Scheduling Queries to Improve the Freshness of A Website

Haifeng Liu Wee-Keong Ng Ee-Peng Lim

Centre for Advanced Information Systems, School of Computer Engineering
Nanyang Technological University, Singapore 639798, SINGAPORE
{awkng,p144330195}@ntu.edu.sg

Abstract

The WWW is a new advertising media in recent years where corporations utilize it to increase their exposure to consumers. For a very large website whose content is derived from some source database, it is important to maintain its *freshness* in response to changes to the base data. This issue is particularly significant for websites presenting fast changing information such as stock exchange information and product information. In this paper, we formally define and study the *freshness* of a website that is refreshed by scheduling a set of queries to fetch fresh data from the databases. Then, we propose several online scheduling algorithms and compare the performance of the algorithms on the freshness metric. Our conclusion is verified by empirical results.

Keywords: Internet Data Management, View Maintenance, Query Optimization, Hard Real-Time Scheduling

1 Introduction

The popularity of the World-Wide Web (WWW) has made it a prime vehicle for disseminating information. More and more corporations and individuals advertise themselves through websites in recent years. The relevance of database concepts to the problems of managing and querying Web information has led to a significant body of recent research addressing these problems. Three main classes of tasks related to information management on the WWW were proposed in [3]: modeling and querying the Web, information extraction and integration, website construction and restructuring. Usually, all Web pages can be classified into three categories as follows:

(i) *Static* This kind of Web pages present information that either does not change over time, or it rarely changes. Examples are personal home pages.

(ii) *Dynamic* This kind of Web pages are dynamically computed, usually by a CGI script at run-time when a user submits a query form with the required input parameters. The content of such Web pages varies according to various input parameters.

(iii) *Semi-dynamic* These Web pages have their contents derived from some source databases and they change in response to updates to the source databases. In relative to dynamic pages, we refer to them as semi-dynamic pages because they remain static and do not change automatically unless a user initiates a refresh request explicitly. A refresh request is mostly initiated due to updates to the base data. An example of this kind of page can be found at `http://www.fish.com.sg` where a list of stock exchange information is refreshed frequently in response to updates to base data.

As the World Wide Web continues its rapid growth, the number of semi-dynamic Web pages with information extracted from source databases increases rapidly too. A crucial problem arises when base data change at a high frequency and there is a need to keep a large set of semi-dynamic pages up-to-date in response to the changes since nobody is interested in stale data on the Web (an obsolete stock price may lead an investor relying on the Web to make a loss). We have proposed a generic model for timely refreshing semi-dynamic websites in [5], where a *Web cell* has been defined as a portion of a Web page that is derived by the result of a *refresh query* against some base tables that change at specific frequencies. In order to keep a data-intensive website up-to-date, we design a *cellbase* that materializes the Web cells hosted on the website and schedule the set of refresh queries to update the cells in the cellbase. The website is considered fresh only if its cellbase can be kept up-to-date. More details can be found in [5].

This paper focuses on formally measuring and studying the *freshness* of a cellbase such that the website can be kept the most up-to-date by choosing a proper scheduling algorithm since we may apply different algorithms to schedule refresh queries to pull fresh data from source databases. Another approach that can be applied to improve the cellbase freshness by optimizing refresh query set is discussed in [6].

The paper is organized as follows: In Section 2 we first introduce some basic notations and definitions that are used throughout the paper. We define and propose several scheduling algorithms to schedule the executions of refresh queries in Section 3. In Section 4, we formally define the metric cellbase freshness and present a few of basic properties of it. We then have a brief discuss on how the feasibility of a refresh query set and different scheduling algorithm affect the cellbase freshness in Section 5. Section 6 has a comparison study of the scheduling algorithms on cellbase freshness and the analytical results are verified by the simulation results. We review related work in Section 7 and draw the conclusions of the paper in Section 8.

2 Basic Definitions

We formally define several basic concepts in this section.

Definition 1 (Feature Table) *For a Web cell c_i derived from a set of base tables T_i, we call one of its base table b_i^f as its feature table where $b_i^f \in T_i$ and b_i^f has the minimal update period among all base tables in T_i.* ∎

Although the content of a cell is affected by the updates of all its base tables, we focus on the *update pattern* of the feature table only. We explain the reason in Theorem 1 later.

Definition 2 (Update Pattern of Feature Table) *Let b_i^f be the feature table of cell c_i. If b_i^f is updated at a constant period U_i, the duration of each update is assumed to be the same and the first update completes at time $t_{i,1}$, then we know the time instants when the subsequent updates complete ($t_{i,j}$ is the completion time of the jth update):*

$$\begin{aligned} t_{i,2} &= t_{i,1} + U_i \\ t_{i,3} &= t_{i,1} + 2U_i \\ &\vdots \quad\quad \vdots \\ t_{i,j} &= t_{i,1} + U_i(j-1) \\ &\vdots \quad\quad \vdots \end{aligned} \qquad \text{(Eq. 1)}$$

Therefore, the tuple $\langle U_i, t_{i,1} \rangle$ is the set of all time instants when updates happen at the feature table b_i^f and we denote the tuple as the update pattern of b_i^f. ∎

When feature table b_i^f of cell c_i is updated with update pattern $\langle U_i, t_{i,1} \rangle$, the time axis is divided into time intervals with the same length as U_i by the sequence of time instants when updates finish at the feature table. We call such each time interval the *update interval* of b_i^f which corresponds to the *refresh interval* of c_i (Figure 1).

Definition 3 (Refresh Interval) *A refresh interval of cell c_i is the time interval between two consecutive completions of a feature table update. Clearly, the length of each refresh interval of c_i is the same as the update period U_i of feature table b_i^f.* ∎

A *refresh request* for a cell is raised whenever its feature table has finished one update. We denote $r_{i,j}$ as the jth refresh request for cell c_i raised at time $t_{i,j}$ when c_i's jth update finishes. Thus, a refresh interval of a cell is identified by the time interval between two consecutively raised

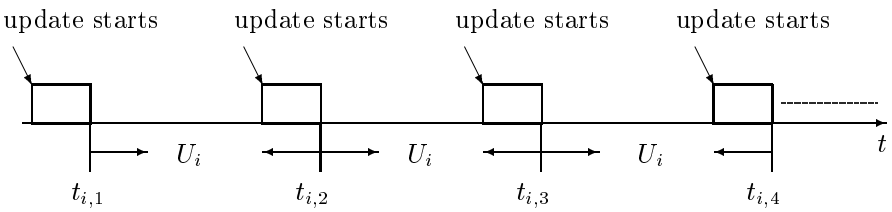

Figure 1: Periodical update pattern of the feature table of cell c_i

refresh requests of the cell. Since all refresh intervals have the same length as U_i, we also call U_i as the *refresh period* of cell c_i.

Generally, the freshness of a cell is affected by the updates on all its base tables. A cell is kept fresh if its refresh query can always fetch the newest data from base tables before the next update on the base tables. We give a formal definition of *freshness* of a cell in Section 4. However, among the set of base tables of one cell, it suffices to keep track of only the updates of the feature table as expressed by the following theorem:

Theorem 1 *If there is one refresh query performed within each update interval of the feature table of a cell, then there is at least one refresh query performed within each update interval of the other base tables of the cell.* ∎

The proof is straightforward as all base tables are updated periodically and the feature table has the minimum update period.

Definition 4 (Timely Refresh) *Let q_i be the refresh query of the cell c_i. If one execution of q_i begins and ends within each refresh interval of c_i, we say that q_i timely refreshes c_i.* ∎

If c_i is timely refreshed by q_i, then we say that all refresh requests of c_i have been *timely satisfied*. Otherwise, for a refresh request $r_{i,j}$ raised at time $t_{i,j}$, if no execution of q_i is performed within the interval $[t_{i,j}, t_{i,j} + U_i]$, we say that $r_{i,j}$ has been *missed* and is not timely satisfied. Note that the *deadline* for $r_{i,j}$ is the time instant $t_{i,j+1} = t_{i,j} + U_i$ when the next request $r_{i,j+1}$ is raised. We say that a refresh request $r_{i,j}$ is *pending* at time t if $t \leqslant (t_{i,j+1} - E_{q_i})$ and no query q_i is executed during $[t_{i,j}, t]$, where E_{q_i} is the execution time of q_i. After time instant $t_{i,j+1} - E_{q_i}$, the pending request $r_{i,j}$ will be discarded by the scheduler and is missed.

Definition 5 (Refresh Pattern) *We call the update pattern $\langle U_i, t_{i,1} \rangle$ of the feature table of cell c_i the refresh pattern of c_i.*

Since we are concerned with refreshing a set of cells materialized in a cellbase, we now define some concepts associated with a cell set. Let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of cells in a cellbase

where for each c_i, $1 \leqslant i \leqslant n$, there exists a refresh query q_i that takes E_{q_i} time units to execute to yield c_i. Let $\Phi(q_i)$ denote the result set of query q_i. Then, $\Phi(q_i) = \{c_i\}$, $1 \leqslant i \leqslant n$, is singleton, and we say that q_i is *atomic*. We define a *complex* refresh query p as one whose result set has more than one element (i.e., $|\Phi(p)| \geqslant 1$) and that takes E_p time units to execute. We emphasize here that the result of an atomic refresh query can be used to refresh only one cell whereas the result of a complex query can be decomposed and distributed to refresh multiple cells, i.e., the result of the complex query *covers* these cells. In this work, we are not concerned with the detail of how the result of a complex query is distributed to multiple cells. We assume that such a distribution can be performed with negligible cost.

Definition 6 (Candidate Query Set) *Given a cell set C and a query set $Q(C)$, if the result of $Q(C)$ covers all cells in C, i.e., $\Phi(Q(C)) = \bigcup_{q \in Q(C)} \Phi(q) = C$, we say $Q(C)$ is a candidate query set for C.* ∎

We refer to the initial refresh query set $Q_0(C) = \{q_1, q_2, \ldots, q_n\}$ of C where q_i's are atomic as a *trivial* candidate query set for C. To save database access and to reduce processor usage, we wish that candidate query set includes as less elements as possible. Since the result of one complex query may cover multiple cells, a candidate query set involving complex queries may have less elements than a trivial candidate query set. We discuss candidate query sets that include complex queries in [6].

