BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

INCREASING EFFECTIVNESS OF CDN NETWORK

ZVÝŠENÍ EFEKTIVITY SÍTĚ CDN

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MARK BARZALI**
AUTOR PRÁCE

SUPERVISOR **Ing. MATĚJ GRÉGR, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

Abstract

This work outlines the Content Delivery Network (CDN) concept and defines the current problems these networks address. Subsequently, the CDN architecture at Seznam.cz is introduced. The study involves obtaining performance metrics of Seznam.cz's CDN, specifically network utilization and cache filling, to gain insights into the current state of the CDN. The results include measurements from proxy servers in Seznam.cz's CDN, which will be further compared with future enhancements. Overall, outcomes reveal the current status of the CDN within the existing architecture and provide valuable insights for potential network optimizations. Then are discussed approaches that could help to increase effectiveness of the cache in company, implementation details and final results.

Abstrakt

Tato práce představuje koncept sítě pro doručování obsahu Content Delivery Network (CDN) a vymezuje současné problémy, kterým tyto sítě čelí. Následně je představena architektura CDN ve společnosti Seznam.cz. Studie zahrnuje získání výkonnostních metrik CDN Seznam.cz, konkrétně využití sítě a zaplňování mezipaměti, pro přehled o současném stavu CDN. Výsledky obsahují měření z proxy serverů v CDN Seznam.cz, která budou dále porovnávána s budoucími vylepšeními. Celkově výsledky odhalují aktuální stav CDN v existující architektuře a poskytují cenné poznatky pro potenciální optimalizace sítě. Dále jsou diskutovány přístupy, které by mohly pomoct zvýšit efektivitu vyrovnávací paměti ve firmě, detaily implementace a konečné výsledky.

Keywords

CDN, vyrovnávací paměť, varnish, CHR, proxy

Klíčová slova

CDN, cache, varnish, CHR, proxy.

Reference

BARZALI, Mark. *Increasing effectivness of CDN network.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Matěj Grégr, Ph.D.

Increasing effectivness of CDN network

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Matěje Grégra Ph.D. Další informace mi poskytli pan Ing. Petr Kuběna. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Mark Barzali

May 9, 2024

Contents

List of Figures

List of abbreviations

CDN	Content Delivery Network. 5, 7, 40
CHR	Cache Hit Rate. 6, 12, 13, 25, 28–30, 34, 36, 37
DASH	Dynamic Adaptive Streaming over HTTP. 19
DDoS	Distributed Denial of Service. 16
HLS	HTTP Live Streaming. 19
IPTV	Internet Protocol Television. 9, 15
LRU	Least Recently Used. 35
RAM	Random-Access Memory. 10, 26
SSL	Secure Sockets Layer. 16
TLS	Transport Layer Security. 16
TTL	Time To Live. 28, 29, 38
URL	Uniform Resource Locator. 8, 10
VCL	Varnish Configuration Language. 11
VM	Virtual Machine. 17
VOD	Video on demand. 9, 25

Chapter 1

Intro

Since the invention of the internet, traffic flowing through worldwide web networks has been steadily rising up. This relentless surge creates load on servers, proxies, switches, routers, and many other network infrastructure components over the network.

Despite the concurrent rise in internet traffic, hardware capacity is also advancing with engineers trying hard to optimize the flow of incoming and outgoing data. One of the most prevalent and globally adopted strategies to prevent overload of servers and provide fast response to requesting clients is the implementation of Content Delivery Networks or CDNs.

The primary objective of this thesis is to provide a comprehensive overview of the current state of the CDN's cache, emphasizing its dedicated role in storing and delivering the media content. By addressing the nuanced caching network issues tailored to Seznam.cz, this work seeks potential optimizations and improvements in the CDN's caching architecture.

The study employs an approach to achieve these objectives, involving the measurement and analysis of relevant metrics gathered from CDN proxy servers. This data forms the basis for a comparative study, aiding in identifying areas for enhancement and optimization within the existing CDN infrastructure in the company Seznam.cz.

In essence, this work strives to contribute to the evolving discourse on CDN architectures by offering insights into the current state of the CDN cache at Seznam.cz and laying the groundwork for future advancements in content delivery optimization.

In chapter 2. an introduction to the world of CDNs is presented. The chapter commences with an overview of CDN, encompassing its various solutions. The CDN ecosystem is also broken down, detailing the roles of origin servers, clients, and providers. The section further explores the communication among CDN components and the types of resources managed by CDN. The examination concludes with a detailed focus on the crucial role of proxy servers, incorporating caching strategies.

The 3. chapter expands on the CDN foundation; this section centers specifically on caching. The concept of Cache Hit Rate (CHR) is introduced, and various caching techniques are explored. This encompasses a comprehensive discussion of inter-cluster and intra-cluster caching strategies, providing insights into the mechanisms that optimize content delivery through effective caching.

The 4. chapter applies insights from earlier sections to Seznam.cz. A concise overview of the platform is presented, elaborating on the role of CDN within Seznam. This involves discussing the content typical for Seznam.cz and an outline of specific requirements for CDN implementation. The technologies employed through Seznam and CDN are explored, covering aspects such as running applications, data storage, video encoding, quality mea-

surement, monitoring, and caching. The evaluation of the current solution's effectiveness and the measurements of the present CDN architecture are integral components of this concluding section.

5. chapter describes possible ways that could have positive impact on cache effectivity and increase CHR. In this chapter is discussed potential changes that could be applied to physical level and changes that could be changed in software approach, including rethinking logical topology of routing user traffic. Next thing that is writen down is way to predict that new changes will bring raise of cache hit rate, there are compared two ways - simulation model and recreation of production-like environment, discussed better approach to use and why. Then chapter continues with results of testing and makes a conclusion is it worth to implement given approach or not.

6 chapter *Real Life Implementation* describes how did new changes influenced production metrics. Shows visualtions of new graphs and discusses why did graphs change. It shows graphs of RAM usage, Cache hit rates, LRU nukes and state of network. Chapters ends with conclusion based on changes in graphs compared with previous ones.

7. chapter „Possible improvements“ brings up topics that could be took in account to increase effectiveness of cache usage, increase CHR or improve CDN at other moments and discusses some potential problems with new topology like sliding window that brings double object cache time in cache storage.

Last 8. chapter *Conclusion* sums up a thesis and brings an overview of whole work done.

Chapter 2

Closer view to CDN

A CDN is designed to prevent server overload caused by enormous data throughput by creating a collaborative network of elements that implement transparent for end-user solutions.

Mr. Pathan and his colleagues have authored a book titled „Content Delivery Networks“ [6]. In this book, they wrote:

The typical functionalities of a CDN include:

- *Request redirection and content delivery services, to direct a request to the closest suitable CDN cache server using mechanisms to bypass congestion, thus overcoming flash crowds [7], or SlashDot [5] effects.*
- *Content outsourcing and distribution services to replicate and cache content from the origin server to distributed Web servers.*
- *Content negotiation services to meet a specific need of each user(or group of users).*
- *Management services, to manage the network components, to handle accounting, and to monitor and report on content usage.*

To wrap it up, CDN primarily aims to create a network service that manages tasks required for faster, reliable content delivery from web servers to clients.

2.0.1 CDN solutions

Currently, we observe two main approaches - to use a ready CDN solution or invest in developing own CDN. Both of them have their pros and cons.

- **CDN as a service**.
  ‘

  One of the biggest and most famous CDN providers is *Cloudflare* . Also companies like Amazon (CloudFront) and Microsoft (Azure CDN) are offering CDN as a service.
- **Building own Content Delivery Network**
  Indeed, while specialized CDN providers possess extensive knowledge and experience in developing CDN networks and offering comprehensive solutions, they may not always be able to cover all specific or unique use cases, requirements, or tailored policies of every company. In such cases, organizations often find it necessary to start developing their own CDN in search of a solution able to cover all their specific requirements.

As I work for a company with its own Content Delivery Network, this work mainly focuses on in-house CDN solution.

2.1 Ecosystem of CDN

An architecture of CDN includes three essential components - *origin server*, *CDN distributor*, and *client*.

2.1.1 Origin server

An origin server is a machine capable of generating responses based on client requests, typically using the Uniform Resource Locator (URL) to determine the content to provide. An instance of the origin server can be another proxy or a backend application that can process and respond to this request.

2.1.2 Client

A client, often called an „end-user," can encompass a person or another machine that submits a valid request for a service.

In this work, the term „client" broadly encompasses human users and automated systems.

2.1.3 CDN provider

A CDN provider is an element in infrastructure(often referred to as a *web accelerator*[1]) that helps to reduce the load of the origin server, increase reliability, provide faster response to the client. This can be achieved by caching and replicating content across multiple CDN providers. Large CDN networks try to locate their servers as closely as possible to end users to ensure faster and more immediate responses to clients.

In the context of this work, the term „CDN provider" may be replaced by a „CDN proxy," „edge server," or „edge proxy."

2.2 Communication of CDN components

Indeed, there is no one-size-fits-all formula for creating and making everything work. The behavior of the CDN network should be configured to align with the specific use cases that a team or organization aims to achieve. However, many patterns and recommendations are available to guide solving assigned tasks.

Many methods are available for configuring an effective Content Delivery Network, ranging from basic load balancing to more advanced solutions like Anycast solution[2]. However, as was mentioned above, fundamental principles of building a CDN have remained relatively consistent since the 2000s. These principles involve a typical flow where a client requests content from an origin server, and the request passes through an edge proxy, which checks if it is possible to serve content requested by the client without burdening the origin server.

[1]Network component that reduces access time to web resources. That component may use caching, prefetching, load-balancing, and many other methods to achieve the aim of web acceleration. See how Nginx Organisation describes Web acceleration

[2]In the context of CDN, Anycast is a method to connect the clients to the geographically nearest data center.

