Time-sensitive Computation of Aggregate Functions over Distributed Imprecise Data

Qi Han, Matthew Ba Nguyen, Sandy Irani and Nalini Venkatasubramanian
School of Information and Computer Science
University of California, Irvine, CA 92697-3425
{qhan,nguyenmb,irani,nalini}@ics.uci.edu

Abstract

Many distributed applications in the real world now require real time services in which aggregate queries need to be computed over a set of values. These applications can often tolerate varying degrees of inaccuracy in the results. System designers, on the other hand, would like to provide services with low inaccuracy and minimal management overhead. In this paper, we focus on addressing the tradeoffs between timeliness, accuracy and cost for data aggregation in distributed environments. Specifically, we address the problem of time-sensitive computation of aggregate queries(count, sum and min) over a set of values represented by intervals with lower and upper bounds. These intervals are approximations based on most recent values about distributed sources. In order to meet the precision constraints from users, a subset of sources needs to be probed for exact values. We first propose algorithms for batch selection of the probing set, where selection is done before probing without the knowledge of the actual values. In addition, we propose an iterative selection approach where the selection of the next probing source depends on the previous returned value.

1. Introduction

Recent advances in communication, mobile computing and embedded systems have enabled a variety of real-time distributed applications (e.g. stock information exchange, network fault diagnosis, multimedia streaming, environmental sensing etc.). These applications often make decisions based on timely results aggregated over distributed data from varying sources.

Consider project RAIL(Real-time Active Inference and Learning) [9] from IBM as an an example. As distributed computer systems and networks continue to grow in size and complexity, real-time fault localization and problem diagnosis become significantly more challenging. As a result, more sophisticated tools are needed that can assist in performing these management tasks by both responding quickly and accurately to the ever-increasing volume of system measurements, and also actively selecting the minimum number of most-informative tests to run. In other words, it is crucial to build an active, real-time information-gathering and inference system that can "ask the right questions at the right time". A distributed system can be represented as a dependency graph, where nodes can be either hardware elements (e.g., workstations, servers, routers) or software components/services, and links can represent both physical and logical connections between the elements. The following questions can be asked:
Q_1. how many nodes are overloaded?
Q_2. what is the total latency along a path $N_1 \rightarrow N_2 \rightarrow \cdots \rightarrow N_k$?
Q_3. what is the bottleneck (minimum link bandwidth) along a path $N_1 \rightarrow N_2 \rightarrow \cdots \rightarrow N_k$?
Typically, it is desirable to answer these questions in a timely fashion so that diagnosis can be done as quickly as possible.

Answering the above aggregate queries poses several challenges. While obtaining current and precise answers is desirable, techniques to collect and maintain accurate information may not be cost-effective. One strategy would be to collect fresh values from each source every time a query is posed; given roundtrip latencies in unreliable networks, this process might violate timing constraints of the queries. Another possibility would be to require all sources to periodically report changes in readings; this is likely to be expensive in dynamic distributed environments with a large number of sources and fast changing source data. Fortunately, many applications can tolerate data imprecision at a certain level; hence, maintaining absolutely accurate data becomes unnecessary. Therefore, system designers can use the error tolerance of the executing applications to reduce the consumption of communication and computation resources. In our work, we take advantage of an applications' tolerance of

information imprecision to balance the timeliness and accuracy requirements of a request.

Previous research has addressed the tradeoff between accuracy and cost. Given the data intensive nature of these applications, a database is a must for the system to function efficiently. Storing data in a database using a range with a lower bound and an upper bound [2] (instead of using a single instantaneous value) has been identified as an effective strategy to speed up data retrieval while reducing message exchange overhead. Queries such as Q_1,Q_2 and Q_3 can be executed over the cached ranges and return a range which contains the associated value. When a user poses a query, he can specify a precision constraint indicating how wide a range is tolerable in the answer. If the data precision in the database is high enough, user requests can be satisfied by only consulting the repository; otherwise, the data sources can be queried/probed for current exact values, which improves the accuracy at the price of communication overhead. This tradeoff between database accuracy and maintenance overhead has been addressed [14] under the constraint of user precision requirements. When the query is an aggregate query over more than one source, probing all related distributed sources can be expensive both in communication and computation. We can usually satisfy the accuracy requirements of the aggregate functions with exact values from only a subset of the sources. Strategies for probing set selection were proposed [15] that deliver sufficiently accurate answers at minimal cost.

