Energy-Efficient Processing of Complex Queries over a Wireless Broadcast Data Stream

Yon Dohn CHUNG†, Member and Chang-Sup PARK††, Nonmember

SUMMARY  Energy-efficiency is one of the main concerns in the wireless information dissemination system. This paper presents a wireless broadcast stream organization scheme which enables complex queries (e.g., aggregation queries) to be processed in an energy-efficient way. For efficient processing of complex queries, we propose an approach of broadcasting their pre-computed results with the data stream, wherein the way of replication of index and pre-computation results are investigated. Through analysis and experiments, we show that the new approach can achieve significant performance enhancement for complex queries with respect to the access time and tuning time.

key words: broadcast generation scheme, wireless broadcasting, complex queries, pre-computation, mobile databases

1. Introduction

As the wireless technologies are rapidly gaining popularity, wireless information systems are currently realized in a real environment [1]-[20]. In the wireless information systems, mobile clients carry small, battery powered handheld devices such as PDA's and mobile phones with limited data processing capabilities. In wireless environments, data broadcasting is widely used due to the bandwidth restriction of wireless communication [1], [3]-[6], [8]-[20]. The server disseminates data through a broadcast channel, and mobile clients listen on the channel and retrieve desired information without sending requests to the server.

In wireless data broadcasting, indexing and replication approaches are widely used for energy and latency efficiency reasons. The index information, which is broadcast on the air intermixed with data, is a kind of directory information commonly represented as a set of <data ID, address> tuples. By reading the index information, mobile clients can be informed of the address of their target data. So, they can access the data directly without scanning the full wireless data stream. Also, by replicating hot data items more frequently than cold ones, the overall access time performance can be improved.

In the literature, various indexing methods have been proposed for tuning time efficiency. Some examples are Tune_opt, Access_opt, (1, M), Distributed Indexing [7], [13] for key-based queries, DSI [15] for spatial queries [17], and B2V-Tree [4] for content-based retrieval queries. However, none of them effectively handles complex queries, e.g. aggregation queries such as AVG and SUM. If the broadcast data stream were indexed by one of the previous methods, the mobile clients would have to read the entire data stream for processing these complex queries, and thus, much tuning energy would be consumed.

In this work, we suggest a novel broadcasting paradigm where the pre-computed results of selected complex queries are delivered on the broadcasting channel in company with the data and index stream. For this paradigm, we propose a broadcast stream organization scheme, where the data records, index records, and pre-computed query results are optimally placed on the stream. We also consider replication of the query results, as well as index information, within the broadcast stream to improve the access time performance of complex query processing. We provide analytical and experimental evaluation on the performance of the proposed methods with respect to the access time and tuning time. Also, through the application of the proposed scheme to a real system environment, the energy-efficiency improvement is observed.

The rest of the paper is organized as follows. We first discuss some background information and related work in Sect. 2. After introducing the approach of broadcasting pre-computed query results in Sect. 3, new scheduling methods are proposed and analyzed in Sect. 4. We evaluate effectiveness of the proposed methods through various experiments and practical implications in Sect. 5 and draw a conclusion in Sect. 6.

2. Background and Related Work

Data broadcasting has many applications in wireless information systems due to its beneficial characteristics such as bandwidth-efficiency (i.e., the broadcast channel is shared by many clients), energy-efficiency (i.e., listening on the broadcast channel requires no energy for sending request messages to the server, where the amount of energy required for sending data is tens of hundreds of that for receiving data), and scalability (i.e., the number of clients listening on a broadcast channel is not limited) [2], [10], [13].

In data broadcasting systems, a data server periodically broadcasts data items or records to a large client population. Each period of the broadcast is called a broadcast cycle, while the content of a broadcast cycle is called the bcast. Each client listens to the broadcast and fetches data as they
arrive on the air. The smallest logical unit of a broadcast is called a bucket. Buckets are the analog to blocks for disks. Each bucket may contain several data records (or index entries).

