Dynamic Allocation of Abundant Data Along Update Sub-Cycles to Support Update Transactions in Wireless Broadcasting

Ahmad al-Qerem
Department of computer science,
Zarqa University Zarqa, Jordan

***Abstract*—Supporting transactions processing over wireless broadcasting environment has attracted a considerable amount of research in a mobile computing system. To allow more than one conflicting transactions to be committed within the same broadcast cycle, the main broadcast cycle need to be decompose into a sub cycles. This decomposition contains both the original data to be broadcast in Rcast cycle and the updates come from the committed transactions on these data items called Ucast cycle. Allocation of updated data items along Ucast cycles is highly affecting the concurrency among conflicting transactions. Given the conflicting degree of data items, one can design proper data allocation along Ucast Cycles to increase the concurrency among conflicting transactions. We explore in this paper the problem of adjusting abundant data allocations to respond in effective way to the changes of data conflicting probability, and develop an efficient algorithm ADDUcast to solve this problem. Performance of our adjustment algorithms is analyzed and a system simulator is developed to validate our results. It is shown by our results that the ADDUcast is highly increased the average number of committed transactions within the same broadcast cycle.**

Keywords—data broadcast; Dynamic Allocation; concurrency control; cycle decomposition; Abundant Data

I. INTRODUCTION

Data broadcast is becoming a promising way to disseminate information to a large population of mobile clients by mean of transaction. Unlike the conventional client server approach, where a data item have to be send many times to deliver the requested data even in the case of read –only transactions. Broadcast has the potential to satisfy all outstanding requests for the same data with a single response. It increases the efficiency of shared bandwidth and improves the system throughput. However, existing mobile technologies have to face several constraints such as limited network bandwidth, frequent disconnections and insufficient battery power. To cope with these constraints, there has been many studies on data transmission techniques using wireless data broadcasting [1], [7] and [8]. Generally a mobile client sends requests to the server and receives the response. For sending requests, mobile clients have to consume uplink bandwidth. The response time also can dramatically increase when the server is heavily loaded by the requests from a large number of clients. However, wireless data broadcast models can overcome these problems. For example, [1] proposed the broadcast disks model. The server continuously and repeatedly broadcasts all data in the database using a single or multiple wireless communication channels. Clients wait for the data in need to come up on the channel, and retrieve data from the channel. In this system, the number of mobile clients does not affect their access time. With its good scalability, wireless data broadcast is used in various mobile applications, e.g. auctions, electronic bidding, stock trading, weather information and traffic information broadcasts [9]. In these applications, the consistency among data items is likely to be violated by update transactions [8], [10] and [16]. Thus, a concurrency control scheme is needed to preserve data currency and consistency for mobile transactions. However, conventional concurrency control methods cannot be directly applied to mobile transaction processing [10], [11]. On the other hand, when an application involves wireless mobile clients that run multiple-operation transactions and dynamically update the server database, those updates have to be appear in the next broadcast cycle. Earlier show up of such updates is highly improving the performance. In this paper we aim to decompose the main broadcast cycle into sub cycles which only contain a subset of data items in the database. This decomposition contains both the original data to be broadcast and the updates come from the committed transactions on these data items. At sub cycle level, it is more powerful for both update and read-only transactions which allow more than one conflicting transactions to be committed on the same broadcast cycle. Moreover, by using sub cycle the currency of the data may be higher since the delays in performing the updates are shorter. There have been many research efforts reported in the literature that tackle the concurrency problems in wireless broadcast environments, such as Update First Ordering (UFO) [5], Multi-Version Broadcast [2, 3], Serialization Graph [2, 3], Broadcast Concurrency Control with Time Stamp Interval (BCC-TI) [6], F-Matrix [4], and Certification Report [7]. The drawbacks of these methods have been analyzed in [12, 13]. In general, some of these methods only support client read-only transactions, and some of them could have substantial processing overhead.

The major contribution of this paper is determining the proper allocation of abundant updated data item along the remaining Ucast cycles and dynamically adjustment of such allocation to support update transactions responding in effective way to the changes of data conflicting probability, and develop an efficient algorithm DADUcast to solve this problem. According to the extensive analysis and comprehensive performance evaluation, the proposed approach shows a satisfactory performance in transactions processing on

broadcast environments. It is shown by our results that the DADUcast is highly increased the average number of committed transactions within the same broadcast cycle.

The rest of this paper is organized as follows: Section II describes the system architecture. Section III illustrates the problem of abundant data and how it could affect the transaction processing. Section IV proposes the ADDUcast approach in more details. Section V presents the performance evaluation of our approach and show the superiority of the proposed algorithm. Finally, we conclude the paper in Section IV.

II. System Model

The system model of adopted in this paper allows both read-only transactions (reading from air without being known by the server) and update transactions (modifying data by sending requests through a low bandwidth channel after completion at client). All operations are executed in order.

The broadcast cycle is divided into multiple sub-cycles of two types alternatively see Fig.1: read cast called Rcast and update cast called Ucast in alternative way. The former contains the data items which were predefined scheduled for the main broadcast cycle but distributed along many Rcast whereas the Ucast content is dynamics and changed based on the data being updated by the committed transactions in the previous Rcast cycles. Between any two Rcast cycles, there is a reserved space for Ucast to accommodate identities for all the data objects which are updated by transactions in the server after the first sub-cycle begins.