Directly applying the approaches described above to distributed real-time applications will not be effective, since queries not only have accuracy constraints, but also have time constraints which specify the latest time by which the results of an aggregate query are expected to be available. Depending on the characteristics of sources and the network paths leading to the sources, the cost and also the latency of probing vary from source to source. Probing set selection should take into account the probing latency and query time constraints. Our previous work has addressed the timeliness/accuracy/cost tradeoffs in real-time collection of single source value [7]. This paper complements previous work by addressing how the server selects an appropriate subset of sources to probe so that the overall probing cost is minimized without violating accuracy and timeliness requirements of aggregate queries. Specifically, three representative aggregate queries ($count$, sum and min) are considered. Note that computation of $average$ can be derived from $count$ and sum. We propose batch selection algorithms which choose a set of sources in advance that guarantees adequate precision regardless of actual exact values. We also propose an iterative selection algorithm that probe sources iteratively until the precision constraint is met. In addition, we illustrate in which context the iterative approach is preferrable to the batch method.

The rest of this paper is organized as follows. We formulate the problem in Section 2. In Section3, we conduct a worst case analysis on the computation of aggregation functions $count$, sum and min. Furthermore, we present an average case analysis for min. In Section 4, performance results are presented. We discuss related work and conclude in Section 5.

2. Problem Characterization

In this section, we describe system, source and query models used in this paper, and develop a formal characterization of the real-time computation of aggregate queries.

System Model: Our system consists of a number of sources and one server that maintains a database. In a real system, a server may correspond to a data aggregation hub for a set of sources. Various aggregation hubs, taken together, may form the overall distributed database. We, therefore, address the aggregation problem in a simplistic setting of a single access point (server) attached to a set of sources. Our solution can serve as a building block for large scale distributed systems.

Source Model: The system consists of a set of n sources. Let $S = \{1, \ldots, n\}$. Each source stores a variable that can change frequently. For each $i \in S$, let e_i denote the exact value of the variable stored at the i^{th} source. The database stores an approximation for each exact value. The approximation of e_i is represented by a range r_i with lower bound l_i and upper bound u_i: $r_i = [l_i, u_i]$. For any interval $[l, u]$, we will denote interval length $u - l$ by $|[l, u]|$. Whenever the source value e_i changes to e_i', source s_i checks whether r_i is still a valid approximation for the new value. If e_i falls outside r_i, a new approximation of e_i' is sent to the server to update the database. Otherwise, there is no need to transmit the update to the server, hence reducing communication overhead.

Query Model: Queries are executed over the cached ranges at the server. Each incoming query requests the value of $f(e_1, \ldots, e_n)$ (in this paper, the function is f_{count}, f_{sum} or f_{min} for $count$, sum or min), and is associated with an accuracy constraint A indicating its tolerance to error in answer precision. Furthermore, a query has a latency bound D which requires each query to be answered within D time units. The only thing we know about each e_i is that it lies in the range $[l_i, u_i]$. Given that each e_i lies in the range $r_i = [l_i, u_i]$, it can be determined that $f(e_1, \ldots, e_n)$ lies in some range which will be denoted by $f(r_1, \ldots, r_n)$. We define the *uncertainty* of $f(r_1, \ldots, r_n)$ as the length of the interval. If the error tolerance of the query is larger than the data error, it is processed without any communication with the source. Otherwise, the approximation offered by the database is insufficient, the server may request the exact value from part of the remote sources. The sources re-

spond with current exact values and new approximations to be used by subsequent queries. If a subset P of sources is probed, then this may increase the accuracy of the estimate for $f(e_1, \ldots, e_n)$. Let $r_i^P = r_i$ if $i \notin P$ and let $r_i^P = [e_i, e_i]$ if $i \in P$, then the new interval for $f(e_1, \ldots, e_n)$ becomes $f(r_1^P, \ldots, r_n^P)$. However, there is a delay d_i associated with the probing as well as a cost c_i. The cost of probing a subset of sources is the total cost of probing each of them, i.e., $\sum_{i \in P} c_i$. The latency of computing function f denoted by T_f due to probing depends on how the probing is conducted: if the probing is performed in parallel, then $T_f = \max_{i \in P} d_i$; otherwise, if the probing is issued sequentially, then $T_f = \sum_{i \in P} d_i$.

Given a request with an accuracy constraint A and a time constraint D, we would like to identify a subset of the variables $P \subseteq S$ to be probed such that the accuracy and delay constraints are satisfied and the cost is minimized. In other words, we would like to

$$\text{minimize } \sum_{i \in P} c_i,$$
$$\text{subject to}$$
$$(1)\ |f(r_1^P, \ldots, r_n^P)| \leq A,$$
$$(2)\ T_f \leq D.$$

We will use S_D to denote the subset of sources whose delay d_i is at most D. Note that a successful computation not only finishes by the deadline, but also returns the answer at the desired accuracy. However, many deadline misses or accuracy violations can occur depending on system load and time/accuracy constraints. When the time and accuracy constraints can not be met at the same time, we give higher priority to time constraints.

