UNASSSUMING VIEW-SIZE ESTIMATION TECHNIQUES IN OLAP
An Experimental Comparison

Kamel Aouiche and Daniel Lemire
LICEF, University of Quebec at Montreal, 100 Sherbrooke West, Montreal, Canada
kamel.aouiche@gmail.com, lemire@acm.org

Keywords: probabilistic estimation, skewed distributions, sampling, hashing

Abstract: Even if storage was infinite, a data warehouse could not materialize all possible views due to the running time and update requirements. Therefore, it is necessary to estimate quickly, accurately, and reliably the size of views. Many available techniques make particular statistical assumptions and their error can be quite large. Unassuming techniques exist, but typically assume we have independent hashing for which there is no known practical implementation. We adapt an unassuming estimator due to Gibbons and Tirthapura: its theoretical bounds do not make unpractical assumptions. We compare this technique experimentally with stochastic probabilistic counting, LOGLOG probabilistic counting, and multifractal statistical models. Our experiments show that we can reliably and accurately (within 10%, 19 times out 20) estimate view sizes over large data sets (1.5 GB) within minutes, using almost no memory. However, only GIBBONS-TIRTHAPURA provides universally tight estimates irrespective of the size of the view. For large views, probabilistic counting has a small edge in accuracy, whereas the competitive sampling-based method (multifractal) we tested is an order of magnitude faster but can sometimes provide poor estimates (relative error of 100%). In our tests, LOGLOG probabilistic counting is not competitive. Experimental validation on the US Census 1990 data set and on the Transaction Processing Performance (TPC H) data set is provided.

1 INTRODUCTION

View materialization is presumably one of the most effective technique to improve query performance of data warehouses. Materialized views are physical structures that improve data access time by precomputing intermediary results. Typical OLAP queries defined on data warehouses consist in selecting and aggregating data with queries such as grouping sets (GROUP BY clauses). By precomputing many plausible groupings, we can avoid aggregates over large tables. However, materializing views requires additional storage space and induces maintenance overhead when refreshing the data warehouse.

One of the most important issues in data warehouse physical design is to select an appropriate configuration of materialized views. Several heuristics and methodologies were proposed for the materialized view selection problem, which is NP-hard (**?**). Most of these techniques exploit cost models to estimate the data access cost using materialized views, their maintenance and storage cost. This cost estimation mostly depends on view-size estimation.

Several techniques have been proposed for view-size estimation: some requiring assumptions about the data distribution and others that are "unassuming." A common statistical assumption is uniformity (**?**), but any skew in the data leads to an overestimate of the size of the view. Generally, while statistically assuming estimators are computed quickly, the most expensive step being the random sampling, their error can be large and it cannot be bounded a priori.

In this paper, we consider several state-of-the-art statistically unassuming estimation techniques: GIBBONS-TIRTHAPURA (**?**), probabilistic counting (**?**), and LOGLOG probabilistic counting (**?**). While relatively expensive, unassuming estimators tend to provide a good accuracy. To our knowledge,

this is the first experimental comparisons of unassuming view-size estimation techniques in a data warehousing setting.

2 RELATED WORK

Haas et al. (**?**) estimate the view-size from the histogram of a sample: adaptively, they choose a different estimator based on the skew of the distribution. Faloutsos et al. (**?**) obtain results nearly as accurate as Haas et al., that is, an error of approximately 40%, but they only need the dominant mode of the histogram, the number of distinct elements in the sample, and the total number of elements. In sample-based estimations, in the worst-case scenario, the histogram might be as large as the view size we are trying to estimate. Moreover, it is difficult to derive unassuming accuracy bounds since the sample might not be representative. However, a sample-based algorithm is expected to be an order of magnitude faster than an algorithm which processes the entire data set.

Probabilistic counting (**?**) and LOGLOG probabilistic counting (henceforth LOGLOG) (**?**) have been shown to provide very accurate unassuming view-size estimations quickly, but their estimates assume we have independent hashing. Because of this assumption, their theoretical bound may not hold in practice. Whether this is a problem in practice is one of the contribution of this paper.

