Efficient Query Subscription Processing in a Multicast Environment

Arturo Crespo Orkut Buyukkokten Hector Garcia-Molina

July 28, 1998

Abstract

This paper examines query subscription merging in a distributed environment where multicast channels are used to deliver information. It describes methods for reducing the cost of delivering information by merging overlapping data in the query answers and by using the multicast channels effectively.

1 Introduction

Ongoing advances in communications, including the proliferation of the Internet and intranets, the development of mobile and wireless networks, and the impending availability of high bandwidth links to the home, have fueled the development of a wide range of new information-centered applications. Many of these new applications involve data dissemination, that is, the delivery of data from a set of *producers* to a (typically) larger set of *consumers*. Examples of dissemination-based applications include information feeds (e.g., stock and sports tickers of news wires), traffic information systems, electronic newsletters, and entertainment delivery [FZ96].

Current subscription services make three simplifying assumptions. First, each query is processed independently. Second, the answers to each query subscription are transmitted separately. And third, clients are considered "dumb" processes; this is, the subscription service does not require any post-processing on the answers sent to them. We believe that these assumptions restrict the usefulness of query subscription service and increase the cost of processing query subscriptions. For example, consider the extreme case where n clients issued exactly the same query in their query subscriptions. A standard subscription service will process and transmit the answers to those queries n times. This is wasteful and could be avoided by *merging* all those queries into a single query and processing and transmitting the answer only once.

The cost of processing a set of queries may be reduced even further if we consider merging not only identical queries but also queries with answers that contain a large amount of overlapping data. By merging these queries, the server has to process fewer queries and the amount of information sent may be reduced. [1] On the negative side, the merged answers may contain some data that is

[1] As we will see later, in some cases, merging queries might result in an *increase* of the data sent

irrelevant for some of the clients. As a result, the clients need to apply an *extraction query* over the answer to obtain the answer of their original queries. For example, if we have the queries $q_1 : \sigma_{2 \leq A \leq 40} R(A)$ and $q_2 : \sigma_{3 \leq A \leq 41} R(A)$, we can merge them into $q_3 : \sigma_{2 \leq A \leq 41} R(A)$. The server can then process this single query and send the result, $ans(q_3)$, to the clients that issued q_1 and q_2. Then, the client that issued q_1 will need to *extract* the answer of q_1 from $ans(q_3)$ by applying the extraction query $q_1 : \sigma_{A \leq 40}(q_3)$, and the client that issued q_2 will need to use a similar extraction query to eliminate all elements less than 3. By merging q_1 and q_2 into q_3, we have reduced both the amount of work done by the server to process the query and the amount of information sent to the clients; however, this has been at the expense of having to post-process the messages at the clients.

2 Motivating Example

To motivate our presentation we will use the Battlefield Awareness and Data Dissemination initiative (BADD). The goal of the BADD initiative is to develop an operational system that delivers to combat troops an accurate, timely, and consistent picture of the battlefield and provides access to key transmission mechanisms and worldwide data repositories. Figure 1 outlines the relevant components of the BADD initiative.

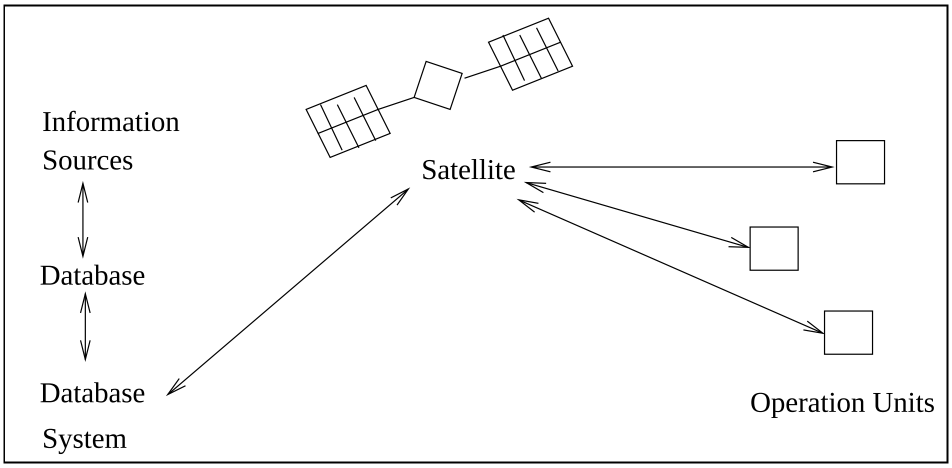

Figure 1: The BADD Scenario

In the BADD scenario a database is updated using *information sources*. A DBMS receives queries from *operation units* via satellite and answers those queries. The operation units are limited capacity computers that can perform simple operations on the data received.

A common request in the BADD environment is information (troop presence, weather, topography, etc.) about a geographical area. For these request, we will need information sources that associate to each object a geographical location. For example, if the data source is a relational database, it can have the schema $R(longitude, latitude, otherattributes)$, where the pair $(longitute, latitude)$ defines the placement of the object and $otherattributes$ is the description of the object. This database can be visualized as in Figure 2(a). The dots in the figure represent the objects that have a given longitude and latitude. As stated before, operational units will query this database for objects inside a geographical area. For simplicity, we will assume that such area is a rectangle, defined by two coordinates (c_1, c_2) and (c_3, c_4). The queries over the database will have the form: $\sigma_{(c_1 <= latitude <= c_3) \wedge (c_2 <= longitude <= c_4)} R$. In Figure 2(b), we represent this query visually. Although we will use this simple scenario as our running example, we want to stress that our system can handle more complicated queries and database schemas.

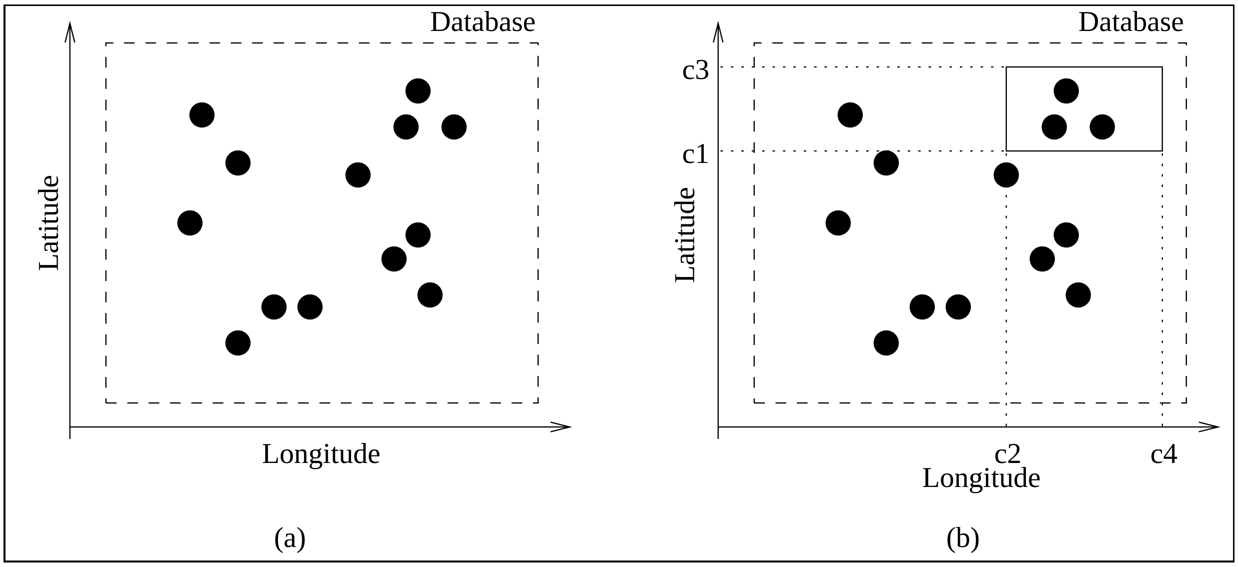

Figure 2: A Sample BADD Database

In the BADD environment, queries tend to be cluster in a few geographical areas. Obviously, if two clients issue identical queries, we want to process and transmit the answer of the queries only once. Beyond that, if the queries are "similar," but not identical, we can decide to merge them into a single query. For example, in Figure 3, we have 2 queries q_1 and q_2; these queries are very similar and it may be advantageous to merge then in to a single query $mrg(\{q_1, q_2\})$. Note that the answer to the new query will contain objects that were not in the answer of q_1, or in the answer of q_2, or both. The operating units that receive the answer of $mrg(\{q_1, q_2\})$ must be able to derive from it the answers to q_1 and q_2. In this paper, we explore the problem of merging similar queries in a multicast environment. First, in Section...

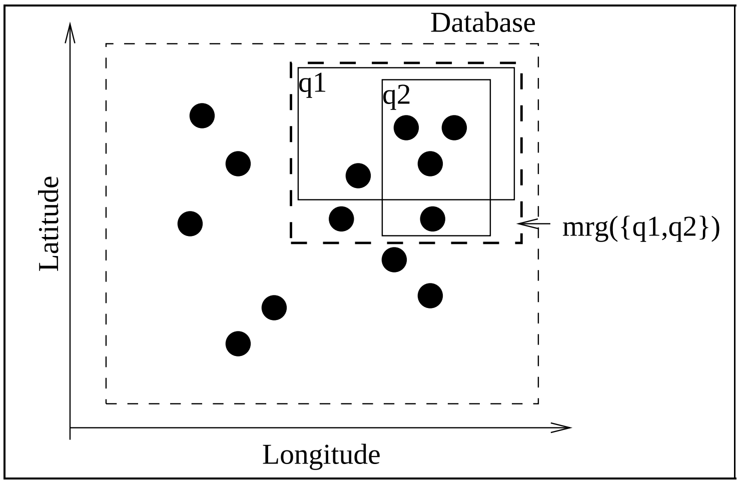

Figure 3: Query Merging

3 Problem Specification

Our objective is to reduce the cost of answering a set of query subscriptions made by clients to a server. We attempt to reduce the cost by finding a (possibly) different set of queries, with lower processing and transmission costs, from which the clients can derive the answer to their original queries. In this section, we discuss our conceptual model in detail.

3.1 Conceptual Model

The conceptual model for a query subscription service is shown in Figure 4. In this model, we have a set of clients, $c_1, ..., c_n$, that require information. The information need of c_i is described by a set of subscriptions. Each subscription consists of a query and its timing requirements (e.g., how often it should be run). For simplicity, we assume that all subscriptions have identical timing requirements. Thus, we can view the subscriptions of client c_i simply as a set of queries Q_i. We call Q the set of all queries received by the server.

