An architecture design for low aborting rate (LAR) Concurrency control in mobile databases

Fatemeh Abdi Saghavaz

fasaabdi@nima.ac.ir

Abstract: In the mobile environments, due to some specific features such as the bandwidth restriction, displacement in various geographical regions and disconnection, the efficient and cost-effective design of the concurrency control mechanisms, require techniques that are completely different from the distributed databases. Although the concurrency control protocols which are suggested for the distributed environments can be developed for the mobile environments, their efficiency might be very different from that of the distributed environments[9]. In the suggested plan, we offered a new architecture for the mobile database in which a set of cells form a region and one of these cells under the name 'mobile transaction manager'(MTM) is responsible for the transactions concurrency control which is very suitable for the mobile databases. Also, the suggested concurrency control mechanism which is adopted from an optimistic approach is able to significantly reduce the aborting rate of the transactions in the mobile environment using the transactions early termination mechanism and ignoring some conflictions.
[Fatemeh Abdi Saghavaz. **An architecture design for low aborting rate (LAR) Concurrency control in mobile databases.** *Researcher* 2014;6(7):1-10]. (ISSN: 1553-9865). http://www.sciencepub.net/researcher. 1

Keywords: Mobile Database, Optimistic Concurrency Control, Conflicting Transaction

1. Introduction

It is clear that the fulfillment of fast access to the information in the mobile networks level is dependent upon the fast process of transaction and an increase in their concurrency. The concurrency control is responsible for controlling the performance of concurrent transactions and has a direct and significant effect on the efficiency of the transaction process [3]. Numerous approaches are proposed to improve the concurrency in the mobile environments[2,5,6,8]. Some of them use the lock-based approaches that often have a high blocking rate. Others are from the optimistic approaches (OCC) that are the cause of often improper aborts in the validation phase[1,4].

Compared to the locking methods, the optimistic approaches seem to be attracting for mobile environments due to their nature of being free from deadlocks and less communication overload [8]. Therefore, in this paper, we decided to take measures to help cover the shortcomings of the optimistic concurrency control mechanisms and

Evict those transactions condemned to abortion from the colony of the system active transactions by early detecting them and prevent the proliferation of conflictions among other transactions and reduce the system concurrency.

2. The architecture of the suggested system

As we know the mobile database architecture consists of a Fixed sever, Mobile Hosts, Mobile Server, Mobile Support Station (MSS) and Control Server. The region under the network coverage is divided into various zones each of which is again divided into several cells. The Fixed Servers are computers that are connected to the fixed network and do not have the mobility feature. These computers have the duty to process the transactions, manage the information and respond to queries. Breaking down the transaction and sending it to another fixed station in the network which can execute that sub-transaction. Another part of this architecture is the Mobile Hosts. These hosts are computers that are moving through the wireless network and have the ability to connect to the network via the wireless interfaces[7]. In this part, data storage and transaction management do not take place. The mobile servers are similar to the fixed servers except for this difference that they are capable of mobility and making connection to the wireless network. The main difference between the fixed server and the mobile server is related to the transactions breakdown and processing. In the mobile database architecture, the connection of each cell with other cells is facilitated via a wireless interface specific to that cell which is called the Mobile Support Station and holds the address of all cells. In each network, there is only one Control Server that is responsible for maintaining the physical location of the Mobile Hosts in that network and also the global concurrency control, management and recovery of transactions [4].

In the suggested plan, a set of mobile support stations (cells) form 'a zone'. In each zone, anMSS is considered to be the mobile transaction manager (MTM) (fig 1). It should be mentioned that in the presented figure the dotted lines indicate the wireless communications and the connected lines indicate the wired communications. Upon arriving at each zone, the mobile host stores the number of the zone MTM

