A Break for Workaholics: Energy-Efficient Selective Tuning Mechanisms for Demand-Driven-Based Wireless Environment

Kian-Lee Tan
Dept. Information Systems & Computer Science
National University of Singapore
Lower Kent Ridge, Singapore 119260
email: tankl@iscs.nus.sg

Jeffrey X. Yu
Dept. Computer Science
Australian National University
Canberra, ACT 0200, Australia
email: yu@cs.anu.edu.au

Abstract

Existing work on demand-driven-based wireless environments has largely focused on energy-efficient caching strategies. While these schemes minimize the number of uplink requests (and hence conserve energy), they are still not adequate as clients must continue to monitor the broadcast for data that is not found in the cache or has been invalidated. Other work on disseminating data via periodic broadcasting of the data file has developed techniques that organize data to allow clients to selectively tune to the desired portion of the broadcast. Such schemes, unfortunately, cannot be applied to demand-driven-based context because demand-driven data cannot be predetermined. In this paper, we study the issue of selective tuning in a demand-driven-based environment. We propose and study three strategies that allow clients to repeatedly toggle between doze mode and active mode until the desired objects are obtained. One of the strategies is stateful-based in the sense that the server is aware of the schedule of doze-off/awake time determined by the clients. The other two strategies are stateless-based approaches where the clients' schedules of doze-off/awake time depend on cues broadcast by the server. We conducted a performance study and our results demonstrate that the proposed schemes are energy efficient without sacrificing on the average access times of object retrievals. Furthermore, our results show that none of the algorithms is superior in all cases.

Keywords Selective tuning, tuning time, wireless network, demand-driven, energy-efficient

1 Introduction

In our increasingly mobile world, the ability to access information on demand at any location can satisfy people's information needs as well as conferring on them a competitive advantage. In fact, it is expected that, in the near future, tens of millions of users will carry small, battery-powered, portable computers equipped with wireless communicators that enable access to a large number of information repositories. Such a new form of computing/communication environment has been referred to as *mobile computing.* The potential market for mobile computing applications is estimated to be billions of dollars annually [6]. For example, passengers will access airline schedules, investors will access stock activities, travelers will access weather and/or traffic conditions.

Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia, April 1–4, 1997.

A wireless communication channel comprises two channels, a downlink channel and an uplink channel. In this paper, we restrict our discussions to a single downlink and a single uplink channel. In practice, there may be multiple downlink and uplink channels. Data objects can be delivered to mobile clients in two modes: the *broadcasting* mode, and the *on-demand* mode. Under the broadcasting mode, data is broadcast periodically on a downlink channel. Mobile clients will "listen" to the channel and download data by filtering the incoming data stream. This method of disseminating data has the advantage that it is independent of the number of clients tuning to the channel. On the other hand, in the on-demand mode, clients submit query requests on an uplink channel and servers respond by sending the data to the clients on the downlink channel. We shall refer to the former as a *broadcast-based* approach, and the latter as a *demand-driven-based* approach.

As argued in [1, 2, 8, 9], power conservation is a key issue for small palmtops that typically run on small AA batteries. For an "average user", the power source is expected to last 2 to 3 hours before replacing or recharging becomes necessary. Therefore, it is important to manage the power resource effectively to extend the battery life. Several recent papers [8, 9, 10] propose efficient techniques for organizing broadcast data such that clients can *selectively tune* to the desired records. The effect of selective tuning is that the battery consumption can be reduced, and hence the effective battery life is extended. However, selective tuning mechanisms are largely studied in the context of broadcast-based environment. Such schemes, unfortunately, cannot be applied to demand-driven-based context because demand-driven data cannot be predetermined. Existing work on demand-driven wireless environment has largely focused on energy-efficient caching strategies. While such schemes minimize the number of uplink requests (and hence conserve energy), they are still not adequate as clients must continue to monitor the broadcast for data that is not found in the cache or has been invalidated. To our knowledge, there is no reported study on selective tuning techniques for demand-driven-based environments.

In this paper, we study the issue of selective tuning in a demand-driven-based environment. To facilitate selective tuning, clients should repeatedly toggle between the *low* energy consuming *doze* mode and the CPU operational *active* mode until the desired objects are obtained. In the active mode, clients will listen to the broadcast for up to t_a time units for the objects. In the doze mode, clients will "rest" for up to t_d time units. Like the broadcast-based counterparts, the selective tuning schemes can be categorized into *stateful* and *stateless* schemes. The difference between the two schemes is that in the former clients determine the schedule for doze-off/wake-up, while the server controls the schedule under the latter approach.

We propose a stateful-based scheme and two stateless-based schemes. The stateful-based scheme is a *static* scheme where the doze-off and awake time is predetermined and fixed by the clients. The stateless-based schemes employ *cues* broadcast by server that allow clients to doze-off and wake up as and when necessary. An extensive simulation study was conducted and our results demonstrate that the proposed schemes are energy efficient without sacrificing on the average access times of object retrievals. Our results further show that none of the proposed schemes is superior in terms of the average access times, average initial response times and average tuning times.

Our work can also be seen as a supplement to existing work on cache invalidation schemes in demand-driven-based wireless environments. There, energy consumption can be reduced by salvaging the content of the client cache after a disconnection. However, once reconnected, the clients will have to be active to download any cached objects that have been invalidated (and are needed to answer a query). Here, we are interested in how energy consumption can be minimized during the period *when the clients are operating in a connected mode* (i.e., *workaholics*!). This is especially critical since the CPU will be operating in the energy consuming active mode.

