Optimized Query Planning of Continuous Aggregation Queries in Dynamic Data Dissemination Networks

Rajeev Gupta
IBM India Research Lab
New Delhi, India
grajeev@in.ibm.com

Krithi Ramamritham
Indian Institute of Technology
Mumbai, India
krithi@cse.iitb.ac.in

ABSTRACT

Continuous queries are used to monitor changes to time varying data and to provide results useful for online decision making. Typically a user desires to obtain the value of some aggregation function over distributed data items, for example, to know (a) the average of temperatures sensed by a set of sensors (b) the value of index of mid-cap stocks. In these queries a client specifies a coherency requirement as part of the query. In this paper we present a low-cost, scalable technique to answer continuous aggregation queries using a content distribution network of dynamic data items. In such a network of data aggregators, each data aggregator serves a set of data items at specific coherencies. Just as various fragments of a dynamic web-page are served by one or more nodes of a content distribution network, our technique involves decomposing a client query into sub-queries and executing sub-queries on judiciously chosen data aggregators with their individual sub-query incoherency bounds. We provide a technique of getting the optimal query plan (i.e., set of sub-queries and their chosen data aggregators) which satisfies client query's coherency requirement with least cost, measured in terms of the number of refresh messages sent from aggregators to the client. For estimating query execution cost, we build a continuous query cost model which can be used to estimate the number of messages required to satisfy the client specified incoherency bound. Performance results using real-world traces show that our cost based query planning leads to queries being executed using less than one third the number of messages required by existing schemes.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web Based Services

General Terms

Algorithms, Management, Measurement, Performance, Design.

Keywords

Content distribution networks, Dynamic data, continuous aggregation queries, data coherency, query dissemination cost.

1. INTRODUCTION

Many data intensive applications delivered over the Web suffer from performance and scalability issues. Content distribution networks (CDNs) solved the problem for static content using caches at the edge nodes of the networks. CDNs continue to evolve to serve more and more dynamic applications [1, 2]. A dynamically generated web page is usually assembled using a number of static or dynamically generated fragments. The static fragments are served from the local caches whereas dynamic fragments are created either by using the cached data or by fetching the data items from the origin data sources. One important question for satisfying client requests through a network of nodes is how to select the best node(s) to satisfy the request. For static pages content requested, proximity to the client and load on the nodes are the parameters generally used to select the appropriate node [3, 4]. In dynamic CDNs, while selecting the node(s) to satisfy the client request, the central site (top-level CDN node) has to ensure that page/data served meets client's coherency requirements also. Techniques to efficiently serve fast changing data items with guaranteed incoherency bounds have been proposed in the literature [5, 6]. Such dynamic data dissemination networks can be used to disseminate data such as stock quotes, temperature data from sensors, traffic information, and network monitoring data. In this paper we propose a method to efficiently answer aggregation queries involving such data items.

In data dissemination schemes proposed in literature [5, 6], a hierarchical network of data aggregators is employed such that each data aggregator serves the data item at some guaranteed incoherency bound. *Incoherency of a data item at a given node is defined as the difference in value of the data item at the data source and the value at that node.* Although CDNs use page-purge [8] based coherency management, we assume that in dynamic data dissemination networks, these messages carry the new data values thereby an invalidation message becomes a refresh message. For maintaining a certain incoherency bound, a data aggregator gets data updates from the data source or some higher level data aggregator so that the data incoherency is not more than the data incoherency bound. In a hierarchical data dissemination network a higher level aggregator guarantees a tighter incoherency bound compared to a lower level aggregator. Thus data refreshes are pushed from the data sources to the clients through the network of aggregators. Dissemination networks for various data items (possibly from different data sources) can be overlaid over a single network of data aggregators as shown in Figure 1. Thus, from a data dissemination capability point of view, each data aggregator (DA) is characterized by a set of (s_i, c_i) pairs, where s_i is the data item which the DA can disseminate at an incoherency bound c_i.

Example 1: In a network of data aggregators managing data items S_1-S_4, various aggregators can be characterized as-

D1: $\{(S_1, 0.5), (S_3, 0.2)\}$

D2: $\{(S_1, 1.0), (S_2, 0.1), (S_4, 0.2)\}$

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

Aggregator *D1* can serve values of S_1 with an incoherency bound greater than or equal to 0.5 whereas *D2* can disseminate the same data item at a looser incoherency bound of 1.0 or more. Usually, client is interested in an aggregation of these dynamic data items at a certain incoherency bound. These continuous queries are used to monitor changes in dynamic data and provide results useful for online decision making. For generating the result of a query, data from multiple sources is required. As a result, the query has to be evaluated either at data aggregators [9] or at the client. In this paper we assume existence of data dissemination network of multiple data items to answer a class of queries termed, continuous incoherency bounded weighted aggregation queries, which are formally defined next.

1.1 Continuous Incoherency-Bounded Weighted Aggregation Queries

A continuous weighted aggregation query can be formally written as:

$$V_s^q(t) = \sum_{i=1}^{i=n^q} s_i(t) \times w_i^q \quad (1)$$

V_s^q is the value of a client query q involving n^q data items with the weight of the i^{th} data item being $w^q{}_i$, $1 \le i \le n^q$. $s_i(t)$ is the value of the i^{th} data item at the data source at time t. Such a query encompasses SQL aggregation operators SUM and AVG besides general weighted aggregation queries such as portfolio queries, involving aggregation of stock prices, weighted with number of shares of stocks in the portfolio. Due to space limitations we are not presenting execution schemes for other aggregation queries such as MIN/MAX. Interested readers are referred to [10] for the extended version of this paper.

Let the value of i^{th} data item, in Equation (1), known to the client/DA be $d_i(t)$. Then the data incoherency is given by $|s_i(t)-d_i(t)|$. For a data item which needs to be disseminated at an incoherency bound C the data refresh is sent to the client or lower level DA, if the $|s_i(t)-d_i(t)|$ is more than C. If user specified incoherency bound for the query q is C^q, then the dissemination network has to ensure that:

$$\left| \sum_{i=1}^{n^q} (s_i(t) - d_i(t)) \times w_i^q \right| \le C^q \quad (2)$$

Whenever data values at sources change such that query incoherency bound is violated, the updated value(s) is disseminated to the client. If the network of aggregators can ensure that the i^{th} data item has incoherency bound C_i then the following condition ensures that the query incoherency bound C^q is satisfied:

$$\sum_{i=1}^{n^q} C_i \times w_i^q \le C^q \quad (3)$$

A client specified query incoherency bound needs to be translated into incoherency bounds for individual data items or sub-queries such that Equation (3) is satisfied. It should be noted that Equation (3) is sufficient condition for satisfying the query incoherency bound but not the necessary. This way of translating the query incoherency bound into the sub-query incoherency bounds is required if data is transferred between various nodes using only *push* based mechanism. Such a translation is not required in either a *pull* based mechanism as shown in our earlier paper [9] or combinations of *push* and *pull*. In this paper we consider only *push* based data dissemination among servers, DAs and clients. Next we present the summary of our approach towards executing the continuous multi-data weighted additive aggregation query with the objective of minimization of number of refreshes from data aggregators to the client. Our technique can be used for various popular applications where different clients require aggregation of multiple data items at their individual incoherency bounds. Monitoring stock portfolios is one such popular application which we use for performance measurements.

