Development of an Agent-based Graph Database Benchmarking Dataset

Michelle Dea

Term Report Fall 2023

Project Committee:

Prof. Munehiro Fukuda, Chair
Prof. Wooyoung Kim, Member
Prof. David Socha, Member

1. Project Overview

The Multi-Agent Spatial Simulation (MASS) library is a parallel programming library using agent-based modeling to simulate a number of collective behaviors, ex: biological agents, and computational geometry algorithms [1]. MASS at its core is comprised of two main components, Places and Agents. Places represents a matrix of elements dynamically allocated over a cluster of nodes. These elements can exchange information with any other element, i.e. Place. Agents can perform computations, and can migrate between different Places, allowing them to interact with other Agents and Places.

When considering big data computing, frameworks such as MapReduce and Spark are some of the more common tools being used to process large volumes of data. However, these tools mainly deal with data in the form of text. For more complex data structures, such as graphs, where the data points are interconnected, a more suitable approach is to use a graph database. More specifically, agents can be leveraged to support analysis of graphs, as they can be deployed into data structures mapped over distributed memory. In this way, we can build a graph database application that can construct graphs over distributed memory.

Harshit Rajvaidya's recent work implemented an agent-based graph database using a MASS application, i.e. MASS Graph Database (MASS Graph DB) [2]. More specifically, his work used a combination of OpenCypher queries, and Agents to perform those queries. As part of his work, he conducted performance comparisons against two popular graph database systems: Neo4j, and RedisGraph. Both of which support OpenCypher queries. Rajvaidya's performance comparison was done using very simple graph datasets he created. The use of these simple datasets was largely to showcase the querying capabilities of MASS Graph DB, rather than execution speed. As part of a continuation of his work, my project focuses on creating several larger graph datasets (at a minimum 500 nodes, and 500 edges/relationships per dataset) for benchmarking purposes with a focus on evaluating execution speed and spatial scalability.

It was recently announced that the application RedisGraph will soon be sunsetted. In the interest of longevity, a new graph database application was selected for my project. Instead of RedisGraph, I will be using ArangoDB for benchmarking purposes.

We will be comparing our performance against Neo4j and ArangoDB to evaluate the potential of MASS Graph DB. We can identify the strengths and weaknesses of an agent-based graph database in comparison to these popular graph databases by using graph datasets that vary in different aspects, specifically type of data and size of the data. Specifically, I will be doing a comparative analysis of the execution speed (query throughput) and spatial scalability (query response time) of these three graph databases. To ensure the comparison is fair, I will be creating a tool that randomly generates the graph dataset based on certain parameters, such as topology and size, and I will be developing a way for the resulting dataset to be loaded into the three different graph databases. For my project, I identified three popular use cases for graph databases. These include recommendation engines, fraud detection, and social networks. The datasets for

each of these use cases are unique in their topology, which is why a parameter for this tool will account for topology to generate a realistic random graph for benchmarking.

The purpose of this work is to be able to measure execution performance of MASS against Neo4j and RedisGraph using these benchmarking applications. As part of these benchmarking applications, standard queries will be written to provide an efficient way for future researchers to benchmark their MASS Graph DB programs.

2. Goals

The main goal of this project is to create a benchmarking dataset and tools to simplify benchmarking MASS Graph DB against popular graph databases. The motivation behind this is to create tools such that future researchers can devote their time to improving the MASS Graph DB, rather than spending resources on investigating methods to conduct performance or execution testing. This work will also standardize how future MASS Graph DB changes will be evaluated, establishing consistency and improving accuracy when it comes to performance testing. To further outline the specific goals for this project, here are the explicit tasks I hope to accomplish:

1. Create tools to randomly generate several graph datasets that can be loaded into MASS Graph DB, Neo4j, and ArangoDB. These datasets are modeled after graphs pulled from real world applications based on the use cases mentioned above, and will be automatically generated with some randomness.
2. Create standard queries for each of the datasets that can be run across each graph database for measuring performance.
3. Identify the strengths and weaknesses of the MASS Graph DB compared to Neo4j and ArangoDB.
4. Create an efficient process or tool for running these programs for future researchers.

3. Achievements

This quarter began with the review of Harshit's previous work. I took a look into the GraphDB branch of the MASS Java application repository, specifically within the GraphDB folder. I started my code review with the GraphDBAgent, GraphDBHandler, and GraphDBNode. The GraphDBHandler held the main program to be run, which indicated use of two simple CSV files (Book1.csv and Book2.csv) as the program's input. Initially, I tried running this program to see the output of the match query. Unfortunately, the program would not compile successfully. This led to a further review of how the GraphDB Agent and GenericGraph work together in the GraphDB application. From this investigation, a fellow student, Lilian Cao, who is working on extending the capabilities of MASS Graph DB, concluded it was better to rebuild the Graph DB application without the use of GenericGraph, as it is structured in a way that is not flexible for extension of additional queries. From here, I shifted my focus away from familiarizing myself with Harshit's previous work, to identifying a new database to use for benchmarking.

