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. 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. 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, 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.

I. INTRODUCTION

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. i. Password should be easy to remember.

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 \( v(t) \) denote the value of the \( i \)" data item at the data source at time \( t \); and let the value the data item known to the client be \( u(t) \). Then the data incoherency at the client is given by \(| v(t) - u(t) |\). 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., \(| v(t) - u(t) | > C \).

**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. It is 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 such network of data aggregators, data refreshes occur from data sources to the clients through one or more data aggregators.

**Fig. 1** Data dissemination network for multiple data items

Here it is 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 \((d, c)\) pairs, where \( d \) is a data item which the DA can disseminate at an incoherency bound \( c \). 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.

**II. PROBLEM STATEMENT**

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

\[
V_q(t) = \sum_{i=1}^{n_q} (V_{q_i}(t) \times W_{q_i})
\]  

(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 \( w_{q_i} \). Suppose the result for the query given by Equation (1) needs to be
continuously provided to a user at the query incoherence bound $C_q$. Then, the dissemination network has to ensure that:

$$\sum_{i=1}^{n}(v_i(t) - u_i(t)) \times w_i \leq C_q$$

(2)

Whenever data values at sources change such that query incoherence bound is violated, the updated value should be refreshed to the client. If the network of aggregators can ensure that the $i_0$ data item has incoherence bound $C_{q_{i_0}}$ then the following condition ensures that the query incoherence bound $C_q$ is satisfied:

$$\sum_{i=1}^{n}(C_{q_{i}} \times w_{q_{i}}) \leq C_q$$

(3)

### III. DATA DISSEMINATION COST MODEL

The presented model is to estimate the number of refreshes required to disseminate a data item while maintaining a certain incoherence bound. There are two primary factors affecting the number of messages that are needed to maintain the coherence requirement:

(a) The coherence requirement itself and

(b) Dynamics of the data

Consider a data item which needs to be disseminated at an incoherence 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 $|v(t) - u(t)| > C$ 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 where each step is correlated with its previous step.

**a) Data Dynamics Model**

Specifically, the cost of data dissemination for a data item will be proportional to data $sumdiff$ defined as

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

(4)

Where $s_i$ and $s_{i-1}$ are the sampled values of a data item at $i^{th}$ 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.

**b) Incoherence Bound Model**

Data source pushes the data value whenever it differs from the last pushed value by an amount more than $C$. Client estimates data value based on server specified parameters. 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:

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

(5)

Thus, it hypothesize that the number of data refresh messages is inversely proportional to the square of the incoherency bound.

c) Combining data dissemination models

Number of refresh messages is proportional to data sum-diff $R$, and inversely proportional to square of the incoherency bound. Further, it need not disseminate any message when either data value is not changing ($R = 0$) 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 $R/C^2$. In the next section, it use this data dissemination cost model for developing cost model for additive aggregation queries.

IV. 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 then need to estimate its dissemination cost. If data items are disseminated separately, the query sum-diff will be:

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

(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 sum-diff will be:

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

(7)

$R_{query}$ is clearly less than or equal compared to $R_{data}$. Thus IT need to estimate the sum-diff of an aggregation query (i.e., $R_{query}$) given the sum-diff 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. Here develop the query cost model in two stages.

a) Modeling Correlation between Data Dynamics

From Equations (6) and (7) it 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 increase 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, it 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 $\rho$ is the correlation measure then $R_{query}$ can be written as:

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

(8)

The correlation measure $\rho$ is defined such that $-1 \leq \rho \leq 1$. So, $R_{query}$ will always be less than $|w_p R_p + w_q R_q|$ 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. For data items P and Q, $\rho$ can be calculated as:

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

(9)

b) Query based Normalization

Suppose there is need 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 $C = 2C$ 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 $R^2/C^3$) 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 $W_j$ and incoherency bound $C$ is the same as query with weights $nW_j$ and incoherency bound $nc$. So, while normalizing need to ensure that both, query weights and incoherency bounds, are multiplied by the same factor.

Normalized query $sumdiff$ is given by

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

(10)

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:
V. 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 \( a_k \in A(1 \leq k \leq N) \), set \( D_k \subseteq D \) of data items disseminated by the data aggregator \( a_k \) and incoherency bound \( t_{kij} \), which the aggregator \( a_k \) can ensure for each data item \( d_{kij} \in D_k \).
2. Client query \( q \) and its incoherency bound \( C_q \). An additive aggregation query \( q \) can be represented as \( \sum_{i=1}^{n_q} w_{qi} d_{qi} \).

Where \( w_{qi} \) is the weight of the data item \( d_{qi} \); for \( 1 \leq i \leq n_q \).