Clients send their sets of queries to a *server*. The server periodically processes the queries against a database, and sends *answers* to the clients. Before processing queries, the server runs a merge algorithm that combines "similar" queries. The output of the merge process is a collection $\mathcal{M} = \{M_i\}$ where each M_i contains the queries that are merged. The queries in each M_i are merged into a single query, $mrg(M_i)$. We use $ans(q)$ to represent the answer to query q. Thus, the server generates $ans(mrg(M_i))$ for each M_i in $\mathcal{M}$. For completeness, we require that $\cup_i Q_i = \cup_i M_i$. Similarly, we require that $ans(q) \subseteq ans(mrg(M_i))$ for every $q \in M_i$. We call the difference between the answer to the merged query sent to a client, $ans(mrg(M_i))$, and the original query, $ans(q)$, the *irrelevant information for* q sent to the client.

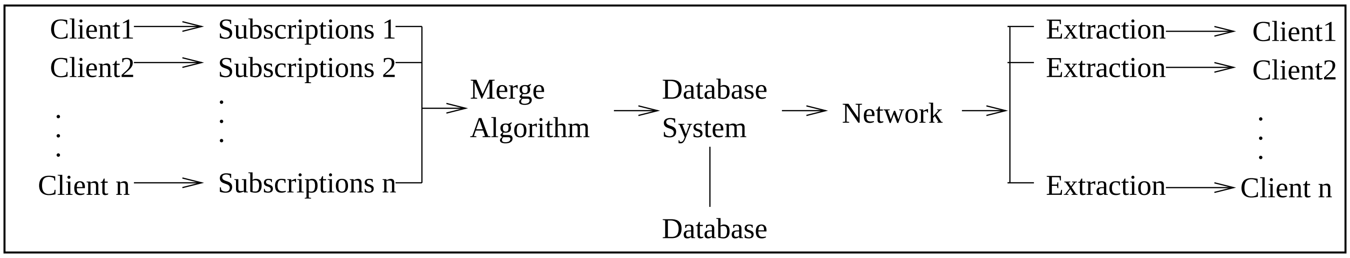

Figure 4: Subscription Service Conceptual Model

To illustrate these concepts, say client c_1 submits queries $Q_1 = \{x, y\}$, and c_2 submits $Q_2 = \{z\}$. The server may merge them into $M_1 = \{x, z\}$ and $M_2 = \{y\}$. Then the server runs $mrg(M_1)$ and $mrg(M_2)$ against the database and generates $A_1 = ans(mrg(M_1))$ and $A_2 = ans(mrg(M_2))$. Note that A_1 needs to be sent to both c_1 and c_2 and that each must apply an *extractor* to obtain the desired answer. For example, the extractor, e, that c_1 applies to A_1 should yield $e(A_1) = ans(x)$. Thus, when the server sends A_1 out, it must include a *header* containing the following information:

- A list of clients that should receive A_1.
- For each such client c, one or more pairs, (e, q), where e is an extractor and q is a query identifier. The extractor e is what client c needs to apply to obtain the answer to its original query q.

Note that more than one (e, q) pair is needed if multiple c queries are involved in A_1. If clients do not need to know what queries generated answers, then the query identifiers are not required. In our example, the information sent with A_1 would be $c_1 : (e_x, x)$ and $c_2 : (e_z, z)$. Client c_1 then applies $e_x(A_1)$ to obtain its answer to query x while c_2 applies $e_z(A_1)$ to obtain its answer.

There are many options for implementing extractors. For example, the server could tag each individual answer object with the identifier of the query that generates the object, or with the identifier of the client that should receive the object. Then each extractor only needs to look for the appropriate tags. In some cases, the extractor for a query is the query itself. In particular, this happens when queries only have selections and projections. A related issue is which component generates an extractor. Above we assumed that extractors were generated by the server and sent with answers. However, if the client can deduce its extractors (e.g., if the extractor is the original query itself), then the server needs not send them.

Note that our basic model does not specify what kind of network is used to send queries to the server and answers to the clients. In the next section, when we discuss cost, we will introduce a simple broadcast model. This model is then extended in Section 7.

3.2 Geographic Queries

To illustrate our conceptual model, we will formalize the geographic query example presented in Section 2. For simplicity, we consider the database to be a single relation R, that has position attributes (e.g., "latitude" and "longitude"), as well as other attributes describing that position.

A geographic query has the form $\sigma_{(c_1<=latitude<=c_3)\wedge(c_2<=longitude<=c_4)}R$. The database server receives a set Q of geographic queries and produces a set $\mathcal{M}$ of merged geographic queries. An example of a merge procedure for our geographical scenario was presented in Section 2. Let us now expand on this and other merge procedures. In Figure 5, we illustrate three different merge functions that can be used in the geographic query scenario. In the figure, the solid lines represent the queries, and the dotted line represents the result of the merge procedure. Figure 5(a) shows the *bounding rectangle merging procedure*, the merging procedure introduced in Section 2. This procedure merges a set of 2-dimensional selection queries into a single 2-dimensional selection query. The merging is done by finding the most restrictive geographic query that includes the original queries. We can visualize this merged query as the smallest rectangle that bounds the original queries. The bounding rectangle merge procedure is very simple (and therefore fast to execute). Additionally, it is easy to extract the answers to the original queries from the answers to the merged query, as we just need to re-apply the original geographical query on the received answers. However, a disadvantage is that the answer includes objects that will be irrelevant to some or all of the input queries.

There are other possible merge functions for the geographic query scenario. Figure 5(b) shows the *bounding polygon merging procedure*. This procedure also generates a single merged query, but, the query may have disjunctions. Although, the merge query contains less irrelevant information than the bounding rectangle merge procedure, irrelevant information is still present (the area of the polygon outside each query is irrelevant to the query). We can again use the original query as the extraction function for this merging procedure. Figure 5(c) shows a merge procedure that completely eliminates irrelevant information. However, five "merged" queries are generated. A client implementing the extraction function for this merging procedure needs to combine the answers to the five merged queries in order to find the answer to the original query.

In summary, there are many choices for merge functions that trades off complexity of query, complexity of the extractor and amount of irrelevant information added.

In this paper, we have assumed that $|mrg(M)| = 1$. However, our model can be easily extended to the case when $|mrg(M)| > 1$ by taking the union of the answers in $mrg(M)$ and creating a single answer.

4 The Cost Model

As described in Section 3.1, the server receives a set of queries Q and outputs a set $\mathcal{M}$ where each of its elements is a set of queries to be merged. The query merging problem is to find the set $\mathcal{M}$ with the minimum cost. The input for the query merging problem is a cost function $cost()$, a merge

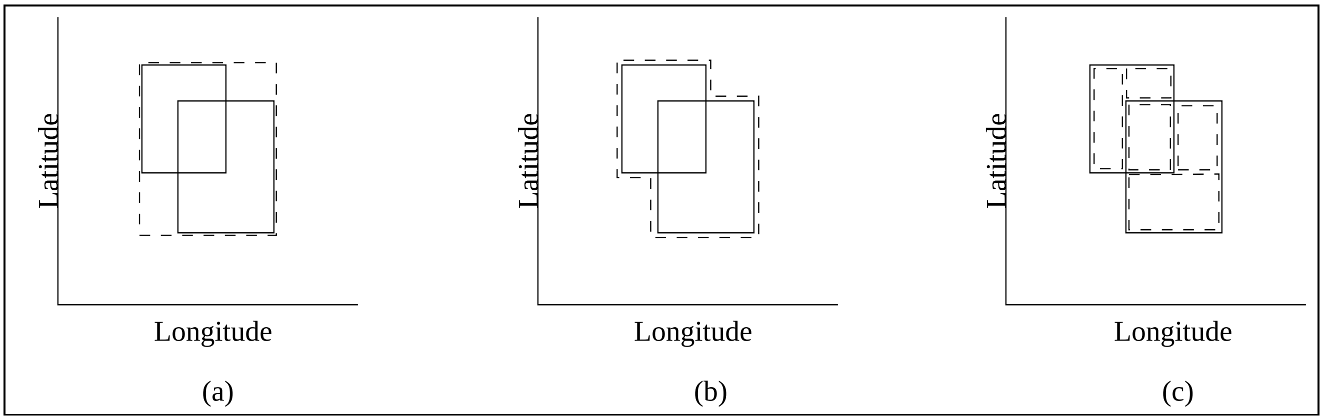

Figure 5: Three different merge procedures

procedure $mrg()$, and a set of queries Q. The output is a collection $\mathcal{M}$ such that the total cost, $cost(\mathcal{M})$, is minimized.

The cost of processing the queries and sending the answers back is represented by the total resources consumed. The total resources consumed are the sum of all the resources used by the server, the network, and the clients. The costs involved in our model can be summarized as follows:

- Server cost to run the merging algorithm.
- Server cost to process the merged queries.
- Cost of transmitting the answer of the merged query.
- Client cost of applying the extraction procedure.

In order to compute the amount of resources used, we need to estimate the size of query answers. Such estimate can be obtained using well-known database system techniques [MCS88]. We use $size(q)$ to denote the estimated size of q's answer. The total size of all the answers (equal to $size(mrg(M_1)) + size(mrg(M_2)) + ... + size(mrg(M_m))$) will be denoted as $size(\mathcal{M})$. This is the total amount of data that the server needs to transmit to the clients.

As stated before, clients need to apply an extraction query to each message they receive. We will denote the size of the irrelevant information for query q_i by $u_i = size(mrg(M_j)) - size(q_i)$, provided $q_i \in M_j$. We will call $U(Q, \mathcal{M})$ the sum of all u_i ($U(Q, \mathcal{M}) = \sum_{q_i \in Q}(u_i)$). We assume that the cost of applying the extraction query is proportional to the size of the irrelevant information contained in the message.

The amount of resources used by each component of the system can be computed as follows:

- Server cost: In a subscription service, the cost of the merging algorithm itself is insignificant when compared to the cost of running the queries many times. Therefore we will ignore the cost of executing the merging algorithm.

  The other component of the server cost is the time for retrieving the answers from the database. The data retrieval time depends on the number of queries in the collection $\mathcal{M}$, and on the size of the answers of the queries $mrg(M_1)$, $mrg(M_2)$, ..., $mrg(M_m)$. Here we assume that the cost of processing the queries is a weighted sum of the number of queries and the query sizes, where k_1 and k_2 are the proportionality constants.

