A Novel Scheduling Algorithm for Supporting Periodic Queries in Broadcast Environments

Guohui Li, Quan Zhou, and Jianjun Li

Abstract—Being a proven efficient approach to answering queries that have common data needs, data broadcast has received much attention in the past decade, especially for dynamic and large-scale data dissemination. An important class of emerging data broadcast applications must monitor multiple data items continuously in order to enable data-driven decision making. For such applications, an important problem that must be addressed is how to disseminate data to periodic continuous queries so that all the requests can be satisfied while the bandwidth utilization is minimized. To our best knowledge, the only known work on this topic is the RM-UO algorithm proposed in [27]. However, the RM-UO algorithm simply utilizes the S_r algorithm introduced in [13] to transform the original queries into 2-harmonic tasks, which would lead to a considerable waste of available bandwidth. In this paper, based on the observation that some queries can be merged to save bandwidth consumption, we propose two merging polices namely Multiple Query Merging (MQM) and Redundant Query Merging (RQM), and show that both can lead to notable bandwidth savings. Further, to disseminate data to periodic continuous queries, we implement a unified scheduling algorithm called UM, which combines both MQM and RQM. Extensive experiments have been conducted to compare our UM algorithm with RM-UO, and the results show that UM outperforms RM-UO considerably in terms of wireless bandwidth consumption and query service ratio.

Index Terms—On-demand data broadcast, periodic continuous queries, query merging, real-time scheduling

1 Introduction

With the rapid development of wireless technology, data broadcasting has become a popular data accessing method in wireless communication environments, due to its distinguishing feature of satisfying all pending requests for the same data item with a single transmission. Recently, mobile data services with real-time support has received particular attention in the research community. A time-critical request is associated with a deadline imposed by either the application or the user, the result of a request is useful only if all the required data items can be received before the deadline. For such kind of applications, some real-time broadcast scheduling algorithms have been proposed [14], [6], [22], which take queries' deadlines into account. But unfortunately, almost all the existing real-time broadcast scheduling algorithms only support single data item [2] or one-shot queries [7], [16], and thus can be of low performance, or even not applicable in some practical applications. For example, a stock investor wants to refresh his stock information once every minute to decide whether or not to trade his stock. Since the stocker wants to get continuous services to guarantee that a relatively accurate stock price can be obtained in a limited time interval, existing data broadcasting methods which do not consider the time constraints of queries cannot support such periodic query processing efficiently. Therefore, it is essential to design new scheduling algorithms to process periodic queries with real-time requirement.

To our best knowledge, the work in [27] presented the first and only algorithm (RM-UO) which achieves timing predictability for admitted periodic continuous queries. This algorithm is designed to broadcast data items and supports processing of continuous queries with fixed periods. Specifically, it assumes that mobile clients send their data requests to the server with implicit deadlines, which are equal to their corresponding periods.

Review of RM-UO. The RM-UO algorithm consists of the following three steps.

- Data set division and Distance-Constrained (DC) tasks generation.
- Transform the DC tasks into two-harmonic periodic tasks, by using the S_r algorithm proposed in [13].
- Schedule the generated two-harmonic periodic task set by the Rate Monotonic (RM) [20] algorithm.

In the first step, the data items which one query needs to access are divided into two subsets: the *unique* set and the *shared* set. Each data item in the *unique* set of a query has the shortest period among all the queries requiring it. It should be pointed out that, the data items in the *unique* sets are different. After the data set division operation, the DC task set is generated which only includes the *unique* sets, and the *shared* sets are ignored, since the data items in the *shared* set of a query can be shared from the *unique* set of other queries.

In the second step, the S_r algorithm is invoked to generate a two-harmonic task set from the DC task set [13]. We call a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ 2-harmonic if the period of any task τ_k, denoted by T_k, is $2^x (x \in 0 \cup N^+)$ times of other task periods which are not greater than T_k. For example, the task set with periods $T_1 = 3$, $T_2 = 6$, $T_3 = 6$, $T_4 = 24$ is

- *The authors are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei, P.R. China.*
  E-mail: {guohuili, jianjunli}@hust.edu.cn, zhouquanwh@gmail.com.

Manuscript received 4 Aug. 2014; revised 8 Jan. 2015; accepted 19 Jan. 2015. Date of publication 5 Feb. 2015; date of current version 30 Oct. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMC.2015.2398417

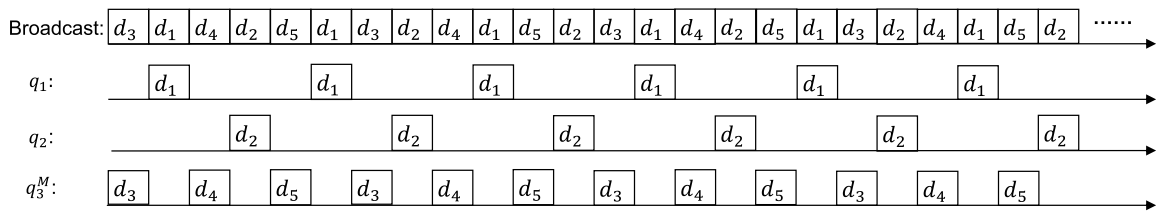


Fig. 1. A feasible schedule for the simple example in Section 1.

two-harmonic. Detail of the S_r algorithm is described as follows: 1) First, for each DC task τ_i, S_r computes $\gamma_i = T_i/2^{\lceil \log_2 \frac{T_i}{T_1} \rceil}$, where T_1 represents the shortest period of all the tasks; 2) Based on each different γ_i (with duplicates removed), the period of each DC task τ_k is transformed into $\gamma_i \cdot 2^x$, where $x = \lfloor \log_2 \frac{T_k}{\gamma_i} \rfloor$. After the transformation, the total utilization of the tasks is calculated. Finally, the transformed task set with the minimum utilization is selected to be the final two-harmonic task set.

In the third step, the RM-UO algorithm calculates the bandwidth consumption of the two-harmonic task set (generated in the second step) first. If the bandwidth consumption is no larger than 100 percent, then by the result in [13], the two-harmonic task set is schedulable under RM. Otherwise, RM-UO fails to handle this task set and thus cannot guarantee to provide services for the query set.

Shortcoming of RM-UO. Although RM-UO is an effective algorithm to schedule periodic queries to satisfy their time constraint, there is still much room for improvement. Specifically, we observe that some task sets which cannot be scheduled under RM-UO are actually schedulable. The main reason lies in that, in RM-UO, the transformation from DC tasks to two-harmonic tasks would result in considerable waste of the broadcasting bandwidth.

Consider a simple example in which there are five queries with their periods to be 4, 5, 6, 6 and 8, respectively. The data items they need to access are d_1, d_2, d_3, d_4 and d_5 respectively. Under the RM-UO algorithm, each original query is considered independently and transformed into a two-harmonic task. Specifically, these five queries will be transformed to five two-harmonic periodic tasks each with a new period 4, 4, 4, 4 and 8, respectively. Obviously, the server will refuse to provide services for these five queries, since the total bandwidth utilization of the first four queries $(4 \cdot \frac{1}{4})$ has reached the upper bound 100 percent. But actually, under the multiple query merging (MQM) merging policy which will be introduced in this paper, the last three queries with periods 6, 6 and 8 can be merged into one virtual task τ_3^M with period 2. Then, the five queries can be processed by the server, since the total bandwidth consumption is $\frac{1}{4} + \frac{1}{4} + \frac{3}{6} = 1$. As an illustration, Fig. 1 depicts the feasible broadcast sequence for the five queries.

The above example indicates that, for a query set, sometimes a few queries can be merged to save the consumption of the wireless broadcast bandwidth while maintaining the schedulability of the query set. Moreover, in RM-UO, when transforming the DC-tasks into two-harmonic ones, the periods of some queries are shortened. But, sometimes there is no need to broadcast data items with such a shortened period, which means the period transforming operation would also lead to bandwidth waste. Inspired by these two observations, we propose two methods namely Multiple Query Merging and Redundant Query Merging (RQM) to merge queries, with the objective of saving bandwidth utilization and hence accommodating more queries. We also formally prove the correctness of our algorithms on scheduling the query sets. Further, to disseminate data to periodic continuous queries, we implement a unified scheduling algorithm called UM which combines both MQM and RQM. In summary, the main contributions of this paper can be summarized as follows.

- Based on the observation that some queries can be merged in the transformation step, we propose the Multiple Query Merging policy. We also propose a scheduling algorithm MQM-UO to schedule the query set derived by MQM, and theoretically prove the correctness of it on scheduling the queries to get desired data items before their deadlines.
- Based on the observation that using the S_r algorithm to transform the DC-tasks to two-harmonic tasks would lead to unnecessary bandwidth waste, we further propose another merging policy Redundant Query Merging. We also propose a scheduling algorithm RQM-UO to schedule the query set derived by RQM, and theoretically prove the correctness of it on scheduling the queries to get desired data items before their deadlines.
- We implement a unified scheduling algorithm namely UM which combines both MQM and RQM. Extensive experiments have been conducted to compare UM versus RM-UO, and the experimental results demonstrate that UM can result in significant performance improvement compared to RM-UO, in terms of service ratio and bandwidth consumption.

The remainder of this paper is organized as follows: Section 2 gives the system model along with some basic assumptions. In Section 3, we introduce two merging policies, MQM and RQM, based on which we present the UM algorithm in Section 4. Experimental results are shown in Section 5. Section 6 reviews some related work. Finally, the conclusion of this paper with a brief discussion on future work is given in Section 7.

2 Model and Assumptions

2.1 System Model

The on-demand broadcasting environment we considered in this paper is shown in Fig. 2. The server schedules the broadcasting data based on the queries issued from clients. When a mobile client wants to get service, it sends requested data items and the deadline of the query to the server and then keeps on monitoring the broadcasting

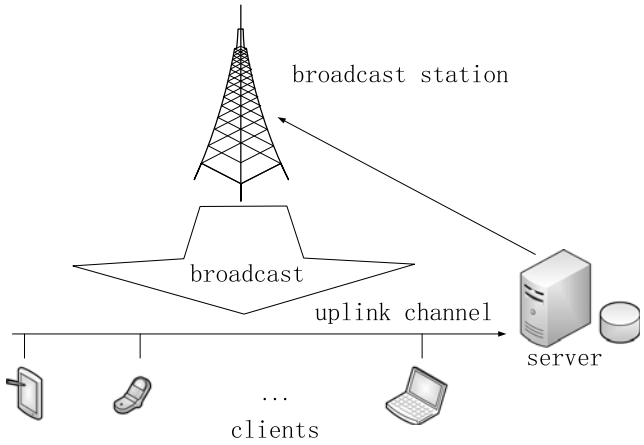


Fig. 2. System model.

channel. Once the server receives a query, it checks the feasibility of providing service for this query. If the server decides to provide service for the query, the requested data items of the query are added into the broadcast channel. Note that, the same as [27], in this paper, queries are assumed to be periodic and continuous. Moreover, our research focuses on the multi-receiver, single-channel environments. So, all the mobile clients monitor the same channel and there is no channel-changing processing.

