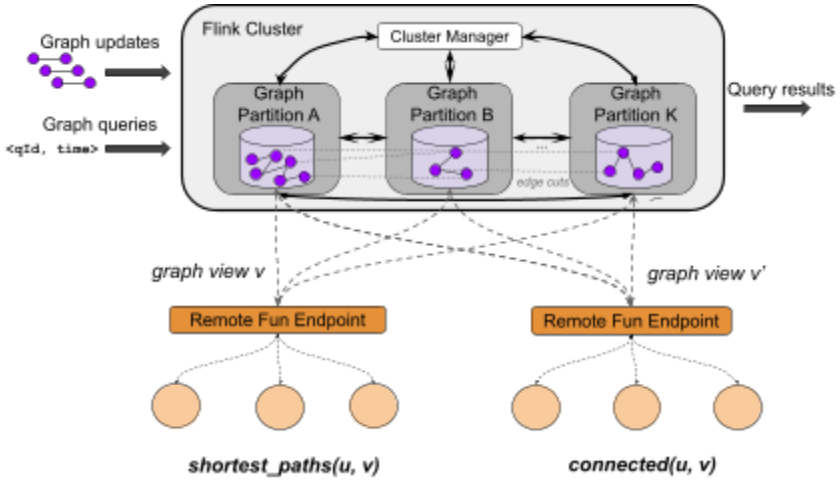
Streaming graph analytics with Flink Stateful Functions

Introduction

In this project, you will design and implement a temporal graph processing API and library on top of Apache Flink's Stateful Functions. You will have to think about the time semantics, partitioning strategies, graph representation in managed state, and how to translate graph algorithms to Stateful Functions topologies.

Background

Graph streaming analytics is an emerging application area that aims to extract knowledge from evolving networks in a timely and efficient manner. Graph streams are sequences of timestamped events that represent relationships between entities: user interactions in social networks, online financial transactions, driver and user locations in ride-sharing services. Graph streams are continuously ingested from external, often distributed, sources and are modeled either as streams of edges or as vertex streams with associated adjacency lists.



Stateful Functions is a framework for building general-purpose distributed event-driven applications. You can think of it as "Functions as a Service on top of Streaming", where functions are executed by dataflow operators and the Flink runtime orchestrates their execution. Functions can run embedded into flink tasks or as remotely as lambdas.

Stateful Functions provide a programming interface similar to that of *actors*: functions can send and receive messages and they can create and maintain state. Compared to Flink programs, Stateful Functions offer much greater flexibility: they can communicate with any other function in an arbitrary way, so they let us build iterative computations. This characteristic makes Stateful Functions a promising basis for graph processing.

Project goal

You will design and implement a library of streaming graph algorithms on top of Stateful Functions. Your implementation should support two types of input streams: (i) graph updates, i.e. edge additions, deletions, and modifications, and (ii) temporal queries. Your library should propagate graph updates to the correct Flink partition and maintain the evolving graph up-to-date. Query events should retrieve the relevant graph view, spawn graph computations on local or remote functions, and send the results to the egress. The graph state should be organized carefully in order to support both low-latency updates and time-based search. For example, a possible query might be "*return all k-hop neighbors of vertex u as of timestamp t*". Your system should retrieve a graph view containing edges with timestamp >= t from managed state and then spawn a set of functions to compute k-hop neighbors on this view. Since Stateful Functions decouple state from computation, the system should be able to keep serving graph updates while the query is running.

Stretch Goals

1. Add support for cross-graph analytics. Consider the case where input streams can represent updates not just for a single graph but for several. We can use Flink's managed state to keep a collection of graphs updated and spawn functions to answer cross-graph queries. For example, combine a social network graph with a real-time location graph in order to provide personalized recommendations to users based on their friends' preferences.
2. Add support for fixed-point graph algorithms. In order to achieve this goal, you will have to think about progress tracking in an asynchronous environment. How can we know when an algorithm has converged without flooding the network with messages?

Required skill set

- Experience with Java.
- Familiarity with graph algorithms.

Where to start

- Flink Stateful Functions: https://ci.apache.org/projects/flink/flink-statefun-docs-stable/
- Chronos: A Graph Engine for Temporal Graph Analysis: https://dl.acm.org/doi/pdf/10.1145/2592798.2592799
- Snap datasets: https://snap.stanford.edu/data/#temporal
- Yelp dataset: https://www.yelp.com/dataset