Definition 7 (Refresh Pattern Set) *The collection of refresh patterns of all cells in C can be represented as $R(C) = \{r_i | r_i = \langle U_i, t_{i,1} \rangle, 1 \leqslant i \leqslant n\}$, then we say $R(C)$ is the* refresh pattern set *for C.* ∎

3 Scheduling Algorithm

In our work, a *scheduling algorithm* is a set of rules that determine the refresh query to be executed at a particular moment. Given a set C of cells with refresh pattern set $R(C)$ and candidate query set $Q(C)$, the output of a scheduling algorithm is a *schedule*. Let $\mathbb{N}^+$ denote the set of non-negative numbers and $\mathcal{R}$ denote the set of refresh requests sets, a schedule is defined as a mapping $S : \mathcal{R} \times \mathbb{N}^+ \rightarrow \mathbb{N}^+$ such that for each $t \in \mathbb{N}^+$ when processor is idle and $\mathcal{R}_t \in \mathcal{R}$ is the set of pending refresh requests at time t, then $S(\mathcal{R}_t, t) = i$ means that query q_i is scheduled at time t to satisfy the request $r_{i,j} \in \mathcal{R}_t$ from c_i. Intuitively, $S(\mathcal{R}_t, t) = 0$ indicates that $\mathcal{R}_t = \varnothing$ and the processor is idle. We say that a schedule is *feasible* if all refresh requests of C can be timely satisfied by scheduling queries in $Q(C)$. Below, we give an example of a cell set C with a feasibly-scheduled candidate query set.

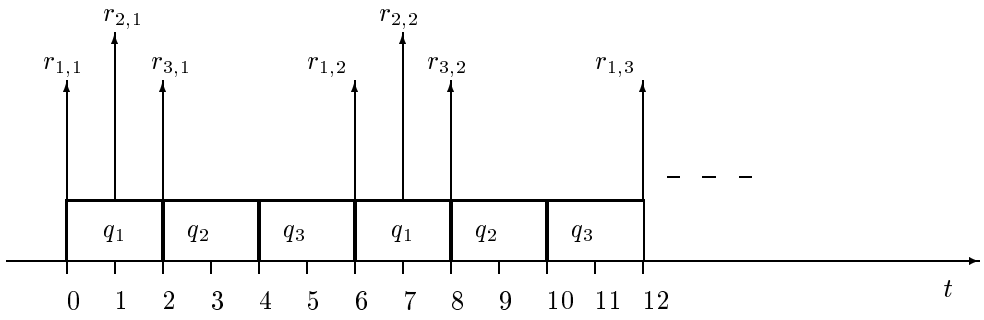

Figure 2: A feasible candidate query set.

Example 1 Let $C = \{c_1, c_2, c_3\}$, $Q_0(C) = \{q_1, q_2, q_3\}$, $R(C) = \{\langle 6, 0\rangle, \langle 6, 1\rangle, \langle 6, 2\rangle\}$, $E_{q_1} = E_{q_2} = E_{q_3} = 2$. As shown in Figure 2, in each refresh intervals of c_1, c_2 and c_3 the corresponding refresh queries q_1, q_2 and q_3 can be successfully executed. Thus, $Q_0(C)$ is feasible. ∎

The main task of a scheduling algorithm is to determine which refresh query should be executed when the processor is idle and there are several pending requests such that a feasible schedule can be made. In our work, all scheduling algorithms are *stationary*.

Definition 8 (Stationary Schedule) *A scheduling algorithm is* stationary *if it schedules the same query to execute at each time instant when the same permutation of pending refresh requests appears.* ∎

Note that two permutations of a set of refresh requests are the same, if

(i) the cells raising the refresh requests are the same,

(ii) the raised order of requests are the same, and

(iii) the time intervals between two neighboring requests are the same.

Suppose at time t, there are two pending refresh requests $r_{i,u}$ and $r_{j,v}$ from cell c_i and c_j respectively (assume $r_{i,u}$ is raised before $r_{j,v}$) and that a scheduling algorithm S determines that q_i should be executed at t. Suppose at $t + T$, there are two pending requests $r_{i,u'}$ and $r_{j,v'}$ from c_i and c_j (c_i raises the request before c_j) and the processor is idle, the executed query determined by S at $t+T$ is still q_i rather than q_j. Thus, S is a stationary scheduling algorithm and it produces a stationary schedule. Otherwise, if at time $t + T$ q_j is executed instead, then S is not stationary.

We have some properties for a schedule:

(i) $S(\varnothing, t) = 0$, i.e., the processor continues to be idle at time t if there is no pending request at t.

(ii) $S(\mathbb{R}_{t_1}, t_1) = S(\mathbb{R}_{t_2}, t_2)$ if the permutations of $\mathbb{R}_{t_1}$ and $\mathbb{R}_{t_2}$ are the same (stationary property).

A scheduling algorithm can be *offline* or *online*. An offline scheduling (analysis) has full knowledge of the state of the refresh system including the refresh pattern of a cell set and the execution times of refresh queries. It is always executed before the real schedule is performed at run-time. An online scheduling, on the other hand, does not know when an execution of refresh query would be completed if it has been performed by the scheduler.

Our algorithms are based on the modern real-time system where tasks are scheduled in a priority manner. At any point in time, the ready job with the highest priority executes. Most systems use a fixed priority assignment according to which all jobs in a task have the same priority. Examples of fixed priority policies are *Rate Monotonic* (RM) [2] or *Deadline Monotonic* (DM) [7]. The priority of a task under RM is proportional to the rate at which jobs in the task are released, while the priority of a task under DM is inversely proportional to the relative deadline of the task. Priorities may also be assigned dynamically. The most common dynamic priority scheduling policy is *Earliest Deadline First* (EDF) [2] which assigns priorities to jobs in order of their absolute deadlines.

In our work, a *refresh manager* working in a autonomously-pull mode sequentially executes refresh queries to refresh the cellbase [5]. When a cell needs to be refreshed periodically, the scheduler periodically initiates refresh requests to initiate itself to perform corresponding refresh query. Refresh requests may have four different states: PENDING, FUTURE, SATISFIED and MISSED at different time instants. If the initiating time of a request is later than the current time, then its state is FUTURE. If the initiating time of a request is earlier than the current time and the initiating time of the next request coming from the same cell is later than the current time, then the state of the request is PENDING. If a request has been satisfied by executing a corresponding query completed before the deadline (the time when the next request is initiated), then it is SATISFIED, otherwise, it is MISSED. Certainly, for a set of cells with random refresh requirements, the manager cannot guarantee to refresh all cells in time by working with a specific scheduling algorithm. That is, some refresh requests of the cells may be missed.

We propose several online scheduling algorithms, which include both fixed and dynamic priority assignment policies, to compare their performance for refreshing a set of cells with specific refresh requirements. The algorithms are common in deploying a list to store the pending refresh requests. The scheduler always chooses and fetches one pending request from the list according to the priority assignment policy and performs a corresponding query to satisfy the request. After the execution of the query has been completed, the state of the request is changed to SATISFIED or MISSED according to the timing constraint. And the scheduler will also change the state of all other requests. Previous pending requests may become MISSED and previous future requests may become

PENDING and are required to be inserted into the pending requests list. The scheduler will wait if there is no any pending requests until some future requests become PENDING with time elapsed. The algorithms will stop running if no more refresh requests will be raised in the future. The running steps of algorithms is shown in Figure 3. The difference among the algorithms lies in the policy to choose one pending request to refresh the corresponding cell. We describe them in detail as follows:

Earliest Deadline First (EDF) This is a traditional scheduling algorithm in real-time sporadic task systems. The algorithm always chooses the pending request whose deadline is the nearest among all pending requests in the list. Here the deadline for a request equals to the initiating time of the next request from the same cell.

Shortest Refresh Period First (SRPF) The algorithm always chooses the pending request initiated from a cell which has the shortest refresh period among all cells which have pending refresh requests in the current list. Clearly, it is equivalent to the traditional Rate Monotonic algorithm.

Longest Refresh Period First (LRPF) In a contrarily manner to SRPF, the algorithm always chooses the pending request initiated from a cell which has the longest refresh period among all cells which have pending refresh requests in the current list.

Minimal Execution Time First (MiEF) The above three algorithms do not consider the impact of execution time of refresh query which really plays a great role in scheduling result since all refresh query are competing to occupy the single processor. Thus, MiEF gives the refresh priority to the pending request which can be satisfied by the execution of a refresh query whose execution time is the shortest among the refresh query set of current pending request list. All execution times of refresh queries are computed when they are executed at the first time.

Maximal Execution time First (MaEF) Contrary to MiEF, MaEF assigns the highest priority to the request which is satisfied by the refresh query having the longest execution time among the refresh query set of pending request list.

4 Freshness

So far, if a candidate query set is feasible, then the refresh queries can be successfully scheduled to timely refresh the corresponding cells. However, after an update of the feature table of a cell, "timely refreshing the cell" can be started earlier or later providing that the refresh can be finished

```
at starting time, set the first refresh requests of all cells PENDING;
set other refresh requests FUTURE;
while (true) {
    if pending requests list is not empty {
        fetch the pending request determined by policy of the algorithm;
        executing the corresponding refresh query;
        change the state of the request to SATISFIED or MISSED;
        remove the request from the pending requests list;
        update the state of all pending and future requests;
    } else {
        if there is no more future refresh requests
            break;
        wait until the time when the nearest future request is initiated;
        insert the request into pending requests list;
    }
}
```

Figure 3: Online scheduling algorithms for websites refresh

before the next update of the feature table. The earlier the refresh starts, the fresher the cell is. To measure "how fresh" a single cell (and a cellbase) is (are), we formally define a *freshness* metric in this section.

4.1 Cell Freshness of a Refresh Interval

Definition 9 (Cell Freshness of A Refresh Interval) *For cell c_i that is to be refreshed by query q_i with refresh pattern $\langle U_i, t_{i,1}\rangle$, its freshness $F_{i,j}$ achieved in the jth refresh interval $[t_{i,j}, t_{i,j+1}]$ is defined as follows:*

$$F_{i,j} = \begin{cases} \frac{1}{U_i}(t_{i,j+1} - s_j) & \text{if } q_i \text{ starts the execution at } s_j \text{ and ends at } e_j, \text{ where } s_j \geqslant t_{i,j} \text{ and } e_j \leqslant t_{i,j+1} \\ 0 & \text{otherwise} \end{cases}$$

∎

From Eq. 1, we know that $t_{i,j+1} = t_{i,1} + jU_i$, thus we have

$$F_{i,j} = \frac{t_{i,1}}{U_i} + j - \frac{1}{U_i}s_j$$

when $s_j \geqslant t_{i,j}$ and $e_j \leqslant t_{i,j+1}$, i.e., $s_j \geqslant t_{i,1} + (j-1)U_i$ and $s_j + E_{q_i} \leqslant t_{i,1} + jU_i$. From the definition above, we see that the cell freshness of a refresh interval is always between 0 and 1.

