Adaptive Power-Aware Prefetch in Wireless Networks

Liangzhong Yin, *Student Member, IEEE* and Guohong Cao, *Member, IEEE*

***Abstract*—Most of the prefetch techniques used in the current cache management schemes do not consider power constraints of the mobile clients and other factors such as the size of the data items, the data access rate, and the data update rate. In this paper, we address these issues by proposing a power-aware prefetch scheme, called adaptive value-based prefetch (AVP) scheme. The AVP scheme defines a value function which can optimize the prefetch cost to achieve better performance. Also, AVP dynamically adjusts the number of prefetches to get better tradeoff between performance and power. As stretch is widely adopted as a performance metric for variable-size data requests, we show by analysis that the proposed approach can indeed achieve the optimal performance in terms of stretch when power consumption is considered. Simulation results demonstrate that our algorithm significantly outperforms existing prefetching algorithms under various scenarios.**

***Index Terms*—Cache management, invalidation report, power conservation, prefetch, stretch.**

I. Introduction

BROADCASTING has been shown to be an effective data dissemination technique in wireless networks by many studies [2], [3], [13], [14]. With this technique, clients access data by simply monitoring the broadcast channel until the requested data appear in the channel. The broadcasting model exploits the asymmetric nature of the wireless channel, where more bandwidth is available for the downlink (server-to-client), but less bandwidth for the uplink (client-to-server). Further, the model is scalable since the bandwidth consumption is independent of the number of clients in the system.

Although data broadcasting has many advantages, it also introduces some problems. For example, waiting for the data to appear in the broadcast channel may increase the query latency. One way to alleviate this problem is to cache frequently accessed data on the client side [3], [7], [19], [20]. In this way, the client can serve many requests from the local cache without sending uplink requests. This not only reduces the average data access delay but also reduces the uplink and downlink bandwidth consumption.

To further reduce the access latency and improve the cache hit ratio, prefetching techniques can be used. Prefetching has many advantages in mobile environments since wireless networks such as wireless local area networks (LANs) or cellular networks support broadcasting. When the server broadcasts data on the broadcast channel, clients can prefetch interested data without increasing the bandwidth consumption. Note that if the requested data item is not prefetched earlier, the client has to send an uplink request when receiving the query. This not only increases the query delay but also increases the uplink bandwidth requirement. Since the uplink bandwidth is very expensive in wireless networks, prefetching should be used frequently. However, prefetching consumes a large amount of system resources such as battery power on the client side. Although prefetching can make use of the broadcast channel, clients still need to consume power to receive and process the data. Further, they cannot power off the wireless network interface, which consumes a large amount of power even when it is in the idle mode [18]. Since most mobile clients are powered by battery, it is important to prefetch the right data when designing prefetching schemes. Unfortunately, most of the prefetch techniques used in the current cache management schemes [7], [12] do not consider power constraints of the mobile clients and other factors such as the data size, the data access rate, and the data update rate. To address these issues, we first propose a value-based (VP) scheme, which makes prefetch decisions based on the value of each data item considering various factors such as access rate, update rate, and data size. As stretch [1], [19] is widely adopted as a performance metric for variable-size data requests, we show by analysis that the VP scheme can indeed achieve the optimal performance in terms of stretch. Then, we extend the VP scheme and present two adaptive value-based prefetch (AVP) schemes, which can achieve a balance between performance and power based on different user requirements. Extensive simulations are provided and used to justify the analysis. Compared to previous schemes, the proposed schemes can reduce the energy consumption and improve the system performance in terms of stretch under various scenarios.

The rest of the paper is organized as follows. Section II describes the system model and the performance metrics. In Section III, we present the VP and AVP schemes. Section IV evaluates the performance of the VP and AVP schemes. Related work is provided in Section V. Section VI concludes the paper.

II. Preliminaries

A. System Model

We use a pull-based broadcasting model, which consists of a single server and a number of clients. At the server side, there is a database of n data items: $d_1, d_2, \ldots, d_n$. The server is responsible for maintaining the database and serving the requests of the mobile clients. At the client side, caches are used to save frequently accessed data. When a client needs to access a data item that cannot be found locally, it sends out a query to the server through the uplink channel. On receiving the request, the

Manuscript received August 21, 2002; revised January 4, 2003, May 2, 2003; accepted May 26, 2003. The editor coordinating the review of this paper and approving it for publication is S. Sarkar. This work was supported in part by the National Science Foundation under Grant CAREER CCR-0092770 and Grant ITR-0219711.

The authors are with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802 USA (e-mail: yin@cse.psu.edu; gcao@cse.psu.edu).

Digital Object Identifier 10.1109/TWC.2004.833430

server sends the reply through the common broadcast channel. Similar to previous work [5], [7], [19], [20], data can only be updated by the server.

B. Cache Invalidation Model

Cached data may become invalid due to server update. To ensure cache consistency, the server broadcasts invalidation reports (IRs) every L seconds. The IR consists of the current timestamp T_i and a list of tuples (d_x, t_x) such that $t_x > (T_i - w*L)$, where d_x is the data item *id*, t_x is the most recent update timestamp of d_x, and w is the invalidation broadcast window size. In other words, IR contains the update history of the past w broadcast intervals. However, any client who has been disconnected longer than w IR intervals cannot use the report, and it has to discard the cached items. Every client, if active, listens to the IRs and invalidates its cache accordingly. To answer a query, the client listens to the next IR and uses it to decide whether its cache is valid or not. Since the client has to wait for the next IR before answering a query, the average query delay is the sum of the actual query processing time and half of the IR interval. If the IR interval is long, the delay may not be able to satisfy the requirements of many clients.

In order to reduce the query latency, Cao [7] proposed to replicate the IRs m times within an IR interval. As a result, a client only needs to wait at most $(1/m)$th of the IR interval before answering a query. Hence, latency can be reduced to $(1/m)$th of the latency in the previous schemes. Since the IR contains a large amount of update history information, to save the broadcast bandwidth, after one IR, $m-1$ updated invalidation reports (UIRs) are inserted within an IR interval. Each UIR only contains the data items that have been updated after the last IR was broadcasted. In this way, the size of the UIR becomes much smaller compared to that of the IR. Although this approach can reduce the query latency when there is a cache hit, clients still need to wait for the data to be delivered if there is a cache miss. To improve the cache hit ratio, the UIR approach also actively prefetch data that are available in the broadcast channel. All the cached data are marked as prefetchable and, whenever a data item is broadcasted, it will be prefetched if the cached copy has expired. In this paper, the UIR approach is used for cache management.

C. Performance Metrics

One widely used performance metric is the response time, i.e., the time between sending a request and receiving the reply. It is a suitable metric for homogeneous settings where different data requests have the same "size." However, the data requirements of users and applications are inherently diverse, and then to encapsulate all responses into a single-size broadcast would be unreasonably wasteful. Therefore, unlike some previous work [6], [12], we do not assume that the data items have the same size. When data requests are heterogeneous, response time alone is not a fair measure given that the individual requests significantly differ from one another in their service time, which is defined as the time to complete the request if it was the only job in the system. In this paper, we adopt an alternate performance measure, namely the stretch [1] of a request, defined to be *the ratio of the response time of a request to its service time*[1]. The rationale behind this choice is based on our intuition; i.e., clients with larger jobs should be expected to be in the system longer than those with smaller requests. The drawback of minimizing response time for heterogeneous workloads is that it tends to improve the system performance of large jobs (since they contribute the most to the response time). Minimizing stretch, on the other hand, is more fair to all job sizes.

