Scientific Discovery through Weighted Sampling

Lefteris Sidirourgos, Martin Kersten, Peter Boncz
Database Architectures, CWI
Amsterdam, The Netherlands
{lsidir, mk, boncz}@cwi.nl

Abstract—Scientific discovery has shifted from being an exercise of theory and computation, to become the exploration of an ocean of observational data. Scientists explore data originated from modern scientific instruments in order to discover interesting aspects of it and formulate their hypothesis. Such workloads press for new database functionality. We aim at sampling scientific databases to create many different impressions of the data, on which the scientists can quickly evaluate exploratory queries. However, scientific databases introduce different challenges for sample construction compared to classical business analytical applications. We propose adaptive weighted sampling as an alternative to uniform sampling. With weighted sampling only the most informative data is being sampled, thus more relevant data to the scientific discovery is available to examine a hypothesis. Relevant data is considered to be the focal points of the scientific search, and can be defined either a priori with the use of functions, or by monitoring the query workload. We study such query workloads, and we detail different families of weight functions. Finally, we give a quantitative and qualitative evaluation of weighted sampling.

Keywords—Scientific Data Management; Sampling;

I. INTRODUCTION

Scientific discovery has shifted from being an exercise of theory and computation, to become the exploration of an ocean of observational data. This transformation was identified by Jim Gray as the 4th paradigm of scientific discovery [1]. State-of-the-art observatories, digital sensors, and modern scientific instruments produce every day petabytes of information. This scientific data is stored in massive datacenters for later analysis. From the data management viewpoint, the capture, curation, and analysis of data are not computation-intensive processes any more, but a data-intensive ones. The explosion in the amount of scientific data presents a new “stress test” for database systems design. Meanwhile, the scientists are confronted with new questions: how can relevant and compact information be found from such a flood of data?

The predominant answer given by the data management community is the raw power of big datacenter installations, complemented by new technologies focusing on scalable distribution of data and operations [2]. When it comes to curating and analysing vast amounts of data one can hardly argue that another approach is feasible. However, scientific discovery implies also a more delicate and refined task, that of forming the scientific question, a hypothesis to be later tested for correctness. Such formations of scientific questions or hypotheses, as well as the initial proof-of-validity, constitute an ad-hoc exploratory scientific query workload. During an iterative and interactive query session, the scientists pose queries in order to examine the nature of the stored data, discover interesting aspects of it, formulate their hypotheses, but also test the syntactical correctness of their queries.

Exploratory scientific workloads press for new database functionality. Scientists are interested in an initial educative answer without having to spend too many valuable resources. Nevertheless, it is necessary to provide bounds on the quality of the answer and limit the execution time [3], [4], [5]. Our approach to this new challenge is sampling the database to create many different “snapshots” of the data, called impressions, on which the scientists can evaluate their queries. Impressions come in various sizes and are organised in a multi-layer hierarchical manner. The size of an impression determines both the quality of the result, and the time spent on query processing. The rule of thumb is the smaller an impression, the faster an answer will be given, while the bigger an impression is, the better the quality of the answer will be. With such a collection of impressions, accompanied by a bounded query execution engine, we envision a complete framework for scientific data management with bounds on runtime and quality [5]. This framework serves the purpose of data-intensive scientific discovery. Once a scientific hypothesis is formulated, and a correct set of queries is written, the scientist may then validate his findings over the entire dataset.

In this work we are concerned with the problem of how to create and maintain samples, to be used in impressions. The predominant technique in the literature is uniform sampling over the data. However, scientific databases introduce different properties for sample construction compared to classical business analytical applications. For starters, a scientific search can be more focused, drilling into details, while a business strategy is made by observing general trends along the breadth of the data. Accurately answering focused queries stresses existing sampling methods. Secondly, data-type domains are different. Scientific data has many double precision values, often with strong skew. On the other hand, classical OLAP applications usually have predefined categorical data types with low cardinality (e.g., product type, country, etc.). Another difference of scientific discovery is
that group-by and aggregations, such as average and sums, are less common. In fact, most of the research in sampling presents solutions on overcoming errors introduced by rare high values that throw the estimations off track, or by very small group that won’t appear in a sample.