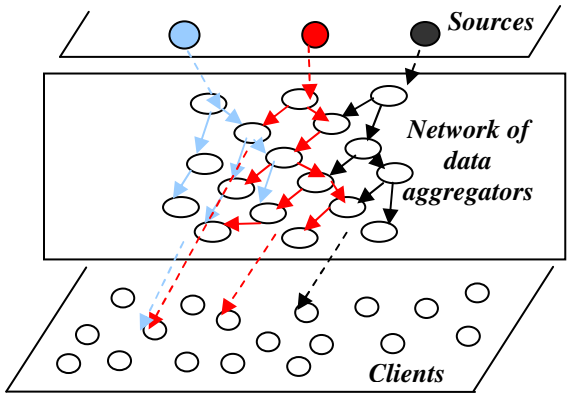

Figure 1: Data dissemination network for multiple data items

1.2 Summary of Approach and Contributions

Consider a client query $Q1$=50 S_1 + 200 S_2 + 150 S_3 with a required incoherency bound of 80 (in a stock portfolio S_1, S_2, S_3 can be different stocks and incoherency bound can be $80).We want to execute this query over data aggregators given in *Example1* minimizing number of refreshes. There are various options for the client to get the data items:

- The client may get the data items S_1, S_2 and S_3 separately. The query incoherency bounds can be divided among data items in various ways while satisfying Equation 3. In this paper, we show that getting data items independently is a costly option. This strategy ignores facts that the client is interested only in the aggregated value of the data items and various aggregators can disseminate more than one data item.
- If a single DA can disseminate all three data items required to answer the client query, the DA can construct a composite data item corresponding to the client query (S_q=50 S_1 + 200 S_2 + 150 S_3) and disseminate the result to the client so that the query incoherency bound is not violated. It is obvious that if we get the query result from a single DA, the number of refreshes will be minimum (as in this case data item updates may cancel out each other, thereby keeping the query result within the incoherency bound). As different data aggregators disseminate different subsets of data items, no data aggregator may have all the data items required to execute the client query which is indeed the case in *Example1*. Further, even if an aggregator can disseminate all the data items, it may not be able to satisfy the query coherency requirements. In such cases the query has to be executed with data from multiple aggregators.
- Another option is to divide the query into a number of sub-queries and get their values from individual DAs. In that case, the client query result is obtained by combining the results of

more than one sub-query. For the DAs given in *Example1*, the query *Q1* can be divided in two alternative ways:

plan1: *D1* {50 S_1 + 150 S_3}; *D2* {S_2}

plan2: *D1* {S_3}; *D2* {50 S_1, + 200 S_2}

i.e., in *plan1* result of sub-query 50 S_1 + 150 S_3 is disseminated by *D1* and that of S_2 (or 200 S_2) by *D2*. Combining them at the client gives the query result.

- Selecting the optimal plan among various options is not-trivial. As a thumb-rule, we should be selecting the plan with lesser number of sub-queries. But that is not guaranteed to be the plan with the least number of messages. Further, we should select the sub-queries such that updates to various data items appearing in a sub-query have more chances of canceling each other as that will reduce the need for refresh to the client (Equation 2). In the above example, if updates to S_1 and S_3 are such that when S_1 increases, S_3 decreases, and vice-versa, then selecting *plan1* may be beneficial. We give an algorithm to select the query plan based on these observations.

- While solving the above problem of selecting the optimal plan we ensure that each data item for a client query is disseminated by one and only one data aggregator. Although a query can be divided in such a way that a single data item is served by multiple DAs (e.g., 50 S_1 + 200 S_2 + 150 S_3 is divided into two sub-queries 50 S_1 + 130 S_2 and 70 S_2 + 150 S_3); but in doing so the same data item needs to be processed at multiple aggregators, increasing the unnecessary processing load. By dividing the client query into disjoint sub-queries we ensure that a data item update is processed only once for each query (For example, in case of paid data subscriptions it is not prudent to get the same data item from the multiple sources).

- The query incoherency bound needs to be divided among sub-query incoherency bounds such that, besides satisfying the client coherency requirements, the chosen DA (where the sub-query is to be executed) is capable of satisfying the allocated sub-query incoherency bound. For example, in *plan1* allocated incoherency bound to the sub-query $50S_1 + 150S_3$ should be greater than 55 (=50*0.5+150*0.2) as that is the tightest incoherency bound which the aggregator *D1* can satisfy. We prove that the number of refreshes depends on the division of the query incoherency bounds among sub-query incoherency bounds. Similar result was proved for data incoherency bounds in [11].

Thus, what we need is a method of (a) optimally dividing client query into sub-queries and (b) assigning incoherency bounds to them; such that (c) selected sub-queries can be executed at chosen DAs and (d) total query execution cost, in terms of *number of refreshes*, is minimized. **We prove that the problem of choosing sub-queries while minimizing query execution cost is an *NP*-hard problem**. **We give efficient approximation algorithms to choose the set of sub-queries and their corresponding incoherency bounds for a given client query.** In contrast, all related work in this area [11, 12] propose getting individual data items from the aggregators which, as we show in this paper, leads to large number of refreshes. For solving the above problem of optimally dividing the client query into sub-queries, we first need a method to estimate query execution cost for various alternative options. **A method for estimating the query execution cost is another important contribution of this paper.** As we divide the client query into sub-queries such that each sub-query gets executed at different aggregator nodes, the query execution cost (i.e., *number of refreshes*) is the sum of the execution costs of its constituent sub-queries. We model the sub-query execution cost as a function of following parameters:

(a) Dissemination costs of the individual data items involved. The data dissemination cost is dependent on data dynamics and incoherency bound associated with the data. We model the data dynamics using a *data synopsis model*, and the effect of the incoherency bound using an *incoherency bound model*. These two models are combined to get the estimate of the data dissemination cost.

(b) A correlation measure of data dynamics, quantifying the chance that the updates of two data items will cancel each other out such that a sub-query of their sum will incur less refreshes than disseminating the individual data changes. We use *cosine similarity* between data items for this purpose. This parameter is widely used in information retrieval domain [20].

