# Estimating the Output Cardinality of Partial Preaggregation with a Measure of Clusteredness

Sven Helmer, Thomas Neumann, Guido Moerkotte
helmer|tneumann|moer@pi3.informatik.uni-mannheim.de

Fakultät für Mathematik und Informatik,
University of Mannheim, Germany

## Abstract

We introduce a new parameter, the clusteredness of data, and show how it can be used for estimating the output cardinality of a partial preaggregation operator. This provides the query optimizer with an important piece of information for deciding whether the application of partial preaggregation is beneficial. Experimental results are very promising, due to the high accuracy of the cardinality estimation based on our measure of clusteredness.

## 1 Introduction

A query optimizer has the option of introducing partial preaggregation (PPA) into a query plan containing a grouping with an aggregation. The motivation is to improve the performance by reducing the amount of data very early during query execution. Partial preaggregation means delaying the complete aggregation and partially aggregating the data in a first step.

As long as there is sufficient buffer space left, we do a regular aggregation in main memory. Once we run out of buffer space, we have to change the procedure. A regular (full) aggregation algorithm will start swapping out group records to disk to be able to continue the aggregation in main memory. Usually, the group records written to disk are partitioned so that records which belong to the same group will end up in the same partition. The final aggregation is done by scanning the partitions and aggregating the group records they contain (now the aggregation result of a partition should fit into the buffer). When aggregating partially, on the other hand, we do not write group records to disk while clearing space in the buffer, but instead send the records to the next operator in the operator tree of the query. The main advantage of this approach is a reduction in data volume while avoiding costly disk I/O. With this method, however, the aggregation may only be partial, i.e., it may contain more than one record per group. So we will have to perform a complete aggregation toward the end of the query evaluation. Ideally, we expect to benefit from the additional work of the early partial aggregation, due to a significant reduction in data volume. Unfortunately, this is not always the case. The overhead of PPA only pays off if its output (the number of records per group) is small enough. The smaller the ratio of output to input, the quicker the query evaluation, because the less data has to be handled. As the query optimizer has to decide whether to apply PPA, it needs to estimate the output cardinality accurately.

Several factors influence the quality of an early aggregation step: the number of groups, the number of tuples in the input, the distributions of the group sizes, the size of the buffer, the buffer replacement strategy used by the algorithm and, last but not least, the clusteredness of the data. While the former factors have already been investigated in some depth, the clusteredness of the



**Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003**

data has not received much attention so far. Nevertheless, it is a very important (if not the most important) parameter. When the clusteredness of the data is favorable, this parameter dominates all other negative effects (as we will see in Section 6, when looking at the experimental results). Let us illustrate this briefly by an (admittedly extreme) example. Assume a relation $R$ with the attributes $A_1, A_2, \ldots, A_r$. We want to group the tuples on the integer values of attribute $A_i$ and aggregate some other attributes. During query evaluation, the tuples of $R$ are fed into a partial preaggregation operator, for example in this order: $\langle 4,4,4,4,4,4,3,3,3,3,1,1,1,2,2\rangle$ (each number stands for a tuple, for brevity we represent each tuple by its value for $A_i$). The tuples could arrive just as well in the following order (due to the physical ordering on disk): $\langle 4,3,1,2,4,3,1,2,4,3,1,4,3,4,4\rangle$. While the second stream contains exactly the same tuples as the first one, it will be more difficult to handle. In the first case, we need just enough memory to hold a single group to do a perfect aggregation (resulting in the smallest output possible). The only difference between the two streams is the apparent clusteredness of the first stream.

This observation motivated us to include the clusteredness of data in predicting the output cardinality of PPA. We developed a measure of clusteredness adapted to the requirements of grouping and aggregation and show how the measure can be used for estimating the output cardinality of PPA. With the help of an experimental evaluation we demonstrate that the output cardinality of two different PPA strategies can be predicted very accurately.

This paper is structured as follows. In the next section we deal with the related work on aggregation. After that we briefly introduce some preliminaries. We present our measure in Section 4 and show how to estimate the output cardinality in Section 5. Section 6 contains the experimental evaluation, while Section 7 gives a conclusion and an outlook.

## 2 Related Work

The importance of data orderedness has been realized in areas apart from databases, e.g. in sorting. There are numerous publications on adaptive sorting, where the influence of orderedness of data on different sorting algorithms is measured (see [17] for a survey). The measures of disorder used in adaptive sorting are not appropriate for aggregation because they make too great demands. When aggregating, we are not interested in how sorted the data is, clustered data is perfectly fine (in the example of the introduction, the first stream was not sorted).

Also related to our work are publications dealing with the estimation of the cardinality of projections on relations [4, 8, 12, 22, 26, 27]. That work is different insofar as it tries to predict the cardinality of a deterministic operator's final result. PPA is an intermediate, nondeterministic operator (the final aggregation is done later) and its output is influenced by many other parameters, e.g. the strategy of the algorithm or the buffer size.

There has also been a renewed interest in the topic of aggregation and duplicate elimination. Recent work on PPA is covered by [10, 23, 24, 30, 31]. Up to now, however, all models assumed randomized data and did not consider clustered data. Several researchers have investigated the possibilities of improving query evaluation and optimization with regard to aggregation [11, 18, 21]. There has also been work on materializing aggregates for speeding up the processing, especially in data warehouse environments [3]. Furthermore, there are ongoing activities looking into the exchange of precision for performance [1, 9, 15]. Finally, we should mention the studies concerning the extension of standard aggregate functions [6, 7, 13, 29].

The work that comes closest to our approach is the collection of numerous publications on buffer and cache replacement strategies (see [16, 28] for an overview). A cache miss is similar to not finding a group record in the buffer of PPA. One major difference, however, is the behavior in the startup phase. A cache has many misses during the initial phase as it fills up. PPA, on the other hand, just allocates a new group record (as long as free space remains in the buffer). In this sense, the allocation should not be seen as a miss.