To facilitate discussion, we assume that all data items are of the same size. Therefore, the time to transfer all data items from the server to a mobile client is the same and the time needed to broadcast one data item is called a *time unit*. In the rest of this paper, time length is measured in the number of *time units*. Note that even though it is assumed that the sizes of all data items are equal, the algorithms proposed in this paper can be easily adapted to the situation where data items are of different sizes.

2.2 Notations and Assumptions

We assume that there is a data set $D = \{d_i | 1 \leq i \leq m\}$ maintained in the broadcast server, and there is a set of N queries $\mathcal{T} = \{q_1, \ldots, q_N\}$, where q_i represents the ith query. Each query q_i is characterized by two parameters (T_i', S_i), where T_i' is q_i's period and S_i indicates the set of data items q_i wants to access. $\mathcal{T}$ is sorted in ascending order of queries period, i.e., if $i < j$, then $T_i' \leq T_j'$. The same as in [27], the set of data items a query q_i requests is partitioned into two sets: the *unique* set and the *shared* set. The *unique* set of q_i consists of the data items which are accessed by q_i and q_i is the front-most one among the queries requiring them. The *shared* set of q_i consists of the rest data items accessed by q_i. Note that, each data item only belongs to the *unique* set of the query which has the shortest period among all the queries accessing it. For example, suppose there is a task set $\{q_1, q_2\}$ which has been sorted in non-decreasing order of there periods, q_1 requires d_1 and d_2, while q_2 requires d_2 and d_3. Based on the description above, d_1 and d_2 should be set in the *unique* set of q_1, since q_1 is the front-most one among the queries requiring them. Due to the same reason, d_3 should be set in the *unique* set of q_2. Furthermore, we can get that the *shared* set of q_1 is $\emptyset$, while the *shared* set of q_2 is $\{d_2\}$, since d_2 has been set in the *unique* set of q_1. Note that, since query q_i can share the data items in its *shared* set from other queries, q_i is transformed into a DC task which does not include the data items in its *shared* set. Obviously, the *unique* set of a DC task may contain multiple data items. In this case, the DC task is further transformed into multiple tasks each having the same period as the original task but only one single data in its *unique* set. In this way, a new DC task set is obtained and the ith DC task in the newly generated task set is represented by τ_i, with its *unique* set to be $\{d_i\}$. In order to distinguish the periods of the original queries from that of the tasks in the newly generated DC task set, we use T_i to represent the period of τ_i. Note here T_i is the same as the period of the query from which τ_i is generated. To facilitate discussion, in the remaining of this paper, we assume a DC-task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ has been generated with only one data item d_i in each task τ_i's *unique* set. Moreover, in the following discussion, when we talking about that a task set is schedulable, we mean all the queries in the corresponding query set can get desired data items before their deadlines.

3 Merging Queries

In this section, we present our two merge policies MQM and RQM in Sections 3.1 and 3.2, respectively.

3.1 The MQM Policy

As pointed out in Section 1, RM-UO may lead to waste of bandwidth consumption when transforming DC tasks into two-harmonic tasks. In this section, we propose a novel method called Multiple Queries Merging, which can merge multiple tasks to save the bandwidth, but without affecting any time constraints of the queries. The same as in [27], we also need to first transform the DC task set into a two-harmonic periodic task set. To distinguish, we use τ_i^* to represent the transformed result of the ith DC task τ_i. For any DC task τ_i, after the transform operation, its period will be shortened to $T_i^* = \gamma \cdot 2^x$, where x is an integer and $\frac{T_1}{2} < \gamma \leq T_1$. Note here γ is called the *key unit* for the two-harmonic task set, and is calculated by the method described in Section 1. It should be pointed out that transforming the DC-task set into a two-harmonic one is only an intermediate step in our method, we still need to check whether some tasks can be merged and study how to merge them. Before detailing the merging policy of our method, we first give three definitions.

Definition 1. *For each DC task τ_i, if the period of the corresponding two-harmonic task after transformation is equal to $\gamma \cdot 2^x$, then the value of 2^x, denoted by k_i, is defined as the coefficient of τ_i.*

For example, suppose there is a DC task set and the *key unit* (γ) of the corresponding two-harmonic task set is 3. Then, for task τ_i whose period T_i satisfies $3 \cdot 2^2 < T_i < 3 \cdot 2^3$, we know the period of τ_i^* must be $3 \cdot 2^2$, thus τ_i's coefficient k_i is 2^2.

Definition 2. *For each DC task τ_i with period T_i and its corresponding two-harmonic task τ_i^* with period T_i^*, we define the set which consists of all the tasks with indexes no smaller than i and periods after transformation equal to T_i^*, as the Possible Merging task set of τ_i, denoted by PM_i. More formally,*

$PM_i = \{\tau_j | j \geq i \ T_j^* = T_i^*\}$. The number of the elements in PM_i is denoted by N_i.

Definition 3. *For a DC task τ_i, let $\beta = 2^{\lfloor \log_2 N_i \rfloor}$, $\alpha = \lfloor \frac{\beta \cdot T_i}{T_i^*} \rfloor$, and x be the greatest common divisor of α and β. If $\alpha > \beta$, then we define τ_i as a $(z, 2^y)$-task, where $z = \frac{\alpha}{x}$ and $2^y = \frac{\beta}{x}$.*

We will detail the reason why we set β, α, z and 2^y this way later in this section. Below we introduce a lemma to be used in the merging policy.

Lemma 1. *If there exists a $(z, 2^y)$-task τ_i, then there are at least 2^y tasks with their indexes no smaller than i and periods after transformation (to two-harmonic tasks) equal to T_i^*.*

Proof. Based on Definition 3, there are $2^y = \frac{\beta}{x} \leq \beta$ and $\beta = 2^{\lfloor \log_2 N_i \rfloor}$. So, we have $2^y \leq 2^{\lfloor \log_2 N_i \rfloor} \leq N_i$. Based on Definition 2, N_i is the number of the tasks whose indexes are no smaller than i and periods after transformation equal to T_i^*. So there must be at least 2^y tasks with their indexes no smaller than i and periods after transformation equal to T_i^*, the lemma thus follows. □

We are now ready to present our MQM policy. First, the same as RM-UO, we transform the DC task set into a two-harmonic task set. Then, for each DC task $\tau_i (1 \leq i \leq n)$, we calculate N_i based on Definition 2. Next, we traverse the DC task set to find the $(z, 2^y)$-tasks. Once a $(z, 2^y)$-task τ_i is found, we have the following two cases to be considered:

- If there are no less than $(z-1)$ tasks located behind τ_i, then starting with τ_i, we merge the z continuous tasks to obtain a new task τ_i^M with period $T_i^M = \gamma \cdot \frac{k_i}{2^y}$;
- Otherwise, if the number of the tasks located behind τ_i is less than $(z-1)$, then all the tasks starting with τ_i are merged into one task τ_i^M with period $T_i^M = \gamma \cdot \frac{k_i}{2^y}$.

We call the above merging policy MQM. Note that based on Lemma 1, there are at least $(2^y - 1)$ tasks located behind τ_i. For simplicity, we use m_i to denote the number of the tasks to be merged with task τ_i, it is then clear to see that $2^y \leq m_i \leq z$. To record the data items which are accessed by the tasks being merged, we assume the virtual task τ_i^M requires a virtual data item set VD_i, where $VD_i = \{d_i, d_{i+1}, \ldots, d_{i+m_i-1}\}$.

After conducting the merging operation by MQM, the other problem is how to broadcast the data items so that all the queries can be satisfied.

Algorithm MQM-UO. In [27], RM is used to schedule the generated two-harmonic task set. In this work, we can also obtain a two-harmonic task set after the merging operation, but the schedule is different from that in RM-UO. Specifically, we first invoke the RM algorithm to generate a schedule for the two-harmonic task set in a hyper period. Then, when determining the data items to be broadcast, we need to consider the following two cases: 1) For the virtual task which is generated by the MQM policy, the data items which are in the virtual data item set will be broadcast sequentially and periodically; 2) For the tasks which are derived by transforming the DC tasks directly, the same as RM-UO, the corresponding data item accessed by the task will be broadcast. To help understand the first case, consider that there is a virtual task τ_i^M, which is the result of merging $\{\tau_i, \tau_{i+1}, \ldots, \tau_{i+m_i-1}\}$ by executing the MQM policy. Then, when τ_i^M is selected to execute based on the task schedule, the broadcasting server will sequentially and periodically choose a data item from $VD_i = \{d_i, d_{i+1}, \ldots, d_{i+m_i-1}\}$ to broadcast. Since we also only broadcast the data items in the *unique* set, similar to [27], we call this scheduling algorithm MQM-UO. The pseudo-code of MQM-UO is shown in Algorithm 1.

Algorithm 1. MQM-UO

Input: a two-harmonic task set, the current time t
Output: the data item to be broadcast at t

```
1 begin
2   Utilize the RM algorithm to derive the task schedule for the
    two-harmonic task set within one hyper period;
3   Find the task to be executed at time t;
4   if the task to be executed is a virtual task then
5     Sequentially and periodically return a data item in the
      virtual data item set accessed by the virtual task;
6   else
7     Return the data item accessed by the task;
8 end
```

An astute reader may concern whether the queries can get their data on time after merging some tasks by the MQM policy. The answer is affirmative, as shown in the following theorem.

Theorem 1. *Given a periodic continuous query set $\{q_1, q_2, \ldots, q_m\}$ and its corresponding DC task set $\{\tau_1, \tau_2, \ldots, \tau_n\}$, if $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{\frac{1}{n}} - 1)$, then this periodic continuous query set $\{q_1, q_2, \ldots, q_n\}$ is schedulable under* MQM-UO.

