Efficient Dissemination of Aggregate Data over the Wireless Web *

Mohamed A. Sharaf
University of Pittsburgh
msharaf@cs.pitt.edu

Yannis Sismanis
University of Maryland
isis@cs.umd.edu

Alexandros Labrinidis
University of Pittsburgh
labrinid@cs.pitt.edu

Panos K. Chrysanthis
University of Pittsburgh
panos@cs.pitt.edu

Nick Roussopoulos
University of Maryland
nick@cs.umd.edu

ABSTRACT

The proliferation of wireless technologies along with the large volume of data available online are forcing us to rethink existing data dissemination techniques for data over the Web, and in particular for aggregate data. In addition to scalability and response time, data delivery to mobile clients with wireless Web connectivity must also consider energy consumption. In this work, we present a hybrid scheduling algorithm (DV-ES) for broadcast-based data delivery of aggregate data over the wireless Web. Our algorithm efficiently "packs" aggregate data for broadcast delivery and utilizes view subsumption at the mobile client, which allow for faster response times and lower energy consumption.

1. INTRODUCTION

Undoubtedly, the Web has brought massive amounts of information to everyone's fingertips, changing forever the way we learn the news, perform research, do business, etc.. The proliferation of wireless technologies, combined with the emergence of web services standards, will enable a new class of applications that empower mobile users to access great amounts of data in a seamless and efficient manner. We collectively refer to such infrastructure as the *"wireless Web"*.

There are many motivating applications for the wireless Web. In this work, we are focusing on applications that utilize aggregate, summarized data. Mobile decision making is one such application, in which executives have access to time-sensitive, business-critical data from their mobile, hand-held devices. Scientists on the field would also rely on similar wireless Web technologies in order to access historic data for their research or to be able to retrieve summarized sensor measurements. We expect that the ease of deployment of wireless Web technologies will also change existing applications (currently running on stationary, wired environments) to take advantage of the mobility it provides.

The ease of deployment and the high mobility of the wireless Web users will provide new opportunities but also bring forth limitations. On the one hand, using the wireless data communication medium allows the use of broadcast/multicast schemes which have better scalability than traditional unicast communication in disseminating data. On the other hand, being mobile poses energy consumption considerations in addition to the traditional response time improvement goals.

In this paper, we are addressing the issue of time and energy efficient delivery of aggregate data to mobile clients over the wireless Web. In wireless and mobile networks, broadcasting is the primary mode of operation for the physical layer. This means that if multiple clients request the same data at approximately the same time, the server may aggregate these requests, and only broadcast the data once. In this way, the low bandwidth of wireless link is better utilized, clearly improving user perceived performance. Several scheduling algorithms have been proposed that attempt to achieve maximum aggregation of requests [1, 3, 11, 2, 10].

In a decision making 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 aggregate data. 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., *supplier*, *product*, *customer*, *time* or *region*).

Decision making queries typically operate on aggregate data (i.e. summarized, consolidated data), derived from fact tables. The needed data can be derived using the *data cube* operator [4]. 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 views. Given that the data cube is an expensive operator, often views are pre-computed and stored at the server as *materialized views*. Basically, a view is an aggregation query, where the dimensions for analysis are the *Group-By* attributes and the measures of interest are the aggregation attributes.

In the case of queries to aggregate data, there is an interesting property which may exist between two queries: a

*This work is supported in part by NSF award ANI-0123705 and in part by the DoD-Army Research Office under Award No. DAAD19-01-1-0494. Prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

International Workshop on the Web and Databases (WebDB).
June 12–13, 2003, San Diego, California.

set of aggregate data V^d requested by a client may be used to derive the set of aggregate data V^a requested by another client. This happens when V^d corresponds to a detailed version of V^a, which is more abstract. In such a case, V^a has a derivation dependency on V^d, and V^d subsumes V^a.