We propose using weighted sampling as an alternative to uniform sampling. Our goal is to get the most out of scientific data by sampling only the interesting parts. For a given statistical test, the power of the test can always be increased by increasing the sample size. Clearly, with larger sample sizes, the probability of accepting a hypothesis is increased as the sample size increases. Weighted sampling increases the sample size around areas of interest. Thus, although the total resources dedicated to sample storage is constant, queries that target areas of interest give better approximated answers and with higher confidence levels. The areas of interest are defined as the focal points of the scientific search. It can be a priori defined (e.g., interest only on spikes in the data) or it can be observed during a query session. Once the focal points have been established, a weight is assigned to each newly ingested tuple. This is done with the help of weight functions, which can also be adaptive to any changes of focal points over time. With adaptive weighted sampling queries are answered faster, are less error prone, and fewer physical resources are used.

II. RELATED WORK

Sampling is an important database operation primarily used for extracting statistical information over the data without the need for accessing the entire base. Fundamental work on database sampling is due to Olken and Rotem [6]. The typical usage of samples in modern data management systems is to provide an estimation of the result size [7], [8], [9] which is then used by the cost model of a query optimiser, or to facilitate approximate query answering [10], [11]. Biased sampling has been used in the past for non-uniform sampling methods based on weight functions [3], [12]. Dynamic sample selection [12] to answer approximate queries is motivated by “ad hoc, exploratory data analysis” but in a business analytic environment. In such a setting, the data is explored in a breadth search manner looking for global trends, as opposed to scientific application where a scientist searches in depth, focusing into the details. Congressional samples are also constructed for answering approximate group-by queries [13]. This technique is also known to the literature as stratified sampling [14]. Concise and Counting sampling techniques include the number of occurrences of a value on the sample [15]. ICICLES [16] treats the results of a query as being newly appended tuples that should be sampled, thus forcing frequently appearing results to be more likely to be part of the sample. BlinkDB [3] uses samples to achieve interactive query execution times over a distributed environment, by extending the Hive/HDFS execution stack.

III. SCIENTIFIC WORKLOAD ANALYSIS

To improve the way scientists can explore large science datawarehouses, a much better understanding of their query interaction is needed. Fortunately, little is known publicly to guide us. A notable exception is the SDSS Skyserver DR7 database which contains astronomy data. We study its publicly accessible query logs to extract query patterns. Skyserver contains data for over half a billion sources. A source in astronomy is any object in the sky that emits energy, for example a star or a nebula. Every source is identified in the database with a unique id, and can be located by its position in the sky. The position is marked with two coordinates, the right ascension (ra), and the declination (dec). We study the queries fired against the Skyserver DR7 over the span of one month. The total number of queries logged is 1,242,767 and originated from 2,646 distinct IP addresses. Out of all queries, 402,108 queries had an unknown origin so could not be associated with a specific user. The 50 most active users accounted for over 90% of the remaining workload (774,924 queries).

An interesting side observation is the increased number of queries originating from crawlers of known search engines indexing the so called deep web. Our goal is to identify patterns in the queries posed by the scientists over time. We consider the coordinates ra and dec to be a natural choice of attributes to plot, since they correspond to the area in the sky that an astronomer looks at. Due to lack of space, we next show the query habits of only four – but representative – users. They are all either astronomy observatories or research institutes and universities from Asia, Europe, and America.

Figure 1 depicts the querying habits of these representative users. Each query searches for sources over an area that Skyserver covers. We plot the values of ra and dec as given in the queries’ predicates. In addition we add a third dimension that shows the exact sequence these queries were fired. We plot 4 graphs for each user: i) a 3-d plot that in the x-axis shows the time sequence of the queries and in the z- and y- axis, the ra and dec respectively, ii) the ra over the query sequence, iii) the dec over the query sequence, and iv) the ra and dec. The top left user of Figure 1 (purple colour) scans over the entire Skyserver many times throughout the query sequence. For this user a uniform sample may serve his needs better. The next user, seen in the top right corner of Figure 1 (yellow colour) scans over the same stripes of data over and over throughout the span of the entire month. His queries are very concentrated on selected areas of interest. Similarly the next user, shown on the left lower corner of Figure 1 (green colour) is interested in three narrow areas of the Skyserver and for a small period of time. Finally, the last user shown on the right lower corner of Figure 1 (blue colour) exhibits an exploratory query pattern, where the focal point is constantly moving towards a more limited area.
IV. Weighted Sampling