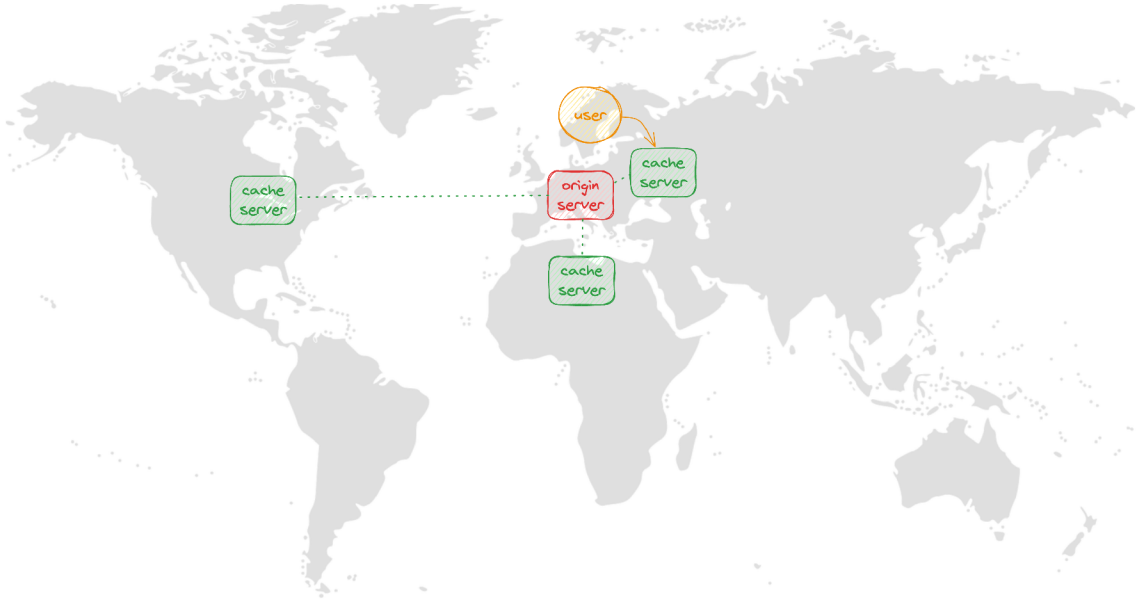


Figure 2.1: CDN ecosystem

If possible, the origin server is spared the processing of this request effort. Otherwise, the request is sent to the server, and the response is routed back through the CDN proxy, which can perform operations with this data, cache it, and finally return it to the client.

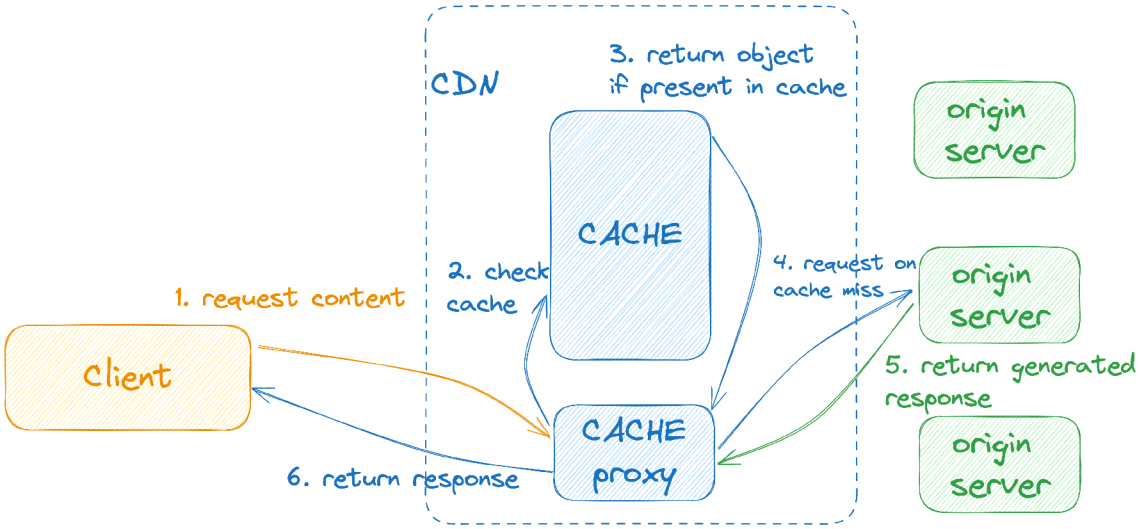


Figure 2.2: Example of a simplified CDN workflow.

2.3 Resources CDN takes care of

CDN is a system capable of handling a wide array of content types. This includes various forms of media content files such as images, pictures, Video on Demand(VOD), any audio files, live streams, Internet Protocol Television(IPTV), as well as text-based content, con-

figuration data, or HTML pages and even specific segments of HTML documents, among many other types.

2.4 Role of proxy

Proxy is an intermediate element between the client and server, which has several responsibilities:

- **Security**. The proxy can be an additional layer of security to prevent potential harmful traffic from going to the server.
- **Filtering traffic**. Filtering is used to block specific URLs.
- **Load balancing** is used in a network with multiple servers that can handle requests to distribute work on servers, ensuring that the workload is evenly balanced and preventing single server overload or handling cases when one of the servers is down and redirecting request to a server that is running.
- **Caching** is storing copies of content delivered to the client in case this content will be requested again by the same user or other users.
- **Access Control**. Proxies can enforce access control policies, allowing or denying access to specific resources based on client credentials, IP addresses, or other criteria.
- **Logging**. Proxies can monitor savings by going through them for further analysis, error detection, or other purposes.

In this section, not all possible uses of a proxy are defined, such as using it to obscure a client's real IP or location, as these aspects are irrelevant for this work.

2.4.1 Caching on proxy

Caching is an essential part of any Content Delivery Network, so it is appropriate to work for the proxy as described above. The way proxies *can* work with caching is described below.

A proxy should be configured to store and manage a cache for caching to be effective. This configuration may include some rules that describe:

- **When to cache content**. It may be based on request URL, specific response header, and, in some cases, on request/response IP, date, and time.
- **For how long this content should be stored on the proxy**. Specifying a time duration for which content should be held on the proxy ensures that content will be regularly fetched again to keep it up-to-date.
- **How to store content**. The proxy can store content in Random-Access Memory (RAM) using different techniques for faster retrieval or store it on a hard disk drive for larger storage capacities. Additionally, options like partial caching can be configured to store only specific parts of content.
- **When content should be cleaned up**. Over time, it may be necessary to purge entire or parts of the proxy's cache to ensure the availability of fresh content.

2.4.2 Proxies specialized in caching

Nowadays, several companies offer proxy server solutions. Most popular are *Nginx*[3], *Apache HTTP Server*[4] and *Varnish*[5].

For this work, Varnish proxy and partly Nginx proxy will be considered.

As Thijs Feryn writes in his book dedicated to Varnish:
Originally, Varnish was a reverse caching proxy: a proxy server that speaks HTTP you put in front of your web servers. Varnish heavily reduces the load and the latency of your web servers.[4]

Varnish has its configuration language called VCL.

The second proxy that will be covered is Nginx, which is defined in the Nginx Cookbook written by Derek DeJonghe:

NGINX is one of the most widely used web servers available today, in part because of its capabilities as a load balancer and reverse proxy server for HTTP and other network protocols [3]

Both solutions are written in the C language, ensuring fast processing of requests and tasks. However, their main difference lies in their primary development focus. Varnish primarily focuses on being an HTTP accelerator, while Nginx mainly aims to be a web server with caching options.

This leads us to a logical conclusion: while these tools may be capable of handling some functions implemented in others, they are designed to excel in distinct areas. Consequently, using these two technologies in various ecosystems is a common practice. This collaborative approach allows organizations to harness the strengths of each tool, creating a more robust and versatile solution.

[3] https://www.nginx.com/
[4] https://httpd.apache.org/
[5] https://varnish-cache.org/

Chapter 3

Closer view to caching

The **cache hit rate** (CHR) is a metric that is calculated by dividing the amount of cache hits[1] by the total amount of request that came to proxy(sum of cache hits and cache misses[2]) as seen in formula 3.1. This metric shows how effective cache proxies are working.

$$\text{CHR} = \frac{N_{\text{Cache Hits}}}{N_{\text{Cache Hits}} + N_{\text{Cache Misses}}} \tag{3.1}$$

This formula calculates the cache hit rate, representing the efficiency of the cache usage. Where $N_{\text{Cache Hits}}$ is the number of requests that was served with cache, $N_{\text{Cache Misses}}$ is the number of requests that was redirected to origin server as cache did not contain a requested object. So $N_{\text{Cache Hits}} + N_{\text{Cache Misses}}$ is the number of total requests that came to poxy.

Most websites with primarily static content can reach high CHR. Websites that serve dynamic content[3] have this ratio usually lower. However, one of the primary targets for CDN engineers is to make this ratio as high as possible.

3.1 Caching techniques

For complex solutions where availability and reliability are crucial, it's common practice to construct a cluster, array, or mesh of caching proxies.

This approach offers the opportunity to replicate cache on different instances of cache proxies. Doing so ensures that if one of the cache proxies becomes unavailable, then other cache proxies could handle request, and provide a cache that was present on the inactive proxy. This redundancy enhances fault tolerance and guarantees uninterrupted access to cached data.

In the book Content Delivery Networks[6], authors define two caching techniques - inter-cluster and intra-cluster.

- Intra-cluster caching

[1] Cache hit - a situation when a request that came to reverse caching proxy was handled by proxy internal cache system and the request was not directed to the original server.

[2] Cache miss - a situation when proxy serving cache does not possess requested resource, and should request is redirected to an origin server that can handle it.

[3] Dynamic web content is a type of content generated in response to the exact incoming requests and varies depending on factors such as time or by the client that created this request.

  - Query-based scheme
  - Digest-based scheme
  - Directory-based scheme
  - Hashing-based scheme
  - Semi-Hashing-based scheme
- Inter-cluster caching
  - Query-based scheme

3.1.1 Intra-cluster caching

This section describes ways of communicating proxy servers that are part of the cluster, described in the work of Buyya Rajkumar, Mukaddim Pathan, and Athena Vakali [6].

The main principle of a *query-based* schema is that if the cluster node does not have the object stored in the cache that the client requested - it sends a broadcast request to all nodes registered in the cluster and waits for its response to decide if content should be served from the origin server or not. So, suppose no node in the cluster contains the appropriate cache entry. In that case, the client may experience significant delays, as we have to wait for the response of the slowest node and, after this, may wait to process the request by the origin server. Furthermore, the act of sending broadcast requests can lead to network flooding, which can have a severe impact on infrastructures with limited bandwidth.

To overcome problem with network flooding and a potentially long time to respond developers tried solution based on storing digest of the content stored on other proxy cache servers. Unfortunately, to keep the digest, there is a need to allocate some memory (that could store cache objects needed to increase the CHR) for this purpose. The second disadvantage is *traffic of update messages* when the proxy caches a new object - it should notify others about its new digest.