Proof. Based on [27], given a periodic continuous query set $\{q_1, q_2, \ldots, q_m\}$, if its corresponding DC task set $\{\tau_1, \tau_2, \ldots, \tau_n\}$ satisfies the following condition:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{\frac{1}{n}} - 1)$$

then this periodic continuous query set is schedulable under RM-UO. Now we need to prove the query set is also MQM-UO schedulable. Clearly, if there does not exist any task which can be merged by the MQM policy, then MQM-UO is the same as RM-UO, and the claim holds obviously.

Now we consider the case that some tasks can be merged by MQM. Without loss of generality, suppose there is a $(z, 2^y)$-task τ_i. Then based on Definition 3, we know τ_i's period T_i satisfies the following condition:

$$T_i^* \cdot \frac{z}{2^y} \leq T_i < 2 \cdot T_i^*. \quad (1)$$

Note that, although τ_i's period is shortened to $T_i^* = \gamma \cdot k_i$, its actual period remains unchanged and the mobile client still requires d_i every T_i time units. As mentioned above, by the MQM policy, the period of τ_i^M is $T_i^M = \gamma \cdot \frac{k_i}{2^y}$. So, if we broadcast $\{d_i, d_{i+1}, \ldots, d_{i+m_i-1}\}$ cyclically with period T_i^M, each of them will be broadcast

with a fixed period $\gamma \cdot \frac{k_i}{2^y} \cdot m_i = \gamma \cdot k_i \cdot \frac{m_i}{2^y}$. Obviously, all these tasks τ_i, $\tau_{i+1}, \ldots, \tau_{i+m_i-1}$ can obtain their *unique*-data, since by $2^y \le m_i \le z$ and Formula (1), there is $\gamma \cdot k_i \cdot \frac{m_i}{2^y} \le \gamma \cdot k_i \cdot \frac{z}{2^y} \le T_i \le T_{i+1} \le \cdots \le T_{i+m_i-1}$.

Next we argue that other tasks can also share data from these m_i tasks correctly. First, we prove that for any task τ_k with d_i in its *shared* set, τ_k can share d_i in each of its period. Based on the data item division method of a query, only the tasks whose periods are not shorter than T_i can hold d_i in their *shared* sets. So we have $T_k \ge T_i \ge \gamma \cdot k_i \cdot \frac{z}{2^y} \ge \gamma \cdot k_i \cdot \frac{m_i}{2^y}$. Note that, under MQM-UO, d_i is broadcast with period $\gamma \cdot k_i \cdot \frac{m_i}{2^y}$. So, τ_k can get d_i in each of its period. Following a same way, we can prove that the tasks with d_{i+1}, d_{i+2}, ..., or d_{i+m_i-1} in their *shared* set can also get these data items in their periods respectively. The proof is thus finished. □

The following theorem formally shows the improvement of MQM-UO as compared to RM-UO, in terms of bandwidth consumption.

Theorem 2. *Suppose the* MQM *policy is executed on m_i continuous tasks starting with a $(z, 2^y)$-task τ_i, then the bandwidth saving of this operation is* $\frac{1}{T^*_{i+2^y}} + \frac{1}{T^*_{i+2^y+1}} + \cdots + \frac{1}{T^*_{i+m_i-1}}$.

Proof. Under RM-UO, the m_i continuous tasks consume the bandwidth in the size of $\frac{1}{T^*_i} + \frac{1}{T^*_{i+1}} + \cdots + \frac{1}{T^*_{i+m_i-1}}$. Since τ_i is a $(z, 2^y)$-task, by Lemma 1, we know there are at least $(2^y - 1)$ tasks which are sorted behind τ_i will be transformed into two-harmonic tasks with period T^*_i. So, we have $\frac{1}{T^*_i} + \frac{1}{T^*_{i+1}} + \cdots + \frac{1}{T^*_{i+m_i-1}} = \frac{1}{T^*_i} + \frac{1}{T^*_{i+1}} + \cdots + \frac{1}{T^*_{i+2^y-1}} + \frac{1}{T^*_{i+2^y}} + \cdots + \frac{1}{T^*_{i+m_i-1}} = \frac{2^y}{T^*_i} + \frac{1}{T^*_{i+2^y}} + \cdots + \frac{1}{T^*_{i+m_i-1}} = \frac{2^y}{\gamma \cdot k_i} + \frac{1}{T^*_{i+2^y}} + \cdots + \frac{1}{T^*_{i+m_i-1}}$. Under MQM-UO, by executing the MQM policy on the tasks from τ_i to τ_{i+m_i-1}, the newly generated virtual task has a bandwidth consumption of $\frac{2^y}{\gamma \cdot k_i}$. So, the bandwidth saving is $\frac{1}{T^*_{i+2^y}} + \frac{1}{T^*_{i+2^y+1}} + \cdots + \frac{1}{T^*_{i+m_i-1}}$. □

Reason for setting β, α, z and 2^y. Now, we explain the reason why we set $\beta = 2^{\lfloor \log_2 N_i \rfloor}$, $\alpha = \lfloor \frac{\beta \cdot T_i}{T^*_i} \rfloor$, $2^y = \frac{\beta}{x}$ and $z = \frac{\alpha}{x}$ in Definition 3. First, we consider the case where $m_i = z$. Without loss of generality, suppose τ_i is a $(z, 2^y)$-task, since $m_i = z$, we know z is the number of the tasks merged into τ^M_i, and $T^M_i = \frac{T^*_i}{2^y}$ is the period of τ^M_i. Someone may concern why the period of the new task τ^M_i must be in the form of $\frac{T^*_i}{2^k} (k \in N)$. This is because we need to guarantee that the new task set after conducing the MQM policy is still a two-harmonic one. Since the data items in VD_i are broadcast one after another in sequence, we know the broadcasting period of each data item is equal to $z \cdot T^M_i = z \cdot \frac{T^*_i}{2^y}$. Note that, we must guarantee that the broadcasting period of each item is not larger than the original period of its corresponding DC task. So, there is $z \cdot \frac{T^*_i}{2^y} \le T_i$, which means $z \le \frac{T_i \cdot 2^y}{T^*_i}$. Since z is the total number (an integer) of the tasks which can be merged, in order to select a bigger z to save more bandwidth consumption, we have $z = \lfloor \frac{2^y \cdot T_i}{T^*_i} \rfloor$. As described above, under the RM-UO algorithm, the total bandwidth consumption of $\{\tau_i, \ldots, \tau_{i+z-1}\}$ is $\sum_{j=i}^{i+z-1} \frac{1}{T^*_j}$. Under the MQM-UO algorithm, the bandwidth consumption of τ^M_i is $\frac{2^y}{T^*_i}$. In order to guarantee

$$\frac{2^y}{T^*_i} \le \sum_{j=i}^{i+z-1} \frac{1}{T^*_j} = N_i \cdot \frac{1}{T^*_i} + \sum_{j=i+N_i}^{i+z-1} \frac{1}{T^*_j},$$

we must have $2^y \le N_i$. Moreover, it should be pointed out that, during one MQM execution, the bandwidth saving is determined by the total consumption of the two-harmonic tasks ($\{\tau^*_{i+2^y}, \ldots, \tau^*_{i+z-1}\}$) which is generated by the transformation of the last $z - 2^y$ tasks. Since $z - 2^y$ is a non-decreasing function with respect to 2^y, in order to maximize the value of $z - 2^y$, we need to choose the largest 2^y which satisfies $2^y \le N_i$. Consequently, we have $2^y = 2^{\lfloor \log_2 N_i \rfloor}$.

Note that the above analysis focuses on the case where $m_i = z$. However, it is also possible that $m_i < z$. In this case, if $m_i > N_i$, then MQM-UO can still save some broadcast bandwidth. But, if $m_i = N_i$ and we still set $z = \lfloor \frac{2^y \cdot T_i}{T^*_i} \rfloor$ and $2^y = 2^{\lfloor \log_2 N_i \rfloor}$, then there is not any bandwidth savings by merging the m_i tasks with MQM, since $\frac{2^y}{T^*_i} = \sum_{j=i}^{i+m_i-1} \frac{1}{T^*_j}$. In order to get bandwidth savings by MQM, we first select an x which is the greatest common divisor of $\lfloor \frac{2^y \cdot T_i}{T^*_i} \rfloor$ and $2^{\lfloor \log_2 N_i \rfloor}$, and then set $2^y = 2^{\lfloor \log_2 N_i \rfloor}/x$ and $z = \lfloor \frac{2^y \cdot T_i}{T^*_i} \rfloor / x$ to make the new number of tasks which are to be merged into one task is bigger than 2^y in most cases, with the objective of saving the bandwidth consumption as much as possible. In summary, we have $\beta = 2^{\lfloor \log_2 N_i \rfloor}$, $\alpha = \lfloor \frac{\beta \cdot T_i}{T^*_i} \rfloor$, $2^y = \frac{\beta}{x}$ and $z = \frac{\alpha}{x}$.

We now give an example to illustrate MQM-UO.

Example 1. Suppose there are eight queries and their corresponding DC tasks are $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8\}$, with their periods respectively to be $\{4, 4, 4, 16, 24, 24, 24, 24\}$. By the RM-UO algorithm, the periods of the DC tasks will be transformed (shortened) to $\{4, 4, 4, 16, 16, 16, 16, 16\}$ with $\gamma = 4$. Obviously, the total bandwidth consumption is $\frac{17}{16} > 100\%$, which means RM-UO fails to handle this query set. Different from RM-UO, we do not schedule these two-harmonic tasks directly, but merge some DC tasks by the MQM policy to generate a new two-harmonic task set. Based on Definition 3, we find that τ_5, τ_6 and τ_7 are $(3, 2^1)$-tasks and their coefficients are 4. So, instead of shortening the periods of them to 16, we merge τ_5, τ_6 and τ_7 to obtain a new task τ^M_5, with period $\frac{\gamma \cdot k_5}{2} = 8$. After this merging operation, the DC task set is changed to $\{\tau^*_1, \tau^*_2, \tau^*_3, \tau^M_5, \tau^*_4, \tau^*_8\}$ with periods $\{4, 4, 4, 8, 16, 16\}$. As can be seen, this new two-harmonic task set's bandwidth consumption is 100 percent, which means it can be scheduled by MQM-UO. Fig. 3 shows the feasible schedule for the newly generated task set.

3.2 The RQM Policy

In this section, we introduce another method which can be used to further reduce the waste of the bandwidth consumption. We start by giving an example as follows.

