Handling Non-linear Polynomial Queries over Dynamic Data

Shetal Shah [#1], Krithi Ramamritham [#2]

[#] *Indian Institute of Technology Bombay, India*
[1] shetals@cse.iitb.ac.in
[2] krithi@cse.iitb.ac.in

***Abstract*— Applications that monitor functions over rapidly and unpredictably changing data, express their needs as *continuous* queries. Our focus is on a rich class of queries, expressed as *polynomials* over multiple data items. Given a set of polynomial queries at a coordinator $\mathcal{C}$, and a user-specified *accuracy bound* (tolerable imprecision) for each query, we address the problem of assigning *data accuracy bounds* or filters to the source of each data item. Assigning data accuracy bounds for non-linear queries poses special challenges. Unlike linear queries, data accuracy bounds for non-linear queries depend on the current values of data items and hence need to be recomputed frequently. So, we seek an assignment such that a) if the value of each data item at $\mathcal{C}$ is within its data accuracy bound then the value of each query is also within its accuracy bound, b) the number of data *refreshes* sent by sources to $\mathcal{C}$ to meet the query accuracy bounds, is as low as possible, and c) the number of times the data accuracy bounds need to be *recomputed* is as low as possible.**

In this paper, we couple novel ideas with existing optimization techniques to derive such an assignment. Specifically, we make the following contributions: (i) Propose a novel technique that significantly reduces the number of times data accuracy bounds must be recomputed; (ii) Show that a small increase in the number of data refreshes can lead to a large reduction in the number of recomputations; we introduce this as a tradeoff in our approach; (iii) Give principled heuristics for addressing negative coefficient polynomial queries where no known optimization techniques can be used; we also prove that under many practically encountered conditions our heuristics can be close to optimal; and (iv) Present a detailed experimental evaluation demonstrating the efficacy of our techniques in handling large number of polynomial queries.

I. Introduction

Applications of on-line monitoring belong to a wide spectrum: preventive monitoring to track unusual network activity, monitoring to exploit exchange disparities, critical health monitoring, etc. Monitoring needs are usually expressed as *continuous queries* over data that changes continuously and unpredictably, e.g., network traffic information, stock prices and sensor data. In this paper, we focus on a class of continuous queries where a user is interested in the *value* of a *polynomial* over many data items. This rich class of queries, formally defined in Section I-A, covers a wide range of applications like:

1. **Financials**: (a) In a *global portfolio query*, the user is interested in stocks across exchanges in different countries. In this case, the query posed will be:

$\sum$*(Number of stocks of company* i $*$ *current stock price of company* i *in country* j $*$ *currency exchange rate of country* j*)*

Here, both the stock price in foreign currency and the currency exchange rate change continuously.

(b) In an *arbitrage query*, the user is interested in exploiting the difference in the price of securities or foreign exchange in different markets. For example:

$\sum$*(Amount of foreign exchange* $*$ [[*buy price at exchange* i $*$ *currency exchange rate of country* j] - [*sell price at exchange* k $*$ *currency exchange rate of country* l]]

2. **Monitoring Dynamic Physical Phenomena**: Regularly monitored physical phenomena often depend on the value of polynomials. For example,

A disaster management group is interested in tracking an (approximately circular) oil spill. Assuming that sensors track points on perimeter of the spill, the centre of the circle (x_0, y_0) can be approximated as the average of these points. Let (x_1, y_1) be a point on the perimeter. Then the query tracking the area under the spill is: $\pi((x_1 - x_0)^2 + (y_1 - y_0)^2)$. Here both (x_0, y_0) and (x_1, y_1) change continuously.

The data required for monitoring could be distributed amongst many sources. User queries are posed at *coordinators*. The sources refresh the coordinators with the data necessary to continuously answer these queries. Since data changes rapidly, the volume of data refreshes required to handle the queries is high. Resource constraints like CPU/battery life at the sources, underlying network capacity, etc., place a restriction on the amount of data that can be sent to evaluate a user query.

Fortunately, users of rapidly changing data can tolerate a certain amount of imprecision in the query results. The amount of tolerable imprecision will vary from, for e.g., a fraction of a cent to a few dollars, depending on the usage of the query. We refer to the *maximum* imprecision that a user can tolerate in a query value as a *Query Accuracy Bound* or QAB. Consider a user query Q with an accuracy bound B at a coordinator $\mathcal{C}$. Let S be the virtual source[1] representing distributed sources $S_1, \cdots, S_m$. Let $V(S, Q)$ and $V(\mathcal{C}, Q)$ denote the value of Q at S and $\mathcal{C}$, respectively, at time t. To maintain Q's accuracy bound, we must have, at all times, $|V(\mathcal{C}, Q) - V(S, Q)| \leq B$.

[1]We use this abstraction only for conceptual understanding and ease of explanation.

QAB: Query Accuracy Bound
DAB: Data Accuracy Bound
PQ: Non-linear Polynomial Query
PPQ: Positive-coefficient Polynomial Query
LAQ: Linear Aggregate Query
ddm: data dynamics model

Fig. 1. List of Acronyms

$V(S,x)$, $V(S,y)$	$V(S,Q)$	$V(\mathcal{C},x)$ $V(\mathcal{C},y)$	$V(\mathcal{C},Q)$	*Remarks*
2, 2	4	2, 2	4	
3, 2	6	3, 2	6	S pushes x to $\mathcal{C}$
3.9, 2.9	11.31	3, 2	6	b_x, b_y **not valid**

Query $= xy : 5$, $b_x = 1$, $b_y = 1$

Fig. 2. DABs for PQs *depend* on current data values

Typically, many queries will be assigned to a coordinator $\mathcal{C}$. We would like the data at $\mathcal{C}$ refreshed such that the accuracy bounds of *all* the queries assigned to it are met. Our approach assigns a *filter* or a *Data Accuracy Bound* (DAB) to the *source* of each data item. The data accuracy bound, b_x, is the tolerable imprecision for data item x for $\mathcal{C}$. If $V(S,x)$ and $V(\mathcal{C},x)$ are the value of x at S and $\mathcal{C}$ respectively, then, to maintain this DAB, we must have, at all times, $|V(\mathcal{C},x) - V(S,x)| \leq b_x$.

We assume that the sources *push* data to $\mathcal{C}$, such that the DABs are maintained. For e.g., if the value of x last pushed by source S to C is 5 and $b_x = 1$, then S will next refresh $\mathcal{C}$ when the value of x at S becomes ≤ 4 or ≥ 6.

A. Polynomial Queries (P:B)

Let sources $S_1, \cdots, S_m$ serve data items $\{x_1, \cdots, x_n\}$. These could be individual data items like stock prices, aggregate functions like frequency count or averages of streamed values over a sliding window. In this paper, we focus on user queries which are polynomials in multiple data items. We define a polynomial query below.

A **Polynomial Query** (PQ) is a query where the user is interested in the value of a polynomial, e.g., global portfolios, volumes, areas, etc. A PQ can be written as:
$w_1({x_1}^{p_1}{x_2}^{q_1}\cdots{x_m}^{r_1}) + \ldots + w_n({x_1}^{p_n}{x_2}^{q_n}\cdots{x_m}^{r_n}) : B$
Here, each w_i is a constant representing the weight attached to the i^{th} term in the polynomial; B is a constant representing the QAB, $B > 0$; $p_i, q_i, \cdots, r_i$ are the exponents of the data items, $p_i, q_i, r_i, w_i, B \in \mathcal{R}$. The *degree* of the polynomial is the maximum of $(p_i + q_i + \cdots + r_i)$ over all terms.