Another architecture of the cache cluster is the directory-based scheme. This solution builds upon the previously mentioned idea of storing digests on the server but transitions from a decentralized system to a centralized one. In this approach, all traffic servers sent to each other are routed to a central server, which maintains all mapping data. This approach significantly reduces the volume of packets coming through a cluster network, up to N[4] times! However, there is a trade-off to this optimization. Like any centralized system, this suffers from the unavailability of a central element, the server director. Moreover, the fact that the director must serve all other proxies can exacerbate this vulnerability.

Next method does not depend on a centralized element but needs some significant part of the memory to store metadata or create a traffic flood. This schema is based on hash functions. The prerequisite of this solution is that each server should keep the addresses(IP or domains) of other nodes in the cluster. The main idea is that each server has the same hashing function, and based on the hash of the URL request, it picks from the hash-circle designated server and redirects to it. This approach solves the problems of previous schemes. Unfortunately, this method has a disadvantage too - as we configured cluster hash-circle for nodes, there is no possibility of adding new nodes - the hash-based scheme does not scale.

[4]Where N is several servers

In a semi-hashing-based approach, a local CDN server dedicates a specific portion of its memory space to cache the most famous content for its local users. The remaining portion is allocated to collaborate with other CDN servers through a hashing function.

3.1.2 Inter-cluster caching

Inter-cluster communication is responsible for redirecting requests from one cluster to another.

Inter-cluster caching has only one scheme to offer, as digest or directory-based schemes use big amounts of data to communicate between cluster nodes, and any hash-based scheme is unsuitable for this solution as a representation of CDN servers of different clusters is normally distributed geographically. [6]

Therefore, only a query-based solution is appropriate for inter-cluster communication.

Chapter 4

CDN at Seznam.cz

This work will be dedicated to the Content Delivery Network implemented in the company from the Czech Republic - ***Seznam.cz***[1].

Seznam.cz is a Czech internet portal and search engine. The company was founded in 1996 by *Ivo Lukačovič* and later became one of the Czech Republic's first internet directories and search engines.

The company is headquartered in Prague. Since its inception in 1998, the search engine and business directory gradually expanded to offer more services. By the beginning of 2013, the company operated over 25 different services. As of 2014, Seznam.cz services had over 6.75 million unique monthly visitors on the Czech internet.

Seznam.cz operates three data centers - Osaka, Nagoya, and Kokura.

Seznam.cz is a substantial company that manages many services and delivers significant volumes of content to users daily. Its dedicated CDN department is responsible for developing highly reliable systems capable of handling substantial loads efficiently.

Seznam.cz encompasses its television and television studio, a streaming platform, a news website, a web page with recorded live streams, a marketplace, and a web platform for podcasts and radio. By the end of 2023, Seznam started developing its own IPTV.

Consequently, Seznam possesses a large volume of data that includes media content like audio and video files and images. Additionally, the company hosts services that use static content like JavaScript files.

The Content Delivery Network in Seznam handles all types of content.

4.1 Requirements for CDN in Seznam

CDN makes an effort to achieve the following requirements:

- **Content distribution and streaming**, as a core requirement of any CDN, the network should efficiently distribute web content, including storing content into the network.
- **Caching** serves as the heart of the ecosystem, particularly given the vast volumes of content traversing Seznam's network; Caching plays a pivotal role in alleviating the load on origin servers or proxies responsible for content generation.
- The CDN should be easily to **scale** to handle traffic spikes; the solution should also load balance load between nodes.

[1] https://seznam.cz

- **Security** managing DDoS protection, SSL/TLS encryption, content validation, and preventing unauthorized access to content inside CDN, detect potentially harmful content.
- **Monitoring and analyzing** The CDN should provide real-time and historical analytics to monitor traffic, usage, and performance.
- **High availability**
- **High reliability**
- **Support for video encoding**

4.2 Architecture of Seznam and CDN

In this section, we will delve into the architecture implemented by Seznam and its CDN department.

4.2.1 Reliability and Availability

In the present day, it is essential to put into operation systems that can achieve *zero downtime*[2]. Various tools and techniques exist to achieve this objective, which will be explored in greater detail later in this work.

As previously mentioned, Seznam owns and operates three data centers. All of them are located in the Czech Republic, as the target clients are people who live there. This approach empowers the company to establish high-available and high-reliable production systems in case of any problem in one of the data centers; services would be active, running, and ready to serve users in the other two. Indeed, adopting this way necessitates meticulous planning, configuration, and testing, which, in turn, translates to a more significant investment of *human-hours*[3], the payoff is the achievement of high availability.

High reliability signifies the ability of a system to store and provide data without corruption consistently. This goal can be achieved in many ways, including

- Implementing a **redundancy**, when content will be replicated through several sources (servers, cloud services, etc.), and in the case of lost one, another could offer an exact copy of stored content.
- Creating a **backup** of the content provided. This solution ensures the opportunity to recover lost data from a backup source.
- Making a **data validation**.

... and others.

[2] *Zero downtime*, in the context of software, stands for the state of an application with continuous availability and operation without interruption even.

[3] Human-hour or person-hour is one of the metrics for estimation of average work done by one person

4.2.2 CDN location and data storage

As all three data centers are located relatively geographically close, and are located close to a target auditory, CDN does not prioritize building a solution connecting each user to the nearest data center.

Instead, it primarily focuses on problems of high load and quality of service.

One of the critical factors for successfully deploying applications is the selection of the right technologies and tools that meet the specific requirements of the task. Equally important is the planning and construction of the architectural design.

Seznam widely employs solutions provided by the OpenStack *OpenStack*[4] platform. One of the solutions offered by this platform is *Swift*[5], an object storage system. This storage solution aids the CDN by offering built-in redundancy. Being an object storage system, it simplifies the management of stored objects. *Swift* was designed to work with significant volumes of data and optimized for durability and availability, which makes it a good tool for working in a Content Delivery Network. Communication with object storage like Swift is carried out through its API.

Another storage solution used is *CEPH*[6]. CEPH is a distributed storage system that aims to be scalable and reliable. Unlike Swift, CEPH can be mounted into a VM or container, making it a valuable choice for transferring data between applications over a network without requiring direct API connections.

4.2.3 Monitoring

Monitoring is essential in any infrastructure, serving as an indispensable component. Its primary functions include the early detection of issues and the prevention of catastrophic consequences. Monitoring also allows for analyzing various states of an application, facilitating tasks like performance comparisons between different versions. It aids in planning resource availability and tracking their status while providing alerting mechanisms to notify administrators or developers when the application deviates from its expected and proper functioning.

Nowadays, there are lots of solutions that help set up Monitoring. There is a lot of work on ELK [7] and EFK [8]. Despite the wide usage of these two stacks, Seznam does not use data controllers like *Logstash* or *Fluentd*; instead, it develops applications that can be compatible (and generate good-formatted) with *Kibana* [9] input format.

Kibana is not the only tool used to monitor states of application. The monitoring stack in Seznam comprises several technologies, including *Prometheus*, *Thanos*, and *Grafana*.

- **Prometheus** serves as a tool for collecting metrics from applications, storing this data, and offering flexible searching capabilities for these metrics
- **Thanos** is an open source extension for Prometheus that provides high availability, long-term storage, and global querying of Prometheus data. It was created to address

[4] Read closer about what solutions are implemented and provided by the platform you can read on their website https://www.openstack.org/

[5] https://wiki.openstack.org/wiki/Swift

[6] Read more about CEPH on its web page https://ceph.com/en/

[7] *ElasticSearch Logstash Kibana* technology stack, read more about you can in work of Chen, Lei and Liu, Jian and Xian, Ming and Wang, Huimei [2]

[8] *ElasticSeatch Fluentd Kibana* technology stack, it is mentioned in Zhang, Haiyang, and Zeng, Hao work[8]

[9] Kibana is an open-source project for the visualization of logs.

the challenges of scaling Prometheus for long-term storage and making it more suitable for production use cases.

- **Grafana** is a platform for data visualization. It allows the creation of interactive, customizable dashboards for visualizing data from various sources, including Prometheus and Thanos.

4.2.4 Edge Caching

Caching in Seznam's Content Delivery Network is implemented between the CDN and end clients and within the CDN infrastructure. The cache located closer to the client is referred to as the „Edge Cache," in this work.

The management and optimization of the Edge Cache are critical for Seznam's CDN team. Cache storage is not a unlimited resource, and therefore, effective control and maintenance of used memory for caching are essential. The caching system within Seznam has several key requirements:

1. **Effortless Creation of New Cache Objects:** Seznam's caching system should be able to transparent the generation of new cache objects.
2. **Content Selection for Caching:** Deciding what content to store in the cache is a delicate balance. Seznam's CDN components must make intelligent decisions about what content is most suitable for caching. This involves assessing the popularity and volatility of content and considering factors like user access patterns and geographical distribution.
3. **Cache Maintenance:** Over time, cached objects may become outdated or irrelevant. To keep the caching on the same level of effectiveness, the system should be capable of identifying and removing old cached objects that have expired or no longer match the content available on the origin servers. This process ensures that users consistently receive accurate and updated content.
4. **Cache Invalidation:** In certain situations, there might be a need to invalidate all cache objects before they expire naturally. The caching system should provide mechanisms for proactive cache invalidation, allowing the CDN team to refresh the cache when necessary.

Seznam's CDN caching system leverages these capabilities to enhance content delivery, reduce latency, and improve the overall user experience. Effective cache management and maintenance are crucial to achieving these objectives, making the Edge Cache a vital component in Seznam's content delivery strategy.

In response to the CDN network's specific requirements mentioned earlier, Seznam's development team is following an approach that involves implementing Varnish proxies as the core application of the edge caching layer. This technology, which underpins the CDN's edge caching infrastructure, has been instrumental in enhancing content delivery and optimizing the traffic that comes through the network.

Seznam operates multiple edge proxies that serve as intermediaries for client-server communications. Each of these proxies has a substantial amount of RAM – an impressive 256 gigabytes. A straightforward calculation means approximately three terabytes of RAM are collectively available across the CDN for caching purposes. On the surface, this storage

capacity might seem enormous, but in practice, it is consistently near total capacity for an organization as dynamic and content-rich as Seznam. Even with this substantial amount of cache storage, edge proxies have an average CHR at 83,6% daily as seen in graph 4.1.