Lemma 1 *For cell c_i with refresh pattern $\langle U_i, t_{i,1}\rangle$, if its jth refresh request is timely satisfied by executing the refresh query q_i, then we have*

$$\frac{E_{q_i}}{U_i} \leqslant F_{i,j} \leqslant 1$$

where E_{q_i} is the execution time of q_i. ∎

The proof follows Definition 9, we shall not show it here.

According to the timeliness of response to the updates of feature table, we classify the freshness of a cell into two levels: *tight freshness* and *loose freshness*. We say that a cell c_i is *tight fresh* during its jth refresh interval if $F_{i,j} = 1$. This means that its refresh query can be instantaneously executed at the time when the feature table finishes an update. The tight freshness is required when users are critical to the timeliness of information presented on the Web. They hope that Web information can be refreshed as soon as possible in response to an update on the base data. Web pages that present the real-time stock exchange information are examples of such requirements. On the other hand, we say that c_i is *loose fresh* during its jth refresh interval if $0 < F_{i,j} < 1$. In this case, the execution of its refresh query may be started after the beginning of the interval and finished before the end of the interval. Clearly, if $F_{i,j} = 0$ then c_i is not fresh in the refresh interval $[t_{i,j}, t_{i,j} + U_i]$ and the refresh request $r_{i,j}$ initiating the interval has been missed. Note that in Example 1, if the cells are required to be tight fresh, then the candidate query set is infeasible.

In Example 1, according to Definition 9, $F_{1,1} = (6-0)/6 = 1$, $F_{2,1} = (7-2)/6 = 0.83$ and $F_{3,1} = (8-4)/6 = 0.67$. Actually, we can observe from the schedule in Figure 2 that c_1 will always achieve a freshness of 1 during all its refresh intervals. Cell c_3 achieves a smaller value of freshness than c_2 because q_3 has to wait for two units of time to execute after the feature table of c_3 has been updated once at time 2 whereas q_2 only waits for one unit of time to execute after an update on c_2's feature table at time 1.

4.2 Freshness over A Period of Time

When a set of refresh queries have been scheduled to refresh a set of cells, we are concerned with the freshness achieved over a period of time rather than the freshness of a refresh interval. Thus, we define freshness over a period of time as follows:

Definition 10 (Interval Freshness) *Given a cellbase C where each cell c_i $(1 \leqslant i \leqslant n)$ has refresh pattern $\langle U_i, t_{i,1}\rangle$, we define the freshness of c_i over any arbitrary time interval $[t_1, t_2)$ as:*

$$F_i(t_1, t_2) = \sum_{j=u}^{v} F_{i,j}/(u - v + 1),$$

where $r_{i,u}$ and $r_{i,v}$ are the first and last refresh requests of c_i raised within $[t_1, t_2)$ respectively. The overall cellbase freshness achieved in $[t_1, t_2)$ is defined as

$$F_C(t_1, t_2) = \frac{1}{n} \sum_{i=1}^{n} F_i(t_1, t_2).$$

∎

To describe a special time interval which is the basic unit of steady period introduced in Theorem 2, we define it as *refresh cycle* of a cellbase.

Definition 11 (Cellbase Refresh Cycle) *For a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C) = \{\langle U_1, t_{1,1}\rangle, \langle U_2, t_{2,1}\rangle, \ldots, \langle U_n, t_{n,1}\rangle\}$, if L is the least common multiple of U_1, U_2, ..., U_n, i.e., $\mathrm{lcm}(\mathrm{U_1}, \mathrm{U_2}, \ldots, \mathrm{U_n}) = \mathrm{L}$, then we say that any time interval $[t, t+L)$ (t is any non-negative integer) is a refresh cycle of C.* ∎

Lemma 2 *Given a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C) = \{\langle U_1, t_{1,1}\rangle$, $\langle U_2, t_{2,1}\rangle$, ..., $\langle U_n, t_{n,1}\rangle\}$, if the time axis from time t is divided onwards into continuous refresh cycles $[t+kL, t+(k+1)L)$, where k is any non-negative integer and $L = \mathrm{lcm}(\mathrm{U_1}, \mathrm{U_2}, \ldots, \mathrm{U_n})$, then the permutation of refresh requests raised in each refresh cycle is the same.*

Proof: Consider any refresh request $r_{i,j}$ raised at $t_{i,j}$ within $[t, t+L)$, we only need to prove that refresh requests of c_i are also raised at instants $t_{i,j} + kL$, $k is any positive integer$, within the corresponding refresh cycles $[t+kL, t+(k+1)L)$. This is true because L is a multiple of any U_i and c_i is raising its refresh requests with period U_i. ∎

Theorem 2 *For a cellbase C (when refresh queries have been scheduled to refresh cells), there must exist a time instant t_s and an integer Y such that some refresh requests are raised at t_s, and starting from t_s, freshness of C over each time period with length Y is the same, i.e., $F_C(t_s+kY, t_s+(k+1)Y)$ is the same for any non-negative integer k. We call t_s the* steady time instant *and Y the* length of steady period.

Proof: To prove that such a steady time instant t_s really exists, for a time instant t when there are raised requests, we first observe properties at t that may affect the behavior of a scheduling algorithm. We list all *scheduling properties* at t as follows:

(i) Permutation of pending refresh requests at t.

(ii) Processor state. Is processor idle at t? If processor is busy at t, then how much time is left for it to complete the current execution ?

(iii) Permutation of refresh requests raised after t.

For a stationary scheduling algorithm, if we can find two time instants t_1 and t_2 such that all properties mentioned above at these two time instants are entirely the same, then the scheduling decisions made from t_1 is totally same as the scheduling decisions made from t_2. That is, the permutation of scheduled queries during $[t_1, t_2)$ is the same as the permutation of scheduled queries during $[t_2, t_2 + (t_2 - t_1))$. Clearly, this leads to that scheduling properties at $t_2 + (t_2 - t_1)$ are also the same as those at t_1 and t_2. Repeatedly, we can find infinite time instants $t_2 + k(t_2 - t_1)$, k is an integer and $k \geqslant 2$, which have the same scheduling properties. We say these time instants have the same *steady property*.

Therefore, according to Definition 9 and Definition 10, we have $F_i(t_1, t_2) = F_i(t_2 + k(t_2 - t_1), t_2 + (k+1)(t_2 - t_1))$ for any cell c_i and $F_C(t_1, t_2) = F_C(t_2 + k(t_2 - t_1), t_2 + (k+1)(t_2 - t_1))$, k is any non-negative integer. Thus, if t_1 is the earliest time instant which has the steady property, then the steady time instant $t_s = t_1$ and the steady length $Y = t_2 - t_1$.

To prove our theorem, our task now is to find two time instants t_1 and t_2 which have the same steady property. Based on Lemma 2, we know that for any two time instants t_a and t_b which have a distance of one or more refresh cycles, the permutations of refresh requests during $[t_a, t_b)$ and $[t_b, t_b + (t_b - t_a))$ are the same. Because the number of cells in the cellbase is finite, the number of cases for possible permutations of pending requests at a time instant is also finite. Likewise, as the execution time of each query is constant and finite, the number of cases for waiting time to be idle at a time instant is finite also. However, our scheduling algorithm runs for an infinite time, thus after a sufficient long time, there must be two time instants t_1 and t_2 which have a distance of one or more refresh cycles and have the same scheduling properties.

There are infinite time instants having the same steady property, we take the earliest time instant as our steady time instant t_s and the length between t_s and the next instant having the same property as the steady length Y. Therefore, we have $F_C(t_s + kY, t_s + (k+1)Y)$ is the same for any non-negative integer k. ∎

Let's illustrate the steady property introduced above with the following example:

Example 2 Give a cellbase $C = \{c_1, c_2, c_3\}$ with refresh pattern set $R(C) = \{\langle 5, 0\rangle, \langle 5, 1\rangle, \langle 5, 2\rangle\}$ and trivial candidate query set $Q_0(C) = \{q_1, q_2, q_3\}$ where $E_{q_1} = 4$, $E_{q_2} = 2$ and $E_{q_3} = 2$, if we schedule the refresh queries with the EDF algorithm, we obtain the sequence of executed queries as shown in Figure 4. We can see that time instants $5 + 10k$, k is any non-negative integer, have the same steady property. Lets compare the scheduling properties at time instants 5 and 15 respectively. At time instant 5,

(i) the permutation of pending requests is $r_{1,2}$,

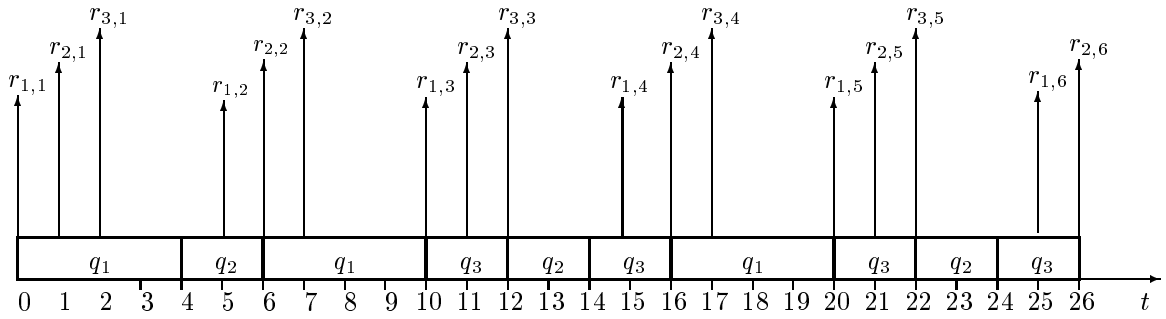

Figure 4: The sequence of executed queries in Example 2.

(ii) the processor is busy at 5 and it will be idle at 6,

(iii) the permutation of pending requests from 5 to 15 is $r_{1,2}$, $r_{2,2}$, $r_{3,2}$, $r_{1,3}$, $r_{2,3}$, $r_{3,3}$.

While at time instant 15,

(i) the permutation of pending requests is $r_{1,4}$,

(ii) the processor is busy at 15 and it will be idle at 16,

(iii) the permutation of pending requests from 15 to 25 is $r_{1,4}$, $r_{2,4}$, $r_{3,4}$, $r_{1,5}$, $r_{2,5}$, $r_{3,5}$.