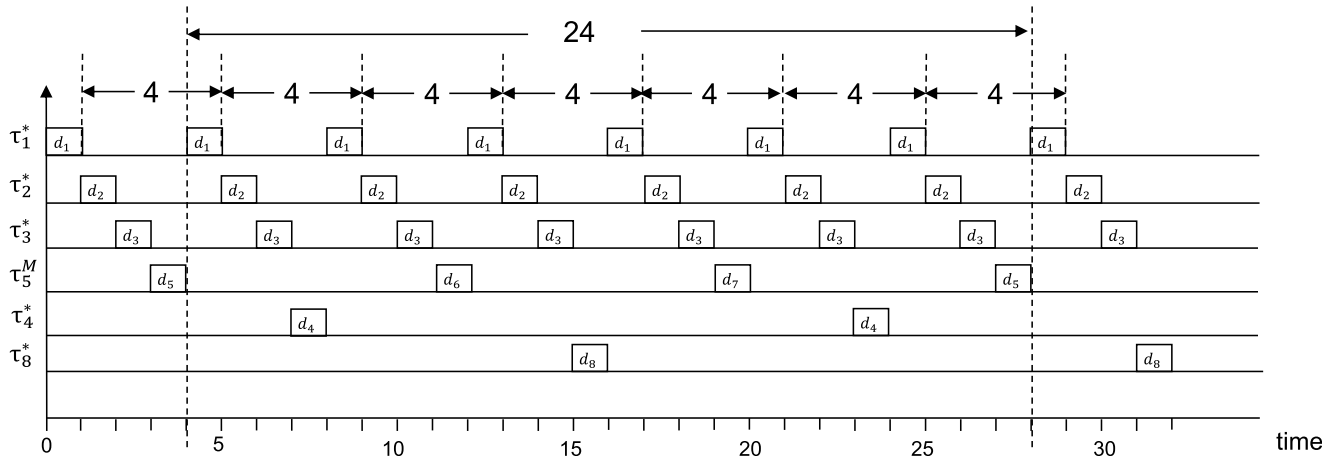


Fig. 3. Schedule of the queries in Example 1 by MQM-UO.

Example 2. In Example 1, suppose a new task τ_9 with period 48 is added, i.e., the new task set τ is $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8, \tau_9\}$, with their periods to be $\{4, 4, 4, 16, 24, 24, 24, 24, 48\}$. By the RM-UO algorithm, the periods of the DC tasks will be transformed to $\{4, 4, 4, 16, 16, 16, 16, 16, 32\}$. It is clear that RM-UO still cannot handle this query set since the total bandwidth consumption is $\frac{35}{32} > 100\%$. By MQM-UO, the DC task set will be changed to $\{\tau_1^*, \tau_2^*, \tau_3^*, \tau_5^M, \tau_4^*, \tau_8^*, \tau_9^*\}$ with periods $\{4, 4, 4, 8, 16, 16, 32\}$. Since the bandwidth consumption is $\frac{33}{32} > 100\%$, it means MQM-UO also cannot handle this query set.

At first glance, it seems that τ in Example 2 cannot be scheduled to get services. But actually, if we schedule it in a more wise way, these nine tasks can still get their services simultaneously. We first introduce a useful lemma.

Lemma 2. *Suppose there is task τ_i with d_i in its* unique *set, and τ_i's period T_i will be shortened to T_i^* under the RM-UO algorithm, then in the broadcast sequence of RM-UO, there must exist a τ_i's instance which can get d_i twice , but at most twice, in a time-interval of T_i.*

Proof. Without loss of generality, suppose d_i appears in the broadcast channel at time unit m for the first time. Then under RM-UO, d_i must be broadcast at each time unit $m + k \cdot T_i^*$ where k is a non-negative integer. Let $x = \left\lfloor \frac{m}{T_i - T_i^*} \right\rfloor$, since τ_i's period T_i will be shortened to T_i^*, i.e., $T_i > T_i^*$, it is clear that x is a non-negative integer. Then we know under the RM-UO algorithm, d_i must be broadcast at time unit $m + x \cdot T_i^*$ and $m + (x+1) \cdot T_i^*$. By $x = \left\lfloor \frac{m}{T_i - T_i^*} \right\rfloor$, we have $x \cdot T_i \leq m + x \cdot T_i^* < m + (x+1) \cdot T_i^* < (x+1) \cdot T_i$. This means the $(x+1)$th instance of τ_i can get d_i twice from the broadcast channel, and the two broadcasting time units are $m + x \cdot T_i^*$ and $m + (x+1) \cdot T_i^*$, respectively. Moreover, since for any instance of τ_i, there is $T_i^* < T_i < 2 * T_i^*$, we know τ_i cannot get d_i more than twice. In summary, the claim is proved. □

Obviously, it is a waste of bandwidth to broadcast data item d_i twice during the execution of any instance of τ_i in a time length of T_i. In the following discussion, the second broadcasting of d_i during the execution of an instance of task τ_i as described in the above lemma is called a *potentially redundant* broadcast of d_i. Based on this definition, we can derive that a *potentially redundant* broadcast of d_i appears at time t, if and only if the following equation holds:

$$\left\lfloor \frac{t}{T_i} \right\rfloor = \left\lfloor \frac{t - T_i^*}{T_i} \right\rfloor. \tag{2}$$

Note that, for such a *potentially redundant* broadcasting of d_i, sometimes it can be removed from the broadcast channel to save the broadcasting bandwidth.

Now we give a lemma to specify when we can remove a *potentially redundant* broadcasting of d_i, so as to save the bandwidth consumption. We first give a definition.

Definition 4. *Assume there are two adjacent DC tasks τ_i and τ_{i+1}, with their coefficients to be k_i and k_{i+1}, respectively. If $k_i < k_{i+1}$, i.e., $T_i^* < T_{i+1}^*$ (or $2T_i^* \leq T_{i+1}^*$ due to that they are all two-harmonic periods), and the period of τ_i will be shortened under RM-UO, then τ_i is called an R-task.*

For instance, in Example 2, $k_8 = 4 (T_8^* = 16)$, $k_9 = 8 (T_9^* = 32)$ and the period of τ_8 will be shortened from 24 to 16 under RM-UO. So, Based on Definition 4, τ_8 is an R-task.

Lemma 3. *If τ_i is an R-task with its* unique *set to be $\{d_i\}$, then the potentially* redundant *broadcasting of data item d_i can be removed from the broadcast channel but without affecting the satisfaction of the queries which require d_i.*

Proof. Obviously, for task τ_i, removing the potentially *redundant* broadcasting of d_i will not affect its execution. Now we discuss the execution of the tasks whose *shared* units including d_i.

Under the RM-UO algorithm, d_i is broadcast with period T_i^*. So, d_i definitely will appear in any time interval of length $2 \cdot T_i^*$. Assume there is a task τ_j with d_i is in its *shared* set. Based on the definition of R-task and the rules of item division, we know $T_j > T_i$. Because τ_i and τ_{i+1} are two adjacent DC tasks, and the DC-tasks are sorted in non-decreasing order of period, we know j must be larger than i, which indicates that $T_j \geq T_{i+1}$. Since τ_i is an R-task, based on Definition 4, we have,

$$T_j \geq T_{i+1} \geq T_{i+1}^* \geq 2 \cdot T_i^*.$$

As we have argued above, d_i appears in any time interval of length $2 \cdot T_i^*$, then d_i must be broadcast in each of τ_j's period. As a conclusion, if τ_i is an R-task, then all the *redundant* broadcasting of d_i can be deleted and the schedulability of the task set will not be affected. The lemma thus follows. □

Obviously, for those R-tasks, removing the *redundant* broadcasting data items can leave space to accommodate some other data items. Now the remaining problem is how to use these space, we first give a definition as follows.

Definition 5. *Given a DC task set, assume there is an R-task τ_i with period T_i, the first task τ_j with its period satisfying $T_j \geq \lceil \frac{T_i}{T_i - T_i^*} \rceil \cdot T_i^*$, is defined to be the RM-task of τ_i.*

Based on the above definition, we are now ready to present our merging policy. Assume there is an R-task τ_i and τ_j is its RM-task. Then, by replacing potentially redundant broadcasting of d_i with d_j, τ_i and τ_j can be merged into a new task τ_i^R with period $T_i^R = T_i^*$. We call such a task merging policy Redundant Query Mering.

After merging the R-task and RM-task to obtain a new one by RQM, the other problem is how to broadcast the data items so that all the queries can be satisfied.

Algorithm 2. RQM-UO

Input: a 2-harmonic task set, the current time t
Output: the data item to be broadcast at t

```
1  begin
2    Utilize the RM algorithm to derive the task schedule for
     the 2-harmonic task set within one hyper period;
3    Find the task to be executed at time t;
4    if the task to be executed is a virtual task then
5      Find the R-task and RM-task of the virtual task;
6      if broadcasting the data item accessed by the R-task is redun-
       dant then
7        Return the data item accessed by the RM-task;
8      else
9        Return the data item accessed be the R-task;
10   else
11     Return the data item accessed by the task;
12 end
```

Algorithm RQM-UO. The same as MQM-UO, we first utilize the RM algorithm to derive the task schedule for the two-harmonic task set within one hyper period. When determining the data items to be broadcast, we also need to consider two cases: 1) For a virtual task which is generated by the RQM policy, we first find the R-task and RM-task of the virtual task. Then, we check whether broadcasting the data item accessed by the R-task is redundant, by Equation (2). If the answer is negative, we broadcast this data item. Otherwise, we broadcast the data item accessed by the RM-task; 2) For the tasks which are derived by transforming the DC tasks directly, the same as MQM-UO and RM-UO, we simply broadcast the corresponding data items accessed by these tasks. To help understand the first case, suppose at time t, a virtual task τ_i^R needs to execute, and τ_i^R is the result of merging τ_i (R-task) and τ_j (RM-task). We first check whether broadcasting d_i is *potentially redundant* by verifying if Equation (2) holds for τ_i. If the answer is affirmative, it means broadcasting d_i is redundant (since τ_i is an R-task) and we broadcast d_j. Otherwise, we broadcast d_i. Similar to MQM-UO, since we only broadcast the data items in the *unique* set, we call this scheduling algorithm RQM-UO. The pseudo-code of RQM-UO is shown in Algorithm 2.

The following theorem illustrates this merging and data replacing policy does not affect the query set's schedulability.

Theorem 3. *Given a periodic continuous query set $\{q_1, q_2, \ldots, q_m\}$ and its corresponding DC task set $\{\tau_1, \tau_2, \ldots, \tau_n\}$, if $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{\frac{1}{n}} - 1)$, then this periodic continuous query set $\{q_1, q_2, \ldots, q_n\}$ is schedulable under* RQM-UO.