Outputs:
1. For (1 \( \leq k \leq N \)), i.e., sub-query for each data aggregator \( a_k \).
2. For (1 \( \leq k \leq N \)) i.e., incoherency bounds for all the sub queries. Thus, to get a query plan here need to perform following tasks:

1. Determining sub-queries: For the client query \( q \) get sub queries \( q_{k,s} \) for each data aggregator.
2. Dividing incoherency bound: Divide the query incoherency bound \( C_q \), among sub-queries to get \( C_{q,k,s} \).

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

Optimization objective: Number of refresh messages is minimized. For a sub query \( q_{k,s} \), the estimated number of refresh messages is given by \( k R_{q,k} / C_{q,k} \), where \( R_{q,k} \) is the sumdiff of the sub query \( q_{k,s} \), \( C_{q,k} \) is the incoherency bound assigned to it and \( k \), the proportionality factor, is the same for all sub queries of a given query \( q \). Thus total number of refresh messages is estimated as:

\[
Z_q = k \sum_{k=1}^{N} \frac{R_{q,k}}{C_{q,k}^2}
\]

(12)

Hence \( Z_q \), needs to be minimized for minimizing the number of refreshes.
Constraint 1: \( q_k \) 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 \( d_{q_k} \) required for the sub-query \( q_k \), \( d_{q_k} \in D_k \).

Constraint 2: 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, need to ensure that sum of sub-query incoherencies is less than or equal to the query incoherency bound. Thus

\[
\sum C_{q_k} \leq C_q
\]  
(13)

Constraint 3: Sub-query incoherency bound is satisfied: Data incoherency bounds at \( a_k \) should be such that the sub-query incoherency bound \( C_{q_k} \) can be satisfied at that DA. The tightest incoherency bound \( T_{q_k} \) which the data aggregator \( a_k \) can satisfy for the given sub-query \( q_k \) can be calculated as

\[
T_{q_k} = \sum_{n_{q_k}} (W_{q_i} \times t_{q_j} | d_{q_i} = d_{q_j})
\]

For satisfying this constraint ensure \( C_{q_k} > T_{q_k} \).

Following is the outline of my approach for solving this constraint optimization problem as detailed in the rest of this chapter: it proves that determining sub-queries while minimizing \( Z_q \), as given by Equation (12), is NP hard. If the set of sub-queries \( (q_k) \) is already given, sub-query incoherency bounds \( C_{q_k} \) 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, it present two heuristics for determining sub-queries while satisfying as many constraints as possible (Constraint1 and Constraint2 to be precise). Then it 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 are used for determining sub-queries. Then, using the set of sub-queries, the method outlined is used for dividing in incoherency bound

VI. OPTIMAL ALLOCATION OF QUERY INCOHERENCY BOUND AMONG SUB-QUERIES

If know the division of the client query into sub-queries, using Equation (11) can calculate sumdiffvalues of all the sub-queries. Thus, it need to minimize \( Z_q \), given by Equation (12) subject to Constraint2 (query incoherency bound is satisfied) and Constraint3 (sub-query incoherency bound is satisfied). Then it can get a close form expression by solving Equation (12) with Equation (13) using Lagrange Multiplier scheme. In that scheme minimizes

\[
\sum_{k=1}^{N} \left( R_{q_k} / C_{q_k}^2 \right) + \lambda \left( \sum_{k=1}^{N} C_{q_k} - C_q \right)
\]
For a constraint $\lambda$ to get values of $C_{qk}$ as
\[ C_{qk} = C_q R_{qk}^{1/3} \left( \sum_{k=1}^{N} R_{qk}^{1/3} \right) \]  
\[(14)\]
i.e., without the Constraint3, sub-query incoherency bounds should be allocated in proportion to $R_{qk}^{1/3}$. Use this expression to develop heuristics for optimally dividing the client query into sub-queries. If it also considers Constraint3 then, it can model the problem of minimization of $Z$, (while satisfying Constraint2 and Constraint3) 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. It used gradient descent method ($fmincon$ function in MATLAB) to solve this nonlinear optimization problem to get the values of individual sub-query incoherency bounds for a given set of sub queries. Here next describe two greedy heuristics to determine sub-queries while using the formulations developed in this section.

a) Greedy Heuristics for Deriving the Sub-queries

The algorithm gives the outline of greedy algorithm for deriving sub-queries. First, 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 $50d_1 + 200d_2 + 100d_3$. For the data aggregators $a_1$ and $a_2$, the maximal sub-query for $a_1$ will be $m_1=50d_1 + 100d_3$, 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 $O(l_ql_{max}\max \mid D_q \mid)$.

