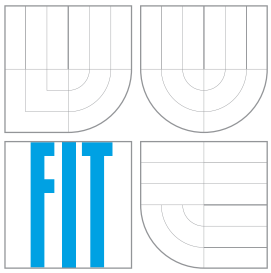
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTELIGENTNÍ DISTRIBUCE SOUBORŮ V CDN

INTELLIGENT FILE DISTRIBUTION IN CDN

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK KALETA

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. ŠKODA PETR

BRNO 2014

Abstrakt

Tato práce se zabývá algoritmy pro rozdělování obsahu v CDN na jednotlivé uzly systému. Srovnává lokální a globální algoritmy pro ukládání dat na zdrojové a cachovací servery, jednotlivé přístupy cachování a rychlost reakce a kvalitu případné předpovědi zátěže. Je vytvořen vysokoúrovňový simulátor CDN, navržen popis mapování pomocí matic a navrženo několik algoritmů včetně genetických k mapování souborů na jednotlivé servery

Abstract

This work deals with algorithms for distributing and mapping content on nodes in CDN system. Compares local and global algorithms for loading files on origin and edge servers. A high level CDN simulator is made. A matrix based approach for mapping content on CDN servers is proposed along with tranformation for solution of mapping optimalisation through genetic algorithms.

Klíčová slova

CDN, Systém distribuce obsahu, vyvažování, předvídání zátěže

Keywords

CDN, Content delivery network, loadbalancing, load prediction

Citace

Marek Kaleta: Inteligentní distribuce souborů v CDN, diplomová práce, Brno, FIT VUT v Brně, 2014

Inteligentní distribuce souborů v CDN

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana magistra Petra Škody.

.....
Marek Kaleta
31. července 2014

Poděkování

Děkuji vedoucímu práce a spolupracovníkům z firmy Seznam.cz za odbornou pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Tématem této práce je návrh, implementace a simulace algoritmů pro distribuci obsahu v systému CDN společnosti Seznam s přihlédnutím ke specifikům této CDN.

Kapitola 2 popisuje obecné systémy CDN, požadavky na ně kladené, jejich dělení, historii a současný stav. Popisuje jednotlivá řešení problémů CDN a jejich výhody. V kapitole 3 je představena druhá generace sytému CDN Seznamu na úrovni architektury, nástinu funkce komponent a základních případů užití a systém je zařazen do klasifikace ustavené v předchozí kapitole. Parametry a typickou zátěž systému Seznam CDN pak popisuje a analyzuje kapitola 4. Kapitola dále nastiňuje budoucí vývoj tohoto systému.

Pro ověření navrhovaných algoritmů bylo nutné vytvořit simulátor CDN. Jeho návrhem se zabývá kapitola 5, jeho testováním a ověřením na běžných provozních požadavcích pak kapitola 6. V ní jsou také nalezeny pravděpodobné limity současného stavu CDN Seznamu beze změn navržených v této práci.

Popisu a rozboru problému distribuce dat z hlediska CDN Seznamu a popisu některých z algoritmů se věnuje kapitola 7, jejich porovnání a konkrétnímu návrhu a implementaci vybraných algoritmů pak kapitola 8. Konečně v 9. kapitole jsou algoritmy otestovány a porovnány v simulovaném prostředí na běžné zátěži, předpokládané budoucí zátěži, simulované špičkové zátěži, jejich reakce na změny konfigurace clusteru CDN a to buďto selháním některých strojů nebo rozšířením kapacity CDN jak změnou parametrů tak přidáním strojů. Dále je nalezen limit jejich zátěže na současné konfiguraci Seznam CDN. Na základě těchto testů je zvolen nejvhodnější kandidát na integraci do systémů Seznamu a nastíněny možnosti dalšího vývoje.

Celkové zhodnocení práce je provedeno v kapitole 10.

Kapitola 2

Sítě pro doručování obsahu

Růst velikosti dat a provozu v Internetu zapříčinil problémy s kapacitou serverů obsluhujících požadavky uživatelů. Dalším problémem centralizované obsluhy požadavků byly prudké výkyvy zátěže [4] a z toho plynoucí přílišná zátěž nejen výdejových serverů, ale i další síťové infrastruktury. Tyto problémy lze řešit mimo jiné (data gridy, distribuované databáze, peer2peer sítě) pomocí sítí pro doručování obsahu.

