Content Delivery Networks - Gradient Descent Approach for Optimization of the Network Cost and the Cache Hit Ratio

Yiheng Niu, Xiaoyu Wu, Xuehan Yi, Yuecheng Shi, Xiyou Xu
Khoury College of Computer Science, Northeastern University, Vancouver, Canada
e-mail: niu.yih@northeastern.edu

I. INTRODUCTION

A content delivery network (CDN) is a distributed network of servers that can efficiently deliver web content to users[7]. With the deployment of CDN, website visitors are experiencing faster page loading times[7]. Basically, a CDN minimizes latency by storing cached content on edge servers in point-of-presence (POP) locations that are close to end users[7].

Since many technology companies have provided cloud services, the implementation of a CDN is no longer a problem. Therefore, our goal is to minimize the total cost while maintaining an acceptable CDN performance. In this report, we hope to explore the interrelation between the total cost and the Cache Hit Ratio (CHR) by implementing a Machine Learning algorithm that simulates quickly.

Specifically, the CDN cost can be reduced by finding out the optimal time-to-live (TTL), which is the time of the web content stored in local cache nodes. This is because the content demand varies with the change of time and space, which allows the removal of some content that is no longer requested. CHR, the ratio of dividing the total number of cache hits by the sum of the total number of cache hits and the number of cache misses, is expected to increase as TTL increases. Moreover, the brute-force algorithm is too slow to run the simulation. Thus, our problem is to come up with an algorithm that executes the analysis of determining the best TTL much faster.

A. key Insights

In this paper, we are using gradient descent approach for analyzing the best TTL to set to the stored contents on the local cache nodes in Content Delivery Networks. In our model, we will consider the total costs of CDN as well as the cache hit ratio of the content that stored in the cache. We try to minimize the costs and maximize the CHR and choose the best trade-off between costs and CHR. Also, we will design and implement the gradient descent algorithm to reduce the simulation time to get the optimized TTL comparing to the Brute Force algorithm.

B. Contribution

Expecting main contributions are stated as bellow:

- **Approach:** We learn Content Delivery Networks using the traffic model. We also investigate reinforcement learning algorithms like gradient descent.
- **Implementation:** We will implement the gradient descent algorithms to find the optimal trade-off between the network cost(including cache storage cost and request cost) and cache hit ratio. One optimal approach using Brute Force will be set as our standard for evaluating the other approach - a machine learning-based approach, a gradient descent approach. Using reinforcement learning, our gradient descent approach is expected to reduce the execution time to find the optimal TTL for a certain content. At least TTL should be very close to one from Brute Force. Our first goal is to modify the action choosing algorithm based on [1], aiming to have a shorter execution time or result closer to optimal. Some possible modifications include having smaller step values, etc.
- **Evaluation:** We evaluate our algorithms by utilizing CEONS simulator [2].

II. RELATED WORKS

The CDN with the most significant point of presence is usually chosen by the content provider (PoP). Smaller, less expensive CDNs may not offer comparable performance to the larger ones. However, evaluating the number of points of presence is insufficient to assist content providers in picking the best CDN [10].

CDN servers are costly to construct and operate since the server capacity that can be devoted to the distribution of a single media file is restricted, and the supplier and/or edge consumers of this media file incur a non-trivial cost [11]. Based on monitoring data accessible in the Data Repository, we could apply Machine Learning methods to create a predictive model for every tuple area-content-format [12], and such a ML-based model can be used to forecast the evolution of a group of content's relative popularity. Local control loops with advanced monitoring concepts and machine learning techniques enable re-configuration to adjust resources to changing situations [13].

When considering the cost of cache node and the cache hit ratio of the content simultaneously, Machine-Learning seems to be an appropriate method to predict content demand because it is less restrictive for prediction tasks and it allows traffic prediction to better fit non-linear features in the inter-data center. To minimize a cost/loss function, there is a method called Gradient descent that is extensively used in machine learning (ML). Gradient descent (GD) is an iterative first-order optimization process for determining a function's local minimum and maximum [14].