Complementary to the informative user-study described above, we can now proceed with designing new techniques for choosing those data parts of a relational database that better serve the exploration of scientists. We study the process of sampling a relational database for the purpose of providing data impressions. An impression is a different view of the data, that may hide details, or bring in the foreground details of a specific characteristic of the data. The scientist then may explore the impressions for scientific discovery. The benefit of an impression lies in the fact that it contains only data relevant to the query, thus making the exploration easier and faster. The concept of an impression is detached from the underlying method of creating it. Our approach is based on sampling, while other approaches may be applicable.

The predominant technique for sampling in the literature is uniform sampling. However, uniform sampling assumes that all data are of equal importance. This may be true for traditional analytical workloads with aggregates and group-by operators, but for the exploratory scientific workload is not. A scientist more often wishes to explore the details rather than the overall picture. To address the unique properties of scientific discovery, we introduce a framework for performing weighted sampling. In weighted sampling, not all tuples are sampled with the same probability. A weight function determines the probability \( P \) with which a tuple is included in the sample. The choice and implementation of the weight function depends on the demands put forward by the application.

**Definition 1**: A weight function \( w \) is any function that assigns a non-negative real number \( w_i \) to the \( i \)-th member of the population to be sampled. That is,

\[
w : \mathbb{N} \rightarrow \mathbb{R}_+, \ w(i) \mapsto w_i
\]

Notice that we do not require the range of the weight function \( w \) to be between \([0, 1]\). Special care is needed to normalise the weight functions in order to produce the correct probability values. Hence, the probability for the \( i \)-th member of the population to be sampled is

\[
P(i) = \frac{w_i}{\sum_{j=1}^{N} w_j}
\]

where \( N \) is the total size of the population. However, we will also experiment with weight functions that have a range between \([0, 1]\) and thus no such normalisation is needed.

Given a tuple \( t \) with \( A, B, C, \ldots \) attributes, a weight function \( w \) may be defined in such way that more than one attribute takes part in the computation of the weight of that tuple. We use \( w(t) \) to notate the weight of tuple.
t even if it is a combination of many attributes, i.e.,

$$w(t) = w(t.A, t.B, t.C, \ldots)$$

We present three families of weight functions to address different application requirements. The first set of functions are data independent, the second data dependent, and the third query dependent.

A. Data Independent Weight Functions

In many scientific applications the latest observations are of most interest. For example, in seismographic monitoring the earthquakes of the last 24 hours or the last week is the focal point of the seismologists. Another example are the latest measurements for the meteorologists. Therefore, it can be useful to maintain a sample that contains the latest events. For example, given the query

\[
\text{SELECT * FROM Events AS ev WHERE ev.timestamp - current_timestamp < INTERVAL '24 hours'};
\]

a weighted sample created such that most of the past 24 hour events are included will give a more informative and complete result, than a uniform sample. We define the following data independent weight function, that adds more weight to the latest events.

**Definition 2**: Given a tuple \(t\), and a sample \(S\) of size \(s\), a data independent weight function \(w_{ls}\) that chooses the latest appended tuples over the older ones is defined as

$$w_{ls}(t) = \frac{k}{D}$$

where \(D\) is a parameter that defines the size of the timeframe for the latest events, and \(k \leq s\) is the ratio of latest over older events to be included in the sample \(S\). If \(D < s\) then \(w_{ls}(t) = 1\), that is all new tuples are included on the sample.

