ICICLES: Self-tuning Samples for Approximate Query Answering

Venkatesh Ganti[*] Mong Li Lee Raghu Ramakrishnan

{vganti,leeml,raghu}@cs.wisc.edu

Department of Computer Sciences, University of Wisconsin-Madison

Abstract

Approximate query answering systems provide very fast alternatives to OLAP systems when applications are tolerant to small errors in query answers. Current sampling-based approaches to approximately answer aggregate queries over foreign key joins suffer from the following drawback. All tuples in relations are deemed equally important for answering queries even though, in reality, OLAP queries exhibit locality in their data access. Consequently, they may waste precious real estate by sampling tuples that are not required at all or required very rarely.

In this paper, we introduce *icicles*, a new class of samples that tune themselves to a dynamic workload. Intuitively, the probability of a tuple being present in an icicle is proportional to its importance for answering queries in the workload. Therefore, an icicle consists of more tuples from a subset of the relation that is required to answer more queries in the workload. Consequently, the accuracy of approximate answers obtained by using icicles is better than a static uniform random sample. We show, analytically, that for a certain class of queries reflected by the workload, icicles yield more accurate answers. In a detailed experimental study, we examine the validity and performance of icicles.

[*]Supported by a Microsoft Graduate Fellowship.

Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.

1 Introduction

Advances in information collection and management have enabled businesses to build large data warehouses containing data about customers and their daily business activity. The realization that careful analysis of this data yields valuable business intelligence led to the emergence of *on-line analytic processing (OLAP)* systems for decision support. The goal of these systems is to provide interactive response times to aggregate queries. However, very large database sizes may not allow true interactivity despite careful design and development of an OLAP system.

Approximate query answering (AQUA) systems are being developed with a goal to reduce response times to true levels of interactivity [VL93, AGPR99b, CMN99, PG99, SFB99, MS00]. That most decision support applications can tolerate approximate answers to queries is exploited by AQUA systems to achieve truly interactive response times.

Several approximate query answering approaches for different data domains have been proposed: sampling-based [AGPR99b, AGPR99a], histogram-based [PG99], clustering-based [SFB99], probabilistic [MS00], and wavelet-based [VW99] approaches. Both histogram-based and wavelet-based approaches assume that attributes of underlying relations are numerical. The sampling-based approach does not require such assumptions. Moreover, only sampling-based approaches have been explored for approximately answering aggregate queries over joins of relations. Therefore, we consider sampling-based techniques in this paper.[1]

Gibbons et al. introduce the concept of a *join synopsis* which, informally, is a uniform random sample over foreign-key joins of relations [AGPR99b]. Depending on the database schema they compute a set of join synopses to approximately answer aggregate queries. However, such a static sampling approach has the following drawback. All tuples within each relation (or a foreign-key join of a set of relations) are assumed to be equally important while collecting a random sample. However, in practice, OLAP

[1]This is not to say that the sampling-based approach is qualitatively better than others. In fact, other approaches may be enhanced in the same way we enhance a sampling-based approach.

queries follow a predictable repetitive pattern, and exhibit locality in their data access [DSRN98]. Therefore, treating all tuples uniformly while sampling wastes precious main-memory on tuples that are not required for answering queries.

As an example, consider the data warehouse of the `Widget Tuners` company that maintains all sales information of widgets in United States over the past ten years. A salesperson in the Madison office of `Widget Tuners` may only be interested in analyzing the sale of widgets in Wisconsin over the past one or two years. In such cases, maintaining a uniform random sample over the entire relation wastes real estate because it samples sales information on regions outside of Wisconsin for the last ten years, and the early eight year window of Wisconsin sales. It may waste more than 95% of the available main memory on sampling unnecessary information. But, since we do not a priori know the focus of analysis, a straight-forward approach of jacking up the sampling rate in a given region of the relation cannot be applied.

In this paper, we address the above-mentioned drawback of a static sampling strategy to answer aggregate queries over foreign key joins. The intuition behind our approach is to incrementally maintain a sample, called *icicles*,[2] in which the probability of a tuple being selected is proportional to the frequency with which it is required to answer queries (exactly). That is, the probability is proportional to the number of query predicates in the workload that the tuple satisfies. Therefore, an icicle is expected to consist of more tuples in regions that are accessed more frequently to answer queries. We exploit this property to provide more accurate answers to aggregate queries. Our contributions in this paper are:

1. We introduce a new class of samples, called *icicles*, to capture the data locality exhibited by a set of aggregate queries over foreign key joins.

2. We develop a dynamic algorithm that tunes an *icicle* with respect to the most recent knowledge of the workload (Section 3).

3. We develop estimators for average, count, and sum aggregates to answer queries using icicles (Section 4).

4. We analytically show that icicles are better than static samples for approximately answering a certain class of queries that conform to the workload (Section 6).

5. In an extensive experimental evaluation, we compare the performance of icicles with (static) join synopses (Section 7).

2 Icicles

In this section, we discuss informally the intuition behind icicles, the algorithm for maintaining icicles, and the problem in using traditional estimators with icicles. We start

[2]The number of accesses to tuples in a workload-sensitive sample relative to that over tuples in the relation resembles an *icicle*.

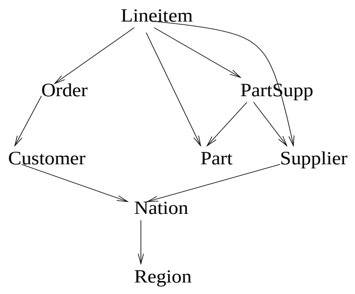

Figure 1: Foreign Key Relationships in TPC-D

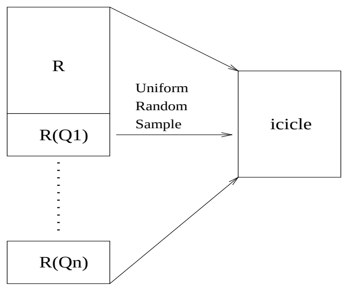

Figure 2: Logical Relation and an icicle

with a brief discussion of earlier work on sampling-based approaches for approximately answering aggregate queries.

Gibbons et al. [AGPR99b] and Chaudhuri et al. [CMN99] showed that a join of random samples of base relations may not be a random sample of the join of base relations. Gibbons et al. exploit the property that relations in many decision support applications are connected by foreign key relationships. For instance, in a star schema, the fact table and dimension tables are connected by foreign key relationships, and queries over such schema typically involve foreign key joins. They introduced the concept of a *join synopsis*, which is the join of a uniform random sample of the fact table with a set of dimension tables. Any aggregate query involving the fact table can now be answered approximately using exactly one of a small number of synopses.

Example 2.1 As a running example, we consider the TPC-D database [Cou95] shown in Figure 1. It consists of the following relations: `Lineitem (LI), Order (O), Customer (C), Part (P), PartSupp (PS), Supplier (S), Nation (N), and Region (R)`. `LI` is the fact table and `O, P, S` are constrained by foreign key relationships with the `LI` table. The complete schema is shown as a directed graph in Figure 1 wherein a directed edge from relation A to relation B captures the existence of a foreign key constraint from A to B. A query Q derived from query 5 in the TPC-D benchmark is shown below. Note that all joins in the query are foreign key joins. Therefore, this query can be answered approximately using a join synopsis on the join of `LI, C, O, S, N,` and `R` relations.

