Information Systems 63 (2017) 66–79

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Comparing columnar, row and array DBMSs to process recursive queries on graphs

Carlos Ordonez, Wellington Cabrera *, Achyuth Gurram

Department of Computer Science, University of Houston, Houston, TX 77204, United States

ARTICLE INFO

Available online 26 April 2016

Keywords:
Graph
SQL
Recursive query
Matrix
Reachability
Query optimization

ABSTRACT

Analyzing graphs is a fundamental problem in big data analytics, for which DBMS technology does not seem competitive. On the other hand, SQL recursive queries are a fundamental mechanism to analyze graphs in a DBMS, whose processing and optimization are significantly harder than traditional SPJ queries. Columnar DBMSs are a new faster class of database system, with significantly different storage and query processing mechanisms compared to row DBMSs, still the dominating technology. With that motivation in mind, we study the optimization of recursive queries on a columnar DBMS focusing on two fundamental and complementary graph problems: transitive closure and adjacency matrix multiplication. From a query processing perspective we consider the three fundamental relational operators: selection, projection and join (SPJ), where projection subsumes SQL group-by aggregation. We present comprehensive experiments comparing recursive query processing on columnar, row and array DBMSs to analyze large graphs with different shape and density. We study the relative impact of query optimizations and we compare raw speed of DBMSs to evaluate recursive queries on graphs. Results confirm classical query optimizations that keep working well in a columnar DBMS, but their relative impact is different. Most importantly, a columnar DBMS with tuned query optimization is uniformly faster than row and array systems to analyze large graphs, regardless of their shape, density and connectivity. On the other hand, there is no clear winner between the row and array DBMSs.

1. Introduction

Recursion is fundamental in computer science: many algorithms are naturally recursive, such as graph algorithms. Recursion has been incorporated into SQL via recursive queries [15,12,19]. Unfortunately, recursion is not available in all DBMSs and its implementation varies widely despite an ANSI SQL standard. In fact, most row DBMSs offer recursive queries (e.g. Postgres, Oracle, Teradata, IBM DB2, MS SQL Server), but they are not currently available in most columnar DBMSs (e.g. MonetDB, Vertica, C-Store, with the exception of SAP Hana [4]). This lack of querying capability is no coincidence as recursive queries represent one of the most challenging class of queries. A current trend in analytic database systems and data warehousing are so-called column stores [27] (also called column-oriented databases or columnar database systems), which have been shown to provide an order of magnitude performance improvement in evaluating analytical queries on large tables, mixing joins and aggregations. Since we are concerned about systems used in practice we focus on fully functional column-based database systems (e.g. supporting SQL, basic ACID properties, parallel evaluation, basic fault tolerance), which we simply call “columnar DBMSs” to contrast them with “old” row-oriented DBMSs. Within big data analytics, graph problems

* Corresponding author.
E-mail address: wcabrera@cs.uh.edu (W. Cabrera).

http://dx.doi.org/10.1016/j.is.2016.04.006
0306-4379/

are particularly difficult given the size of data sets, the complex structure of the graph (density, shape) and the mathematical nature of computations (i.e. graph algorithms). With that motivation in mind, we study the optimization of recursive queries on a columnar DBMS to analyze large graphs.

1.1. Motivation

We focus on the evaluation of queries with linear recursion, which solve a broad class of difficult problems including reachability, shortest paths, network flows and hierarchical aggregation. Some motivating examples include the following. Assume that there is a human resources database, with a table containing employee/manager information (a self-relationship in ER modeling terms). That is, there are two columns corresponding to the employee and the manager. Typical queries include: "list employees managed directly or indirectly by manager X", "how many people are managed by Y?", "list managers with at least 10 managed employees", "sum the salaries of all employees managed by Z". Assume that you have a database with flight information from multiple airlines. In this case the input table has a departing city and an arriving city, with associated cost and distance. Representative queries include: "give me all cities I can arrive to departing from this airport with no more than 2 connections", "find the cheapest flight between every pair of cities", "count the number of cities reachable with no more than 3 connections", "for every pair of cities count how many potential flights there are". As a more modern example related to the Internet, consider a social network like Facebook or Twitter, where typical queries include the following. "How many people are indirectly related to other persons with up to two common acquaintances?", "is there anyone in group A who knows someone in group B, and if so, how many connections are there between both groups?", "if one person spreads some news/gossip, how many people can be reached?". We should mention we do not tackle queries mixing negation and recursion which represent a harder class of queries. In summary, recursive queries open up the possibility to exploit a columnar DBMS to solve many fundamental graph problems.

Efficient processing of recursive queries is a fundamental problem in the theory of databases, where Datalog is the most prominent declarative language [1]. In contrast, research on recursive queries in SQL is rather scarce and existing research has only focused on row storage, the dominating storage for the past three decades. This is due to variations in recursive query implementation (despite the ANSI standard), the difficulty in understanding how recursion and query optimizations are combined and the common perception that a DBMS is hard to tune. However, graph problems are becoming more prevalent and more graph-structured data sets are now stored on SQL engines. To the best of our knowledge, optimization of recursive queries has not been revisited with columnar DBMSs. Having columnar DBMSs as the main motivation to perform graph analytics, these are some representative research issues: can relational DBMSs tackle large graphs or should they get out of the way and let other no-SQL systems do the job? are columnar DBMSs indeed faster? is it necessary to adapt classical query optimization techniques to columnar DBMSs? are there considerations to change or improve existing storage or indexing techniques? are there new considerations to accelerate recursive joins, the most demanding relational operator? can aggregation help reducing the size of intermediate results and perhaps query evaluation time? does the graph structure and connectivity impact query processing time, as it happens in Hadoop/noSQL systems? is recursion depth a big hurdle in dense graphs, as past research has shown? We attempt to provide clear answers to these questions.

1.2. Contributions

This is an overview of our research contributions. We start by reviewing a unified Seminaïve algorithm that works on both column and row DBMSs, based on automatically generated SQL queries. As a major contribution of our paper, we establish a connection between two graph problems and two recursive queries: transitive closure evaluated with a recursive join and adjacency matrix multiplication evaluated with a recursive query combining join and aggregation. We revisit query optimization of SPJ queries showing even though recursive query optimization is a well studied problem, there are indeed new research issues on columnar DBMSs. In order to study scalability and query optimizations with predictable results, we introduce a flexible graph generator that allows simulating graphs representing the Internet and social networks. Finally, we present a benchmark comparing a columnar DBMS, a row DBMS and an array DBMS, covering a wide spectrum of database technologies available today.

1.3. Article outline

Section 2 , a reference section, introduces graph and relational database definitions and gives an overview of storage mechanisms. Section 3 presents our main technical contributions: the standard Semi-naïve algorithm to evaluate recursive queries, SQL queries for each algorithmic step, query optimizations for relational operators, and their algebraic query transformations, highlighting differences in a columnar DBMS. We also include a time complexity analysis per relational operator. Section 4 compares query processing in a columnar DBMS with two prominent DBMSs: row-based and array. Experiments also evaluate the impact of each query optimization on graphs with different structure, density and connectivity. Section 5 discusses closely related work, focusing on SQL query optimization. Section 6 summarizes theoretical contributions, experimental findings, and directions for future research.

2. Definitions

This is a reference section which introduces standard graph definitions from a discrete mathematics perspective, the relational database model and basic SQL queries