Storing and Querying Social Graph Data on a Variety of Distributed Systems

Christine F. Reilly
Skidmore College
Saratoga Springs, New York, USA
creilly@skidmore.edu

1. INTRODUCTION

We propose the exploration of a data model for social graph data that can be implemented on a variety of distributed storage systems. Online social networks provide many benefits to our society, but also have a number of drawbacks and negative impacts. Currently, a small number of companies hold private information about the operation of the majority of online social network services. The lack of publicly available information leads to difficulties in understanding and asking questions about the impacts that online social networks have on society.

The storage system that supports an online social network platform must be able to store graph data at a huge scale, have a fast query response, and support frequent updates. By providing a knowledge about how to deploy a data model for social graph data using publicly available distributed storage systems, database systems experts can use our knowledge to help society gain improved understanding and control of online social networks. The information that has been published by the companies that operate social network services [2, 3, 4], as well as systems for distributed data storage (Apache Hadoop) and graph storage (JanusGraph, Neo4j, and [6, 7]) provide a starting point. Prior research has demonstrated that the distributed graph storage systems that are publicly available do not have the capacity to support the fast query response and frequent updates that are required for large scale social network workloads [1, 5].

2. DATA MODEL

A data model for social graph data, along with a corresponding application programming interface (API), provides the ability for applications to interact with any storage system that utilizes this data model. The abstract data model and API that has been published by researchers from Facebook is simple and supports the operations that are typical in a online social network application [2, 4]. This model represents social network objects (users, places, photographs, events, etc.) as graph nodes, and connections between objects as graph edges. Two logical tables store the social graph data: one table for nodes and one table for edges.

9th Biennial Conference on Innovative Data Systems Research (CIDR '19) January 13-16, 2019 , Asilomar, California, USA.

3. CHALLENGES

The database systems community faces a number of challenges related to implementing this social graph data model using publicly available storage systems.

Benchmarks: In order to test and compare storage systems, benchmark workloads and datasets must be published. These benchmarks should mimic a variety of typical social network use patterns. Generated datasets need to represent the social graph structure size.

Distributed Database Systems: The abstract data model can be implemented as a database schema, and the API can be implemented as database queries. The data distribution algorithms must account for the social graph structure in order to support fast queries.

Distributed File Systems: The abstract data model can be mapped onto flat files using the slotted page approach, similar to that used by relational database systems. The API can be implemented as methods that read and write these flat files. By considering the social graph structure when determining what file the data about a node or edge is placed into, the write methods in the API can account for the social graph structure. Index structures are required in order to support fast queries.

4. REFERENCES

[1] I. Abdelaziz et al. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. *Proceedings of the VLDB Endowment*, 10(13), 2017.

[2] T.G. Armstrong et al. LinkBench: A database benchmark based on the Facebook social graph. In *SIGMOD*, 2013.

[3] D.F. Bacon et al. Spanner: Becoming a SQL system. In *SIGMOD*, 2017.

[4] N. Bronson et al. TAO: Facebook's distributed data store for the social graph. In *USENIX ATC*, 2013.

[5] A. Khandelwal et al. ZipG: A memory-efficient graph store for interactive queries. In *SIGMOD*, 2017.

[6] B. Shao et al. Trinity: A distributed graph engine on a memory cloud. In *SIGMOD*, 2013.

[7] K. Zhao and J.X. Yu. All-in-one: Graph processing in RDBMSs revisited. In *SIGMOD*, 2017.