# Efficient Exploration of Large Scientific Databases

**Etzard Stolte, Gustavo Alonso**

Department of Computer Science
Swiss Federal Institute of Technology (ETHZ)
ETH Zentrum, CH-8092 Zürich, Switzerland
{stolte, alonso}@inf.ethz.ch

## Abstract

One of the most challenging aspects of scientific data repositories is how to efficiently explore the catalogues that describe the data. We have encountered such a problem while developing HEDC, HESSI Experimental Data Center, a multi-terabyte repository built for the recently launched HESSI satellite. In HEDC, scientific users will soon be confronted with a catalogue of many million tuples. In this paper we present a novel technique that allows users to efficiently explore such a large data space in an interactive manner. Our approach is to store a copy of relevant fields in segmented and wavelet encoded views that are streamed to specialized clients. These clients use approximated data and adaptive decoding techniques to allow users to quickly visualize the search space. In the paper we describe how this approach reduces from hours to seconds the time needed to generate meaningful visualizations of millions of tuples.

## 1 Introduction

Scientific databases storing continuous observations of natural phenomena will soon store petabytes of data [26, 11]. Finding relevant data points is a major challenge in these systems. We have faced this problem during the development of HEDC (the HESSI Experimental Data Center), a data repository that will store the observations of the recently launched HESSI satellite (High Energy Solar Spectroscopic Imager). Simply stated, HESSI observes the sun at different energy ranges and stores a count of the number of electrons detected as a function of time. HEDC stores this data stream (*raw data*) and builds catalogues indexing *events* of interest (*derived data*) such as solar flares or gamma ray bursts. The catalogues will also serve to record user analyses so as to facilitate sharing of scientific results. It is expected that these catalogues will soon contain several million tuples. Although HEDC is not an exceptional case, existing systems provide little or no support for browsing such catalogues, expecting users to explore them by posing queries. Given the data volume involved, this is simply not realistic and often leads to valuable data being lost in the system.

Several approaches have been proposed to tackle this problem [15]. Unfortunately, we are not aware of any system capable of coping with the volume of data involved in HEDC. For instance, [16, 24] mention a scalability of up to one million tuples, but provide no indication of the time necessary to compute the visualizations. In comparison, HEDC requires a solution scalable to 10 million tuples per user request, unlimited size of the entire catalogue, and response time below a few seconds for many simultaneous users.

In this paper we present such a solution. The basic idea is to reorganize the catalogues as a number of multi-dimensional arrays and allow users to specify ranges in any of the dimensions. Based on these ranges the information is then presented in a compact and efficient manner using *density* (number of tuples per bin) and *extent* (location and extent of each tuple or cluster of tuples) plots. To achieve the necessary performance for interactive use, we also implement several important optimizations. First, the arrays are pre-processed and sorted according to the most relevant attributes. Then they are partitioned across the dimensions to form the equivalent of materialized views. Since the partitioned views tend to be large, we encode them using a wavelet transformation. Decoding takes place at the client side to minimize the load at the server (otherwise interactive exploration would require a very powerful server). To further speed up the generation of the plots, the client works on approximated and aggregated versions of the original data.

Using this approach, we have been able to reduce the time it takes to create a 2-dimensional (*2D*) density plot for 1 million tuples from 22 minutes (using conventional tools) to 1.6 seconds. We have also tested the scalability of the system to up to 8.4 million tuples per plot, taking only 5 seconds to dynamically generate such plots. Such a performance improvement is par-

**Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002**

ticularly relevant because the techniques we propose are not specific to HEDC or astrophysics data. They can be used with a wide variety of data and coupled with many different visualization tools. Therefore, we believe the paper makes crucial contributions both towards more efficient scientific databases and a better understanding of how to explore large data spaces using approximated and wavelet transformed data.

In what follows, we first introduce the context and our goals and discuss related work (Section 2). We then describe the solution in detail (Section 3) and provide an extensive experimental evaluation (Section 4). The paper concludes with a discussion of the main ideas (Section 5), a brief description of how these ideas have been implemented (Section 6), and the conclusion (Section 7).

## 2 Motivation and Goals

### 2.1 HESSI and HEDC

The HESSI, High Energy Solar Spectroscopic Imager, satellite[1] was launched on the 5th of February, 2002. Its goal is to provide the data necessary to improve our understanding of solar flare physics. Its only scientific instrument is an array of detectors that continuously record the energy and time of impact for each photon reaching the detectors. Data is produced at a rate of 2.0 GB per day and the duration of the mission is planned to be 2 to 3 years. The observations are first buffered at the satellite and then forwarded to a ground station at pre-established intervals. Before publication in the form of a collection of files, this raw-data stream is calibrated, analyzed for possibly relevant events, segmented along the time axis, and packaged into units of roughly 40MB. For each relevant event detected, some summary data and a number of *data products* are generated which are then attached to the corresponding raw-data unit. This extra information constitutes the *basic catalogue* of events. In about 24 hours, the raw-data is sent to HEDC[2] and two other repositories at the Space Science Lab (UC Berkeley) and at NASA's Goddard Space and Flight Center (GSFC).

HEDC, the HESSI Experimental Data Center, has been built to optimize the scientific return of the HESSI mission by providing better exploration tools and a more sophisticated data repository. Thus, when the raw-data units reach HEDC, they are once more searched for relevant events. This time with algorithms that detect a wider range of events such as solar flares, gamma ray bursts or quiet periods. The result of this new search is the *standard catalogue*, which is also used to record any analyses later performed by users. Each entry in the catalogue is a tuple indicating the type of analysis performed, its parameters, and additional information used to classify the entry.

Towards the end of the HESSI mission, the raw-data volume will be about 1.2 TB with at least 2 TB of user generated data products. The basic and standard catalogue are expected to contain several million tuples. Currently, more than 210 GB of HESSI data and around 25 GB of measurements taken by the Phoenix-2 Broadband spectrometer (Bleien, Switzerland) are available at HEDC.

### 2.2 Exploring Large Tuple Catalogues

Searching for relevant events in large scientific databases poses two basic problems. The first one is the transformation of raw data into scientifically meaningful information (in [25] we show how to speed up such procedure). The second problem is how to navigate the catalogues summarizing the scientifically useful information (i.e., analyses already performed). Since working with the raw data is a time consuming, manual process, many users rely instead on the catalogues as the starting point for data exploration. A good approach to visualize large catalogues, suggested to us by HEDC users, is to provide a graphical representation of how the existing analyses are distributed over the parameter space. The idea is based on the experience that although not all scientists share an identical notion of what is relevant, a general consensus exists. As a result, analyses tend to cluster around well defined regions of the data. Being able to quickly locate such regions and ascertain their characteristics would be a great help when exploring the available data. Following this idea, HEDC supports two forms of visual representation: *density* and *extent* plots.

### 2.3 Density Plots

HEDC currently supports density plots with up to 3 dimensions. Each dimension can be used to represent either a single attribute (e.g., peak rate) or two correlated attributes specifying a range (e.g., start and end time of an observation). In a density plot, the range associated with each dimension is divided into intervals of the same size. These intervals define one-, two-, or three-dimensional bins depending on the number of dimensions involved. For each bin, we count the number of tuples that correspond to the intervals defining the bin. In this regard, density plots are not unlike histograms except that both the ranges and intervals in each dimension are not known in advance. For presentation purposes, density plots can be post-processed by a rendering algorithm (e.g., to interpolate boundaries, introduce transparencies, etc). Figure 1 shows a contour rendered 2D density plot for 128 bins and 4 attributes (2 attributes per dimension). The plot shows the number of tuples available in each bin (darker color indicating more tuples), thereby allowing users to quickly identify areas where relevant data is located (*hot-spots*).