One notable aspect of Seznam's edge caching infrastructure is the distribution of these proxies. They form an array of proxies, collectively creating a single caching layer. Significantly, these proxies operate independently, enhancing fault tolerance and load distribution. This distributed approach ensures that cached content is available and responsive if several machines are down.

While the fault tolerance inherent in the approach described above is a notable advantage, it also comes with a significant drawback - substantial data duplication. Imagine a scenario where several hundreds of requests are made for the same object. In this situation, the load-balancing process will attempt to distribute these requests to different nodes. However, the consequence of this strategy is that the same content is cached across all the involved proxies. While offering redundancy and reliability, this practice is not the most efficient way of managing data. It results in a significant waste of memory resources, multiplying the data storage requirements by a factor of N, where N represents the number of proxies engaged in the load-balancing process. Utilizing memory resources while avoiding unnecessary data duplication is a persistent challenge in such a dynamic and content-rich environment.

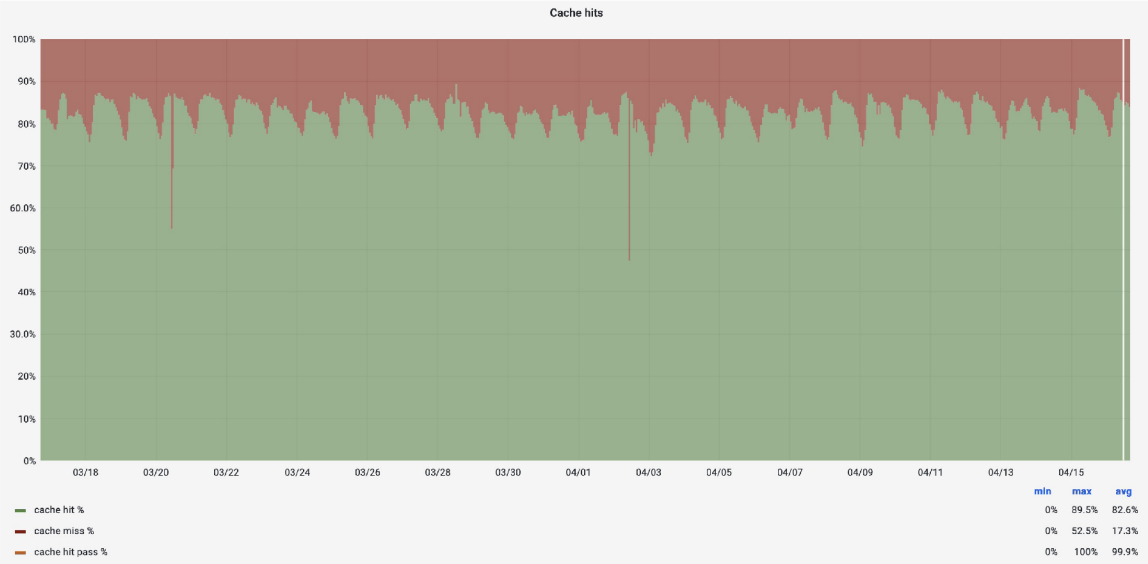


Figure 4.1: Graph displaying 30-day statistics of caching on edge proxies

4.2.5 Internal caching

Seznam's Content Delivery Network (CDN) is an adaptable and miscellaneous system that handles a wide array of tasks, ranging from uploading and storing images to applying filters on videos and generating media playlists for live streaming, including HLS and DASH. These processes are developed in alignment with the fundamental principles of Microservice architecture. As such, incoming tasks to the CDN are precisely classified and delegated to a specific microservice. These microservices can take various forms, such as APIs or running daemons, each designed to perform a particular function.

For the CDN to operate optimally, all microservices should be configured appropriately and keep up-to-date information about the CDN's current state. To ensure this synchronization process, various components within the CDN need regular updates and data exchanges between components. Examples of the types of such traffic that flow into the CDN encompass obtaining configuration data for an encoder, initiating the encoding process, fetching authentication data to determine a user's permissions for specific actions, or retrieving the status of particular tasks.

Given the high volume of tasks that can be executed concurrently within the CDN, an efficient caching system is employed. Unlike the Edge Cache, which primarily serves content closer to clients, this internal CDN cache serves a different purpose. It is relatively small and does not demand ample RAM resources on a machine. Consequently, it can be efficiently implemented as a cache within a Kubernetes Pod's container, often working with a running Nginx server inside. This internal cache enhances the CDN's performance by reducing the need for frequent, resource-intensive data retrieval from other components, ensuring quicker response times and better resource management.

In certain scenarios, there are situations where a component needs to retain data that will be required more than once. Several approaches exist to store this data efficiently for future use in such cases. One of the most prevalent methods employed at Seznam is using the Redis [10] database, which serves as a cache layer for these purposes.

This approach offers the advantage of making cached objects accessible across multiple processes, whether running on the same machine or different machines within the network. Components delegate to the Redis, fulfilling various essential functions, including cache invalidation and, in some instances, mitigating data races associated with a particular object.

The internal caching level of the CDN architecture not only improves the system's efficiency but contributes to its ability to handle large workloads.

[10] Redis, which stands for Remote Dictionary Server, an open-source, in-memory data storage and caching system. https://redis.io

4.3 Effectiveness of current solution

The current state of architecture can be characterized through metrics. In this context, metrics collected from edge proxies with Varnish are stored in Prometheus and are accessible through the Thanos API. Thanos aggregates this data, providing an interface for further analysis and visualization. The aggregated metrics are then rendered in graphical format on Grafana.

4.4 Measurement of present architecture

This section will showcase graphs with metrics generated during a single workday. The graphs provide insights into the performance of the network proxies connected to it.

Cache Hits

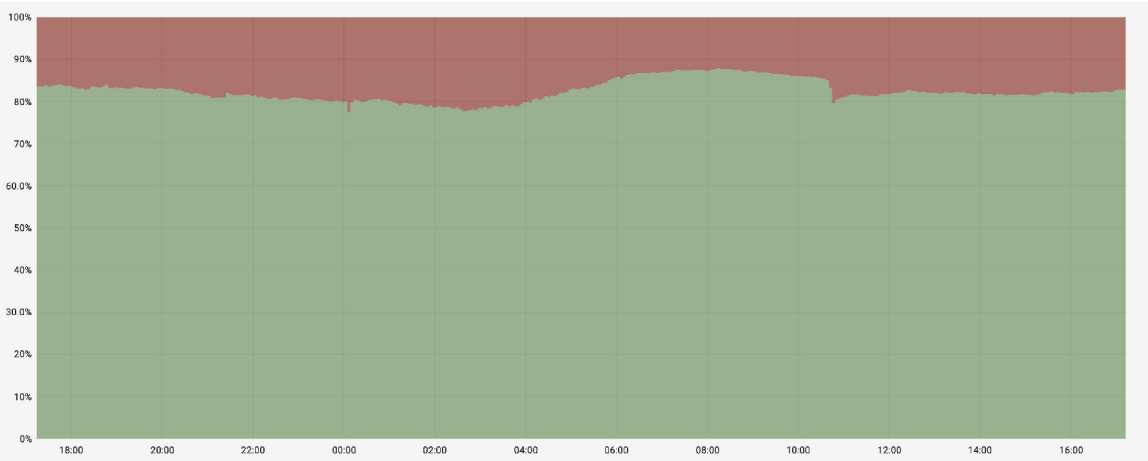


Figure 4.2: Graph displaying one-day statistics of caching hits on proxies

The depicted graph (Figure 4.2) provides an overview of caching hits on proxies over a single day. The horizontal axis represents the timeline, capturing the progression of time throughout the day. The vertical axis displayed the cache hit rate and the percentage of requests successfully served from the cache.

Cache Memory

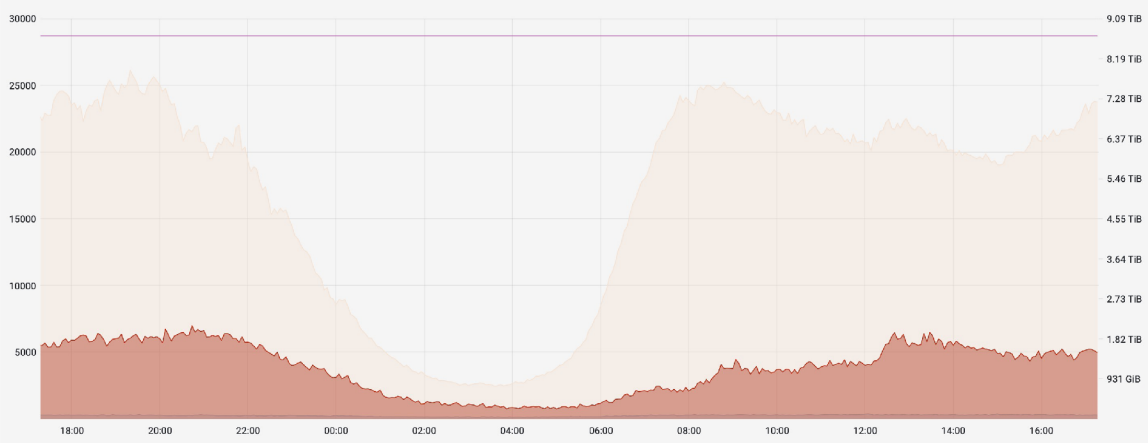


Figure 4.3: Graph displaying one-day statistics of caching Memory on proxies

The figure 4.3 illustrates the one-day statistics of caching Memory on proxies, describes memory management dynamics. The horizontal axis represents the timeline, and the vertical axis quantifies the amount of Memory.

The Graph contains three aspects of caching memory management:

- **LRU Moved (White area):** This area depicts the volume of cache entries subject to the Least Recently Used (LRU) policy and consequently moved within the caching Memory.
- **LRU Nuked (Red area):** The red area signifies instances where cache entries were forcefully evicted from storage or „nuked“ to make room for a new object due to the LRU policy. This could indicate high memory pressure or the need to prioritize more relevant or frequently accessed data.

