Design and Implementation of a Cloud-based and Distributed Graph Engine for Large Datasets

Qing Cao* and John A. Stankovic†

*Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN, US
†Department of Computer Science
University of Virginia, Charlottesville, VA, US

***Abstract*—In this position paper, we consider the problem of query processing related to graph problems using a cloud computing platform for extremely large-scale complex networks. Existing work suffers from three major shortcomings: they cannot run in parallel over many servers in cloud-computing platforms, they are too slow in generating solutions for many simultaneous queries, and they require too much overhead to store the outputs from their algorithms. In this paper, we propose a cloud-based graph processing engine for finding approximate results with an elastic accuracy for extremely large datasets, where we aim to achieve a much higher throughput than existing methods. As a case study, we demonstrate how this engine can find near-shortest paths. Specifically, we develop algorithms based on the engine to take advantage of two ideas: pre-computing and storing algorithm-specific digests of graph topology in each node, and generating solutions for queries by exploiting node level parallelism that naturally emerges after nodes are assigned to multiple servers. We also list several challenges we have identified to develop this proposed engine. We believe that such a graph processing engine is useful for many types of tasks such as social networks, biological networks, transportation scheduling, and others.**

I. Introduction

Various graph models are commonly used to analyze real-world phenomenon, such as online social networks (e.g., LinkedIn or Facebook), biological networks, road networks, among others. Due to the ever increasing number of nodes, many seemingly straightforward operations have become challenging when extremely large graphs are involved. For example, consider the classical problem of finding the shortest paths between any two nodes. Existing solutions to the problem of finding the shortest path, such as the well-known Floyd-Warshall algorithm, takes $O(V^3)$ time to find all-pair shortest paths. However, such approaches suffer from two limitations in scalability: first, they cannot be easily modified to handle many simultaneous queries over extremely large graphs, which may comprise of millions or even billions of nodes. In such cases, finding solutions will be way too slow for most applications. Second, they are centralized, which means that they can not be easily implemented on distributed computer clusters, such as cloud computing platforms, in an efficient manner.

Motivated by these limitations, our main position in this paper is that we need a distributed, approximate, and cloud-based graph based engine for processing a large number of simultaneous queries over many graph features. For example, hosting a large-scale social network service usually requires many queries on the degrees of nodes and paths connecting nodes to be answered efficiently. In such cases, the engine can provide the results in a timely manner, therefore, greatly simplifying the task of building large-scale complex server-side applications that rely on the query results to provide services for users.

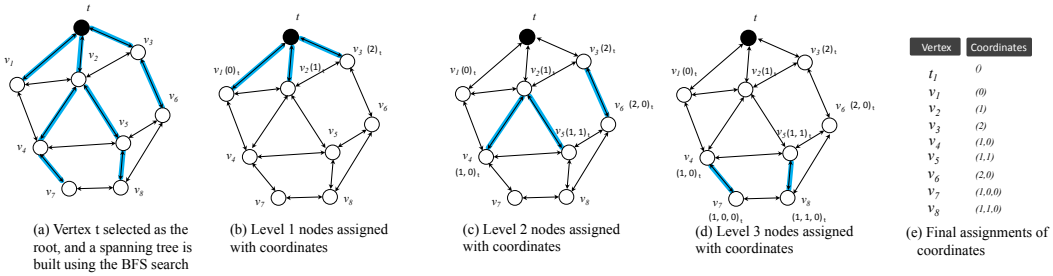
This proposed work and position is enabled by the recent progress on cloud computing platforms. In recent years, cloud computing platforms have rapidly been developed for providing elastic computing infrastructure based on users' demands. Through the use of broadband networks, virtualization, resource management, and data center technologies, even users with limited budgets can quickly deploy their scalable services for handling a large number of customers. The success of this rent-for-use model is bringing considerable changes to the research community as well, and various recent initiates, such as the NSFCloud infrastructure, has attracted great interest to deploy research prototypes that would not be possible without such infrastructure support. Similarly, this work can benefit greatly from this new generation of innovative infrastructure of cloud computing, where our engine can be deployed as a service on such clouds to provide elastic services to end users.

We hope with the support from cloud experimental facilities by NSF, we can develop the engine to run on these facilities, where it will assign nodes to servers based on factors such as their degrees, computational loads, and query volumes. This is used to partition the computational overhead evenly across cloud servers. Note that there are two features of this engine that make it novel. First, it is designed to provide approximate, rather than accurate, answers to queries, where we hope to generate *nearly optimal* solutions so that we can save greatly on the overhead and computation cost. To this end, applications that generate queries (on the client side) can provide their desired accuracy levels so that the engine will allocate proportional computational and communication resources to achieve differentiated levels of accuracies. Second, it is designed to execute on multiple servers in a distributed manner, so that it can fully exploit the underlying computation power and storage space.