Gibbons and Tirthapura (**?**) derived an unassuming bound (henceforth GIBBONS-TIRTHAPURA) that only requires pairwise independent hashing. It has been shown recently that if you have k-wise independent hashing for $k > 2$ the theoretically bound can be improved substantially (**?**). The benefit of GIBBONS-TIRTHAPURA is that as long as the random number generator is truly random, the theoretical bounds have to hold irrespective of the size of the view or of other factors.

All unassuming estimation techniques in this paper (LOGLOG, probabilistic counting and GIBBONS-TIRTHAPURA), have an accuracy proportional to $1/\sqrt{M}$ where M is a parameter noting the memory usage.

3 ESTIMATION BY MULTIFRACTALS

We implemented the statistically assuming algorithm by Faloutsos et al. based on a multifractal model (**?**). Nadeau and Teorey (**?**) reported competitive results for this approach. Maybe surprisingly, given a sample, all that is required to learn the multifractal model is the number of distinct elements in the sample F_0, the number of elements in the sample N', the total number of elements N, and the number of occurrences of the most frequent item in the sample $m_{\max}$. Hence, a very simple implementation is possible (see Algorithm **??**). Faloutsos et al. erroneously introduced a tolerance factor ε in their algorithm: unlike what they suggest, it is not possible, unfortunately, to adjust the model parameter for an arbitrary good fit, but instead, we have to be content with the best possible fit (see line 9 and following).

Algorithm 1 View-size estimation using a multifractal distribution model.

1: **INPUT:** Fact table t containing N facts
2: **INPUT:** GROUP BY query on dimensions $D_1, D_2, \ldots, D_d$
3: **INPUT:** Sampling ratio $0 < p < 1$
4: **OUTPUT:** Estimated size of GROUP BY query
5: Choose a sample in t' of size $N' = \lfloor pN \rfloor$
6: Compute g=GROUP BY(t')
7: let $m_{\max}$ be the number of occurrences of the most frequent tuple $x_1, \ldots, x_d$ in g
8: let F_0 be the number of tuples in g
9: $k \leftarrow \lceil \log F_0 \rceil$
10: **while** $F < F_0$ **do**
11: $\quad p \leftarrow (m_{\max}/N')^{1/k}$
12: $\quad F \leftarrow \sum_{a=0}^{k} \binom{k}{a} (1 - (p^{k-a}(1-p)^a)^{N'})$
13: $\quad k \leftarrow k+1$
14: $p \leftarrow (m_{\max}/N)^{1/k}$
15: **RETURN:** $\sum_{a=0}^{k} \binom{k}{a} (1 - (p^{k-a}(1-p)^a)^{N})$

4 UNASSUMING VIEW-SIZE ESTIMATION

4.1 Independent Hashing

Hashing maps objects to values in a nearly random way. It has been used for efficient data structures such as hash tables and in cryptography. We are interested in hashing functions from tuples to $[0, 2^L)$ where L is fixed ($L = 32$ in this paper). Hashing is uniform if $P(h(x) = y) = 1/2^L$ for all x, y, that is, if all hashed values are equally likely. Hashing is *pairwise independent* if $P(h(x_1) = y \wedge h(x_2) = z) = P(h(x_1) = y)P(h(x_2) = z) = 1/4^L$ for all x_1, x_2, y, z. Pairwise independence implies uniformity. Hashing is k-wise independent if $P(h(x_1) = y_1 \wedge \cdots \wedge h(x_k) = y_k) = 1/2^{kL}$ for all x_i, y_i. Finally, hashing is (fully) independent if it is k-wise independent for all k. It is believed that

independent hashing is unlikely to be possible over large data sets using a small amount of memory (**?**).

Next, we show how k-wise independent hashing is easily achieved in a multidimensional data warehousing setting. For each dimension D_i, we build a lookup table T_i, using the attribute values of D_i as keys. Each time we meet a new key, we generate a random number in $[0,2L)$ and store it in the lookup table T_i. This random number is the hashed value of this key. This table generates (fully) independent hash values in amortized constant time. In a data warehousing context, whereas dimensions are numerous, each dimension will typically have few distinct values: for example, there are only 8,760 hours in a year. Therefore, the lookup table will often use a few Mib or less. When hashing a tuple $x_1, x_2, \ldots, x_k$ in $D_1 \times D_2 \times \ldots D_k$, we use the value $T_1(x_1) \oplus T_2(x_2) \oplus \cdots \oplus T_k(x_k)$ where $\oplus$ is the EXCLUSIVE OR operator. This hashing is k-wise independent and requires amortized constant time. Tables T_i can be reused for several estimations.