A **Positive-coefficient Polynomial Query** (PPQ) is a polynomial query with all weights (coefficients) positive.

We refer to queries with degree 1 as linear aggregate queries. In a **Linear Aggregate Query** (LAQ), the user is interested in the value of a weighted sum of many data items and is expressed as $\sum_i w_i x_i : B$. This class handles a variety of queries like averages across data items, traffic monitoring and network monitoring.

LAQs require simpler solutions than non-linear PQs and hence we treat LAQs separately. Due to space constraints, we are unable to present our work on LAQs here. This work is available at [1]. In this paper, we focus on non-linear polynomial queries, i.e., queries with degree greater than 1.

B. Problem Statement

Let sources $S_1, \cdots, S_m$ serve data items $\{x_1, \cdots, x_n\}$. Given a set G of PQs at a coordinator $\mathcal{C}$, our goal is to assign a DAB b_x to the source of each data item x, such that the following three conditions are met.

Condition 1: For each data item x, if the value of x is within b_x at $\mathcal{C}$ then values of all the queries at $\mathcal{C}$ are within their QABs. Formally, $\forall x, |V(\mathcal{C},x) - V(S,x)| \leq b_x$
$\Rightarrow \forall_{Q \in G}, |V(\mathcal{C},Q) - V(S,Q)| \leq B_Q$.
Condition 2: The number of refreshes sent from the sources to $\mathcal{C}$ required to meet the QABs, is as low as possible.
Condition 3: The number of times the DABs need to be recomputed is as low as possible.

Condition 1 ensures that in the absence of communication and computation delays, the QABs of all queries will be met at all times. However, since these delays are a reality, there may be time intervals during which the QABs will not be met. Hence, we would like a communication efficient assignment, i.e., an assignment where the number of refreshes required to be sent by the sources to $\mathcal{C}$, is as low as possible (Condition 2). By reducing the number of refreshes, the load on the network reduces, resulting in lower communication delays. Further, for each refresh that $\mathcal{C}$ receives, $\mathcal{C}$ checks which QABs will be violated and for those queries, sends the value of the query result to the respective users. The lower the number of refreshes at $\mathcal{C}$, the lesser is the computational load on $\mathcal{C}$ and the smaller the delay perceived by the user.

To motivate the need for Condition 3, consider a simple example query as illustrated in Figure 2. Here, the user is interested in the product of two data items, x and y, and the QAB is 5, i.e., $Q = xy : 5$. We need to calculate b_x and b_y, the DABs for x and y respectively. Initially, let the value of x and y at both the virtual source S and $\mathcal{C}$ be 2, i.e., $V(\mathcal{C},x) = V(S,x) = 2$ and $V(\mathcal{C},y) = V(S,y) = 2$. Then the value of the query: $V(\mathcal{C},Q) = V(S,Q) = 2 * 2 = 4$. S does not need to push data to $\mathcal{C}$ as long as the query value ≥ -1 and ≤ 9, i.e., the query validity interval is $[-1, 9]$ at $\mathcal{C}$.

Consider a DAB assignment $b_x = b_y = 1$. This is a valid assignment. As long as x and y do not change by 1, xy does not change by 5. Suppose x increases by 1 and S informs $\mathcal{C}$ of this change. Then $V(\mathcal{C},Q) = V(S,Q) = 2 * 3 = 6$. Note that now *the DABs are no longer* valid*, i.e, they no longer ensure that the QAB will be met*. For instance, if $V(S,x)$ were to change from 3 to say 3.9, and $V(S,y)$ were to change from 2 to 2.9, S will not send any update to $\mathcal{C}$, ($0.9 < b_x$, b_y), even though the change in query value at S, $V(S,Q) = 3.9 * 2.9 - 6 = 5.31$, is more than 5, the QAB.

The crucial fact to note here is that changes in the value of data items may render a valid DAB assignment invalid. More specifically, whenever the value of a data item x changes by the DAB, and hence $\mathcal{C}$ receives an update for x, it will result

in a recomputation of the DABs[2]. Each recomputation has a cost associated with it. Apart from the actual computational cost involved in a recomputation, it may typically also result in some extra communication messages [2], [3], [4], [5] or even a reorganization of the data dissemination network [6]. For instance, after each recomputation, some of the sources have to be informed about the new DABs. Assuming a query uses data items from m sources, after a query is recomputed, in the worst case, a DAB-change message will be sent to all m sources. For polynomial queries, this implies that *a data refresh message from* one *source may result in* m *DAB-change messages - one to each source*. A recomputation can therefore be more expensive than a refresh. Hence, it is critical to minimize not only the number of refreshes (Condition 2) but also the number of DAB recomputations (Condition 3).

C. Contributions: Efficient solutions for Polynomial Queries

Finding DABs for non-linear queries is a difficult problem [5], [7]. In this paper, we present *efficient* and *practical solutions* for handling a rich class of non-linear queries, viz, polynomial queries. We present an *optimal* solution for a positive-coefficient polynomial query and two heuristics for general polynomials, one of which is provably near-optimal for an important subset.

The key factors which make such a solution possible are: (i) identification that recomputations can be expensive and should be reduced, (ii) a novel idea to reduce the number of recomputations, and (iii) effective use of existing optimization techniques to derive optimal and near optimal solutions, wherever possible. Our contributions include:

An Optimal Solution to Minimize the Number of Refreshes for a PPQ: Finding DABs for a PQ is simpler if (i) only Conditions 1 and 2 (Section I-B) have to be satisfied, and (ii) all coefficients of the polynomial are positive. We solve this restricted problem, in Section III-A.1, in the following manner: we first derive a *necessary and sufficient condition* that individual DABs must satisfy so that the QAB is met (Condition 1). To meet Condition 2, we need to *estimate* the number of refreshes for a particular assignment. To do so, we make assumptions on the way data changes and consider two different data dynamics models (*ddm*s) (i) data changes monotonically at a uniform rate and (ii) data changes as a random walk. These *ddms* have also been used earlier in [4], [6], [8]. We make use of this estimate, and using geometric programming techniques, arrive at a solution *optimal* in the number of refreshes.

A Novel Dual-DAB Approach to Reduce the Number of Recomputations for a PPQ: Addition of Condition 3 increases the complexity of the problem considerably. As explained in Section I-B, every refresh arriving at a coordinator leads to a recomputation. Hence, intuitively, to reduce the number of recomputations, one should reduce the number of refreshes. We propose a counter-intuitive approach in Section III-A.2 - it actually *increases the number of refreshes to reduce the number of recomputations*. The key idea in our approach is to assign not one but *two* DABs for each data item, which together enable us to meet Conditions 1, 2 and 3. It introduces a tradeoff between the number of recomputations and the number of refreshes *leading to a large reduction in the number of recomputations for only a slight increase in the number of refreshes*. This approach coupled with existing optimization techniques gives a solution which is optimal in the total cost: (number of refreshes + number of recomputations).

Effective Heuristics to Handle a General Polynomial: To the best of our knowledge there is *no efficient optimization technique which can be used to find the optimal solution for a general PQ*, i.e., a polynomial with positive and negative coefficients. In Section III-B, we propose two heuristics based on the observation that a general polynomial can be written as the *difference* of two positive-coefficient polynomials. Furthermore we prove that, *under certain conditions*, which we feel occur in practice, *the solution given by one of the heuristics is close to the optimal solution*.

Solutions for Multiple PPQs: The formulation for one PPQ can be extended to handle multiple PPQs. However, because this simple extension has some drawbacks, we propose an alternate solution in Section IV.

