AIDOA: An Adaptive and Energy-Conserving Indexing Method for On-Demand Data Broadcasting Systems

Jiun-Long Huang

***Abstract*—Since only a modest improvement in battery lifetime is expected in the next few years, energy conservation is raised as a key factor in the design of mobile devices. In view of this, we propose in this paper an energy-conserving on-demand data broadcasting system that employs the data indexing technique. Different from prior work, the power consumption of turning on and turning off the wireless network interfaces is considered. In addition, we also employ a server cache to reduce the effect of the time to retrieve data items from the corresponding data servers. Specifically, we first analyze the access and tuning times of data requests, and propose an Adaptive Index and Data Organizing Algorithm (AIDOA) to adjust the degree of buckets according to the system workload. Several experiments are then conducted to evaluate the performance of algorithm AIDOA. The experimental results show that algorithm AIDOA is able to greatly reduce the power consumption at the cost of a slight increase in the average access time and dynamically adjust the index and data organization to adapt to the change of system workload.**

***Index Terms*—Data indexing, energy conservation, mobile information system, on-demand data broadcasting.**

I. Introduction

OWING to the constraints resulting from power-limited mobile devices and low-bandwidth wireless networks, designing a power-conserving mobile information system with high scalability and high bandwidth utilization becomes an important research issue and hence attracts a significant amount of research attention. In recent years, data broadcasting has been proposed to address such a challenge and has been recognized as a promising data dissemination technique in mobile computing environments [1], [4], [5], [10], [11]. Most research works on data broadcasting focus on generating a proper broadcast program or designing scheduling algorithms to minimize the *average access time*, which is defined as the average time elapsed from the moment a client issues a query to the point that the desired data item is read.

As shown in [17] and [19], only a modest improvement (about 20% to 30%) in battery lifetime is expected in the next few years. Hence, energy conservation is raised as a key factor in the design of mobile devices. Consider a Nokia 5510 that supports Advanced Audio Coding and Moving Pictures Experts Group Audio Layer 3 playing. Compared to the power consumed on playing music, the wireless network interface (WNI) consumes much more energy (as much as 70% of the total power in Nokia 5510) [22]. Hence, reducing the power consumption on WNIs is an effective means to reduce the overall power consumption. Most devices can operate in two modes, i.e., *active* mode and *doze* mode. Many studies show that the power consumed in active mode is much higher than that consumed in doze mode. For example, a typical wireless personal computer card (e.g., ORiNOCO) consumes 60 mW during the doze mode and 805–1400 mW during the active mode [19]. As a consequence, to reduce the power consumption, the mobile devices should stay in doze mode for as long as possible.

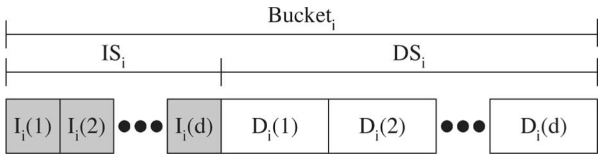


Fig. 1. Index structure.

To evaluate the effect of data indexing algorithms on energy conservation, the *tuning time*, which is defined as the time that a mobile device operates in active mode to retrieve a data item, is introduced in [12]. Since employing data indexing will unavoidably introduce some overhead in access time, the data indexing algorithms should reduce the tuning time as much as possible at the cost of producing an acceptable increase in access time. Since the size of an index item is usually much smaller than that of a data item, the increment in access time is usually small. As a result, many research works study the design of data indexing algorithms in push-based data broadcasting environments [20], [21]. However, most studies on on-demand data broadcasting focus on the design of scheduling algorithms [2], [5] to reduce the average access time, and only a few of them consider the employment of data indexing in on-demand data broadcasting environments [13] to reduce the average tuning time.

In [13], Lee *et al.* proposed an indexing algorithm for on-demand data broadcast systems. As shown in Fig. 1, the proposed broadcast program is made up of a series of buckets, and each bucket consists of one index segment and one data segment. A data segment contains a series of data items, while an index segment consists of the index items of the data items in the corresponding data segment. For a bucket, the number

Manuscript received December 13, 2005; revised June 20, 2006. This work was supported in part by the National Science Council of Taiwan, R.O.C., under Contract NSC95-2221-E-009-220. This paper was recommended by Associate Editor H. R. Rao.

The author is with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: jlhuang@cs.nctu.edu.tw).

Digital Object Identifier 10.1109/TSMCA.2007.914739

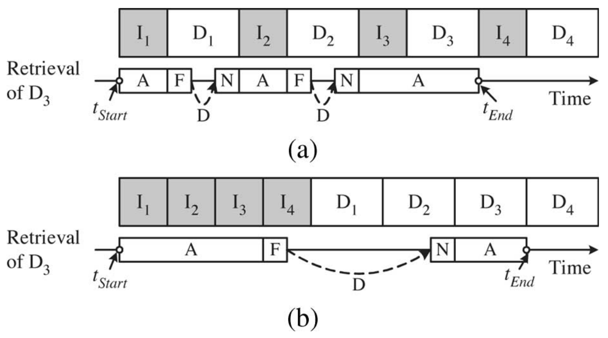


Fig. 2. Example organizations of index and data items. (a) Organization one. (b) Organization two.

of data items in the corresponding data segment is called the *degree* of the bucket. The information in an index item, for example, $I_i(1)$, consists of the identifier of the corresponding data item $D_i(1)$, the data size of $D_i(1)$, and the time that $D_i(1)$ in bucket i will be broadcast on the broadcast channel. In addition, by the information in the current index segment, a mobile device is able to determine the broadcast time of the index segment of the next bucket.

Although inserting index items into the broadcast program is able to significantly reduce the average tuning time at the cost of a slight increase in average access time [13], however, the proposed data indexing method proposed in [13] has the following drawbacks.

- *Does not consider the power consumption of turning on and turning off the WNIs.*

  As pointed out in [18], turning on and turning off the WNIs consume some time and energy, and the transition times of a WNI from active mode to doze mode and from doze mode to active mode are both on the order of tens of milliseconds. Consider two organizations of index and data items shown in Fig. 2.[1] Suppose that a mobile device tunes to the broadcast channel at time t_{Start} and finishes the retrieval of the desired data item at time t_{End}. Without considering the power consumption of turning on and turning off the WNI, the power consumptions of organization one and organization two are equal. However, when the power consumption of turning on and turning off the WNIs is considered, organization two outperforms organization one.

  Therefore, we argue that the design of an energy-conserving data-indexing method should take the power consumption of turning on and turning off the WNIs into account to obtain a precise power consumption estimation. To the best of our knowledge, there is no prior work on data indexing in on-demand broadcast considering the power consumption of turning on and turning off the WNIs, which thereby distinguishes our paper from others.
- *Does not consider the data fetch time.*

  Most studies on indexing in on-demand data broadcasting are under the premise that all the data items are *immediately* available for a data broadcasting system [13]. However, as pointed out in [6], the data fetch time cannot be neglected since it is infeasible to store all the data items in the local cache of the system. Hence, the traditional data broadcasting systems [5] may not perform well. As a consequence, we argue that the indexing algorithm in on-demand data broadcasting should also consider the data fetch time to attain a higher efficiency.
- *Does not adapt to the change of system workload.*

  In mobile computing environments, schemes with a static degree may not be able to adapt to the change of system workload. Such a phenomenon shows the necessity of designing an adaptive algorithm to dynamically adjust the degree of buckets to adapt to the change of system workload. To the best of our knowledge, all prior works on data indexing in on-demand broadcast employ a static degree, and none of them is able to adapt to the change of system workload.

In view of this, we propose in this paper an energy-conserving on-demand data broadcasting system by employing the data indexing technique. Different from the prior work on data indexing on on-demand data broadcasting, the power consumption of turning on and turning off the WNIs is considered. Specifically, we first analyze the access and tuning times of data requests, and propose an Adaptive Index and Data Organizing Algorithm (AIDOA) to adjust the degree of buckets according to the system workload. In essence, algorithm AIDOA consists of two phases, i.e., statistics collection phase and adjustment phase, and periodically switches back and forth between these two phases. The system collects some statistic information of all the served data requests in the statistics collection phase, and the collected information is used to adjust the degree of buckets in the adjustment phase according to the derived analytical results. In addition, we employ a server cache to eliminate the performance degradation caused by the data fetch time. We also propose a program generation algorithm and a cache replacement policy to cooperate with algorithm AIDOA. Several experiments are then conducted to evaluate the performance of algorithm AIDOA. The experimental results show that due to the dynamic adjustment on the degree of buckets, the scheme that uses algorithm AIDOA outperforms the other schemes with a static degree in most cases.

The rest of this paper is organized as follows. Section II describes the proposed system architecture and the power consumption model used in this paper. Section III shows the analytical model of the proposed system architecture. Based on the analytical model, we propose algorithm AIDOA in Section IV. In addition, the companion program generation algorithm and the cache replacement policy are proposed in Section V. The experimental results are shown in Section VI to evaluate the performance of algorithm AIDOA, and finally, Section VII concludes this paper.

II. Preliminaries

A. System Architecture

We adopt the index structure proposed in [13], and the adopted index structure is shown in Fig. 1. As shown in Fig. 3, the proposed system architecture consists of the following components.

Scheduler: The scheduler is in charge of receiving and processing the data requests submitted by the mobile devices.

[1]The descriptions of symbols "A," "D," "F," or "N" will be given in Table I in Section III-B.

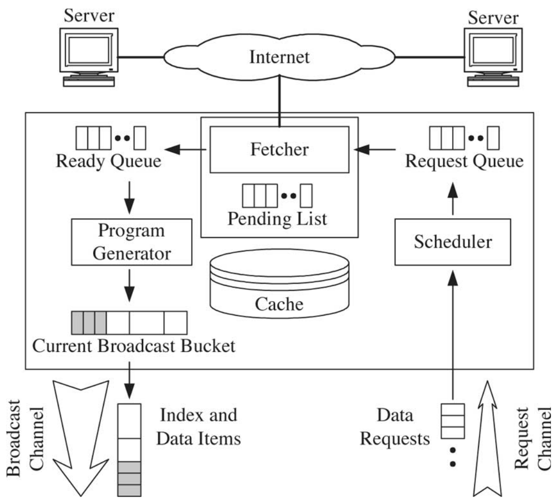


Fig. 3. System architecture.

After receiving a data request, for example, Req_i, the scheduler will sequentially search the ready queue, pending list, and request queue to check whether there exists a data request, for example, Req_j, with the same required data item as Req_i. When Req_j is in the pending list, the scheduler merges Req_i into Req_j. When Req_j is in the ready queue (respectively, the request queue), the scheduler will merge Req_i into Req_j and updates the priorities of all the data items in the ready queue (respectively, the request queue) according to the employed scheduling algorithm, such as First-In–First-Out, Longest Wait First (LWF), RxW [5], and so on. Otherwise, when Req_j does not exist, the scheduler will insert Req_i into the request queue and update the priorities of all the data items in the request queue according to the employed scheduling algorithm.