`SELECT` COUNT(*), AVG(LI_Extendedprice),

```
        SUM(LI_Extendedprice)
FROM LI, C, O, S, N, R
WHERE C_Custkey=O_Custkey AND O_Orderkey=LI_Orderkey
AND LI_Suppkey=S_Suppkey AND C_Nationkey=N_Nationkey
AND N_Regionkey=R_Regionkey AND R_Name=North America
AND O_Orderdate≥01-01-1998 AND O_Orderdate≤12-31-1998;
```

In the above TPC-D example query that typifies decision support queries, an analyst is interested in sales in the North American continent for the year 1998. Analysts in North America are unlikely to be very interested in sales in other continents. Therefore, all their queries may be focused on the sales after 1998 in North America. In such cases, a uniform random sample of the entire relation wastes precious main-memory by drawing a significant number of tuples from sets of tuples that are not as "important" for answering queries. The space is better utilized if more sample tuples are gathered from 1998 sales in North America resulting in a significant improvement of the accuracy of approximate answers.

Note that we do not a priori know the subset or sets of subsets on which the analysis is focused. Moreover, the focus may change with time. Therefore, we have to automatically identify the focus of a query workload and sample accordingly. We introduce a new class of workload-sensitive samples, called *icicles*, that capture the data locality exhibited by queries in the workload. *An icicle is a uniform random sample of a multiset of tuples L, which is the union of R and all sets of tuples that were required to answer queries in the workload.* We call L the *extension* of R with respect to the workload.

We maintain an icicle of a relation R (or a foreign key join of relations) as follows. To start with, L is exactly the same as R, and an icicle is a uniform random sample of R. For each query Q on R, we derive a multiset L' by appending to L the set of all tuples in R that satisfy the query predicate (and hence were required to answer Q). The icicle is now updated to be a random sample of the new multiset L'. We use the reservoir sampling algorithm [Vit85] to maintain the invariant that an icicle is a random sample of L. (Note that the algorithm for updating icicles only accesses the new block of data that is logically appended to L.) Therefore, if a particular set of tuples is retrieved for a lot of queries then an icicle is expected to consist of a greater proportion of tuples from this set.

2.1 Icicle-based Estimators

An icicle is a non-uniform sample of the original relation. Hence, traditional estimators for aggregate operators cannot always be applied directly. For instance, consider the TPC-D query Q shown in Example 2.1. Suppose the size of the lineitem relation LI consists of 1000000 tuples and the sample consists of 1000 tuples. Suppose $LI(Q)$ consists of 100000 tuples. A uniform random sample of LI is expected to consist of 100 tuples from $LI(Q)$. Starting with a random sample of relation LI, if the *count query* is executed five times, the number of tuples from $LI(Q)$ in the icicle is expected to increase by five times to 500. However, the size of the extension of L increases marginally by an amount $5 \cdot |LI(Q)|$ to a value 1500000. It is easy to see that the traditional estimation of scaling up the count from a sample S of LI by a factor of $\frac{|LI|}{|S|}$ is way off the mark!

We need new estimators for providing more accurate answers. In Section 4, we derive icicle-based estimators for the average, count, and sum aggregate operators. The basic idea is to maintain additional information to adjust for the selection bias in an icicle. We maintain a set of *frequencies*, one per tuple in the relation, where the frequency of a tuple is the number of times it was required to answer queries in the workload.

2.2 Quality Guarantees

The accuracy of an approximate answer is expected to increase with the number of tuples in the sample that were actually used to compute it. Since we expect an icicle to contain a greater proportion of tuples from frequently accessed sets of tuples, the quality of answers computed using these sets is better. If the number of queries that use such frequent sets is much greater than the number of queries that do not, the average quality of answers improves as well. Under the assumption that the past workload reflects the future workload, we expect that an icicle provides more accurate answers than a statically collected uniform random sample of the relation. In Section 6, we show analytically that for a broad class of queries computing averages, icicles are expected to provide more accurate answers than simple uniform random samples. However, if an occasional query does not conform to the workload or if the workload changes drastically, then the accuracy of the approximation for that query or for a set of queries is worse than a static sample. But, as shown in Section 7, icicles adapt quickly to a workload. Therefore, such sudden changes in a workload will deteriorate performance only for a short period of time.

2.3 Sampling Over Foreign Key Joins and Base Relations

Due to the results in [AGPR99b, CMN99], we focus on answering aggregate queries involving only foreign key joins (especially, joins between fact tables and dimension tables). Since join synopses are uniform random samples over foreign key joins of a set of base relations, any query in the targeted class can be answered using only one of a small number of join synopses.[3] Therefore, maintaining join synopses is conceptually equivalent to maintaining a set of uniform random samples, each drawn from a single (logical) relation. Therefore, in the algorithmic description, we assume that samples are being drawn from single relations. However, in our experimental study, we evaluate the performance of icicles for answering aggregate queries over foreign-key joins. To avoid any confusion with the precise

[3]The total number of such foreign key joins in a star schema is limited ([AGPR99b]).

definition of a join synopsis (see [AGPR99b]), we use the term *static sample* instead of join synopsis.

3 Icicle Maintenance

In this section, we introduce our notation and definitions. We then describe the algorithm for maintaining icicles with respect to a dynamically changing workload.

3.1 Definitions and Notation

We now introduce our terminology beginning with some basic notation. Following the reasons discussed earlier, we assume that icicles are drawn from single relations. In this paper, we study the class of *average*, *count*, and *sum* aggregate operators.

We use $A(Q)$ to denote a query involving the aggregate A (one of average, count, or sum). A *workload* $\mathcal{Q}$ is a set of aggregate queries. If S is a multiset of tuples, we use $\tilde{S}$ to denote the set of distinct tuples in S. Let S_1 and S_2 be two multisets. We use $S_1 \uplus S_2$ to denote the multiset-union of S_1 and S_2. That is, an element is repeated in the result once for every appearance in either S_1 or S_2.

For the remainder of this section, we assume that R is a relation and $\mathcal{Q}$ is a workload. We use $A(Q, R)$ to denote the *exact answer* for the aggregate query $A(Q)$ over the relation R.

Given R and a query $A(Q)$, the *query-relevant subset* $R(Q)$ *of* R *for* $A(Q)$ is the set of tuples in R that satisfy the query predicate. $R(Q)$ is also the *minimal set* of tuples in R that are required to exactly answer $A(Q)$. Note that the subset $R(Q)$ of R is independent of the query processing algorithm. An algorithm may scan the entire relation R and determine the (minimal) subset required to answer $A(Q)$, or it may retrieve the minimal subset through an index-scan.