Through extensive simulations we show that:

- Our method of dividing query into sub-queries and executing them at individual DAs requires less than one third of the number of refreshes required in the existing schemes.

- For efficient execution, more dynamic data item should be part of sub-query involving larger number of data items.

Our method of executing queries over dynamic data dissemination network is practical since it can be implemented using a mechanism similar to URL-rewriting [4] in CDNs. Just like in a CDN, the client sends its query to the central site. For getting appropriate aggregators (edge nodes) to answer the client query (web page), the central site has to first determine which data aggregators have the data items required for the client query. If the client query can not be answered by a single data aggregator, the query is divided into sub-queries (fragments) and each sub-query is assigned to a single data aggregator. In case of a CDN, web page's division into fragments is a page design issue, whereas, for continuous aggregation queries, this issue has to be handled on per-query basis by considering *data dissemination capabilities* of data aggregators as represented in *Example* 1.

1.3 Outline of the Paper

We give a formal mathematical definition of the query plan selection problem in Section 2. Query cost model for a multi-data incoherency bounded aggregation query is developed in Section 3. The query cost model uses the data dissemination model presented in Section 3.1 and *cosine similarity* measure which is explained in Section 3.2. In Section 4, we first prove that the optimization problem presented in Section 2 is *NP-hard* then we give approximate algorithms for the problem. In Section 5, performance evaluation done using real-world traces is presented to show that our sub-query based query evaluation scheme executes the client query at less than one third cost compared to other known schemes. Related work is presented in Section 6 and the paper concludes in Section 7.

2. QUERY PLAN SELECTION PROBLEM

In this section, we give a formal definition of the optimization problem described in the previous section. We are given a set D of data aggregators, set S of data items and one-to-many mapping f: $D \rightarrow (S, C)$ where $C \subset \Re$ is a sub-set of real number representing incoherency bounds for various data items (in the set S) at

aggregators in D. Each incoming client query q over the data items set $S^q \subseteq S$ has corresponding weights given as a set W^q. Thus the query can be represented as set of tuples of ⟨*data_item*, *weight*⟩, i.e., $q = \{\langle s^q, w^q \rangle\}$ with the query incoherency bound C^q. We need to perform the following two tasks such that the number of refreshes to the client is minimum:

Task1: Divide the client query $q = \{\langle s^q, w^q \rangle\}$ into sub-queries $q_k = \{\langle s_k^q, w_k^q \rangle\}$ so that $\cup q_k = q$ i.e., although different sub-queries may be executed at different aggregators, combining their results gives the value of the client query.

Task2: Allocate each sub-query q_k, with its incoherency bound C_k, to data aggregators.

While fulfilling the following conditions:

Condition1: Query incoherency bound is satisfied, i.e., $\sum_k C_k \leq C^q$.

The sub-query q_k should be assigned to a data aggregator $d_i \in D$ iff:

Condition2: The chosen aggregator should have all the data items appearing in the sub-query i.e. $\prod_S(q_k) \subseteq \prod_S(f(d_i))$. Here Π indicates project operator in relational algebra.

Condition3: Data incoherency bounds at the selected data aggregator $c_j = \prod_C(\sigma_{s=s_k^q(j)}(f(d_i))$ should be such that $C_k \geq X_k$ where $s_k^q(j)$ is the j^{th} data item appearing in the sub-query q_k and X_k is the tightest incoherency bound the aggregator can ensure for the given sub-query. X_k can be calculated as: $X_k = \Sigma c_j w_j^q$. Here σ indicates select operator in relational algebra.

3. QUERY COST MODEL

Before developing the query cost model we first summarize the model to estimate the number of refreshes required to disseminate a data item at certain incoherency bound. For simulation experiments we use data items from sensor network and stock data domains as explained in our previous work [9]. Stock traces of 45 stocks were obtained by periodically polling *http://finance.yahoo.com*. Sensor network data used were temperature and wind sensor data from Georges Bank Cruises Albatross Shipboard [13]. Due to paucity of space we present results using stock data only but similar results were obtained for sensor data as well [14]. For detailed analysis and simulation results, readers can refer to the extended version of the paper [10].

3.1 Data Dissemination Cost

Cost of disseminating a data item at a certain given incoherency bound C can be estimated by combining two models:

- *Incoherency bound model* is used for estimating dependency of data dissemination cost over the desired incoherency bound. As per this model, we have shown in [10] that the number of data refreshes is inversely proportional to the square of the incoherency bound ($1/C^2$). Similar result was earlier reported in [5] where the data dynamics was modeled as a random-walk process.

$$\textit{Data dissemination cost} \propto 1/C^2 \tag{4}$$

- *Data Synopsis Model* is used for estimating the effect of data dynamics on number of data refreshes. We define a data dynamics measure called, *sumdiff*, to obtain a synopsis of the data for predicting the dissemination cost. The number of update messages for a data item is likely to be higher if the data item changes more in a given time window. Thus we hypothesize that cost of data dissemination for a data item will be proportional to *sumdiff*, defined as:

$$R_s = \sum_i |\, s_i - s_{i-1} \,| \tag{5}$$

where s_i and s_{i-1} are the sampled values of the data item at i^{th} and $(i\text{-}1)^{th}$ time instances (consecutive ticks). In [10] we corroborate the above hypothesis using simulation over a large number of data items. Pearson product moment correlation coefficient (PPMCC) [19] values, used for quantifying linearity between data *sumdiff* and number of refreshes required to maintain a fixed incoherency bound, were found to be between 0.90 and 0.96 for various values of incoherency bounds. *Sumdiff* value for a data item can be calculated at the data source by taking running average of difference between data values at the consecutive ticks. A data aggregator can also estimate the *sumdiff* value by interpolating the disseminated values.

Thus, the estimated dissemination cost for data item S, disseminated with an incoherency bound C, is proportional to R_s/C^2. Next we use this result for developing the query cost model.

3.2 Query Dissemination Cost

Consider a case where a query consists of two data items P and Q with weights w_p and w_q respectively; and we want to estimate its dissemination cost. If data items are disseminated separately, the query *sumdiff* will be:

$$R_{data} = w_p R_p + w_q R_q = w_p \sum |\, p_i - p_{i-1} \,| + w_q \sum |\, q_i - q_{i-1} \,| \tag{6}$$

Instead, if the aggregator uses the information that client is interested in a query over P and Q (rather than their individual values), it makes a composite data item $w_p p + w_q q$ and disseminates that data item then the query *sumdiff* will be:

$$R_{query} = \sum |\, w_p (p_i - p_{i-1}) + w_q (q_i - q_{i-1}) \,| \tag{7}$$

R_{query} is clearly less than or equal compared to R_{data}. Thus we need to estimate the *sumdiff* of an aggregation query (i.e., R_{query}) given the *sumdiff* values of individual data items (i.e., R_p and R_q). Only data aggregators are in position to calculate R_{query} as different data items may be from different sources. We develop the query dissemination model in two stages.