Fetcher: The fetcher repeatedly retrieves the data request with the highest priority from the request queue and fetches the required data item from the corresponding data server via Internet. Cache is employed to reduce the performance degradation caused by the data item fetch time. To fetch a data item, the fetcher first checks whether the required data item is cached in the local cache. If yes, the fetcher will mark the cached data item as LOCKED and insert the data request into the ready queue. Then, the fetcher will retrieve the data request with the highest priority from the request queue and repeat the above procedure.

Otherwise, when the desired data item is not cached, the fetcher will submit a data request message to the data server of the required data item and insert the data request into the pending list. Then, the fetcher will check the number of pending data requests and will stop if the number of pending data requests is equal to a predetermined threshold. Otherwise, the fetcher will repeat the above procedure until the number of pending data requests is equal to a predetermined threshold or the request queue is empty.

When a data server responds with a data item, the fetcher will retrieve the corresponding data request from the pending list and insert the data request into the ready queue. In addition, the fetcher will insert the received data item into the cache. Several cached data items may be replaced by the employed replacement policy when the free space of the cache is not enough to store the received data item.

Program Generator: The program generator employs a program generation algorithm to compose all the buckets of the broadcast programs. After a bucket is generated, the index and data items in the current bucket are sequentially broadcast. The program generator will start to compose another bucket after all the index and data items in the current bucket have been broadcast.

B. Power Consumption Model

Denote the time for a mobile device to switch the WNI from active mode to doze mode as T_{On} and the time to switch the WNI from doze mode to active mode as T_{Off}. To evaluate the power consumption of turning on and turning off the WNIs, we assume that the power consumption of a mobile device spending in time intervals T_{On} (respectively, T_{Off}) is equal to that of a mobile device staying in the active mode for time $\alpha_1 \times T_{\text{On}}$ (respectively, time $\alpha_2 \times T_{\text{Off}}$). Similar to [22], the values of α_1 and α_2 can be obtained by profiling.

Denote the traditional (i.e., without considering the turning-on and turning-off time of WNIs) average tuning time of a data request as T_{Tuning}. To evaluate the overall power consumption, we define the *effective tuning time* of a data request as $T_{\text{Tuning}}^{\text{Eff.}} = T_{\text{Tuning}} + n_1 \times \alpha_1 \times T_{\text{On}} + n_2 \times \alpha_2 \times T_{\text{Off}}$, where n_1 and n_2 are the numbers of times of turning on and turning off the WNI, respectively, and T_{Tuning} is the traditional tuning time. To ease the presentation, we use the term tuning time to represent the effective tuning time and assume that $\alpha_1 = \alpha_2 = 1$ in the rest of this paper.

III. Analytical Model

A. Client Access Protocol

After submitting a data request, a mobile client will retrieve the desired data item according to the employed client access protocol. We adopt the client access protocol described in [20], and the protocol consists of the following phases.

- *Initial probe phase:* After submitting a data request, the mobile device tunes to the broadcast channel and listens on the broadcast channel to wait for the appearance of an index segment.
- *Index search phase:* The mobile device enters the index search phase after retrieving an index segment. In the index search phase, the mobile device determines whether the desired data item will be broadcast in the corresponding data segment. If not, the mobile device will switch to doze mode and then switch back to active mode when the next index segment is broadcast. Otherwise, the mobile device will enter the data retrieval phase.
- *Data retrieval phase:* If the desired data item will be broadcast in the current data segment, the mobile device will retrieve the time that the desired data item will be broadcast from the current index segment and switch to doze mode. Then, when the desired data item is broadcast, the mobile device will switch back to active mode and retrieve the desired data item.

Consider the example shown in Fig. 4, where a mobile device submits a data request. Let t_{Start} be the time that the mobile device starts to listen on the broadcast channel after submitting the data request, and let t_{End} be the time that the mobile device

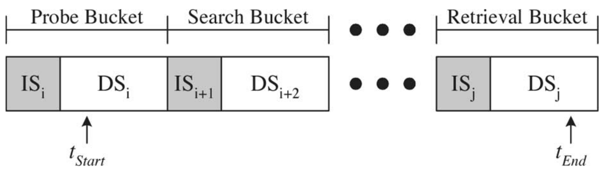


Fig. 4. Categories of buckets.

TABLE I
SYMBOLS OF TIME FRAMES

Symbol	Description
A	The mobile device is in active mode
D	The mobile device is in doze mode
F	The mobile device is turning off its WNI
N	The mobile device is turning on its WNI

receives the desired data item. According to the employed client access protocol, the buckets within the time interval from t_{Start} to t_{End} can be divided into the following three categories.

1) *Probe bucket:* The bucket that t_{Start} lies on is called the probe bucket. In Fig. 4, $Bucket(i)$ is the probe bucket. There is only one probe bucket for each data request.
2) *Search bucket:* The bucket whose index segment is retrieved by the mobile device and whose data segment is skipped by the mobile device is called the search bucket. In Fig. 4, $Bucket(i+1), Bucket(i+2), \ldots, Bucket(j-1)$ are all search buckets. For a data request, there may be zero, one, or multiple search buckets.
3) *Retrieval bucket:* The bucket that t_{End} lies on is called the probe bucket. That is, the retrieval bucket is the bucket where the mobile device retrieves the desired data item. In Fig. 4, $Bucket(j)$ is the retrieval bucket. For each data request, there is only one probe bucket. In addition, the probe bucket and the retrieval bucket of a data request may be the same or different.

B. Derivations of Access Time and Tuning Time

To facilitate the following derivations, we have the following assumptions. 1) All the data items are of equal size S_D. 2) The time to broadcast a data item (i.e., S_D/B) is larger than $T_{\text{On}} + T_{\text{Off}}$. Note that both assumptions are not the limitations of ADIOA and are made only to ease the derivations in Sections III and IV. Hence, they will be relaxed in Sections V and VI.

In $Bucket(i)$, denote the moment that the mobile device starts to turn on and turn off the WNI as $t_{\text{WakeUp}}(i)$ and $t_{\text{Sleep}}(i)$, respectively. In addition, we also denote the starting and ending times of $Bucket(i)$ as $Bucket(i).Start$ and $Bucket(i).End$, respectively. For a data request, we also partition the time interval from t_{Start} to t_{End} into several segments, and each segment is marked as "A," "D," "F," or "N." The descriptions of these four symbols are given in Table I.

According to the relationship of the probe and retrieval buckets, a data request may be belonging to one of the following two types.

1) Type I: The Probe and Retrieval Buckets Are the Same: As shown in Fig. 5, in a Type I data request, t_{Start} and t_{End} are within the same bucket. In addition, according to the employed client access protocol, t_{Start} must be located in the index segment. Otherwise, t_{Start} and t_{End} will not be in the same bucket, and such a result conflicts with the definition of Type I data requests. To minimize the power consumption, $t_{\text{Sleep}}(i)$ is determined as the moment that the mobile device has finished the retrieval of the corresponding index item of the desired data item, and $t_{\text{WakeUp}}(i)$ is determined as the moment that the mobile device has to start to turn on the WNI to retrieve the desired data item.

We observe from Fig. 5 that one Type I data request will increase the aggregate access Qtime of all the data requests by $t_{\text{End}} - t_{\text{Start}}$. On the other hand, the contribution of a Type I data request on the aggregate tuning time of all the data requests is determined by the length of the time interval $(t_{\text{Sleep}}(i), t_{\text{WakeUp}}(i))$. When $t_{\text{WakeUp}}(i) - t_{\text{Sleep}}(i) > T_{\text{Off}}$, the data request will increase the aggregate tuning time by

$$t_{\text{Sleep}}(i) - t_{\text{Start}} + t_{\text{End}} - t_{\text{WakeUp}}(i) + T_{\text{On}} = t_{\text{Sleep}}(i) - t_{\text{Start}} + \frac{S_D}{B} + T_{\text{On}} + T_{\text{Off}}.$$

Otherwise, when $t_{\text{WakeUp}}(i) - t_{\text{Sleep}}(i) \le T_{\text{Off}}$ (i.e., the mobile must start to turn on the WNI before the WNI has been turned off), the time interval $(t_{\text{Sleep}}(i), t_{\text{WakeUp}}(i))$ is too short to turn on and then turn off the WNI. Hence, the data request will increase the aggregate tuning time by $t_{\text{End}} - t_{\text{Start}}$.

2) Type II: The Probe and Retrieval Buckets Are Different: The time interval $(t_{\text{Start}}, t_{\text{End}})$ of a Type II data request consists of one probe bucket; zero, one, or multiple search buckets; and one retrieval bucket. Next, we will separately derive the contributions of the probe bucket, search buckets, and retrieval bucket of a Type II data request on the aggregate access and tuning times of all the data requests.

a) Probe bucket: Consider the example shown in Fig. 6. According to the location of t_{Start}, the Type II data requests can be divided into the following two subtypes.

1) *Type II.I: t_{Start} is in the index segment.* Consider a Type II.I data request. Since the desired data item is not in the probe bucket [i.e., $Bucket(i)$], the probe bucket of a Type II.I data request will increase the aggregate access time of all the data requests by $Bucket(i+1).Start - t_{\text{Start}}$.

   On the other hand, to maximize the power saving, the mobile device should start to turn off the WNI after retrieving the latest index item in IS_i and must turn on the WNI on $Bucket(i+1).Start$ to retrieve the first index item in IS_{i+1}. Hence, $t_{\text{WakeUp}}(i)$ is equal to $Bucket(i+1).Start - T_{\text{On}}$. As a consequence, a Type II.I data request will increase the aggregate tuning time of all the data requests by $t_{\text{Sleep}}(i) - t_{\text{Start}} + T_{\text{Off}} + T_{\text{On}}$.
2) *Type II.II: t_{Start} is in the data segment.* When t_{Start} is in the data segment, according to the employed client access protocol, the mobile device has to listen on the broadcast channel to wait for the appearance of the index segment of the next bucket (i.e., IS_{i+1}). Hence, in $Bucket(i)$, the mobile device is in the active mode from t_{Start} to $Bucket(i+1).Start$, and the contributions of the probe bucket of a Type II.II data request on the aggregate access and tuning times are both $Bucket(i+1).Start - t_{\text{Start}}$.