Definition 3.1 The *frequency relation* $F(R, \mathcal{Q}) = \{f(t, R, \mathcal{Q}) | t \in R\}$ is a set of frequencies one for every tuple $t \in R$ where, $f(t, R, \mathcal{Q}) = \sum_{Q \in \mathcal{Q}} t \in R(Q)$. That is, the frequency with respect to $\mathcal{Q}$ of a tuple $t \in R$ is the number of queries in $\mathcal{Q}$ whose query-relevant subsets contain t. The *frequency sum* $\sigma(R, \mathcal{Q})$ is the sum of all frequencies in $F(R, \mathcal{Q})$. That is, $\sigma(R, \mathcal{Q}) = \sum_{t \in R} f(t, R, \mathcal{Q})$. $\odot$

A workload where all tuples in a relation are retrieved equally frequently to answer queries in the workload is called the *uniform workload*. Let $F(R, U)$ represent a uniform workload on R. Then, for any $t \in R$, $f(t, R, U) = c$ for some non-negative integer c. Without loss of generality, we assume that $c = 1$.

Definition 3.2 The *extension* $L(R, \mathcal{Q})$ *of* R *with respect to* $\mathcal{Q}$ is defined as follows.

$$L(R, \mathcal{Q}) = R \uplus_{Q \in \mathcal{Q}} R(Q)$$

An *icicle* $S(R, \mathcal{Q})$ of R with respect to $\mathcal{Q}$ is a uniform random sample of $L(R, \mathcal{Q})$. $\odot$

3.2 Maintenance Algorithm

We now describe our algorithm to maintain an icicle over a single relation. We then generalize it to a set of icicles over a set of relations.

Intuitively, we maintain an icicle such that, at all times, the probability of a tuple's presence in the icicle is proportional to its "importance" in answering queries in a workload. We quantify the importance of a tuple to be its frequency with respect to the workload. Hence, we maintain an icicle such that a tuple is selected with a probability proportional to its frequency. However, we do not want to scan the complete relation whenever we update the icicle. The efficient incremental maintenance of an icicle is possible due to the following two observations. First, a uniform random sample of the extension of a relation R with respect to a workload ensures that each tuple's selection in the icicle is proportional to its frequency. Second, incremental maintenance of a random sample of the extension of R only requires the segment $R(Q)$ of the relation each time a new query $A(Q)$ is answered by the system. We use the *reservoir sampling* algorithm [Vit85] to maintain a random sample of the extension.

As mentioned in Section 2.1, we maintain the frequency relation to derive good estimates for aggregate queries. For clarity in presentation, we treat the maintenance of the frequency relation $F(R, \mathcal{Q})$ independently. However, in Section 5, we describe in detail the maintenance of the frequency relation. For now, we assume that a routine to update frequencies of a set of tuples exists.

```
Algorithm 3.1 UpdateIcicle($L(R, \mathcal{Q})$, $S(R, \mathcal{Q})$, $A(Q)$)
/* $L(R, \mathcal{Q})$:extension of R, $S(R, \mathcal{Q})$:icicle, $A(Q)$:new query */
begin
    Step 1: Compute $R(A(Q))$.
    Step 3: Set $L(R, \mathcal{Q}) = L(R, \mathcal{Q}) \uplus R(A(Q))$.
    Step 2: Update $S(R, \mathcal{Q})$ such that it is a uniform
            random sample of the updated $L(R, \mathcal{Q})$.
    Step 4: Increment the frequencies in $F(R, \mathcal{Q})$ of
            each tuple $t \in R(A(Q))$.
end
```

The pseudocode for the algorithm to update icicles is presented above. At any instant, an icicle $S(R, \mathcal{Q})$ is a uniform random sample of $L(R, \mathcal{Q})$. Conceptually, we stream each tuple being appended to $L(R, \mathcal{Q})$ through a reservoir sampling procedure [Vit85] that maintains a uniform random sample of $L(R, \mathcal{Q})$.[4] When a new query $A(Q)$ is added to the workload $\mathcal{Q}$, we stream tuples in $R(Q)$ through the reservoir sampling procedure to update the icicle $S(R, \mathcal{Q})$ such that it is a uniform random sample of $L(R, \mathcal{Q}) \uplus R(Q)$. Even though the pseudocode serializes steps 1 and 2, we pipeline the streaming operator into the

[4]The reservoir sampling algorithm maintains a uniform random sample of the relation as follows: it scans the relation one tuple at a time and randomly decides whether to insert it into the sample. If the tuple is added to the sample, another tuple in the sample is kicked out.

reservoir sampling procedure. Thus, we do not need to simultaneously store all of $R(Q)$ in-memory. Also, note that $L(R, \mathcal{Q})$ is not explicitly materialized at all.

3.2.1 Workload Restriction

Note that the icicle-maintenance algorithm accesses all tuples in the set $R(Q)$ that satisfy the query predicate of a query $A(Q)$. Therefore, the amount of work done to update an icicle is almost the same as the work done to answer the query exactly. If the workload includes queries where a user is happy with just an approximate answer, the overhead in maintaining icicles is significant. Therefore, we *restrict* the workload to only consist of queries in the system that require exact answers to queries.[5] This restriction allows us to update icicles for free because tuning an icicle with respect to a new query $A(Q)$ requires the set $R(Q)$ of tuples that satisfy the query predicate. Observe that this set of tuples would have been retrieved anyway for answering the query, and hence we do not perform any additional work for updating the icicle if we piggyback the update with the processing for the query.

Restricting the workload to only consist of exactly answered queries is reasonable because we expect a typical data analysis session to proceed as follows. Analysts derive their initial conclusions on data characteristics based on an approximate or cursory analysis where they can tolerate small errors. To confirm their initial conclusions realized from an interactive exploratory session, they issue one or more queries to be answered exactly. Therefore, the final exactly-answered queries are actual queries of interest and are likely to be representative of the true workload. Because the interactive approximate querying session leads an analyst to the final set of interesting queries, it is indispensable.

Note that duplicates of tuples may be present in an icicle because a tuple may be required several times to answer queries in a workload. It is possible to collapse such duplicates together by maintaining a *count* of the number of times a tuple appears in the icicle. Gibbons et al. showed that this approach is better than maintaining duplicates [GM98]. To avoid muddling the performance study of icicles with improvements due to collapsing duplicates, we stick with the straight-forward implementation of icicles that allows duplicate tuples. However, note that this optimization only improves the performance of icicles.

Example 3.1 We illustrate our icicle-maintenance algorithm through an example. Consider a hypothetical *Widget-Tuners* relation shown in Figure 3. We add a new *frequency* attribute to the schema of the relation to hold frequencies of tuples. A hypothetical icicle created from a hypothetical workload is shown in Figure 4. Suppose the following query is asked by an analyst to compute the average of all sales in the month of April.

```
SELECT average(*) FROM Widget-Tuners
WHERE Date.Month='April';
```

[5] Alternative restrictions like choosing only a subset of all queries may also be possible.

We require the two April tuples corresponding to orders 5 and 6 in Figure 3 to answer the above query exactly. The set $\{OrderId = 5,\ OrderId = 6\}$ consisting of these two tuples is (conceptually) appended to the extension of Widget-tuners. We now update the icicle such that it is a random sample of the updated extension. Suppose the updated icicle is as shown in Figure 5. Note that we increment the frequencies of tuples corresponding to orders 5 and 6, and that these updates are reflected in the updated icicle.

3.2.2 Maintaining Multiple Icicles

Previously, we described an algorithm for maintaining an icicle on a single relation. We now extend it to multiple relations. The extension relies on the partitionability of the workload into a set of workloads, one per relation.

