Vikrant Verma
CSS 497
Wi26 Term Report

Introduction

DSLab is actively developing a distributed graph database system built upon the MASS (Multi-Agent Spatial Simulation) library. The original GUI for visualizing the resulting graph was based on Cytoscape. While Cytoscape is capable of visualizing static and small-to-medium sized graphs, it presents significant limitations regarding the high-performance 3D rendering and general interactivity required to visualize the complex, dynamic, and large-scale graphs generated by the MASS library.

My mission last quarter and this quarter has been to establish a more robust and feature-rich visualization framework to replace Cytoscape. This framework must specifically support:

- High-performance 3D rendering for large graph topologies.
- Real-time, dynamic updates to the graph structure (nodes and edges).
- Visualization and real-time tracking of independent Agent entities interacting with the graph topology.
- Support for property graphs.

This paper details the visualization infrastructure, showcasing the final implementation of a real-time, dynamic 3D graph viewer that I have spent the last two quarters building.

Survey

Graphosaurus was selected because it provides the necessary 3D rendering foundation using modern WebGL techniques, which ensures high performance and interactivity, even with massive datasets.

While Graphosaurus was initially designed for static graphs, its foundational suitability for 3D visualization made it ideal. The choice was validated by successfully transforming the library to support the following:

- Dynamic Support: I engineered a custom, socket-based architecture to enable real-time data injection over WebSockets, a critical feature for a live MASS simulation.

- Agent Tracking: The 3D environment was easily extended to support Agent objects, allowing us to visualize and track independent entities moving within the graph.

The shift to Graphosaurus was an architectural necessity to support the full complexity of a dynamic property graph system.

Design and Implementation

While Graphosaurus provided the necessary 3D rendering foundation, it lacked the dynamic capabilities required for our project out of the box, as it was primarily designed for visualizing static graphs. To bridge this gap, I forked the repository and engineered a dynamic loading architecture. I implemented a custom Graphosaurus server that utilizes message passing over WebSockets, allowing nodes and edges to be injected into the visualization in real-time rather than relying on pre-loaded static datasets. Additionally, I successfully implemented "Agent" objects within the graph environment, allowing us to visualize and track independent entities as they interact with the topology.

During the winter quarter, I implemented full support for key-value pair property graphs, enabling richer metadata for both nodes and edges. This was made possible in part by the existing `PropertyGraph` functions that expose the necessary functionality for gathering property vertex information to send to the frontend. Properties are sent in the message to the frontend in key-value pairs that can be named arbitrarily and will show when clicked on.

Instructions on how to use this version of Graphosaurus can be found here, along with the rest of the repository: https://github.com/vikv1/mass-graphosaurus/blob/master/README.md

The changes to MASS Java have not been merged yet, but the main components are within the following files that I have added:

- **GraphosaurusListener.java** – WebSocket client that polls agent locations and sends updates via a queue utilized by a messenger thread.
- **GraphosaurusMessage.java** – Message types for the Graphosaurus protocol
- **AgentLocationTracker.java** – Tracks agent positions and detects changes. Also tracks what information has already been sent to the frontend to avoid resending information.

Figure 1 below shows the new architecture for the Graphosaurus integration.

