Proceedings of the
IASTED International Conference

COMMUNICATIONS, INTERNET, AND INFORMATION TECHNOLOGY

Editor: M.H. Hamza

November 18-20, 2002
St. Thomas,
US Virgin Islands

A Publication of The International Association of Science and Technology for Development — IASTED

ACTA Press
Anaheim • Calgary • Zurich

ISBN: 0-88986-327-X

Proceedings of the IASTED International Conference on **Communications, Internet & Information Technology**, held November 18-20, 2002 St. Thomas, US Virgin Islands.

Sponsors

The International Association of Science and Technology for Development (IASTED)

- Technical Committee on Information Systems
- Technical Committee on Telecommunications
- Technical Committee on the Web, Internet and Multimedia

Editor

M.H. Hamza

Special Session Organizer

L. Kocarev
University of California at San Diego, USA

For each IASTED conference, the following review process is used to ensure the highest level of academic content. Each full manuscript submission is peer reviewed by a minimum of two separate reviewers on the International Program Committee / Additional Reviewers list. The review results are then compiled. If there is a conflict in reviews, the paper is sent to a third reviewer.

Abstracted by INSPEC

ACTA Press
P.O. Box 5124
Anaheim, CA 92814-5124
USA

ACTA Press
#80, 4500 - 16th Ave. NW
Calgary, AB T3B 0M6
Canada

ACTA Press
P.O. Box 354
CH-8053 Zurich
Switzerland

Publication Code: 376

Proc. of IASTED International Conference on Communications, Internet, and Information Technology (CIIT 2002) (Nov. 2002)

A COLLABORATIVE QUERY PROCESSING METHOD FOR A DATABASE BROADCASTING SYSTEM

Masakazu Kashita[†] Tsutomu Terada[‡] Takahiro Hara[*] Masahiko Tsukamoto[*] Shojiro Nishio[*]

[†]Dept. of Information Systems Eng., Graduate School of Engineering, Osaka University
[‡]Cybercommunity Division, Cybermedia Center, Osaka University
[*]Dept. of Multimedia Eng., Graduate School of Information Science and Tech., Osaka University

Abstract
In a database broadcasting system, the server periodically broadcasts contents of database to mobile clients such as portable computers and PDAs. In this system, there are two possible methods for query processing: one is that a client stores all data which are necessary for processing a query and then processes it locally, and the other is that the server processes a query and broadcasts the query result to the query issue client. However, a client cannot get query results when the disk space of the client is not enough for query processing in the former method or when overfull queries are issued in the latter method. In this paper, to resolve these problems, we propose a method which processes a query by collaboration of the server and clients. In our method, the server adds identifiers to tuples appearing in the query result. Thus, the client can store only necessary tuples by referring to the identifiers. In addition, the server broadcasts rules which define the client's behavior for receiving data, and then, the query results are constructed on the client by using the rules. In this way, clients can get query results efficiently.

Key Words
Database broadcasting, ECA rule, Query processing, Mobile environment

1 Introduction

The recent evolution of wireless communication technologies has led to an increasing interest in information systems in which data is disseminated via broadcast channels. In such systems, a server broadcasts various data periodically via a broadband channel, and a client picks out and stores necessary data. Since the data delivery cost of the server little increases even if the number of clients increases, the server can disseminate data with high quality and high throughput independent of the number of clients.

Many researches have been done so far to improve the performance of data broadcasting systems. They include data scheduling techniques at the server side [1, 4, 7, 8, 9, 10], caching techniques at the client side [1], update propagation techniques[2], integration of push-based and pull-based techniques[3], and data pre-fetching techniques[5]. Most of them deal with broadcast data only as a *data item*, which is a collection of data, and do not address performance improvement by considering contents, characteristics and types of broadcast data. However, since there are various types of data such as hyper-link texts and tuples of relational database, data processing considering data types is very important to improve system performance.

In this paper, we assume a data broadcasting system in which a server broadcasts contents of a relational database and clients issue database queries to retrieve data from the database. We call such a system a *database broadcasting system*. The goal of this paper is to realize efficient query processing in a database broadcasting system. We propose a query processing method in which the server and a client collaboratively process queries. This method reduces response time, and disk space required for query processing.

The remainder of this paper is organized as follows: In section 2, we explain a database broadcasting system and two basic methods for query processing. In section 3, we describe our proposed collaborative query processing method. In section 4, we evaluate the performance of our method. In section 5, we conclude this paper.