Thus, scheduling properties at 5 and 15 are the same and we cannot find a time instant earlier than 5 which has the steady property. Therefore, 5 is the steady time instant and $15-5=10$ is the length of steady period. We have that $F_C(5+10k, 5+10(k+1))$ is the same for any non-negative integer k. ∎

Corollary 1 *The length Y of steady period of a cellbase is the same as or multiple of the length L of refresh cycle of the cellbase, i.e., $Y = kL$ where k is a positive integer.* ∎

From Theorem 2, the freshness of a cellbase is the same over each steady period after entering the steady time instant. Thus, to compare the freshness achieved by different scheduling algorithms, it is sufficient to only measure the freshness achieved over the first steady period. We explicitly define the metric as follows:

Definition 12 (Steady Freshness) *Given a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C)$ and candidate query set $Q(C)$, if t_s is the steady time instant and Y is the length of steady period while refresh queries are scheduled by a stationary algorithm, then the steady cell freshness F_i for c_i is defined as $F_i = F_i(t_s, t_s + Y)$ and the steady cellbase freshness $F(C)$ is defined as $F(C) = F_C(t_s, t_s + Y)$.* ∎

The following lemma says that the steady time instant is always found when the processor is idle if the candidate query set can be scheduled to be loose feasible.

Lemma 3 *Given a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C)$ and candidate query set $Q(C)$, if $Q(C)$ can be feasibly scheduled to keep all cells loose fresh, then the processor is idle at the steady time instant t_s.*

Proof: Assume that the processor is busy at t_s. Let Y be the length of steady period, then t_s and $t_s + Y$ have the same steady property, i.e., the processor is busy at both instants and needs the same length time to be idle, the permutation of pending requests is the same at both instants and the permutations of requests within $[t_s, t_s + Y)$ and $[t_s + Y, t_s + 2Y)$ are the same too.

Suppose that t_1 is the nearest idle instant before t_s where no query is being executed and at least one request is raised and t_2 is the time instant when the query executed at t_s completes its execution. Thus, time interval $[t_1, t_2]$ is spent to execute queries to satisfy all requests raised during $[t_1, t_s)$ since no request is missed. After Y time units, the permutation of requests during $[t_1 + Y, t_s + Y)$ is the same as during $[t_1, t_s)$ because Y is the multiple of refresh cycle of C. Since we have a feasible schedule and all requests should be satisfied, we need the same length of time as $t_2 - t_1$ to satisfy requests raised during $[t_1 + Y, t_s + Y)$. Since $t_s + Y$ has the same property as t_s and the processor should be idle at $t_s + Y + (t_2 - t_s) = t_2 + Y$, the processor must be idle at $t_1 + Y$. Otherwise, there is no sufficient time to satisfy all requests raised during $[t_1 + Y, t_s + Y)$.

Therefore, the processor is idle at both t_1 and $t_1 + Y$. Moreover, the permutations of requests during $[t_1, t_1 + Y)$ and during $[t_1 + Y, t_1 + 2Y)$ are the same too. That is to say, t_1 and $t_1 + Y$ has the same scheduling property and they are the time instants having the same steady property. However, a contradiction arises while t_1 is earlier than t_s and t_s is the steady time instant. Thus, the assumption is untenable and the lemma does hold. ∎

Theorem 3 *Given a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C)$ and candidate query set $Q(C)$, if $Q(C)$ can be feasibly scheduled to keep all cells loose fresh, then the length Y of steady period is the same as the length L of refresh cycle of C, i.e., $F(C) = F_C(t_s, t_s + L)$ where t_s is the steady time instant and $L = \mathrm{lcm}(\mathrm{U}_1, \mathrm{U}_2, \ldots, \mathrm{U}_n)$.*

Proof: We need to prove that the processor has the same scheduling properties at t_s and $t_s + L$, that is:

(i) The permutations of pending requests at both instants are the same.

(ii) The processor at both instants is either idle or busy with the same busy length left.

(iii) The permutations of requests raised during $[t_s, t_s + L)$ and $[t_s + L, t_s + 2L)$ are the same.

From Lemma 2, we know the third property above is clearly true. And also we know from Lemma 3, the processor must be idle at t_s. Thus, we only need to prove that:

(i) The permutations of pending requests at both instants are the same. Suppose the permutations of pending requests at two instants are denoted respectively as $P(t_s)$ and $P(t_s + L)$, then we need to prove $P(t_s) = P(t_s + L)$.

(ii) The processor is idle at $t_s + L$.

We assume the above two properties are not true, then we have the following possible cases:

(i) Assume that the processor is idle at $t_s + L$ while $P(t_s) \neq P(t_s + L)$. Because processor is idle at t_s and $t_s + L$, $P(t_s)$ and $P(t_s + L)$ are respectively composed of requests raised at t_s and $t_s + L$. However, according to Lemma 2, the requests raised at t_s and $t_s + L$ should be totally the same, i.e., $P(t_s) = P(t_s + L)$. Thus, the assumption does not hold.

(ii) Assume that the processor is busy at $t_s + L$ while $P(t_s) = P(t_s + L)$. Thus, for satisfying requests raised during $[t_s, t_s + L)$, the requirement for processor usage is more than L time units. Since permutations of requests during refresh cycles $[t_s + L, t_s + 2L)$, $[t_s + 2L, t_s + 3L)$, ... are the same as the permutation during $[t_s, t_s + L)$ and no requests are missed, the same processor usage is required during each cycle, then processor should be busy at $t_s + 2L$, $t_s + 3L$, ... and it won't be idle at any instant $t_s + kL$ where k is a positive integer. Thus, there is no any time instant having the same scheduling properties as t_s. This is a contradiction with the fact that t_s is the steady time instants. So the assumption does not hold.

(iii) Assume that the processor is busy at $t_s + L$ while $P(t_s) \neq P(t_s + L)$. Clearly, pending requests at $t_s + L$ are more than those at t_s since pending requests at $t_s + L$ include not only requests raised at $t_s + L$ which are the same as pending requests at t_s but also those raised before $t_s + L$. We denote those pending requests at $t_s + L$ raised before $t_s + L$ as $P'(t_s + L)$. Then, more than L time units are used to satisfy the requests raised during $[t_s, t_s + L)$ excluding those in $P'(t_s + L)$. Similarly as the second case, we know that no time instant $t_s + kL$ where k is a positive integer has the same scheduling properties as t_s. This is a contradiction with the fact that t_s is the steady time instants. Therefore, the assumption does not hold.

Based on the above proof, the initial assumption does not hold. The theorem is proved. ∎

Corollary 2 *Given a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C)$ and candidate query set $Q(C)$, if $Q(C)$ can be feasibly scheduled to keep all cells loose fresh, then for all scheduling algorithms which can feasibly schedule $Q(C)$, the first steady periods start at the same instant, i.e., their steady time instants are the same.*

For cellbase C in Example 1, the steady time instant is 0 and the length of steady period is $Y = L = \text{lcm}(6, 6, 6) = 6$, thus we have $F(C) = F_C(0, 6) = \frac{1}{3}(F_1(0, 6) + F_2(0, 6) + F_3(0, 6)) = \frac{1}{3}(F_{1,1} + F_{2,1} + F_{3,1}) = \frac{1}{3}(1 + 0.83 + 0.67) = 0.83$.

5 Feasibility, Scheduling Algorithm, and Freshness

When the cellbase is required to be tight fresh, if there exist one scheduling algorithm which schedules the executions of refresh queries to make all cells in the cellbase tight fresh in their every refresh intervals, then this set of queries is *tight* feasible. Likewise, when the cellbase is required to be loose fresh, we say a candidate query set is *loose feasible* if queries in the set can keep all cells loose fresh in their every refresh intervals when they are scheduled by some algorithm. Simply to say, if a candidate query set is loose feasible, then it can be schedulable without any refresh requests missed. Clearly, a tight feasible candidate query set must be also loose feasible and a tight infeasible candidate query set may still be loose feasible. However, the converse does not hold. Note that in the rest of the paper, without additional notes we are referring to a loose infeasible candidate query set when we talk about an infeasible candidate query set. We have studied the feasibility determination problem in [4].

For a loose feasible candidate query set, it is possible that some scheduling algorithms may be unable to feasibly schedule queries whereas there must exist one scheduling algorithm which can produce a feasible schedule for queries.

Example 3 Give a cell set $C = \{c_1, c_2, c_3\}$ with refresh pattern set $R(C) = \{\langle 8, 0\rangle, \langle 4, 1\rangle, \langle 8, 2\rangle\}$, and candidate query set $Q_0(C) = \{q_1, q_2, q_3\}$ where $E_{q_1} = 2$, $E_{q_2} = 1$, $E_{q_3} = 3$, if $Q_0(C)$ is scheduled by the *Maximal Execution time First* (MaEF) algorithm, then cells cannot be kept loose fresh (the refresh request $r_{2,1}$ is missed, see Figure 5). However, if $Q_0(C)$ is scheduled by EDF algorithm, then all cells can be kept loose fresh and no refresh requests are missed. Thus, $Q_0(C)$ is loose feasible. ∎

We say that a scheduling algorithm is *global* if it can produce feasible schedules for all feasible candidate query sets. According to [11], we know that EDF algorithm is global for candidate query sets.

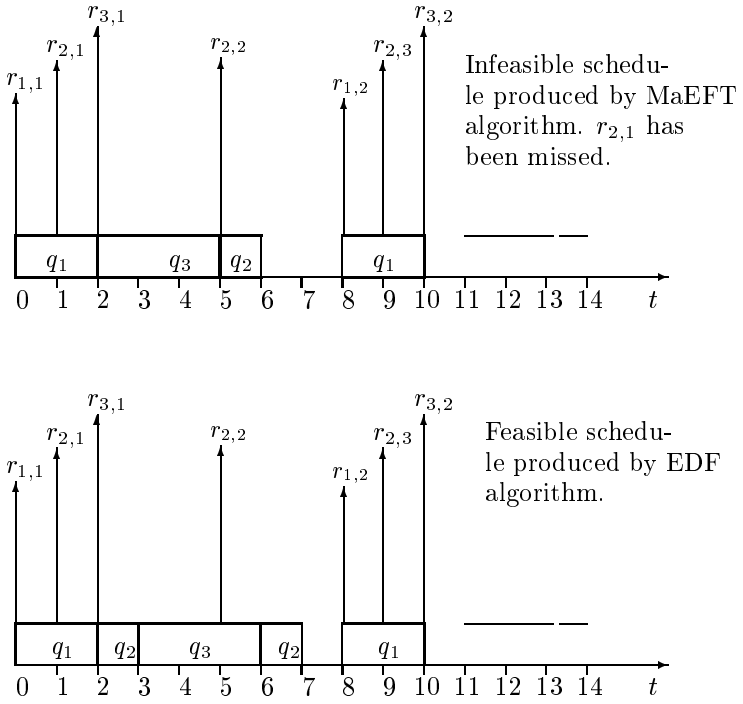

Figure 5: Two different schedules of $Q_0(C)$ in Example 3 produced by two algorithms.

