SciBORQ: Scientific data management with Bounds On Runtime and Quality

Lefteris Sidirourgos
CWI
Amsterdam, the Netherlands
lsidir@cwi.nl

Martin Kersten
CWI
Amsterdam, the Netherlands
mk@cwi.nl

Peter Boncz
CWI
Amsterdam, the Netherlands
boncz@cwi.nl

ABSTRACT

Data warehouses underlying virtual observatories stress the capabilities of database management systems in many ways. They are filled, on a daily basis, with large amounts of factual information derived from intensive data scrubbing and computational feature extraction pipelines. The predominant data processing techniques focus on parallel loads and map-reduce feature extraction algorithms. Querying these huge databases require a sizable computing cluster, while ideally the initial investigation should run interactively, using as few resources as possible.

In this paper, we explore a different route, one based on the observation that at any given time only a fraction of the data is of primary value for a specific task. This fraction becomes the focus of scientific reflection through an iterative process of ad-hoc query refinement. Steering through data to facilitate scientific discovery demands guarantees for the query execution time. In addition, strict bounds on errors are required to satisfy the demands of scientific use, such that query results can be used to test hypotheses reliably.

We propose SciBORQ, a framework for scientific data exploration that gives precise control over runtime and quality of query answering. We present novel techniques to derive multiple interesting data samples, called *impressions*. An *impression* is selected such that the statistical error of a query answer remains low, while the result can be computed within strict time bounds. *Impressions* differ from previous sampling approaches in their *bias* towards the focal point of the scientific data exploration, their *multi-layer* design, and their *adaptiveness* to shifting query workloads. The ultimate goal is a complete system for scientific data exploration and discovery, capable of producing quality answers with strict error bounds in pre-defined time frames.

1. INTRODUCTION

Scientific instruments produce huge amounts of information which is stored in large data warehouses. Examples are virtual observatories populated with astronomical data, or the Grid, a computer cluster spanning the globe with experimental data originating from the Large Hadron Collider at CERN. The data produced is so large that in many cases a decade of intense exploration by the scientists passes by before new observations are obtained and safe conclusions are drawn. The predominant data processing techniques focus on massivly parallel loads and distributed processing on a computer cluster. Although these approaches allow efficient execution of complicated and computationally intensive workflows, they do not provide interactive and low-cost means for the scientists to make an initial exploration over the daily produced data. The demand for data intensive scientific discovery led Jim Gray to call the community to arms to face the challenge of the "Fourth Paradigm" [11]. Facing this challenge calls for a database architecture exhibiting features different from contemporary ones.

A significant portion of the processing time goes into loading data into the science data warehouse and to prepare it for fast retrieval. The daily ingest may involve data sizes that are already hard to manage. Indexing may take an exorbitant amount of time, otherwise, massive data parallel processing is needed later on. Even a raw scan is hindered by the sequential bandwidth required. Our hypothesis is that in many real-life situations the scientist is initially satisfied with a properly chosen database sample as a starting point for determining a query scenario. This scenario, once proven correct and relevant, can be run in depth against all data overnight. The key challenge is to determine what constitutes a good set of sampled data, such that the interests of the scientist are met and the computational tasks run efficiently, thus providing interactive query performance. Traditional approximate query answering and online aggregation methods do not satisfy the requirement of complete control over both resource consumption and query result error bounds.

In this paper we describe SciBORQ [1], a novel architecture that extracts multiple samples of a science database, called *impressions* hereafter, to facilitate data exploration with guarantees on execution time and tight error bounds. The approach taken generalises the *sampling* techniques originally designed to maintain synopsis and histograms for query optimisation. Contrary to existing work, *impressions* are large samples *biased* towards the scientist's interest as captured by taking note of the query workload. SciBORQ constantly *adapts* towards the shifting focal points of real time data exploration. *Adaptive biased sampling* is more suitable under the observation that given a limitation on size, it is better to pick more tuples from the areas of interest so as to minimise the error bounds. New challenges emerge, such as providing correct estimators, satisfactory error bounds, and execution time guarantees.

