Size estimation; Partitioning

Rasmus Pagh

Core problem: Size estimation

- The sizes of intermediate results are important for the choices made when planning query execution.
- Time for operations grow (at least) linearly with size of (largest) argument. (Note that we do not have indexes for intermediate results.)
- The total size can even be used as a crude estimate on the running time.

Classical approach: Heuristics

- In the book a number of heuristics for estimating sizes of intermediate results are presented.
- This classical approach works well in some cases, but is unreliable in general.
- The modern approach is based on maintaining suitable *statistics* summarizing the data. (Focus of lecture.)

Some possible types of statistics

- Random sample of, say, 1% of the tuples. (NB. Should fit main memory.)
- The 1000 most frequent values of some attribute, with tuple counts.
- Histogram with number of values in different ranges.
- Last 10-12 years: ”Sketches”.

On-line vs off-line statistics

- **Off-line**: Statistics only computed periodically, often operator-controlled (e.g. Oracle). Typically involves sorting data according to each attribute.
- **On-line**: Statistics maintained automatically at all times by the DBMS. Focus of this lecture.

Maintaining a random sample

- To get a sample of expected size 1% of full relation:
  - Add a new tuple to the sample with probability 1%.
  - If a sampled tuple is deleted or updated, remember to remove from or update in sample.

Estimating selects

- To estimate the size of a select statement $\sigma_C(R)$:
  - Compute $|\sigma_C(R')|$, where R' is the random sample of R.
  - If the sample is 1% of R, the estimate is 100 $|\sigma_C(R')|$, etc.
  - The estimate is reliable if $|\sigma_C(R')|$ is not too small (the bigger, the better).

Sampling using a hash function

- Basic idea:
  - Want sampling decision based on the value of a particular attribute A.
  - Use a hash function $h : U \rightarrow \{0, \ldots, 99\}$ and sample the tuples with hash value 0 on the value of A.
  - Use same hash function for all relations – the sampling is *dependent*.

Estimating join sizes?

- Suppose you want to estimate the size of a join statement $R_1 \bowtie R_2$.
- You have random samples of 1% of each relation. Two cases:
  - Independent sampling.
  - Sampling using a hash function on join attr.
- **Question**: How do you do the estimation?

Estimating join sizes I

- Compute $|R'_1 \bowtie R'_2|$, where R_1' and R_2' are *independent* samples of R_1 and R_2.
- If samples are 1% of the relations, estimate is

$$100^2|R'_1 \bowtie R'_2|$$

Estimating join sizes II

- Compute $|R_1' \bowtie R_2'|$, where R_1' and R_2' are *hashing based* samples of R_1 and R_2 on the join attribute.
- If samples are 1% of the relations, estimate is $100|R_1' \bowtie R_2'|$
  - less chance of a zero estimate,
  - but the *variance* may be large.
- Notice that for this purpose we do not need to store R_1' and R_2' – it suffices to store the frequency of each item of attribute A.