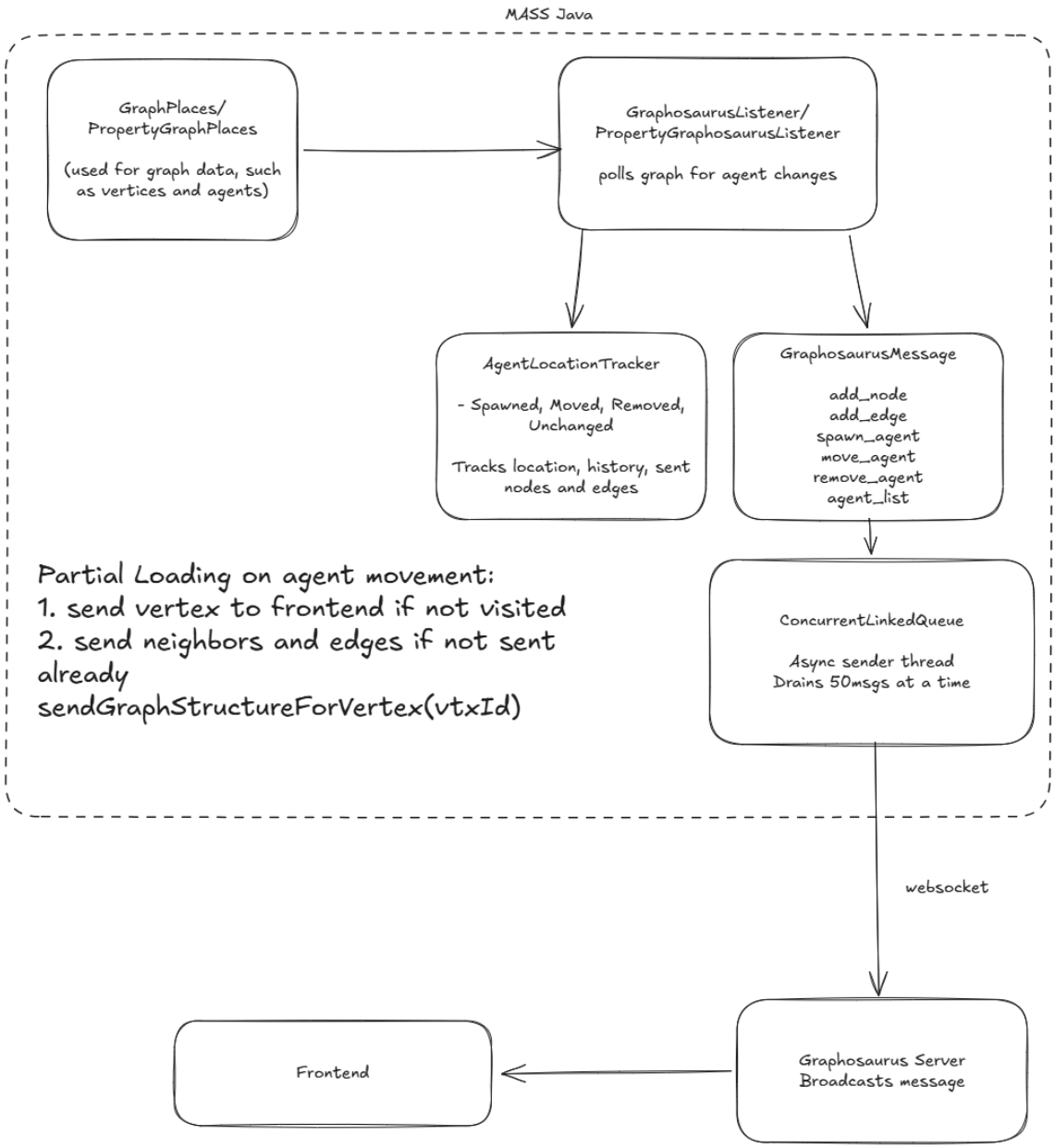

Figure 1

To enable the MASS Graphosaurus integration, the following flags can be passed, which will cause `GraphPlaces` to create the new listener and automatically broadcast all changes to Graphosaurus.

```
java -Dgraphosaurus.enabled=true \
     -Dgraphosaurus.websocket.url=ws://localhost:8080 \
     -Dgraphosaurus.poll.interval=500 \
     …
```

Figure 2 depicts the typical data flow from MASS to the Graphosaurus frontend, including an example message payload.