2.1 Definice

Podle [22, 31] je síť pro doručování obsahu (dále Content Delivery Network, zkratka CDN) kolekce spolupracujících síťových prvků v Internetu, kde je obsah replikován na několik webových serverů k zabezpečení transparentního a efektivního doručení obsahu k uživatelům. Spolupráce mezi jednotlivými komponentami CDN může probíhat v homogenních i heterogenních prostředích.

CDN bylo vytvořeno pro překonání omezení internetu především z hlediska uživatelem vnímané *Quality of Service* (QoS). Přínosem CDN tedy jsou služby pro zvýšení síťové propustnosti, dostupnosti a udržení správnosti obsahu.

Základními funkcemi CDN tedy je:

- Přesměrování požadavků na obsah na nejbližší vhodný cache server
- Distribuce obsahu, replikace nebo cachování obsahu na dostatečné množství cache serverů
- Služby vyjednávání pro poskytování takových služeb, které vyžaduje zákazník CDN.
- Management a monitorování jednotlivých komponent sítě

V kontextu CDN, *doručení obsahu* (content delivery) popisuje akci obsloužení uživatele požadovaným obsahem. *Obsahem* mohou být libovolná digitální data. Obsah se skládá ze dvou hlavních částí: enkódovaná média a metadata. Enkódovaná média mohou být statická, dynamická nebo proudová, tedy dokumenty, audio a video, obrázky, a webové stránky. Metadaty se rozumí popis obsahu umožňující jeho identifikaci, nalezení, distribuci a interpretaci. Obsah může být uložený do CDN nebo živě přenášený.

V rámci modelu OSI může být CDN chápána jako nová virtuální vrstva nad aplikační vrstvou. Pro transport pak používá protokoly aplikační vrstvy (HTTP, RTSP) [10].

Vztah k CDN mají tři hlavní entity: Dodavatel obsahu, poskytovatel služeb CDN a koncový uživatel. Dodavatel obsahu dodává data, která mají být distribuována. Ta se nachází

na *origin serveru*. Poskytovatel CDN poskytne dodavateli obsahu infrastrukturu potřebnou k doručení obsahu koncovým zákazníkům.

Pro CDN jsou pak používány *cachovací* nebo *replikační* servery, někdy také nazývané *edge* servery. Celá skupina edge serverů je pak nazývána *cluster*. CDN distribuuje obsah na jednotlivé edge servery tak, že obsah sdílí jméno a URL. Koncový uživatel je přesměrován na optimální edge server a CDN je tak pro něj transparentní.

Hlavními komponentami CDN pak jsou:

- komponenty pro doručení obsahu - origin a edge servery
- komponenty pro směrování požadavků - směrují požadavky koncových uživatelů na vhodný edge server.
- distribuční komponenta - přesunuje data z origin serveru na edge
- monitorovací a účtovací komponenta - monitoruje zátěž systému, dodává podklady pro účtování dodavatelům obsahu.

2.2 Historie

Vývoj CDN lze rozdělit 4 fází: před CDN a tři generace CDN.

Řešení před vznikem CDN spoléhaly nejprve na navyšování výkonu výdejového serveru a to po hardwarové stránce, případně pomocí serverových farem. Serverové farmy sdílí zátěž, která je na ně rozdělována pomocí přepínačů pracujících na vyšších úrovních modelu OSI. Nevýhodou takových řešení bylo především neoptimální využití kapacity jednotlivých strojů a výdej byl stále limitován kapacitou síťové infrastruktury, jelikož řešení bylo stále centralizované.

Dalším možným řešením bylo cachování provozu na úrovni ISP. Díky umístění *cache* velmi blízko k uživateli tak nejen snížilo průměrnou zátěž serverů poskytovatele dat, ale i celkovou zátěž síťové infrastruktury a také latenci. Evolučním krokem pak bylo hierarchické cachování, které poskytovalo další úsporu přenášených dat [5]. Fakt, že obsah je do *cache* nahráván podle požadavků uživatelů, však vede k tomu, že origin server i jeho připojení stále musí být dimenzováno na špičkové zatížení.

Na konci 90. let 20. století byla uvedena první *Content delivery network*. První generace CDN se soustředila především na poskytování statických a dynamických webových dokumentů [14, 29].

Za druhou generaci CDN se pak považují sítě zaměřené především poskytování streamovaného audia, videa, videa on demand, případně obsahu s vysokou mírou interaktivity s uživatelem. Třetí generací CDN pak může být postupná evoluce k plně decentralizované infrastruktuře.