In earlier work [7, 8], we have exploited the above *subsumption* property by considering aggregate data as views, to increase sharing among clients and go beyond the exact matching of requests, thus allowing for further reduction in access time and power consumption. In this paper, we investigate another technique which also embodies the subsumption property to reduce the size of the broadcast. Specifically, we utilize the Dwarf technology to compress a view and all its subordinate views in order to achieve aggregation of multiple requests. Further, we combine the Dwarf-based scheme with the View-based aggregation to achieve further broadcast efficiency and scalability.

The rest of this paper is organized as follows. In the next section, we introduce a model to support aggregate data dissemination over the wireless Web. Overviews of the related work on dwarf and subsumption based scheduling are presented in Sections 3 and 4, respectively. In Section 5, we present *DV-ES*, our hybrid on-demand scheduling algorithm. Our simulation testbed and experiments are presented in Sections 6.

2. SYSTEM ARCHITECTURE

Our assumed architecture is based on *broadcast pull*. The Web server is responsible for maintaining and disseminating the aggregate data. A client sends a request for aggregate data on the *uplink channel* and then listens to the downlink channel for a response. A client can be in one of two modes, either in *active* mode, tuning and listening to the broadcast, or in *doze* mode, where the client is idle waiting for the response, with the receiver switched off [5].

A client request G is characterized by the set of its Group-By attributes A and measure attributes M. Hence, we represent a request as $G^{A,M}$ and the corresponding view as $V^{A,M}$. Without loss of generality, in this paper we are assuming only one measure attribute, hence we can drop the M part from the definition and use V^A to fully describe a view. Under this definition, a view V^{A1} *subsumes* view V^{A2}, if $A2 \subseteq A1$. In this case, V^{A2} is *dependent* on V^{A1}. We denote the number of dimensional attributes in the set A as $|A|$ and the size of view V^A in bytes as $|V^A|$.

The smallest logical unit of a broadcast is called a *packet* or *bucket*. A broadcast view 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 packets. The descriptor packet contains a view descriptor which semantically identifies the aggregation dimensions, the number of attribute values or tuples in the view and the number of data packets accommodating that view.

We use bit encoding to represent both the semantics of a 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 corresponds to one of the dimensions $a_1, a_2, ..., a_n$. If view V^A has dimension a_x ($a_x \in A$), then the bit at position x is set to 1, otherwise it is a zero.

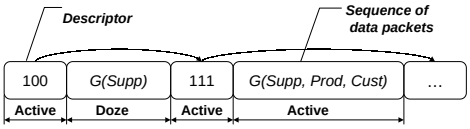


Figure 1: A Client Access to Broadcast

After a client submits a request for view V^R on the uplink channel, it follows a three-phase access protocol: (1) *initial probe*; (2) *semantic matching*; and (3) *view retrieval*.

In the initial probe phase, the client tunes to the downlink channel and uses the nearest packet header to locate the next descriptor packet. The semantic matching phase starts when the client finds the first descriptor packet, say for view V^B. In this phase, the client will "detect" whether view V^B satisfies its request exactly or whether the answer can instead be "inferred" from view V^B.

Depending on the matching result and the scheduling algorithm used (as we will see below), the client will either switch to the final retrieval phase or it will stay in the matching phase. In the former, the client stays in active mode tuning to the next sequence of data packets to read (download) view V^B. In the latter case, it will switch to doze mode reducing power consumption. Using the offset in the packet header, the client wakes up just before the next opportunity to read the next descriptor packet in the broadcast, after which, the semantic matching process is repeated. The access protocol is shown in Figure 1.

2.1 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 *wait time* and *tune time*.
- **Tune Time:** It is the time spent by the client listening to the downlink channel either reading a descriptor packet or data packets containing the requested view. During tuning, the client is in active mode.
- **Wait Time:** The total time a client spends waiting between descriptor packets until it finds a matching one. A client is in doze mode during the wait time.