Apart from this difference, most of the papers on buffer and cache replacements strategies compare different algorithms experimentally and do not model the buffer hit probability. Analytical studies as in [2, 14, 25] make assumptions different from ours (e.g. different data distributions or complex modeling of hardware cache properties).

## 3 Preliminaries

Before describing and analyzing our measure and PPA, we have to introduce the terms we use throughout the paper. When talking about aggregation, we mean classical, decomposable aggregate functions as used in SQL. We group and aggregate a relation $R$ containing the tuples $t_1, t_2, \ldots, t_n$, so the cardinality of $R$, denoted by $|R|$, is $n$, where the subscripts of the tuples supply the order of appearance. By $t_i.A_j$ we mean the value of attribute $A_j$ of tuple $t_i$. The (final) aggregation will yield $m$ different groups: $G = \{g_1, g_2, \ldots, g_m\}$. The size of a group $g_i$ (i.e., the number of tuples belonging to this group) is denoted by $|g_i| = s_i$. $G$ partitions $R$, i.e., each tuple of $R$ belongs to exactly one group. Furthermore, a PPA operator will have at its disposal a buffer of size $B$, measured in the number of group records.

## 4 A Measure of Clusteredness

As seen in the introductory example, blocks of identical grouping values are very favorable for PPA. One step towards determining the clusteredness is to find the tuple positions in $R$ where the value of the grouping attribute $A_y$ changes. For this, we define a function $f$ (on the domain $1, 2, \ldots n$):

$$f(i) = \begin{cases} 1 & \text{if } t_{i-1}.A_y \neq t_i.A_y \\ 0 & \text{if } t_{i-1}.A_y = t_i.A_y \end{cases} \quad (1)$$

with $i \geq 2$ and $f(1) = 1$. Consequently, the total number of changes $c$ is:

$$c = \left( \sum_{i=1}^{n} f(i) \right) - m \quad (2)$$

We subtract $m$ at the end, since we do not want to count the first appearance of each grouping value. Thus, perfectly clustered data counts as having 0 changes.

Taking $c$ as a measure of clusteredness is not accurate enough. For example, we may have the same number of changes in two different scenarios, but a totally different behavior of PPA. In one scenario we may just alternate between two different groups (which is quite favorable from the viewpoint of PPA), while in the other we cycle between many different groups (more than the buffer can hold).

Inspired by measures in adaptive sorting based on inversions [17], we enhanced our approach. In a sequence $X = \langle x_1, x_2, \ldots, , x_n \rangle$, $(i, j)$ is an inversion if $i < j$ and $x_i > x_j$. Usually, the number of inversions or the largest distance determined by an inversion is used as a measure of disorder. As a straightforward transfer to aggregation is not possible, we settled on the distance between tuples having the same grouping value, i.e., we want to know how far back in the relation the last visited tuple with this grouping value is located. Measuring the distance in terms of positions within $R$ does not make sense, because this does not consider the clusteredness of the data in between. On account of this shortcoming, we measure the average distance in terms of changes between the occurrence of two identical grouping values.

Let $T_k = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_{s_k}} \rangle$ denote the sequence of tuples for which $A_y$ (the attribute we are grouping on) is equal to $k$ (w.l.o.g. we assume, for ease of notation, that the value of $A_y$ for group $g_k$ is equal to $k$). We now want to add up all the changes in the value of $A_y$ that occur between tuple $t_{i_2}$ and its predecessor $t_{i_1}$, between $t_{i_3}$ and $t_{i_2}$, up to $t_{s_k}$ and $t_{s_k - 1}$. With the help of $f(i)$ we can do this easily:

$$\delta_k = \sum_{j=2}^{s_k} \sum_{l=i_{j-1}+1}^{i_j} f(l) = \sum_{l=i_1+1}^{i_{s_k}} f(l) \quad (3)$$

Let us look at an example. Given the sequence $\langle 4, 1, 1, 2, 4, 3, 3, 4, 4, 2 \rangle$ and $A_y = 4$, we have $T_4 = \langle t_1, t_5, t_8, t_9 \rangle$. Between tuple $t_1$ and $t_5$ we change to two different blocks of values, between $t_5$ and $t_8$ to one block, and between $t_8$ and $t_9$ we have no changes. This means, that $\delta_4 = 3$. The values for $\delta_1, \delta_2$ and $\delta_3$ can be calculated analogously, resulting in $\delta_1 = 0, \delta_2 = 3$ and $\delta_3 = 0$.

We want to attain a measure of clusteredness $d$, which expresses the average number of changes between the appearance of two identical grouping values. The lower this number is, the more clustered the data will be (and vice versa). However, we are only interested in distances that are larger than zero changes. Tuples with identical grouping values next to each other will certainly lead to the absorption of one of them (which is ideal for PPA). Averaging over all non-zero distances, we get:

$$d = \begin{cases} \frac{\sum_{k=1}^{m} \delta_k}{c} & \text{if } c \neq 0 \\ 0 & \text{else} \end{cases} \quad (4)$$

In the case $c = 0$, we have perfect clustering and thus, all $\delta_k$ will also be equal to 0. For our example above, this means $d = 2$, since $c = 3$ (see formula (2)). This corresponds to the intuitive view. There are three non-zero distances: 1 and 2 for $A_y = 4$ and 3 for $A_y = 2$.

One could object that the query optimizer could just run PPA to find out whether it is beneficial and remember the output cardinality for later queries (as we have to traverse the relation anyway to calculate our measure). However, there is an important difference. In the case of test runs, statistics would have to be gathered for each anticipated buffer size. In our case, $c$ and $d$ are collected once and can be used for all different buffer sizes because our measure is totally independent of the buffer size used by the PPA algorithm later on.