  $Cost_{server} = k_1 * |\mathcal{M}| + k_2 * size(\mathcal{M})$

- Network cost:

  Our network model is initially based on a broadcast medium; namely, one where all messages are sent to all clients. In a later section, we will modify our model to allow other kinds of networks. The network cost will be proportional to two factors. First, the network resources consumed are proportional (by factor k_3) to size of the data being transmitted ($size(\mathcal{M})$). Since we expect the size of the header to be very small compared to the size of the data, we will ignore the size of the header when computing the size of an answer message. Second, in some cases we may need to establish network connections or "logical channels" for each M_i set. Messages then just include a logical channel id, and clients can subscribe to one or more channels. The cost of maintaining logical channels (e.g. table space in the routers, or operating system connection overhead) is proportional (by a factor k_4) to the number of messages transmitted.

  $Cost_{network} = k_3 * size(\mathcal{M}) + k_4 * |\mathcal{M}|$

- Clients' cost:

  The Clients' cost has two components. First, the cost of all clients having to apply the extraction function to the information received is proportional (by k_5) to the amount of irrelevant information received by the client. Second, as messages are broadcast, clients need to spend resources in checking if a message is directed to them; the cost of these resources is proportional (by factor k_6) to the number of merged queries and the number of clients.

  $Cost_{clients} = k_5 * U(Q, \mathcal{M}) + k_6 * num(Clients) * |\mathcal{M}|$

Using the three cost components, we can compute the total cost as:

$Cost_{total} = Cost_{server} + Cost_{network} + Cost_{clients}$

$Cost_{total} = k_1 * |\mathcal{M}| + k_2 * size(\mathcal{M}) + k_3 * size(\mathcal{M}) + k_4 * |\mathcal{M}| + k_5 * U(Q, \mathcal{M}) + k_6 * num(Clients) * |\mathcal{M}|$

$Cost_{total} = (k_1 + k_6 * num(Clients) + k_4) * |\mathcal{M}| + (k_2 + k_3) * size(\mathcal{M}) + k_5 * U(Q, \mathcal{M})$

If we assume that the number of clients is constant, we can introduce new constants K_M, K_T and K_U with the properties $K_M = (k_1 + k_6 * num(Clients) + k_4)$, $K_T = k_2 + k_3$ and $K_U = k_5$, so that

$Cost_{total} = K_M * |\mathcal{M}| + K_T * size(\mathcal{M}) + K_U * U(Q, \mathcal{M})$.

Here we are presenting a general cost model. In particular scenarios, some of these constants may be zero. In some cases particular constants or costs can be ignored.

5 The Query Merging Problem

In this section, we study the complexity of the query merging problem. First, we study the 2-Query Merging Problem, which is the special case of the query merging problem when $|Q| = 2$. We are considering the 2-Query Merging Problems because there are polynomial algorithms that can solve it. Then, we will consider the n-Query Merging Problem which, as we will see at the end of the section, is NP-Complete.

5.1 The 2-Query Merging Problem

We define the 2-Query Merging Problem as the query merging problem with $|Q| = 2$. In other words, using our cost model, we want to decide if it is worthwhile to merge two queries q_1 and q_2 into a merged query q_3.

For compactness, in the following discussion, let us denote $size(q_i)$ as S_i. Therefore, the cost of processing and transmitting queries q_1 and q_2 separately will be $K_M + K_T * S_1$ and $K_M + K_T * S_2$ respectively, for a total cost of $2K_M + K_T(S_1 + S_2)$. If we merge the queries into a single query q_3, the total cost will be $K_M + K_T * S_3 + K_U * U(Q, \mathcal{M})$, where $U(Q, \mathcal{M}) = 2 * S_3 - S_1 - S_2$. We derive the $U(Q, \mathcal{M})$ term in the following way: if we send a message with only $ans(q_1)$, the client receives an answer with size S_1. If we send a message with $ans(q_3)$ instead, the client will receive an answer of size S_3. The difference $(S_3 - S_1)$ is the size of the irrelevant results received by the client. We can use a similar derivation for the other client and conclude that the size of the irrelevant information for the other client is $(S_3 - S_2)$. Therefore the total size of irrelevant information is $U(Q, \mathcal{M}) = 2 * S_3 - S_1 - S_2$.

From these expressions, it is easy to derive a decision rule that tells us exactly when it is beneficial to merge two queries (this is, if the second cost we computed is less than the first cost). Therefore, it is beneficial to merge q_1 and q_2 if $K_M + K_T * [S_1 + S_2 - S_3] + K_U * [S_1 + S_2 - 2 * S_3] > 0$.

Unfortunately, the general problem ($|Q| > 2$) is significantly harder, since there are many ways to combine a set of queries into merged queries. For example, if we have three queries as input, it could be the case that it is not worthwhile merging any pair of them, but it is worthwhile merging the three queries into a single query. On the other hand, it could be the case that it is worthwhile merging one specific pair, but not the other pair and not the three queries. In conclusion, we would have to consider all possible ways to partition the input queries into subsets. For each possible partition we compute a cost, and then we pick the partition with minimum cost. This approach leads to an exponential algorithm. In fact, we will show, in the next section, that the query merging problem is NP-complete.

Let us use our geographical database scenario to show a case when merging three queries is optimal, although merging any pair is not. In Figure 6, we show three queries over our geographical database.

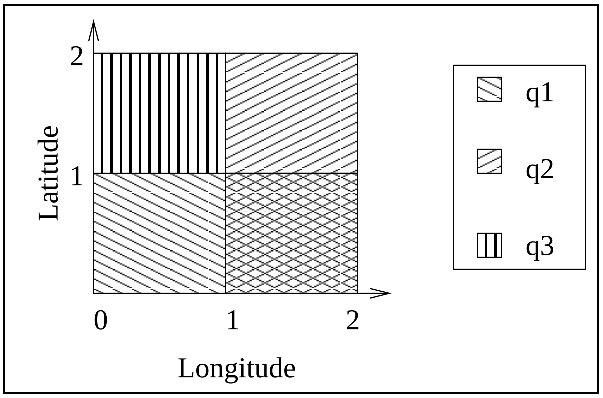

Figure 6: 3-query Merging Example.

In the following discussion, we will assume that the answer of a query over each square unit in the diagram has size S and that we are using the bounding rectangle merge procedure. Therefore, $size(q_1) = size(q_2) = 2S$, $size(q_3) = S$, and $size(mrg(q_1, q_3)) = size(mrg(q_2, q_3)) = size(mrg(q_1, q_2)) = size(mrg(q_1, q_2, q_3)) = 4S$. Since there are three queries, there are five ways to merge then: we can merge 2 of them (3 combinations), we can merge all of them, or we can keep them separately. The costs of each of the five merging cases are summarized below:

In the Appendix 1, we derive the costs of all the five possible ways of merging the queries. Merging all of the queries is advantageous, although merging any pair is not, when all of the following equations are satisfied:

$$S > \frac{K_M}{4K_U} \quad \text{and} \quad S > \frac{K_M}{5K_U - K_T} \quad \text{and} \quad S < \frac{2K_M}{7K_U - K_T} \tag{1}$$

These equations are satisfiable; for instance, if we pick $S = 1$, $K_M = 10$, $K_T = 9$, and $K_U = 4$ all the equations will be true.

5.2 Complexity of the Query Merging Problem

In this section we will prove that the Query Merging Problem is NP-complete. We will prove it by showing that an algorithm that solves the Query Merging Problem can also solve the Set Covering Problem, a well known NP-complete problem.

Definition of the Minimum Set Covering Problem (MSCP): Given a collection $\mathcal{L}$ of subsets of a finite set C, a set cover is a subcollection $\mathcal{L}' \subseteq \mathcal{L}$ such that the union of all subsets in $\mathcal{L}'$ is equal to C and $\mathcal{L}'$ has minimum size.

Example of MSCP: Given, $\mathcal{L} = \{\{1, 2\}, \{2, 3\}, \{1\}\}$ and $C = \{1, 2, 3\}$, the solutions of the MSCP are $\mathcal{L}' = \{\{1\}, \{2, 3\}\}$ and $\mathcal{L}' = \{\{1, 2\}, \{2, 3\}\}$.

In order to solve an instance of the MSCP using the Query Merging Problem, we need to map the input of the MSCP into the Query Merging Problem as follows:

- The set of queries Q will be equal to C
- The constants in the cost function, $Cost_{total} = K_M * |\mathcal{M}| + K_T * size(\mathcal{M}) + K_U * U(Q, \mathcal{M})$, will be $K_M = 0$, $K_U = 0$, and $K_T = 1$.
- We define symbols $\lambda_0, ...\lambda_{n-1}$ representing each element of L, and the symbol λ_n representing any set not in L. We define the merge function as:

$$mrg(S) = \begin{cases} \lambda_i & \text{if } S \text{ is the ith element of } L \\ \lambda_n & \text{otherwise} \end{cases}$$

Additionally, we will assume that the database manager will be able to process only the symbols $\lambda_0, ...\lambda_{n-1}$. The function $size(q)$ will be defined as 1 if the queries can be processed ($q = \lambda_0, ...\lambda_{n-1}$) or ∞ ($q = \lambda_n$) if not.

Under this assumption is easy to see that $cost(\mathcal{M})$ is transformed into:

$$cost(\mathcal{M}) = \begin{cases} \infty & \text{if } \exists M \in \mathcal{M} \text{ such that } M \notin \mathcal{L} \\ |\mathcal{M}| & \text{otherwise} \end{cases}$$

With this input, an algorithm that solves the query merging problem, will find a collection $\mathcal{M} = \{M_i\}$ such that the union of all M_is is C and $cost(\mathcal{M})$ is minimized. Therefore, $\mathcal{M}$ is a solution of the MSCP as minimizing $cost(\mathcal{M})$ means that the size of $\mathcal{M}$ is minimum and all $M_i \in \mathcal{L}$.

This proves that the query merging algorithm is NP-Complete with an arbitrary assignment of the K_M, K_T and K_U proportionality constants, as well as an arbitrary $size()$ function. However, there be might be values of the proportionality constants for which the problem is polynomial. Specifically, if $K_T = K_U = 0$, then the problem has a trivial solution: merge all queries into a single one.

6 Algorithms for the Query Merging Problem

In this section we introduce exhaustive and heuristic algorithms for the query merging problem. We will compare the performance of these algorithms in Section 9.2.

6.1 Exhaustive Algorithm

An exhaustive approach for solving the query merging problem is presented in Figure 7.

First, we form the superset of Q, $\mathcal{S}(Q)$, which contains all possible sets formed by combining elements of Q. Note that $\mathcal{S}(Q)$ contains all potential merges of the queries in Q. Then, we generate the superset of $\mathcal{S}(Q)$, $\mathcal{S}(\mathcal{S}(Q))$. The set $\mathcal{S}(\mathcal{S}(Q))$ is a superset of all the possible solutions to the query merging problem. An element of $\mathcal{S}(\mathcal{S}(Q))$ is a solution only if it provides a total cover of Q. That is, the element includes all the queries in Q. After eliminating all the elements that are not solutions, the final step of the algorithm is to use the cost model to evaluate the remaining elements of $\mathcal{S}(\mathcal{S}(Q))$. The solution with the lowest cost is the optimal solution. This algorithm is outlined in Figure 7.

