On-Demand Broadcast: New Challenges and Scheduling Algorithms[*]

Mohamed A. Sharaf and *Panos K. Chrysanthis*
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
{msharaf,panos}@cs.pitt.edu

Abstract

With the rapid growth in wireless technologies and the cost effectiveness in deploying wireless networks, wireless computers are quickly becoming the normal front-end devices for accessing enterprise data. In this paper, we are addressing the issue of efficient delivery of business-critical data in the form of summary tables to wireless clients equipped with OLAP front-end tools. Towards this, we propose a new heuristic, on-demand scheduling algorithm, called *STOBS-α*, that aggregates requests and broadcasts the results only once to all clients. STOBS-α exploits the structural dependencies among summary tables to maximize data sharing based on aggregation and does not assume fixed length or uniform broadcasts. The effectiveness of our proposed heuristic was evaluated experimentally using simulation with respect to both access time and fairness, as well as power consumption in the case of mobile clients.

1 Introduction

With the rapid growth in wireless technologies and the cost effectiveness in deploying wireless networks, wireless devices are quickly becoming alternative platforms for accessing enterprise data. This combined with the increased popularity of palmtop and hand-held computers as well as the availability of light yet powerful laptop computers, mobile computers will become the normal front-end devices hosting sophisticated business applications.

One such sophisticated business application which is central to the success of any enterprise is the support of decision making. Without an effective decision support system, enterprises will be unable to exploit opportunities as they appear anywhere and anytime. For good decision making, executives and managers need to count on up-to-date, business-critical data, being instantly available on their hand-held and wireless computers [1]. Such data are typically in the form of summarized information tailored to suit the users' analysis interests.

Traditionally, decision makers use OLAP (On-Line Analytical Processing) tools to execute decision support queries on the enterprise data warehouse or data mart. OLAP tools provide multidimensional views of data for decision-making and support [7]. The multidimensional data model abstracts data in the form of a *data cube*

[*]This work is supported in part by National Science Foundation award ANI-0123705 and by National Center for Disease Control.

[1]Good mobile decision making is also critical in a public health surveillance system as the one we are developing at the University of Pittsburgh (www.health.pitt.edu/rods), especially during emergencies and periods of crisis

where dimensions are the subject of interests (aggregated attributes) and the cell values are the measures of interest [13]. An OLAP server may store multiple summary tables (subcubes) for efficient access by queries issued by OLAP tools at the client. An interesting property of summary tables which we call *derivation dependency*, is that one summary table can be derived from one or more summary tables.

As an example, consider the case of stock brokers on the floor of a Stock Exchange guided by investors on the galleries, all of them with hand-held devices or notebooks. In this interactive environment, brokers and investors decide on stock purchases and sales by analyzing stocks along multiple parameters (dimensions). Different dimensions are more important than others and the importance of dimensions as well as the popularity of stocks shifts following the developments in the market place. Consequently, the access rate to the stocks that are considered *hot* and the different dimensions shift as well. This analysis process can be viewed as a sequence of sessions. Each session typically starts with some summarized tables involving few dimensions and, if necessary, drills down to high dimensional tables for more detailed information.

In this paper, we are addressing the issue of efficient delivery of summary tables to wireless clients (e.g., on a company wireless intranet) equipped with OLAP front-end tools. In wireless networks, broadcasting is the primary mode of operation for the physical layer. Thus, broadcasting is the natural method to propagate information in wireless links and guarantee scalability for bulk data transfer. Specifically, data can be efficiently disseminated by any combination of the following two schemes: *broadcast push* and *broadcast pull*. These exploit the asymmetry in wireless communication and the reduced energy consumption in the receiving mode. Client devices are assumed to be small and portable, and most often rely for their operation on the finite energy provided by batteries. Servers have both much larger bandwidth (downlink) available than client devices and more power to transmit large amounts of data.

In broadcast push the server repeatedly sends information to the clients without explicit client requests. Any number of clients can monitor the broadcast channel and retrieve data as they arrive on the broadcast channel. If data is properly organized to cater to the needs of the clients, such a scheme makes an effective use of the low wireless bandwidth and is ideal to achieve maximal scalability [1, 15, 14].

In broadcast pull, the clients make explicit requests for data. If multiple clients request the same data at approximately the same time, the server may aggregate these requests, and only broadcast the data once. Such a scheme also makes an effective use of the low wireless bandwidth and clearly improves user perceived performance. Several scheduling algorithms have been proposed that attempt to achieve maximum aggregation [3, 17, 9, 28, 29].

Assuming the traditional OLAP server basic functionality, the broadcast pull or *on-demand* environment as shown in Figure 1 is the most suitable for supporting wireless OLAP query processing. Interestingly, the broadcast scheduling problem arising in the wireless OLAP system exhibits the above mentioned derivation

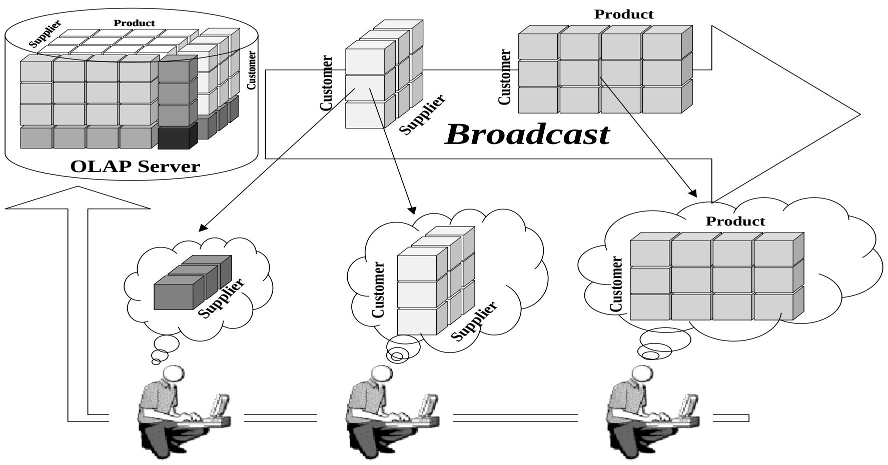

Figure 1: Wireless OLAP System

dependency feature. That is, every client request is for one of the summary tables and a table requested by a client may subsume the table requested by another client. However, each table satisfying a particular request incurs a different processing cost, and this cost needs to be considered in selecting the specific table to broadcast at a given point. Since request aggregation is commonly used by general content delivery scheduling algorithms for efficient data dissemination, the derivation dependency property adds a new optimization dimension to the request aggregation process that allows further broadcast efficiency and scalability.

In this paper, we propose a new, heuristic, on-demand scheduling algorithm called *Summary Tables On-Demand Broadcast Scheduler* (*STOBS*-α). STOBS-α is non-preemptive and considers the varying sizes of the summary tables. The unique characteristic of STOBS-α is its α-optimizer that exploits the derivation dependency among the summary tables to increase sharing among clients that goes beyond the exact match of requests of all the current on-demand scheduling approaches.