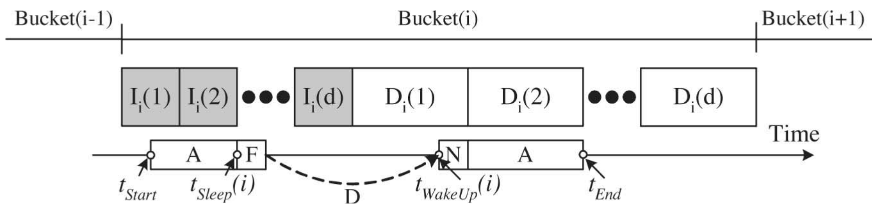

Bucket(i-1)
Bucket(i)
Bucket(i+1)
I_i(1) I_i(2) I_i(d) D_i(1) D_i(2) D_i(d)
Time
A F N A D
t_Start t_Sleep(i) t_WakeUp(i) t_End

Fig. 5. Probe bucket in a Type I data request.

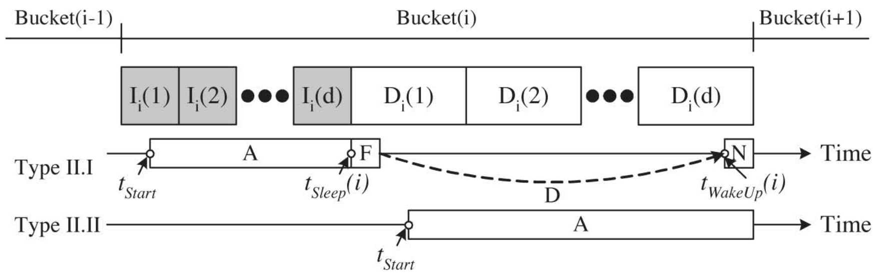


Fig. 6. Probe buckets in Type II.I and Type II.II data requests.

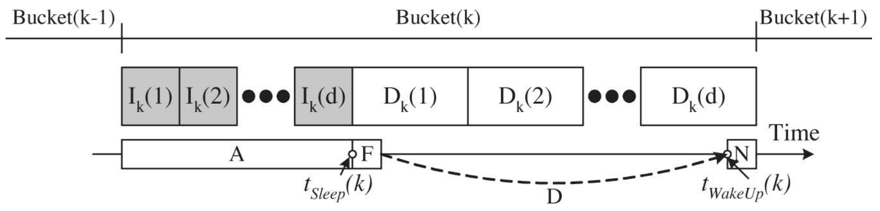


Fig. 7. Search bucket in a Type II data request.

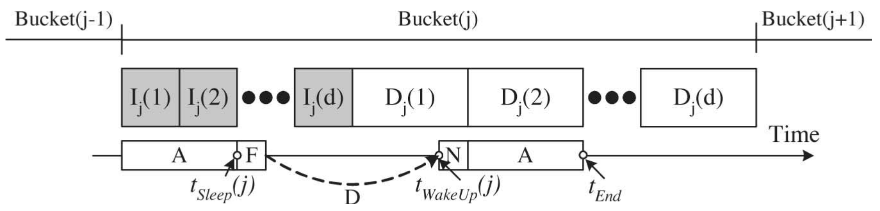

Bucket(j-1)
Bucket(j)
Bucket(j+1)
I_j(1) I_j(2) I_j(d) D_j(1) D_j(2) D_j(d)
Time
A F N A D
t_Sleep(j) t_WakeUp(j) t_End

Fig. 8. Retrieval bucket in a Type II data request.

b) Search bucket: Consider the example shown in Fig. 7. In a search bucket, the mobile device operates in the active mode to retrieve the index segment and starts to turn off the WNI after retrieving all the index items in the index segment. Then, the mobile device has to start to turn on the WNI to ensure that the mobile device just enters the active mode on $Bucket(k+1).Start$. Hence, in a search bucket $Bucket(k)$, the respective contributions on the aggregate access and tuning times of all the data requests are

$$Bucket(k+1).Start - Bucket(k).Start = d \times \frac{S_\text{I} + S_D}{B}$$

$$t_\text{Sleep} - Bucket(k).Start + T_\text{On} + T_\text{Off} = d \times \frac{S_\text{I}}{B} + T_\text{On} + T_\text{Off}.$$

c) Retrieval bucket: Consider the example shown in Fig. 8. In the retrieval bucket, the mobile device sequentially retrieves the index items in the index segment until the index item of the desired data item has been retrieved. Then, the mobile starts to turn off the WNI to wait for the appearance of the desired data item. To retrieve the desired data item, the mobile device has to start to turn on the WNI so that the mobile device is able to enter the active mode in the moment that the desired data item is just being broadcast. Hence, the retrieval bucket of a Type II data request will increase the aggregate access time by $t_\text{End} - Bucket(j).Start$. In addition, the retrieval bucket of a Type II data request will increase the aggregate tuning time by

$$t_\text{Sleep}(j) - Bucket(j).Start + t_\text{End} - t_\text{WakeUp}(j) + T_\text{On} + T_\text{Off}$$
$$= t_\text{Sleep}(i) - Bucket(k).Start + \frac{S_D}{B} + T_\text{On} + T_\text{Off}$$

when $t_\text{WakeUp}(j) - t_\text{Sleep}(j) > T_\text{On}$. Otherwise, the data request increases the total tuning time by $t_\text{End} - Bucket(j).Start$.

With the above discussions, for a Type II data request, its contributions on the aggregate access and tuning times are equal to the summations of the access and tuning times, respectively, of its probe bucket, search buckets, and retrieval bucket.

IV. AIDOA

With the analysis in Section III, in this section, we propose algorithm AIDOA to dynamically adjust the degree of buckets according to the system workload. Basically, algorithm AIDOA consists of two phases, i.e., statistics collection phase and degree adjustment phase, and periodically switches between the statistics collection phase and the degree adjustment phase. In the statistics collection phase, the server will keep track of the information of all the data requests, and the recorded information will be used to guide the adaptation procedure in the successive execution of the adjustment phase.

A. Statistics Collection Phase

In each execution of the statistics collection phase, the server will collect the statistic information of all the data requests served in the current execution of the statistics collection phase. A data request is *served* when the desired data item has been broadcasted.

Two data structures ($Stat_{\rm I}$ and $Stat_{\rm II}$) are defined to store the collected information of Type I and Type II (including Type II.I and Type II.II) data requests, respectively. The details of $Stat_{\rm I}$ and $Stat_{\rm II}$ are as follows.

Details of $Stat_{\rm I}$

- *ReqNo:* The number of Type I data requests served in the current statistics collection phase.
- *AggAT:* Aggregate Access Time of Type I data requests served in the current statistics collection phase.
- *AggTT:* Aggregate Tuning Time of Type I data requests served in the current statistics collection phase.

Details of $Stat_{\rm II}$

- *ReqNo:* The number of Type II data requests served in the current statistics collection phase.
- *AggATP/AggTTP:* Aggregate Access/Tuning Time of Probe buckets of Type II data requests served in the current statistics collection phase.
- *AggATS/AggTTS:* Aggregate Access/Tuning Time of Search buckets of Type II data requests served in the current statistics collection phase.
- *AggATR/AggTTR:* Aggregate Access/Tuning Time of Retrieval buckets of Type II data requests served in the current statistics collection phase.

Each field, except $ReqNo$, of $Stat_{\rm I}$ and $Stat_{\rm II}$ has an *average* version with new names by replacing the prefix Agg with Avg. For example, the field $AvgAT$ of $Stat_{\rm I}$ indicates the *average* access time of all Type I data requests served in the current statistics collection phase. We also define the structure *Request* to indicate the data requests that are merged together. The elements in the request queue, pending list, and ready queue are all instance of the structure *Request*. An instance of the structure *Request* is said to be in the server when it is in the request queue, pending list, or ready queue. The details of structure *Request* are as follows.

Details of structure *Request*

- *ReqNo*: The number of data requests that are merged together and are represented by the instance of *Request*.
- *AvgTIS*: Average Time In Search buckets of the data requests represented by the instance of *Request*.

After receiving a data request, the server first determines the type of this data request. If the data request is belonging to Type I, the server calculates the contributions of the data request on the aggregate average and tuning times based on the analysis in Section III-B1 and accordingly updates $Stat_{\rm I}$. Since being able to be served by the current bucket, a Type I data request will neither be merged into a structure *Request* nor be inserted into the request queue, ready queue, and pending list.

On the other hand, when the data request is belonging to Type II, the server first checks whether it can be merged into an instance of structure *Request* in the server. If yes, the server accordingly updates the fields (i.e., $ReqNo$ and $AvgTIS$) of the instance of the structure *Request*. Otherwise, the server creates a new instance of the structure *Request* and inserts the instance into the request queue. Finally, the server calculates the contribution on the aggregate access and tuning times of the probe bucket of the data request according to the derivations in Section III-B2 and accordingly updates $Stat_{\rm II}$.

While an instance of *Request*, for example, r, is retrieved from the ready requests,[2] the server first calculates the average number of search buckets that each data request in r has by

$$AvgSBNo \leftarrow \frac{Bucket(j).start - r.ATIS}{d \times (S_D + S_{\rm I})}.$$

The contributions of these search buckets on the aggregate access and tuning times can be obtained from the derivations in Section III-B2, and $Stat_{\rm II}.AggATS$ and $Stat_{\rm II}.AggTTS$ are accordingly updated. The server calculates the time that the desired data item of r can be retrieved (i.e., $t_{\rm End}$). Finally, with $t_{\rm End}$, the server calculates the aggregate contributions of the retrieval buckets of all the data requests in r on the aggregate access and tuning times according to the derivations in Section III-B2 and accordingly updates $Stat_{\rm II}.AggATR$ and $Stat_{\rm II}.AggTTR$. The algorithmic form of the procedure to update $Stat_{\rm II}$ when an instance of the structure *Request* is served is as follows.

```
Procedure RequestServed($Request\ r$)
  1: $Stat_{\rm II}.ReqNo \leftarrow Stat_{\rm II}.ReqNo + r.ReqNo$
  2:     $AvgSNo \leftarrow (Bucket(j).start - r.ATIS) / (d \times (S_D + S_{\rm I}))$
  3:     $Stat_{\rm II}.AggATS \leftarrow Stat_{\rm II}.AggATS + (d \times ((S_{\rm I} + S_D)/B)) \times AvgSBNo \times r.ReqNo$
  4:     $Stat_{\rm II}.AggTTS \leftarrow Stat_{\rm II}.AggTTS + (d \times (S_{\rm I}/B) + T_{\rm On} + T_{\rm Off}) \times AvgSBNo \times r.ReqNo$
  5: Calculate $t_{\rm End}$ of $r$
  6:     $Stat_{\rm II}.AggATR \leftarrow Stat_{\rm II}.AggATR + (t_{\rm End} - Bucket(j).Start) \times r.ReqNo$
  7: Let $TTR$ be the tuning time of $r$ in the retrieval bucket
  8:     $Stat_{\rm II}.AggTTR \leftarrow Stat_{\rm II}.AggTTR + TTR \times r.ReqNo$
```