Experimental Evaluation - Using Simulation and PlanetLab: Our results show that (i) our approach of having *two* DABs coupled with optimization techniques gives significantly superior results as compared to the use of single DABs, (ii) geometric programming techniques are practical to use, and (iii) to handle a *large number of PQ*s, *a solution which reduces the number of recomputations is a must* and this is what we provide. We have also implemented our solutions on a real-world test-bed, PlanetLab, results from which corroborate the findings from simulation studies (Section V).

D. Outline

Related work is discussed in Section II. Sections III and IV discuss our approach for a single PQ and multiple PQs, respectively. Section V presents the experimental evaluation of our techniques and Section VI concludes with future work.

II. Related Work

Algorithms to handle continuous approximate queries over distributed sources have been a focus of many research works [2], [4], [5], to name a few. [7] gives a comprehensive survey of the various issues in handling queries over data streams.

The problem of assigning DABs, given QABs has been addressed in many recent papers. The queries considered, architectures and refresh model used, vary from paper to paper. Data at a coordinator, $\mathcal{C}$, can be *refreshed* either by (a) $\mathcal{C}$ *pull*ing data values from respective sources, at specific points in time, or (b) sources *push*ing relevant new data values to $\mathcal{C}$. In [8], the authors use pull and focus on reducing the number of refreshes for multiple LAQs over multiple data items by using random walk $\mathtt{ddm}$ to calculate the next time to pull. Pull-based schemes rely heavily on the accuracy of $\mathtt{ddm}$s.

[2]This complication does not surface for LAQs and hence they admit simpler solutions.

Our approach uses push and guarantees that in a zero delay network, the QABs will always be met. Though we use ddms to estimate the number of refreshes, as shown Section V, our reliance on the accuracy of the model is low.

In [9], the source calculates the DAB (precision interval) for each refresh using a combination of push and pull. In [10], Kalman Filters are used to model data changes at sources to reduce the number of refreshes. Both these approaches do not handle multiple queries over multiple data items.

Solutions described in [3], [4], [5], [11] use push. In [11], the source cooperates with the users to periodically modify the DABs based on the bandwidth available. Our focus is to calculate the DABs based on QABs specified by the user.

In [4], an adaptive algorithm to handle single and multiple LAQs is proposed, where periodically DABs are recalculated and some of the sources are informed of the changed DABs.

In [3], a special LAQ, i.e., the sum of frequency counts, is considered. The QAB is a function of the query value and two algorithms are proposed to calculate DABs. However, both [3] and [4] do not consider non-linear queries and the solutions require frequent recomputations.

To the best of our knowledge, [5] is the first work to consider a general class of non-linear queries. The problem considered in the paper is: Given vectors $v_1, \cdots, v_n$ at n different sources, and any non-linear function on the weighted average of these vectors, when does a change at a source imply that value of the function is greater than a threshold? The solutions presented use geometric techniques to reduce the number of refreshes required. While the queries handled are more general than ours, their techniques do not yield optimal solutions for our problem (see Section V). The focus of our work is to reduce not only the number of refreshes but also the number of recomputations of DABs. Hence, we explicitly consider the cost of recomputing the DABs, leading to very different solutions. As we show in Section V, reducing just the number of refreshes is not enough, if a large number of queries are to be handled. Another key feature in our work is the explicit use of the estimate of the rate of change of data, which helps us obtain better solutions.

In [6], techniques are proposed to build a content dissemination network for dynamic data that preserves DABs. We use the work of [6] to experimentally evaluate the performance of our techniques on a network of distributed coordinators.

III. Deriving DABs: Single Query Case

We describe our solution for a positive-coefficient polynomial (PPQ) in Section III-A and that for a general PQ in Section III-B.

A. A Positive-coefficient Polynomial Query

We formulate the problem of assigning DABs for a PPQ as a geometric optimization problem. We now present our approach for a simple PPQ.

1) Optimal Refresh Approach: Meeting Conditions 1 and 2: We explain our formulation for a PPQ using a simple global portfolio query (Query 1(a)) as an example: a query Q where the user is interested in the product of two data items, $Q = (xy) : B$. Assume that Q is assigned to a coordinator $\mathcal{C}$. We proceed with the derivation of our formulation in a step-wise manner, treating each condition at a time.

Sources : S, $S_1, \cdots, S_m$ Coordinator : $\mathcal{C}$
Polynomial: P, P_1, P_2 Query : $Q = P : B$
Data Items : $x_1, \cdots, x_n, y_1, \cdots y_n$
Value of data item x at source S : $V(S, x)$
Value of data item x at $\mathcal{C}$: $V(C, x)$, V_x
Primary DAB for x : b_x Secondary DAB for x : c_x
Cost of a recomputation : μ
Rate of change of data item x : λ_x

Fig. 3. List of Notations

To handle Condition 1, we first derive a *necessary and sufficient condition* for the DABs for Q. Let V_x, V_y be the current values of x, y at the virtual source S and coordinator $\mathcal{C}$[3]. Then the value of Q is V_xV_y at S and $\mathcal{C}$. Let b_x and b_y be the DABs for x and y respectively. The *maximum difference in the query values at S and C will be when both x and y increase or both decrease*. Now suppose that x increases by b_x and y increases by b_y at S. Then, the value of Q at S is: $(V_x+b_x)(V_y+b_y) = V_xV_y+V_xb_y+V_yb_y+b_xb_y$. To meet Q's QAB, we need, $V_xV_y+V_xb_y+V_yb_x+b_xb_y-V_xV_y \leq B$,

$$\text{i.e., } V_xb_y + V_yb_x + b_xb_y \leq B \tag{1}$$

If x had decreased by b_x and y by b_y, then we would have $V_xV_y - (V_x - b_x)(V_y - b_y) = V_xb_y + V_yb_x - b_xb_y \leq B$. This is implied by Equation (1).

It is easy to see that Equation 1 is a necessary and sufficient condition for a DAB assignment. Any DAB assignment which satisfies the above condition is a feasible assignment. The overheads involved in the dissemination of the data items will *depend* on and vary with each assignment.

One of our goals is to minimize the number of refreshes required at $\mathcal{C}$ per unit time. To know the number of refreshes that a certain DAB might incur, we need to know how the data changes. We assume two different data dynamics models (i) data changes monotonically and (ii) data changes as a random walk. In this section, we describe our formulation for the monotonic data model. We extend this formulation to the random walk model in Section III-A.5. Let λ_i be the rate of change of data item x_i in unit time. Recall that x_i is refreshed when the value changes by b_i. Then, for the monotonic data model, the number of refreshes per unit time for x_i, is $\frac{\lambda_i}{b_i}$.

The total number of refreshes per unit time to meet the DABs is then $\frac{\lambda_x}{b_x} + \frac{\lambda_y}{b_y}$. Since we want to minimize the number of refreshes, we can formulate the following non-linear optimization problem, for Q, as:

$$\min \ \frac{\lambda_x}{b_x} + \frac{\lambda_y}{b_y} \quad \text{subject to} \quad V_xb_y + V_yb_x + b_xb_y \leq B$$

This formulation satisfies Condition 1 and 2.

We see from Equation 1 that the DABs depend on the *current* value of the data items. This means that each time x

[3]We use V_x instead of $V(C, x)$ for brevity

changes by b_x or y changes by b_y, we will need to recalculate the DABs. If this is not done then we cannot guarantee Condition 1, which is that the DABs ensure that the QABs will be met. Since every refresh to $\mathcal{C}$ results in a recomputation, a possible solution to reduce the number of recomputations at $\mathcal{C}$ would be to reduce the number of refreshes further. But our DAB calculation is already optimal in the number of refreshes. Our solution described next, slightly *increases* the number of refreshes to considerably reduce the number of recomputations.