4.2 Probabilistic Counting

Our implementation of (stochastic) probabilistic counting (**?**) is given in Algorithm **??**. Recently, a variant of this algorithm, LOGLOG, was proposed (**?**). Assuming independent hashing, these algorithms have standard error (or the standard deviation of the error) of $0.78/\sqrt{M}$ and $1.3/\sqrt{M}$ respectively. These theoretical results assume independent hashing which we cannot realistically provide. Thus, we do not expect these theoretical results to be always reliable.

Algorithm 2 View-size estimation using (stochastic) probabilistic counting.

INPUT: Fact table t containing N facts
INPUT: GROUP BY query on dimensions $D_1, D_2, \ldots, D_d$
INPUT: Memory budget parameter $M = 2^k$
INPUT: Independent hash function h from d tuples to $[0, 2^L)$.
OUTPUT: Estimated size of GROUP BY query
$b \leftarrow M \times L$ matrix (initialized at zero)
for tuple $x \in t$ **do**
  $x' \leftarrow \pi_{D_1,D_2,\ldots,D_d}(x)$ {projection of the tuple}
  $y \leftarrow h(x')$ {hash x' to $[0, 2^L)$}
  $\alpha = y \bmod M$
  $i \leftarrow$ position of the first 1-bit in $\lfloor y/M \rfloor$
  $b_{\alpha,i} \leftarrow 1$
$A \leftarrow 0$
for $\alpha \in \{0, 1, \ldots, M-1\}$ **do**
  increment A by the position of the first zero-bit in $b_{\alpha,0}, b_{\alpha,1}, \ldots$
RETURN: $M/\phi 2^{A/M}$ where $\phi \approx 0.77351$

Algorithm 3 View-size estimation using LOGLOG.

INPUT: fact table t containing N facts
INPUT: GROUP BY query on dimensions $D_1, D_2, \ldots, D_d$
INPUT: Memory budget parameter $M = 2^k$
INPUT: Independent hash function h from d tuples to $[0, 2^L)$.
OUTPUT: Estimated size of GROUP BY query
$\mathcal{M} \leftarrow \underbrace{0, 0, \ldots, 0}_{M}$
for tuple $x \in t$ **do**
  $x' \leftarrow \pi_{D_1,D_2,\ldots,D_d}(x)$ {projection of the tuple}
  $y \leftarrow h(x')$ {hash x' to $[0, 2^L)$}
  $j \leftarrow$ value of the first k bits of y in base 2
  $z \leftarrow$ position of the first 1-bit in the remaining $L-k$ bits of y (count starts at 1)
  $\mathcal{M}_j \leftarrow \max(\mathcal{M}_j, z)$
RETURN: $\alpha_M M 2^{\frac{1}{M}\sum_j \mathcal{M}_j}$ where $\alpha_M \approx 0.39701 - (2\pi^2 + \ln^2 2)/(48M)$.

4.3 GIBBONS-TIRTHAPURA

Our implementation of the GIBBONS-TIRTHAPURA algorithm (see Algorithm **??**) hashes each tuple only once unlike the original algorithm (**?**). Moreover, the independence of the hashing depends on the number of dimensions used by the GROUP BY. If the view-size is smaller than the memory parameter (M), the view-size estimation is without error. For this reason, we expect GIBBONS-TIRTHAPURA to perform well when estimating small and moderate view sizes.

Algorithm 4 GIBBONS-TIRTHAPURA view-size estimation.