Theorem 4 *If candidate refresh query set $Q(C)$ is loose feasible for a cell set C with refresh pattern set $R(C)$, then EDF scheduling algorithm can feasibly schedule the executions of queries in $Q(C)$.* ∎

Theorem 5 *If a candidate query set $Q(C) = \{q_1, q_2, \ldots, q_n\}$ is feasible to keep the set C of cells with refresh pattern set $R(C) = \{\langle U_1, t_{1,1}\rangle, \langle U_2, t_{2,1}\rangle, \ldots, \langle U_n, t_{n,1}\rangle\}$ loose fresh, then $\frac{1}{n}\sum_{i=1}^{n}\frac{E_{q_i}}{U_i} \leqslant F(C) \leqslant 1$.*

Proof: If $Q(C)$ is loose feasible, then any cell $c_i \in C$ $(1 \leqslant i \leqslant n)$ can be kept loose fresh. Thus, according to Lemma 1, we have $\frac{E_{q_i}}{U_i} \leqslant F_{i,j} \leqslant 1$ for any j. Then, according to Definition 12, $\frac{E_{q_i}}{U_i} \leqslant F_i \leqslant 1$. Therefore, we have $\frac{1}{n}\sum_{i=1}^{n}\frac{E_{q_i}}{U_i} \leqslant F(C) \leqslant 1$ since $F(C) = \frac{1}{n}\sum_{i=1}^{n} F_i$. ∎

The theorem below says that value 1 of cellbase freshness can be achieved by a tight feasible candidate query set.

Theorem 6 *If a candidate query set $Q(C)$is feasible to keep the set C of cells tight fresh, then $F(C) = 1$.*

Proof: If $Q(C)$ is tight feasible, then for any i, j, we have $F_{i,j} = 1$, thus according to Definition 12, $F_i = 1$. Therefore, we have $F(C) = 1$ since $F(C) = \frac{1}{n}\sum_{i=1}^{n} F_i$. ∎

An important factor to affect the freshness of the cellbase is the scheduling algorithm deployed by the refresh scheduler. Let us illustrate it with an example below.

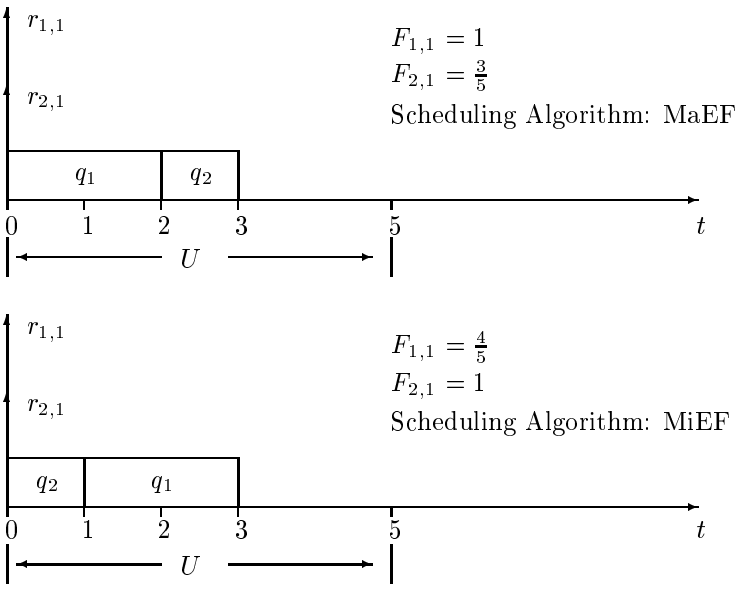

Figure 6: Queries in $Q(C)$ in Example 4 are scheduled by two different algorithms.

Example 4 Let $C = \{c_1, c_2\}$ be a cellbase with refresh pattern set $R(C) = \{\langle 5, 0\rangle, \langle 5, 0\rangle\}$ and candidate query set $Q(C) = \{q_1, q_2\}$ where $E_{q_1} = 2$ and $E_{q_2} = 1$. If $Q(C)$ is scheduled by MaEF algorithm, the first steady period is $[0, 5)$, $F_1 = F_1(0, 5) = F_{1,1} = 1$ and $F_2 = F_2(0, 5) = F_{2,1} = \frac{3}{5}$, then $F(C) = \frac{1}{2}(1 + \frac{3}{5}) = 0.8$. However, if $Q(C)$ is scheduled by MiEF algorithm, although the first steady period is still $[0, 5)$, $F_{1,1} = \frac{4}{5}$, $F_{2,1} = 1$, then $F(C) = \frac{1}{2}(1 + \frac{4}{5}) = 0.9$ (Figure 6) and the freshness has been improved. ∎

However, for a special group of cells which have the same refresh period and whose refresh queries have the same length of execution time, we show in the theorem below that the freshness of cell set will not be altered when different scheduling algorithms are deployed if the refresh query set is loose feasible.

Theorem 7 *Given a cellbase* $C = \{c_1, c_2, \ldots, c_n\}$ *with refresh pattern set* $R(C) = \{\langle U_1, t_{1,1}\rangle, \langle U_2, t_{2,1}\rangle, \ldots, \langle U_n, t_{n,1}\rangle\}$ *where* $U_1 = U_2 = \ldots = U_n = U$*, if its candidate query set* $Q(C) = \{q_1, q_2, \ldots, q_n\}$ *where* $E_{q_1} = E_{q_2} = \ldots = E_{q_n} = E$ *is loose feasible, then the cellbase freshness* $F(C)$ *is the same for all scheduling algorithms which feasibly schedule* $Q(C)$ *.*

Proof: We would prove that for any two different scheduling algorithms S_1 and S_2, if they both feasibly schedule $Q(C)$ and S_1 achieves the freshness $F(C)_1$ and S_2 achieves the freshness $F(C)_2$, then $F(C)_1 = F(C)_2$. This is clearly true if $Q(C)$ is tight feasible according to Theorem 6. So we only consider the case when $Q(C)$ is not tight feasible but loose feasible.

The difference between algorithms lies in the decision policies they deploy to decide which request should be satisfied by executing a query when multiple requests are pending at a time instant. We will prove that although S_1 and S_2 have different decision policies, we have $F(C)_1 = F(C)_2$. From Corollary 2, S_1 and S_2 have the same first steady period $[t_s, t_s + L)$. Suppose that c_i raises refresh request $r_{i,i'}$ $(1 \leqslant i \leqslant n)$ during $[t_s, t_s + U)$, since $U_1 = U_2 = \ldots = U_n = U$, we have $L = U$ and $F(C) = \frac{1}{n}\sum_{i=1}^{n} F_{i,i'}$. If we can prove that for S_1 and S_2 the value of $\sum_{i=1}^{n} F_{i,i'}$ is the same, then we would have $F(C)_1 = F(C)_2$.

Assume that for any two requests $r_{u,u'}$ and $r_{v,v'}$ pending at a time instant $t_p \in [t_s, t_s + U)$, S_1 schedules to execute q_u at time t_1 and execute q_v at time t_2 $(t_2 > t_1)$ whereas S_2 executes q_u at time t_2 and q_v at t_1 respectively. (Note that if $E_{q_u} \neq E_{q_v}$, this exchanging of execution order may not be feasible). According to Definition 9, for both S_1 and S_2, we have

$$F_{u,u'} + F_{v,v'} = \frac{1}{U}(t_{u,u'} + t_{v,v'} + 2U - t_1 - t_2).$$

Since $r_{u,u'}$ and $r_{v,v'}$ is freely selected, we have $\sum_{i=1}^{n} F_{i,i'}$ is the same for S_1 and S_2. Thus, $F(C)_1 = F(C)_2$. ∎

Another factor which can improve the cellbase freshness is the cardinality of candidate query set. If we can reduce the number of queries in the candidate query set of the cellbase, performing less queries would be expected to refresh the cells. Intuitively, we know that more cells can be refreshed earlier since less queries are competing to occupy the scheduler. Thus, the cellbase freshness would be improved.

We would analyze and compare the performance on cellbase freshness of several scheduling algorithms in detail in the next section and develop an optimization technique to reduce the cardinality of candidate query set in [6].

6 Comparison Study on Scheduling Algorithms

6.1 Introduction

In this section, we address the issue of improving the cellbase freshness by applying an appropriate online scheduling algorithm. Since we have shown that the maximal freshness 1 can be achieved by scheduling a tight feasible candidate query set, we only consider the case that the candidate query set is not tight feasible, namely, it is either loose feasible or loose infeasible. However, we must note that obtaining a scheduling algorithm that maximizes the overall cellbase freshness is a difficult problem. Even for a *clairvoyant* scheduler, i.e, one that knows all the parameters of the refresh pattern set and the candidate query set, the problem of finding the maximum freshness can be shown NP-Hard. Before proving this result, we first observe the load of executed queries incurred by requests within the steady period of a cellbase.

Lemma 4 *Given a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with candidate query set $Q(C) = \{q_1, q_2, \ldots, q_n\}$, if for each cell c_i $(1 \leqslant i \leqslant n)$, the refresh query q_i is scheduled n_i times within the steady period $[t_s, t_s + Y)$ (the execution of q_i is started within $[t_s, t_s + Y)$) where t_s is the steady time instant, then we have $\sum_{i=1}^{n} n_i E_{q_i} \leqslant Y$ where E_{q_i} is the execution time of q_i.*

The lemma is straightforward proved from the fact that the time instants t_s and $t_s + Y$ have the same scheduling property. That is, the processor either is idle or has the same length busy duration left at both instants.