In active mode, a client device consumes energy that is orders of magnitude higher than in doze mode. For this reason, tune time has been traditionally used to evaluate the power consumption of a system in a mobile environment. However, the energy dissipated in doze mode becomes more significant when the client has to wait for a long time until its request is satisfied. Hence, in this paper we will adopt a weighted energy consumption cost model that includes the active and doze factors.

3. DWARF TECHNOLOGY

The *Dwarf Structure* [9] is a highly compressed structure for computing, storing and querying data cubes. It efficiently addresses both the storage space problem and the computation cost problem by factoring out prefix and suffix redundancies. Each redundancy is identified *prior* to its computation which results in significant computational

savings during creation. To help clarify the nature of the redundancies let us consider a cube with three dimensions a, b and c.

Prefix redundancy occurs when two Group-Bys share a common prefix (like Group-Bys abc and ab). For example, in the fact table shown in Table 1, store $S1$ appears a total of seven times in the corresponding cube and more specifically in the Group-Bys: $\langle S1, C2, P2\rangle$, $\langle S1, C3, P1\rangle$, $\langle S1, C2\rangle$, $\langle S1, C3\rangle$, $\langle S1, P1\rangle$, $\langle S1, P2\rangle$ and $\langle S1\rangle$. The same happens with prefixes of size greater than one. This kind of prefix redundancy is factored out in the Dwarf Structure by storing each unique prefix just once. Dense areas of the cube benefit more by eliminating prefix redundancies, since there is no need to repeat the same values of the dimensions over all possible Group-Bys. It has been shown that for very dense cubes the corresponding dwarf is much less than the fact table.

Store	Customer	Product	Price
S1	C2	P2	$70
S1	C3	P1	$40
S2	C1	P1	$90
S2	C1	P2	$50

Table 1: Fact Table for cube Sales

Suffix redundancy occurs similarly when two Group-Bys share a common suffix (like Group-Bys abc and bc). For example, for the fact table in Table 1, since customer $C1$ shops only from store $S2$ then, for any value x of dimension `Product`, the Group-Bys $\langle S2, C1, x\rangle$ and $\langle C1, x\rangle$ always have the same aggregate values. This happens because the $\langle C1, x\rangle$ Group-By aggregates all the tuples of the fact table with customer $C1$ for any combination of stores. In this case however there is only one store $S2$ and the aggregation degenerates to the aggregation already performed for the Group-By $\langle S2, C1, x\rangle$. Such redundancies are very often in sparse datasets and even more apparent in cases of correlated dimension values – which are very common in real datasets. Suffix redundancies are factored out in the Dwarf Structure by *coalescing* the space of the corresponding suffixes. It has been demonstrated that the space required to store the Dwarf Structure for sparse datasets is many times smaller than that required by alternative techniques such as materialized views.

The Dwarf Structure is self-sufficient since it does need to access or reference the fact table in order to answer for any of the views of the cube and it provides an implicit index mechanism without needing any additional index for querying it. In addition any query can be answered with just one scan/traversal over the dwarf structure.

Sub-Dwarfs The intuition of using the Dwarf as a compressed broadcast unit can be depicted in Figure 2. For a four-dimensional cube of uniform data and a cardinality of one hundred for each dimension, we enumerate all possible views using the binary representation described in Section 2. For each view, the size of the materialized view (in bytes) along with the size of the corresponding *sub-dwarf* is depicted. The sub-dwarf corresponds to the dwarf that is having the corresponding view as fact table (i.e, the original fact table where all the dimensions, that are represented by 0 in the view binary representation, are projected out). The materialized view is an unindexed relation that holds all the