## 5 Estimating the Output Cardinality

How do we use the measure from the last section for estimating the output size of a partial preaggregation algorithm? We analyze two different algorithms, one using a random replacement strategy and the other using an optimal strategy (allowing lookahead).

The most common strategy, LRU, is quite difficult to analyze. Developing a mathematical model for it is very complex and, up to now, one had to rely on simulation results [23]. However, the optimal strategy is a strict lower bound, whereas random replacement establishes a practical upper bound [16]. We will see later how the LRU strategy can be bounded by our estimates.

### 5.1 Random Replacement Strategy

First, we give a general description of this strategy before analyzing the cardinality of its output. The basic algorithm for partially aggregating tuples is quite simple. We step through the tuples of $R$ in their order of appearance (from $t_1$ to $t_n$). For each grouping value of $A_y$ seen so far, we keep a group record in the buffer (with a maximum capacity of $B$ group records). For each newly arriving tuple, we check if the corresponding group record is present in the buffer. If so, we combine this tuple with the data already aggregated; the tuple is *absorbed*. Otherwise, we create a new group record. This works perfectly until we run out of buffer space. At this point, we have to swap out a group record each time a tuple arrives for which no appropriate group record exists in the buffer (we make room for the newly created group record). In case of a complete aggregation operator, this can happen in several different ways, e.g. by writing runs to disk in sort-based techniques [19] or by swapping out partitions in hash-based techniques [19]. In the case of PPA, we just output the displaced group records. In this context we need a strategy to determine the records which are to be swapped out. For the random replacement strategy, we select the records to be displaced randomly.

#### 5.1.1 A First Analysis

In the following sections, we need to be able to estimate the (average) probability $P$ that the record belonging to a group is not in the buffer. A very simple approximation for $P$ is $1 - \frac{B}{m}$ (e.g. used by Larson in [23] for uniformly distributed data). However, this neglects the fact that in almost all realistic cases the cardinalities of the groups will vary widely. Therefore, we use the following estimation (for a derivation see Appendix A):

$$P \approx \frac{\sum_{i=1}^{m} \prod_{j=0}^{B-1} \left(1 - \frac{s_i}{n - j \cdot \hat{s}_i}\right)}{m} \quad (5)$$

with $\hat{s}_i = \frac{n - s_i}{m - 1}$ (and $s_i$ the number of tuples in group $i$).

Let us now analyze the output cardinality. We assume that the buffer is already filled with group records and that we have seen each group at least once (we will soon relax these constraints). In Section 4, we have determined the total number of changes $c$ when traversing $R$. However, not all of these changes lead to the removal (and subsequent output) of a group record from the buffer. We also know that on average, $d$ changes have taken place since the last time we encountered the value of $A_y$ of the current tuple. The probability that a group record has been swapped out after $x$ swaps is $1 - \left(\frac{B-1}{B}\right)^x$. As we know the average probability $P$ that a group record is not present in the buffer momentarily, about $d \cdot P$ of the changes lead to swaps (and consequently

contribute to the output). So we can estimate the cardinality of the output $o$ by

$$o \approx m + c\left(1 - \left(\frac{B-1}{B}\right)^{d \cdot P}\right) \quad (6)$$

#### 5.1.2 Refining the Estimation

Having just one average value $d$ for the distance does not consider the fact that the distances between appearances can vary widely. To get a more accurate picture, we divide the distances $\delta_k$ into different classes, depending on their value. We define $D$ different distance classes using a partitioning $\Omega = (\omega_0, \omega_1, \ldots, \omega_D)$ (with $\omega_0 = 0$, $\omega_D = \infty$), such that $\delta_k(\omega_u)$ encompasses all distances that are between $\omega_{u-1}$ and $\omega_u$. We define $\Delta_k(\omega_u)$ to be the positions of tuples whose attribute values $A_y$ are equal to $k$ and whose distance to their predecessor is between $\omega_{u-1}$ and $\omega_u$:

$$\Delta_k(\omega_u) = \{i_j | t_{i_j} \in T_k, j > 1 \ \wedge \ \omega_{u-1} < \sum_{l=i_{j-1}+1}^{i_j} f(l) \leq \omega_u\} \quad (7)$$

We do not include the lower bound of each interval in order to filter out the tuples that have a distance of zero to their predecessor (as these are irrelevant for our measure). Now $\delta_k(\omega_u)$ is simply

$$\delta_k(\omega_u) = \sum_{i_j \in \Delta_k(\omega_u)} \sum_{l=i_{j-1}+1}^{i_j} f(l) \quad (8)$$

In order to calculate the average distance $d(\omega_u)$ for each class, we also have to divide up $c$, the total number of changes, into the different classes:

$$c(\omega_u) = \sum_{k=1}^{m} |\Delta_k(\omega_u)| \quad (9)$$

Combining these steps, we get $D$ different values (for $u = 1, 2, \ldots, D$):

$$d(\omega_u) = \begin{cases} \frac{\sum_{k=1}^{m} \delta_k(\omega_u)}{c(\omega_u)} & \text{if } c(\omega_u) \neq 0 \\ 0 & \text{else} \end{cases} \quad (10)$$

Consequently, we also have to adapt Equation (6) to take the different distance classes into account:

$$o \approx m + \sum_{i=1}^{D} c(\omega_i)\left(1 - \left(\frac{B-1}{B}\right)^{d(\omega_i) \cdot P}\right) \quad (11)$$