3. Time-sensitive Computation of Aggregate Functions

In order to achieve the goal of minimizing the probing cost under the time and accuracy constraints of user queries, two basic approaches to probing set selection can be applied. One is *batch selection*, where the entire set of sources to probe is selected before the probings actually occur. In this approach, the precision constraint must be guaranteed for any possible precise values for the sources in probing set. The second approach is *iterative selection*, where the source is probed one at a time, computing an answer after each probing and stopping when the answer is precise enough. This is an online approach, where the user is presented with an answer that gradually refines to be more precise over time. In this case, the goal is to shrink the answer as fast as possible. In this section, we present batch selection algorithms for three aggregate queries: $count$, sum and min. Furthermore, we motivate and present an iterative selection algorithm for computing min, where the next source to be probed depends on the previous returned value.

3.1. Batch Selection of Source Probing Set

For each request, we compute the answer based on the stored approximations in the database. If the answer does not satisfy the accuracy constraints of the user request, we decide on a set of sources to probe for exact values in order to improve the answer precision. In the following, we present batch selection algorithms for computing $count$, sum and min.

Computing $count$ (Algorithm BATCH_COUNT): The problem is to calculate the number of source values that fall inside an input range $r = [l, u]$, that is

$$f_{count} = |\{e_i | e_i \in [l, u]\}|.$$

For each variable range r_i there are three cases to consider: r_i is completely contained in r, r_i is disjoint from r, and r_i intersects but is not contained in r. We can divide the variables into three sets: I (inside),O (outside) and U (uncertain) depending on which of the three cases the variable is in. In other words,

- if $l_i \geq l$ and $u_i \leq u$, then $i \in I$;
- if $u_i \leq l$ or $l_i \geq u$, then $i \in O$;
- if $l_i < l \geq u_i$ or $l_i \geq u < u_i$, then $i \in U$.

Therefore, we know that

$$|I| \leq |f_{count}| \leq |I| + |U|, \text{i.e.,} f_{count} = [|I|, |I| + |U|].$$

If $|U| > A$, then we must probe $|U| - A$ variables to determine the function f_{count} within the desired accuracy. Each time a variable is probed, we learn its exact value and can decide whether or not it lies in the range r. Thus, each probe reduces the uncertainty of the answer by exactly 1. Since we have a deadline, we will only consider those variables whose delay d_i is at most the deadline D. This set of variables is called S_D. If $|U \cap S_D| < |U| - A$, then we can not determine the function $count$ to within the desired accuracy. If $|U \cap S_D| \geq |U| - A$, then we order the variables in $|U \cap S_D|$ according to increasing cost and probe the first $|U| - A$ variables in this ordering.

Computing sum (Algorithm BATCH_SUM): The problem is to calculate the sum of all the source values, that is

$$f_{sum} = \sum_{i=1}^{n} e_i.$$

To satisfy time constraints, we do not consider probing those variables whose delay is larger than D. For the remaining set of sources,S_D, if we compute the query based on the stored intervals in the database, then the smallest possible sum occurs when all values are the lower bounds, and the largest possible sum occurs when all values are the upper bounds, i.e.,

$$f_{sum} = [\sum_{s_i \in S_D} l_i, \sum_{s_i \in S_D} u_i].$$

Choosing the subset of sources to probe is equivalent of choosing the subset of sources not to probe: $\bar{P} = S - P$. We observe that

$$|f_{sum}| = \sum_{s_i \in S_D} u_i - \sum_{s_i \in S_D} l_i = \sum_{s_i \in S_D} (u_i - l_i).$$

After probing all sources $s_i \in P$, we have $u_i - l_i = 0$, so these values contributes nothing to the uncertainty. i.e., after probing, $|f_{sum}| = \sum_{s_i \in \bar{P}} (u_i - l_i)$. These equalities combined with the accuracy constraint $|f_{sum}| \leq A$ give us the constraint

$$\sum_{s_i \in \bar{P}} (u_i - l_i) \leq A.$$

The optimization objective is to satisfy this constraint while minimizing the total cost of probing sources in P. We observe that minimizing the total cost of the sources in P is equivalent to maximizing the total cost of the sources not in P. Therefore, the optimization problem can be formulated as choosing $\bar{P}$ so as to

$$\begin{array}{l} \text{maximize } \sum_{s_i \in \bar{P}} c_i \\ \text{subject to } \sum_{s_i \in \bar{P}} (u_i - l_i) \leq A. \end{array}$$

As also observed in [15], this problem can be reduced to $0/1$ Knapsack problem, which is known to be NP-Complete, but there is a polynomial-time approximation scheme for the Knapsack problem that for any ϵ will return a solution within $(1 + \epsilon)$ of optimal in time $O(\frac{1}{\epsilon} n^3)$ [13].