The remainder of this paper is organized as follows. In the next section, we present the context of our study, introduce the concept of selective tuning and review related work. Section 3 presents a stateful-based selective tuning strategy. In Section 4, we present two stateless-based selective tuning strategies that differ in the cues that the server broadcast. Section 5 reports on the performance study conducted and the results, and finally, we conclude in Section 6 with directions for future research.

2 Preliminaries

2.1 The Context

Our model of a wireless mobile environment, as shown in Figure 1, is similar to that specified in [3, 8]. The mobile environment consists of two distinct sets of entities: a large number of mobile clients (MC) and relatively fewer, but more powerful, *fixed hosts* (or database servers) (DS). The fixed hosts are connected through a *wired network*, and may also be serving local terminals. Some of the fixed hosts, called *mobile support stations* (MSS), are equipped with wireless communication capability. Each MSS can communicate with MCs that are within its radio coverage area called a *wireless cell*. A wireless cell can either be a cellular connection or a wireless local area network. All MCs that have identified themselves with a particular MSS are considered to be *local* to that MSS. A MC can directly communicate with a MSS if the mobile host is physically located within the cell serviced by the MSS. The servers manage and service on-demand requests from mobile clients. Based on the requests, the objects are retrieved and sent via the wireless link to the mobile clients.

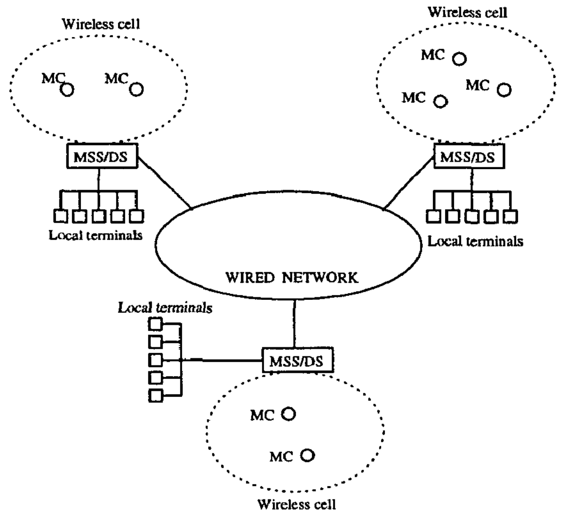

Figure 1: The wireless computing environment.

For simplicity, we also made several reasonable assumptions. First, at any given instant of time, a

MC may logically belong to only one cell; its current cell defines its location. Next, each database server has a complete copy of the database, i.e. the database is fully replicated across all the DSs. Furthermore, data is read-only, i.e., there are no updates either by the clients or at the DS. Finally, clients do not cache objects. Caching is an orthogonal issue and has been studied in [3, 13, 20].

There are three timings that are critical to mobile clients. The *access time* is the time elapsed from the moment the client submits a request to the point when all the resultant objects are downloaded by the client. The *initial access time* is the time between the submission of the request and the arrival of the first resultant object. A short initial time will allow users to view some resultant objects while waiting for the rest. This will also minimize the frustration of impatient clients. Finally, the *tuning time* is the amount of time the client spent on listening to the channel. It essentially serves as a measure on the amount of power consumed. Traditionally, once a client submits a query, it will listen to the channel until all the resultant objects are received. This leads to the tuning time being equal to the access time. Since listening to the channel requires the CPU to be in full operation (*active mode*), it is a power-consuming operation that should be minimized to cut down on power consumption.

2.2 Power Management via Selective Tuning

One of the factors that can limit the functionality of mobile computers is the lifetime of a battery. The need to recharge/replace batteries every 2-3 hours may be too disruptive for them to be widely accepted. What makes it worse is the predictions by battery experts of the modest improvement in battery capacity of only 20%-30% over the next 5-10 years [5, 18]. Hardware vendors have been driven to come up with processors and memories that require less power. For example, the Hobbit chip by AT&T is a processor that can operate in two modes [17]. In the *active* mode, it consumes 250 mW, and in the *doze* mode, it requires only 50 μW.

To exploit the capability of dual-mode processors to minimize energy consumption, Imielinski, Viswanathan and Badrinath introduce the concept of *selective tuning* for broadcast-based environments [8, 9]. Instead of listening to the channel (which requires the CPU to be in active mode) all the time, the basic idea is to organize the data such that the CPU can operate mostly in the less power consuming *doze mode*, and will "wake up" to listen to the channel only when the data of interest is broadcast. This is realized by multiplexing indexes with the data file during the broadcast. The technique was shown to reduce the tuning time (and hence energy consumption) significantly. However, it is not suitable for demand-driven based applications since the data objects to be transmitted by the server are not predictable and vary from query to query.