1: Generate $\mathcal{S}(Q)$, the superset of the set Q.

2: Generate $\mathcal{S}(\mathcal{S}(Q))$, the superset of $\mathcal{S}(Q)$.

3: Create the set of solutions, A, with the elements of $\mathcal{S}(\mathcal{S}(Q))$ that form a total cover of Q.

4: Evaluate the cost of each element of A using the cost model and select the element with the minimum cost.

5: The element with the minimum cost is the optimal solution.

Figure 7: The Exhaustive Algorithm

The exhaustive algorithm solves the problem, but it has a doubly exponential complexity on the number of queries. Step 1 generates a set $\mathcal{S}(Q)$ with $2^{|Q|}$ elements. Then, step 2 generates a total of $2^{2^{|Q|}}$ elements. Therefore the algorithm will have a cost of $O(2^{2^{|Q|}})$ which is impractical for all but the smallest Q.

6.1.1 More Efficient Exhaustive Algorithms: The Partition Algorithm

The exhaustive algorithm presented in the previous sections is doubly exponential. However, there exists a better algorithm for exhaustively solving the query merging problem when the cost model ensures the *single-allocation property*: each q_i in the solution is in one and only one element of $\mathcal{M}$. The single-allocation property means that if we want to process a set of queries $\{q_1, q_2, q_3, q_4\}$, we do not need to consider merged queries such as $\mathcal{M} = \{\{q_1, q_2, q_3\}, \{q_1, q_4\}\}$ where a q_i (in this case q_1) is in more than one element of $\mathcal{M}$.

The cost model presented in Section 4 ensures the single-allocation property. We will show this by arguing that given a candidate solution $\mathcal{M}$ that does not have the single-allocation property, we can always build a new set $\mathcal{M}'$ that follows the single-allocation property and has the same or lower cost. The candidate solution $\mathcal{M}'$ is simply formed by eliminating all but one of the duplicate queries in $\mathcal{M}$. Obviously, the new candidate solution, is a valid solution of the problem (if the original solution was also a valid solution); and, it has the same or lower cost because eliminating a duplicate q_i, reduces or maintain the values of all the factors in our cost model. Specifically, $|\mathcal{M}'|$ can not increased as we are not adding new members to $\mathcal{M}$; $size(\mathcal{M}')$ is the same or lower as there is less data to transmit, and $U(Q, \mathcal{M})$ is also same or lower as each set potentially has less useless data.

We called the exhaustive algorithm that exploits the single-allocation property, the partition algorithm. Intuitively, the Partition Algorithm generates each possible candidate solution. The algorithm uses a search tree as an auxiliary data structure to ensure that each candidate solution is generated only once. In Figure 8, we present an example of the Partition Algorithm with input $Q = \{q_1, q_2, q_3\}$. We build the tree by considering one query at a time. At the end of each iteration, the leaves of the tree will contain all possible partitions of the queries considered up to that point. In the figure, in Step 0, the tree only has a single node with an empty list. In Step 1, we consider all partitions that can be created with q_1. In this case, the only possible partitions is $\{q_1\}$. In Step 2, we consider adding q_2 to the current partition. There are only two partitions, one that has q_1 and q_2 as separate queries and another, that merges them together. Similarly, in Step 3, we generate the partitions that include q_3. After all partitions are generated, in Step 4, we compute the cost of each one, and select the partition with the minimum cost (which is the optimal solution).

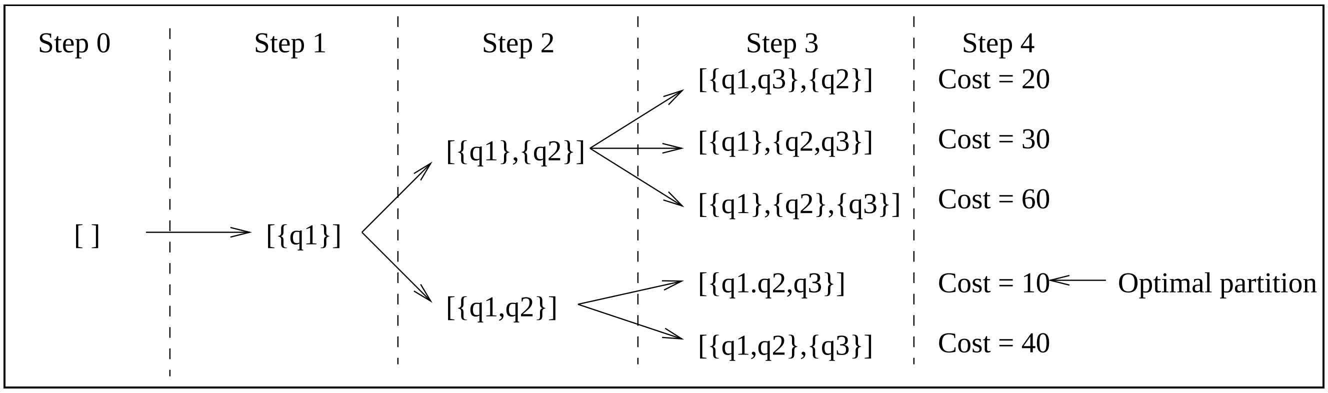

Figure 8: An Example of the Partition Algorithm

Formally, the algorithm associates to each node N of the search tree a list of partition of the queries $N.\mathcal{L} = [M_1, M_2, ..., M_m]$. The lists of partitions associated to the nodes at level h contain all possible partitions for the queries q_1, q_2, ..., q_h. At each step of the algorithm, for each node, we create as many children as elements are in the list of partitions plus one. Each of the children has the parent partition, but with the new element considered in that step, added to one of its partitions. The additional child, adds the new element as a separate partition. The algorithm is presented in Figure 9.

Input : $Q = \{q_1, q_2, ..., q_m\}$.
Output : Optimal solution for the query merging problem.

Create root node N with $N.\mathfrak{L} = []$. $h := 1$.

For $h = 1$ to $|Q|$

For each node N at level $h-1$ with $N.\mathfrak{L} = [M_1, M_2, ..., M_{|N.\mathfrak{L}|}]$
Create children N_0, N_1, ..., $N_{|N.\mathfrak{L}|}$ with $N_i.\mathfrak{L} = N.\mathfrak{L}$
Append $\{q_h\}$ to $N_0.\mathfrak{L}$
For each child N_i, $i \geq 1$ add q_h to the ith element of $N_i.\mathfrak{L}$

For each leaf node N evaluate the costs of $N.\mathfrak{L}$

The leaf with the minimum cost is the optimal solution.

Figure 9: Partition Algorithm

The number of cases that the partition algorithm needs to consider is equivalent to the number of ways in which n labeled balls can be put in n indistinguishable boxes. This number is known as the *Bell Number* which for large values of n is $O(n^n)$ [??]. Although this may seem as a small improvement over the exhaustive algorithm of the previous section; it significantly extends the values of n for which we can use an exhaustive algorithm. For example, if the partition algorithm takes 1 millisecond to find the optimal solution for $n = 6$, the exhaustive algorithm would take 30 centuries.

6.2 Heuristic Algorithms

6.2.1 Pair Merging Algorithm

The Pair Merging Algorithm takes a greedy approach to solve the query merging problem. The foundation of this algorithm are two simplifying assumptions. First, we assume that the cost model has the single-allocation property. Second, and more important, we assume that local decisions (i.e. deciding which pairs of queries to merge) will lead to the correct global solution. Note that the second assumption is, in general, incorrect (we showed in Section 5.1 that it does not hold). However, the assumption allows us to efficiently obtain solutions that, in practice, are very close to the real "optimal" solution (we will present performance results in Section 9.2). The complexity of the Pair Merging Algorithm is $O(|Q|^2)$.

The Pair Merging Algorithm maintains a set of sets of queries. Initially, each set contains each single query. Then for all pairs of sets the algorithm computes the change in the total cost when merging the two sets. The pair that produces the largest decrease in cost is chosen and the sets are replaced by their union. The algorithm continues picking and merging sets until no merging of any pair decreases the total cost. The decrease in total cost after merging sets can be by computed as using a generalization of the formula given in Section 5.1 for solving the 2-query merging problem.

Specifically, let us define M_a and M_b as the sets containing the queries $\{q_{a_1}, q_{a_2}, ..., q_{a_p}\}$ and $\{q_{b_1}, q_{b_2}, ..., q_{b_r}\}$ respectively; and M_m as the set formed by the union of these two sets ($M_m = \{q_{a_1}, q_{a_2}, ..., q_{a_p}, q_{b_1}, q_{b_2}, ..., q_{b_r}\}$).

The size of the answer of query q_i is given by $S_i = size(q_i)$. S_a and S_b are the total sizes of all the answers to the queries in sets M_a, M_b respectively. R_a, R_b and R_m are the total sizes of the answers of the queries which are obtained by merging all the queries in sets M_a, M_b and M_m respectively.

$S_a = S_{a_1} + S_{a_2} + ... + S_{a_p}$

$S_b = S_{b_1} + S_{b_2} + ... + S_{b_r}$

$R_a = size(mrg(\{q_{a_1}, q_{a_2}, ..., q_{a_p}\}))$

$R_b = size(mrg(\{q_{b_1}, q_{b_2}, ..., q_{b_r}\}))$

$R_m = size(mrg(\{q_{a_1}, q_{a_2}, ..., q_{a_p}, q_{b_1}, q_{b_2}, ..., q_{b_r}\}))$

$Cost_{old}$ is the total cost of processing the sets without merging them and $Cost_{new}$ is the total cost if we merge them.

$Cost_{old} = K_M + K_T * R_a + K_U * \{(R_a - S_{a_1}) + (R_a - S_{a_2}) + ... + (R_a - S_{a_p})\} + K_M + K_T * R_b + K_U * \{(R_b - S_{b_1}) + (R_b - S_{b_2}) + ... + (R_b - S_{b_r})\}$

$Cost_{old} = 2 * K_M + K_T * (R_a + R_b) + K_U * (p * R_a + r * R_b - S_a - S_b)$

$Cost_{new} = K_M + K_T * R_m + K_U * ((R_m - S_{a_1}) + (R_m - S_{a_2}) + ... + (R_m - S_{a_p}) + (R_m - S_{b_1}) + (R_m - S_{b_2}) + ... + (R_m - S_{b_r})\}$

$Cost_{new} = K_M + K_T * R_m + K_U * \{(p + r)R_m - S_a - S_b\}$

$Cost_{old} - Cost_{new} = K_M + K_T * (R_a + R_b - R_m) + K_U * \{p * R_a + r * R_b - (p + r)R_m\}$

In conclusion, the algorithm will merge the two sets with the highest positive value of $Cost_{old} - Cost_{new}$.