#### 5.1.3 Yet Another Refinement

We are now going to relax the assumption that the buffer is filled right from the start and that we have already seen a tuple of each group. We assume that, when arriving at tuple position $l$ in $R$, we know the number $v_l$ of different grouping values observed so far. (We can estimate this number by a formula given by Larson [23] for uniformly distributed data or by obtaining them from a histogram.) Estimating the probability $P$ that a group record is not in the buffer now involves $l$ instead of $n$ tuples and $v_l$ instead of $m$ different groups. So,

$$P(l) \approx \begin{cases} 0 & \text{if } v_l \leq B \\ \frac{\sum_{i=1}^{v_l} \prod_{j=0}^{B-1}\left(1 - \frac{\frac{l}{n} s_i}{l - j \cdot \tilde{s}_i}\right)}{v_l} & \text{if } v_l > B \end{cases} \quad (12)$$

with $\tilde{s}_i = \frac{l\left(1 - \frac{s_i}{n}\right)}{v_l - 1}$. We do not want to calculate the probability $P$ anew for each tuple in $R$, so we check the number of observed groups for the $z$ positions $l_1, l_2, \ldots, l_z$ (we partition the relation $R$ into what we call group classes). For each interval $l_{j-1}$ to $l_j$, $v_j$ is the number of groups seen when reaching the tuple at position $l_j$. So, for each interval we are working with an upper bound for the number of groups, resulting in a slight overestimation of the output cardinality. After having introduced the intervals into Equation (11), we get:

$$o \approx m + \sum_{i=1}^{D} c(\omega_i) \sum_{j=1}^{z} \left(\frac{l_j - l_{j-1}}{n}\right) \left(1 - \left(\frac{B-1}{B}\right)^{d(\omega_i) \cdot P(l_j)}\right) \quad (13)$$

with $l_0 = 0$ and $v_0 = 0$.

### 5.2 Optimal Replacement Strategy

The optimal replacement strategy always swaps out the group record whose next appearance in

the stream is the farthest off (this technique was already proposed for block replacements in virtual memory management [5]). Obviously, this strategy cannot be implemented without look-ahead capability. However, it is still interesting to know the best possible case of a PPA operator, as this can serve as a lower bound for the performance of all other algorithms.

In order to estimate the output size $o_{opt}$, we modify Equation (13) in the following way. All values whose distance to their last appearance is smaller than the buffer size are absorbed, since in the optimal case, we can make certain that they are kept in the buffer. So (with sign(0) = 0) we get

$$o_{opt} \approx m + \sum_{i=1}^{D} c(\omega_i) \sum_{j=1}^{z} \left( \frac{l_j - l_{j-1}}{n} \right) \left( 1 - \left( \frac{B-1}{B} \right)^{d(\omega_i) \cdot P(l_j) \cdot \text{sign}(\max(0, d(\omega_i) - B))} \right) \quad (14)$$

# 6 Experimental Evaluation

We implemented both algorithms (random replacement strategy and optimal replacement strategy) in order to validate our cost model. It was important to test our measure with many different data sets. For that reason, we generated synthetic data sets in which we could control the clusteredness (see Section 6.1). We tested the effects of several parameters on our estimation, namely number of distance and group classes, buffer size, distribution of group sizes, and the cardinality of the relations.

## 6.1 Generating the Data

As we wanted to investigate the effect of the clusteredness of the data, we generated the data in the following way. We started by creating relations containing tuples that were perfectly clustered in regard to the grouping attribute. This was done for different (group value) distributions: uniform, Zipf, and normal. In each case we decreased the clusteredness by randomly swapping two tuples within the relation several times. In the following graphs we state the number of swaps done in each relation to specify the clusteredness. We also created totally randomly ordered relations (denoted "shuffled" in the following). We also tested the cardinality estimation on real-world data taken from detectors in the HERA-B experiments at DESY (in this case, we grouped after the z-positions of particles traveling through the detector).

## 6.2 Random Replacement

### 6.2.1 Number of Distance Classes

In Section 5.1.2, we introduced distance classes with the goal to increase the precision of our estimation. In Figure 1, we plotted the size of the output (y-axis) of the partial aggregation operator depending on the clusteredness of the input (x-axis) for different numbers of distances classes (1, 2, 4, and 14) with $w_i = \lceil n^{\frac{1}{D}} \rceil^i$ for $1 < i < D$. For these measurements, we had 500 groups (with 20 elements each for uniformly distributed data), a buffer size of 100 groups, and assumed one group class (i.e., the buffer is filled and we have seen all 500 groups right from the start).

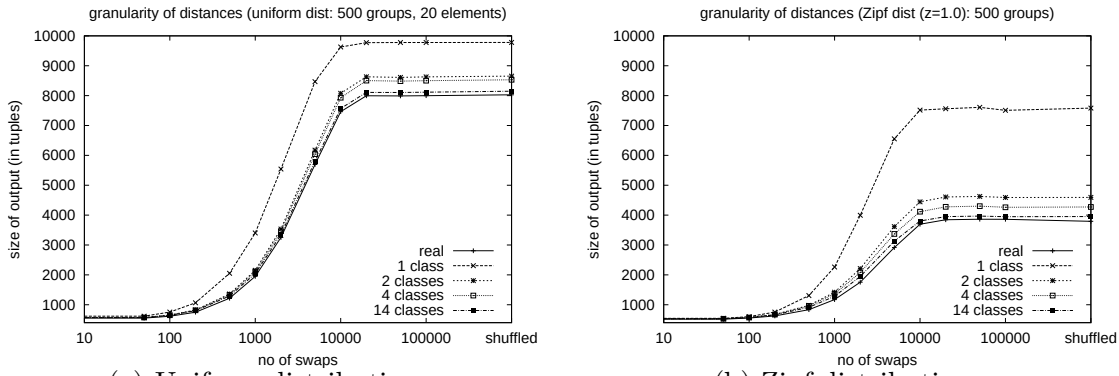
In Figure 1(a), the results for uniformly distributed data are depicted, while Figure 1(b) shows those for Zipf distributed data. In both cases, it can be clearly seen that by increasing the number of different distance classes, the theoretically predicted output size rapidly approaches the measured value. This is also a nice feature of our cost model. The more resources are available for estimating the output cardinality, the better the result will be. With 10 to 15 distance classes we were able to keep the deviation of our estimation from the actually measured value within 5%.