To facilitate selective tuning for demand-driven-based environments, mechanisms are needed to allow clients to repeatedly toggle between *doze* mode and *active* mode until the resultant objects are obtained. In the active mode, clients will listen to the broadcast for up to t_a time units for the objects. In the doze mode, clients will "rest" for up to t_d time units. There are two categories of mechanisms in which these can be achieved:

- **Stateful Server.** Schemes belonging to this category have the following characteristics:
  - clients determine the schedule for doze-off/wake-up, i.e., the t_a and t_d time units;
  - once clients submit their requests, they doze-off/wake-up according to the schedule;
  - the server is made known of the clients' schedules; this can be achieved by submitting the schedule together with the query;
  - the server follows the schedules of clients in responding to their requests, i.e., it will only download objects of clients during the active period, and avoid transmitting objects during the doze off period. Note that the server transmits the IP-address of the client, together with the resultant objects.
- **Stateless Server.** Schemes belonging to this category have the following characteristics:
  - after submitting query requests, clients look out for *cues* from the server;
  - server transmits *cues* together with data objects; the *cues* essentially indicate the content of the data that follows the *cues* and the time slot in which the next set of *cues* will be broadcast;
  - based on the *cues*, a client can doze off if its requests cannot be met, and wake up only when the next set of *cues* is broadcast.

Clearly, under the stateful approach, the client has control over the schedule in which it dozes off/wakes up. But, the server has to be very sophisticated and complex. The server must maintain and keep track of the schedules of all clients that have submitted queries. In addition, there may be several exchanges (or negotiations between clients and server) before a final "optimal" schedule can be determined. On the other hand, the stateless approach is simple, though it requires clients to abide by the cues of the server.

2.3 Related Work

Work on conserving energy consumption for demand-driven-based wireless environments has focused on caching and cache invalidation strategies [3, 11, 20]. Caching data objects at the clients can lead to a shorter access time (if the data objects queried are in the cache). This also implies a corresponding reduction in energy consumption. For objects that are not found in the cache or have been invalidated due to disconnection, cache invalidation schemes proposed in [11, 20] attempt to minimize the number of uplink requests by salvaging as much valid cache data as possible. This is critical since transmitting data consumes more energy than receiving data, and minimizing both the amount of data to receive/transmit will save energy.

There is also some work done on reducing power consumption by spinning down the disk when it is idle [4, 14, 16]. However, most of these work focus on a stand-alone portable rather than accessing data from a repository through wireless networks.

Exploiting selective tuning mechanisms in broadcast-based approaches have been investigated in [7, 8, 9, 10, 13, 19]. Essentially, the techniques multiplex indexes with the data objects broadcast. Following the indexes, clients will know when the desired objects will be broadcast, and so they can operate in the doze mode most of the time. The work in [8, 9, 13] proposed several indexing schemes – tree-based [8], hash-based [9] and signature-based [13] – for uniform data broadcast (where all objects have the same average access time). The use of secondary indexes for selective tuning in uniform data broadcast was addressed in [10]. Indexing nonuniform data broadcast (where popular objects have shorter average access time than the less popular ones) was explored in [19].

Minimizing energy consumption has also been addressed in the context of query optimization in mobile databases. In [1], Alonso and Ganguly studied the generation of energy-efficient plans. They proposed two-dimensional dynamic programming search algorithms for query optimization. In these algorithms, the optimization criterion involves minimizing the energy spent by the client, while maximizing the overall server throughput.

3 Stateful Selective Tuning Scheme

In a stateful selective tuning scheme, there are several issues that need to be addressed:

- How did the client determine its schedule?
- How did the server schedule the queries?
- What measures must be taken to ensure that the clients will eventually receive the results?

As a first cut, we propose a simple strategy that employs simple heuristics to handle the above issues. The scheme works as follows:

- the ratio of the amount of time spent in active and doze mode is fixed at t_a/t_d; once the query is submitted (t_a and t_d are piggy-backed to the query), the client will be active for t_a time units, after which it will doze off for t_d time units, and the process repeats.
- the server examines query requests using a FIFO policy. When a query is examined, from its t_a and t_d values and its arrival time, t_{arr}, the server is able to *predict* whether the corresponding client is in active or doze mode. Let $t_{ad} = t_a + t_d$, and $t_c = t_{arr} - \epsilon_c$. Here, t_c is an estimation (by the server) of the time that the client submits its query, and ϵ_c is the delay between the submission of the query by the client to the receiving of it by the server. This corresponds to the transmission time of the query over the uplink channel. We determine k, which is the smallest integer that satisfies the expression:

  $$t_c + k \cdot t_{ad} < t_{now} < t_c + (k+1) \cdot t_{ad}$$

  where t_{now} denotes the current time. The client is expected to be in active mode in t_x if isActive(t_x) is true as follows.

  $$t_c + m \cdot t_{ad} < t_x < t_c + m \cdot t_{ad} + t_a$$

  where m is an integer, $m \geq k$. It is worth noting that the delay ϵ_s for the client to receive the IP-address must be taken into consideration when t_x is considered. If the client is not in active mode, then the next query will be examined, and the process repeats.
- the client is also assigned a timeout period, $t_{timeout}$, so that if the client still has not received its first result after $t_{timeout}$ time units, then it will resubmit its query. This is to avoid the client from missing the data (especially since the client and server may not be synchronized).

The access protocol of the client is as follows:

1) submit a query (or list of referenced objects) and a schedule;
2) tune into the downlink channel, and listen for t_a time units;
   a) examine the objects broadcast;

b) if the query is being served (i.e., IP-address is in the broadcast), then download all resultant objects and STOP;

3) go into doze mode and wake up after t_d time units;

4) if timeout, goto 1; otherwise goto 2.

Under this scheme, once the IP-address is broadcast, all the data objects required by client with the associated IP-address are also broadcast. The scheme, however, fails to exploit much data sharing. For example, a client may have requested a subset of the data that is being broadcast, but it is unable to access them (since the objects of a query are identified by the IP-address). This will also lead to a long initial access time.

