Tyler Ekaireb
Professor Gilbert

CS 292F
April 27th, 2022

Final Project Proposal

Introduction

In order to handle large computational problems, or service the computing needs of many users at once, it is necessary to make use of multiple computers in coordination. Typically, these computers must communicate over a network in order to work in tandem and accomplish their shared goal. Yet, this form of cooperation tends to incur significant overhead. A great many applications would benefit from a reduction of this overhead, necessitating more optimal distributions of the workload in order to minimize network communication. If the workload is modeled as a graph, then graph partitioning can be used to divide the work among nodes while severing as few dependencies as possible.

While finding optimal solutions is not computationally feasible for all but the smallest of problems, the spectral methods we learned in class provide effective heuristics for finding near-optimal solutions. I plan to build a system that will make use of these techniques to automatically divide a distributed application among nodes. I will also implement at least one other graph partitioning algorithm to compare against the spectral algorithm.

Base Deliverables:

1. Motivate problem with distributed microservice application deployment (conceptual)
2. Implement graph partitioning using the Fiedler vector (spectral method)
3. Implement additional partitioning method for comparison, such as the Kernighan-Lin algorithm (non-spectral method)
4. Compare performance across various metrics

"Reach" Deliverables:

5. Create system to generate a dependency graph from a distributed application based on a given workload
6. Simulate deployments of a distributed application based on graph partitions
   a. Include a more realistic end-to-end performance benchmark
7. Implement more graph partitioning methods for comparison

Evaluation

I plan to create several graphs of varying sizes and use existing graphs from several repositories, which should be representative of distributed applications in order to test the graph partitioning algorithms. In the simplest case, I may use unweighted graphs to represent all connections between different microservices as equivalent. However, I would like to use weighted graphs to represent more strongly associated microservices (i.e., those that issue more requests to one another). I will test both the quality of the partitions (e.g., balance of the partitions and the sum of the edges cut) and the performance of the algorithms (e.g., execution time and how they scale as the graphs grow larger).

I found a dataset with unweighted dependency graphs, known as the Microservice Dataset[1], which includes the dependency graphs of twenty microservice applications, one of which is shown in Figure 1. I would like to use a few of these graphs in my tests, as they are representative of actual applications. However, they are all quite small, so I will also include larger graphs from the SuiteSparse Matrix Collection[2] to evaluate the graph partitioning algorithms.

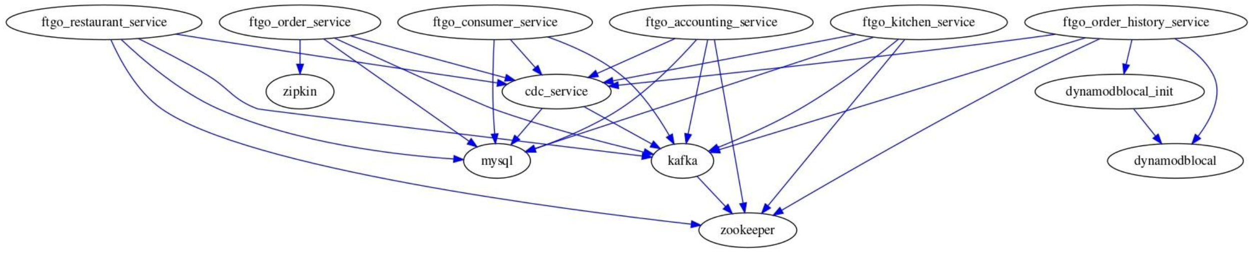

Figure 1: A sample dependency graph from the Microservice Dataset

If time allows, I would like to motivate the problem more directly. I hope to create a system to monitor simple distributed applications and generate graphs based on their execution under a particular workload. Each node will represent a microservice, and each edge weight will quantify the network traffic between microservices over the course of the sample workload. These graphs will then be fed to the graph partitioning algorithms in order to derive optimal deployments. Instead of only comparing the graph partitions from a purely theoretical standpoint, I will simulate these generated deployments and evaluate the algorithms based on the optimality of the deployment. I may calculate the total amount of network traffic (less is better), or impose network delays in my simulation and calculate the average latency and throughput of each deployment.

If I have more time, I may implement additional graph partitioning algorithms. I would evaluate them in the same way as I plan to evaluate the spectral and non-spectral methods: quality of the graph partition based on balance of the partitions and the sum of the severed edges, as well as runtime performance of the algorithm itself, including absolute performance against the other algorithms and how it scales with larger inputs (i.e., time complexity).

[1] https://github.com/clowee/MicroserviceDataset
[2] https://sparse.tamu.edu/