In order to get the best TTL, there are some approaches that use reinforcement learning to converge the solution. There is another value-based learning algorithm used in

these approaches: Q-learning Algorithm. It relies on a 2-dimensional table to record the value in each state given an action. An agent selects the best action based on the value in the table, then updates the value. The agent keeps playing until reaching the terminal state. The algorithm stops iteration until the variance of the table is below a threshold. While Q-learning attempts to predict the reward of a specific action performed in a specific state, Gradient descent does not. In our project, we do not need a q-table so the Gradient descent method appears to be a better fit for the task. In the Q-learning technique, we might assign a random TTL starting value at the start of the simulation and calculate its cost recursively [15]. Then, depending on the value of the award, we try out modifications of TTL value to see the change of cost improvement. To compare with the Gradient descent algorithm, the process of searching the best TTL in Q learning is slower because the Q-learning may have useless probing. Gradient descent could reduce useless probing as long as the learning rate is selected properly, and most of the time it will move in the right direction with appropriate learning rates.

- **Gradient Descent:** Gradient Descent is an optimization algorithm used to find the minimum of a function by iteratively moving in the direction of steepest descent[8]. In machine learning, gradient descent is often used to update the parameters of the model[8]. The Content Delivery Networks paper has shown us the graph of cost analysis[1]. Our goal is to move from the top to the bottom point, which gives us the TTL that has the lowest cost. Starting at the top of the curve, we take the first step in the direction specified by the positive gradient. Then we recalculate the gradient and take another step in the direction it specifies. We repeated such a process iteratively until we reached the minimum point in the graph[8].
- **Learning Rate:** The size of the steps is called the learning rate. There is a trade-off in choosing the learning rate. With a high learning rate, we are able to cover more distance each step, but we risk overshooting the lowest point since the slope of the curve is continuously changing[8]. With a low learning rate, the calculation is more precise but it takes more time to get to the minimum[8].

III. CONTENT DELIVERY NETWORK MODEL

In the content delivery network model we set up, the main server sends web content parts to the local servers located in various regions in order to accelerate edge users on requesting the contents, and minimize overall costs consists of storage costs and requests costs (see equation 1). This model allows us to record the total cost of CDN as well as the cache hit ratio of the stored web contents on the local cache nodes. The size and design of the network is based on Akamai network[9]. As shown in Figure 1, the model contains a main server and many edge users in different regions. Each region has its local cache node that stores the cached version of web content parts. When users request web contents, local cached version content will be checked. If it exists, content will be sent from local cache node otherwise content will be sent from main server to local cache node.

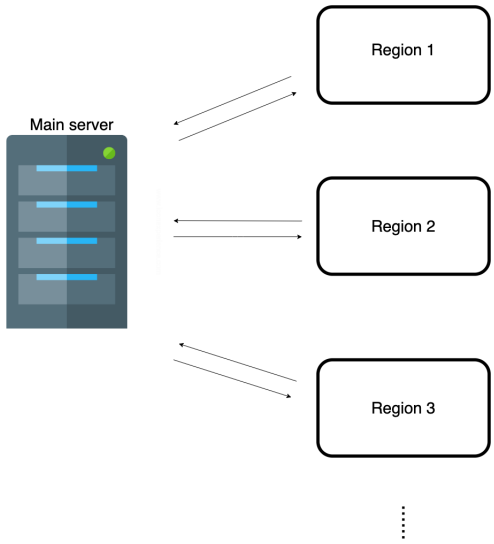


Fig. 1. Main Server and Regions

For us to abstract useful and relevant information from monitoring the data, KPIs (Key Performance Indicators) to collect is identified as follow:

- Cache Hit Ratio (CHR)
- Cost of Cache Node (in $)

Contents are characterized using two key values: size and popularity. Size determines storage and request cost of the content, and popularity determines the probability of content to be requested. Each instance of content is split into several content parts, emulating different episodes of a TV show or movies with different language settings. These content parts have different sizes and can be stored in the local cache nodes and requested by users. Whenever content parts were requested by users, information about which content part was requested and from which region was kept.

In the implemented model, all requests go through the local cache nodes. The cache nodes can only request contents from the main server. The main server has the ability to delete or update the content on cache nodes. The cost of each cache node is different based on different regions in the network and different number of stored content parts on the local storage.

The cache node in each region stores the content parts and has the ability to record the number of requests for these content parts. The cache node sets a TTL based on the popularity of the content. When the requested content part is not stored on local cache node, the node will request the content part from main server and store it locally with a certain TTL (as shown in Figure 2).