Computing min (Algorithm BATCH_MIN): The problem is to pinpoint the minimum value among all the source values, that is

$$f_{min} = \min_{i=1}^{n} (e_i).$$

Before any sources are probed, a lower bound for the min function is $\min_{i \in S} l_i$ and an upper bound for the min function is $\min_{i \in S} u_i$. If $\min_{i \in S} u_i - \min_{i \in S} l_i > A$, then in the worst case, it will be necessary to probe each variable whose left endpoint l_i is less than $\min_{i \in S} u_i - A$. Let this set of variables be L. With the delay constraints, we can not consider probing any variable whose delay is greater than D. Thus, we are limited to those variables in S_D. We will probe all the variables in $L \cap S_D$. However, it is possible that even after these probes, we do not have the value of the minimum pinned down to the desired level of accuracy. Furthermore, it may not be possible to know until the probes are completed and their values are returned.

Computing the minimum of a set of variables in our model presents interesting questions that do not arise in the context of computing the f_{sum} or f_{count} . The difference is that in f_{sum} or f_{count}, one knows in advance exactly the benefit of probing any particular variable. In the case of f_{count}, probing any variable decreases the uncertainty by 1. In the case of f_{sum}, probing variable e_i decreases the uncertainty by $u_i - l_i$. Thus, one can decide in advance which variables to probe. However, the situation with f_{min} is different. Consider the following example. Suppose we have two sources s_1 and s_2 whose ranges are $[0, 5]$ and $[1, 6]$, respectively. Suppose that we would like to find an interval of length at most one that contains the minimum. If we probe s_1 and its value is 2, we know that the minimum must lie in the interval $[1, 2]$ and the accuracy constraint is met. However, if the value is 5, the interval for the minimum value is $[1, 5]$ and we must probe s_2. Thus, the number of probes required may vary depending on the values of the sources. In the following subsection, we present an iterative approach to selecting probing set for computing min. In other words, we decide what to probe next based on previous probing result.

3.2. Iterative Selection of Source Probing Set

The analysis of min in the previous section is a worst-case analysis which assumes that the values returned always maximize the remaining uncertainty. One may consider an average-case approach in which we assume that the value of each variable is distributed uniformly over its range. Note that we still try to guarantee that the minimum value lies within a range of size at most A. This means that the same set of sources may have to be probed in the worst case. However, it is possible that it may be beneficial in the average case to probe some sources before others.

Algorithm ITERATIVE_MIN: Consider a source s_i with range $[l_i, u_i]$. Let u_{min} denote the current upper bound for the range of the minimum value:$\min_{i \in S} u_i$. Now we consider probing the source s_i. The new upper bound for the minimum value will be $\min\{e_i, u_{min}\}$. Assuming that e_i is uniformly distributed over the range $[l_i, u_i]$, we can calculate the expected amount by which the range for the minimum decreases when e_i is probed. This is:

$$\int_{l_i}^{u_{min}} \frac{u_{min} - x}{u_i - l_i} dx = \frac{(l_i - u_{min})^2}{2(u_i - l_i)}.$$

We will call this the *benefit* of probing s_i and will denote it by $ben(e_i)$. We could probe sources in S_D one at a time and make each probing decision based on the values that have already been returned. Taking into account the cost and latency involved, we define *benefit to cost ratio (BCR)* as $\frac{ben(e_i)}{c_i * d_i}$. We probe the source that maximizes BCR. After each probe, the current range for the minimum is updated. If it is still larger than A, the benefits for each variable are updated and a new source is chosen for probing based on the updated values.

4. Performance Evaluation

The objective of this simulation is to study in detail the performance of the proposed algorithms by comparing them with other simpler algorithms. For each type of aggregate query discussed in this paper, we choose three baseline policies (GREEDY_X, LAZY_X and RANDOM_X, with X being COUNT, SUM or MIN) for comparison:
(a) GREEDY_X: This algorithm probes all the related sources for current exact values. The answer returned is based on all the newly obtained fresh values. Therefore the total cost involved is $\sum_{i \in S} c_i$ and the latency is $\max_{i \in S} d_i$.
(b) LAZY_X: This algorithm probes none of the sources, so the answer of a query is purely dependent upon the accuracy of the stored values in the cache. Clearly, both cost and latency involved is 0.
(c) RANDOM_X: This algorithm randomly picks a subset of sources for probing, so the answer of a query is based on partially accurate and partially stale values.