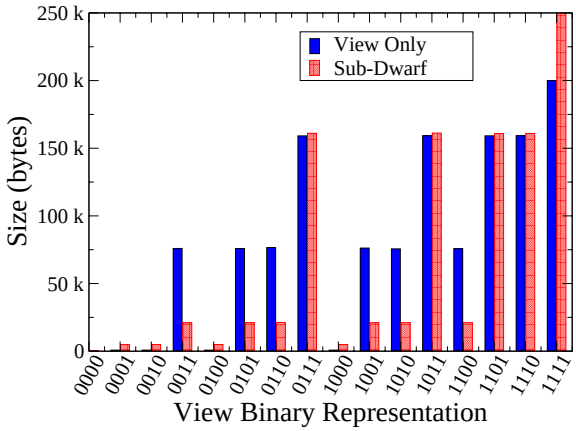


Figure 2: Sub-Dwarfs vs Materialized Views

tuples of the corresponding view. For this specific case we assumed that the values of each dimension are mapped to four-byte integers and that there is one measure represented by a float (four-byte) value.

It is important to point out that while the materialized view contains only the tuples of a single Group-By query, the corresponding sub-dwarf contains *all subordinate*[1] views. For example, for view 0110, the materialized view only holds the tuples of the view 0110, while the sub-dwarf contains all subordinate views 0000, 0010, 0100 and 0110. The subsumption operation in the sub-dwarf case degenerates to just a simple scan over the dwarf without any online aggregation.

The basic idea is that in many cases it is better to broadcast a sub-dwarf instead of the view since its size is much smaller than the size of the sum of the subordinate views. In addition, for dense views, the sub-dwarf is much smaller than the corresponding materialized itself making it "economic" to transmit even if there are no request for the subordinate views.

We observe that for views 0011, 0101, 0110, 1001, 1010 and 1100 the size of the sub-dwarf is much smaller than the size of the corresponding materialized view. The reason is that these views are quite dense since they contain only two dimensions and the savings by factoring out prefix redundancies is tremendous. For views 0111, 1011, 1101 and 1110 (i.e., all views with three dimensions) we observe that the size of the dwarf is slight larger than the corresponding view. In cases where the request workload accesses a lot of their subordinate views it could be beneficial to broadcast the sub-dwarfs instead of the corresponding views. Sub-Dwarfs of just one dimension (i.e, 0001, 0010, 0100 and 1000) are bigger than the corresponding materialized views due to various implementation overheads. The sub-Dwarf that corresponds to 1111 corresponds actually to the full cube since it contains all possible subordinate views.

4. SUBSUMPTION BASED SCHEDULER

In general, accesses for aggregate data have the following properties: *heterogeneity* (views can be of different dimensionality and of various sizes), *skewed access* (requests from clients usually form a hot spot within the data cube), and *subsumption* (it is often possible to use one detailed view to

[1] views that can be computed from the fact table.

compute other summarized ones).

In previous work [7], we developed (*SBS-α*), the first on-demand broadcast scheduling algorithm that exploits the subsumption property of summary tables to increase sharing among clients. The *Subsumption-Based Scheduler* (*SBS-α*) consists of two components: A basic selection component, which captures the heterogeneity and skewed access properties and the *α-optimizing* component that exploits the subsumption property to reduce access time with minimum extra overhead in terms of energy.

SBS-α uses *Longest Total Stretch First* (LTSF) algorithm [1] as its basic selection component, where the stretch of a pending request is the ratio of the time the request has been in the system thus far to its service time. Hence, the stretch of request i for view V^X can be represented as: $\frac{W_i^X}{|V^X|/B}$, where W_i^X is the wait time of request i, $|V^X|$ is the size of V^X, and B is the bandwidth. Since the bandwidth is constant we can re-write the stretch as $\frac{W_i^X}{|V^X|}$.

In *SBS-α*, the server queues up the clients' requests as they arrive. When it is time for the server to make a decision which view to broadcast next, it computes the total stretch value for each view that has at least one outstanding request. The view with the highest total stretch value (say V^b) is selected to be broadcast, as in [1].