III. AVP Scheme

The proposed AVP scheme consists of two parts. The first part is the VP scheme, which identifies valuable data items for prefetching. The second part is the AVP scheme, which determines how many data items should be prefetched.

A. VP Scheme

In this subsection, we present a value-based function which allows us to gauge the worth of a data item when making a prefetch decision. The following notations are used in the presentation:

- n number of data items in the database;
- $\overline{u_i}$ mean update arrival rate for data item i;
- $\overline{a_i}$ mean access arrival rate for data item i;
- x_i ratio of update rate to access rate for data item i, i.e., $x_i = \overline{u_i}/\overline{a_i}$;
- p_{a_i} access probability of data item i, $p_{a_i} = \overline{a_i}/\sum_{k=1}^{n} \overline{a_k}$;
- p_{u_i} probability of invalidating cached data item i before next access;
- l_i access latency for data item i;
- f_i delay of retrieving data item i from the server;
- s_i size of data item i;
- v cache validation delay;
- D set of all the data items in the database;
- C_k set of cached data items after the kth access;
- U_k set of data items updated between the kth and the $(k+1)$th access;
- P_k set of data items prefetched after the kth access.

The value function is used to identify the data to be prefetched. Intuitively, the ideal data item for prefetching should have a high access probability, a low update rate, a small data size, and a high retrieval delay. Equation (1) incorporates these factors to calculate the value of a data item i.

$$\text{value}(i) = \frac{p_{ai}}{s_i}(f_i - v - p_{u_i} \cdot f_i). \tag{1}$$

This value function can be further explained by the data access cost model, shown in Fig. 1. If item i is not in the cache, in terms of the stretch value, it takes f_i/s_i to fetch item i into the cache. In other words, if i is prefetched to the cache, the access cost can be reduced by f_i/s_i. However, it also takes $((v + P_{u_i} f_i))/s_i$ to validate the cached item i, and update it if necessary. Thus, prefetching the data can reduce the cost by $((f_i - v - p_{u_i} \cdot f_i))/s_i$ for each access. Since the access probability is p_{a_i}, the value of prefetching item i is $p_{a_i}/s_i(f_i - v - p_{u_i} \cdot f_i)$.

[1]Note that in broadcast systems, the service time for a request is the requested data size divided by the bandwidth. For simplicity, in this paper, we remove the constant bandwidth factor and use the data size to represent the service time.

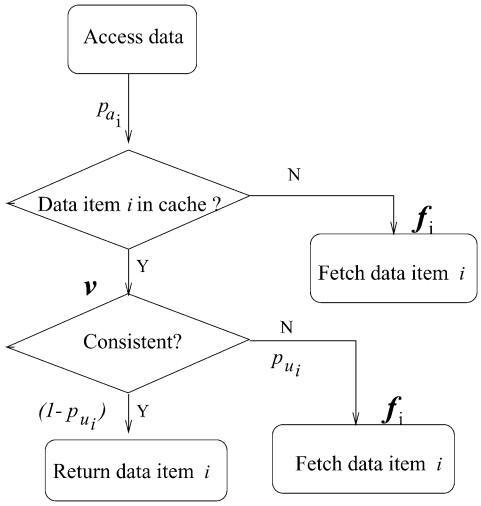

Fig. 1. Data access cost model.

The VP scheme decides which data item should be prefetched based on the value function. The VP scheme is defined as follows. Suppose a client can prefetch N_p data items, the VP scheme prefetches the N_p items which have the highest value based on the value function. Note that VP is not responsible for determining how many items (N_p) should be prefetched. N_p is determined by the adaptive scheme, which will be discussed in Section III-D.

B. Analysis of the Value Function

In this subsection, we prove that the value-based prefetch scheme can minimize the access cost given that the number of prefetches is limited. We assume that the arrivals of data update and data access follow Poisson distribution. Then, the interarrival time (t_i^a) of data access for item i, and the interarrival time (t_i^u) of the update follow exponential distributions with means of $\overline{a}_i$ and $\overline{u}_i$. The update event (U_i) for a data item i during the period from the current time to the arrival time of the next query can be probabilistically written as

$$\begin{aligned} p_{u_i} &= \Pr(U) = \Pr\left(t_i^u < t_i^a\right) \\ &= \int_{t_i^a=0}^{\infty} \int_{t_i^u=0}^{t_i^a} f\left(t_i^a\right) g\left(t_i^u\right) d_{t_i^u} d_{t_i^a} \\ &= \frac{\overline{u_i}}{\overline{u_i} + \overline{a}_i}. \end{aligned} \tag{2}$$

Therefore, the value function can be rewritten as

$$\begin{aligned} \text{value}(i) &= \frac{p_{a_i}}{s_i}\left(f_i - v - \frac{\overline{u}_i}{\overline{a_i} + \overline{u_i}} \cdot f_i\right) \\ &= \frac{p_{a_i}}{s_i}\left(f_i \cdot \frac{\overline{a_i}}{\overline{a_i} + \overline{u_i}} - v\right) \\ &= \frac{p_{a_i}}{s_i}\left(\frac{f_i}{1+x_i} - v\right). \end{aligned} \tag{3}$$

We evaluate the access cost by calculating the stretch of the kth access, which can be defined as

$$S_k = \sum_{1 \le i \le n} p_{a_i} \cdot \frac{l_i}{s_i}. \tag{4}$$

Equation (4) can be rewritten considering cache hits and cache misses

$$S_k = \sum_{i \in C_k} \frac{p_{a_i} \cdot l_i}{s_i} + \sum_{i \in (D - C_k)} \frac{p_{a_i} \cdot l_i}{s_i}. \tag{5}$$

In case of a cache hit, there are two cases: cache hit with an up-to-date copy and cache hit with an obsolete copy. Thus, (5) can be rewritten as

$$\begin{aligned} S_k = \sum_{i \in C_k} \frac{p_{a_i} \cdot l_i}{s_i} \cdot p_{u_i} + \sum_{i \in C_k} \frac{p_{a_i} \cdot l_i}{s_i}(1 - p_{u_i}) \\ + \sum_{i \in (D - C_k)} \frac{p_{a_i} \cdot l_i}{s_i}. \end{aligned} \tag{6}$$

The access latency l_i is equal to f_i when there is a cache miss. When there is a cache hit, $l_i = v$ when the cache hit is an up-to-date copy, and $l_i = v + f_i$ when the cache hit is an obsolete copy. Combining (2) and (6), we get

$$\begin{aligned} S_k &= \sum_{i \in C_k} \left(\frac{p_{a_i} \cdot (v + f_i)}{s_i} \cdot \frac{\overline{u_i}}{\overline{u_i} + \overline{a_i}}\right) \\ &\quad + \sum_{i \in C_k} \left(\frac{p_{a_i} \cdot v}{s_i}\left(1 - \left(\frac{\overline{u_i}}{\overline{u_i} + \overline{a_i}}\right)\right)\right) \\ &\quad + \sum_{i \in (D - C_k)} \frac{p_{a_i} \cdot f_i}{s_i} \\ &= \sum_{i \in C_k} \left(\frac{p_{a_i} \cdot \left(v + \frac{f_i \cdot \overline{u}_i}{\overline{u_i} + \overline{a_i}}\right)}{s_i}\right) + \sum_{i \in (D - C_k)} \frac{p_{a_i} \cdot f_i}{s_i}. \end{aligned} \tag{7}$$

Theorem 1: Prefetching items with high value can achieve lower stretch than any other prefetch schemes given that the number of prefetches is limited.