We would note that the above scheme is a *static* scheme in that once t_a and t_d are determined, they cannot be changed. Clearly, an optimal $[t_a, t_d]$ pair depends on the load of the system (i.e., number of clients in the mobile environment). When the number of clients is small, the access time of a query is expected to be low. In this case, it is probably more suitable to set a small doze off time or a large active time. On the other hand, if the number of clients is large, the access time is expected to be high, and so the doze off time can be larger to save up in the tuning time. We shall discuss this issue further in the experimental study.

4 Stateless Selective Tuning Schemes

For the stateless schemes, the basic protocol between clients and server is as follows:

- Besides broadcasting the data objects, the server also broadcasts *cues*. The *cues* essentially provide hints to the content of the data that follows them. We also include the time slot in which the next set of *cues* will be broadcast at the end of the *cues*.

- Clients submit query requests. Moreover, clients also listen to the *cues*. From the *cues*, clients can determine whether the data objects being broadcast in the segment are their resultant objects. If the data objects are for them, then they will listen to the segment; otherwise, they will doze off for the period of time that is determined by the length of the segment (derived from the time slot in which the next set of *cues* is to be broadcast). In this way, clients can toggle between the doze mode and active mode repeatedly until all the resultant objects are received.

In some sense, the scheme is similar to the indexing schemes used in broadcast-based

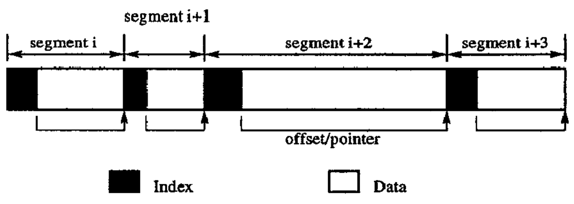

Figure 2: A broadcast to facilitate selective tuning.

approaches. We can consider the *cues* as a form of indexes, and view the dissemination of data as segments, where each segment comprises a set of *cues* and its associated data objects (see Figure 2). However, unlike the broadcast-based approaches, depending on the number of clients in the system and the size of the query results, segments are of variable length. Moreover, the content of the segments is also different and unpredictable.

Clearly, depending on the *cues* used, we can derive different selective tuning strategies.

4.1 Client Identity as Cues

The first strategy we derive uses the client identity as the *cue*. Since IP-address of a client is unique, we shall restrict our discussion to IP-address. Each segment then comprises:

- a list of IP-addresses
- an offset that indicates when the next segment will be broadcast.
- all the resultant data objects of clients whose IP-addresses are being broadcast.

This strategy includes all the resultant data for a query in a single segment. The specific access protocol for a client is as follows:

1) submit a query (or list of referenced objects);

2) tune into the downlink channel, and listen for the beginning of the next segment;

3) read the *offset* to determine the address of the next segment;

4) read and examine the list of IP-addresses;

   a) if the client's IP-address is being broadcast, examine the data objects that are broadcast and download the referenced objects;

   b) if the client's IP-address is not being broadcast, go into doze mode and tune in at the broadcast of the next segment (based on the offset); goto 3.

Like the stateful-based scheme, this scheme cannot exploit much data sharing, since all the data objects are broadcast together with the IP-address in the same segment. The same object can appear multiple times in a segment. Hence, it is expected to have a long initial access time also.

Clearly, the strategy is a *multicasting* and *batching* mechanism. The batch size is a factor that can affect the effectiveness of the strategy. If the number of clients batched is small, then it may be repeatedly switching from one mode to the other. This may be bad when the number of resultant data objects per client is small, in which case, clients will be waking up almost immediately when they doze off. On the other hand, too large a number of clients may also be bad as clients in the segment may be listening to all the objects in the broadcast, resulting in a longer tuning time too.

Another factor that affects the performance of the strategy is the number of clients in the system. When the number of clients is small, it makes no sense to have a large batch of clients in the segment since the access time will be excessively large due to the additional waiting time for the required number of clients to arrive. However, for large number of clients, a large batch size may be useful as this will help to better utilize the channel bandwidth.

We adopt an adaptive approach. The server determines the batching size based on the number of queries in the waiting queue and a predetermined *threshold* value (for the maximum number of queries to be batched). When the number of queries is less than the threshold, all the queries will be served in the current segment; otherwise only the number corresponding to the threshold will be served. Figure 3 illustrates the scheme. Here, we assume that the threshold is set to four queries, i.e., at most four queries will be batched. At t_1, the waiting queue contains only two queries. As such, both queries will be served and their data objects transmitted in a single segment. At t_2 which corresponds to the time when segment 1 has completed and the time to transmit the next segment, the queue has increased to five queries. Since the threshold is set to four, we can only serve four queries. At t_5, there is no job in the queue. As a result, nothing needs to be transmitted. At t_6, a query arrives and is served immediately since it is the only job and the channel is available.

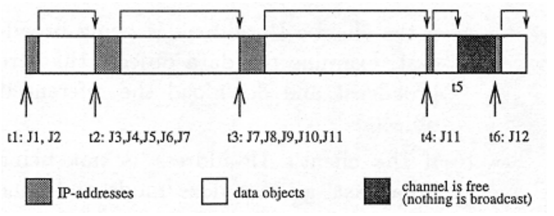

Figure 3: Client identity selective tuning scheme.