Ignoring the subsumption semantic of data cubes, a client that requested V^r, which is derivable from V^b, will wait until V^r is broadcast even if V^b is available sooner. In the extreme case of utilizing the subsumption property, the client will use V^b to derive V^r regardless of the relative cardinality of both tables and potentially incurring unnecessary extra energy. This increase energy is due to the extra time a client has to spend tuning to a detailed version of the aggregate rather than a summarized one and the accompanying high energy consumed in the active mode. For that reason, parameter α is used to define the degree of flexibility in using the subsumption property (that captures the degree of sharing) and it works as follows.

At the server side, upon deciding the broadcast of view V^b, the server discards every pending request for a view V^r that can be derived from V^b and satisfies the following property (α *rule*): The ratio of the difference in size between views V^b and V^r to view V^b is less than the α value. Formally, V^r can be discarded and is not broadcast, iff V^b is broadcast, $r \subset b$ and $\frac{|V^b|-|V^r|}{|V^b|} \leq \alpha$, where $\alpha \in [0,1]$.

At $\alpha = 0$ there is no flexibility in using views; the client access is restricted to exact match and SBS-0 is equivalent to LTSF. At $\alpha = 1$, the case of extreme flexibility, a client can use any subsuming matching view. Picking a reasonable value for α will balance the trade-off between reducing the wait time (doze energy consumption) and increasing the tune time (active energy consumption).

5. PROPOSED HYBRID ALGORITHM

We propose to integrate the two presented technologies (Dwarf and SBS-α) into a hybrid on-demand broadcast scheduling algorithm that employs view subsumption (similarly to SBS-α) and uses Dwarf cubes as a compact representation form for aggregate data. We named our hybrid algorithm *DV-ES(α)* to reflect the objects that can be included on the broadcast by the server (V for Views, or D for Dwarfs), and the operations that a client can perform (S for subsumption, or E for extraction from the Dwarf).

Under DV-ES(α), the server can broadcast either entire dwarf sub-cubes or plain views for a given set of dimensions (i.e., for a given Group-By query). Accordingly, a client may receive one of the following:

- the view that exactly matches its query, or
- a detailed view which subsumes the one it originally requested, or
- a dwarf sub-cube that contains the view.

As an example to illustrate the above cases, consider a client request for view (A, B). The server can serve this request by either broadcasting a view or a dwarf. In the case of sending a view, it can either be the exact view (i.e., (A, B)), or an ancestor view (e.g., (A, B, C)). The same applies in the case of broadcasting a dwarf. That is, it can be either the dwarf corresponding to (A, B), or a dwarf corresponding to an ancestor view, say (A, B, C), which will physically contain view (A, B) as well as all possible combination of views that can be derived from (A, B, C).

In the case where the server disseminates a dwarf sub-cube D^X, the client will download D^X and extract its requested view. Note that D^X will contain V^X and all the views that are descendant from V^X, hence, a client can use D^X to either extract V^X or any other views that is dependent on V^X. On the other hand, if the server decided to disseminate V^X rather than D^X, then a client will still be able derive any view that is subsumed by V^X.

The server decides whether to broadcast a view or dwarf based on the sizes: if the size of the view is less than the smallest dwarf that contains this view, then the view is selected, otherwise the corresponding dwarf sub-cube is selected. The intuition is to save bandwidth in the cases where the size of the dwarf is larger than the corresponding view.

DV-ES(α) is similar to SBS-α, consisting of an LTSF component for scheduling and the α-optimizing component to exploit the dependency between views. However, the implementation of these component is slightly different.

In DV-ES(α), the LTSF scheduling component is modified to consider the two different representations of broadcast objects, namely dwarfs or views, when selecting the object to be broadcast. Specifically, since the object representation that yields the smaller size is selected, the stretch of a request i for view V^X is redefined as $\frac{W_i}{min(|D^X|,|V^X|)}$ where $|D^X|$ is the size of the dwarf corresponding to V^X.