Unlike synopsis and histograms, which are traditionally used for approximate query answering, the size of an *impression* may be many gigabytes rather than just kilobytes or megabytes. Query processing is designed such that a query may be evaluated against multiple impressions, according to the specific user demands on er-

[1] pronounced as *cyborg*

5th Biennial Conference on Innovative Data Systems Research (CIDR '11) January 9-12, 2011, Asilomar, California, USA.

ror and time bounds. Therefore, multiple *impressions* of different size and focus are derived. Depending on the policy chosen, some scientists would be keen to keep the latest observations in their samples, while others may only be interested in events close to a point of interest. Others may be interested in the outliers, i.e., peaks or troughs of the data instead of average values. Finally, for practical data exploration, it is imperative to control the statistical errors that might occur when a database query is executed, or bound the processing time to an acceptable limit, for example, "*give me the most representative result you can obtain within 5 minutes*".

The key features of SciBORQ can be summarised as follows:

- SciBORQ consists of *impressions*, which are created and updated incrementally during parallel database loads, such that a scientist's interest captured by an *impression* is satisfied.
- *Impressions* adaptively reflect the focal point of scientific exploration, which is derived from the query workload.
- *Bounded query processing* is facilitated by recursively defined *impressions* with strict control over their response time, disk space, and statistical quality, leading to a *multi-layer* data exploration framework.

The research opportunities under such an architecture are promising. *Biased adaptive multi-layered samples* are an entirely new concept, introducing new areas for research in database theory and system design. Moreover, the specifics of sampling over a read-optimised columnar architecture have not been studied in detail yet – leaving ample space for exploration and rethinking of already established sampling techniques. Bounded query answering calls for developing a new query processing framework that can keep errors under control by resorting to using more detailed impressions, or in the extreme case, the base data. Finally, although there is an articulated desire from the scientific community to provide database engines with control over the execution time [22], no significant steps have been done towards the realisation of such a system.

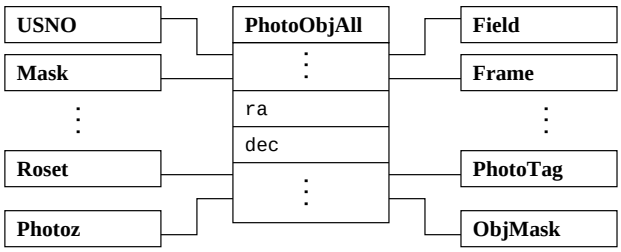
The rest of the paper is organised as follows. Section 2 presents one of the scientific data warehouses that motivates our work. Section 3 presents the design of SciBORQ. Section 4 details the concept of adaptive and biased sampling. Section 5 presents related work, followed by Section 6 with conclusions and future work.

2. SCIENTIFIC DATA WAREHOUSES

The proposed multi-layer query processing framework is targeted towards an ongoing astronomy applications, the Sloan Digital Sky Survey SkyServer. The SciBORQ implementation is designed to work on top of MonetDB [17], a modern column-store database system with a proven track record in various fields [12, 13, 16]. MonetDB is already integrated with the aforementioned application as the underlying data management system.

2.1 Sloan Digital Sky Server

The Sloan Digital Sky Server realisation in SkyServer [2] is a well-known and complex science data warehouse. Its schema encompasses several tens of relational tables. Figure 1 shows a summarised view of the schema. The main fact table `PhotoObjAll` contains hundreds of columns and several billion tuples. Each tuple contains information about an astronomical image. Attribute `ra` refers to the right ascension, and `dec` to the declination of the image in the sky. More information is incorporated by joining the foreign key attributes of the main fact table to the dimension tables. In addition, the SkyServer schema contains tens of views and functions to facilitate data exploration. A fully functional implementation of this 4TB database is available for MonetDB. The publicly accessible query logs provide a basis to derive areas of interest. A large percentage of the queries have the form shown in the lower part of Figure 1. Table `Galaxy` is a view of `PhotoObjAll` with many foreign key joins. This view presents the *galaxy* information according to the astronomers' desire. The function `fGetNearbyObjEq` returns all objects found in a nearby area specified by `ra=185` and `dec=0`. The scientists' interest can be satisfied, even if only the data around the coordinates `ra` and `dec` are available, and not the entire data set. The area described by the query predicate is the focal point of exploration. Often this focal point is limited to a small part of the sky. Queries can run anywhere from a few seconds on a large cluster, to tens of minutes on a single machine.




```
select *
from Galaxy as G,
dbo.fGetNearbyObjEq(185,0,3) as N
where G.objID = N.objID
```

Figure 1: SkyServer Schema and Query

The SkyServer application is prototypical for emerging projects, such as Pann-Stars and LSST. The system is used by around 2000 astronomers worldwide to support and drive their research. The majority of users, however, consists of amateur astronomers challenging the system with a large and complex query load.