Proof: Let U_k represent the data that have been modified since the last invalidation report and P_k represent the data that are prefetched after the kth access. Then, $C_{k+1} = C_k - U_k + P_k$

$$\begin{aligned} S_{k+1} &= \sum_{i \in C_{k+1}} \left(\frac{p_{a_i} \cdot \left(v + \frac{f_i \cdot \overline{u_i}}{\overline{u_i} + \overline{a_i}}\right)}{s_i}\right) + \sum_{i \in (D - C_{k+1})} \frac{p_{a_i} \cdot f_i}{s_i} \\ &= S_k + \sum_{i \in U_k} \left(\frac{p_{a_i}}{s_i}\left(\frac{f_i}{1+x_i} - v\right)\right) \\ &\quad - \sum_{i \in P_k} \left(\frac{p_{a_i}}{s_i}\left(\frac{f_i}{1+x_i} - v\right)\right). \end{aligned} \tag{8}$$

In (8), $\sum_{i \in U_k} \left(p_{a_i}/s_i((f_i/1+x_i) - v)\right)$ cannot be reduced because it is caused by the server update. Equation (8) implies that the cost can be reduced by prefetching those items that can maximize $\sum_{i \in P_k} \left(p_{a_i}/s_i((f_i/1+x_i) - v)\right)$. This is exactly what the proposed scheme tries to do: prefetching those items with the maximum sum of value. For any other prefetch scheme that prefetch the same number of data items, let the set of prefetched data items be P'_k and the cost of the $(k+1)$th access be S'_{k+1}, according to the proposed scheme

$$\sum_{i \in P_k} \left(\frac{p_{a_i}}{s_i}\left(\frac{f_i}{1+x_i} - v\right)\right) > \sum_{i \in P'_k} \left(\frac{p_{a_i}}{s_i}\left(\frac{f_i}{1+x_i} - v\right)\right).$$

Thus

$$S_{k+1} < S'_{k+1}.$$

This proves that the proposed scheme can minimize the cost among all prefetch schemes that prefetch the same number of data items. ∎

The proposed value-based function is calculated in terms of stretch since the performance metric is stretch. Actually, this value-based function can be easily extended for other performance metrics. For example, if the performance metric is query delay, the value function will be changed to $\text{value}(i) = p_{a_i}(f_i - v - p_{u_i} \cdot f_i)$. Similar techniques can be used to prove that this value function can minimize the query delay.

C. Parameter Estimation

To implement the AVP scheme, we need to estimate parameters f_i, $\overline{a_i}$, and $\overline{u_i}$ since they are not constant. To estimate f_i, we adopt the exponential aging method, which has been used in TCP to estimate the round-trip delay. It combines both the historical data and the currently measured data to estimate the parameters. Whenever a data item i is fetched from the server, f_i is recalculated as the following:

$$f_i = \alpha \cdot f_i^{\text{new}} + (1 - \alpha) \cdot f_i$$

where f_i^{new} is the currently measured data retrieval delay and f_i on the right side of the formula is the calculated f_i before the last retrieval of item i.

Although this formula can be used to estimate f_i, it is not suitable for estimating $\overline{a_i}$ and $\overline{u_i}$ since the access rate and the update rate should be "aged" in the absence of data access. That is, the values of $\overline{a_i}(\overline{u_i})$ should be decreased even if there is no data access (update) over some period of time. We apply techniques, which have been used in [17], to estimate $\overline{a_i}$ and $\overline{u_i}$. This method uses K most recent samples to estimate $\overline{a_i}$ and $\overline{u_i}$ as follows:

$$\overline{a_i} = \frac{K}{T - T_{\overline{a_i}}(K)}$$
$$\overline{u_i} = \frac{K}{T - T_{\overline{u_i}}(K)}$$

where T is the current time, $T_{\overline{a_i}}(K)$ and $T_{\overline{u_i}}(K)$ are the time of the Kth most recent access and update. If there are less than K samples, all the available samples are used to estimate the value. As shown in [17], the best performance can be achieved with small value of K (2 or 3). Thus, the spatial overhead to store these samples is very small. The estimation of $\overline{a_i}$ should be done at the client side since different clients may have different access pattern. However, it is impossible for clients to estimate $\overline{u_i}$ since the data updates occur at the server side. Therefore, the server estimates $\overline{u_i}$ of each data item and piggybacks it to clients when the data item is broadcast.

D. AVP Scheme

Due to limitations of battery technology, the energy available for a wireless device is limited and must be used prudently. If the prefetched data item is not accessed or is invalidated before it is accessed, the energy spent on downloading this item will be wasted. To avoid wasting power, it is important that clients only download the data with high value, but such a strict policy may adversely affect the performance of the system and increase the query delay.

Each client may have different available resources and performance requirements, and these resources such as power may change over time. For example, suppose the battery of a laptop lasts for three hours. If the user is able to recharge the battery within three hours, power consumption may not be an issue, and the user may be more concerned about the performance aspects such as the query latency. However, if the user cannot recharge the battery within 3 h and wants to use it a little bit longer, power consumption becomes a serious concern. Since N_p controls the number of data to be prefetched and then affects the tradeoff between performance and power, we propose adaptive schemes to adjust N_p to satisfy different client requirements.

1) Value of N_p: When N_p reduces to 0, there will be no prefetch. As N_p increases, the number of prefetches increases and the power consumption also increases. Since the maximum number of data items to be prefetched is limited by the cache size, N_p is also limited by this number. Intuitively, the query delay decreases as the number of prefetches increases. However, this is not always true considering the overhead to maintain cache consistency. In our cache invalidation model, a client needs to wait for the next IR to verify the cache consistency. This waiting time may increase the query delay compared to the approaches without prefetch. The cost has been quantified in (1), where v is the cache invalidation delay. Due to the cost of v, the value of a data item may be negative. If $\text{value}(i)$ is negative, prefetching item i not only wastes power but also increases the average stretch. Therefore, N_p should be bounded by $N_p^{\max}$, which is limited by the client cache size and the data value; i.e., a client will not prefetch items with negative values.

The tradeoff between performance and power can be achieved by adjusting N_p. In Sections III-D2 and 3, we present two adaptive schemes: the AVP_T (T for time) scheme which dynamically adjusts N_p to reach a target battery life time, and the AVP_P (P for power) scheme which dynamically adjusts N_p based on the remaining power level.

2) AVP_T: Adapting N_p to Reach a Target Battery Life: A commuter normally knows the amount of battery energy and the length of the trip between home and office. With these resource limitations, the commuter wants to achieve the lowest query delay. This is equivalent to the problem of adapting N_p to reach a target battery life and minimize the average stretch. Suppose a battery with E joule lasts T_1 seconds when $N_p = N_p^{\max}$, and T_2 seconds when $N_p = 0$. It is possible to adjust N_p to reach a target battery life time $T \in [T_1, T_2]$. In AVP_T, the client monitors the power consumed in the past. If it consumes too much power in the past and cannot last T seconds, N_p is reduced. On the other hand, it increases N_p when it found that it has too much power left. Certainly, N_p is bounded by $N_p^{\max}$. Fig. 2 shows the details of the AVP_T scheme.

3) AVP_P: Adapting N_p Based on the Power Level: When the energy level is high, power consumption is not a major concern and then trading off energy for performance may be a good option if the user can recharge the battery soon. On the other

Notations:

- E: the amount of initial energy.
- T: the target battery lasting time.
- P_{avg}: the estimated average power consumption.
- T_{old}, E_{old}: the time to obtain the last P_{avg}, and the energy level at that time.
- T_{new}, E_{new}: the current time and the current energy level.
- N_p^{max}: the maximum N_p.

