Reading Assignment 1: PREF

Locality-aware Partitioning in Parallel Database Systems

Summary:

One major challenge when horizontally partitioning large amounts of data is to reduce the network costs for a given workload and a database schema. The authors of this paper propose a novel partitioning scheme called predicate-based reference partition (PREF) that allows co-partitioning of sets of tables based on given join predicates, which can avoid expensive remote join operations. They also present two automated partitioning design algorithms to maximize data-locality. One algorithm only needs the schema and data, whereas the other algorithm additionally takes the workload as input. The authors show that their automated design algorithms can partition database of different complexity and thus help to effectively reduce the runtime of queries under a given workload when compared to existing partitioning approaches.

In the PREF scheme part, the paper gives the definition of PREF, which improves the efficiency of query processing by reducing runtime. It also creates two additional bitmap indexes for query processing: The first index dup indicates for each tuple $r \in R$ if it is the first occurrence, where duplicates that result from PREF partitioning can be easily eliminated during query processing. The second index hasS indicates for each tuple $r \in R$ if there exists a tuple $s \in S$ which satisfies p(r, s), which tells whether the tuple is linked in the table.

In the query processing part of the paper focuses on how queries need to be rewritten for correctness and how to optimize SQL queries optimizes query processing. The rewriting rules only support SPJA queries (Selection, Projection, Join and Aggregation), while nested SPJA queries are supported by rewriting each SPJA query block individually.

In the scheme and workload driven automated partitioning part, the paper talks about the main optimization goal, which is maximizing data-locality under the given partitioning schemes, among those materialization configurations with the same highest data-locality, the one with the minimum data-redundancy should be chosen.

For the scheme-driven part, the authors use an undirected labeled and weighted graph for the given schema firstly, and extract a subset of edges Eco from GS that can be used to co-partition all tables in GS such that data-locality is maximized. For a given connected GS, the desired set of edges Eco is the maximum spanning tree. Finally, they enumerate all possible partitioning configurations that can be applied for the MAST to find out which partitioning configuration introduces the minimum data-redundancy.

For the work load-driven part, firstly the algorithm creates a separate schema graph GS(Qi)

for each query where edges represent the join predicates in a query. Afterwards, the authors compute the MAST(Qi) for each GS(Qi) like the previous one. In the second step, the authors merge MASTs of individual queries in order to reduce the search space of the algorithm. Given the MASTs, the merge function creates the union of nodes and edges in the individual MASTs. finally, the authors use a cost-based approach to further merge MASTs.

In the evaluation part, the authors used the TPC-H benchmark as well as the TPC-DS benchmark to show the partitioning scheme has better performance on several aspects:
(1) the efficiency of parallel query processing over a PREF partitioned database
(2) the costs of bulk loading a PREF partitioned database
(3) the effectiveness of our two automatic partitioning design algorithms
(4) the accuracy of our redundancy estimates and the runtime needed by these algorithms under different sampling rates.

It illustrates how PREF reduces query runtimes and improves overall system efficiency.

At the end of the paper, the authors reference historical approaches in parallel database systems in the 1990s and pointed the importance of advanced partitioning schemes with the rapid development of computing resources.

The concluding section summarizes the benefits of different PREF partitioning schemes, while the schema-driven algorithms work reasonably well for small schemata, the workload-driven design algorithm is more efficient for complex schemata with a bigger number of tables. It also discusses potential developments about using algorithms on dynamic data, OLTP workloads and pruning technologies.

Here's the detailed summary of the paper, noticed that I mainly focused on the key parts of it, that are the PREF scheme and two different algorithms, which meant the later parts such as evaluations will be omitted.

ABSTRACT

One major challenge when horizontally partitioning large amounts of data is to reduce the network costs for a given workload and a database schema. The authors of this paper propose a novel partitioning scheme called predicate-based reference partition (PREF) that allows co-partitioning of sets of tables based on given join predicates, which can avoid expensive remote join operations. They also present two automated partitioning design algorithms to maximize data-locality. One algorithm only needs the schema and data, whereas the other algorithm additionally takes the workload as input. The authors show that their automated design algorithms can partition database of different complexity and thus help to effectively reduce the runtime of queries under a given workload when compared to existing partitioning approaches.