3. SciBORQ ESSENTIALS

The key to multi-layer query processing is to extract samples from the database, the *impressions*, such that bounded query processing functionality is precisely controlled. Impressions are of different size, ranging from a few kilobytes to many gigabytes. Depending on their size, an impression fits either in the CPU cache, or the main memory of a workstation, or resides on the disk of a laptop or even a cluster. Therefore, this flexibility has a direct impact on the execution time of a query, the number of results produced, and the answer quality. Impressions bear commonalities with data synopsis and histograms but their purpose, functionality, and applicability go beyond that. In this section we sketch the landscape of the SciBORQ system.

3.1 System Parameters

Size. Impressions have different sizes with different degrees of detail. The memory footprint of an impression is directly proportional to the error bounds and the processing time that can be promised. The larger the impression, the longer the processing time and the smaller the error bounds. The user is able to define the desired size of an impression to serve her needs.

Focal point. An impression gathers data according to a sampling strategy. However, the sampling need not be from the entire database, but can be from specific areas of interest. The focal point of an impression is defined to be exactly this area of interest. The

[2] http://www.sdss.org

predicates and the join conditions of the queries in a workload determine what is important for the scientist and what not. For example, in the SkyServer paradigm, by requesting objects of the galaxy with the `fGetNearbyObjEq` function, effectively the scientist is defining one of the focal points for an impression.

Layers. SciBORQ is a multi-layer hierarchical and parallel collection of impressions. Impressions are defined to serve different purposes and needs of the application user. In traditional systems, there is one (typically small) synopsis of the data (i.e., sample, catalogue statistics) used by the query optimiser or for approximate query answering. However, in SciBORQ multiple impressions of different sizes and focal points are constructed. Each less detailed impression is derived from a previous more detailed one. In such a derivation, the focal point of the larger impression is inherited by the smaller, but many such hierarchies of impressions exist. If the error bounds during query execution are not met, the process continues on a larger impression of the same hierarchy. Moreover, smaller impressions on higher layers are more efficient to maintain since they only touch the data of the impression one layer below, and not the entire base. This is important, since small impressions need fast reflexes to efficiently adapt to query workload shifts.

Correlations. Impressions do not contain just a single attribute or relation, but may span the entire database logical schema. Each one of them may reflect the total of the base tables, or since SciBORQ is designed for read optimised column stores, may contain a subset of the attributes of a table. If the need rises, more columns can be added. Past work [3, 4, 18, 21] demonstrates how join attributes across relations are achieved with uniform sampling, and it can be adjusted to our case, too. This way, the correlations between join attributes are maintained, leading to more precise query results.

Adaptive. An impression constantly adapts to the focal point of the scientist's exploration, such that it contains more data from the areas of interest. To achieve this objective, there are two phases where an impression has the opportunity to re-adjust its focus: as a side-effect of query processing and, alternatively, by triggering impression maintenance on subsequent incremental loads. SciBORQ recognises tuples that are potentially interesting for the workload that has been observed up until now, thus increasing their chance of being part of the corresponding impression. This strategy ensures better resolution around the focal points.

3.2 Bounded Query Processing

Quality of results. An important feature of the SciBORQ design is the quality guarantees given for the query results. Any scientific exploration, no matter how generic, is useful only if strong error bounds are provided. Although traditional sampling techniques provide confidence levels on the results, error bounds will deteriorate due to correlations and complicated query plans. If a scientist is prepared to accept only a specific upper limit on the error, he will be left unsatisfied. A new query execution engine is needed that can dynamically keep the error bounds under control. In SciBORQ, if the error bound requested is not met during execution, the query evaluation moves to an impression on a lower level, with a higher level of detail, to confine the error margin. Ultimately, this can lead to the base columns for a zero error margin. The implementation of such functionality is feasible because of the special runtime optimisation capabilities of a system such as MonetDB that materialises intermediate results and provides the hooks to dynamically change the query plans [15]. In addition, since query processing is column oriented, some operators with low statistical confidence can run on a larger impression of the same hierarchy, while other operators can ran on a smaller one.

```
populate the sample smp with the first n tuples;
cnt := n;
while (tpl := block_until_next_tuple())
      cnt++;
      rnd := floor(cnt*random());
      if (rnd < n)
         smp[rnd] := tpl;
      end
end
```

