Query Planning for Continuous Aggregation Queries over a Network of Data Aggregators

Rajeev Gupta, and Krithi Ramamritham, *Fellow IEEE*

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 value of portfolio for a client; or the AVG of temperatures sensed by a set of sensors. In these queries a client specifies a coherency requirement as part of the query. We present a low-cost, scalable technique to answer continuous aggregation queries using a network of aggregators 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 for getting the optimal set of sub-queries with their incoherency bounds which satisfies client query's coherency requirement with least number of refresh messages sent from aggregators to the client. For estimating the number of refresh messages, we build a 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.

Index Terms—Algorithms, Continuous queries, Distributed query processing, Data dissemination, Coherency, Performance.

1 INTRODUCTION

Applications such as auctions, personal portfolio valuations for financial decisions, sensors based monitoring, route planning based on traffic information, etc., make extensive use of dynamic data. For such applications, data from one or more independent data sources may be aggregated to determine if some action is warranted. Given the increasing number of such applications that make use of highly dynamic data, there is significant interest in systems that can efficiently deliver the relevant updates automatically. As an example, consider a user who wants to track a portfolio of stocks in different (brokerage) accounts. Stock data values from possibly different sources are required to be aggregated to satisfy user's requirement. These aggregation queries are long running queries as data is continuously changing and the user is interested in notifications when certain conditions hold. Thus, responses to these queries are refreshed continuously. In these continuous query applications, users are likely to tolerate some inaccuracy in the results. That is, the exact data values at the corresponding data sources need not be reported as long as the query results satisfy user specified accuracy requirements. For instance, a portfolio tracker may be happy with an accuracy of $10.

Data incoherency: Data accuracy can be specified in terms of *incoherency of a data item, defined as the absolute difference in value of the data item at the data source and the value known to a client of the data. Let* $v_i(t)$ denote the value of the i^{th} data item at the data source at time t; and let the value the data item known to the client be $u_i(t)$. Then the data incoherency at the client is given by $|v_i(t)-u_i(t)|$. For a data item which needs to be refreshed at an incoherency bound C a data refresh message is sent to the client as soon as data incoherency exceeds C, i.e., $|v_i(t)-u_i(t)| > C$.

Network of data aggregators: Data refresh from data sources to clients can be done using *push* or *pull* based mechanisms. In a push based mechanism data sources send update messages to clients on their own whereas in pull based mechanism data sources send messages to the client only when the client makes a request. We assume the push based mechanism for data transfer between data sources and clients. For scalable handling of push based data dissemination, network of data aggregators are proposed in the literature [5, 7, 22]. In such network of data aggregators, data refreshes occur from data sources to the clients through one or more data aggregators.

In this paper we assume that each data aggregator maintains its configured incoherency bounds for various data items. From a data dissemination capability point of view, each data aggregator (DA) is characterized by a set of (d_i, c_i) pairs, where d_i is a data item which the DA can disseminate at an incoherency bound c_i. The configured incoherency bound of a data item at a data aggregator can be maintained using any of following methods: (a) The data source refreshes the data value of the DA whenever DA's incoherency bound is about to get violated. This method has scalability problems. (b) Data aggregator(s) with tighter incoherency bound help the DA to maintain its incoherency bound in a scalable manner as explained in [5,7].

• *Rajeev Gupta is with IBM Research, New Delhi. E-mail: grajeev@in.ibm.com.*
• *Krithi Ramamritham is with Indian Institute of Technology, Mumbai. E-mail: krithi@cse.iitb.ac.in.*

Manuscript received (May 05, 2010).

Example 1: In a network of data aggregators managing data items d_1-d_4, various aggregators can be characterized as-

a_1: {(d_1, 0.5), (d_3, 0.2)}

a_2: {(d_1, 1.0), (d_2, 0.1), (d_4, 0.2)}

Aggregator a_1 can serve values of d_1 with an incoherency bound greater than or equal to 0.5 whereas a_2 can disseminate the same data item at a looser incoherency bound of 1.0 or more. In such a network of aggregators of multiple data items all the nodes can be considered as peers since a node a_i can help another node a_k to maintain incoherency bound of the data item d_1 (incoherency bound of d_1 at a_i is tighter than that at a_k) but the node a_i gets values of another data item d_2 from a_k.

1.1 Aggregate Queries and their Execution

In this paper we present a method for executing continuous multi-data aggregation queries, using a network of data aggregators, with the objective of minimizing the number of refreshes from data aggregators to the client. First we give two motivating scenarios where there are various options for executing a multi-data aggregation query and one must select a particular option to minimize the number of messages.

Scenario1: Consider a client query $Q = 50d_1 + 200d_2 + 150d_3$, where d_1, d_2, d_3 are different stocks in a portfolio, with a required incoherency bound of \$80. We want to execute this query over the data aggregators given in *Example1,* minimizing the number of refreshes.

Scenario2: In a sensor network, consider an AVG query over a *target* set of sensors (say, d_1, d_2 and d_3) injected at a *query* node. In-network aggregation is used for energy efficient propagation of aggregates [29]. For constructing an aggregation tree, connecting the *target* sensors and the *query* node, each node can select a path to the query node based on certain *preference factor*. We want to select the in-network aggregation path such that the aggregation query gets executed with the minimum number of messages.

In both the cases a limited number of options are available for executing the aggregation query. In this paper we will use *Scenario1* as the running example but results obtained and conclusions drawn are applicable to both the scenarios. Specifically we answer the question: *Given a client query posed over a hypothetical database of multiple data sources, what sub-queries should be posed at various data aggregators so that the number of refreshes from these aggregators to the client can be minimized?* We use additive aggregation queries to develop our approach in detail and towards the end of the paper describe how *max/min* queries can be handled.

For answering the multi-data aggregation query in *Scenario1*, there are three options for the client to get the query results. Firstly, the client may get the data items d_1, d_2 and d_3 separately. The query incoherency bound can be divided among data items in various ways ensuring that query incoherency is below the incoherency bound. In this paper, we show that getting data items independently is a costly option. This strategy ignores the fact that the client is interested only in the aggregated value of the data items and various aggregators can disseminate more than one data item.

Secondly, 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 (d_q=50 d_1 + 200 d_2 + 150 d_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 data item updates may cancel out each other, thereby maintaining the query results 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 refresh 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.

A third 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 multiple sub-queries. For the DAs given in *Example1*, the query Q can be divided in two alternative ways:

Plan1: Result of sub-query 50 d_1 + 150 d_3 is served by a_1 whereas value of d_2 is served by a_2.

Plan2: Value of d_3 is served by a_1 whereas result of sub-query 50 d_1 + 200 d_2 is served by a_2.

In both the plans, combining the sub-query values at the client gives the query result. But, selecting the optimal plan among various options is not-trivial. Intuitively, 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. In the above example, if updates to d_1 and d_3 are such that when d_1 increases, d_3 decreases, and vice-versa, then selecting *plan1* may be beneficial. We give a method to select the query plan based on these observations. While solving the above problem, 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 d_1 + 200 d_2 + 150 d_3 is divided into two sub-queries 50 d_1 + 130 d_2 and 70 d_2 + 150 d_3), in doing so the same data item is processed at multiple aggregators, increasing the unnecessary processing load (further, in case of paid data subscriptions it is not prudent to get the same data item from multiple sources). By dividing the client query into disjoint sub-queries we ensure that a data item update is processed only once for each query.

Sub-query incoherency bounds are required to be derived using the 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*, incoherency bound allocated to the sub-query $50d_1 + 150d_3$ should be greater than

55 (=50*0.5+150*0.2) as that is the tightest incoherency bound which the aggregator a_1 can satisfy for the sub-query $50d_1 + 150d_3$. We show that the number of refreshes also depends on the division of the query incoherency bounds among sub-query incoherency bounds. A similar result was reported for data incoherency bounds in [11]. Next we present problem statement formally and our contributions.