2.3 Dělení a názvosloví CDN

2.3.1 Metoda nahrávání dat do cache

Lze dělit podle toho, zda edge servery vyžadují data, která na ně mají být nahrány (pull), nebo jsou data na edge servery nahrány pomocí jiných komponent CDN (push). Dále lze dělit podle vzájemné spolupráce edge serverů při vydávání dat.

Při nekooperativním pull nahráváním do cache je požadavek na data přesměrován na nejbližší edge server. V případě, že data se na edge serveru nenachází (cache miss), jsou stažena z origin serveru.

Kooperativní pull nahrávání se od nekooperativního liší tím, že edge server v případě cache-missu spolupracují na nalezení nejbližší platné kopie dat, která je poté nahrána na edge server.

Kooperativní push nahrávání znamená, že CDN udržuje mapování mezi obsahem a jednotlivými edge servery. Nahrání dat na edge server je řízeno CDN, jednotlivé edge servery spolupracují mezi sebou při updatech a distribuci dat. Díky existenci mapování dat je požadavek přesměrován na nejbližší vhodný edge server, případně origin server.

Výběr edge serverů, na které mají být nahrány repliky dat, může být proveden různými heuristikami: [12] uvádí čtyři heuristiky: random, popularity, greedy-single a greedy-global. [28] uvádí další greedy heuristiky vyvažující zátěž edge serverů. Heuristiky na základě latence přístupu jsou lat-cdn [20] a jeho evoluce il2p [19], který bere v úvahu i zátěž edge serverů.

2.3.2 Způsob nalezení dat

Nalezení edge serveru, který má cachovaná data, se dělí podle toho, zda se data vyhledávají uvnitř clusteru nebo v okolních clusterech.

intra-cluster

Nalezení dat na edge serverech jednoho clusteru může být *query-based*, *digest-based*, *directory-based*, *hashing-based* a *semi-hashing-based*.

query-based Při tomto postupu se edge server dotazuje všech ostatních edge serverů [33]. Výhodou tohoto postupu je absence centrálního prvku, nevýhodami pak značný provoz a zpoždění, jelikož dotazující musí čekat na odpovědi od všech ostatních serverů.

digest-based Dotazování okolních serverů eliminuje digest-based postup, kdy si každý server drží přehled o obsahu, který spravují ostatní edge servery v clusteru. Z toho vyplývá, že každý server musí informovat všechny ostatní edge servery o veškerých změnách v obsahu, který cachuje. Udržování přehledu používá značný provoz.

directory-based Tento postup zmenšuje provoz digest-based postupu, kdy je přehled o obsahu na jednotlivých edge serverech udržován na jediném, centrálním, serveru. Jednotlivé edge servery pak centrální server notifikují o změnách a dotazují se jej v případě cache-missu.

Tím snižuje provoz, ale centrální server může být úzkým hrdlem a v případě jeho selhání způsobí problémy výdeje v celém clusteru.

hashing-based Při hashing-based postupu jednotlivé edge servery používají stejnou hashovací funkci. Konkrétní obsah je pak spravován právě tím edge serverem, jehož ip odpovídá výstupu hashovací funkce pro URL kontrétního souboru [13, 30].

Tento postup má nejnižší provozní zátěž, relativně snadnou implementaci a nejvyšší efektivitu sdílení. Jeho problémem je absence škálování a také to, že i dotazy z lokalit blízkých konkrétnímu edge serveru musí být často obslouženy jiným serverem.

semi-hashing-based Problém škálování a lokálních dotazů řeší semi-hashing-based postup nalezení dat [9, 17], kdy je část edge serveru vyhrazena pro cachování často vyžadovaných dat, o kterých ale nejsou notifikovány ostatní edge servery v clusteru.

Tento postup udržuje výhody hashing-based postupu jen s poněkud nižší efektivitou sdílení při současné eliminaci nevýhody častého vydávání souborů z nelokálních edge serverů.

inter-cluster

Mezi clustery není vhodné používat postupy založené na hashování, protože jednotlivé clustery se mohou nacházet v geograficky značně vzdálených lokalitách. Digest-based postup není vhodný z důvodu velmi velkého síťového provozu, použitelný by mohl být v kombinaci s directory-based inter-cluster postupem lokalizace souborů.