A mobile transaction can validate its prefetched data consistency autonomously by accessing the Ucast cycle. Our goal is to reveal the write set of the committed transactions as soon as possible. Unfortunately, this may jumble the original schedule. We can tradeoff the immediate show up of the updated data items from the previous sub cycle and delaying all those updates until the beginning of a next sub cycle which will be accommodated in front of pre assigned Rcast index. As a result, the identities of all the data items which are updated by any transactions are included in the Ucast space after the current broadcast sub-cycle.

The identities of all the extra data items with no rooms at the current UCAST are included in the next UCAST based on their conflicting probability. At the same time, since all information of latest committed update transactions is dynamically loaded at the beginning of some Rcast index segment, a mobile transaction only needs to tune to the channel at specific periods to retrieve the requested data item and the control information associated with it.

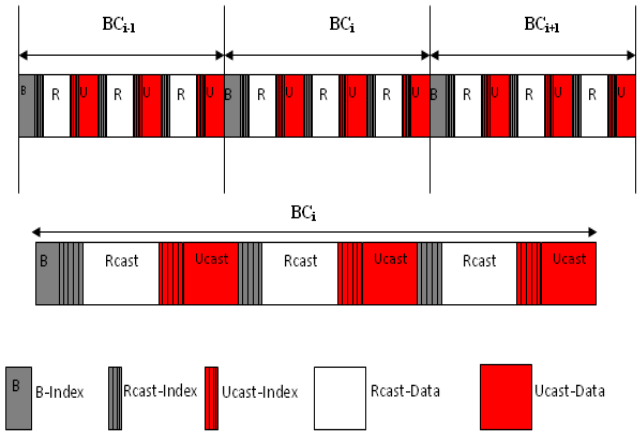

Fig. 1. Broadcast Cycle Decomposition

Data broadcast usually requires a client to be active all the time in order to monitor the data units that go by. This leads to unacceptable energy consumption on wireless mobile equipments, for which power saving is a very essential issue. To save power in data broadcast models, indexing schemes are proposed in [17]. The Basic idea of indexing is to insert pointers for data broadcast in a future schedule into a broadcast cycle. Consequently, a client application can go to doze mode after it accesses this pointer, and only wakes up at the time the requested data unit is on the air. Several index schemes have been proposed in. In all indexing schemes, an index tree of all data in a broadcast cycle is inserted to the schedule. Pointers to each real data units are located at the leaves of the index tree while a route to a specific leaf can be found following the pointer from the tree root. In our work the B-index contains only information about the starting time of both Rcast-index and Ucast-index based on the number of sub cycles and the sizes for each of them; this is easy know as all Ucast are of equal size to and the Rcast cycles is previously determined at the beginning of each broadcast cycle. In addition to the index information provided by Rcast-index and Ucast-index, Rcast-index contains a pointer for the next Rcast. the Ucast-Index contains the actual index of the data items to be broadcasted and each index is associated with control information to validate its pre fetched data items such as sub cycle number as well as the conflicting degree which will be used in allocation and adjustment procedure as we will be explain later in this paper. Fig.2 shows the structure of index information for each element in Ucast-index in addition to the pointer indicating the next Rcast-index segment as the client may arrive at any time during the major broadcast cycle.

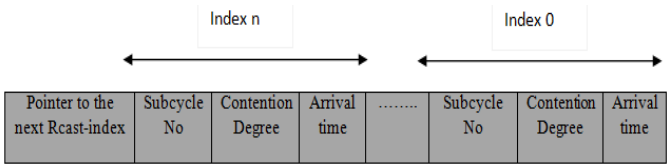

Fig. 2. Information associated with each index in Ucast cycle

III. PROBLEM DESCRIPTION

By exploiting the feature of tree generation with variant-fan-out [14], [15] a heuristic algorithm to distribute the abundant data items along the coming Ucast cycle is developed. The tree obtained in [14] is called Ucast allocation tree where the depth of the allocation trees corresponds to the number of Ucast cycles, and those leaf nodes in the same level of the allocation tree correspond to those data items to be put in the same Ucast cycle. Fig 3 shows a hierarchical broadcast program with two random Ucast allocation tree where the upper level corresponding to the current Ucast cycle is allocated with two data items(assuming the Ucast size =2) and each other lower levels is distributed along the next coming Ucast2 ,Ucast3 respectively**.** As such, the data items in the upper level must be accommodated first satisfy more transactions than those data items in the lower level.

