Fixed-Precision Approximate Continuous Aggregate Queries in Peer-to-Peer Databases

Farnoush Banaei-Kashani
Computer Science Department
University of Southern California
Los Angeles, CA 90089
Email: banaeika@usc.edu

Cyrus Shahabi
Computer Science Department
University of Southern California
Los Angeles, CA 90089
Email: shahabi@usc.edu

***Abstract*—In this paper, we propose an efficient sample-based approach to answer fixed-precision approximate continuous aggregate queries in peer-to-peer databases. First, we define practical semantics to formulate fixed-precision approximate continuous aggregate queries. Second, we propose "Digest", a two-tier system for correct and efficient query answering by sampling. At the top tier, we develop a query evaluation engine that uses the samples collected from the peer-to-peer database to continually estimate the running result of the approximate continuous aggregate query with guaranteed precision. For efficient query evaluation, we propose an extrapolation algorithm that predicts the evolution of the running result and adapts the frequency of the continual sampling occasions accordingly to avoid redundant samples. We also introduce a repeated sampling algorithm that draws on the correlation between the samples at successive sampling occasions and exploits linear regression to minimize the number of the samples derived at each occasion. At the bottom tier, we introduce a distributed sampling algorithm for random sampling (uniform and nonuniform) from peer-to-peer databases with arbitrary network topology and tuple distribution. Our sampling algorithm is based on the Metropolis Markov Chain Monte Carlo method that guarantees randomness of the sample with arbitrary small variation difference with the desired distribution, while it is comparable to optimal sampling in sampling cost/time. We evaluate the efficiency of Digest via simulation using real data.**

I. Introduction

A peer-to-peer database is a fragmented database which is distributed among the nodes of a peer-to-peer network, with both the data and the network dynamically changing. In this paper, we focus on answering continuous aggregate queries in peer-to-peer databases, where the underlying peer-to-peer network of the database is inherently unstructured (as opposed to DHT-based structured peer-to-peer networks). Continuous queries [20] allow users to obtain new results from the database without having to issue the same query repeatedly. Continuous queries are especially useful with peer-to-peer databases which inherently comprise of large amounts of frequently changing data. For example, in a weather forecast system with thousands of interconnected stations the system administrator can issue a continuous aggregate query of the form:

> *"Over next 24 hours, notify me whenever the average temperature of the area changes more than 2 ^oF."*

Or in a peer-to-peer computing system with distributed resources, users can issue the following query to determine when there is enough memory space available to schedule their tasks:

> *"Notify me whenever the total amount of available memory is more than 4GB."*

However, considering the large size and the high rate of change in peer-to-peer databases, exact continuous aggregate queries are inevitably inefficient, if not infeasible. Exact answers are rarely necessary, and even if needed, a consistent approximation can converge to the exact result with arbitrary precision. Therefore, in this paper we consider *approximate* continuous aggregate queries.

Previous approaches for approximate query answering are not applicable to peer-to-peer databases. The model based approaches [6] are parameterized, where with peer-to-peer databases parameters are unknown and variable. The histogram based [11] and the precomputed-sample based [13] data reduction approaches are not appropriate either. Although dynamically updated, with the high rate of change in peer-to-peer databases maintaining histograms and precomputed samples is intolerably costly. The large set of techniques proposed for approximate continuous aggregate query over data streams [2] naturally assume the data are collected centrally and are being received in sequence, where none of these assumptions hold for the data in peer-to-peer databases. Finally, the current on-the-fly sampling approaches, mostly developed for query size estimation [14], are limited to snapshot (or one-time) aggregate queries.

In this paper, we propose an approach for answering fixed-precision approximate continuous aggregate queries by on-the-fly sampling from peer-to-peer databases. Our query answering system, called *Digest*, evaluates approximate continuous aggregate queries by continual execution of the approximate snapshot aggregate queries, where each snapshot query is evaluated by sampling the database. The snapshot queries probe the database and accordingly the running result of the continuous query is updated. As we elaborate below, with continuous queries the main issue transcends how to execute each snapshot query, but how to execute snapshot queries continually such that while the fixed precision requirements of the continuous query are guaranteed, the query is answered efficiently by deriving minimum number of samples.

With fixed-precision approximate continuous aggregate queries, the required (or fixed) precision of the approximate result is defined by the user in terms of 1) the *resolution* of the result in capturing the changes of the actual running aggregate value (e.g., the result reflects the changes of the average temperature iff a change is more than 2 ^{o}F), and 2) the *confidence* (or accuracy) of the result *at each time* as compared to the exact aggregate value at that time. Using continual snapshot queries to answer such queries, the resolution of the result is determined by the frequency of the snapshots, and the confidence of the result depends on the number of the samples derived to approximate the result of each snapshot query. Therefore, for efficient evaluation of the continuous queries while guaranteeing the fixed precision, both the frequency of the snapshot queries and the number of the samples derived at each snapshot query must be minimized while the resolution requirement and the confidence requirement of the query are still satisfied, respectively. Digest provides solutions for both of these optimization problems.

Digest is implemented as a two-tier system, with a sampling operator at the bottom tier and a query evaluation engine at the top tier. The sampling operator implements a distributed sampling algorithm to derive arbitrary random samples from the peer-to-peer databases (with arbitrary topology and tuple distribution). The query engine uses the samples collected from the database to evaluate the snapshot queries. To minimize the frequency (or equivalently, the number) of the snapshot queries, the query engine exploits an *extrapolation algorithm* that predicts the evolution of the running aggregate value based on its previous behavior and adapts the frequency of the continual snapshot queries accordingly. With this approach, the more varying the aggregate value, the more becomes the frequency of the snapshot queries in order to maintain the resolution of the result, and when the aggregate value is steady the frequency of the snapshot queries decreases accordingly to avoid redundant sampling.

On the other hand, to minimize the number of the samples derived at each snapshot query, the query engine employs a *repeated sampling algorithm*. Repeat sampling draws on the observation that across successive snapshot queries the values of the database tuples are expected to be autocorrelated and, therefore, exploiting the regression of the value of a sampled tuple at the current query on that of the previous query can improve the accuracy of the current estimate. Repeated sampling uses regression estimation to achieve the required confidence using fewer samples as compared with the straightforward independent sampling which ignores the correlation between the snapshots.

We study both the extrapolation algorithm and the repeated sampling algorithm analytically. Besides, we demonstrate their effectiveness via simulation using real data. We show that the combined effect of our extrapolation and repeated sampling algorithms can improve the efficiency of the query evaluation up to 320% as compared with the straightforward continual query execution with fixed frequency and independent sampling.

The remainder of this paper is organized as follows. In Section II, we define the semantics of the fixed-precision approximate continuous aggregate queries. Section III presents an overview of the Digest architecture. In Section IV, we describe the query evaluation component of Digest, and follow by explaining the distributed sampling component in Section V. Section VI presents the results of our empirical study on Digest. In Section VII, we briefly discuss the remaining related work. Finally, Section VIII concludes the paper and discusses the future directions of this research.