[1] http://hesperia.gsfc.nasa.gov/hessi/
[2] http://www.hedc.ethz.ch/

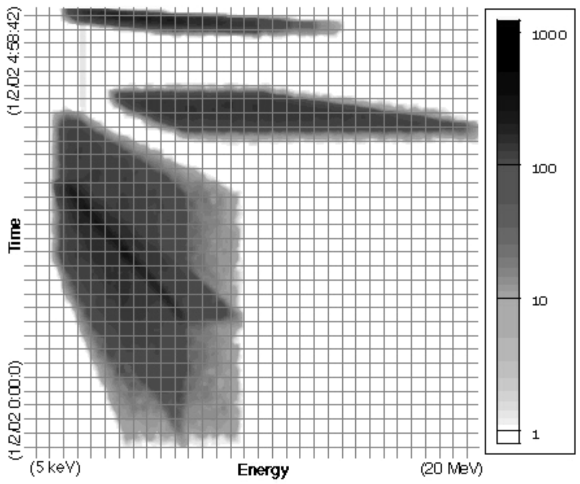

Fig. 1: Density plot (128*128 bins, $524{\cdot}10^3$ tuples).

Excluding the (optional) rendering step, the runtime behavior of density plots is independent of the number of hot-spots, produces little network traffic, and requires only minimal processing capacity on the client side. The challenge with density plots is how to generate them fast enough for the exploration procedure to be interactive even if they involve millions of tuples.

### 2.4 Extent Plots

Density plots aggregate information on a per bin basis. Extent plots also work with the same type of bins but do not aggregate the information. Every tuple falling into a bin is represented (a tuple can cover several bins). For efficiency purposes, if the number of elements exceeds a given threshold, the extent plot might group tuples and display them as a single object (see section 3.6). Figure 2 shows such a 2D extent plot based on 2 attributes per dimension, rendering every tuple as a transparent square. The intensity of the square indicates the number of tuples it represents. As a result of this rendering approach, areas with many overlapping tuple ranges appear as brighter patches.

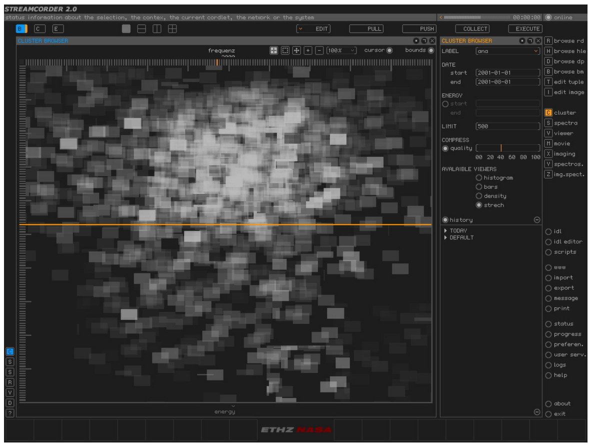

Fig. 2: Extent plot over energy and time

In contrast to density plots, extent plots require the actual tuple data. Their runtime behavior (for simple algorithms) is thus roughly proportional to the number of tuples to be processed. As a result, extent plots tend to put a high load on server and network. As with density plots, the problem then arises how to dynamically construct extent plots so that they can be used interactively.

### 2.5 Related Work

Using graphical representations to navigate data is not a new idea. In our case, the challenge comes from the volume of data involved and the requirement to provide an interactive solution. In this regard, tools for exploratory data visualization already exist. Projects such as VQE [7], DEVise [18], VisDB [16], Tioga [27] or Polaris [24] support interactive database exploration, often through *visual queries*. Systems like Spotfire [23] and XGobi [6] also allow brushing and zooming, which can be used to refine the search. Most tools also provide some form of dimension reduction to speed up processing (e.g. [16, 23]). Unfortunately, neither these systems, nor any of the techniques for which published experimental results exists can cope with the requirements of HEDC. The largest data-sets used for visualization we could find were in [15, 16], where a theoretical limit of 1 million tuples is mentioned and experimental results are shown only for up to $65{\cdot}10^4$ tuples. In [24] a test with 6 million tuples is mentioned, but no performance measures are given.

Materialized views are a well know technique to speed up data access [1, 4, 5, 17]. To provide different quality and resolution levels on views we use wavelets. Alternatively, one could use methods such as sampling [9, 10], histograms [13], or techniques such as quadtrees or octrees [8]. However, the quality of wavelet encoded views is generally higher for range-sum queries over dense [29] and sparse data cubes [28]. Further work has shown that view updates are possible when wavelets are being used[19]. [3] generalized this work to include aggregated and non-aggregated views. Wavelet encoding of multi-dimensional arrays has already been implemented [22], but without considering the overall costs, especially the time spent for decoding. There is also some work on data encoding [2] as a means to find and analyze the data but only applied to individual data-sets. Other work has focused on query processing inside wavelet space [3] or through some density function [21]. However, and to our knowledge, there has been no work done in the area of interactive database visualization equivalent to the one reported in this paper.

Histograms capture distribution statistics in a space efficient fashion. They have been found to work well for numeric value domains, and have long been used to support cost-based query optimization [12], approximate query answering [3, 21] and data mining [14].

In principle, density plots are histograms but unlike other approaches, such as [19, 28], our technique is suitable for any quality up to lossless reconstruction of the original attribute vector, with flexible bin size and a dynamic selection of attributes.

## 3 Our Methodology

This section describes how to visualize large tuple collections. The starting point is to combine selected attributes (3.1) of pre-processed tuples into low-dimensional views (3.2), partition and encode these views (3.3), and upload them to the DBMS (3.4). During query processing, the DBMS matches the query attributes with the appropriate view segments, which the clients download (in part or full), decode (3.5) and process for visualization (3.6).

### 3.1 Select Fields for Exploration

HEDC supports interactive exploration based on certain *attributes* of the tuples in the catalogues. The catalogues reference astro-physical analyses, meta-data describing these analyses, and collections thereof. The actual information is distributed across 21 tables and encompasses several hundred fields. For simplicity in handling, exploration of multiple attributes (2D and 3D plots) is restricted to attributes on the same table. This is not really a limitation since different tables contain different types of events that are typically independent of each other. The decision to include an attribute into a view is based on user requirements and attribute type. For instance, in the context of HEDC, exploration is not performed based on ordinal or textual attributes and, therefore, these can be ignored for this purpose. We currently use 8 attributes for exploration purposes (peak rates at two different energy levels, start and end time of the observation, minimum and maximum energy levels observed, and x/y position of the satellite with respect to the sun). Depending on future user feedback, this number might rise to a maximum of 20 to 30 attributes. Except for the peak rates, these attributes are pair-wise related and each pair can be used to define an interval over the corresponding dimension.

