Analysis of the Server Problem on the Cycle

R&D Project Final Report

by

Aamod Kore
110050004

Guide: **Prof. Sundar Vishwanathan**

Department of Computer Science and Engineering
Indian Institute of Technology – Bombay
Mumbai 400076, India

May 5, 2014

Abstract

Abstract

Contents

Chapter 1

Introduction and Overview

1.1 The k-server problem

The k-server problem can be stated informally as the problem of moving around k servers, in a metric space or a weighted graph, to service requests that appear online at points in the metric sapce or at nodes of the weighted graph. A formal definition can be as follows.

Let M be a metric space and $d : M \times M \to \mathbb{R}$ be the distance function such that d is non-negative, symmetric and follows the triangle inequality. For simplicity, we allow distinct points in M to be at zero distance. Thus M is a *pseudometric* rather than a metric. We call an ordered set of k points in M to be a configuration. Let M^k denote the set of all possible configurations in M.

We extend the notion of d from M to M^k. Let $\mathsf{d} : M^k \times M^k \to \mathbb{R}$ denote the distance between configurations in M. For $C_1, C_2 \in M^k$, $\mathsf{d}(C_1, C_2)$ is the value of the minimum-weight perfect matching between the points of C_1 and C_2. In short, it is the minimum distance travelled by k servers to change their configuration from C_1 to C_2.

The server problem $\mathsf{S} = (k, \mathsf{M}, C_0, \mathsf{r})$ is defined by the number of servers k, a metric $\mathsf{M} = (M, d)$, an initial configuration $C_0 \in M^k$ and a sequence of requests $\mathsf{r} = (r_1, \ldots, r_m)$ where each $r_i \in M$ is a point in the metric. The solution is given by a seqence of configurations $(C_1, \ldots, C_m)$, where each $C_i \in M^k$, such that for all $t = 1, \ldots, m$, $r_t \in C_t$. The objective is to minimize the cost of the solution, given by $\sum_{t=1}^{m} \mathsf{d}(C_{t-1}, C_t)$, which is the total distance travelled by the servers.

An ***online algorithm*** computes each configuration C_t based on only the past, that is only on $r_1, \ldots, r_t$ and $C_1, \ldots, C_{t-1}$. An ***offline algorithm*** may also use knowledge of future requests $r_{t+1}, \ldots, r_m$. Thus an offline algorithm knows the entire request sequence before computing the solution.

1.1.1 Competitive Ratio

For a given initial configuration C_0 and a sequence of requests $\mathsf{r} = (r_1, \ldots, r_m)$, let $\mathsf{COST}_A(C_0, \mathsf{r})$ denote the cost of an algorithm A and let $\mathcal{O}(C_0, \mathsf{r})$ denote the cost of an optimal solution. Then we say that algorithm A has ***competitive ratio*** ρ if for every C_0 and r,

$$\mathsf{COST}_A(C_0, \mathsf{r}) \leq \rho \cdot \mathcal{O}(C_0, \mathsf{r}) + \Phi(C_0),$$

where the term $\Phi(C_0)$ depends only on the initial configuration C_0 and is independent of the request sequence r.

The competitive ratio of an alorithm can also be called its approximation ratio. If an algorithm has competitive ratio ρ, it is said to be ρ-competitive.

Rather unexpectedly, it is likely that the competitve ratio of an algorithm is independent of the metric space, provided it has more than k distinct points. This is given by the k-server conjecture [11].

Conjecture 1 (k-server conjecture) *For every metric space with more than k distinct points, the competitive ratio of the k-server problem is exactly k.*

The k-server conjecture is supported by all results on the problem since it was first stated. The conjecture has neither been proved nor disproved and is open, but has been proved for $k = 2$ and some special metric spaces.

We specially consider the metric space of the cycle.