The huge effect of the data clusteredness can also be seen nicely in Figure 1. The size of the output for shuffled data is 1500% larger than that for perfectly clustered data.

### 6.2.2 Number of Group Classes

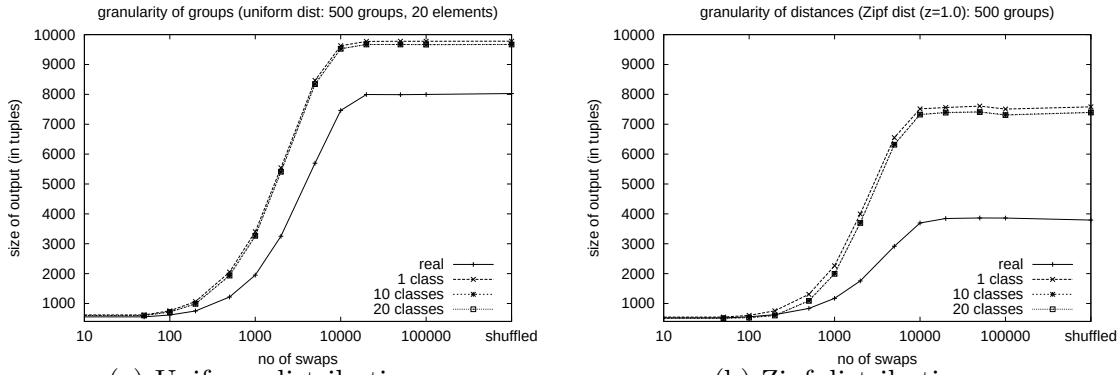
Now we drop the assumption that all groups have been seen right from the start and apply Equation (13) from Section 5.1.3. This time, we use one distance class ($\Omega = \{0, \infty\}$) and several different group classes (1, 10, and 20) to see which of the refinements improves the accuracy more. Again, we had 500 groups (with 20 elements each for uniformly distributed data) and a buffer size of 100 groups.

Figure 2(a) shows the results for uniformly distributed data, Figure 2(b) those for Zipf dis-




(a) Uniform distribution (b) Zipf distribution

Figure 1: Number of distance classes




(a) Uniform distribution (b) Zipf distribution

Figure 2: Number of group classes

tributed data. Although we are able to improve the precision, the effect of increasing the number of group classes is not as significant as that of the number of distance classes. We conclude that it is more important to allocate a larger amount of the available resources to the number of distance classes than to the amount of group classes.

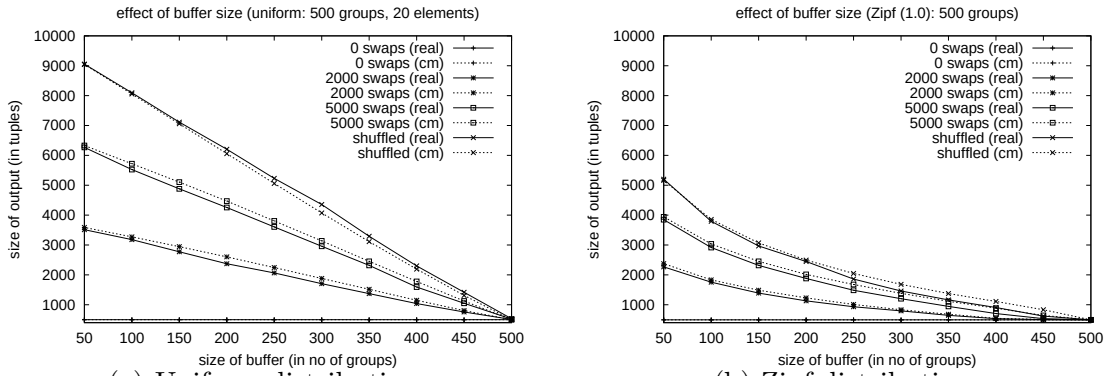
### 6.2.3 Influence of Buffer Size

We want to be able to give an accurate picture of the behavior of the partial preaggregation operator for different buffer sizes (which is a very important parameter for the optimizer). On the x-axis we have the size of the buffer (in number of groups) and on the y-axis the size of the output. For different levels of clusteredness, we plotted pairs of curves: one for the measured value (real) and one for the estimated value (cm, for cost model). The curves of each pair are always close together, both for uniformly distributed data (Figure 3(a)) and for Zipf distributed data (Figure 3(b)). (The number of distance classes was set to 14 and the number of group classes to 10.)

