June 05, 2017

System and Methods for Query Planning Using Entity Population Information

Jue Wang

Follow this and additional works at: http://www.tdcommons.org/dpubs_series

Recommended Citation
http://www.tdcommons.org/dpubs_series/541

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.
System and Methods for Query Planning Using Entity Population Information

Abstract: In a hierarchical data domain, labels can be assigned to data entities of the hierarchical data domain. A data processing system managing the hierarchical data domain can compute population information indicative of the number of data entities associated with each label. The data processing system can also generate label-based indexing including indices of data entities associated with each label. Upon receiving a query indicative of a respective query, the data processing system can make a smart decision whether to use label-based indexing or to perform a full scan of the hierarchical domain to process the received query. Such smart decision can include determining a number of data entities associated with the label based on the population information, and comparing the number of data entities to a threshold.

The present disclosure relates generally to improving query processing performance for hierarchical databases such as databases storing data associated with advertisement campaigns. In particular, the present disclosure relates to query planning based on entity population information associated with labels assigned to entities in the hierarchical database. Entities of the hierarchical database can be assigned labels allowing label-based searching/access of the database. Upon receiving a query indicative of a given label, a data processing system can select between different options of processing the received query based on the number of entities associated with the label indicated by the query.

Efficient and reliable management of distributed database systems, such as online advertising systems, banking systems, e-commerce systems or stock market systems, calls for fast, interactive and reliable data query/analysis over a huge corpse of data. In distributed database systems, respective data can change continuously at a fast pace. As such, allowing fast,
interactive and reliable data query/analysis poses serious technical and practical challenges. Reports provided by query/analysis tools are expected to reflect changes to the data occurring in real time throughout the distributed system. Also, processing a given query may involve fetching huge amount of data (such as tens or hundreds of Giga Bytes (GBs)). As such, providing fast and timely responses to such queries is a technically challenging goal given the complexity of the distributed database systems involved and the amount of data fetched or processed in responding to each query.

Many distributed database systems employ data partitioning to allowing for efficient fetching of data. Employed data partitioning schemes include vertical or horizontal partitioning, round-robin partitioning or hash partitioning. Such partitioning schemes do not improve query performance substantially in hierarchical databases with hierarchical parent-child relationship between entities. In particular, generating a report in such databases can involve, more than often, fetching through a huge amount of entity data along the hierarchy of entities. Some distributed database systems employ data sharding, for instance, according to data size criteria or data structure criteria to allow parallel processing of data shards. While parallel processing of data shards can improve query processing speed, such parallel processing still involves fetching all the data associated with a given database. Processing all the data for each query results in inefficient use of processing resources such as central processing units (CPUs) and input/output (I/O) interfaces (or ports). Also, processing all the data associated with the database is often too expensive to be performed interactively in real time, especially when the database stores a huge amount of data.

In a hierarchical database, respective data entities are arranged according to hierarchical parent-child relationship. In some implementations, a hierarchical database can include one or
more trees with each node of the tree(s) representing a data entity. For instance, in a hierarchical
database maintaining advertising data, each tree can represent data associated with a respective
advertiser including one or more advertisement campaign entities (or nodes). In some instances,
each tree can represent an advertisement campaign associated with a corresponding advertiser.
Each campaign entity can be associated with a plurality of advertisement group entities as child
tentities. Each advertisement group entity can be associated with a plurality of keywords, a
plurality of products, or a combination thereof. The keywords or the products can be represented
within the hierarchical database as child entities of corresponding advertisement group entities.
The data entities can be assigned labels. Data in the hierarchical database can then be searched
based on the labels assigned to the data entities. That is, a query indicative of a label can
represent a request for data associated with data entities to which that label is assigned.
Responsive to such query, the data processing system receiving the query can provide the
requested data by accessing the data entities associated with that label.

FIG. 1 is a diagram illustrating a hierarchical database employing label inheritance. A
label is a term or expression assigned to one or more data entities. A label can be indicative of a
characteristic, feature, name, classification or group that is common to all the data entities to
which the label is assigned. The diagram in FIG. 1 shows direct labels (continuous-line
rectangles) and inherited labels (dashed-line rectangles). A direct label is a label that is directly
assigned by the processing system to a corresponding data entity, whereas an inherited label is a
label inherited by a child or descendent data entity from a corresponding parent or ancestor data
tility. For example, the label “Brand” is directly assigned to data entity “Campaign X” and is
inherited by all descendent data entities associated with the “Campaign X” data entity. A label
associated with a respective data entity can be direct and inherited at the same time.
FIG. 1: A hierarchical database employing label inheritance

In processing a query indicative of a respective label, the data processing system may scan all the data entities in the hierarchical database to identify the data entities associated with the label indicated in the query, and generate a label report based on data associated with the identified entities. A label report can include a global summary view of statistics aggregated for data entities to which a given label is assigned (either directly or by inheritance) or a set of views depicting data entities associated with the given label. Processing techniques described herein are applicable to both types of label reports and other label reports (of other types) that may be generated in response to a query indicative of a respective label.

In some implementations, a data processing system managing a hierarchical data domain can be configured to allow assigning labels to data entities in the data domain. Labels can be assigned by a user having access to the data processing system. The data processing system can
automatically apply label inheritance to data entities descendent from ancestor data entities associated with direct labels. The data processing system can also be configured to compute hierarchical population information in association with one or more labels. The hierarchical population information for a given label can include information indicative of the number of data entities at each hierarchical level associated with the given label. For instance, considering the diagram in FIG. 1, the hierarchical population information for the label “Brand” can include indication(s) of the number of “Campaign” data entities, the number of “AdGroup” data entities, the number of “Product” (or “PT”) data entities, and the number of “Keyword” (or “KW”) data entities associated with that label. That is, the hierarchical population information for a label is indicative of the population of entities (e.g., in numbers) associated with that label.