2 Database Broadcasting System

In this section, we explain a database broadcasting system and basic query processing methods in a database broadcasting system, which we call the *client method* and the *on-demand method*.

Figure 1 shows the concept of a database broadcasting system. The server broadcasts contents of a relational database via a broadcast channel and clients issue a query to retrieve necessary data from the database. We also put the following assumptions:

Contents: The server periodically broadcasts contents of a relational database.

Clients: Clients have a small disk storage, low battery, and low CPU power.

Dual downlink channels: The broadcast channel from the server to clients is divided into two channels. The

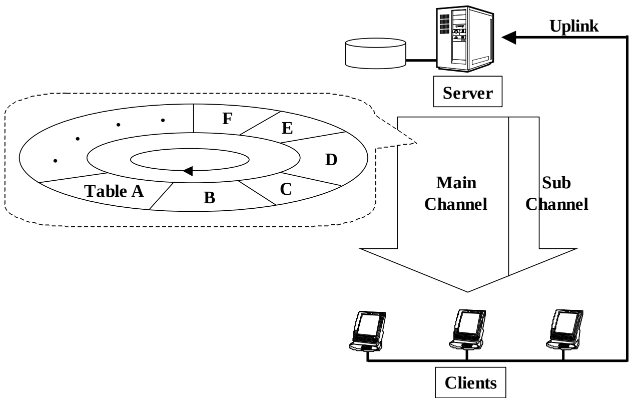


Figure 1. A database broadcasting system.

contents of the database are broadcast via the broadband *main-channel* and other data is broadcast via the narrow-band *sub-channel*.

Uplink channels: There is a narrow-band communication channel from a client to the server.

There are two basic methods for query processing in a database broadcasting system.

Client method: A client stores all broadcast tables that are necessary for processing the query. Then, the client processes the query using the stored tables.

On-demand method: A client sends a query to the server via the uplink. The server processes the received query and sends the query result back to the client via the sub-channel.

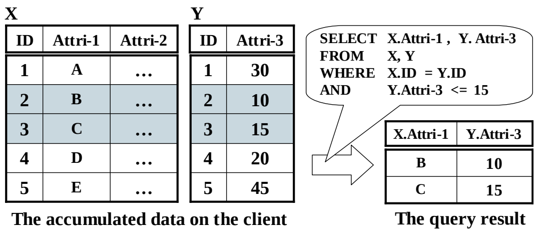
In the following, we explain the both methods in detail.

2.1 Client method

The processing procedure in the client method is as follows:

1. A client issues a query described in SQL.
2. The client monitors the main-channel, and stores broadcast tables which are necessary for processing the query on its disk.
3. The client processes the query after all the necessary tuples are stored.

In this method, even if the number of clients that issue queries heavily increases, every client can receive all necessary data within one broadcast cycle. Moreover, since a client gets the query result without using the uplink, the client method can work in an environment where there is no uplink. However, this method has the following disadvantages:



X

ID	Attri-1	Attri-2
1	A	…
2	B	…
3	C	…
4	D	…
5	E	…

Y

ID	Attri-3
1	30
2	10
3	15
4	20
5	45

```
SELECT  X.Attri-1 ,  Y. Attri-3
FROM    X, Y
WHERE   X.ID  = Y.ID
AND     Y.Attri-3  <= 15
```

X.Attri-1	Y.Attri-3
B	10
C	15

Figure 2. Query processing in the client method.

Large disk space consumption Since a client has to store all necessary tables for processing the query, there is the possibility that the client does not have enough disk space. For example, suppose that the server broadcasts a database consisting of tables *X* and *Y*, and a client issues a query as shown in Figure 2. In this case, the client firstly stores the tables *X* and *Y* on the local disk, and then, processes the query. The gray areas in the tables show tuples that appear in the query result. Therefore, the client wastes the local disk space to store unnecessary tuples and attributes that do not appear in the query result.

High processing cost

Query processing imposes a heavy load to a mobile client that has low computational power, and thus, it may occupy most of the client's work. Moreover, since the processing cost changes according to the complexity of the query, it is difficult to expect the impact on the client.

2.2 On-demand method

The processing procedure in the on-demand method is as follows:

1. A client issues a query and sends it to the server via the uplink.
2. The server processes the query and broadcasts the query result via the sub-channel.
3. The client receives the query result.