### 3.2 Construct Views

Instead of using the tuples directly, we reorganize them into *views*. A view is a two dimensional array of size $\mathcal{T} * \mathcal{A}$, where $\mathcal{T}$ is the number of tuples included in the view, and $\mathcal{A}$ the number of attributes included in the view. In a view, each cell $i, j$ corresponds to the value of the $j^{th}$ attribute in the $i^{th}$ tuple. As we will later discuss, the view is sorted according to one of the attributes to improve compressed storage size and speed up visualization. Thus, a view is a selection over the relevant part of the catalogue eliminating all attributes that will not be used for exploration.

Our wavelet encoder expects all attribute values to be representable as positive valued 27-bit integers without overflow. This means that negative values need to be shifted into the positive range. Larger integer values need to be encoded into two cells of the view, with the corresponding increase in storage and processing time. The same technique is needed to retain precision in floating point numbers. In HEDC, this problem is greatly simplified by the nature of the data. Negative integers can be simply shifted within the $[0..2^{27}]$ range without creating overflow. Similarly, the two floating point attributes used (the two peak rates) can be scaled into a single cell without any measurable loss of precision. In all cases, the values are scaled back into their original domain before creating the corresponding plot. Overall, the increase in storage space due to view construction is expected to remain below 30% of the total RDBMS size.

### 3.3 Partition and Encode Views

To improve performance and facilitate handling, the views are partitioned along the primary sort key, resulting in *view segments*. Optimal segment size depends on a multitude of system parameters. For our system setup, we have found segments containing attributes of around 1 million tuples to be appropriate. This yields an average encoded segment size of, e.g., 2.2 MBs for a view with 6 attributes.

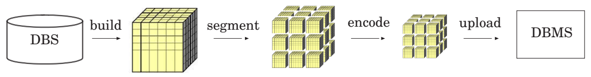

Fig. 3: View encoding steps.

To further optimize segment handling, we compress them. Compression takes place by encoding the segments using a wavelet transformation. A wavelet transformation was chosen as the best option given our requirements: lossless compression, high compression ratio, progressive transfer and fast and robust decoding. Even though the compression ratio depends on the actual attribute values being encoded, the other requirements hold true for any numeric data source. In this, we follow many others who use wavelet decomposition in signal processing [2], clustering [22], query processing [21, 3, 20] or indexing [30]. To implement the transformation, we licensed the source code of a commercial wavelet package[3]. It supports variable bit size representations for all data processing operations (quantization, line-based DWT, transformations), file seeking (to extract the coefficients of selected parameter sub spaces) and arbitrary segment sizes of up to $2^{32}$ bytes in size. It does not require explicit signal extension at boundaries and is thus memory efficient. The C++ source code is a platform independent implementation (gcc greater or equal to version 3.0). For the

[3]http://maestro.ee.unsw.edu.au/~taubman/kakadu/

integration we made minor adjustments and wrapped the library in JNI (Java Native Interface) calls.

### 3.4 Upload View Segments

Views are created by batch jobs and stored in external files. Each file contains a view segment. For each one of these files, context and summary information is extracted and stored in reference tuples (indexed according to the summary information). During query processing, these indexes are used to find the proper files. We mark segments as outdated when 1% of its tuples have changed. To determine when to update a view, a daily batch job counts all insertions, deletions and edits for each segment. View update is done off-line, as part of the periodic updates performed on the system (mainly, when new data is delivered from the satellite).

### 3.5 Decode View Segments

The visualization step occurs at the client. Thus, the encoded view segments need to be shipped to the client and decoded there before the corresponding plot can be constructed (Figure 4). During the decoding process (before building the plot) we adapt the *quality* and also the *resolution* of the view. Both adaptations are intrinsic to wavelet decoders and greatly influence resource requirements as well as performance.

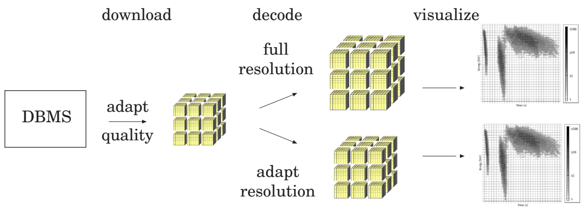

Fig. 4: Resolution adaptation during decoding.

Quality adaptation takes place by varying the number of coefficients used to reconstruct the view during decoding. The resulting array size is the same as the original but the contents of each cell are an approximation of the real value. A view reconstructed using 100% of the coefficients is a full-quality view, otherwise it is a low-quality view. The advantage of quality adaptation is that decoding and transfer times are significantly smaller if only a fraction of the coefficients are used.

Resolution adaptation takes place by varying the size of the decoded view. The decoded view can be made as large as the original (full-resolution, i.e., the resulting array has the same number of cells as the original) or it can be made smaller by multiples of 2 (low-resolution). The reduction for a plot over $\mathcal{T}$ tuples for $\mathcal{B}$ bins is controlled by the resolution factor $r = [0..\log_2(\frac{\mathcal{T}}{\mathcal{B}})]$, with $r = 0$ implying full-resolution. In a low-resolution view the original size of $\mathcal{T} * \mathcal{A}$ is reduced to $\frac{\mathcal{T}}{2^r} * \mathcal{A}$. The reduction occurs through an averaging of groups of $2^r$ tuples.

Resolution adaptation is only possible if the visualization routine allows inputs of variable size. If that is the case, low-resolution views typically result in considerable faster processing times both for decoding and for constructing the plots. In addition, the resulting view is much smaller. For example, uncompressed 2D extent plots with 2 attributes per dimension over $8.4{\cdot}10^6$ tuples require 174 MBs of RAM, whereas the approximated plot at 1/32th the resolution needs only 5.5 MBs of RAM.

### 3.6 Calculate Plots

Once the view has arrived at the clients and has been decoded, the corresponding plot can be calculated. For reasons of space, we discuss only how this is done for density plots; extent plots use a similar algorithm.

Our algorithm for producing a density plot computes the values for equi-distant bins directly from the decoded view segment. The process is straightforward and the only complication is due to having to take resolution adaptation into account. Given below is the simplified pseudo code for computing 3D density plot (measure: count) with bin size $b1 * b2 * b3$ over 3 attributes $(a1, a2, a3)$ with resolution $r$ calculated over view $v$. The 3D array *count* contains the data to plot.

```
for (int i=0; i<number of tuples; i++)
    p1 = v[i,a1] / b1;
    p2 = v[i,a2] / b2;
    p3 = v[i,a3] / b3;
    count[p1,p2,p3] += 2^r;
```

The algorithm proceeds as follows. For every row in the view, the bin position is calculated and the bin value updated. The bin position $(p1, p2, p3)$ is determined by dividing the 3 attribute values by their respective bin sizes. The count is determined by the resolution factor, so that at full-resolution the count is incremented by $2^0$, and by $2^r$ for low-resolution views since every entry represents $2^r$ actual tuples.

The runtime behavior of this algorithm is linear with respect to the number of array elements. The effect of low-resolution views is obvious in this case as it reduces the number of rows in the views by a factor of $\frac{1}{2^r}$. In theory, the resolution could be reduced until the number of rows in the decoded view equals the number of bins in the plot. This would yield the fastest possible way to calculate a plot from the encoded view, although the approximation error would be very high. This can be done with 1D plots. For 2D and 3D plots it does not work because the non-sorted attributes are less correlated and the averaging becomes random.