1.1.2 The Cycle Metric Space

The cycle metric space on p points $\mathcal{C}_p$ is an undirected cyclic graph that consists of p nodes connected cyclically. Thus formally, consider the undirected graph $G = (V, E)$ where $V = \{a_1, \ldots, a_p\}$ and for $j = 1, \ldots, p-1$, $\{a_j, a_{j-1}\} \in E$, and $\{a_p, a_1\} \in E$. Let the distance function $d : V \times V \to \mathbb{N}$

be such that, $d(a,a) = 0$, for all $a \in V$, and $d(a,b)$ denotes the length of the shortest path between the nodes a and b, for $a, b \in V$ and $a \neq b$.

The metric space denoted by $\mathcal{C}_p = (V, d)$ is the cycle metric space on p points.

For the cycle, given a point a, we denote the point furthest away from a (the diametrically opposite point) as $\overline{a}$.

$$\overline{a} = \arg\max_{x \in V} d(a, x)$$

In this report, we mainly focus on finite cycles. It has been proven for all metric spaces that the competitive ratio for $k = 2$ servers is 2 [11]. We take a look at the case for $k = 3$ servers.

1.2 Related Work

The server problem was first defined by Manasses, McGeogh and Sleator [11] in 1988. It was a special case of the online metrical task systems problem stated by Borodin et al. [3, 4] earlier. Manasse et al. showed few important results – They showed that no online algorithm can have competitive ratio less than k, as long as the metric space has more than k distinct points. Further, they showed that the competitive ratio is exactly 2 for for the special case of $k = 2$ and that it is exactly k for all metric spaces with $k + 1$ points. With this evidence, they posed the k-server conjecture (Conjecture 1).

Computer experiments on small metric spaces verified the conjecture for $k = 3$. The conjecture was shown to hold for the line (1-dimensional Euclidean space) [5] and for tree metric spaces [6]. An optimal algorithm for the offline problem was also established [5]. In 1994, a dramatic improvement was shown by Koutaoupias et al. [10] which established the work function algorithm and showed that it has competitive ratio $2k - 1$. This remains the best known bound [9] and there has been limited progress on the server problem. In a recent survey [9], Koutsoupias analyses some major results about the problem, specially concerning the 1-dimensional Euclidian metric, tree metrics and metric spaces with $k + 1$ points.

Two special cases of the server problem we are interested in are the 3-server problem [7, 2] and the k-server problem on a cycle [8]. Any progress on these problems may lead to new paths to attack the k-server conjecture. For both these cases, nothing better than the $2k - 1$ bound is known.

Chapter 2

Algorithms for the Server Problem

2.1 Optimal Algorithm

We take a look at at on offline strategy to find a solution to the k-server problem, provided the entire request sequence is given in advance. The dynamic programming algorithm of Manasse et al. [11] is especially suited for cases where the number of requests dramatically exceeds the number of points in the metric. It has $O(nm\binom{m}{k})$ running time and at least $O(\binom{m}{k})$ space usage for n requests and m points in the metric. Later in 1991, Chrobak et al. gave an $O(kn^2)$ algorithm for the k-server problem to serve a sequence of n requests known in advance on a metric [5].

The algorithm, given by Chrobak et al., reduces the k-server problem to a minimum-cost maximum flow problem in a directed acyclic graph (DAG). If there are k servers $s_1, \ldots, s_k$ and a sequence of n requests $r_1, r_2 \ldots, r_n$, we create a DAG as follows –

- The vertex set is $V = \{s, s_1, \ldots, s_k, r_1, r_1, r'_1, r_2, r'_2, \ldots, r_n, r'_n, t\}$
- The node s is the source and t is the sink
- For every $i \in \{1, \ldots, k\}$, there is an edge of cost 0 from s to s_i and an edge of cost 0 from s_i to t
- For every $j \in \{1, \ldots, n\}$, there is an edge of cost 0 from r'_j to t

- For each pair $(i, j) \in \{1, \ldots, k\} \times \{1, \ldots, n\}$, there is an edge from s_i to r_j of cost equal to the distance between the location of the ith server in the initial configuration and the location of the jth request

- For every $i < j$ there is an endge from r'_i to r_j of cost equal to the distance between the ith and the jth requests

- For every $i \in \{1, \ldots, n\}$, there is an edge from r_i to r'_i of cost $-K$, where K is an extremely large real number.