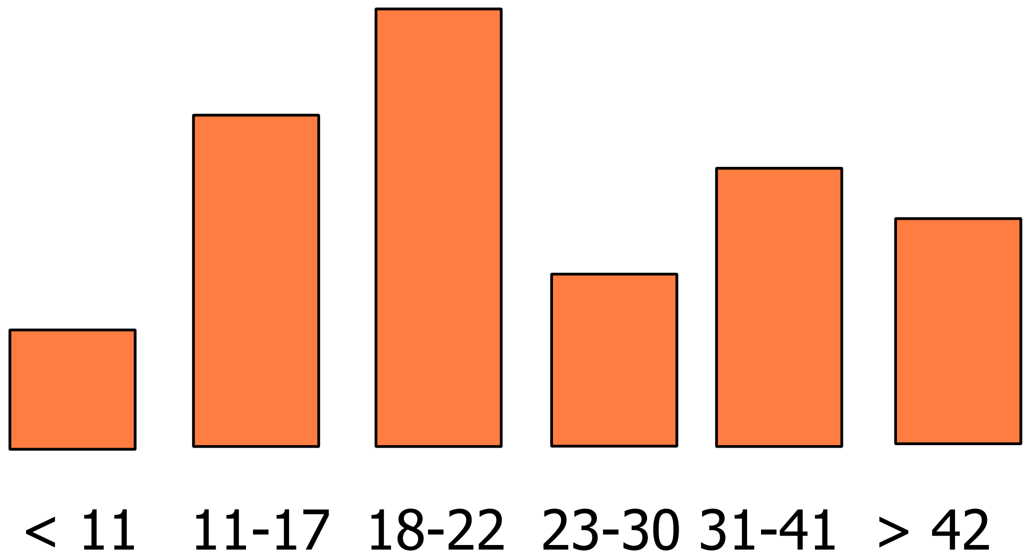
Keeping a sample of bounded size

<u>Reservoir sampling (Vitter '85):</u>

- Initial sample consists of s tuples.
- A tuple inserted in R is stored in sample with probability s/(|R|+1).
- When storing a new tuple, it replaces a randomly chosen tuple in the existing sample (unless sample has size < s due to a deletion).

Histogram

- Number of values/tuples in each of a number of intervals. Widely used.



- Question: How do you use a histogram to estimate selectivity?

Sketch-based estimation

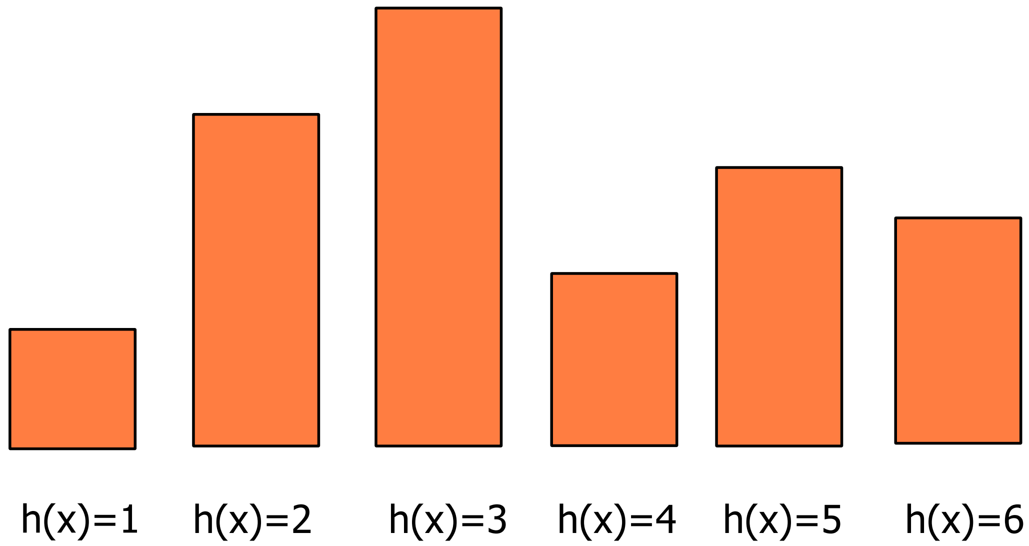
- Idea:
  Maintain some information about the relations (a ”sketch”) that:
  - Is much smaller than the size of the relations themselves. (Smaller than a useful sample.)
  - Can easily be updated when tuples are inserted and deleted.
  - Allows accurate prediction of sizes of subresults (key: joins, selections).

Aside: Approximate answers

- In some applications, we may be happy with an approximation of an aggregate, say, and need only access the sketch.
- In some applications (e.g. data streams) we must accept some inaccuracy do be able to get answers.
- Not focus here.

Random histograms

- A basic technique used in sketching is a histogram where the column for each key is chosen at random, using a hash function.



Experiment

Let's make a random histogram of the class ages (5 columns).

What can we say about:

- `Select … year=1975`?
- Self-join on year?

x	h(x)	x	h(x)
1960	0	1975	0
1961	2	1976	2
1962	4	1977	4
1963	1	1978	1
1964	3	1979	3
1965	0	1980	0
1966	2	1981	2
1967	4	1982	4
1968	1	1983	1
1969	3	1984	3
1970	0	1985	0
1971	2	1986	2
1972	4	1987	4
1973	1	1988	1
1974	3	1989	3

Count-min sketches (CM'05)

- Sketch of an attribute A of relation R.
- The sketch consists of k independent, random histograms $X_i[1..n]$. , using hash functions $h_1,...,h_k$.
- An (over)estimate of $|\sigma_{A=y}(R)|$ is $\min_i(X_i[h_i(y)])$.
- The estimate is not too far off if most of the tuples have values from a small set (size n, say) - i.e. there is high *skew*. (See paper RD07.)