B. Degree Adjustment Phase

In each execution of the degree adjustment phase, the server will adjust the degree (i.e., the value of d) of buckets

[2] Readers can refer to Section V to see how the system retrieves instances of *Request* from the ready queue.

according to the statistic information collected in the precedent execution of the statistics collection phase. Let $T_{\text{Access}}(d)$ and $T_{\text{Tuning}}(d)$ be the average access and tuning times, respectively, when the degree of broadcast programs is d. For each field, the value of the *average* version is equal to the value of the *aggregate* version divided by the number of data requests. For example, the value of $Stat_{\text{I}}.AvgTT$ is equal to $Stat_{\text{I}}.AggTT/Stat_{\text{I}}.ReqNo$. Then, according to the analysis in Section III, we have

$$\begin{aligned}T_{\text{Access}}(d) =& W_{\text{I}} \times (Stat_{\text{I}}.AvgAT) + W_{\text{II}}\\ &\times (Stat_{\text{II}}.AvgATP + Stat_{\text{II}}.AvgATS\\ &\quad + Stat_{\text{II}}.AvgATR)\\ T_{\text{Tuning}}(d) =& W_{\text{I}} \times (Stat_{\text{I}}.AvgTT) + W_{\text{II}}\\ &\times (Stat_{\text{II}}.AvgTTP + Stat_{\text{II}}.AvgTTS\\ &\quad + Stat_{\text{II}}.AvgTTR)\end{aligned}$$

where W_{I} and W_{II} are the weights of the Type I and Type II data requests, respectively. The values of W_{I} and W_{II} are defined as the ratios of the numbers of Type I and Type II data requests. Hence, we have

$$\begin{aligned}W_{\text{I}} &= \frac{Stat_{\text{I}}.ReqNo}{Stat_{\text{I}}.ReqNo + Stat_{\text{II}}.ReqNo}\\ W_{\text{II}} &= \frac{Stat_{\text{II}}.ReqNo}{Stat_{\text{I}}.ReqNo + Stat_{\text{II}}.ReqNo}.\end{aligned}$$

In addition, $T_{\text{OverAll}}(d)$ is employed as the metric of system performance and is defined as

$$T_{\text{OverAll}}(d) = \beta \times T_{\text{Access}}(d) + (1-\beta) \times T_{\text{Tuning}}(d).$$

In the above equation, β is an administrator-specified parameter to reflect the relative importance of the average access time [i.e., $T_{\text{Access}}(d)$] and the average tuning time [i.e., $T_{\text{Tuning}}(d)$]. Hence, there is no optimal setting of β. The objective of the degree adjustment phase is to determine the new value of d to minimize $T_{\text{OverAll}}(d)$. However, since globally minimizing $T_{\text{OverAll}}(d)$ is difficult, algorithm AIDOA is designed to find the new value of d, for example, d_{Next}, where $T_{\text{OverAll}}(d_{\text{Next}})$ is the local minimum. That is, we will find a value of d_{Next} so that $T_{\text{OverAll}}(d_{\text{Next}})$ is smaller than $T_{\text{OverAll}}(d_{\text{Next}}+1)$ and $T_{\text{OverAll}}(d_{\text{Next}}-1)$. Since the exact values of $T_{\text{Access}}(d_{\text{Next}})$ and $T_{\text{Tuning}}(d_{\text{Next}})$ when $d_{\text{Next}} \neq d_{\text{Curr.}}$ cannot be obtained from the collected statistic information, we adopt the following approximation method to estimate $T_{\text{Access}}(d_{\text{Next}})$ and $T_{\text{Tuning}}(d_{\text{Next}})$.

Let $Stat_{\text{I}}^{d_{\text{Next}}}$ and $Stat_{\text{II}}^{d_{\text{Next}}}$ be the approximations of the values of $Stat_{\text{I}}$ and $Stat_{\text{II}}$ when the degree of buckets is d_{Next}. Then, we have the following lemmas.

Lemma 1: $Stat_{\text{I}}^{d_{\text{Next}}}.AvgAT$ and $Stat_{\text{I}}^{d_{\text{Next}}}.AvgTT$ can, respectively, be approximated by

$$\begin{aligned}Stat_{\text{I}}^{d_{\text{Next}}}.AvgAT &= Stat_{\text{I}}.AvgAT + (d_{\text{Next}} - d_{\text{Curr.}}) \times \frac{S_{\text{I}}}{B}\\ Stat_{\text{I}}^{d_{\text{Next}}}.AvgTT &= Stat_{\text{I}}.AvgTT.\end{aligned}$$

Lemma 2: $Stat_{\text{II}}^{d_{\text{Next}}}.AvgATP$ and $Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTP$ can, respectively, be approximated by

$$\begin{aligned}Stat_{\text{II}}^{d_{\text{Next}}}.AvgATP =& \frac{S_{\text{I}}}{S_{\text{I}}+S_D} \times Stat_{\text{II.I}}^{d_{\text{Next}}}.AvgATP\\ &+ \frac{S_D}{S_{\text{I}}+S_D} \times Stat_{\text{II.II}}^{d_{\text{Next}}}.AvgATP\\ Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTP =& \frac{S_{\text{I}}}{S_{\text{I}}+S_D} \times Stat_{\text{II.I}}^{d_{\text{Next}}}.AvgTTP\\ &+ \frac{S_D}{S_{\text{I}}+S_D} \times Stat_{\text{II.II}}^{d_{\text{Next}}}.AvgTTP\end{aligned}$$

where

$$\begin{aligned}Stat_{\text{II.I}}^{d_{\text{Next}}}.AvgATP =& Stat_{\text{II}}.AvgATP\\ &+ (d_{\text{Next}} - d_{\text{Curr.}}) \times \left(\frac{S_{\text{I}}}{B} + \frac{S_D}{B}\right)\\ Stat_{\text{II.I}}^{d_{\text{Next}}}.AvgTTP =& Stat_{\text{II}}.AvgTTP\\ &+ (d_{\text{Next}} - d_{\text{Curr.}}) \times \frac{S_{\text{I}}}{B}\\ Stat_{\text{II.II}}^{d_{\text{Next}}}.AvgATP =& Stat_{\text{II}}.AvgATP\\ &+ (d_{\text{Next}} - d_{\text{Curr.}}) \times \frac{S_D}{B}\\ Stat_{\text{II.II}}^{d_{\text{Next}}}.AvgTTP =& Stat_{\text{II}}.AvgTTP\\ &+ (d_{\text{Next}} - d_{\text{Curr.}}) \times \frac{S_D}{B}.\end{aligned}$$

As mentioned in Lemma 2, setting the degree of buckets from $d_{\text{Curr.}}$ to d_{Next} will increase the numbers of index and data items in each probe bucket of the Type II data requests by $d_{\text{Next}} - d_{\text{Curr.}}$. Suppose that these extra index and data items are from the search buckets. Then, we have Lemma 3.

Lemma 3: $Stat_{\text{II}}^{d_{\text{Next}}}.AvgATS$ and $Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTS$ can, respectively, be approximated as

$$\begin{aligned}Stat_{\text{II}}^{d_{\text{Next}}}.AvgATS =& AvgSBNo_{\text{Next}}\\ &\times d_{\text{Next}} \times \frac{(S_{\text{I}}+S_D)}{B}\\ Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTS =& AvgSBNo_{\text{Next}}\\ &\times \left(d_{\text{Next}} \times \frac{S_{\text{I}}}{B} + T_{\text{Off}} + T_{\text{On}}\right)\end{aligned}$$

where

$$AvgSBNo_{\text{Next}} = \frac{Stat_{\text{II}}.AvgATS \times B}{d_{\text{Next}} \times (S_{\text{I}}+S_D)} - \frac{d_{\text{Next}} - d_{\text{Curr.}}}{d_{\text{Next}}}.$$

Lemma 4: $Stat_{\text{II}}^{d_{\text{Next}}}.AvgATR$ and $Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTR$ can be approximated as

$$\begin{aligned}Stat_{\text{II}}^{d_{\text{Next}}}.AvgATR =& Stat_{\text{II}}.AvgATR\\ &+ (d_{\text{Next}} - d_{\text{Curr.}}) \times \frac{S_{\text{I}}}{B}\\ Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTR =& Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTR.\end{aligned}$$

The approximations of $T_{\text{Access}}(d_{\text{Next}})$ and $T_{\text{Tuning}}(d_{\text{Next}})$ can be calculated based on the above approximations. From the above lemmas, we have the following observations.

1) Increasing the value of the degree will increase the average tuning time in probe buckets since the number of

data items in a bucket increases. In addition, increasing the value of the degree will also reduce the aggregate tuning time of search buckets since the average number of search buckets decreases. The increase of the average tuning time in probe buckets and the decrease of the aggregate tuning time of search buckets are, respectively, the benefit and the cost of increasing the value of the degree.

2) To minimize the average tuning time, decreasing the value of the degree is encouraged when the average access time is short. It is because that decreasing the value of the degree will reduce the average tuning time in probe buckets by slightly increasing the aggregate tuning time of search buckets. Such an increase results from the increase in the average number of search buckets.

We then devise the procedure DegreeAdjustment to find the value of d_{Next}, where $T_{\text{OverAll}}(d_{\text{Next}})$ is the local minimum. In the procedure DegreeAdjustment, the server first checks whether increasing or decreasing the value of the degree will reduce the value of $T_{\text{OverAll}}(d_{\text{Next}})$. After that, the server repeatedly increases or decreases the value of the degree by one until $T_{\text{OverAll}}(d_{\text{Next}})$ is the local minimum. Finally, the system sets the value of the degree (i.e., $d_{\text{Curr.}}$) to the return value of the procedure DegreeAdjustment. The algorithmic form of the procedure DegreeAdjustment is as follows.

```
Procedure DegreeAdjustment
Note: The new value of $d$ (i.e., $d_{\text{Next}}$) is returned
 1: if ($T_{\text{OverAll}}(d_{\text{Curr.}}+1) < T_{\text{OverAll}}(d_{\text{Curr.}})$) then
 2:     $\delta \leftarrow 1$
 3: else if ($T_{\text{OverAll}}(d_{\text{Curr.}}-1) > T_{\text{OverAll}}(d_{\text{Curr.}})$) then
 4:     $\delta \leftarrow -1$
 5: else
 6:     return $d_{\text{Curr.}}$
 7: $d_{\text{Next}} \leftarrow d_{\text{Curr.}}$
 8: while ($T_{\text{OverAll}}(d_{\text{Next}}+\delta)) < T_{\text{OverAll}}(d_{\text{Next}})$) do
 9:     $d_{\text{Next}} \leftarrow d_{\text{Next}} + \delta$
10: return $d_{\text{Next}}$
```

C. Complexity Analysis

To derive the worst time complexity of algorithm AIDOA, we consider the case that no request merge occurs. Suppose that the number of received requests in one execution of the statistics collection phase is n. Then, the time complexity of one execution of the statistics collection phase is $O(n)$ since the time complexity of one execution of the procedure RequestServed is $O(1)$. Suppose that the maximal value of the degree is d_{Max}. The time complexity of the procedure DegreeAdjustment is $O(d_{\text{Max}})$. Since algorithm AIDOA executes the procedure DegreeAdjustment once in each execution of the degree adjustment phase, the time complexity of one execution of the degree adjustment phase is $O(d_{\text{Max}})$. To implement algorithm AIDOA, we have to spend the storage space to store structures $Stat_{\text{I}}$ and $Stat_{\text{II}}$. Since the sizes of structures $Stat_{\text{I}}$ and $Stat_{\text{II}}$ are fixed and are independent of n, the space complexity of algorithm AIDOA is $O(1)$.