Proof. Based on [27], given a periodic continuous query set $\{q_1, q_2, \ldots, q_m\}$, if its corresponding DC task set $\{\tau_1, \tau_2, \ldots, \tau_n\}$ satisfies the following condition:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{\frac{1}{n}} - 1)$$

then this periodic continuous query set is schedulable under RM-UO. Now we need to prove the query set is also RQM-UOschedulable. Clearly, if there does not exist any task which can be merged by the RQM policy, then RQM-UO is the same as RM-UO, and the claim holds obviously.

Now we consider the case that some tasks can be merged by RQM. Without loss of generality, suppose τ_i is an R-task and τ_j is its RM-task. Based on Lemma 3, we know that, after merging τ_i with τ_j by RQM, any task which requires d_i can get it from the wireless channel correctly during their execution. So, in order to prove the claim in this theorem, we only need to show that under the RQM policy, the broadcasting of d_j will not affect the timing constraints of the tasks which need to access d_j.

Suppose d_i is broadcast at time unit $m (m < T_i^*)$ for the first time and let $x = \frac{T_i}{T_i - T_i^*}$, now regarding x, we have the following two cases to be considered.

- x is an integer. In this case, we first assume that $y = \lfloor \frac{m}{T_i - T_i^*} \rfloor + 1$. Then by $x = \frac{T_i}{T_i - T_i^*}$, we can get,

$$(x - 1) \cdot T_i = x \cdot T_i^*. \tag{3}$$

By $y = \lfloor \frac{m}{T_i - T_i^*} \rfloor + 1$, we can get,

$$y \cdot (T_i - T_i^*) > m \geq (y - 1) \cdot (T_i - T_i^*). \tag{4}$$

Due to the periodic execution of the queries, it is obvious that under RQM-UO, d_i will be broadcast at the two time units: $m + (y - 1 + k \cdot x) \cdot T_i^*$ and $m + (y + k \cdot x) \cdot T_i^* (k = 1, 2, \ldots)$. Moreover, for any k, there must be an instance of τ_i which releases at time unit $(y - 1 + k \cdot x - k) \cdot T_i$ and ends at time unit $(y + k \cdot x - k) \cdot T_i$. By Formulas (3) and (4), we have,

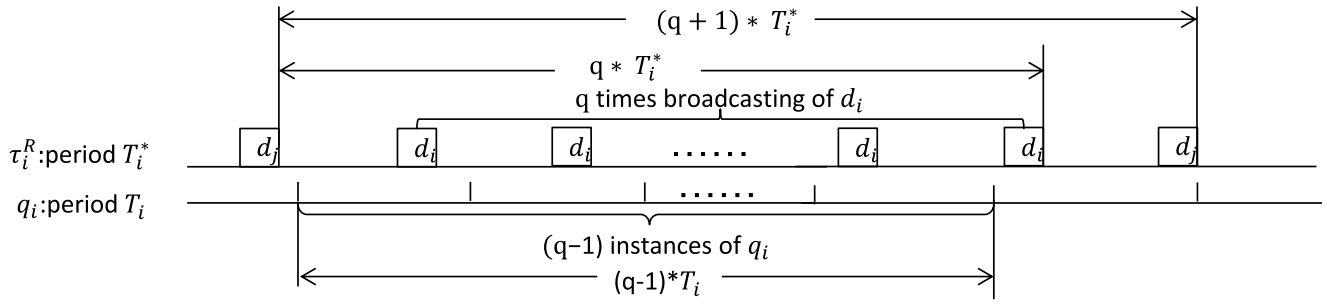


Fig. 4. Illustration of Case 2 (x is not an integer) in Theorem 2.

$$(y-1+k\cdot x-k)\cdot T_i \le m+(y-1+k\cdot x)\cdot T_i^* < m+(y+k\cdot x)\cdot T_i^* < (y+k\cdot x-k)\cdot T_i,$$

which means the $(y+k\cdot x-k)$th instance of τ_i can get d_i twice at the time units $m+(y-1+k\cdot x)\cdot T_i^*$ and $m+(y+k\cdot x)\cdot T_i^*$ respectively. This means all the broadcastings of d_i at time unit $m+(y+k\cdot x)\cdot T_i^*$ are *redundant*. According to the RQM policy, these *redundant* broadcasting items will be replaced by d_j, and d_j will be broadcast with period $x\cdot T_i^*$. Note that, task τ_j is the RM-task of τ_i. By the definition of RM-task (Definition 5), $T_j \ge \lceil \frac{T_i}{T_i-T_i^*} \rceil \cdot T_i^* = x\cdot T_i^*$, we can conclude that the broadcasting of d_j will not affect its time constraints under RQM-UO. Furthermore, based on the division rule of data items, we are sure that the periods of the tasks sharing d_j from τ_j must be longer than T_j. So, under the RQM policy, all these tasks can get d_j in each of their periods.

- x is not an integer. Note that according to the RQM policy, d_j will only replace the *redundant* broadcasting of d_i. For any two neighboring broadcastings of d_j, we assume there are q broadcastings of d_i between them. We consider the time interval between the first broadcastings of d_j and the last broadcasting of d_i. As shown in Fig. 4, the length of the interval is equal to $q*T^*$ and it must contain $(q-1)$ complete instances of τ_i, since the time interval between the broadcasting of d_i and d_j or two continuous broadcasting of d_i is T_i^* time units. So, we have $(q-1)\cdot T_i < q\cdot T_i^*$, which means $q < \frac{T_i}{T_i-T_i^*}$. Moreover, since q is an integer, we have $q+1 \le \lceil \frac{T_i}{T_i-T_i^*} \rceil$, i.e., $(q+1)\cdot T_i^* \le \lceil \frac{T_i}{T_i-T_i^*} \rceil \cdot T_i^*$. This means, under the RQM policy, d_j is broadcast at least once in every time interval of $\lceil \frac{T_i}{T_i-T_i^*} \rceil \cdot T_i^*$. Since τ_j is the RM-task of τ_i, based on the definition of RM-task, we have $T_j \ge \lceil \frac{T_i}{T_i-T_i^*} \rceil \cdot T_i^*$. So, all the instances of τ_j can get d_j before their deadlines. Based on the division rule of data times, the periods of the tasks which require d_j must be longer than $\lceil \frac{T_i}{T_i-T_i^*} \rceil \cdot T_i^*$. Therefore, all the tasks which require d_j can get d_j in each of their periods.

Based on the above discussion, the theorem follows. □

The following theorem formally shows the improvement of RQM-UO as compared to RM-UO, in terms of bandwidth consumption.

Theorem 4. *By executing the* RQM *policy once, a bandwidth consumption in the size of the utilization of the RM-task after period transformation can be saved, as compared to RM-UO.*

Proof. Without loss of generality, suppose the RQM policy is executed to merge an R-task τ_i and its RM-task τ_j. Under RM-UO, the total bandwidth occupied by τ_i and τ_j is $\frac{1}{T_i^*}+\frac{1}{T_j^*}$. By executing the RQM policy, the newly generated task τ_i^R only consumes bandwidth in the size of $\frac{1}{T_i^*}$. So, the bandwidth saving is $\frac{1}{T_j^*}$, which is the utilization of the RM-task τ_j after period transformation. □

Now let us go back to Example 2. After merging the tasks by MQM, it can be found that task τ_8 is the first R-task in the newly produced task set, and τ_9 is the RM-task of τ_8. Then, by RQM, τ_8 and τ_9 can be merged to generate a new task τ_8^R with period $T_8^R = 16$. Clearly, this new two-harmonic task set is schedulable since the total bandwidth happens to be 100 percent. As an illustration, Fig. 5 shows two broadcast cycles of τ_8^R.

4 Algorithm Implementation Issues

A whole on-demand broadcast system consists of three active components: *Arrival-query-handler* and *Scheduler* running at the server side, and *Receiver* executing on the

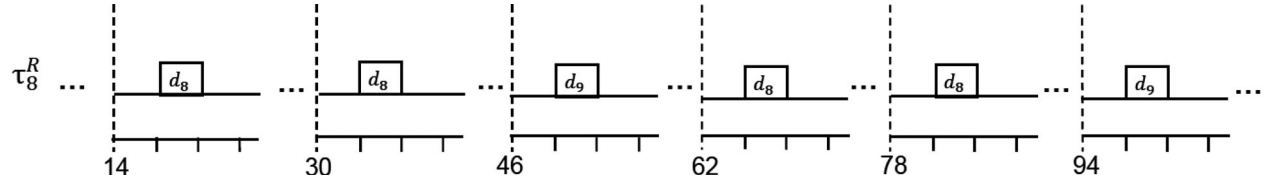


Fig. 5. A piece of schedule of the query set in Example 2 by RQM-UO.

client side. *Arrival-query-handler* maintains a pending query list and inserts newly issued queries into the list. *Scheduler* generates the broadcast schedule, releases pending queries and removes expired queries at appropriate instants. *Receiver* simply keeps listening to the broadcast channel and retrieves the needed data items as they appear. In this work, we adopt the same *Arrival-query-handler* and *Receiver* algorithms as in [27]. Hence, we mainly concentrate on detailing the design of the *Scheduler*. Specifically, we will discuss the implementation details of the proposed MQM and RQM polices, and introduce a new scheduling algorithm namely UM which combines RQM with MQM. We start by defining two data structures OQ (original query profile) and TP (task profile list) as follows:

- OQ is described as a six-tuple in the form of $< id, period, items, Uitems, Sitems, k >$, where id is a unique number for identifying a query; $period$ is the period of this query; $items$ includes all the data items the query requires; $Uitems$ and $Sitems$ denote the *unique* and *shared* sets of the query, respectively; and k is the number of items which are in $Uitems$. Note that, once a query is accepted, an OQ is initialized. $OQList$ consists of all the OQs which are sorted in non-decreasing order of their periods. If the queries having the same periods, then they are sorted in non-decreasing order of the number of the data items in their *unique* sets.
- TP is defined as an eight-tuple in the form of $< id, op, pp, anum, flag, itemlist, vdiset, position >$, where id is a unique number for identifying a task; op is the original period of the query from which the task is generated; pp is the period of the two-harmonic task which is generated from this task by the S_r algorithm; $anum$ is N_{id}, which is introduced in Definition 2; $flag$ is used to identify the merging policy (MQM or RQM) adopted when generating the task. Specifically, $flag = 0$ means that the task is directly transformed from a query without adopting any merging policy. $flag = 1$ and $flag = 2$ indicate that the task is produced by using the MQM and RQM policies, respectively; $itemlist$ is a list used to record the data items. If $flag = 0$, only the single data item in the *unique* set of task TP is in the $itemlist$; If $flag = 1$, task TP is produced by using the MQM policy and $itemlist$ records the *unique* sets of the m_i tasks respectively which are merged into one task TP. If $flag = 2$, then task TP is produced by using the RQM policy and $itemlist$ records the *unique* sets of the R-taskτ_i and its RM-task τ_j, sequentially. It should be pointed out that, when the RM-task is a virtual task generated by executing MQM, we use $vdiset$ to record the data items which are accessed by the virtual task, and the second element in $itemlist$ is set to be null. When $flag = 2$, $position$ is used to record the position of the last broadcast of the first data item in $itemlist$, so that we can determine whether it is unnecessary to broadcast the data item in the *unique* sets of the R-task at a special time unit. Note that, a TP has only one single unique data item and it is generated by transforming an OQ. Moreover, all the TPs constitute a list $TPList$ and they are sorted in non-decreasing order of $TP.pp$.