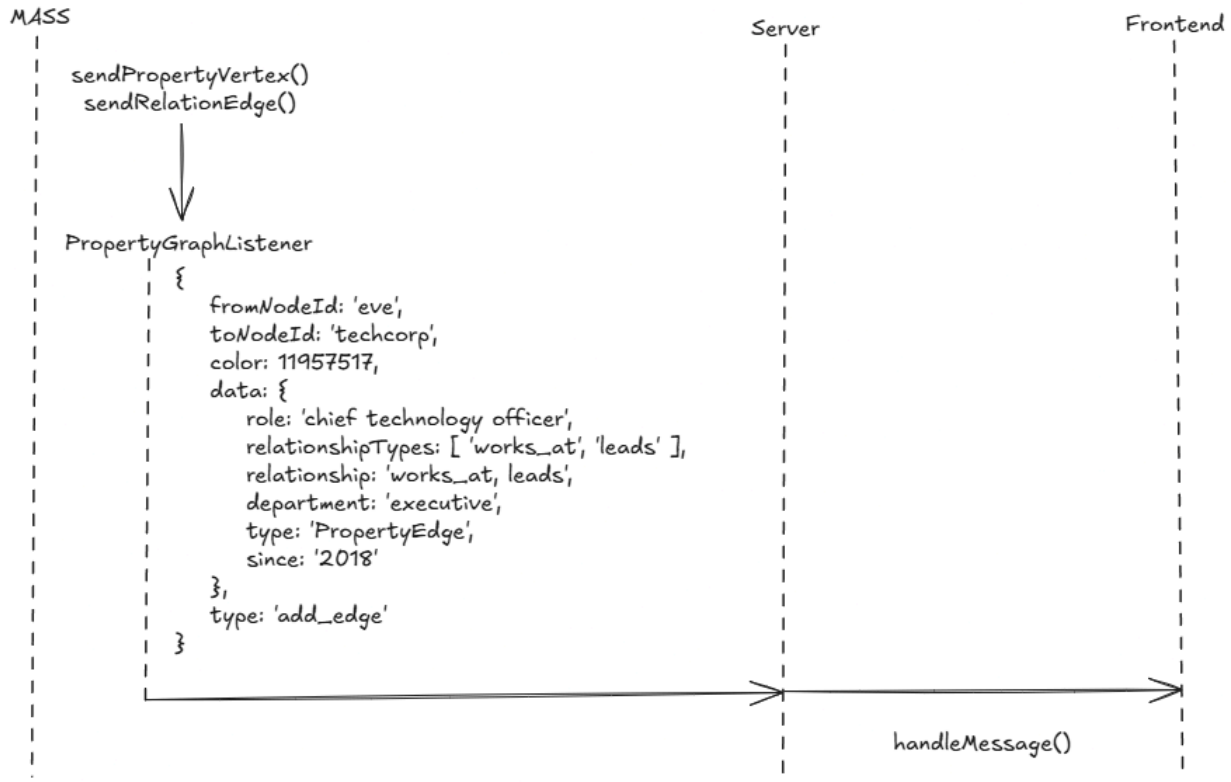

Figure 2

Evaluation

The following execution snapshots, presented in Figures 3, 4, and 5, provide evidence that the core goals for the Winter quarter were successfully met. The image displays a working 3D graph visualization featuring nodes, edges, and a property viewer that depicts properties sent from MASS. This dynamic, socket-based environment successfully replaces the static 2D limitations of Cytoscape. The implementation of property graphs for MASS using the forked Graphosaurus library exceeded my initial expectations, although performance at high node counts is an issue that is yet to be addressed.