Recall that our target class of queries is the class of aggregate (average, count, and sum) queries over foreign key joins. As discussed earlier, this class is equivalent to a class of single-table queries where tables may be formed from foreign-key joins of a set of relations. Any query may be answered using exactly one icicle on a relation. (It is a fallout of the fact that icicles are "tuned" synopses.) Therefore, the overall workload can be *partitioned* into several independent workloads, one for each relation. Due to the properties that the workload is partitionable and that queries access a single table, we can independently maintain icicles on relations. That is, the maintenance of an icicle on one relation does not depend on how an icicle on a different relation is maintained. Given a certain amount of space to maintain icicles, we distribute the available space among them and maintain each icicle independently. Any space-partitioning strategy to distribute space among icicles may be used here. We refer the reader to the work by Gibbons et al. [AGPR99b] for a detailed discussion of several strategies. We now formalize this procedure.

Let $R_1, \ldots, R_k$ be k relations. Let $\mathcal{Q} = \mathcal{Q}_1 \uplus \cdots \uplus \mathcal{Q}_k$ be a workload where queries in $\mathcal{Q}_i$ only access the relation R_i. Let M be the amount of space allocated for maintaining icicles. We maintain k icicles as follows. Using any space allocation strategy, allocate space M_i to an icicle $S(R_i, \mathcal{Q}_i)$ on relation R_i, $1 \leq i \leq k$. For a query $Q \in \mathcal{Q}_i$, we update the icicle $S(R_i, \mathcal{Q}_i)$ using the *UpdateIcicle* Algorithm.

4 Estimators for Aggregate Queries

In the previous section, we described how we maintain an icicle with respect to a changing workload. We now describe the methodology of answering aggregate queries using the icicle. Due to the presence of duplicates and the selection bias (or non-uniformity) in an icicle, traditional estimators (e.g.,[Olk93, AGPR99b]) do not apply directly. An example was discussed in Section 2.1 to illustrate this problem. We now derive icicle-based estimators for the average, count, and sum aggregate operators. We first discuss the intuition behind our aggregate estimators, and then formalize them in Lemma 4.1. (Due to space constraints, we do not present the proof of the lemma.)

OrderID	Date	Price	Frequency
1	01-Jan-1998	100	3
2	01-Feb-1998	200	1
3	01-Feb-1998	150	1
4	01-Mar-1998	125	1
5	01-Apr-1998	120	$2 \rightarrow 3$
6	01-Apr-1998	140	$2 \rightarrow 3$

Figure 3: Widget-tuners

OID	Date	Price	Frequency
1	01-Jan-1998	100	3
4	01-Mar-1998	125	1
6	01-Apr-1998	140	2

Figure 4: An icicle of Widget-tuners

OID	Date	Price	Frequency
1	01-Jan-1998	100	3
5	01-Apr-1998	120	**3**
6	01-Apr-1998	140	**3**

Figure 5: Updated icicle of Widget-Tuners

When the sum or average is being computed, we use X to denote the aggregated attribute.

Average: We assume that for any query $A(Q)$ the value a tuple $t \in R(Q)$ assumes for the aggregated attribute X is independent of the probability that t is present in the icicle. That is, the distribution of values of X in the set of tuples accessed by the query from the icicle is the same as that in $R(Q)$. Under this assumption, the average of the set of distinct sample tuples that satisfy a given average query is a good estimate of the average. Note that our assumption is similar to the (implicit) assumption made by Gibbons et al. [AGPR99b]. Their assumption was that the value distribution of an attribute is independent of query predicates which, in turn, determine the probability of a tuple's presence in an icicle.

Count: Intuitively, we expect each tuple in an icicle to correspond to a set of tuples in the original relation R. We first compute an estimate of the number of tuples in R that contribute one tuple t in the icicle. We call this value the *expected contribution* of t. We then estimate the count aggregate to be the sum of expected contributions of all tuples in the icicle that satisfy the given query.

Sum: Under the assumption that for any query $A(Q)$, the value a tuple $t \in R(Q)$ assumes for the aggregated attribute X is independent of the probability that t is present in the icicle, the average and the count aggregates are independent of each other. Therefore, the estimate of the sum aggregate is given by the product of the average and count estimates.

Lemma 4.1 Let $A(Q)$ be an aggregate query over the relation R, and $S = S(R, \mathcal{Q})$ be an icicle over R with respect to a workload $\mathcal{Q}$. Let $F(R, \mathcal{Q})$ be the frequency relation. Let the probability of a tuple t being present in S be independent of the probability that t assumes a certain value for an aggregated attribute X. Then $E[A(Q, R)]$

$$= \begin{cases} E[A(Q, \tilde{S})], & \text{if A=avg} \\ \frac{\sigma(R,\mathcal{Q})}{|S|} \sum_{t \in S(Q)} \frac{1}{f(t,R,\mathcal{Q})} & \text{if A=count} \\ E[avg(Q, \tilde{S})] \cdot E[count(Q, S)], & \text{if A=sum} \end{cases}$$

5 Maintaining the Frequency Relation

Recall that we maintain the frequency associated with each tuple for estimating count queries. We now discuss an efficient procedure for maintaining the frequency relation $F(R, \mathcal{Q})$ of a relation R with respect to a workload $\mathcal{Q}$.

To hold the frequency for each tuple, we add a new *frequency* attribute to the relation R. To start with, the frequency of each tuple is set to 1. The frequency of a tuple is incremented whenever it is accessed to answer a query. Therefore, whenever we update an icicle with a new query, we update frequencies of all tuples required to answer the query. Consequently, the frequency column is update-intensive. Even though these updates may be performed off-line when the system is idle, they still constitute a significant overhead. We now describe an efficient procedure for updating frequencies of tuples that exploits the following two properties. First, frequencies need not be maintained up-to-date. That is, it is not absolutely necessary to immediately update the frequency of a tuple whenever it is accessed by a query. Second, data analysis queries exhibit data locality. Therefore, frequencies of a significant portion of often-accessed tuples can be maintained in main memory.

The intuition behind our approach is to delay updating the frequency of a tuple until the magnitude of the change in frequency crosses a certain threshold. The delay helps to batch several related updates (either on the same tuple or on distinct tuples) together thus significantly reducing the overall number of updates. We maintain a main-memory hash table consisting of two attributes: *tuple identifier* and *count*. The *count* for a tuple t in the hash table is the number of times t was required to answer queries since the last time the frequency $f(t, R, \mathcal{Q})$ of t has been updated on disk. The hash table maintains a set of tuples and the counts associated with them.

Whenever a tuple t that is required to answer a query, is read into main memory, we check if t is present in the hash table. If not, we add t to the hash table and set its count to 1. If t is present in the hash table, we increment its count in the hash table. If the count crosses a threshold C (fixed a priori), we update the frequency $f(t, R, \mathcal{Q})$ of t on the disk and delete t from the hash table. We may further optimize this algorithm by updating frequencies of all other tuples in the hash table that belong to the same disk page as t.

5.1 Special Case of the Average Operator

We now discuss a very important optimization for the special case when we are only interested in answering queries that compute averages. We observe that we do ***not*** need to maintain the frequency relation if we restrict ourselves to answering queries with the average operator.