Note, that we can obtain the expression for solving the 2-query merging problem given in Section 5.1 by making $R_a = S_1$, $R_b = S_2$, $R_m = S_3$, and the number of queries in each set equal to one ($p = r = 1$).

The Pair Merging algorithm, as presented, needs to compute the cost of doing all possible merges in every step. However, in each step, only two of the sets (the ones that we decide to merge) have changed. The other sets remain the same so we can use all the computation involving them in the next step. Specifically, in step k of the algorithm, there are $(|Q| - k + 1) * (|Q| - k)/2$ possible pairs; of those, only $k - 1$ are new pairs. The rest were all candidate pairs that were considered in the previous iteration. Note, that the fact that those candidates were not chosen in a previous iteration, does not preclude them to be chosen later (as long as they have a positive benefit). To avoid computing the costs again for those sets, in each step, we save all computed costs in a *Profit Table*. Before computing the cost of merging a set, we check in the Profit Table to see if the cost was already computed; if it was, we take it from the table; if it was not, we compute it and add it to the table. At the end of each step, after selecting the pair of sets to be merged, we remove all entries of the Profit Table that are related with those sets.

1: $\mathcal{M} = \{\{q_i\}\}$ (put each q_i into a separate set M_i).

2: Initialize Profit Table by computing the change in total cost for all pairs of queries.

3: While there are positive entries in the profit table

   Find, using the profit table, the pair of sets M_a and M_b that decreases the total cost most.

   $\mathcal{M} = \mathcal{M} - M_a$; $\mathcal{M} = \mathcal{M} - M_b$

   $\mathcal{M} = \mathcal{M} \cup \{M_a \cup M_b\}$

   Update the profit table with the new possible pairs.

4: Return $\mathcal{M}$.

Figure 10: Pair Merging Algorithm

6.2.2 Directed Search Algorithm

The weakness of the Pair Merging Algorithm is that it can be trapped into a local minimum of the cost function and miss the global minimum. The Directed Search Algorithm tries to reduce this weakness by running the Pair Merging algorithm several times with a different initial state. At the end the algorithm outputs the lowest of all the "minimums" found. Additionally, instead of always merging pairs, we allow extracting one query from a set.

Intuitively, in each step, the algorithm consider all possible pairs constructed by merging two sets or separating them.

The Directed Search Algorithm is presented in Figure 11.

The complexity of this algorithm is $O(|Q|^2 * T)$, where T is the number of times we run the algorithm. If T is a constant, then the complexity is $O(|Q|^2 * T)$.

1: For $i = 1$ to T:

2: Start with a random state.

3: do:

4: For each pair of sets in $\mathcal{M}$ compute the cost of combining them.

5: For each query q of each set S in $\mathcal{M}$ compute the costs of $S - \{q\}$.

6: Perform the operation that minimizes the cost the most.

7: Repeat until no beneficial operation is possible.

Figure 11: Directed Search Algorithm

6.3 Clustering Algorithm

The Clustering Algorithm takes a "divide and conquer" approach to the query merging problem. The foundation of the algorithm is the definition of a "distance" metrics between queries. If the distance between two queries is "far" enough, we can safely ignore all combinations of merged queries that contain those queries. A graphical intuition of this approach is presented in Figure 12.

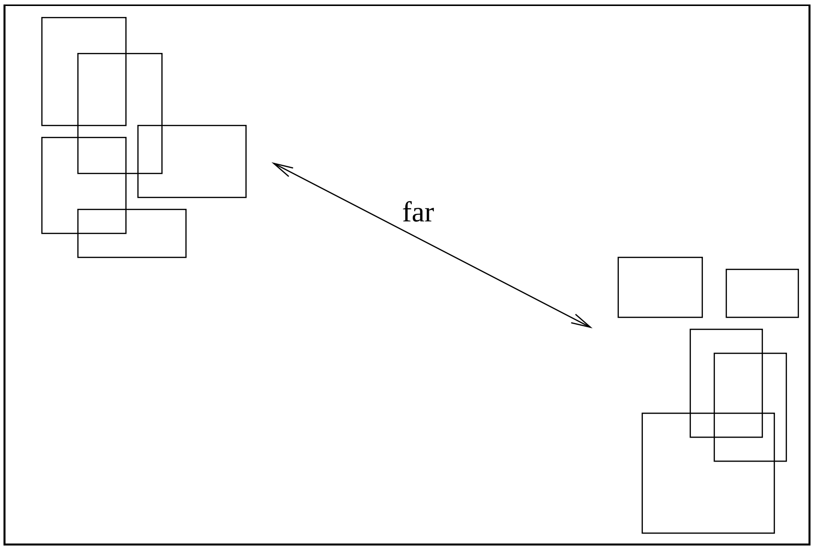

Figure 12: Clustering Algorithm Scenario

Given 2 queries, q_i and q_j, the decision of merging them together (with whatever other queries they have already been merged) will have the following impact in the cost model:

- $K_M * |\mathcal{M}|$: In the best case, by merging these two queries together, $|\mathcal{M}|$ will drop to 1.
- $K_T * size(\mathcal{M})$ In the best case (namely when q_1 is equal to q_2), the change in $size(\mathcal{M})$ will be 0.
- $K_U * U(Q, \mathcal{M})$ In the best case, $U(Q, \mathcal{M})$, will only increase by $2 * size(mrg(\{q_1, q_2\})) - size(\{q_1\}) - size(\{q_2\})$.

Therefore, two queries can only be at the same time in the same merged set when:

$K_M + K_U * 2 * size(mrg(\{q_1, q_2\})) - size(\{q_1\}) - size(\{q_2\}) > 0$.

If we can compute the intersection of two queries, we can have a more restricted condition. Specifically, if we define "$\cap$" in the usual way (this is $q_3 = q_1 \cap q_2$ if and only if $ans(q_3) = ans(q_1) \cap ans(q_2)$), then the decision of merging them together will change, in the best case, the second factor to $2 * size(mrg(\{q_1, q_2\})) - size(\{q_1\}) - size(\{q_2\} + size(\{q_1 \cap q_2\})$.

Therefore, in this case the condition is:

$$K_M + (K_T + K_U) * 2 * size(mrg(\{q_1, q_2\})) - size(\{q_1\}) - size(\{q_2\}) + K_T * size(\{q_1 \cap q_2\}) > 0$$

7 Query Merging in a Multiple Channel Broadcast Environment

7.1 Overview

In this section, we will explore a richer model of the network. In this new model, we have a transmission mechanism that disseminates data through a fixed number of physical multicast channels. This mechanism can be a satellite with a fixed number of broadcast channels as described in the BADD scenario. In this extended model, we are given the same initial set of queries as before. We have a fixed set of channels Ch_1, Ch_2..., Ch_c to disseminate the answers to queries. The answers are broadcast on one of the channels, and clients who need that information are told to listen to that channel. A client can listen to one or more of these channels. In this case, along with the data sent through the channels, we can add to the header client identifiers that are supposed to get the information, or alternatively the clients can check all the data that goes through the channel and filter it. The second approach requires additional computation overhead at the client site. However, in a dynamic scenario where the queries are added and removed from the system, the client can cache the incoming data and check if the information already exists before requesting data.

7.2 Problem Description

In the model described earlier, we assumed that the answers of the queries were sent to specific sets of clients and that the network broadcast all messages. In this new model, we consider the network as a sets of multicast channels. We define a multicast channel network, as one where messages can be sent to (and only to) a specific set of clients. We will assume that clients can listen to only one channel and the answers to the queries are sent through a limited number of multicast channels. In addition to the query merging problem, we now have to allocate channels to clients. We cannot consider these two problems separately as shown in the following example:

Say we have the queries q_1, q_2, ..., q_5 with the corresponding clients associated with them:

c_1 : q_1,q_2

c_2 : q_3,q_4

c_3 : q_5

Suppose after running the merging algorithm, the queries are distributed into a collection $\mathcal{M} = \{M_1, M_2, M_3\}$ as follows:

$M_1 = \{q_1, q_5\}$

$M_2 = \{q_2, q_3\}$

M_3 ={q_4}

Each destination has to receive the answers of the queries requested. Let us consider a system with two channels. Say we allocate M_1 and M_2 to Channel 1 and M_2 and M_3 to channel 2. Client 1 can listen to channel 1, client 2 can listen to channel 2 and client 3 can listen to channel 1.

M_1, $M_2 \rightarrow Ch_1 \rightarrow c_1$, c_3

M_2, $M_3 \rightarrow Ch_2 \rightarrow c_2$

However, the answers of the queries q_3 in channel 1 and q_2 in channel 2 is not required by the corresponding clients. Thus we can reduce the size of data transmitted by using the following allocation of queries to channels:

M_1, $q_2 \rightarrow Ch_1 \rightarrow c_1$, c_3

q_3, $M_3 \rightarrow Ch_2 \rightarrow c_2$

This example shows that we can not consider the query merging and channel allocation problems separately for an efficient information delivery using multiple channels.

8 Algorithms for Channel Allocation

In this section, we will introduce two kinds of algorithms; the exhaustive algorithm, and the heuristic algorithm. The exhaustive algorithm checks all possible cases and is guaranteed to find the optimal solution. However, for a large number of clients and channels, the complexity of this algorithm makes it impractical to use. The heuristic algorithm reduces the number of cases considered. Although the solution found is a good one, it may not be the optimum.

8.1 Exhaustive Algorithm for Channel Allocation

The exhaustive solution of this problem is to generate all possible allocations of clients to channels. Given clients $C = \{c_1, c_2, ..., c_p\}$, we try to generate all cases by generating a search tree. The leaves of this tree gives us all possible allocations.

We associate to each node N of this tree a list of sets of clients $N.L = [C_1, C_2, ..., C_m]$ where C_i is the set of clients which are allocated to channel Ch_i. The sets of clients associated to the nodes at level h contain all possible combinations for the clients c_1, c_2, ..., c_h allocated to the channels.

The algorithm is presented in Figure 13.

The cost of answering the queries sent by clients C_i through channel Ch_i can be computed using our cost model and applying the Pair Merging Algorithm (described in section 5.3.1) to all the queries sent by the clients C_i. The total cost at leaf N is computed by summing up each individual cost associated with the sets of clients $N.L$. The leaf with the minimum cost gives the allocation that is the optimal solution.

Input : $C = \{c_1, c_2, ..., c_p\}$, $NumberOfMulticastChannels$
Output : Optimal solution for the query merging problem.