As we use the same technique for density as well as for extent plots, their runtime behavior is identical. Resolution adaptation applied to extent plots determines the maximum number of objects to be plotted.

With $8.4 \cdot 10^6$ tuples and a resolution of 1/32th, for example, a maximum number of $262 \cdot 10^3$ data points can be drawn. In practice though, the plotting algorithm will further condense these to around 1024 data points, which in the unlikely case of an even distribution of the tuples across the parameter space, would each represent 8192 tuples. For instance, a 2D plot over time and energy, size and width of the square would represent the average time and energy values of the 8192 tuples included in the data point.

## 4 Experiments

In this section we evaluate the ideas proposed above. We first describe the experimental setup (4.1). We then motivate our choice of error-measures (4.2). As a first step in the evaluation, we consider different alternatives to our approach (4.3) before describing in detail the effect of using views (4.4), encoding these views with wavelets (4.5), and performing a quality adaptation (4.6) and a resolution adaptation (4.7). We also provide the adaptation parameters used in HEDC that allow visualization of any number of tuples up to 8 million in less than 5 seconds (4.8).

### 4.1 Setup

HEDC has been implemented using a 3-tier architecture on top of an Oracle 8.1.7 RDBMS and a number of analysis servers. HEDC can be accessed through either a Web based client using a conventional browser or a Java-based client, the *StreamCorder*. All the experiments described below have been performed with the StreamCorder, running on a Java 1.2.2 JVM on a dual Pentium II 450 MHz PC with 512 MB RAM, a FireGL graphics card, and a 100Mb/s Ethernet network connection to the server. Unless otherwise mentioned, all measurements represent the average for 100 test runs.

HEDC contained at the time of writing roughly $300 \cdot 10^3$ tuples created through processing data from other solar observatories. For the tests, additional data-sets with up to $8.4 \cdot 10^6$ tuples were generated and evaluated. Each generated data-set represents a realistic tuple distributions in attribute space. All tests access every tuple stored in the database (selectivity = 100%). The tuples in all data-sets are contained within the boundaries of the same attribute value ranges, and show the same three hot-spots. Around 10% of the tuples are not associated with one of the three hot-spots, and represent systematic background scans and investigations independent of the major events. Client side decoding and visualization processing was done on a dedicated computer. Visualization times do not include the optional post-processing, such as contour rendering (which adds a constant factor of about 1 minute to each plot).

### 4.2 Error Measures

The goal of the experiments is to determine the improvement in overall response time, and to measure the visualization error introduced. We measure the error of the approximated views and of the resulting plots as the quadratic norm over all $n$ view cells or all $n$ plot bins. Both absolute error ($E_a$)

$$E_a() \quad = \quad \frac{1}{n} \sum_{i=1}^{n} \|f^i - l^i\|^2 \qquad (1)$$

and the relative error ($E_r$)

$$E_r() \quad = \quad \frac{1}{n} \sum_{i=1}^{n} \left( \frac{\|f^i - l^i\|}{max(c, f^i)} \right)^2, \ c > 0 \qquad (2)$$

are calculated, with $f^i$ a full-resolution and $l^i$ the corresponding low-resolution attribute or plot vector. The absolute error is useful to document a trend, for example, to measure the impact of progressive quality- and resolution-adaptation on plot accuracy. Yet, to evaluate the actual impact of the approximation independent of the value range, the relative error is more helpful. In the case of approximated views and extent plots, the errors capture the difference with the full-quality and full-resolution views. In the case of the density plots, the error reflects the difference of a plot bin value to the actual bin value.

### 4.3 Standard JDBC Approach

As a baseline for the measurements, we use the approach followed by most existing systems. This is based on direct connections from the clients to the server through, e.g., a JDBC interface. As an alternative, we also consider a direct connection to the server from the client but using an export utility instead of the JDBC interface. In both cases, the data is located through the same type of queries. In the case of density plots, typically aggregation functions (such as *count*, *avg*, *min* or *max*) are requested for equi-distant bins along continuous attributes. A 1D density plot will then execute $\mathcal{B}$ range queries (one for every bin), a 2D plot will execute $\mathcal{B}^2$ queries, and so forth.

Let us consider the example of a density plot based on pairs of attributes over $b$ bins, with the 4 queried fields being start and end of the observation time and maximum and minimum energy. Our first naive attempt using standard *count*$(*)$ queries on indexed fields in combination with set operations took hours to finish a 128*128 plot over a million tuples. To optimize, we eliminated the full table access in exchange for creating and dropping temporary materialized views and subsequent index range scans. For our Oracle 8.1.7 installation, the following code fragment was the fastest SQL code we could devise:

```
DROP MATERIALIZED VIEW;
COMMIT;

CREATE MATERIALIZED VIEW temp AS
SELECT maxEng, minEng FROM hle_ana
     WHERE ( start >= '2002-01-01 00:16:40')
       AND   start < '2002-01-01 02:20:55'))
UNION
SELECT maxEng, minEng FROM hle_ana
     WHERE ( end >= '2002-01-01 00:16:40')
       AND   end < '2002-01-01 02:20:55')
);

SELECT count(*) FROM (
     SELECT rownum FROM temp
     WHERE (maxEng >= 2709) AND (maxEng < 3000 )
UNION
     SELECT rownum FROM temp
     WHERE (minEng >= 2709) AND (minEng < 3000 )
);
```

Figure 5 displays the time necessary to compute plots with 128 bins for 2 (1D) and 4 (2D) attributes using this SQL statement as a function of the selectivity of the query. Even the performance of this optimized SQL/JDBC code is short of interactive database visualization by several orders of magnitude. In the case of 2 attributes, a plot of only $65 \cdot 10^3$ tuples needs 1.4 seconds and 174 seconds for $8.4 \cdot 10^6$ tuples. Due to the quadratic runtime behavior of the underlying algorithm, processing time explodes in the case of 4 attributes from 2 minutes to 13.46 hours as the number of tuples increases. Extent plots are even more expensive to compute since, in addition to the query processing, the attribute data has to be downloaded to the client.

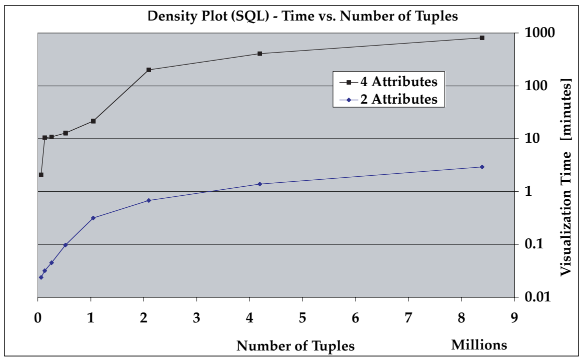

Fig. 5: 1D/2D density plot time vs. data-set size.

Given the amount of data involved, using the JDBC interface is certainly not the best option. An alternative often used is to export the data rather than extracting it through a query. For this purpose, we have used the Oracle Export Utility which, given a query, returns a file compressed in a proprietary format. For comparison purposes, we have used a 2D plot over $10^6$ tuples with 128*128 bins and 4 attributes. The results are shown in tables 1 and 2. Using the export utility, the overall time to compute a density plot is about half the time needed when using a JDBC interface, yet nowhere near the response time needed for an interactive exploration. The additional plot time for the export case is due to the need to extract the data from the file sent.