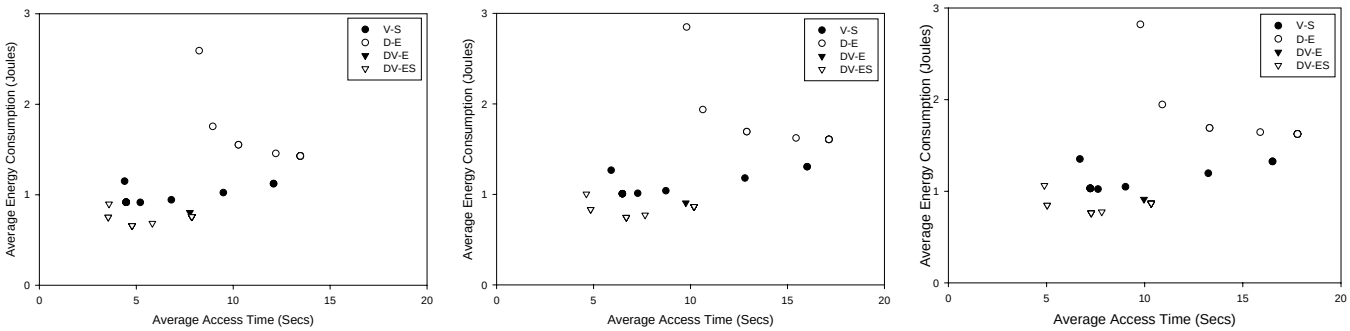
Also, the α optimizing component is modified to incorporate the fact that a request for V^r can be satisfied either by sending a view or by sending the equivalent dwarf (whichever is smaller in size). The α rule is redefined as follows:

if selection = dwarf then
  if $\frac{|D^b|-min(|D^r|,|V^r|)}{|D^b|} \leq \alpha$ *then eliminate* G^r
else if selection = view
  if $\frac{|V^b|-min(|D^r|,|V^r|)}{|V^b|} \leq \alpha$ *then eliminate* G^r

where G^r is the request for V^r.

One way that a client could decide whether or not to use a particular object on the broadcast is by applying the same α-rule used at the server. However, this will require a client to have full information about the sizes of all views and their corresponding dwarfs.

We adopted an alternative way which assumes that clients have no prior knowledge about the sizes of views and dwarfs.



(a) 50 clients (b) 250 clients (c) 500 clients

Figure 3: Energy vs Time for all scheduling algorithms

Instead, the server explicitly informs the clients which request can be satisfied by a given view or dwarf on the broadcast. This is achieved by augmenting each descriptor packet with a views encoding bitmap. The length of this bitmap is equal to the number of possible views. If a view can be inferred from the broadcast object and it satisfies the α-rule, the bit corresponding to this view is set to 1, otherwise it is set to 0. This approach has a negligible overhead in terms of bits added to the descriptor packet and it only requires a client's knowledge of the multidimensional schema at the server which is already part of the meta-data information periodically broadcast by the server.

6. EVALUATION

We implemented a simulator to evaluate the performance of our proposed *DV-ES(α)*.

6.1 Experimental Setup

Algorithms In addition to the SBS-α, in our evaluation we included two "base-case" algorithms in order to better understand the behavior of DV-ES(α). In the first algorithm, the server always broadcasts packed dwarf cubes, whereas in the second algorithm we emulate the hybrid scheduling algorithm, but remove the client's subsumption ability.

For clarity in presentation, we name these algorithms with the X-Y scheme that we used to name our proposed **DV-ES(α)** algorithm, where X stands for the data objects that can be included in the broadcast (V for Views, or D for Dwarfs) and Y is the operations that clients can perform (S for subsumption, or E for extraction from the Dwarf). Under this scheme, the SBS-α algorithm is named **V-S(α)**, the first base case algorithm is named **D-E(α)**, and the second base case algorithm is named **DV-E(α)**. Parameter α controls the degree of flexibility in deriving views or extracting them from a higher dimensional ancestor.