in its memory. This information enters a two dimensional array in which one dimension includes the MSS numbers and the relevant MTM number is inserted in another dimension. Suppose that MSSs 1, 2 and 3 are under the supervision of MTM number 1 and the MSSs 4, 5 and 6 are under the supervision of the MTM number 2. Both The mobile host and the MSSs have this array. The MTM of each zone has a copy of all MSS data under its cover. When a transaction is issued from a mobile host, it gets divided into some sub-transactions which will be distributed to various MSSs. The sub-transactions that are executed in the MSSs of a zone will also be executed in the MTM of that zone identically with the same scheduling. In fact, MTM is an intensive pattern from the whole MSSs of a zone and performs the execution of sub-transactions under the same applied scheduling in MSSs. The advantage of this architecture is that it does not require a coordinator when committing the transactions so as to send the committing command to all MSSs involved in the execution of the transaction and only sends it to the mobile transaction managers of each zone (MTMs). Hence, the exchange of information in the network will be reduced and committing will occur faster. In each zone, the number of MSSs is constant and the number of mobile hosts is variable. Each MTM keeps its list of constant and variable members and updates them. If a mobile host goes from one cell to another cell which is under the zone MTM cover, no change will be made in the list. But if the mobile host exits a zone under an MTM cover and enters a new zone, (gives its previous MSS/Id to the new MSS as a part of hand-off.Then the new MSS checks the received Id with its two dimensional array. If any differences arise, which means the mobile host entered a new zone, the MSS sends the join ( ) message (through the wired network) to the zone manager which places the mobile host in its list, and simultaneously informs the previous MSS of removing the mobile host from the previous zone manager's list (He sends the Leave ( ) message to the previous MSS and it in turn sends the message to the MTM of its zone). It should be mentioned that the exchanged messages across the network will be transferred at the fixed network level which can be ignored due to the high speed of these networks compared to the mobile networks. The zone manager should have enough information about its zone data, MSSs under cover and the mobile hosts which enter or exit the zone and also the transactions that are issued from these hosts and work on the data to be able to manage the concurrency and data processing. Ergo, there is a quadruplet set called 'the access set' in the form of d_i:<$time_i$,O_i,MSS_i,MH_i> for each data in the MTM which indicates that in $time_i$, the O_i operation (the subscript indicates the transaction and O is the read and write operations ($O_i \in (r_i, w_i)$) performed an operation on the data $d_i$from a transaction which its origin is the MH_i and is in the MSS_i. $i_n$dicates the read operation and w_i indicates the write operation. Information was created under the name 'report' from the union of access sets belonging to various data in the MTM memory which is stored in the MTM memory of each zone. The report containing information is about an operation that the transaction performs temporarily in the read phase and is supposed to be evaluated in the validation phase. Therefore, after these transactions are aborted, this information will be continually updated and cannot be reflected in the database until the end of the final evaluation. Also, MTM keeps a variable called $R(t_i)$ for each transaction which indicates the number of commands that have been read by the t_i transaction. One unit is added to $R(t_i)$ each time a command is performed. This variable indicates the transactions working progress and serves a purpose when comparing their performance rate and making decisions to select a victim to get aborted in the 'intermediate validation phase'.

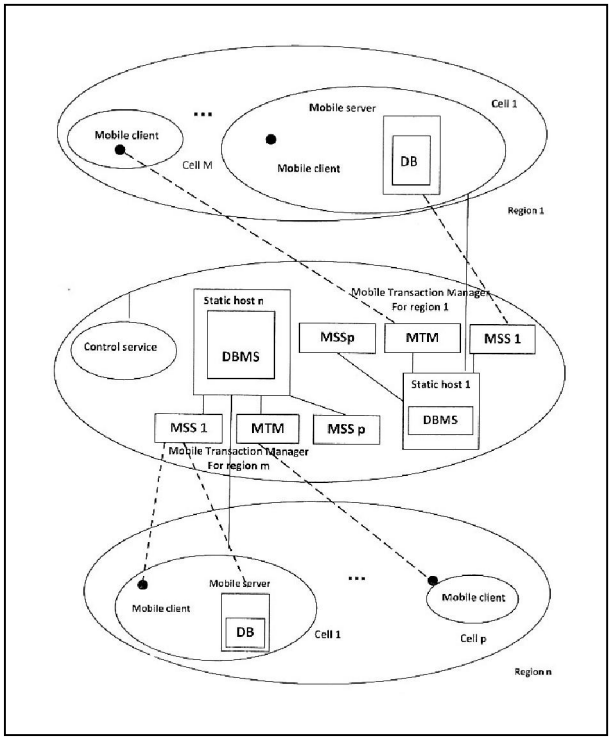

Figure 1: Mobile database architecture, taking into account regional manager of mobile transactions

3.Concurrency control under the suggested architecture

In this section, we propose an approach which is called 'low aborting rate' optimistic concurrency control(LAR). This idea resulted from the fact that

we can ignore aborting some of those apparently conflicting transactions which are to be aborted in the validation phase, via the application of more serializable scheduling. On the other hand, with early recognition of the problematic transactions which are condemned to be aborted in the future (final validation phase), we can economize significantly in consumption of the system resources which are very important in the mobile environment and prevent more conflictions. This way the system output and the transactions concurrency will increase. In this paper, we assume that all of the protocols are strict in that the transactions perform the operations in a private workspace and its results are not visible for other transactions until it is committed. In this situation, some of the conflicting transactions are not inevitable to abort. Let's pay attention to the following example to explain the problem.

Example 1) we have the transactions T_1, T_2 and scheduling with the following commands:

$$T_1 = R(b)w(a)$$

$$T_2 = R(a)w(e)$$

$$Sch_1 : R_1(b)w_1(a)R_2(a)V_1$$

Here, a read-after-write (in short: R-after-W) confliction on 'a' occurred. After w_1 (a), the r_2 (a) command was executed. As a result, when T_1 arrives at its validation phase (V_1), the T_2 will be aborted according to the forward validation [10]. With a bit of contemplation, we find out that in the optimistic approaches, T_2, in fact, read the data 'a' before the effect of command w_1(a) appears in the bank (T_2 cannot read w_1(a) in a private workspace, and therefore reads the old value of the data in the bank)[4]. Hence, the R-after-W confliction can create the $T_2 \rightarrow T_1$ priority with regards to the invisibility of the private workspace under the strict protocol. Ergo, if T_1 can (in case of not violating the principles which we will mention) delay its validation; until first the T_2 is committed, there is no need to abort T_2.

It should be noted that allowing the $T_2 \rightarrow T_1$ scheduling with such transpired confliction is only possible due to the strict protocol assuming conditions which will be mentioned further and is not true under the conditions in which the transactions changes are applied instantly in the bank.