RedisGraph was originally one of the graph databases I planned on using for benchmarking MASS' Graph application's performance against. However, Redis put out an end-of-life notice for RedisGraph. This meant we should consider a different graph

database to use for benchmarking purposes, one that would have longevity so that future researchers down the line could continue to use this application for performance comparison. In my research, I came across reviews of different graph databases that touched on features such as complexity, user base, and use cases. Among the databases I considered were OrientDB, Dgraph, TigerGraph, and ArangoDB. The reason ArangoDB was selected amongst these contenders came down to a few key factors. Firstly, ArangoDB is cloud agnostic. With the rising popularity of using cloud providers for application development, this is important for future work as it is flexible enough to work with any cloud provider. ArangoDB can also be deployed on any operating system (OS). This avoids issues where a future research is dependent on a specific OS. ArangoDB uses its own language, AQL, for querying its database. While this does increase complexity since there is an associated learning curve, the language is a combination of conventional coding and SQL, which increases its adaptability for developers who may not be familiar with AQL. There is also a good user base, which is an advantage when it comes to troubleshooting since other users likely ran into similar issues.

After selecting a new graph database to use, I turned my attention to identifying practical datasets to use as the initial datasets for benchmarking. Graph databases have a variety of use cases, of these the more common ones include: recommendation engine, fraud detection, and networks (ex: social media or network management). For recommendation engine, I will be using the Amazon Product Co-Purchasing Network [3] dataset. For fraud detection, I have selected a Cryptocurrency Transactions [3] dataset. And for a social media network, I have selected a Twitch dataset [4]. Next quarter, I will be using these datasets as the foundation for my random graph generator tool. This tool will take in a parameter such as topology or use case and return a randomized graph that is modeled after these practical datasets.

With the datasets selected, I worked on transforming the datasets into a format that could be imported into Neo4j and ArangoDB. I started with the Amazon dataset, and wrote a Java program to convert the data into an importable format for Neo4j. I then used OpenCypher queries to import the data into Neo4j, and successfully created the graph.

I used the same dataset for importing into ArangoDB. For ArangoDB, I used the terminal to execute commands to import the data from the CSV files.

4. Results

This quarter, I focused on establishing the foundational pieces for performance testing once the MASS Graph DB is ready. Part of this was being able to successfully create a graph database within Neo4j. I started with the Amazon dataset that contained over 3 million edges, and over 400,000 nodes. I wrote a program that updated the format of the data into an importable csv for Neo4j.

The size of this graph was too large for my machine to import, as Neo4j currently runs locally on my personal laptop. As discussed with Professor Fukuda, MASS can generally handle graphs with around 8000 nodes [5]. This led me to create a smaller graph with 7900 nodes instead.

To create a graph such that the products (nodes) were connected to co-purchased products, I created two sets with the same nodes and then used the edges information to connect the nodes using a BOUGHT_WITH relationship to establish a co-purchasing relationship. A snippet of the resulting graph is shown in Figure 1 below.

