Models and Techniques for Approximate Queries in OLAP

Alfredo Cuzzocrea
University of Calabria, Italy

INTRODUCTION

Since the size of the underlying data warehouse server (DWS) is usually very large, response time needed for computing queries is the main issue in decision support systems (DSS). Business analysis is the main application field in the context of DSS, as well as OLAP queries being the most useful ones; in fact, these queries allow us to support different kinds of analysis based on a multi-resolution and a multi-dimensional view of the data. By performing OLAP queries, business analysts can efficiently extract *summarized knowledge*, by means of SQL aggregation operators, from very large repositories of data like those stored in massive DWSs. Then, the extracted knowledge is exploited to support decisions in strategic fields of the target business, thus efficiently taking advantage from the amenity of exploring and mining massive data via OLAP technologies. The negative aspect of such an approach is just represented by the size of the data, which is enormous, currently being tera-bytes and peta-bytes the typical orders of data magnitude for enterprise DWSs, and, as a consequence, data processing costs are explosive.

Despite the complexity and the resource-intensiveness of processing OLAP queries against massive DWSs, client-side systems performing OLAP and data mining, the most common application interfaces versus DWSs, are often characterized by small amount of memory, small computational capability, and customized tools with interactive, graphical user interface supporting *qualitative*, *trend analysis*. For instance, consider the context of retail systems. Here, managers and analysts are very often more interested in the product-sale plot in a fixed time window rather than to know the sale of a particular product in a particular day of the year. In others words, managers and analysts are more interested in the trend analysis rather than in the *punctual*, *quantitative analysis*, which is, indeed, more proper for OLTP systems. This consideration makes it more convent and efficient to compute approximate answers rather than exact answers. In fact, typical decision-support queries can be very resource intensive in terms of spatial and temporal computational needs. Obviously, the other issue that must be faced is the accuracy of the answers, as providing fast and totally wrong answers is deleterious. All considering, the key is proving fast, exploratory answers with some guarantees on their degree of approximation.

On the other hand, in the last few years, DSS have become very popular: for example, sales transaction databases, call detail repositories, customer services historical data, and so forth. As a consequence, providing fast, even if approximate, answers to aggregate queries has become a tight requirement to make DSS-based applications efficient, and, thus, has been addressed in research in the vest of the so-called approximate query answering (AQA) techniques. Furthermore, in such data warehousing environments, executing multi-steps, query-processing algorithms is particularly hard because the computational cost for accessing multi-dimensional data would be enormous. Therefore, the most important issues for enabling DSS-based applications are: (1) minimizing the time complexity of query processing algorithms by decreasing the number of the needed disk I/Os, and (2) ensuring the quality of the approximate answers with respect to the exact ones by providing some guarantees on the accuracy of the approximation. Nevertheless, proposals existent in literature devote little attention to the point (2), which is indeed critical for the investigated context.

BACKGROUND

Multi-dimensional models represent data as univocally associated to positions in a multi-dimensional space and support query and mining tasks according to a multi-resolution view of data. The growing attention towards such models was stirred up by recent advances of OLAP systems, which allow us to efficiently support data analysis for a wide range of modern application fields ranging from Business Intelligence to sensor network data management and QoS-based (quality of service) systems. Traditionally, OLAP technology was adopted for supporting just-in-time (summarized) knowledge extraction in decision-making processes of very large organizations, but its reliability and effectiveness has made OLAP engines a (very) popular component of a plethora of data-intensive systems. *Data cube* (Gray, Bosworth, Layman, & Pirahesh, 1996) is the fundamental data model of the OLAP technology, and effectively supports the above mentioned analysis goals. According to such a model, multi-dimensional data are organized in cubes that are characterized by a set of dimensions and a set of measures. The first set, which contains the analysis parameters or, more properly,

the *functional attributes*, allows us to univocally locate *data cells storing measures*, which are the values of interest for the target decision process, and on which various kinds of OLAP queries, which retrieve data by means of multi-dimensional aggregations, are executed.

Range queries (Ho, Agrawal, Megiddo, & Srikant, 1997) are an important class of OLAP queries that are very often executed on data cubes. They are defined as the application of a given *SQL aggregation operator* (such as SUM, COUNT, AVG, etc.) over a set of selected contiguous ranges in the domains of the dimensions. For instance, a *n*-dimensional range-SUM query over a *n*-dimensional data cube *A* can be generally formulated as follows:

$$SUM(l_0 : h_0, l_1 : h_1, \ldots, l_{n-1} : h_{n-1}) = \sum_{l_0 \le i_0 \le h_0} \sum_{l_1 \le i_1 \le h_1} \cdots \sum_{l_{n-1} \le i_{n-1} \le h_{n-1}} A[i_0][i_1]\ldots[i_{n-1}]$$

such that $\langle l_k : h_k \rangle$ is the range defined on the dimension d_k.