1. INTRODUCTION

Modern parallel database systems horizontally partition large amount of data in order to provide parallel data processing capabilities for analytical queries which face the challenge of network costs. A common technique to reduce it is reference partitioning, but it also has some shorts such as incoming foreign keys are not supported. The authors present a novel partitioning scheme called PREF to use a join predicate (called partitioning predicate). PREF may lead to full replication of a table in the worst case and they designed several algorithms to solve this problem.
Their automatic partitioning algorithms consists of two types which maximum the data-locality and minimize data redundancy. One algorithm only needs the schema and data, whereas the other algorithm additionally takes the workload as input.

2. PREDICATE-BASED REFERENCE PARTITIONING

2.1 Definition and Terminology

It gives the definition of PREF partitioning scheme. by using the PREF partitioning scheme, all tables in a given join path of a query can be co-partitioned as long as there is no cycle in the query graph. Finding the best partitioning scheme for all tables in a given schema and workload that maximizes data-locality under the PREF scheme, however, is a complex task.
For query processing, the authors create two additional bitmap indexes when PREF partitioning a table R by S: The first bitmap index dup indicates for each tuple $r \in R$ if it is the first occurrence (indicated by a 0 in the bitmap index) or if r is a duplicate (indicated by a 1 in the bitmap index). That way duplicates that result from PREF partitioning can be easily eliminated during query processing. The second index hasS indicates for each tuple $r \in R$ if there exists a tuple $s \in S$ which satisfies p(r, s). That way, anti-joins and semi-joins can be optimized.

2.2 Query Processing

In the following, the authors discuss how queries need to be rewritten for correctness, if PREF partitioned tables are included in a given SQL query Q. This includes adding operations for eliminating duplicates resulting from PREF partitioned tables and adding re-partitioning operations for correct parallel query execution. Furthermore, the authors also discuss rewrites for optimizing SQL queries (e.g., to optimize queries with anti-joins or outer joins). All these rewrite rules are applied to a compile plan P of query Q. Currently, rewriting rules only support SPJA queries (Selection, Projection, Join and Aggregation), while nested SPJA queries are supported by rewriting each SPJA query block individually

Rewrite Process: The rewrite process is a bottom-up process, which decides for each operator o ∈ P if a distinct operation or a re-partitioning operation (i.e., a shuffle operation) must be applied to its input(s) before executing the operator.
Note that our distinct operator is not the same as the SQL DISTINCT operator. Our distinct operator eliminates only those duplicates which are generated by our PREF scheme. Duplicates from PREF partitioning can be eliminated using a disjunctive filter predicate that uses the condition dup=0 for each dup attribute of a tuple in an (intermediate) result. A normal SQL DISTINCT operator, however, can still be executed using the attributes of a tuple to find duplicates with the same values. It also talks about the other three operations(projection, join, aggregation).

2.3 Bulk Loading

In order to insert a new tuple r into table R, we have to identify those partitions Pi(R) into which a copy of a tuple r must be inserted. Therefore, we need to identify those partitions Pi(S) of the referenced table S that contain a partitioning partner (i.e., a tuple s which satisfies the partitioning predicate p for the given tuple r).
For efficiently implementing the insert operation of new tuples without executing a join of R with S, we create a partition index on the referenced attribute of table S. The partition index is a hash-based index that maps unique attribute values to partition numbers i.
Finally, updates and deletes over a PREF partitioned table are applied to all partitions. However, we do not allow that updates modify those attributes used in a partitioning predicate of a PREF scheme (neither in the referenced nor in the referencing table).