Recall that estimators for the count and the sum aggregate operators require frequencies of tuples in their computation while the computation of the estimate for the average operator does not (Section 4). Also, note that the frequency relation is not required for any purpose other than estimating count and sum aggregate operators. Therefore, if we restrict the class of queries to be answered approximately to the class of aggregate queries computing averages, then we do not need to maintain frequencies of tuples. Thus, restricting the query class significantly reduces the costs for maintaining icicles.

6 Quality Guarantees

In this section, we compare the quality of answers obtained from icicles with answers obtained from static samples. When queries in a workload exhibit data locality, then icicles consist of more tuples from frequently accessed subsets of the relation. The accuracy of an approximate answer improves with the number of tuples used to compute it. Therefore, if a new query reflects the data locality of earlier queries in the workload, then the icicle-based answer to this query is more accurate than that based on a uniform random sample of the relation. We now prove that icicles provide more accurate answers for a certain class of queries computing averages. Intuitively, this class consists of queries "focused" with respect to the workload that created an icicle.

We say that a query $A(Q)$ is *focused* with respect to a workload $\mathcal{Q}$ if any tuple in the set $R(Q)$ is, on an average, accessed more often by queries in $\mathcal{Q}$ than queries in a uniform workload. That is, for a query $A(Q)$ over the relation R, the average frequency with respect to $\mathcal{Q}$ of a tuple in $R(Q)$ is greater than that for a uniform workload. We now formalize this notion below.

Definition 6.1 Let R be a relation and $\mathcal{Q}$ be a workload. Let $F(R,\mathcal{Q})$ be the frequency relation of R with respect to $\mathcal{Q}$. Let $\sigma(R,\mathcal{Q}) = \sum_{t \in R} f(t,R,\mathcal{Q})$. We say that an aggregate query $A(Q)$ is *focused* with respect to $\mathcal{Q}$ if

$$\frac{\sum_{t \in R(Q)} f(R,t,\mathcal{Q})}{\sigma(R,\mathcal{Q})} \geq \frac{|R(Q)|}{|R|}$$

$\odot$

Lemma 6.1 Let R be a relation, $\mathcal{Q}$ be a workload, and $F(R,\mathcal{Q})$ be the frequency relation of R with respect to $\mathcal{Q}$. Let $\sigma(R,\mathcal{Q}) = \sum_{t \in R} f(t,R,\mathcal{Q})$. Let $S(R,\mathcal{Q})$ be an icicle created by $\mathcal{Q}$ and S be a uniform random sample of R. The expected fraction of $R(Q)$ tuples in S is $\frac{|R(Q)|}{|R|}$, and the expected fraction of $R(Q)$ tuples in $S(R,\mathcal{Q})$ is $\frac{\sum_{t \in R(Q)} f(t,R,\mathcal{Q})}{\sigma(R,\mathcal{Q})}$.

Theorem 6.1 *Let R be a relation, $\mathcal{Q}$ be a workload on R, and $F(R,\mathcal{Q})$ be the frequency relation of R with respect to $\mathcal{Q}$. Let $S(R,\mathcal{Q})$ be an icicle on R. Let S be a uniform random sample on R. If a query $average(Q)$ is focused with respect to $\mathcal{Q}$, then the icicle $S(R,\mathcal{Q})$ yields a more accurate estimate for $average(Q)$ than the uniform random sample S.*

The above theorem follows from Lemma 6.1. We now present a straight-forward extension of Theorem 6.1 to multiple relations. The extension merely relies on the partitionability of the workload and that the amount of space allocated to each icicle and the corresponding uniform static sample on a relation are equal.

Corollary 6.1 *Let $R_1, \ldots, R_k$ be k relations. Let the workload $\mathcal{Q} = \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k$ be such that any query $Q \in \mathcal{Q}_i$ only accesses relation R_i. For $1 \leq i \leq k$, let S_i be a uniform random sample of R_i and $S(R_i,\mathcal{Q}_i)$ be an icicle on R_i with respect to the workload $\mathcal{Q}_i$ such that $|S_i| = |S(R_i,\mathcal{Q}_i)|$. Then a query $average(Q_i)$ focused with respect to the workload $\mathcal{Q}_i$ will be answered more accurately by $S(R_i,\mathcal{Q}_i)$.*

7 Performance Evaluation

In this section, we evaluate the accuracy of answers, obtained from icicles, to aggregate queries for a wide variety of workloads, and their adaptability to the characteristics of a workload.

7.1 Experimental Testbed

We now describe our experimental setup and briefly outline our experimental procedure. We use the TPC-D decision support benchmark to generate a variety of workloads. Using a scale factor of 0.3, we generate 300 megabytes of test data.

Workload Characteristics and Parameters: We fix the number of queries in a workload to be 40. We use the query $Q_{workload}$ shown in Figure 6 to generate workloads. ($Q_{workload}$ is derived from the query Q_5 of the TPC-D benchmark.) The query has two parameters: *region* and *start date*. Region takes values from the set $\{$AFRICA, AMERICA, ASIA, EUROPE, MIDDLE-EAST$\}$. The start date may vary between 01-01-1993 and 06-31-1998. A workload is characterized by the *number of frequently accessed groups* of tuples in a relation, and the *relative size* of each group. The number of frequently accessed groups is varied using the *region* attribute. We vary the sizes of

```
SELECT COUNT(*), AVG(LI_Extendedprice), SUM(LI_Extendedprice)
FROM LI, C, O, S, N, R
WHERE C_Custkey=O_Custkey AND O_Orderkey=LI_Orderkey AND LI_Suppkey=S_Suppkey AND
      C_Nationkey = N_Nationkey AND N_Regionkey = R_Regionkey AND
      R_Name = [region] AND O_Orderdate ≥ Date[startdate] AND O_Orderdate ≤ 12-31-1998
```

Figure 6: $Q_{workload}$: Template for generating workloads

```
SELECT COUNT(*), AVG(LI_Extendedprice), SUM(LI_Extendedprice)
FROM LICOS-icicle, N, R
WHERE C_Nationkey = N_Nationkey AND N_Regionkey = R_Regionkey AND
     R_Name = [region] AND O_Orderdate ≥ Date[startdate] AND O_Orderdate ≤ 12-31-1998
```

Figure 7: Template for obtaining approximate answers

frequently-accessed groups by varying the *start date* parameter (both month and year components) appropriately. Note that the relative size of a group of tuples accessed by a query equals the *selectivity* of the query predicate. We generate queries in the workload by setting (with an element of randomness) the region and start date parameters of $Q_{workload}$.

The join synopsis $LICOS$ we use to answer the query $Q_{workload}$ in Figure 6 is a uniform random sample of the join of `lineitem` LI, `customer` C, `order` O, and `supplier` S relations. An icicle is a tuned sample of this synopsis. The query, corresponding to $Q_{workload}$, for obtaining an approximate answer from an icicle is shown in Figure 7. (A similar query is used for using a join synopsis.) We fix the sizes of an icicle or a static sample on $LICOS$ join to be 1% of the size of the join of all four relations.