APPROXIMATE QUERY ANSWERING TECHNIQUES

There is a clear taxonomy of methods proposed for supporting approximate query answering: We distinguish between methods based on *pre-computation* and methods based on *online computation*. Methods belonging to the first class compute *synopses*, which are succinct representations of the original data, in an off-line mode, and we use them instead of the original data for answering queries, thus obtaining approximate answers. Methods belonging to the second class perform sampling at query time, so that answers can be continually improved (by progressively enlarging the data sample) under user control. Many techniques for compressing data cubes and evaluating range queries over their compressed representation have been proposed. Several compression models, which have been originally defined in different contexts, have been used to this end. In the following section, we focus our attention on both the most popular techniques such models are based on, that is, *histograms*, *wavelets*, and *sampling*.

Histograms

Ioannidis and Poosala (1999) introduced the use of histograms for providing approximate answers to set-valued queries. Histograms are data structures obtained by partitioning a given data domain into a number of mutually disjoint blocks, called *buckets*, and then storing for each bucket some aggregate information, such as the sum of their items. Histograms were originally proposed for query size estimation inside query optimizers, and to summarize multi-dimensional data distributions (Poosala & Ioannidis, 1997; Poosala, Ioannidis, Haas, & Shekita, 1996). In the latter case, the data distribution to be compressed consists of the frequencies of values of the attributes in a given relation, and queries are evaluated by performing linear interpolation on the stored aggregate values. Subsequently, histograms have been effectively used to estimate range queries in OLAP (Poosala & Ganti, 1999). Many techniques for building multi-dimensional histograms have been proposed in literature, each of them based on particular properties of the data distributions characterizing the input domains. Typically, statistical and error metrics-based properties are taken into account, and greedy algorithms are considered to mitigate the computational cost of processing very large data cubes. Among all the alternatives, we focus our attention on the following histograms, mainly because they describe approaches that we retain having a similar spirit to ours: Equi-Depth (Muralikrishna & DeWitt, 1998), MHist (Poosala & Ioannidis, 1997), and GenHist (Gunopulos, Kollios, Tsotras, & Domeniconi, 2000) histograms.

Given a *n*-dimensional data domain *D*, the Equi-Depth histogram $H_{E\text{-}D}(D)$ is built as follows: (1) fix an ordering of the *n* dimensions $d_0, d_1, \ldots, d_{n-1}$; (2) set $\alpha \approx n$-th root of desired number of buckets; (3) initialize $H_{E\text{-}D}(D)$ to the input data distribution of *D*; (4) for each *i* in {0, 1, …, *n*-1} split each bucket in $H_{E\text{-}D}(D)$ in α equi-depth partitions along d_i; and finally (5) return resulting buckets to $H_{E\text{-}D}(D)$. This technique presents some limitations: fixing α and a dimension ordering can result in poor partitions, and, consequently, there could be a limited level of *bucketization*.

The MHist histogram overcomes the Equi-Depth histogram performances, as stated by experimental results shown in (Poosala & Ioannidis, 1997). The MHist build procedure depends on the parameter *p* (specifically, such histograms are denoted by MHist-p histograms): contrarily to the previous technique, at each step, the bucket *b* in $H_{MH}(D)$ (i.e., the output histogram) containing the dimension d_i whose marginal is the most in need of partitioning is chosen, and it is split along d_i into *p* (e.g., $p = 2$) buckets.

GenHist histograms are a new class of multi-dimensional histograms that are different from the previous ones with respect to the build procedure. The key idea is the following: given an histogram *H* with h_b buckets on a *n*-dimensional input data domain *D*, the proposed technique builds the output histogram $H_{GH}(D)$ by finding n_b overlapping buckets on *H*, such that n_b is an input parameter. To this end, the technique individuates the number of distinct regions that is much larger than the original number of buckets h_b, thanks to a greedy algorithm that considers increasingly-coarser grids. At each step, the algorithm selects the set of cells *J* of highest density and moves enough randomly-selected points from *J* into a bucket to make *J* and its neighbor cells "close-to-uniform." Thus, the innovative contribution is to define a truly multi-dimensional split policy, based on the concept of *tuple density*.