**SC4052 <Cloud Computing> Assignment #1 Report**

N2304863G LEE HEECHAN

**1. Introduction**

The Transmission Control Protocol plays a vital role in ensuring data gets from one point to another reliably. Congestion control mechanism is also important to run TCP system, which helps prevent network congestion. One of the key algorithms is Additive Increase Multiplicative Decrease. AIMD is a simple and powerful algorithm that controls data transmission rates based on parameters and window sizes. In this report, we'll explore how changing certain parameters—like the α and β constants—within the AIMD algorithm and test various calculation methods. As a result, we’ll check the fairness, performance, and whether it converges to results similar to the theoretical values.

**2. Methodology**

I have used Python 3.11, Numpy 1.26.4, Matplotlib 3.8.2 and tqdm 4.66.2.

**3. Trials**

3-1. Standard

*Iteration: 1000 times*

*Capacity: 100*

*α: 1, β: 0.5*

*Initial x1: 1, x2: 5*

Basic standard condition is performed to ensure that the code worked properly. It quickly converged to the fairness line. Since the constant values are the same, it converges to the theoretical A value, the eigenvalue [1, 1].

텍스트, 라인, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

Figure 1. Result of 3-1

3-2. Alpha Variation

*Iteration: 1000 times*

*Capacity: 100*

*α1: 2, α2: 1, β: 0.5*

*Initial x1: 1, x2: 10*

For checking that values of constants affect the fairness line and theoretical value of eigenvalue, the alpha value of x1 is set as twice of X2, and accordingly, the theoretical eigenvalue becomes . The actual experimental result is approaching that eigenvalue like the graph.

텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

Figure 2. Result of 3-2

3-3. Highspeed TCP-like

*Iteration: 1000 times*

*Capacity: 100*

*Initial x1: 1, x2: 10*

The important point of Highspeed TCP is to adjust values in proportion to window size. The larger the window size, the more it increases and the more it reduces when it goes over capacity. Therefore, it's not the same with the paper, but I made it adjust the value in proportion to the window size. It can be checked in the code.

We can see that it converges to fairness line much faster than standard case.

텍스트, 라인, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명

Figure 3. Result of 3-3

3-4. With great power comes great responsibility

*Iteration: 1000 times*

*Capacity: 100*

*α: 1, β: 0.5*

*Initial x1: 1, x2: 50*

It is implemented to be reduced the window size of user with larger size in half when it exceeds the capacity.

In result, it doesn't converge, but it doesn't go far significantly from the fairness line.

라인, 텍스트, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명

Figure 4. Result of 3-4

3-5. Adjusting constants by network performance

*Iteration: 1000 times*

*Capacity: 100*

*Initial α: 1, β: 0.5*

*Initial x1: 1, x2: 10*

The parameter values are adjusted dynamically depending on the network performance. It might seem like Highspeed TCP, but it adjusted the value in proportion to the current remaining capacity, not the window size. Network performance is defined as .

It is fast to converge as Highspeed TCP, but it is more stable around the capacity limitation.

텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

Figure 5. Result of 3-5

3-6. Time-based fairness

*Initial Remain Data x1, x2: Varied for each trials*

*Capacity: 100*

*Initial* *α: 1, β: 0.5*

*Initial x1: 1, x2: 10*

Based on 3-5’s algorithm, now our perspective is changed in a ‘time fairness’. It means that previous methods are based on ‘resource fairness’ to try to provide same window sizes, but in this trial the system try to give more window size where more data transmission is needed. To compare this, the code of 3-5 was modified to create a 3-6-1 code, and in 3-6-2 it was decided to distribute the gradient of alpha in proportion to the amount of data remaining for transmission. It considered the packet drop to precisely check the effect of each algorithm. Also, when one of the tasks was finished, the other had to use all of the capacity.

**Table 1**. *Iteration times by the amount of data from x1, x2.*

| Initial x1 data | Initial x2 data | A: x1 end  (iteration times) | A: x2 end | B: x1 end | B: x2 end |
| --- | --- | --- | --- | --- | --- |
| 2000 | 1000 | 35 | 24 | 34 | 33 |
| 5000 | 2000 | 79 | 48 | 77 | 78 |
| 10000 | 3000 | 143 | 72 | 143 | 145 |

A is 3-6-1, and B is 3-6-2. Since the total amount and capacity are determined, it can be seen that there is no significant difference in the time it takes. It was good that the code moved as intended, but it didn't seem to help with any real improvements.