INPUT: Fact table t containing N facts
INPUT: GROUP BY query on dimensions $D_1, D_2, \ldots, D_d$
INPUT: Memory budget parameter M
INPUT: k-wise hash function h from d tuples to $[0, 2^L)$.
OUTPUT: Estimated size of GROUP BY query
$\mathcal{M} \leftarrow$ empty lookup table
$t \leftarrow 0$
for tuple $x \in t$ **do**
  $x' \leftarrow \pi_{D_1,D_2,\ldots,D_d}(x)$ {projection of the tuple}
  $y \leftarrow h(x')$ {hash x' to $[0, 2^L)$}
  $j \leftarrow$ position of the first 1-bit i y (count starts at 0)
  if $j \leq t$ **then**
    $\mathcal{M}_{x'} = j$
    while size$(\mathcal{M}) > M$ **do**
      $t \leftarrow t + 1$
      prune all entries in $\mathcal{M}$ having value less than t
RETURN:.2^t size$(\mathcal{M})$

The theoretical bounds given in (**?**) assumed pairwise independence. The generalization below is from (**?**) and is illustrated by Figure **??**.

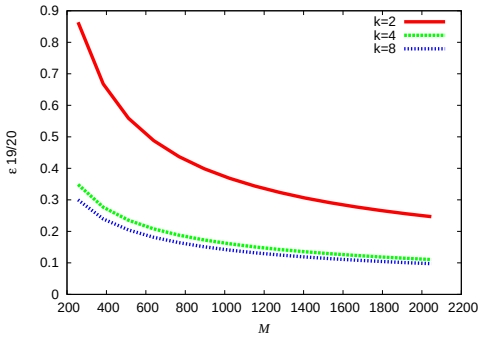


Figure 1: Bound on the estimation error (19 times out of 20) as a function of the number of tuples kept in memory ($M \in [128, 2048]$) according to Proposition **??** for GIBBONS-TIRTHAPURA view-size estimation with k-wise independent hashing.

Proposition 1 *Algorithm* **??** *estimates the number of distinct tuples within relative precision* ε*, with a* k*-wise independent hash for* $k \geq 2$ *by storing* M *distinct tuples* ($M \geq 8k$) *and with reliability* $1-\delta$ *where* δ *is given by*

$$\delta = \frac{k^{k/2}}{e^{k/3}M^{k/2}}\left(2^{k/2} + \frac{8^{k/2}}{\varepsilon^k(2^{k/2}-1)}\right).$$

More generally, we have

$$\delta \leq \frac{k^{k/2}}{e^{k/3}M^{k/2}}\left(\frac{\alpha^{k/2}}{(1-\alpha)^k} + \frac{4^{k/2}}{\alpha^{k/2}\varepsilon^k(2^{k/2}-1)}\right).$$

for $4k/M \leq \alpha < 1$ *and any* $k, M > 0$.

In the case where hashing is 4-wise independent, as in some of experiments below, we derive a more concise bound.

Corollary 1 *With 4-wise independent hashing, Algorithm* **??** *estimates the number of distinct tuples within relative precision* $\varepsilon \approx 5/\sqrt{M}$*, 19 times out of 20 for* ε *small.*

Proof. We start from the second inequality of Proposition **??**. Differentiating $\frac{\alpha^{k/2}}{(1-\alpha)^k} + \frac{4^{k/2}}{\alpha^{k/2}\varepsilon^k(2^{k/2}-1)}$ with respect to α and setting the result to zero, we get $3\alpha^4\varepsilon^4 + 16\alpha^3 - 48\alpha^2 - 16 = 0$ (recall that $4k/M \leq \alpha < 1$). By multiscale analysis, we seek a solution of the form $\alpha = 1 - a\varepsilon^r + o(\varepsilon^r)$ and we have that $\alpha \approx 1 - 1/2\sqrt[3]{3/2}\varepsilon^{4/3}$ for ε small. Substituting this value of α, we have $\frac{\alpha^{k/2}}{(1-\alpha)^k} + \frac{4^{k/2}}{\alpha^{k/2}\varepsilon^k(2^{k/2}-1)} \approx 128/24\varepsilon^4$. The result follows by substituting in the second inequality. □

	US Census 1990	**DBGEN**
# of facts	2458285	13977981
# of views	20	8
# of attributes	69	16
Data size	360 MB	1.5 GB

Table 1: Characteristic of data sets.

