DS Course Project (2022): Repetitive Shortest Paths

Project idea for “Engineering Distributed Infrastructure: A MSc Course”: go/ds-course

idyczko@google.com

Main goal

The main goal of this project is to design and implement a system capable of finding shortest paths in a graph, repetitively. The graph will be delivered by the project mentor. By “repetitiveness” we mean an ability to scalably serve find-shortest-path queries with given start and target vertices, after the system stabilizes.

Graph properties

The graph delivered by the project mentor will be large enough not to fit into a single machine to enforce distributed representation. It will also be relatively sparse ($|E| = O(|V|)$), weighted and planar. Additionally, the 2D layout of the vertices will be provided to guide the sharding process and enable using heuristics to find shortest paths. Given the graph properties, a natural application of this system might be a routing server running on a static road map.

Memory constraints

Additional resource constraints may be put onto the used infrastructure to make the problem relevant for smaller graphs as well. These constraints will only limit used memory: the team will be instructed to only use a certain type of machines and to keep the memory usage across the cluster below predefined threshold after the system stabilizes. System is considered stable when all the preprocessing has been completed and it is ready to serve queries.

Main focus

Though there is a lot of nuance to designing and implementing a system as described above, the main focus of this project is to cover the following points:

- expressing a well known CS problem in a scalable way,
- sharding state without data duplication,
- designing and implementing an algorithm to run on the distributed state.

However, this is only the author's vision! If your team wishes to work within the boundaries of the problem above, but focus on other details, don’t hesitate to reach out - I am sure we can make it a more enjoyable challenge with some tuning. It is important the team understands they are not expected to create “the perfect system” in every possible

dimension of this problem. Solving this problem holistically is challenging, so please, choose your battles - pick the areas you want to focus on.

Possible extensions and alternative focuses

Here are some possible extensions to the problem and alternative focuses you might want to consider. If something looks interesting to you, we can add it to your team's "main focuses" list, or swap it for one of the items there. This list is nowhere near finished - so don't worry if something you'd like to work on hasn't been mentioned.

- **Graph change support**: add/remove edge, add/remove vertex. You may assume neither the edge nor the vertex breaks planarity, but you don't have to, if you want to solve this problem yourself. Additionally, you can consider how the change affects preprocessed data - if you are using some "clever" approach to find shortest paths, you might have decided to store some additional data on the machines, which might be inconsistent with the new changes.
- **Density-sensitive sharding**: you might want to consider the graph's local density when sharding the data. This extension may be connected to the previous one, as changing the graph also changes the local density, so it might be a good idea to enable the system to recalculate the sharding.
- **Dynamic horizontal scaling:** you may design your system to reorganize when it is receiving increased load. You can do it globally (for the whole graph) or locally (for the graph regions with increased queries). It is reasonable to assume that the more dense regions will receive more queries, which justifies density-sensitive sharding.
- **Fault recovery**: make your system resilient to lost state - if one of the system units crashes, its state should be recreated to enable serving further queries.
- **Data redundancy**: make the sharded state resistant to single-machine failures, i.e. a single machine failure shouldn't (greatly) affect performance - we shouldn't have to wait for the system to recreate the partially lost state before being able to serve further queries.