Now we state the following theorem:

Theorem 8 *Give a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C) = \{\langle U_1, t_{1,1}\rangle, \langle U_2, t_{2,1}\rangle, \ldots, \langle U_n, t_{n,1}\rangle\}$ and candidate query set $Q(C) = \{q_1, q_2, \ldots, q_n\}$, to find a schedule of refresh queries in $Q(C)$ such that the maximum freshness $F(C)$ of C can be achieved is a NP-Hard problem.*

Proof: We only need to prove that the corresponding decision problem is a NP-Complete problem, i.e, we need to prove that the following problem is a NP-Complete problem: given such a cellbase C and a positive value W $(0 < W \leqslant 1)$, is there a schedule of refresh queries such that the achieved freshness $F(C) \geqslant W$?

We will give a polynomial time transformation from the KNAPSACK problem [14] to the above problem.

An instance of the KNAPSACK problem consists of a finite set U, a size $s(u) \in \mathbb{Z}^+$, a value $v(u) \in \mathbb{Z}^+$ for each $u \in U$ and bounds $B, K \in \mathbb{Z}^+$. The problem is to determine if there is a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leqslant B$ and such that $\sum_{u \in U'} v(u) \geqslant K$.

The transformation is performed as follows. Let $U = \{u_1, u_2, \ldots, u_m\}$, $B \in \mathbb{Z}^+$, $K \in \mathbb{Z}^+$, and $s(u_i), v(u_i) \in \mathbb{Z}^+$ for $1 \leqslant i \leqslant m$ constitute an arbitrary instance of the KNAPSACK problem. We create a special instance of our problem by constructing a cellbase $C = \{c_1, c_2, \ldots, c_m\}$ with candidate query set $Q(C) = \{q_1, q_2, \ldots, q_m\}$ such that cell refresh periods $U_1 = U_2 = \ldots = U_m = B$ and the steady period is $[t_s, t_s + B)$. Let the execution time E_{q_i} of q_i be $s(u_i)$ and $R = \{r_1, r_2, \ldots, r_n\}$ be the set of refresh requests raised within $[t_s, t_s + B)$, the scheduling algorithm chooses to schedule a subset $Q' \subseteq Q(C)$ to satisfy a subset of R. Thus, the cellbase freshness $F(C) = \sum_{q_i \in Q'} F_i/n$, where F_i is the freshness for c_i achieved in the steady period when r_i is satisfied. Let $F_i = cv(u_i)/K$ and $W = c/n$ (c is an arbitrary constant number such that $K/B \max\{s(u_i)/v(u_i)\} \leqslant c \leqslant \min\{n, K \min\{1/v(u_i)\}\}$; otherwise, we cannot guarantee $s(u_i)/B \leqslant F_i \leqslant 1$ (from Lemma 1) and $W \leqslant 1$). Clearly, the construction can be done in polynomial time. From Lemma 4, we have $\sum_{q_i \in Q'} E_{q_i} \leqslant B$, i.e., $\sum_{u_i \in U'} s(u_i) \leqslant B$. If $F(C) \geqslant W$, then $\sum_{q_i \in Q'} F_i/n \geqslant c/n$, i.e.,

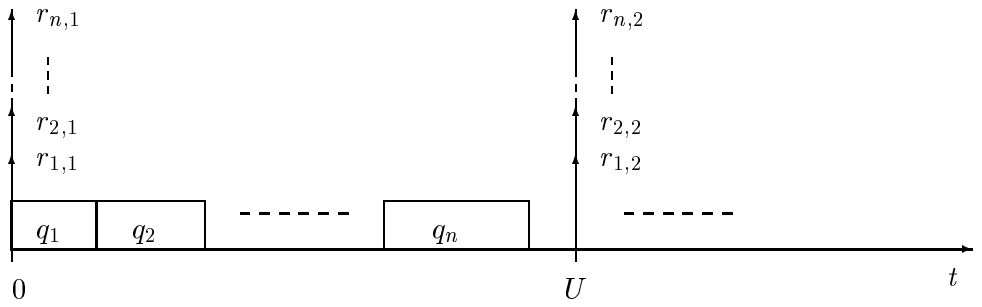

Figure 7: Cells with the same refresh period are refreshed by MiEF

$\sum_{q_i \in Q'} cv(u_i)/K \geqslant c$. Thus, a schedule that achieves the freshness $F(C) \geqslant W$ can be found if and only if we can find a U' such that $\sum_{u_i \in U'} v(u_i) \geqslant K$ with the constraint $\sum_{u_i \in U'} s(u_i) \leqslant B$ while let $Q(C) = U$ and $Q' = U'$.

Therefore, a solution to our instance can be used to solve an arbitrary instance of KNAPSACK problem. Since Knapsack is known to be NP-Complete then our problem to find a schedule that achieves a freshness greater than a given positive number is NP-Complete. Thus, the corresponding optimization problem of finding the maximum freshness is NP-Hard [14]. ∎

Since it is hard to find a schedule which can maximize the freshness of a cellbase, we propose several approximation scheduling algorithms in this section and compare their performance with simulation results.

6.2 Comparison Analysis

In this section, we analyze and compare the performance of the algorithms proposed in the preceding section.

Give a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C) = \{\langle U_1, t_{1,1}\rangle, \langle U_2, t_{2,1}\rangle, \ldots, \langle U_n, t_{n,1}\rangle\}$ and candidate query set $Q(C) = \{q_1, q_2, \ldots, q_n\}$, we examine the freshness achieved by EDF, MiEF, MaEF, SRPF and LRPF respectively in the following two cases: (For simplicity of the analysis, we assume that the cellbase is kept fresh by all schedules, and the steady time instant is 0 and all cells raise their refresh requests at 0 during all schedules.)

6.2.1 Uniform Refresh Periods

In this case, all cells in C have the same refresh period $U_1 = U_2 = \cdots = U_n = U$ and the steady length is U too. Thus, SRPF and EDF (all requests have the same deadline) have no effects on the cellbase freshness, we do not need to consider them in this case, and we only compare MiEF and MaEF.

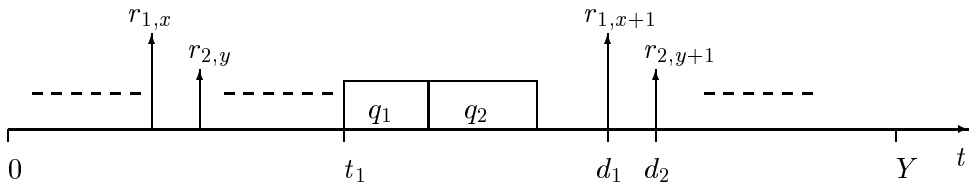

Figure 8: Two adjacent refresh queries during a schedule

Suppose $E_{q_1} \leqslant E_{q_2} \leqslant \cdots \leqslant E_{q_n}$ and $\sum_{i=1}^{n} E_{q_i} \leqslant U$, MiEF schedules the refresh queries as shown in Figure 7. Thus, according to Definition 12, the freshness achieved by MiEF

$$F(C) = \frac{1}{n}\left(\frac{U}{U} + \frac{U - E_{q_1}}{U} + \frac{U - (E_{q_1} + E_{q_2})}{U} + \cdots + \frac{U - (E_{q_1} + E_{q_2} + \cdots + E_{q_{n-1}})}{U}\right)$$

$$= \frac{1}{nU}(nU - (n-1)E_{q_1} - (n-2)E_{q_2} - \cdots - E_{q_{n-1}});$$

If we change the scheduling algorithm to MaEF, then the sequence of executed queries is inversed in Figure 7 and the freshness achieved by MaEF

$$F'(C) = \frac{1}{nU}(nU - (n-1)E_{q_n} - (n-2)E_{q_{n-1}} - \cdots - E_{q_2});$$

Clearly, $F'(C) < F(C)$, MiEF achieves a higher cellbase freshness than MaEF. Note that this result is based on the assumption that schedules made by MiEF and MaEF are all feasible. We can draw the same conclusion even when the schedules made by MiEF and MaEF are infeasible. For infeasible cases, some refresh requests would be missed. However, MiEF can satisfy more number of refresh requests than MaEF when the same number of requests are competing to occupy the same length of time interval because MiEF performs the refresh queries with shorter execution time first. Thus, less number of 0 values are contributed to the cellbase freshness achieved by MiEF than MaEF, which leads to a higher freshness achieved by MiEF.

6.2.2 Random Refresh Period

In this case, the cells in C may have different refresh periods. It is difficult to compute the cellbase freshness by averaging freshness achieved by each cell. However, to compare the effect of different schedule policies, we may just observe the effect by varying the policy within a short interval of the schedule.

Consider two adjacent refresh queries q_1 and q_2 in a schedule for C shown in Figure 8. Suppose the steady period is $[0, Y]$, q_1 and q_2 are executed at t_1 and $t_1 + E_{q_1}$ respectively to satisfy the refresh requests $r_{1,x}$ and $r_{2,y}$. From Definition 9, the freshness achieved in the x_{th} refresh interval

of c_1 and the freshness achieved in the y_{th} refresh interval of c_2 are computed as:

$$\begin{aligned} F_{1,x} &= (d_1 - t_1)/U_1 \\ F_{2,y} &= (d_2 - t_1 - E_{q_1})/U_2 \end{aligned} \quad \text{(Eq. 2)}$$

Thus, according to Definition 12, the respective steady freshness for c_1 and c_2 are:

$$\begin{aligned} F_1 &= \frac{1}{n_1}(b_1 + F_{1,x}) \\ F_2 &= \frac{1}{n_2}(b_2 + F_{2,y}) \end{aligned} \quad \text{(Eq. 3)}$$

where n_1, n_2 are the respective number of raised refresh requests by c_1 and c_2 during the steady period, $n_1 = \lceil Y/U_1 \rceil$, $n_2 = \lceil Y/U_2 \rceil$, and b_1 and b_2 are the respective freshness during other refresh intervals of c_1 and c_2. Therefore, the cellbase freshness

$$F(C) = \frac{1}{n}(F_1 + F_2 + w) \quad \text{(Eq. 4)}$$

where w is the freshness contributed by the other cells except for c_1 and c_2.