In any way, our scheduling will be as follows:

$$R_1(b)w_1(a)R_2(a)[V_1]w_2(e)V_2\,C_2\,V_1\,C_1.$$

[V_1] indicates the delay in executing V_1 and the serializability order is $T_2 \rightarrow T_1$

Example 2) Now, pay attention to the example of write-after-read (in short: W-after-R) confliction. We see the same transactions with different scheduling.

$$sch_2 : R_1(b)R_2(a)w_1(a)V_1$$

Again, according to the forward validation, T_2 is condemned to abort, yet since T_2 had read the data 'a' before it was applied in the bank, and did not commit any violations –which will be explained- so far, we can ignore its abort and delay the T_1 validation until the T_2 is first committed.

$$R_1(b)R_2(a)w_1(a)[V_1]w_2(e)V_2\,C_2\,V_1\,C_1.$$

The scheduling result is the $T_2 \rightarrow T_1$ serialization.

By allowing such scheduling, we will be able to reduce the aborting rate in OCC. It should be mentioned that the write-after-write (in short: W-after-W) confliction will not cause any priority among the transactions.

3-1 The suggested serializable graph

By allowing the 'read' to come before the 'write' and literally precede it without observing any principles, and in the case of writing for others to come 'prior' and the write to be 'preceded' by them, a complicated graph with cycles will form. Therefore, we will explain specific principles and criteria which will simplify our serializable priority graph into a two-part graph. We call this two-partserializable priority graph under the priority protocol 'serializable priority graph ' (fig 2).

3-2 The priority protocol

Priority among the transactions will be posed only when the conflictions are R-after-W or W-after-R. For simplification, we call the transaction which writes a data the 'write transaction' of that data and the one which reads a data the 'read transaction' of that data. Ergo, in the R-after-W confliction, the read transaction will precede the write transaction. Knowing the type of transaction whether it is 'posterior' or 'prior' and after the type of priority is specified, we can draw the serializable priority graph of the transactions which indicates their priority. We call graph G (V, E) the serializable priority graph of a system that follows the strict protocol. The instructive information of these graphs in MTMs is exchanged via reports and is specified sooner if a problem arises in the graph, which appears as a cycle. In any way, E indicates the edge among the transactions which specifies their priority. Assume $T_i \rightarrow T_j$ and an edge that is drawn from T_i to T_j, namely T_i, read the data written by T_j in its workspace. If we let the transactions with R-after-W or W-after-R confliction, precede each other without any obstacles, this creates a complicated serializable priority graph.

For simplification, the prior and posterior transactions observe the priority protocol rules. The purpose is to allow the conflicting transactions to be either prior or posterior and not both. (One form of

the possibility of a cycle appearing in a graph is that a node should have both an input and an output edge.)

● **The priority protocol rules.**

1- For read (T_i)- after – write (T_j):

T_i is allowed to proceed the T_j (Prior=T_i, Posterior=T_j) If there is no other transaction prior to T_i (posterior=T_i), T_j will not be prior to any other transactions (Prior=T_j).

2- For write (T_i) – after – read (T_j):

T_j is allowed to precede the T_i (Posterior= T_i) If T_i is not prior to any other transactions (Prior=T_i) and T_j does not have the posterior role by any other transactions.

A transaction can be independent which means it has no confliction with any other transactions or can be dependent which means it has conflictions with other transactions. The serializable priority graph which follows the priority protocol has two parts in which the edge always goes from the transactions to the posterior ones and not visa versa. Transactions that follow the priority protocol will not create any cycles in the graph.

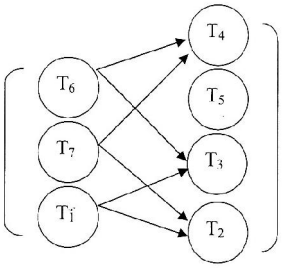

Figure 2: The serializable priority graph under the priority protocol

3-3 The violating transactions

It is possible that the posterior transaction (for example T_i) wants to be prior to another transaction (T_i gets the prior role too) or the $T_j$transaction which is posterior itself wants to be prior to the T_i (T_j is prior too) and/or it is possible that the prior transaction (for example T_k) is prior to another prior transaction such as T_L (T_L has the posterior role too) or another transaction become prior to T_k (in this case, T_k will get the posterior role too). These transactions that violate the priority protocol rules are called the violating transactions. In the following, we show such an example:

Example 3): The transactions, namely T_1, T_2 and T_3 are given with their related scheduling:

$$T_1 = R(b)\, w(a)$$

$$T_2 = R(a)\, w(e)$$

$$T_3 = R(e)$$

$$sch_3 : \underbrace{R_1(b)\ w_1(a)\ R_2(a)}\ w_2(e)\ R_3(e)$$

Violating the priority protocol
the R-after-W confliction

When T_2 reads the data 'a', which is written by the transaction T_1 (in the T_1 workspace and not in the bank), faster committing of T_2 $(T_2 \rightarrow T_1)$ will be established.

When T_3 wants to read 'e', it thinks that it can be prior to T_2, but since T_2 has already become prior, now it cannot play the role of posterior for T_3. ($T_3 \rightarrow T_2 \rightarrow T_1$ a cycle is probable to appear). In case a transaction violates a rule, it will be aborted.

