Parallel Computing 58 (2016) 140–156

Contents lists available at ScienceDirect

Parallel Computing

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

An asynchronous traversal engine for graph-based rich metadata management

Dong Dai[a], Philip Carns[b], Robert B. Ross[b], John Jenkins[b], Nicholas Muirhead[a], Yong Chen[a],*

[a] *Computer Science Department, Texas Tech University, United States*
[b] *Mathematics and Computer Science Division, Argonne National Laboratory, United States*

A R T I C L E   I N F O

Article history:
Received 5 December 2015
Revised 14 May 2016
Accepted 16 June 2016
Available online 23 June 2016

Keywords:
Parallel file systems
Rich metadata management
Property graph
Graph traversal
Graph partitioning

A B S T R A C T

Rich metadata in high-performance computing (HPC) systems contains extended information about users, jobs, data files, and their relationships. Property graphs are a promising data model to represent heterogeneous rich metadata flexibly. Specifically, a property graph can use vertices to represent different entities and edges to record the relationships between vertices with unique annotations. The high-volume HPC use case, with millions of entities and relationships, naturally requires an out-of-core distributed property graph database, which must support live updates (to ingest production information in real time), low-latency point queries (for frequent metadata operations such as permission checking), and large-scale traversals (for provenance data mining).

Among these needs, large-scale property graph traversals are particularly challenging for distributed graph storage systems. Most existing graph systems implement a “level-synchronous” breadth-first search algorithm that relies on global synchronization in each traversal step. This performs well in many problem domains; but a rich metadata management system is characterized by imbalanced graphs, long traversal lengths, and concurrent workloads, each of which has the potential to introduce or exacerbate stragglers (i.e., abnormally slow steps or servers in a graph traversal) that lead to low overall throughput for synchronous traversal algorithms. Previous research indicated that the straggler problem can be mitigated by using *asynchronous* traversal algorithms, and many graph-processing frameworks have successfully demonstrated this approach. Such systems require the graph to be loaded into a separate batch-processing framework instead of being iteratively accessed, however.

In this work, we investigate a general asynchronous graph traversal engine that can operate atop a rich metadata graph in its native format. We outline a traversal-aware query language and key optimizations (*traversal-affiliate caching* and *execution merging*) necessary for efficient performance. We further explore the effect of different graph partitioning strategies on the traversal performance for both synchronous and asynchronous traversal engines. Our experiments show that the asynchronous graph traversal engine is more efficient than its synchronous counterpart in the case of HPC rich metadata processing, where more servers are involved and larger traversals are needed. Moreover, the asynchronous traversal engine is more adaptive to different graph partitioning strategies.

* Corresponding author.
E-mail address: yong.chen@ttu.edu (Y. Chen).

http://dx.doi.org/10.1016/j.parco.2016.06.002
0167-8191/© 2016 Elsevier B.V. All rights reserved.

1. Introduction

A high-performance computing (HPC) platform commonly generates huge amounts of metadata about different entities including jobs, users, files, and their relationships. Traditional metadata, which describes the predefined attributes of these entities (e.g., file size, name, and permissions), has been well recorded and used in current systems. Rich metadata, which describes the detailed information about entities and their relationships, extends traditional metadata to an in-depth level and can contain arbitrary user-defined attributes. A typical example of rich metadata is provenance or lineage, which maintains a complete history of a dataset, including the processes that generated it, the user who started the processes, and even the environment variables, parameters, and configuration files used during execution [1]. Property graphs, which are an extension of traditional graphs with property annotations on vertices and edges, are a promising data model for rich metadata management in HPC systems because of their ability to represent not only metadata attributes but also the relationships between them. Distributed property graph databases such as Neo4j [2], DEX [3], OrientDB [4], G-Store [5], and Titan [6] have been developed to assist in managing large property graphs.

We are developing a rich metadata management system based on the new concept of unifying metadata into one generic property graph [1]. In addition to storing the property graphs, a major requirement in the rich metadata use case is to effectively answer graph traversal queries from metadata management utilities, such as provenance queries, hierarchical data traversal, and user audit. Graph traversal usually serves as the basic building block for various algorithms and queries. In fact, it is so fundamental that traversal of simple graphs[1] has been used as a benchmark metric (Graph500) for measuring the performance of supercomputers [7,8]. Traversal for property graphs is likewise critical and needs efficient implementation.

Typically, the core execution engine of graph traversal is implemented by following the general structure of the parallel “level-synchronous” breadth-first search (BFS) algorithm, dating back three decades [9,10]. Given a graph *G*, level-synchronous BFS systematically explores *G* from a source vertex *s* level by level. The *level* is the distance or hops it travels. BFS implies that all the vertices at level k from vertex s should be “visited” before vertices at level $k+1$; hence, global synchronization is needed at the end of each traversal step. The “level-synchronous” breadth-first search structure has been adopted not only in graph databases but also in many distributed graph-processing frameworks, including Pregel [11], Giraph [12], and GraphX [13]. The Bulk Synchronous Parallel (BSP) model is popular in this context because of its simplicity and performance benefits under balanced workload.

However, such global synchronization could cause serious performance problems in our property graph-based metadata management case for several reasons. First, as an on-line database system, our system allows concurrent graph traversals for different management tasks. The interferences among traversals easily create stragglers [14,15], which can cause poor resource utilization and significant idling during global synchronization. Second, the imbalance of the graph partitions, along with the possible variations in attribute sizes among different vertices and edges, leads to highly uneven loads on different servers (an indication of stragglers) while traversing. The wide existence of small-world graphs in HPC metadata (e.g., degree of vertices follows the power-law distribution [1,16]) makes this problem even worse. Third, in HPC metadata property graphs, possible graph traversal steps could be much larger than the graph diameter, which traditionally limits the maximal traversal steps in simple graphs. For example, the six degrees of separation theory exists in social networks [17]. Specifically, in our use case, different attributes of the same vertex or edge can be used in different steps. Longer traversals introduce more synchronizations and lead to a higher chance of performance penalty caused by stragglers.

Previous work indicated that asynchronous approaches have the potential to minimize the effects of load imbalance across different cores in multicore machines [18]. GraphLab [19], PowerGraph [20], and other distributed frameworks [21,22] have investigated the use of asynchronous execution models, which could implement the traversal operations in general. However, these approaches are more suitable for the distributed, batch-oriented graph computation that runs on the entire graph, instead of interactive traversal and query of subgraphs, which are common in our HPC-rich metadata management system.

In this research, we explore the design and implementation of an asynchronous traversal engine. We propose optimizations, including *traversal-affiliate caching* and *execution merging*, to fully exploit the performance advantage of the asynchronous traversal engine. In addition, we explore the effect of different graph-partitioning strategies on graph traversal engines to show the advantage of the asynchronous engine. Also proposed is a general traversal language to describe diverse patterns of property graph-based rich metadata management. We show that the asynchronous engine can support this language with detailed progress report functionality comparable to that of a synchronous engine. The main contributions of this work are fourfold.

- Analysis and summary of the graph traversal patterns in property graph databases for HPC rich metadata management. Based on these patterns, we propose a graph traversal language to support them.
- Design and implementation of an asynchronous distributed traversal engine. Critical optimizations are also proposed for the asynchronous traversal engine: *traversal-affiliate caching* and *execution merging* to improve the performance.
- Analysis of the effects of vertex-cut vs. edge-cut graph partitioning on graph traversal.
- Evaluation and demonstration of the performance benefits compared with synchronous traversal engine on both synthetic graphs and real-world graphs, as well as under different graph-partitioning strategies.

[1] The simple graph indicates a graph defined as a set of nodes connected by weighted edges in this study.