1.2 Problem Statement and Contributions

Value of a continuous weighted additive aggregation query, at time t, can be calculated as:

$$V_q(t) = \sum_{i=1}^{n_q} (v_{qi}(t) \times w_{qi}) \tag{1}$$

V_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_{qi}, $1 \le i \le n_q$. 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.

Suppose the result for the query given by Equation (1) needs to be continuously provided to a user at the query incoherency bound C_q. Then, the dissemination network has to ensure that:

$$\left| \sum_{i=1}^{n_q} (v_{qi}(t) - u_{qi}(t)) \times w_{qi} \right| \le C_q \tag{2}$$

Whenever data values at sources change such that query incoherency bound is violated, the updated value should be refreshed to the client. If the network of aggregators can ensure that the i^{th} data item has incoherency bound C_{qi} then the following condition ensures that the query incoherency bound C_q is satisfied:

$$\sum_{i=1}^{n_q} (C_{qi} \times w_{qi}) \le C_q \tag{3}$$

The 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 a sufficient condition for satisfying the query incoherency bound but not 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.

We need a method for (a) optimally dividing a client query into sub-queries and (b) assigning incoherency bounds to them such that (c) the derived sub-queries can be executed at chosen DAs and (d) total query execution cost, in terms of number of refreshes to the client, is minimized.

We prove that the problem of choosing sub-queries while minimizing query execution cost is an NP-hard problem. We give efficient 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 [5,11,12] 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 the 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 dissemination costs of the individual data items involved. The data dissemination cost is dependent on data dynamics and the incoherency bound associated with the data. We model the data dynamics using a data dynamics 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.

To empirically evaluate our approach, we use real-world data from the sensor and stock market domains [9]. Sensor network data used were temperature and wind sensor data from Georges Bank Cruises Albatross Shipboard [13]. Stock traces of 45 stocks were obtained by periodically polling *http://finance.yahoo.com*. We collected 20000 samples for each data item with a period of 5 seconds. Appendix A gives statistical properties of some of these stock traces. In this paper we present results using stock data only but similar results were obtained for sensor data as well [8]. Our simulation studies show that for continuous aggregation queries:

- *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 reducing the number of refreshes more dynamic data items should be part of sub-query involving larger number of data items.

Our method of executing queries over a network of data aggregators is practical since it can be implemented using a mechanism similar to URL-rewriting [4] in content distribution networks (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 explained in *Example* 1.

We would like to differentiate the current work with that of designing a network of data aggregators for a specific set of client queries. Whereas we propose a method to answer a client query using a given network of data aggregators; if the client queries are fixed, one can use the client queries to optimally construct a network of data aggregators as in [5,7,22]. Our aim of minimizing the number of messages between aggregators and client compliments the works of [5,7,22]. Together, they can be used to minimize the total number of messages between data sources and clients.

1.3 Outline of the Paper

The cost model for data dissemination is developed in Section 2. In Section 3, we present the query cost model for the additive aggregation queries. It uses the data dissemination model and a measure for capturing correlation between data dynamics. Optimal query planning for additive queries is presented in Section 4. Results of performance evaluations of algorithms described in Section 4 are presented in Section 5. Section 6 discusses optimal query planning for MAX queries. Most conclusions drawn for this class of queries are similar to that for additive aggregation queries. Related work is presented in Section 7. Discussion about various aspects of our work, conclusions and future work are presented in Section 8. Table 1 gives summary of various symbols used in the paper and their descriptions.

Table1: Important symbols and their meaning

Symbols	Description
A	Set of aggregators in the network.
N	Number of data aggregators (DAs).
D	Set of data items disseminated by the network.
C	Incoherency bounds of data items.
a_k	k^{th} data aggregator, $1 \le k \le N$
D_k	Set of data items disseminated by the k^{th} DA.
d_{kj}	j^{th} data item disseminated by the k^{th} DA.
t_{kj}	Incoherency bound which a_k can ensure for d_{kj}.
q	Client query.
C_q	Incoherency bound for q.
n_q	Number of data items in q.
d_{qi}	i^{th} data item of the query q.
$v_{qi}(t)$	Value of the i^{th} data item of the query q at time t.
w_{qi}	Weight of the data item d_{qi} for the query q.
$V_q(t)$	Value of the query q at time t.
q_k	Sub-query of q to be executed at a_k .
C_{qk}	Incoherency bound of q_k.
R_q	*Sumdiff* of the query q.
ρ	Correlation measure between data items
α	Query satisfiability parameter

2 DATA DISSEMINATION COST MODEL

In this section we present the model to estimate the number of refreshes required to disseminate a data item while maintaining a certain incoherency bound. There are two primary factors affecting the number of messages that are needed to maintain the coherency requirement: (a)the coherency requirement itself and (b)dynamics of the data.

2.1 Incoherency Bound Model

Consider a data item which needs to be disseminated at an incoherency bound *C*, i.e., new value of the data item will be pushed if the value deviates by more than *C* from the last pushed value. Thus the number of dissemination messages will be proportional to the probability of $|v(t)-u(t)|$ greater than *C* for data value *v(t)* at the source/aggregator and *u(t)* at the client, at time *t*. A data item can be modeled as a discrete time random process [10] where each step is correlated with its previous step. In push based dissemination, a data source can follow one of the following schemes:

(a) Data source pushes the data value whenever it differs from the last pushed value by an amount more than *C*.

(b) Client estimates data value based on server specified parameters [12, 16]. The source pushes the new data value whenever it differs from the (client) estimated value by an amount more than *C*.

In both these cases, value at the source can be modeled as a random process with average as the value known at the client. In case (b), the client and the server estimate the data value as the mean of the modeled random process whereas in case (a) deviation from the last pushed value can be modeled as zero mean process. Using Chebyshev's inequality [10]:

$$P(|v(t)-u(t)| > C) \propto 1/C^2 \tag{4}$$

Thus, we hypothesize that the number of data refresh messages is inversely proportional to the square of the incoherency bound. A similar result was reported in [5] where data dynamics were modeled as random-walks.

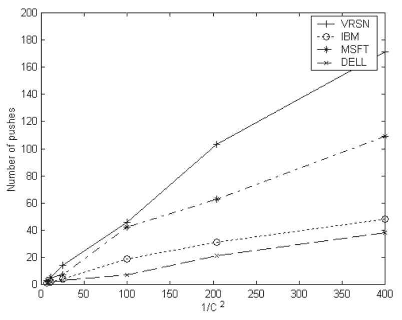
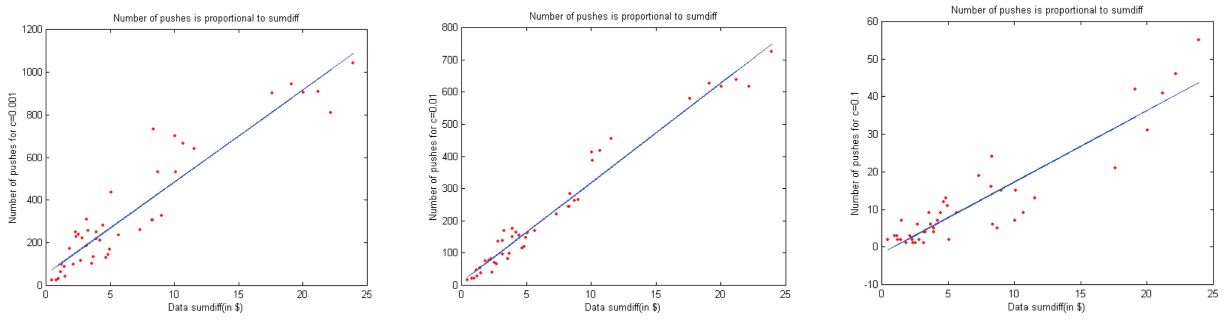