The AVP_T scheme is as follows:

$P_{avg} = \alpha \cdot P_{avg}$
$\quad + (1-\alpha) \cdot (E_{old} - E_{new})/(T_{new} - T_{old});$
if ($|P_{avg} - E/T|/(E/T) > 0.05$) **then**
$\quad N_p = int(max(1, N_p) \cdot$
$\quad\quad (1 - 2 \cdot (P_{avg} - E/T)/(E/T)));$
else {
$\quad$ **if** ($P_{avg} - E/T > 0$) **then** $N_p++;$
$\quad$ **else** $N_p--;$
}
if ($N_p < 0$) **then** $N_p = 0;$
else if ($N_p > N_p^{max}$) **then** $N_p = N_p^{max};$

Fig. 2. AVP_T scheme.

hand, when the energy level is low, the system should be power-aware to prolong the system running time to reach the next battery recharge time. Based on this intuition, the AVP_P scheme dynamically changes N_p based on the power level. Let a_k be the percentage of energy left in the client. When a_k drops to a threshold, the number of prefetches should be reduced to some percentage, say $f(a_k)$, of the original value. Some simple discrete function can be as follows:

$$f(a_k) = \begin{cases} 100\% & 0.5 < a_k \le 1.0 \\ 70\% & 0.3 < a_k \le 0.5 \\ 50\% & 2 < a_k \le 0.3 \\ 30\% & 0.1 < a_k \le 0.2 \\ 10\% & a_k \le 0.1. \end{cases} \tag{9}$$

At regular interval, the client re-evaluates the energy level a_k. If a_k drops to a threshold value, $N_p = N_p \cdot f(a_k)$. The client only marks the first N_p items in the cache, which have the maximum value, as prefetchable. In this way, the number of prefetches can be reduced to prolong the system running time. Since this is a discrete function, N_p does not need to be frequently updated and the computation overhead is low.

Note that a simple policy which is neither too aggressive nor conservative might result in similar average stretch and lifetime as the VAP_P scheme if the battery runs out before recharge. However, if the user recharges the battery frequently, this simple policy may not be a good option since it saves power at the cost of delay, but power consumption is not a concern at this time. In contrast, our adaptive scheme tries to tradeoff power for performance at the beginning, and become power-aware when the client cannot recharge in time.

IV. Performance Evaluation

To evaluate the performance of the proposed methodology, we developed a simulation model similar to that employed in [7], [19]. We compare the VP approach and the AVP approaches with the prefetch scheme used in UIR [7] under various workload and system settings.

TABLE I
Simulation Parameters

Number of data items (n)	2000
Number of clients	100
s_{min}	0.5k
s_{max}	10k
Mean update time (T_{update})	100 seconds
Broadcast interval (L)	20 seconds
Broadcast window (w)	10 interval
Broadcast bandwidth	144Kbps
Uplink bandwidth	14.4Kbps
Relative cache size	20% of total database size
Mean query generate time (T_{query})	100 seconds
Zipf distribution parameter θ	0.9

A. Simulation Model and Parameters

1) Client Model: Each client generates a single stream of read-only queries. The mean query generate time for each client is T_{query}. The access pattern follows a Zipf-like distribution [4] which has been frequently used to model nonuniform distribution. In the Zipf-like distribution, the access probability of the ith $(1 \le i \le n)$ data item is represented as follows.

$$p_{a_i} = \frac{1}{i^\theta \sum_{k=1}^{n} \frac{1}{k^\theta}}$$

where $0 \le \theta \le 1$. When $\theta = 1$, it is the strict Zipf distribution. When $\theta = 0$, it becomes the uniform distribution.

2) Server Model: The server broadcasts IRs and UIRs periodically to the clients. The IR/UIR broadcast messages are assigned the highest priority whereas the rest of the messages are of equal priority. This ensures that IRs and UIRs can be broadcast regularly over the wireless channels with the broadcast interval specified by L. The IR interval is set at 20 s and the UIR is replicated four times within each interval. The other messages are served on a first-come-first-serve basis. Should the server be in the middle of a transmission when an IR or UIR has to be sent, the IR/UIR broadcast is deferred till the end of the current packet transmission. There are n data items at the server side. The size (s_i) of item i grows linearly from $s_{\min}$ to $s_{\max}$ as i increases. As a result of joint distribution of access pattern and data size, the item with smaller size will be accessed more frequently than bigger ones. This has been commonly observed in traces [4], [11].

The server generates a single stream of updates separated by an exponentially distributed update interval time. The whole database is divided into two subsets: the frequently-updated subset which contains the first 20% of the data items (*id* from 0 to $n * 20\% - 1$) in the database, and the rarely-updated subset which contains the rest of the data. Eighty percent of the updates are randomly distributed inside the frequently-updated subset. The rest of the updates are randomly distributed in the rarely-updated subset. It is assumed that the bandwidth is fully utilized for broadcasting IRs and UIRs and serving client requests. The server processing time is considered to be negligible. Most of the system parameters and their default values are listed in Table I.

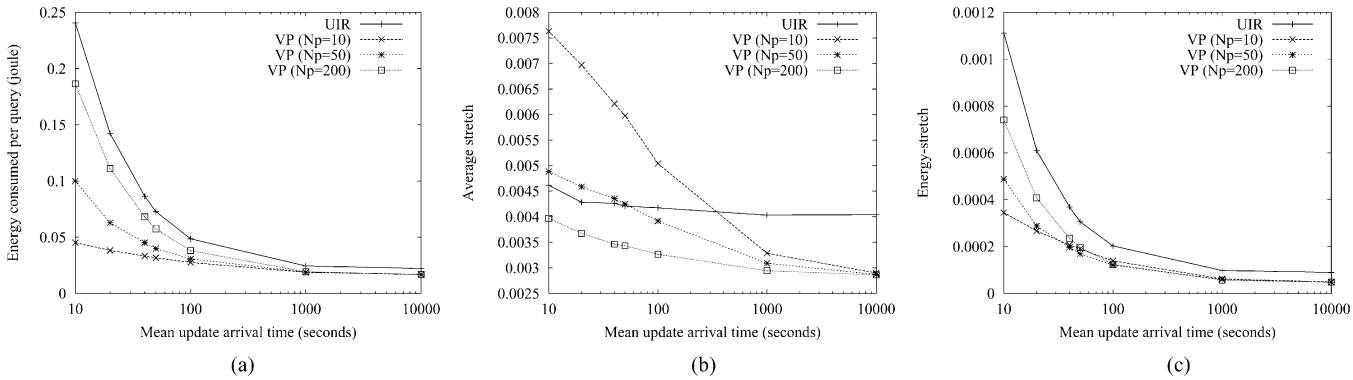

Fig. 3. Performance under varying update arrival time. (a) Energy consumption. (b) Average stretch. (c) Energy-stretch.

The energy consumption model adopted in this paper is similar to the model used in [6], where the transmission power dissipation is 0.5 W and the receiving power dissipation is 0.2 W for the wireless network interface. In order to focus on the energy consumption of data transmission and receiving, we do not consider the energy consumed by other components. In order to better understand the overall performance of the system, we introduce a new metric called energy-stretch, which is defined as

$$\text{energy-stretch} = \text{stretch} * \text{energy-consumed}.$$

The energy-stretch value gives a better indication of the system performance because it considers two important parameters: energy consumption and stretch. The simulation results consist of two parts: one to evaluate the performance of the VP scheme and the other to evaluate the performance of the AVP schemes.