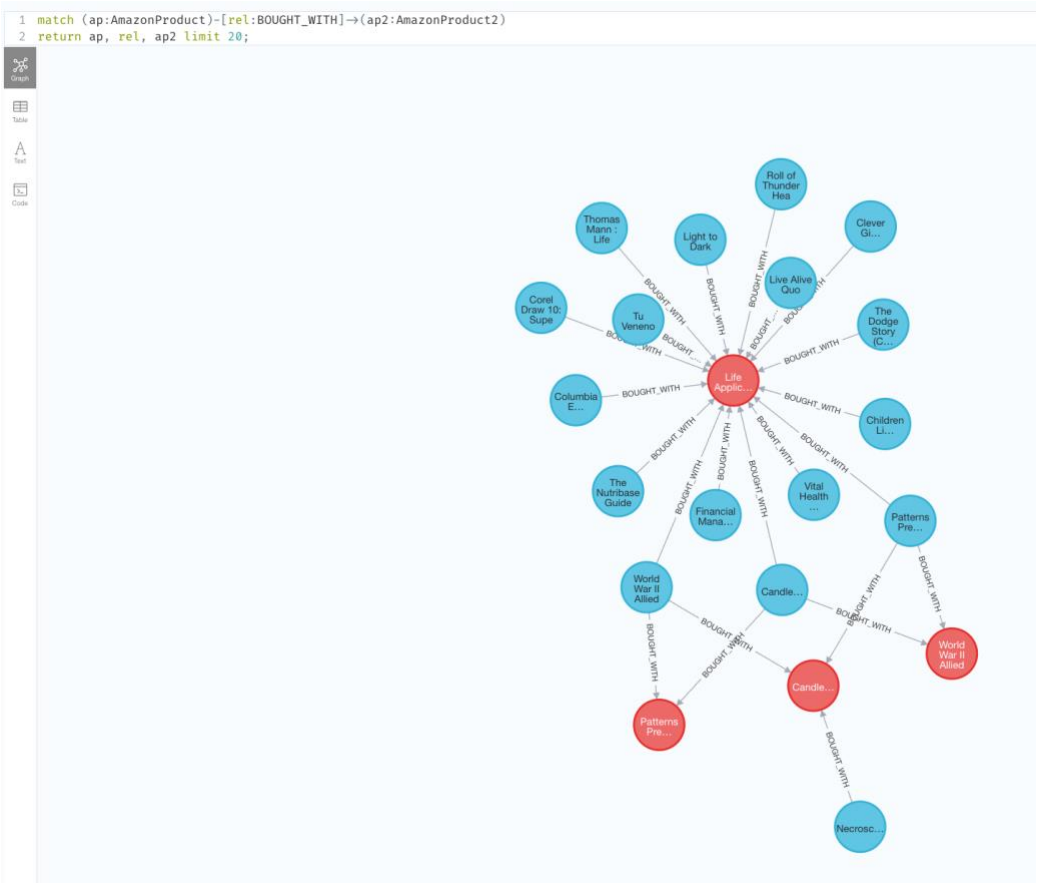


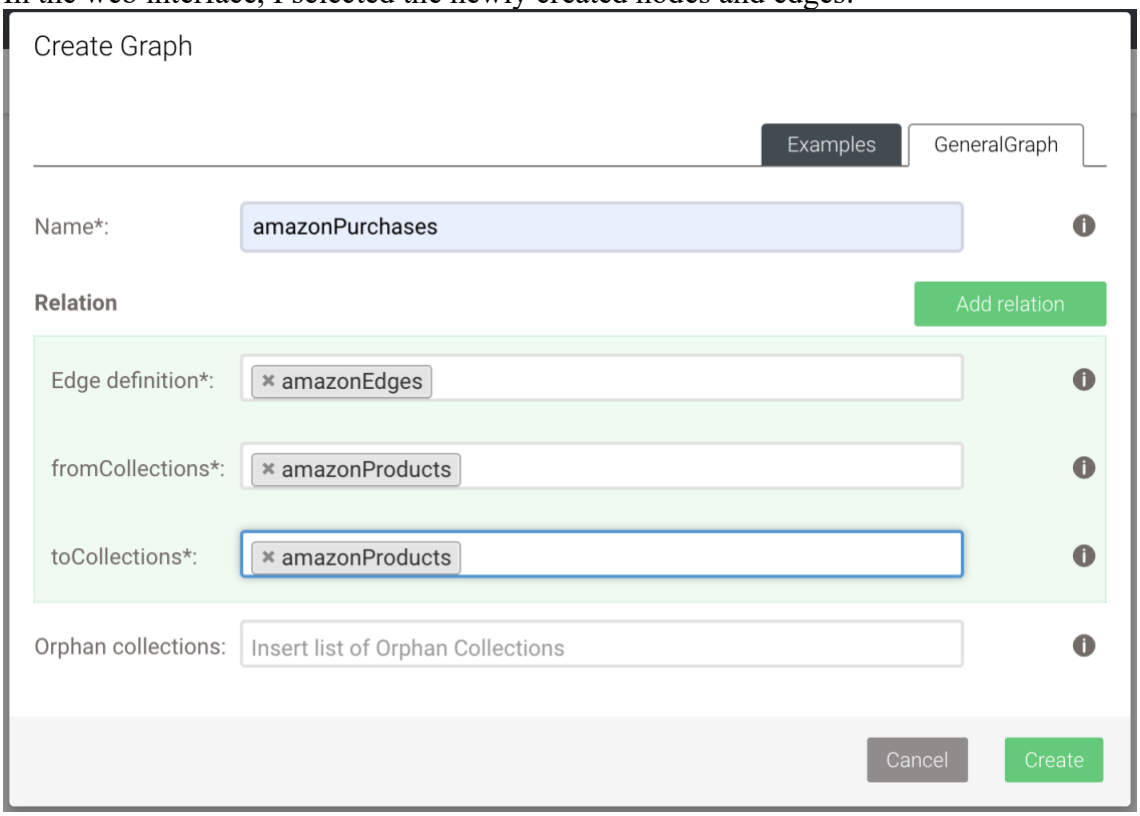
Figure 1: Display of Amazon graph in Neo4j limited to 20 nodes