1: Create root node N with $N.L = []$. $h := 1$.

2: For each node N at level $h - 1$ with $N.L = [C_1, C_2, ..., C_{|N.L|}]$

   2.1 If $|N.L| < NumberOfMulticastChannels$, create child N_0 with $N_0.L = N.L$ and append c_h to $N.L$.

   2.2 Create children N_1, ..., $N_{|N.L|}$ with $N_i.L = N.L$

   2.3 For each child N_i, $i \geq 1$ add $\{c_h\}$ to the ith element of $N_i.L$

3: $h = h + 1$. If $h \leq |C|$ go to step 2.

4: For each leaf node N evaluate the cost for the distribution $N.L$.

5: The leaf with the minimum cost is the optimal solution.

Figure 13: The Exhaustive Algorithm for Channel Allocation

8.2 Heuristic Algorithms for Channel Allocation

The goal of this algorithm is to find a distribution of the users to channels that will minimize the total cost associated with the processing and transmission of the queries. This algorithm starts with an initial distribution of clients to channels. Then it reallocates the clients by moving them between channels in the direction of decreasing total cost. The algorithm uses the hill climbing technique in which we move through the search space in the direction which reduces the cost. It does not guarantee to find the optimal solution since it can get stuck at a local minimum in the search space.

First, we need to come up with the initial distribution of clients to channels. We can increase the probability of finding an optimal solution and can decrease the computation time by putting clients that have more overlapping query answers in the same channel.

Let $Q_C = \{q_1, q_2, ..., q_n\}$ be a set of queries sent by clients $C = \{c_1, c_2, ..., c_p\}$ and $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ be the collection computed using the Pair Merging Algorithm. The cost of answering the queries sent by the clients C using a single channel can be calculated as follows:

$Cost_C = K_M * |\mathcal{M}| + K_T * size(\mathcal{M}) + K_U * \sum_{i \le m} \sum_{q_j \in M_i} (size(M_i) - size(q_j))$

The reduction in cost, denoted by $Cost_\Delta$ by putting the clients c_a and c_b in the same channel is:

$Cost_\Delta = Cost_{\{c_a\}} + Cost_{\{c_b\}} - Cost_{\{c_a, c_b\}}$. We denote the current channel for assigning next client by CCh and a list consisting of $[c_a, c_b, Cost_\Delta]$ triples by L.

The algorithm is as follows:

$CCh := 1$. Calculate the $Cost_\Delta$ for all pairs c_a and c_b. Store the triple $[c_a, c_b, Cost_\Delta]$ in L.

While L is not empty

> Pick the triple $[c_a, c_b, Cost_\Delta]$ in L with maximum $Cost_\Delta$ value. Allocate clients c_a and c_b to channel CCh. Delete all the triples that have either c_a or c_b from L.
> $CCh = CCh + 1 (mod c)$.

While there is a client c_i that is not assigned to any channel

> Put c_i in channel CCh.
> $CCh := CCh + 1 (mod c)$

Figure 14: The Algorithm for Initializing Distribution

After finding the initial distribution, we decrease the total cost further by iteratively making changes. The algorithm proceeds as follows:

Two lists are associated with each channel Ch_i. List L_i keeps track of the clients that listen to Ch_i, and list I_i keeps track of the queries whose answers are sent through Ch_i. The queries in each channel are merged using the Pair Merging Algorithm and the answers are sent through that channel.

- For each channel, we run the merging algorithm with the queries of the clients using that channel given as input. Using the collection $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ given for this channel, we calculate the cost associated with this channel.

  $Cost_{Ch_i} = K_M * |\mathcal{M}| + K_T * size(\mathcal{M}) + K_U * U$

  We store the costs associated with each channel in a table T.

- For each client c_i and channel Ch_j, we consider the decrease of cost by moving that client c_i to channel Ch_j. Suppose the client c_i is in channel Ch_k $(k \neq j)$. In order to find the reduction in cost, we calculate the new costs associated with the channels Ch_k and Ch_j. We subtract these costs from the previous costs of these channels stored in T. We choose the client and the channel that caused the most reduction in cost, If there is such a client,we move client c_i to channel Ch_j, update the lists L_k, I_k, L_j, I_j and T. Then we repeat this step.

- For each channel Ch_i, we allocate the channels by assigning the clients in L_i to channel Ch_i.

9 Performance Evaluation

In order to test the efficiency of the algorithms developed, an application program has been implemented. The application simulates an environment in which the queries are given on a two-dimensional database (see Figure 15. The database elements consist of two attributes and the queries received by the server are range queries over the database. The application consists of three main modules. The first module provides input to the system. It generates a set of clients which send queries to the system. The second module runs the query merging algorithm for a given set of queries and parameters as input for considering only one channel for transmission. The final module finds the distribution of users to channels that will minimize the overall cost.

9.1 Generating Input

In most environments, it is quite likely that the input given by clients generates a pattern which creates groups of queries that are located near each other. Some portions of the database are likely to be more dense than others. For instance if the database specifies the atmospheric conditions, the portions of the database denoting the regions with higher population is more likely to get query requests. Similarly, in a battlefield situation, the queries in regions which have more combat troops will be much denser. Therefore, rather than generating random input queries, a clustering effect

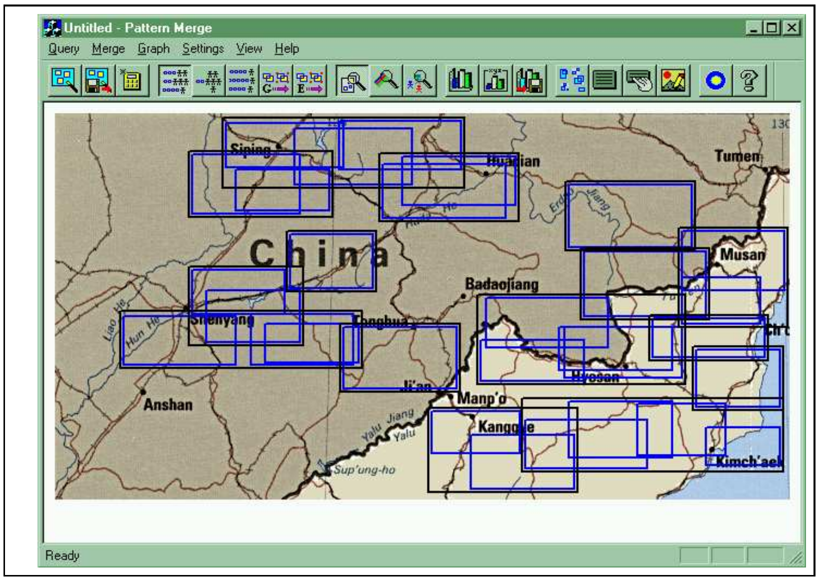

Figure 15: Screen Shot of the Simulator

has been added to the input generating module, in which some queries tend to create small crowds. The input generated is a hybrid of random and clustering queries.

The input is generated depending on three variables namely cf, sf and df. The clustering effect is given by the parameter cf. cf is the ratio of the number of queries generated using clustering to the number of queries in total. sf is the ratio of number of queries in a cluster to the number of queries generated using clustering. df is the density of the clusters. For each cluster, a random origin is generated on the two-dimensional database and the queries associated with that cluster are distributed using normal distribution around the origin.

$$f(r) = \frac{1}{\sqrt{2\pi df}} e^{-\frac{r^2}{2df^2}}$$

r is the distance of the query to the origin. This function gives the ratio of queries at a particular distance from the origin to the total number of queries.

Another parameter in generating input is the size of the queries. The minimum and maximum ranges for both attributes are given. The size of each query is selected randomly between these ranges.

9.2 Experiments

In this section we study the performance of the Query Merging Algorithm and Channel Allocation Algorithm. The performance was tested with the simulator describe previously. Sample queries were generated and both the exhaustive and heuristic algorithms were used to evaluate the results.

The main parameters of our model are follows:

- Query Generating Variables: cf, sf and df.
- Cost Variables: K_M, K_T, K_U, K_D.
- Database Variables: Range of the attributes of the database.
- Query Variables: Number of queries received by server, the minimum and maximum query sizes.
- Channel Allocation Variables: Number of users and number of multicast channels.

Obviously the exhaustive algorithms give the optimal solution for a given set of queries. Therefore we can not expect to find the solutions our the algorithms described to be superior to the optimal solution. In order to evaluate the efficiency of our algorithms, we wish to address the following questions:

- What is the probability that the algorithms find the optimal solution?
- If the algorithms do not find the optimal solution, how far are the solutions to the optimal ones?

For a given set of queries let us say that the initial cost of answering them without merging is $Cost_{initial}$. The exhaustive algorithm considers all possible answers and picks up the one with the minimum cost. The heuristic algorithm finds a solution whose cost is not more than the initial cost and not less than the optimal cost. We denote these costs by $Cost_{optimum}$ and $Cost_{heuristic}$ respectively. We measure the distance of the heuristic solution to the optimal solution as follows: $Performance_{heuristic} = \frac{Cost_{heuristic} - Cost_{optimum}}{Cost_{initial} - Cost_{optimum}}$

This formula gives the relative cost of the heuristic algorithm compared to the initial and optimal solutions. For instance a value of 0.0% indicates the solution is optimum and a value 100.0% indicates the cost is the same as the initial cost which means there is no merging.

9.3 Performance of Query Merging

The Pair Merging Algorithm for Query Merging gives us the queries that should be merged based on our cost model. The parameters have a significant affect of the solution generated. For some queries, the pair merging algorithm does not find the optimal solution for certain values of cost constants as described in Section 4.3. In order not to get too optimistic results, we ran the simulator for different sets of parameters. We observed for some sets of parameters (cf, sf, df, K_M, K_T, K_U, K_D) the solution was further from the optimum. We picked these parameters as constants. The remaining parameters (database size, query size, number of queries) were ranged over a fixed interval.

The number of possible combinations to merge n queries is equal to $B(n)$, which is the nth Bell number. For large values of n, $B(n) \approx n^n$ For example, for $n = 12$, $B(n) = 4,213,597$, for $n = 15$, $B(n) = 1,382,958,545$, for $n = 30$, $B(n) \approx 10^{23}$. These values indicate that it is futile to obtain optimal solutions to the query merging problem with exhaustive search. Therefore the number of queries were selected to be equal or fewer than 12 which makes the computation feasible.

For two queries, there are two options, merging these two queries and not merging them. Therefore we omitted this trivial case as the Pair Merging Algorithm is guaranteed to find the best solution for 2 queries. Figure 16 shows the probability that the pair merging algorithm finds the optimal solution. On the average this probability is 97%. Figure 17 shows the distance of the pair merging solution to the optimal solution. On the average this value is 0.6343%.