### 6.2.4 Distribution of Group Sizes

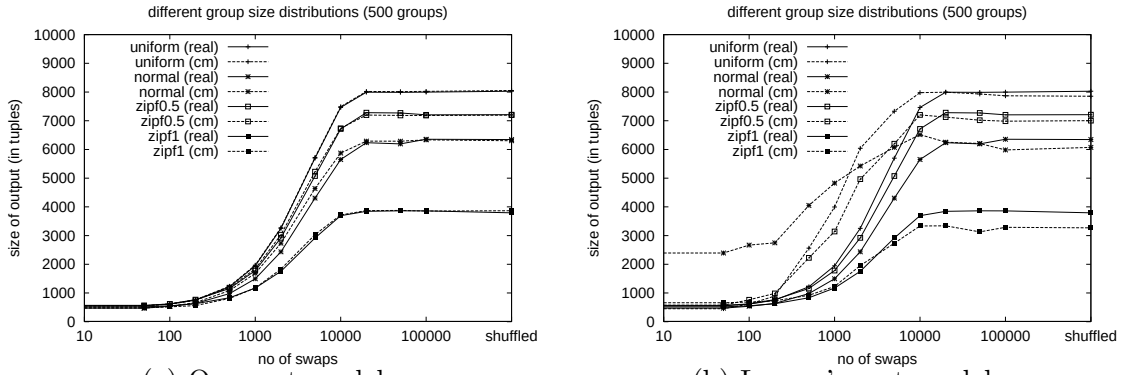
Our technique makes no assumption about the distribution of the group size. Nevertheless, it should be able to handle many different distributions correctly. In Figure 4(a), we plotted pairs of curves for several different distributions of the group size. We used uniformly distributed group sizes, two Zipf distributed group sizes (with $z = 0.5$ and $z = 1$), and normally distributed group sizes (with $\mu = 300$ and $\sigma = 70$). As we can clearly see, the output size of the partial preaggregation operator can be estimated accurately, independently of the chosen distribution.

For comparison, we have plotted the results for the formulas taken from [24]. We have fed them with the exact values for the group sizes and the exact value for the point when the first buffer overflow occurs. The results are depicted in Figure 4(b). Note that these formulas estimate the




(a) Uniform distribution (b) Zipf distribution

Figure 3: Effect of buffer size




(a) Our cost model (b) Larson's cost model

Figure 4: Effect of distribution

output size of an LRU-strategy. So, the numbers should be slightly smaller than the measure values. Quite contrary to these expectations, the estimations are usually larger than the measured values (as can be seen in Figure 4(b)). For normally distributed data, the estimation is off up to a factor of 5.

### 6.2.5 Relation Cardinality

In Figure 5, we show our results for investigating the effect of the relation size on our estimation (the left hand side depicts the results for uniformly distributed data, the right hand side for Zipf distributed data). We started out with a relation containing 250 groups with 10 members each (for uniformly distributed data). In each step, we doubled the number of groups and members up to 4000 groups with 160 members each. The buffer size was set to $\frac{1}{10}$ of the number of groups in each case. We show the curves for maximal clusteredness, $\frac{1}{5}$ relation size, $\frac{1}{2}$ relation size, and shuffled. As we can see in the results, the estimation is not distorted by larger numbers, but scales very well with the relation cardinality.

### 6.2.6 Real-world Data

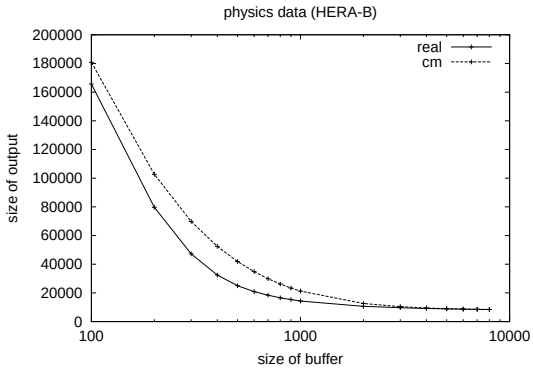
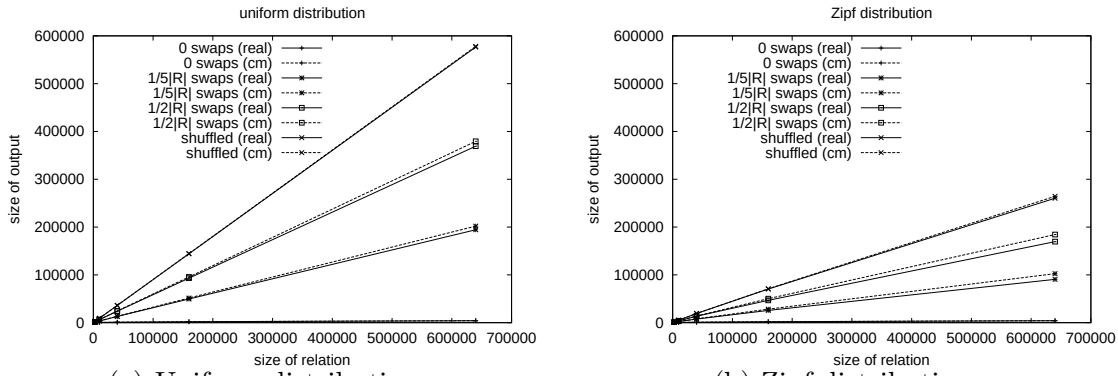



Figure 6: Real-world data

For experiments with real-world data we used 500,000 particle tracks grouped into 8403 different groups. In Figure 6, we plotted the results for different buffer sizes. Although the results




(a) Uniform distribution (b) Zipf distribution

Figure 5: Effect of relation cardinality

for the generated data are slightly more accurate, the predicted output cardinalities follow the measured ones closely.