It is shown [5] that the maximum flow in this graph is k and a maximum flow with minimum cost can be found in $O(kn^2)$ time. Further, the flow can be decomposed into k edge-disjoint $s \to t$ paths, the ith path passing through s_i. In the optimal solution for the problem, the ith server will serve exactly those requestscontained in the $s \to t$ path passing through s_i.

2.2 Work Function Algorithm

In 1994, Koutsoupias et al. [10] established a natural online algorithm based on the dynamic programming approach, called the ***work function algorithm*** (WFA). For metric M, an initial configuration C_0 and a sequence of requests $\mathsf{r} = (r_1, \ldots, r_t)$, we define for every configuration $X \in M^k$, the function $w(C_0, (r_1, \ldots, r_t), X)$ to be the cost of the optimal solution which starts at the configuration C_0, passes through (serves the requests) $r_1, \ldots, r_t$ (in that order) and ends at the configuration X. Therefore,

$$w(C_0, (r_1, \ldots, r_t), X) = \min\{\sum_{i=1}^{t+1} \mathsf{d}(C_{i-1}, C_i) \mid C_i \in M^k \wedge r_i \in C_i \wedge C_{t+1} = X\}$$

For a given server problem, we have a fixed C_0 and fixed $\mathsf{r} = (r_1, \ldots, r_n)$ and w becomes a real function of M^k. Such a function is called the *work function* and is denoted as $w_{C_0,(r_1,\ldots,r_t)}$ or simply as w_t, since C_0 and $(r_1, \ldots, r_t)$ are fixed. The value of the work function $w_t(X) = w(C_0, (r_1, \ldots, r_t), X)$ can be computed in a dynamic programming approach as,

$$w_i(X) = \min_{Z \in M^k, r_i \in Z} \{w_{i-1}(Z) + \mathsf{d}(Z, X)\}$$

using the base values $w_0(X) = \mathsf{d}(C_0, X)$.

To service a request r_t, with the current configuration C_{t-1}, the work function algorithm moves to the configuration C_t such that $r_t \in C_t$ and such that the quantity $w_t(C - t) + \mathsf{d}(C_{t-1}, C_t)$ is minimized.

It has been proven [10, 9] that the work function algorithm for the 2-server problem has competitive ratio 2, and that the work function algorithm for the general k-server problem has competitive ratio of at most $2k - 1$ [9].

Chapter 3

Tests and Results

3.1 Implementation of Algorithms

The implementation of the algorithms and test cases is done in Python 3.2.3.

3.1.1 Optimal Algorithm

We implemented the optimal offline algorithm (Section 2.1). For this we had to implement an algorithm to evaluate the minimum cost maximum flow of an directed acyclic network. For the minimum cost maximum flow problem, we use minimum cost augmentation [12] using successive shortest paths [1] based on the cost of edges. The flow along the minimum cost path is successively augmented to the flow of the network and the residual network is again iteratively augmented, until no path from the source to the sink remains.

Since edges may have negative costs, we cannot use Dijkstra's algorithm. We need to find shortest path in a network where edges have negative weights, which calls for use of the Bellman-Ford algorithm. However, Bellman-Ford algorithm would take worst case $O((k+n)^2 n)$ time complexity, for k servers and a sequence of n requests. Instead we establish node potentials and reduced costs for edges using Bellman-Ford algorithm. Since reduced costs are non-negative and do not change shortest paths between nodes, we use Dijkstra's algorithm for finding shortest path. After augmenting the flow along the found path, the node potentials and reduced costs need to be updated.

`NetworkFlow.py` contains the implementation for the minimum cost maximum flow algorithm. The `FlowNetwork` class creates the directed acyclic network. Functions `add_vertex` and `add_edge` help build the required network. The function `max_flow` and `min_cost_max_flow` find the maximum flow and the maximum flow with minimum cost respectively.

