An Analysis of K-Server algorithm performance

Stefan Caldararu[1] and Marc Renault[2]

[1] *Undergraduate Student with Department of Computer Science, UW-Madison*
[2] *Professor in the Department of Computer Science, UW-Madison*

May 4, 2023

Department of Computer Science
University of Wisconsin – Madison, USA

Abstract

Write something here...

Keywords: add keywords...

Contents

1 Introduction

1.1 Outline

This paper considers the K-Serversproblem, and the practical development of algorithms within a C++environment. Additionally, we both provide a literature review, and multiple forms of analysis of a variety of algorithms. In sec. 1.2, we provide a problem description for the K-Serversproblem. In section 2 a brief literature overview is provided, and then sec. 2.2 describes different performance metrics that can be used when looking at these algorithms. In section 3, multiple popular K-Serversalgorithms are described, and a brief list of benefits and drawbacks are provided. We then describe the software implementation for the algorithms in sec. 4, and some analysis of numerical experiments in sec. 5. Finally, we provide some analysis of results in sec. 6, and propose future research thrusts in sec. 7.

1.2 Problem Description

An instance of the K-Serversproblem can be described by a metric space $M = (X, d)$, a number of servers $k > 1$, and an input sequence $\sigma = (r_1, r_2, r_3, ..., r_n)$, which each correspond to a point in the metric space. Each of the k servers is assigned an initial starting location within the metric space (generally this is assumed to be the first k requests of the input sequence). Following this, The sequence of reqeusts is processed one at a time. When a request comes in, a given algorithm ALG must decide on one of the servers to service the request. It must then move said server from it's current location x to the request r_i. This incurrs a cost of $c = d(x, r_i)$. The goal is to have ALG incurr the smallest possible cost while servicing all of the requests in the sequence [1].

There are two major distinctions to be made between different classes of algorithms. The first is the classification of a "lazy" algorithm - one that only moves a server in order to service the current request. Non-lazy algorithms will process a request, and then potentially also move other servers preemptively in order to prepare for future requests. It is important to note that for any non-lazy algorithm that performs well, there is a parallel lazy algorithm that performs just as well, if not better. We can describe this algorithm as follows: suppose we have our non-lazy algorithm, ALG. We have the algorithm $LAZY$ service requests with the same serverst that ALG services requests. Suppose that ALG moves a server from location x_1 to location x_2, and then later services request r_i with this server. $LAZY$ will be servicing r_i with the same server, except it will be moving from x_1 instead of x_2. By the triangle inequality, $d(x_1, r_i) \leq d(x_1, x_2) + d(x_2, r_i)$. By applying this principle throughout the request sequence, we will see that the cost of $LAZY$ will be as good if not better than that of ALG. This shows us that we can create a lazy algorithm from a non-lazy algorithm by maintaining "ghost" locations for servers in relation to how the non-lazy algorithm would use them. Then, we can determine which server the non-lazy algorithm would use, and then service that location with the correct server [1].

The second major distinction is between "online" and "offline" algorithms. An offline algorithm receives the entire request sequence at once, and so as a result is able to make decisions on what server to use for the current request based off of future requests. In contrast, an online algorithm receives the request one at a time, and so is only able to make decisions based off of past requests, the current server configuration, and the current request. While online algorithms are at a severe dis-

advantage due to this, real world applications often rely on the performance of these algorithms [1]. Applications range from disk access optimization, such as the two headed-disk problem **include citation**, to **include other examples, police/firetruck service?**.

2 Background

2.1 Literature Overview

2.2 Analysis Methods

Competitive Analysis

The most popular method for analyzing the performance of online algorithms for a problem is competitive analysis. Here, we assume a "malicious adversary", who is attempting to make our algorithm ALG perform as poorly as possible in relation to the performance of the optimal algorithm OPT. The malicious adversary is allowed to come up with any finite input, and our competitive ratio is determined by this. If for any finite input, we are able to guaruntee that ALG has a cost within a certain ratio c of OPT (allowing for a constant additive factor), then we say that our algorithm is c-competitive [1]. So, if we define $ALG(\sigma)$ as the cost ALG incurs while processing request σ, then we have the following definition:

Definition 2.1. Algorithm ALG is said to be ***c*-competitive** if for every finite request sequence σ, $ALG(\sigma) \leq c \cdot OPT(\sigma) + \alpha$ for some constant α.

Competitive analysis is in some sense similar to "worst case" analysis, where we try to see how poorly an algorithm will ever perform. This may not always be as useful in practice, as there are algorithms that perform very well on most inputs, but given specific inputs may not be competitive at all. That is, given some value c, a finite length input can be found such that the algorithm doesn't satisfy the above definition. Additionally, a lower bound if k-competitive has been shown for any online algorithm. This means that if we are looking at the 3-Servers problem, the best competitive ratio an online algorithm can achieve is 3 [1].

Direct Analysis

Direct analysis is a similar techinque to competitive analysis, in that we directly compare the performance of ALG to the performance of OPT on a request. Rather than looking across all request sequences of any finite length, we determine a specified length for our request sequence. Then,

Max/Max Ratio

Bijective Analysis

3 Algorithm Description

Random Algorithm

Greedy Algorithm

Optimal Algorithm

Work-Function Algorithm

Double Coverage Algorithm

K-Centers Algorithm

4 Software implementation details and API

Write about the software implementation, threading, how the algorithms were implemented, etc.

5 Numerical Experiments

What experiments were conducted, and the results from the experiments...

6 Analysis

6.1 Analysis

Describe the results from the data that was collected, what is interesting, different results...

7 Conclusion

Write something here...

References

[1] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis.* Cambridge University Press, 1998.