Relative Error: For each query $A(Q)$ on the relation R, we compute the *relative error* for an answer $\hat{A}_S(Q, R)$ obtained using a sample S on R as follows. (We assume that the exact answer $A(Q, R) > 0$.)

$$\text{relative-error}(S, Q) = \frac{|\hat{A}_S(Q, R) - A(Q, R)|}{A(Q, R)}$$

Plots for each Workload: For each workload, we show three plots. The first plot (indexed by the legend *static sample*) corresponds to the errors incurred by a static uniform random sample (join synopsis) of the join of LI, C, O, and S. The second plot (indexed by the legend *icicle*) shows the errors from an icicle as it evolves with the workload. That is, after each query in the workload is answered by the system, we update the icicle; the subsequent query is answered by the updated icicle. The third plot (indexed by the legend *icicle-complete*) shows the errors from a tuned icicle created by executing the entire workload. That is, we first create an icicle "tuned" with respect to queries in a workload. We then answer all queries in the (same) workload (again) using the tuned icicle and compute the error. Intuitively, this plot corresponds to the best possible accuracy from an icicle of a given size. Of course, this is not a feasible evaluation strategy; it is only intended as a way to compare how our proposed evaluation compares in terms of a sampling strategy that has foreknowledge of the entire workload. We only show the plots for average and count estimators because the behavior of the sum estimator is already captured by these two estimators.

7.2 Quality of Approximate Answers

In this experiment, we study the accuracy of aggregate query answers obtained from icicles on both synthetic and real datasets.

7.2.1 Varying Selectivities

In this experiment, we fix the number of frequently accessed groups at 1, and vary the relative size of a frequently accessed group. Queries in the workload are derived from $Q_{workload}$. We fix the region parameter to be ASIA. We study the behavior on two such workloads. In the first workload, we fix the year component of the start date to 1998 and vary the month component between $January$ and $June$. In the second workload, we fix the year component of the start date to 1997 and vary the month component between $January$ and $December$. Note that the size of the frequently accessed group in the first workload (year=1998) is much smaller than that for the second workload (year=1997). We first show the 1998 plots (Figures 8 and 9), and then the 1997 plots (Figures 10 and 11).

Figures 8 and 9 demonstrate the *rapid* decrease in relative error of query answers obtained from icicles as more queries focused on a core set of tuples are added to the workload, and the icicle updated with tuples required to answer them. Note that the relative error of query answers obtained from icicles decreases significantly from 25% to 5%. Also, observe that the icicle plot converges to the icicle-complete plot even before 20 queries are answered. The quick convergence of the icicle plot to the icicle-complete plot demonstrates that icicles adapt quickly to the workload characteristics.

The results for the 1997 plots are shown in Figures 10 and 11. The conclusions are similar. Even though the relative error in answers from icicles stabilizes around 5% (as for 1998 plots), the relative improvement in accuracy due to the use of icicles is less than that for 1998 be-