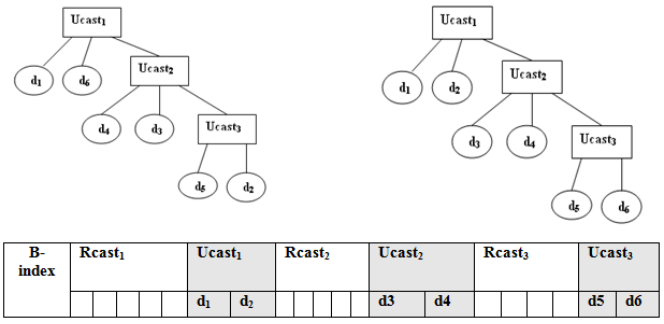

Fig. 3. Updated data item during **$Rcast_1$** = **{d1, d2, d3, d4, d5, d6}** and the size of **$Ucast_1$** = **2**. The abundant updated data items {**d_3, d_4, d_5, d_6**} need to be distributed a long the next coming Ucast cycles to maximize the number of committed transaction within one broadcast cycle.

Note, however, that the algorithm in [14] is designed for the situation where the contention degree of the data items being updated determined at the beginning of broadcast cycle. This consider non practical, as the transactions who wait for this data may need to span multiple broadcast cycle in order to have an opportunity to finish their execution and commit. Clearly, without adapting to the change of update frequencies, the broadcast program determined off-line will unavoidably lead to degraded performance. Thus, with the broadcast programs generated by [14], it is important for the broadcast programs to dynamically adapt to the change of the update frequencies during broadcast cycle so as to retain the performance and increase the freshness of the data items for both read-only and update transactions.

The problem we study can be best understood by the illustrative example in Fig.4. Assume that the data items d_i, $1 \leq i \leq 6$ are of the same size and the number of transaction validating at the server during $Rcast_i$ and $Rcast_{i+1}$ are 5. Denote that the conflicting degree of data item di as $Cd(d_i)$ which is representing the number of validating transactions in which d_i exists in its write set.

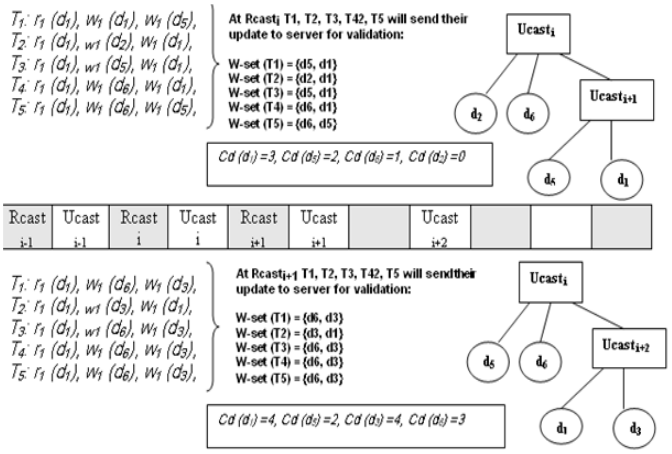

Fig. 4. Illustrative example

Denote the total number of data items being updated in a certain Rcast cycle as n, and a data item as d_i, $1 \leq i \leq n$. The number of Ucast Cycle in a broadcast cycle is S. Recall that $CR(d_k)$ is the conflict ratio of d_k in $Rcast_i$ and estimated as

$$CR(d_k) = \frac{Cd(d_k)}{\sum_{i=1}^{n} Cd(d_i)} \quad \text{.....(1)}$$

Multiplying the conflicting ratio of each data item d_k $CR(d_k)$ by the expected cost of restarting transactions due to conflicting ratio of that data item and summing up the results Same as in [1][19], the expected restarting cost Rc for each data item in the Ucast i is formulated as:

$$Rc = \sum_{x=1}^{N_i} \frac{N_i - x}{N_i} \quad \text{.....(2)}$$

Where Ni us the number of data item allocated in $Ucast_i$. Suppose that the number of updated data items in $Rcast_v$ has j data items, where $j > |Ucast_V|$ and $Uacst_v$ accommodate $d_1, d_2, .. d_{i-1}$, then $d_i, d_{i+1}, ..., d_j$ .. That mean, the updated data items in a $Racst_v$ cannot be accommodated in the corresponding Ucast cycle and have to be transferred to the next Ucast cycle. Unfortunately, the conflicting degree of the updated data items in a certain Rcast with no room in the corresponding Ucast is dynamically changed because of transactions who fetched those data items in Rcasti and finish its execution at Rcast k where k>i. The accumulation of the abundant updated data items come from the latest Rcast cycle need to be distributed properly over the coming Ucast tacking into considerations a dynamic change of their conflicting degree. The conflicting cost *C* of $Ucast_v$ in an allocation tree that has (j-i-1) abundant data items $d_i, d_{i+1} \ldots d_j$ is defined as:

$$C_{i,j} = \sum_{k=1}^{j-i-1} \frac{(j-i-1)-k}{j-i-1} \sum_{\ell=i}^{j} CR(d_\ell) \quad \text{.....(3)}$$