More specifically, \(D\) is equal to the expected amount of tuples to be inserted in the database during the timeframe of interest. For example, if we are interested in the earthquakes that took place on the last 24 hours, and there are on average 1000 events recorded per day, then \(D\) is set to 1000 or more. Similarly, if we are interested in events of the last week, then \(D\) is set to 7000, and so on. Assuming that the size of the sample is \(s\), we set the parameter \(k\) to be equal to \(s\) if the application’s interest is on only new events. Otherwise, if \(k < n\) then there will be \(k\) latest events in the sample, and \(n - k\) older events on average. By tuning \(k\) we can set the ratio between old and new events in the sample.

We refer to this family of weight function as *latest seen*, or **ls** for short. This application of weighted sampling is particular useful for examining the latest observations, or for grabbing the newest additions in the database and forwarding them to third parties.

B. Data Dependent Weight Functions

Weight functions that depend on the data are the most generic ones. The goal of such functions is to apply filters to the data. Only tuples that satisfy these filters are eligible for sampling. The existing **VIEW** infrastructure of relational databases often allows the definition of such filtering. For example, an SQL statement

```
CREATE VIEW name AS
SELECT * FROM PhotoObj WHERE ra < 180;
```

will create a view containing only sources that have \(ra\) smaller than 180. A uniform sample over such a view will produce the desired result. However, the same effect can be achieved with a simple weight functions of the form

$$w(t) = \begin{cases} 
1 & \text{if } t.ra \leq 180 \\
0 & \text{otherwise}
\end{cases}$$

Nevertheless, data dependent weight functions can have more complicate expressions and multiple weights compared to what can be achieved with existing relational database functionality. A generic multi-part data dependent weight function can be defined as follows.

**Definition 3**: Given a tuple \(t\) and with attributes \(A, B, C, \ldots\) a multi-part data dependent weight function is defined as

$$w_{m}(t) = \begin{cases} 
w_{1} & \text{if } expr_{1}(t) \\
w_{2} & \text{if } expr_{2}(t) \\
\vdots & \\
0 & \text{otherwise}
\end{cases}$$

such that at most one of \(expr_{1}, expr_{2}, \ldots\) can be true.

Besides these generic data dependent weight functions, more specialised functions are necessary for certain applications. In scientific applications more often than usual, scientists are interested in the outliers rather than the average value distribution. This demand is entirely opposite from traditional application, where outliers are considered bad cases and need to be avoided because they can drastically alter the result of an approximated answer. In scientific exploration however, outliers may be of the utmost importance. For example, in seismic data, a big earthquake is much more important than the many small regular earthquakes that repeatedly occur around the globe. Similarly in astronomy, objects that emit light in a high spectrum are more suitable for observation. For this reasons, we define a family of weight functions that assign heavier weights to tuples that are further from the usual observed values. We refer to these functions as **z-weight** because they are inspired by the **standard score** \(z\) for identifying outliers.

**Definition 4**: Given a tuple \(t\) and an attribute \(t.A\) such that we are interested in its outlier values, the **z-weight** function is defined as

$$w_{z}(t) = \frac{|t.A - \mu|}{\sigma}$$

where \(\mu\) is the mean of the values of attribute \(t.A\) and \(\sigma\) the standard deviation.
Essentially, the $z$-weight function assigns a higher weight to values deviating more from the mean. The distance from the mean is then divided by the standard deviation to overweight the outliers. If $(t.A - \mu) < \sigma$, that is if the value of the attribute is between the expected deviation from the mean, then $w_z(t) < 1$, otherwise $w_z(t) > 1$. In order to calculate the $w_z$ function, both the mean and the standard deviation of the values of $t.A$ are needed. These can be easily computed during the loading phase with the use of an online algorithm such as the one detailed in [17]. Finally, in order to increase the probability of more outliers to be sampled, and less of the common values, the numerator of the fraction $\frac{t.A-\mu}{\sigma}$ can be squared as many times as needed. In the experimental evaluation we show results also for the cases of $\frac{(t.A-\mu)^2}{\sigma^2}$ and $\frac{(t.A-\mu)^4}{\sigma^4}$.

C. Query Dependent Weight Functions

Query dependent weight functions are steered by the observed user’s interest in the data. The interest is monitored by first identifying the attributes of the data that contain relevant scientific observational values rather than annotations or metadata. Then, by storing – typically in a histogram – the values that appear in the predicates of the query workload. We can then define a weight function that promotes those tuples that satisfy more often the predicates of the workload.