Metric	Min	Max	Total
LRU Moved	2405.12	26.1k	12032.4k
LRU Nuked	710.38	6.9k	2742.5k

Table 4.1: Cache Memory Statistics

State of Network

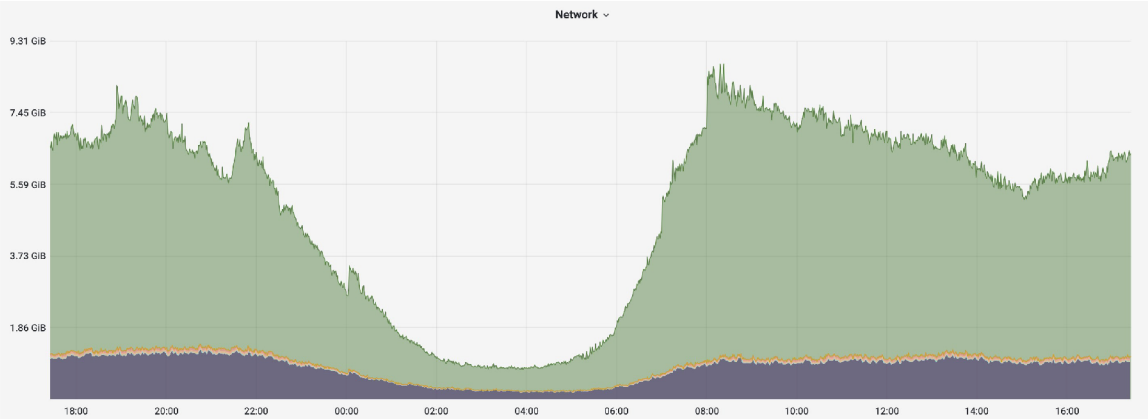


Figure 4.4: Graph displaying one-day statistics of network state

The figure 4.4 represents a network state. The horizontal axis signifies a timeline, delineating the progression of time throughout the day, while the vertical axis quantifies the network metrics.

- **Frontend (Green Area):** The green area illustrates the metrics associated with the frontend.
- **Backend Default (Purple Area):** The purple area represents metrics related to the default backend. Also, other backends are overlapped by the purple zone.

Component	Max	Avg	Total
Frontend	8.29 GiB	4.85 GiB	3.41 TiB
Backend	1.79 GiB	1.08 GiB	778 GiB
Backend Default	1.64 GiB	1.01 GiB	366 GiB
Backend Fallback_backend	6.70 MiB	374 KiB	132 MiB
Backend Filtrilon1	55.9 MiB	34.0 MiB	12.0 GiB
Backend Filtrilon2	58.3 MiB	33.9 MiB	12.0 GiB

Table 4.2: Network Statistics 1 day

Requests

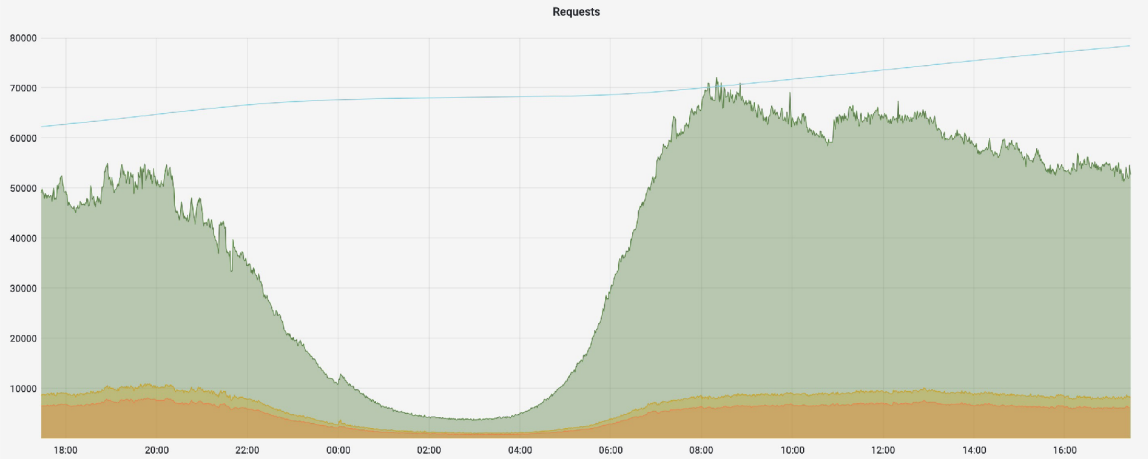


Figure 4.5: Graph displaying one day and showing the amount of requests going through the proxies

The graph (Figure 4.5) illustrates the one-day statistic representing the volume of requests going through the proxies. The horizontal axis is a timeline, and the vertical axis is the number of requests.

- **Frontend (Green Area):** The green area signifies the number of requests from the frontend.
- **Backend (Yellow Area):** The yellow area represents the requests from a backend. The cyan line depicts the cumulative sum of request rates over a 5-minute interval.
- **Fetched Objects (Orange Area):** The orange area represents the number of objects fetched.

Component	Max	Avg
Frontend (Green Area)	72.7k	41.9k
Backend (Yellow Area)	11.3k	7.3k
Fetched Objects (Orange Area)	8.7k	5.8k

Table 4.3: Request Statistics for the Specified Components

Chapter 5

Ways to Improve CDN's cache effecienty

Seznam.cz in 2024 year made a strategic decision to increase caching capabilities of company's proxies. Improving cache efficiency on proxies includes increasing efficiency and effectiveness of storing and retrieving stored in internal cache content. This can involve implementing some new caching algorithms, upgrading hardware infrastructure including network links, or just optimizing software configurations. By focusing on this cache improvement, Seznam aims to deliver its content to end-users more quickly, enhancing the total user experience.

A higher CHR indicates that a larger portion of user requests can be served directly from the edge-cache without any need to access backend servers. This reduces a workload on a backend infrastructure and can lead to resource savings and improved performance of process. When content is often accessed and is stored in the cache on proxy, this reduces the need for extra and repeated requests to backend servers. This not only saves compute resources but also reduces network traffic and latency. By optimizing the CHR, Seznam.cz can minimize the amount of requests going straight to its backend servers, leading to improved reliability of its services. Higher CHR also contributes to a more efficient usage of resources, allowing Seznam to allocate computing resources more effectively to handle peak loads.

Content Delivery Networks play a main role in delivering content efficiently to users by caching and distributing content closer to the end-users by reverse edge proxies and web accelerators. Improving cache store efficiency on proxies can significantly reduce the workload comming to CDN backends from users, particularly for tasks such as generating content for Video on Demand (VOD) and Live streaming.

Generating content for both VOD and Live streaming involves resource intensive tasks such as parsing data, processing and generating media playlists in formats like HLS or DASH. By offloading some of this work to the cache, CDN backends can work more efficiently and serve more requests that are unique. User gets faster content downloading, as content can be served directly from the cache without need to wait until backend process data sent to it.

5.1 Improvements on hardware level

Implementing some improvements at the hardware layer, may increase the performance, capacity, and efficiency of the system that is serving a web accelerator or HTTP server.

- **Increase amount of RAM** Increasing amount of RAM on a devcie may allow it to store and access more data. More RAM enables the system to operate with larger dataset.
- **Use hard disk as additional storage and virtual cache space** - Using the hard disk as additional storage and virtual cache space can increase the system's capacity and improve its ability to store and retrieve more data than with only RAM used. Hard disks offer high-capacity storage at relatively lower costs compared to RAM. But hard disks has its' own drawback - they are slow, RAMs are much faster reading and storing data than hard disks. Therefore, there is a big trade off for company and maintainers of cache server. One of the common use-case for companies to store content within combination of RAM and hard disks is storing data that is accessed small amount of times per day into a HDD / SSD. By storing less frequently accessed data on the hard disk, it frees up RAM space for more more so-called *hot-files*.

5.2 Improvements on software side

More effective caching system may be as a result of software updates, features and maintaince. Varnish servers may be configured in wide range of combinations. Topology may change to serve cache more effectively. By modifying the network topology with several proxies, engineers may optimize cache delivery and improve efficiency of a served content. Implementing a hashing (as was previously mentioned in 3.1.1) algorithm can help distribute requests ***evenly*** across cache servers, reducing the load on individual servers and improving overall caching efficiency by sending user requests for same data to a responsible Varnish instance that is owner or should be an owner of this cache object. However, hashing algorithms have their challenges - it is typically implemented at Layer 7 (application layer), which may introduce overhead and complexity on load balancers that should distriburte requests. Hashing at Layer 7 involves parsing application layer data from request, such as URLs or HTTP headers(to extract cookies, for example), to figure out which neighbour is the possible owner of a cache object. Sharding is using a consistent hashing[1] algorithm, which serves stability on resize of hash circle of cache servers. Hash circle resize may occur on server joining group or leaving it. Group leaving may be caused by network partition, machine overhead, machine reboot and other cases when server stops responding unexpectedly. It is more dangerous than joining as it should be detected immediately and apply rules that would stop routing to this node. Possible fallbacks are discussed later in this work.

There is a possibility to build several layers of caching proxies where the deeper proxy is located (from client's side) the less popular content it stores in its storage.

This work targets and discusses one- and two-layered topology of a Varish web accelerators. Architecture involiving only one layer of proxies is self-routing and is quite popular in

[1] Consistent hashing - is special case of hashing process which is good for handling resize of group included in hash. On resize only $\frac{n}{m}$ keys have to be mapped again where n is the number of keys already stored and m is the number of participants in hash ring

community of engineers. Self-routing means that instance of Varnish could possibly be intended to „send“ it to this particular instance, to itself. When this particular case happens, node is starting the standard flow of processing incoming request - lookup in cache, on miss or when object in cache is in its grace period, request is forwarded up to backend and then cached based on internal logic of web accelerator. Self-routing topology of nodes in cluster is quite easy to implement, the most fundamental thing to account with is that the hashing algorithm should be same on each instance, and hash cirlce has to have same amount of keys or participant that are enrolled in backend calculation, otherwise it's possible to stack in infinite loop of request juggling.