3-7. Reinforcement Learning

*Capacity: 100*

*α: 1*

*Initial x1: 30, x2: 45*

For one of the machine learning methods, reinforcement learning is applied because congestion control consists of series of action decisions whether to increase or decrease window sizes. Actions are simply defined, increasing or decreasing each window size by 1. If the window size is negative or the sum of window sizes exceed the capacity, the penalty is given. To encourage the system to find bigger throughput and more fair point, reward is defined as . By Deep Q Network (DQN) method, 100,000 times of episodes is investigated in training phase. Then testing start with same initial window size and traverse the coordinates on Q table.

Based on this reinforcement learning algorithm, the algorithm immediately finds the fairness point and grow the total throughput. This shape of the graph is distinctive to other trials. Since each attempt is newly trained, different results come out in each attempt, as Figure 6.

라인, 도표, 텍스트, 그래프이(가) 표시된 사진

자동 생성된 설명 라인, 도표, 그래프, 텍스트이(가) 표시된 사진

자동 생성된 설명

Figure 6. Results of 3-7 (Reinforcement learning show varied result for every trial)

**4. Discussion**

When thinking about TCP Congestion Control, one thing to keep in mind is 'fairness'. In the case of Highspeed TCP as well as basic AIMD systems, it is an algorithm to achieve resource fairness faster. In Trials 3-1 and 3-3, we can see the difference between the general TCP and Highspeed TCP algorithms. The significant difference can be felt in the graph. Highspeed TCP approached the Fairness line around 8 times, while 3-1 approached the Fairness line after more than 80 iterations. This is a big difference.

Trial 3-5 is a trial to add modifications in Highspeed TCP. In 3-5, the alpha and beta values are determined by the margin remaining in the capacity of the current system. This new system can be considered as the opposite of Highspeed TCP. The larger the window size, the greater the value for Highspeed TCP, but in 3-5, the smaller the window size (= the larger the system capacity), so the larger the adjusting values are. The time to access the fairness line is similar to Highspeed TCP, but it can be seen that the change in window size on the graph is quite stable. In the graph of Highspeed TCP(3-3), points are spread throughout the fairness line, but in the case of 3-5, points are located within a limited area. It is expected that packet loss due to capacity shortage will be less than that of Highspeed TCP. This tuning is meaningful, and it is expected that more stable algorithms could be obtained by adjusting parameters.

Until now, for resource fairness, each user got the same alpha and beta values. However, 3-2, 3-4, and 3-6 are algorithms that apply different policies for each user by environment. 3-2 simply doubled the alpha value of x1 and shows a result close to the theoretical eigenvector. In the case of 3-4, only the window size of the user with larger window size is reduced when the capacity is exceeded. This did not converge, but it 'resonates' around the fairness line.

Trial 3-6 is a trial to match the processing time of two users similarly. In this attempt, the window size is changed based on how much data remains for each user. This is a data center, not a normal network, so we can try it. This is because it is easy to obtain information about the state of user or server in the same data center, and it is easy to modify. As a result of the experiment, the capacity and the total amount of data to process were the same, so the time it took was similar in the end. However, this is because when one side ends first, it is implemented so that all resources are available to the other side. If the system had to process additional requests after one completion as real data center, the original method (3-6-1) takes longer. (If it is implemented to start to process a new request even when one side is completed, 49 iterations are required to process when the amount of data at x1 is 2000. It is a big difference compared to 35 times.) The 3-6-2 method will help roughly match the processing time of the requests in the case of processing a big problem into several pieces, such as MapReduce.

In Trial 3-7, machine learning was used to find the optimal window size. Reinforcement learning was used, and it was not easy to set a reward calculation formula with overall throughput, fairness between two users, and capacity of system. It shows a completely different shape from the graph of the existing AIMD algorithm, and it is moving efficiently to obtain the benefits of fairness and throughput. Unfortunately, the disappointed point was the speed of finding the optimal point. It is quite slower than Highspeed TCP now. It would be better to train with more diverse actions, but it was unable due to my limited knowledge. If I have a little more understanding of ML and there is an environment to train more heavily, it would be able to obtain more effective congestion control algorithm by ML. Since the actual data center processes a lot more users and a lot of data, applying machine learning will be the one of the best solutions to respond to various and complex situations.

The point of this assignment is how to control the window size within the data center environment. Traditional TCP AIMD algorithms are designed to gradually find fair points because servers handle unknown requests from unknown users. However, within a data center where more information is available and modifications can be made, there will be a way to get ideal results faster. Using the data throughput information required for each request (3-6), or AI-based system can quickly find a point that can achieve maximum efficiency while preventing congestion (3-7). Fairness of resources is important, but it is thought that data center can find more efficient ways by considering latency, throughput, and service-specific factors.

**Appendix. Source codes**

Github: <https://github.com/Gabul99/SC4052-Assignment1>