With Eq. 2, Eq. 3 and Eq. 4 together, we obtain

$$F(C) = \frac{(b_1 n_2 U_1 U_2 + d_1 n_2 U_2 - n_2 U_2 t_1 + b_2 n_1 U_1 U_2 + n_1 U_1 d_2 - n_1 U_1 t_1 + n_1 n_2 U_1 U_2 w) - n_1 U_1 E_{q_1}}{n n_1 n_2 U_1 U_2}$$

Let $a = b_1 n_2 U_1 U_2 + d_1 n_2 U_2 - n_2 U_2 t_1 + b_2 n_1 U_1 U_2 + n_1 U_1 d_2 - n_1 U_1 t_1 + n_1 n_2 U_1 U_2 w$, $b = n n_1 n_2 U_1 U_2$, then

$$F(C) = \frac{a - n_1 U_1 E_{q_1}}{b} \quad \text{(Eq. 5)}$$

Now we change our scheduling policy to see if the cellbase freshness is affected. We just interchange q_1 and q_2 executed during the interval $[t_1, t_1 + E_{q_1} + E_{q_2}]$ such that q_2 is executed before q_1 while we keep the other sequence unchanged during the schedule shown in Figure 8. With the same method, we compute the changed freshness

$$F'(C) = \frac{a - n_2 U_2 E_{q_2}}{b} \quad \text{(Eq. 6)}$$

Comparing Eq. 5 with Eq. 6 and having $n_1 U_1 = n_2 U_2 = Y$, we observe that the freshness is improved ($F'(C) > F(C)$) only if $E_{q_2} < E_{q_1}$, i.e., MiEF performs better than MaEF. Also, we see that those algorithms (SRPF and LRPF) that assign schedule priority based on the length of cell refresh period would not affect the resultant cellbase freshness.

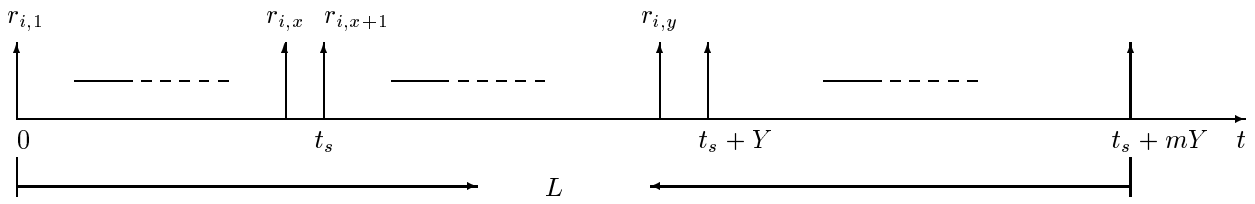

Figure 9: Steady freshness and freshness over a long period

6.3 Performance Metrics

The first and foremost objective of scheduling algorithms is to improve the freshness of the cellbase. Given a cellbase C, to compute its freshness achieved by a schedule, according to Definition 12, we must find out the steady time instant t_s and the length of steady period Y. However, this is a difficult task due to the complex refresh pattern set of C. Fortunately, according to Theorem 2, we observe that after entering the steady time instant t_s, the cellbase achieves the same freshness over every steady period Y. Based on this observation, we build the following theorem to compute the (steady) freshness of a cellbase.

Theorem 9 *The steady freshness $F(C)$ of a cellbase C is equal to the freshness of C over a sufficiently long time period.*

Proof: Suppose that the steady time instant is t_s and the length of steady period is Y, according to Definition 12, the steady freshness $F(C) = F(C)(t_s, t_s + Y)$. Let $C = \{c_1, c_2, \ldots, c_n\}$ and L be a sufficiently large positive integer, we need to prove $F(C)(t_s, t_s + Y) = F(C)(0, L)$. Consider an arbitrary cell $c_i \in C$, if we can prove $F_i(t_s, t_s + Y) = F_i(0, L)$, then the theorem is proved based on $F(C)(t_1, t_2) = \frac{1}{n}\sum_{i=1}^{n} F_i(t_1, t_2)$.

Suppose $L = t_s + mY$ (m is sufficiently large) and c_i raises requests $r_{i,1}, r_{i,2}, \ldots, r_{i,x}$ within $[0, t_s)$ and raises requests $r_{i,x+1}, r_{i,x+2}, \ldots, r_{i,y}$ within $[t_s, t_s + Y)$ (Figure 9), then

$$F_i(0, L) = \lim_{m\to\infty} \frac{\sum_{j=1}^{x} F_{i,j} + m\sum_{k=x+1}^{y} F_{i,k}}{x + m(y - x)} = \frac{\sum_{k=x+1}^{y} F_{i,k}}{y - x} = F_i(t_s, t_s + Y)$$

Thus, the theorem is proved. ∎

Therefore, to measure the freshness of a cellbase achieved by a schedule, we only need to run the schedule for a sufficiently long time and compute the freshness over this long period.

Intuitively, we find that the more the missed refresh requests are produced by a schedule, the smaller the freshness of the cellbase is. This is because that a missed refresh request contributes 0 value to the final freshness of the cellbase. Therefore, we define the percentage of missed refresh

requests among all request raised during a schedule as another measure of performance of scheduling algorithm. It is denoted as

$$M(C) = \frac{m}{r} \times 100\%$$

where m is the missed number of refresh requests during the schedule and r is the total number of raised requests during the schedule.

6.4 Simulation Results

In this section, we simulate refreshing a cellbase by a program and verify the previous performance analysis of different scheduling algorithms by empirical results.

6.4.1 System Environment

We perform the simulation program on a PIII 550 Gateway PC where Red Hat 6.0 is running. Sybase Adaptive Server Enterprise 11.9.2 provides an experimental database as the data source of the cellbase.

6.4.2 Input Parameters

We have set a list of important parameters for tuning behaviors of the simulation program. By varying the values of parameters, we achieve different configurations for the source database, the cellbase and the scheduling algorithm. Thus, different experiments can be conducted. The important parameters are clarified as follows:

NUMBER OF TABLES It is for setting how many tables are stored in the source database.

NUMBER OF COLUMNS The parameter specifies how many columns are included in the base tables.

NUMBER OF TUPLES The parameter specifies how many tuples are stored in the base tables.

NUMBER OF CELLS The parameter specifies how many cells should be composed for the cellbase.

BASE UPDATE PERIOD This parameter is used to provide a base update period for the base tables when they are updated in constant or regular pattern.

UPDATE PATTERN This parameter determines the update pattern of the base tables. The value of the parameter can be *uniform*, *regular* or *random*. When the base tables changes in the uniform pattern, they are updated with the same period, while in regular case update periods of all tables are multiples of the value of parameter BASE UPDATE PERIOD, and in random case tables are updated with the random period.

REQUEST PATTERN It determines how the base tables begin to change. The value can be *synchronous* (all tables begin to change at the same time) or *asynchronous* (all tables begin to change at arbitrary times).

SCHEDULE ALGORITHM This parameter determines which scheduling algorithm is employed to schedule the refresh work. It can be EDF, SRPF, LRPF, MiEF, MaEF.

REFRESH TIMES This parameter specifies how much times the cells should be refreshed, i.e., how many refresh requests a cell raises during the schedule. With this parameter, we control the running time of the simulation.

CELL COMPOSE PATTERN It determines where the content of composed cell comes from. The value can be *atomic* (from one base table) or *complex* (from multiple base tables).

6.4.3 Experiments

We test the performances of scheduling algorithms: EDF, SRPF, LRPF, MiEF, and MaEF with the different parameter settings. The source database has 20 tables and each table stores 1000 tuples. Given a cellbase $C = \{c_1, c_2, \ldots, c_n\}$ with refresh pattern set $R(C) = \{\langle U_1, t_{1,1}\rangle, \langle U_2, t_{2,1}\rangle, \ldots, \langle U_n, t_{n,1}\rangle\}$ and trivial candidate query set $Q_0(C) = \{q_1, q_2, \ldots, q_n\}$, the workload of the refresh system is measured by the *utilization factor*, which is defined as $\sum_{i=1}^{n} E_{q_i}/U_i$ and represents the fraction of computing time consumed by the queries over the lifetime of the system.

We have two different experimental settings as follows:

(i) (a) UPDATE PATTERN: uniform
    (b) BASE REFRESH PERIOD: 5sec

(ii) (a) UPDATE PATTERN: random
    (b) BASE REFRESH PERIOD: 5sec
    (c) RANDOM SCOPE: 5
    (d) RANDOM SEED: 1

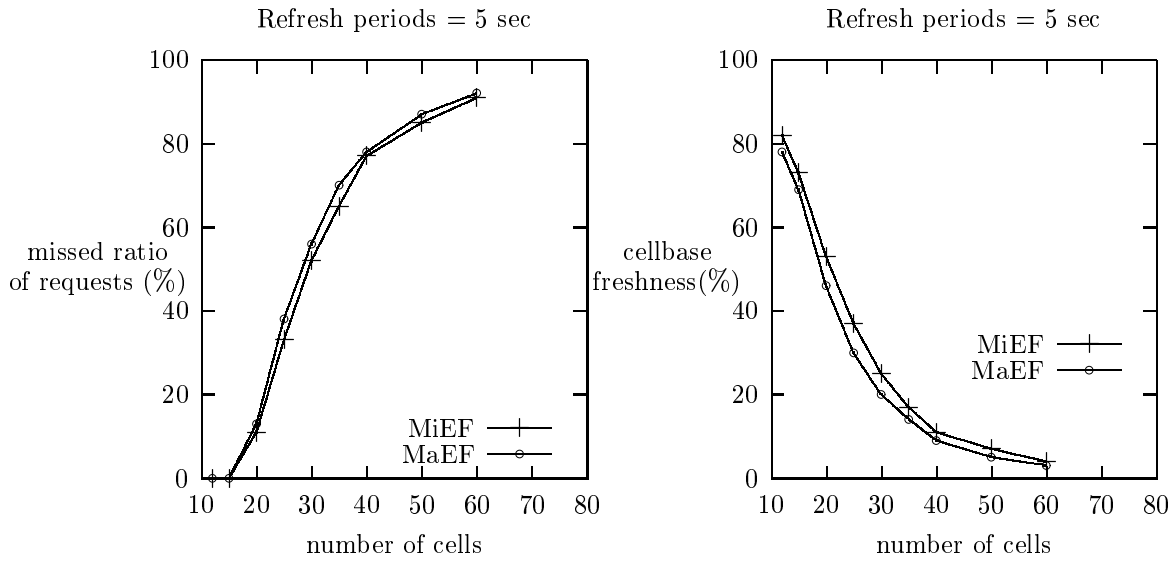

Figure 10: Cellbase freshness achieved by MiEF and MaEF when cells have the same refresh period

For each configuration (experiment), we adjust the workload by varying the number of cells in the cellbase (i.e., the number of refresh queries) and perform the simulation program scheduled by different algorithms respectively.