II. Approximate Continuous Query

We model an unstructured peer-to-peer network as an undirected graph $G(V, E)$ with arbitrary topology. The set of vertices $V = \{v_1, v_2, \ldots, v_r\}$ represent the set of the network nodes, and the set of edges $E = \{e_1, e_2, \ldots, e_q\}$ represent the set of the network links, where $e_i = (v_j, v_k)$ is a link between v_j and v_k. As nodes autonomously join and leave the network, the member-set of V, and accordingly, that of E vary in time. Consequently, the set sizes r and q are also variable and unknown a priori. We assume the rate of the changes in G is relatively low as compared to the sampling time (i.e., the time required to draw a sample from the peer-to-peer database), such that the network can be assumed almost static during each sampling occasion (although it may change significantly between successive sampling occasions).

For a peer-to-peer database stored in such an unstructured peer-to-peer network, without loss of generality we assume a relational model. Suppose the database consists of a single relation $R = \{u_1, u_2, \ldots, u_N\}$. R (a multiset) is horizontally partitioned and each disjoint subset of its tuples is stored at a separate node. The number of tuples stored at the node v_i is denoted by m_{v_i}. The member-set of R also varies in time; the changes are either due to the changes of V, as nodes with new content join the network (as if inserting tuples) and existing nodes leave and remove their content (as if deleting tuples), or as the existing nodes autonomously modify their local content by insertion, update, and deletion.

With such a model for peer-to-peer databases, we define our basic query model for the continuous aggregate queries as follows. Consider the queries of the form:

SELECT $op(expression)$ FROM R

where *op* is one of the aggregate operations AVG, COUNT, or SUM, and $expression$ is an arithmetic expression involving the attributes of R. Suppose Q is an instance of such queries. Assuming a discrete-time model (i.e., the time is modelled as a discrete quantity with some fixed unit), the *snapshot* aggregate query Q_t is the query Q evaluated at time t. Correspondingly, the *continuous* aggregate query Q^c is the query Q evaluated continuously (i.e., repeated successively without intermission) for all $t \geq t_0$, where t_0 is the arrival time of Q^c. The result of Q^c is $X[t]$, a discrete-time function where for all $t \geq t_0$, $X[t]$ is the aggregate-value result of the snapshot query Q_t.

For instance, in a peer-to-peer computing system each node (a node consists of one or more computing units) keeps

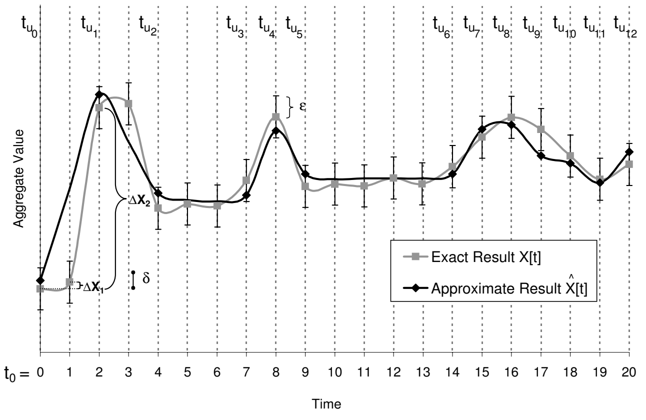


Fig. 1. Fixed-Precision Approximate Continuous Aggregate Query

a current record of its available resources by maintaining tuples of the form $u_i = \langle \text{cpu}, \text{memory}, \text{storage}, \text{bandwidth} \rangle$, one tuple for each local computing unit. Considering $R(\text{cpu}, \text{memory}, \text{storage}, \text{bandwidth})$ as a single-relation peer-to-peer database representing the resources available in such a peer-to-peer computing system, the following continuous query returns X, the total amount of the space currently available throughout the system, as a function of time:

SELECT SUM($\text{memory} + \text{storage}$) FROM R

With a *fixed-precision approximate* version of the exact continuous query Q^c defined above, the exact result $X[t]$ is approximated by an estimate $\widehat{X}[t]$ with guaranteed precision (see Figure 1). Our model for approximate queries includes three extra user-defined parameters, δ, ϵ, and p, to specify the desired precision of the estimation. With δ, user specifies the *resolution* of $\widehat{X}[t]$ in capturing the incremental changes in $X[t]$ as it evolves in time. To answer an exact query, $X[t]$ is updated (i.e., re-evaluated by snapshot query) at every time instant t, regardless of the amount of change in X since the last update at $t-1$. However, with approximate queries smaller changes below some threshold may be insignificant to the user/application and, therefore, not required to be reflected in the estimated result $\widehat{X}[t]$. The parameter δ defines this application-specific threshold. Suppose t_{u_i} is the most recent time at which the result $\widehat{X}[t]$ is updated (initially, $t_{u_0} = t_0$). For $t > t_{u_i}$, the approximate query is not required to re-update the result until $t = t_{u_{i+1}}$, where $t_{u_{i+1}}$ is the earliest time at which $\Delta X \geq \delta$ (by definition $\Delta X = |X[t_{u_{i+1}}] - X[t_{u_i}]|$). For all times t in the interval $(t_{u_i}, t_{u_{i+1}})$, $\widehat{X}[t]$ can be estimated without update/re-evaluation, e.g., by "holding" (i.e., $\widehat{X}[t] = \widehat{X}[t_{u_i}]$) or interpolation. With this semantic for approximation, the smaller changes of X during the intervals $(t_{u_i}, t_{u_{i+1}})$ for $i = 0, 1, 2, 3, \ldots$ are filtered out of the estimated result. Back to our running example mentioned above, changes on the order of several megabytes in the total space may not be noteworthy for a distributed task scheduling application and/or may be too costly to monitor. In such a case, e.g., $\delta = 1GB$ might be an effective choice to formulate an approximate query that is both useful and practical. It is important to note that in addition to allowing for optimization of the query efficiency, δ provides useful functionality to the user. Consider each update of the running query result raises an alarm for the user. With δ, user can avoid false alarms. For example, consider a scenario where a weather reporter does not want to report the average temperature unless the change in the average is more than $\delta = 5^o F$.

Next, the parameter ϵ indicates the maximum tolerable absolute error in the estimate $\widehat{X}[t]$ at *each* time t_{u_i}. The approximate query should guarantee $|\widehat{X}[t_{u_i}] - X[t_{u_i}]| \leq \epsilon$ for all i. The interval $[X[t_{u_i}] - \epsilon, X[t_{u_i}] + \epsilon]$ is termed the *confidence interval* of the estimation at time t_{u_i}, with $X[t_{u_i}] - \epsilon$ and $X[t_{u_i}] + \epsilon$ as the lower and upper *confidence limits*, respectively. The provided guarantee is probabilistic and with the parameter p user specifies the desired *confidence level* of the guarantee, i.e., the probability that the estimation $\widehat{X}[t_{u_i}]$ is actually confined within the confidence interval. The user-defined parameters ϵ and p together determine the required confidence of the estimate $\widehat{X}[t_{u_i}]$. Note that, approximate query generalizes exact query; an exact query is an approximate query with $\delta = 0$, $\epsilon = 0$, and $p = 1$.