Where $l_q$ is number of data items in the query $max |D_q|$, 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 ($\psi$) 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 it describe two criteria ($\psi$) 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.

a) Minimum Cost Heuristic

As there is need to minimize the query cost, a sub-query with minimum cost per data item can be chosen in each iteration of the algorithm criterion $\psi = \text{minimize} \left( R_m^{1/3} C_q^{1/3} \mid m \right)$. But from Equation (14) it can see that the sub-query incoherency bounds should be allocated in proportion to $R_m^{1/3}$. Using Equations (12) and (14)
From Equation (15), it is clear that for minimizing the query execution cost it should select the set of sub queries so that \( \sum R_{qk}^{1/3} \) is minimized. It can do that by using criterion \( \psi = \text{minimize} (R_{m}^{1/3} / m) \) in the greedy algorithm. Once it get the optimal set of sub-queries it can use Equation (15) and Constraint 3 (\( C_{qk} \geq T_{qk} \)) to optimally allocate the query incoherency bound among them using any of the convex optimization techniques. But this method of first deriving sub-queries and then allocating the incoherency bounds has a problem which is described next.

b) Maximum Gain Heuristic

Now here presents 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, here calculate the relative gain of executing it by finding the \text{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:

\[
G_m = \frac{\sum w_i d_i}{\sqrt{\sum w_i^2 R_i^2 + \sum \sum \rho_{ij} w_i R_i R_j}} - 1
\]

Where \( R_i \) is \text{sumdiff} of the data item \( d_i \) this algorithm can be implemented by using criterion \( \psi = \text{maximize} (G_m / m) \) to get the set of sub-queries and corresponding DAs. Then it uses the convex optimization method to allocate incoherency bounds among sub queries. To tackle the query satisfiability issue the query gain Equation (16) is modified to:

\[
G'_m = G_m - \frac{\sum w_i T_i}{C_q R_{m}^{1/3}}
\]

where \( T_i \) is tightest incoherency bound that can be satisfied for the data item \( d_i \); and \( R'' \), is the sub-query \text{sumdiff} Reasons for selecting the particular extended objective function are same as ones outlined for the \text{min-cost} heuristic. To summarize, for a given client query and a network of data aggregators, first it get the maximal sub-queries for all data aggregators. Here it uses 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.
VII. QUERY PLANNING FOR MAX QUERIES

This describes 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:

\[ V_q(t) = \max(v_{qi}(t), 1 \leq i \leq n_q) \]

For MAX queries, relationship between the query incoherency bound and required data incoherency bounds. According to one such formulation, if the network of aggregators can ensure that the ith data item has incoherency bound \( C_i \), then the following condition ensures that the query incoherency bound \( C_q \) is satisfied:

\[ C_i \leq C_q, \forall i, 1 \leq i \leq n_q \]

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 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.

a) Query Cost Model

Let us consider a query \( Q = \text{MAX}(A, B) \), 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 \( R \) 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:

\[ R_q \approx \sum_{i=1}^{n_q} R_i \left( \prod_{j=1, j \neq i}^{n_q} p(x_i > x_j) \right) \leq \max(R_i | 1 \leq i \leq n_q) \]

where \( p(x_i > x_j) \) is the probability that value of ith data item is more than value of jth data item.

It has the values of data item sumdiffs but for getting the probabilities it 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 it uses upper bound of the expression given by Equation (20) as query sumdiff. Approximation used is the maximum of sumdiffs of data items involved. Now it considers the optimized execution of MAX queries using the above mentioned query cost model.
b) Greedy Heuristics

It uses greedy algorithm for solving the query planning problem with different set of sub-query selection criteria ($\psi$). In the min-cost heuristic it selects 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):

\begin{align*}
\text{result} & \leftarrow \phi \\
\text{while } M_q \neq \phi \\
\{ & \\
\text{Choose a sub-query } m_i \in M_q \text{ with criterion } \psi \\
\text{result} & \leftarrow \text{result} \cup m_i; M_q \leftarrow M_q - \{m_i\}; \\
\text{For each data item } d \in m_i \\
\text{For each } m_j \in M_q \\
\text{if } m_i = \phi & \hspace{1cm} M_q \leftarrow M_q - \{m_i\} \\
\text{else calculate sumdiff for modified } m_j; \\
\text{return result}
\}
\end{align*}

VIII. CONCLUSION

Here, presented a cost based approach to minimize the number of refreshes required to execute an incoherency bounded continuous query. It is 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 incoherence bounds. Developed an important measure for data dynamics in the form of sumdiff which, is a more appropriate measure compared to the widely used standard deviation based measures.

For optimal query execution it divides the query into sub-queries and evaluates each sub-query at a judiciously chosen data aggregator. Performance results show that by my method the query can be executed using less than one third the messages required for existing schemes. It is showed that the following features of the query planning algorithms improve performance:

1) Dividing the query into sub-queries (rather than data items) and executing them at specifically chosen data aggregators.
2) 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.
REFERENCES