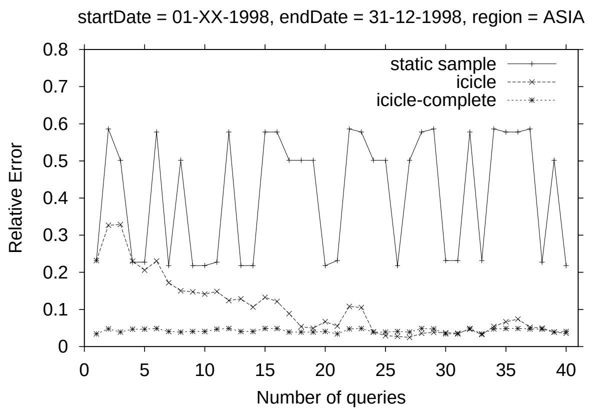

Figure 8: Average Queries

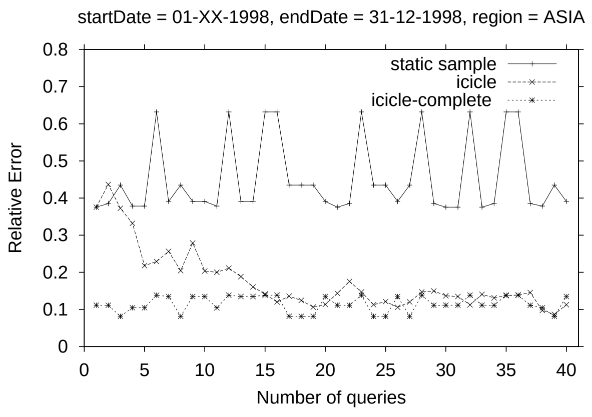

Figure 9: Count Queries

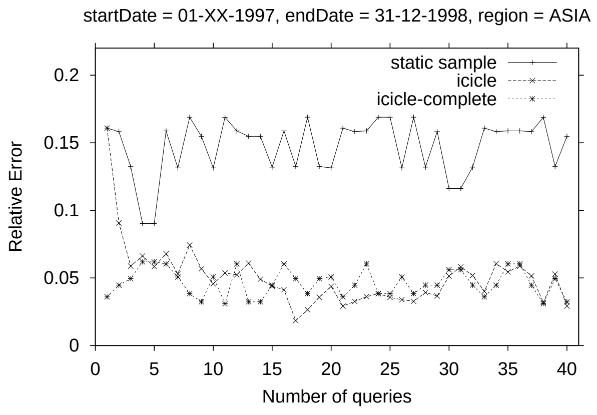

Figure 10: Average Queries

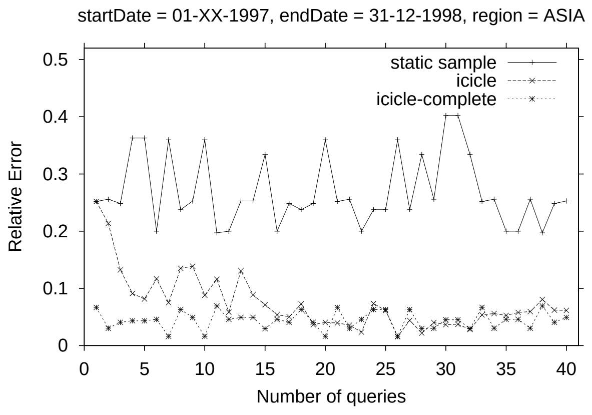

Figure 11: Count Queries

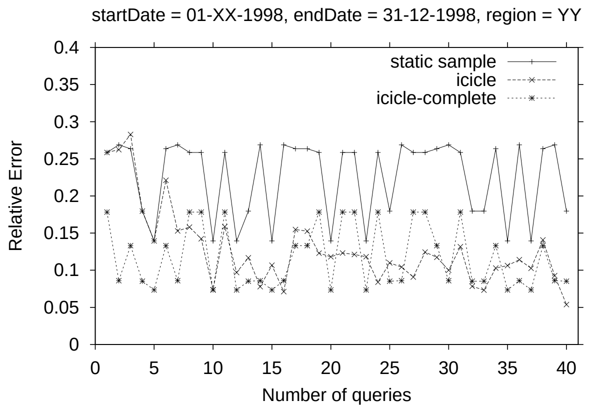

Figure 12: Average Queries

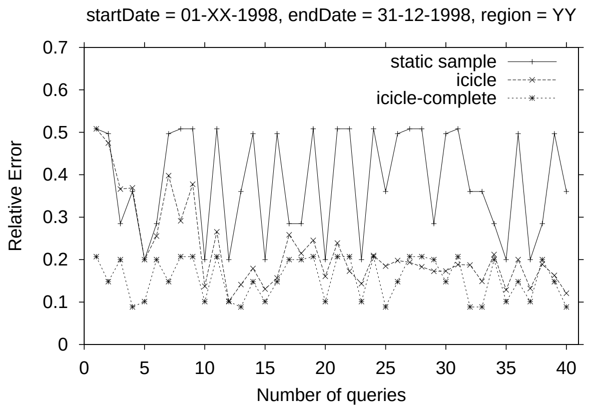

Figure 13: Count Queries

cause both icicles and static samples start at lower error values. Also, the convergence of the icicle plot to the icicle-complete is much faster than that for 1998. These observations are as expected because the absolute number of both 1997 and 1998 tuples in a uniform random sample of the $LICOS$ join is higher than that for 1998 alone.

The plots are not smooth because all queries in the workload are distinct from each other as the values for parameters to $Q_{workload}$ are chosen randomly. Therefore, the exact answers and relative errors of the estimated answers are different and vary significantly from each other. The variation is, in turn, depicted in the plots.[6] The fluctuations in errors are higher when the year component of the start date is fixed at 1998 because the absolute value of the result is much smaller for 1998. And, varying the month component causes relatively larger fluctuations in the exact result as well. These fluctuations are, in turn, exhibited by the relative error plots because the exact answer appears in the denominator of the expression for the relative error.

7.2.2 Varying Number of Groups and Selectivities

In this experiment, we increase the number of frequently accessed groups in a workload. The relative fraction is varied around a fixed constant. We generate such a workload by creating queries from $Q_{workload}$ where the region parameter is uniformly randomly selected from the set {AFRICA, AMERICA, ASIA, EUROPE, MIDDLE-EAST}. As in the previous experiment, we generate two workloads. In the first workload the year component of the start date is fixed at 1998 and the month component varied between $January$ and $June$. In the second workload, the year component is fixed at 1997 and the month component is varied between $January$ and $December$. Each workload has 5 frequently accessed groups, one per region.

The results for the first workload are shown in Figures 12 and 13, and those for the second workload in Figures 14 and 15. The conclusions from these plots are similar to those drawn in the previous section. The relative errors of answers decrease steadily when icicles are used to answer queries. The icicle plot converges quickly to the icicle-complete plot.

7.2.3 A Mixed Workload

In this section, we vary both the number of frequently accessed groups as well as relative sizes of each group. The region is randomly picked from {AFRICA, AMERICA, ASIA, EUROPE, MIDDLE-EAST} while the year component of the start date is randomly picked to be between 1993 and 1998. Note that both the year and region parameters are chosen uniformly at random from their domains. Hence, the workload is not focused on specific sets of tuples. Figures 16 and 17 show the results of this experiment. Therefore, as expected, the improvement due to the use of icicles is not significant. The important point to observe here though is that *icicles are as good as static samples*, if not better.

[6] This argument holds for almost all plots that follow.

```
SELECT count(*), average(price), sum(price)
FROM Sales-icicle
WHERE orderDate ≥ [start date]
```

Figure 20: Template Query on Mail Order Data

7.2.4 Mail Order Dataset

In this section, we present the performance of icicles on a real mail order dataset. Besides other information, the dataset has the following attributes: customer identifier, order date, style, price, quantity, cost, and gender code ('M' for men and 'F' for women). The dataset consists of 45000 tuples. We generate a workload by varying the start date parameter in the template query shown in Figure 20. The start date is varied over an interval of 31 days in May. (Overall, the date attribute varies between 08-01-1995 and 12-31-1996.) The size of the icicles and static samples constructed on this workload are set such that the sample has 3000 tuples.

We present the results from this experiment in Figures 18 and 19. The results and conclusions are similar to those on the TPC-D data. Again, icicles outperform static samples by a significant margin thus validating our approach to maintain more sample tuples from frequently accessed groups.

8 Related Work

The application of statistical techniques to approximate query answering has recently received attention. However, to the best of our knowledge, there is no technique that tunes data summaries (samples or any other class of summaries) with respect to a dynamic workload. We now briefly discuss previous approaches for approximate query answering, which we have not already touched upon.

We have already discussed the approach based on join synopses [AGPR99b]. Chaudhuri et al. explored the issue of introducing sampling as a relational operator [CMN99]. Hellerstein et al. proposed a framework for providing a series of monotonically improving approximate answers to a single query [HHW97].

Barbara et al. [BDF$^+$97] present a survey of various statistical estimation techniques for data reduction, some of which have been applied for approximately answering aggregate queries. For instance, histogram-based approaches [PG99], wavelet-based approaches [VWI98, VW99], mining based approaches [SFB99, MS00] have been explored. We believe that our ideas in this paper are applicable to these summaries as well even though actual (maintenance) techniques will be different. In the APPROXIMATE query processor, Vrbsky and Liu develop notions of approximations to set-valued answers and develop computation techniques [VL92]. Their techniques are not based on statistical summaries of the data.

