Information Systems 41 (2014) 45–73

Contents lists available at ScienceDirect

Information Systems

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

Hybrid query execution engine for large attributed graphs

Sherif Sakr [a,*], Sameh Elnikety [b], Yuxiong He [b]

[a] National ICT Australia, UNSW, Sydney, Australia
[b] Microsoft Research, Redmond, WA, United States

ARTICLE INFO

Article history:
Received 19 December 2012
Received in revised form
26 October 2013
Accepted 29 October 2013
Recommended by: K.A. Ross
Available online 12 November 2013

Keywords:
Data models
Query
Access methods
Graphs
Graph queries
SPARQL

ABSTRACT

Graphs are widely used for modeling complicated data such as social networks, bibliographical networks and knowledge bases. The growing sizes of graph databases motivate the crucial need for developing powerful and scalable graph-based query engines. We propose a SPARQL-like language, G-SPARQL, for querying attributed graphs. The language enables the expression of different types of graph queries that are of large interest in the databases that are modeled as large graph such as pattern matching, reachability and shortest path queries. Each query can combine both structural predicates and value-based predicates (on the attributes of the graph nodes/edges). We describe an algebraic compilation mechanism for our proposed query language which is extended from the relational algebra and based on the basic construct of building SPARQL queries, the Triple Pattern. We describe an efficient hybrid Memory/Disk representation of large attributed graphs where only the topology of the graph is maintained in memory while the data of the graph are stored in a relational database. The execution engine of our proposed query language splits parts of the query plan to be pushed inside the relational database (using SQL) while the execution of other parts of the query plan is processed using memory-based algorithms, as necessary. Experimental results on real and synthetic datasets demonstrate the efficiency and the scalability of our approach and show that our approach outperforms native graph databases by several factors.

1. Introduction

Graphs are popular data structures which are used to model structural relationship between objects. Recently, graph query processing has attracted a lot of attention from the database research community due to the increasing popularity of graph databases in various application domains. In general, existing research on graph databases and graph query processing can be classified into two main categories. The *first* category represents graph databases which consist of a large number of small graphs (usually called *Transactional Graph Database*) such as bioinformatic applications [28], cheminformatics applications [31] and repositories of business process models [46]. In this category, there are two types of queries that are commonly studied in the literature:

(a) *Subgraph query* which aims to find all the graphs in the database such that a given query graph is a subgraph of them [58,59].
(b) *Supergraph query* that aims to find all the graphs in the database which are subgraphs of the given query graph [11,60].

The *second* category of graph databases is usually represented as one (or a very small number) of the large graphs such as social networks [8], bibliographical networks [56]

* Corresponding author.
E-mail addresses: ssakr@cse.unsw.edu.eu (S. Sakr), samehe@microsoft.com (S. Elnikety), yuxhe@microsoft.com (Y. He).

0306-4379/$ - see front matter © 2013 Published by Elsevier Ltd.
http://dx.doi.org/10.1016/j.is.2013.10.007