IV. ALGORITHMS

For the benchmark solution, we use a brute force approach to get the relationship between TTL with cost and CHR. The brute force method iterate all the TTL and get the corresponding cost and CHR in that situation. Having all the iteration situation, it makes comparison and get the optimal solution. This approach can get the best TTL for less cost and

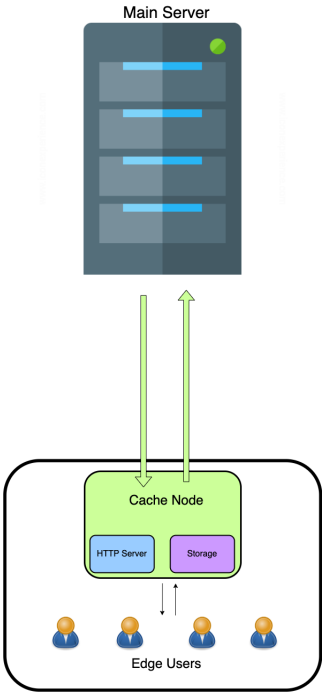


Fig. 2. Cache Node

better CHR, but it is not practical for the real-life networking solutions, as it has a long execution time.

In order to overcome this problem and get a solution which has a practical execution time, we analyzed the relationship between TTL with cost and CHR from the results generated from Brute Force and Q-Learning[6]. The curve indicates the relationship between TTL and CHR is continuous and logarithmic, which rises more slowly as TTL increases(as shown in Figure 3). The curve indicates the relationship between TTL and cost is continuous and only has one optimum minimum(as shown in Figure 4).

As the CHR has little optimization after TTL is large enough, we set up a threshold for CHR. If the cache hit ratio with the currently assigned TTL is less than the minimum acceptable cache hit ratio, we don't consider it as the best solution and keep searching the bigger TTL to get a better CHR while also has a lowest cost. As for searching the TTL for the lowest cost, we use gradient descent to find the optimal solution. Gradient descent is used when the parameters cannot be calculated analytically and must be searched for by an optimization algorithm. In our problem we want to find the only one optimum minimum, where gradient descent works well. So in order to overcome the problem of lengthy simulations, we implement a gradient descent algorithm. We then compare the execution time and the optimality gap.

After iterating every TTL from 1 to 30, a Cost per 100000 requests - TTL pattern that is linear and has one local minimum concave is observed. It is obvious that gradient descent with some refinements could be adopted. The algorithm takes minor steps to explore cost, the step length is calculated by learning rate times derivative in a given point, it will ultimately reach a local minimum, every cost and cache hit ratio are stored and sorted to avoid unnecessary exploration and get the smallest cost with acceptable cache hit ratio(CHR). In case of there is no acceptable CHR among all exploration

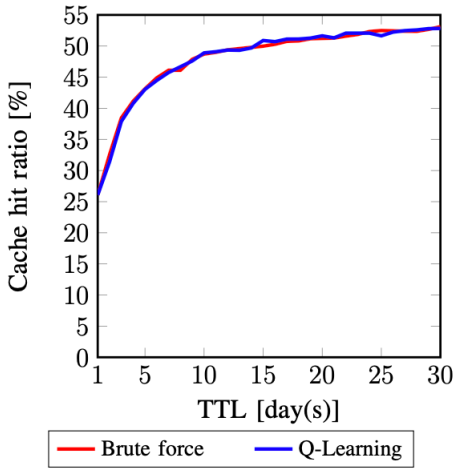


Fig. 3. CHR analysis

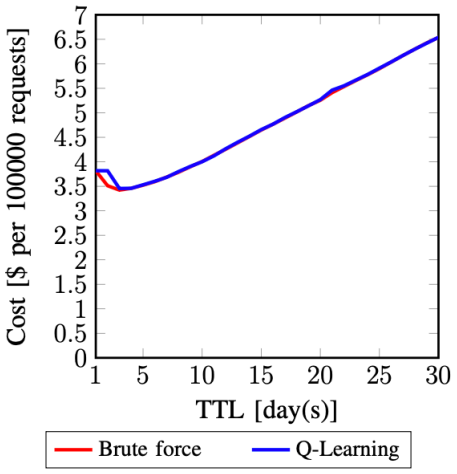


Fig. 4. Cost analysis

TTLs in one iteration, multiple iterations is adopted to make this algorithm general.

In comparison to brute force, gradient descent takes less time since it explores much less TTLs. It might not converge in a global minimum, though, it could reach to a relative low cost TTL in most cases since when a local minimum cost is found, it is more likely to be global according to Cost - TTL pattern in real world, also the high efficiency would convince company to adopt this algorithm.

A. Gradient descent pseudocode

There are some parameters and data structure in the program.