To answer an exact continuous aggregate query, snapshot queries must be executed *continuously*, each evaluated for an *exact* result; hence, termed *continuous-exact* snapshot queries. Alternatively, an approximate continuous aggregate query can be answered by executing the more flexible and general *continual-approximate* snapshot queries. With continual-approximate queries, the less frequent the snapshot queries and the less accurate the approximation by each snapshot query, the less the cost of evaluating the continuous aggregate query, but also the less the precision (i.e., the resolution and the confidence, respectively) of the result. That allows a trade-off between the precision and the cost of obtaining the result, such that while the fixed-precision approximate query is correctly satisfied, the cost of evaluating the query can be optimized for efficiency. An extreme case of the trade-off is with continuous-exact snapshot queries to answer exact continuous aggregate queries. With continuous-exact snapshot queries, both the frequency of the snapshot queries and the accuracy of the approximation by each snapshot query are maximal, such that the estimated result of the continuous query is exact (i.e., $\widehat{X}[t] = X[t]$) while it costs the most to evaluate. Next, we present *Digest*, a query answering system that executes continual-approximate snapshot queries by sampling the database, and optimizes the frequency and accuracy of the snapshot queries to answer approximate continuous aggregate queries both correctly (with guaranteed precision) and efficiently.

III. Digest: Overview

Figure 2 depicts the two-tier architecture of *Digest*. Each node of the peer-to-peer database operates its own individual instance of Digest to answer the continuous queries received from the local user. As discussed in Section I, the query evaluation engine at the top tier exploits an extrapolation algorithm (see Section IV-A on *continual querying*) and a repeated sampling algorithm (see Section IV-B on *approximate querying*) to optimize the number of the samples derived to

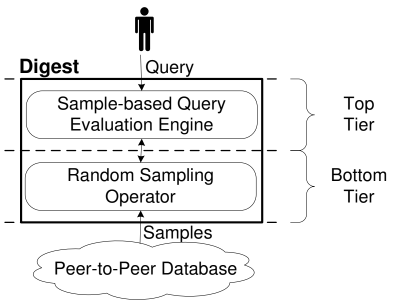


Fig. 2. Two-Tier Architecture of Digest

answer continuous aggregate queries. In addition to the query evaluation engine, Digest benefits from a sampling operator at the bottom tier that (in collaboration with other instances of Digest distributed throughout the peer-to-peer network) efficiently derives random samples from the peer-to-peer database. Here, we describe the interface of the sampling operator; the distributed sampling algorithm developed to implement the sampling operator itself is presented in Section V.

The sampling operator of Digest implements a distributed sampling algorithm to draw random samples (sample nodes and correspondingly sample tuples) from the peer-to-peer databases with arbitrary network topology and tuple distribution. The interface of the sampling operator is defined as follows. First, consider a generic weight function **w** that assigns a weight w_v to each node v of the database. For instance, $\mathbf{w_1} = \{\forall v \in V \mid w_v = 1\}$ is a uniform weight function, and $\mathbf{w_2} = \{\forall v \in V \mid w_v = m_v\}$ is a (possibly) nonuniform weight function with which each node is weighted according to the number of the tuples m_v stored at the node. We assume that with **w** the weight of each node is a function of the *local* properties of the node (such as the content-size m_v of the node, the degree of connectivity of the node, the reputation of the node, the accuracy and relevance of the tuples stored at the node, etc.), and the assigned weight is not necessarily normalized. Given such a weight function **w** as input, once invoked the sampling operator S randomly derives a sample *node* v from V such that $p_v = w_v / \sum_{u \in V} w_u$, where p_v is the probability of sampling the node v. In other words, the distribution of the sampling probability among the nodes is proportional to the distribution of the weight according to the desired (uniform or nonuniform) weight function **w**. As we show in Section V, with our distributed sampling algorithm each node only needs to know the weights of its local neighbors; hence, no need to acquire global information.

While the sampling operator S proposed in this paper can be used to draw sample *nodes* based on any generic weight function, as we discuss in Section IV-B1 with Digest we employ S to draw uniformly random sample *tuples* from R using the weight function **w**$= \{\forall v \in V \mid w_v = m_v\}$, where m_v is the number of tuples stored at each node. For this purpose, first S is invoked with the weight function **w** to derive a sample node with a sampling probability proportional to its content-size. Thereafter, the content of the sampled node is uniformly sampled to derive a sample tuple. The combination of the two samplings, i.e., the distributed node sampling via S and the local tuple sampling from the sampled node, uniquely specifies the random distribution of the sampled tuple in the *entire* R, which in this case is uniform as desired. This sampling scheme is termed *two-stage sampling* [7]. Instead, one can use *cluster sampling* with which *all* tuples of the sampled node are drawn as a batch sample. However, since with most P2P applications the contents of a node are highly correlated (high intra-cluster and low inter-cluster correlation), cluster sampling results in imprecise estimations with these applications. Therefore, with Digest we prefer two-stage sampling to cluster sampling.

With the above two-stage sampling scheme, sampling the tuples stored at the sampled node is performed locally; hence, it is standard and inexpensive. The sampling operator S, which implements the more complicated and costly *distributed node sampling*, ensures suboptimal performance (comparable to that of the optimal sampling) in terms of communication cost and sampling time, while guaranteeing randomness of the derived sample node with arbitrary small error (i.e., variation difference) as compared to the desired sampling probability distribution (see Section V for details).

IV. Sample-based Query Evaluation

As mentioned in Section I, to evaluate an approximate continuous query one needs to provide solution for two sub-problems: 1) *continual querying*, i.e., to determine when to execute the next snapshot query, and 2) *approximate querying*, i.e., to minimize the number of samples required to answer each snapshot query. An integrated technique that incorporates solutions for the two sub-problems simultaneously is ideal but complicated. With Digest, we address the two sub-problems separately. First, at each occasion Digest uses an extrapolation algorithm to decide when to execute the next snapshot query (Section IV-A). Thereafter, Digest uses a repeated sampling algorithm to evaluate the snapshot query with the minimum number of samples required to satisfy the confidence requirements of the query (Section IV-B). This process is repeated while the continuous query is running. Below, we describe these algorithms.

A. Continual Querying

Suppose the most recent snapshot query is executed at time $t_{u_i} = k$ (see Figure 3). For continual querying, we should predict the next update time $t_{u_{i+1}}$ such that $|X[t_{u_{i+1}}] - X[t_{u_i}]| \geq \delta$. To predict $t_{u_{i+1}}$, in brief our approach is to fit a curve to the previously observed values of X to predict its values in the near future with guaranteed error bounds. If the ratio of the change/variation of $X[t]$ in time is unbounded, the future values of X are unpredictable and, therefore, inevitably continual querying reduces to continuous querying to ensure the required resolution. However, the aggregate value $X[t]$ is expected to be a smooth function of time with considerable autocorrelation in short time intervals that bounds its variation (e.g., consider the variation of the total amount of the available space in a peer-to-peer computing system). Hereafter, by convention we model such a smooth function $X[t]$ with bounded

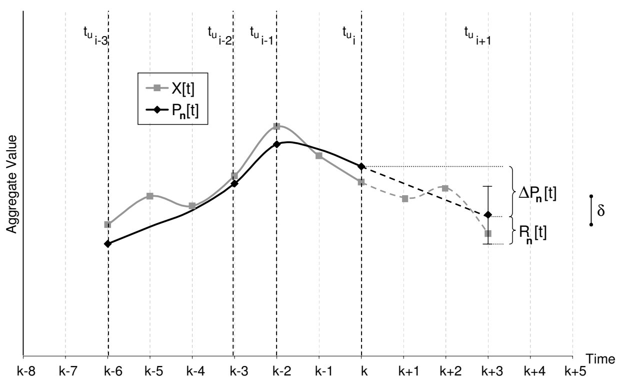


Fig. 3. Computing $t_{u_{i+1}}$ by Polynomial Extrapolation: at $t = t_{u_{i+1}}$, we have $|\Delta P_n[t_{u_{i+1}}]| + |R_n[t_{u_{i+1}}]| > \delta$

variation as an *analytic* function of time, i.e., we assume $X[t]$ possesses derivatives of all orders and agrees with its Taylor series in the neighborhood of every point.