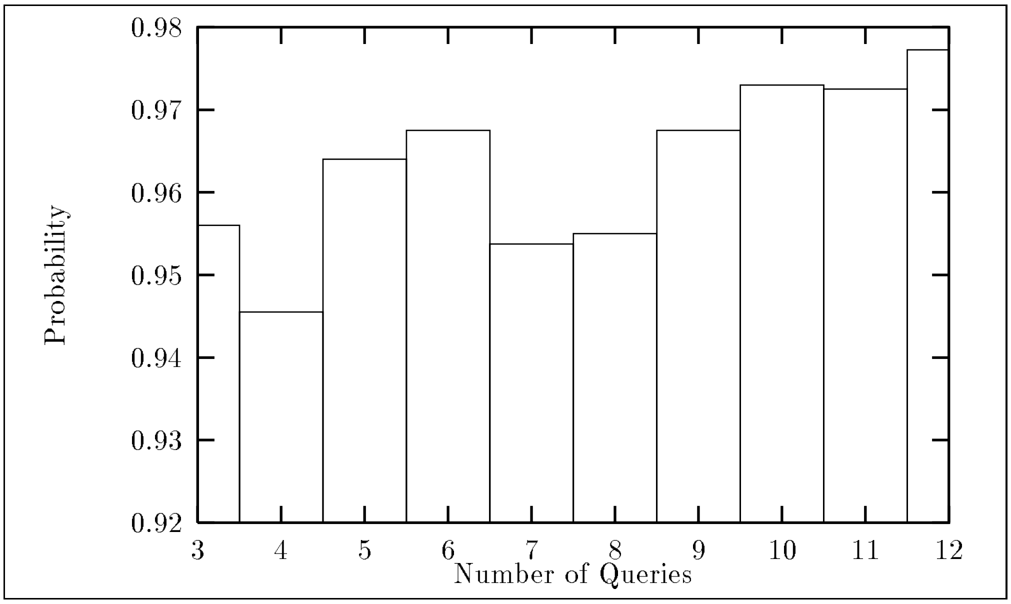

Figure 16: Probability of finding optimal solution for the Pair Merging Algorithm

9.4 Performance of Channel Allocation

The Heuristic algorithm for channel allocation gives us the distribution of the users to channels. In this model we have additional parameters such as the number of channels and the number of clients. It is very difficult to answer these questions in absolute terms as there are many parameters to consider. Therefore, we considered representative scenarios to evaluate the performance. In the channel allocation algorithm, the merging process is applied to each channel separately. The number of possible combinations of allocating u users to ch channels and merging the n queries in this channel then becomes $ch^u * B(n/u) = O(ch^u * n^n)$ assuming that each user sends an equal number of queries. The exhaustive algorithm and the heuristic algorithm finds a distribution by computing the costs associated with each channel. We can apply the pair merging algorithm to the

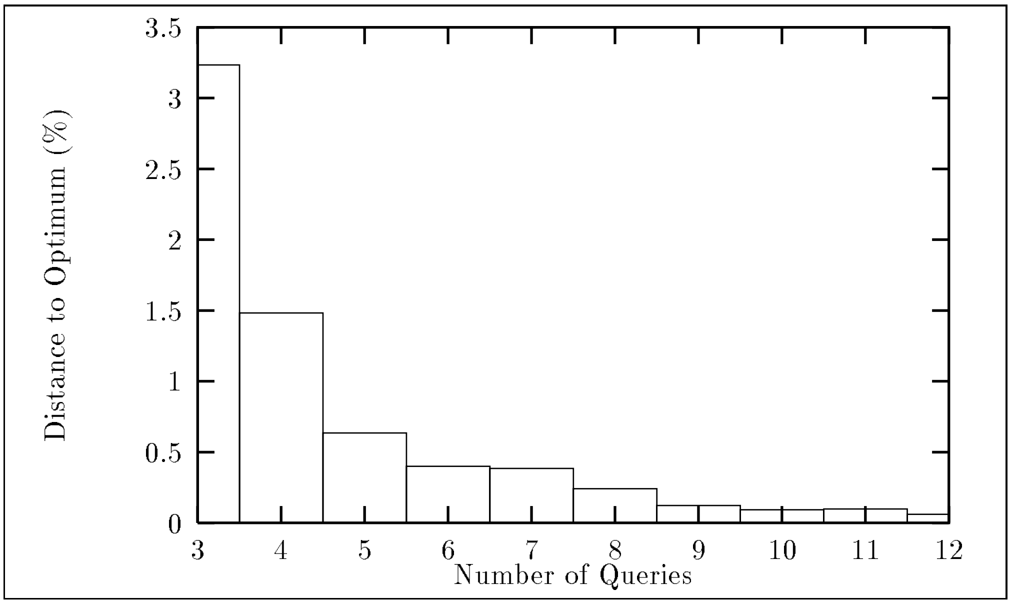

Figure 17: Distance to optimal solution for the Pair Merging Algorithm

queries in each channel which will give us the complexity of $O(ch^u * n^2)$. This will not effect the overall performance result of the channel allocation algorithm since changing the merge function does not effect the allocation algorithm. Thus using the pair merging function for merging will lead to less computation time. Hence we can increase the number of queries.

The Heuristic algorithm for channel allocation uses the hill climbing approach. We evaluated the performance of this algorithm with 2 different starting points (initial distributions). The first one is uses the algorithm described in section 8.2. The second one picks up a random distribution as the initial distribution. However, we can improve the result by applying the algorithm with different starting points and picking up the solution that gives the lowest cost. Figure 18 summarizes the probability of finding the optimal solution depending on the initial starting point. If the initial distribution is random, the probability is 85.5% and if the initial distribution takes into account minimizing starting cost, the probability is 81.8%. As we can see, starting at a random node in the search space decreases the chance of getting stuck at a local minima. If we use both approaches and pick the better solution, we can get a slightly better result which is 88.6%. We also measured the performance of this algorithm using the same distance metric. As Figure 4 shows, the average distance of the solution to the optimal one which is 0.1697% on the average.

10 Related Work

- Julie Basu

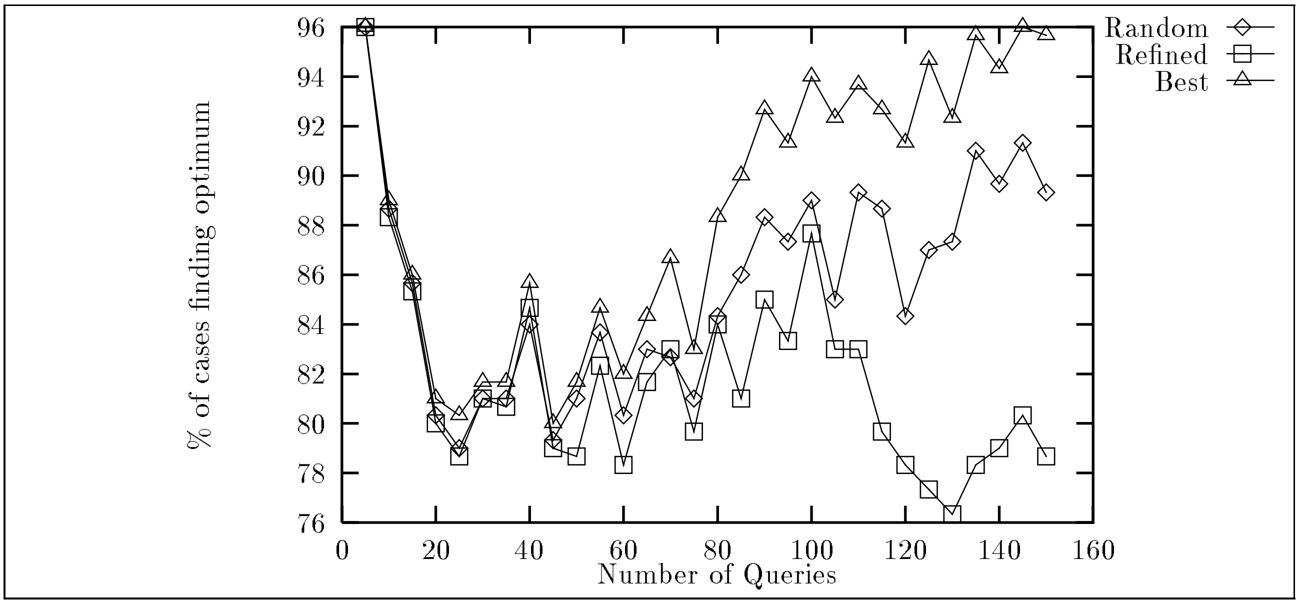

Figure 18: % of cases finding the optimal solution

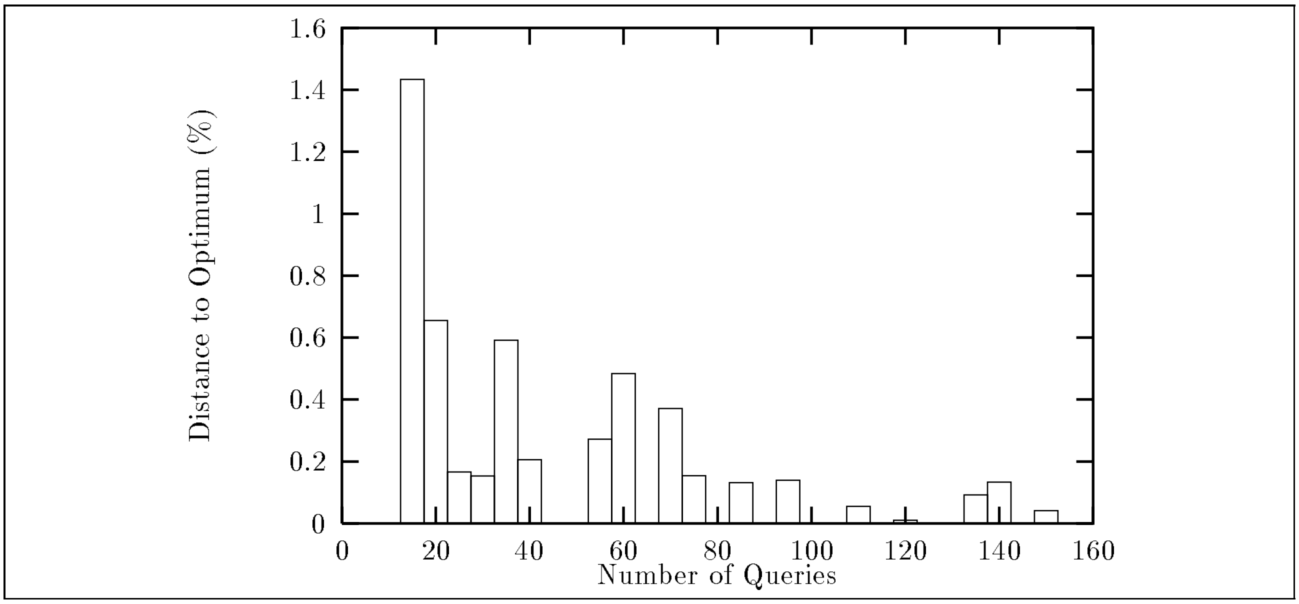

Figure 19: Distance to Optimal Solution for Exhaustive Channel Allocation Algorithm