- **Iteration:** iteration time to do for loop to prevent the program from running in infinite loop.
- **Difference:** difference between previous ttl cost and current ttl cost, always positive.
- **Precision:** a criteria set to measure if the difference of cost is small enough between two steps, so that we can stop.
- **Delta:** a little small change based on current ttl in order to calculate the derivative.
- **PriorityQueue:** a priority queue ordered by cost and cache hit ratio.
- **Map:** Used to get pre-calculated cost.

Algorithm 1 findLocalMinimum

currentTTL = *initialTTL*;
minCacheHitRatioAcceptable :
for(i = 0; i < *iteration* and *difference* > *precision* ; i++)
derivative = (calculateCost(*currentTTL* + *delta*) - calculateCost(*currentTTL* - *delta*)) / (2 * *delta*);
previousTTL = *currentTTL* ;
currentTTL = *currentTTL* - *learningRate* * *derivative*;
difference = Math.abs(*currentTTL* - *previousTTL*);
end for
while(*PriorityQueue* is not empty):
ttl , *cost* , *cacheHitRatio* = *PriorityQueue.poll()* ;
if(*cacheHitRatio*>=*minCacheHitRatioAcceptable*)
break;
end if
end while
return -1;

Algorithm 2 calculateCost

if that *ttl* is stored in Map, return the cost of it;
end if
cost , *cacheHitRatio* = simulation(*TTLtoCaculate*);
put *TTLtoCaculate* and its cost into memory map;
put *TTLtoCaculate*, its cost and cache hit ratio into priority queue;
return *cost* ;

To decrease repeated computation, the program stores currentTTL and its cost and cache hit ratio into both a hash map and a priority queue. The map is used for quickly getting pre-calculated cost. A priority queue is ordered by cost first, and then cache hit ratio. It will be used to get a ttl that has a qualified cache hit ratio (higher than minCacheHitRatioAcceptable) in an order from lowest cost to highest. If there is no ttl that has enough cache hit ratio, the program would iterate another time with a new random ttl.

V. SIMULATION SETUP

We simulate a process that client could request some contents from different regions in nearly a year, to lower the length of internet latency, cache is requested first. The program setups the basic components first. Regions are divided by 6 continents with respective time zone offset, different regions have different storing price in Amazon Web Services. One cache node is assigned for each region. We also created 3000 clients which are randomly distributed in 6 regions. 100 web contents are created and each content is split into parts ranging from 10 to 100. The contents are assigned with 4 popularity levels with corresponding possibilities, and each level has following possibilities to be requested:

- Very Popular - 80%
- Popular - 50%
- Regular - 20%
- Obsolete - 5%

Next we assign each client with some contents that he/she is going to request based on the popular level of those contents, request week, day, and hour. In different day in a week and different hour in a day, client has different chance to request contents.

Then the request simulation runs for every hour in every week in 350 days. It first clears out all the content parts that are expired. Then the program will execute request for each client, if the content part is stored in his/her regions's local cache node, we know he/she hits the cache and if not, the cache node would access content from the main server and store the content part in local cache node. Meanwhile, parameters including the request count, cost of storage and sending request to local cache node and main server cache hit count are also updated.

The cost for each cache node is calculated by five components.

$$C_a = C_r + C_c + C_{ser} + C_{st} + C_h \tag{1}$$

where:

C_a - Overall cost;
C_r - Cost of request;
C_c - Cost of transferring contents to client from cache nodes;
C_{ser} - Cost of transferring contents to cache nodes from server;
C_{st} - Cost of storage requests;
C_h - Cost of storage hosting.

VI. RESULT

We execute our gradient descent algorithm against the benchmark brute force algorithm in three simulation scenarios with different simulation run time (50 weeks, 25 weeks, 12 weeks). We collect the KPIs, Cache Hit Ratio (CHR) and Cost of Cache Node, and then compare the TTL result and the execution time for each scenario(as seen in Figure 5, Figure 6 and Figure 7).