In the wireless data broadcasting, people consider two performance measures for data access: access time and tuning time [1]–[8], [11]–[20]. The former is the duration from the time of query submission (i.e., start to listen on the broadcast channel) to the time of complete download of target information. The latter is the duration when a mobile device remains in the active mode during the access time, which directly relates to the amount of energy consumption of the mobile device. (It is known that, in the case of the Hobbit chip from AT&T, the power consumption in the active mode is 250 mW, whereas that in the doze mode is 50 μW [13]. That is, the ratio of power consumption in the active mode to the doze mode is 5,000.)

Figure 1 illustrates the access time and tuning time of wireless data access. When a mobile client tunes into a data stream on the air, it moves to the next nearest index area. By reading the index, the mobile client can be informed of the target address and thus can move to the target address and download the data. In the figure the access time is ‘d,’ whereas the tuning time is the sum of selective tuning periods of ‘a,’ ‘b,’ and ‘c.’ In the case where there is no index, the whole data stream should be scanned for mobile clients to find the target data, which requires a large tuning time. (The tuning time is the same to the access time.)

There have been some studies for improving the access time performance via effective scheduling of broadcast data [1], [5], [6], [8], [14], [20] and reducing the tuning time via indexing methods which place some index information intermixed with the data on the broadcast stream [3], [4], [7], [13], [17], [19].

The Access_opt method [13] uses no index for the wireless data broadcasting. Since the index information is itself an overhead with respect to the access time performance, this method shows the best access time performance. However, this approach has the worst tuning time performance. The Tune_opt method [13] places the global index information in front of a broadcast stream. Therefore, it has the best tuning time performance, but the worst access time performance due to the overhead from a large index. The (1, M) index method [13] replicates the index information m-times equi-distantly over data in a bcast. By varying the value of m, it can control the preference of the access time vs. tuning time efficiency. The Distributed Index [13] method divides the index into small units and distributes small indices all over the broadcast stream. Since a small index is placed in front of relevant data items, the access time overhead is minimized while the tuning time efficiency is maximized. In this method, some index information is replicated over the stream. The DSI (Distributed Spatial Index) method [15] is for spatial queries [17], wherein a spatial index structure is constructed based on geographic attribute values and placed before the data records. The B2V-Tree index [4] supports content-based retrieval queries by constructing the index structure using the hash values of the contents of data records.

All the above index methods are focused on a specific type of queries and are constructed for providing relevant access paths to the mobile clients. However, this paper focuses on complex queries, such as the queries performing aggregation over the entire data, which cannot be handled efficiently by only exploiting access paths. When such queries are processed in a mobile client over the broadcast stream indexed by a conventional method, the whole data stream should be read into the clients’ memory. Therefore, we suggest a new approach that broadcasts the pre-computed results of the complex queries in addition to base data records and their index. In this paper, we use the term ‘complex queries’ for the queries which cannot be processed using simple index paths and thus require reading of entire data stream, and we call the queries searching for data with a specific attribute value the ‘normal queries.’

3. The Pre-Computation Approach

Because the canonical index, which consists of directory or path information to relevant data records, is not effective for processing the complex queries, we suggest a pre-computation approach where the results of some complex queries are pre-computed and delivered to mobile clients via the broadcasting channel together with data records and index. The queries that are frequently requested by lots of clients are selected for pre-computation. Figure 2 shows a typical layout of the broadcast stream using this approach. The PQR (Pre-computed Query Results) region contains the records for the pre-computed results of complex queries, which consist of the query definition and its result value.

When using this pre-computation approach, the access protocol of mobile clients is as follows: If the query to process is a normal one, the conventional index is used for fast look up. Otherwise, PQR region is searched and the corre-
if (the query is one of the selected complex queries) {
    if (the current bucket is not the start bucket of a PQR region)
        move to the next nearest PQR region;
    find the result of the given query in the PQR region;
    return the result;
} else {
    if (the current bucket is not the start bucket of an INDEX region)
        move to the next nearest INDEX region;
    find an index entry for the given query by reading index;
    move to the target data address and read the data;
    return the data;
}

Fig. 3 The broadcast stream access and query processing algorithm.

The broadcast stream access and query processing algorithm.