3.2.1 Quantifying correlation between dynamics of data

From Equations (6) and (7) we can see that if two data items are correlated such that if value of one data item increases, that of the other data item also increases, then R_{query} will be closer to R_{data} whereas if the data items are inversely correlated then R_{query} will be less compared to R_{data}. Thus, intuitively, we can represent the relationship between R_{query} and *sumdiff* values of the individual data items using a correlation measure associated with the pair of data items. Specifically, if ρ is the correlation measure then R_{query} can be written as:

$$R_{query}^2 \propto (w_p^2 R_p^2 + w_q^2 R_q^2 + 2\rho w_p R_p w_q R_q) \tag{8}$$

The correlation measure is defined such that $-1 \leq \rho \leq +1$, so, R_{query} will always be less than $|w_pR_p+w_qR_q|$ (as explained earlier) and always be more than $|w_pR_p-w_qR_q|$. The correlation measure ρ can be interpreted as *cosine similarity* [20] between two streams represented by data items P and Q. Cosine similarity is a widely used measure in information retrieval domain where documents are represented using a vector-space model and document similarity is measured using cosine of angle between two document representations. For data streams P and Q, ρ can be calculated as:

$$\rho = \frac{\sum (p_i - p_{i-1})(q_i - q_{i-1})}{\sqrt{\sum (p_i - p_{i-1})^2}\sqrt{\sum (q_i - q_{i-1})^2}} \quad (9)$$

3.2.2 Query normalization

Suppose we want to compare the cost of two queries: a SUM query involving two data items and an AVG query involving the same data items. Let the query incoherency bound for the SUM and the AVG queries be $C_1=2C$ and $C_2=C$, respectively. From Equation (8), *sumdiff* of the SUM query will be double that of the AVG query (as the weight of each data item in the SUM query is double of that in the AVG query). Hence, query evaluation cost (as per R_i/C_i^2) of the SUM query will be half that of the AVG query (as SUM query incoherency bound is double). But, intuitively, disseminating the SUM of two data items, at double the incoherency bound should require the same number of messages as their AVG. Thus, there is a need to *normalize* query costs. From a query execution cost point of view, a query with weights w_i and incoherency bound C is same as query with weights $\alpha\, w_i$ and incoherency bound αC. So, while normalizing we need to ensure that both, query weights and incoherency bounds, are multiplied by the same factor. Normalized query *sumdiff* is given by:

$$R_{query}^2 = \frac{(w_p^2R_p^2 + w_q^2R_q^2 + 2\rho w_pR_pw_qR_q)}{(w_p^2 + w_q^2 + 2\rho w_pw_q)} \quad (10)$$

i.e., the value of the normalizing factor $s$should be $1/\sqrt{w_p^2 + w_p^2 + 2\rho w_pw_q}$. The value of the incoherency bound has to be adjusted by the same factor. Normalization ensures that queries with arbitrary values of weights can be compared for execution cost estimates. From Equations (9 and 10) the value of query *sumdiff* can be estimated at a data aggregator node if it has all the required data items disseminated to it. An aggregator can use interpolated values of data items to estimate ρ as it is not (always) likely to have all the data updates. In the extended version of the paper [10] we present an efficient method (using [21]) to calculate ρ which can also be used when the corresponding data items are not being disseminated by the same data aggregator. Equation (10) can be extended to get query *sumdiff* for any general weighted aggregation query given by Equation (1) as:

$$R_Q^2 = \frac{\sum_{i=1}^{n} w_i^2R_i^2 + 2\sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} \rho_{ij}w_iw_jR_iR_j}{\sum_{i=1}^{n} w_i^2 + 2\sum_{i=1}^{n}\sum_{j=1, j\neq i}^{n} \rho_{ij}w_iw_j} \quad (11)$$

3.2.3 Validating the query cost model

To validate the query cost model we performed simulations by constructing more than 50 weighted aggregation queries using the stock data with each query consisting of 3-7 data items with data weights uniformly distributed between 1 and 10. For each query the number of refreshes was counted for various normalized incoherency bounds between 0.01 and 0.5. Figure 2 shows that the number of messages is proportional to the normalized query *sumdiff* as calculated using Equation (11) if their normalized incoherency bounds are same. In this case PPMCC value is found to be 95%. Similarly, Figure 3 shows the dependence of the number of refreshes on $1/C^2$ to prove that the relationship that holds between them for single data item also holds for a query with multiple data items. The query cost model can be used in various applications of query assignment, load balancing, optimal order of processing, etc. In the next section, we use this query cost model for our query plan problem to optimally divide a client query into sub-queries and execute it over a network of data aggregators so that the number of refreshes can be minimized

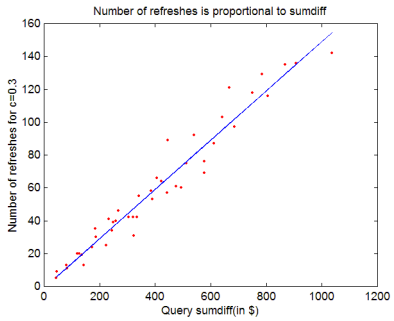

Figure 2: Variation of query cost with query *sumdiff* (Normalized C=0.3)

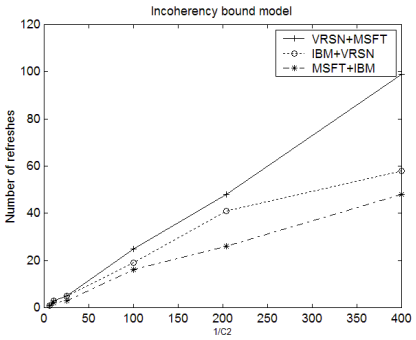

Figure 3: Number of refreshes for varying query incoherency bounds

4. EXCEUCTING QUERIES USING SUB-QUERIES

For executing an incoherency bounded continuous query, a query plan is required which includes the set of sub-queries, their individual incoherency bounds and data aggregators which can execute these sub-queries. We need to find the optimal query execution plan which satisfies client coherency requirement with the least number of refreshes. As explained in Section 1, what we need is a mechanism to:

Task 1: Divide the aggregation query into sub-queries; and

Task 2: Allocate the query incoherency bound among them.

while satisfying the following conditions identified in Section 2:

condition 1. Query incoherency bound is satisfied.

condition 2. The chosen DA should be able to provide all the data items appearing in the sub-query assigned to it.

condition 3. Data incoherency bounds at the chosen DA should be such that the sub-query incoherency bound can be satisfied at the chosen DA.