Figure 1. Number of pushes vs. incoherency bounds

Validating the analytical model: To corroborate the above analytical result we simulated data sources by reading values from the sensor and stock data traces, described in Section 1.2, at periodic instances. For these experiments, each data value at the first tick is sent to the client. Data sources maintain last sent value for each client. The sources read new value from the trace and send the value to its clients if and only if not sending it will violate the client's incoherency bound *C*. For each data item the incoherency bound was varied and refresh messages, to ensure that incoherency bound, were counted. Figure 1 shows the curves for the number of push messages, for four representative share price data items, as their corresponding incoherency bounds, and hence $1/C^2$, are varied. Besides validating the analytical model, these results provide one important insight into the dissemination mechanism. As the incoherency bound decreases, the number of messages increases as per analytical model, but there is a saturation effect for very low values of the incoherency bound (i.e., right part of the curve). This is due to the fact that the data items have limited number of discrete changes in the value. For example, if the sensitivity of a temperature sensor is one degree then number of dissemination messages will not increase even if incoherency bound is decreased below 1^o.



(a) C=0.001 (b) C=0.01 (c) C=0.1

Figure 2. Number of pushes vs. data *sumdiff*

2.2 Data Dynamics Model

We considered two possible options to model data dynamics. As a first option, the data dynamics can be quantified based on standard deviation of the data item values [25]. We take an example to show why standard deviation is not a good measure of data dynamics in our case: Suppose data values in consecutive instances for a data item d_1 are {0, 4, 0, 4, 0, 4, 0, 4} whereas for another data item d_2 values are {0, 0, 0, 0, 4, 4, 4, 4}. Suppose both data items are disseminated with an incoherency bound of 3. It can be seen that the number of messages required for maintaining the incoherency bound will be 7 and 1 for data items d_1 and d_2 respectively whereas both data items have the same standard deviation (=2.138). Thus we need a measure which captures data changes along with its temporal properties. This motivates us to examine the second measure.

As a second option we considered Fast Fourier Transform (FFT) which is used in the digital signal processing domain to characterize a digital signal. FFT captures number of changes in data value, amount of changes, and, their timings. Thus FFT can be used to model data dynamics but it has a problem. To estimate the number of refreshes required to disseminate a data item we need a function over FFT coefficients which can return a scalar value. The number of FFT coefficients can be as high as the number of changes in the data value. Among FFT coefficients, 0th order coefficient identifies average value of the data item whereas higher order coefficients represent transient changes in the value of data item. We hypothesize that the cost of data dissemination for a data item can be approximated by a function of the 1st FFT coefficient. Specifically, the cost of data dissemination for a data item will be proportional to data *sumdiff* defined as:

$$R_s = \sum_i | s_i - s_{i-1} | \quad (5)$$

where s_i and s_{i-1} are the sampled values of a data item S at ith and $(i$-1)th time instances (i.e., consecutive ticks). In practice, *sumdiff* value for a data item can be calculated at the data source by taking running average of difference between data values for consecutive ticks. For our experiments, we calculated the *sumdiff* values using exponential window moving average with each window having 100 samples and giving 30% weight to the most recent window.

Validating the hypothesis: We did simulations with different stocks being disseminated with incoherency bound values of $ 0.001, 0.01 and 0.1. This range is 0.1 to 10 times the average standard deviation of the share price values. Number of refresh messages is plotted with data *sumdiff* (in $) in Figure 2. The linear relationship appears to exist for all incoherency bound values. To quantify the measure of linearity we used Pearson product moment correlation coefficient (PPMCC) [19], a widely used measure of association, measuring the degree of linearity between two variables. It is calculated by summing up the products of the deviations of the data item values from their mean. PPMCC varies between –1 and 1 with higher (absolute) values signifying that data points can be considered linear with more confidence. For three values of incoherency bounds 0.001, 0.01, and 0.1; PPMCC values were 0.94, 0.96 and 0.90 respectively i.e., average deviation from linearity was in the range of 5% for low values of C and 10% for high values of C. Thus, we can conclude that, for lower values of the incoherency bounds, linear relationship between data *sumdiff* and the number of refresh messages can be assumed with more confidence. A larger error for larger values of C can be explained as follows:

As per the hypothesis, a larger value of data sumdiff should result in more refreshes. But that may not be true when either (1) there are low amplitude changes in the consecutive data values, smaller than the incoherency bound, which increases the data *sumdiff* value without requiring the dissemination of messages; or (2) there are high spikes such that they are much higher compared to the incoherency bound leading to more than proportional increase in the data sumdiff. The first case will be more prevalent for high values of the incoherency bound whereas the second case will be more pronounced for low values of the incoherency bound. Thus the linear relationship between the data *sumdiff* and number of refreshes has more error for very low values and very high values of incoherency bounds. As low amplitude perturbations are more prevalent than high amplitude spikes in most data items, the linear relationship is more accurate for lower values of incoherency bounds.

2.3 Combining Data Dissemination Models

Number of refresh messages is proportional to data *sumdiff* R_s and inversely proportional to square of the incoherency bound (C^2). Further, we can see that we need not disseminate any message when either data value is not changing (R_s = 0) or incoherency bound is unlimited ($1/C^2$=0). Thus, for a given data item S, disseminated with an incoherency bound C, the data dissemination cost is proportional to R_s/C^2. In the next section, we use this data dissemination cost model for developing cost model for additive aggregation queries.

3 Cost Model for Additive Aggregation Queries

Consider an additive query over 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:

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

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

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

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 a position to calculate R_{query} as different data items may be disseminated from different sources. We develop the query cost model in two stages.

3.1 Modeling Correlation between Data Dynamics

From Equations (6) and (7) we can see that if two data items are correlated such that as the value of one data item increases that of the other data item also increases, then R_{query} will be closer to R_{data}. On the other hand 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:

$$R^2_{query} \propto (w_p^2 R_p^2 + w_q^2 R_q^2 + 2\rho w_p R_p w_q R_q) \qquad (8)$$

The *correlation measure* ρ is defined such that $-1 \le \rho \le +1$. So, R_{query} will always be less than $|w_p R_p + w_q R_q|$ (as explained earlier) and always be more than $|w_p R_p - w_q R_q|$. The above relation can be better understood from its similarity with the standard deviation of the sum of two random variables [10]. For data items P and Q, ρ can be calculated as:

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

In Appendix B, we discuss a method for efficiently calculating ρ.

3.2 Query based 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 set of 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. Hence, query evaluation cost (as per R/C^2) of the SUM query will be half that of the AVG query. But, intuitively, disseminating the AVG of two data items at a given incoherency bound should require the same number of refresh messages as their SUM with double the incoherency bound. 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 the same as query with weights Ωw_i and incoherency bound ΩC. 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:

$$R^2_{query} = (w_p^2 R_p^2 + w_q^2 R_q^2 + 2\rho w_p R_p w_q R_q) / (w_p^2 + w_q^2 + 2\rho w_p w_q) \qquad (10)$$

i.e., the value of the normalizing factor for R_{query} should be $1/\sqrt{w_p^2 + w_q^2 + 2\rho w_p w_q}$. 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. Equation (10) can be extended to get query *sumdiff* for any general weighted aggregation query given by Equation (1) as:

$$R_Q^2 = \frac{\sum_{i=1}^{n_q} w_{qi}^2 R_i^2 + \sum_{i=1}^{n_q} \sum_{j=1, j\ne i}^{n_q} \rho_{ij} w_{qi} w_{qj} R_i R_j}{\sum_{i=1}^{n_q} w_{qi}^2 + \sum_{i=1}^{n_q} \sum_{j=1, j\ne i}^{n_q} \rho_{ij} w_{qi} w_{qj}} \qquad (11)$$

3.3 Validating the Query Cost Model

To validate the query cost model we performed simulations by constructing 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 incoherency bounds such that their normalized values (using normalization factor as in Equation (11)) are between 0.01 and 0.5. Figure 3(a) shows that the number of messages is proportional to the normalized query *sumdiff* as calculated using Equation (11) if their normalized incoherency bounds are the same. In this case PPMCC value is found to be 95%. Similarly, Figure 3 (b) shows the dependence of the number of refreshes on $1/C^2$ to illustrate that the relationship that holds between them for single data item also holds for a query with multiple data items. We use this query cost model for query planning which is presented next.