2) Dual DAB Approach: Reducing the Frequency of Recomputations: We now propose a technique which (i) helps us reduce the number of recomputations, (ii) indicates when the next recomputation should occur, and (iii) guarantees that between two recomputations, the DABs ensure that the QAB is met. The technique introduces a tradeoff between the number of recomputations and refreshes received at $\mathcal{C}$, where the number of recomputations is reduced at the cost of a *small* increase in the number of refreshes.

Our idea is to assign *two* DABs to a data item x, namely,
(i) a *smaller* primary DAB which is used to evaluate the user query and which ensures that the QAB is met, and,
(ii) a *larger* secondary DAB which defines the range of values of x for which the primary DAB is *valid*, i.e., will ensure query accuracy. When the value of x goes beyond this range, the DABs have to be recomputed. For the query above, let the primary DABs be b_x and b_y and the secondary DABs be c_x and c_y. Then as long as the value of x is within $V_x \pm c_x$ and the value of y is within $V_y \pm c_y$, the primary DABs are valid.

We explain the intuition of our approach with the help of the example query $xy : 5$. If $\lambda_x = \lambda_y = 1$, then optimal refresh assignment is $b_x = b_y = 1$. This is the assignment discussed in Figure 2. Now consider an assignment, shown in Figure 4, where $b_x = b_y = 0.5$. Initially, the value of x and y at both S and $\mathcal{C}$ is 2. As before x increases by 1 at S and S pushes this change to $\mathcal{C}$. The new values of x and y at S and $\mathcal{C}$ are now 3 and 2, respectively. Note that the DABs $b_x = b_y = 0.5$ are still valid ((3 + 0.5) * (2 + 0.5) - 6 = 8.75 -6 = 2.5 $<$ 5). However, if we had done the optimal assignment, we would have now recomputed the DABs (as explained in Section I). Suppose x now changes to 3.5 and y changes to 2.5. Notice that the DABs are still valid.

The key insight is that *by assigning more stringent DABs, we can avoid recomputation over a larger range of data values*. However, more stringent DABs imply an increase in number of refreshes. S will now send all changes ≥ 0.5 instead of sending changes ≥ 1. In the example above, the primary DABs are valid till x increases to 5.5, and y increases to 4.5, (5.5 + 0.48 * 4.5 + 0.48 = 29.78 - 24.75 $>$ 5). Here, $c_x = (5.5-2) = 3.5, c_y = (4.5-2) = 2.5$ serve as the secondary DABs.

For a given set of primary DABs, one can have many secondary DABs. In the example above, $c_x = 2.5, c_y = 3.5$ are also valid secondary DABs. The decrease in the number of recomputations and the increase in the number of refreshes will vary with each assignment. The critical question then is how to choose the two DABs. We address this question now.

$V(S,x)$, $V(S,y)$	$V(S,Q)$	$V(\mathcal{C},x)$ $V(\mathcal{C},y)$	$V(\mathcal{C},Q)$	b_x, b_y
2, 2	4	2, 2	4	valid
3, 2	6	3, 2	6	valid
3.5, 2.5	8.75	3.5, 2.5	8.75	valid
3.9, 2.9	11.31	3.9, 2.9	11.31	valid
5.5, 4.5	24.75	5.5, 4.5	24.75	**invalid**

Query $= xy : 5$, $b_x = 0.5, b_y = 0.5$

Fig. 4. Reducing the number of recomputations

Calculating Primary and Secondary DABs

Our solution is to incorporate the primary and secondary DABs into the optimization formulation as follows.

The secondary DABs define the range of data values for which the primary DABs are valid. This means that b_x is valid from $V_x - c_x$ to $V_x + c_x$. Hence, for the user query $(xy) : B$, we need to satisfy:

$$(V_x + c_x + b_x)(V_y + c_y + b_y) - (V_x + c_x)(V_y + c_y) \leq B$$

$$\text{i.e., } (V_x + c_x)b_y + (V_y + c_y)b_x + b_x b_y \leq B \qquad (2)$$

and $(V_x - c_x)(V_y - c_y) - (V_x - c_x - b_x)(V_y - c_y - b_y) \leq B$

$$\text{i.e., } (V_x - c_x)b_y + (V_y - c_y)b_x - b_x b_y \leq B \qquad (3)$$

We can see that any assignment which satisfies Equation (2) will also satisfy Equation 3. Equation (2) is also the *necessary and sufficient condition* for the two DABs.

Since the secondary DABs define the range for primary DABs, they should be larger than the primary DABs. This gives us the additional constraints: $c_x \geq b_x$ and $c_y \geq b_y$. Assuming that the data changes *monotonically*, the number of refreshes arriving at $\mathcal{C}$ per unit time due to the primary DABs is: $\frac{\lambda_x}{b_x} + \frac{\lambda_y}{b_y}$.

The tradeoff between the number of recomputations and number of refreshes is modelled through a constant μ, $\mu \geq 0$. The computational cost of one recomputation, additional communication or reorganizational overheads due to this recomputation, etc., are approximated as μ messages.

A recomputation occurs when the value of x at $\mathcal{C}$ changes by more than $V_x \pm c_x$ or the value of y changes by more than $V_y \pm c_y$. The time between recomputations due to x is $\frac{c_x}{\lambda_x}$ and due to y is $\frac{c_y}{\lambda_y}$. Hence, the number of recomputations, R, is the maximum of $\frac{\lambda_x}{c_x}$ and $\frac{\lambda_y}{c_y}$.

We need to minimize both the number of refreshes arriving at $\mathcal{C}$ and also the number of recomputations. Hence, the objective function now becomes: $\frac{\lambda_x}{b_x} + \frac{\lambda_y}{b_y} + \mu * R$.

The optimization problem now becomes:

$$\min\left(\frac{\lambda_x}{b_x} + \frac{\lambda_y}{b_y} + \mu * R\right) \quad \text{subject to}$$

$$(V_x + c_x)b_y + (V_y + c_y)b_x + b_x b_y \leq B$$

$$c_x \geq b_x; \quad c_y \geq b_y; \quad \frac{\lambda_x}{c_x} \leq R; \quad \frac{\lambda_y}{c_y} \leq R$$

This is a non-linear optimization problem which can be solved by geometric programming techniques [12].

Note that the sources only need to be aware of the primary DABs. The secondary DAB is required only at the coordinator to check if a recomputation of the DABs is warranted.

3) **Calculation of μ:** The parameter μ approximates the sum total of all overheads of a recomputation as the number of messages. The overheads are: (i) Actual computational cost of recomputing, (ii) DAB-change messages sent to the sources and (iii) Other miscellaneous costs such as cost of reorganization, etc. These costs vary with the architecture used hence we show how to compute μ with this example: Consider a data dissemination network [6]. Assume that the network has 5 sources and that a recomputation results in a reorganization of the network. We estimate μ as follows: Let the actual cost of recomputation be nominal amounting to 0 messages. After each recomputation, the sources need to be informed about the new DABs, resulting in extra messages: Assuming that all 5 sources are notified, this results in 5 messages. Also, the recomputation results in a reorganization. Even if a reorganization takes only 1 sec, for an average message delay of 200 ms, this equates to 5 more messages, bringing the cost of a recomputation(μ) to 10 messages.

Effect of μ: For larger μ, the primary DABs will be more stringent and the secondary DABs will be larger. This implies that, as μ increases, the range for which the primary DABs are valid, increases, leading to fewer recomputations. It also implies an increase in the number of refreshes at $\mathcal{C}$.