In the on-demand method, since the query processing is completely done by the server and a client receives only the query result, the client needs no work space for query processing. When there is no waiting query to be processed at the server, a client can receive the query result immediately. However, since the sub-channel is exhausted when queries are issued frequently or when the size of query results is very large, it takes long time for the client to receive the query result.

3 Collaborative Method

As described in the section 2, there is the possibility that queries cannot be processed in the client method and that it takes long time to get the query result in the on-demand method. To solve these problems, we propose the *collaborative method* for efficient query processing in a database broadcasting system. In this section, we explain our proposed method.

3.1 Overview

The collaborative method reduces the disk space required for query processing as compared with the client method, and reduces the response time as compared with the on-demand method. This is accomplished by processing a query collaboratively by the server and a client. The followings are characteristics of the collaborative method:

Server assistance of query processing: The server analyses a query received from a client and adds identifiers to tuples that appear in the query results. Since the client can store only the necessary data referring to the identifiers, the disk space required on the client is reduced.

Specification of client's behaviors by rules: The server creates rules which specify how a client receives and processes the broadcast data, and sends them via the sub-channel. Based on the received rules, a client automatically receives necessary tuples and reconstructs the query result. Since the size of a rule is generally much smaller than that of a query result, the sub-channel is rarely exhausted as compared to the on-demand method.

3.2 ECA rule

In the collaborative method, rules are used to specify the behaviors of clients. As the description formula, we use the *ECA rule* which is a behavior definition language in active database systems. An active database system processes prescribed actions in the response to the occurrence of an event generated inside/outside of databases[6]. An ECA rule consists of *Event*, *Condition*, and *Action*. Event is the occurring event in the system, Condition is the condition to execute actions, and Action is the operation to be performed when the conditions are satisfied. The use of ECA rules for specifying clients' behaviors brings the following advantages:

- Since system behaviors are specified in the event-driven manner, query processing can be represented as a collection of rules, which specify the behaviors on receiving data, processing received tuples, and so forth.
- Since all functions on query processing are represented as a set of rules, they can be easily customized by updating, adding, and deleting ECA rules.

Events and actions that can be used in our method are shown in Table 1 and Table 2. Additionally, we provide

Table 1. Events.

Name	Event
SELECT	Retrieval of data from a table
INSERT	Insertion of tuples into a table
DELETE	Deletion of tuples from a table
UPDATE	Update of tuples in a table
RECEIVE	Arrival of data
TIMER	Activation of a specified timer

Table 2. Actions.

Name	Action
QUERY([*query*])	Operation to the database
ENABLE_ECA([*Rule ID*])	Activation of rules
DISABLE_ECA([*Rule ID*])	Deactivation of rules
INSERT_ECA([*Rule ID*])	Storage of rules
DELETE_ECA([*Rule ID*])	Deletion of rules
SET_TIMER([*Condition of a timer*], [*Time*])	Setting of a new timer
KILL_TIMER([*Timer ID*])	Deletion of a timer
STORE([*Query ID*],[*Table name*], [*Table name*], [*Attribute*], ...)	Storage of tuples
MATCH([*Query ID*],[*Table name*], [*Matching table name*],[*Attribute*], ...)	Storage of tuples in combining with stored ones
DISPLAY([*Query ID*])	Display of a query result

two system variables, *NEW data* and *OLD data*. When an event occurs, the system variables are set to certain values according to Table 3, and they can be used in describing rules.

3.3 Query processing algorithm

The query processing procedure in the collaborative method is as follows:

1. **Query initiation and transmission**

   A query is issued by a client and it is sent to the server via the uplink.

2. **Addition of identifiers**

   It is assumed that each tuple in the broadcast database has two extra attributes for the *query identifier* and the *combination identifier*. The query identifier is a unique identifier that is given by the server to each query. The combination identifier is an identifier to match tuples when a query result is constructed from multiple tables, such as the case of a join operation. The server analyzes the received query and investigates which tuples appear in the query result. Then, the server adds the query identifier and the combination identifier in those tuples. Referring to the two attributes, the client can store only tuples appearing in the query result. Since each of the two attributes has a capacity to store several identifiers at the same time, the system can process several queries in one broadcast cycle even if some tuples appear in the results of more than one query.