Figure 2: Reservoir algorithm R

Execution time. In today's systems, the amount of results that a query produces can only be limited by a count barrier, i.e., LIMIT clause in SQL. Indirectly, the query execution time can be controlled likewise. Its implementation relies on "cutting" the execution pipeline when enough tuples have been produced or the predefined timeout is triggered. The main problem with this approach is that the *first* N results are returned, where *first* is defined arbitrarily by the order in which the data is processed. This order can be either user defined (e.g., an ascending numerical order), or the order in which the data was appended to the relation, or, finally, defined by an index for fast retrieval. In all cases, such a cut does not necessarily produce representative results for the entire data population, but merely the lucky N first tuples. Moreover, in the presence of blocking operators, such as *'sort'*, *'group by'*, etc., all data has to be read to produce the correct answer, and thus the pipeline cannot be cut. Query processing in SciBORQ is much different in that respect. The parameters of impressions are defined and maintained during updates, such that SciBORQ always guarantees an upper limit on time execution while producing results that are sampled from the entire database. In such an architecture the equivalent query with a `LIMIT 100` clause will not return the first 100 results, but the 100 results satisfying the impression. In the SkyServer example, instead of finding all objects near an area of the galaxy by evaluating the function `fGetNearbyObjEq` against the entire `PhotoObjAll` fact table, and then returning only a few results, the function is evaluated against an impression. If the number of results cannot be obtained from that impression, query processing may continue to a lower level that contains more sampled tuples from `PhotoObjAll`.

3.3 Impressions Construction

Impressions are deployed either as part of a database loading step or extracted from an existing database. In the first case, they are constructed with little overhead during the load phase, without the need to visit the base tables after the data is stored. The construction algorithms reside in the load process, considering each tuple as it is being loaded, much like a stream, and deciding if it should be part of an impression or not. Because daily ingests of new data are common in scientific data warehouses, the algorithms for creating an impression also support incremental updates.

The incremental construction of impressions follow the *reservoir algorithms* paradigm [24]. Reservoir algorithms have a) a fixed capacity of tuples that can fit in the sample, b) process the data sequentially, and c) each tuple has the same probability of being part of the sample. The size of the sample is kept constant by throwing out a random tuple to make room for a newly arrived one. Figure 2 outlines the general reservoir algorithm for maintaining a sample of size n. The decision to include or not a tuple in the sample is equal to flipping a coin with probability of acceptance $\frac{n}{cnt+1}$, where cnt is the number of tuples seen so far. Our algorithms stress the definition of reservoir algorithms, since in SciBORQ tuples are not chosen uniformly.

```
populate the sample smp with the first n tuples;
while (tpl := block_until_next_tuple())
      rnd := random();
      if ((D*rnd) < k)
         smp[floor(n*rnd)] := tpl;
      end
end
```

Figure 3: Last Seen Impression construction

Scientific observations have a strong temporal component. It is often more important to retain recent tuples than ones that have been investigated several times already. This leads to a *Last Seen* focused impression, where tuples that were recently added have a greater probability of being retained. To achieve this, instead of picking a tuple with probability $\frac{n}{cnt+1}$, we use the fixed probability $\frac{k}{D}$, where D can be tuned to be close to the expected daily ingest of new tuples, and $k = n$ if only new tuples are desired, or $k < n$ for a ratio of $\frac{k}{n}$ new tuples in the sample. In such a strategy, older tuples have a bigger chance of being thrown out from the reservoir. Figure 3 outlines this algorithm. The *Last Seen* approach is useful in cases where observations have a timestamp which is used in query predicates.

The second strategy for determining the scientist's interest is based on a more complex infrastructure of query logging. Every query ran against the complete database touches a subset of the base tables that are relevant to the data exploration. An approach would be to keep this set as an impression. The MonetDB *recycler* component already facilitates this functionality [13]. Here we seek an algorithm such that the probability of keeping a tuple is proportional to the *distance* of the values of that tuple from the values requested by the query workload. For each predicate of a query, the requested values are logged in histograms. These histograms do not contain the entire value space of an attribute, but only a portion. Given the workload knowledge, for each ingested tuple a weight is calculated and used to *bias* the sample towards the tuples with higher weight. In the next section we present the essentials of *biased sampling* and how the weight is calculated.