Assuming that $X[t]$ is an analytic function, our continual querying algorithm predicts the evolution of the analytic function $X[t]$ by polynomial extrapolation using the Taylor series expansion. Based on the Taylor's theorem, at the neighborhood of t_{u_i}, $X[t]$ can be approximated by a degree-n Taylor polynomial $P_n[t]$:

$$P_n[t] = X[t_{u_i}] + (t - t_{u_i})X'[t_{u_i}] + \frac{(t - t_{u_i})^2}{2!}X''[t_{u_i}] + ... + \frac{(t - t_{u_i})^n}{n!}X^{(n)}[t_{u_i}] \quad (1)$$

The upper bound error for this approximation of $X[t]$ with the Taylor polynomial is the Lagrange remainder $R_n[t]$, such that $|X[t] - P_n[t]| < |R_n[t]|$, where:

$$R_n[t] = \frac{(t - t_{u_i})^{n+1}}{(n+1)!}X^{(n+1)}[c_t] \quad (2)$$

with $c_t \in [t_{u_i}, t]$ a constant minimizing $X^{(n+1)}[t]$ in this interval.

To find $t_{u_{i+1}}$, first $P_n[t]$ is computed by fitting a degree-n polynomial to $n+1$ previous values of $X[t]$ at $t = (k-n)$, $t = (k-n+1)$, ..., and $t = k$. We use the well-known Levenberg-Marquardt Method (based on non-linear least squares fitting via trust regions) for fitting. This method is known for robust estimation of the Taylor polynomial. Note that the exact values of $X[t]$ are unknown unless the snapshot queries are exact. Instead of the exact values, assuming sufficiently accurate approximate snapshot results, we compute $P_n[t]$ using $n+1$ previous values of $\widehat{X}[t]$ at $t = t_{u_{i-n}}$, $t = t_{u_{i-n+1}}$, ..., and $t = t_{u_i}$ ($n = 3$ in Figure 3). Next, having $P_n[t]$ as an approximation with bounded error for $X[t]$, $t_{u_{i+1}}$ is derived by extrapolation as the minimum t satisfying:

$$|P_n[t] - P_n[t_{u_i}]| + |R_n[t]| > \delta \quad (3)$$

Note that the upper bound error $|R_n[t]|$ of the polynomial approximation is a decreasing function of n. Therefore, the higher the degree of the polynomial approximation, the tighter is the error bound and, thus, the predicted update time $t_{u_{i+1}}$ is less conservative, which makes the continual querying more efficient. Also, while $t < t_{u_n}$, a degree-n polynomial approximation is not applicable. During the bootstrapping period, i.e., the interval $[t_{u_0}, t_{u_n})$, our continual querying algorithm implements continuous querying instead of continual querying.

B. Approximate Querying

1) Independent Sampling: To provide the background for explaining the repeated sampling algorithm, in this section we briefly review the query evaluation process for an approximate AVG query based on the classical independent sampling algorithm.

With the independent sampling algorithm, to answer an AVG snapshot query Q_t

SELECT AVG($expression$) FROM R

n sample tuples u_1, u_2, ..., u_n are derived from R, uniformly random and with replacement. The time interval (beginning at t) during which the database is probed to draw samples for evaluating Q_t is called the *sampling occasion* for Q_t. During the sampling occasion, each sample is drawn by first calling the sampling operator S with the weight function $\mathbf{w} = \{\forall v \in V \mid w_v = m_v\}$ to derive a sample node with a sampling probability proportional to its content-size. Next, the content of the sampled node is uniformly sampled to derive a sample tuple. Suppose the value of the $expression$ when applied to the sample tuple u_i is denoted by y_i. Based on the derived samples, the result ($X =$) $\overline{Y} = 1/N \sum_{i=1}^{N} y_i$ of the AVG query is estimated by the unbiased and consistent estimator:

$$\widehat{\overline{Y}} = 1/n \sum_{i=1}^{n} y_i \quad (4)$$

The number of the samples n is computed such that with probability p the estimate $\widehat{\overline{Y}}$ is within the confidence interval $[\overline{Y} - \epsilon, \overline{Y} + \epsilon]$. One can use the standard central limit theorem to compute n. Let $\sigma^2 = 1/N \sum_{i=1}^{N}(y_i - \overline{Y})^2$ be the true variance of y_i in R, and $\widehat{\sigma}^2 = 1/n \sum_{i=1}^{n}(y_i - \widehat{\overline{Y}})^2$ the estimated variance. For sufficiently large n, it follows from the central limit theorem that the random variable $\widehat{\overline{Y}}$ has a normal distribution with mean $\overline{Y}$ and variance σ^2/n, or equivalently, the standardized random variable $\sqrt{n}(\widehat{\overline{Y}} - \overline{Y})/\sigma$ has a normal distribution with mean 0 and variance 1. Therefore:

$$\Pr\{|\widehat{\overline{Y}} - \overline{Y}| \le \epsilon\} = \Pr\left\{\left|\frac{\sqrt{n}(\widehat{\overline{Y}} - \overline{Y})}{\sigma}\right| \le \frac{\epsilon\sqrt{n}}{\sigma}\right\} \approx 2\left(\Phi(\frac{\epsilon\sqrt{n}}{\widehat{\sigma}}) - 1/2\right) \quad (5)$$

where Φ is the standard cumulative normal distribution function. Let l_p be the $(p+1)/2$ quantile of this distribution (i.e., $\Phi(l_p) = (p+1)/2$). To derive n, we set the rightmost term in Equation 5 equal to p and solve the equation for n:

$$n = \left(\frac{\widehat{\sigma} l_p}{\epsilon}\right)^2 \quad (6)$$

2) Repeated Sampling: With independent sampling, each snapshot query is answered independently, disregarding the results and the samples derived for the previous queries. However, across successive queries the values of the database tuples are expected to be autocorrelated and, therefore, exploiting the regression of the value of a sampled tuple at the current sampling occasion on that at the previous occasion can improve the accuracy of the current estimate. Alternatively, by regression estimation one can achieve the same accuracy/confidence using fewer samples at each sampling occasion; hence, more efficient query evaluation. *Repeated sampling* relies on this observation to improve the efficiency of independent sampling while still satisfying the confidence requirement of the query. Below, we explain the repeated sampling algorithm in details for evaluation of the AVG queries. Other types of aggregate queries are evaluated similarly. We begin with regression estimation for the evaluation of the 2^{nd} snapshot query (i.e., the special bootstrapping snapshot query) of a continuous aggregate query. Later, we extend the analysis for evaluation of the general k^{th} snapshot query.