3. **Creation of ECA rules**

Table 3. NEW data and OLD data.

Event	NEW	OLD
SELECT	Referenced tuples	-
INSERT	Inserted tuples	-
DELETE	-	Deleted tuples
UPDATE	Updated tuples	Old tuples
RECEIVE	Contents of arrived packet	-
TIMER	Timer identifier	-

```
SELECT   X.Attri-1 ,  Y. Attri-2
  FROM   X, Y
 WHERE   X.ID  = Y.ID
   AND   Y.Attri-2  <=  15
```

Figure 3. A query example.

The server creates ECA rules for clients to receive the necessary data and reconstructing the query result on the client. Several templates for typical ECA rules are provided for efficient rule description. If possible, the server creates ECA rules by setting query parameters to the templates. Otherwise, it creates rules completely by itself. The server also creates ECA rules for tuning the main-channel only when the necessary tuples are broadcast. This enables clients to shorten the monitoring time, and thus, reduce the power consumption.

4. **Broadcast of ECA rules**

   The server broadcasts the ECA rules via the sub-channel. The client continues to monitor the sub-channel until the necessary ECA rules are broadcast, and then, receives and stores them.

5. **Reconstruction of query result**

   Based on the received ECA rules, the necessary tuples are received and the query result is automatically reconstructed by combining these tuples.

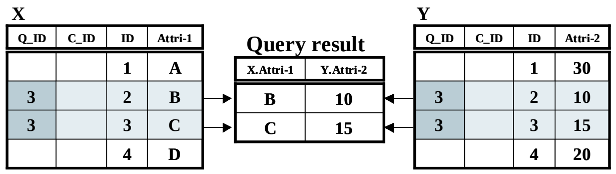
6. **Release of identifiers**

   The server expects the time when the client completes to receive the necessary tuples, and after the time, it releases all the identifiers added for the query issued by the client.

3.4 Query example

We show an example of query processing. Let us suppose that a client issues a query shown in Figure 3, and the server assigns a query identifier to the query as 3. As shown in Figure 4, the server investigates tuples which appear in the query result and writes the number 3 in the attribute for query identifier, Q_ID, on these tuples. Then, as shown in Figure 5, the server investigates pairs of tuples that are combined to reconstruct the query result and writes the combination identifiers in the attribute, C_ID, on these tuples.

Finally, the server creates ECA rules and sends them to the clients. In this example, the set of created ECA rules



X

Q_ID	C_ID	ID	Attri-1
		1	A
3		2	B
3		3	C
		4	D

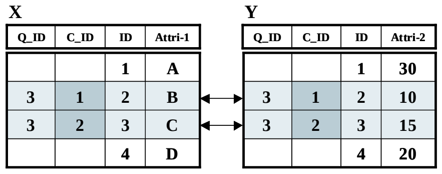
Query result

X.Attri-1	Y.Attri-2
B	10
C	15

Y

Q_ID	C_ID	ID	Attri-2
		1	30
3		2	10
3		3	15
		4	20

Figure 4. Addition of query identifiers.



X

Q_ID	C_ID	ID	Attri-1
		1	A
3	1	2	B
3	2	3	C
		4	D

Y

Q_ID	C_ID	ID	Attri-2
		1	30
3	1	2	10
3	2	3	15
		4	20

Figure 5. The addition of combination identifiers.

is shown in Figure 6. ‘Rule3-1’ is a rule for creating a table for storing the query result. ‘Rule3-2’ and ‘Rule3-3’ are for setting a timer to trigger the rule for displaying the query result. ‘Rule3-4’, ‘Rule3-5’, ‘Rule3-6’, and ‘Rule3-7’ are for receiving the broadcast data. ‘Rule3-8’ and ‘Rule3-9’ are for displaying the query result, and delete all rules and timers.

4 Evaluation

In this section, we evaluate our method by comparing with the two basic methods from the following three points of view:

1. Disk space consumption

   The disk space is required by a client to process a query. Note that this does not include the space for storing the query result.

2. Response time

   The elapsed time from the query initiation to the receipt of the query result.