3-4 The transaction execution model

As we know, in the optimistic concurrency control approach, transactions are allowed to continue their job until they arrive at the committing point without any obstacles. Then, they will be validated before being committed. In the traditional optimistic concurrency control, performing the transactions consists of three phases, namely read, write and validation. The transactions fate will be sealed in the last phase[10] [11]. Validation can take place in one of the two ways, namely 'backward validation' and 'forward validation'. Most of the existing OCC protocols use the forward validation due to its flexibility when selecting the transaction condemned to restarting in terms of criticality or priority from among then transactions being validated or the conflicting active transaction [4]. While in the backward validation, the candidates are only those transactions under validation. Also, the forward approach recognizes the conflictions quicker which leads to economization in time and the system sources. Hence, the forward validation will be used in the suggested plan. In the suggested concurrency control mechanism, performing the transaction has four phases, namely the read, intermediate validation, final validation and the write phase.

3-4-1 Reading Phase

At this stage, the transactions are conducted freely, but they all write operations in the working space which is accessible only for private transactions, so they are not visible to other transactions.

3-4-2Intermediate validation phase

By using this phase, conflicting transactions can be detected early and before the final validation phase are completed.In this phase, if a conflicting transaction that follows priority protocol rules can be prior. Otherwise, it will abort and restart. Perhaps it is assumed that the priority protocol and observing its rules causes an increase in the aborting rates. While

according to this approach, if a transaction meets the conditions of the examples (1) and (2), based on the forward validation it is condemned to abort, but according to the priority protocol one such confliction between two transactions will be ignored, yet in case any confliction arises which is against the principles of the priority protocol, (which is disproved by the forward validation approach too) it cannot be ignored based on the priority protocol.

In the suggested plan, MTM performs the intermediate validation operation periodically, for example every L seconds. In a way that those steps that are taken from the transaction until the intermediate validation moment will be compared to other concurrent transactions. In case a non-negligible confliction arises, with regards to the priority protocol, the transaction is condemned to restart is chosen in terms of criticality or priority. We will assign a tag to each transaction which shows the type of transaction based on priority.

This tag can be adjusted as prior, posterior or independent. In order to help commit the transactions that follow the arranged priorities, a 'before-list' is made which is the list of transactions that are prior to this transaction. Similarly, if the transaction is a prior transaction (the type of transaction is recognizable from the tag in MTM of each region), an 'after- list' will be made for it that includes transactions which took the role of posterior. When the transaction becomes final, the system can use this information in the direction of updating the prioritization information of the posterior transactions.

As noted above, MTM performs validation on a periodic basis every L seconds. To avoid repetition of tests a "check point" can be inserted into transaction reading set. However, despite using of intermediate validation phase, transaction failure in final validation phase is probable. For example, the intermediate validation could be done every 7 seconds and transaction execution time is 17 seconds. Last intermediate validation will be happen in the 14^{th} second, while it may be failed in the 15^{th} second.

Figure (3) shows intermediate validation algorithm.

```
If there is a Read- After- Write or Write-after-Read
confrict
{ if transaction follows priority
protocol,
proceed with the operation;
else
with comparing the R(ti) Value of conflicting transactions
abort transaction with the less value of R(ti)
```

Figure 3: intermediate validation phase

As noted above, in this phase the reading set of the transactions is compared to reading set of other concurrent transactions. Intermediate validation is performed on MTM. As we know MTM keeps an access set di:<$time_i$,o_i,Mss_i,MH_i> for each of its data. This information is continually updated based on the transactions reading set which is temporary. If we study them, we can infer the confliction of those transactions which simultaneously get access to a data. Whenever a transaction is aborted, its related information is removed from the access set of those data with which the transaction worked. During the intermediate validation, this information is exchanged among the MTMs that cooperated with one another during the performance of the transaction, because it is possible that the data 'di' gets accessed by those transactions that are outside of a zone under MTM coverage. After getting information from other MTMs, the serializability graph of each zone will be created during the evaluation, the transaction is aborted sooner if a cycle is found. It should be mentioned that these reports are exchanged periodically and concurrent with applying the intermediate validation phase by the MTMs. Each MTM is responsible for sending the report definitely to other engaged MTMs. In the following, you can see the report-making and sending algorithm:

1- For each di<$time_i$,o_i,Mss_i,MH_i> which is in each MTM, Id of each Mss_i is scanned.

2- IfMssi belongs to the zone under MTM coverage, it means that the data is accessed locally. Therefore, we will insert di:<$time_i$,o_i,Mss_i,MH_i> into the related report.

3- Otherwise (i.e. non-local access to the data), we will insert the access set into the report and then send it to the MSS which is the supervisor of the non-local MTM.

4- We will combine the local report with reports from other MTMs.

```
For each di:<timei,oi , Mssi, Mhi> in MTM
{if (di is accessed locally)
add access- set to the report ;
else { add access- set to report and send it for MTM
which Mssi belongs to it } }
Combine Report with external reports from another
MTM
```

Figure 4: The report-making and sending algorithm

In each intermediate validation, an evaluation is carried out in order to identify the conflicting transactions by receiving and combining reports.