Before showing the UM algorithm, we first detail MQM and RQM.

Algorithm 3 shows the pseudo-code of merging tasks by MQM. For presentation convenience, we let

$$\begin{cases} \alpha(i) & = \left\lfloor \frac{2^{\lfloor \log_2 TP[i].anum \rfloor} \cdot TP[i].op}{TP[i].pp} \right\rfloor \\ \beta(i) & = 2^{\lfloor \log_2 TP[i].anum \rfloor}, \end{cases}$$

where i is a cursor used to traverse $TPList$. First, we traverse the $TPList$ to find the first task τ_i which satisfies $\alpha(i) > \beta(i)$ (line 4). Once such a task τ_i is found, we know it is a $(z, 2^y)$-task, where $z = \frac{\alpha(i)}{x}$, $2^y = \frac{\beta(i)}{x}$, and x is the greatest common divisor of $\alpha(i)$ and $\beta(i)$ (line 5). Next, we check the number of the tasks behind τ_i. Note here since the size of $TPList$ is n, $TP[n-1]$ is the last TP in $TPList$. If $i + z - 1 \leq n - 1$, we know there are at least $(z-1)$ tasks behind τ_i, and the z tasks (from τ_i to τ_{i+z-1}) can be merged into one task τ_i^M with period $T_i^M = \gamma \cdot \frac{k_i}{2^y}$ (lines 6-8). Otherwise, all the tasks starting from τ_i will be merged (lines 9-11). Finally, we order the newly generated task set $TPList$ in non-decreasing order of pp (line 13) and return the result. The time complexity of MQM is $O(n \cdot \log n)$, since traversing $TPList$ takes $O(n)$ time, while ordering the newly generated $TPList$ takes at most $O(n \cdot \log n)$ time.

Algorithm 3. MQM

```
   Input: TPList;
   Output: TPList derived by MQM
 1 begin
 2   i = 0; temp = 0;
 3   while TP[i] is not null do
 4     if α(i) > β(i) then
 5       x = GCD(α(i), β(i));
 6       if i + α(i)/x − 1 ≤ n − 1 then
 7         Merge tasks from TP[i] to TP[i + α(i)/x − 1];
 8         Delete tasks from TP[i + 1] to TP[i + α(i)/x − 1];
 9       else
10         Merge tasks from TP[i] to TP[n − 1];
11         Delete tasks from TP[i + 1] to TP[n − 1];
12     i++;
13   Order tasks in TPList in non-decreasing order of pp;
14   Return TPList;
15 end
```

Algorithm 4 presents the pseudo-code of RQM. For presentation convenience, we let $F(i) = \lceil \frac{TP[i].op}{TP[i].op - TP[i].pp} \rceil$. The same as in Algorithm 3, we first traverse $TPList$ to find the first task τ_i which satisfies $TP[i].pp \neq TP[i+1].pp$ and $TP[i].pp \neq TP[i].op$. Based on Definition 4, we know τ_i is an R-task (line 4). Next, we traverse $TPList$ with cursor j to find the RM-task τ_j of τ_i, and merge them (lines 5-9). Note here j starts from $i+1$. This is because based on Definition 5, we have $T_j \geq \lceil \frac{T_i}{T_i - T_i^*} \rceil \cdot T_i^* \geq 2 \cdot T_i^* > T_i^*$, which indicates that $j \geq i+1$. It is not difficult to see the time complexity of Algorithm 4 is $O(n^2)$, since it contains two while loops.

Algorithm 4. RQM

Input: $TPList$
Output: $TPList$ derived by RQM
1 **begin**
2 $i = j = 0$;
3 **while** $TP[i]$ *is not null* **do**
4 **if** $TP[i].pp \neq TP[i+1].pp$ and $TP[i].pp \neq TP[i].op$ **then**
5 $j = i + 1$;
6 **while** $TP[j]$ *is not null* **do**
7 **if** $TP[j].op \geq F(i) \cdot TP[i].pp$ **then**
8 Merge $TP[i]$ with $TP[j]$;
9 Delete $TP[j]$; Break;
10 $j++$;
11 $i++$;
12 return $TPList$;
13 **end**

Now, we are ready to show our UM algorithm, which consists of the following three steps:

- Divide the data item set of every query into two sets, *unique* set and *shared* set, and generate a DC task set.
- Transform the newly generated DC tasks into two-harmonic tasks, and merge them by MQM and RQM (this step obtains a new two-harmonic task set).
- Schedule the two-harmonic tasks by the UM algorithm, which combines RQM-UO and MQM-UO.

Algorithm 5 shows the first two-steps of UM. Similar to RM-UO, we first divide the item set and transform original queries into two-harmonic tasks (line 2). Then, the MQM and RQM schemes are invoked sequentially to merge some tasks, with the objective of saving the bandwidth consumption. After that, the $utilization$ of $TPList$ is checked to see whether the newly generated two-harmonic task set can be scheduled. If $utilization$ of $TPList$ does not exceed 100 percent, we use the RM algorithm to generate the broadcasting sequence BS of $TPList$ (line 6). Note here BS is also a $TPList$ which consists of multiple BSs, and each $BS[i]$ corresponds to one $TP[i']$.

Algorithm 5. Admission Control of UM

Input: $OQList$
Output: Scheduling sequence BS of queries in $TP\ List$
1 **begin**
2 Generate 2-harmonic task list $TP\ List$ of $OQ\ List$ by S_r and compute $anum$ for each TP;
3 Invoke Algorithm MQM (with $TP\ List$ as input);
4 Invoke Algorithm RQM (with $TP\ List$ derived in the last step as input);
5 **if** $utilization$ of $TP\ List \leq 100\%$ **then**
6 Use RM to generate the scheduling sequence BS of $TP\ List$ within one hyper-period;
7 return BS;
8 **end**

After computing the initial broadcasting sequence BS, we use the scheduler of UM (Algorithm 6) to identify the data item to be broadcast at each time unit. As mentioned above, for each $BS[i]$ in BS, it corresponds to a $TP[i']$. Suppose the current time unit is t and $BS[t]$ is selected to be broadcast, we have the following three cases to be considered:

- If $BS[t].flag = 0$, i.e., $BS[t]$ represents a task derived by directly transforming (using S_r) the originally query, then the data item in $TP[t'].itemlist$ is returned (line 3);
- If $BS[t].falg = 1$, i.e., $BS[t]$ represents a task derived by merging some tasks with MQM. Then based on the MQM policy, we circularly choose the data items in $TP[t'].itemlist$ (line 5) and return it;
- If $BS[t].falg = 2$, i.e., $BS[t]$ represents a task derived by merging some tasks with RQM. Then based on the RQM policy, the *redundant* broadcast of the data item accessed by an R-task will be replaced by the data item accessed by its RM-task. We use the $position$ (the initial value of $position$ is set to be a negative number) of each TP to help check whether one broadcast is *redundant*. Specifically, if the profile of $TP[t']$ satisfies the following condition:

$$\left\lfloor \frac{TP[t'].position}{TP[t'].op} \right\rfloor = \left\lfloor \frac{t}{TP[t'].op} \right\rfloor \quad (5)$$

then the broadcast of the first item in $TP[t'].itemlist$ is *redundant*, and we have two cases to be considered. If $TP[t'].itemlist[1]$ is null, it means the RM-task is a virtual task derived by MQM, and we cyclically choose the data item from $TP[t'].vdiset$ and return it (line 9). Otherwise, we simply return $TP[t'].itemlist[1]$ (line 11). Finally, if Equation (5) does not hold, we return the first data item in $TP[t'].itemlist$ (line 13).

Algorithm 6. Scheduler of UM

Input: BS; t—the current time unit
Output: The data item to be broadcast at time t
1 **begin**
2 **if** $BS[t].flag = 0$ **then**
3 Return the data item in $TP[t'].itemlist$;
4 **if** $BS[t].flag = 1$ **then**
5 Cyclically choose the data item in $TP[t'].itemlist$ and return it;
6 **if** $BS[t].flag = 2$ **then**
7 **if** *the profile of* $TP[t']$ *satisfying Equation* (5) **then**
8 **if** $TP[t'].itemlist[1]$ *is null* **then**
9 Cyclically choose the data item in $TP[t'].vdiset$ and return it;
10 **else**
11 Return $TP[t'].itemlist[1]$;
12 **else**
13 Return $TP[t'].itemlist[0]$;
14 **end**

Algorithm 5 has a time complexity of $O(n^2)$, while Algorithm 6 has a time complexity of $O(1)$. But for a given query set, Algorithm 5 only needs to be executed once, while Algorithm 6 is invoked at each broadcast slot.

As stated in Theorems 2 and 4, both MQM and RQM algorithms can lead to considerable bandwidth savings as compared to RM-UO. In UM, we first invoke the MQM

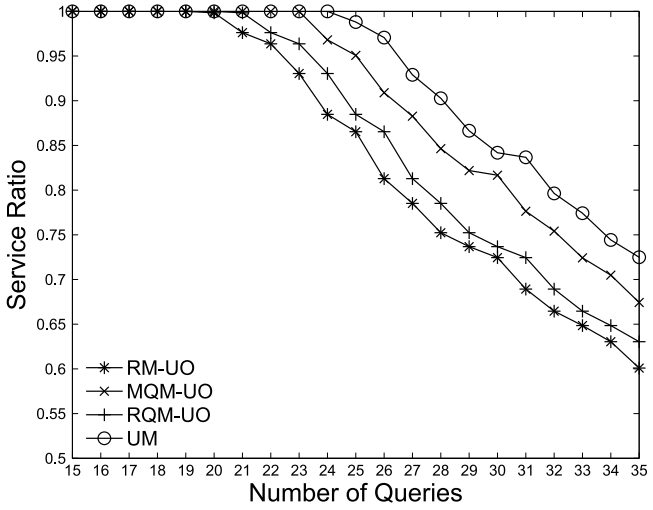


Fig. 6. Service ratio comparison with $m = 5{,}000$.

algorithm, and then the RQM algorithm to generate the final task set for scheduling, which means UM combines the advantages of both MQM and RQM. Hence, the performance improvement of UM is also more significant, as illustrated in our experimental study.