a) Evaluating 2^{nd} Snapshot Query: Suppose the sample-set is of size n in both the first and the second sampling occasions of a continuous AVG query. Let u_{ik} denote the state of the sampled tuple u_i at occasion k (i.e., the current attribute-values of the tuple u_i at occasion k), and let y_{ik} denote the corresponding value of u_{ik} when evaluated by the $expression$ in the AVG query at occasion k. At the first occasion, there is no prior information to utilize. Therefore, all n samples u_1, u_2, ..., u_n are new samples derived from the database and the result of the first snapshot query is simply estimated based on the independent sampling algorithm using u_{11}, u_{21}, ..., u_{n1} as the samples. At the second occasion, each sampled tuple u_i is either replaced by a new sample $u_{i'}$ from the database (with the current state $u_{i'2}$), or retained and re-evaluated to its current state u_{i2}. Thus, the sample-set at the second occasion consists of n samples u_1, ..., u_g, $u_{g'}$, ..., $u_{n'}$, where g is the number of retained samples. A new sample $u_{i'}$ must be derived using the sampling operator S and incurs communication overhead to locate, whereas a retained sample u_i is already located and is only retrieved to be re-evaluated (to refresh u_{i1} to u_{i2}) after a possible state update in between the two sampling occasions. If a sample tuple is deleted or the node storing the tuple leaves the network, the tuple is always replaced. Repeated sampling uses the current state of the new samples, i.e., $u_{g'2}, ..., u_{n'2}$, for regular estimation as in the first occasion, while utilizing the current state of the retained samples, i.e., $u_{12}, ..., u_{g2}$, for regression estimation (where y_{i2} regresses on y_{i1} as the auxiliary regression variate). The final estimate of the repeated sampling algorithm for the result of the second snapshot query is a combined estimate, a weighted sum of the regular estimate and the regression estimate from the new portion and the retained portion of the sample-set, respectively.

With the above estimation scheme, an optimal sample replacement policy is required to determine the proportion of the new and retained portions of the sample-set such that the combined estimate of the result is optimal (i.e., the most accurate estimate with minimum variance). With two extreme cases of the replacement policy, the samples are either all replaced or all retained. As we show below, none of these policies are optimal.

With repeated sampling we establish the optimal replacement policy as follows. Suppose among n samples at the k^{th} sampling occasion (here $k = 2$), g samples are retained from the previous occasion and the rest f samples ('f' for fresh) are new samples. Let $\overline{y}_{kf}$, $\overline{y}_{kg}$, and $\overline{y}_k$ denote the average value of the samples in the new portion, the retained portion, and the entire sample-set (both portions together) at the k^{th} sampling occasion, respectively. Correspondingly, the values of the regular estimate, regression estimate, and the combined estimate for the result $\overline{Y}_k$ of the k^{th} snapshot query are denoted by $\widehat{\overline{Y}}_{kf}$, $\widehat{\overline{Y}}_{kg}$, and $\widehat{\overline{Y}}_k$, respectively.

Estimator	Variance
$\widehat{\overline{Y}}_{2f} = \overline{y}_{2f}$	$\frac{\widehat{\sigma}^2}{f} \qquad = \frac{1}{W_{2f}}$
$\widehat{\overline{Y}}_{2g} = \overline{y}_{2g} + b(\overline{y}_1 - \overline{y}_{1g})$	$\frac{\widehat{\sigma}^2(1-\widehat{\rho}^2)}{g} + \widehat{\rho}^2\frac{\widehat{\sigma}^2}{n} = \frac{1}{W_{2g}}$

TABLE I
REGULAR AND REGRESSION ESTIMATORS AT 2^{nd} OCCASION

The estimators $\widehat{\overline{Y}}_{2f}$ and $\widehat{\overline{Y}}_{2g}$ and their corresponding variances are defined in Table I. $\widehat{\sigma}_2^2 = 1/n\sum_{i=1}^{n}(y_{i2} - \overline{y}_2)^2$ is an estimate of the true variance $\sigma_2^2 = 1/N\sum_{i=1}^{N}(y_{i2} - \overline{Y}_2)^2$ of y_i in R at the second occasion, and we have $\widehat{\sigma}_2 \approx \widehat{\sigma}$ $(= \widehat{\sigma}_1)$. The estimator $\widehat{\overline{Y}}_{2f}$ is simply the average value of the samples $\overline{y}_{2f}$ in the new portion of the sample-set. For the regression estimate $\widehat{\overline{Y}}_{2g}$, we consider a linear regression with the regression coefficient $b = \frac{\widehat{\sigma}_{1,2}}{\widehat{\sigma}_1^2}$. The parameter b is an estimate of the true regression coefficient $B = \frac{\sigma_{1,2}}{\sigma_1^2}$, where $\sigma_{1,2}$ is the covariance of y_{i1} and y_{i2} in the entire R, and $\widehat{\sigma}_{1,2}$ is its estimate in the sample-set. Similarly, $\widehat{\rho} = \frac{\widehat{\sigma}_{1,2}}{\widehat{\sigma}_1\widehat{\sigma}_2}$ is an estimate of the true correlation coefficient $\rho = \frac{\sigma_{1,2}}{\sigma_1\sigma_2}$.

The combined estimate $\widehat{\overline{Y}}_2$ is derived as the sum of the two independent estimates $\widehat{\overline{Y}}_{2f}$ and $\widehat{\overline{Y}}_{2g}$ weighted inversely by their variance:

$$\widehat{\overline{Y}}_2 = \alpha\widehat{\overline{Y}}_{2f} + (1-\alpha)\widehat{\overline{Y}}_{2g} \tag{7}$$

where $\alpha = \frac{W_{2f}}{W_{2f}+W_{2g}}$. By the least squares theory the variance of $\widehat{\overline{Y}}_2$ is:

$$\text{var}(\widehat{\overline{Y}}_2) = \frac{1}{W_{2f} + W_{2g}}$$

which from Table I works out as:

$$\text{var}(\widehat{\overline{Y}}_2) = \frac{\widehat{\sigma}^2(n - g\widehat{\rho}^2)}{n^2 - g^2\widehat{\rho}^2} \tag{8}$$

The minimum variance $\text{var}_{min}(\widehat{\overline{Y}}_2)$ is calculated from Equation 8 by derivation with respect to g. This gives the optimal partitioning of the sample-set as:

$$g_{opt} = \frac{n}{1 + \sqrt{1 - \widehat{\rho}^2}} \qquad f_{opt} = \frac{n\sqrt{1 - \widehat{\rho}^2}}{1 + \sqrt{1 - \widehat{\rho}^2}} \tag{9}$$

and with optimal partitioning, the minimum variance is derived as:

$$\text{var}_{min}(\widehat{\overline{Y}}_2) = \frac{\widehat{\sigma}^2}{2n}(1 + \sqrt{1 - \widehat{\rho}^2}) \tag{10}$$

Note that if $g = 0$ (all samples replaced, like independent sampling) or $f = 0$ (all samples retained), the estimate variance (see Equation 8) is equal to that of the independent sampling, i.e., $\widehat{\sigma}^2/n$ $(\approx \sigma^2/n)$. However, with optimal partitioning $(g = g_{opt})$ repeated sampling improves the variance with the ratio:

$$\beta = \frac{(\widehat{\sigma}^2/n) - \mathrm{var}_{min}(\widehat{\overline{Y}}_2)}{\mathrm{var}_{min}(\widehat{\overline{Y}}_2)} = \frac{1-\sqrt{1-\widehat{\rho}^2}}{1+\sqrt{1-\widehat{\rho}^2}} \tag{11}$$

Based on Equation 11, depending on the correlation $|\widehat{\rho}|$ (≤ 1) between values of the tuples at successive occasions, repeated sampling can improve the accuracy of the estimation over that of the independent sampling by up to 100% (with maximum correlation $|\widehat{\rho}| = 1$). Also, as the correlation $|\widehat{\rho}|$ increases, with optimal partitioning a larger portion of the samples are retained because regression estimation is more effective. However, unless the correlation is maximum, repeated sampling replaces a considerable portion of the samples to account for the tuple insertions, deletions, and pathological updates.

b) Evaluating k^{th} Snapshot Query: Query evaluation at the k^{th} occasion is a generalization of that of the second occasion. Details are included in the extended version of this paper.