Unbiased sketches

- Suppose we want an estimate whose expected value is $|\sigma_{A=y}(R)|$.
- Fast-Count idea (Thorup-Zhang '04):
  - Subtract the expected "noise" from $X_i[h_i(y)]$.
  - To reduce error, take mean for i=1,...,k.
- Alternative (Fast-AGMS '96, '05):
  - Consider the *difference* of two values from the random histogram.
  - By symmetry, they have the same expected noise, and the result is unbiased!
- In both cases, we get a guarantee (see RD07): Estimate is "close" with "good" probability, depending on how much space is used.

Join size from sketches

- For Fast-Count, Fast-AGMS the ”inner product” of the (unbiased) estimator vectors is an estimator for the join size:

  $(-3,2,0,4)\cdot(-2,-1,3,2) =$
  $-3\cdot(-2)+2\cdot(-1)+0\cdot 3+4\cdot 2 = 12$

- This estimator has large variance. We can reduce variance by taking the *average* of many estimators.
- Sometimes, a *median* of averages gives better (provable) guarantees.

And now a warmup on…

partitioning!

Motivating examples

- *Your company maintains a database of 20 years of business transactions, but performance is only important for the last year's data.*
- *A relation has a ”country” attribute, and most queries are about one particular country (not the same every time).*
- Could of course sort by date and country, respectively. But there is a more ”lightweight” solution.

Motivating examples, cont.

- *An important query uses a full table scan, but need only two attributes out of 20.*
  - Too expensive to maintain a covering index.
  - And no need for sorted order.
- *A few attributes of a table are rarely accessed.*
  - Make full table scans slower.
  - Buffer will contain less useful data.

Partitioning

- If many queries need just a ”predictable” subset of the rows and/or a subset of the columns, it may be a good idea (especially if there are full table scans) to:
  - partition horisontally (by row) and/or
  - partition vertically (by column).
- Each partition is stored similarly to a normal table, including indexes.
- Some DBMSs have features that make the partitioning *transparent*.

Partitioning in Oracle (not XE)

- Only horisontal partitioning.
  - Can do vertical partitioning via foreign keys.
- Range and hash partitioning:

```
CREATE TABLE sales
    ( invoice_no NUMBER,
      sale_year  INT NOT NULL,
      sale_month INT NOT NULL,
      sale_day   INT NOT NULL )
PARTITION BY RANGE (sale_year, sale_month, sale_day)
    ( PARTITION sales_q1 VALUES LESS THAN (1999, 04, 01),
      PARTITION sales_q2 VALUES LESS THAN (1999, 07, 01),
      PARTITION sales_q3 VALUES LESS THAN (1999, 10, 01),
      PARTITION sales_q4 VALUES LESS THAN (2000, 01, 01) );

CREATE TABLE scubagear
     (id NUMBER,
      name VARCHAR2 (60))
PARTITION BY HASH (id)
   PARTITIONS 4;
```

Partitioning in Oracle

List partitioning:

```
CREATE TABLE q1_sales_by_region
   (deptno number,
    deptname varchar2(20),
    quarterly_sales number(10, 2),
    state varchar2(2))
PARTITION BY LIST (state)
   (PARTITION q1_northwest VALUES ('OR', 'WA'),
    PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
    PARTITION q1_northeast VALUES  ('NY', 'VM', 'NJ'),
    PARTITION q1_southeast VALUES ('FL', 'GA'),
    PARTITION q1_northcentral VALUES ('SD', 'WI'),
    PARTITION q1_southcentral VALUES ('OK', 'TX'));
```

Discussion of partitioning

- Vertical partitioning means that retrieving or inserting a tuple requires more I/Os.
- Horizontal partitioning allows faster updates than buffered indexes **if** the database buffer is large enough to keep the ”active” block of each partition.
- Maintaining a suitable list or range partitioning is tedious…

This afternoon

- Guest lecture by Kaare Jelling Kristoffersen, LECTOR ApS talks about:

Partitioning and Green IT