With ArangoDB, I am using a docker container to run the database. The web interface can be accessed via http://localhost:8529/. ArangoDB organizes their data into collections, which in this case can be either a collection of nodes, or a collection of edges. The import here is a bit trickier, as you need to specify how the edges correspond to the node collection. I was able to do so using this command by specifying a prefix of the node collection:

```
arangoimport --file "amazonProduct.csv" --type csv --collection
"amazonProducts" --create-collection true

arangoimport --file "amazonEdges.csv" --collection amazonEdges --
create-collection true --type csv --create-collection-type edge --translate
FromNodeId=_from --translate ToNodeId=_to --from-collection-prefix
amazonProducts --to-collection-prefix amazonProducts
```

In the web interface, I selected the newly created nodes and edges:



A snippet of the resulting graph is displayed in Figure 2 below.

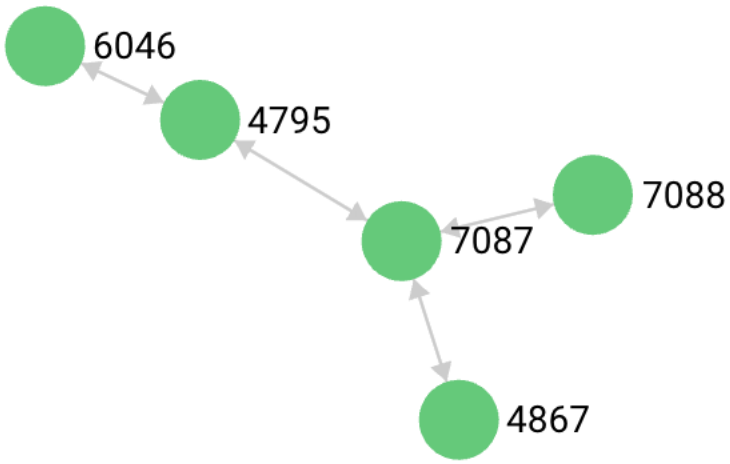


Figure 2: The Amazon graph displayed in ArangoDB.

5. Next Quarter's Plan

Due to some technical difficulties with getting up to speed on some of the syntax for these graph databases, I will be adjusting my plans for next quarter to include the following:

Winter 2024

Week	Tasks
January Week 1 & 2	Update csv formatter program to use " | " as a delimeter for MASS importing Import Amazon data into MASS using the QueryDB branch developed by Lilian Convert the remaining graphs into the appropriate formats for all three graph database applications. Create standard set of queries for benchmarking – beginning with "Create" command
January Week 3 & 4	Start work on random graph generator • Identify inputs/parameters for generator • Create functions to output graph in CSV format Finalize standard queries for benchmarking program
February Week 1 & 2	Create program for automating benchmarking GraphDB application performance using benchmarking datasets • Program should be easy to run and compatible graphs generated from random graph generator Include thorough instructions for running program
February Week 3 & 4	Continue working on automated benchmarking program
March Week 1 & 2	Write up quarter term report

6. Summary

This quarter I was able to accomplish the foundations of what I hoped to achieve, which was familiarizing myself with graph database creation within Neo4j and ArangoDB. The syntax and nuances for both systems proved a bit challenging as I have not worked with these databases before. With Neo4j, it is worth noting that by establishing constraints on the database, the commands used to import data need to altered to reflect the established constraints. For example, specifying that each node in the graph will contain a unique ID requires the user to specify that there will be no null IDs in their Cypher queries.

The limitation of using my personal device to run these databases introduced a limitation of the size of graph I am able to create. As a solution to this, we may be able to leverage the school Hermes machines to run the database applications. Though I suspect a challenge here will be accessing the web interface in order to execute queries and visualize the graph. This will be part of my future work, to explore the capabilities in this area.

7. References

1. "MASS: A parallelizing library for multi-agent spatial simulation." [Online]. Available: http://depts.washington.edu/dslab/MASS/
2. "An Agent-based Graph Database" White Paper, Accessed on: June 30, 2023. [Email]. Available: via Professor Munehiro Fukuda.
3. Leskovec, J., & Krevl, A. (2014, June). SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data
4. "Recommendation on Live-Streaming Platforms: Dynamic Availability and Repeat Consumption"
   Jérémie Rappaz, Julian McAuley and Karl Aberer
   RecSys, 2021
5. "Agent-based Graph Applications in MASS Java and Comparison with Spark" Term Paper. [Online]. Available: https://depts.washington.edu/dslab/MASS/reports/CarolineTsui_whitepaper.pdf