V. Design of Program Generation Algorithm and Cache Replacement Policy

After determining the new value of the degree, the program generator will accordingly generate the successive buckets. Since the data items may be cached in the server cache, the adopted program generation algorithm should cooperate with the employed cache replacement policy. Each cached data item is initially marked as LOCKED, and only the cached data items in the UNLOCK state are candidates of replacement. To facilitate the design of the cache replacement policy, the system maintains a min heap *Cand* that stores all the data items in the UNLOCKED state according to their priorities. The definition of the priority of a data item will be given later in this section. Note that in this and the following section, we relax the assumption that all the data items are of the same size and denote the size of D_i as $size(D_i)$ and the average data size as S_D.

The server maintains a list $bucket$ that contains the index items and data items of the current bucket. Initially, $bucket$ is empty. Then, the server retrieves $d_{\text{Curr.}}$ data items from the head of the ready queue, inserts them into $bucket$, and marks them as LOCKED. In addition, the corresponding index items of the data items in $bucket$ are also inserted into $bucket$. Then, the server sequentially broadcasts the index items and data items in $bucket$. Once an item has been broadcast, it will be removed from $bucket$. If the item is a data item, it will be marked as UNLOCKED. Once $bucket$ becomes empty, the server retrieves $d_{\text{Curr.}}$ data items from the head of the ready queue and repeats the above procedure. The algorithmic form of the proposed program generation algorithm is as follows.

```
Algorithm ProgramGeneration
 1: while (true) do
 2:   $bucket \leftarrow$ BucketGeneration()
 3:   while ($bucket$ is not empty) do
 4:     $item \leftarrow$ the head of $bucket$
 5:     Remove the head of $bucket$
 6:     Broadcast $item$
 7:     if ($item$ is a data item) then
 8:       Mark $item$ as UNLOCKED
 9:       Calculate the priority of $item$ and insert $item$
          into $Cand$
```

```
Procedure BucketGeneration
 1: $bucket \leftarrow empty$
 2: for ($i = 1$ to $d_{\text{Curr.}}$) do
 3:   if (ready queue is empty)} then
 4:     break
 5:   Fetch a data item (denoted as $item$) from the head of
      ready queue
 6:   Mark $item$ as LOCKED
 7:   Append $item$ into $bucket$
 8: Insert the corresponding index items of the data items
    in $bucket$ into the head of $bucket$
 9: return $bucket$
```

We now consider the design of the server cache. Similar to other cache replacement policies, we define an evict function to determine the cache priorities of all the data items. The profit of caching a data item is defined as the overall data fetch time saving when the data item is cached. The cost of caching a data item is defined as the size of the data item. The cache replacement policy is designed to maximize the aggregate profit of all the cached data items under the limitation on the aggregate cost (i.e., size) of all the cached data items. Hence, the cache priority of a data item D_i is defined as

$$priority(D_i) = \frac{fetch(D_i) \times rate(D_i)}{size(D_i)}$$

where $fetch(D_i)$ is the time for the server to fetch D_i from the data server of D_i, and $rate(D_i)$ is the request rate of D_i. When retrieving D_i from the corresponding data server, the server calculates the value of $fetch(D_i)$ and stores it for further uses. The server also stores the time of the previous cache hit of D_i, which is denoted as $t_{\mathrm{PrevHit}}(D_i)$. In addition, for each cache hit of D_i, $rate(D_i)$ is set to

$$\frac{1}{t_{\mathrm{CurHit}} - t_{\mathrm{PrevHit}}(D_i)}$$

where t_{CurHit} is the time of the current cache hit of D_i. After the calculation of the request rate of D_i, $t_{\mathrm{PrevHit}}(D_i)$ is set to t_{CurHit}.

The proposed cache replacement policy is as follows. When a data item, for example, D_i, is retrieved from the data server, it will be inserted into the cache. When inserting D_i into the cache, the server first checks whether the cache is of enough free space for D_i. If yes, the system stores D_i into the cache, calculates $priority(D_i)$, and marks D_i as LOCKED. Otherwise, the system repeatedly removes "the data item with the smallest priority among all the data items in $Cand$" from $Cand$ until the free space of the cache becomes enough. Then, the system stores D_i into the cache, calculates $priority(D_i)$, and marks D_i as LOCKED. The algorithmic form of the proposed cache replacement policy is as follows.

```
Algorithm CacheReplacement($D_i$)
  1: **while** ($FreeSpace < size(D_i)$) **do**
  2:     Let $D_j$ be the data item with the smallest priority
         among all other data items in $Cand$
  3:     Remove $D_j$ from cache
  4:     $FreeSpace \leftarrow FreeSpace + size(D_j)$
  5: Insert $D_i$ into cache
  6: Calculate the $priority(D_i)$
  7: Mark $D_i$ as LOCKED
```

Suppose that the data items in $Cand$ are organized as a min heap. In addition, let n_{Replace} be the number of data items to be replaced. Therefore, the time complexity of one execution of the algorithm CacheReplacement is $O(n_{\mathrm{Replace}} \times \log |Cand|)$.

VI. Performance Evaluation

A. Simulation Model

We take the LWF as the underlying scheduling algorithm to prioritize the data requests in the request queue and the ready queue. The server provides one request channel and one broadcast channel with network bandwidth of 38.4 and 384 kb/s, respectively. Analogous to [8], we assume that there are 4000 data objects, and the sizes of the data objects follow a lognormal distribution with a mean of 7 kB. The size of a data request message and an index item is set to 128 bytes. The times to turn on and turn off the WNIs are both set to 30 ms. The access probability of the data objects follows a Zipf distribution, which is widely adopted as a model for real Web traces [3], [7]. The parameter of the Zipf distribution is set to 0.75 with reference to the analyses of real Web traces [7], [15]. Since the small objects are much more frequently accessed than the large ones [9], we assume that there is a negative correlation between the object size and its access probability. The default capacity of the cache is set to $0.01 \times \sum$ object size, and the fetch delays of the data objects follow an exponential distribution with a mean of 2.3 s [8]. Similar to [16], the number of users in the network is set to 250. The service holding time and the service reestablishing time for each user are set to exponential distributions with means of 10 min and 1 h, respectively. The service reestablishing time is defined as the time interval between the moment that a user terminates the service and the moment that the user establishes the service again. We also assume that the interarrival times of data requests of each user follow an exponential distribution with a mean 10 s [14]. The value of β is set to 0.5 to simulate the environment that the average access time and the average tuning time are of equal importance (Table II).

TABLE II
Default System Parameters

Parameter	Value
Data object number	4000
Data object sizes	Lognormal dist. (mean 7 KB)
Data access probabilities	Zipf dist. with parameter 0.75
Cache capacity	$0.01 \times \sum$object size
Object fetch delay	Exponential dist. with μ = 2.3
Client number	250
Service holding time	Exp. dist. with μ = 10 minutes
Service establishing time	Exp. dist. with μ = one hour

To evaluate the performance of the proposed degree adjustment method in algorithm AIDOA, the algorithm proposed in [13] (referred to as algorithm Static) is modified to cooperate with the cache replacement policy and the program generation algorithm proposed in Section V. Hence, the difference between algorithm AIDOA and algorithm Static is only on the ability of adjusting the degree of buckets. Based on the algorithm Static, we devise two schemes, i.e., Static-2 and Static-8, which set the degree of buckets to 2 and 8, respectively, and the values of the degree of buckets are fixed throughout the simulation. In addition, scheme AIDOA employs algorithm AIDOA and initializes the degree of buckets to 2. Hence, scheme AIDOA will dynamically adjust the degree of buckets according to the system workload. Note that all these three schemes employ a server cache to eliminate the performance degradation caused by the data fetch time.

B. Effect of Average Data Size

In this experiment, we investigate the effect of the average data size on the average access and tuning times. The average data size is set from 2 to 11 kB, and the experimental results are shown in Fig. 9(a) and (b), respectively. Due to increasing the load of the broadcast channel, it is intuitive that increasing the

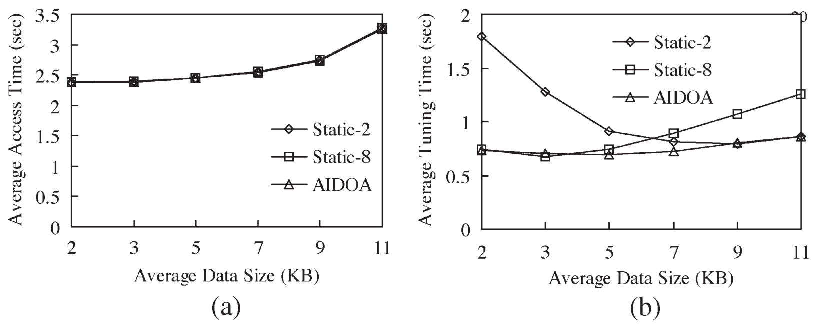


Fig. 9. Effect of average data size. (a) Average access time. (b) Average tuning time.

average data size results in the increase in the average access time. In addition, when the average data size is large enough, the load of the broadcast channel is high, and hence, a slight increase in the average data size will cause significant increases in the average access time. Since the sizes of the index items are much smaller than those of the data items, the effect of the degrees of broadcast programs in the average access time is quite small.