The performance of our new heuristic was evaluated experimentally using simulation by comparing it to relevant heuristics previously proposed in literature, namely FCFS, SSTF, and RxW. To the best of our knowledge, RxW is currently the best performing preemptive scheduler reported in the literature [5]. Our experimental results have shown that STOBS-α outperforms the RxW, reducing the access time by up to 75%.

For mobile clients, savings in power consumption is particularly important since they operate on batteries. Power consumption is also becoming a key issue for all other computer products given the negative effects of heat. Heat adversely affects the reliability of the digital circuits and increases costs for cooling [24]. *STOBS* achieves power reductions up to 30% less than RxW, while reducing the average access time by 63%. The latter saving could be increased to 75%, while preserving the same power consumption as RxW by adjusting the value α of the optimizer.

The rest of this paper is organized as follows. The next section presents an overview of the related work in

OLAP technologies and broadcast-based data dissemination techniques. In Section 3, we discuss our assumed wireless OLAP environment. In Section 4, we review various scheduling heuristics previously proposed, and in Section 5, we present *STOBS*, our new on-demand scheduling algorithm. Our simulation testbed and experimental results are presented in Sections 6 and 7 respectively. Section 8 concludes the paper with a summary and discussion of the future work.

2 Background and Related Work

In a decision support environment, sets of facts are analyzed along multiple dimensions. This led to the development of the multidimensional data model that represents a set of facts in a multidimensional space in a way that facilitates the generation of summarized data and reports [18]. In this model, data is typically stored using a star schema. The star schema consists of a single fact table storing the measures of interest (e.g., *sales*, or *revenue*) and a table for each dimension (e.g., *product*, *time*, or *region*).

OLAP queries typically operate on summarized, consolidated data derived from fact tables. The needed consolidated data by an OLAP query can be derived using the *data cube* operator [10]. The data cube operator is basically the union of all possible *Group-By* operators applied on the fact table. A data cube for a schema with *N* dimensional attributes, will have 2^N possible subcubes. Given that the data cube is an expensive operator, often subcubes are pre-computed and stored as summary tables at the server. Basically, a summary table can be modeled as an aggregation query, where the dimensions for analysis are the *Group-By* attributes and the measures of interest are the aggregation attributes. A detailed summary table T_d can be used to derive a more abstract one T_a. In such a case, the abstract table T_a has a derivation dependency on T_d. this property *derivation dependency*. For example, in Figure 1, by adding the measure values across *customer*, the detailed table (*supplier*, *customer*) can be used by a client to extract the abstract table (*supplier*).

The idea of using summary tables to derive one from another has been widely used in materialized views selection. The objective is to select the appropriate set of tables for storing (materialization), so that to speed up future query processing, while meeting the space constraints [13, 11, 12, 19, 25]. To facilitate the selection process, the search lattice was introduced in [13]. The search lattice is a directed graph to represent the subcubes space that captures the derivation dependencies among subcubes. For example Figure 2 shows the lattice for the (*Supplier(S), Product(P), Customer(C)*) schema.

In this paper, we also use the property of derivation dependency of the summarized tables and the idea of search lattice in selecting the appropriate tables to broadcast over wireless links, such that the user perceived latency is minimized.

As mentioned in the introduction, broadcast pull and broadcast push are the basic methods for efficient data dissemination in wireless networks. Dykeman et. al. pointed out that First Come First Serve (FCFS) scheduling

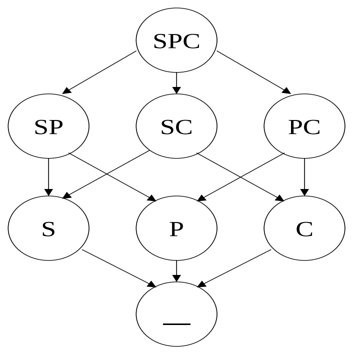

Figure 2: Data Cubes Lattice

would provide poor access time for a broadcast pull environments [9] and proposed several alternative efficient algorithms in [9, 29]. We will elaborate more on these algorithms in Section 4. The RxW algorithm was proposed in the context of non-preemptive environments with homogeneous request, i.e., request for data of the same size [5]. The heterogeneous requests problem, i.e., requests of varying sizes, is approached in [3], through a preemptive algorithm called MAX. In [17], the authors gave an offline algorithm of O(1) for minimizing the average access time in on-demand systems, while [6] studied minimizing the maximum access time.

In broadcast push, techniques for data organization have been investigated. For example, the broadcast disks organization was introduced in [1], while a scheme to generate non-uniform broadcasts that support range queries is described in [27]. For fast search and selective tuning, several indexing techniques have been proposed (e.g., [15, 4]). The work in [8] introduced techniques for reducing data dissemination costs in a subscription environment, by exploring the idea of merging queries with overlapping answers. Research on balancing the broadcast push of information and broadcast pull data delivery methods appeared in [2, 26].

3 Wireless OLAP Model

In this section, we are presenting our model for the wireless OLAP environment. Our assumed architecture is based on broadcast pull scheme as shown in Figure 1. The OLAP server is responsible for maintaining and disseminating the summary tables. We are assuming that all the lattice subcubes are ready at the server, which is a reasonable assumption, specially for relatively small size data marts. The Essbase system (according to [13]) is an example of commercial product that materialize all the possible summary tables.

A client sends an uplink request for a table on the *uplink channel*. Then it listens to the downlink channel for a response. A client can be in one of two modes, either a *tune* mode, listening to the broadcast, or in a *wait* mode, where the client is idle waiting for the response. Clients depend on the server to satisfy all their requests; they are not accessing any local storage and previous answers are not locally cached for future use.

An uplink request Q is characterized by the set of its Group-By attributes D. Hence, we represent a request as Q^D and the corresponding table as T^D. A summary table T^{D1} *subsumes* table T^{D2}, if $D2 \subseteq D1$, similarly, T^{D2} is *dependent* on T^{D1}. We denote the number of dimensional attributes in the set D as $\mid D \mid$.

The smallest logical unit of a broadcast is called a *packet* or *bucket*. A broadcast table is segmented into equal sized packets, where the first one is a *descriptor* packet. Every packet has a header, specifying whether it is data or descriptor packet, the offset (time step) to the beginning of the next descriptor packet, and the offset of the packet from the beginning of its descriptor packet. The descriptor packet contains a table descriptor which has an *identifier* describing the aggregation dimensions of the table being broadcast, the number of attribute values or tuples in the table and the number of data packets accommodating that table. We are assuming that no single data packet is occupied by tuples from different tables. We refer to the period required to broadcast a table as a *broadcast cycle*. That is, each summary table is broadcast within a broadcast cycle that starts with the table descriptor packet and broadcast cycles have variable duration.

Here we used bit encoding to represent the client request and the descriptor packet identifier. The representation is a string of bits; its length is equal to the number of the complete schema dimensions and each bit position is equivalent to one of the dimensions d_1, d_2, ..., d_n.