4. Broadcast Stream Organization Methods

In the organization of broadcast streams, index replication is popularly used for access time performance reasons [4], [6], [8]. With the replicated index regions in a bcast, mobile clients can easily find an index area and thus retrieve target data in a short access time. However, since too much repetition of index may increase the latency, special care must be taken in determining the number of index replications in a bcast. Moreover, in our pre-computation approach, PQR must be considered for replication. In this section, we present a broadcast stream organization scheme, where the replication of index and PQR is involved, and its performance analysis with respect to the access time and tuning time.

A broadcast stream (bcast) is assumed to be organized as \(<\text{PQR}, \text{INDEX}, \text{DATA}>\). Here, \(\text{PQR}\) denotes the sequence of buckets in which the pre-computed query results are stored, \(\text{INDEX}\) does that of index (i.e., directory) information, and \(\text{DATA}\) does that of data records. \(\Delta\text{PQR}, \Delta\text{INDEX}\) and \(\Delta\text{DATA}\) denote the sizes (i.e., the number of buckets) of \(\text{PQR}, \text{INDEX}\) and \(\text{DATA}\) regions, respectively.

First, we consider the case of non-replication, denoted by the \((1, 1, 1)\) method, where \(\text{PQR}\) and \(\text{INDEX}\) are delivered once in a broadcast cycle. And then, the replication method, the \((1, m, n)\) method, is investigated. ‘\(m\)’ is the number of replications of index and ‘\(n\)’ is the number of replications of \(\text{PQR}\), respectively, in a bcast. Figure 4 shows the broadcast stream structure of the \((1, m, n)\) method. We examine two performance aspects (access time and tuning time in terms of the number of buckets) for normal queries \(\text{NQ}\) and complex queries \(\text{CQ}\).

4.1 Stream without Replication- \((1, 1, 1)\) Method

The access time is determined by two components: \(\text{probeWait}\) and \(\text{bcastWait}\). The \(\text{probeWait}\) is the time required for finding the index area, which is in probability a half of the time between two consecutive index regions. The \(\text{bcastWait}\) is the time required for downloading target data records after index reading. Therefore, the average access time of a normal query \(\text{AvgAT}(\text{NQ})\) is computed in terms of the number of buckets as follows:

\[
\text{AvgAT}(\text{NQ}) = \text{probeWait} + \text{bcastWait} = \frac{\Delta\text{bcast}}{2} + \Delta\text{INDEX} + \Delta\text{DATA}/2
\]

The average tuning time \(\text{AvgTT}(\text{NQ})\) is analyzed as follows: After reading a bucket at the query start position in the bcast, one half of the index region in average must be listened (i.e., tuned) to find an index entry for the target records. Then, the data bucket of target data records will be tuned by using the addresses found before. Here, we assume that the data records are clustered in the stream by the search key attribute and the target data records are stored into one bucket. Also, the firstly tuned bucket is not the starting bucket of an index region. (If it is the starting bucket of an index region, the first ‘\(1\)’ is not necessary in the below formula.)

\[
\text{AvgTT}(\text{NQ}) = 1 + \frac{\Delta\text{INDEX}}{2} + 1
\]

For the complex queries, the mobile clients do not use index. Instead, they search the PQR region and find the pre-computed results of their target queries. So, the average access time of a complex query \(\text{AvgAT}(\text{CQ})\) is determined by the sum of the \(\text{probeWait}\) (for \(\text{PQR}\)) and \(\text{bcastWait}\) as follows:

\[
\text{AvgAT}(\text{CQ}) = \text{probeWait} + \text{bcastWait} = \frac{\Delta\text{bcast}}{2} + \Delta\text{PQR}/2
\]

\[
\text{AvgTT}(\text{CQ}) = 1 + \frac{\Delta\text{PQR}}{2}
\]

