1 Abstract

Since the inception of computer science, data retrieval speed was a major bottleneck for numerous applications. The problem got only worse following the introduction of the Internet. To overcome this limitation the concept of caching was introduced. The solution is to store the data which is used the most closer to the location where it is used or store it in a storage with higher access speed. A large number of caching policies have been proposed but most of them require ad-hoc tuning of different parameters and none emerges as a clear winner across different request traces. For this reason, most of the practical caching systems adopt LRU (Least Recently Used) policy because of its simplicity and relatively good performance.

During the internship, we explore the possibility of the application of machine learning algorithms to solve the caching problem. We propose a caching policy which utilizes feedforward neural network and overperforms state of the art policies on both synthetic and real-world request traces. We also examine other attempts of application of machine learning techniques to handle the problem and compare their performance with the approach proposed by us.

2 Introduction

2.1 Caching problem

The invention of the computer allowed scientists to process vast amounts of data faster than ever before. However, soon a significant bottleneck was discovered - data retrieval speed. The introduction of the Internet only increased the influence of this problem. According to Cisco, annual global IP traffic is predicted to reach 3.3 zettabytes by 2021[1]. A massive increase in traffic volume naturally increases the load on the infrastructure. To improve the performance in various applications and to reduce the impact of the traffic growth the concept of caching was introduced. The idea behind this concept is to put the actively used data into storage from which it can be retrieved quicker. The goal is to reduce latency, shorten data access times and improve input/output. Since the workload of most of the applications is highly dependent upon I/O operations, caching positively influences applications performance. Previously described goals can be achieved by using a storage device which is physically closer to the data consumer or which has a higher data access speed. However, most of the time storage capacity is limited in such devices, or the use cost is higher. To maximize the utility of the storage devices various caching policies have been introduced.

2.2 Caching policies

Belady's min algorithm is proven to be optimal[2]. The general idea behind this algorithm is to evict from the cache objects which are requested furthest in the future compared to other objects in the cache. However, this information is not available in the real-time setting thus this algorithm cannot be deployed in a practical system.

First In First Out (FIFO) - one of the first proposed caching policies. Simple to implement and deploy but eventually has been replaced by more

sophisticated algorithms with better performance.

Least Recently Used (LRU) is the natural evolution of FIFO and the most commonly used caching replacement policy. It offers comparably good performance and does not require a lot of extra storage or CPU time.

Least Frequently Used (LFU) in some cases overperforms LRU but requires to track the number of requests for all of the objects observed. This disadvantage limits the number of applications of LFU.

Adaptive Replacement Cache (ARC)[3] is a caching policy introduced by IBM[4] in 2004. It offers better performance than LRU while keeping low computational resources requirements. Considered to be state of the art.

While a large number of caching policies has been introduced, there is still a room for improvement in comparison to the optimal algorithm. Moreover, since, as said before, the amount of web traffic is expected to rise, even a small improvement in caching policy performance could lead to significant cost savings in long-term. To compare the performance of caching policies we are going to use cache hit ratio[5] metric which is the most commonly used and effective metric for cache performance evaluation.

2.3 Neural networks

Following recent successful attempts of application of neural networks[6] for complex task solving[7, 8, 9] a question arises - is it possible to apply Neural Networks to learn close to optimal caching policy online? To tackle this problem, we will try to apply simple feedforward fully connected neural network with a goal to construct a new caching policy which would overperform existing methods. The primary challenge is to overcome the dependence on the future information by estimating it using neural networks.

2.4 Report organization

In the beginning, we will discuss related work in the area.

Then we continue by discussing what data is required to develop and test the proposed caching policy. For ease of development, a controlled and customizable environment is required. Thus we will discuss techniques to generate synthetic data which is good at representing the real world. We will continue by discussing what real-world data is used to test the performance of the proposed policy.

After that, we will discuss in more detail the concept of neural networks, intended use of neural networks for caching, the iterative process of tuning the architecture of the network.

In the last part, we will propose an architecture of a caching policy which is utilizing a neural network in the process of making a caching decision. We will compare the performance of the proposed policy with other approaches including the state of the art approaches.

References

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2016–2021." Available at `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`.

[2] L. A. Belady, "A study of replacement algorithms for virtual storage computers," *IBM Systems Journal*, pp. 78–101, January 1966.

[3] N. Megiddo and D. S. Modha, "Outperforming lru with an adaptive replacement cache algorithm," *Computer*, vol. 37, no. 4, pp. 58–65, April 2004.

[4] IBM. `https://www.ibm.com/`.

[5] B. Davison, "A survey of proxy cache evaluation techniques," *Proceedings of the Fourth International Web Caching Workshop*, pp. 67–77, April 1999.

[6] K. Hornik, M. Stincheombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[7] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-R. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39–48, April 2015.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis,

"Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 28 January 2016.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Multilayer feedforward networks are universal approximators," *Advances in Neural Information Processing Systems*, vol. 27, 2014.