If a table T^D has dimension $d_x \in D$, then the bit at position *x* is set to 1, otherwise it is a zero. For example, assume the (*supplier, product, customer*) schema. The representation of the (*supplier, customer*) summary table will be 101. This scheme can be easily extended to include tables with more than one measure and different aggregation functions such as *Sum, Avg, Min, Max*. But, without loss of generality, we are assuming only one measure attribute and *Sum* as the aggregation function in this paper.

When a client submits a request for table T^R on the uplink channel, it immediately tunes to the downlink channel, examining descriptor packets. When it finds a descriptor packet, say for table T^B, the client classifies T^B as:

1. *Exact match*: if the aggregation dimensions in T^B are the same as T^R (i.e., $R = B$).
2. *Subsuming match*: if T^B subsumes T^R, and T^B is not an exact match for T^R (i.e., $R \subset B$ and $R \neq B$).
3. *No match*: if it is neither an exact match nor a subsuming match (i.e., $R \not\subseteq B$).

For example, assume R is (*supplier, product*), then B_1 = (*supplier, product, customer*) is a subsuming match, while B_2 = (*product*) and B_3 = (*supplier, customer*) are examples of no match.

Depending on the kind of match (as we will see in Sections 4 and 5), the client will either tune to the next sequence of data packets to read (download) table T^B or it will wait for the next broadcast cycle.

3.1 Performance Metrics

The performance of any scheduler in a wireless environment can be expressed in terms of:

- **Access Time:** It is the user perceived latency from the time a request is posed to the time it gets the response. Its two components are the *tune time* and *wait time*.

- **Tune Time:** It is the total period of time spent by the client listening to the downlink channel either reading a descriptor packet or a stream of data packets containing the requested summary table.

- **Wait Time:** The total period of time a client spends waiting for a descriptor packet to appear on the downlink channel until it finds a matching one. A client network interface is switched off during the wait time and the client does not listen to the channel.

In our model, requests are for different tables of different sizes. Hence, an equal quality of service for all requests is neither feasible nor fair. However, requests can be logically grouped in classes, where the quality of service is different from one class to another. Accordingly, we adopt *fairness* as one of the proposed scheduler objectives. In calculating fairness we used the notion of *stretch* [3], both defined as follows:

- **Service Time:** The time that takes the server to completely transmit a table on the downlink channel in an non-preemptive manner.

- **Stretch:** The stretch of a request is defined as the ratio of the response time (access time) of a request to its service time.

- **Fairness:** The fairness measure is computed as the standard deviation of the requests stretch values [22].

The service time depends on the size of the table and hence the notion of stretch normalizes the access time of a request with respect to the size of the resulting table. A low standard deviation implies the fairness of scheduling policy for all jobs, while a higher value means that the policy is unfair towards some class [22].

4 Scheduling Algorithms

In this section, in order to set the stage for the presentation of our new scheduling algorithm, we will discuss in some more detail four of the major scheduling policies previously proposed in the literature: *First-Come First-Served*, *Shortest Service Time First, Most Requests First,* and *RxW*. In Section 7, we will evaluate our algorithm by comparing its performance in terms of these algorithms.

- **First Come First Served (FCFS):** In this simple scheduling policy, requests are served and broadcast in their arrival order.

- **Shortest Service Time First (SSTF):** The scheduler serves the data item which has the minimum resource requirements first. In our case, the downlink channel is the shared resource. Hence, the requested data item with the smallest size is broadcast first.

- **Most Requests First (MRF):** The scheduler selects the most popular item to broadcast, i.e., the data item with the maximum number of pending requests.

- **RxW:** This scheme combines the benefits of MRF and FCFS. At each broadcast cycle, the server selects the data item with the highest $R \times W$ value to broadcast. The R value is the number of requests for that item, while the W value is the longest wait time for a request to that data item.

The SSTF policy is biased in favor of requests for small objects. Requests for large data items are prone to starvation. In contrast, FCFS is not susceptible to starvation. However, it can be particularly bad for small requests as they may need to wait in the queue until the broadcast of a large data item is finished and a new scheduling point is reached. The intuition underlying RxW is that "hot" or popular data items are disseminated as soon as possible yet it avoids starvation of "cold" or less popular data items by means of an aging scheme.

It has been shown that RxW outperforms the FCFS and the MRF techniques in reducing the average access time under a skewed data access pattern [5]. The performance of RxW was not previously compared to SSTF, as the RxW was designed to handle requests for homogeneous data items (same size) and the service time for all requests at the server side is the same.

As it is clear from the preceding discussion, the already existing variation of scheduling algorithms are performing well for some cases, but they might fall short in others. Hence, the selection of a broadcast scheduler should be tightly coupled to the application context, which may have its specific data structure and access pattern characteristics. Accessing OLAP data cubes is one example of such contexts, which might need a special scheduler that is capable of boosting the access performance in a broadcast environment and we are proposing such algorithm in the next section.

5 Summary Tables On-Demand Broadcast Scheduler (*STOBS*-α)

The profile for OLAP summary tables access has the following key features:

1. Heterogeneity: summary tables are of different dimensionality (number of dimensional attributes) and varying sizes.
2. Skewed Access: Request from OLAP clients usually form a hot spot within the data cubes lattice. Most of the time queries are accessing low dimensionality tables and they often drill down for detailed ones.
3. Derivation dependency: it is often possible to use one detailed table to extract other tables.

The *Summary Tables On-Demand Broadcast Scheduler* (*STOBS*-α) that we are proposing in this section, consists of two components: A *normalizing* (basic selection) component, which captures the first and second features above and the α-*optimizing* component that exploits the third feature above to control the degree of sharing.

In *STOBS*, the server queues up the clients requests as they arrive. For each request Q^X for a summary table T^X, the server maintains the following three values:

- R: The number of requests for T^X. This value is incremented with every arrival of a request for T^X.
- A: The time the first request Q^X has been waiting for table T^X.
- S: The size of table T^X.

When it is time for the server to make a decision which table to broadcast next, it computes the $\frac{R \times A}{S}$ value for each request in the queue. The request with the highest value is selected to be broadcast.

The parameter α defines the degree of flexibility in broadcasting a summary table and eliminating from the broadcast some of its dependent tables. For example, for $\alpha = 2$, if the server selects a table T^X to broadcast, then the server discards every request in the queue for a table T^Y that can be derived from T^X and is up to two levels lower in the data cubes lattice. (Recall from Figure 2 that the position of a table in the lattice is lower than the position of any table from which it can be derived.) Formally, T^Y can be discarded and is not broadcast iff T^X is broadcast, $Y \subset X$ and $\mid X \mid - \mid Y \mid \leq \alpha$.

Consequently, a client can use a table T^X that subsumes the table T^Y it originally requested and is up to two levels higher in the data cubes lattice. Formally, a client that requested T^Y can use T^X iff, $Y \subset X$ and $\mid X \mid - \mid Y \mid \leq \alpha$.