4.2 Stream with Replication - \((1, m, n)\) Method

Here, we analyze the performance aspects of index and \(\text{PQR}\).
replicated streams. Let \( m \) and \( n \) be the numbers of index and PQR replications, respectively. Then, the size of a broadcast stream \( \text{bcast} \) is \((m \cdot \Delta \text{INDEX} + n \cdot \Delta \text{PQR} + \Delta \text{DATA})\). The average access times of \( NQ \) and \( CQ \) are computed as follows:

\[
\text{AvgAT}(NQ) = \text{probeWait} + \text{bcastWait} \\
= [\Delta \text{bcast}/2m] + \Delta \text{INDEX} \\
+ [(\Delta \text{bcast} - \Delta \text{INDEX})/2] \\
= [(m+1)\Delta \text{bcast}/2m + \Delta \text{INDEX}]/2] \\
= ([m+2]m\Delta \text{INDEX} + (m+1)\Delta \text{DATA} \\
+ (m+1)n\Delta \text{PQR})/2m]
\]

\[
\text{AvgAT}(CQ) = \text{probeWait} + \text{bcastWait} \\
= [\Delta \text{bcast}/2n] + [\Delta \text{PQR}/2] \\
= [(\Delta \text{DATA} + m\Delta \text{INDEX})/2n] + \Delta \text{PQR}
\]

The tuning times are not dependent on the replications of index and PQR. Therefore, they are the same to those of non-replicated streams as follows:

\[
\text{AvgTT}(NQ) = 1 + [\Delta \text{INDEX}/2] + 1 \\
\text{AvgTT}(CQ) = 1 + [\Delta \text{PQR}/2]
\]

If we assume that \( p \) and \( 1 - p \) are the frequency ratios of normal queries and complex queries \((0 \leq p \leq 1)\), the weighted access time (WAT) and tuning time (WTT) can be represented as follows:

\[
\text{WAT} = p \times \text{AvgAT}(NQ) + (1-p) \times \text{AvgAT}(CQ) \\
\text{WTT} = p \times \text{AvgTT}(NQ) + (1-p) \times \text{AvgTT}(CQ)
\]

Because the tuning time is independent of the replications, we only have to consider the weighted access time for selecting optimal values of \( n \) and \( m \). The formula \( \text{WAT} \) is represented as a function of two variables \( m \) and \( n \), as follows:

\[
\text{WAT}(m, n) = \Delta \text{INDEX}(mp + 2p + m/n - mp/n) \\
+ \Delta \text{DATA}((m+1)p/m + (1-p)/n) \\
+ \Delta \text{PQR}(np(m+1)/m + 2 - 2p)
\]

Figure 5 illustrates the above formula, where \( \Delta \text{PQR} \), \( \Delta \text{INDEX} \) and \( \Delta \text{DATA} \) are set to 50, 100, and 5000 buckets, respectively.

5. Performance Evaluation

5.1 Experiments and Results

In this section, we demonstrate effectiveness of the proposed broadcast generation scheme by experiments. We developed a simulation system for generating and accessing a broadcast data stream and implemented two methods, \((1, 1, 1)\) and \((1, m, n)\), presented in Sect. 4, as well as the conventional \((1, 1)\) and \((1, m)\) methods [8] which do not consider complex queries. For experiments, different types of broadcast streams were synthetically generated by those methods. They contain 16,384 data records and sequential index information in a \( \text{bcast} \). The sizes of a data and index records are 256 bytes and 8 bytes, respectively, hence the sizes of a data and index regions are 4 MB and 128 KB. The streams produced by the \((1, 1, 1)\) and \((1, m, n)\) methods also include PQR regions of 512 records, each of which stores a definition of a complex query and its pre-computed result. The \( \text{PQR} \) record size is 128 bytes long and thus the size of a PQR region is 64 KB. For the experiments on the \((1, m)\) and \((1, m, n)\) methods, we generated the streams having various numbers (from 1 to 20) of replicas of index and/or PQR information in a \( \text{bcast} \). To evaluate performance of data accesses in a client, we executed 50,000 queries over each broadcast stream and measured access times and tuning times in the number of buckets. The same queries were evaluated on the different types of broadcast streams to compare the performances of the proposed and previous methods. We suppose that 20% of the test queries are complex aggregation queries and the rest are normal data searches. We also assumed that the streams were transmitted and accessed in the unit of a bucket whose size is 4 KB.