Finally, we can incorporate biased sample construction across many-to-many joins and foreign key joins by following each join path [3], or by using weighted sampling [4]. However, due to the special nature of impressions (i.e., incremental and adaptive biased sampling), these traditional sampling techniques have to be adapted to *wait* for the joining tuples to arrive during subsequent loads.

4. BIASED SAMPLING

Biased sampling is achieved by assigning weights to tuples such that those that belong to areas of past interest have a higher probability to be part of an impression than other, irrelevant ones. Intuitively, the upside is that queries that target the area of interest have tighter error bounds. The downside is that the confidence of queries that span widely outside of these areas is lower. Assigning weights to the probability of picking an item leads to a *non-central hypergeometric distribution*. Specifically, our setting is described by the *Fisher's non-central hypergeometric distribution* [6]. These mathematical tools provide the theory to calculate the variance, the mean, and the support function of the biased sample.

Biased sampling is steered by the observed interest in the data. This is achieved by first identifying the attributes of the data that contain relevant scientific observation values rather than annotations or metadata. In the SkyServer setting of Figure 1, these attributes, for the main fact table `PhotoObjAll`, are for example `ra` and `dec`, which give the position of the observed objects in the sky. They appear as parameters of the `fGetNearbyObjEq` function. For many of the queries in the workload, these attributes are part of the `WHERE` clause. Given a query workload – which is defined over a period of time or over a predefined number of queries – the *predicate set* is the set of all values of the interesting attributes that are requested by the queries. During incremental load of data into the `PhotoObjAll` fact table, tuples are sampled with a bias to the areas of the sky that previously appeared in the predicate set.

```
struct histo_stats{int c=0;
                   float m=0;
                  } hs[β];
N = 0;
while (v := block_until_next_value())
      N++;
      i := floor((v-min)/w);
      hs[i].c++;
      hs[i].m=(hs[i].m×(hs[i].c-1)+v)/hs[i].c;
end
```

Figure 5: Histogram maintenance over the predicate set

The values in the predicate set are regarded as points that *suggest* the entire distribution of values of interest. A *kernel density estimator (kde)* is used to estimate this interest. Kernel density estimators have been used to approximate the distribution of a sampled space [20]. They are smoother than histograms because they avoid rounding errors, and there is no dependency on the endpoints or the width of the bins of a histogram. Moreover, since they are continuous, and not discrete, they give a better view of the neighbour area of the observed values. Assume a set of N data points $x_1, \ldots, x_N$ as they appear in the predicate set of a query workload. The kernel density estimator estimates the expected total workload and is given by the function:

$$\hat{f}(x) = N^{-1} \sum_{i=1}^{N} K_h(x - x_i)$$

where $K_h(\cdot) = h^{-1}K(\cdot/h)$ where K is a kernel function and h the bandwidth. A common choice of K is the standard normal (Gaussian) distribution $\phi(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$. Function $\hat{f}$ is an estimator of the density function f of requested values, given N data points.

Figure 4 depicts two equi-width histograms that correspond to the distribution of 400 values as observed in the predicate set for attributes `ra` and `dec`. An important parameter is the choice of h, called the *bandwidth* of the kde. The red lines of Figure 4 show the density function estimation of the values in the histograms, as approximated by function $\hat{f}$ with a carefully chosen bandwidth. Notice that a large h will *oversmooth* the distribution (green lines in Figure 4), while a small h will *undersmooth* (blue lines in Figure 4). Choosing the correct approximation for the bandwidth h is hard and has been an area of intense research [14]. Moreover, computing $\hat{f}$ for a new value x involves re-iterating over all observed values $x_1, \ldots, x_N$. This implies that for every newly ingested tuple t_{new} the computation of $\hat{f}(t_{new})$ involves reading all N previously observed values of the predicate set. We adjust the kde to our setting to overcome these shortcomings as follows.