5 Experimental Evaluation

This section presents the performance evaluation of the proposed algorithms versus RM-UO via simulation experiments. We first describe the experimental setup in Section 5.1, and then present our experimental results and discussions in Section 5.2.

5.1 Experimental Setup

To our best knowledge, RM-UO is the only known work that can support continuous periodic queries with time constraints. Hence, in this section, we focus on comparing our algorithms with RM-UO. Specifically, we compare the performance of the following four algorithms: RM-UO, MQM-UO, RQM-UO and UM. We have developed a simulator written in C++ which imitates the on-demand broadcasting environments in which data items for periodic continuous queries are scheduled. It should be pointed out that when a new query arrives, all the four algorithms will check the feasibility at first. If the feasibility test cannot be passed, then the new query cannot be served. Hence, it is meaningless to compare the deadline miss ratios of the queries which have got their services, since the results are all 0 under the four algorithms. In fact, a good scheduling algorithm should save more broadcast bandwidth to serve more queries. In view of this, in this work, we use the service ratio of a set of queries and the total bandwidth consumption of a set of accepted queries to measure the performance of the scheduling algorithms. Note that when testing the bandwidth consumption, our experiments are designed with the premise that all the queries can get their services.

To simulate a wide variety of queries, we choose five classes of queries, whose parameters are originally from [27]. In particular, each query period belongs to one of the following five ranges: $[50, 60]$, $[100, 120]$, $[140, 150]$, $[250, 300]$ and $[500, 590]$. The number of data items accessed by a query belonging to the five period ranges is $[1, 2]$, $[3, 5]$, $[5, 8]$, $[10, 15]$ and $[15, 20]$. Note here we use the similar baseline values for the parameters as [27], for the following reasons: 1) To enable easy comparison; 2) Our objective is to evaluate the relative performance characteristics of the approaches, not their absolute levels. Within each category, a query's period and the number of data items to be accessed by the query are randomly selected from the corresponding ranges. For each particular level in each simulation set, we randomly generate $1{,}000$ qualified query sets and take the average. For each point plotted in the figure, the simulations continued until a confidence interval of 95 percent with half-width of less than 5 percent about the mean was achieved.

5.2 Experimental Results

5.2.1 Service Ratio

In the first set of experiments, by fixing the number of data items to be $m = 5{,}000$, we first test the service ratios with different number of queries. The number of the queries N is set to be 15 initially, and increases with a step size of 1, until it reaches 35. Fig. 6 shows the service ratio of the four algorithms. It can be seen that when N is no larger than 20, feasibility can be easily achieved and all algorithms yield 100 percent service ratio. As N increases, the service ratio performance of all schemes drops to some extent in different degrees. This is because with limited bandwidth, some queries cannot get services any more when N is relatively large. Among the four algorithms, UM has the best service ratio performances, followed by MQM-UO and RQM-UO. The largest performance gap between UM and RM-UO is about 20 percent. MQM-UO outperforms RQM-UO on the service ratio aspect because it can merge more tasks (or queries) to save more bandwidth consumption. UM exhibits the best performance since it combines both the advantages of MQM-UO and RQM-UO.

Next, by fixing the number of the queries to be $N = 30$, we test the service ratio performance with different number of data items (m) in the database. m is limited in $[100, 2{,}500]$ and increases with a step size of 100. The performance of the four algorithms on the service ratio is shown in Fig. 7. As can be seen, the service ratios of the four algorithms decrease sharply with the increase of m. This is because with the growth of m, the probability of one item accessed by multiple queries decreases. UM has the best service ratio among the four algorithms, due to the same reason as in Fig. 6. Moreover, we can see that in this experiment, the largest performance gap between UM and RM-UO is about 21 percent.

5.2.2 Bandwidth Consumption

In the second set of experiments, we compare the bandwidth consumption of the four algorithms. Firstly, we test the bandwidth consumption with different number of queries. In order to guarantee that all the queries can get their services, we let $m = 5{,}000$ and vary the range of N to $[5, 18]$. The bandwidth consumption comparison is shown in Fig. 8. As can be seen, UM preserves the best performance on bandwidth utilization, during almost the whole range of

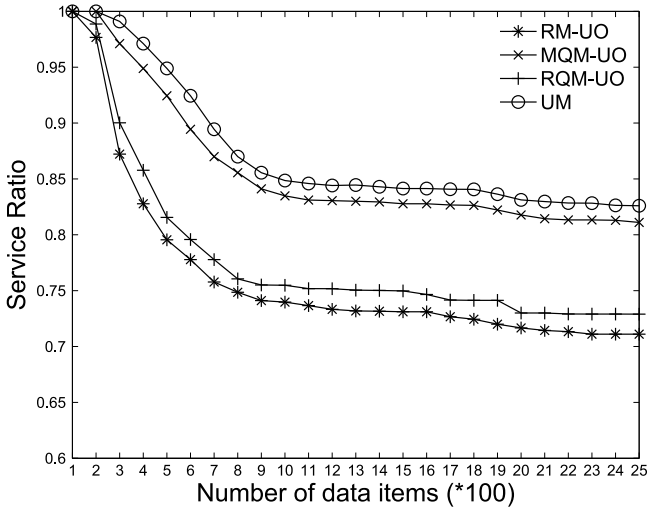


Fig. 7. Service ratio comparison with $N = 30$.

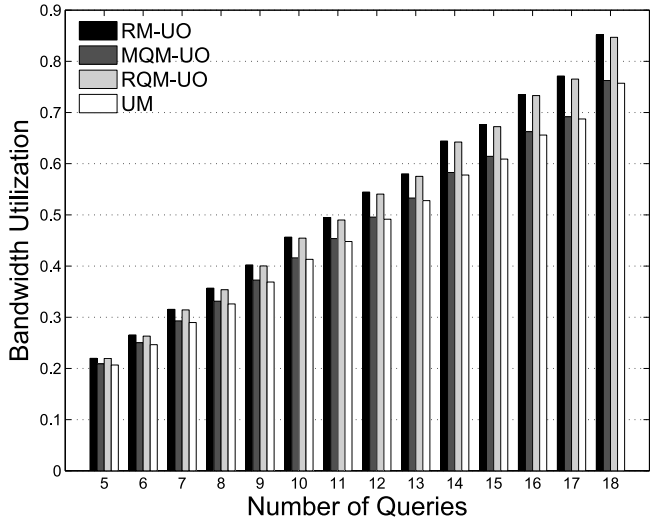


Fig. 8. Bandwidth utilization comparison with $m = 5{,}000$.

parameter settings. It can also be observed that with the growth of N, the superiority of UM compared to RM-UO becomes more significant. The largest performance gap between UM and RM-UO is about 11 percent when $N = 18$.

Secondly, by fixing the number of the queries to be $N = 30$, we test the bandwidth consumption performance with different number of data items (m) in the database. To guarantee that all the queries can get their services, m is limited in $[30, 100]$ and increases with a step size of 10. Fig. 9 shows the bandwidth utilization comparison of the four algorithms. It can be seen that UM still outperforms the other three algorithms, and the performance gains become more significant as m becomes larger. When $m = 100$, the largest performance gap (about 10 percent) between UM and RM-UO can be achieved.

In summary, both MQM-UO and RQM-UO outperform RM-UO due to that they can merge queries to save bandwidth consumption and thus accommodate more queries. MQM-UO outperforms RQM-UO since it can merge more queries. UM outperforms the other three algorithms in terms of the resulted bandwidth utilization and service ratio in almost the whole range of parameter setting, because it combines the advantages of both MQM and RQM.

6 Related Work

Scheduling algorithms play an important role in data dissemination in mobile environments. Since the publication of the seminal work by Acharya et al. [1], various scheduling algorithms have been proposed to determine the broadcast sequence of data items [2], [4], [9], [12], [15], [24], [25], [26], [29], [30]. In general, these algorithms can be classified into two categories: *push-based* and *on-demand*. The push-based approach periodically broadcasts a set of predetermined data items, based on some prior knowledge of data access patterns, regardless of individual information requirements. Hence, push-based broadcast is useful for applications with relatively stable access patterns. In contrast, on-demand broadcast is more suitable and widely used for dynamic and large-scale data dissemination. Moreover, when dealing with queries with time constraints, on-demand broadcast is more suitable than push-based method due to that it takes the client's individual requirements into consideration to determine broadcasting period of the data items.

For on-demand broadcasting environments with real-time constraints, Lam et al. studied the issue on how to ensure consistency and currency of data items and proposed the OUFO algorithm in [17]. Fang et al. proposed the ACR algorithm which always broadcasts the data item with the most deadlines to meet at each time unit in [11]. Xu et al. [28] considered both urgency and access frequency of data items, and proposed the Slack time Inverse Number of pending requests (SIN) algorithm with a theoretical upper bound of deadline miss rates. In [19], Lee et al. presented a pyramid preemptive algorithm (PRDS) which considers data urgency, data size and the number of pending requests. In addition to the above firm deadline scheduling, Dewri et al. studied in [10] the soft deadline broadcast scheduling problem and proposed an evolution strategy based stochastic hill-climber. However, all of the above studies are focused on single item requests, single channel real-time broadcast scheduling.