Figure (5) shows the algorithm for identifying the conflicting transactions.

The period of sending report is identical with the period of performing a validation; i.e. As soon as an MTM receives a new report it combines it with its report and creates a single report. The act of intermediate validation is so simple. It is enough to specify the data that at one time (by studying the $time_i$ belonging to the access-set of each data) were not accessed by the conflicting transactions. This way, we can recognize the conflicting transactions. As we mentioned earlier, MTMs retains a variable called $R(t_i)$ for each transaction which is indicative of the progress being made in the transaction process. By comparing $R(t_i)$ of the two conflicting concurrent transactions, we choose the transaction with the smaller variable as the victim for aborting.

```
{for each di:<time_i,o_i,MSS_i,MH_i> in the combined report of MTM_i
{for each dj:<time_j ,o_j, MSS_j, MH_j> in the combined report of MTM_j
{if (di=dj and t_i<t_j )
Terminate t_i ;/* if the read timestamp <update timestamp*/
} } }
```

Figure(5): The algorithm for identifying the conflicting transactions

3-4-3 The final validation phase

In this phase, the evaluation will be carried out in front of all of the transactions that are being performed concurrently. Identification of the conflictions takes place by comparing the write set of the transaction under evaluation with the read set of the active transaction. The transaction that is aborted in this phase should be restarted immediately. In order to carry out the final validation, it is not necessary for all commands to be evaluated from the beginning and evaluating the commands after the last check point will suffice. This way, the transaction won't wait too much for the final validation and the speed of its being committed and as a result the concurrency of the system goes up. A transaction that enters this phase immediately requests for the right on the data that changed in the private work space. A timer is set for each transaction that undergoes validation and its value decreases over time (we will adjust the timer value as mush as the time until which we expect the transaction to be committed). The transactions being validated will be adjusted in two states based on the before-list and after-list.

The first state: All of the prior transactions will be committed as long as the timer has not reached 0.

The second state: The transactions that are prior are not committed completely. But timer turns 0 (timer=0). In this case, the transaction being validated will abort the prior transaction and it will be committed itself.

```
/* when final validation phase of transaction t_istarts*/
set timer for t_i ;
if timer is equal to 0 {
abort all the transactions which precede t_i ;
t_i commits ;
} Else {
t_i waits
(until all preceding transaction terminates);
t_i commits ;
      }
```

Figure(6): final validation phase

3-4-4The writing phase

In this phase, after the final validation, the result of the transaction operation in the read phase is transferred from the workspace to the database. This way, the transaction private records will be visible for other transactions.

In order to explain the proposed algorithms and the suggested architecture, consider the following example:

Assume we have a network with the following specifications:

The network has 18 mobile support stations with numbers 1 to 18 and has 3 regions. Each region has 6 mobile support stations. MTM_1 of the first region covers MSSs 1 to 6, MTM_2 cover s MSSs 7 to 12 and MTM_3 covers MSSs 13 to 18, respectively. When the mobile host enters each region, the number of related MSSs and the mobile transaction manager of the region, enter a two-dimensional array which is in mobile host memory.

Consider the transactions T_1, T_2 and the following scheduling. Assume the data 'a' has full repetition in all MSSs. The place of execution for each instruction is mentioned under each command.

Example 4)

$$T_1 : R(a)\, w(a)$$

$$T_2 : R(a)\, w(a)\, R(b)\, R(c)\, w(c)$$

$$Sch_4 : \underbrace{R_1(a)}_{Mss_1},\ \underbrace{R_2(a)}_{Mss_5},\ \underbrace{w_1(a)}_{Mss_7},\ \underbrace{R_2(b)}_{Mss_7},\ \underbrace{[V_1]}_{Mss_7},\ \underbrace{w_2(a)}_{Mss_4},\ \underbrace{R_2(c)\, w_2(c)}_{Mss_3},\ \underbrace{ab_2}_{Mss_6},\ V_1, c_1$$

The execution of the transactions T_1 and T_2 is distributed among the MSSs of the regions 1 and 2. r_1 (a) and r_2 (a) are executed in region number one. The information related to the data 'a' is written in the

access set of the data 'a' (This set is registered in a file called 'report' which is related to the MTM of each region). When transaction T_1, writes 'a' in the second region, according to the priority protocol rules, T_1 takes the role of posterior via T_2 ($T_1 \rightarrow T_2$) i.e. when T_1 arrives at its final validation phase, it will be delayed [V_1]. It is because T_2 became prior to it due to the earlier r_2 (a).

When T_2 wants to write 'a', w_2 (a), i.e. it wants to take the posterior role via the T_1 and the command r_1 (a) ($T_1 \rightarrow T_2$) and this intermediates a violation of the priority protocol rule. With the occurrence of the intermediate validation and the exchange of reports between the MTMs of regions 1 and 2, one of the violating transactions should be aborted. Since T_2 becomes prior to the transaction T_1 which is being validated (T_1 arrived earlier at the final validation phase), in order to avoid longer validation time T_1, the T_2 transaction will be aborted (ab_2). Finally, T_1 is validated and committed. It is observed that the T_2 transaction is aborted before arriving at the final validation phase via the intermediate validation.