More complicated case of cluster architecture with sharding process is the one in which engineers imlpementing two layers physically or virtually(instances of the first and the second layer could easily run on one machine). When we have more complicated topology its aslo gives us space to expirement and play with request forwarding. Its possible to implement sharding on the first layer and leave second as just long-time cache, make both layers as independent clusters, and the last is make the second layer routing while leaving the front one as just shield from the most popular requests. However, all three implementation assumes that the first layer would serve the most popular requests from users, while second one would be a long-term cache that holds object that could be cached for a long period of time. In this work we discuss a third written concept with non-clustered first layer and self-routing cluster of nodes on the second one. This topology has big advantage - covered case of hot-spotting [2]

5.3 Implementation direction decision

Desicion was to implement 2 layered topology with sharding on the second layer of nodes. As was mentioned before the first line of Varnishes should serve the most popular objects from its cache storages. Also one of interesting implementation details is that second instance of web accelerator going to take place on the same machine where the first one is running. This gives us two advantages:

- There is no need to create an additional VM or set up a new baremetal server and therefore minimizes deployment complexities. Moreover, this approach reduces the need for additional hardware infrastructure and simplifies hardware management.
- In case that front node is going to forward request to the second proxy, traffic stays within the confines of the local machine, eliminating the need for data transmission across the network. This not only accelerates response times but also reduces the load on network bandwidth.

As second layer should not be visible and accessable by clients of a company listening port of it should not be exposed to an external network. So port was opened on a internal VLAN with a dedicated interface for intra-cluster communication. As nodes that are participating in one hashing ring for sharding are located in one datacenter, communication in cluster is fast and not leaving datacenter, instead it is just forwarded between server racks in server rooms.

[2]Hot-spotting or hot shards appear because organic laod patterns drive more traffic to one particular shard [1]

5.4 Topology efficency prediction

This chapter describes how prediction could be made and results extracted to confirm that the new topology would 'behave' and operate better, usage of a cache storage would be more effective and total CHR would be increased with followig decreasing of request amount to Seznam's backends. There is two possible ways how to measure an effectiveness of a new topology - build an exact copy of current production solution and change its configuration, then start to simulate a production data-flow with metrics scraping from Varnishes; the second way would be to build a simulation model that would allow us to simulate behaviour of proxies. First approach seems to give us the results that are approximating to the real one as it real software, exactly same that is running in production, requests are recreated from a day logs. However, this solution could benefit real good in case of accuracy of results, despite on TTL of objects, a cost of implementation is enourmous in human hours and resources. There is need to configure new virtual machines or setup bare metal servers. On the other hand, second way gives us less trust-worthy results. Simulation model should be tested on dataset and model should be proved to serve and give a result that refers to real ones. But in case that model would be valid and gives objective results reflecting production CHR it could save a lot of time and speed up process of development, simulating and gives an opportunity to run simulation in parallel to test different topologies, their configurations and get results much faster. Based on the fact that for a business it is good to get results faster which brings faster profit, simulation way was chosen as a way to prove efficency of the future topology.

5.4.1 Topology recreation

To recreate topology for simulation preference fell on GoLang[3] programming language. This language is good in both important aspects for our simulation goal. It is **fast** and it is a **high-level** language, combination of two gives us a proper way to implement model in much faster way than in any low-level language like *C* or *Rust*; and property that language is compiled gives opportunity to collect results from simulation much faster than if the the model was implemented in language like *Python*. Moreover, one of the best feature that is implemented in *GoLang* is that binary files does not have any external dependency! That allows to copy binary to any machine with same processor architecture and running on same OS(Linux-based systems and OS X compatible as both follows UNIX standards).

The implemented model is stored and can be found on Github/Drakorgaur; source code is under Apache 2.0 license.

5.4.2 Dataset

Dataset for the simulation process was day-long log files from all edge proxies on one datacenter, that mean that it contains logs from all 12 proxies through one day. Logs were merged, sorted by timestamp and formatted to save disk space. After filtering log would contain space it took in cache and url to calculate key under which object would be stored.

```
ubuntu@pomo-huge-disk:~/$ ls -lah logs/dataset.log
-rw-rw-r-- 1 ubuntu ubuntu 147G Feb 14 12:02 logs/dataset.log
```

[3]GoLang or just Go - is a compiled high-level programming language with static typing designed and implemented by Google

5.4.3 Results and analysis

Despite the fact that model does not take into account TTL of cache objects, it gave pretty good results for architecture that was implemented in 2023 year on CDN.


```
{
  "proxy-23": {
    "cache": {
      "hit": 71245273,
      "hit_ratio": 0.8507157800455882,
      "miss": 12502172,
      "total": 83747445
    },
    "cache_size": 176000000000,
    "cache_used": 175996432943,
    "routes_to": [
      "default"
    ],
    "routing": {}
  }
}
```


Above is presented one of metric set extracted from model of proxy with Varnish. Total cache hit rate is set to 85% with full memory used. During the start of 2024 year machines that hosted Varnish instances got a new RAM plates which doubled available RAM to 512Gb. Althoug there was boost in RAM amount, not all RAM could go to Varnish itself for allocating memory for cache objects, there is need to leave some memory for other processes including Varnish daemon, lurker, OS running and etc. With new configuration plan where the Varnish layer got a 372Gb to store user requests, there was a opportunity to predict how CHR would change.

```
{
  "proxy-23": {
    "cache": {
      "hit": 55188813,
      "hit_ratio": 0.8819648515623947,
      "miss": 7386031,
      "total": 62574844
    },
    "cache_size": 372000000000,
    "cache_used": 371999997182,
    "routes_to": [
      "default"
    ],
    "routing": {}
  }
}
```


Simulation model showed us a result of CHR set to 88.2%, after applying changes and reconfiguring Varnish daemons and scraping metrics from them we have got approximately same raise.

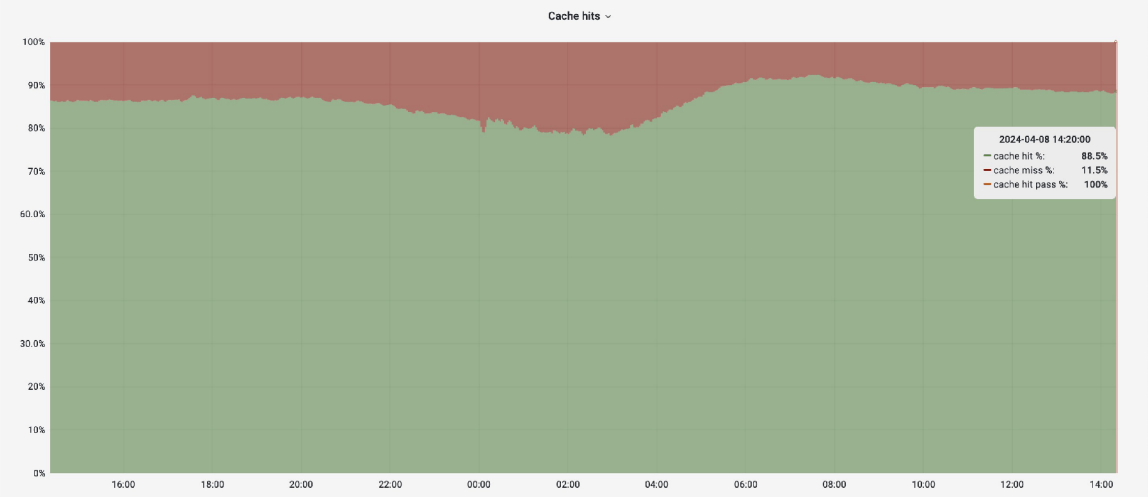


Figure 5.1: Graph displaying one-day statistics of caching hits on proxies with 372Gb of RAM given for cache storage

CDN engineer team can trust the model, dispite the fact that model gives only final CHR. Expectation based on model prediction met the real production results. That gave a freedom to actions and modeling two layered topology. Even that expirements was made more than four, below is described the four the most interesting ones.

Two best cases from table ?? for Seznam's Content Delivery Network would be configurations - first, where 74 on the first layer and 298 on the second gigabytes of RAM; and second where the first layer has 149 and the second one has 223 gigabytes of RAM. Prediction is for both cases on CHR are more than 96% this means that on comparement with current 88% half of requests that are passed to backend servers would be served by

Table 5.1: 2 Layer Cache Configuration

1L Size	2L Cache	CHR 1L	CHR 2L	CHR
74 GB	298 GB	81%	83%	**96.77**%
149 GB	223 GB	85%	77%	**96.55**%
223 GB	149 GB	87%	62%	95.06%
298 GB	74 GB	88%	48%	93.76%

Cache proxies! Taking to the account that Seznam has huge network flow half of requests would mean a lot of saved CPU processing time, less loaded backends, databases etc.

5.4.4 Decision

So decision was made to move aside with solution where we have 74 gigabytes of RAM on the first layer and pass left 298 gigabytes to the second layer. This should give cache hit rate boost *by predictions of model* from 88.2% to 96.77%. Both instances would be running on same machine while still exposing to users only one port - port of the first layer, meanwhile the second layer would serve as huge network storage or in some kind of virtual RAM for Varnish, storing only unique content from end users. Both layers are communicating via VLAN with possibility given to CDN team from Seznam's admin to create a dedicated physical connection. But by estimation current links could easily handle a new load with increased traffic passing between nodes that are running Varnish instances on them. By design all routing and business logic should do the first layer of proxies, while the second layer is just a simple cache storage with some additional functionalities like headers incapsulation, retries and etc. On first attempt decision was made to store objects in cache on second layer instances with same TTL as they are stored on the second one.

Chapter 6

Implementation and deployment

Implementation includes creating new configuration files for Varnish daemon in *VCL*[1], creating new systemd service that takes care of new daemon of Varnish proxy that listening on dedicated interface and storing metadata in directory different from the default one, which is used by Varnish deamon serving first layer connections; directory changed by '-n' flag in Varnishd and other useful tools from Varnish software like *Varnishadm* or *Varnishlog*. Then is added script for automative restart or reload of Varnish daemons when VCL or systemd files are changed to fullfill CD[2] in the CDN team. Also there is need to setup new metric exporter implemented aside with Prometheus - this tool is called Prometheus Varnish Exporter and could be found on jonnenauha's Github. Prometheus Varnish Exporter is just translating Varnish metrics to format that Prometheus could understand and exposed this metrics on a specific port. As a matter or fact, there is need to connect them to CDN's Thanos store and scrape to have observability on Seznam's new implementation of proxy nodes. After new thanos configuration there is need to update Grafana's dashboards to work with new metrics as total cache hit rate is now calculated in an another way than it was a year ago.

After all preparations were done and new topology was tested it was a time to release new topotogy to production and get new metrics. There was a decision to make a rollout to one server room only so in a case of error or instability we could serve content from the other ones. Also rollback plan was prepared to turn lost datacenter on as fast as possible to remove load from other datacenter rooms. In the middle of april Seznam was successfully running with one datacenter set to new topology. After one week of stability rest of datacenters were put on new topology as well.