Workload We present experiments results using synthetic datasets. The base fact table has uniformly distributed data, where the default number of tuples is 100,000 and the default number of dimensions is 6 (we used a cardinality of one hundred for each dimension).

To test the system under a realistic workload, requests are generated by the clients according to Zipf distribution with the Zipf parameter θ=0.5. Queries are sorted according to their size, so that queries to small size views occur with higher probability than queries to detailed ones.

Time is reported in *Seconds* and the energy consumption is in *Joules*. We considered a wireless LAN where the broadcast channel has a bandwidth of 10 Mbps. Clients are equipped with the ORiNOCO World PC Card [6]. The card operates on a 5V power supply, using 9mA at doze mode and 185 mA at receiver, active mode.

6.2 Experiments

Figure 3 Figure 3 depicts the average access time and energy consumption for the different algorithms for values of α between 0.0 and 1.0 with 0.1 steps. For all algorithms, the access time decreases by increasing the value of α. So, for a certain algorithm, traversing the points from right to left corresponds to increasing the value of α.

Figure 3 shows the poor performance of the D-E algorithm, where only dwarfs are disseminated. DV-ES will always outperform D-E since in the cases where the dwarf size is bigger than the corresponding view, it broadcasts the view. Compared to the other algorithms, D-E will perform better only when the access is skewed toward aggregates whose dwarf representation is smaller than the corresponding view. This clearly is not the case in the given workload as shown in Figure 3.

We also noticed that the performance of DV-E for values of α from 0.0 to 0.9 is similar to that of DV-ES(0) (coinciding points in the graph). This shows that DV-E is almost insensitive to the parameter α. The explanation is that under the DV-E algorithm: 1) a dwarf is selected if its size is smaller than the corresponding view, and 2) a client extracts a view from this dwarf if the view minimum size representation is within an α distance from the selected dwarf. Given our workload, the first condition leads to selecting dwarfs to satisfy medium dimensionality queries, while selecting views to satisfy low and high dimensionality queries. However, most of the time these medium dimensionality dwarfs are orders of magnitude larger in size than the view version of their descendants. Hence, it requires a high value of α in order for extraction to take place (i.e., α=1), which is the single point presented in Figure 3.

Finally, it is the clear that the DV-ES algorithm outperforms V-S in both time and energy reductions. DV-ES utilizes the cases where the dwarf is smaller than the corresponding view and it allows for both extraction and subsumption. This improvement over V-S is more significant as the request rate increases (i.e., increasing the number of

clients from 50 to 500 as in cases (a), (b), and (c) in Figure 3), which demonstrates the scalability of DV-ES.

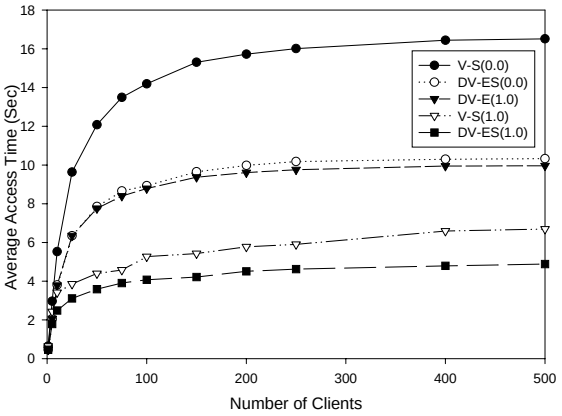


Figure 4: Access Time

Figure 4 We plot the response time of V-S, DV-E, and DV-ES in Figure 4 where the number of clients ranges from 50 to 500 and for values of $\alpha = 0$ and 1. All algorithms exhibit similar behavior: the average access time increases, but ultimately levels as the number of clients is increased. This behavior is normal for broadcast data delivery to clients with shared interests.

