A Quick Introduction to Approximate Query Processing

CS286, Spring'07
Minos Garofalakis

Decision Support Systems

• Data Warehousing: Consolidate data from many sources in one large repository.
  - Loading, periodic synchronization of replicas.
  - Semantic integration.
• OLAP:
  - Complex SQL queries and views.
  - Queries based on spreadsheet-style operations and "multidimensional" view of data.
  - Interactive and "online" queries.
• Data Mining:
  - Exploratory search for interesting trends and anomalies. (Another lecture)

Introduction & Motivation

Outline

• Intro & Approximate Query Answering Overview
  - Synopses, System architectures, Commercial offerings
• One-Dimensional Synopses
  - Histograms, Samples, Wavelets
• Multi-Dimensional Synopses and Joins
  - Multi-D Histograms, Join synopses, Wavelets
• Set-Valued Queries
  - Using Histograms, Samples, Wavelets
• Discussion & Comparisons
• Advanced Techniques & Future Directions
  - Dependency-based, Workload-tuned, Streaming data

Fast Approximate Answers

• Primarily for Aggregate Queries
• Goal is to quickly report the leading digits of answers
  - In seconds instead of minutes or hours
  - Most useful if can provide error guarantees
  
  E.g., Average salary
  $59,000 +/- $500 (with 95% confidence) in 10 seconds
  vs. $59,102.45

• Achieved by answering the query based on samples or other synopses of the data
• Speed-up obtained because synopses are orders of magnitude smaller than the original data

Approximate Query Answering

Basic Approach 1: Online Query Processing

- e.g., Control Project [HHW97, HH99, HAR00]
- Sampling at query time
- Answers continually improve, under user control
**Approximate Query Answering**

**Basic Approach 2: Precomputed Synopses**

- Construct & store synopses prior to query time
- At query time, use synopses to answer the query

- Like estimation in query optimizers, but
  - reported to the user (need higher accuracy)
  - more general queries

- Need to maintain synopses up-to-date

- Most work in the area based on the precomputed approach
  - e.g., Sample Views [OR92, Oik93], Aqua Project [GMP97a, AGP99, etc]

---

**The Aqua Architecture [GMP97a, AGP99]**

**Picture without Aqua:**
- User poses a query Q
- Data Warehouse executes Q and returns result
- Warehouse is periodically updated with new data

**Picture with Aqua:**
- Aqua is middleware, between the user and the warehouse
- Aqua Synopses are stored in the warehouse
- Aqua intercepts the user query and rewrites it to be a query Q’ on the synopses. Data warehouse returns approximate answer

---

**Online vs. Precomputed**

**Online:**
- Continuous refinement of answers (online aggregation)
- User control: what to refine, when to stop
- Seeing the query is very helpful for fast approximate results
- No maintenance overheads
- See [HH01] Online Query Processing tutorial for details

**Precomputed:**
- Seeing entire data is very helpful (provably & in practice)
  (But must construct synopses for a family of queries)
- Often faster: better access patterns, small synopses can reside in memory or cache
- Middleware: Can use with any DBMS, no special index striding
- Also effective for remote or streaming data

---

**Commercial DBMS**

- Oracle, IBM Informix: Sampling operator (online)
- IBM DB2: “IBM Almaden is working on a prototype version of DB2 that supports sampling. The user specifies a priori the amount of sampling to be done.”
- Microsoft SQL Server: “New auto statistics extract statistics [e.g., histograms] using fast sampling, enabling the Query Optimizer to use the latest information.” The index tuning wizard uses sampling to build statistics.
  - see [CN97, CN98, CN98]

**In summary, not much announced yet**

---

**Approximate Query Processing using Data Synopses**

- How to construct effective data synopses??
**Outline**

- Intro & Approximate Query Answering Overview
- One-Dimensional Synopses
  - Histograms: Equi-depth, Compressed, V-optimal, Incremental maintenance, Self-tuning
  - Samples: Basics, Sampling from DBs, Reservoir Sampling
  - Wavelets: 1-D Haar-wavelet histogram construction & maintenance
- Multi-Dimensional Synopses and Joins
- Set-Valued Queries
- Discussion & Comparisons
- Advanced Techniques & Future Directions

---

**Histograms**

- Partition attribute value(s) domain into a set of buckets
- Issues:
  - How to partition
  - What to store for each bucket
  - How to estimate an answer using the histogram
- Long history of use for selectivity estimation within a query optimizer [Kao80], [PSC84], etc.
- [PIH96] [Poo97] introduced a taxonomy, algorithms, etc.

---

**I-D Histograms: Equi-Depth**

- Can **maintain** using one-pass algorithms (insertions only), or:
- Use a backing sample [SGM97b]: Maintain a larger sample on disk in support of histogram maintenance
  - Keep histogram bucket counts up-to-date by incrementing on row insertion, decremented on row deletion
  - **Merge** adjacent buckets with small counts
  - **Split** any bucket with a large count, using the sample to select a split value, i.e., take median of the sample points in bucket range
  - Keeps counts within a factor of 2: for more equal buckets, can recompute from the sample

---

**I-D Histograms: Compressed**

- Create singleton buckets for largest values, equi-depth over the rest
- Improvement over equi-depth since get exact info on largest values, e.g., join estimation in DB2 compares largest values in the relations

**Construction:** Sorting + O(B log B) + one pass; can use sample

**Maintenance:** Split & Merge approach as with equi-depth, but must also decide when to create and remove singleton buckets [SGM97b]
1-D Histograms: V-Optimal

[IP95] defined V-optimal & showed it minimizes the average selectivity estimation error for equality-join & selections.