## 4.4 Server-Side Optimization: Views

An obvious optimization over these approaches is to use materialized views rather than computing them every time. Moreover, these views do not really need to be extracted using queries or an export utility. Since they need to be shipped to the client anyway, they can as well be stored in files. This adds a small penalty when several files need to be combined to produce the requested range but this penalty turns out to be negligible compared with other costs.

In a first approximation, we simply store the segmented views as gnu-zipped files. Using the same test data as for the JDBC and export utility cases, the advantages of using views materialized in files can be seen in tables 1 and 2. The use of a view increases plotting performance by a factor of 70 (6 seconds), as it involves merely locating the relevant view segment, downloading it to the visualization client, decoding it and then processing it. In the case of HEDC, locating the view segment and returning an URL to download requires 2 to 4 indexed queries, so that performance is very good (in our experience, much better than having the view stored in the database)

| Time[s] | sql/jdbc | export | view.gz |
|---|---|---|---|
| query | 440.45 | 264.02 | 0.25 |
| download | - | - | 2.30 |
| decode | - | - | 2.13 |
| plot | - | 1.31 | 1.31 |
|  | 440.45 | 265.32 | 5.98 |

Tab. 1: Times [s] to generate density plots ($10^6$ tuples).

For extent plots, the procedure is very similar except that the actual tuple data must also be sent to the client. This is extremely slow using the JDBC interface (22 minutes) (Table 2). In all cases, once the data is on the client side, the simple extent plot algorithm chosen for this example completes in less than 6 seconds. The same applies to the export tool except that the data is much more compressed and, therefore, the costs for downloading are smaller.

| Time[s] | sql/jdbc | export | view.gz |
|---|---|---|---|
| query | 440.45 | 263.85 | 0.25 |
| download | 820.9 | - | 2.30 |
| decode | - | - | 2.13 |
| plot | 5.56 | 5.56 | 5.56 |
|  | 1266.91 | 269.41 | 10.24 |

Tab. 2: Times [s] to generate extent plots ($10^6$ tuples).

With these results, it is clear that interactive exploration can only be done when using view segments materialized outside the database. Still, the approach used so far requires over 10 seconds for a extent plot with one million tuples. To obtain real interactivity, further optimizations are necessary.

## 4.5 Wavelet Encoded Views

Using wavelets allows to implement the quality and resolution adaptation necessary for interactive exploration. However, the fact that the view may contain several un-correlated attributes introduces some limitations on the level of adaptation that is feasible.

The compression ratio for wavelet encoded views is influenced by the correlation among the attributes. Figure 6 displays the relative error for 4 attributes combined into a view and sorted along attribute 1 (start time of an observation). Attribute 1 and 2 (end time of an observation) are highly correlated, whereas attributes 3 (minimum energy) and 4 (maximum energy) are not correlated with the time based attributes, so that they represent a random signal. As a result, a large number of wavelet coefficients is needed to encode attributes 3 and 4, and (compared to attributes 1 and 2) also a higher percentage is needed to decode to a reasonable quality level. In figure 6 the error remains very small (0.01) for all 4 attributes up until around 20% of the coefficients. While the error for the first 2 attributes continues at very low levels beyond that threshold, the error for the attributes 3 and 4 increases dramatically. Thus, to achieve plots of reasonable quality over this data-set, more than 20% of the encoded view segment file should be used. A 1D plot of the first 2 attributes, on the other hand, would yield very good approximation quality using only 1% of the coefficients.

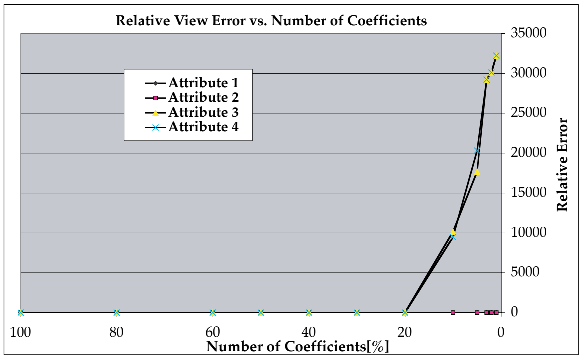

Fig. 6: Relative view error vs. view quality.

Figure 7 displays the storage requirements for views composed of attributes correlated and not correlated with the sorting attribute. Shown is the size of a single view segment versus increasing number of tuples with 2, 4 and 6 attributes. For 2 correlated attributes (1 and 2), a data-set encompassing $8.4 \cdot 10^6$ tuples requires 6 MBs of storage. A view with an additional two, non-correlated attributes of equal type increases storage requirements by more than a factor of four to 27 MBs. Adding more correlated or un-correlated attributes (of identical or equal type) will increases segment size by 6 or 27 MBs respectively.

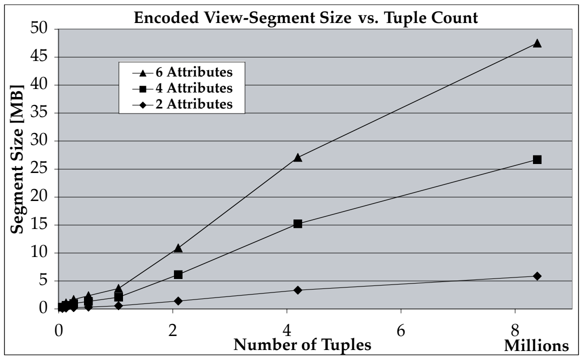

Fig. 7: Encoded view segment size vs. tuple count.

Even accounting for this factor, wavelet encoding yields a high compression ratio. Table 3 displays the amount of disk space required to hold increasing number of tuples for various storage methods. All methods allow a lossless reproduction of the tuples and show a more or less linear increase in storage size. The sizes do not scale evenly, as the tuple distributions for different sizes were statistically similar, but not identical. With 27 MB for $8.4 \cdot 10^6$ tuples, wavelet encoded view segments require less than 20% of the storage used by an Oracle RDBMS (just the table space, without indexes). The byte file generated the Oracle Export utility yields the highest compression ratio (16 MB). An ASCII view file will require 174 MB, which is still less than half of the DBMS size. The simple gnuzip compression of these ASCII segments reduces the size for $8.4 \cdot 10^6$ tuples to less than 40 MBs.

| Tuples[$10^3$] | dbs | export | view | view.gz | view.wav |
|---|---|---|---|---|---|
| 65 | 6 | 0.13 | 1.55 | 0.47 | 0.30 |
| 13 | 11 | 0.26 | 3.10 | 0.89 | 0.60 |
| 262 | 22 | 0.51 | 5.58 | 1.37 | 0.95 |
| 524 | 45 | 1.00 | 9.74 | 1.89 | 1.33 |
| 1048 | 90 | 2.00 | 16.11 | 2.88 | 2.09 |
| 2097 | 179 | 4.00 | 41.61 | 8.77 | 6.13 |
| 4194 | 359 | 8.00 | 91.77 | 21.24 | 15.20 |
| 8389 | 718 | 16.00 | 174.63 | 37.05 | 26.68 |

Tab. 3: View Size in MB for various storage methods.