When a mobile host of the transaction T_1 enters the second region, the commit command ($c_{1)}$ will be issued. The commit command has been sent from the cell 7 coordinator (according to the information stored in the mobile host memory) to the managers of the visited regions. After the final validation (by exchanging reports between the two managers involved in executing the transaction), the committing act will take place. The changes in data will be distributed across the two MTMs data and then according to the repetition protocols[1], these changes will be applied in all of the desired mobile environment banks.

4. Comparison of the suggested plan and the traditional optimistic approach

In order to evaluate the performance of the concurrency control mechanisms, there are various parameters such as the response time, the effective output and the probability of transactions conflict. In this part, we compare two approaches, namely the traditional optimistic concurrency control mechanism and the suggested plan based on the framework of these parameters.

4-1The time complexity function of the transactions' responsiveness

The average of transactions' response time has a direct relationship with the size of transaction and the number of conflicts in the system [3].

$$R(M) = (k+1)s(\overline{M}_a) + kp_c w \qquad (1)$$

In equation (1), K is the transaction size; and p_c is the probability of conflict for each data requested by the transaction. $S(\overline{M}_a)$ is the average of the processing time for each transaction step in a system with M active transactions. The conflicting transactions restart after the specified delay (W). If this transaction is condemned to abort, according to the suggested plan and based on the period adjustment of the intermediate validation phase, a specific transaction might be identified after executing k/2 of its size (on average). Therefore, the average of the conflicting transactions' response time decreases as follows which is shown with index (our):

$$R(M)_{our} = (\frac{k}{2}+1)S(\overline{M}_a) + \frac{kp_c W}{2} \qquad (2)$$

4-2 The transactions' effective output

The transactions' effective output can be obtained from equation (3) and equals to dividing the number of the database transaction by the transactions' response time [3]:

$$T(M) = \frac{M}{R(M)} \Rightarrow T(M) = \frac{M}{(k+1)s(\overline{M}_a) + kp_c W} \qquad (3)$$

Since the amount of response time for the transactions decreases in the suggested plan according to equation (2), the transactions' effective output is as follows which is indicated by the index (our):

$$T(M)_{our} = \frac{M}{R(M)_{our}} \Rightarrow T(M) = \frac{M}{(\frac{k}{2}+1)S(\overline{M}_a) + \frac{kp_c W}{2}} \qquad (4)$$

It goes without saying that as the R(M) decreases, the amount of the conflicting transactions' effective output increases.

4-3 The relationship between the conflict probability and the transaction size

Regarding the fact that n_c is the average of the number of conflicts for each transaction, it equals $n_c=kp_c$. When a transaction requests data, the probability of data conflict (p_c) for the i^{th} data request equals [3]:

$$P_c = \frac{\overline{N}-i}{D-i} \simeq \frac{(M-1)\overline{L}}{D} \simeq \frac{(M-1)k}{2D} \qquad (5)$$

$\overline{L}$ Indicates the number of data that are accessed by the transaction; $\overline{N}$ is the average of the number of data which is utilized by other transactions (M-1); and D is the total number of the database data. Hence, the data conflict probability after performing the last step of a transaction equals:

$$p_W = 1-(1-p_c)^k \simeq kp_c \sim \frac{(M-1)k^2}{2D} \qquad (6)$$

This conflict probability is obtained after the last data required by the transaction was requested in the last step (k^{th}) of performing the transaction; while in the suggested plan, we can decrease this

percentage to the proposed value in equation (7) by evaluating the transaction in the intermediate validation phase and identifying the transaction condemned to abort by executing k/2 its size.

$$p_W = \frac{(M-1)k^2}{2D} \quad k = k/2 \Rightarrow p_W^{our} =$$

$$(1-p_c)^{k/2} \simeq k/2\, p_c \sim \frac{(M-1)k^2}{8D} \qquad (7)$$

4-4 The average of the transactions' validation time

In the optimistic techniques, transactions undergo evaluation after the reading phase is completed and before committing .Considering the probability of data conflict and the number of active transactions, the average of the evaluation time equals [3]:

$$E = \sum_{j=1}^{K} \frac{2j}{k(k+1)} \quad [(k-j)\ (S(\overline{M}_a) + p_c] \qquad (8)$$

j is the number of data that are accessed by the active transactions. In the proposed approach, committing of the transactions takes place after making some arrangements with a limited number of MTM_s and it occurs faster compared to other concurrency control approaches in which the coordinator exchanges information with a large number of MSS involved in the operation. On the other hand, the status of the active and delayed transactions can be determined faster using the intermediate validation in general and the data conflict decreases as a result. It is clear that the decrease in $S(\overline{M}_a), j, p_c$ affects the final validation time average of transactions. Besides, according to the intermediate validation phase effect and the fact that at least half of the transaction size is evaluated in the pre-mentioned phase, the rest of the steps will be evaluated after the check point in the final validation phase. Ergo, the average of the final validation time decreases as follows:

$$E = \sum_{j=1}^{K} \frac{2j}{k(k+1)} \quad [(k-j)\ (S(\overline{M}_a) + p_c]$$

$$k = k/2 \Rightarrow E_{our} = \sum_{j=1}^{K/2} \frac{2j}{\frac{k}{4}(k+\frac{1}{2})} \quad [(k/2 - j)\ (S(\overline{M}_a) + p_c] \quad (9)$$

5- The evaluation environment

This system is evaluated via the MATLAB software in which the parameters are considered as depicted in table (1). It should be mentioned that the selected values in this table are common in many of the evaluations in the mobile environment. [3][4]. As we know, the optimistic approaches are suitable for the environments in which the number of reading operations is more than the writing operations. Also, since the transaction size in the mobile environment is not much, the maximum size is taken to be 20.