More specifically, the predicate set is defined to be all the values of the interesting attributes that are requested by the queries. These values are regarded as points that suggest the entire distribution of values of interest. Therefore, weighted sampling should follow that distribution instead of the distribution of the stored values, which would have been the case of uniform sampling. A histogram $H$ over the predicate set is maintain by increasing the counter of the bin that a value falls into. Given a histogram $H$ the following query dependent weight function $w_q$ is defined.

Definition 5: Given a tuple $t$ and a histogram $H$ build over the predicate set, the query dependent weight function $w_q$ is defined as

$$w_q(t) = H[i].c$$

where $i$ is the bin of the histogram $H$ that tuple $t$ falls into, and $c$ is the counter of that bin.

We can also normalise the weight of tuple $t$ in order to be smaller than 1. The normalise weight $w'_q$

$$w'_q = \frac{w_q(t)}{\sum_{i=1}^{n} H(i).c}.$$

Intuitively, the probability of sampling tuple $t$ is equal with the percentage of the total surface of the distribution of the predicate set it covers. The more queries have predicates that fall in a bin, the more likely is that a tuple of that bin is sampled. If the predicates are not equalities, but ranges then the counters of all bins that fall into that range are increased by one.

V. EXPERIMENTAL EVALUATION

In our experimentation we wish to examine the behaviour of the different families of the proposed weight functions and their effectiveness in query answering. For this we used the data and query logs of Skyserver as described in Section III. The data of Skyserver have been mirrored locally and loaded to the MonetDB [18] database system. The realisation of Skyserver in MonetDB uses over 2.6 terabytes of disk space, excluding the indexes. All algorithms for weighted sampling, including the proposed extensions for SQL, were implemented as part of the MonetDB system.

The first set of experiments study the properties of the weighted samples created with the query dependent weight functions. We build the histograms needed over the predicate set as given by the query workload of the four users in Section III. To emulate the process of regular ingests of data, we loaded the tuples of the PhotoObj relation of Skyserver with an append procedure in batches. The entire process of loading all tuples takes just over 20 minutes for a total of 500 million tuples. The loading time for each batch shows no noticeable variation with or without the creation of an impression. This is due to the lightweight implementation of the reservoir algorithm. The execution is performed by a separate thread, which renders the sampling overhead unnoticeable compared to the time spent on parsing
and loading the data.

Figure 2 shows the different query dependent weighted samples for the four users. The x-axis plots the right ascension (ra) found on the PhotoObj relation. The y-axis counts how many sources with the specific ra value have been found in the sample. In addition we plot the histograms capturing the query preferences of each user to be easily compared with the samples. The bottom graph of Figure 2 shows (in red) the distribution of sources over ra for a uniform sample. The samples are configured to contain 10,000 tuples. However, no matter the size of the sample, the distribution of the number of sources in the sample remains the same. All plots in Figure 2 are of the same scale.

The top plot of Figure 2 depicts the query dependent weighted sample maintained for the purple user. This is the user whose query workload scans the entire Skyserver. We previously suggested that a uniform sample will work as well. However, there are some differences compared to the uniform sample. For the values of ra bigger than 300 there is a more concentration of interest, thus more sources compared to the uniform sample are included. The second plot of Figure 2 shows the weighted sample created for the second (orange) user, as well as the histogram for the predicate set of the query workload. The queries of this user focus on sources between 0 < ra < 50 and 300 < ra < 360. In this case, the weighted sampling is superior to uniform sampling. It manages to only include sources from the areas of interest. This is translated to more sources relevant to the queries and thus better approximation with tighter error bounds. The remaining two plots of Figure 2 exhibit different distributions between the weighted sampling and the uniform one. When compared with the help of the histograms of the predicate sets, one can notice that the weighted samples follow the distribution of the query workload instead of that of the stored data. The general observation of the first set of experiments is that the query dependent weight functions and the adaptation of the reservoir algorithm produce the desired result.