### 6.3 Optimal Replacement

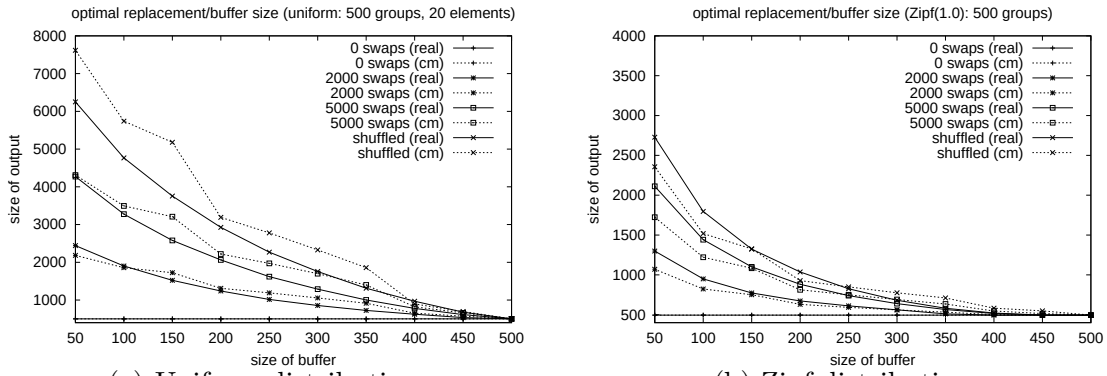
Figure 7 shows the results for the optimal replacement strategy (the left hand side for uniformly distributed data, the right hand side for Zipf distributed data). We plotted the output size depending on the buffer size for different levels of clusteredness (we used 14 distance and 10 group classes). The precision of the cardinality estimation for the optimal replacement strategy cannot really keep up with that for the random replacement strategy, but it is still very good (we had a maximal error of 38%). Also the curve is not as smooth. This is due to the sharp cutoff in Equation (14). All distances $d(\omega_i)$ below $B$ are totally ignored, while all above count regularly (as in random replacement). In reality, the cut is not as distinct and we are looking for a better way to model this case. The curve run can be smoothed by introducing more distance classes at the price of higher computation costs for the estimation.

## 7 Conclusion and Outlook

Estimating the output cardinality of PPA accurately is a necessary prerequisite for a query optimizer to reach a decision on applying it. Previous analyses of PPA did not consider the clusteredness of the data, but assumed randomized data. This is unfortunate, because most real-world data is clustered and this has a large influence on the output cardinality. We developed a measure of clusteredness and showed how it can be used to estimate this cardinality accurately. The quality of our approximations was demonstrated by thorough experiments.

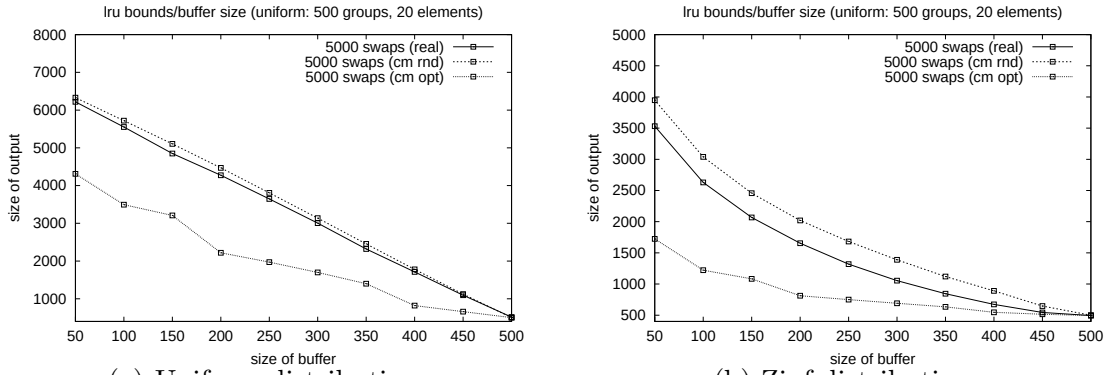
At the moment, we are working on two topics: extending our model to cover PPA with a LRU strategy and developing techniques for calculating the clusteredness of a relation very efficiently (similar to the work done in [20]). As already mentioned, developing a formal model for the LRU replacement strategy is quite complex. However, with our measure and estimations for the random and optimal replacement strategy, we can give an upper and lower bound for the output cardinality of the LRU strategy. In Figure 8, we plotted our predictions with measured values for an LRU strategy PPA algorithm. For uniformly distributed data an LRU strategy is on par with random replacement, for Zipf distributed data it lies between the random and the optimal case. This confirms the results of Smith in [28], who claimed that an LRU-strategy performs about 10% better than random replacement for realistic data.

At the moment, we recommend our technique for mostly static databases. It is not suited very well for dynamic environments in which frequent recalculations of our measure are necessary, due to a complexity of $O(n\lceil\frac{m}{b}\rceil)$ for calculating $d$ (where $n$ is the number of tuples, $m$ the number of groups, and $b$ the buffer space available for calculation). Estimating the output cardinality is much cheaper, since the complexity here is $O(Dz)$, where $D$ is the number of distance and $z$ the number of group classes. This also means, that we can trade speed for accuracy. After the promising start with our measure, we are quite confident that the calculation of $d$ can be accelerated with the help of a similar trade-off by not looking at the whole relation and all groups, but




(a) Uniform distribution (b) Zipf distribution

Figure 7: Effect of buffer size for optimal replacement




(a) Uniform distribution (b) Zipf distribution

Figure 8: Bounding the LRU strategy

at some fraction.

**Acknowledgments**

We thank Vasco Amaral for providing the experimental data from HERA-B at DESY. A word of thanks also goes to the anonymous referees for their helpful hints on improving the paper.

## References

[1] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. In *SIGMOD Conference*, pages 487–498, 2000.

[2] Anant Agarwal, Mark Horowitz, and John L. Hennessy. An analytical cache model. *TOCS*, 7(2):184–215, 1989.

[3] Sameet Agarwal, Rakesh Agrawal, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the computation of multidimensional aggregates. In *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 506–521, 3–6 1996.

[4] Rafiul Ahad, K. V. Bapa Rao, and Dennis McLeod. On estimating the cardinality of the projection of a database relation. *TODS*, 14(1):28–40, 1989.

[5] L.A. Belady. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[6] Michael Benedikt and Leonid Libkin. Exact and approximate aggregation in constraint query. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 102–113. ACM Press, 1999.

[7] Chee Yong Chan, Wenfei Fan, Pascal Felber, Minos N. Garofalakis, and Rajeev Ras-

togi. Tree pattern aggregation for scalable XML data dissemination. In *VLDB'02, Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China*, pages 826–837, 2002.