The conflict ratio CR (d_i) is adjusted during Rcast cycles while transactions execution. In essence, the value of $C_{i,j}$ is related to the average restarting cost resulting from the

conflicting degree of abundant data items in Ucast v. Theoretically, generating a broadcast program can be viewed as a partition problem for data items. Given the number of Ucast in a broadcast cycle and the contention degree all data items in the corresponding Rcast, we shall determine the proper set of data items that should be allocated to each Ucast with the purpose of maximizing the number satisfactory request within one broadcast cycle. The problem of distribution updated data items over a K Ucast cycles can be viewed as a discrete maximization problem: Given a list of n data items with their contention degree, partition them into K parts so that the number satisfactory requests of all data items is maximized. As pointed out in previous section, a distribution updated data items over a K Ucast cycles can be represented as an allocation tree with a height of K. Note that the leaf nodes in the same level of the allocation tree correspond to a set of data items to be put in the same Ucast cycle. The value of $C_{i,j}$ is related to the average delay result from the conflicting degree of leaf nodes in level v. This paper investigates the problem of adjusting the data broadcast program at a certain points (i.e. Ucast cycle) to satisfy as much transactions as possible. In order not to distract readers from the main theme of this paper for dynamically adjusting broadcast programs, readers interested in the details of update transactions processing in broadcast data model are referred to [13][16]. Once the change of conflicting degree is larger than the predetermined value, algorithm ADDUcast will be executed to reach the new configuration with minimal conflicting. In accordance with the conflicting degree of data items at Rcast1 and the number of Ucast cycles given, the allocation tree was determined based on the their committed time. It can be seen in Fig 4, at time Rcast $_{i+1}$ allocation tree differ from the one at Rcast $_i$ due to the change of conflicting ratio. Consequently, data items should be moved among levels within the given allocation tree in response to the change of conflicting ratio of data items. Clearly, such movements have an impact on the average response time for all transactions.

IV. Allocation and Adjustments Approach

We devise in this paper an algorithm, referred to as algorithm ADDUcast, to dynamically adjust the broadcast programs by shuffling data items among different levels in the allocation tree. in this paper we propose an algorithm, referred to as algorithm ADDUcast, to dynamically adjust the abundant data allocations by shuffling data items among different Ucasts in order to reflect the contention status therapy increasing the currency and utility of those data items. The process of algorithm ADDUcast can be decomposed into two phases, namely (1) the basement adjustment phase and (2) the smooth adjustment phase. In the basement adjustment phase, algorithm ADDUcast moves data items among the remaining Ucast cycles so as to enable the costs of most Ucasts in the allocation tree to be smaller than or equal to average cost as we describe in algorthim1. Then, for smooth adjustment, algorithm ADDUcast adjusts the data items between consecutives Ucast with the objective of minimizing the total cost of these two consecutive Ucasts.

Since the basement adjustment intends to let the total cost of allocation tree be evenly allocated to all levels, it is possible that some data nodes would move back and forth between neighboring levels.

For execution efficiency, the number of runs for the basement adjustment is limited to be K-1. Basement tuning described in *algorithm 3* is developed to move data items in Ucast i so as to satisfy the purpose of the basement adjustment. By exploiting the basement adjustment, data items are roughly allocated to each Ucast of an allocation tree with the costs of most Ucasts are smaller than or equal to average cost.

The Ucast status table UST is created to record the cost of each Ucast in the allocation tree, and the number of rows in UST is equal to the number of Ucasts in a broadcast cycle (see *algorithm 1* line *7-10*). UST(i).D = UST(i).C −UST(i).P. Where UST(i).P is the cost of data items in Ucast i previously, whereas the value of UST(i).C represent the cost of data items in Ucast i regarding the conflicting probability of the latest Rcast cycle. Also, UST(i).check is a Boolean variable used to indicate whether the basement tuning is performed or not.

1. Algorithm 1: Abundant data Distribution over Ucast cycle ADDUcast
2. Input:
3. K: number of remaining Ucast cycles
4. Usize: number of room in each Ucast cycle.
5. UST: The Ucast status table (UST) with K rows.
6. Output: Minimal conflict Ucast allocation tree up to next Broadcast Cycle
7. /* Construction and initialization of UST content */
8. **for** each row $_i$ in UST **do**
9. **{** UST(i).D=UST(i).C-UST(i).P;
10. UST(i).Checked=false **}**
11. /* Vector Δ has K-1 elements which record the conflicting cost difference between two consecutive Ucasts */
12. For each element i in vector Δ **do**
13. **{** Δ [i]:= |UST(i).C − UST(i + 1).C|
14. Base-checking=0 **}**
15. /* The Basement adjustment phase */
16. Select $r_i \in$ UST such that UST(i).D is maximal;/*Select row of maximal difference */
17. repeat
18. begin
19. **if** (i==1)
20. call Basement Tuning (i, i+1);
21. **else if** (i==k)
22. call Basement Tuning (i, i-1);
23. **else**
24. **{**select the row j where j∈ (i − 1, i + 1) such that ST(j).G is false and Δ [j] is maximal;
25. Call Basement Tuning (i, j); **}**
26. base-checking ++;
27. update UST and Vector Δ accordingly ;
28. select the row i from UST where UST(i).C is maximal and UST(i).Check is false;
29. end
30. **until** base-checking ==K-1;
31. Call smooth adjustment phase