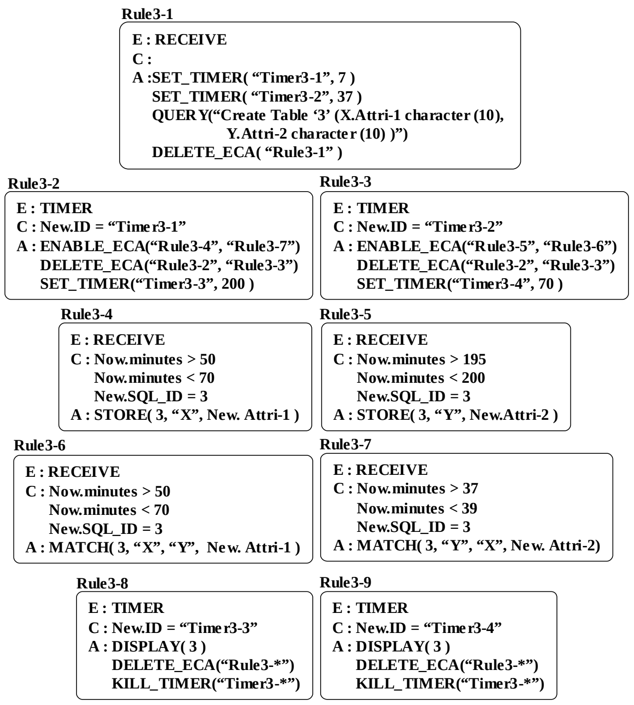
3. Tuning time

   The time during when a client listens the broadcast channels.

4.1 Evaluation model

In this evaluation, we choose the database schema and the query model assuming an information service in a shopping center. In this service, the server broadcasts the information on shops, goods and so forth. Clients equipped with PDAs walk around and receive the broadcast data. The clients issue queries that request data including images, e.g., ‘I want images of *goods A* and the name list of shops which deals in it’.

There are two tables for each of genres such as apparel, interior, and restaurant; one of the tables is a shop table {shopID, shop name, image} and the other is a goods table {goodsID, shopID, goods name, image}. The shop



```
Rule3-1
E : RECEIVE
C :
A :SET_TIMER( "Timer3-1", 7 )
   SET_TIMER( "Timer3-2", 37 )
   QUERY("Create Table '3' (X.Attri-1 character (10),
             Y.Attri-2 character (10) )")
   DELETE_ECA( "Rule3-1" )

Rule3-2
E : TIMER
C : New.ID = "Timer3-1"
A : ENABLE_ECA("Rule3-4", "Rule3-7")
    DELETE_ECA("Rule3-2", "Rule3-3")
    SET_TIMER("Timer3-3", 200 )

Rule3-3
E : TIMER
C : New.ID = "Timer3-2"
A : ENABLE_ECA("Rule3-5", "Rule3-6")
    DELETE_ECA("Rule3-2", "Rule3-3")
    SET_TIMER("Timer3-4", 70 )

Rule3-4
E : RECEIVE
C : Now.minutes > 50
    Now.minutes < 70
    New.SQL_ID = 3
A : STORE( 3, "X", New. Attri-1 )

Rule3-5
E : RECEIVE
C : Now.minutes > 195
    Now.minutes < 200
    New.SQL_ID = 3
A : STORE( 3, "Y", New.Attri-2 )

Rule3-6
E : RECEIVE
C : Now.minutes > 50
    Now.minutes < 70
    New.SQL_ID = 3
A : MATCH( 3, "X", "Y",  New. Attri-1 )

Rule3-7
E : RECEIVE
C : Now.minutes > 37
    Now.minutes < 39
    New.SQL_ID = 3
A : MATCH( 3, "Y", "X", New. Attri-2)

Rule3-8
E : TIMER
C : New.ID = "Timer3-3"
A : DISPLAY( 3 )
    DELETE_ECA("Rule3-*")
    KILL_TIMER("Timer3-*")

Rule3-9
E : TIMER
C : New.ID = "Timer3-4"
A : DISPLAY( 3 )
    DELETE_ECA("Rule3-*")
    KILL_TIMER("Timer3-*")
```

Figure 6. Created ECA rules.

Table 4. Parameters.