Objective : Number of refreshes should be minimized.

Let the client query be divided into N sub-queries $\{q_k: 1 \le k \le N\}$; with R_k being *sumdiff* of k^{th} sub-query and C_k being incoherency bound assigned to it. As given is Section 3, the dissemination cost of a sub-query is estimated to be proportional to R_k/C_k^2. Thus query cost estimate is given by:

$$Z = \sum_{k=1}^{N} (R_k / C_k^2) \quad (12)$$

While allocating sub-query incoherency bounds we need to ensure that the query coherency requirement C is satisfied (*condition1)*; i.e.,

$$\sum_{k=1}^{N} C_k \le C \quad (13)$$

For satisfying *condition2*, sub-queries should be such that all its data items can be disseminated by the chosen DA. Let X_k be the tightest incoherency bound (defined in Section 2) the chosen DA can satisfy for q_k. For the *condition3*, we have to ensure that $C_k \ge X_k$ for each sub-query q_k and its assigned data aggregator. Z needs to be minimized for minimizing the number of refreshes as per the *objective*.

Before attempting the *hard* problem of optimizing Z, let us first consider a simpler problem where values of C_k are given. In this simpler problem we divide the client query into sub-queries to minimize the estimated execution cost (Z) without considering the optimal division of the query incoherency bound into sub-query incoherency bounds. Besides working as a step towards a solution for the whole problem this case can also be used where allocation of incoherency bounds to sub-queries is done independent of the data dynamics. For example, it may be pre-decided that incoherency bounds for all data items will be the same. Thus, for a given query and its incoherency bounds, the sub-query incoherency bounds can be obtained. Next we prove that this simpler version of the problem is NP-hard.

4.1 Finding Optimal Query Plan is NP-hard

For proving that the problem is NP-hard, **we use reduction from 3-dimensional matching (3DM) problem**. For a given client query and DA, let us first define maximal sub-query as the largest part of the query which can be disseminated by the DA (i.e., the maximal sub-query has all the query data items which the DA can disseminate at the required incoherency bound). For example, for a client query $20S_1 + 25S_2 + 35S_3$ with incoherency bound 80; let the pre-decided incoherency bound for each data item be 1. For the data aggregators *D1* and *D2* given in *Example 1*, the maximal sub-query for *D1* will be $q_1 = 20S_1 + 35S_3$, whereas for *D2* it will be $q_2 = 20S_1 + 25S_2$.

3DM Problem: Given three sets X, Y and Z, each with N elements, and a set $M \subseteq X \times Y \times Z$, is there a subset $M1 \subseteq M$ such that $|M1| = N$ and no two elements of $M1$ agree in any coordinate?

We use a slightly different (decision) version of the optimization problem to reduce the 3DM problem. To solve the 3DM problem we reduce it to a SUM query of $3N$ items:

- The SUM query: $\sum_{i=1}^{N} (x_i + y_i + z_i)$ for $x_i \in X$, $y_i \in Y$, $z_i \in Z$.
- We assume that all the data items have the same *sumdiff* values of 1; cosine similarity between all the data items is 0; and all data items are allocated an incoherency bound of 1.
- For each element $(x_i, y_j, z_k) \in M$, we assume the existence of a data aggregator disseminating these three data items only.
- In the decision version of optimal plan problem we ask whether there exists a query plan with query cost estimate value $N/3$.

If a query plan with cost estimate value $N/3$ exists; it implies that the query plan has N queries with 3 items each (that will lead to query cost value of 1/3 per sub-query as per Equation (11) whereas any other combination of sub-queries will lead to more cost). Three data items from each chosen data aggregators form a triplet for the set *M1* which solves 3DM. **Because of space constraints we are not giving the complete proof of NP-hardness of the original problem**. In general, there is no known approximate algorithm for such a problem. It should be noted that performing *Task1* for achieving the *objective* is NP-hard, so we give two greedy heuristics in next two sub-sections; whereas *Task2* can be performed optimally with *conditions1-3* while achieving the *objective*. In our approach, we first try to perform *Task1*, while satisfying as many *conditions* as possible, and then optimally perform *Task2* while satisfying all the conditions.

4.2 Minimum Cost Heuristic

Figure 4 shows the outline of greedy heuristics where different criteria (ψ) can be used to select sub-queries. In this section we describe the case where the estimate of query execution cost is minimized in each step of the algorithm (*min-cost*) whereas in the next section we present the case where gain due to executing a query using sub-queries is maximized (*max-gain*).

4.2.1 Query Plan with Pre-decided Incoherency Bound Allocation

For the given client query (q) and mapping between data aggregators and the corresponding {*data-item*, *data incoherency bound*} pairs (f: $D \rightarrow (S, C)$) maximal sub-queries can be obtained for each data aggregator. Let A be the set of such maximal sub-queries. In this set, each query $a \in A$ can be disseminated by a designated data aggregator at the assigned incoherency bound. For each sub-query $a \in A$, its *Sumdiff* R_a is calculated using Equation 11. Using the set A and sub-query *sumdiffs*, we use the algorithm outlined in Figure 4 to get the set of sub-queries minimizing the query cost. In this Figure each sub-query $a \in A$ is represented by the set of data items covered by it. As we need to minimize the query cost, a sub-query with *minimum cost per data item* is chosen in each iteration of the algorithm i.e., criteria $\psi \equiv$ minimize $(R_a/C_a^2|a|)$. All data items covered by the selected sub-query are removed from all the remaining sub-queries in A before performing the next iteration.

```
Result ← Ø
while A ≠ Ø
   choose a sub-query a∈A with criteria ψ
   Result ← Result ∪a
   A←A-{a}
   for each data element e ∈a
      for each b∈A
         b←b-{e}
         if b = Ø
            A←A-{b}
         else
            calculate sumdiff for modified b
return Result
```