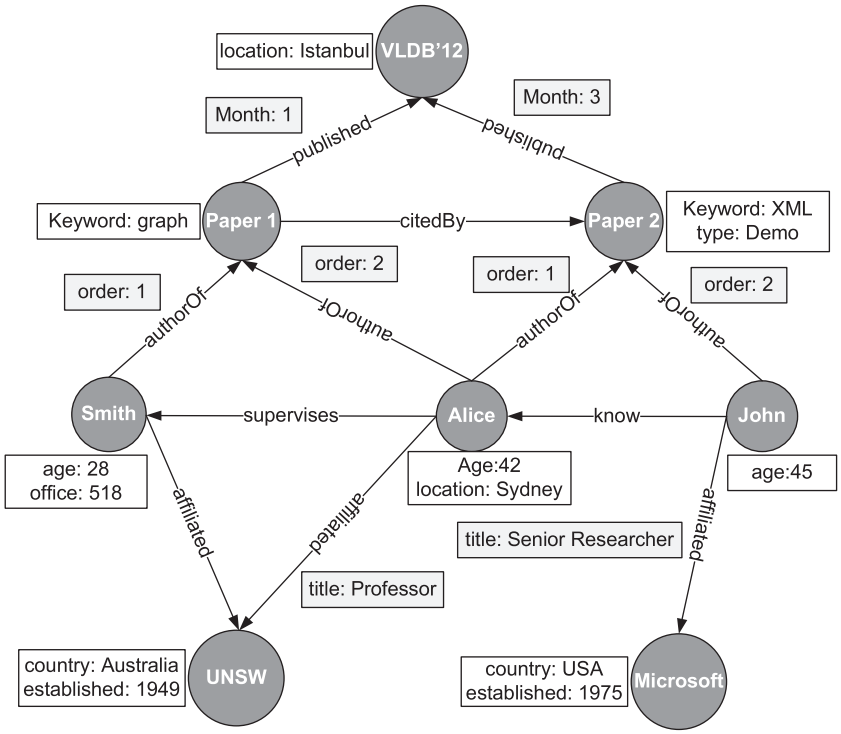

Fig. 1. An example attributed graph.

and knowledge bases [53]. In this category, there are three common types of queries:

(a) *Pattern match query* that tries to find the existence(s) of a pattern graph (e.g. path, star, subgraph) in the large graph [62,63].
(b) *Reachability query* that verifies if there exists a path between any two vertices in the large graph [13,30].
(c) *Shortest path query* which represents a variant version of the reachability query as it returns the shortest path distance (in terms of number of edges) between any two vertices in the large graph (if the two vertices are connected) [12,57].

In this paper, we focus on query processing in the *second* category of graph databases. In many real applications of this category, both the graph topological structures in addition to the properties of the vertices and edges are important. For example, in a social network, a vertex can be described with a property that represents the *age* of a person while the topological structure could represent different types of relationships (directed edges) with a group of people. Each of these relations can be described by a *start date* property. Each vertex is associated with a basic descriptive attribute that represents its *label* while each edge has a *label* that describes the type of relationship between the connected vertices. The problem studied in this paper is to query a graph associated with attributes (called as *attributed graph*) based on both structural and attribute conditions. Unfortunately, this problem did not catch much attention in the literature and there is no solid foundation for building query engines that can support a combination of different types of queries over large graphs. Formally, an *attributed graph* is denoted as $(V, E, L_v, L_e, F_v, F_e, \Lambda_v, \Lambda_e)$ where V is the set of vertices; $E \subseteq V \times V$ is the set of edges joining two distinct vertices; L_v is the set of vertex labels; L_e is the set of edge labels; F_V is a function $V \rightarrow L_v$ that assigns labels to vertices and F_e is a function $E \rightarrow L_e$ that assigns labels to edges; $\Lambda_v = \{a_1, a_2, \ldots, a_m\}$ is a set of m attributes that can be associated with any vertex in V. Each vertex $v \in V$ can be described with an attribute vector $[a_1(v), \ldots, a_m(v)]$, where $a_j(v)$ is the attribute value of vertex v on attribute a_j. $\Lambda_e = \{b_1, b_2, \ldots, b_n\}$ is a set of n attributes that can be associated with any edge in E. Each edge $e \in E$ can be described with an attribute vector $[b_1(e), \ldots, b_n(e)]$ where $b_k(e)$ is the attribute value of edge e on attribute b_k.

Fig. 1 shows a snippet of an example large graph where a vertex represents an entity instance (e.g. *author*, *paper*, *conferences*) and an edge represents a structural relationship (e.g. *co-author*, *affiliated*, *published*). In addition, there are attributes (e.g. *age*, *keyword*, *location*) that describe the different graph vertices while other attributes (e.g. *order*, *title*, *month*) describe the graph edges. In practice, a user may need to pose a query on the large graph that can involve more than one of the common graph query types. Examples of these queries are:

(1) Find the names of two authors, `X` and `Y`, where `X` and `Y` are connected by a path (sequence of edges) of any length (number of edges), the author `X` is affiliated at `UNSW`, the author `Y` is affiliated at `Microsoft` and