Name	Content	Value
g	Number of genres	10
n	Number of shops per a genre	40
t	Number of goods per a shop	100
s	Size of a tuple[Bytes]	5000
i	Max number of identifiers per a tuple	100
b_m	Bandwidth of the main-channel[Mbps]	10
b_s	Bandwidth of the sub-channel[Mbps]	1
s_q	Size of a query[Bytes]	100
d	Interval of query initiation[second]	1 – 20
r	Tuple usage rate[%](the rate of tuples in a goods table included in a query result)	5
s_e	Size of a ECA rule[Bytes]	140

table has the attributes 'shopID' as the primary key, the shop name, and the map image. The goods table has the attributes 'goodsID' as the primary key, the goods name, the identifier of the shop which deals in the goods, and the graphical image of the goods.

A query is described in SQL. For the sake of simplicity, we suppose all tables are of the same size. Moreover, a client issues only natural join queries with the shop table and the goods table in the same genre.

Table 4 shows parameters and their values used in this evaluation. These values are based on the case of a real shopping center.

4.2 Disk space consumption

In this section, we evaluate the disk space consumption in each of the three methods. The disk space consumptions in the three methods are expressed by the following formulas:

- Client method: $D_{cli} = sn\,(t+1)$
- On-demand method: $D_{on} = 0$
- Collaborative method: $D_{col} = sn\,(tr/100+1)$

In the client method, since a client has to store all tables required for processing the query, the disk space consumption is the total size of all tables. In the on-demand method, since the query result itself is broadcast, no work space is needed on the client. In the collaborative method, the disk space consumption is calculated by multiplying the size of a goods table by the *tuple usage rate* and adding the size of a shop table to it. Although the rates of tuples in goods and shop tables that appear in the query result depend on the query, for the sake of simplicity, we suppose that the rate in a shop table is 1 and that in a goods table is r.

Figure 7 shows the disk space consumption in each of the three methods when changing the tuple usage rate. The tuple usage rate has no impact on the disk space consumption in the client method, because all tables related to a query are stored on the client. In the collaborative method, the disk space consumption linearly increases as the tuple usage rate gets higher. It is shown that when the tuple usage rate is low, the collaborative method can greatly reduce the disk space consumption as compared with the client method. Even in the worst case, it gives the same disk space consumption as the client method. When the tuple usage is lower than 5%, the disk space consumption in the collaborative method is lower than 1MByte. Thus, a client with low memory space of 10MByte such as a PDA, cannot store all the necessary tables in the client method, but can store all the necessary data in the collaborative method.

4.3 Response time

The response time in each of the three methods is represented by a formula described in Appendix. Figure 8 shows the response time when changing the interval of query initiation. The client method is not affected by the interval of the query initiation. In the on-demand method and the collaborative method, as the interval of query initiation gets shorter, the response time gets suddenly longer from certain points. We call these points the *query interval limits*. This result shows that the sub-channel is exhausted at the query interval limits. The response time in the collaborative method is a little longer than that in the client method where the interval of query initiation is longer than the query interval limits. This is because the broadcast data size increases due to the growth of the total size of identifiers. The query interval limit in the collaborative method is much shorter than that in the on-demand method, because the size of ECA rules is much smaller than that of query results. This shows that the collaborative method works well even when queries are frequently issued. However, the response time of the on-demand method is very short when the query initiation frequency is low.

Figure 9 shows the response time when changing the tuple usage rate, r. The response time of the on-demand method is heavily affected by the tuple usage rate. As the tuple usage rate gets larger, the size of query results gets larger, and thus, the response time and the query interval limit get longer. The response times of the other methods are not affected by the tuple usage rate.

Figure 10 shows the impact of the max number of identifiers in a tuple, i, on the response time of the collaborative method. As the max number of identifiers gets larger, the query interval limit of the collaborative method gets shorter.

Figure 11 shows the impact of the bandwidth of the sub-channel, b_s, on the response time. As the bandwidth of the sub-channel gets larger, in the on-demand method, the response time becomes shorter because query results can be sent in a short time. Moreover, since the sub-channel are hardly exhausted the query interval limit gets shorter.

4.4 Tuning time

We evaluate the tuning time in each method. The tuning time in each method is represented by a formula described in Appendix. Figure 12 shows the tuning time when changing the interval of query initiation.

It is shown that the tuning times in the client method and the collaborative method are much shorter than the response times shown in Figure 8. This is because clients can know when the necessary tuples appearing in the query result will be broadcast by using the index in the client method or by using the ECA rules in the collaborative method. However, in the on-demand method, since the client must keep watching the sub-channel until the query result is broadcast, the tuning time is equivalent to the response time.