Next we study the z-weight function for detecting outliers. In this data dependent weight function, sources with values that deviate from the mean are more likely to be included in a sample. To perform this experiment, a different set of attributes from the PhotoObj relation has to be chosen. The ra and dec attributes mark the position of the sources, thus they are not relevant for outliers. Instead we use the measure of the redshift of a source, denoted with r, which is more relevant for our study. The redshift follows a power-law distribution where most of the sources have very similar value and only few of them vary largely.

In Figure 3 the distribution of the values of the redshift r of a uniform sample are plotted (in red). Most of the values are concentrated between 22 and 23. We define, besides the uniform sample, three z-weighted samples. One z-weight sample is based on the 1st degree weight function, i.e., $(t.A - \mu)$, while the second and third on the 2nd, i.e., $(t.A - \mu)^2$, and 4th $(t.A - \mu)^4$ degree, respectively. The comparison between those three weighted samples reveals the existence of outliers that are missed otherwise by the uniform sample. The smallest value of r that the uniform sample contains is around 10, despite the existence of sources, in the tail of the distribution of $r$, that have much lower values. Similarly, the 1st degree of a z-weight sample considers only few outliers and reduces the sources that fall around the mean of the $r$ values. Nevertheless, such a weighted sample is still useful for certain applications. If, however, the goal is to include many outliers, then the 4th degree z-weight function is more suitable. In this case, many sources with values of $r$ down to 8 are included. Also, more outliers are part of the weight sample making the approximation of queries about them less error prone.

The final set of experiments concern the last-seen weight function. In this application, only the latest appended tuples are included in the sample. Due to the ever evolving nature of such sampling we depict the start and the end of a series of consecutive experiments. We divide the PhotoObj relation into batches of 10 million tuples. We then append these batches one by one into the PhotoObj relation. An impression with the last-seen weight function is defined
over PhotoObj with parameters $D = 10,000,000$ and $k = 5,000$, while the size of the sample is set to be 10,000.

Figure 4 depicts the state of both the uniform and the weighted sample after the load of the first 10 million tuples, and after the load of the last batch of tuples. The total number of tuples in the end is over 500 million. The uniform samples are shown on the upper plots of Figure 4 and the last-seen weighted samples at the lower part. The uniform sample always resembles the distribution of $ra$ values found in PhotoObj after each append of the next 10 million tuples. Thus, at the end of the loading process the uniform samples is exactly like the one shown in the bottom plot of Figure 2. However, at the start of the process, the uniform sample resembles only the distribution of the $ra$ values loaded up until that point. For the last-seen weighted sample the story is different. Given that this sample includes only the last 5000 tuples that have been appended and 5000 from the previous loads, the resulting distribution is much different. Notice that $D$ is set to be equal to 10 million since that is the size of the batches of tuples that we append. In the left lower plot of Figure 4 it is shown that although the weighted sample has a spike around the value 230 of $ra$, there are still some remaining tuples between 0 to 50, and 150 to 250. They are the remaining tuples of the previous appends. That is, since $k = 5000$ and the size of the sample is $s = 10000$, it follows that there should be about 5000 new tuples and 5000 older ones. The plots of the last-seen weighted sample have similar properties for all the consecutive batch appends. At the end, the weighted sample only contains tuples from the last append as shown on the right lower plot of Figure 4. The spikes in the value distribution have moved to the values from the last tuples of the last batch.

VI. Conclusions

Sampling is an important operation of a database system. It is used for approximating query results, for advising query planers, and for maintaining indexes. In this paper, we propose a new use for sampling, that of refining a database query planers, and for maintaining indexes. In this paper, we propose weighted sampling as an alternative to uniform. We have described three different families of weight functions. To support our approach, we analysed the data and query logs obtained by a real-world scientific application, namely Skyserver. Finally, we have shown experimentally that weighted sampling is more suitable for choosing those specific areas of the data that better answer focused query workloads, such as those found in the scientific discovery paradigm. The future challenge is to redesign the execution engine around bounded query execution that employs impressions to improve performance under user-defined error bounds and execution times using multiple, differently sized impressions.

REFERENCES