Figure 4: Greedy algorithm for query plan selection

4.2.2 Optimizing query execution cost

Now we consider the overall problem to select the optimal set of sub-queries while simultaneously dividing the query incoherency bound among them. In this case we get the set of maximal queries (*A*) without considering the minimum incoherency bounds that the data aggregators can satisfy (i.e., *condition3*). In this algorithm we first get the optimal set of sub-quires without considering the *condition3* and then allocate incoherency bound among them using *condition1* (Equation (13)) and *condition3*. Lagrange multiplier scheme can be used to solve for incoherency bounds (from Equations 12 & 13) so that *Z* is minimized:

$$C_k = R_k^{1/3} C / (\sum_{k=1}^{N} R_k^{1/3}) \qquad (14)$$

i.e., without the constraints of *condition3*, sub-query incoherency bounds should be allocated in proportion to $R_k^{1/3}$. Using Equations (12) and (14) we get:

$$Z^{1/3} = \frac{1}{C^{2/3}} \sum_{k=1}^{N} R_k^{1/3} \qquad (15)$$

From Equation (15), it is clear that for minimizing the query execution cost we should select the set of sub-queries so that $\sum R_k^{1/3}$ is minimized. We can do that by using criteria $\psi \equiv$ *minimize* ($R_a^{1/3}/|a|$) in the algorithm described in Figure 4. Once we get the optimal set of sub-queries we can use the Equation (13) and *condition3* ($C_k \geq X_k$) to optimally allocate the query incoherency bound among them. This allocation problem can be solved by various convex optimization techniques available in the literature such as gradient descent method, barrier method etc. We used gradient descent method (*fmincon* function in MATLAB) to solve this non-linear optimization problem to get the values of individual sub-query incoherency bounds. But this method of first selecting sub-queries and then allocating the incoherency bounds has a problem which is described next.

4.2.3 Satisfiability of Condition 3

In the solution described in the previous section, we select the set of sub-queries (and corresponding DAs) and then allocate the query incoherency bound among them using convex optimization techniques. But the problem of incoherency bound allocation among chosen DAs may not have any feasible solution. There may be situations where, although the data dissemination network is able to satisfy the query coherency requirements but once the set of sub-queries (and corresponding DAs) is selected the incoherency bound allocation is not possible.

Example 2: Consider a client query $50S_1 + 200S_2 + 150S_3$ with the incoherency bound of 80 and data dissemination network consisting of two aggregators *D1* and *D2* as given in *Example 1*. There are (at-least) two possible query plans to answer the above query:

Plan1: *D1* ($50S_1 + 150\ S_3$); *D2* (S_2)

Plan2: *D1* (S_3); *D2* ($50S_1 + 200\ S_2$)

In Section 4.2.2 we are selecting sub-queries having minimum $\sum R_k^{1/3}$, thus based on data dynamics it is possible that we select *plan2* as the optimal plan. But from the data incoherency bounds that aggregators *D1* and *D2* can ensure, we see that it is not possible for *plan2* to satisfy the client specified incoherency bound as minimum incoherency bound that can be satisfied by the selected aggregators (*X*=50*1 +200*0.1 +150*0.2 =100) is greater than the query incoherency bound (=80). Thus although there exists a plan (*plan1*) which can satisfy the client query incoherency bound, while minimizing the query execution cost the above method cannot ensure that such a plan will be selected.

What we need is a compromise between the query satisfiability and performance. In Section 4.2.2 we are selecting the sub-queries without considering the data incoherency bounds for the selected data aggregators. We correct that by selecting sub-queries using $\sum (R_a^{1/3} + \frac{\alpha X_a}{C R_a^{1/3}})$ as *substitute objective function* instead of $\sum R_a^{1/3}$. The second term ensures that while selecting the optimal plan we *prefer* the data aggregators having tighter data incoherency bounds (lower values of X_a) thus higher chances of satisfying the query. The tuning parameter (α) can used to balance the objectives of minimizing query execution cost through sub-queries selection and meeting the query coherency requirements. We use $X_a / C R_a^{1/3}$ in the second term as, according to Equation (14), optimal incoherency bound allocation is likely to be done proportional to $C R_a^{1/3}$. In Section 5.2, we measure effects of the tuning parameter α on the query satisfiability.

4.3 Maximum Gain Heuristic

In this section we present an algorithm which, instead of minimizing the estimated query execution cost, maximizes the estimated gains of executing client query using sub-queries. In this algorithm, for each sub-query, we calculate the *relative gain* of executing it by finding the *sumdiff* difference between cases when each data item is obtained separately and when all the data items are aggregated as a single sub-query. Thus, the relative gain for a sub-query $w_p p + w_q q$ can be written as:

$$G_{query} = \frac{(w_p R_p + w_q R_q)}{\sqrt{(w_p^2 R_p^2 + w_q^2 R_q^2 + 2\rho w_p w_q R_p R_q)}} - 1 \qquad (16)$$

This algorithm can be implemented by using criteria $\psi \equiv$ *maximize* ($G_{query}/|a|$) to get the set of sub-queries and corresponding DAs. Then we use the convex optimization method outlined in Section 4.2 to allocate incoherency bounds among sub-queries. To tackle the query satisfiability issue the query gain Equation (16) is modified to:

$$G'_{query} = G_{query} - \frac{\alpha(w_p X_p + w_q X_q)}{CR_{query}^{1/3}} \quad (17)$$

where X_p is minimum incoherency bound that can be satisfied for the data item P; C is query incoherency bound and R_{query} is the query *sumdiff* ($=\sqrt{(w_p^2 R_p^2 + w_q^2 R_q^2 + 2\rho w_p w_q R_p R_q)}$). Reasons for selecting the particular *substitute objective function* are same as ones outlined in Section 4.2.3. In the next Section, through performance results, we show that this algorithm performs better than the *min-cost* heuristic.

5. PERFORMANCE EVALUATION

For performance evaluation we simulated the data dissemination networks of 25 stock data items over 25 aggregator nodes such that each aggregator can disseminate combinations of up to 10 data items with data incoherency bounds chosen uniformly between \$0.005 and 0.02. Then we created 500 portfolio queries such that each query has up to 10 randomly (uniformly) selected data items with weights varying between 2 and 10. These queries were executed with incoherency bounds between 0.3 and 1.0 (i.e., 0.03-0.1% of the query value). In the first set of experiments, we kept the data incoherency bounds at the data aggregators very low so that query satisfiability can be ensured.

5.1 Comparison of algorithms

For comparison with our algorithms, presented in the previous section, we consider various other query plan options. Each query can be executed by disseminating individual data items or by getting sub-query values from DAs. Set of sub-queries can be selected using *sumdiff* based approaches or any other random selection. Sub-query (or data) incoherency bound can either be pre-decided or optimally allocated. Various combinations of these dimensions are covered in the following algorithms:

1. No sub-query, equal data incoherency bound (*naïve*): In this algorithm, the client query is executed with each data item being disseminated independent of other data items in the query. Incoherency bound is divided equally among the data items. This algorithm acts as a baseline algorithm.

2. No sub-query, optimal incoherency bound (*optc*): In this algorithm also data items are disseminated separately but incoherency bound is divided among data items so that total number of refreshes can be minimized. This algorithm is similar to the one presented in [11]. Here, the incoherency bound is allocated dynamically using Equation (14).