[1] VCL - Varnish Configuration Languages
[2] CD - continious deployment

6.1 Metric Gathering and comparement

6.1.1 RAM usage and warmup

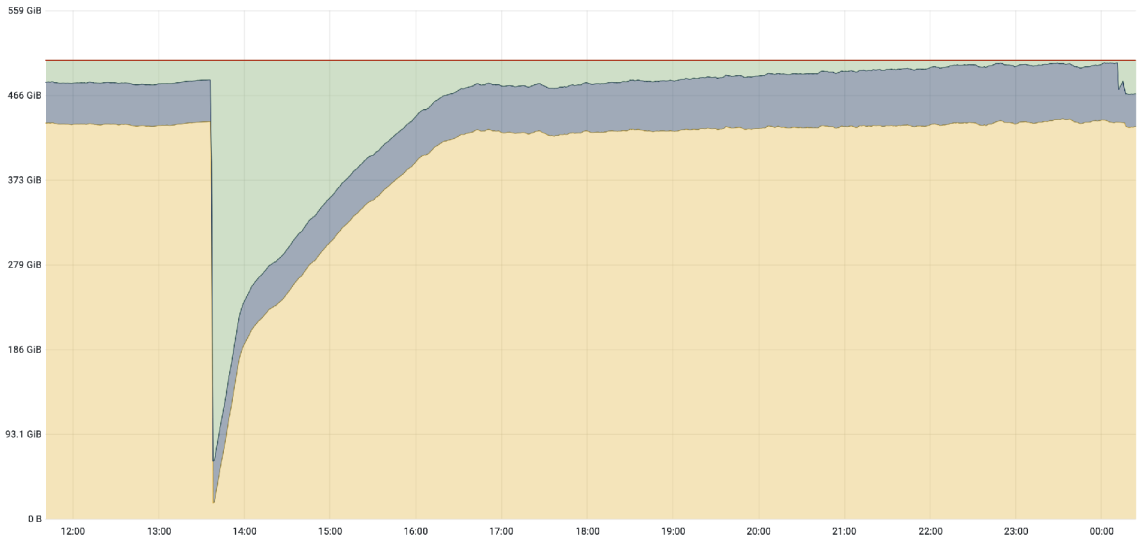
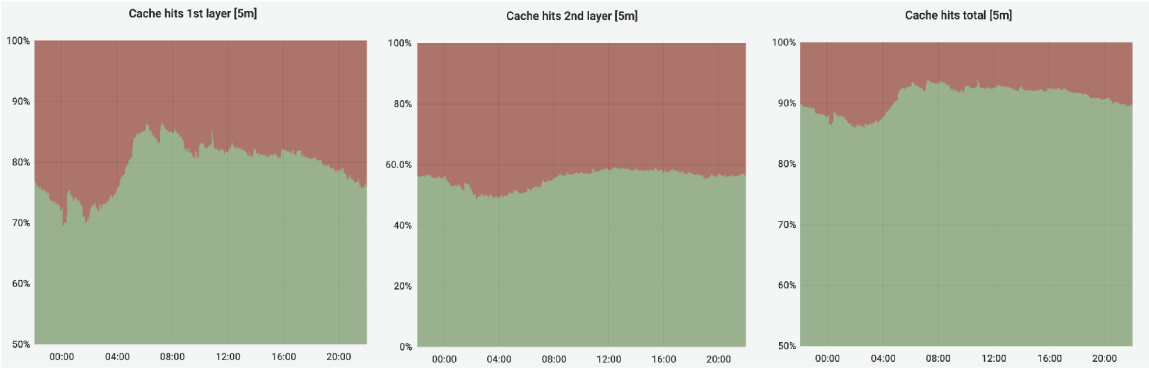


Figure 6.1: RAM Usage After Restart

After rollout of new configuration there is warm-up time needed by proxy to fullfill its cache store. As shown on 6.1 proxy has interval of 2 hours to end its warm-up period. Arround 14:00 there was an angle change of RAM usage graph, RAM usage value at this time is arround 190GB RAM, that is because smaller storage did fill and since this point of time only bigger cache storage continuing to fill and store new objects.

6.1.2 Cache Hit Rate



(a) (b) (c)

Figure 6.2: Cache Hit Rate through 1 day of production traffic with new architecture

Figures 6.2 based on metrics extracted from proxies of new topology visualizes 3 cache hit rates:

- Figure 6.2a shows a CHR that is average between all proxies in one cluster on the first layer. *[5m]* means that PromQL[3] uses function rate with uses per-second rate of increase of a metric within the specified time range of 5 minutes. Graph has *cache hit* minimum set to 72.0%, maximum set to 88.2% and average to 81.4%.
- The next figure 6.2b has same query but on metrics that come from Varnish instances on 2nd logical layer. It shows *cache hit* minimum as 51.0%, maximum set to 62.2% and average to 58.3%.
- The last figure 6.2c, as is understood from its name it shows a total cache hit rate for a new solution and a new topology. This CHR is calculated by next formula:

$$\mathrm{CHR}_{\mathrm{total}} = \mathrm{CHR}_{1\mathrm{L}} + (1 - \mathrm{CHR}_{1\mathrm{L}}) * \mathrm{CHR}_{2\mathrm{L}} \tag{6.1}$$

  This formula calculates CHR as a count of hits in a cache on the first layer plus number of hits on the second that are performing on requests comming from a cache miss on the first layer of proxies. Cache hit is set as following: minimum is 86.5% maximum is set to 94.3% and average is set 91.7%

6.1.3 Memory usage and LRU nukes

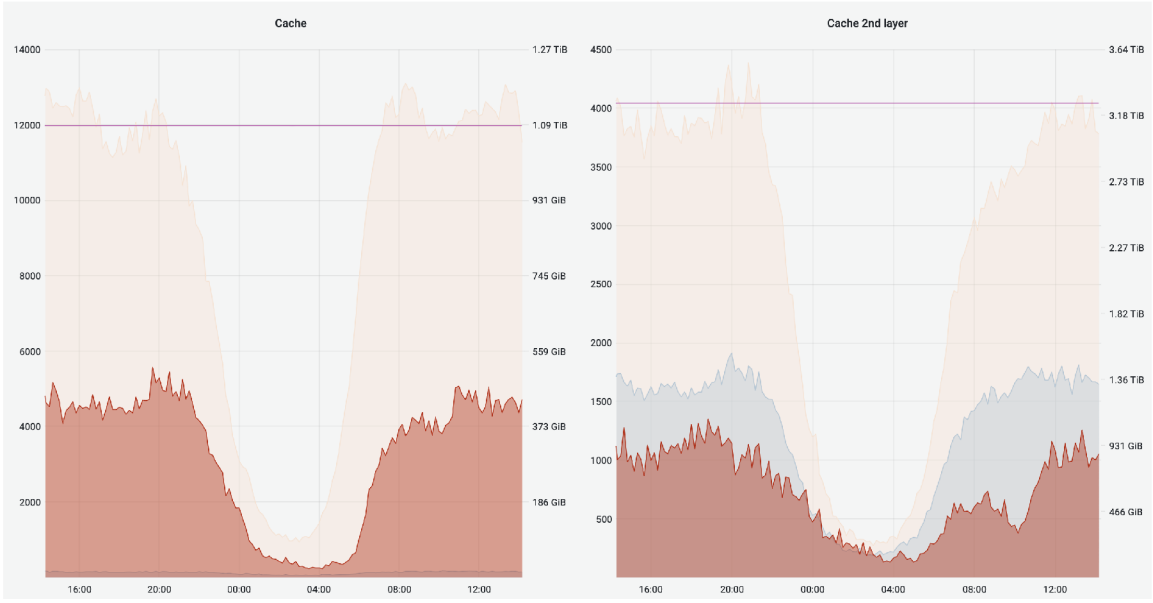


Figure 6.3: Cache Usage 2 Layer arhitecture 1 day.png

Table 6.1: Data Summary for first graph

	Min	**Max**	**Current**
LRU moved (light red)	943.47	13k	11.5k
LRU nuked (red)	228.40	5.5k	4.7k

[3]PromQL - Prometheus Query Language, functional query language for queries to Prometheus database

Table 6.2: Data Summary for second graph

	Min	Max	Current
LRU moved (light red)	271.80	4.3k	3.8k
LRU nuked (red)	130.93	1348.20	1053.51

Visulizations of Cache usage on proxies is quite useful and gives a few things for next analysis. First, LRU Nukes are less common for the second layer of proxies, which means that cache space is used really effectively and Varnish less often removes old objects from cache storage to store a new ones. The second interesting conclusion that could be made out from graphs is that objects has it expiration time reach much more often comparing with objects that are handled by the first layer.

6.1.4 Network state

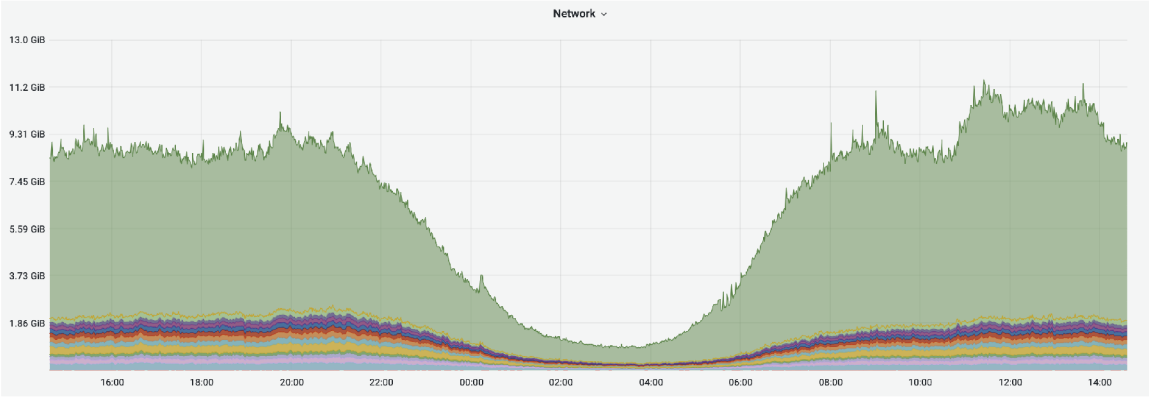


Figure 6.4: Network State 2 layer topology

Network flow is increased comparing with previous solution by 20% on avarage. Which means that on serving same amount of user requests we're creating on 20% packets comming between proxies, but this graph does not shows requests count comming to backends from Varnish instances.