B. Simulation Results: VP

1) Effects of the Mean Update Arrival Time: Fig. 3 shows the effects of the mean update arrival time on the energy consumption, the average stretch, and the energy-stretch. The figure shows various VP approaches with different N_p value. As we know, the number of prefetches increases as N_p increases. As a result, the energy consumption increases and the average stretch drops. This has been verified by the simulation results in Fig. 3(a) and (b). The $N_p = 10$ approach has the lowest power consumption, but it has the highest stretch. Fig. 3(c) shows their energy-stretch value. Although $VP(N_p = 10)$ has the lowest energy-stretch when $T_{\text{update}} = 10s$, it has higher energy-stretch than $VP(N_p = 50)$ when $T_{\text{update}} > 40$ s, and it has a very high average stretch. Generally speaking, $VP(N_p = 50)$ has low stretch while consuming similar power as $VP(N_p = 10)$ when $T_{\text{update}} > 40$ s. We use $N_p = 50$ in the following experiments when VP scheme is used unless specified otherwise.

Compared to the UIR approach, the VP approach saves a large amount of power since the UIR approach aggressively prefetches data to fill up the client cache. From Fig. 3(a), we can also see that four approaches have similar power consumption when $T_{\text{update}} = 10\,000$ s, because only limited number of data items in the cache are invalidated when T_{update} is very large and, hence, the clients only need to prefetch a few data items. As the mean update arrival time drops, many data items are updated, and the number of prefetches increases. As a result, the power consumption increases. However, the increasing trend is different. The UIR approach has the largest power consumption increase whereas $VP(N_p = 10)$ has the lowest. This is due to the fact that $VP(N_p = 10)$ limits the number of prefetches to 10.

From Fig. 3(b), we can see that the UIR approach does not follow the same trend as other VP approaches. When the mean update arrival time is low, UIR has lower stretch than $VP(N_p = 10)$ and $VP(N_p = 50)$ since UIR allows aggressive prefetching, while VP only prefetches N_P items with high access rate and low update rate (based on the value function). When the update arrival time is low, data items are frequently invalidated. Due to aggressive prefetch, UIR achieves better performance at the cost of high energy consumption as shown in Fig. 3(a). As the mean update time increases, various VP approaches outperform UIR, since VP only prefetches the data with high value. Comparing $VP(N_p = 200)$ with UIR, we can see the advantage of VP. Although $VP(N_p = 200)$ only prefetch 200 data items, which is much less that of the UIR approach, it still outperforms UIR in terms of energy, stretch, and energy-stretch.

2) Comparisons With Simple Prefetch Schemes: A prefetch scheme may utilize the fact that the Zipf-like distribution favors items with small *id*. One simple and reasonable prefetch scheme is to prefetch those k hottest (frequently accessed) data items. In this subsection, we evaluate this simple prefetch scheme with VP. In Fig. 4, SIMP-k represents the prefetch scheme which prefetches k hottest data items. As expected, smaller k results in lower energy consumption and higher average stretch. Overall, the VP scheme has the best performance considering both energy consumption and average stretch. Although the simple prefetch scheme considers the access frequency factor, it does not consider other factors such as data size, cache invalidation cost. Thus, it underperforms the VP scheme.

From Fig. 4, we can also see that the energy consumption of the UIR approach and the VP approach both increase with the mean query generate time. This can be explained as follows. When the mean query generate time increases, if other parameters do not change, the update rate within the time of two query interval increases. In other words, increasing the mean query generate time without changing the update arrival time should have the same effect of reducing the update arrival time without changing the mean query generate time. As explained

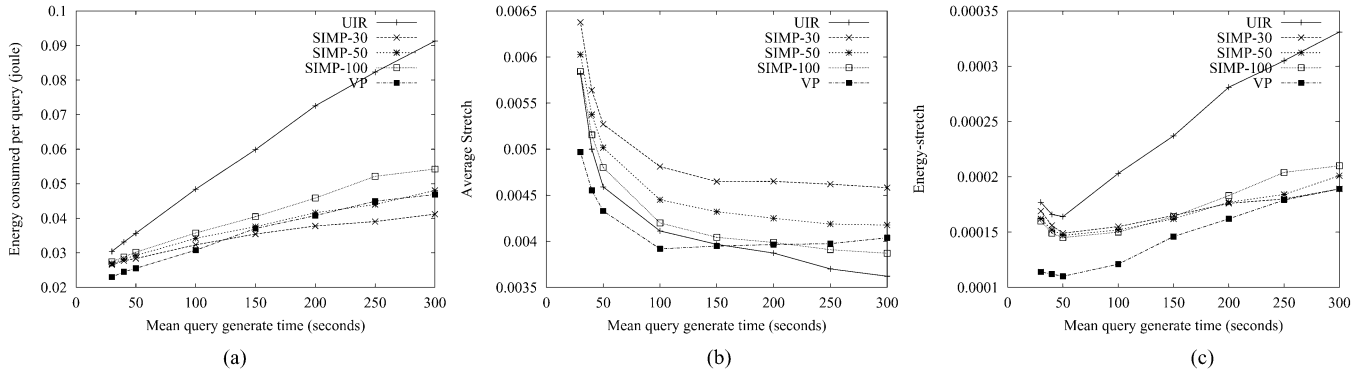

Fig. 4. Comparisons with simple prefetch schemes. (a) Energy consumption. (b) Average stretch. (c) Energy-stretch.

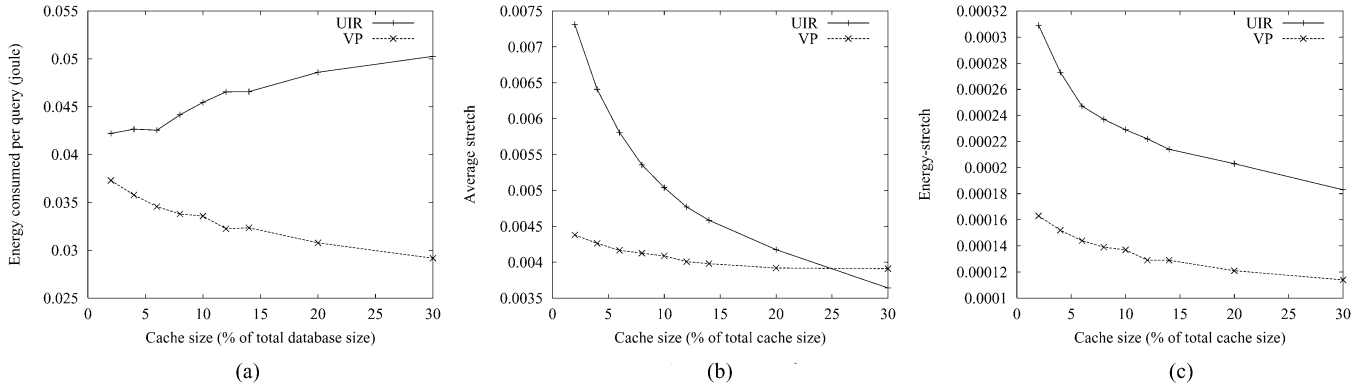

Fig. 5. Performance as a function of the cache size. (a) Energy consumption. (b) Average stretch. (c) Energy-stretch.

in Section IV-B1, the energy consumption increases if the update arrival time drops, and then the energy consumption increases as the mean query generate time increases.

3) Effects of the Cache Size: Fig. 5 shows the effects of the cache size on the average stretch and the energy consumption of the UIR approach and the VP scheme. As can be seen, UIR and VP have similar energy consumption when the cache size is small since the number of items to be prefetched is limited by the cache size in the UIR approach. As the cache size increases, the number of prefetches in the UIR approach increases and its energy consumption also increases. This is different from VP. Since N_p is fixed, the data to be prefetched is also fixed in VP. Moreover, when the cache size increases, the cache hit ratio increases, and then more queries can be served from the cache without sending requests and receiving data. Thus, the amount of energy consumed per query actually drops. This explains why the energy consumption per query in VP drops as the cache size increases. As shown in Fig. 5(b), due to aggressive prefetching, the average stretch of the UIR approach drops below that of the VP approach when the cache size increases to 25%. When considering both power consumption and stretch, VP always outperforms UIR as shown in Fig. 5(c).