Figure 6 shows the average tuning time and access time of the test queries over the broadcast streams generated by the considered methods. As shown in Fig. 6 (a), the \((1, 1, 1)\)
and \((1, m, n)\) methods have the same tuning time performance in processing complex queries, which is an order of magnitude better than that of \((1, 1)\) and \((1, m)\) methods. The average tuning times in processing normal queries are the same for all the methods. The tuning time of a query is independent on the number of replications of index or PQR information in \((1, m, n)\) methods.

Figure 6 (b), (c), and (d) show the access time performances of the considered methods. For \((1, m)\) and \((1, m, n)\) methods, the results of the instance methods that have achieved the minimum average access time for the complex, normal, and entire queries, respectively, in the experiments are selected and compared with the results of the other methods. Among all the considered instances of the \((1, m, n)\) method, for example, the \((1, 1, 20)\) method which has no index replication and maximum number of PQR replication have obtained the shortest access time for complex queries on average, and the \((1, 6, 1)\) method showed the best access time performance for the normal queries. We included the results of the \((1, 1)\) and \((1, 1, 1)\) methods in the three figures redundantly to compare them with those of the various instances of \((1, m)\) and \((1, m, n)\) methods.

As expected, when only the index is used in the broadcast streams as in \((1, 1)\) and \((1, m)\) methods, access time for the normal queries is shorter than that for the complex queries, while the broadcast streams in \((1, 1, 1)\) and \((1, m, n)\) methods which also contain PQR information have better access time performance for the complex queries than for the normal queries. Without considering replication of the index or PQR information in a bcast, the average access time of \((1, 1, 1)\) method in processing complex queries is about 65\% shorter that that of \((1, 1)\) method while its average access time for normal queries is only 0.6\% longer than that of \((1, 1)\) method. For the whole queries, its access time is 16\% shorter than that of \((1, 1)\) method on average.

As shown in Fig. 6 (b), \((1, 1, 20)\) method, which is the best instance of \((1, m, n)\) method for complex queries, has the average access time for complex queries which is only about 3\% of that of \((1, 1)\) method and the best instance of \((1, m)\) method. However, its access time performance for normal queries is worse than those of other methods. Figure 6 (c) indicates that \((1, 6, 1)\) method, i.e., the best instance of \((1, m, n)\) method for normal queries, has the average access time which is 32\% shorter than that of \((1, 1, 1)\) method. It is only 1\% larger than that of the \((1, 6)\) method, the best instance of \((1, m)\) method for normal queries. For the whole queries, its average access time is about 39\% and 15\% better than that of the \((1, 1, 1)\) and \((1, m)\) methods, respectively. Finally, Figure 6 (d) compares the access time performances of the best instances of \((1, m, n)\) and \((1, m)\) methods for the whole queries. The average access time of \((1, 6, 4)\) method is about 45\% and 23\% better than that of \((1, 1)\) method and \((1, 5)\) method, respectively. For the complex queries, it is only about 11\% and 13\% of that of \((1, 1)\) and \((1, 5)\) method, respectively.

Figure 7 presents the comparison of access time performances of \((1, m, n)\) methods with different numbers of PQR replications in a bcast. Note that as the number of replicas of the pre-computed results increases, the average access time for complex queries decreases drastically until it is saturated at about 6 replicas of the PQR regions, while the average access time for normal queries grows at a steady linear rate. As a result, we can observe that the broadcast stream having

\[\text{With this result we can verify the correctness of the analysis in the previous section.}\]
Fig. 7 Access time performances of $(1, 6, n)$ methods with different values of $n$.

Finally, we evaluated average tuning time and access time performances of the proposed methods for the whole queries as increasing the portion of the complex queries within the given set of user queries from 0 to 1 by 0.1. Figure 8 shows the results of the best instances of $(1, m)$ and $(1, m, n)$ methods for each value of the ratio, as well as the performances of $(1, 1)$ and $(1, 1, 1)$ methods. The upper figure indicates that as the complex query ratio increases, the average access times of $(1, 1, 1)$ and $(1, m, n)$ methods decrease linearly while those of $(1, 1)$ and $(1, m)$ methods increase almost at the same rate. This clearly presents the impact of additional placement and replication of PQR information within a bcast stream on the access time performance as the portion of complex aggregation queries grows in the set of client queries. We can observe that $(1, 1, 1)$ method outperforms $(1, m)$ method when the portion of complex queries becomes larger than about 30% of the whole queries. Thus, in that case, it is more desirable to add a region of pre-computed results for the complex queries instead of replicating index information in the bcast. We also note that the higher the complex query ratio is, the better the access time performance of $(1, m, n)$ method with respect to $(1, m)$ method which replicates only index information in the bcast. The lower figure shows the average tuning time of $(1, m)$ method grows in proportion to the complex query ratio since the entire data should be read for processing each complex query, while that of $(1, m, n)$ method, which is very short due to the use of pre-computed results in the PQR region in processing complex queries, decreases more as the complex query ratio increases.