6.2 Conclusion

Even though the simulation model overestimated cache hit rates on the second logical layer of proxies, a great result was achieved with an increasing cache hit rate using the provided changes and improvements in Varnish proxy configuration. Furthermore, this is only a change in the configuration and way proxies communicate with their backends; the backend for a concrete proxy can be another proxy or the real backend that serves client requests. As expected, network traffic has increased, but not in the way we can not handle it, and future improvements to physical links may improve our ability to handle network traffic between proxies. With provided „shared“ and sharded cache for proxies we achived uniqness of content on the second layer with more efficient usage of RAM available to CDN's team, this brings less amount of LRU nukes of objects on second layer, but it brings higher

rate of object expiration in cache comparing to the first layer of two layered topology and comparing to previous implementation of topology.

Even that cache hit rate is increased by several percents, it's a big deal for company. The higher CHR the harder to improve it far. So average CHR raise from 83.6% to 91.7% is quite big for Seznam.

Chapter 7

Possible improvements

Even that cache hit rate was increased to an average value of 91.7% there is still options to improve work and effecincy of serving user requests by changes in configuration and hardware. This chapter takes a look on this possible future improvements that should help proxies serve content more effiecient.

7.1 Different cache time on layers

As mentioned in the section 5.4.4, decision was made to leave TTL on both logical levels same, which than can be observed on graph 4.1. This brings idea to play and configure TTL of objects that are storing on 2nd layer to increase time accessable to object from cache instead passing it to the backend.

7.2 Different cache sizes for layers

As was mentioned in previous section lot of objects are getting state of expired and removed from cache by Varnish. This is not a bad factor, but that means Varnish is not removing it efficiently to store new data, instead some part of memory became available and empty, just waiting to store a new object. This also gives us a understanding that if we have available space, we could move it on first layer to handle more requests there with increasing CHR of the first layer and not loosing the last on the second layer.

7.3 Warmup section

Varnish shard director which picks backend with sharding process has a option of warmup, which brings opportunity to select a different backend with variable probability. This probably is not going to increase an amount of objects that are server from cache, but it brings an useful factor - it is creating a backup for case of unavailibility of proxy holding unique object. So Varnish caclulates new virtual hash ring without proxy that picked from current ring, and picks new potentional responsible backend for this object and instead of proxying request to real owner of this object Varnish sends request to its „deputy“ creating fallback in case of network partition of other problems that would disconect nodes.

7.4 Inter datacenter clustering

Current solution is based on independent clusters that are located in different datacenters. This means that „unique“ objects through the cache are not really so unique, they are duplicated across datacenter rooms. Real uniqness could be solved by creation of one cluster across all datacenters. This way brings opportunity to decrease memory amount provided to second layer and give it to the first one, while keeping or increasing total memory of the second logical layer. This brings following trade-offs for increased cache hit rate:

- Latency would be increased to pass request between datacenters as servers would be geographically located far. But with assumtion that all datacenters are located in Czech, and moreover in the same city, delay should not be so tangible.
- Another issue is a network partition between datacenters. This means that datacenters are just loosing about half of objects that are stored in the second layer of different datacenter when can not see the last one. Despite this huge hole in cache storage on this case, most of objects are still stored on the first layer giving a big buffer and saving backends from an overload. Continuing with handling network partition as was mentioned in 7.3, Varnish could create backups of objects, but it does not give a waranty that objects would be duplicated with probability across different rooms.

7.5 Physical link update

Earilier in this work was discussed that there is possibility to set up new links for intra-cluster communication and increase it bandwidth and link speed. This change can help to remove worries about increasing network traffic comming from proxy to proxy, and a slightly decreasing the latency for end user, however cache storage and usage would not be more efficient.

7.6 Sliding window handling

With new two layered topology there appeared a new potential problem - double time caching due to sliding window of requests. As illustrated in 7.1 diagram, when no proxy has an object, and request is coming to proxy number one (on the first layer) it passes request to owner of object on the second layer, it has the object in cache and should ask backend. Let’s assume backend returned picture and TTL of the object is one hour, then proxies returning response to the initiator of communication. Then after 55 minutes another request comes to proxy number two, it has no object stored, then it is forwarded to owner, which has this object, but this particular object have TTL only 5 minutes left, and despite this front Varnish is storing picture in cache for another hour. Simple calculation gives us a time of 1 hour and 55 minutes of total cache time instead one hour for pictures.

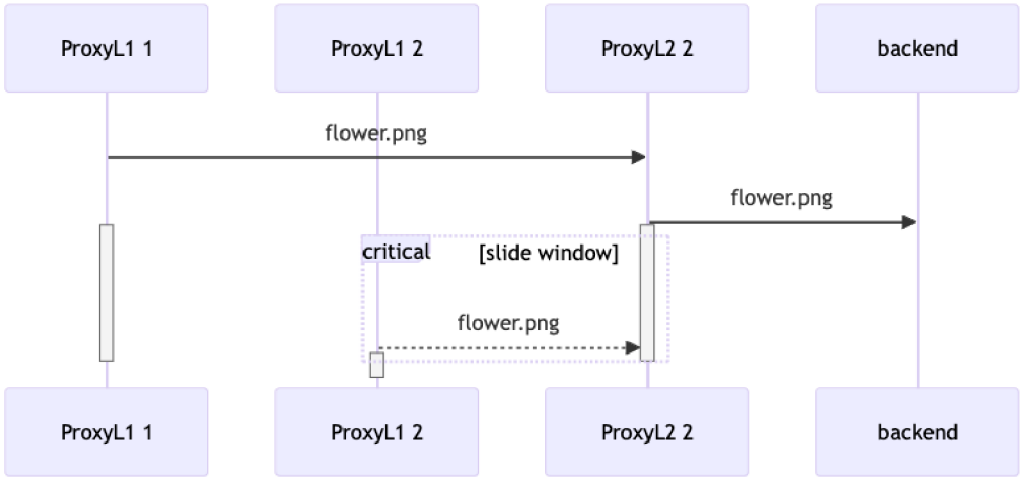


Figure 7.1: Sliding Window Problem

This could be a problem for freqently refreshing object in cache. Despite the fact, that old architecture had the same problem, this is still problem. And with two layer topology we have one source of truth - Varnish instance on the second layer, the real owner of cache object. This Varnish instance can propagates to front proxies TTL that *left* for object and this would bring to expiration of object on all proxies in approximately same point of time.

Chapter 8

Conclusion

Main target of work was to increase a cache hit rate and therefore increase performance of CDN in Seznam.cz. In this work I described main concepts of Content Delivery Networks together with how CDN is implemented in company Seznam.cz and what main parts are participating in process of serving content from CDN. Then we discussed what role cache plays in CDNs and its particular role in Seznam, which techologies are used to cache different contents on different levels, what policies does Seznam has on its cache objects, and what alternatives Seznam has. The solution implemented by Seznam.cz was reviewed, as for its proxy nodes that were running Varnish instances to cache objects for end-users and remove load from backends and give them space to process uncachable or hardly-cacheble content instead. By provided graphs from Seznam's monitoring it was discussed state of Varnish web accelerators. After this work contains discussion how is it possible to improve total amount of cache hits and serving more efficient requests from cache instead of passing user requests to the backends. Chapter 5 gives few points of improvement direction to raise cache hit rate and how is it possible to confirm and prove that proposal solution would really work in production environment and would increase total objects serving from cache. In the end of chapter 5, why did I choose to build a simulation model, which on real production dataset of request could predict effectiveness of new topology and therefore I could pick a solution that would increase cache hit rate the most. Due to positive results of simulation I started to reconfigure proxies for staging and production environments. Following by chapter 6 it is described process of implementation of a chosen solution, work also shows results extracted from new solution and compares important metrics to previous. Results showed that in average per day cache hit rate was raised about 10 percent, from 82-83% to 91-92%. Also this chapter contains discussion about discrepancy in cache hit rate prediction of the simulation model and real results scraped from proxies.

The result of this bachelors thesis, is increased cache hit rate of CDN solution by reconfiguration of proxies and they routing table by creating a cluster of nodes with several layers, where first layer is responsible for serving the most popular content for end user and the second layer is responsible for the storing object that should have long term of living in cache. This topology brought significant decreasing of request comming to backends and therefore CPU processing time. However, trade-off for this improvement was increased network load between proxies as proxy on request calculates the node-owner of object and redirects request to it, creating single point of truth for cache system. Also work covered specification of hot spotting problem and how it is handled in a new solution of proxy topology.

The chapter 7 provides some notes which should be taken into account to improve caching system or make it more sustainable starting from software and configuration details to hardware improvements.

Bibliography

[1] Burns, B. *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. O'Reilly Media, Inc, Sebastopol, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2018. ISBN 1491983647.

[2] Chen, L., Liu, J., Xian, M. and Wang, H. Docker Container Log Collection and Analysis System Based on ELK. In: *2020 International Conference on Computer Information and Big Data Applications (CIBDA)*. 2020, p. 317–320. DOI: 10.1109/CIBDA50819.2020.00078.

[3] DeJonghe, D. *NGINX Cookbook: Advanced Recipes for High-Performance Load Balancing*. 1st ed. O'Reilly Media, Inc., 2022. ISBN 1492078484.

[4] Feryn, T. *Varnish 6 by example: A practical guide to web acceleration and content delivery with Varnish 6 technology*. Vulkan, 2021. ISBN 9189179978.

[5] Huston, G. *Ipv4: How long do we have?* [online]. 2003. Available at: https://www.potaroo.net/papers/ipj/2003-v6-n4-ipv4/ipv4.html.

[6] Rajkumar, B., Pathan, M. and Vakali, A. *Content delivery networks*. Springer Science Business Media, 2008. ISBN 3540778861.

[7] Cisco, D. L., Cisco, P. C. and Texas, M. D. U. of. *WCIP: Web Cache Invalidation Protocol* [online]. 2001. Available at: https://datatracker.ietf.org/doc/html/draft-danli-wrec-wcip-01.

[8] Zhang, H. and Zeng, H. Design and implementation of blockchain platform operation and maintenance support system based on Kubernetes+EFK framework. In: *ISCTT 2021; 6th International Conference on Information Science, Computer Technology and Transportation*. 2021, p. 1–8.