Idea: Select buckets to minimize frequency variance within buckets.

- [JKM98] gave an O(\(N^2\)) time dynamic programming algorithm
  - \(F(k)\) is freq. of value \(k\); \(AVGF(k)\) = avg freq for values \(k\).
  - \(SSE(k) = \sum_{i=k+1}^{n} F(i)^2 - (n+1)/N \cdot AVGF(k)\)^2
  - For \(i=1..N\), compute \(F(i) = \sum_{k=1..i} F(k)\) & \(F(i) = \sum_{k=1..i} F(k)^2\)
  - Then can compute any \(SSE(i)\) in constant time.
  - Let \(SSEP(k) = \min SSE\) for \(F(1..k)\) using \(k\) buckets.
  - Then \(SSE(k) = \min_{i=1..k} (SSEP(k) + SSE(k+1..N))\), i.e.,
suffices to consider all possible left boundaries for \(i\)th bucket.
  - Also gave faster approximation algorithms.

Answering Queries: Equi-Depth

Answering queries:
- select count(*) from \(R \) where \(R.A = 15\)
- approximate answer: \(F \cdot |R|/B\), where
  - \(F\) = number of buckets, including fractions, that overlap the range
  - error guarantee: \(\pm 2 \cdot |R|/B\)

Answering Queries: Histograms

- Answering queries from 1-D histograms (in general):
  - (Implicitly) map the histogram back to an approximate relation,
    & apply the query to the approximate relation.

  - Continuous value mapping [SAC79]:

  - Uniform spread mapping [PH96]:

Self-Tuning 1-D Histograms

1. Tune Bucket Frequencies:

   [AC99]

   - Compare actual selectivity to histogram estimate
   - Use to adjust bucket frequencies

   - Divide *Error proportionately, d-damping factor

2. Restructure:

   - Merge buckets of near-equal frequencies
   - Split large frequency buckets

   Also Extends to Multi-D

Sampling: Basics

Idea: A small random sample \(S\) of the data often well-represents all the data.

- For a fast approx answer, apply the query to \(S\) & "scale" the result.
- E.g., \(R.A = 0.1\), \(S\) is a 20% sample
  - select count(*) from \(R\) \& where \(R.A = 0\)
  - select 5 * count(*) from \(S\) \& where \(S.A = 0\)
  - Est. count = 5 \(* 2 \approx 10\), Exact count \(\approx 10\)

Unbiased: For expressions involving count, sum, avg: the estimator is unbiased, i.e., the expected value of the answer is the actual answer, even for (most) queries with predicates.

- Leverage extensive literature on confidence intervals for sampling
  - Actual answer is within the interval [\(a,b\)] with a given probability

E.g., 54,000 \(\pm 600\) with prob \(\approx 90\%\)
**Sampling: Confidence Intervals**

<table>
<thead>
<tr>
<th>Method</th>
<th>90% Confidence Interval (σ)</th>
<th>Guaranteed?</th>
</tr>
</thead>
</table>
| Central Limit Theorem   | 1.64 * σ(R)/√|σ(R)|[5]| at 1% = 5%
| Haefling                | 1.22 * MAX(MIN / apr1([5])) always |
| Cholesky (known σ(R))   | 1.16 * σ(R)/sqrt(|σ(R)|) always |
| Cholesky (estimated σ(R)) | 1.16 * σ(R)/sqrt(|σ(R)|) at 1% = 5%

Confidence intervals for Average: select avg(R.A) from R

- (Can replace R.A with any arithmetic expression on the attributes in R)
- σ(R) = standard deviation of the values of R.A; σ(R) = s.d. for S.A
- If predicates, 5 above is subset of sample that satisfies the predicate
- Quality of the estimate depends only on the variance in R & [5] after the predicate: So 10K sample may suffice for 10B row relations
- Advantage of larger samplers: can handle more selective predicates

**Sampling from Databases**

- Sampling disk-resident data is slow
  - Row-level sampling has high I/O cost:
    - must bring in entire disk block to get the row
  - Block-level sampling: rows may be highly correlated
  - Random access pattern: possibly via an index
  - Need acceptance/rejection sampling to account for the variable number of rows in a page, children in an index node, etc.

- Alternatives
  - Random physical clustering: destroys "natural" clustering
  - Precomputed samples: must incrementally maintain (at specified size)
    - Fast to use: packed in disk blocks, can sequentially scan, can store as relation and leverage full DBMS query support, can store in main memory

**One-Pass Uniform Sampling**

- Best choice for incremental maintenance
  - Low overheads, no random data access

- Reservoir Sampling [Vit85] Maintains a sample S of a fixed-size M
  - Add each new item to S with probability M/N, where N is the current number of data items
  - If add an item, evict a random item from S
  - Instead of flipping a coin for each item, determine the number of items to skip before the next to be added to S
  - To handle deletions, permit [S] to drop to L + M, e.g., L = M/2
    - remove from S if deleted item is in S, else ignore
    - If [S] = M/2, get a new S using another pass (happens only if delete roughly half the items & cost is fully amortized) [68P97b]

**Biased Sampling**

- Often, advantageous to sample different data at different rates (Stratified Sampling)
  - E.g., outliers can be sampled at a higher rate to ensure they are accounted for; better accuracy for small groups in group-by queries
  - Each tuple j in the relation is selected for the sample S with some probability Pj (can depend on values in tuple j)
  - If selected, it is added to S along with its scale factor sf = 1/Pj

  - Answering queries from S: e.g.,
    - select sum(R.a) from R where R.b < 5
    - select sum(S.a * S.sf) from S where S.b < 5
  - Unbiased answer: Good choice for Pj’s results in tighter confidence intervals