V. Random Sampling Operator

Given a weight function $\mathbf{w} = \{\forall v \in V \mid w_v\}$ as input, the sampling operator S should randomly derive a sample node v from V with the sampling probability distribution $p_v = w_v / \sum_{u \in V} w_u$. We implement our sampling operator based on a distributed sampling algorithm inspired by the Markov Chain Monte Carlo (MCMC) methods for sampling from a desired probability distribution. To sample from a distribution, first an MCMC method constructs a Markov chain that has the desired distribution as its stationary distribution. With such a Markov chain, starting a traversal of the chain from any initial state, under certain conditions the distribution of the covered states of the chain converges to the stationary distribution of the chain after a sufficiently large number of steps. Once converged, the current state is returned as a sample from the desired distribution.

With our distributed sampling algorithm, we consider a peer-to-peer network as a Markov chain, with nodes as the states of the chain, and links as the transitions between the states. Also, our algorithm uses random-walking sampling agents that are forwarded from node to node to emulate the state transition process. To sample the peer-to-peer database, the sampling operator at a node initiates a random walk (a sampling agent). If the forwarding probabilities of the random walk (corresponding to the transition probabilities of the constructed Markov chain) are properly assigned such that the stationary distribution of the walk is equivalent to the desired sampling distribution p_v, after a sufficiently large number of steps the distribution of the nodes covered by the random walk converges to p_v and the current node is returned to the originating node as the sampled node. In the rest of this section, first we describe how our distributed sampling algorithm employs the *Metropolis* Markov Chain construction algorithm to assign the forwarding probabilities of the random walk for the desired stationary distribution p_v. Second, we present our result that determines the number of steps required to converge to the desired distribution with arbitrary difference.

A. Forwarding Probabilities

Let the undirected connected graph $G(V, E)$ model a peer-to-peer network with arbitrary topology. A random walk that starts at a node v_0 (the originating node), arrives at a node v_t at time t and with certain forwarding probability moves to a neighbor node v_{t+1} at time $t+1$. Suppose π_t denotes the distribution of the node v_t such that $\pi_t(i) = \Pr(v_t = i)$, for all $i \in V$. Let $P = (P_{ij})$, $i, j \in V$, denote the forwarding matrix of the random walk, where P_{ij} is the probability that the random walk moves from node i to node j. $P_{ij} = 0$ if i and j are not adjacent. By definition, we have $\pi_{t+1} = \pi_t P = \pi_0 P^{t+1}$, where π_0 is the distribution of the originating node v_0. The following existence result is classic [17]:

Theorem 1: If P is irreducible (i.e., any two nodes are mutually reachable by random walk) and P is aperiodic (which will be if G is non-bipartite), then π_t converges to the unique stationary distribution π such that $\pi P = \pi$ independent of the initial distribution π_0.

The Metropolis algorithm [16] is designed to assign the forwarding probabilities P_{ij} such that the stationary distribution π corresponds to a desired distribution (uniform or nonuniform) such as p_v:

Theorem 2: Consider the graph $G(V, E)$ and let d_i denote the degree of the node i in G. For each neighbor j of i, the forwarding probability P_{ij}, $i \neq j$, is defined as follows:

$$P_{ij} = \begin{cases} \frac{1}{2}(\frac{1}{d_i}) & \text{if } \frac{p_i}{d_i} \leq \frac{p_j}{d_j} \\ \frac{1}{2}(\frac{1}{d_j})(\frac{p_j}{p_i}) & \text{if } \frac{p_i}{d_i} > \frac{p_j}{d_j} \end{cases} \tag{12}$$

and $P_{ii} = 1 - \sum_{j \in neighbors(i)} P_{ij}$. Then, with the forwarding matrix P, p_v is the unique stationary distribution of the random walk on G.

The proof for Theorem 2 is complicated [16]. Intuitively, the Metropolis algorithm modifies a regular random walk with uniform forwarding probability to a biased random walk with forwarding probabilities that depend on the desired sampling probability of the neighbor nodes. The Metropolis forwarding matrix P is irreducible [9]. Also, the laziness factor $1/2$ adds a virtual self-loop to each node of the G, which makes G non-bipartite and P aperiodic. Thus, convergence of the Metropolis follows from Theorem 1.

Note that using the Metropolis algorithm, S implements a fully distributed sampling process with which it does not require to know/compute the global normalization factor $\sum_{u \in V} w_u$ (to calculate $p_v = w_v / \sum_{u \in V} w_u$) in order to assign the forwarding probabilities P_{ij}. Each node i determines its local forwarding probabilities P_{ij} (j is a neighbor of i) individually and only based on the local information. According to Equation 12, to determine P_{ij}, i only needs to know the *ratio* w_j/w_i ($= p_j/p_i$), which it computes by obtaining the weight w_j from its neighbor j.

B. Convergence Time

To determine how rapidly π_t converges to p_v, first consider the following definitions and the subsequent classic result (Theorem 3):

Definition 1: The total-variance difference (or simply variance difference) between two distributions π_t and p_v is defined as $\|\pi_t, p_v\| = \frac{1}{2}\max_{v_0}\sum_i |\pi_t(i) - p_i|$.
The variance difference is a measure to quantify the total difference between two probability distributions, and we have $0 \leq \|\pi_t, p_v\| \leq 1$.

Definition 2: For $\gamma > 0$, the mixing time is defined as $\tau(\gamma) = \min\{t | \forall t' \geq t, \|\pi_{t'}, p_v\| \leq \gamma\}$.
The mixing time $\tau(\gamma)$ is the time (i.e., number of time steps) it takes for π_t to converge to p_v to within the difference γ, such that $\|\pi_t, p_v\| \leq \gamma$. The following theorem bounds the mixing time when the random walk is on a graph G with arbitrary topology [10]:

Theorem 3: Let $p_{v_{min}} = \min_i p_i$, then $\tau(\gamma) \leq \theta_P^{-1}\log((p_{v_{min}}\gamma)^{-1})$, where θ_P is the eigengap of the forwarding matrix P.
The eigengap of P is defined as $\theta_P = 1 - |\lambda_2|$, where λ_2 is the second eigenvalue of the matrix. Thus, the larger the eigengap, the more rapidly the random walk converges to the desired distribution.

However, computing the exact eigengap of P for peer-to-peer networks with large size and dynamic topology is difficult, if not infeasible. Instead, one can utilize the geometric bounding approach [10] to derive a bound for the eigengap. Considering power-law graph as a generic and realistic model for the topology of the peer-to-peer networks [19], we use the geometric bounding approach to derive the mixing time (or convergence time) of the random walk on a graph G with power-law topology:

Theorem 4: Suppose G is a random graph with the node degree distribution $p_k \propto k^{-\alpha}$, where $2 < \alpha < 3$. Then, $\tau(\gamma)$ is of order $O(N^{-\alpha}\log\gamma^{-1}\log^4 N)$, where N is size of the network.