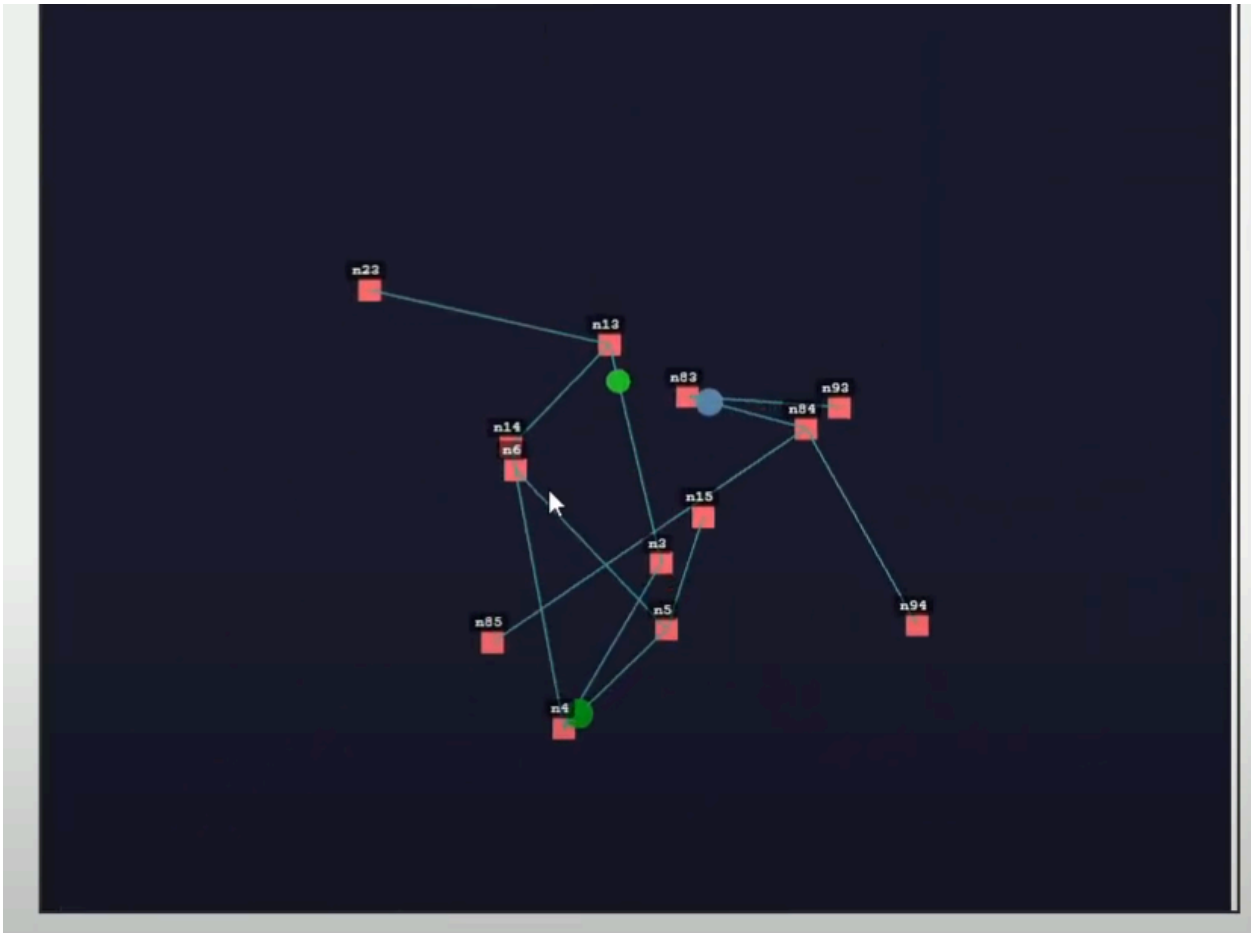

Figure 3

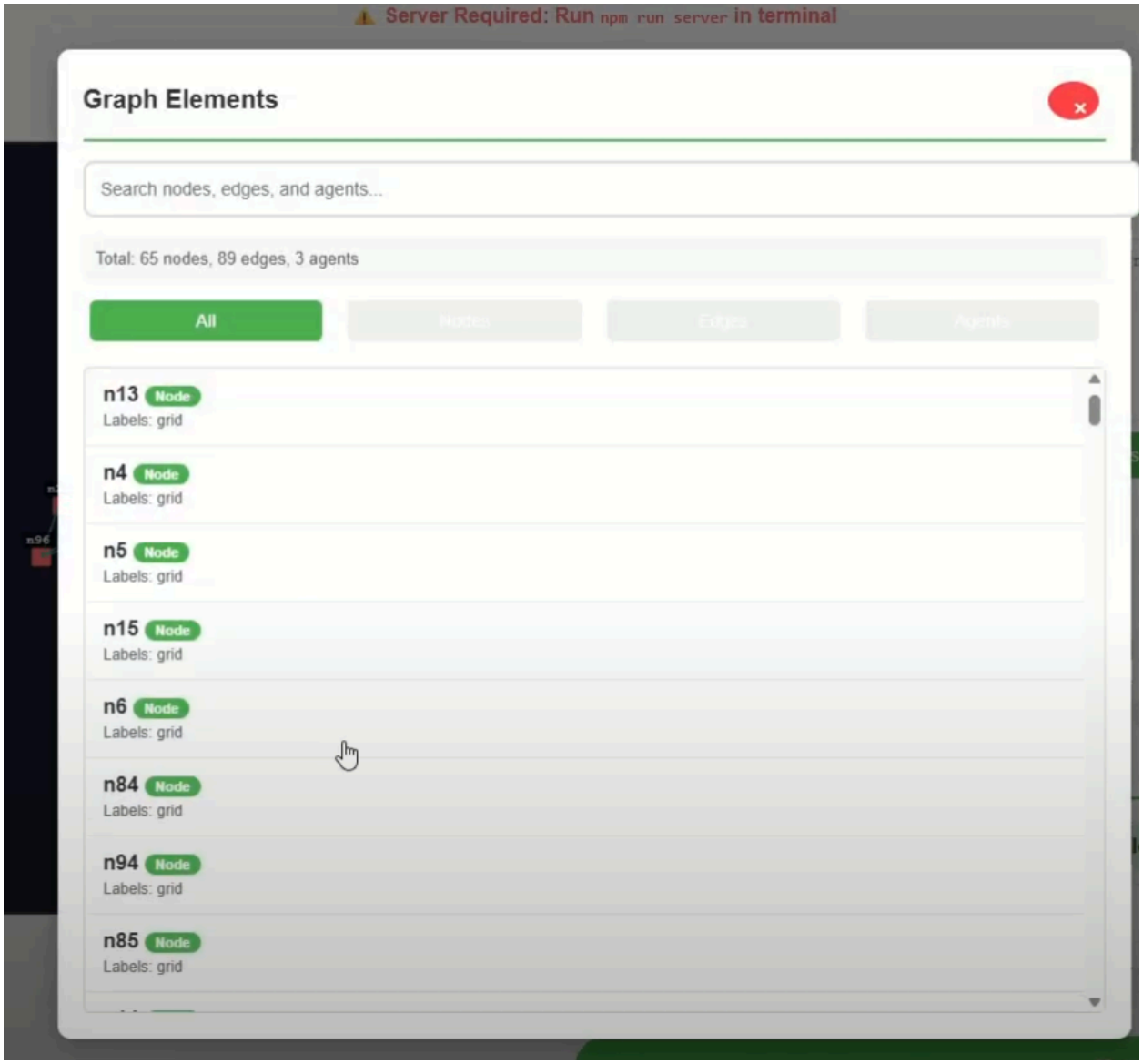
Server Required: Run npm run server in terminal
Graph Elements
Search nodes, edges, and agents...
Total: 65 nodes, 89 edges, 3 agents
All
n13 Node
Labels: grid
n4 Node
Labels: grid
n5 Node
Labels: grid
n15 Node
Labels: grid
n6 Node
Labels: grid
n84 Node
Labels: grid
n94 Node
Labels: grid
n85 Node
Labels: grid