4.2 Data Groups as Cues

An alternative strategy is to partition a database into groups, and use the group-ids as the *cues*. Like [20], the grouping function can be simply a modulo function, or can be different for different types of objects, or can be generated on-the-fly (such as using signatures). Whatever a grouping function is chosen, it must be agreed upon between the server and the mobile clients. In this case, the *cues* is a list of group-ids that indicates the groups that the data in the segment belong to. However, not all the objects in the group need to be broadcast. In other words, the group-ids provide only a hint on the objects that will be broadcast, but not the actual objects that will be broadcast. Clients will examine the group-ids; if their desired objects belong to the groups, it will listen to the broadcast, otherwise it will doze off. Under this scheme, the access protocol for a client is as follows:

1) submit a query (or list of referenced objects);
2) tune into the downlink channel, and listen for the beginning of the next segment;
3) read the $offset$ to determine the address of the next segment;
4) read the list of group-ids, and examine them;
    a) if the objects referenced by the client's query belong to one of the groups being broadcast;
        i) examine the data objects that are broadcast and download the referenced objects, if any;
        ii) if all the objects have been downloaded, then STOP; otherwise doze off and tune in at the broadcast of the next segment (based on the offset), goto 3;
    b) if none of the groups broadcast are needed by the client, go into doze mode and tune in at the broadcast of the next segment; goto 3.

This scheme has the advantage that, like broadcast-based approaches, all clients whose data objects belong to the groups in the current segment can tune in (even though the required objects may not be in the current segment). However, it is expected to result in a larger tuning time as "false drops" (i.e., the client requests for an object whose group-id is being broadcast, but the object is not being broadcast) may occur.

The number of groups is a factor that is critical to the effectiveness of this approach. Too small a number of groups will increase the chances of false drops (since the number of objects per group is large), and hence result in a larger tuning time. Too large a number of groups may also result in poor utilization of the channel as more group-ids has to be transmitted. Like the IP-based scheme, the server determines the number of queries to batch adaptively (based on a *threshold* value). However, in this scheme, it is the group ids that form the *cues*, i.e., the group ids from all the resultant objects of the batched queries. Since different objects may belong to the same group, we only need to broadcast one group-id for these objects. Thus, the number of groups can vary from segment to segment.

Another factor that can affect the group-based scheme is the data access patterns. If the distribution of clients' accesses is uniform, then the false drop is expected to be small. On the contrary, if the distribution is skewed, then false drop may increase (especially when the group size is large).

5 A Performance Study

To evaluate the performance of the proposed strategies, we developed an event-driven simulator to model a wireless environment as described in Section 2. The model was implemented using the Modula-2-based DeNet simulation package [15].

The model comprises a set of clients and a server. The server contains a database of D objects. The objects are partitioned into G equal-sized groups. Each object is O bytes. The object id is O_{id} bits, and the group id is G_{id} bits. The server will batch at most the results of *batchSize* queries for downloading. Clients enter the wireless cell covered by the server, submit queries, and leave the cell once the data objects are received. The IP address of client is IP bits long. Clients' arrival is a Poisson process with mean interarrival time of λ. We assume that all queries are read-only and are processed in the mobile computer. The object ids of referenced objects are transmitted to the server and the server sends the data objects back. The number of objects referenced in a query is uniformly distributed between $Q/2$ and $3Q/2$, where Q is the mean. To determine the objects to be retrieved, we adopt a two phase strategy. In the first phase, the groups in which the objects belong to are determined, and in the second phase, objects within the groups are selected. To model the access frequency of the groups, we use the Zipf-like distribution [12]. Under the distribution, if the total number of accesses on the database is *totalAccess*, then the number of accesses of the i^{th} most frequently accessed group is given by the following expression:

$$access_i = \frac{totalAccess}{i^\theta \cdot \sum_{j=1}^{D} \frac{1}{j^\theta}}$$

where θ is known as the Zipf factor. When $\theta = 0$, the distribution becomes uniform, and all groups are equally likely to be accessed. On the other hand, when $\theta = 1$, then the distribution becomes the highly skewed pure Zipf distribution [21]. Objects within a group are picked randomly.

The uplink and downlink channel bandwidths are fixed at ω_{uplink} Kbps and $\omega_{downlink}$ Kbps respectively. For the stateful-based algorithm, the amount of doze time and active time are given by t_d and t_a respectively. The notations and definitions, together with the default values, for all the simulation parameters are summarized in Table 1.

Notation	Definition (Default Values)
D	server database size (10000 objects)
λ	mean interarrival time of mobile host (query), Poisson distribution (0.2)
Q	mean number of objects referenced by a query (20)
θ	reference skew by query Zipf distribution factor (0.8)
G	number of groups (100)
O	object size (256 bytes)
O_{id}	object id size (64 bits)
IP	IP address (32 bits)
G_{id}	group id size (8 bits)
$\omega_{downlink}$	downlink channel bandwidth (100 Kbps)
ω_{uplink}	uplink channel bandwidth (25 Kbps)
batchSize	maximum batch size (5 queries)
t_a	amount of time in active mode (1 sec)
t_d	amount of time in doze mode (5 sec)

Table 1: System and workload parameters.

The average initial response time, the average access time and the average tuning time of the queries are used as metrics to compare the various strategies. We have conducted a large number of experiments, and can only present the more interesting ones here.