## 4.6 Network Optimization: Quality

Figure 6 demonstrated the high quality of approximated views at small percentages of coefficients used when the attributes are sorted and correlated. Figure 9 displays the absolute error of density plots based on such approximated views. Shown is the absolute error of a 128*128 bin density plot versus decreasing quality for a data-set of $524 \cdot 10^3$ tuples. For this data-set first the error for the 4-attribute plot and then the one for the 2 attribute plot increases at 50% from near zero up to 630/945 at 5% of the coefficients.

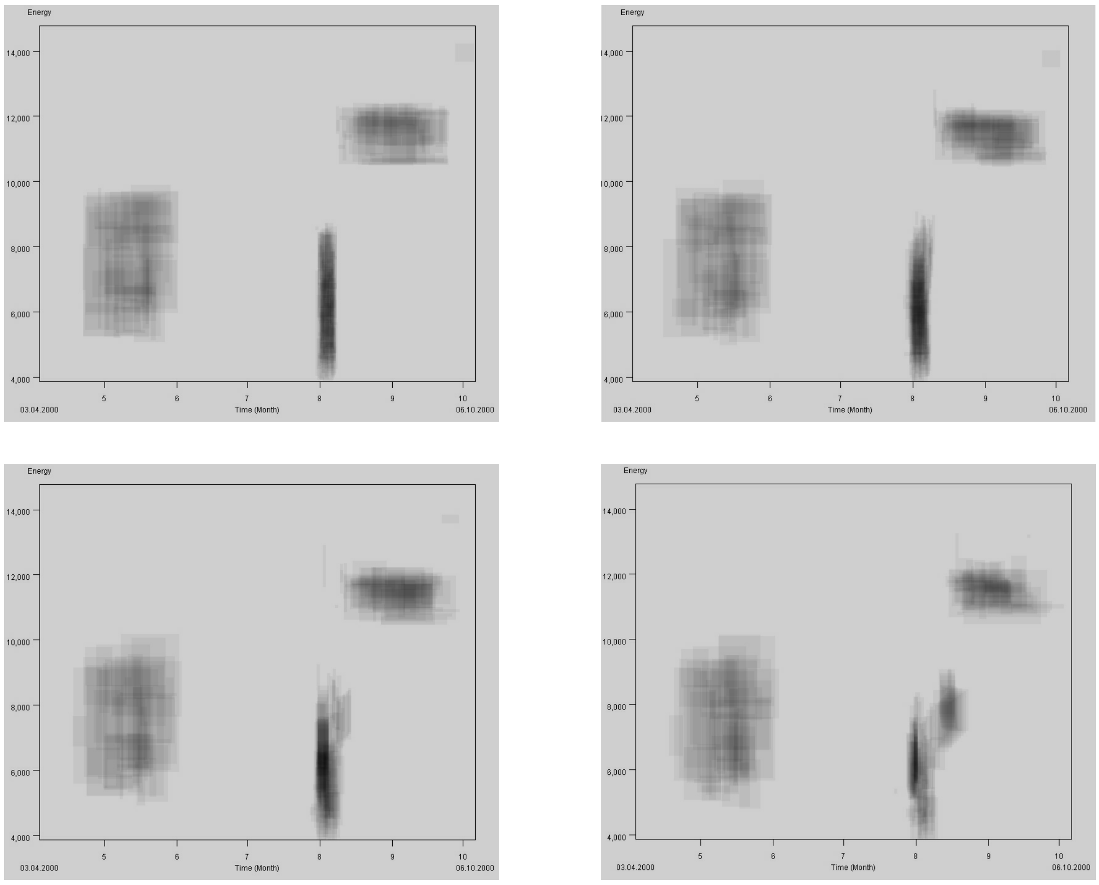

Fig. 8: 2D Extent plot (4 attributes) based on 100% (top left), 60% (top right), 30% (bottom left) and 10% (bottom right) of the coefficients.

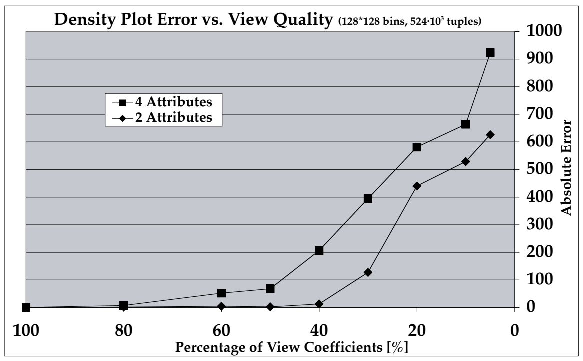

Fig. 9: Abs. density plot error vs. view quality.

What is the impact of such an absolute error of 700 on the visual accuracy of the plot? In the case of density plots the difference between the full-quality plot (Figure 1) and a plot based on 10% (Figure 10) is quite small and the 3 hot-spots are still easily discernible. For extend plots (2 attributes, 512 objects) the result is similarly good, with usable plots even at 10% of the coefficients. Clearly visible is the reduced accuracy of the time attribute, as the view was sorted along energy. This shows, that quality adaption can be applied to wavelet encoded views holding any numeric data, even random values.

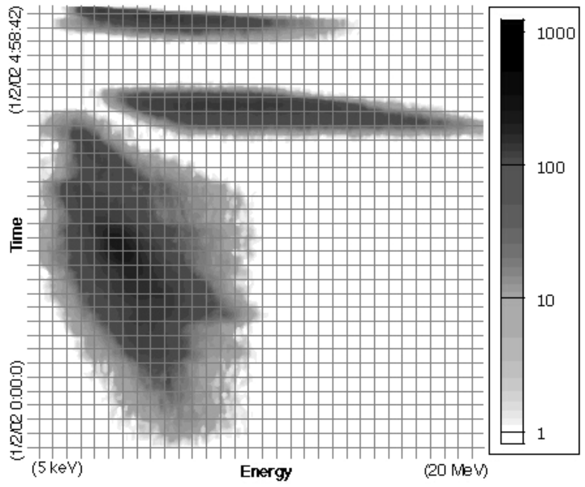

Fig. 10: Density plot (128*128 bins, $524 \cdot 10^3$ tuples, 10% view quality).

Yet, on a fast Ethernet connection quality reduction is not a big advantage, mostly because download time is small compared to decoding time. Figure 11 displays the times spent querying, downloading, decoding and then calculating the density plot with decreasing view quality for $4.2 \cdot 10^6$ tuples. Although the overall time is reduced by half, if view segments with 10% coefficients

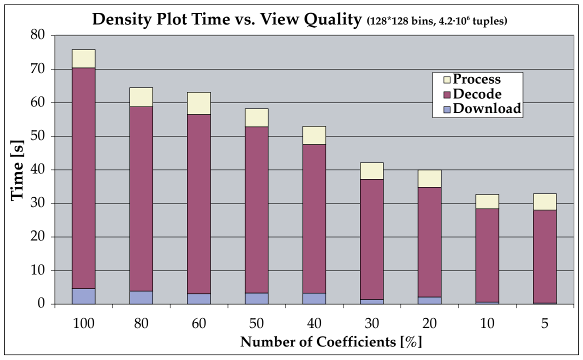

Fig. 11: Overall plot time vs. view quality.

are utilized, the dominant factor in all cases is the decoding time. It is this decoding time that needs to be reduced to achieve interactive exploration.