- Predicate Locking
- Conjunctive Queries
- SATIN Project: Data Engineering Bulletin [DP96b]
- Badd Project Related Conferences. The papers from these two conferences are very general, but they are good citation when defining the "BADD" problem. MILCOM conference hold at Monterey, CA, USA 2-5 Nov. 1997. Sponsored by IEEE and AFCEA (Armed Forces Communication & Electronic Association) [BB97][LBD+97][SB97][Lu97a][DSLL97]. BADD Conference (SPIE) Digitization of the Battlefield II Orlando, FL, USA 22-24 April 1997. Sponsored by SPIE (The International Society for Optical Engineering) [Lu97b][SDSV97][LS97][DMS97].
- Cellular Telephony/Telecommunication research. Low level approach to the problem. They don't deal with subscription or repeated queries (except one paper, where "queries" are request for a data page). Some interesting algorithms in the area of channel allocation. However, they don't attempt to do query merging. [HLL98][CM97][LHL97].
- Data Dissemination Research: A very active group in Maryland (Franklin and Zdonik). First with research in "brodacast disks," then, in a more general data dissemination framework. The OOPSLA paper is a good introduction to the topic [FZ98][FZ97][AFZ97][AAFZ95][FZ96].
- Good snapshot of 1996 state of the art: Data Engineering Bulletin, Volume 19, Number 3, September 1996 Special Issue on Data Dissemination [DP96b][DP96a][Bes96].
- Stanford References: SIFT [YGM95].
- Data Dissemination Companies: Now that I understand this area better, I'm surprised how weak current companies are (maybe with the exception of TIBCO). Most of their "push" technology is nothing more than a "automatic pull," which is translated in millions of unicast connections (which are obviously impossible to scale). This is a consequence of the weakness of IP and (in a lesser extent HTTP). Definetely a good business opportunity here. Companies include TIBCO (http://www.tibco.com) [Cha98], Pointcast (http://www.pointcast.com) [RD98], Backweb (http://www.backweb.com), Marimba (http://www.marimba.com), Airmedia (http://www.airmedia.com).
- Other references: [MCS88], [CR88] [SS72] [TTS+98]

11 Future Work

- non uniform object space.
- dynamic scenario:
- queries are continuous, and return new objects added to database. I think the analysis should be the same, except that instead of talking about objects returned by queries, we talk of objects per second disseminated.

- We already have a set of queries that have been merged, and a new query arrives. Can we incrementally compute a new partition, without starting from scratch?
- different dissemination mechanism This one is explained below, and introduces some issues that are orthogonal to query merging.
- cashing of queries in the client
- splitting a query between 2 clients:

  Example: given this three queries: $0 < x < 3$, $0 < x < 4$, $x < 2$. A possible way to merge them is to make q'_1: $0 < x < 4$ and q'_2 $x < 4$, to compute q_3, we need to use both q'_1 and q'_2. This complicates the problem and shows a more complicated relation between the set covering and the predicate finding problem.

12 Conclusions

Appendix 1

- Not merging any query:

  $Cost = 3K_M + K_T(2S + 2S + S)$

  $Cost = 3K_M + 5K_T S$

- Merging q_1 and q_2 :

  $Cost = K_M + K_T size(q_3) + K_M + K_T size(mrg(q_1, q_2)) + K_U(2size(mrg(q_1, q_2)) - (size(q_1) + size(q_2)))$

  $Cost = 2K_M + 5K_T S + 4K_U S$

- Merging q_1 and q_3 (Same as merging q_2 and q_3):

  $Cost = K_M + K_T size(q_2) + K_M + K_T size(mrg(q_1, q_3)) + K_U(2size(mrg(q_1, q_3)) - (size(q_1) + size(q_3)))$

  $Cost = 2K_M + 4K_T S + 5K_U S$

- Merging all queries:

  $Cost = K_M + K_T size(mrg(q_1, q_2, q_3)) + K_U(3size(mrg(q_1, q_2, q_3)) - (size(q_1) + size(q_2) + size(q_3)))$

  $Cost = K_M + 4K_T S + 7K_U S$

In conclusion, merging q_1, q_2 and q_3 is more advantageous than merging any pair if:

$$
\begin{aligned}
3K_M + 5K_TS &< 2K_M + 5K_TS + 4K_US \\
3K_M + 5K_TS &< 2K_M + 4K_TS + 5K_US \\
3K_M + 5K_TS &> K_M + 4K_TS + 7K_US
\end{aligned}
$$

As a result if the following equations are satisfied, it will be optimal to merge all three elements but it's not beneficial to merge any pair.

$$
\begin{aligned}
S &> \frac{K_M}{4K_U} \\
S &> \frac{K_M}{5K_U - K_T} \\
S &< \frac{2K_M}{7K_U - K_T}
\end{aligned}
$$

These equations are satisfiable; for instance, if we pick $S = 1$, $K_M = 10$, $K_T = 9$, and $K_U = 4$ all the equations will be true.

References

[AAFZ95] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communication environments. In *1995 ACM SIGMOD International Conference on Management of Data*. ACM, May 1995.

[AFZ97] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *SIGMOD 1997. ACM SIGMOD International Conference on Management of Data*. ACM, May 1997.

[BB97] F. Bauer and M. Brown. A reliable multicast transport protocol for a global broadcast service-based network. In *MILCOM 97*, volume 2, pages 988–92. IEEE, 1997.

[Bes96] A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proceedings of ICDE'96*, March 1996.

[Cha98] A. Chan. Transactional publish/subscribe: the proactive multicast of database changes. In *1998 ACM SIGMOD International Conference on Management of Data*, June 1998.

[CM97] M.S. Corson and J.P. Macker. Global quality of service-based and reliable data dissemination via asymmetric direct broadcast satellite channels. In *Proceedings of 19th Pacific Telecommunications Conference (PTC'97)*, Jan 1997.

[CR88] Joseph C. Culberson and Robert A. Reckhow. Covering polygons is hard (preliminary abstract). In *29th Annual Symposium on Foundations of Computer Science*, pages 601–611, White Plains, New York, oct 1988. IEEE.

[DMS97] R.J. Douglass, J. Mork, and B.R. Suresh. Battlefield awareness and data dissemination (badd) for the warfighter. In *Proceedings of the SPIE*, volume 3080, pages 18–24, 1997.

[DP96a] Son Dao and Brad Perry. Efficient dissemination of information on the internet. *Bulletin of the Technical Committee on Data Engineering*, 19(3):48–54, September 1996.

[DP96b] Son Dao and Brad Perry. Information dissemination in hybrid satellite/terrestrial networks. *Bulletin of the Technical Committee on Data Engineering*, 19(3):12–18, September 1996.

[DSLL97] J. Dukes-Schlossberg, Yongwon Lee, and N. Lehrer. Iids: intelligent information dissemination server. In *MILCOM 97 Proceedings*, volume 2, pages 635–9. IEEE, 1997.

[FZ96] Michael Franklin and Stan Zdonik. Dissemination-based information systems. *Bulletin of the Technical Committee on Data Engineering*, 19(3):20–30, September 1996.

[FZ97] M. Franklin and S. Zdonik. A framework for scalable dissemination-based systems. In *OOPSLA '97. Object Oriented Programming Systems Languages and Applications.* ACM, Oct 1997.

[FZ98] M. Franklin and S. Zdonik. "data in your face": push technology in perspective. In *1998 ACM SIGMOD International Conference on Management of Data*, June 1998.

[HLL98] Qinglong Hu, Dik Lun Lee, and Wang-Chien Lee. Optimal channel allocation for data dissemination in mobile computing environments. In *Proceedings of 18th International Conference on Distributed Computing Systems*, May 1998.

[LBD+97] R. Lindell, J. Bannister, C. DeMatteis, M. O'Brien, J. Stepanek, M. Campbell, and F. Bauer. Deploying internet services over a direct broadcast satellite network: challenges and opportunities in the global broadcast service. In *MILCOM 97 Proceedings*, volume 2, pages 906–10. IEEE, 1997.

[LHL97] W.-C. Lee, Qinglong Hu, and Dik Lun Lee. Channel allocation methods for data dissemination in mobile computing environments. In *Proceedings. The Sixth IEEE International Symposium on High Performance Distributed Computing.* IEEE, Aug 1997.

[LS97] M.B. Lazaroff and P.A. Sage. Any information, anywhere, anytime for the warfighter. In *Proceedings of the SPIE*, volume 3080, pages 35–42, 1997.

[Lu97a] N.H. Lu. Network interface to tactical communications. In *MILCOM 97 Proceedings*, volume 2, pages 645–9. IEEE, 1997.

[Lu97b] N.H. Lu. Network interfaces to tactical communications. In *Proceedings of the SPIE*, volume 3080, pages 59–70, 1997.

[MCS88] Michael V. Mannino, Paicheng Chu, and Thomas Sager. Statiscal profile estimation in database systems. *ACM Computing Surveys*, 20(3), September 1988.

[RD98] S. Ramakrishnan and V. Dayal. The pointcast network. In *1998 ACM SIGMOD International Conference on Management of Data*, June 1998.

[SB97] B. Schein and S. Bernstein. A forward error control scheme for gbs and badd. In *MILCOM 97 Proceedings*, volume 2, pages 710–15. IEEE, 1997.

[SDSV97] T.P. Stephenson, B. DeCleene, G. Speckert, and H.L. Voorhees. Badd phase ii. dds information management architecture. In *Proceedings of the SPIE*, volume 3080, pages 49–58, 1997.

[SS72] H.M. Salkin and J. Saha. Set covering: algorithms, results and codes. In *Bulletin of the Operations Research Society of America*, volume 20, suppl.2, Nov 1972.

[TTS+98] Min Tan, Mitchell D. Theys, Howard Jay Siegel, Noah B. Beck, and Michael Jurczyk. A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment. In *Proceedings of teh 7th International Computing Workshop (HCW'98)*. IEEE, 1998.

[YGM95] T.W. Yan and H. Garcia-Molina. Sift-a tool for wide-area information dissemination. In *Proceedings USENIX Winter 1995 Technical Conference*, Jan 1995.