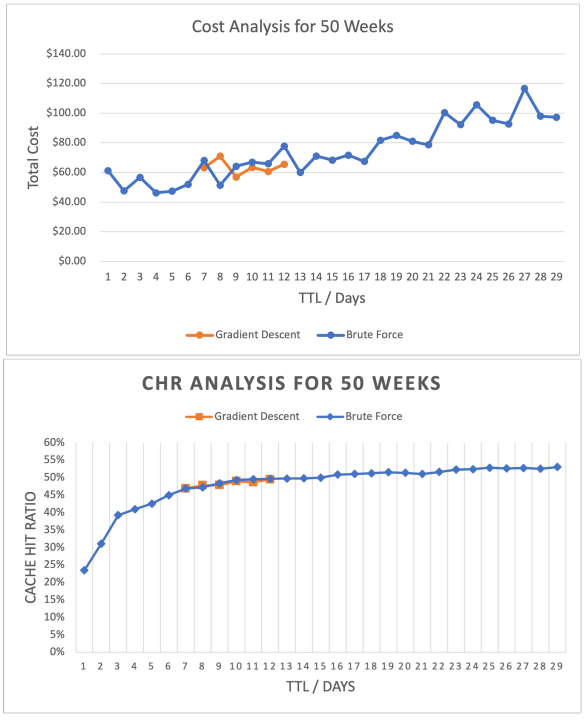
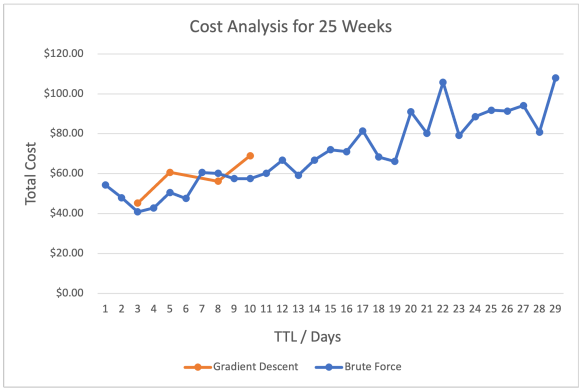


Fig. 5. Simulation Results for 50-weeks duration



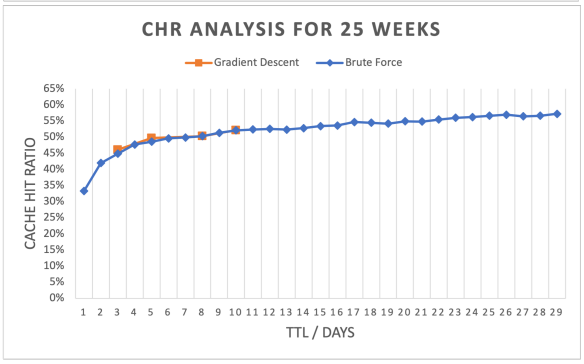
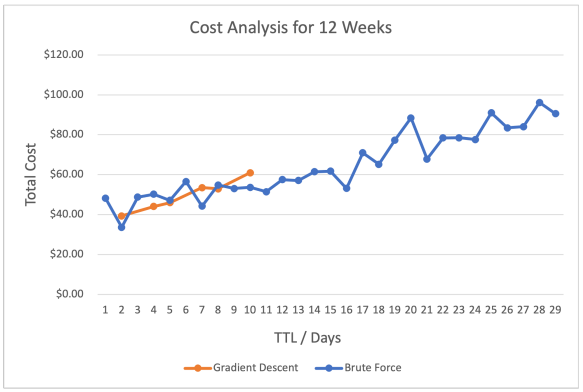


Fig. 6. Simulation Results for 25-weeks duration



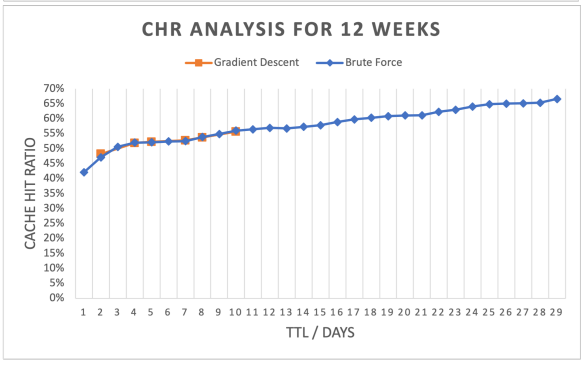


Fig. 7. Simulation Results for 12-weeks duration

Our observation is that the cost drops down initially when the TTL increases at first, and then after a TTL point it rise up and keep growing as TTL increases. As for CHR, with TTL increases, it has a significant grow in the beginning, and then the changes become slower to not that visible. This pattern shows that as TTL increases, we store the content longer in the nodes, which will give us a better CHR. The cost will reduce initially as we don't need to request content from the server, but after a certain TTL, the cost will rise up because of a larger storage cost.

With both the brute force and gradient descent algorithm, we get very similar pattern and result between cost, CHR and TTL in different simulation scenarios. It is worth noticing that gradient descent approach get the result faster as a more effective method. Gradient descent does not need to iterate all the TTL to find the most optimum result, which makes the execution time better.