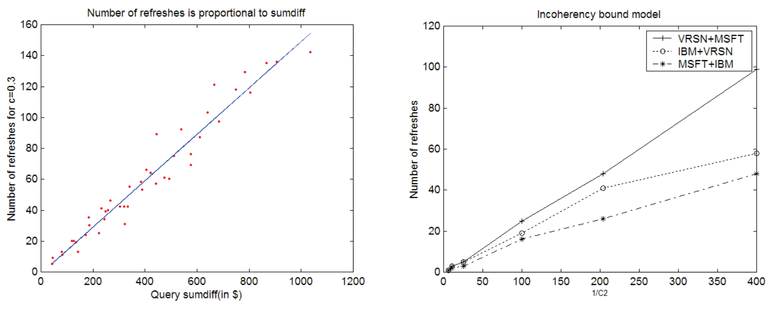


Figure 3: Query cost validation with varying (a) *Sumdiff* (b) Incoherency bound

4 Query Planning for Weighted Additive Aggregation Queries

For executing an incoherency bounded continuous query, a query plan is required. The query planning problem can be stated as:

Inputs: (1) A network of data aggregators in the form of a relation *f* (*A*, *D*, *C*) specifying the *N* data aggregators $a_k \in A$ ($1 \le k \le N$), set $D_k \subseteq D$ of data items disseminated by the data aggregator a_k and incoherency bound t_{kj} which the aggregator a_k can ensure for each data item $d_{kj} \in D_k$.

(2) Client query *q* and its incoherency bound C_q. An additive aggregation query *q* can be represented as $\sum w_{qi} d_{qi}$, where w_{qi} is the weight of the data item d_{qi} for $1 \le i \le n_q$.

Outputs: (1) q_k for $1 \le k \le N$, i.e., sub-query for each data aggregator a_k.

(2) C_{qk} for $1 \le k \le N$, i.e., incoherency bounds for all the sub-queries.

Thus, to get a query plan we need to perform following tasks:

1. *Determining sub-queries*: For the client query *q* get sub-queries q_ks for each data aggregator.
2. *Dividing incoherency bound*: Divide the query incoherency bound C_q among sub-queries to get C_{qk} s.

For optimal query planning, above tasks are to be performed with the following objective and constraints:

Optimization objective: Number of refresh messages is minimized. In Section 3, we have proved that, for a sub-query q_k, the estimated number of refresh messages is given by $\kappa R_{qk}/C_{qk}^2$ where R_{qk} is the *sumdiff* of the sub-query q_k, C_{qk} is the incoherency bound assigned to it and κ, the proportionality factor, is the same for all sub-queries of a given query *q*. Thus total number of refresh messages is estimated as:

$$Z_q = \kappa \sum_{k=1}^{N} \frac{R_{qk}}{C_{qk}^2} \tag{12}$$

Hence Z_q needs to be minimized for minimizing the number of refreshes.

Constraint1 *q_k is executable at a_k*: Each DA has the data items required to execute the sub-query allocated to it, i.e., for each data item d_{qki} required for the sub-query q_k, $d_{qki} \in D_k$.

Constraint2 *Query incoherency bound is satisfied*: Query incoherency should be less than or equal to the query incoherency bound. For additive aggregation queries, value of the client query is the sum of sub-query values. As different sub-queries are disseminated by different data aggregators, we need to ensure that sum of sub-query incoherencies is less than or equal to the query incoherency bound. Thus,

$$\sum C_{qk} \le C_q \tag{13}$$

Constraint3 *Sub-query incoherency bound is satisfied*: Data incoherency bounds at a_k (t_{kj} for $d_{kj} \in D_k$) should be such that the sub-query incoherency bound C_{qk} can be satisfied at that DA. The tightest incoherency bound T_{qk}, which the data aggregator a_k can satisfy for the given sub-query q_k, can be calculated as $T_{qk} = \sum_{n_{qk}} (w_{qi} \times t_{qj} | d_{qi} \equiv d_{kj})$. For satisfying this *constraint* we ensure $C_{qk} \ge T_{qk}$.

Following is the outline of our approach for solving this constraint optimization problem as detailed in the rest of this section: In Section 4.1, we prove that determining sub-queries while minimizing Z_q, as given by Equation (12), is *NP hard*. In Section 4.2 we show that, if the set of sub-queries (q_k) is already given, sub-query incoherency bounds C_{qk}s can be optimally determined to minimize Z_q. As optimally dividing the query into sub-queries is *NP-hard* and there is no known approximation algorithm, in Section 4.3, we present two heuristics for *determining sub-queries* while satisfying as many constraints as possible (*Constraint1* and *Constraint2* to be precise). Then we present variation of the two heuristics for ensuring that *sub-query incoherency bound is satisfied* (*Constraint3*). In particular, to get a solution of the query planning problem, the heuristics presented in Section 4.3 are used for *determining sub-queries*. Then, using the set of sub-queries, the method outlined in Section 4.2 is used for *dividing incoherency bound*.

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 [15].

3DM Problem: Given three sets *X*, *Y* and *Z*, each with *n* elements, and a set $M \subseteq X \times Y \times Z$, does there exists a subset $M' \subset M$ such that every element of sets *X*, *Y* and *Z* occur in M' once and only once? (The cardinality of M' will be *n* if it does exist).

We use a decision version of the query planning problem to reduce the 3DM problem. To solve the 3DM problem we reduce it to a SUM query of *3n* items, and incoherency bound *n*, as given in Appendix C. In the appendix, we prove that the best query plan having query cost of *3n* will consist of *n* sub-queries each with 3 data items and sub-query incoherency bound of 1. If we can find such an optimal plan, three data items from the chosen data aggregators form a triplet for the set M' while ensuring that each and every element of sets *X*, *Y* and *Z* occurs once and only once in M'. Conversely, there can be cases when:

(1) The sum query can not be satisfied as no combination of data aggregator can disseminate all the query data items. In this case, we can easily see that there can not be any $M' \subset M$ such that all elements of *X*, *Y* and *Z* be in M'.

(2) Cost of selected optimal plan is more than 3*n*. That implies that the plan has at-least one data aggregator disseminating sub-query with less than 3 data items (see Appendix C). Since we get M' using all the elements of the selected data aggregators (step 5 in the appendix), some elements are repeated in M'.

In both of these cases 3DM is answered in negation. Thus, using 3DM we have proved that optimal planning problem is *NP*-hard. For the purpose of the next sub-section (4.2) we assume that we have already *determined sub-queries* while satisfying *Constraint1* and we show that *incoherency bound division* can be performed optimally while satisfying *Constraint2* and *Constraint3*.