3. SCHEMA-DRIVEN AUTOMATED PARTITIONING DESIGN

3.1 Problem Statement and Overview

In other words, while the main optimization goal is maximizing data-locality under the given partitioning schemes, among those materialization configurations with the same highest data-locality, the one with the minimum data-redundancy should be chosen.
The first step is to create an undirected labeled and weighted graph GS = (N, E, l(e ∈ E), w(e ∈ E)) for the given schema S (called schema graph). While a node n ∈ N represents a table, an edge e ∈ E represents a referential constraint in S. Moreover, the labeling function l(e ∈ E) defines the equi-join predicate for each edge (which is derived from the referential constraint) and the weighting function w(e ∈ E) defines the network costs if a remote join needs to be executed over that edge. The weight w(e ∈ E) of an edge is defined to be the

size of the smaller table connected to the edge e.
As a second step, we extract a subset of edges Eco from GS that can be used to co-partition all tables in GS such that data-locality is maximized. For a given connected GS, the desired set of edges Eco is the maximum spanning tree (or MAST for short). The reason of using the MAST is that by discarding edges with minimal weights from the GS, the network costs of potential remote joins (i.e., over edges not in the MAST) are minimized and thus data-locality as defined above is maximized.
Finally, in the last step we enumerate all possible partitioning configurations that can be applied for the MAST to find out which partitioning configuration introduces the minimum data-redundancy.

3.2 Maximizing Data-Locality

In order to maximize data-locality for a given schema graph GS that has only one connected component, we extract the maximum spanning tree MAST based on the given weights w(e ∈ E). The set of edges in the MAST represents the desired set Eco since adding one more edge to a MAST will result in a cycle which means that not all edges can be used for co-partitioning.

3.3 Minimizing Data-Redundancy

In Listing 1, we show the basic version of our algorithm to enumerate different partitioning configurations (PCs) for a given MAST. For simplicity (but without loss of generality), we assume that the schema graph GS has only one connected component with only one MAST. Otherwise, we can apply the enumeration algorithm for each MAST individually. The enumeration algorithm gets a MAST and a non-partitioned database D as input and returns the optimal partitioning configuration for all tables in D. The algorithm therefore analyzes as many partitioning configurations as we have nodes in the MAST (line 5-15). Therefore, we construct partitioning configurations (line 7-9) that follow the same pattern: one table is used as the seed table that is partitioned by one of the seed partitioning schemes (or SP for short) such as hash partitioning and all other tables are recursively PREF partitioned on the edges of the MAST. For each partitioning configuration new PiC, we finally estimate the size of the partitioned database when applying new PiC and compare it to the optimal partitioning configuration so far (line 12-14). While seed tables in our partitioning design algorithms never contain duplicate tuples, PREF partitioned tables do. In order to estimate the size of a database after partitioning, the expected redundancy in all tables which are PREF partitioned must be estimated. Redundancy is cumulative, meaning that if a referenced table in the PREF scheme contains duplicates, the referencing table will inherit those duplicates as well. For example, in Figure 2 the duplicate orders tuple with orderkey = 1 in the ORDERS table results in a duplicate customer tuple with custrkey = 1 in the referencing CUSTOMER table. Therefore, in order to estimate the size of a given table, all referenced tables up to the seed (redundancy-free) table must be considered.

3.4 Redundancy-free Tables

we adopt the enumeration algorithm described in Section 3.3 as follows:
(1) We also enumerate partitioning configurations which can use more than one seed table.
(2) We prune partitioning configurations early that add data-redundancy for tables where a user-given constraint disallows data-redundancy

4. WORKLOAD-DRIVEN AUTOMATED PARTITIONING DESIGN

In the first step our algorithm creates a separate schema graph GS(Qi) for each query Qi ∈ W where edges represent the join predicates in a query. Afterwards, we compute the MAST(Qi) for each GS(Qi).
In the second step, we merge MASTs of individual queries in order to reduce the search space of the algorithm. Given the MASTs, the merge function creates the union of nodes and edges in the individual MASTs.
In the last step (i.e. a second merge phase), we use a cost-based approach to further merge MASTs. In this step, we only merge MAST(Qj) into MAST(Qi) if the result is acyclic and if we do not sacrifice data-locality while reducing data-redundancy (i.e., if |D P (Qi+j)| < |D P (Qi)|+|D P (Qj)| holds).

Xiaoyan Xue
16.11.2023