Table (1): The characteristics of the evaluation environment

Values	Parameters
λ_u	The input rate of write transaction is 5 per second.
λ_r	20 read-only transactions enter the system each second.
m	The probability of the mobile host's movement from one cell to another equals 0.1
M	The total number of transactions that enter the system is between 50 to 250
D	We consider the database size to be 250. By size, we mean the number of data items in the bank.
K	The maximum size of transactions is considered to be k=20
W	Delay to restart a transaction is taken to be 10 time unit.
$P_c = \frac{(M-1)k}{D}$	The probability of one write command conflict with another read command equals 2

5-1 The evaluation results

Regarding the relationships which were proposed in the previous part, the suggested approach is compared to the traditional OCC approach and the results of evaluation are depicted as some charts.

According to the suggested plan, most of the conflicting transactions in the intermediate validation phase will abort except for those which commit failure after the last intermediate validation. Therefore, the system's response time for conflicting transactions decreases as figure (7). It is clear that as the time decreases for the conflicting transactions, the total average of the response time decreases for all the transactions that are a combination of the conflicting transaction and those without conflicts. As can be seen in figure (7); the response time of both approaches increases as the size of transactions increases. In the traditional OCC approach, the time equals 450 time units considering the values depicted in table (1). Each transaction gets restarted all over again after getting aborted with a w(10) delay of the time unit. $S(\overline{M}_a)$ is the average of the processing time for each transaction step in a system with M active transactions which equaled 0.2 [3] and p_c which happened to be 2 after placing the values of table (1).

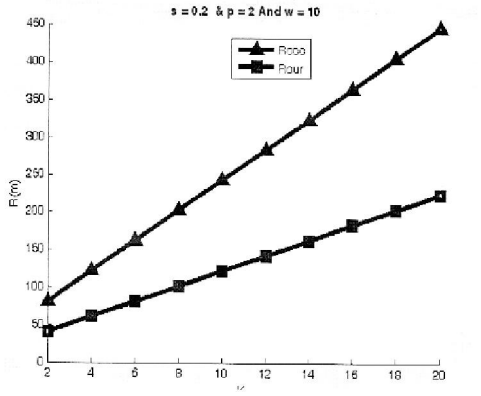

Figure (7): The comparison of the transactions' response time average between the traditional OCC and the suggested approach.

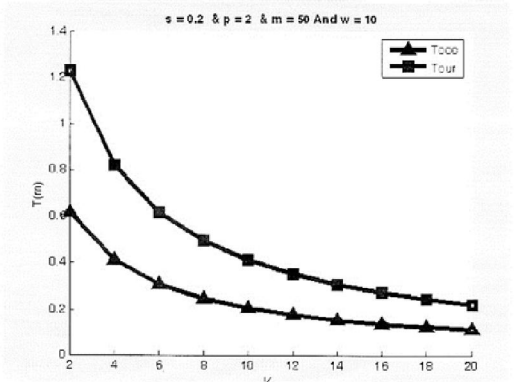

Figure (8): The comparison of the average of the transactions' effective output between the traditional OCC and the suggested approach

The effective output of transaction is obtained by dividing the ratio the total number of transactions by the response time average. Needless to say, the effective output of each transaction increases when the transactions' response time decreases according to figure (7). Figure (8) indicates an improvement in the suggested plan. The increase in the transactions' size led to an increase in the conflict probability and the response time increases as well. In both approaches, the increase in transactions' response time led to a decrease in the transactions' effective output. But, the transactions' effective output shows lesser decrease in the suggested plan due the improvement in the response time. In such a way that we obtain $0.3<T_{our}<1.2$ and $0.2<T_{occ}<0.6$ when we consider the preliminary values of table (1).

It is clear that the more the transaction size, the more the conflict probability in the system. If this transaction is a transaction condemned to abort, in traditional OCC techniques, the total transaction size

will be evaluated in the final evaluation phase; but in the suggested technique, at least half of the transaction size will be evaluated in the intermediate phase; So if this is a transaction condemned to fail, it does not wait for the evaluation and prevents more conflicts by getting early aborted. As it is shown in figure (9), in both approaches, the increase in transaction size leads to an increase in the conflict probability. Regarding the evaluation environment parameters, in the traditional OCC approach, this conflict probability is between 0 to 20 and in the suggested plan, it is between 0 to 4.

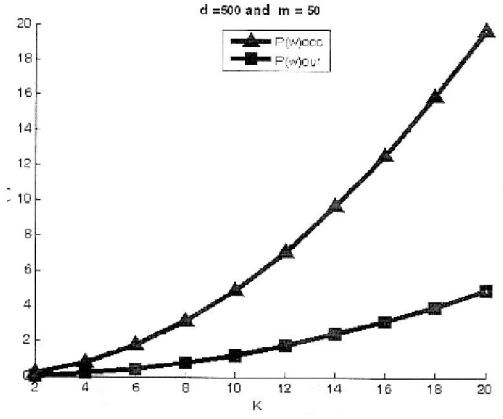

Figure (9): The relationship between the conflict probability and the transaction size and its comparison in the traditional OCC and the suggested plans.