Vector Δ has K-1 elements that record the cost difference between two consecutive Ucast cycles (see *algorithm 1 from line 12 to line 14*). As can be seen in basement adjustment algorthim1, algorithm ADDUcast makes sure that most Ucasts of the allocation tree meet the requirement of the basement adjustment. Since the casual adjustment intends to let the total

cost of allocation tree be evenly allocated to all levels, it is possible that some data nodes would move back and forth between neighboring levels. By take advantage of basement adjustment, data items are roughly allocated to each Ucast of based on corresponding allocation tree with the costs of most Ucasts are smaller than or equal to average cost. For execution efficiency, the number of runs for the basement adjustment is limited to be K-1. Procedure basement tuning is developed to move data items in Ucast i so as to satisfy the purpose of the basement adjustment.

```
1.  Construct a priority queue PQ;
2.  /* A priority queue returns the element with the minimal
    conflicting value */
3.  for each element i in Vector Δ do
4.  Insert Δ [i] into the PQ;
5.  While (PQ is not empty) and Ucast.CurrentSize < USize
6.  begin
7.  remove the element i from PQ;
8.  if (UST(i).C<UST(i+1).C)
9.  /* if there is no movement between Ucast i and Ucast i+1,
    moving equals to -1 */
10. Moving=push up (i, i+1);
11. CurrentSize = CurrentSize +1
12. else
13. Moving=pull down(i, i+1);
14. CurrentSize = CurrentSize -1
15. if (the movement occur: moving <> -1)
16. Update the elements in PQ and UST accordingly;
17. end
```

Then, algorithm ADDUcast employs the smooth adjustment *algorithm 2* to adjust data items between consecutive Ucasts. As can be seen from *line 2 to line 18 of algorithm 2*, consecutive Ucasts are examined on finding potential movements with the purpose of minimizing the total cost of consecutive Ucasts. Specifically, in *line 8 of algorithm 2*, the sequence of performing the smooth tuning is determined by identifying the largest cost difference among those between consecutive Ucasts (i.e., the largest value in Δ). After identifying the consecutive Ucasts (e.g., level I and level i+1) to perform the smooth tuning, one should determine the data movements between these Ucast. Note that there are two kinds of movements, i.e., pushing up and polling down. Judiciously applying these movements is able to reduce the total cost of these two consecutive Ucasts. Clearly, if the cost of Ucast i is smaller than Ucast i+1, we should move data items from level i+1 to level i and vice versa.

From *line 7 to line 18, algorithm 2* adjusts data items in consecutive Ucasts iteratively with the objective of minimizing the total cost until there is no further adjustment required (i.e., queue PQ is empty).

```
1.  Algorithm 3: Basement Tuning (Ucast i, Ucast j)
2.  { sort those data items in Ucast i according their conflict
    probabilities;
3.  if (i<j)
4.  begin
5.  while UST(i).C > Σ_{i=1}^{k} UST(i).C / k do
6.  Move the data item in the rightist side of Ucast i to Ucast j
    and update ST(i).C accordingly;
7.  end
8.  else
9.  begin
10. while UST(i).C > Σ_{i=1}^{k} UST(i).C / k do
11. move the data item in the leftist side of Ucast i to Ucast j
    and update ST(i).C accordingly;
12. end
13. }
```

Once determining the movement's direction among Ucasts, we should determine the number of data items considering the available space in hosting Ucast. Such a number determine based on the conflict reduction gain result from the movement which is limited also by the available space. To do so, we will use the move up and move down procedures as follows: Suppose that data items in each Ucasts are sorted according to the descending order of conflicting probability.

Conflict reduction gain occurs by moving N data items from $Ucast_i$ to $Ucast_{i+1}$ can be estimated by: *CRG-U*(N) = $(C_{i,k} + C_{k+1,j}) - (C_{i,k+p} + C_{k+p+1,j})$ assuming that $Ucast_i$ has k−i+1 data items, d_i, d_{i+1}, ..., d_k and Ucast i+1 has j−k data items, d_{k+1}, d_{k+2}, ..., d_j . The following procedure move-up Fig.5 determine the set of data items in $Ucast_{i+1}$ to be moved upward to $Ucast_i$ in order to maximize the conflict reduction gain between these consecutive Ucasts.

```
Procedure Move-up (Ucasti, Ucasti+1)
{Determine N' such that CRG-U (N') = max{1≤N≤j−k {CRG-U
(N)};
/* determine the maximal value of CRG-U (N)) such that N 1≤N≤ j-k.
*/
if CRG-U (N') > 0
Move data items d_{k+1}, d_{k+2}, ..., d_{k+N"} to Ucasti;
else
N' =-1; /* no movement is performed since there is no cost-effective
movement. */ }
```

Fig. 5. Move-up procedure

In the same way of Move-up procedure, the Move-down procedure in Fig.6 evaluate the set of data items in Ucast i to be moved downward to $Ucast_{i+1}$with the purpose of maximizing the conflict reduction gain of these consecutive Ucasts.