5 EXPERIMENTAL RESULTS

To benchmark the quality of the view-size estimation against the memory and speed, we have run test over the US Census 1990 data set (**?**) as well as on synthetic data produced by DBGEN (**?**). The synthetic data was produced by running the DBGEN application with scale factor parameter equal to 2. The characteristics of data sets are detailed in Table **??**. We selected 20 and 8 views respectively from these data sets: all views in US Census 1990 have at least 4 dimensions whereas only 2 views have at least 4 dimensions in the synthetic data set.

We used the GNU C++ compiler version 4.0.2 with the "-O2" optimization flag on a Centrino Duo 1.83 GHz machine with 2 GB of RAM running Linux kernel 2.6.13–15. No thrashing was observed. To ensure reproducibility, C++ source code is available freely from the authors.

For the US Census 1990 data set, the hashing look-up table is a simple array since there are always fewer than 100 attribute values per dimension. Otherwise, for the synthetic DBGEN data, we used the GNU/CGI STL extension `hash_map` which is to be integrated in the C++ standard as an `unordered_map`: it provides amortized $O(1)$ inserts and queries. All other look-up tables are implemented using the STL `map` template which has the same performance characteristics of a red-black tree. We used comma separated (CSV) (and pipe separated files for DBGEN) text files and wrote our own C++ parsing code.

The test protocol we adopted (see Algorithm **??**) has been executed for each estimation technique (LOGLOG, probabilistic counting and GIBBONS-TIRTHAPURA), GROUP BY query, random seed and memory size. At each step corresponding to those parameter values, we compute the estimated-size values of GROUP BYs and time required for their computation. For the multifractal estimation technique, we computed at the same way the time and estimated size for each GROUP BY, sampling ratio value and random seed.

US Census 1990. Figure **??** plots the largest 95$^{\text{th}}$-percentile error observed over 20 test estimations for various memory size $M \in \{16, 64, 256, 2048\}$.

Algorithm 5 Test protocol.

```
1: for GROUP BY query q ∈ Q do
2:   for memory budget m ∈ M do
3:     for random seed value r ∈ R do
4:       Estimate the size of GROUP BY q with m memory budget and r random seed value
5:       Save estimation results (time and estimated size) in a log file
```

For the multifractal estimation technique, we represent the error for each sampling ratio $p \in \{0.1\%, 0.3\%, 0.5\%, 0.7\%\}$. The X axis represents the size of the exact GROUP BY values. This 95^{th}-percentile error can be related to the theoretical bound for ε with 19/20 reliability for GIBBONS-TIRTHAPURA (see Corollary **??**): we see that this upper bound is verified experimentally. However, the error on "small" view sizes can exceed 100% for probabilistic counting and LOGLOG.

Synthetic data set. Similarly, we computed the 19/20 error for each technique, computed from the DDBGEN data set . We observed that the four techniques have the same behaviour observed on the US Census data set. Only, this time, the theoretical bound for the 19/20 error is larger because the synthetic data sets has many views with less than 2 dimensions.

Speed. We have also computed the time needed for each technique to estimate view-sizes. We do not represent this time because it is similar for each technique except for the multifractal which is the fastest one. In addition, we observed that time do not depend on the memory budget because most time is spent streaming and hashing the data. For the multifractal technique, the processing time increases with the sampling ratio.

The time needed to estimate the size of all the views by GIBBONS-TIRTHAPURA, probabilistic counting and LOGLOG is about 5 minutes for US Census 1990 data set and 7 minutes for the synthetic data set. For the multifractal technique, all the estimates are done on roughly 2 seconds. This time does not include the time needed for sampling data which can be significant: it takes 1 minute (resp. 4 minutes) to sample 0.5% of the US Census data set (resp. the synthetic data set – TPC H) because the data is not stored in a flat file.