4) Extending to General Global Portfolio Queries: A global portfolio query is written as $\sum w_i x_i x_j : B$. The formulation discussed in Section III-A can be applied to each term $w_i x_i x_j$ in this query. Ensuring we assign only one primary and secondary DAB to each data item, we arrive at the following optimization formulation.

$$\min \sum \frac{\lambda_i}{b_i} + \mu * R \quad \text{subject to}$$

$$\sum w_i[(V_{x_i} + c_{x_i})b_{x_j} + (V_{x_j} + c_{x_j})b_{x_i} + b_{x_i}b_{x_j}] \leq B$$

$$\forall x : c_x \geq b_x; \qquad \frac{\lambda_x}{c_x} \leq R$$

We can similarly derive the formulation for any PPQ. Geometric programming techniques are powerful enough to handle positive-coefficient polynomials with positive, negative or fractional exponents [12].

5) Extending to the Random Walk Data Model: In Section III-A, we discussed our formulation for the monotonic data model. It is easy to extend our formulation to other ddms. In this section, we extend it to the random walk data model.

As derived in [4], for the random walk model, the number of refreshes per unit time for data item x_i is $\frac{\lambda_i^2}{b_i^2}$. The total number of refreshes is then $\sum \frac{\lambda_i^2}{b_i^2}$. This changes the formulation to

$$\min \sum \frac{\lambda_i^2}{b_i^2} + \mu * R \qquad \text{subject to}$$

$$\sum w_i[(V_{x_i} + c_{x_i})b_{x_j} + (V_{x_j} + c_{x_j})b_{x_i} + b_{x_i}b_{x_j}] \leq B$$

$$\forall x : c_x \geq b_x; \qquad \frac{\lambda_x^2}{c_x^2} \leq R$$

B. A Polynomial Query with Positive and Negative Coefficients

Consider a general polynomial query (PQ) $Q = xy - uv : B$. In this query, we have a term with a negative coefficient, viz, uv. Existence of this term increases the complexity of the problem considerably: now the necessary and sufficient condition for Q, and hence the constraints in the formulation will have negative terms. Geometric optimization techniques require the objective function and constraints to be positive-coefficient polynomials and hence these techniques can no longer be used for Q. In fact, to the best of our knowledge, there is *no* known efficient technique which can be used to obtain an optimal solution for Q. The best we can hope for are solutions *close* to the optimal solution.

1) Key Observation and Definitions: We observe that a polynomial P can be represented as the *difference of two positive-coefficient polynomials*, P_1 and P_2, i.e., as $P = P_1 - P_2$, where P_1 is a sub-polynomial of P containing all positive coefficient terms and P_2 is the sub-polynomial containing all negative coefficient terms of P. This simple but key observation leads to the two heuristics in Section III-B.2.

Definitions: Two polynomials P_1, P_2 are said to be *independent*, if they have *no* data items in common. For example, if $P_1 = xy$ and $P_2 = uv$, then P_1 and P_2 are independent. If however, P_1 and P_2 have *at least one* data item in common then they are said to be *dependent*. For instance, if $P_1 = x^2$ and $P_2 = xy$, then P_1 and P_2 are dependent.

2) Two Heuristics for a PQ (P:B): We now propose heuristics *Half and Half* and *Different Sum*, to find DABs for a PQ.

Heuristic 1: Half and Half

Given $P : B$, we split P into two positive-coefficient sub-polynomials P_1 and P_2 such that $P = P_1 - P_2$. We also split the QAB uniformly and solve separately for $P_1 : \frac{B}{2}$ and $P_2 : \frac{B}{2}$. For any data item, the DAB for coordinator $\mathcal{C}$ is the minimum amongst the primary DABs calculated for P_1 and P_2.

It is easy to see that this heuristic ensures correctness. If the value of P changes by more than B then this implies that the value of either P_1 or P_2 had changed by at least $\frac{B}{2}$.

Dividing the bound equally between the two queries may not be optimal. Finding an optimal division for the QAB is a very complex problem in itself, depending on the sizes and degrees of each polynomial, the current values of data items, etc. As a result, we propose the next heuristic.

Heuristic 2: Different Sum

As before, we split P into P_1 and P_2. However, instead of solving for P_1 and P_2 separately, in this heuristic, we find the optimal solution for the PPQ: $P_1 + P_2 : B$.

This heuristic, though counter-intuitive, gives solutions close to the optimal for independent polynomials, under certain conditions. Before we discuss optimality issues, we first prove that *Different Sum* ensures correctness. Let P_1 and P_2 be any two *independent* positive-coefficient polynomials. Consider two queries: $Q = P_1 - P_2 : B$ and $Q' = P_1 + P_2 : B$. Then,

Claim 1: DABs which satisfy Q' also satisfy Q.

For simplicity, we prove this when $P_1 = xy$ and $P_2 = uv$. Then $Q = xy - uv : B$ and $Q' = xy + uv : B$.

Similar to Equation (2), the necessary and sufficient condition for the DABs for Q is:

$$(V_x + c_x + b_x)(V_y + c_y + b_y) - (V_u - c_u - b_u)(V_v - c_v - b_v)$$
$$-((V_x - c_x)(V_y - c_y) + (V_u - c_u)(V_v - c_v)) \leq B$$

i.e., $(V_x + c_x)b_y + (V_y + c_y)b_x + (V_u - c_u)b_v +$
$(V_v - b_v)b_u + b_x b_y - b_u b_v \leq B$ (4)

Similarly, after simplification, the condition for the DABs for Q' is: $(V_x + c_x)b_y + (V_y + c_y)b_x + (V_u + c_u)b_v +$
$(V_v + c_v)b_u + b_x b_y + b_u b_v \leq B$ (5)

We can see that the terms that occur in the two inequalities are the same, except in sign - some being negative in Equation (4). Hence the lhs of Equation (4) is atmost that of Equation (5). This implies that if *the DABs satisfy Equation (5) they also satisfy Equation (4)*. The proof for *any* two positive-coefficient polynomials P_1 and P_2 is similar and can be found in [1].

When does *Different Sum* give near-optimal solutions?

We now prove that when (i) the queries are independent and (ii) the optimal DABs for $P_1 - P_2$ are *small* with respect to the corresponding values of data items, then, the optimal DABs of $P_1 + P_2$ are *close* to the optimal DABs of $P_1 - P_2$.

Let P_1 and P_2 be two *independent* polynomials. Let b_i, c_i be the optimal primary and secondary DABs for every data item x_i in $P_1 - P_2$. Let R be the optimal number of recomputations for $P_1 - P_2$. Let α be a constant, $0 < \alpha \leq 1$ and d be the degree of $P_1 - P_2$. Let V_i be the value of data item x_i. Then we formally prove that

Claim 2: (A) If $\forall i, \alpha \geq \frac{8*d^2*(b_i+c_i)}{V_i - c_i}$ then $b_i(1-\alpha)$, $c_i(1-\alpha)$, $\frac{R}{1-\alpha}$ are feasible for $P_1 + P_2$, and (B) the total cost of this solution for $P_1 - P_2$, under the monotonic[4] data model is, at most a factor of $\frac{1}{1-\alpha}$ worse than the optimal for $P_1 - P_2$.

Due to space constraints, we are unable to present the proof for Part A here. This non-trivial proof is available in [1].