5.1 Exp 1: On Stateful Scheme

In this experiment, we study the stateful tuning mechanism. Specifically, we look at the effect of varying the amount of time to be in active and doze mode. Figures 4(a) and 5(a) show the results when the amount of active time is fixed while the amount of doze time is varied from 1 to 10 sec. Figures 4(b) and 5(b) show the results when the amount of doze time is fixed while the amount of active time is varied from 0.2 to 2.0 sec. The results in Figure 4 is for a heavily loaded context ($\lambda = 0.2$), while that in Figure 5 represents a lightly loaded scenario ($\lambda = 0.8$).

From the results, we note the tradeoffs in the choice of the amount of doze time and active time.

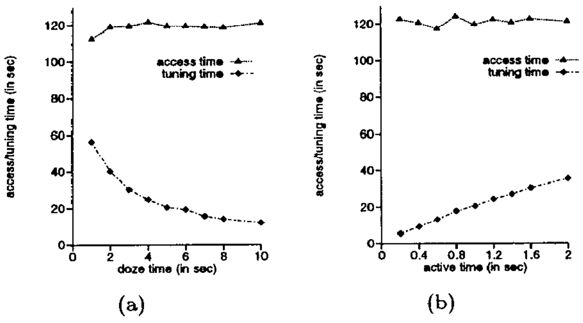

Figure 4: On stateful schemes ($\lambda = 0.2$)

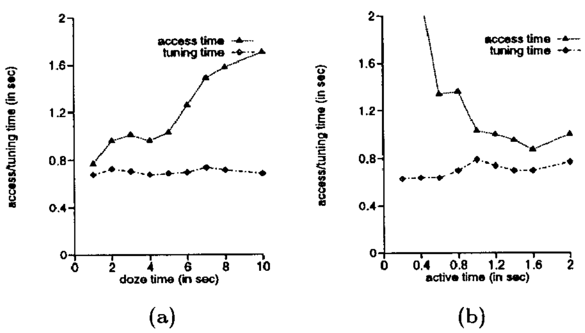

Figure 5: On stateful schemes ($\lambda = 0.8$)

When the system is heavily loaded (Figure 4(a)), choosing a high doze time may reduce tuning time. The relative difference in access time is not significant as the access time is high. On the other hand, when the system is lightly loaded (Figure 5(a)), the access time increases with a longer doze time. Moreover, the difference in access time becomes more significant. The tuning time is fairly constant when the system load is light since most of the clients would have been served before they need to switch into doze mode.

The result is slightly different when we vary the amount of active time. As shown in Figure 4(b), under heavy load, as the amount of active time increases, the tuning time also increases. This is expected since the access time is large which in turn implies that the number of times that the clients must be in active mode also increases. However, the access time when the system load is light decreases with higher active time (see Figure 5(b)). This is so since a long active time would mean that the chances of dozing off is kept low (i.e., most of the clients need not doze off).

Since the workload cannot be predetermined, an adaptive mechanism is needed for optimal performance. We shall leave this issue for future research, and fix t_d and t_a at the default values.

5.2 Exp 2: On IP-based Scheme

In this experiment, we study the IP-based stateless scheme. We also look at two variations of the IP-based scheme. The first is essentially the basic strategy that broadcasts all objects of a query.

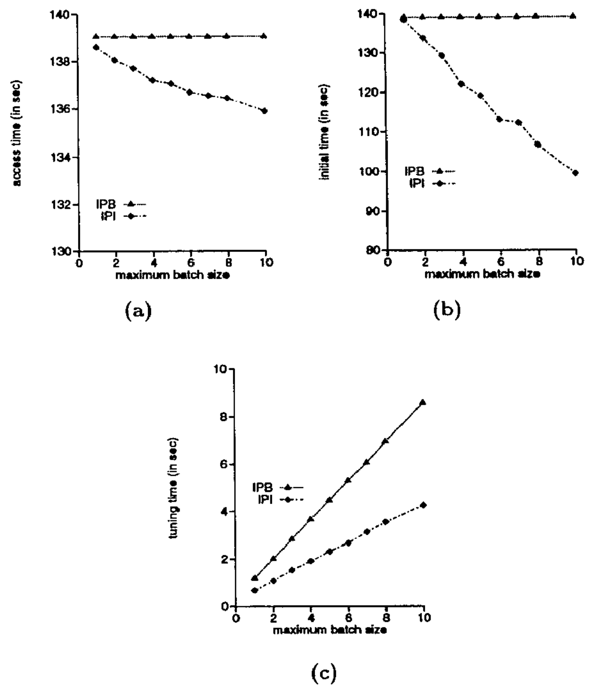

Figure 6: Comparing IP-based stateless schemes.

The second, however, exploits batching to avoid downloading objects multiple times in a batch. We shall denote the first method as IPB (IP-basic) and the second method as IPI (IP-Improved). Figure 6 shows the results of the average access time, average tuning time and average initial access time as the maximum batch size increases from 1 to 10.

As expected, we note that IPI outperforms IPB in all cases since it downloads fewer objects in each batch. We also note that there is a tradeoff between reduction in access/initial time and increased tuning time as the batch size increases. Access time is reduced since a larger batch size will imply more common objects within a batch. Tuning time increases as more clients will have to examine more objects before all their results are received. In view of these results, for the subsequent experiments, we shall restrict to IPI and use the default batch size.