Proof: Omitted. ∎

Considering this result, the mixing time, i.e., the sampling cost/time of our sampling operator S, is suboptimal, comparable to the optimal mixing time which is achieved by a centralized algorithm [5].

VI. Experiments

We conducted a set of experiments via simulation using real data to study the performance of Digest. We implemented a multi-threaded simulator in C++ and used two Enterprise 250 Sun servers to perform these experiments.

A. Experimental Methodology

To study Digest empirically, we used two sets of real data: the TEMPERATURE dataset and the MEMORY dataset. The TEMPERATURE data are collected from a set of interconnected weather forecast stations from JPL/NASA, and the MEMORY data are collected from the nodes of the SETI@HOME peer-to-peer computing system. Each weather station/node collects recent readings from one or more temperature sensor units, and each node at SETI@HOME may include one or more computing units (multiple units in the case of the nodes that are clusters). Each dataset consists of timestamped tuples with a single attribute (temperature and available memory space, respectively), where each tuple records the current value of the attribute at a particular time at a particular unit (sensor unit and computing unit, respectively). Whenever the value of the attribute is modified (i.e., autonomously updated, or inserted/deleted due to the node or unit join/leave), a new tuple is appended to the dataset to record the modification. Tuples are collected from a large set of nodes over a specific duration. The nodes and units of the weather forecast network are almost stable whereas those of SETI@HOME join and leave the network more frequently. Table II lists the dataset parameters.

We simulated the weather forecast network and the peer-to-peer computing network with two networks of the correspondingly same size, with mesh and power-law topologies, respectively. Each node of the network represents a node of the real network and emulates the updates of the local attributes according to the most recent recorded values in the corresponding dataset. The nodes of the two networks are Digest enabled. To perform the experiments, we considered a continuous `AVG` query of the form:

`SELECT AVG(`a`) FROM` R

where a is the recorded attribute. The duration of the continuous query is equal to the duration of recording for each dataset (see Table II). We picked random nodes from the networks to issue the queries and combined the results of the queries to derive a statistically reliable estimation of the result, wherever applicable.

As an aside, we should mention that we used the sampling operator S in batch mode, i.e., to derive n samples we invoke S for n times simultaneously, which initiates n random walks with overlapping convergence time, to expedite the experiments. Also, once converged for the first time, to derive successive samples we continue the random walk from where it stops. In this case, the time to re-converge is reduced from the mixing time to the *reset* time, which is much shorter than the mixing time of the random walk.

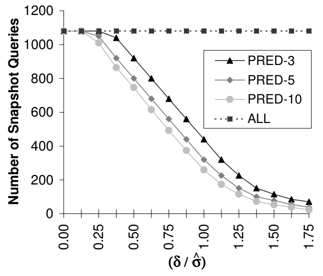
B. Experimental Results

We studied the efficiency of Digest by considering the improvement due to the extrapolation algorithm and the repeated sampling algorithm, individually and combined.

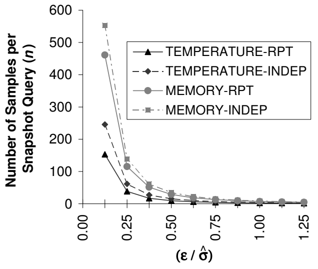
1) Effect of the Extrapolation Algorithm: Figure 4-a illustrates the results of our experiment with the extrapolation algorithm. For this experiment, we report our results for the TEMPERATURE dataset, focusing on different variations of

	TEMPERATURE	MEMORY
Number of Tuples	8640000	95445
Number of Units	8000	1000
Number of Nodes	530	820
Duration of Recording	18 months	1 hour
Frequency of Updates	Twice per day	Continuous
$\widehat{\rho}$	0.89	0.68
$\widehat{\sigma}$	8	10

TABLE II
Parameters of the Datasets

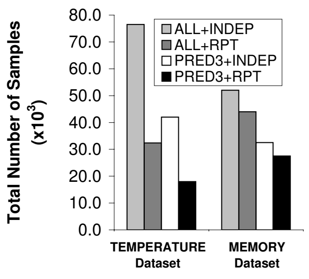


a. Extrapolation Algorithm

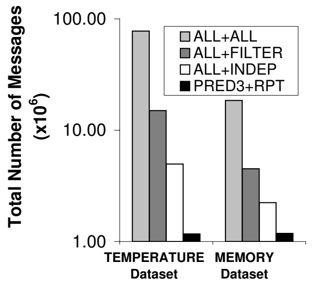


b. Repeated Sampling Algorithm

Fig. 4. Effect of the Algorithms



a. Total Number of Samples



b. Total Communication Cost

Fig. 5. Efficiency of Digest

the extrapolation algorithm (we observe similar trend with the MEMORY dataset). In Figure 4-a, PRED-k denotes the extrapolation algorithm when k previous values are used for prediction. The PRED-k algorithms are compared with the naive continuous querying algorithm (ALL), which executes snapshot queries at all time steps. The time step (i.e., the discrete-time unit) for executing snapshot queries is 12 hours, equal to the data update period with the TEMPERATURE dataset. With a fixed confidence (for the reported result, $\epsilon = 2$ and $p = 0.95$), we vary the required resolution δ of the query (in the figure, it is normalized to the variance $\widehat{\sigma}$), and observe the number of the snapshot queries executed to maintain the resolution using different algorithms.

As depicted in Figure 4-a, all of the extrapolation algorithms behave similarly. With small δ (relative to $\widehat{\sigma}$), there are not many sampling occasions that an extrapolation algorithm can skip and, therefore, the performance is similar to ALL. However, with larger resolution thresholds, the extrapolation algorithms significantly outperform the naive continuous querying algorithm by gracefully eliminating the redundant snapshot queries according to the required resolution. For example, with $\delta = 8$ (i.e., $\delta/\widehat{\sigma} = 1$), the number of the snapshot queries executed to answer the query are up to 75% reduced.

2) Effect of the Repeated Sampling Algorithm: Figure 4-b shows the results of our experiment with the repeated sampling algorithm (RPT) as compared with the independent sampling algorithm (INDEP). For this experiment, we used both of the datasets. Assuming a fixed resolution ($\delta/\widehat{\sigma} = 1$, where $\widehat{\sigma}$ is known for each dataset) and fixed confidence level ($p = 0.95$), we vary the required confidence interval ϵ of the query, and observe the (average) number of the samples required *per snapshot query* to satisfy the confidence requirement of the query using each algorithm. Note that here, for RPT we report the *total* number of the samples required per snapshot query, including both the retained and the fresh samples (for INDEP, all samples are fresh). This is to isolate and show the effect of considering the correlation in reducing the total number of the required samples with RPT. In Section VI-B3, we investigate another advantage of RPT due to the retained samples. Although the retained samples must be re-evaluated, they incur negligible communication cost to derive; therefore, in fact with RPT only the fresh samples actually cost to derive from the database (refer to Section IV-B2).