The first step is to maintain statistical information of equi-width histograms for the attributes of interest to the scientific exploration. These values are exactly the ones requested by the queries of the workload and not the entire value domain. For the previous example of SkyServer and attributes `ra` and `dec`, we maintain statistics for two histograms [3]. These histograms are different from the

[3]multi-dimensional histograms are more attractive, but for simplicity of the example we use two distinct histograms. Alternative ap-

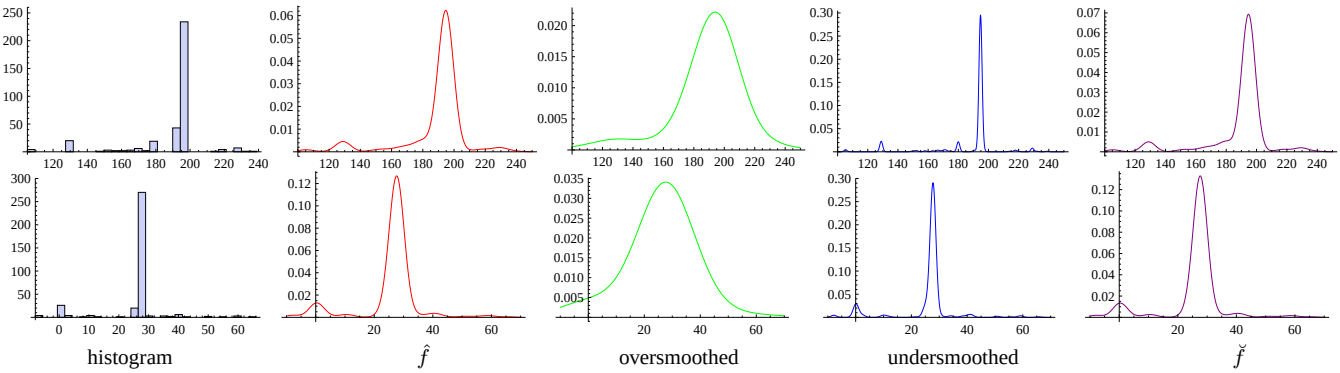


Figure 4: 1st row is for predicate `'ra'` and 2nd row for `'dec'`

ones shown in Figure 4, since they are not fully materialised as the figure suggests. Only the statistical aspects of the histograms are needed: the number of values that fall in a specific bin and their mean value. More specifically, the domain of each attribute is divided into β equal-width bins. The width is denoted by w. For each bin $b_i, i \in \{1, \ldots, \beta\}$ two values are maintained: the count c_i that corresponds to the number of values that fall in bin b_i, and the mean m_i that is defined to be the mean of all values belonging to the same bin b_i. Figure 5 outlines the code of maintaining the statistics of the bins of a histogram build over the requested values of one attribute, i.e., its predicate set. The min value of the domain, the width w, and number of bins β are considered to be known beforehand. The variable N contains the total number of values that have been observed in the predicate set.

The statistics of the histograms provide the means to determine the distribution of the interest, and based on them, a weight is assigned to each newly appended tuple. We adjust the kde function to consider only the mean values m_i of the β bins multiplied by the count c_i instead of iterating over all observed values $x_1, \ldots, x_N$. The resulting estimator function is now defined as:

$$\breve{f}(x) = \frac{1}{N \times w} \sum_{i=1}^{\beta} c_i \times \phi\Big(\frac{x - m_i}{w}\Big)$$

where β is the total number of bins, w the width of the bins, and c_i the count and m_i the mean of the i-th bin. Since $\beta \ll N$, and β is fixed, $\breve{f}(x)$ can be computed in constant time. Also

$$\int K_w(u) = 1 \text{ and } \textstyle\sum_{i=1}^{\beta} c_i = N \Rightarrow$$
$$\textstyle\int \sum_{i=1}^{\beta} c_i K_w(u) = N \times \int K_w(u) = N \Rightarrow$$
$$\textstyle\int \breve{f}(x) = N^{-1} \int \sum_{i=1}^{\beta} c_i K_w(u) = N^{-1} \times N = 1.$$

Thus, function $\breve{f}$ is an estimation of the probability density function that describes the relative likelihood for value x to occur in the predicate set. The purple line of Figure 4 shows the density function computed with $\breve{f}$. It is almost identical with the estimation from $\hat{f}$, while it only iterates over a few constant number of bins, and the bandwidth is always equal to the width of the bins.