4) Effects of the Zipf Parameter θ: The Zipf parameter θ determines the "skewness" of the access distribution. Fig. 6 shows the effects of the access pattern on the system performance. When $\theta = 0$, the access pattern follows uniform distribution. That is, all the data items have the same access probability. At this time, prefetch is less effective and the energy-stretch value is high. As θ increases, more data accesses are focused on items with small *id* and the energy-stretch drops. Fig. 6 demonstrates that our approach always outperforms the UIR approach. Even when the access pattern follows uniform distribution ($\theta = 0$), our approach still performs better. It also shows that our approach works for a wide spectrum of access patterns.

5) Effects of the Access Pattern Variation: In practice, the client access pattern may change. To model such environments, we make the following modifications. The data access distribution is shifted one item every δ IR intervals. More specifically, suppose item i is the data to be accessed according to the Zipf distribution. An offset Δ is added to i after each δ IR intervals so that $i + \Delta$ is the actual data item that will be accessed. Δ is initialized to 0, and increased by one every δ IR intervals. The smaller δ is, the faster the access pattern changes. To make it reasonable, we also apply the same offset to the update so that the items with high update rate still have high access rate.

Fig. 7 shows the results when the access pattern changes. If the access pattern changes rapidly, the average energy consumption is very high because clients need to spend more energy to prefetch and request data from the server. It can be seen that the VP approach always outperforms the UIR approach.

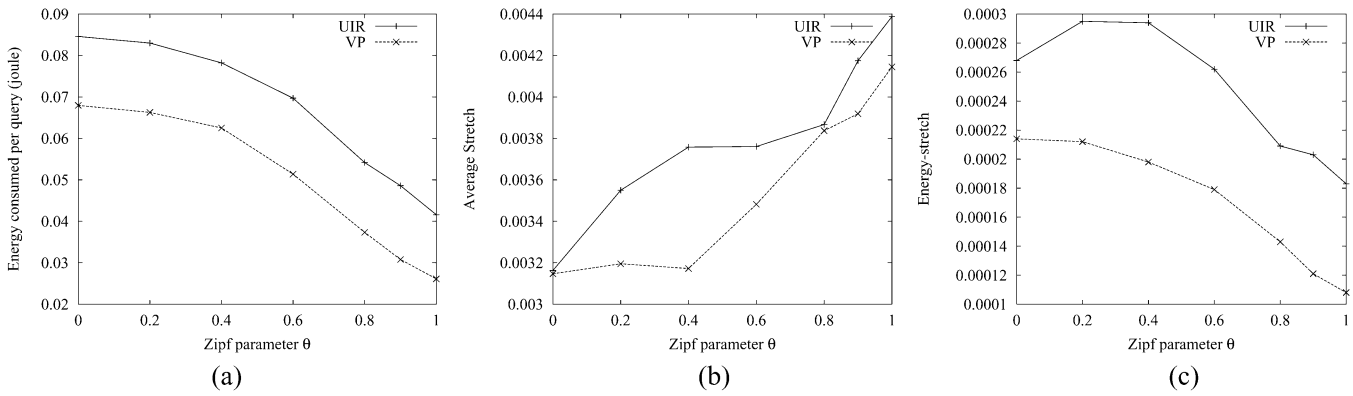

Fig. 6. Effects of the Zipf parameter θ. (a) Energy consumption. (b) Average stretch. (c) Energy-stretch.

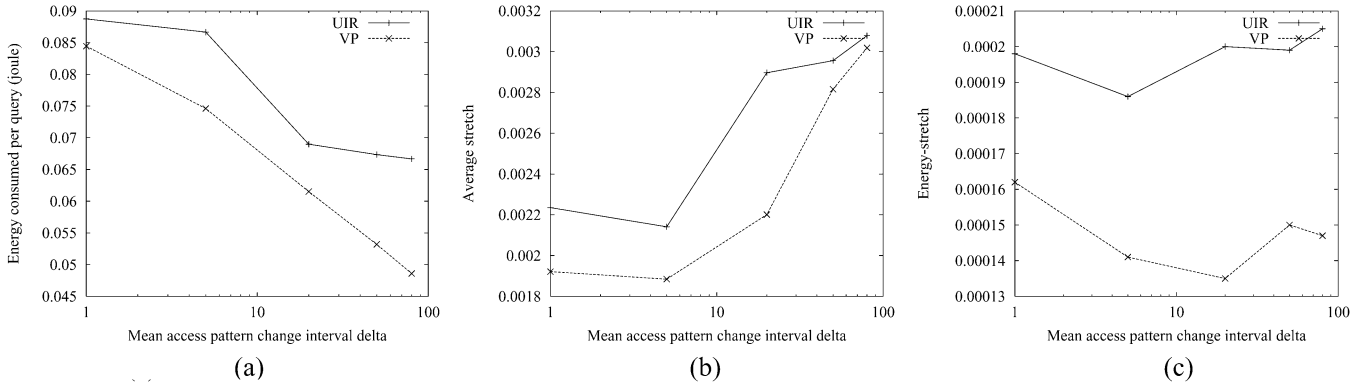

Fig. 7. Effects of the access pattern variation. (a) Energy consumption. (b) Average stretch. (c) Energy-stretch.

C. Simulation Results: Adaptation

1) Adapting N_p to Reach a Target Battery Life: Fig. 8 evaluates the performance of the AVP_T scheme after the clients are assigned 5000 J initially. By simulation, we found that the battery life is 190 ks when $VP(N_p = 0)$ is used. Although the cache size is 20% of the database size, considering the data distribution pattern and the data size difference, the client cache can hold as many as 840 data items. We set $N_p^{\max}$ to be 840, but we should know that clients will not prefetch data items with negative value. Based on simulation results, the battery life is 110 ks when $VP(N_p = 840)$ is used. To evaluate the performance of the AVP_T scheme, we consider the following target battery life (T).

- $T = 100$ ks. In this case, denoted as AVP_T(100, 0), a very short battery life time is expected, and N_p has an initial value of 0.
- $T = 200$ ks. In this case, denoted as AVP_T(200, 840), the battery life is too long to be reachable, and N_p has an initial value of $N_p^{\max}$.
- $T = 140$ ks. There are two cases: N_p has an initial value of 0, denoted as AVP_T(140, 0); N_p has an initial value of of $N_p^{\max}$, denoted as AVP_T(140, 840).

As shown in the Fig. 8(a), the adaptive schemes can quickly adjust N_p to reach the target battery life if it is possible. For $AVP\_T(200, 840)$ and $AVP\_T(100, 0)$, because the target battery life is not reachable, these two schemes can only adjust N_p to reach the target as close as possible. The small difference of the battery life time between $AVP\_T(200, 840)$ and $VP(N_p = 0)$ is caused by the aggressive initial prefetch of the $AVP\_T(200, 840)$ scheme. For the two adaptive schemes with moderate target life, $AVP\_T(140, 0)$ and $AVP\_T(140, 840)$, both reaches the target battery life time. The difference between the actual battery life and the target battery life is less than 3%. Fig. 8(c) shows the cumulative average stretch from the time to collect results. Fig. 8(d) shows the average stretch of every 2000 s, which is the time for N_p to be re-evaluated. As can be seen from Fig. 8(c), the difference between the final average stretch of $AVP\_T(140, 0)$ and $AVP\_T(140, 840)$ is less than 4% Fig. 8(d) also shows that $AVP\_T(140, 0)$ and $AVP\_T(140, 840)$ have similar stretch after the first 60 ks.