## 4.7 Client-Side Optimization: Resolution

The decoding time can be reduced through resolution adaptation. Figure 12 displays the absolute error of density plots versus decreasing decoding output size (resolution). Measurements were done for 8 data-sets ranging from $65{\cdot}10^3$ to $8.4{\cdot}10^6$ tuples. As the number of coefficients (quality) used was 10%, the error is greater than zero for the original resolution (1/1). Logically, the absolute error increases with the size. At lower resolutions the absolute error increases from original to 1/1024th of the original resolution by a factor of 2 to 3.

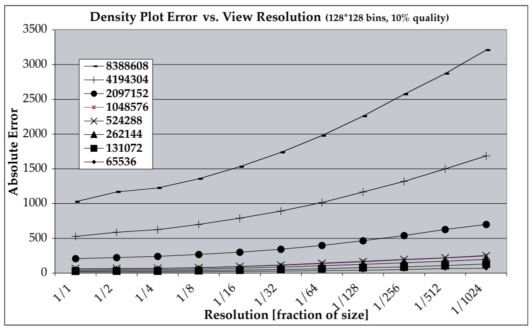

Fig. 12: Abs. plot error vs. view resolution.

Even at those resolutions the error is quite low compared to low quality approximations, which showed a much steeper decrease in accuracy. This is due to the fact that the number of data points generated after decoding still greatly exceeds the number of plot bins. Thus, a plot based on a data-set with $65{\cdot}10^3$ tuples in size still over-samples at a resolution of 1/1024th each bin with 65 values. Figure 13 shows the same density plot seen at 10% quality in figure 10, but this time at 1/16th of the resolution. The three hot-spots are still clearly visibly, but the value ranges are more condensed. At 10% quality the un-correlated energy attributes (min-/max-energy) are more distorted than the much better approximated time attributes (start-/ end-time).

Figure 14 displays the overall visualization speedup for a 128*128 density plot for $10^6$ tuples with decreasing resolution. As wavelet decoding speed is roughly linear with the number of view cells [29], especially higher dimensional views profit from resolution adaptation. At lower resolutions, network speed becomes proportionally more relevant. Extent plots show a similar behavior with longer visualization times.

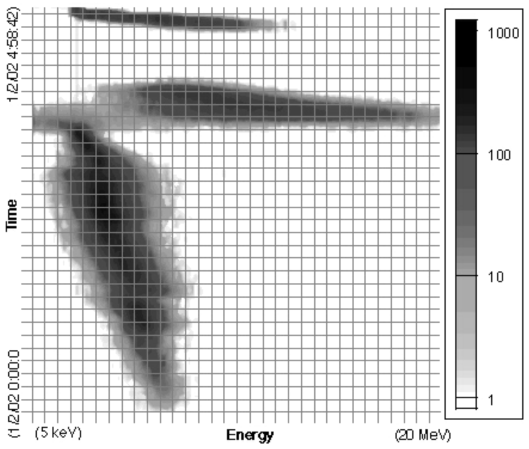

Fig. 13: Density plot (128*128 bins, $524{\cdot}10^3$ tuples, 10% view quality, 1/16th the resolution).

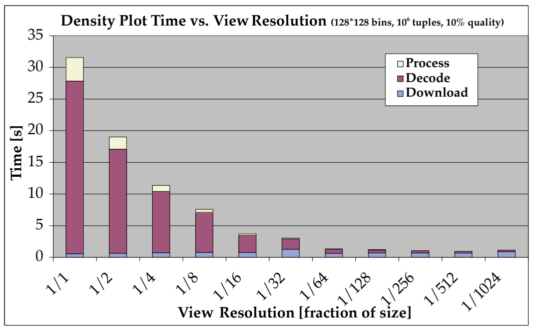

Fig. 14: Overall plot time vs. view resolution.

## 4.8 Interactive Database Exploration

Besides simple query processing, our method reduces the DBMS to a mere file server. The users adapt quality and resolution to their needs and the resources at hand. Thus server load is low, and even with large data-sets the clients determine the performance and accuracy of their visualizations. Table 4 and Figure 15 display the overall time to visualize data-sets of increasing size. For each data-set size, we have selected the quality- and resolution-setting so that the overall visualization for density plots remains below 5 and the time for extent plots below 10 seconds. These are the limits we have set for interactive exploration within HEDC. Given our Ethernet connection, users tend to keep quality high, but adjust resolution more strongly. Modem users might choose the opposite strategy. Beyond 20% quality and 1/2 the resolution, the relative error is low and the choice therefore not important.

| | 64 | 256 | 512 | 1MT | 2MT | 4MT | 8MT |
|---|---|---|---|---|---|---|---|
| Q. [%] | 100 | 100 | 100 | 50 | 20 | 10 | 10 |
| Res. | 1/1 | 1/2 | 1/2 | 1/2 | 1/2 | 1/16 | 1/32 |
| Den. [s] | 1.6 | 4.6 | 4.2 | 4.9 | 5.0 | 3.7 | 4.5 |
| Ext. [s] | 2.2 | 5.1 | 5.19 | 6.5 | 8.7 | 4.58 | 5.3 |

Tab. 4: Examples of interactive adaptation.

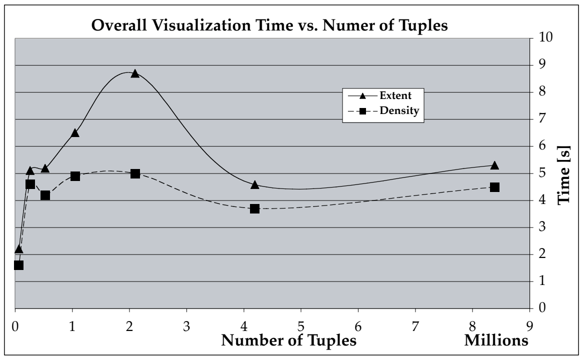

Fig. 15: Overall times with interactive quality and resolution adaptation vs. number of tuples.

## 5 Discussion

The main contribution of the ideas presented is the fact that interactive visualizations can be produced in a dynamic manner. To do so requires several optimizations. The first one is not to base visualizations on direct queries over the data. As the experiments show, this approach simply does not scale and explains the limitations of many database visualization tools. Instead, materialized views are needed. Again, our experiments show that these views need to be kept outside the database to get the necessary performance. However, the views we use are different from what has been proposed before. Our views are simple selections over the original data, we neither summarize this information nor pre-process it in any manner (histograms, for instance). The reason to perform such pre-processing steps is to speed up response time. In our case, we have the necessary response time while keeping complete flexibility about ranges, bin sizes, and aggregation parameters. This is a significant advantage over existing proposals that fix these parameters at the time the view is generated. While this is acceptable and even advantageous in other applications, it can be quite limiting in scientific databases where there is no way to determine an aggregation unit that will work for all cases. This is particularly important in terms of quality and resolution adaptations, which are fully controlled by the user and not by the system (e.g., through pre-computed views that already determine the quality or the resolution).