The value of α ranges from 0 to the maximum data cube dimensionality MAX. At $\alpha = 0$ there is no flexibility in using summary tables and the client access is restricted to exact match. At $\alpha = MAX$, a client will use the first subsuming matching table. In cases where $0 < \alpha < MAX$, the client will use the first table that subsumes its original request and is up to α levels higher than it in the data cubes lattice.

In order to distinguish between *STOBS*-α and the algorithms in Section 4 that are restricted to exact match, we are calling the algorithms mentioned in Section 4 *strict algorithms*, STOBS-0 is a strict algorithm as well, while the family of STOBS-α, where $\alpha > 0$, are *flexible algorithms*. The value for α is known to the server and clients (it can be part of the table descriptor information).

As an example, consider the search lattice shown in Figure 3, in which nodes are summary tables. Q^X is a request to the 4-dimension table (d_1, d_2, d_3, d_4). Assume the search lattice nodes shown in figure, are the tables for which there exist at least one request and $\alpha = 2$. Also, assume that table T^X corresponding for Q^X is selected for broadcast. Then, clients requests for tables (d_1, d_2), (d_1, d_3), (d_1, d_2, d_3), and (d_1, d_2, d_4) will be satisfied by T^X. While Clients requested tables (d_1), (d_2), (d_3), and (d_4) will just wait for the next broadcast cycles.

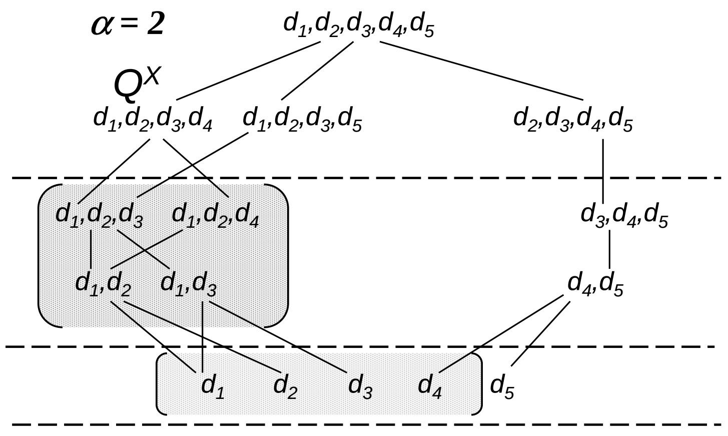

Figure 3: Flexibility

The intuition for the algorithm is to capture all the specific features of summary tables access in an on-demand broadcast environment. The $\frac{R \times A}{S}$ encapsulates all the factors affecting response access time. The α parameter controls the degree of flexibility. The advantage of the flexibility is to find another aspect of common interest other than the exact strict one. The drawback is the extra time a client has to spend tuning to a detailed table rather than a summarized one. Picking a reasonable value for α will balance the trade-off between reducing the wait time and increasing the tune time. As in [13], we are assuming a linear cost model for aggregate query processing, where a table scan is required to compute the result. This processing time can simply overlap with the tuning phase and is determined by the memory transfer rate. Given that the memory transfer rate is much higher than that of the wireless network which is the real bottleneck, the cost of any extra required filtering and extraction is very small and may be neglected.

As an example for the flexibility trade-off, consider the case where α is set to 2. In case of request for table T^{high}, where $|X| - |high| \leq 2$. If the $\frac{R \times A}{S}$ value for the request for table T^{high} is still not high enough, then disseminating T^X will reduce the wait time by a client requested T^{high}. On the contrary, a client requested table T^{low}, where $|X| - |low| > 2$, if T^X is disseminated, the client requested T^{low} would rather wait for the next broadcast cycles to avoid the costly tune time of downloading T^X.

Let us now consider a simple numeric example that highlights the differences in scheduling decisions and average access time between the algorithms in Section 4 and the algorithm we have just proposed. Table 1 shows the example settings, where there are four pending requests Q_1, Q_2, Q_3, and Q_4 for four different tables T_1, T_2, T_3, and T_4. The R_i, A_i, and S_i values for request Q_i are as described above. Additionally, we are assuming that table T_2 is derivable from T_4. Each scheduler has to make a decision what is the sequence of tables to broadcast given the queue status at each broadcast cycle. In this snapshot, the four requests constitute the whole workload, i.e., no more requests will arrive at the server.

	Q_1	Q_2	Q_3	Q_4
R_i	2	1	1	2
A_i	5	4	10	14
S_i	20	25	50	60

Table 1: Example Settings

Algorithm	Broadcast Sequence	Average Access Time	Broadcast Size
FCFS	T_4, T_3, T_1, T_2	116.1	155
SSTF	T_1, T_2, T_3, T_4	90.3	155
RxW	T_4, T_1, T_3, T_2	102.8	155
STOBS-0	T_1, T_4, T_2, T_3	85.3	155
STOBS-2	T_1, T_4, T_3	77.0	130

Table 2: Example Results

Table 2 shows the broadcast sequence generated by each algorithm (left most table is the first to be broadcast), the corresponding average access time, and the broadcast size. Assume that the transmission time of a table is equal to its size, hence, the transmission time for table T_1 is 20 units and its access time using FCFS is equal to $A_1 + S_4 + S_3 + S_1$, where $(A_1 + S_4 + S_3)$ is the wait time and S_1 is the tune time.

The sequences generated by the FCFS depends only on each request Age (A), while that from SSTF is size (S) dependent. We can see in the sequence generated by RxW that the large table T_4 is the first to be broadcasted due to its high A and R values. However that gave an average access time higher than the SSTF algorithm. In STOBS-0, sending the small popular table T_1 first, followed by T_4, gave the lowest access time among the strict algorithms. As it is possible to derive T_2 from T_4, STOBS-2 selects T_4 for transmission and discards T_2, converting the wait time for T_2 into a tune time to T_4 and in addition eliminating part of the wait time for T_3.

6 Performance Evaluation Testbed

We implemented a system simulation model to evaluate the potential gains using the *STOBS*-α algorithm by comparing it to the relevant scheduling algorithms previously proposed in literature, namely, SSTF, FCFS, and RxW.

We modeled the environment as a single server with a set of clients. There is a single downlink broadcast channel over which all data is disseminated to the clients and a single uplink channel that clients use to send uplink requests. We are assuming that clients are able to complete any uplink request in a single uplink packet. For the purposes of this simulation, we have ignored all communication errors.

We generated a synthesized lattice for a six-dimensional data cube. The sizes of lattice subcubes are computed as in [16], where a subcube is given a binary code C. The binary code is similar to the bit encoding we used for identifying cubes on broadcast. Then the subcube size (number of tuples) is set to C^2. The final cube size is the product of generated number of tuple and the number of attributes (dimensional and measure attributes), hence, the unit for size is the number of attribute values in a table. Using a six-dimensional data cube results in a maximum value for α=6. Due to similarity in performance between close values of α and for the sake of readability, we are only presenting results where α is set to 0, 2, 4, and 6.

The way we generated the lattice ensures diversity in subcube sizes and significant size difference between a cube and all its dependent cubes. In the generated lattice, cubes at the bottom left area have small sizes while those at top right have larger sizes. This setting will results in 64 (2^6) possible queries.