Although the values of the degrees of broadcast programs only slightly affect the average access time of all the schemes, they result in significant effects in the average tuning time. As shown in Fig. 9(b), scheme Static-8 performs well only when the average data size is small, and scheme Static-2 performs well only when the average data size is large. As observed in Section IV-B, increasing the value of the degree will increase the average tuning time in the probe bucket. In addition, increasing the value of the degree also decreases the number of search buckets and hence reduces the average tuning time in the search buckets. When the average data size is large, the average access time is also long. Employing a large value of degree will reduce the average tuning time in the search buckets by reducing the number of search buckets (i.e., reducing the number of times of turning-on and turning-off the WNIs) and increase the average tuning time in the probe bucket. However, due to the tradeoff between the average tuning times in the probe bucket and the search buckets, the value of the degree cannot be set to be too large. In addition, we can also observe from Section IV-B that decreasing the value of the degree will decrease the average tuning time in the probe bucket and increase the number of search buckets. Therefore, the scheme with small values of degree outperforms schemes with large values of degree in the case with a small average data size. Hence, the value of the degree cannot be set to be too small either. Different from schemes Static-2 and Static-8, since scheme AIDOA is able to dynamically adjust the value of the degree to a proper value according to the system workload, scheme AIDOA outperforms schemes Static-2 and Static-8 in most cases.

C. *Effect of Turning-On and Turning-Off Times of WNIs*

The effect of the turning-on and turning-off times of WNIs is measured in this subsection, and the experimental results are given in Fig. 10. In this experiment, we assume that $T_{\mathrm{On}} = T_{\mathrm{Off}}$ and set the value of T_{On} and T_{Off} from 5 to 60 ms. As shown in Fig. 10(a), the values of T_{On} and T_{Off} do not affect the average access time of schemes Static-2 and Static-8. It is because that in these two schemes, the degrees of broadcast

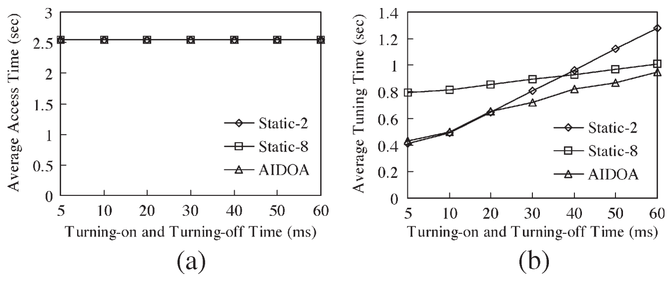


Fig. 10. Effect of turning-on and turning-off time. (a) Average access time. (b) Average tuning time.

buckets are fixed, and the values of T_{On} and T_{Off} do not affect the organizations of broadcast programs. On the other hand, although scheme AIDOA is able to dynamically adjust the degree of buckets, the influence of T_{On} and T_{Off} on the average access time of scheme AIDOA is small since the size of the index items is much smaller than that of the data items.

Consider the average tuning time of these schemes shown in Fig. 10(b). According to the observations in Section IV-B, increasing the value of the degree will increase the average tuning time in the probe bucket and reduce the aggregate tuning time in the search buckets. Since the benefit of increasing the value of degree is in proportion to the values of T_{On} and T_{Off}, scheme Static-2 performs well when T_{On} and T_{Off} are small. On the contrary, scheme Static-8 outperforms scheme Static-2 in the case with large T_{On} and T_{Off}. Although producing more power consumption in the probe bucket than scheme Static-2 does, scheme Static-8 is still able to reduce the overall power consumption since being able to greatly reduce the power consumption on turning-on and turning-off the WNIs by reducing the average number of search buckets. On the other hand, with dynamic adjustment in the degree, scheme AIDOA is able to determine a suitable value of degree for the current system workload and hence outperforms schemes Static-2 and Static-8 in most cases.

D. *Effect of the Number of Users*

In this subsection, we evaluate the scalability of these schemes in the average access and tuning times by increasing the number of users from 200 to 450. The experimental results are shown in Fig. 11.

Due to the characteristics of data broadcasting, it is intuitive that increasing the number of users results in a smoothly increasing average access time. As shown in Fig. 11(a), when the number of users is small, increasing the number of users

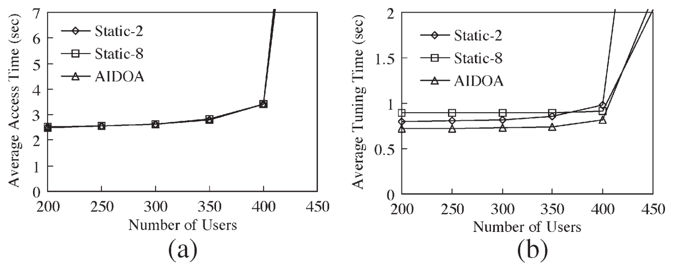


Fig. 11. Effect of the number of users. (a) Average access time. (b) Average tuning time.

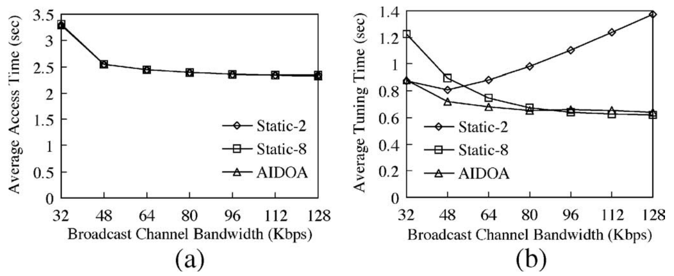


Fig. 12. Effect of bandwidth of broadcast channel. (a) Average access time. (b) Average tuning time.

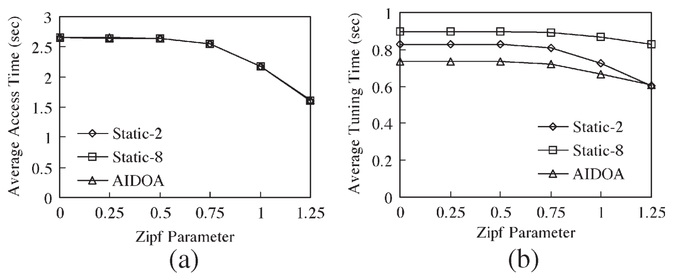


Fig. 13. Effect of skewness of data requests. (a) Average access time. (b) Average tuning time.

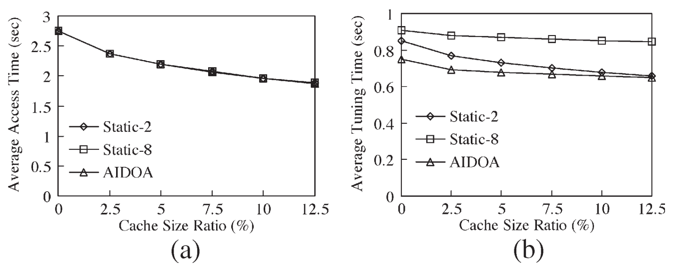


Fig. 14. Effect of cache size ratio. (a) Average access time. (b) Average tuning time.

produces a slight increase in the average access time since the system load is still light. However, when the number of users is large enough, the system load becomes high, and increasing the number of users results in drastic increases in the average access time. In addition, since the size of the index items is much smaller than that of the data items, the average access time of these schemes is close.

Fig. 11(b) shows the average tuning time of these schemes with the number of users varied. We observe that when the number of users is not large, scheme Static-2 outperforms scheme Static-8. In addition, when the number of users is large, the average tuning time of scheme Static-2 becomes longer than that of scheme Static-8. This phenomenon agrees with the observations in Section IV-B, where in the average tuning time the cases with short average access time favor schemes with small values of degree and the cases with long average access time favor schemes with large values of degree. Therefore, when the number of users is large enough, the average tuning time of scheme Static-2 becomes much longer than that of scheme Static-8 since the average access time of these schemes both becomes drastically increasing. Since being able to adjust the values of degree according to the system workload, scheme AIDOA outperforms schemes Static-2 and Static-8 when the number of users is not large. In addition, the average tuning time of scheme AIDOA is close to that of scheme Static-8 when the number of users is large.

E. Effect of Bandwidth of the Broadcast Channel

In this experiment, we investigate the effect of the bandwidth of the broadcast channel on the average access and tuning times by setting the bandwidth from 32 to 128 kb/s. The experimental results are shown in Fig. 12.

It is intuitive that increasing the bandwidth of the broadcast channel decreases the average access time. However, as shown in Fig. 12(a), the effect of increasing the bandwidth of the broadcast channel on the average access time diminishes when the bandwidth is large enough. It is because that the average access time comprises several components, such as the fetch time and the broadcast time of data items, the waiting time of data requests spending in queues, and the transmission time of data requests on the request channel. Since increasing the bandwidth of the broadcast channel only reduces the broadcast time of data items, the effect of increasing the bandwidth of the broadcast channel on the average access time is limited. Similar to the precedent experiments, the average access time of these schemes is close.

Now consider the experimental results on the average tuning time shown in Fig. 12(b). As observed in Fig. 12(b), scheme Static-8 performs well only when the bandwidth of the broadcast channel is high. Similarly, scheme Static-2 performs well only when the bandwidth of the broadcast channel is low. Consider the scenario of increasing the value of degree. As observed in Section IV-B, with the same increment in the value of degree, increasing the bandwidth of the broadcast channel will reduce the cost of increasing the value of degree by reducing the average tuning time in the probe bucket. Since most average tuning time in the search buckets comes from turning-on and turning-off the WNIs, the average tuning time in the search buckets does not affect by the increase of the bandwidth of the broadcast channel. Therefore, increasing the bandwidth of the broadcast channel favors increasing the value of degree. Similarly, decreasing the bandwidth of the broadcast channel makes increasing the value of degree more costly. On the other hand, scheme AIDOA is able to adjust the value of degree according to the system workload and hence outperforms schemes Static-2 and Static-8 in most cases. This phenomenon shows the advantage of scheme AIDOA due to its adaptability to the system workload.

F. Effect of Skewness of the Access Probabilities of Data Requests

We evaluate in this experiment the effect of skewness of the access probabilities of data requests by setting the value of the Zipf parameter from 0 to 1.25. The larger the value of the Zipf parameter, the more skewed are the access probabilities of data requests. In addition, the value of the Zipf parameter is set to 0 to indicate the case that the access probabilities of data requests are equal. It is intuitive that when the access probabilities become more skewed, more data requests are merged together, and the average access time becomes shorter. The result shown in Fig. 13(a) agrees with this intuition.

Fig. 13(b) shows the average access time of all the schemes with the value of the Zipf parameter varied. As observed in Fig. 13(b), the average tuning time decreases as the value of the Zipf parameter increases. It can be explained by the observations in Section IV-B that in the average tuning time, cases with short average access time favor schemes with small values of degree. Therefore, scheme Static-2 outperforms scheme Static-8 particularly in cases with a high value of the Zipf parameter. With the change of skewness of the access probabilities, scheme AIDOA is able to adjust the value of the degree to adapt to such a change and hence attains a better performance.

G. Effect of Cache Size

This experiment evaluates the effect of cache size on the average access and tuning times. The experimental results are shown in Fig. 14. Similar to [8], the cache size is determined as "cache size ratio $\times$ the summation of the sizes of all data items." The case that the value of the cache size ratio is set to 0 indicates the case that the server does not employ cache.