II. Case Study of Finding Shortest Paths

A. Problem Formulation

As a case study, we consider the following query processing task based on this engine: how we can answer queries regard-



(a) Vertex t selected as the root, and a spanning tree is built using the BFS search

(b) Level 1 nodes assigned with coordinates

(c) Level 2 nodes assigned with coordinates

(d) Level 3 nodes assigned with coordinates

(e) Final assignments of coordinates

Fig. 1. This figure shows forming a tree structure and then assigning coordinates to nodes based on their relative orderings in the tree.

ing possible paths between any two nodes in extremely large graphs. The nature of the graphs can be flexible, ranging from simple graphs to complex networks such as those that follow the power law distributions of node degrees. As mentioned in the introduction, we expect the users will query the graph processing engine in the cloud infrastructure for node distances, while the engine will respond to the users in a timely and scalable manner.

B. Approach Overview

Our approach is inspired by the previous work on graph embeddings found in the literature for simple graphs such as transportation graphs, which in turn was based on the classical landmark based approaches. The overall idea of such frameworks is that they pre-compute and store a digest of graph topology in each node, so that any distance query can be answered by initiating a routing procedure from the the source node to the destination node. We observe that our approach for this specific case study highlights the two cornerstones for the graph processing engine: we aim to exploit *algorithm-specific digests* and *node-initiated sessions* to build various algorithm implementations. Note that the digests and node-initiated sessions are similar to templates: their particular semantics are defined by the algorithms. This is very similar to the way other cloud-based data processing algorithms, such as MapReduce, are developed: the developers will need to fill in the semantics of functions, while the data processing engine only provides templates.

C. Algorithm Design

In this section, we describe the details of *digests* and *node sessions* of our algorithm. We next describe these two stages separately.

The precomputation step involves choosing a few nodes as landmarks, and then computing for every node a "topology digest" based on these landmarks. These digests will be used in the approximate path finding step later. Briefly, as illustrated in Figure 1, this process starts by selecting a few nodes as roots, denoted as t_0, t_1, etc. These nodes will be used to construct trees. Usually, we select roots based on their degrees: those nodes that have the highest degrees will be chosen as the roots. Next,we perform a breadth-first-search operation from the root to every other nodes in the tree structure. Therefore, for each node in the tree, we can find its children nodes, based on which we can assign them with coordinates. Figure 1(b) to Figure 1(d) show the children node coordinates. The coordinates assigned to each node is shown in Figure 1(e), where the distances and topology stored at each node becomes their path digests.

In the second stage, we start new tasks initiated at each node for finding paths between a pair of nodes. For example, if a query is for finding the shortest path from v_4 to v_6 in Figure 1, we can start a session from the server storing v_4 independently from other queries. Therefore, such tasks are referred to as parallel node sessions that can be started at multiple servers simultaneously, and their algorithms are based on the path digests we collected earlier. The goal is to get the a small error, that is, the number of hops should be similar to the optimal path. We have developed different ways to implement this specific algorithm, and they can lead to different approximation ratios by using varying amount of computational resources.

We have carried out preliminary studies using a single workstation, where the results demonstrate that the approach is promising, as tested on datasets with Twitter, Facebook, and Google webpage graphs. We hope to run this engine on cloud platforms in the future when facilities are available.

III. Challenges and Conclusions

We have identified a few challenges that we will investigate in the future, such as the best ways to allocate nodes to cloud servers to achieve load-balancing, how to make the approximate solutions to be scalable to dataset sizes, and how to handle dynamic graphs with changes to their vertices and edges. Such research challenges present problems that researchers need to address in the future. Furthermore, a good implementation will also adapt to the underlying networking fabric of the cloud processing platform, so that it can take advantage of the shorter latencies between servers that are nearer to each other.

Our overall vision is that we can provide graph query processing as a generic cloud-based service, and propose to design a cloud-based engine for graph processing. We note that our approach naturally allows users to choose either to keep their graph datasets private, or make the datasets public. The former may be applied to the scenarios where the datasets contain sensitive data, while the latter can be graph datasets that are scientific by nature, such as biology graphs or geographic maps. If successful, we believe that a generic graph processing engine will have a wide range of applications and bring benefits for the research community.