4.2 Optimal Allocation of Query Incoherency Bound among Sub-queries

If we know the division of the client query into sub-queries, using Equation (11) we can calculate *sumdiff* values of all the sub-queries. Thus, we need to minimize Z_q given by Equation (12) subject to *Constaint2* (*query incoherency bound is satisfied*) and *Constaint3* (*sub-query incoherency bound is satisfied*). We can get a close form expression by solving Equation (12) with Equation (13) using Lagrange Multiplier scheme (See Appendix D). In that scheme we minimize $\sum_{k=1}^{N}(R_{qk}/C_{qk}^2)+\lambda(\sum_{k=1}^{N}C_{qk}-C_q)$ for a constant λ to get values of C_{qk} s as:

$$C_{qk}=C_q R_{qk}^{1/3}/(\sum_{k=1}^{N}R_{qk}^{1/3}) \quad (14)$$

i.e., without the *Constaint3*, sub-query incoherency bounds should be allocated in proportion to $R_{qk}^{1/3}$. In Section 4.3, we use this expression to develop heuristics for optimally dividing the client query into sub-queries. If we also consider *Constaint3* then, we can model the problem of minimization of Z_q (while satisfying *Constaint2* and *Constaint3*) as a (non-linear) convex optimization problem. The non-linear convex optimization problem can be solved using 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 for a given set of sub-queries. In the next sub-section we describe two greedy heuristics to *determine sub-queries* while using the formulations developed in this section.

```
result ← $\phi$;
while $M_q \neq \phi$
    choose a sub-query $m_i \in M_q$ with criterion ψ;
    result ← result $\cup m_i$; $M_q \leftarrow M_q - \{m_i\}$;
    for each data item $d \in m_i$
        for each $m_j \in M_q$
            $m_j \leftarrow m_j - \{d\}$;
            if $m_j = \phi$    $M_q \leftarrow M_q - \{m_j\}$;
            else calculate sumdiff for modified $m_j$;
return result
```

Figure 4: Greedy algorithm for query plan selection

4.3 Greedy Heuristics for Deriving the Sub-queries

Figure 4 gives the outline of greedy algorithm for deriving sub-queries. First, we get a set of *maximal* sub-queries (M_q) corresponding to all the data aggregators in the network. The *maximal* sub-query for a data aggregator is defined 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). For example, consider a client query $50d_1 + 200d_2 + 150d_3$. For the data aggregators a_1 and a_2 given in *Example 1*, the maximal sub-query for a_1 will be $m_1=50d_1 + 150d_3$, whereas for a_2 it will be $m_2=50d_1 + 200d_2$. For the given client query (q) and relation consisting of data aggregators, data-items, and data incoherency bounds ($f(A, D, C)$) maximal sub-queries can be obtained for each data aggregator by forming sub-query involving all data items in the intersection of query data items and those being disseminated by the DA. This operation can be performed in $O(|q|.max|D_k|)$ where $|q|$ is number of data items in the query, $max|D_k|$ is the maximum number of data items disseminated by any DA. For each sub-query $m \in M_q$, its *sumdiff* R_m can be calculated using Equation (11). Different criteria (ψ) can be used to select a sub-query in each iteration of various greedy heuristics. All data items covered by the selected sub-query are removed from all the remaining sub-queries in M_q before performing the next iteration. It should be noted that sub-queries for DAs can be null.

Now we describe two criteria (ψ) for the greedy heuristics; 1) *min-cost*: estimate of query execution cost is minimized, and 2) *max-gain*: estimated gain due to executing the query using sub-queries is maximized.

4.3.1 Minimum Cost Heuristic

As we need to minimize the query cost, a sub-query with *minimum cost per data item* can be chosen in each iteration of the algorithm given by Figure 4, i.e., criterion ψ≡ minimize ($R_m/C_m^2|m|$). But from Equation (14) we can see that the sub-query incoherency bounds should be allocated in proportion to $R_k^{1/3}$. Using Equations (12) and (14) we get:

$$Z_q^{\ 1/3}=\frac{1}{C_q^{2/3}}\sum_{k=1}^{N}R_{qk}^{1/3} \quad (15)$$

From Equation (15), it is clear that for minimizing the query execution cost we should select the set of sub-queries so that $\sum R_{qk}^{1/3}$ is minimized. We can do that by using criterion ψ≡ *minimize* ($R_m^{1/3}/|m|$) in the greedy algorithm. Once we get the optimal set of sub-queries we can use Equation (15) and *Constraint3* ($C_{qk}\geq T_{qk}$) to optimally allocate the query incoherency bound among them using any of the convex optimization techniques as discussed in Section 4.2. But this method of first deriving sub-queries and then allocating the incoherency bounds has a problem which is described next.

4.3.2 Satisfiability of sub-query incoherency bound

In the solution described in the previous section, we select the set of sub-queries (and corresponding DAs) and then allocate the incoherency bound among them using a convex optimization technique. But the problem of incoherency bound allocation among chosen DAs may not have any feasible solution. There may be situations where, although the given network of data aggregators is able to satisfy the query coherency requirements but once the set of sub-queries is selected the incoherency bound allocation is not possible. Such a situation can be illustrated with the help of the network of data aggregators consisting of two DAs a_1 and a_2 as given in *Example 1*. Consider a client query $Q=50d_1 + 200d_2 + 150d_3$ with an incoherency bound of 80. As discussed in Section 1, there are (at-least) two possible query plans (*Plan1* and *Plan2*) to answer this query. As suggested in the previous subsection, we select sub-queries having minimum $\sum R_{qk}^{1/3}$, thus based on data dynamics it is possible that we select *plan2* as the optimal plan. But from the specification of aggregators a_1 and a_2 in *Example1*, we see that it is not possible for *plan2* to satisfy the client specified incoherency bound as tightest incoherency bound that can be satisfied by the selected aggregators (T_{plan2}=50*1 +200*0.1 +150*0.2 =100) is greater than the query incoherency bound (=80). Thus although there exists a plan (T_{plan1}= 50*0.5+ 200*0.1+ 150*0.2=75) 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 query satisfiability and performance. Instead of selecting the sub-queries without considering the data incoherency bounds for the selected data aggregators, we select sub-queries using $\sum(R_m^{1/3}+\frac{\alpha T_m}{C_q R_m^{1/3}})$ as *expanded objective function*. The second term ensures that while selecting the optimal plan we *prefer* data aggregators having tighter data incoherency bounds (lower values of T_m) thus higher chances of satisfying the query. The tuning parameter (α) can be used to balance the objectives of minimizing query execution cost through sub-query selection and meeting the query coherency requirements. We use $T_m / C_q R_m^{1/3}$ in the second term as, according to Equation (14), optimal incoherency bound allocation is likely to be done proportional to $C_q R_m^{1/3}$. In Section 5.2, we measure the effects of the tuning parameter α on the query satisfiability.

4.3.3 Maximum Gain Heuristic

Now we present an algorithm which instead of minimizing the estimated query execution cost maximizes the estimated gains of executing the 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 (i.e., maximal sub-query). Thus, the relative gain for a sub-query $\sum w_i d_i$ can be written as:

$$G_m = \frac{\sum_i w_i R_i}{\sqrt{\sum_i w_i^2 R_i^2 + \sum_j \sum_i \rho_{ij} w_i w_j R_i R_j}} - 1 \qquad (16)$$

where R_i is *sumdiff* of the data item d_i. This algorithm can be implemented by using criterion ψ≡ *maximize* (G_m/| *m* |) 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:

$$G_m^{'} = G_m - \frac{\alpha(\sum_i w_i T_i)}{C_q R_m^{1/3}} \qquad (17)$$

where T_i is tightest incoherency bound that can be satisfied for the data item d_i and R_m is the sub-query *sumdiff*. Reasons for selecting the particular *extended objective function* are same as ones outlined for the *min-cost* heuristic.

To summarize, for a given client query and a network of data aggregators, first we get the maximal sub-queries for all data aggregators. We use heuristics described in this section to derive sub-queries. In these heuristics extended objective functions are used to have the desired level of query satisfiability. Then, the technique explained in Section 4.2 is used to allocate the query incoherency bound among the derived sub-queries.