The following metrics are used to compare the performance of different algorithms:

- cost: It is the total cost of probing sources for exact values. If a subset P of source are probed, the total cost will be $\sum_{i \in P} c_i$.
- accuracy ratio α: This metric measures how close the answer interval (of size a) matches the accuracy constraint A. We define $\alpha = \frac{a}{A}$, so smaller α indicates more accurate answer. When $\alpha \leq 1$, the accuracy constraint is met.
- latency ratio τ: Similar to accuracy ratio, this metric measures how close the time d spent answering the query matches the time constrains D. If a subset P of sources are probed in parallel, the delay in obtaining the probed values is $\max_{i \in P} d_i$, so $\tau = \frac{\max_{i \in P} d_i}{D}$; otherwise, $\tau = \frac{\sum_{i \in P} d_i}{D}$. Smaller τ indicates that less time is spent to answer the query. When $\tau \leq 1$, the time constraint is met.
- accuracy satisfaction ratio: It is the percentage of queries with their accuracy constraints satisfied.
- deadline satisfaction ratio: It is the percentage of queries with their time constraints satisfied.

Our performance study consists of the following experiments: (a) evaluation of all the policies for each type of query; and (b) evaluation of the proposed algorithm for heterogeneous requests in terms of varying accuracy and time constraints.

We built a simulation with 100 sources. Each source holds its exact value, and the local cache holds its interval approximation. The minimum interval size is 10 and the maximum interval size is 20. All the interval sizes are generated randomly from 10 to 20, and the lower bounds of these intervals are randomly generated in the range [5, 12]. The cost of probing each source varies randomly from 1 to 10, and the latency of probing each source also varies randomly from 1 to 10. Each query is accompanied by an accuracy constraint and a time constraint. The accuracy constraints are generated randomly from the range [0, 40]. Time constraints are generated randomly from 5 to 15. All the results are averaged over 20 runs of the experiments.

4.1. Basic Performance Results

In this section, we apply our proposed algorithms and the three baseline algorithms to answer the same set of queries over the data of the same changing pattern. The performance is compared against the five metrics introduced above. Figure 1 shows the basic performance results for computing f_{count}. Not surprisingly, GREEDY_COUNT achieves maximum answer accuracy at the price of highest probing cost and latency. In contrast, LAZY_COUNT provides the most coarse answer instantly by not probing any sources. BATCH_COUNT exhibits similar answer accuracy and latency to RANDOM_COUNT with slightly lower probing cost. However, more queries meet their deadlines by using BATCH_COUNT. This is because BATCH_COUNT gives higher priority to time constraints than accuracy constraints, i.e., the best possible answer (in terms of accuracy) is provided only if the deadline is met. In addition, the sources of less probing cost are chosen in BATCH_COUNT, while random selection of sources as in RANDOM_COUNT could result in arbitrary total cost.

The basic performance for f_{sum} is similar to f_{count}. Figure 2 shows the basic performance results for computing f_{min}. Comparing to RANDOM_MIN, BATCH_MIN provides more accurate answers by spending slightly more in probing. However, its deadline satisfaction ratio is much higher. This is because BATCH_MIN does not probe those sources whose probing latency is higher than the query deadline, while RANDOM_MIN does not take probing latency into account. In addition, studies also indicate that the probabilistic analysis of computing f_{min} is beneficial. It reduces the probing cost significantly. Since source probing is