Also, in the presented plan, due to the presence of the intermediate evaluation phase which of course does not create any delays in the transaction process, according to figure (10), the final evaluation time decreases. In both approaches, as the transactions size increases, the evaluation time increases as well. But in the suggested plan, commands will be evaluated after the check point in the final phase. In such a way that we obtain $0.05<E_{our}<0.45$ and $0.1<E_{occ}<1$ when we consider the evaluation environment parameters of table (1).

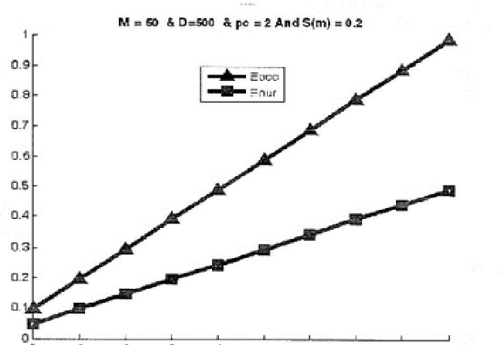

Figure (10): The relationship between the evaluation time and the transaction size and its comparison in the traditional OCC approach and the suggested plan

Conclusion

In the suggested plan, a set of MSSs form a zone and each zone is under the supervision of a station, namely the mobile transactions manager(MTM).

1.Under the proposed architecture, the time-consuming and costly operations such as the intermediate validation, the final validation and committing in the fixed and intensive part of the network, namely MTM will take place. Thus, the pre-mentioned operations will take place quicker and the fate of a transaction (commit or abort) will be determined sooner.

2.When sending the commit command from a mobile host, there is no need for a coordinator to send the commit command to all MTMs involved in performance of the transaction and simply should send it to the managers of each zone, MTMs, (MSS>>MTM). Therefore, the information exchange among the mobile support stations will decrease and so does the network traffic.

3.When committing the transactions, under the 2PC protocol, the coordinator which is responsible for committing should send the committing command to all mobile support station involved in performing the transaction. The coordinator waits until it gets 'yes' from all MSSs or set a specified deadline. If one of the MSSs stop working and does not send the message, the deadline will be missed and the coordinator attempts to abort the transaction [4]. In the suggested plan, in case of a failure in any MSS when receiving the answer 'yes', committing the transaction does not get delayed or often aborted.

4.Using the intermediate validation, the conflicting transactions can be identified and aborted sooner. In this way, we can economize significantly in consuming the system sources which is very vital in the mobile systems.

References

1. Jin jing, omranBukhres, Ahmed Elmagarmid, "Distributed lock management for mobile Transactions", proceedings of the 15th International conference on distributed computing systems (ICDCS), 1995 IEEE.
2. Sung Ho Cho. Jong Min Lee, chong- sun Hwang, "Hybrid concurrency control for mobile computing" proceedings of the High- Performance computing on the international super highway, 1997 IEEE.
3. Thomasian. Alexander, "concurrency control: methods, performance, and Analysis", ACM computing surveys, vol.30, No.1, March 1998.
4. ParticiaSerpando- Alvarado, Claudia Ronancio and Michel ADIBA "A Survey of Mobile Transaction", Distributed and parallel Databases, Kluwer Academic Publisher's March 2004
5. ShapourjoudiBegeillo, FariborzMahmoudi, Mehdi Asadi, "Improving strict 2 phase locking (S2PL) in transactions concurrency control", International conference on convergence Information Technology, 2007 IEEE.
6. Sebastian obermeier, stefanBottcher, "Avoiding infinite blocking of mobile transactions" ll' thInternation Data base Engineering and Application symposium (IDEAS), 2007 IEEE.
7. Jing Li, Jianhua Wang, "A New Architecture Model of Mobile Database Based on Agent," dbta, pp.341-344, 2009 First International Workshop on Database Technology and Applications, 2009
8. Salman Abdul Moiz, Lakshmi Rajamani, "Concurrency Control Strategy to Reduce Frequent Rollbacks in Mobile Environments," cse, vol. 2, pp.709-714,InternationalConference on Computational Science and Engineering, 2009
9. Anne Marie Amja, Abdel Obaid, Normand Seguin,"A Distributed Mobile Database Architecture," apscc, pp.62-69, 2011 IEEE Asia -Pacific Services Computing Conference, 2011
10. Kamal Solaiman, Matthew Brook, Gary Ushaw, Graham Morgan, " A Read-Write-Validate Approach to Optimistic Concurrency Control for Energy Efficiency of Resource-Constrained Systems", Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International
11. Ganeshkohad,Shikhagupta,Truptigangakhedkar,Jogen derraghuvanshi,Umeshahirwar", Concurrency Control issues in Mobile Database", International Journal of Computer Architecture and Mobility, (ISSN 2319-9229) Volume 1-Issue 8, June 2013.

7/11/2014