Proof for part B: Let $b_1, \cdots, b_n$ be the primary DABs, R be the number of recomputations and μ be the cost of a recomputation for an optimal solution of $P_1 - P_2$. The optimal value for objective function, for $P_1 - P_2$, under the assumption that data changes monotonically is $\sum \frac{\lambda_i}{b_i} + \mu R$. Let $b_1', \cdots, b_n'$ be the DABs and R' the number of recomputations for an optimal solution for $P_1 + P_2$. Then the optimal value for objective function, for $P_1 + P_2$ is $\sum \frac{\lambda_i}{b_i'} + \mu R'$. From part A, it follows that $b_i(1-\alpha), c_i(1-\alpha)$ and $\frac{R}{1-\alpha}$ are feasible for $P_1 + P_2$. Hence, $\sum \frac{\lambda_i}{b_i'} + \mu R' \leq \sum \frac{\lambda_i}{b_i(1-\alpha)} + \mu \frac{R}{1-\alpha} \leq \frac{1}{1-\alpha}(\sum \frac{\lambda_i}{b_i} + \mu R)$. Therefore, the value of the objective function, if we use the optimal DABs for $P_1 + P_2$, is at most a factor $\frac{1}{1-\alpha}$ worse than the optimal for $P_1 - P_2$.

Clearly, the smaller the value of α, the better is the solution. For example, consider the query $xy - uv : B$. As long as $b_i, c_i \leq \frac{V_i}{32}$, for each data item i, the solution given by $P_1 + P_2$ is close to the optimal. In practice, the DABs are usually likely to more stringent and we expect near optimal performance for any PQ with independent sub-polynomials.

IV. Multiple PQs

The formulation presented in Section III-A has a natural extension where multiple queries can be handled collectively. Due to space constraints, we briefly describe this solution here. For each data item, the primary DAB is the same across all queries. The secondary DAB, however, depends on the query and hence there are as many secondary DABs as the number of <query, data item> pairs. Also each query has a different number of recomputations, R. The objective function is the sum of the number of messages due to the primary DABs and the number of recomputations of each query (multiplied by the cost of recomputation). We call this solution *AAO* or *A*ll *A*t *O*nce. The detailed formulation is available in [1].

One of the drawbacks of this solution is that the number of variables depends on the number of queries (as the secondary DAB will be different for each data item in each query), making it difficult for existing geometric solvers to handle a large number of queries. Hence, we use the next approach.

Given multiple PQs at coordinator $\mathcal{C}$, we solve for each query independently, using the approach mentioned in Section III-A. For each data item, we then assign the minimum primary DAB across all queries, as the DAB for $\mathcal{C}$. We call this approach *EQI* or *E*ach *Q*uery *I*ndependently.

V. Experimental Evaluation

A. Experimental Methodology

An overview of the experimental methodology follows. Details are given in [1].

Physical Topology: The network topology consists of 20 sources and 1 coordinator. The total number of data items served by the sources is 100. For each data item, we use real stock traces of roughly 3 hours (10000 secs) duration, downloaded from http://finance.yahoo.com. The traces are same as those used in [6]. The PlanetLab experiments are on a topology of 4 sources and 1 coordinator for a trace duration of 4000 secs.

Delays: The communication delays are derived from a heavy tailed Pareto [13] distribution giving a node-node delay of around 100-120 ms. The computational delays at a coordinator, $\mathcal{C}$, are also derived using heavy tailed Pareto distributions. The mean delay, required to check if a query value should be sent to the user, is 4 ms and the mean delay to push the value to the user is 1 ms. We also experimented with other communication and computational delays. Similar delays are associated at a source disseminating data to $\mathcal{C}$.

Queries: PPQs are represented with a set of global portfolio queries (Query 1(a)) and PQs with set of arbitrage queries (Query 1(b)). We generate the queries using an 80-20 model. The data items are divided into two groups, group 1 consisting of 20% of the data items and group 2 consisting of the rest. 80% of the data items in each query belong to group 1. 20% belong to group 2. On an average, each query requested 12-14 data items. The value of the weights is taken uniformly from 1 to 100. We assume that each query has an initial value to start with in order to study the performance of the network in steady-state. For PPQs and PQs, the QAB is set to 1% and 2% of the initial query value, respectively.

Solver: We use a well-known solver, CVXOPT [14]. The Dual-DAB approach took about 40-70ms for a PPQ on a P4,

[4]Under the random walk model, the solution is at most $\frac{1}{(1-\alpha)^2}$ worse than the optimal.

2.66 Ghz machine. *AAO* took 600-750 ms for 10 PPQs.

Model of Data Dynamics: We experiment with two different data models: (i) data changes monotonically and (ii) the data changes as a random walk. For ease of experimentation, the rate of change, λ_i for a data item x_i, is calculated as: We estimate the current rate of change, $\lambda_i(t)$ by sampling the traces at fixed intervals (1 min), and the value of λ_i used is the average of $\lambda_i(t)$ over the complete trace. Results of others ways of calculating λ_i are also reported in [1]. Unless mentioned otherwise, the results presented are for the monotonic data model.

Cost of recomputation: The cost of recomputation, μ, approximates the overheads of a recomputation in terms of the number of messages. Since these costs vary with each query and the architecture used, (Section III-A.3), we experiment with different values of μ = 1, 2, 5, 10, and 20. For each experiment, μ is assumed to be constant. $\mu = 1$ implies that the cost of a recomputation has been approximated by 1 communication message.

Comparison with related work: The work in [5] can be adapted to calculate DABs for polynomial queries. Since the focus of the paper is on reducing the number of refreshes, we compare this work with our Optimal Refresh algorithm (Section III-A). Consider a function $f = \frac{xy}{4}$ and a threshold $B = 50$. Current values of x and y are 40 and 20 respectively. Then the DABs calculated by [5] will be 3.16625 and 2.5 respectively. The assignment given by *Optimal Refresh* is 3.87 and 2.79 (assuming $\lambda_x = \lambda_y$). As the assignment by the *Optimal Refresh* algorithm has less stringent DABs, the number of refreshes will be smaller for the *Optimal Refresh* algorithm. The reason that [5] is sub-optimal is that instead of *one* necessary and sufficient condition (Equation 1) we have to solve n sufficient conditions - one per data item. This results in more stringent DABs.

Metrics: We use four different metrics to study the performance of our algorithms.

1. *Fidelity*: This is the degree to which a query's QAB is met. It is measured as the total length of time for which the bounds are met, normalized by the total length of the observations. The loss in fidelity is simply $(100\% - \text{fidelity})$. The loss in fidelity shown is the mean loss across all queries.
2. *Number of refreshes*: This is the total number of refresh messages which arrive at a coordinator.
3. *Number of recomputations*: This is the total number of recomputations across all queries at a coordinator.
4. *Total cost*: This is the total cost in terms of the number of messages and is given by: *(Number of refreshes)* $+(\mu *$ *(Number of recomputations)*$)$.

B. Performance Results

1) Results for PPQs:

Base Results: We evaluate our Dual-DAB approach (Section III-A.4) and *Optimal Refresh* approach (Section III-A.1 on a set of global portfolio queries ($\sum a_i x_i x_j : B$. In *Optimal Refresh*, the DABs calculated for each query are optimal in *only* the number of refreshes to the coordinator. Once the DABs are calculated for each query (using either Dual-DAB or *Optimal Refresh*, the coordinator asks for each data item at the minimum DAB across all queries (*EQI* - Section IV).

Figure 5 gives the performance of Dual-DAB algorithm for different costs of recomputations, μ, on PlanetLab. From Figure 5(a), we see that the *Dual DAB approach results in a substantial reduction in the total of number of recomputations, across all values of* μ. Even when the cost of a recomputation is approximated by only 1 message ($\mu = 1$), we see that the number of recomputations reduce by more than a factor of 9 as compared to *Optimal Refresh*. For larger values of μ, the number of recomputations reduces even further.