As shown in Fig. 14(a), the average access time decreases as the value of the cache size ratio increases. When the cache size is large, many data items are cached and can be obtained by the server without being fetched from the data servers. In addition, when the cache size is large enough, the benefit of increasing the cache size diminishes since data items with high access rates are cached in the server cache. We also observe from Fig. 14(a) that employing the server cache, which is neglected in prior studies on data indexing for on-demand data broadcasting [13], is able to effectively reduce the average access time.

As the observations in Section IV-B show, when it comes to the average tuning time, cases with a short average access time favor schemes with small values of degree. Hence, scheme Static-2 outperforms scheme Static-8 particularly when the value of the cache size ratio is large. This observation agrees with the results shown in Fig. 14(b). When the value of the cache size ratio is small, both schemes do not perform well since the value of the degree in scheme Static-2 is too small and the value of the degree in scheme Static-8 is too large. On the other hand, scheme AIDOA is able to dynamically adjust the value of the degree to attain a better performance, which shows the advantage of scheme AIDOA.

VII. Conclusion

In this paper, we have proposed an energy-conserving on-demand data broadcasting system that employs the data indexing technique. Different from prior work, the power consumption of turning on and turning off the WNIs was considered. In addition, we also employed the server cache to reduce the effect of the data fetch time. Specifically, we first analyzed the access and tuning times of data requests, and proposed algorithm AIDOA to adjust the degree of buckets according to the system workload. We also devised an approximation method to estimate the effect of increasing and decreasing the values of degree, and employed the approximation method to guide the adjustment of algorithm AIDOA. In addition, the companion program generation algorithm and the cache replacement policy were proposed to cooperate with algorithm AIDOA. Several experiments were then conducted to evaluate the performance of algorithm AIDOA. The experimental results showed that algorithm AIDOA is able to greatly reduce the power consumption at the cost of a slight increase in the average access time and dynamically adjust the index and data organization to adapt to the change of system workload.

Appendix

Proof of Lemma 1: Consider the cases that $d_{\text{Next}} > d_{\text{Curr.}}$. Fig. 15 shows an example of Type I data requests. When the degree of buckets is set from $d_{\text{Curr.}}$ to d_{Next}, $d_{\text{Next}} - d_{\text{Curr.}}$ index item(s) and $d_{\text{Next}} - d_{\text{Curr.}}$ data item(s) will be appended to each index segment and data segment, respectively. As observed from Fig. 15, setting the degree of buckets from $d_{\text{Curr.}}$ to d_{Next} increases the average access time of Type I data requests by $((d_{\text{Next}} - d_{\text{Curr.}}) \times S_{\text{I}})/B$. In addition, we also observe that appending $d_{\text{Next}} - d_{\text{Curr.}}$ index items into each index segment does not affect the average tuning time of Type I requests. Hence, from the above observations, we have

$$Stat_{\text{I}}^{d_{\text{Next}}}.AvgAT = Stat_{\text{I}}.AvgAT + (d_{\text{Next}} - d_{\text{Curr.}}) \times \frac{S_{\text{I}}}{B}$$
$$Stat_{\text{I}}^{d_{\text{Next}}}.AvgTT = Stat_{\text{I}}.AvgTT.$$

We then apply the above equations as the approximations of $Stat_{\text{I}}^{d_{\text{Next}}}.AvgAT$ and $Stat_{\text{I}}^{d_{\text{Next}}}.AvgTT$ in the cases that $d_{\text{Next}} < d_{\text{Curr.}}$, and hence prove Lemma 1. ■

Proof of Lemma 2: Consider the example probe bucket of Type II data requests shown in Fig. 16. Suppose that t_{Start} follows a uniform distribution between $Bucket_i.Start$ and $Bucket_i.End$. Therefore, as observed from Fig. 16, the probabilities of a Type II data request to be Type II.I and Type II.II are $S_{\text{I}}/(S_{\text{I}} + S_D)$ and $S_D/(S_{\text{I}} + S_D)$, respectively. Therefore, by the definition of $Stat_{\text{II}}^{d_{\text{Next}}}.AvgATP$ and $Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTP$, we have

$$\begin{aligned}Stat_{\text{II}}^{d_{\text{Next}}}.AvgATP &= \frac{S_{\text{I}}}{S_{\text{I}}+S_D} \times Stat_{\text{II.I}}^{d_{\text{Next}}}.AvgATP\\ &\quad + \frac{S_D}{S_{\text{I}}+S_D} \times Stat_{\text{II.II}}^{d_{\text{Next}}}.AvgATP\\ Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTP &= \frac{S_{\text{I}}}{S_{\text{I}}+S_D} \times Stat_{\text{II.I}}^{d_{\text{Next}}}.AvgTTP\\ &\quad + \frac{S_D}{S_{\text{I}}+S_D} \times Stat_{\text{II.II}}^{d_{\text{Next}}}.AvgTTP.\end{aligned}$$

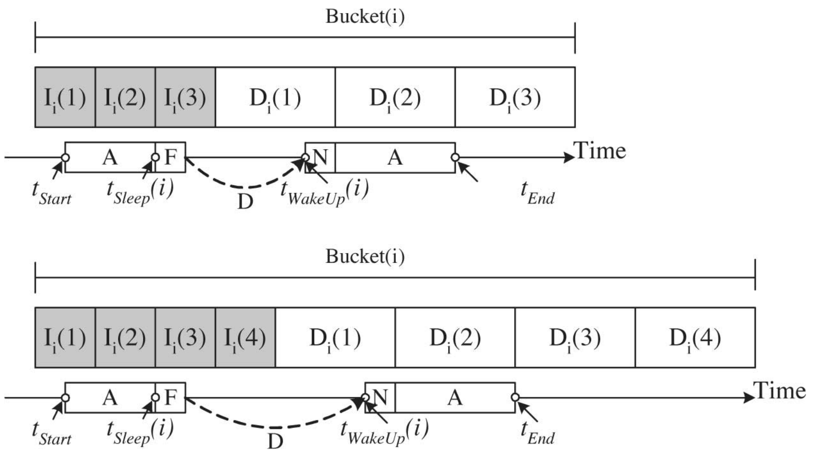

Bucket(i)
I_i(1) I_i(2) I_i(3) D_i(1) D_i(2) D_i(3)
A F N A D
Time
t_Start t_Sleep(i) t_WakeUp(i) t_End
I_i(4) D_i(4)

Fig. 15. Example scenario of Type I data requests that $d_{\mathrm{Curr.}} = 3$ and $d_{\mathrm{Next}} = 4$.

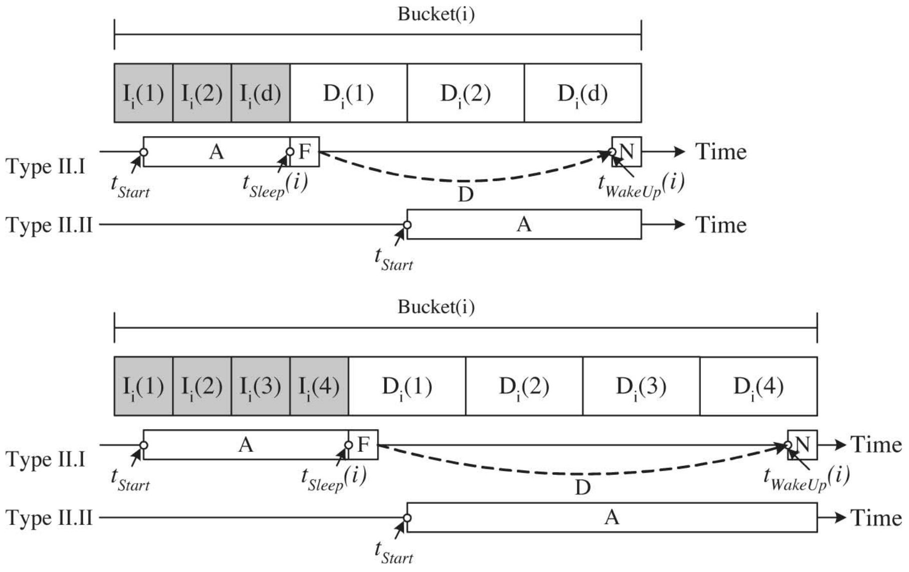

Bucket(i)
I_i(1) I_i(2) I_i(d) D_i(1) D_i(2) D_i(d)
Type II.I
Type II.II
A F N D
Time
t_Start t_Sleep(i) t_WakeUp(i)
I_i(3) I_i(4) D_i(3) D_i(4)

Fig. 16. Example scenario of Type II data requests that $d_{\mathrm{Curr.}} = 3$ and $d_{\mathrm{Next}} = 4$ on the probe bucket.

We now consider the cases that $d_{\mathrm{Next}} > d_{\mathrm{Curr.}}$ and derive the approximations of $Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgATP$, $Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgTTP$, $Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgATP$, and $Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgTTP$. When the degree of buckets is set from $d_{\mathrm{Curr.}}$ to d_{Next}, $d_{\mathrm{Next}} - d_{\mathrm{Curr.}}$ index item(s) and $d_{\mathrm{Next}} - d_{\mathrm{Curr.}}$ data item(s) will be appended to each index segment and data segment, respectively. As observed from Fig. 16, setting the degree of buckets from $d_{\mathrm{Curr.}}$ to d_{Next} increases the average access and tuning times of the probe buckets of Type II.I data requests (i.e., $Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgATP$ and $Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgTTP$) by $((d_{\mathrm{Next}} - d_{\mathrm{Curr.}}) \times (S_{\mathrm{I}} + S_D))/B$ and $((d_{\mathrm{Next}} - d_{\mathrm{Curr.}}) \times S_{\mathrm{I}})/B$, respectively. Hence, we have

$$\begin{aligned} Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgATP &= Stat_{\mathrm{II}}.AvgATP \\ &\quad + (d_{\mathrm{Next}} - d_{\mathrm{Curr.}}) \times \left(\frac{S_{\mathrm{I}}}{B} + \frac{S_D}{B}\right) \\ Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgTTP &= Stat_{\mathrm{II}}.AvgTTP \\ &\quad + (d_{\mathrm{Next}} - d_{\mathrm{Curr.}}) \times \frac{S_{\mathrm{I}}}{B}. \end{aligned}$$

In addition, we also observe that increasing the degree of buckets from $d_{\mathrm{Curr.}}$ to d_{Next} increases the average access and tuning times of the probe buckets of Type II.II data requests (i.e., $Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgATP$ and $Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgTTP$) by $((d_{\mathrm{Next}} - d_{\mathrm{Curr.}}) \times (S_{\mathrm{I}} + S_D))/B$. Therefore, we have

$$\begin{aligned} Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgATP &= Stat_{\mathrm{II}}.AvgATP \\ &\quad + (d_{\mathrm{Next}} - d_{\mathrm{Curr.}}) \times \frac{S_D}{B} \\ Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgTTP &= Stat_{\mathrm{II}}.AvgTTP \\ &\quad + (d_{\mathrm{Next}} - d_{\mathrm{Curr.}}) \times \frac{S_D}{B}. \end{aligned}$$

We then apply the above equations to the approximations of $Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgATP$, $Stat_{\mathrm{II.I}}^{d_{\mathrm{Next}}}.AvgTTP$, $Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgATP$, and $Stat_{\mathrm{II.II}}^{d_{\mathrm{Next}}}.AvgTTP$ in the cases that $d_{\mathrm{Next}} < d_{\mathrm{Curr.}}$, and hence prove Lemma 2. ■

Proof of Lemma 3: In the cases that the degree of buckets is $d_{\mathrm{Curr.}}$, since one data item and the corresponding index item contribute $Stat_{\mathrm{II}}.AvgATS$ by $(S_{\mathrm{I}} + S_D)/B$, the average number of data items in Type II data requests is $Stat_{\mathrm{II}}.AvgATS \times (B/(S_{\mathrm{I}} + S_D))$.