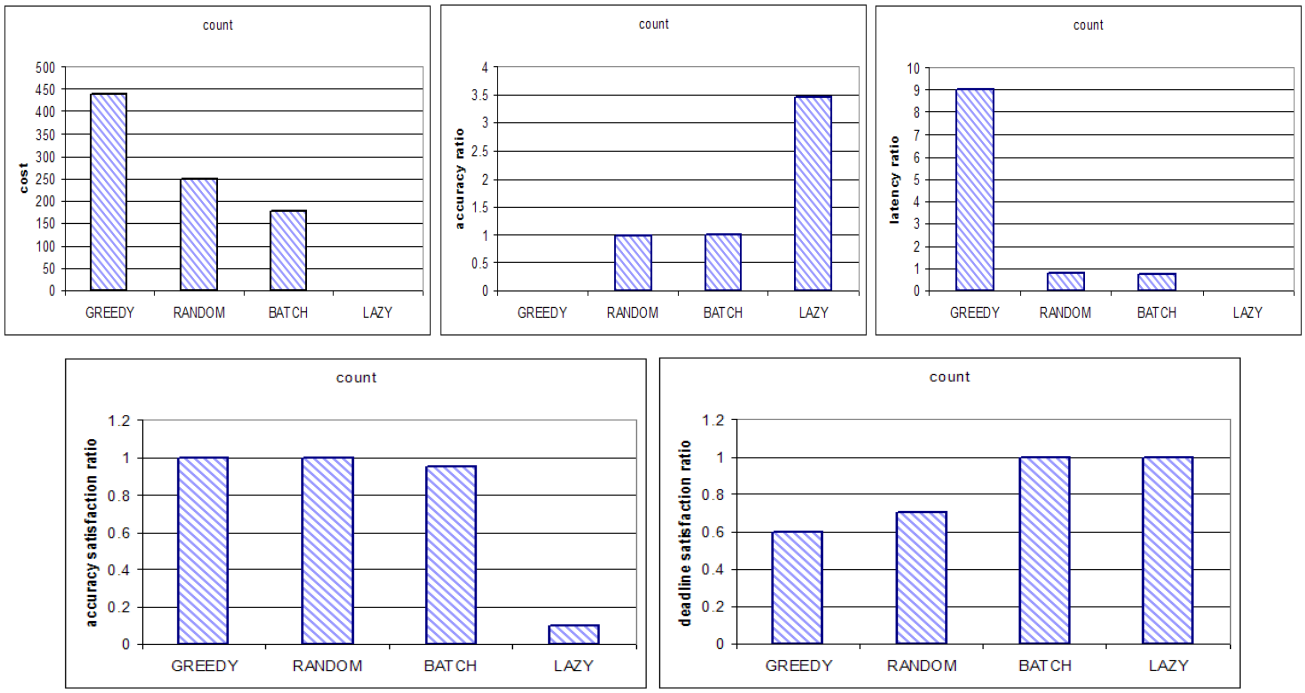

Figure 1. Basic performance comparison for computing f_{count}

done sequentially under ITERATIVE_MIN, the latency of a query is the summation of the latency of each probe. Given the same deadline and this probing limitation, the number of sources to be probed is decreased greatly, which leads to higher accuracy ratio (i.e., less accurate result) and lower accuracy satisfaction ratio.

4.2. Extended Performance Results

We now further study the behavior of BATCH_COUNT, BATCH_SUM and BATCH_MIN as the accuracy constraints of queries vary. Figure 3 shows the performance of BATCH_COUNT under varying accuracy constraints. In order to satisfy the time constraint, only those sources whose probing latency is within the deadline are probed. However, when the accuracy constraint is very tight, this subset cannot provide an accurate enough answer. Therefore, the first part of the curve for cost is horizontal and the first part of the curve for accuracy ratio is decreasing but larger than 1. When the accuracy constraint is relaxed further, fewer probings can provide satisfactory answer. The performance of BATCH_SUM and BATCH_MIN under varying accuracy constraints follows similar trend as BATCH_COUNT. Worthy of mentioning is the sharp increase of accuracy ratio of BATCH_MIN when the accuracy constraint is 11. This is because when the accuracy constraint is relaxed to a certain degree, the cached value can easily satisfy it, therefore no probing is necessary. The accuracy constraint is met, but the accuracy ratio is increased compared to the previous cases where probing is necessary.

Similarly, we evaluate the performance of BATCH_COUNT, BATCH_SUM and BATCH_MIN as deadlines change. In the case of BATCH_COUNT, the several turning points in the curve of probing cost matches exactly the several stages of the algorithm. We only probe the subset S_D whose probing latency is smaller than the deadline. Small deadline leads to small S_D, and S_D increases as the deadline increases. Therefore, the probing cost increases. At the same time, the accuracy ratio decreases, but still larger than 1. When the deadline reaches a point where we can select a subset of S_D to probe, the probing cost is decreased since we select those with less probing costs. When the deadline increases further, no more improvement will be obtained since the same subset of sources will be probed to meet the accuracy and time constraints. For BATCH_SUM, as the deadline increases, more sources can be probed to decrease accuracy ratio. Similar results are exhibited by BATCH_SUM and BATCH_MIN.