Assume a newly ingested tuple t_{new} during incremental load. For simplicity, assume also that t_{new} has only one attribute of interest [4]. A weight is assigned to tuple t_{new} equal to $\breve{f}(t_{new})$. We bias the sample by making the probability of choosing this tuple for an impression proportional to $\breve{f}(t_{new}) \times N$. Function $\breve{f}$ estimates the frequency of appearance of value x in the predicate set. Thus, the more frequent the value, the larger the product $\breve{f}(t_{new}) \times N$, and the higher the probability of choosing t_{new}.

Function $\breve{f}(x)$ can be used in the reservoir setting. An impression has always a predefined size n, thus for a uniform sampling a tuple is accepted with probability n/cnt, where cnt is the number of tuples in the database. For biased sampling the probability of accepting a tuple t is $\breve{f}(t) \times N$, and by normalising this with the desired size of the impression leads to the following probability

$$P(\text{accept } t) = \breve{f}(t) \times N \times \frac{n}{cnt}$$

where N is the size of the observed predicate set, n the size of the desired impression, and cnt the number of tuples in the database. Figure 6 details the biased sampling reservoir algorithm. After a tuple is accepted, another randomly chosen one is thrown out from the sample to make room for the new.

The leftmost histograms of Figure 7 show the distributions of the values of the base data of SkyServer answering the queries used in Figure 4 (more than 600.000 tuples). We create two impressions of 10.000 tuples for each attribute: one based on uniform sampling (red histograms of Figure 7), and one based on biased sampling (purple histograms of Figure 7) steered by the interest shown in Figure 4. The impression created with bias contains many more tuples from the areas of interest, achieving a better representation of data around the focal points.

```
populate the sample smp with the first n tuples;
cnt := n;
while (tpl := block_until_next_tuple())
      cnt++;
      rnd := random();
      if ((cnt*rnd) < (n*N*f̆(tpl)))
         smp[floor(rnd*n] := tpl;
      end
end
```

Figure 6: Biased Sampling reservoir algorithm

5. RELATED WORK

Various techniques on how to construct data synopses, keep summary statistics, and obtain data samples have been proposed in the past [5, 8, 10, 19, 23]. A topic of intense research is how samples can be adjusted to support correlations between join attributes [3, 4, 18, 21]. SciBORQ is also aiming towards efficient inter-column and inter-table sampling. Self-tuning samples were proposed by ICICLES [7]. The results of a query are regarded as newly ingested

proaches are part of future research.

[4] multiple attributes in the same tuple are dealt either with multi dimensional histograms or with a combine function $c(t_{new}) = \breve{f}(t_{new}.att_1) \circ \cdots \circ \breve{f}(t_{new}.att_m)$.

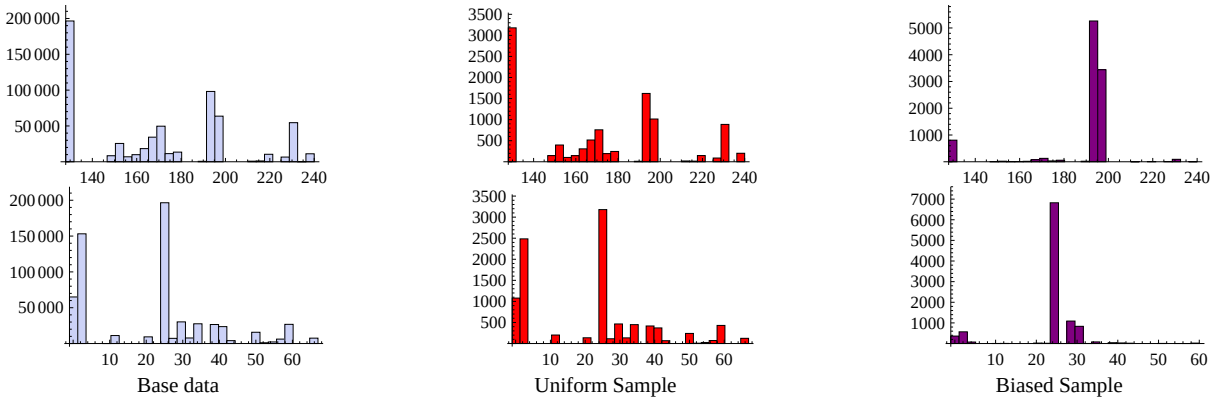


Figure 7: 1st row is for predicate `'ra'` and 2nd row for `'dec'`

data, and the sample is updated accordingly. We intend to investigate this technique for SciBORQ also: a side-effect of a query evaluation is to update an impression using query results. Another tuning approach for histograms was proposed in [1], where the feedback of a query is used to refine histograms to better resemble the base data. Gibbons and Matias envisioned a system similar to ours in their motivation for concise samples [9]. This led to Aqua [2], a system for providing approximate answers to aggregate queries. Both of those are close to our vision, however, they lack the multi-layer design and adaptive biased sampling of SciBORQ that allows the system to adjust the quality guarantees during query execution.