2) Adaptation of N_p Based on the Power Level: In AVP_P, when the energy level becomes less than a threshold, the prefetch rate is reduced. When the energy level becomes critically low, the prefetch rate is further reduced. By reducing the prefetch rate, energy can be saved and the system can last longer. Fig. 9 compares the energy level and the system performance of three schemes: UIR, $VP(Np = 400)$, and AVP_P. In AVP_P, the initial value of N_p is also 400. As can be seen, the UIR approach has higher prefetch rate and consumes more energy, whereas our approaches have lower energy consumption. Due to energy saving, our adaptive approach can last more than 20% longer than the UIR approach.

$VP(Np = 400)$ and AVP_P have the same amount of energy consumption when there are abundant energy. As the energy level falls below 50%, AVP_P reduces the prefetch rate to save energy. Fig. 9(a) shows that AVP_P uses less energy because of the reduction of N_p. As shown in the figure, without adaptation, VP continues to use more energy and eventually runs out of power sooner than AVP_P. On the other hand,

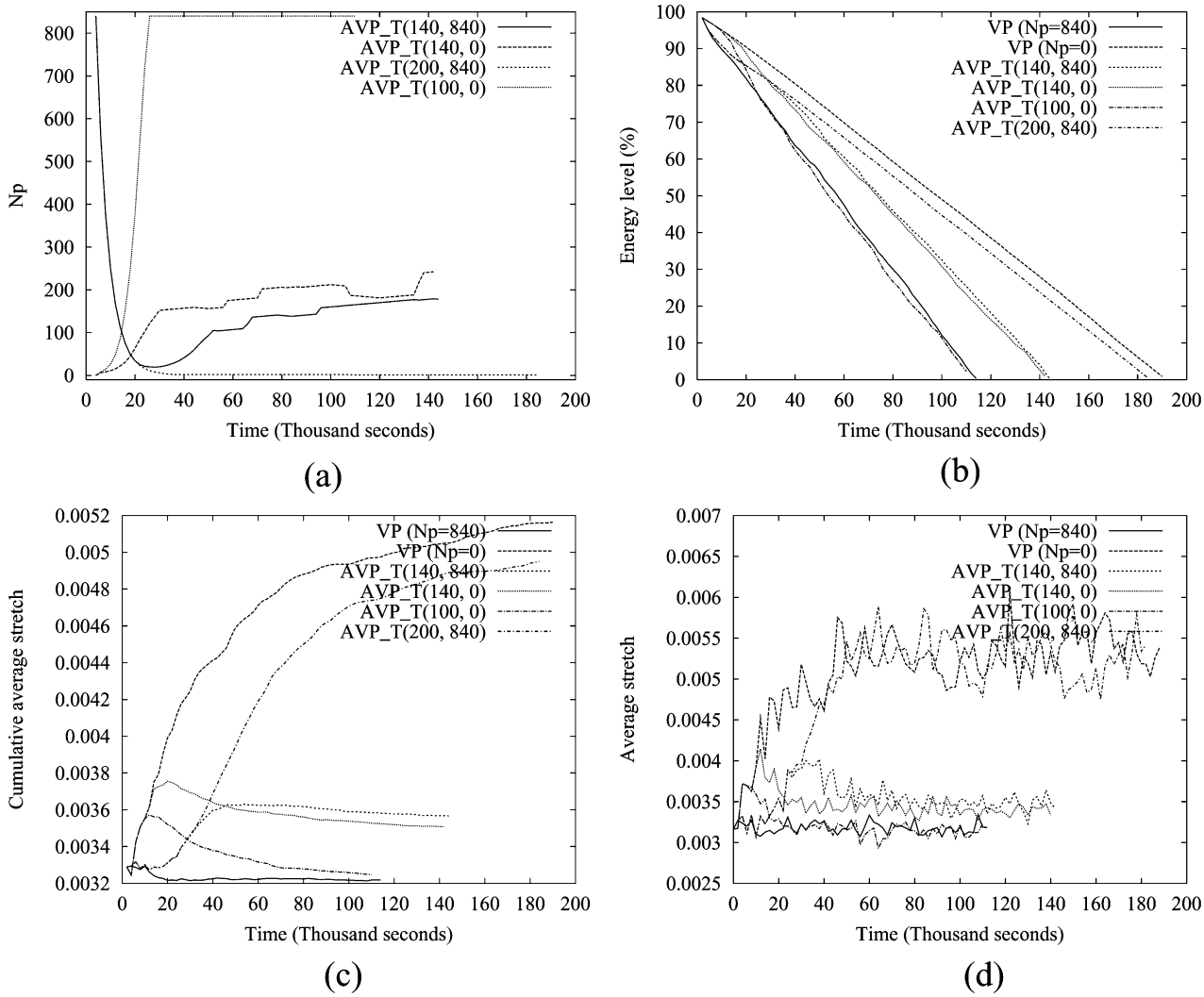

Fig. 8. AVP_T: adapting N_p to reach a target battery life. (a) Adaptation of N_p. (b) Energy level. (c) Cumulative average stretch. (d) Average stretch.

the power saving of AVP_P is at the cost of increasing the average stretch, although the increase is not that significant. As shown in Fig. 9(b), the average stretch of AVP_P is only slightly higher than that of VP. Fig. 9(c) shows that AVP_P manages to achieve low energy-stretch while increasing the battery life.

V. Related Work

Prefetching has been widely used to reduce the response time in the Web environment [8], [9], [15], [16]. Most of these techniques concentrate on estimating the probability of each file being accessed in the near future. Since these techniques are designed for the point-to-point communication environment, they are not suitable for the broadcasting environment in mobile computing systems. Recently, prefetching has been used in many cache management schemes for mobile environments [6], [10], [12] to reduce the query latency and the bandwidth consumption. In Grassi's scheme [12], a push-based model is used. The system performance is optimized using indexing techniques to periodically rebroadcast the hot data. In their prefetch scheme, each data item is assigned a calculated value $fv(i)$, which is a function of the access rate of item i. Based on this value, the client decides whether to prefetch the item or not when it appears in the broadcast channel. If there exists an item j in the client's cache such that $fv(i) > fv(j)$, item j is removed from the cache and replaced by item i. This prefetch scheme fails to address a number of issues such as data size, data update rate, and power consumption.

In [6], Cao proposed an adaptive prefetch scheme. In this scheme, clients record the number of times a cached item being accessed and prefetched, respectively. The client calculates the prefetch access ratio (PAR), which is the number of prefetches divided by the number of accesses, for each item. If PAR is less than one, prefetching the data is useful since the prefetched data may be accessed multiple times. When power consumption becomes an issue, the client marks those cache items with $PAR > \beta$ as *nonprefetch*, where $\beta > 1$ is a system tuning factor, and should be dynamically changed based on the energy consumption. However, no clear methodology as to how and when β should be changed. This scheme, like the previous one [12], does not consider varying data size and the data update rate.