The *CRG-U*(N) for Move-down estimated as = $(C_{i,k} + C_{k+1,j}) - (C_{i,k-p} + C_{k-p+1,j})$ assuming that $Ucast_i$ has k−i+1 data items, d_i, d_{i+1}, ..., d_k and $Ucast_{i+1}$ has j−k data items, d_{k+1}, d_{k+2}, ..., d_j .

```
Procedure Move-down (Ucast_i, Ucast_{i+1})
{Determine N' such that CRG-U (N') = max {1≤N≤ k-i-1 {CRG-
U (N)};
/* determine the maximal value of CRG-U (N)) such that N 1≤N≤ k-
i-1. */
if CRG-U (N') > 0
Move data items d_{k−N"+1}, d_{k−N"+2}, ..., d_k to Ucast_{i+1};
```

else
N' =-1; /* no movement is performed since there is no cost-effective movement. */
}

Fig. 6. Move-down procedure

V. PERFORMANCE EVALUATION

We will investigate the performance of our approach in average response time of the transactions as well as the average restart time of the transactions under different workload and system setting. Table 1 shows the important configuration parameters and their definitions. Read-only and update transactions are simulated based on the ratios defined in the table. Each transaction in the simulation consists of several data access operations and computation operations before or after each data access. A transaction's length is the number of access operations in it. The transaction length is uniformly distributed.

The computation time between two access operations is the operation inter-arrival time. Time between the starts of two consecutive transactions in a simulated system is denoted as transaction inter-arrival time. Each simulation run uses only one client. Large numbers of clients are emulated by using a small average transaction inter-arrival time from one client. Since the amount of abundant data depends on the probability of data conflicts among mobile transactions, we will test our approach when the mobile transactions have different data access patterns. The Zipf distribution with a parameter theta is often used to model non uniform access. It produces access patterns that become increasingly skewed as theta increases. The updates come from the mobile transactions are generated following a Zipf distribution similar to the read access distribution at the client. The update distribution is across the range 1 to UpdateRange. in order to simplify the model, A flat broadcast disk is assumed for selecting data items for broadcast along Rcast cycles.

TABLE I. SIMULATION PARAMETERES

Parameters	Value
decomposition factor	2, 4, 6, 8, 10
Zipf parameter ɵ	0.0–1.0
Database size	10000 items
Ucast size /Rcast Ratio	1/4
Ucast size	20 data items
Transaction size (number of operations)	8,10,12,16
Read operation ratio (for update transactions)	0.5
Read-only transaction ratio	0.7
Average delay between operations	1 (exponentially distributed)
Average delay between transactions	5,10,50 (exponentially distributed)

The number of Ucast and Rcast cycle determined based on the decomposition factor which represent number of Rcast and Ucast cycle in the main broadcast cycle (i.e. when the decomposition factor equal to 2 this mean that the broadcast cycle consists of 2 Ucast and 2 Rcast cycles). Recall that the alternative to the use of dynamic allocation of an abundant data items is to randomly distribute the abundant data along remaining Ucast. A scheme that randomly distributes data items over Ucast cycles, referred to as UD-RAN, is implemented for comparison purposes. To see how the protocols influence the performance, for each configuration the average response time the average restart time are also recorded for UD-RAN and compared with the value obtained under our approach.

Performance of algorithms ADDUcast and UD-RAN is comparatively evaluated It is shown that ADDUcast significantly outperforms RAN due to the proper allocation of data along Ucast that maximize the utilization of data for the active mobile transactions based on the conflict cost heuristic. It is important to see that the advantage of ADDUcast over UD-RAN becomes even more prominent when the skew of updates increases, showing the dynamic allocation and adjustment is even more beneficial when the update is more skewed. See Fig. 7, where the y-axis corresponds to the average response time and the x-axis corresponds to the value of skew factor θ.

In addition, performance of algorithms ADDUcast and UD-RAN is evaluated for different Ucast cycle size. The corresponding results are shown in Fig. 8, where it can be seen that algorithm ADDUcast consistently outperforms algorithm UD-RAN for various values of n. The advantage of ADDUcast over UD-RAN increases as the Ucast cycle size decreases while skew factor is constant. When the size of Ucast is large enough to accommodate all the updated data (i.e.no abndunt data) the performance of ADDUcast are very simller toUD-RAN see Fig. 9.

Fig.7 and Fig.8 show the performance of average response time compared to transaction arrival time among simulated transactions under both UD-RAN and ADDUcast. Each simulation run records the response time of all transactions, which is the time period between a transaction's invoked time and the time it commits. It can include the duration of multiple runs of a transaction if the transaction has ever been aborted and restarted. Average response time is the average response value among all transactions read-only and update transactions without considering the transaction lengths. These figures show the results under different decomposition factor with different Ucast size. It shows that the response time under ADDUcast is less than the UD-RAN as the decomposition factor increase and Ucast size decreases. It also shows that at small Ucast size, the abundant data increase and the allocation of these data items a long remaining Ucast cycles according to ADDUcast is highly decrease the response time of transactions involving in accessing those data items. Actually this is reasonable because the response time is highly affected by the restart rate of transactions as we can see in the next experiments Fig.10, Fig. 11