6. SUMMARY AND FUTURE WORK

In this paper we described a new data exploration architecture for science data warehouses. The key observation is that in most situations a fraction of the data would be a good starting point, provided that the error and processing time bounds are within the requested range.

Biased sampling is a valuable alternative to the predominant uniform sampling techniques, since more data from the areas of interest are sampled. The architecture of SciBORQ is unique in its multi-layer approach, providing the means for runtime execution and (re-)optimisation that will guarantee the desired error bounds, even if they are thrown off track due to correlations. We intend to investigate the theoretical error margins for biased sampling based on known mathematical tools [6] and their propagation through the fundamental query processing operators, and to incorporate multi-dimensional histograms for sampling over relations. Finally, we will explore the connection between query processing time, the size of an impression, and the consumption of resources.

7. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *Proc. of the ACM SIGMOD*, 1999.

[2] S. Acharya, P. Gibbons, and V. Poosala. Aqua: A Fast Decision Support System Using Approximate Query Answers. In *Proc. of the 25th VLDB*, 1999.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join Synopses for Approximate Query Answering. In *Proc. of the ACM SIGMOD*, 1999.

[4] S. Chaudhuri, R. Motwani, and V. Narasayya. On Random Sampling over Joins. *ACM SIGMOD Record*, 28(2), 1999.

[5] Daniel Barbara et al. The New Jersey Data Reduction Report. *IEEE Data Eng. Bull*, 20(4), 1997.

[6] A. Fog. Sampling Methods for Wallenius' and Fisher's Noncentral Hypergeometric Distributions. *Communications in statistics, Simulation and Computation*, 37(2), 2008.

[7] V. Ganti, M. L. Lee, and R. Ramakrishnan. ICICLES: Self-Tuning Samples for Approximate Query Answering. In *Proc. of the 26th VLDB*, 2000.

[8] M. Garofalakis and P. B. Gibbons. Probabilistic wavelet synopses. *ACM-TODS*, 29(1), 2004.

[9] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. of the ACM SIGMOD*, 1998.

[10] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM-TODS*, 27(3), 2002.

[11] T. Hey, S. Tansley, and K. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[12] S. Idreos, M. Kersten, and S. Manegold. Database Cracking. In *Proc. of the 3rd CIDR*, 2007.

[13] M. Ivanova, M. Kersten, N. Nes, and R. Goncalves. An Architecture for Recycling Intermediates in a Column-store. In *Proc. of the ACM SIGMOD*, 2009.

[14] M. C. Jones, J. S. Marron, and S. J. Sheather. A Brief Survey of Bandwidth Selection for Density Estimation. *Journal of the American Statistical Association*, 91(433), 1996.

[15] R. A. Kader, P. Boncz, S. Manegold, and M. van Keulen. ROX: run-time optimization of XQueries. In *Proc. of the ACM SIGMOD*, 2009.

[16] S. Manegold, M. Kersten, and P. Boncz. Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct. In *Proc. of the 35th VLDB*, 2009.

[17] MonetDB. http://monetdb.cwi.nl.

[18] M. Muralikrishna and D. J. DeWitt. Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries. In *Proc. of the ACM SIGMOD*, 1988.

[19] F. Olken and D. Rotem. Simple Random Sampling from Relational Databases. In *Proc. of the 12th VLDB*, 1986.

[20] E. Parzen. On Estimation of a Probability Density Function and Mode. *Annals of Mathematical Statistics*, 33(3), 1962.

[21] V. Poosala and Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proc. of the 23rd VLDB*, 1997.

[22] A. Szalay and R. Brunner. Exploring Terabyte Archives in Astronomy. Invited talk at the IAU Symposium in Baltimore, 1996.

[23] Viswanath Poosala and Venkatesh Ganti and Yannis E. Ioannidis. Approximate Query Answering using Histograms. *IEEE Data Eng. Bull*, 22(4), 1999.

[24] J. S. Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1), 1985.