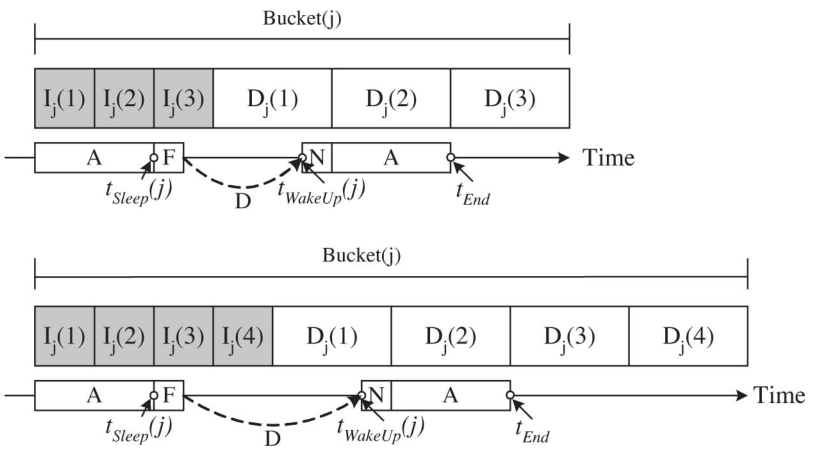


Fig. 17. Example scenario of Type II data requests that $d_{\text{Curr.}} = 3$ and $d_{\text{Next}} = 4$ on the retrieval bucket.

Consider the cases that set the degree of buckets to d_{Next}. For each Type II data request, on average, $d_{\text{Next}} - d_{\text{Curr.}}$ index items and $d_{\text{Next}} - d_{\text{Curr.}}$ data items move from the search buckets to the probe bucket. Therefore, the average number of index and data items in Type II data requests both become $Stat_{\text{II}}.AvgATS \times (B/(S_{\text{I}} + S_D)) - (d_{\text{Next}} - d_{\text{Curr.}})$. Since each search bucket contains d_{Next} index items and d_{Next} data items, the average number of search buckets of Type II data requests is

$$AvgSBNo_{\text{Next}} = Stat_{\text{II}}.AvgATS \times \frac{B}{S_{\text{I}} + S_D} - (d_{\text{Next}} - d_{\text{Curr.}}).$$

Finally, according to the derivations in Section III-B2, since each Type II data request contains $AvgSBNo_{\text{Next}}$ search buckets on average, we have

$$Stat_{\text{II}}^{d_{\text{Next}}}.AvgATS = AvgSBNo_{\text{Next}} \times d_{\text{Next}} \times \frac{(S_{\text{I}} + S_D)}{B}$$

$$Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTS = AvgSBNo_{\text{Next}} \times \left(d_{\text{Next}} \times \frac{S_{\text{I}}}{B} + T_{\text{Off}} + T_{\text{On}}\right). \qquad \blacksquare$$

Proof of Lemma 4: Consider the cases that $d_{\text{Next}} > d_{\text{Curr.}}$ and the example Type II data request shown in Fig. 17. When the degree of buckets is set to from $d_{\text{Curr.}}$ to d_{Next}, $d_{\text{Next}} - d_{\text{Curr.}}$ index item(s) and $d_{\text{Next}} - d_{\text{Curr.}}$ data item(s) will be appended to each index segment and data segment, respectively. As observed from Fig. 17, setting the degree of buckets from $d_{\text{Curr.}}$ to d_{Next} increases the average access time of each Type II data request on the retrieval bucket by $((d_{\text{Next}} - d_{\text{Curr.}}) \times S_{\text{I}})/B$. In addition, we also observe that appending $d_{\text{Next}} - d_{\text{Curr.}}$ index items into each index segment does not affect the average tuning time of each Type II request on the retrieval bucket. Hence, from the above observations, we have

$$Stat_{\text{II}}^{d_{\text{Next}}}.AvgATR = Stat_{\text{II}}.AvgATR + (d_{\text{Next}} - d_{\text{Curr.}}) \times \frac{S_{\text{I}}}{B}$$

$$Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTR = Stat_{\text{II}}.AvgTTR.$$

We then apply the above equations as the approximations of $Stat_{\text{II}}^{d_{\text{Next}}}.AvgATR$ and $Stat_{\text{II}}^{d_{\text{Next}}}.AvgTTR$ in the cases that $d_{\text{Next}} < d_{\text{Curr.}}$, and hence prove Lemma 4. $\blacksquare$

REFERENCES

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, “Broadcast disks: Data management for asymmetric communication environments,” in *Proc. ACM SIGMOD Conf.*, Mar. 1995, pp. 198–210.
[2] S. Acharya and S. Muthukrishnan, “Scheduling on-demand broadcasts: New metrics and algorithms,” in *Proc. 4th ACM/IEEE Int. Conf. Mobile Comput. Netw.*, Oct. 1998, pp. 43–94.
[3] C. Aggarwal, J. L. Wolf, and P. S. Yu, “Caching on the World Wide Web,” *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 94–107, Jan./Feb. 1999.
[4] M. Agrawal, A. Manjhi, N. Bansal, and S. Seshan, “Improving Web performance in broadcast-unicast networks,” in *Proc. IEEE INFOCOM Conf.*, Mar./Apr. 2003, pp. 229–239.
[5] D. Aksoy and M. J. Franklin, “Scheduling for large-scale on-demand data broadcasting,” in *Proc. IEEE INFOCOM Conf.*, Mar. 1998, pp. 651–659.
[6] D. Aksoy, M. J. Franklin, and S. Zdonik, “Data staging for on-demand broadcast,” in *Proc. 27th Int. Conf. Very Large Data Bases*, Sep. 2001, pp. 571–580.
[7] L. Breslau, P. Cao, G. Phillips, and S. Shenker, “Web caching and Zipf-like distributions: Evidence and implications,” in *Proc. IEEE INFOCOM Conf.*, Mar. 1999, pp. 126–134.
[8] C.-Y. Chang and M.-S. Chen, “Exploring aggregate effect with weighted transcoding graphs for efficient cache replacement in transcoding proxies,” in *Proc. 18th IEEE Int. Conf. Data Eng.*, Feb. 2002, pp. 383–392.
[9] S. Glassman, “A caching relay for the World Wide Web,” *Comput. Netw. ISDN Syst.*, vol. 27, no. 2, pp. 165–173, Nov. 1994.
[10] 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.
[11] J.-L. Huang and W.-C. Peng, “An energy-conserved on-demand data broadcasting system,” in *Proc. 6th Int. Conf. Mobile Data Manage.*, May 2005, pp. 234–238.
[12] T. Imielinski, S. Viswanathan, and B. R. Badrinath, “Data on air: Organization and access,” *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 9, pp. 353–372, Jun. 1997.
[13] S. Lee, D. P. Carney, and S. Zdonik, “Index hint for on-demand broadcasting,” in *Proc. 19th IEEE Int. Conf. Data Eng.*, Mar. 2003, pp. 726–728.
[14] C.-W. Lin, H. Hu, and D. L. Lee, “Adaptive realtime bandwidth allocation for wireless data delivery,” *Wirel. Netw.*, vol. 10, no. 2, pp. 103–120, Mar. 2004.
[15] V. Padmanabhan and L. Qiu, “The content and access dynamics of a busy Web site: Findings and implications,” in *Proc. IEEE SIGCOMM Conf.*, Aug./Sep. 2000, pp. 293–304.
[16] M. A. Sharaf and P. K. Chrysanthis, “On-demand data broadcasting for mobile decision making,” *ACM/Kluwer Mobile Netw. Appl.*, vol. 9, no. 4, pp. 703–714, Dec. 2004.
[17] E. Shih, P. Bahl, and M. J. Sinclair, “Wake on wireless: An event driven energy saving strategy for battery operated devices,” in *Proc. 8th ACM/IEEE Int. Conf. Mobile Comput. Netw.*, Sep. 2002, pp. 160–171.

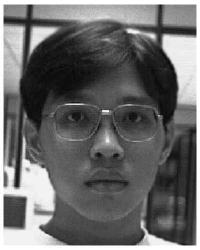
[18] T. Simunic, S. Boyd, and P. Glynn, "Managing power consumption in networks on chips," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 1, pp. 96–107, Jan. 2004.

[19] M. A. Viredaz, L. S. Brakmo, and W. R. Hamburgen, "Energy management on handheld devices," *ACM Queue*, vol. 1, no. 7, pp. 44–52, Oct. 2003.

[20] J. Xu, W.-C. Lee, and X. Tang, "Exponential index: A parameterized distributed indexing scheme for data on air," in *Proc. 2nd ACM/USENIX Int. Conf. Mobile Syst.*, Jun. 2004, pp. 153–164.

[21] J. L. Xu, B. Zheng, W.-C. Lee, and D. K. Lee, "Energy efficient index for querying location-dependent data in mobile broadcast environments," in *Proc. 19th Int. Conf. Data Eng.*, Mar. 2003, pp. 239–250.

[22] H. Zhu and G. Cao, "A power-aware and QoS-aware service model on wireless networks," in *Proc. IEEE INFOCOM Conf.*, Mar. 2004, pp. 1393–1403.



Jiun-Long Huang received the B.S. and M.S. degrees from the National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1997 and 1999, respectively, and the Ph.D. degree from the National Taiwan University, Taipei, Taiwan, in 2003.

He is currently an Assistant Professor with the Department of Computer Science, National Chiao Tung University. His research interests include mobile computing, wireless networks, and data mining.