VI. CONCLUSIONS

The presence of update transactions in wireless broadcast environment make it more difficult to deal with conflicting transactions within the same broadcast cycle. Many researchers tackle the problem of concurrent executions of these

transactions and many of them suggest that broadcast cycle decomposition is a proper technique to increase the concurrency and the system performance as well. As a result, the identities of all the data items which are updated by any transactions in a certain sub cycle we call it Rcast are included in the reserved space after the current broadcast sub-cycle called Ucast. The identities of all the extra data items with no rooms at the current Ucast are included in the next Ucast based on commit sequence of their transactions. Actually, The accumulation of such abundant updated data items is highly affect the system performance and need to be distributed properly over the coming Ucast tacking into considerations a dynamic change of their conflicting degree We explored in this paper the problem of adjusting broadcast programs to cope with the variation of conflicting probability during transactions execution. By exploiting the features of the basement adjustment and the smooth adjustment, we proposed a heuristic based algorithm ADDUcast to adjust abundant data allocations therapy satisfying as much transactions as possible. Performance of algorithm ADDUcast was analyzed and a system simulator was developed to validate our results. It was shown by our simulation results that the allocation achieved by algorithm ADDUcast are highly maintain the freshness of data items with reduced response time. This feature and the efficiency of algorithm ADDUcast justify its practical importance.

