Managing Complexity of Radio Interferometry Observatories Using Graph Technologies

Anatoly Zavyalov, working with Adam Hincks
University of Toronto

Modern observatories, especially those that consist of many dishes, are made up of thousands upon thousands of physical components. One such observatory is the upcoming Hydrogen Intensity and Real-time Analysis eXperiment (HIRAX), which consists of a 32×32 array of radio interferometric telescopes, and will be constructed in the Karoo region of South Africa. The experiment aims to study the evolution of dark energy throughout the universe, as well as detect radio transients such as fast radio bursts and pulsars. In an observatory of such a large scale, it is necessary to be able to track all of its components over time. This includes storing the properties of components as well as how these properties change over time, tracking connections between components, and finding paths from one component to another. For example, we need to track how the antennas are connected to the computer where the data is stored and processed, and what components the signals captured by an antenna go through before they reach the input. Overall, there needs to be a system that allows astronomers to access and update these metadata, as well as perform complex queries like the ones described above. My project is the research and development of such a system for HIRAX that allows astronomers to be able to answer these questions with the help of a Python API and graphical web interface.

Going into the project, we knew that a graph, a mathematical structure consisting of vertices and edges that connect vertices, can store the components and connections between them in a very intuitive way: the vertices may represent components, and the edges may represent the connections between the components. Hence, if this information can be best modeled with graphs, it makes sense to store this data as a graph as well. Unlike conventional relational databases that organize data in tables, graph databases organize data into a native graph structure. We settled on JanusGraph, which is a free and open-source graph database that is actively supported and has a large community, which are all important factors for software used in scientific collaborations like HIRAX. JanusGraph also allows us to connect to it using a Python interface called Gremlin-Python, making it perfect for our needs. We next developed the Python API, tailoring it to our specific needs, and spending lots of time on determining the best way to structure it. After writing the core Python API to interface with the graph database, we started developing the web interface. We wanted the web interface to directly use the Python API (reducing the amount of code we would have to write), so we used the Flask web framework which allows us to call Python functions from the API directly from the web interface. Finally, for the web frontend, we used the React JavaScript library to build a user interface.

Overall, we have created a working Python API as well as the beginnings of a web interface, the functionality of which can be extended by using the existing foundations that we have built so far. A great aspect of our current implementations of the Python API and web interface is their modularity, allowing us to add more features directly on top of existing features and without affecting them in the process.