With the rapid development of real-time broadcast technology, multi-item requests problem and multi-channel

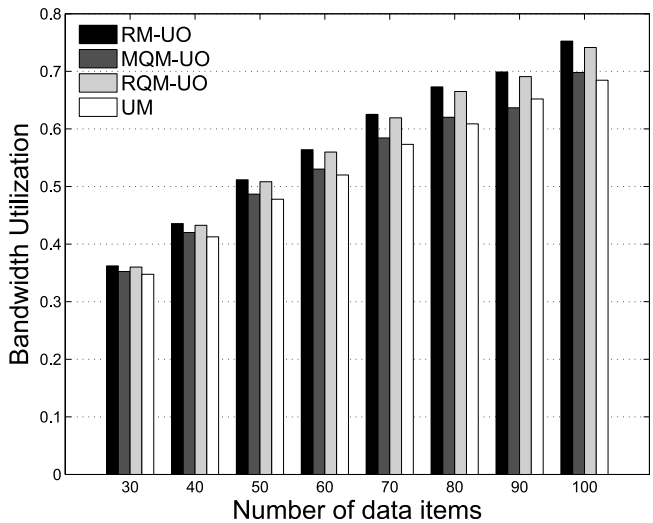


Fig. 9. Bandwidth utilization comparison with $N = 30$.

broadcast scheduling problem have attracted significant attention and a lot of research efforts have been conducted on these topics. Lee et al. [18] presented multiple priority allocation policies to deal with the broadcast scheduling problem in single item requests, multi-channel environment. Chen et al. [7] made a great contribution in the area of multi-item, single-channel broadcast scheduling. They propose an efficient algorithm which considers the urgency, access frequency and request serving status. For multi-item requests, multi-channel broadcast scheduling, Chen et al. gave a theoretical minimum number of channels for scheduling all queries and proposed a data allocation algorithm in [5]. Liu and Lee [21] presented the DUP algorithm which considers both the urgencies of queries and data items, and the DUP algorithm always broadcasts the most urgent data item which is required in the most urgent query. A much more comprehensive discussion on recent research results can be found in [22]. Lv et al. introduced the concept of profit into the broadcast schedule and proposed an efficient scheduling algorithm in [23] which always choose the data item with highest profit to be broadcast at each time unit. It should be pointed out that, all the work mentioned above focused on one-shot broadcast schedule and the goals of them are fundamentally different from ours in providing timing predictability.

The first study on real-time broadcast scheduling problem was conducted by Baruah and Bestavros [3], in which data items are assumed to be broadcast at least once in a given spacing, so that clients can get all their accessing data items in a pre-specified timing bound. The authors also proposed an algorithm which is closely related to the pinwheel scheduling in [3]. However, different form our research environment, the algorithm proposed in [3] is designed for push-based real-time broadcast schedule. In [8], Chung et al. extended [3] to a multi-channel environment and proposed the minimum number of channels to satisfy all the queries. Later, Chen et al. [5] further extended [8] to multi-item requests environment. However, as pointed out in [27], all these work ([3], [5], [8]) neglected the data sharing character between queries in broadcast environment and assumed the read sets of any two queries are disjoint. Different from them, our work in this paper considers the sharing character, in particular, a query can access more than one data item and an overlap between the read sets of any two queries is allowed.

7 Conclusions and Future Work

Research on scheduling for periodic continuous queries in broadcasting environments has received specific attention recently. In this paper, we took a second look at the problem and proposed two merging policies (MQM and RQM) to save the bandwidth consumption of the existing RM-UO algorithm. Based on the two merging policies, a unified merging based broadcast scheduling algorithm (UM) is proposed. Experimental study shows that UM exhibits significant performance improvement compared to RM-UO, in terms of both wireless bandwidth consumption and query service ratio.

For future work, we intend to generalize our current work to multi-channel environments. Moreover, in this work, we assume that all the data items received in each period are valid to the query, it would be interesting to take the temporal constraints of data items into consideration when processing periodic queries with real-time requirement.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive and helpful comments. This work was substantially supported by the State Key Program of National Natural Science of China under Grant No. 61332001, National Natural Science Foundation of China under Grants Nos. 61173049, 61300045, and China Postdoctoral Science Foundation under Grant No. 2013M531696. Jianjun Li is the corresponding author.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data management for asymmetric communication environments," *ACM SIGMOD Rec.*, vol. 24, pp. 199–210, 1995.

[2] N. Bansal, R. Krishnaswamy, and V. Nagarajan, "Better scalable algorithms for broadcast scheduling," in *Proc. 37th Int. Colloquium Automata, Lang. Programm.*, 2010, pp. 324–335.

[3] S. Baruah and A. Bestavros, "Pinwheel scheduling for fault-tolerant broadcast disks in real-time database systems," in *Proc. IEEE Int. Conf. Data Eng.*, 1997, pp. 543–551.

[4] J. Chang, T. Erlebach, R. Gailis, and S. Khuller, "Broadcast scheduling: Algorithms and complexity," *ACM Trans. Algorithms*, vol. 7, no. 4, p. 47, 2011.

[5] C.-C. Chen, C. Lee, and S.-C. Wang, "On optimal scheduling for time-constrained services in multi-channel data dissemination systems," *Inf. Syst.*, vol. 34, no. 1, pp. 164–177, 2009.

[6] J. Chen, G. Huang, and V. C. Lee, "Scheduling algorithm for multi-item requests with time constraints in mobile computing environments," in *Proc. Int. Conf. Parallel Distrib. Syst.*, 2007, vol. 2, pp. 1–7.

[7] J. Chen, V. Lee, and K. Liu, "On the performance of real-time multi-item request scheduling in data broadcast environments," *J. Syst. Softw.*, vol. 83, no. 8, pp. 1337–1345, 2010.

[8] Y.-C. Chung, C.-C. Chen, and C. Lee, "Design and performance evaluation of broadcast algorithms for time-constrained data retrieval," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1526–1543, Nov. 2006.

[9] Y. D. Chung and M.-H. Kim, "Qem: A scheduling method for wireless broadcast data," in *Proc. 6th Int. Conf. Database Syst. Adv. Appl.*, 1999, pp. 135–142.

[10] R. Dewri, I. Ray, I. Ray, and D. Whitley, "Utility driven optimization of real time data broadcast schedules," *Appl. Soft. Comput.*, vol. 12, no. 7, pp. 1832–1846, 2012.

[11] Q. Fang, S. V. Vrbsky, Y. Dang, and W. Ni, "A pull-based broadcast algorithm that considers timing constraints," in *Proc. Int. Conf. Parallel Process. Workshops*, 2004, pp. 46–53.

[12] J. Fernandez-Conde and K. Ramamritham, "Adaptive dissemination of data in time-critical asymmetric communication environments," in *Proc. 11th Euromicro Conf. Real-Time Syst.*, 1999, pp. 195–203.

[13] C.-C. Han, K.-J. Lin, and C.-J. Hou, "Distance-constrained scheduling and its applications to real-time systems," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 814–826, Jul. 1996.

[14] C.-L. Hu, "Fair scheduling for on-demand time-critical data broadcast," in *Proc. IEEE Int. Conf. Commun.*, 2007, pp. 5831–5836.

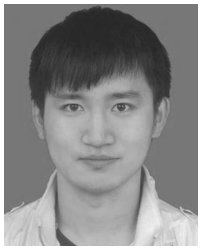
[15] J.-L. Huang and M.-S. Chen, "Dependent data broadcasting for unordered queries in a multiple channel mobile environment," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1143–1156, Sep. 2004.

[16] K.-W. Lam and S.-L. Hung, "Scheduling real-time requests in on-demand broadcast environments," in *Proc. 1st Int. Conf. Netw.-Based Inf. Syst.*, 2007, pp. 258–267.

[17] K.-Y. Lam, E. Chan, H.-W. Leung, and M.-W. Au, "Concurrency control strategies for ordered data broadcast in mobile computing systems," *Inf. Syst.*, vol. 29, no. 3, pp. 207–234, 2004.

[18] G. Lee, Y.-N. Pan, and A. L. Chen, "Scheduling real-time data items in multiple channels and multiple receivers environments," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, 2002, pp. 455–456.
[19] V. C. Lee, X. Wu, and J. K.-Y. Ng, "Scheduling real-time requests in on-demand data broadcast environments," *Real-Time Syst.*, vol. 34, no. 2, pp. 83–99, 2006.
[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[21] K. Liu and V. Lee, "On-demand broadcast for multiple-item requests in a multiple-channel environment," *Inf. Sci.*, vol. 180, no. 22, pp. 4336–4352, 2010.
[22] K. Liu and V. Lee, "Performance analysis of data scheduling algorithms for multi-item requests in multi-channel broadcast environments," *Int. J. Commun. Syst.*, vol. 23, no. 4, pp. 529–542, 2010.
[23] J. Lv, V. Lee, M. Li, and E. Chen, "Profit-based scheduling and channel allocation for multi-item requests in real-time on-demand data broadcast systems," *Data Knowl. Eng.*, vol. 73, pp. 23–42, 2012.
[24] C. J. Su and L. Tassiulas, "Broadcast scheduling for information distribution," in *Proc. IEEE 16th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 1997, vol. 1, pp. 109–117.
[25] C.-J. Su, L. Tassiulas, and V. J. Tsotras, "Broadcast scheduling for information distribution," *Wireless Netw.*, vol. 5, no. 2, pp. 137–147, 1999.
[26] W. Sun, Z. Zhang, P. Yu, and Y. Qin, "Skewed wireless broadcast scheduling for multi-item queries," in *Proc. Int. Conf. Wireless Commun., Netw. Mobile Comput.*, 2007, pp. 1865–1868.
[27] H. Wang, Y. Xiao, and L. Shu, "Scheduling periodic continuous queries in real-time data broadcast environments," *IEEE Trans. Comput.*, vol. 61, no. 9, pp. 1325–1340, Sep. 2012.
[28] J. Xu, X. Tang, and W.-C. Lee, "Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 1, pp. 3–14, Jan. 2006.
[29] D.-N. Yang and M.-S. Chen, "Data broadcast with adaptive network coding in heterogeneous wireless networks," *IEEE Trans. Mobile Comput.*, vol. 8, no. 1, pp. 109–125, Jan. 2009.
[30] B. Zheng, X. Wu, X. Jin, and D. L. Lee, "Tosa: A near-optimal scheduling algorithm for multi-channel data broadcast," in *Proc. ACM Conf. Mobile Data Manag.*, 2005, pp. 29–37.

Guohui Li received the PhD degree in computer science from the Huazhong University of Science and Technology (HUST), China, in 1999. He was promoted to a full professor in 2004, and currently acts as a vice dean of the School of Computer Science and Technology in HUST. His research interests mainly include real-time systems, mobile computing, and advanced data management.



Quan Zhou received the BS degree in computer science from Heilongjiang University, China, in 2009. He is currently working toward the PhD degree at the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. His research interests include real-time scheduling and mobile computing.

Jianjun Li received the PhD degree in computer science from the Huazhong University of Science and Technology (HUST), China, in 2012. He is currently a postdoctoral and lecturer at the School of Computer Science and Technology, Huazhong University of Science and Technology. He serves as the corresponding author of this article. His research interests include real-time systems, real-time databases and advanced data management.