5 Conclusion

In this paper, we proposed the collaborative method for efficient query processing in a database broadcasting environment. The proposed method reduces the client's disk space required for query processing as compared with the client method, and the response time as compared with the on-demand method.

As part of our future work, we plan to implement a system based on our method and perform an evaluation in a real environment. Moreover, we will address a mechanism to automatically select the best method among the client method, the on-demand method, and the collaborative method according to changes of a system situation.

Acknowledgments

This research was supported in part by Grant-in-Aids for Scientific Research on Priority Areas numbered 14019063 from Japan Society for the Promotion of Science.

References

[1] S. Acharya, M. Franklin, and S. Zdonik: "Broadcast Disks: Data Management for Asymmetric Communication Environments," *Proc. ACM SIGMOD*, pp. 199–210 (1995).

[2] S. Acharya, M. Franklin, and S. Zdonik: "Disseminating Updates on Broadcast Disks," *Proc. VLDB Conference*, pp. 354–365 (1996).

[3] S. Acharya, M. Franklin, and S. Zdonik: "Balancing Push and Pull for Data Broadcast," *Proc. ACM SIGMOD*, pp. 183–194 (1997).

[4] D. Aksoy, and M. Franklin: "Scheduling for Large-Scale On-Demand Data Broadcasting," *Proc. IEEE INFOCOM*, pp. 651–659 (1998).

[5] D. Aksoy, M. Franklin and S. Zdonik: "Data Staging for On-Demand Broadcast," *Proc. VLDB Conference*, pp. 571–580 (2001).

[6] G. Lohman, L. Bruce, P. Hamin, and K. Bernhard, "Extentions to Starburst: Object, Types, Functions, and Rules," *Communications of the ACM*, vol. 34, no. 10, pp. 94–109 (1991).

[7] E. Yajima, T. Hara, M. Tsukamoto, and S. Nishio: "Interval Optimization of Correlated Data Items in Data Broadcasting," *Proc. of Int'l Conf. on Advances in Information Systems (ADVIS 2000)*, pp. 127–136 (2000).

[8] E. Yajima, T. Hara, M. Tsukamoto, and S. Nishio: "Scheduling Strategies of Correlated Data in Push-Based Systems," *Information Systems and Operational Research (INFOR)*, pp. 152–173 (2001).

[9] E. Yajima, T. Hara, M. Tsukamoto, and S. Nishio: "Scheduling and Cashing Strategies for Broadcasting Correlated Data," *Proc. ACM Symposium on Applied Computing (ACM SAC 2001)*, pp. 504–510 (2001).

[10] Q. Hu, D. Lee, and W. Lee: "Performance evaluation of a wireless hierarchical data dissemination system," *Proc. MobiCom'99* , pp. 163–173 (1999).

Appendix

The response time and the tuning time are expressed using the parameters in Table 4. We use the following notations which are represented by formulas (1) to (10).

$$S = \frac{2tnsr}{100} \tag{1}$$

$$C_{cli} = \frac{8(t+1)ngs}{b_m} \tag{2}$$

$$C_{col} = \frac{(8s + (2\lfloor \log_2(tn) \rfloor + 1)\, i)\,(t+1)ng}{b_m} \tag{3}$$

$$D_{col} = \frac{72s_e}{b_s} \tag{4}$$

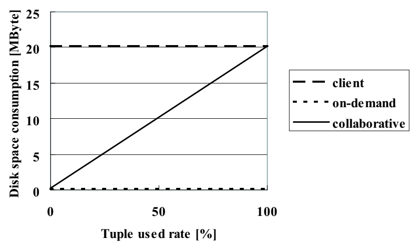


Figure 7. The tuple usage rate vs. the disk space consumption.

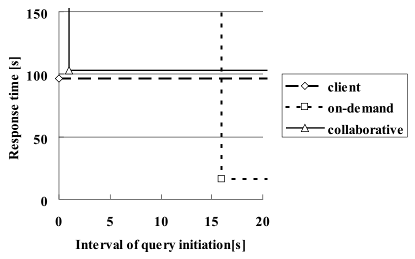


Figure 8. The interval of query initiation vs. the response time.