5. Related Work and Conclusions

There has been extensive research in different caching strategies. One such strategy is adaptive caching, which adjusts when, how and what to cache dynamically as conditions change. For instance, a technique called divergence

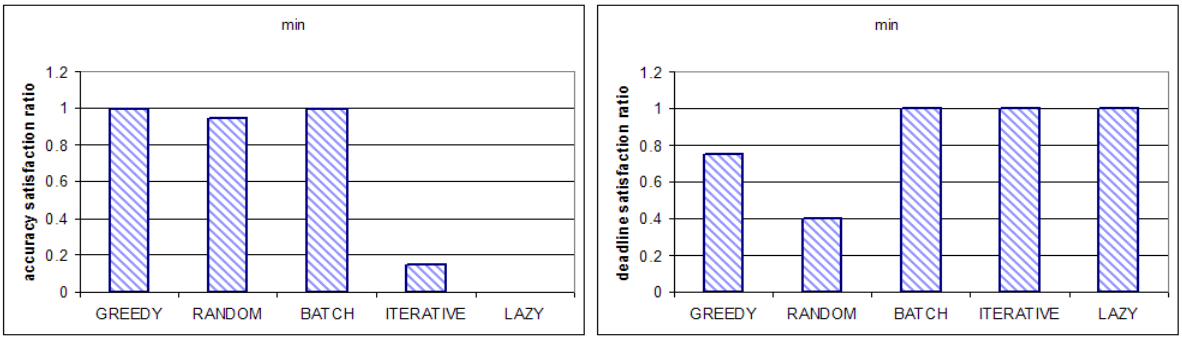

Figure 2. Basic performance comparison for computing f_{min}

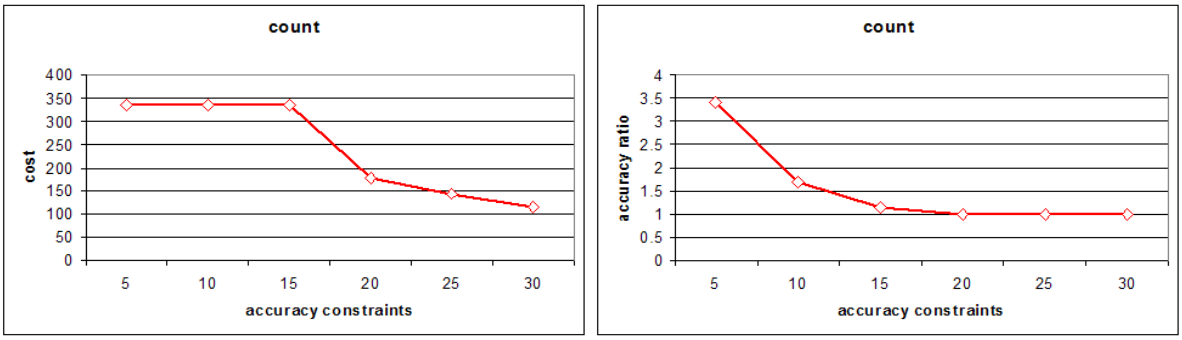

Figure 3. Performance of f_{count} under varying accuracy constraints

caching considers setting the precision of approximate values in a caching environment [8], where precision is inversely proportional to the number of updates to the source value not reflected in the cached approximation, independent of the actual updates. Along the same line, range based data caching for numerical values is used to deal with the tradeoff between performance and precision [14], where intelligent cache refreshment strategies are proposed to ensure queries for each single data item can be answered within their precision constraints. Their work is further applied into the TRAPP system [15], where aggregate queries accompanied by precision constraints are answered by automatically selecting a combination of locally cached bounds and exact master data stored remotely. Furthermore, our previous work considered the scenario where both precision and time constraints are imposed on single-item queries [7]. The algorithms proposed in this paper are for aggregate queries and can be integrated into the framework.

Researchers have been re-investigating traditional data management and processing techniques in the presence of multiple continuous time-varying data streams. Data streaming is now a vibrant research field for both traditional Internet environments (such as OpenCQ, Niagara, Telegraph,STREAM and Aurora) and emerging ad-hoc sensor networks (such as COUGAR and Quasar). Research in this area has addressed a whole gamut of issues such as system architectures, concepts/semantics of continuous queries, QoS specifications for fresh information delivery, system scalability etc. The work presented in this paper is complementary to the research in data streaming.