5.2 Practical Implications

In this section, we show the significance of our work by applying the proposed method into a real world wireless environment such as the Quotrex system [13]. Quotrex is a well-known real-time information delivery system which broadcasts stock-price records on an FM band. In this paper, as the wireless communication, we assume WCDMA (esp. HSDPA), since it is one of the most advanced and popularly used cellular technologies in the world. Mobile clients are assumed to carry PDA-like devices equipped with the Qualcomm’s MSM6280 chipsets [21]. The maximum data rate (DL in HSDPA mode) of this chipset is 3.6 Mbps, and the energy consumption rates are 800 mA and 8 mA for active and doze modes, respectively. For convenience, we focus on the energy consumption required for data communication, excluding data computing, data storing, GUI, and so on.

In the stock price database, an NQ (Normal Query) retrieves specific stock records identified by (an) index attribute(s), such as “Find the stock record for Microsoft Corp.” and “Find the stock record for Google.” On the contrary, a CQ (Complex Query) can be one of the followings (but not limited):

- “Find the top-10 price stock records”
- “Compute the average price of all stock records”
- “Find the most highly purchased stock”
- “Find the top-5 stock records based on the amount of stock exchanges”

As shown in the above measurements, the pre-computation approach significantly improves the energy-efficiency of complex query processing of mobile queries with very limited overhead and complexities. Considering that the popularity of complex queries, such as aggregations,
could never be diminished with the advance of wireless information delivery applications, the use of pre-computation approach and thus the proposed method of broadcast program generation will be recognized an important technique for wireless information systems.

6. Conclusion

In this paper, we proposed a new broadcast generation scheme to support efficient processing of complex queries over the broadcast data in mobile clients. Firstly, we suggested a new paradigm of broadcasting pre-computation results of complex queries in combination with pure data and index on the broadcast stream. The complex query denotes a query which cannot be processed efficiently using conventional index paths because it requires full scanning of the broadcast stream. For the pre-computation paradigm, we studied the placement and replication of index, data and pre-computation results on the stream, and found the optimal replication plan of them.

We carried out performance analysis, experiments and practical implications of the proposed scheme with respect to the access/tuning time and the amount of energy consumption. The results of experiments showed that, compared with the previous index replication method, our POR replication scheme can achieve significant improvement on the access time and tuning time performance in processing complex queries while it scarcely increases the access time for normal queries. For future work, we plan to investigate the problem of selecting an optimal set of complex queries for pre-computation considering their access frequencies.

Acknowledgements

This work was supported by Korea University Research Grant.

References

Yon Dohn Chung received his B.S degree in Computer Science from Korea University, Seoul, Korea, in 1994, and M.S. and Ph.D. degrees in Computer Science from KAIST, Daejeon, Korea, in 1996 and 2000, respectively. In 2006, he joined the faculty of the Department of Computer Science and Engineering, Korea University, Seoul, Korea. His research interests include broadcast databases, sensor networks, MANET/VANET, XML databases, mobile computing and database systems. He is a member of the KIISE, ACM and IEEE.

Chang-Sup Park received his B.S., M.S., and Ph.D. degrees in Computer Science from KAIST, Daejeon, Korea, in 1995, 1997, and 2002, respectively. From 2002 to 2005, he worked for Korea Telecom in Seoul, Korea. In 2005, he joined the faculty of the Department of Internet Information Engineering, The University of Suwon, Korea. His research interests include wireless information systems, XML data management, semantic Web, and multimedia databases. He is a member of the KIISE.