5 Performance Evaluation

For performance evaluation we simulated a network of data aggregators of 200 stock data items over 100 aggregator nodes such that each aggregator can disseminate combinations of 25 to 50 data items. Data items were assigned to different aggregators using *zipf* distribution (*skew*=1.0) assuming that some popular data items will be disseminated by more DAs. Data incoherency bounds, for various aggregator data items, were chosen uniformly between \$0.005 and 0.02. We created 500 portfolio queries such that each query has 10 to 25 randomly (using *zipf* distribution with the same default *skew*) selected data items with weights varying between 2 and 10. These queries were executed with incoherency bounds between 1.0 and 3.0 (i.e., 0.02-0.07% of the query value). Although here we present results for stock traces (man-made data), similar results were obtained for sensor traces (natural data) as well [8]. In the first set of experiments, we kept data incoherency bounds at the data aggregators very low so that query satisfiability can be ensured while keeping default value of α as 0.

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 incoherency bound (*naïve*): In this algorithm, the client query is executed with each data item being disseminated to the client 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 independently but incoherency bound is divided among data items (using Equation (14)) so that total number of refreshes can be minimized..

3. Random sub-query selection (*random*): In this case, sub-queries are obtained by randomly selecting a DA in the each iteration of the greedy algorithm (Figure 4). This algorithm is designed to see how the random selection works in comparison to the *sumdiff* based algorithms.

4. Sub-query selection while minimizing *sumdiff* (*min-cost*): This algorithm is described in Section 4.3.1.

5. Sub-query selection while maximizing gain (*max-*

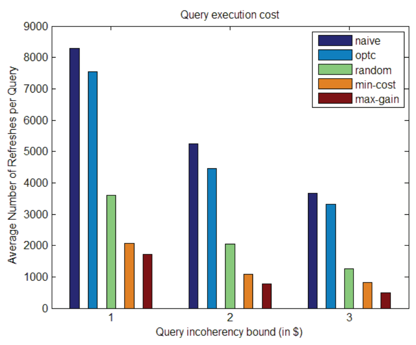
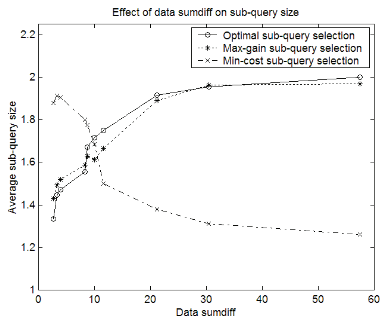
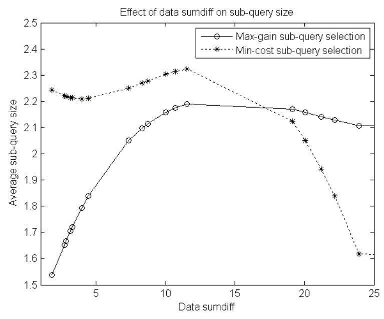


Figure 5: Performance evaluation of algorithms



(a) Query size=3



(b) Query size=5

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

***gain*)**: This algorithm is described in Section 4.3.3.

Figure 5 shows average number of refreshes required for query incoherency bounds of $1-$3. The naïve algorithm requires more than five times the number of messages compared to *min-cost* and *max-gain* algorithms. For incoherency bound of $3, each query, on average, requires 3311 messages if it is executed just by optimizing incoherency bound (*optc*) compared to 487 when we select the query plan using the *max-gain* algorithm. The gains of our algorithms increase further as number of data items disseminated by data aggregators increase (naïve requires more than 10 times the messages when each data aggregate disseminates 50 data items). This happens as, with more data items per DA, sub-query based algorithms result in larger sub-queries and we select sub-queries intelligently.

In the above experiment, for creating queries we selected the query data items with the same *zipf* distribution (*skew*=1.0) as we used for selecting data items to be served by DAs. But if we reduce the skew (i.e., having queries with less popular data items), we found that the performance of sub-query based algorithms suffer. This happens because for better performance, sub-query based algorithms depend on query data items being disseminated by the same DAs. For queries with less popular data items probability of this happening is less, hence, the inferior performance.

Further, although the optimization problem is similar to the covering a set of data items (query) using its subsets (sub-queries) for which the greedy *min-cost* algorithm is considered to be most efficient [28], 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.

5.2 Effects of Algorithmic Parameters

This set of experiments was 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 the 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 the size of the sub-query 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 randomly chosen 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. For these optimal query plans, Figure 6 (a) 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 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 by using *max-gain* algorithm we achieve our objective of including more dynamic data items as part of larger sub-queries. In comparison, for 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 iteration of the greedy algorithm (Figure 4), there is less chance of selecting a sub-query with more dynamic data item. Thus, it is very likely that the most dynamic data item will be disseminated as a single item sub-query resulting in bad performance of the client query. For the *max-gain* and *min-cost* algorithms, similar results were obtained for larger query sizes as well as shown in Figure 6(b). For generating results of Figure 6(b) we simulated 100 data aggregators, each disseminating 3 data items, while each query had 5 data items.

5.2.2 Effect of correlation between data dynamics

To measure the effects of correlation between data dynamics (as measured using *correlation measure*) 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 (each with *sumdiff*=1) 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). We found that by considering correlation measure number of refreshes reduce by approximately 10-12%. This result indicates that for optimal query planning data dynamics and incoherency bound allocation may be more important factor than the *correlation measure*.

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 .01 and 0.04. Value of α was varied between 0 and 20. The case α=0 corresponds to the algorithm without dealing with the query satisfiability. Figure 7 shows 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 decreases 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. At the query incoherency bound of $2, 40% queries can not be satisfied by the optimally selected data aggregators but as we increase the value of α to 10, only 2% queries are unanswered.

The value of α can be chosen to balance the performance and satisfiability of queries. For example, a network of data aggregators 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.

5.3 Overheads of Query Planning

Now we report the time overheads for various query planning operations. We measured these costs by varying the number of data items being disseminated by the network, between 40 and 200. These experiments were done on a WindowsXP machine with 2.53 GHz Intel Core-Duo CPU and 3GB RAM. For various *sumdiff* based algorithms, we need to maintain the *sumdiff* values of various data items (proportional to the number of data items being disseminated) and the correlation measure for each pair of data items (proportional to the square of the number of data items), in addition to the query dependent planning cost. For a trace size of 10000 (for each data item) both -- the cost of maintaining *sumdiff* per data item and the cost of maintaining *correlation measure* for each pair of data items -- were found to be in the range of 50-70 microseconds. Query planning cost (time required to derive sub-queries and their associated incoherency bounds) for *naïve* and *optc* algorithm was found to be approximately 1 microsecond per query whereas the same for the *random*, *minCost* and *maxGain* algorithms was found to be 2.5, 2.2 and 1.7 milliseconds. Higher cost of query planning, for *the sumdiff* based algorithms, is justified by the savings we achieve in terms of number of messages for the whole duration of the continuous query. The query planning cost of *random* and *mincost* is higher as they require more iterations of the algorithm in Figure 4 (i.e., more sub-queries) compared to the *maxGain* algorithm.