Figure 4

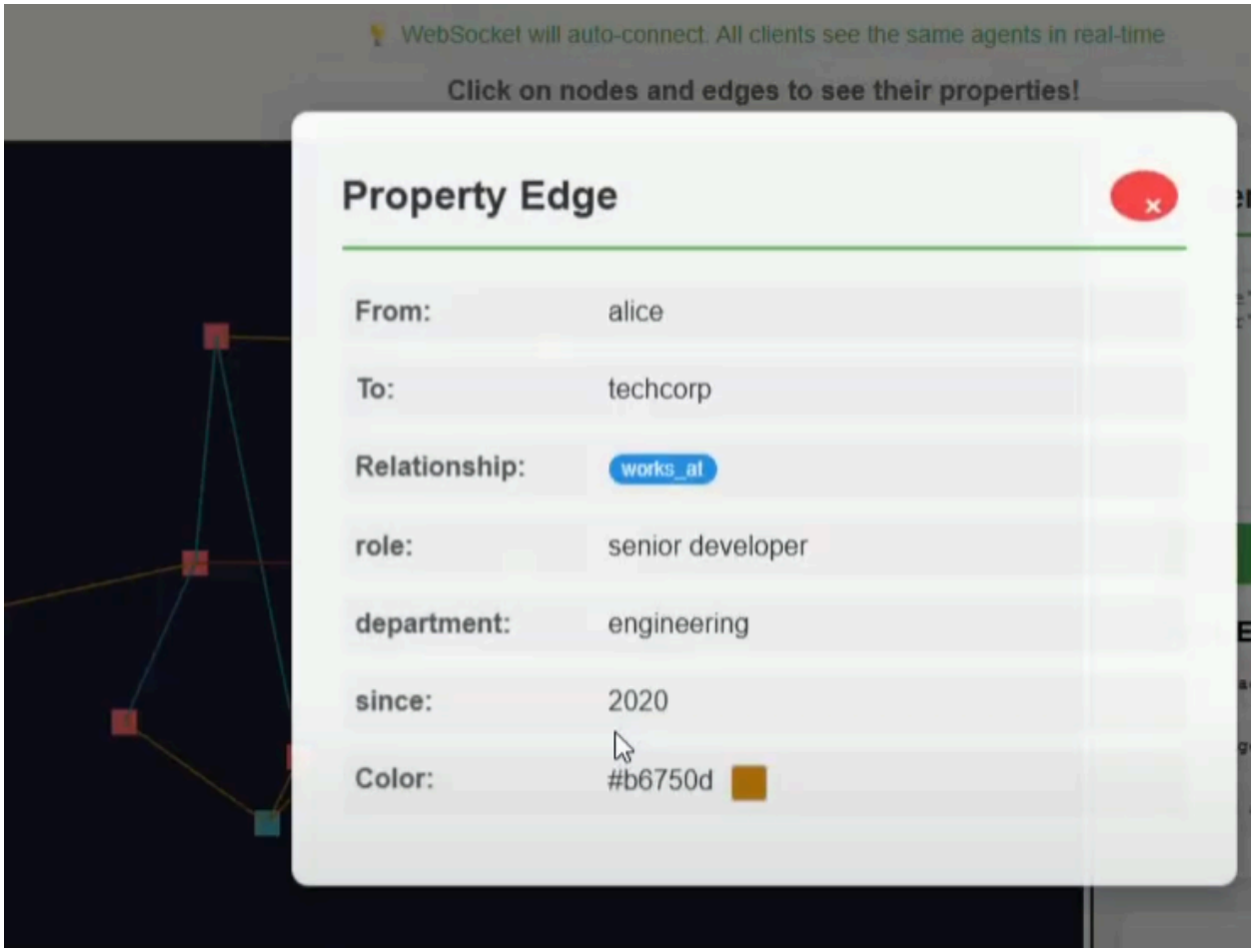

Figure 5

Final Comments

My primary achievement this term was the successful implementation of a dynamic, socket-based 3D environment using the Graphosaurus library, which now supports property graph inspection.

Some future work that would make this project even better:

- Solve performance problems that occur at large node counts. This is largely due to the limitations of browser-based rendering, so potentially we may want rendering to occur before final browser rendering at some middle step.

- Instead of polling GraphPlaces, use some kind of a log similar in function to a write-ahead log, where the GraphosaurusListener can just read the next change and directly send a message rather than getting the whole graph and checking for changes.