Derived summary tables are of different sizes, i.e., they have different degrees and cardinalities. In the simulation, we are assuming that attributes values have the same sizes and a data packet capacity is 10 attribute values. The accuracy of the results is dependent on the number of packets needed by each table. In any system with fixed packet size, internal fragmentation is possible. That is, the last data packet of a given table could be partially empty and two tables of different size might fit within the same number of data packets. This means that two tables of different size, possibly one derived from the other one, will incur the same tuning time. Our scheme trades tuning time for wait time and in order to illustrate the maximum possible performance degradation with respect to tuning time that can be exhibited by our proposed flexible algorithms, we decided to ignore any internal fragmentation in our experiments reported in this paper, and hence we did not round up the number of packets. For example, a table of 73 attribute values will consume 7.3 data packets.

To test the system under a typical workload, requests are generated by the clients according to Zipf distribution with the Zipf parameter (θ) default value is equal to 0.8. Queries are sorted according to their size, so that queries to small size tables occur with higher probability than queries to detailed ones. This reflects the fact that typically an analysis session starts by accessing some consolidated data before focusing on more detailed information. We also experimented with a workload in which the table size is not correlated with the table size, reflecting the situation in which users are accessing specific detailed data as a response, for example, to specific market conditions.

We control the simulation by establishing a fixed number of requests, that is, each client was required to complete a certain number of requests before the experiment would terminate. This ensures fairness in reporting, and eliminates any possibility of reporting partial completion data. A client will pose a new request as soon as it gets an answer to its previous one. We also allow for the variability of the number of clients in the client population.

Table 3 summarizes our simulation parameters and settings. The combination of these parameters allows us to examine the scalability of the system as well as the impact of a changing workload on the algorithm performance.

Parameter	Value
Base Cube Dimensionality	6 dimensions
Possible Requests	64 requests
Packet Capacity	10 attributes values
Zipf Parameter (θ)	0.0 – 0.9 (default 0.8)
Simulation Length	100 requests/client
Number of Clients (Request Rate)	10 – 200 clients
Algorithms	SSTF, FCFS, RxW, STOBS-α
α-optimization	0, 2, 4, 6

Table 3: Simulation Parameters

7 Results

For our evaluation, we took extensive performance measurements. The time reported throughout is in broadcast units. Each of these experiments was repeated 5 times to take statistically correct results.

7.1 Average Access Time

In this experimental setting, the number of clients varied between 1 to 200 client, each client poses 100 requests. The variation in the number of clients reflects different request arrival rates. Requests are generated according to the previously mentioned Zipf distribution with θ equals to 0.8.

Figure 4 shows average waiting time for the strict algorithms, namely SSTF, FCFS, RxW and the STOBS-0 (with $\alpha = 0$). The RxW and STOBS exhibit a similar behavior, with the average access time increasing but ultimately stabilizing as the number of clients is increased. This behavior is the normal for broadcast data delivery to clients with shared interests. For the STOBS-0, balancing between all the decision parameters, it achieves an average access time that is 50% less than FCFS and 30% less than RxW in case of 90 clients. This improvement increased to 60% and 35% respectively at a population of 200 clients.

In case of OLAP data cubes delivery, the sharing can be extended beyond the explicit exact similarity of requests to the implicit derivation dependency between the requested summary tables. This is illustrated in Figure 5, where we used different values for the STOBS algorithm α parameter. Setting α to 2, resulted in an average access time 40% less than the strict STOBS in the case of 100 clients, which implies a 60% less than the RxW. Increasing the degree of flexibility to $\alpha = 4$, gave a 60% reduction in access time than STOBS-0 and 75% reduction than RxW for 100 clients and almost the same value in case of 200 clients.

Figures 6 and 7 depicts the tune and wait components of the access time demonstrated in Figure 5. As expected, increasing the α value leads to the increase in tune time as shown in Figure 6. However, that increase was successfully compensated by a decrease in wait time as shown in Figure 7. It is worth mentioning here, that as the requests arrival rate increases, the wait time becomes the dominant factor in the access time computation