Our approach introduces a tradeoff between the number of recomputations and the number of refreshes. Figure 5(b) shows the actual number of refreshes that arrive at a coordinator. The numbers do not include *any* overhead messages which might result due to recomputations. We can see that though there is an increase in the number of refreshes for $\mu \geq 1$ as compared to *Optimal Refresh*, this increase is small compared to the reduction in the number of recomputations.

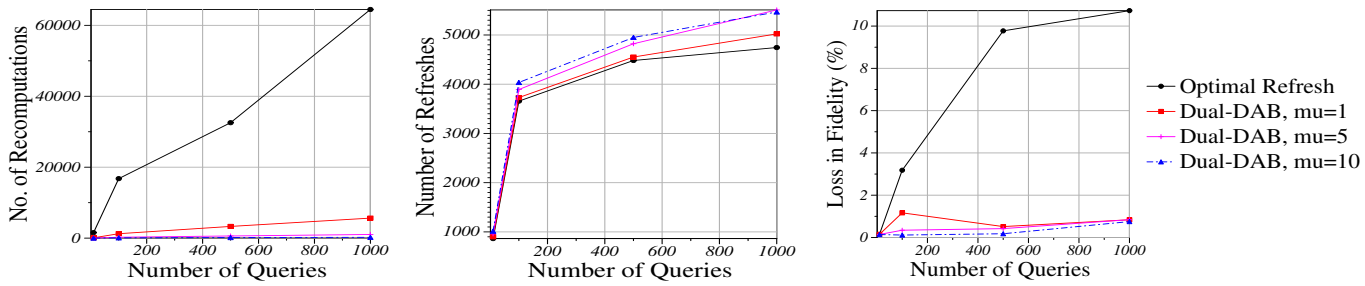
Figure 5(c) shows the trend[5] for loss in fidelity. As expected, the loss in fidelity is substantially lower for our approach. The lower the number of recomputations, the lower is the load on the coordinator and the lesser are the messages in the network, leading to better fidelity for the user.

The results shown in Figure 5 follow the same trend as results from simulation studies. The number of recomputations were higher in our PlanetLab evaluations. We believe that this is due to high variability in the communication and computational delays experienced by our PlanetLab experiments.

Effect of different data dynamics models: We next wanted to see the effect of using different data dynamics models on our approach. We ran our simulations for the random walk data model (Section III-A.5). Due to the nature of the objective function ($\sum \frac{\lambda^2}{b^2}$), the DABs for random walk were less stringent than those for monotonic data model. This resulted in more recomputations and fewer refreshes for random walk model as seen in Figure 6(a) and (b). Figure 6 also shows the effect of using rate of change on our evaluations. We calculated the DABs without using any rate of change information, i.e., $\lambda_i = 1$. These results are represented by curves labelled L1. We observed a higher number of both recomputations and refreshes, for $\lambda_i = 1$, indicating that *using information on the rate of change helps us obtain better solutions*.

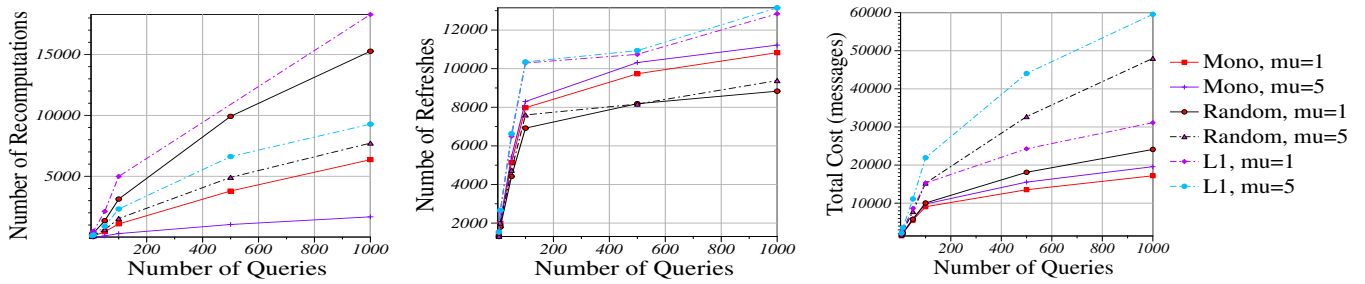
Figure 6(c) gives the total cost in terms of the number of messages for monotonic, random walk and $\lambda = 1$. As μ increases, the total cost for the random walk model and $\lambda = 1$ is higher. This increase is however much less as compared to the total cost of *Optimal Refresh*. For instance, assuming that the cost of a recomputation is 5 messages, the total cost for *Optimal Refresh* (not shown) will be > 300000 which is at least 6 times more than the total cost of L1 for the same cost of recomputation. This indicates that *the Dual DAB approach*

[5]Note that, since the experiments were run on PlanetLab, we faced different delay conditions on each run. Hence, figures here are only an indication the trends seen in the loss in fidelity across different runs.



(a) Total # of Recomputations (b) # Refreshes at the Coordinator (c) Loss in Fidelity

Fig. 5. PlanetLab Results for PPQs: Effect of different values of μ (mu)



(a) Total # of Recomputations (b) Refreshes at the Coordinator (c) # of Recomputations + μ * # Refreshes

Fig. 6. PPQs: Effect of different data dynamics models

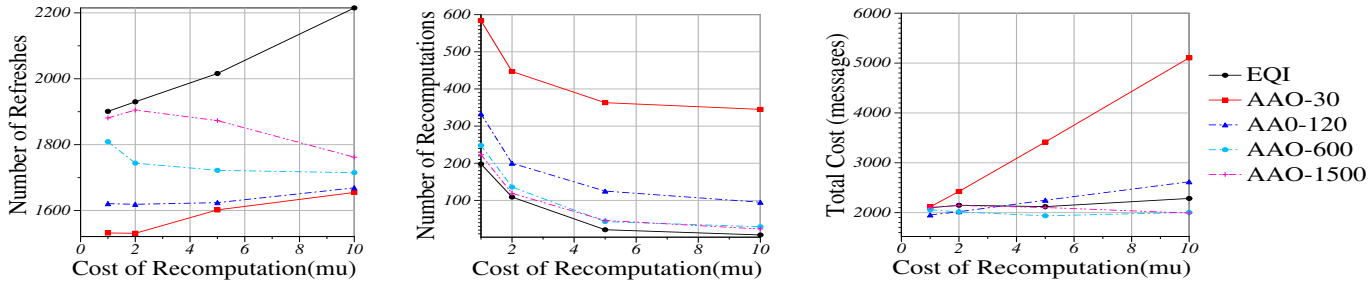
is superior to Optimal Refresh*, independent of data dynamics model and the actual objective function used.*

Effect of Varying Delays: We ran our experiments for different delay conditions. We varied the node-node delays from approximately 30 ms to 500 ms. As the delays increased, we observed a small increase in the loss in fidelity. For *Optimal Refresh*, we also observed an small increase in the number of recomputations ($< 0.5\%$). We also noticed this phenomenon when we increased the computational delays by a factor of 5.