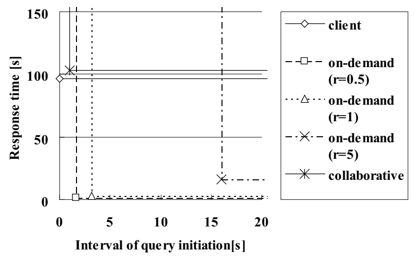


Figure 9. The interval of query initiation vs. the response time when changing the tuple usage rate.

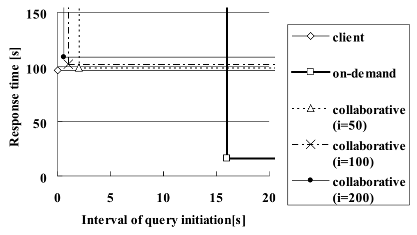


Figure 10. The interval of query initiation vs. the response time when changing the max number of identifiers.

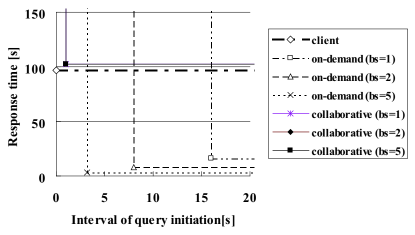


Figure 11. The interval of query generation vs. the response time when changing the bandwidth of the sub-channel.

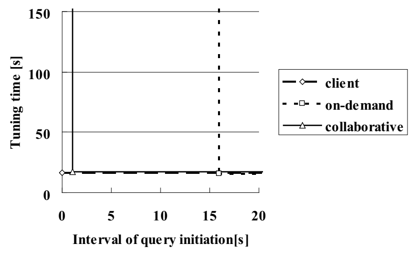


Figure 12. The interval of query initiation vs. the tuning time.

Table 5. Notations for formulation.

Notation	Meaning
S	Size of a query result[Bytes]
C_{cli}	Broadcast cycle(client method)[second]
C_{col}	Broadcast cycle(collaborative method)[second]
Q_{col}	Time until receiving of ECA rules (collaborative method)[second]
Q_{on}	Time untill receiving of a query result (on-demand method)[second]
D_{col}	Time for receiving ECA rules (collaborative method)[second]
D_{on}	Time for receiving a query result (on-demand method)[second]
R_{col}	Time for receiving broadcast data (collaborative method)[second]
R_{cli}	Time for receiving broadcast data (client method)[second]

$$D_{on} = \frac{8S}{b_s} \tag{5}$$

$$F(g,t) = \frac{1}{g}\left(\frac{g^2-1}{2g} + \frac{t^2+1}{2g(t+1)^2} + 1\right) \tag{6}$$

$$R_{col} = C_{col}F(g,t) \tag{7}$$

$$R_{cli} = C_{cli}F(g,t) \tag{8}$$

$$Q_{col} = \begin{cases} \infty & \left(0 < d < max\left[\frac{R_{col}}{i}, D_{col}\right]\right) \\ 0 & \left(max\left[\frac{R_{col}}{i}, D_{col}\right] \le d\right) \end{cases} \tag{9}$$

$$Q_{on} = \begin{cases} \infty & \left(0 < d < \frac{8S}{b_s}\right) \\ 0 & \left(\frac{8S}{b_s} \le d\right) \end{cases} \tag{10}$$

Table 6. Response time and tuning time.

Name	Content
A_{cli}	Response time(client method)[second]
A_{col}	Response time(collaborative method)[second]
A_{on}	Response time(on-demand method)[second]
T_{cli}	Tuning time(client method)[second]
T_{col}	Tuning time(collaborative method)[second]
T_{on}	Tuning time(on-demand method)[second]

In formulas (9) and (10), when the processing rate at the server is lower than the query initiation frequency, the times become very long, and thus, they are expressed by the infinite value.

Using formulas (1) to (10), the response time and the tuning time (Table 5) in each of the three methods are represented by formulas (11) to (16). For the sake of simplicity, we neglect consider the time for sending a query to the server and the computed time for a query processing at a client.

$$A_{cli} = R_{cli} \tag{11}$$

$$A_{on} = Q_{on} + D_{on} \tag{12}$$

$$A_{col} = Q_{col} + D_{col} + R_{col} \tag{13}$$

$$T_{cli} = \frac{C_{cli}}{g} \tag{14}$$

$$T_{on} = Q_{on} + D_{on} \tag{15}$$

$$T_{col} = Q_{col} + D_{col} + \frac{C_{col}}{g} \tag{16}$$