[8] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Towards estimation error guarantees for distinct values. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA*, pages 268–279, 2000.

[9] Surajit Chaudhuri, Gautam Das, Mayur Datara, Rajeev Motwani, and Vivek R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *17th Int. Conf. on Data Engineering*, Heidelberg, Germany, 2001.

[10] Surajit Chaudhuri and Kyuseok Shim. Including Group-By in Query Optimization. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 354–366, Santiago, Chile, 1994.

[11] Surajit Chaudhuri and Kyuseok Shim. An overview of cost-based optimization of queries with aggregates. *Data Engineering Bulletin*, 18(3):3–9, 1995.

[12] Paolo Ciaccia and Dario Maio. Domains and active domains: What this distinction implies for the estimation of projection sizes in relational databases. *TKDE*, 7(4):641–655, 1995.

[13] Sophie Cluet and Guido Moerkotte. Efficient evaluation of aggregates on bulk types. In *Database Programming Languages (DBPL-5), Proceedings of the Fifth International Workshop on Database Programming Languages, Gubbio, Umbria, Italy, 6-8 September 1995*, Electronic Workshops in Computing, page 8. Springer, 1995.

[14] Asit Dan and Donald F. Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *SIGMETRICS*, pages 143–152, 1990.

[15] Alin Dobra, Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD Conference, Madison, Wisconsin*, 2002.

[16] Wolfgang Effelsberg and Theo Härder. Principles of database buffer management. *TODS*, 9(4):560–595, 1984.

[17] Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[18] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 299–310, 1998.

[19] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[20] Sudipto Guha and Nick Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *18th Int. Conf. on Data Engineering*, San Jose, California, 2002.

[21] Alfons Kemper, Donald Kossmann, and Christian Wiesner. Generalised hash teams for join and group-by. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 30–41, 1999.

[22] Arnd Christian König and Gerhard Weikum. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 423–434, 1999.

[23] Per-Åke Larson. Grouping and duplicate elimination: Benefits of early aggregation. Technical Report MSR-TR-97-36, Microsoft, 1997.

[24] Per-Åke Larson. Data reduction by partial preaggregation. In *18th Int. Conf. on Data Engineering*, San Jose, California, 2002.

[25] Lothar F. Mackert and Guy M. Lohman. Index scans using a finite LRU buffer: A validated I/O model. *TODS*, 14(3):401–424, 1989.

[26] Michael V. Mannino, Paicheng Chu, and Thomas Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3):191–221, 1988.

[27] Jeffrey F. Naughton and S. Seshadri. On estimating the size of projections. In Serge Abiteboul and Paris C. Kanellakis, editors, *ICDT'90, Third International Conference on Database Theory, Paris, France, December 12-14, 1990, Proceedings*, volume 470 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1990.

[28] Alan Jay Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, 1982.

[29] S. Sudarshan, Divesh Srivastavaa, Raghu Ramakrishnan, and Catriel Beeri. Extending the well-founded and valid semantics for aggregation. In *Proceedings of the 1993 International Symposium, Vancouver, British Columbia, Canada*, pages 590–608, 1993.

[30] Aris Tsois, Nikos Karayannidisa, Timos K. Sellis, and Dimitri Theodoratos. Cost-based optimization of aggregation star queries on hierarchically clustered data warehouses. In *Proceedings of the 4th Intl. Workshop DMDW'2002, Toronto, Canada*, pages 62–71, 2002.

[31] Weipeng P. Yan and Per-Åke Larson. Eager aggregation and lazy aggregation. In *The VLDB Journal*, pages 345–357, 1995.

## A Derivation of Probability $P$

Deriving $P$ for uniformly distributed data is quite simple. We select $B$ groups randomly from $m$ possible groups without replacement to fill the buffer. We want to know the probability that one particular of those groups will not end up in the buffer:

$$\begin{aligned} P &\approx \frac{\binom{m-1}{1}}{\binom{m}{1}} \frac{\binom{m-2}{1}}{\binom{m-1}{1}} \cdots \frac{\binom{m-B}{1}}{\binom{m-B+1}{1}} \\ &= \frac{\binom{m-B}{1}}{\binom{m}{1}} = \frac{m-B}{m} = 1 - \frac{B}{m} \end{aligned}$$

Now we want to consider the different cardinalities of the groups. When visualizing this situation, we have an urn filled with colored balls and each color represents one group. The cardinality of each group is represented by the number of balls for each color. This is also a selection without replacement. However, when we have chosen a group we have to remove all remaining balls of this color from the urn, since this group cannot be chosen again. As an intermediate step we want to approximate the probability $P(g_i)$ that group $g_i$ is not present in the buffer after filling it. We assume that all other groups $(G \setminus g_i)$ have the same size $\hat{s}_i$ (to simplify the matter), with $\hat{s}_i = \frac{n-s_i}{m-1}$.

$$\begin{aligned} P(g_i) &\approx \frac{\binom{n-s_i}{1}}{\binom{n}{1}} \frac{\binom{n-s_i-\hat{s}_i}{1}}{\binom{n-\hat{s}_i}{1}} \frac{\binom{n-s_i-2\hat{s}_i}{1}}{\binom{n-2\hat{s}_i}{1}} \cdots \frac{\binom{n-s_i-(B-1)\hat{s}_i}{1}}{\binom{n-(B-1)\hat{s}_i}{1}} \\ &= \prod_{j=0}^{B-1} \frac{n-s_i-j\hat{s}_i}{n-j\hat{s}_i} \end{aligned}$$

In order to estimate the average probability $P$ that a group is not in the buffer, we average over all groups:

$$P = \frac{\sum_{i=1}^{m} P(g_i)}{m}$$