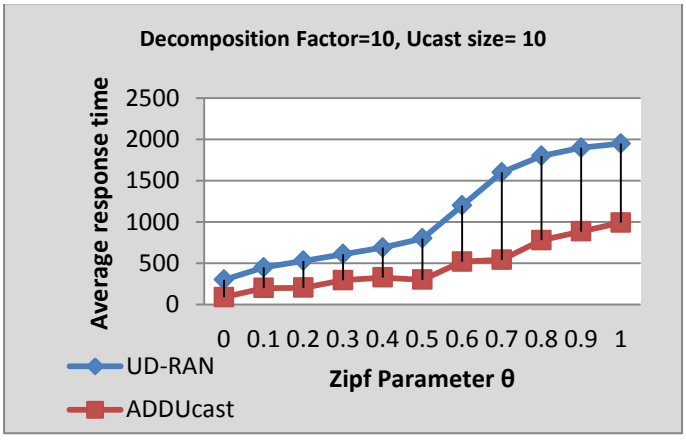

Fig. 7. Average response time with avergae Ucast size

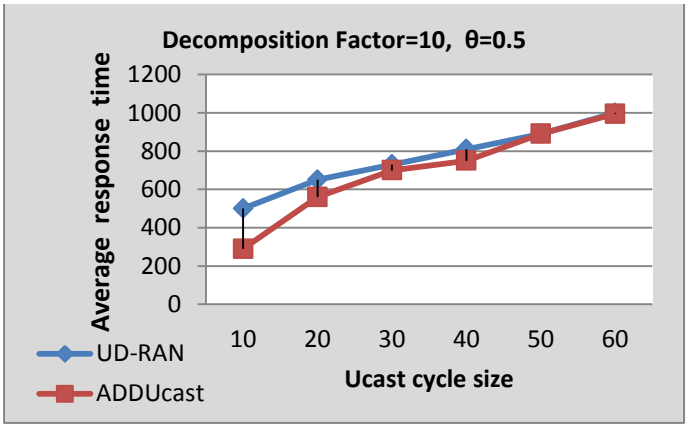

Fig. 8. Average response time under different Ucast cycle size

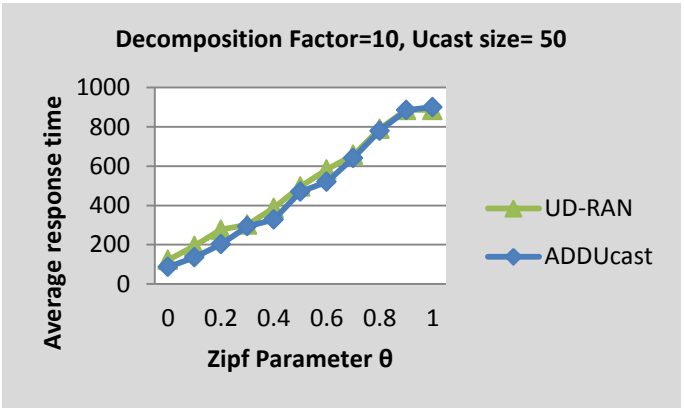

Fig. 9. Average response with large Ucast cycle

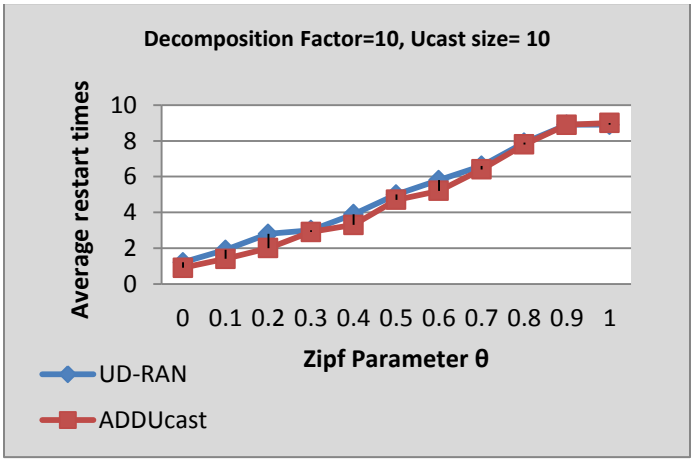

Fig. 10. Average restart times with avergae Ucast size

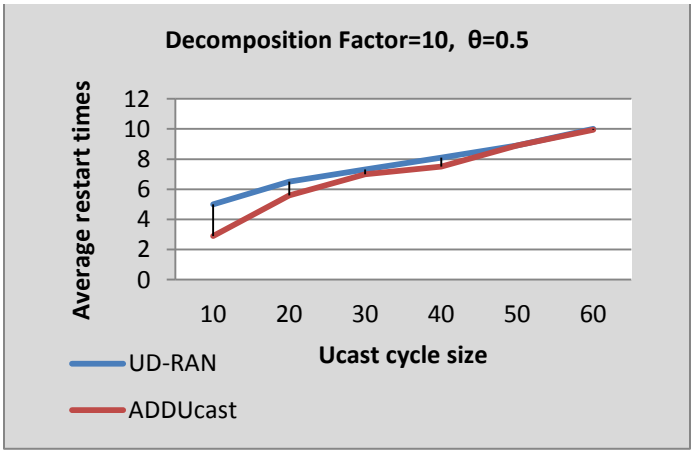

Fig. 11. Average restart times under different Ucast cycle size

ACKNOWLEDGMENT

This research is funded by the Deanship of Research and Graduate Studies in Zarqa Private University /Jordan"

REFERENCES

[1] Acharya, S., Alonso, R., Franklin, M., and Zdonik, S., "Broadcast disks: data management for asymmetric communication environments," Proc. of the ACM SIGMOD Conference, pp.199-210, 1995.

[2] Bernstein, P. A., Hadzilacos, V., and Goodman, N., Concurrency control and recovery in database systems, Addison-Wesley Publishing Company, 1987.

[3] Ozsu, M. T., and Valduriez, P., Principles of distributed database systems, Prentice Hall, 1991.

[4] Lam, K. Y., Au, M. W., and Chan, E., "Broadcast of consistent data to read-only transactions from mobile clients," Proc. of the Second IEEE Workshop on Mobile Computing Systems and Applications, 1999.

[5] Pitoura, E., "Supporting read-only transactions in wireless broadcasting," Proc. of the DEXA98 International Workshop on Mobility in Databases and Distributed Systems, pp. 428-422, 1998.

[6] Pitoura, E., and Chrysanthis, P. K., "Scalable processing of read-only transactions in broadcast push," Proc. Of the 19th IEEE International Conference on Distributed Computing System, 1999.

[7] Lee, SangKeun, Hwang, Chong-Sun, Kitsuregawa, Masaru,. Efficient, energy conserving transaction processing in wireless data broadcast. IEEE Transactions on Knowledge and Data Engineering 18 (9), 1225–1237. September 2006.

[8] Lee, V., Lam, K., Son, S.H., Chan, E, On transaction processing with partial validation and timestamps ordering in mobile broadcast environments. IEEETransactions on Computers 15 (10), 1196–1211 2002.

[9] Cho, H. Concurrency control for read-only client transactions in broadcast disks. IEICE Transactions on Communications E86-B (10), 3114–3122. 2003

[10] Lee, Victor C.S., Lam, Kwok Wa, Kuo, Tei-Wei, Efficient validation of mobile transactions in wireless environments. Journal of Systems and Software, 183–193. 2004.

[11] Lee, SangKeun, Hwang, Chong-Sun, Kitsuregawa, Masaru. Using predeclaration for efficient read-only transaction processing in wireless data broadcast. IEEE Transactions on Knowledge and Data Engineering 15 (6), 1579– 1583. Nov/Dec, 2003.

[12] Imielinski, T., and Viswanathan, S., and Badrimath, B., "Energy efficient indexing on air," Proc. of the ACM SIGMOD Conference, 1994.

[13] Huang, Y., and Lee, Y. H., "Concurrency control protocol for broadcastbased transaction processing and correctness proof," ISCTA PDCS 2001,in press, August 2001.

[14] W.-C. Peng and M.-S. Chen. Dynamic Generation of Data Broadcasting Programs for a Broadcast Disk Array in a Mobile Computing Environment. In Proceeding of the ACM 9th International Conference on Information and Knowledge Management, pages 38—45, November 2000.

[15] Jiun-Long Huang, Ming-Syan Chen: Dynamic Leveling: Adaptive Data Broadcasting in Mobile Computing Environment.MONET 8(4): 355-364 (2003)

[16] Sunggeun Park, Sungwon Jung; An energy-efficient mobile transaction processing method using random back-off in wireless broadcast environments The Journal of Systems and Software vol 82 pp 2012–2022, 2009

[17] Vikas Goel, Ajay Kumar Anil Kumar Ahlawat; A Comparative Study of Energy Efficient Air Indexing Techniques for Uniform Broadcasting International Journal of Computer Applications COMNET-2011

AUTHOR PROFILE

Ahmad Alqerem obtaining a BSc in 1997 from JUSTUniversity and a Masters in computer science from Jordan University in 2002. PhD in mobile computing at Loughborough University, UK in 2008. He is interested in concurrency control for mobile computing environments, particularly transaction processing. He has published several papers in various areas of computer science. After that he was appointed a head of internet technology Depts. Zarka University.