6 QUERY PLANNING FOR MAX QUERIES

In this section we briefly describe the optimal query planning for MAX queries. MIN queries can be handled in the similar manner. A MAX query, where a client wants the maximum of a specified set of data item values, can be written as:

$$V_q(t) = \max(v_{qi}(t), 1 \le i \le n_q) \tag{18}$$

For MAX queries, relationship between the query incoherency bound and required data incoherency bounds is discussed in the literature [11,24]. According to one such formulation, 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:

$$C_i \le C_q, \forall i, 1 \le i \le n_q \tag{19}$$

In these queries even if values of one or more data item change (changing their individual incoherencies) it is possible that query incoherency remains unchanged. Thus, for a given MAX query, it is possible to have an individual

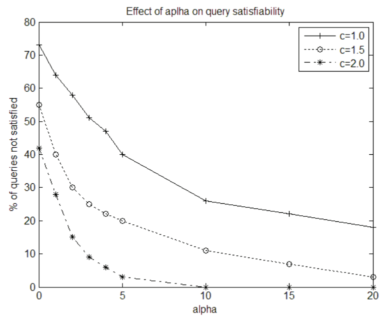
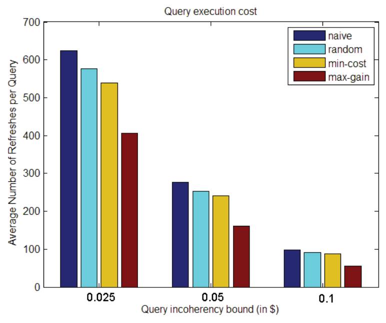
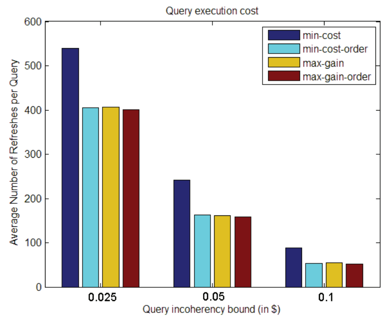


Figure 7: Effect of α on query satisfiability



(a) Comparison of algorithm



(b) Effect of *data dynamics order* on performance

Figure 8: Performance of MAX queries

data (or sub-query) incoherency bound which is more than the query incoherency bound. But such an incoherency bound will depend on instantaneous values of data items thus changing very dynamically. In this paper we do not consider such data value dependent incoherency bounds.

6.1 Query Cost Model

Let us consider a query Q= MAX (A, B), which is used for disseminating *max* of data items A and B from a data aggregator. Let the *sumdiff* values of A and B is R_a and R_b respectively. For a MAX query, the query result is the maximum of data item values. Thus the query dynamics is decided as per the dynamics of the data item with the maximum value. Hence, the query *sumdiff* is nothing but weighted average of data *sumdiffs*, weighted by fraction of time when the particular data item is *maximum*:

$$R_q \approx \sum_{i=1}^{n_q} R_i \left(\prod_{j=1, j\neq i}^{n_q} p(x_i > x_j) \right) \quad \leq \quad \max(R_i \mid 1 \leq i \leq n_q) \tag{20}$$

where $p(x_i > x_j)$ is the probability that value of ith data item is more than value of jth data item. We have the values of data item *sumdiffs* but for getting the probabilities we need to have exact values of data items. As query plan dependent on individual data values (instead of data dynamics) will be too volatile, as a first approximation we use upper bound of the expression given by Equation (20) as query *sumdiff*. Approximation used is the maximum of *sumdiff*s of data items involved. Now we consider the optimized execution of MAX queries using the above mentioned query cost model.

6.2 Optimized Execution

To execute the MAX query using a network of data aggregators, we assign sub-queries to different DAs. Each sub-query is a MAX query over a sub-set of query data items. For optimal planning we need to minimize the sum of sub-query execution costs. As we assign same incoherency bound to all the sub-queries (equals to the query incoherency bound as per Equation (19)), we just need to minimize sum of sub-query *sumdiff* values.

6.2.1 Optimal query planning problem is NP-hard

Optimal query planning problem for MAX queries is *NP-hard*. This can be proved by mapping the set cover problem to this optimal query planning problem.

Set cover problem: Given a universe U and a family S of subsets of U, a cover is a subset ($Cover \subseteq S$) of sets whose union is U. In the set covering optimization problem the task is to find a set covering which uses the fewest sets.

We can map the set cover problem to our query planning problem. The MAX query, corresponding to the set cover problem, will be *max* of all the items in the universe U at an incoherency bound 1. For each set $s \in S$ we assume the existence of a DA disseminating all the elements of s at an incoherency bound of 1. Further, let all data items have *sumdiff* value of 1. From Equation (20), we can see that cost of any sub-query will be 1. Thus cost of the client query, which is sum of cost of its sub-queries, will be same as the number of sub-sets required to get the set cover. It is easy to see that if we can solve the query planning problem optimally we can also solve the set cover problem optimally. Thus, now we give greedy heuristics for the sub-queries selection problem.

6.2.2 Greedy Heuristics

We use greedy algorithm given in Figure 4 for solving the query planning problem with different set of sub-query selection criteria (ψ) like the ones described in Sections 4.3.1 and 4.3.3. In the *min-cost* heuristic we select the sub-query having minimum sub-query *sumdiff* per data item. For the MAX query, sub-query *sumdiff* is nothing but the *sumdiff* of the most dynamic data item in the sub-query. Thus, for the *max-gain* heuristic, the gain of each sub-query is calculated as given in Equation (21):

$$G = \sum_{d_{qi}} R_{diq} - \max(R_{diq}) \tag{21}$$

where R_{diq} is *sumdiff* of ith data item of the query q.

6.2.3 Simulation results

Figure 8(a) shows simulation results for MAX query for various algorithms outlined in Section 5.1. We have not used *optc* algorithm here as all data items have to be served at the query incoherency bound without any optimization in the incoherency bound allocation. Naïve algorithm requires more than 1.5 times messages compared to other efficient sub-query based algorithms. Other results are qualitatively similar to what we obtained for the additive queries with one difference. For both types of queries the *max-gain* algorithm works better than the *min-cost* algorithm but, unlike in additive queries, in case of MAX queries performance of *min-cost* algorithm is closer to that of the *random* algorithm compared to the *max-gain* algorithm. This is a surprising result considering that *min-cost* is the most natural candidate (for *set cover* problem) with approximation guarantee of $log\ n_q$ [6]. For MAX queries, sub-query cost depends on the most dynamic data item. Thus we modified the greedy algorithms by considering the data items in the descending order of *sumdiff*s. For example, in the *max-gain* algorithm we first calculate gains of sub-queries covering the data item having maximum *sumdiff*. We select the one with maximum gain. We repeat the step for the next most dynamic data item and so on. Figure 8(b) shows with this modified greedy approach, performance of *min-cost* and *max-gain* algorithms is almost the same. This can be explained as with the data item ordering enforced, in the *min-cost* heuristic also, we ensure that the most dynamic data item is part of lower cost sub-query, leading to better query plan.

7 RELATED WORK

We divide the related work on scalable answering of aggregation queries over a network of data aggregators into two interrelated topics.

Answering Incoherency Bounded Aggregation Queries: Various mechanisms for efficiently answering incoherency bounded aggregation queries over continuously changing data items are proposed in the literature [9, 11, 14, 16, 20]. Our work distinguishes itself by employing sub-query based query evaluation to minimize number of refreshes. 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 predict data values and pull instances. In comparison, we use push based mechanism to refresh sub-query values at the client. In [11], authors propose push based scheme using data filters at the sources. According to that work, for an aggregation query, the number of refresh messages can be minimized by performing incoherency bound allocation to individual data items such that the number of messages from different data sources is the same. Instead we execute more dynamic data items as part of larger sub-queries while optimally assigning incoherency bounds. While this might lead to different messaging overheads for different DAs as opposed to what is proposed in [11], it does result in minimizing the total number of messages sent by DAs. Like us, authors of [20] also assume that dissemination tree from sensor nodes (data sources) to root (clients) already exists; and they also install error filters on partial aggregates (similar to incoherency bound assigned to sub-queries). But, in our work, each data aggregator can only disseminate data at some pre-specified incoherency bound depending on its capability whereas such a constraint does not exist for [20]. Further, we also give a method to select partial aggregates (sub-queries) to be used for answering the query.