3. Random sub-query selection (*random*): In this case, sub-queries are generated by randomly selecting one data aggregators and allocating it the maximal sub-query consisting of query data items which the aggregator can disseminate. Then the process is repeated for the remaining data items until the whole query is covered. This algorithm is designed to see how the sub-query selection based on query *sumdiff* (Section 4) works in comparison to random selection of sub-queries.

4. Sub-query selection while minimizing *sumdiff* (*min-cost*): This algorithm is described in Section 4.2.

5. Sub-query selection while maximizing gain (*max-gain*): This algorithm is described in Section 4.3.

Figure 5 shows average number of refreshes required for query incoherency bounds of \$0.3, \$0.5 and \$0.8. The naïve algorithm requires more than three times the number of messages compared to *min-cost* and *max-gain* algorithms. For incoherency bound of \$0.8 each query requires 1024 messages if it is executed just by optimizing incoherency bound (*optc*) compared to 255 when we select the query plan using the *max-gain* algorithm. Further, although the optimization problem is similar to the covering a set of data items (query) using its sub-sets (sub-queries) for which the *greedy min-cost* algorithm is considered to be most efficient [7], we see that *max-gain* algorithm requires 20-25% less messages compared to the *min-cost* approach. Reasons for *max-gain* algorithm performing better than other algorithms are explored in the next set of experiments. Although here we presented results for stock traces (man-made data) similar results were obtained for sensor traces (natural data) as well.

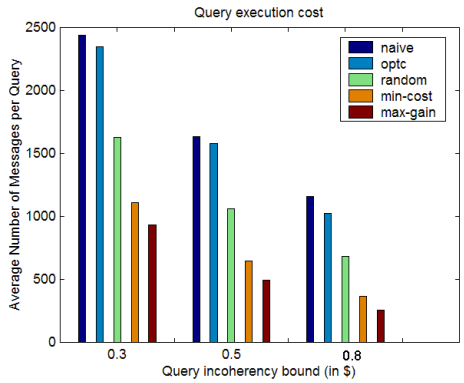

Figure 5: Performance evaluation of algorithms

5.2 Effect of Algorithmic Parameters

These set of experiments were performed to get an insight into various characteristics of our sub-query selection method which lead it to perform better compared to other options. We consider effects of three parameters on sub-query selection and, in turn on query performance: data dynamics, correlation between data dynamics and query satisfiability parameter.

5.2.1 Effect of data dynamics

In this set of experiments, we wanted to see whether there is any definite relationship between data dynamics and sub-query size in which that data item appears. In this experiment with 10 data items, 45 DAs were simulated such that each DA can disseminate a different set of 2 data items. Then 100 queries were created each with 3 data items. In the optimal query plan, each query will be executed with two sub-queries: one consisting of 2 data items and another with single data item (plan with three one item sub-queries will be trivially inefficient). As the query has only 3 data items, only 3 such query plans are possible. We simulated all these options to get the best query plan. Figure 6 shows variation of average sub-query size in which a particular data item appears versus *sumdiff* value of the data item. We can see that if a data item is more dynamic, in the optimal plan, it is more likely to be part of larger sub-query. This is an important observation as it indicates that for efficient query evaluation more dynamic data items should be part of a larger sub-query. This phenomenon can be explained by the fact that by executing a query as a combination of sub-queries will always be more efficient compared to getting the data items independently. By combining

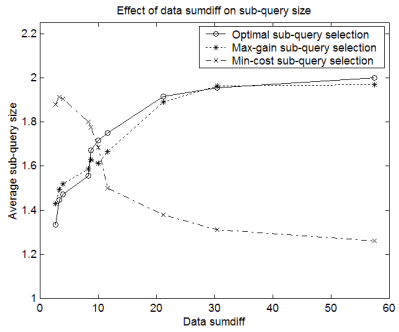

Figure 6: Effect of data *sumdiff* on sub-query size

more dynamic data items we are likely to gain more. For comparison we also show the curve for the sub-query selection based on *max-gain* algorithm. It can be seen that sub-query selection using *max-gain* is approximately same as that selected by the optimal solution. By using *max-gain* algorithm we achieve our objective of disseminating more dynamic data items as part of larger sub-queries. For the *max-gain* algorithm, similar results were obtained for larger query sizes as well. In comparison, in the *min-cost* algorithm most dynamic data item is more likely to be disseminated as single item query. This happens because the *sumdiff* value of a more dynamic data item will be high thus in each step of the *min-cost* algorithm (Figure 4), there is less chance of selecting a sub-query with more dynamic data item. Thus, it is very likely that the highly dynamic data item will be disseminated as a single item sub-query resulting in bad performance of the client query. Still the *min-cost* algorithm performs better compared to *random* algorithm as it tries to execute the query with lesser number of sub-queries.

5.2.2 Effect of correlation between data dynamics

To measure the effects of correlation between data dynamics (cosine similarity) on the query performance, we compared the query performance with the case when all the data items are assumed to be independent (i.e., ρ=0). For performing these experiments we constructed 10 synthetic data traces so that values of ρ for various data item pairs were distributed uniformly between -1 and +1. Then 45 DAs were simulated so that each DA can disseminate 2 data items. 100 queries were generated, each with 4 data items. In this case, each query will get executed with 2 sub-queries of 2 data items each. Combination of sub-queries will be decided based on correlation between data items (*sumdiff* values of all the data items were the same). Table 1 compares the

Table1: Effect of correlation on number of refreshes

Incoherency Bound	Avg. number of msgs when ρ is considered	Avg. number of msgs when ρ is assumed to be 0
0.5	2301	2559
0.8	1092	1215
1.0	754	846

results when cosine similarity is taken into account and when cosine similarity is assumed to be 0 for all data item pairs. It can be seen that by considering cosine similarity number of refreshes reduce by approximately 12%. This result indicates that for sub-query selection data dynamics may be more important factor than the cosine similarity between the data items.

5.2.3 Effect of query satisfiability parameter

To simulate the situation where selected aggregators may not be able to satisfy the query incoherency bounds, we modified the simulation set up used in Section 5.1 to set the minimum data incoherency bounds which DAs can satisfy to be between .015 and 0.04. Value of α was varied between 0-20. The case α=0 corresponds to the algorithm without dealing with the query satisfiability. Figure 7 shows query execution cost and number of unanswerable queries as the value of α is varied. As shown in the figure as the value of α is increased, percentage of the unsatisfied queries decreased for various values of query incoherency bounds.