Computing a function over n inputs is studied in the framework of boolean and/or trees [3], where the source data is accurate and each input has an associated price. However, the authors look for answers without precision requirements or time bounds. Similarly, they considered the problem of competitively finding the maximum of n elements but with additional costs for comparisons between pairs of elements. The problem of computing aggregate functions over approximate data has been addressed to deliver results within specified accuracy constraints while introducing minimal overhead [6, 11, 5], where no time issues are considered. The problem of answering aggregate queries in multi-dimensional databases in an approximate manner is addressed by using a modified tree index data structure multi-resolution aggregate(MRA) to store aggregate information about data at successive levels of resolution as defined by the space partitioning/data group hierarchy. A progressive algorithm is developed to iteratively give increasing quality answers until some error bound is satisfied or timing constraints is reached [12]. The tradeoff

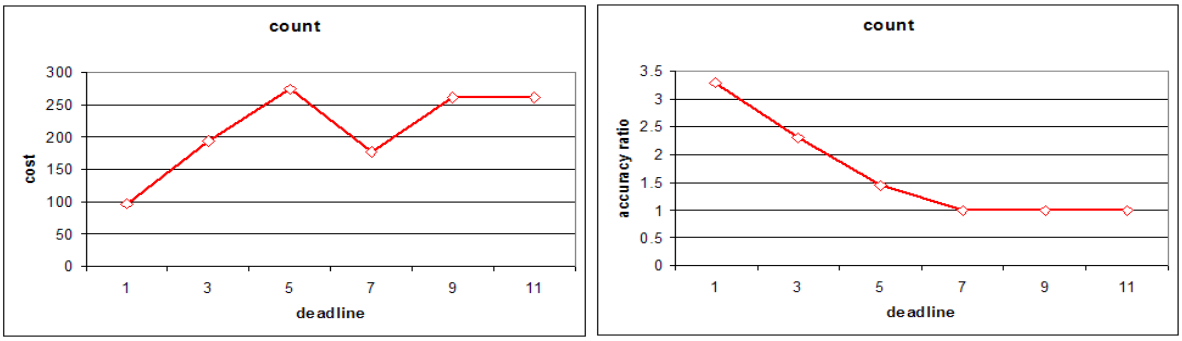

Figure 4. Performance of f_{count} under varying time constraints

between transaction timeliness and data freshness has also been addressed from the database transaction management perspective, such as STRIP [1], ARCS [4] and QMF [10].

Concluding Remarks: In this paper, we studied the problem of computing aggregate functions over a set of values which are bounded by lower and upper bounds and whose exact values can be probed at the cost of various latency. Our worst case analysis provides a bound on the cost to satisfy queries regardless of the values returned when probed. More interesting problems raise when a more sophisticated model such as Gaussian distribution is used to capture the change of data values. It is also interesting to conduct a competitive analysis for answering aggregate queries. Furthermore, handling multiple requests with overlapping interested sets of values will pose more challenging issues. Time-sensitive data aggregation is an essential step to providing real-time information services in distributed environments.

References

[1] B. Adelberg, H. Garcia-Molina, and B. Kao. Applying update streams in a soft real-time database system. In *ACM SIGMOD*, 1995.

[2] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Quality of service based routing: A performance perspective. In *ACM SIGCOMM*, 1998.

[3] M. Charikar, R. Fagin, V. Guruswami, J. Kleinberg, P. Raghavan, and A. Sahai. Query strategies for priced information. In *ACM STOC*, 2000.

[4] A. Datta and I. Viguier. Providing real-time response, state recency and temporal consistency in databases for rapidly changing environments. *Information Systems*, 22(4), 1997.

[5] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. In *Proceedings of STACS*, 2003.

[6] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. In *ACM STOC*, 2000.

[7] Q. Han and N. Venkatasubramanian. Addressing timeliness/accuracy/cost tradeoffs in information collection for dynamic environments. In *Proceedings of IEEE RTSS*, 2003.

[8] Y. Huang, R. Sloan, and O. Wolfson. Divergence caching in client-server architectures. In *IEEE PDIS*, 1994.

[9] I.Rish, M. Brodie, S. Ma, G. Grabarnik, and N. Odintsova. Using adaptive probing for real-time problem diagnosis in distributed computer systems. In *Proceedings of AAI-02/ KDD-02/ UAI-02 workshop on Real-Time Decision Support and Diagnosis Systems*, 2002.

[10] K. D. Kang, S. H. Son, J. A. Stankovic, and T. F. Abdelzaher. A qos-sensitive approach for miss ratio and freshness guarantees in real-time databases. In *The 14th Euromicro Conference on Real-Time Systems*, 2002.

[11] S. Khanna and W. C. Tan. On computing functions with uncertainty. In *ACM PODS*, 2001.

[12] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *ACM SIGMOD*, 2001.

[13] O.H.Ibarra and C. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4), October 1975.

[14] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD*, 2001.

[15] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB*, 2000.