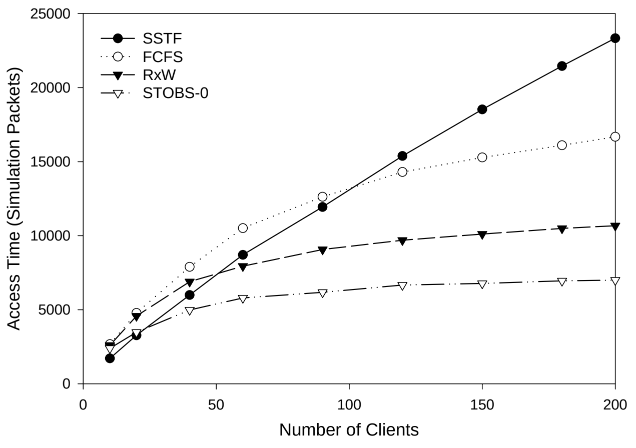

Figure 4: Access Time for strict algorithms

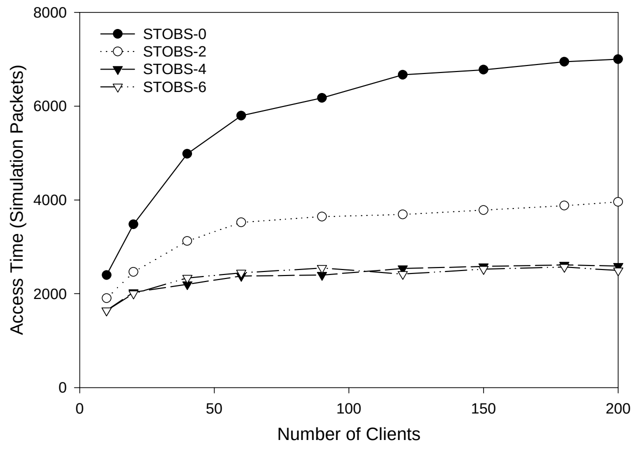

Figure 5: Access Time for *STOBS*-α

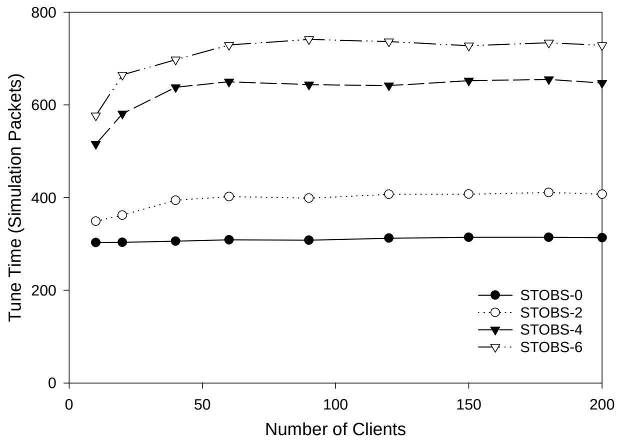

Figure 6: Tune Time for *STOBS*-α

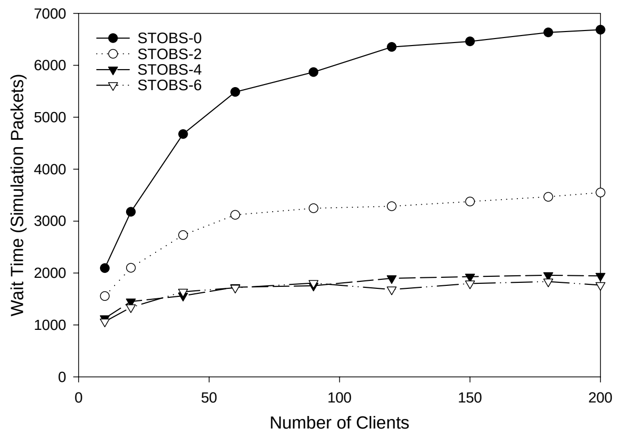

Figure 7: Wait Time for *STOBS*-α

(we can see a 10:1 relation in Figures 6 and 7). This observation supports our idea of tackling the access time reduction by decreasing the wait time even if it yields to a moderate increase in the tune time.

7.2 Fairness

In this experiment, the setting are the same as the previous experiment. However, the metric here is the fairness which is computed as described in Section 3 by calculating the standard deviation of the requests' stretch values. Recall that lower *fairness values* (i.e., values of the standard deviation of stretch) that remain constant across the whole range of request rate indicates better fairness.

The fairness of algorithms is compared in Figure 8. It is clear the unfairness of the FCFS where the decision is biased on the wait time only. Hence, a small request can accumulate high waiting time that can not be absorbed when calculating its stretch. Recalling the experimental setting where there is a high popularity for

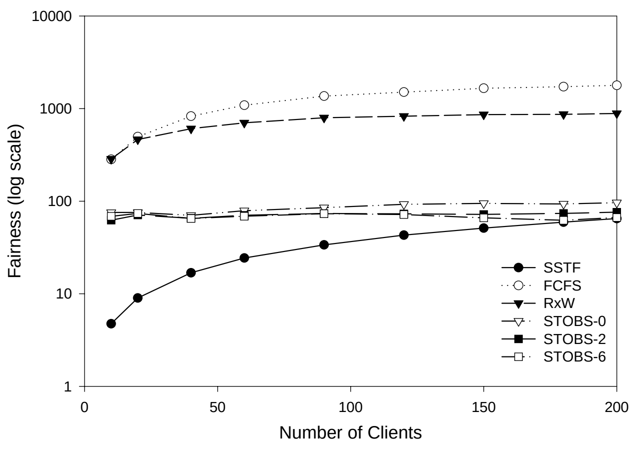

Figure 8: Fairness

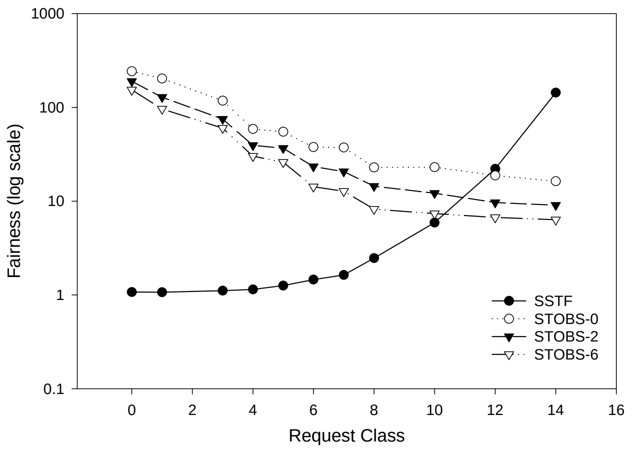

Figure 9: Stretch Per Class

small requests, will explain the improvement of fairness of RxW compared to the FCFS. Interestingly, although SSTF initially exhibits the lowest values of standard deviation of stretch values, it is not the most fair one because of the steady growth of its fairness values. The SSTF fairness degrades as the request rate increases, where the policy is turning down the large requests in favor of the small ones, eventually leading to the starvation of large requests. The STOBS versions exhibit almost a constant value for fairness regardless of the request rate. They are clearly more fair than the FCFS and RxW. Compared to RxW, STOBS-6 is 4 times better in the case of 10 clients and this improvement in fairness increases to 14 times in the case of 200 clients.

In order to get a better insight into the fairness of the STOBS algorithm, we group the requests into classes based on their corresponding table size. As in [3], a request for a table of size between $2^{i-1}+1$ and 2^i (attribute values) belongs to class i. In Figure 9, we have included SSTF as a reference that help in explaining the behavior of STOBS in handling requests of different sizes. The figure magnifies the fairness metric at the point corresponding to workload of 90 clients. The figure shows the obvious biasing of SSTF towards small request and the unfairness towards large ones. In case of STOBS, the service is amortized among the different classes. Increasing the value of α gives more chances and alternative ways for satisfying a single request (especially if it falls within the middle of the class range) which contributes to the improvement in fairness.

7.3 Skewness

In all the previous comparisons, we used the default θ value of 0.8. In this section, we examine the performance for different values of θ, i.e., the degree of skewness of access. Figure 10 shows the average access time for a setting, where the number of clients equals to 100, each posing 100 requests. The Zipf parameter ranges from 0 to 0.9 where for θ= 0, the distribution corresponds to the uniform one.

Since the number of clients (request rate) is kept constant, the increased overlap in client interests allows

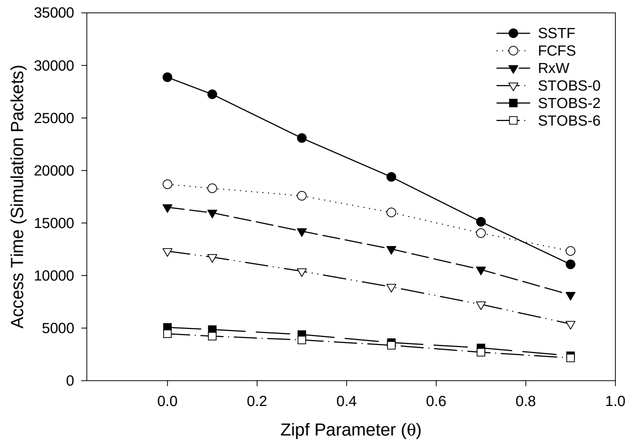

Figure 10: Sensitivity on Low Dimensional Hot spots

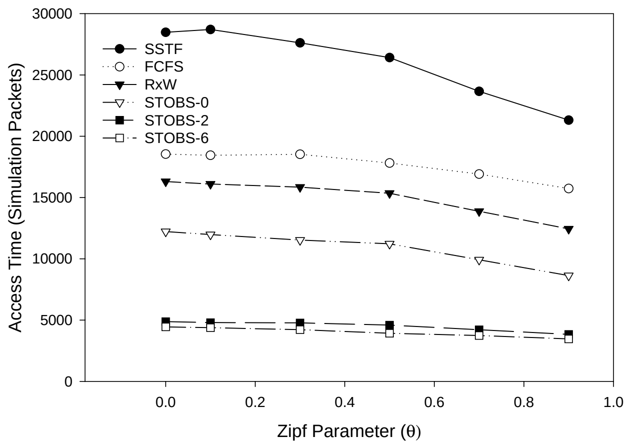

Figure 11: Sensitivity on Random Hot Spots

more efficient use of the broadcast bandwidth. Therefore, as the skew increases all algorithms provide improved reduction in access time. However, the STOBS-2 and STOBS-6 schedulers are also taking advantage of the derivation dependency property between requested tables. Using STOBS-2 reduces the access time by 70% less than the RxW for all θ values.