The data processing system can be configured to generate label-based indexing indicative of data entities associated with each label. For instance, the label-based indexing can include one or more lookup tables (LUTs) for each label listing (and/or pointing to) all the data entities associated with that label. While accessing all data entities when processing each query indicative of a respective label results in inefficient use of processing resources and can lead to inability to provide fast, interactive and reliable responses to such queries, the data processing system can used the label-based indexing to access (and process) only data entities associated with the label in the query. That is, given a query indicative of a respective query, the data processing system can retrieve the label-based indexing data associated with that label. The data processing system can then access the data entities associated with that label directly (without full scan of all the data of the hierarchical domain) by using the information (e.g., pointers or indices) in the retrieved label-based indexing data. As such, the use of the generated label-based indexing can result in significant query processing performance improvement as it allows for
substantial reduction in the number of data entities accessed and/or processed when handling a query indicative of a respective label.

Employing label-based indexing may not provide gains in processing speed or in consumption of processing resources for each query processed. While the use of label-based indexing allows processing a query indicative of a respective label without fetching all data entries in the database, the use of the label-based indexing involves extra processing cost. In particular, using label-based indexing (or LUTs) involves disc seek operations (when retrieving the label-based indexing data associated with a given label) that can be associated with relatively large seek time(s). Specifically, as the number of LUTs to be retrieved in processing a query increases, the respective cumulative seek time increases resulting in longer processing time of the query. As the number of entities associated with a given label increases, the number of corresponding LUTs can also increase. In some instances, the number of entities associated with a given label indicated by a query can be large enough to make the use of label-based indexing less efficient than fetching all the entities in the hierarchical data domain when processing the query.

For example, consider an advertiser having 10 million keywords, 1000 of them are labeled with "2014Q4". Processing these 1000 keywords via index is going to be a lot faster than processing all these 10 million keywords. However, when 9 million out of the 10 million keywords are associated with the label "2014Q4," a full scan of the hierarchical data domain can be more efficient than retrieving the label-based indexing data (e.g., LUTs) associated with the same label and processing only the data entities with indices in the retrieved LUTs. In particular, the processing overhead associated with retrieving and reading the LUTs may be time consuming enough to offset any gain achieved by processing only the data entities associated
with the query. In fact, processing through index can often involve retrieving the LUTs, reading
the LUTs, and a redirection of data scanning based on indices in the LUTs. The redirection of
data scanning often results in breaking the physical locality of underlying data storage, which
makes it not as efficient as direct scanning when the amount of data scanned is relatively large.

The data processing system can be configured to make a smart decision, with regard to
whether to use label-based indexing or perform full scan, when receiving a query indicative of a
respective label. The data processing system can determine a number of entities associated with
the label based on the hierarchical population information for that label. The data processing
system can compare a number of data entities (e.g., “Keywords”) associated the label to a cut-off
threshold and decide based on such comparison whether to perform full scan of the hierarchical
data domain or employ the label-based indexing to access and/or process only data entities
indexed in the LUTs associated with the label. For instance, if the number of data entities
associated with the label is larger than the cut-off threshold is, a full scan is performed.
Otherwise, the LUTs associated with the label are retrieved and used to access only data entities
indexed therein.

The data processing system can also be configured to allocate the processing resources
(e.g., API workers) to process the query based on the number of data entities associated with the
label. For instance, allocation of the API workers can be based on the selected method for
scanning the data entities (i.e., with or without label-based indexing). FIG. 2 below shows two
flow diagrams illustrating, respectively, a regular multi-process API server work flow (on the
left) and an index-based label report generation work flow (on the right). When employing
label-based indexing, the data processing system can, upon receiving a query indicative of a
respective label, retrieve and read the LUT(s) associated with that label. Based on the LUT(s),
the data processing system can determine the label inheritance throughout the hierarchical domain and the data shards associated with the label. The data processing system can then determine the API workers to process the query based on the determined data shards.

The cut-off threshold employed by the data processing system can be obtained through experimental data. Such experimental data can be obtained by processing a plurality of queries indicative of respective labels with and without label-based indexing. For each query, the respective query processing performances in association with the respective number of data entities is recorded. Based on the recorded data, a threshold (indicative of a number of entities) can be determined such that when the number of data entities associated with a label is smaller than (or equal to) the threshold, the use of label-based indexing results in better query processing performance. The threshold can be an integer number or a ratio (e.g., a percentage) with respect to the total number data entities in the hierarchical data domain.
FIG.2 Flow diagrams for workflows using full-scan of and label-based indexing, respectively.

In some implementations, the hierarchical population information can also be used to improve data sharding. For example, for online advertising data domain, the fan-out factor at different hierarchical levels can vary substantially. The fan-out factor can be significantly smaller at levels above the “AdGroup” level than at the “AdGroup” level and levels below. The fan-out factor can also vary substantially between entities at the same level. For example, a large campaign could have hundreds of thousands of “AdGroups” and millions of “Keywords”, while a small campaign could just have a couple hundred of “AdGroups” with less than 20 thousands “Keywords”. So if entity data are sharded at entity boundary so that they can be processed in parallel, the population information can be used to determine whether the current sharding entity is a good point to stop at or is it necessary to recursively shard it. For example, a “Campaign” having 100 “AdGroups” and 10,000 “Keywords” can be processed as a single shard, while a “Campaign having 500 “AdGroups” and 1000,000 “Keywords” can be further sharded, e.g., into 500 shards, with one shard per “AdGroup”.

9