5.3 Exp 3: On Group-based Scheme

In this experiment, we study the effect of batch sizes on the group-based scheme. The results are shown in Figure 7. As shown in the figure, a larger batch size can reduce the access and tuning times slightly. This is because as the batch size increases, more data is transmitted in a segment, and so the chances of false drops will decrease. However, the average initial access time increases as the batch size increases. This is due to the larger number of cues and the longer segment length.

5.4 Exp 4: A Comparative Study

In this experiment, we study the effect of query interarrival time on the strategies by varying the

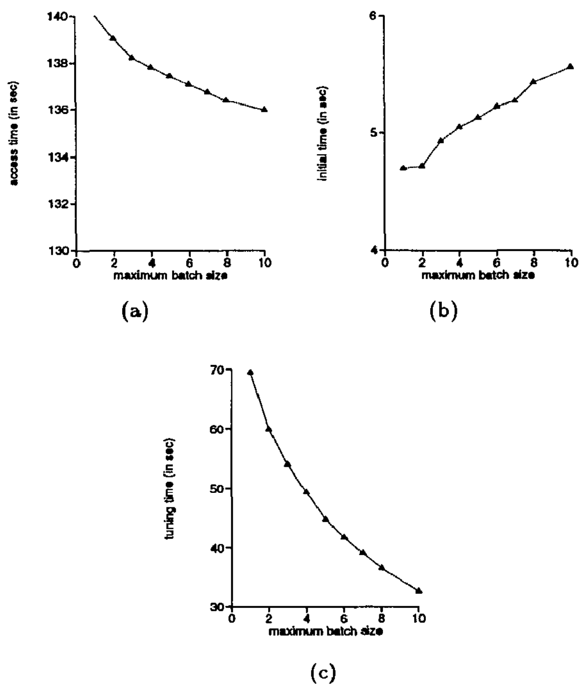

Figure 7: On group-based scheme.

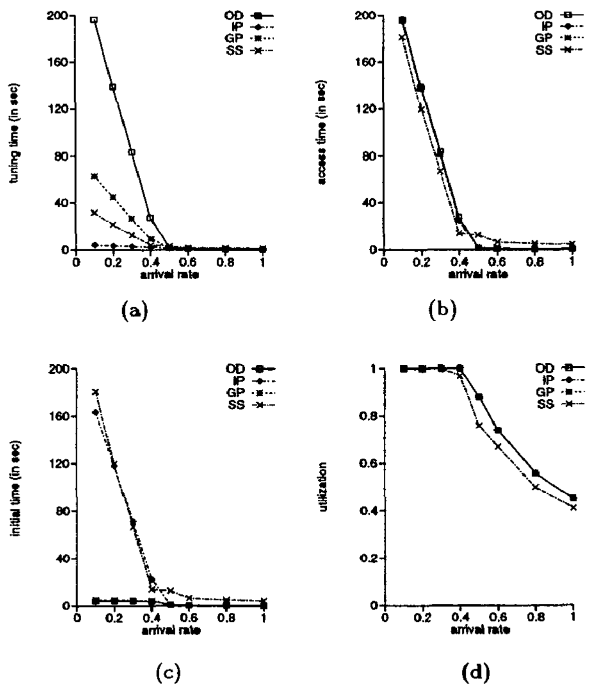

Figure 8: Comparison of various schemes.

interarrival time from 0.1 to 1.0. We shall denote the stateful-based scheme, the IP-based algorithm and the group-based algorithm as SS, IP and GP respectively. The basic exclusive on-demand strategy (denoted OD) is also used as a reference to study the benefits of the proposed schemes. Under scheme OD, clients listen to the broadcast until all data objects have been received (without dozing off at all). The results are shown in Figure 8.

From the results, we note that none of the strategies performs best in all cases. In terms of tuning time (Figure 8(a)), we see that all the proposed schemes (IP,GP and SS) are superior over scheme OD. This demonstrates the effectiveness of the schemes in minimizing power consumption. As expected, when the load is light, all the schemes require the same amount of tuning time (since the opportunity for dozing off is low). However, under high load, the tuning time of the proposed schemes is drastically reduced. For IP, its tuning time can be reduced to only 3% that of OD, while SS and GP can have tuning time of down to 20% and 30% that of OD respectively. GP is less effective because of the simple grouping strategy that we have adopted which can lead to high false drops.

For the average access time (Figure 8(b)), all the schemes perform equally well (or poorly) except for the stateful-based scheme (SS). Algorithm SS performs poorly (in access time) under lightly loaded context, but is superior when the system is heavily loaded. When the load is light, the access time is expected to be low. By allowing clients to doze-off for a long time (relative to the access time) inevitably increases the access time unnecessarily. However, as the system load increases, algorithm SS can lead to better average access time. While this appears unintuitive, three reasons account for this behavior. First, under other strategies (IP and GP) the cues are at the front of the segment, and every client's access time includes all the cues. But, under SS, objects that are broadcast earlier will have a smaller number of IP-addresses to examine. Second, the scheduling policy adopted minimizes the number of times a client needs to doze off since the next query to be addressed is examined in FCFS policy. Finally, under SS, some queries are processed earlier, while others are delayed. It turns out that the net effect is a gain in our study.

It is also worth noting, from Figure 8(c), that OD performs best in the average initial access time. This is so since every client is listening to the broadcast, and objects that are broadcast in response to other queries may also be relevant to it. SS performs the worst in most cases for the same reason given above for its access time performance since the client can only receive its data objects, if any, when it is in active mode.