With this, the solution we propose not only allows interactive exploration but it also provides the basis for much needed improvements in this type of repositories. For instance, *Quality of Service* (QoS) guarantees can be readily implemented. The user can choose an error level, a response time, or a number of plots to produce (the overall load) and the client can automatically adapt resolution and quality accordingly. Similarly, the server may limit the quality when it is overloaded or the client may automatically juggle the two parameters to obtain the best possible plot given the time it takes to download data from the server.

Finally, interactivity also depends on the server being able to provide the data fast enough. One of the advantages of our approach is that the quality adaptation also benefits the server side. In preliminary stress tests over HEDC, we measured client visualization time and server load for up to 32 simultaneous clients doing continuous visualizations over large sets. Given the low load in the server caused by these clients, we are confident the system can support up to 100 concurrent clients all using acceptable levels of quality adaptation.

## 6 Implementation: StreamCorder

The ideas presented in the paper have been implemented in HEDC and are already in use. The implementation has been done as part of a specialized Java client: the *StreamCorder* (see screen shot in Figure 2), which is available for download from the HEDC page[4]. A local DBMS mirrors exactly the schema of the central HEDC repository including global tuple identifiers. During visualization, the StreamCorder coordinates the asynchronous download, caching, decoding and processing of the data. The local database transparently caches query results and manages downloaded view segments. The StreamCorder also contains specialized processing software developed for HESSI data and all the visualization tools discussed in the paper. The idea is for the StreamCorder to support both connected and disconnected operations. Users can download data, process it, and upload the results into HEDC (if they have the proper access rights).

For visualization, users may request a density or extent plot and then select relevant parameter ranges, zoom in, scroll along some parameter range to produce new plots. Users can also switch from the plots to a simple list view to browse the attributes of all tuples that are inside the current attribute selection. By selecting one of these tuples, all the available information (e.g., images) is automatically produced (either from the local database or downloaded from the server, in which case a copy is made in the local cache).

## 7 Conclusions

As the size of databases containing continuous observations of natural phenomena increases, finding relevant portions of the data becomes more difficult and time

[4]http://www.hedc.ethz.ch/release/

consuming. Conventional databases typically do not provide support for exploring large tables except standard query interfaces. Visualization tools that provide graphical representations of the contents of databases are a step in the right direction. Unfortunately, the amount of data in today's systems is much larger than what existing visualization tools can handle. In this paper we have shown how to increase the performance of visual database exploration tools to an interactive level. The techniques used to accomplish this (externally materialized views, partition, wavelet encoding, quality and resolution adaptation) are generic and, therefore, can be used in a wide range of applications. With this, the work presented in the paper is a significant contribution towards a more efficient use of large data repositories.

## References

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th Int. Conference on Principles of Database Systems, Seattle, USA*, pp. 254–263, Jun 1998.

[2] P. Bendjoya, J. Petit, and F. Spahn. Wavelet analysis of the voyager data on planetary rings. *ICARUS*, 105:385–299, 1993.

[3] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Proc. of 26th Int. Conference on VLDB, Cairo, Egypt*, pp. 111–122, Sep 2000.

[4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Int. Conference on Database Engineering (IDBT)*, pp. 190–200, 1995.

[5] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th Int. Conference on Principles of Database Systems, Philadelphia, USA*, pp. 155–166, May 1999.

[6] D. F. Swayne, D. Cook and A. Buja. XGobi: Interactive dynamic data visualization in the X Window System. *Journal of Computational and Graphical Statistics*, 7(1):113–130, Mar. 1998.

[7] M. Derthick, J. Kolojejchick, and S. F. Roth. An interactive visualization environment for data exploration. In *Proc. of the 3rd Int. Conference on Knowledge Discovery and Data Mining (KDD)*, page 2, 1997.

[8] L. Freitag and R. Loy. Comparison of remote visualization strategies for interactive exploration of large data sets. In *Proc. of the 15th Int. Parallel & Distributed Processing Symposium (IPDPS, Los Alamitos, USA)*, pp. 181–183, Apr. 23–27 2001.

[9] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. of the 17th Int. Conference on Principles of Database Systems, Seattle, USA*, pp. 331–342, June 1998.

[10] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proc. of the 16th Int. Conf. on Principles of Database Systems, Tucson, USA*, pp. 171–182, May 1997.

[11] W. Hoschek, J. J., Martinez, A. S. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project. *ACM Workshop on Grid Computing (GRID-00), 17-20 Dec., Bangalore, India*, 2000.

[12] Y. E. Ioannidis. Universality of serial histograms. In *Proc. of the 19th Int. Conference on VLDB, Dublin, Ireland*, pp. 256–267, Aug. 24–27 1993.

[13] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *Proc. of the 25th Int. Conference on VLDB, Edinburgh, UK*, pp. 174–185, Sept. 1999.

[14] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. of the 24th Int. Conference on VLDB, New York, USA*, pp. 275–286, Aug. 1998.

[15] D. A. Keim. Databases and visualization. In *Tutorial Int. Conference on Management of Data, Montreal, Canada*, 1996.

[16] D. A. Keim and H.-P. Kriegel. VisDB: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49, Sept. 1994.

[17] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views (extended abstract). In *Proc. of the 14th Int. Conference on Principles of Database Systems, San Jose, USA*, pp. 95–104, May 1995.

[18] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. DEVise: integrated querying and visual exploration of large datasets. In *Proc. of the 16th Int. Conference on Principles of Database Systems, Tucson, USA*, pp. 301–312, May 1997.

[19] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proc. of 26th Int. Conference on VLDB, Cairo, Egypt*, pp. 101–110, Sept. 2000.

[20] M. Riedewald, D. Agrawal, and A. E. Abbadi. Flexible data cubes for online aggregation. In *Proc. of the 8th Int. Conference on Database Theory (ICDT)*, pp. 159–173, 2001.

[21] J. Shamnugasundaram, U. Fayyad, and P. Bradley. Compressed data cubes for OLAP aggregate query approximation on continuous dimensions. In *Proc. of the 5th Int. Conference on Knowledge Discovery and Data Mining*, pp. 223–232, Aug. 15–18 1999.

[22] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A wavelet based clustering approach for spatial data in vldb. *VLDB Journal*, 8(3–4):289–304, Feb. 2000. Electronic edition.

[23] http://www.spotfire.com.

[24] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, Jan. 2002.

[25] E. Stolte and G. Alonso. Optimizing scientific databases for Client-Side proccessing. In *Proc. of the 8th Int. Conference on Extending Database Technology (EDBT), Prague, Czech Republic*, pp. 390–408, Mar 2002.

[26] A. S. Szalay, P. Z. Kunszt, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner. Designing and mining multi-terabyte astronomy archives: the Sloan Digital Sky Survey. In *Proc. of the 19th Int. Conference on Management of Data, Dallas, USA*, pp. 451–462, May 16–18 2000.

[27] http://datasplash.cs.berkeley.edu/.

[28] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proc. of the 18th Int. Conference on Management of Data, Philadelphia, USA*, pp. 193–204, June 1–3 1999.

[29] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proc. of the 7th Int. Conferences on Information and Knowledge Management (CIKM), Washington, USA*, pp. 96–104, Nov. 3–7 1998.

[30] J. Z. Wang, G. Wiederhold, O. Firschein, and S. X. Wei. Content-based image indexing and searching using daubechies wavelets. *Int. Jounal on Digital Libraries*, 1(4):311–328, 1998.