In Figure 11, we are still varying the value of θ, but we arranged the requests in a random order, so that the small ones are not the popular any more. We can notice that the reduction in access time for the strict algorithms is not as rapid as it is in Figure 10, especially for the SSTF. However, the STOBS-2 and STOBS-6 still maintain the same low average access time, which reached 73% less than RxW in this case.

7.4 STOBS-α and Mobility

An environment where clients are mobile is a special case of the wireless environment, that has even more challenging requirements. Maybe the most important of these requirements is the conservation of energy. Mobile computers operate on batteries which have limited power availability. Hence, power consumption is an issue and its reduction is a performance objective for any proposed algorithm.

Power conservative indexing methods for single-attribute and multi-attribute based queries appeared in [15, 14, 4]. The main idea is, if sufficient indexing information is provided to clients, then the mobile device access pattern to the data stream can alternate between a *doze* mode waiting for data and an *active* mode tuning for required data. In a doze mode the mobile device is consuming power orders of magnitude less than that in the active mode.

Our model supports mobile access. Recall that each packet header contains a *pointer* which is the offset (time step) of the next descriptor packet in the broadcast. After the mobile client poses a request, it starts listening to the downlink channel and tunes to the first descriptor packet. If it can use the up-coming data packets it stays in

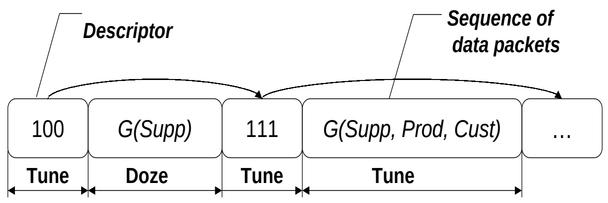

Figure 12: A Client Access to Broadcast

active mode to download the broadcast table. Otherwise, it switches to doze mode reducing power consumption. Using the offset, it wakes up just before the next broadcast cycle (i.e., descriptor packet of the next table on broadcast). When the client wakes up again, it checks the descriptor packet and the process of matching is repeated as shown in Figure 12.

It should be clear that for the FCFS, SSTF, RxW, and STOBS-0 algorithms, the active power is almost the same for all. However, the doze power is a direct consequence of the wait time. So, a reduction in wait time is a reduction in doze power. Using STOBS with $\alpha > 0$ will start the trade-off between the increase in tune time (active power) and the wait time (doze power). In contrast to the access time calculations, where the wait time was the dominant factor, in case of power consumption, the active power is much higher than the doze power.

Figure 13 shows the average consumed power in the active mode when a client is tuned to the broadcast, and in the doze mode when the client is waiting for data to arrive. Figure 14 shows the overall power consumption for the algorithms. We express power in terms of doze mode units assuming that the active:doze ratio to be 20:1 as in the ORiNOCO World PC Card [23] – the power dissipated tuning to one packet is equivalent to that dissipated in dozing for 20 packets transmission time. Here, we used the default experimental settings (θ=0.8).

It is interesting to notice the intersection between the active and doze power consumptions for the STOBS-0 in Figure 13. This means that as the workload increases, the increase in wait time will give more weight to the doze power consumption making its significance equal to that of the active power and even more. Figure 14 shows that 90 STOBS-0 clients will consume on average 20% less power than 90 RxW clients. Setting $\alpha = 2$ achieved a good balance between the active and doze powers, resulting in a reduction of 25% than RxW for 90 clients and 30% for 200 clients.

In order to put these savings in power in perspective, let us consider the practical implications in terms of power units. Consider a wireless LAN, where the broadcast channel has a bandwidth of 1Mbps. Assume each attribute value in our synthesized lattice is of size 10 bytes. And one data packet capacity is 10 attribute values. It will take about 0.8 mSec to broadcast a single packet. Let the clients be equipped with the ORiNOCO World PC Card. The card operates on a 5V power supply, using 9mA at doze mode and 185 mA at receiver mode. Hence, dozing for one packet time will consume 0.8 mSec * 9 mA * 5 V = 36 μJ, while being active tuning to one packet will take 0.8 mSec * 185 mA * 5 V = 740 μJ. To see how much time and energy we are saving, we

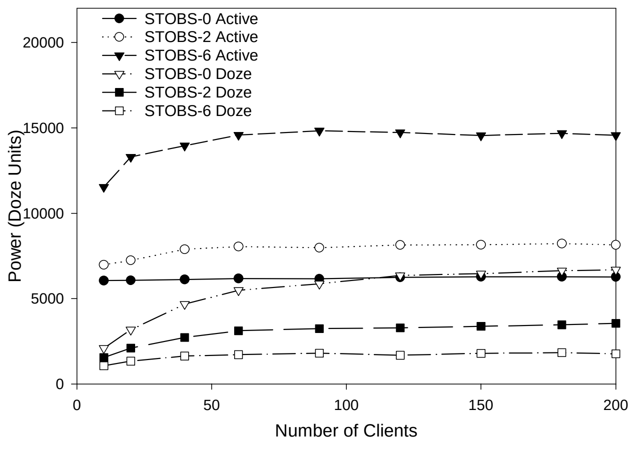

Figure 13: Active and Doze Power

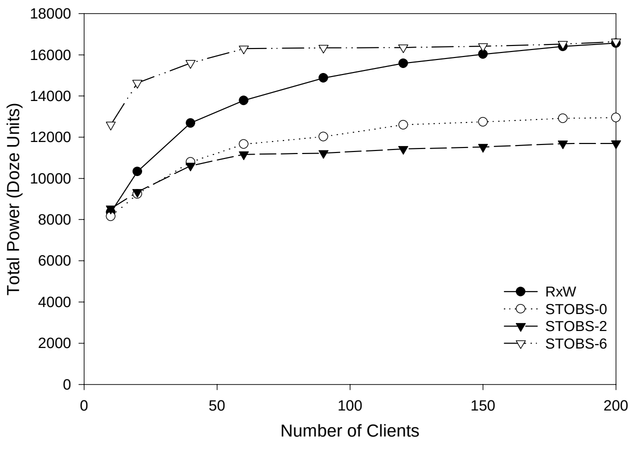

Figure 14: Total Power

are providing a practical numeric comparison between the various algorithms in Table 4. The table summarizes the average access time and the average energy consumption per query for population of 90 clients.