Gitzenis and Bambos [10] proposed a prefetch scheme considering the quality of the wireless channel. Clients prefetch aggressively when the channel quality is good but reduce the

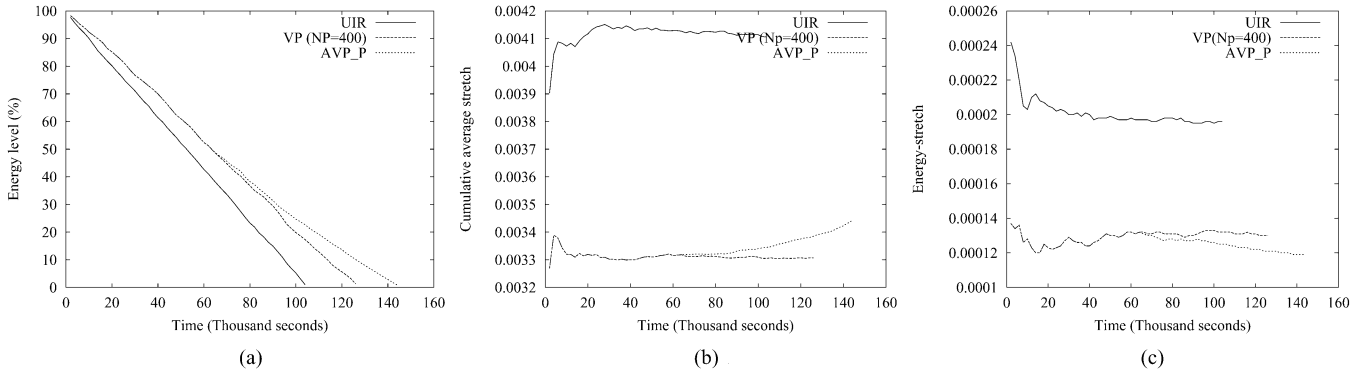

Fig. 9. The energy level and system performance as a function of time. (a) The energy level. (b) Average stretch. (c) The energy-stretch.

prefetch rate when the channel quality becomes poor. Their scheme assumes that the cost of accessing a data item is already given. Our value-base prefetch scheme actually gives a function to identify the value of each data item and prefetch those items that can minimize the overall access cost. Thus, their work complements our work.

VI. Conclusion

Prefetching is an effective technique to reduce the query latency. However, prefetching consumes power. In wireless networks where power is limited, it is essential to correctly identify the data to be prefetched in order to provide better performance and reduce the power consumption. In this paper, a VP scheme was proposed. The proposed scheme evaluates the cost of prefetching a data item by taking into account various factors such as the data size, the access rate, the update rate, and the cache validation delay. In addition to making smarter prefetch decisions, the scheme is designed to be adaptive, adjusting the prefetch rate based on the current energy level. Simulation results verified that the proposed schemes can reduce the energy consumption and improve the system performance in terms of stretch compared to the UIR approach under various system settings.

The major contribution of the paper is the VP function. Although we explored various adaptive approaches, many issues still need further investigation. For example, the channel state information [10] can be used when making prefetch decisions, i.e., N_p can be dynamically adjusted based on the channel state. If the target battery life time is not known or only known with some probability, AVP_T needs to be extended to factor into these uncertainties. If the battery recharge cycle or the user profile is known or known with a high probability, how to enhance AVP_T and AVP_P needs further investigation.

Acknowledgment

The authors would like to thank the editors and the anonymous referees whose insightful comments helped us to improve the presentation of the paper.

References

[1] S. Acharya and S. Muthukrishnan, "Scheduling on-demand broadcasts: New metrics and algorithms," in *Proc. MobiCom*, Oct. 1998, pp. 43–54.

[2] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-based data delivery using broadcast disks," *IEEE Personal Commun.*, vol. 2, pp. 50–60, Dec. 1995.

[3] D. Barbara and T. Imielinski, "Sleepers and workaholics: Caching strategies for mobile environments," *ACM SIGMOD*, pp. 1–12, 1994.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. 18th Annu. Joint Conf. IEEE Computer and Communications Soc.*, 1999, pp. 126–134.

[5] G. Cao, "On improving the performance of cache invalidation in mobile environments," *ACM/Baltzer Mobile Networks Applicat. (MONET)*, vol. 7, no. 4, pp. 291–303, Aug. 2002.

[6] ——, "Proactive power-aware cache management for mobile computing systems," *IEEE Trans. Comput.*, vol. 51, pp. 608–621, June 2002.

[7] ——, "A scalable low-latency cache invalidation strategy for mobile environments," *IEEE Trans. Knowledge Data Eng.*, vol. 15, pp. 1251–1265, Sept./Oct. 2003.

[8] K. Chinen and S. Yamaguchi, "An interactive prefetching proxy server for improvement of WWW latency," in *Proc. 7th Annu. Conf. Internet Soc.* , June 1997.

[9] C. Cuhna and C. Jaccound, "Determining WWW user's next access and its application to pre-fetching," in *Proc. 2nd IEEE Symp. Computers Communications*, 1997, pp. 6–11.

[10] S. Gitzenis and N. Bambos, "Power-controlled data prefetching/caching in wireless packet networks," in *Proc. INFOCOM*, 2002, pp. 1405–1414.

[11] S. Glassman, "A caching relay for the world wide web," *Comput. Networks ISDN Syst.*, vol. 27, no. 2, pp. 165–173, 1994.

[12] V. Grassi, "Prefetching policies for energy saving and latency reduction in a wireless broadcast data delivery system," in *Proc. 3rd ACM Int. Workshop Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, 2000, pp. 77–84.

[13] S. Hameed and N. Vaidya, "Efficient algorithms for scheduling data broadcast," *ACM/Baltzer Wireless Networks (WINET)*, pp. 183–193, May 1999.

[14] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Data on air: Organization and access," *IEEE Trans. Knowledge Data Eng.*, vol. 9, pp. 353–372, May/June 1997.

[15] Z. Jiang and L. Kleinrock, "An adaptive network prefetch scheme," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 358–368, Apr. 1998.

[16] V. Padmanabhan and J. Mogul, "Using predictive prefetching to improve world wide web latency," *Comput. Commun. Rev.*, pp. 22–36, July 1996.

[17] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy cache algorithms: Design, implementation, and performance," *IEEE Trans. Knowledge Data Eng.*, vol. 11, pp. 549–562, July/Aug. 1999.

[18] M. Stemm and R. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices," *IEICE Trans. Commun.*, vol. 80, no. 8, pp. 1125–1131, Aug. 1997.

[19] J. Xu, Q. Hu, W. Lee, and D. Lee, "An optimal cache replacement policy for wireless data dissemination under cache consistency," in *Proc. Int. Conf. Parallel Processing*, Sept. 2001, pp. 267–276.

[20] J. Yuen, E. Chan, K. Lam, and H. Leung, "Cache invalidation scheme for mobile computing systems with real-time data," in *ACM Special Interest Group on Management of Data (SIGMOD) Rec.*, vol. 29, Dec. 2000, pp. 34–39.

Liangzhong Yin (S'02) received the B.E. degree and the M.E. degree in computer science and engineering from the Southeast University, Nanjing, China, in 1996 and 1999, respectively. He is currently working toward the Ph.D. degree at The Pennsylvania State University, University Park.

From 1999 to 2000, he was an Engineer with the Global Software Group, Motorola, Nanjing, China. His research interests include wireless/ad hoc networks, and mobile computing.

Guohong Cao (S'98–A'99) received the B.S. degree from Xian Jiaotong University, Xian, China, and the M.S. degree and the Ph.D. degree in computer science from the Ohio State University, Columbus, in 1997 and 1999, respectively.

He joined the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, in 1999, where he is currently an Associate Professor. He has served on various conference program committees. His research interests include wireless networks, mobile computing, and distributed fault-tolerant computing.

Dr. Cao is an Editor for the IEEE TRANSACTIONS ON MOBILE COMPUTING and the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS. He was a recipient of the Presidential Fellowship at the Ohio State University in 1999 and a recipient of the National Science Foundation CAREER award in 2001.