Firstly, we compare the performance of MiEF and MaEF when all base tables change at a fixed frequency (once every 5 seconds). The results are shown in Figure 10. In this experiment, the workload (utilization factor) is increased from 0.2 to 1.5. From the left figure, we can see that the missed ratio of refresh requests rapidly increases for both MiEF and MaEF algorithms with the increase of the number of refresh queries (workload). This situation is caused by multiple queries which compete to utilize processor and database access. Additional, we observe that MaEF has a little higher missing ratio than MiEF. Correspondingly, the right figure shows that the cellbase freshness decreases rapidly with the growth of the workload, and MiEF achieves higher freshness than MaEF (Because the refresh queries in the experiment designed by us have the similar execution times, we only see a small benefit achieved by MiEF). Note that when the system is overloaded (the utilization factor of the system exceeds 1 when the number of cells exceeds 40), both algorithms achieve the same low cellbase freshness due to the high ratio of missed refresh requests.

Next, we do the experiment to measure the performance of the scheduling algorithms when base tables change at random frequencies, i.e., cells have random refresh periods. The results are shown in Figure 11. In this experiment, the refresh periods for the cells are uniformly distributed within $[5, 10]$ seconds and the workload is gradually increased from 0.2 to 1.5 with the growth of the number of the cells in the cellbase. From the left figure, we obtain the same result as the first

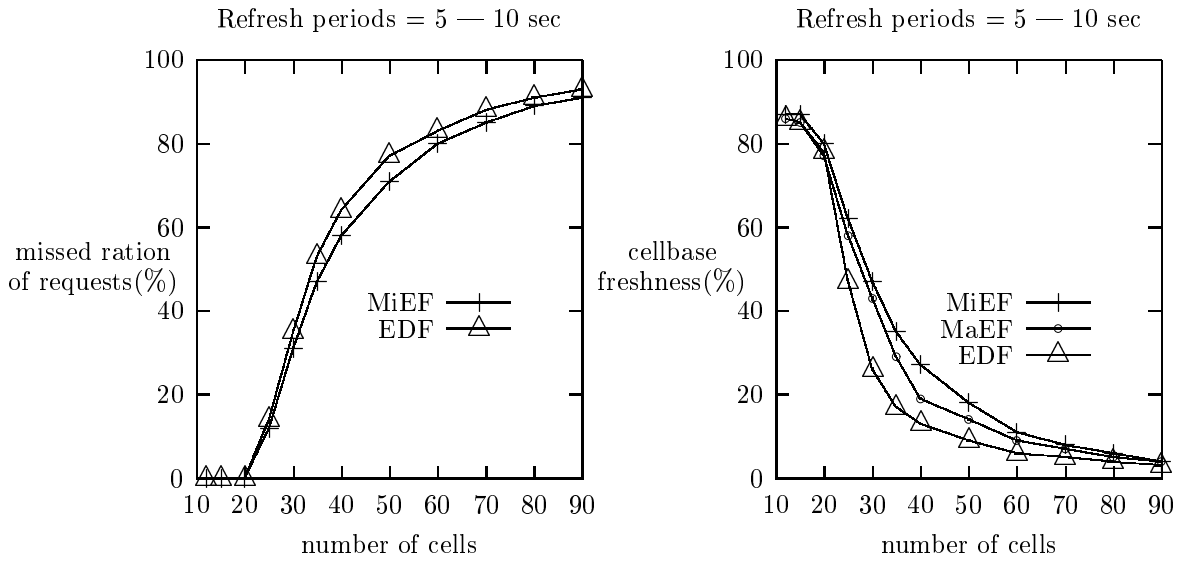

Figure 11: Cellbase freshness achieved by different algorithms when cells have random refresh periods

experiment that the missing ratio of refresh requests increases with the increase of the workload. More than 80% of refresh requests have been missed if the system is overloaded with more than 60 cells (the utilization factor exceeds 1). Also, we see that $MiEF$ has a lower ratio of missed requests than EDF. In order to avoid cluttering the graph, we do not show the curve achieved by $MaEF$ since it almost achieves the same missing ratio of requests, just a little higher. The right figure shows the cellbase freshness achieved by $MiEF$, $MaEF$ and EDF. Clearly, $MiEF$ performs the best and the cellbase freshness rapidly drops with the growth of the workload. When the system is overloaded, all algorithms perform similarly and the cellbase freshness is lower than 10%.

In order to verify that refresh period plays a trivial role on the achieved cellbase freshness, we do another experiment to compare the SRPF and LRPF algorithms. The experiment settings are the same as those in the second experiment. The result is shown in Figure 12. We can clearly see from the figure that two curves almost coincide with each other. That is, SRPF and LRPF do not affect the cellbase freshness.

Therefore, the above experiments have verified the analysis made in Section 6.2. The results prove that MiEF has the advantage over other algorithms on the cellbase freshness regardless of whether the cells have the uniform or random refresh periods. Another conclusion is that the metric of missing ratio of requests keeps consistent with the cellbase freshness metric on comparing the different algorithms, i.e., the higher is the missing ratio of requests the lower is the cellbase freshness.

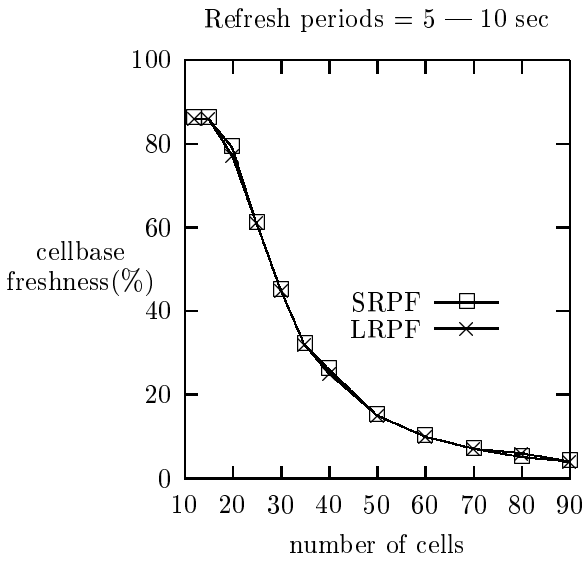

Figure 12: Cellbase freshness achieved by different algorithms when cells have random refresh periods

7 Related Work

To keep websites up-to-date has drawn some interests of researchers recently, although in its infancy. In [1], different Webview (referred as Web cell by us) materialization policies (materialized inside the DBMS, materialized at the Web server and virtual) have been compared. Their results have indicated that materializing at the Web server is a more scalable solution and can facilitate an order of magnitude more users than the virtual and materialized inside the DBMS policies. This result also verifies our approach where a cellbase materializing Web cells is built at the side of Web server. Another similar work has been done in [12] where a document generator system is presented. Their approach providing up-to-date Web information is based on the extensibility infrastructure of object-relational database systems. By utilizing DB triggers and so-called user-defined functions, documents can automatically be generated by the DBMS whenever data is updated. However, this approach will lead to a high workload on databases.

Our approach is to schedule refresh queries to pull fresh data to websites. This work is closely related to the real-time scheduling. A *real-time system* is one in which the correctness of the system depends not only on the logical results, but also on the time at which the results are produced [16]. Since refreshing cells is performed with strict timing constraint that execution of a refresh query must be completed within the interval between two consecutive changes of data source, we may view the refreshing cells as a typical *recurring task* [15] in real-time system that makes repeated requests for processor time. More traditional work can be found in [2, 8, 13, 10]. By utilizing

the real-time scheduling, we can carefully define and analyze the freshness of a website. A similar freshness metric has been defined in [9], however, their work focus on refreshing a local copy of an autonomous data source (website), instead of the original website.

8 Conclusions

As the WWW becomes more and more popular, the trend of using database as the source of Web pages grows rapidly too. Our efforts have been made to maintain the freshness of a data-intensive website in response to changes to the base data. In this paper, we identified website refresh as one significant task of website management. A Web cell has been defined as a portion of Web page that is derived from the result of a query against base data. For refreshing a website, a cellbase has been designed to materialize the Web cells hosted on the website. Thus, our task is to keep the cellbase up-to-date. We first roughly distinguish two levels of freshness of a cellbase: tight freshness and loose freshness according to the timeliness of scheduled refresh queries in response to periodical updates on base data. Then we formally defined the metric cellbase freshness that can be quantitatively measured. Several algorithms have been proposed to schedule the executions of refresh queries. From our analysis and empirical results, we conclude that MiEF algorithm performs the best among the scheduling algorithms on the achieved cellbase freshness. More work is being done to develop optimization technique for refresh query set such that the database access is reduced and the cellbase freshness is improved.

References

[1] Alexandros Labrinidis, Nick Roussopoulos. Webview materialization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Texas, USA, 2000.

[2] C. L. Liu, J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20:46–61, 1973.

[3] D. Florescu, A. Levy, A. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(3), 1998.

[4] Haifeng Liu, Wee-Keong Ng, Ee-Peng Lim. Keeping a very large website up-to-date: Some feasibility results. In *Proceedins of the 1th International Conference on Electronic Commerce and Web Technologies (EC-Web2000)*, Greenwich, UK, September 2000.

[5] Haifeng Liu, Wee-Keong Ng, Ee-Peng Lim. Model and research issues for refreshing a very large website. In *Proceedins of the 1th International Conference on Web-based Information Systems Engineering (WISE2000)*, Hong Kong, June 2000.

[6] Haifeng Liu, Wee-Keong Ng, Ee-Peng Lim. Query integration for refreshing web views. In *Submitted to the 10th International Conference on Data Engineering*, Headiberg, Germany, April 2001.

[7] J. Leung, J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[8] J. Leung, M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

[9] Junghoo Cho, Hector Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of 2000 ACM SIGMOD International Conf. on Management of Data (SIGMOD 2000)*, May 2000.

[10] K. Hong, J. Leung. On-line scheduling of real-time tasks. In *Proceedings of the 9th IEEE Real-Time Systems Symposium.*

[11] K. Jeffay, D. Stanat, C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, pages 129–139, 1991.

[12] Henrik Loeser. Keeping web pages up-to-date with sql:1999. In *Proc. of the Int. Database Engineering and Applications Symposium (IDEAS 2000)*, Yokohama, Japan, Septemper 2000.

[13] A. Mok. Fundamental design problems of distributed systems for the hard real-time environment. *PhD thesis, MIT Laboratory for Computer Science*, 1983.

[14] R. Garey, S. Johnson. *Computers and Intractability, a guide to the theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[15] R. Howell, K. Venkatrao. On non-preemptive scheduling of recurring tasks using inserted idle times. *Information and Computation*, 117:50–62, 1995.

[16] J. A. Stankovic. Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, 28(4), December 1996.