Finally, we note that the utilization of the bandwidth is the same for all schemes except SS which is slightly better (see Figure 8(d)). This also helps to explain why SS performs poorer than the other schemes under light load – it may doze off "unnecessarily" and fail to utilize the bandwidth effectively.

6 Conclusion

In this paper, we have addressed the issue of selective tuning in a demand-driven-based environment.

Specifically, we addressed how energy can be saved when the clients are operating in connected mode. We proposed three strategies that allow clients to repeatedly toggle between *doze* mode and *active* mode until the desired objects are obtained. In the active mode, clients will listen to the broadcast for up to t_a time units for the objects. In the doze mode, clients will "rest" for up to t_d time units. One of the strategies is based on the stateful approach where the clients determine the schedule and the server is made known of the schedule. The other two strategies are stateless-based approaches where the clients' schedules depend on cues broadcast by the server. We conducted a performance study and our results demonstrate that the proposed schemes are energy efficient without sacrificing on the average access times of object retrievals. Furthermore, our results showed that none of the algorithms is superior in all cases in terms of average access times, average initial access times and average tuning times. Thus, the algorithms are useful for different applications.

We plan to extend the work in the following aspects. First, we will explore how to improve the various proposed schemes in terms of robustness and further reduction of the tuning time. For example, the stateful scheme can be fine tuned so that the doze off and awake time can adapt to the changing workload of the system. As another example, different grouping schemes may perform differently for the group-based scheme. Second, we will integrate our schemes with the cache invalidation technique and study the combined effect in minimizing energy consumption. Finally, we plan to implement and study the techniques in a real wireless environment.

References

[1] R. Alonso and S. Ganguly. Query optimization for energy efficiency in mobile environment. In *Proceedings of the 1993 Workshop on Optimization in Databases*, Aigen, Austria, September 1993.

[2] R. Alonso and H. Korth. Database systems in nomadic computing. In *Proceedings of the 1993 ACM-SIGMOD International Conference on Management of Data*, pages 388–392, June 1993.

[3] D. Barbara and T. Imielinski. Sleepers and workaholics: Caching in mobile distributed environments. In *Proceedings of the 1994 ACM-SIGMOD International Conference on Management of Data*, pages 1–12, June 1994.

[4] F. Douglis, P. Krishnan and B. Marsh. Thwarting the power hungry disk. In *Proceedings of the 1994 Winter USENIX Conference*, 1994.

[5] Eager. Advances in rechargeable batteries pace portable computer growth. In *Proceedings of the 1991 Silicon Valley Personal Computer Conference*, 1991.

[6] Y. Huang, P. Sistla and O. Wolfson. Data replication for mobile computers. In *Proceedings of the 1994 ACM-SIGMOD International Conference on Management of Data*, pages 13–24, June 1994.

[7] T. Imielinski and S. Viswanathan. Adaptive wireless information systems. In *Proceedings of the SIGDBS Conference*, pages 19–41, Tokyo, Japan, October 1994.

[8] T. Imielinski, S. Viswanathan and B.R. Badrinath. Energy efficient indexing on air. In *Proceedings of the 1994 ACM-SIGMOD International Conference on Management of Data*, pages 25–36, June 1994.

[9] T. Imielinski, S. Viswanathan and B.R. Badrinath. Power efficient filtering of data on air. In *Proceedings of the 4th International Conference on Extending Database Technology*, pages 245–258, March 1994.

[10] T. Imielinski, S. Viswanathan and B.R. Badrinath. Data on air: Organization and access. In *IEEE TKDE (to appear)*, 1996.

[11] J Jing, O. Bukhres, A. Elmagarmid and R. Alonso. Bit-sequences: A new cache invalidation method in mobile environments. Technical report, Dept. Computer Sciences, Purdue University, 1994.

[12] D.E. Knuth. *The Art of Programming, Vol. 3: Sorting and Searching.* Addison Wesley, 1973.

[13] W.C. Lee and D. Lee. Using signature and caching techniques for information filtering in wireless and mobile environments. *Journal of Distributed and Parallel Databases (to appear)*, 1996.

[14] K. Li, R. Kumpf, P. Horton and T. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the 1994 Winter USENIX Conference*, 1994.

[15] M. Livny. Denet user's guide, version 1.0. Technical report, Computer Sciences Dept, University of Wiscônsin-Madison, 1988.

[16] B. Marsh, F. Douglis and P. Krishnan. Flash memory file caching for mobile computers. In *Proceedings of the 27th HICSS Conference*, pages 451–460, January 1994.

[17] P.V. Argade, et. al. Hobbit: A high-performance, low-power microprocessor. In *Proceedings of COMPCON'93*, pages 88–95, February 1993.

[18] S. Sheng, A. Chandrasekaran and R. E. Broderson. A portable multimedia terminal for personal communications. *IEEE Communications Magazine*, pages 64–75, December 1992.

[19] K.L. Tan and J.X. Yu. Energy efficient filtering of nonuniform broadcast. In *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems*, pages 520–527, May 1996.

[20] K.L. Wu, P.S. Yu and M.S. Chen. Energy-efficient caching for wireless mobile computing. In *Proceedings of the 12th International Conference on Data Engineering*, February 1996.

[21] G.K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison Wesley, 1949.