Then, we create a directed acyclic network based on the number of servers and request sequence and compute the optimal service strategy based on the maximum flow with minimum cost of the created network. The `ServerSpace` class is created for these functionalities. It takes input the initial configuration of the servers and the distance function of the metric. A sequence of requests can then be added and the function `process_requests` processes these requests and produces the output strategy.

3.1.2 Work Function Algorithm

The work function algorithm [10] (desribed in Section 2.2) is implemented in `WorkFunction.py`. We use the dynamic programming approach to find the appropriate server to service the oncoming request. The already computed work function values are stored in the table `stored`, which is a dictionary type data structure. The dictionary data structure ensures fast $O(1)$ access to associations and also makes sure that rows in the table and elements in the rows are created only when needed. Thus, space in memory is occupied by only values of the function that are computed (i.e. there are no empty cells), giving efficient usage of memory.

While an object of the `WorkFunction` class is created, it takes as input the distance function of the metric space and the initial configuration C_0 of the servers. Further, `add_request` and `delete_request` can be used to add or remove requests to or from the end of the request sequence. The function `value` returns the value of the work function given the appropriate arguments. `process_request` takes an integer i and processes the request at the ith position in the sequence based on the work function algorithm and produces the server that serves the request and updates the configuration of the system to after that of the request has been served.

3.2 The Cycle Metric Space

For all tests we considered a cycle with small number of points (mostly $p = 20$ or $p = 36$). Implementing the cycle as a metric space was simple. For the cycle with p points, each point was taken as an integer from 0 to $p - 1$. The distance between two points in the metric is calculated based on their difference modulo p. If the difference (modulo p) is greater than $p/2$ we take the shorter distance along the other direction.

Code 3.1: Code for distance function of a cycle with p points

```
def cycle_metric(a,b) :
        d = (b-a)%p
        if d > p//2 :
                return (p-d)
        else :
                return d
```

3.3 Generating Requests

The request sequence to be generated for the problem holds an important place. A good strategy to produce subsequent requests could have potential ways to attack the general k-server conjecture.

3.4 Evaluating Performance

3.5 Results

Chapter 4

Conclusions

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
1	250	259	205
2	269	278	212
3	250	259	205
4	250	259	205
5	254	261	203
6	259	270	214
7	247	259	203
8	247	259	203
9	260	270	214
10	254	261	203
11	247	262	203
12	260	272	214
13	250	264	209
14	263	281	213
15	255	264	206
16	266	277	214
17	284	293	238
18	285	293	238
19	284	294	241
20	261	270	216

Table 4.1: Performance for $p = 20$ and $n = 100$

Case No.	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{T}$
21	520	531	464
22	498	511	436
23	530	542	469
24	520	530	442
25	529	539	468
26	518	533	464
27	502	509	436
28	514	536	436
29	529	542	462
30	495	505	433
31	510	522	440
32	527	539	466
33	531	542	467
34	522	535	460
35	518	526	442
36	530	539	467
37	505	518	436
38	522	543	441
39	550	564	492
40	535	543	468

Table 4.2: Performance for $p = 20$ and $n = 200$

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

[2] Wolfgang W. Bein, Marek Chrobak, and Lawrence L. Larmore. The 3-server problem in the plane. *Theor. Comput. Sci.*, 289(1):335–354, 2002.

[3] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal online algorithm for metrical task systems. In *STOC*, pages 373–382, 1987.

[4] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[5] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

[6] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

[7] Marek Chrobak and Lawrence L. Larmore. Generosity helps or an 11-competitive algorithm for three servers. *J. Algorithms*, 16(2):234–263, 1994.

[8] Amos Fiat, Yuval Rabani, Yiftach Ravid, and Baruch Schieber. A deterministic $\mathcal{O}(k^3)$-competitive algorithm for the circle. 1991. Manuscript.

[9] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[10] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. In *FOCS*, pages 394–400, 1994.

[11] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333, 1988.

[12] Robert Endre Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.