Performance of AAO and EQI: Since we evaluate the DABs for each query independently, the resultant DABs will be sub-optimal for multiple queries. Hence, we compared the performance of *EQI* with the globally optimal algorithm, *AAO* (Section IV), for a small number of queries (10). In *AAO*, the primary DAB is the same across all queries and the secondary DAB is different for every <query, data item> pair. So, we compared *EQI* with the following approach: The DABs were calculated periodically every T seconds using *AAO*. Between two time periods, whenever a secondary DAB of a query was violated, the Dual DAB approach was used to evaluate the new DABs for that query, and the minimum DAB across all queries was the DAB for the coordinator. Curves marked in Figure 7 with *AAO-T* indicate this approach. Note that we included each *AAO* recomputation in the number of recomputations and though *AAO* is more computationally expensive, the same values of μ were used for ease of comparison.

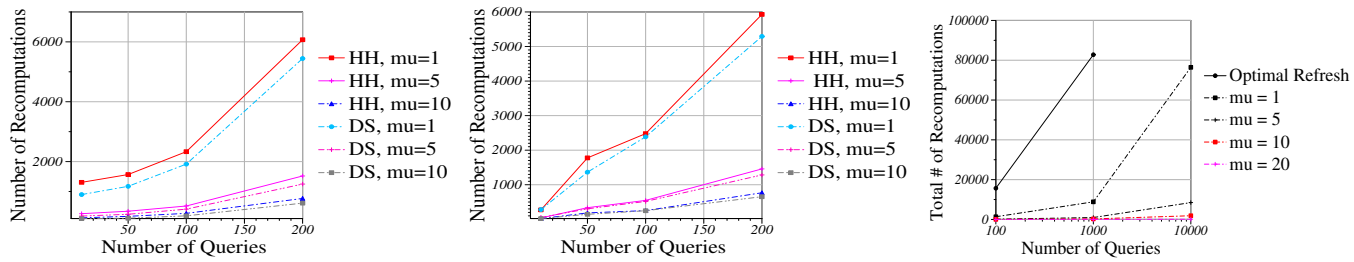
As expected, the primary DABs set by *AAO-T* were less stringent than those set by *EQI* resulting in fewer refreshes and more recomputations as compared to *EQI* as shown in Figure 7(a) and (b). We also observed that as T (period of *AAO* recomputation) increases, the effect of changing the DABs for each query independently affects performance, resulting in a higher number of refreshes for various *AAO-T* curves for the same value of μ. This effect however, is mitigated when $T > 600$ and $\mu \geq 5$. The reason for this is: For larger μ, the validity range given by secondary DABs is larger resulting in fewer per-query recomputations. This implies the bounds set by *AAO* are valid for a larger period reducing the overall number of refreshes required. Hence, we can see that for the curve marked *AAO-1500*, the number of refreshes actually *reduce* with increasing μ. Figure 7(c) shows the total cost in terms of the number of messages for *EQI* and *AAO*. We see that *AAO-30* shows a high total cost reaffirming the fact that frequent recomputations should be reduced. We also see the performance of *EQI* is comparable to *AAO* and hence can be used in practice.

2) General PQs - Arbitrage Queries: In Figures 8 (a) and (b), we compare the performance of *Half and Half* (*HH*) and *Different Sum* (*DS*) of Section III-B. Figure 8(a) shows the results for a set of independent polynomials, $P_1 - P_2 \leq B$ where $P_1 = \sum x_i y_i$ and $P_2 = \sum u_j v_j$. The results indicate that, as the number of queries increase, the number of recomputations is lower for *DS* as compared to *HH*. We found that the number of refreshes were marginally higher ($< 1\%$) in *DS*. We then compared the performance of *HH* and *DS* for dependent polynomials. Here the queries were of the same form as above, except that P_1 and P_2 had data items in common. Here again we see that *DS* performs better *HH* as shown in Figure 8(b). The experimental results indicate that *DS* not only performs well on independent polynomials but also for dependent polynomials. Hence *DS* is our choice for general polynomials.



(a) Number of Refreshes (b) Number of Recomputations (c) Total Cost

Fig. 7. *EQI* versus *AAO* for 10 PPQs



(a) PQs: Independent Queries (b) PQs: Dependent Queries (c) PPQs

Fig. 8. Additional Results: Number of Recomputations

3) Results on a Network: Figure 8(c) shows our results for PPQs on a data dissemination network consisting of 10 coordinators and 2 sources built using the work in [6]. Number of recomputations for *WSDAB* for $10,000$ queries was very high, 604735, reaffirming the fact that for large number of PQs, an approach that reduces the number of recomputations is absolutely essential.

VI. Conclusions

We propose algorithms to calculate DABs, given accuracy bounds associated with a set of polynomial queries over dynamic data. To reduce the overall cost of meeting QABs, we propose a novel idea for a PPQ which helps in substantially reducing the number of recomputations for a small increase in the number of refreshes. We also extend the use of geometric programming techniques for general polynomials. Our experimental results show that

1. It is essential to reduce the number of recomputations when a large number of polynomial queries are to be handled.
2. The reliance of our techniques on the ddm is low. As long as we make a *reasonable* assumption about the data dynamics, our approach gives a substantial reduction in the overall cost of handling polynomial queries.

We have also deployed our algorithms on PlanetLab. The results from our PlanetLab evaluations corroborate the trends observed with emulations.

As future work, we would like to extend our work to handle other kinds of non-linear queries, entity based queries [15] and also exploit possible correlation between data [16].

Acknowledgements: We thank Prof. Sohoni, Prof. Rangaraj, Prof. Sudarshan and Ravi G. for their inputs, IBM-IRL Delhi for partially funding this work and Parul Halwe for deploying this work on PlanetLab.

References

[1] S. Shah and K. Ramamritham, "Handling polynomial queries over dynamic data," http://www.cse.iitb.ac.in/~shetals/techrep.pdf, June 2007.
[2] B. Babcock and C. Olston, "Distributed top-k monitoring." in *ACM SIGMOD*, 2003.
[3] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts." in *ACM SIGMOD*, 2006.
[4] C. Olston and J.Widom, "Adaptive filters for continuous queries over distributed data streams," in *ACM SIGMOD*, 2003.
[5] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," in *ACM SIGMOD*, 2006.
[6] S. Shah, K. Ramamritham, and P. Shenoy, "Resilient and coherency preserving dissemination of dynamic data using cooperating peers," *In IEEE TKDE*, 2004.
[7] G. Cormode and M. Garofalakis, "Streaming in a connected world: Querying and tracking distributed data streams," in *VLDB Tutorial*, 2006.
[8] R. Gupta, A. Puri, and K. Ramamritham, "Executing incoherency bounded continuous queries at web data aggregators," in *WWW*, 2005.
[9] C. Olston, B. Loo, and J.Widom, "Adaptive precision setting for cached approximate values," in *ACM ACM SIGMOD*, 2001.
[10] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive stream resource management using kalman filters," in *ACM SIGMOD*, 2004.
[11] C. Olston and J.Widom, "Best effort cache synchronization with source cooperation," in *ACM SIGMOD*, June 2002.
[12] S. Boyd, S. J. Kim, L.Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," www.stanford.edu/~boyd/gp_tutorial.html.
[13] M. Raunak, P. Shenoy, P. Goyal, and K. Ramamritham, "Implications of proxy caching for provisioning networks and servers," *In ACM SiGMETRICS*, 2000.
[14] CVXOPT, "Cvxopt, convex optimization solver. http://www.ee.ucla.edu/~vandenbe/cvxopt."
[15] R. Cheng, B. Kao, S. Prabhakar, A. Kwan, and Y.-C. Tu, "Adaptive stream filters for entity-based queries with non-value tolerance," in *VLDB*, 2005.
[16] A. Silberstein, R. Braynard, and J. Yang, "Constraint chaining: on energy-efficient continuous monitoring in sensor networks," in *ACM SIGMOD*, 2006.