Figure 4 shows how the access time decreases with increasing α from 0 to 1: $\alpha = 1$ is the case where flexibility in using ancestors is experienced the most, and $\alpha = 0$ is the case where we do not allow using ancestors. It also shows that a significant reduction in access time is achieved by DV-ES compared to V-S, which is even more significant as the load increases. For instance, consider the cases of 50 and 500 clients where $\alpha = 1$. In the case of 50 clients, the access time decreased by 22% compared to V-S(1), while in the case of 500 clients, the reduction achieved by DV-ES(1) was 37% compared to V-S(1) and it is 3 times less than V-S(0).

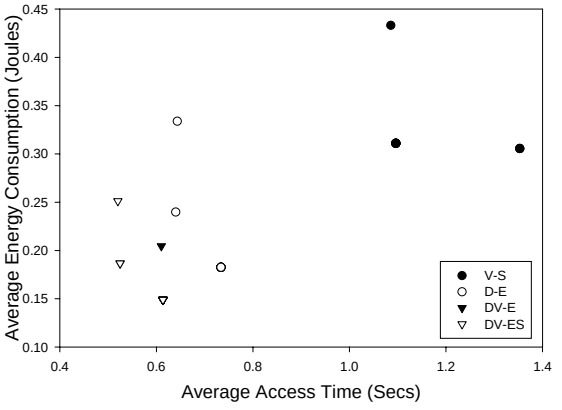


Figure 5: 4-dimension workload

Figure 5 In the previous comparisons we used a fact table of 6 dimensions, whereas in Figure 5 we are experimenting with a fact table that has denser views (by decreasing the number of dimensions to 4, while keeping the cardinality of one hundred for each dimension and the number of tuples as the default 100K). This has the effect of generating fewer aggregate views than the 6-dimensions case, but they are denser. As shown in Figure 5, for a load of 500 clients, all the dwarf-based algorithms (D-E, DV-E, DV-ES) are performing better than V-S, where only views are disseminated. Comparing the performance of DV-ES(1) and V-S(1), we can see that DV-ES(1) outperforms V-S(1) in both access time and energy consumption. For access time, DV-ES(1) achieved a reduction equal to 50% compared to V-S(1), whereas this reduction was only 37% in the case of 6-dimensional fact table shown in Figure 3(c). For energy consumption, the reduction is 40%, as opposed to only 22% in the case of 6-dimensional fact table.

7. CONCLUSIONS

In this paper we proposed and evaluated *DV-ES*, a new on-demand broadcast scheduling algorithm for disseminating aggregated data over the wireless Web. DV-ES integrates the view-derivation properties of SBS-α that goes beyond the exact match of requests and the Dwarf technology that provides a compact represenation for aggregate data. DV-ES achieves reduction both in access time and power consumption by selecting at any given scheduling point to broadcast either a View to be used by the clients to derive their requested data or an entire sub-cube from which clients extract their requested data.

8. REFERENCES

[1] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. *Proc. of 4th ACM/IEEE MobiCom Conf.*, Oct. 1998.

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

[3] 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.

[4] J. Gray, et. al. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals. *Proc. of the ICDE Conf.*, Feb. 1996.

[5] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. *Proc. of the ACM SIGMOD Conf.*, pp. 25–36, May 1994.

[6] ORiNOCO World PC Card. www.orinocowireless.com

[7] M. A. Sharaf and P. K. Chrysanthis. Semantic-based delivery of OLAP summary tables in wireless environments. *Proc. of the CIKM Conf.*, Nov. 2002.

[8] M. A. Sharaf and P. K. Chrysanthis. Facilitating Mobile Decision Making. *Proc. of the 2nd ACM Int'l Workshop on Mobile Commerce*, Sep. 2002.

[9] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, Y. Kotidis Dwarf: shrinking the PetaCube. *Proc. of ACM SIGMOD Conf.*, June 2002.

[10] K. Stathatos, N. Roussopoulos, and J.S. Baras. Adaptive data broadcast in hybrid networks. *The VLDB Journal*, pp. 326–335, 1997.

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