6 DISCUSSION

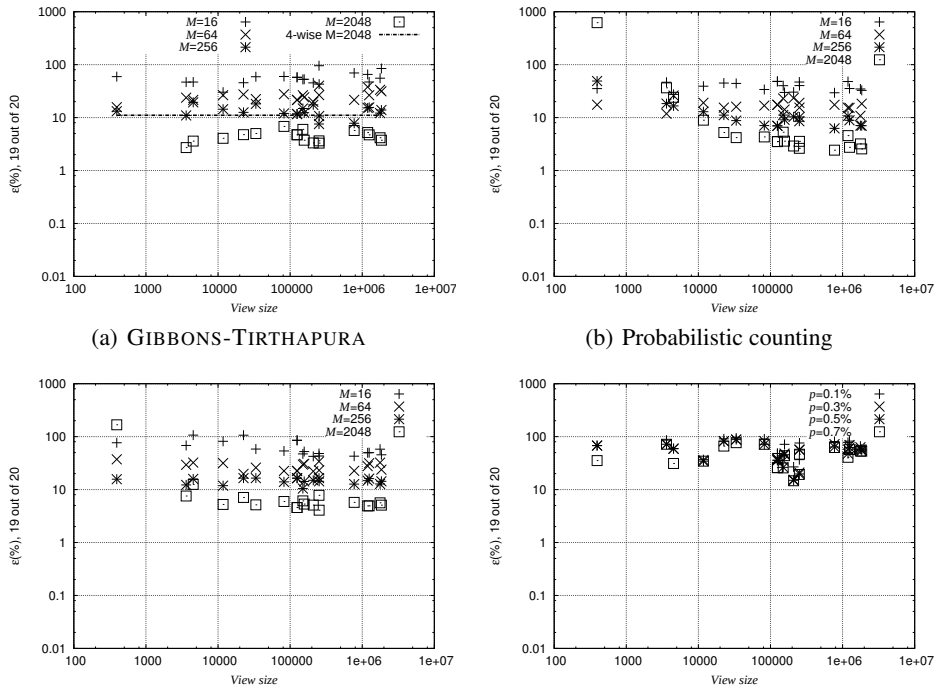
Our results show that probabilistic counting and LOGLOG do not entirely live up to their theoretical promise. For small view sizes, the relative accuracy can be very low.

When comparing the memory usage of the various techniques, we have to keep in mind that the memory parameter M can translate in different memory usage. The memory usage depends also on the number of dimensions of each view. Generally, GIBBONS-TIRTHAPURA will use more memory for the same value of M than either probabilistic counting or LOGLOG, though all of these can be small compared to the memory usage of the lookup tables T_i used for k-wise independent hashing. In this paper, the memory usage was always of the order of a few MiB which is negligible in a data warehousing context.

View-size estimation by sampling can take minutes when data is not layed out in a flat file or indexed, but the time required for an unassuming estimation is even higher. Streaming and hashing the tuples accounts for most of the processing time so for faster estimates, we could store all hashed values in a bitmap (one per dimension).

7 CONCLUSION AND FUTURE WORK

In this paper, we have provided unassuming techniques for view-size estimation in a data warehousing context. We adapted an estimator due to Gibbons and Tirthapura. We compared this technique experimentally with stochastic probabilistic counting, LOGLOG, and multifractal statistical models. We have demonstrated that among these techniques, only GIBBONS-TIRTHAPURA provides stable estimates irrespective of the size of views. Otherwise, (stochastic) probabilistic counting has a small edge in accuracy for relatively large views, whereas the competitive sampling-based technique (multifractal) is an order of magnitude faster but can provide crude estimates. According to our experiments, LOGLOG was not faster than either GIBBONS-TIRTHAPURA or probabilistic counting, and since it is less accurate than probabilistic counting, we cannot recommend it. There is ample room for future work. Firstly, we plan to extend these techniques to other types of aggregated views (for example, views including HAVING clauses). Secondly, we want to precompute the hashed values for very fast view-size estimation. Furthermore, these techniques should be tested in a materialized view selection heuristic.



(a) GIBBONS-TIRTHAPURA (b) Probabilistic counting

(c) LOGLOG (d) Multifractal

Figure 2: 95th-percentile error 19/20 ε as a function of exact view size for increasing values of M (US Census 1990)

ACKNOWLEDGEMENTS

The authors wish to thank Owen Kaser for hardware and software. This work is supported by NSERC grant 261437 and by FQRNT grant 112381.