Due to changed data incoherency bounds of DAs, we found that 20% of queries can not be satisfied even by the data aggregators with tightest data incoherency bounds. Thus, while presenting the results, we remove those queries. At the query incoherency bound of $0.8, 40% are queries can not be satisfied by the optimally

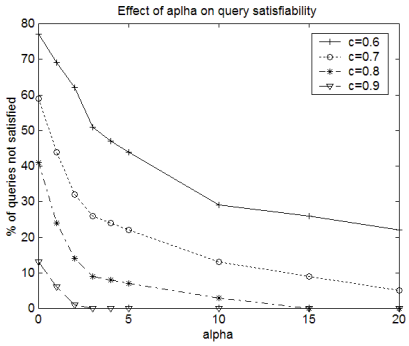

Figure 7: Effect of α on query satisfiability

selected data aggregators but as we increase the value of α to 10, only 3% queries are unanswered. Such a value can be chosen to balance the performance and satisfiability of queries. For example, a dynamic CDN may aim at query satisfiability of 95% for a given distribution of query incoherency bounds. If at any time query satisfiability is below the target, value of α can be increased whereas in case of over achieving the target, the value of α can be decreased to improve the query performance.

5.2.4 Summary of performance results

Following features of the query planning algorithm improve performance:

- Dividing the query into sub-queries and executing them at specifically chosen data aggregators.
- Deciding the query plan using data *sumdiff* based mechanism specifically by maximizing sub-query gains.
- Including more dynamic data as part of a larger sub-query.

We also showed that the *max-gain* algorithm is very close to the optimal algorithm in selecting sub-queries based on data dynamics.

6. RELATED WORK

Various mechanisms for efficiently maintaining incoherency bounded aggregation queries over continuously changing data items are proposed in the literature [11, 12, 16]. Our work distinguishes itself by being sub-query based evaluation to minimize number of refreshes. In [11], authors propose using data filters at the sources; instead we assign incoherency bounds to sub-queries which reduce the number of refreshes for query evaluation, as explained in Section

5. Further, we propose that more dynamic data items should be executed as part of larger sub-query.

In [22], authors present technique of reorganizing a data dissemination network when client requirements change. Instead, we try to answer the client query using the existing network. Reorganizing aggregators is a longer term activity whereas query planning can be done for short as well as long running queries on more dynamic basis.

Pull based data dissemination techniques, where clients or data aggregators pull data items such that query requirements are met, are described in [9,16]. For minimizing the number of pulls, both model the individual data items and predict data values. In comparison, we consider the situation where different sub-queries, involving multiple data items, can be evaluated at different nodes. Further, incoherency bound is applied over the sub-query rather than to individual data items, leading to efficient evaluation of the query. Spatial and temporal correlations between sensor data are used to reduce data refresh instances in [17, 18]. We also consider correlation in terms of cosine similarity between data items, but we use it for dividing client query into sub-queries. Our work can be extended by using temporal and spatial properties of data items for predicting their correlation measures. A method of assigning clients data queries to aggregators in a content distribution network is given in [12]. We do for client queries consisting of multiple data items what [12] does for client requiring individual data items.

7. CONCLUSIONS

This paper presents a cost based approach to minimize the number of refreshes required to execute an incoherency bounded continuous query. For optimal execution we divide the query into sub-queries and evaluate each sub-query at a chosen aggregator. Performance results show that by our method the query can be executed using less than one third the messages required for existing schemes. Further we showed that by executing queries such that more dynamic data items are part of a larger sub-query we can improve performance. Our method of query execution can be implemented using schemes similar to that used in CDNs. Our query cost model can also be used for other purposes such as load balancing various aggregators, optimal query execution plan at an aggregator node, etc. Using the cost model for other applications and developing the cost model for more complex queries is our future work.

Acknowledgement: We would like to thank Venkatesan Chakravarthy and Vinayak Pandit for helpful discussion on various algorithms.

8. REFERENCES

[1] A. Davis, J. Parikh and W. Weihl. Edge Computing: Extending Enterprise Applications to the Edge of the Internet. WWW 2004

[2] D. VanderMeer, A. Datta, K. Dutta, H. Thomas and K. Ramamritham. Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web. ACM Transactions on Database Systems (TODS) Vol. 29, June 2004.

[3] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman and B. Weihl. Globally Distributed Content Delivery, IEEE Internet Computing Sept 2002.

[4] S. Rangarajan, S. Mukerjee and P. Rodriguez. User Specific Request Redirection in a Content Delivery Network, 8th Intl. Workshop on Web Content Caching and Distribution (IWCW), 2003.

[5] S. Shah, K. Ramamritham, and P. Shenoy. Maintaining Coherency of Dynamic Data in Cooperating Repositories. VLDB 2002.

[6] Dynamai: Caching Technology for Dynmaic Content Revealed. www.infoworld.com/articles.

[7] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. SIAM Journal on Computing, vol. 11 (3), 1982.

[8] Zongming Fei. A Novel Approach to Managing Consistency in Content Distribution. WCW 2001

[9] R. Gupta, A. Puri, and K. Ramamritham. Executing Incoherency Bounded Continuous Queries at Web Data Aggregators. WWW 2005.

[10] Optimized Execution of Continuous Queries, APS 2006, www.cse.iitb.ac.in/~grajeev/APS06.PDF

[11] C. Olston, J. Jiang, and J. Widom. Adaptive Filter for Continuous Queries over Distributed Data Streams. SIGMOD 2003.

[12] S. Shah, K. Ramamritham, and C. Ravishankar. Client Assignment in Content Dissemination Networks for Dynamic Data. VLDB 2005.

[13] NEFSC Scientific Computer System http://sole.wh.whoi.edu/~jmanning//cruise/serve1.cgi

[14] Query cost model validation for sensor data. www.cse.iitb.ac.in/~ravivj/BTP06.pdf.

[15] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. SIAM Journal on Computing, vol. 11 (3), 1982.

[16] S. Zhu and C. Ravishankar. Stochastic Consistency and Scalable Pull-Based Caching for Erratic Data Sources. VLDB 2004.

[17] D. Chu, A. Deshpande, J. Hellerstein, W. Hong. Approximate Data Collection in Sensor Networks using Probabilistic Models. ICDE 2006.

[18] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. *VLDB*, 2004.

[19] Pearson Product moment correlation coefficient. http://www.nyx.net/~tmacfarl/STAT_TUT/correlat.ssi /

[20] Lam, W. and Ho, C.Y. Using a Generalized Instance Set for Automatic Text Categorization. SIGIR, 1998.

[21] G. Cormode and M. Garofalakis. Sketching Streams through the Net: Distributed Approximate Query Tracking. VLDB 2005.

[22] S. Agrawal, K. Ramamritham and S. Shah. Construction of a Temporal Coherency Preserving Dynamic Data Dissemination Network. *RTSS* 2004.