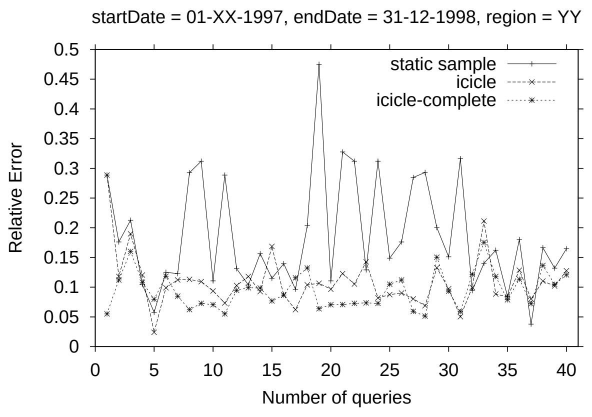

Figure 14: Average Queries

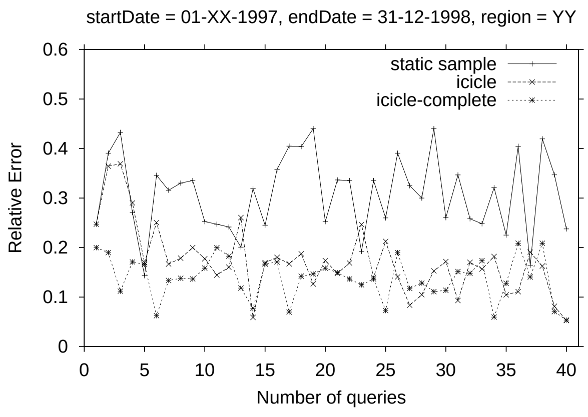

Figure 15: Count Queries

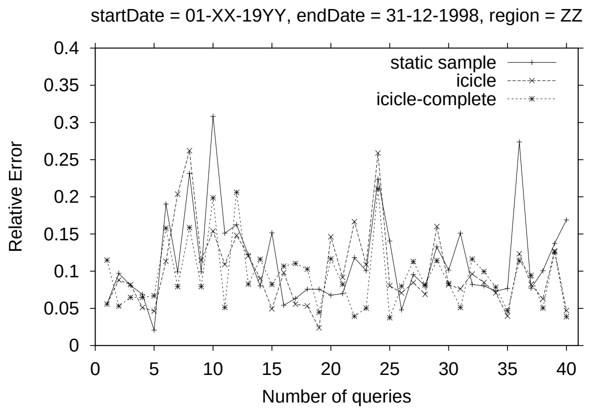

Figure 16: Mixed Workload: Average Queries

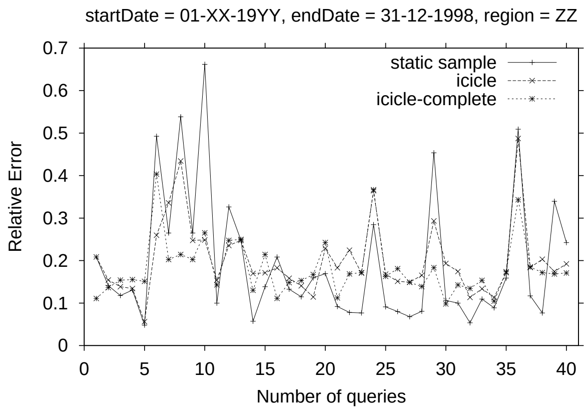

Figure 17: Mixed Workload: Count Queries

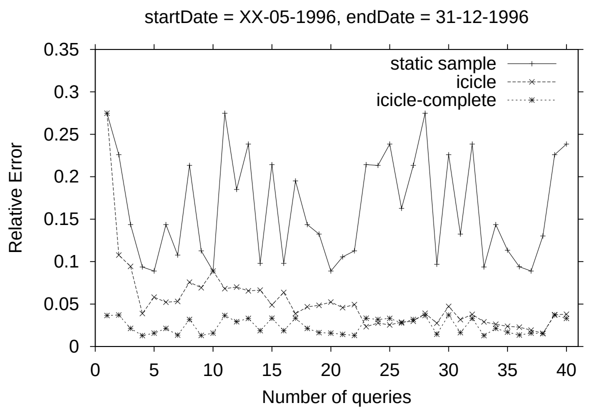

Figure 18: Mail Order Dataset: Average Queries

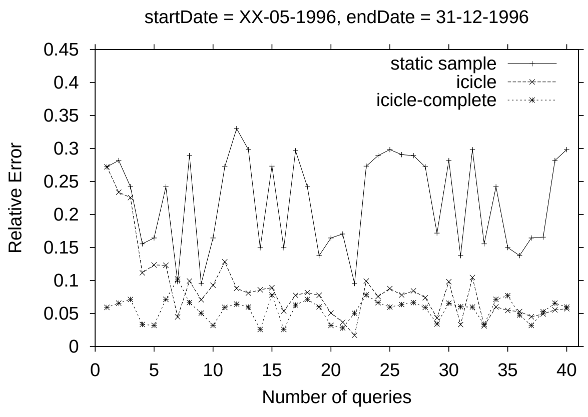

Figure 19: Mail Order Dataset: Count Queries

9 Conclusions and Future Work

We introduced a new class of samples, icicles, that are sensitive to the workload characteristics and described an incremental algorithm to maintain them with respect to a changing workload. We developed new icicle-based estimators for approximately answering aggregate queries. We show analytically and empirically that icicles are better than static samples for a wide variety of workloads. Our empirical studies further show that icicles are especially good when the workload is focused on relatively small subsets of tuples in a relation. In most cases, icicles provide more accurate answers than static samples, and they are at least as good as static samples. Finally, they adapt quickly to the characteristics of a workload.

In future, we intend to extend the maintenance of icicles when the underlying relations are being modified through ad hoc insertions and deletions.

Acknowledgements We thank Donko Donjerkovic for pointing to us that the traditional count estimator is not applicable with icicles. We also thank Kevin Beyer and Prof. Wei-Yin Loh for discussions on statistical estimators.

References

[AGPR99a] S. Acharya, P.B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *Proceedings of ACM SIGMOD international conference on management of data*, Philadelphia, PA, June 1999. Demonstration paper.

[AGPR99b] S. Acharya, P.B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proceedings of the ACM SIGMOD International Conference on Managment of Data*, pages 275–286, Philadelphia, PA, June 1999.

[BDF$^+$97] Daniel Barbara, William DuMouchel, Christos Faloutsos, Peter J. Haas, Joseph M. Hellerstein, Yannis E. Ionnidis, H.V. Jagadish, Theodore Johnson, Raymond T. Ng, and Viswanath Poosala. The new jersey data reduction report. *Data Engineering Bulletin*, 20(4), 1997.

[CMN99] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. In *Proceedings of the ACM SIGMOD International Conference on Managment of Data*, pages 263–274, Philadelphia, PA, June 1999.

[Cou95] Transaction Processing Performance Council, May 1995. http://www.tpc.org.

[DSRN98] Prasad Deshpande, Amit Shukla, Karthik Ramasamy, and Jeffrey Naughton. Caching multidimensional queries using chunks. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Seattle, WA, June 1998.

[GM98] P.G. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the ACM SIGMOD International Conference on Managment of Data*, pages 331–342, Seattle, WA, June 1998.

[HHW97] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In Joan M. Peckman, editor, *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 171–182, May 1997.

[MS00] Heikki Mannila and Padhraic Smyth. Approximate query answering with frequent sets and maximum entropy. In *Proceedings of the Sixteenth IEEE International Conference on Data Engineering (ICDE)*, page 309, San Diego, CA, March 2000.

[Olk93] Frank Olken. *Random Sampling from Databases*. PhD thesis, University of California at Berkeley, 1993.

[PG99] Vishy Poosala and Venkatesh Ganti. Fast approximate answers to aggregate queries on a datacube. In *Proceedings of the 11th International Conference on Scientific and Statistical Database Management*, pages 24–33, Cleveland, Ohio, July 1999.

[SFB99] Jayavel Shanmugasundaram, Usama Fayyad, and Paul Bradley. Compressed data cubes for olap aggregate query approximation on continuous dimensions. In *Proceedings of the ACM SIGKDD fifth international conference on knowledge discovery in databases*, pages 223–232, San Diego, CA, August 1999.

[Vit85] Jeffrey Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.

[VL92] S. Vrbsky and J. Liu. Producing approximate answers to set-valued and single-valued queries with APPROXIMATE. In *Int. Conf. on Information and Knowledge Management, Baltimore*, November 1992.

[VL93] S.V. Vrbsky and J.W.S. Liu. Approximate–a query processor that produces montonically improving answers. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):1056–1068, 1993.

[VW99] Jeffrey Vitter and Min Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the ACM SIGMOD International Conference on Managment of Data*, Philadelphia, PA, June 1999.

[VWI98] Jeffrey Vitter, Min Wang, and Bala Iyer. Data cube approximation and histogram via wavelets. In *Proceedings of the seventh international conference on information and knowledge management*, pages 96–104, Washington D.C., November 1998.