Algorithm	Average Access Time (Secs)	Average Energy Consumption (Joules)
SSTF	9.5	0.64
FCFS	10.1	0.66
RxW	7.2	0.53
STOBS-0	4.9	0.43
STOBS-2	2.9	0.40

Table 4: Practical Results

8 Conclusions

In this paper, we discussed the new challenge of efficient support of mobile decision making. Towards this, we re-emphasized the role of broadcast based data dissemination in supporting efficient access of enterprise data warehouse and consequently enabling good decision making anytime and anywhere. Although the emphasis of our paper was on wireless and mobile computing environments, our results are applicable in wired networks which support multicasting.

More specifically, this paper has made four contributions in the context of on-demand broadcast scheduling:

- It identified the new possibility of request aggregation based on the derivation dependencies among summary tables rather than just based on the exact match of requests of all the current approaches.

- It classified on-demand scheduling algorithms into *strict* and *flexible* based on their ability to broadcast subsuming tables in respond to a given request.

- It proposed a family of heuristics called called *STOBS-α*. The α-optimization parameter controls the degree of flexibility in using available subsuming tables, which provided further reductions in access time and dissipated power. The superiority of the STOBS-α was demonstrated experimentally using simulation.

- It provided a comprehensive evaluation of existing on-demand scheduling algorithms in terms of their suitability to support the dissemination of OLAP summary tables.

In summary, the basic STOBS (α=0) balances the trade-off between all the scheduling parameters resulting in significant reductions in average access time and power consumption for mobile clients. Further, our experiments showed that experiencing a moderate degree of flexibility resulted in a good trade-off between the decrease in wait time and the increase in tune time. This is specially important in case of mobile users, where the former corresponds to low doze power whereas the latter to high active power.

We are currently working on techniques to integrate the flexibility with the scheduling decision. We are also studying the problem in a push subscribe environment. We are planning to investigate the effect of deploying caching at the client side and what will be the appropriate caching mechanisms.

Acknowledgments: The authors would like to thank Karl Gossett for his help with the initial phase of this work. They would also like to thank the anonymous reviewers for their very helpful comments.

References

[1] S.Acharya, R. Alonso, M. Franklin and S. Zdonik. Broadcast Disks: Data Management for Asymmetric Communication Environments. *Proc. of the ACM SIGMOD Conf.*, pp. 199-210, May 1995.

[2] S. Acharya, M.Franklin, and S.Zdonik. Balancing Push and Pull for Data Broadcast. *Proc. of the ACM SIGMOD Conf.*, pp. 183-194, May 1997.

[3] S. Acharya and S. Muthukrishnan Scheduling On-demand Broadcasts: New Metrics and Algorithms. *Proc. of Fourth Annual ACM/IEEE Conf. MobiCom*, pp. 43-54, October 1998.

[4] R. Agrawal and P. K. Chrysanthis. Efficient Data Dissemination to Mobile Clients in E-Commerce Applications. *Proc. of the 3rd Int'l Workshop on E-Commerce and Web Information Systems*, pp. 58-65, June 2001.

[5] D. Aksoy and M. Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Transactions On Networking*, 7(6):846-860, December 1999.

[6] Y. Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. *SODA*, pp. 558-559, 2000.

[7] E.F. Codd. Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate. E.F. Codd and Associates, 1993.

[8] A. Crespo, O. Buyukkokten, and H. G. Molina. Efficient Query Subscription Processing in a Multicast Environment (Extended Abstract). *Proc. of the 16th ICDE Conf.*, February 2000.

[9] H.D. Dykeman, M. Ammar, and J.W. Wong. Scheduling Algorithms for Videotex Systems Under Broadcast Delivery. *Proc. of the 1986 Int'l Conf. on Communications*, pp. 1847-1851, June 1986.

[10] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D.Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals. *Proc. of the ICDE Conf.*, pp. 152-159, February 1996.

[11] H. Gupta. Selection of Views to Materialize in a Data Warehouse. *Proc. of ICDT*, pp. 98-112, January 1997.

[12] H. Gupta, V. Harinarayan, A. Rajaraman, and J.D. Ullman. Index Selection for OLAP. *Proc. of the ICDE Conf.*, pp. 208-219, April 1997.

[13] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing Data Cubes Efficiently. *Proc. of the ACM SIGMOD Conf.*, pp. 205-216, June 1996.

[14] Q. Hu, W.-C. Lee, and D.L. Lee. Power Conservative Multi-Attribute Queries on Data Broadcast. *Proc. of the 16th ICDE Conf.*, pp. 157-166, 2000.

[15] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy Efficient Indexing on Air. *Proc. of the ACM SIGMOD Conf.*, pp. 25-36, May 1994.

[16] P. Kalnis, N. Mamoulis, and D. Papadias. View Selection Using Randomized Search. *DKE*, 42(1): 89-111, 2002.

[17] B. Kalyanasundram, K.R. Pruhs, and M. Velauthapillai. Scheduling Broadcasts in Wireless Networks. *ESA*, pp. 290-301, 2000.

[18] R. Kimball. The Data Warehouse Toolkit. John Wiley, 1996.

[19] Y. Kotidis, N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses *Proc. of the ACM SIGMOD Conf.*, pp. 371-382, June 1999

[20] S. Weissman Lauzac and P. K. Chrysanthis. Utilizing Versions of Views within a Mobile Environment. *Proc. of the 9th Int'l Conf. on Computing and Information*, pp. 201-208, June 1998.

[21] S. Weissman Lauzac and P. K. Chrysanthis. Programming Views for Mobile Database Clients. *Proc. of the 9th Int'l Workshop on Mobility in Databases and Distributed Systems*, pp. 408-413, August 1998.

[22] M. Mehta, and D. J. DeWitt. Dynamic Memory Allocation for Multiple-Query Workloads. *Proc. of the VLDB Conf.*, pp. 354-367, August 1993.

[23] www.orinocowireless.com

[24] T. Mudge. Power: A first class design constraint. *Computer*, 34(4):52–57, 2001.

[25] P. Scheurmann, J. Shim, and R. Vingralek. WATCHMAN: A Data Warehouse Intelligent Cache Manager. *Proc. of the 22nd VLDB Conf.*, pp. 51-62, September 1996.

[26] K. Stathatos, N. Roussopoulos, and J.S. Baras. Adaptive Data Broadcast in Hybrid Networks. *Proc. of the 23rd VLDB Conf.*, pp. 326-335, August 1997.

[27] K.-L. Tan and J.X. Yu. Generating Broadcast Programs that Support Range Queries. *IEEE TKDE*, 10(4):668-672, 1998.

[28] N. H. Vaidya and S. Hameed. Scheduling Data Broadcast In Asymmetric Communication Environments. *ACM/Baltzer Wireless Networks*, 5(3):171-182, 1999.

[29] J.W. Wong. Broadcast delivery. *Proc. of the IEEE*, 76:1566-1577 , 1988.