In [29] authors propose cost-based methods to create in-network-aggregation tree consisting of the *query node*, where an aggregation query is invoked being the root of the aggregation tree, and sensors. Authors of [29] propose combinations of number of hops and remaining energy to select a particular path from various options available between any two nodes. Mapping their problem to the optimal query planning discussed in this paper, each communicating node can work as data source as well as data aggregator. Each node can select sub-queries based on their *sumdiff* values using principles outlined in this paper to minimize the number of message transfers in the network.

In [27], authors use data histograms to optimally assign local thresholds at monitoring sites for threshold monitoring at a central site. Maintaining histogram is a tedious task with more space and time overhead compared to the *sumdiff* based mechanism. Authors of [25] also use Chebyshev's inequality to show that expected communication cost is inversely proportional to square of the error-budget. But, compared to our work, they assume that number of refresh messages is proportional to data variance. As we have explained in Section 2, our *sumdiff* measure takes the order of data value changes into account which variance does not. Spatial and temporal correlations between sensor data are used to reduce data refreshes in [17,18]. We also consider correlation in terms of correlation measure between data items, but we use it for dividing client query into sub-queries. A method of assigning clients data queries to aggregators in a content distribution network is given in [12]. They do for individual data items what we do for queries consisting of multiple data items.

Construction and Maintenance of Network of Data Aggregators: Authors of [5,7,22] describe construction and maintenance of hierarchical network of data aggregators for providing scalability and fidelity in disseminating dynamic data items to a large number of clients. In these works, fidelity is defined as fraction of time when the client coherency requirements are met. Each data aggregator is given client requirements in the form of data items and their respective incoherency bounds. Instead, we use such networks for efficiently answering client's aggregation queries. One can use client queries to optimally construct a network of data aggregators while, on the other hand, one can also use a given network of data aggregators to efficiently answer client queries. Authors of [5,7,22] deal with the first part whereas we have studied the second part. Changes in data dynamics may lead to reorganization of the network of data aggregators which, in turn, may necessitate changes in query plans. It is a chicken and egg problem. Aggregators tree reorganization should be a longer term phenomenon (i.e., each incoming query should not lead to tree reorganization) whereas query plan can change more often depending on data dynamics.

Instead of optimizing fidelity of data items at data aggregators, as proposed in [7], using our work, one can optimize fidelity all the way up to client queries. Fidelity of a data item can be approximately calculated as number of dissemination messages multiplied by the total delay in the message transmission. Authors of [7] assume that each client's data requirements are fulfilled by a single data aggregator. But, in that case, data aggregators may need to disseminate a large number of data items which will lead to processing large number of refresh messages, hence, increase in delay. Thus, each client getting all its data items from a single data aggregator (using single sub-query) is optimal from number of messages point of view but not necessarily from the query fidelity point of view. By using our work, one can model expected number of messages for the client query. Thus, our work can complement the work of [7] for end-to-end (sources-to-client) fidelity optimization.

8 Discussion & Conclusion

This paper presents a cost based approach to minimize the number of refreshes required to execute an incoherency bounded continuous query. We assume the existence of a network of data aggregators, where each DA is capable of disseminating a set of data items at their pre-specified incoherency bounds. We developed an important measure for data dynamics in the form of *sumdiff* which, as we discussed in Section 2, is a more appropriate measure compared to the widely used standard deviation based measures. For optimal query execution we divide the query into sub-queries and evaluate each sub-query at a judiciously chosen data aggregator. Performance results show that by our method the query can be executed using less than one third the messages required for existing schemes. We showed that the following features of the query planning algorithms improve performance:

o Dividing the query into sub-queries (rather than data items) and executing them at specifically chosen data aggregators.

- Deciding the query plan using *sumdiff* based mechanism specifically by maximizing sub-query gains.
- Executing queries such that more dynamic data items are part of a larger sub-query.

We showed that the *max-gain* algorithm is very close to the optimal algorithm in selecting sub-queries based on data dynamics (Figure 6). Query satisfiability parameter (α) is employed for trade-off between query satisfiability and query performance. For any value of the query satisfiability parameter, there is always non-zero probability that a query will not get satisfied by the network of data aggregators. Usually, data aggregators disseminating same data item form a hierarchical network. In that case, even if a data aggregator can not satisfy its assigned query it can again apply the principles outlined in this paper to send a sub-query of the assigned query to its parents (which can disseminate the data item at a tighter incoherency bound). That will lead to poorer performance outlining the tradeoff between the query satisfiability and performance. Developing efficient strategies for multiple invocations of our algorithm, considering hierarchy of data aggregators, is an area for future research.

Another area for future research is changing a query plan as data dynamics changes. We are calculating data *sumdiff* in dynamic manner. If data *sumdiff* changes beyond a certain limit, the chosen query plan may not remain efficient. As a simple scheme, limits on changes to data *sumdiff* can be found for which the selected query plan remains optimal. Our work can also be used for extending the work proposed in [7] for construction and maintenance of a network of data aggregators so that end-to-end (sources-to-client) fidelity can be maximized. Our query cost model can also be used for other purposes such as load balancing various aggregators, multi-query execution, routing sensor data, etc. Using the cost model for these applications and developing the cost model for more complex queries is third area of our future work.

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] T. H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. MIT Press and McGraw-Hill 2001.

[7] Y. Zhou, B. Chin Ooi and Kian-Lean Tan, "Disseminating Streaming Data in a Dynamic Environment: An Adaptive and Cost Based Approach", The VLDB Journal, Issue 17, pg. 1465-1483, 2008.

[8] Query cost model validation for sensor data. www.cse.iitb.ac.in/~grajeev/sumdiff/RaviVijay_BTP06.pdf.

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

[10] Populis, A. Probability, Random Variable and Stochastic Process, Mc. Graw-Hill, 1991.

[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] S. Madden, M. J. Franklin, J. Hellerstein and W. Hong, "TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks", Proc. of 5th Symposium on Operating Systems Design and implementation, 2002.

[15] DS Johnson and MR Garey, Computers and Intractability: A Guide to the theory of NP-completeness. San Francisco, CA: Freeman, 1979.

[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] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos, "Processing Approximate Aggregate Queries in Wireless Sensor Networks", Information Systems vol. 31, Issue 8, Pg. 770-792, 2006.

[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.

[23] Brian Babcock and Chris Olston, "Distributed Top-K Monitoring", SIGMOD 2003.

[24] Adam Silberstein, Kamesh Munagala and Jun Yang, "Energy Efficient Monitoring of Extreme Values in Sensor Networks", SIGMOD 2006.

[25] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin and Y. Zhang, "STAR: Self-Tuning Aggregation for Scalable Monitoring", VLDB 2007.

[26] R. Gupta and K. Ramamritham, "Optimized Query Planning of Continuous Aggregation Queries in Dynamic Data Dissemination Networks", WWW 2007.

[27] S. Kashyap, J. Ramamritham, R. Rastogi and P. Shukla, "Efficient Constraint Monitoring using Adaptive Thresholds", ICDE 2008.

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

[29] P. Edara, A. Limaye and K. Ramamritham, "Asynchronous In-network Prediction: Efficient Aggregation in Sensor Networks", ACM Transactions on Sensor Networks, Volume 4, Number 4, August 2008.

Rajeev Gupta got his BTech from Indian Institute of Technology (IIT) Kharagpur, India in Electronics Engineering. He is currently pursuing his PhD from IIT Mumbai, India in Computer Science. He is working as Researcher at IBM Research, New Delhi, India for last 10 years.
Krithi Ramamritham received the PhD in Computer Science from University of Utah and then joined the University of Massachusetts. He is currently at IIT Bombay as Professor in the Department of Computer Science. He is a fellow of IEEE and a fellow of ACM. He has served on numerous program committees of conferences and workshops. His editorial board contributions include *IEEE Transactions*, the *Real Time Systems Journal*, and the *VLDB Journal*.