Table I. Execution time analysis

Algorithm	50 weeks	25 weeks	12 weeks
Brute Force(ms)	358667	277195	81944
Gradient Descent(ms)	75362	55550	22050

Table II. Optimal TTL analysis

Algorithm	50 weeks	25 weeks	12 weeks
Brute Force(days)	8	4	2
Gradient Descent(days)	9	3	2

We selected three time periods to run our simulation, which are 50 weeks, 25 weeks and 12 weeks respectively. Table I recorded the execution time with both brute force method and gradient descent method. It can be seen that gradient descent is faster than brute force under all three time periods. Table II recorded the optimal TTL found with both brute force method and gradient descent algorithm, accordingly. It can be seen that the best TTL found by gradient descent method is pretty close to that found by brute force method, especially with longer time periods.

VII. CONCLUSION

In this paper, we analyze the Content Delivery Networks with the two most basic key performance indicators, Cache Hit Ratio and Cost of Cache Node. Our goal is to find the best TTL to set to the stored contents on the local cache nodes in Content Delivery Networks. For the Content Delivery Networks to be cost-effective, an auto-scaling cache node should be about to determine the best TTL to set to the stored contents where the cost is optimized and the minimum acceptable cache hit ratio is achieved. We designed and implemented the gradient descent algorithm to get the optimized TTL. Compared to brute force algorithm, our approach gets the optimum result, and also has a shorter execution time. The solution presented in this paper, can be applied to different network topologies used by different network operators.

REFERENCES

[1] Felix de Almeida. Diego, Yen. Jason, Aibin. Michal. (2020). Content Delivery Networks

- Q-Learning Approach for Optimization of the Network Cost and the Cache Hit Ratio. 10.1109/CCECE47787.2020.9255813.

[2] M. Aibin and M. Blazejewski, "Complex Elastic Optical Network Simulator (CEONS)," in 17th International Conference on Transparent Optical Networks (ICTON), Budapest, Hungary, 2015, pp. 1–4.

[3] G. Tang, H. Wang, K. Wu, D. Guo, and C. Zhang, "When more may not be better: Toward cost-efficient CDN selection," in INFOCOM 2018 - IEEE Conference on Computer Communications Workshops, 2018.

[4] D. Xu, S. S. Kulkarni, C. Rosenberg, and H. K. Chai, "Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution," Multimedia Systems, 2006.

[5] Y. Bengio, "Learning Deep Architectures for AI," Foundations and Trends® in Machine Learning, vol. 2, no. 1, pp. 1–127, 2009.

[6] L. Velasco, L. Gifre, and M. Ruiz, "Autonomic Content Delivery Network Service," in International Conference on Transparent Optical Networks, no. 1, 2019, pp. 8–11.

[7] Duongau. (n.d.). What is a content delivery network (CDN)? - azure. What is a content delivery network (CDN)? - Azure | Microsoft Docs. Retrieved February 24, 2022, from https://docs.microsoft.com/en-us/azure/cdn/cdn-overview

[8] Gradient descent. Gradient Descent - ML Glossary documentation. (n.d.). Retrieved March 24, 2022, from https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

[9] Akamai, "Media Delivery Network Map," 2019. [Online]. Available: https://www.akamai.com/us/en/resources/visualizing-akamai/media-delivery-map.jsp

[10] G. Tang, H. Wang, K. Wu, D. Guo, and C. Zhang, "When more may not be better: Toward cost-efficient CDN selection," in INFOCOM 2018 - IEEE Conference on Computer Communications Workshops, 2018.

[11] D. Xu, S. S. Kulkarni, C. Rosenberg, and H. K. Chai, "Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution," Multimedia Systems, 2006.

[12] Y. Bengio, "Learning Deep Architectures for AI," Foundations and Trends® in Machine Learning, vol. 2, no. 1, pp. 1–127, 2009.

[13] L. Velasco, L. Gifre, and M. Ruiz, "Autonomic Content Delivery Network Service," in International Conference on Transparent Optical Networks, no. 1, 2019, pp. 8–11.

[14] Kwiatkowski, R. (2021, May 24). Gradient descent algorithm-a deep dive. Medium. Retrieved March 24, 2022, from https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

[15] Almeida, D.F., Yen, J., Aibin, M. (2020). Content Delivery Networks - Q-Learning Approach for Optimization of the Network Cost and the Cache Hit Ratio. 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), 1-5.