As depicted in Figure 4-b, the behavior of INDEP and RPT follow our analytical results, and RPT consistently outperforms INDEP with both datasets. From the experiments, we measure the average improvement factor $I = n_{indep}/n_{rpt}$ (where n_{indep} and n_{rpt} are the total number of the samples per snapshot query for INDEP and RPT) as 1.63 and 1.21 for the TEMPERATURE and the MEMORY datasets, respectively, which correspondingly translate to 39% and 18% less samples with RPT. As suggested by the results, the benefit of the repeated sampling algorithm is more when applied to the TEMPERATURE dataset; this is expected because of the higher correlation (as indicated by the correlation coefficient $\widehat{\rho}$) as well as less churn at the weather forecast network.

3) Overall Efficiency of Digest: To evaluate the overall efficiency of Digest due to the combined effect of the extrapolation algorithm and the repeated sampling algorithm, we measured the total number of samples required to answer a continuous query (for the reported result, $\delta/\widehat{\sigma} = 1$, $\epsilon/\widehat{\sigma} = 0.25$, and $p = 0.95$) using four different combinations of the algorithms: (ALL + INDEP), (ALL + RPT), (PRED3 + INDEP), and (PRED3 + RPT). We performed this experiment with both of the datasets. As shown in Figure 5-a, with the TEMPERATURE dataset, Digest (i.e., PRED3 + RPT) outperforms a naive solution (i.e., ALL + INDEP) up to 320%. Similar results are obtained for other continuous queries with a full spectrum of different precision parameters; these results reveal a cost-precision trade-off that conform with those represented by the results in Figures 4-a and 4-b (we omit these results due to lack of space).

Thus far, we used the total number of the samples derived to answer the query as the measure of efficiency for the algorithms. Assuming a fixed (in average) communication cost for deriving each sample, this can be translated to the total *communication cost* (i.e., the total number of messages sent from node to node) for answering the query. To factor in and evaluate the communication cost for deriving each sample using our random sampling algorithm, we considered the performance of the Digest (PRED3 + RPT) for the same query ($\delta/\widehat{\sigma} = 1$, $\epsilon/\widehat{\sigma} = 0.25$, $p = 0.95$), this time measuring the total communication cost as the measure of efficiency. Here we compared Digest, which is a sample-based (and pull-based) approach, with the naive-sampling pull-based approach (ALL + INDEP) as well as two non-sampling push-based approaches: 1) the baseline solution (ALL + ALL), with which at every snapshot query all tuples from the entire network are pushed to the querying node to evaluate the query (only supports exact queries), and 2) the filter-based solution (ALL + FILTER) proposed by Olston et al. [18], which installs adaptive filters at the nodes to reduce the number of the data updates pushed to the querying node. With ALL + FILTER, we set the user-defined precision interval $[L, H]$ such that $(H - L) < 2\epsilon$, to compare the approaches under equal conditions.

As shown in Figure 5-b (note that the vertical axis is in logarithmic scale), with Digest we improve the communication cost of evaluating a typical continuous query more than one order of magnitude over that of the filter-based solution (ALL + FILTER), and almost two orders of magnitude over that of the baseline solution (ALL + ALL). Also we observe that even a naive sample-based solution (i.e., ALL+INDEP) substantially outperforms an improved non-sampling solution (i.e., ALL + FILTER). Comparing our results in Figure 5-b versus those in Figure 5-a, we note that as mentioned in Section VI-B2 the improvement of Digest over the naive-sampling solution (ALL + INDEP) almost doubles in terms of the communication cost, which reflects the fact that with repeated sampling the cost of deriving the retained samples is negligible and asymptotically only half of the required samples are fresh samples costly to derive from the database. Finally, based on our results, the average costs of deriving each sample are 65 and 43 messages for the simulated weather forecast network and the SETI@HOME network, respectively, loosely consistent with Theorem 4 that predicts a poly-logarithmic complexity for our random sampling operator.

VII. Related Work

In contrast with Digest and other aggregate query answering approaches with which the query evaluation process occurs out of the network at the querying node, there are a number of techniques recently developed for *in-network* aggregate query processing. The randomized distributed algorithms [4], [8] are communication-intensive and their communication overhead is only justified when all nodes of the network issue the same aggregate query simultaneously. TAG [15] incurs less overhead but with its tree-based aggregation scheme, it is prone to severe miscalculations due to frequent fragmentation of the poorly connected topology of the tree, specially in the dynamic peer-to-peer databases. Also, DHT based aggregation techniques [12] are limited to the peer-to-peer databases with structured topologies.

The most relevant related work is the work by Arai et al. [1] on sample-based approximate aggregation queries in peer-to-peer networks, which is limited to snapshot queries, whereas we focus on continuous queries. Finally, we should mention that we have previously presented the preliminary concepts of the work we detailed in this paper as a poster [3].

VIII. Future Work

We intend to extend this study in three directions. First, we plan to complement our reverse regression algorithm by forward regression, which allows adjusting the previous result. Second, we intend to expand on our contributions in this paper to cover more complex aggregate queries with multiple relations and arbitrary select-join predicates. Finally, with the peer-to-peer databases where the time-scale of the data changes is comparable with the sampling time, our snapshot sampling assumption no longer holds. With such peer-to-peer databases, either the sampling techniques should be improved or new semantics should be defined for continuous queries.

Acknowledgments

This research has been funded in part by NSF grant CNS-0831505 (CyberTrust), the NSF Integrated Media Systems Center (IMSC), unrestricted cash-gift from Google, and in part from the METRANS Transportation Center, under grants from USDOT and Caltrans. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] B. Arai, G. Das, D. Gunopulos, and V. Kalogeraki. Approximating aggregation queries in peer-to-peer networks. In *Proceedings of ICDE*, April 2006.
[2] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, September 2001.
[3] F. Banaei-Kashani and C. Shahabi. Fixed-precision approximate continuous aggregate queries in peer-to-peer databases (poster paper). In *Proceedings of the 24nd International Conference on Data Engineering (ICDE'08)*, April 2008.
[4] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. Submitted for publication.
[5] S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing Markov chain on a graph. *SIAM Review*, 46(4):667–689, 2004.
[6] C. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of SIGMOD*, May 1994.
[7] W. Cochran. *Sampling Techniques*. John Wiley and Sons, 3rd edition, 1977.
[8] E. Cohen and H. Kaplan. Spatially-decaying aggregation over a network: model and algorithms. In *Proceedings of SIGMOD*, June 2004.
[9] P. Diaconis and L. Saloff-Coste. What do we know about the Metropolis algorithm. In *Proceedings of STOC*, May 1995.
[10] P. Diaconis and D. Stroock. Geometric bounds for eigenvalues of Markov chains. *Annals of Applied Probability*, 1(1):36–61, 1991.
[11] D. Donjerkovic, Y. Ioannidis, and R. Ramakrishnan. Dynamic histograms: Capturing evolving data sets. In *Proceedings of ICDE*, February 2000.
[12] L. Galanis and D. DeWitt. Scalable distributed aggregate computations through collaboration in peer-to-peer systems. Submitted for publication.
[13] P. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of SIGMOD*, June 1998.
[14] R. Lipton, J. Naughton, and D. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of SIGMOD*, May 1990.
[15] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *Proceedings of OSDI*, December 2002.
[16] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.
[17] S. Meyn and R. Tweedie. *Markov chains and stochastic stability*. Springer-Verlag, 1993.
[18] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of SIGMOD*, June 2003.
[19] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of MMCN*, January 2002.
[20] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *Proceedings of SIGMOD*, June 1992.