



Department of Mathematics and Computer Science
Database Group

Efficient Graph Processing Using AWS S3

Master Thesis

Aditya Chandla

Supervisors:
dr. Nikolay Yakovets

Eindhoven, July 2024

Abstract

As the size of graphs being queried by graph databases increases, coupling storage and compute increases the cost and limits the flexibility of traditional graph database management systems (GDBMS). As part of this thesis, we explore the viability of an architecture where the compute and storage are independent. This independence is achieved using a distributed cloud storage service (AWS S3) which provides bottomless storage and theoretically unlimited throughput. Using this distributed cloud storage solution, we evaluate the latency of running two common graph traversal algorithms i.e breadth first search (BFS) and depth first search (DFS). We then compare this latency with some other systems which may be used to perform BFS and DFS on large graphs.

Preface

Please write all your preface text here. If you do so, don't forget to thank your supervisor, other committee members, your family, colleagues etc. etc.

Contents

Contents	vii
List of Figures	ix
List of Tables	xi
Listings	xiii
1 Introduction	1
1.1 Structure of the thesis	1
2 Preliminaries	3
2.1 Distributed storage services	3
2.2 Serverless Architectures	3
2.3 Caching and Prefetching in distributed systems	3
3 Experimental Evaluation	5
3.1 System Architecture	5
3.2 Query Evaluation	5
3.3 Baseline Implementation	5
3.4 Improving graph access service	5
3.4.1 Modifying CSR structure	5
3.4.2 Caching	5
3.4.3 Prefetching	5
3.5 Parallelizing graph algorithm service	5
3.5.1 Parallel BFS implementation	5
3.5.2 Parallel DFS implementation	5
3.6 Optimizing partition sizes	5
4 Comparisons	7
4.1 Neo4j	7
4.1.1 Local query evaluation	7
4.1.2 Distributed query evaluation	7
4.2 Apache Flink	7
5 Conclusion and Future Work	9
5.1 Applications of serverless architecture	9
5.2 Threats to applicability of this work	9
5.3 Future Work	9
Bibliography	11
Appendix	13

A My First Appendix**13**

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In a recent survey titled ‘Ubiquity of large graphs’[6], the authors noted that the size of graphs being used in both academia and industry is increasing. Although this survey found that it was common for graphs to have tens of billions of edges, it also found that scalability was the primary concern of all the participants surveyed[6]. In this thesis, we evaluate one way to alleviate this concern of scalability by decoupling storage of base graph from the query evaluation.

Today, all major cloud providers have a distributed cloud storage offering, Google cloud platform offers Google cloud storage[5], Azure offers Azure Blob storage[2], and AWS offers Simple Storage Service or S3[1]. These distributed storage services are already used by Big Data Analytics tools like Snowflake[7] and RDBMSs like Neon[4] for storage of underlying data. The use of these distributed storage services has not yet been explored for storing and querying graph data.

In this paper, we provide an initial analysis of how distributed storage services can be used to query large graphs. While using these distributed storage services, the primary issue that needs to be addressed is of latency. The latency of accessing data in a networked distributed system involves communication over the network which is at least three orders of magnitude more than accessing data from local storage. In return for this increased latency, we get almost unlimited throughput as read operations are distributed across a cluster of thousands of physical machines if not more. Therefore, in this paper, we evaluate ways to reduce the latency of accessing graphs. More precisely, we will focus on the performance of two commonly used graph traversal algorithms: Breadth first search (BFS) and Depth first search (DFS). We use these algorithms on LDBC datasets to provide results for multi-hop traversals.

1.1 Structure of the thesis

In Chapter-2, we provide the necessary background for this thesis. In Section-2.1 we talk about the history, architecture, and characteristics of distributed storage systems. We also give a brief overview of the capabilities of AWS S3, which is the service that we use in this thesis. Then, in Section-2.2, we discuss the serverless architecture of databases and its advantages over traditional architectures. Finally, in Section-2.3 we discuss the caching and prefetching strategies that we employ in this thesis to lower the latency of graph traversals.

Chapter-3 contains almost all the unique contribution of this thesis. We start by describing the architecture of our system and our evaluation methodology in Section-3.1 and Section-3.2. Then, in Section-3.3, we present the performance of a baseline solution which does not use any sophisticated caching or any prefetching. Then, in Section-3.4, we utilize a more granular caching algorithm along with an prefetching strategy which reduces the latency for accessing a node’s neighbours in the graph. After that, in Section-3.5, we discuss the impact of parallelizing the graph traversal algorithms. Finally, in Section-3.6, we explore the impact of different partition sizes on the overall performance of the system for various levels of parallelisms.

Chapter-4 compares our system with other systems which may be used for performing traversals on large graphs. In Section-4.1, we discuss the various methods that Neo4j provides for performing traversals on graphs. We then move on to discussing Neo4J's strategy for scaling to large graphs and possible shortcomings of their approach. We also talk about how these shortcomings also apply to traditional relational databases by extension. Then in Section-4.2, we evaluate the performance of a popular Big data framework named Apache Flink[3] for performing graph traversals and discuss its advantages and disadvantages.

Finally, in Chapter-5,

Chapter 2

Preliminaries

2.1 Distributed storage services

2.2 Serverless Architectures

2.3 Caching and Prefetching in distributed systems

Chapter 3

Experimental Evaluation

Experimental Evaluation

3.1 System Architecture

3.2 Query Evaluation

3.3 Baseline Implementation

3.4 Improving graph access service

3.4.1 Modifying CSR structure

3.4.2 Caching

3.4.3 Prefetching

3.5 Parallelizing graph algorithm service

3.5.1 Parallel BFS implementation

3.5.2 Parallel DFS implementation

3.6 Optimizing partition sizes

Chapter 4

Comparisons

4.1 Neo4j

4.1.1 Local query evaluation

4.1.2 Distributed query evaluation

4.2 Apache Flink

Chapter 5

Conclusion and Future Work

5.1 Applications of serverless architecture

sec:applications

5.2 Threats to applicability of this work

5.3 Future Work

Bibliography

- [1] AWS. Aws s3. <https://aws.amazon.com/s3/>. 1
- [2] Azure. Azure blob storage. <https://azure.microsoft.com/en-us/products/storage/blobs>. 1
- [3] Apache Software Foundation. Apache flink. <https://flink.apache.org/>. 2
- [4] Neon. Neon. <https://neon.tech/>. 1
- [5] Google Cloud Platform. Google cloud storage. <https://cloud.google.com/storage>. 1
- [6] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing. *Proceedings of the VLDB Endowment*, 11(4):420–431, 2017. 1
- [7] Snowflake. Snowflake. <https://www.snowflake.com/en/>. 1

Appendix A

My First Appendix

In this file (`appendices/main.tex`) you can add appendix chapters, just as you did in the `thesis.tex` file for the ‘normal’ chapters. You can also choose to include everything in this single file, whatever you prefer.