Nejlépe použitelným tak zůstává query based postup [18]. V případě cache missu se edge server dotáže některého se serverů okolních clusterů (v případě hash-based inter-cluster postupu nalezení dat se může dotázat případně i přímo správného serveru). Pokud je obsah na clusteru nalezen, cluster odpoví původnímu cache miss edge serveru, jinak dotaz přesměruje na další okolní clustery. Při použití timeoutu na původním cache miss edge serveru a TTL u dotazovacích zpráv tento postup omezuje provoz nutný k vyhledání obsahu a snižuje zpoždění záporné zprávy o nalezení obsahu.

2.3.3 podle update cache

Existují 4 základní postupy pro udržování aktuálních cachovaných dat na edge/cache serverech:

Nejčastější je *periodický update*, kdy origin server každému obsahu určí, jak dlouho může být považován za aktuální. Edge server se pak pravidelně dotazuje, zda nebyl obsah aktualizován. Tím je však generován zbytečný provoz.

Oproti tomu při *propagaci updatu* je nový obsah pushován na všechny edge servery. Tento postup způsobuje vysoký provoz při častých updatech obsahu.

Při *update na žádost* je nový obsah na edge servery propagován pouze pokud byl edge serverem z origin serveru vyžádán.

Jinou metodou propagace updatu je *invalidace*, kdy na edge servery není šířen nový obsah, ale je zneplatněn starý. Nevýhodou tohoto postupu je obtížnější zaručení konzistence dat a z toho plynoucí neschopnost využít pro získání nového obsahu edge servery.

2.3.4 podle směrování požadavků

Úkolem systému směrování požadavků je přesměrovat klienta na vhodný edge server. Vhodnost edge serveru může být ovlivněna vzdáleností v síti, latencí, a zatížením konkrétních edge serverů. Systém směrování požadavků se skládá ze dvou částí: algoritmů směrování, které určují, který edge server bude zvolen pro obsloužení konkrétního požadavku, a mechanismů směrování, které o zvoleném edge serveru informují klienta.

algoritmy

Algoritmy se dělí do dvou základních skupin: adaptivní a neadaptivní. Adaptivní algoritmy berou v úvahu aktuální zatížení serverů CDN a síťových propojení, zatímco neadaptivní tyto parametry opomíjejí. Neadaptivní algoritmy jsou značně jednodušší pro implementaci

a jejich výsledky jsou blízké optimálním, pokud pracují v prostředí předpokládaném při návrhu jejich parametrů. Adaptivní algoritmy zaznamenaly velmi vysokou robustnost [32] v případě abnormální zátěže systému.

Jedním z nejjednodušších neadaptivních algoritmů je *round-robin*, který rozděluje požadavky rovnoměrně mezi všechny edge servery. Tento algoritmus je dostatečný pouze pro směrování v clusteru v jedné lokalitě, navíc za předpokladu, že všechny servery mají podobné parametry a obsloužit požadavek je schopen kterýkoli z nich. V případě, že se prvky systému nachází v různých lokalitách může být směrování neefektivní. Ani loadbalancing není dokonalý, pokud obsloužení různých požadavků vyžaduje různou kapacitu.

Složitější neadaptivní algoritmus může brát v úvahu vzdálenost klienta a serverů a předpokládanou zátěž serverů. Předpokládaná zátěž se odhaduje z počtu obsloužených požadavků na daném serveru. Algoritmus předpokládá, že vzdálenost serveru a klienta a zátěž edge serveru ovlivňují výkon nejvíce. Podle [1] takový algoritmus pracuje uspokojivě z hlediska směrování, avšak klientem pozorovaný výkon je stále špatný.

Cisco Distributed Director [11] obsahuje sadu dalších směrovacích algoritmů. Některé z neadaptivních odhadují relativní výkon serverů z poměru obsloužených požadavků jednotlivými servery. Požadavky jsou pak směrovány na nejvýkonnější servery. Takový algoritmus však trpí pozitivní zpětnou vazbou a požadavky tak mohou být směrovány na přetížené servery.

Variací na intra-cluster caching postupu nalezení obsahu je hashovací algoritmus směrování [13]. Klient je směrován na server s nejnižším id větším než výsledek hashování url požadovaných dat.

CDN Globule používá adaptivní směrovací systém, kde metrikou blízkosti je odhad síťové vzdálenosti klienta k serverům [27]. Odhad je pravidelně obnovován, měření je pasivní. Výsledky však nejsou příliš přesné.

Další možností je jako metriku použít latenci ke klientům [2, 3]. Pro její získání je možné použít výstup z přístupových záznamů nebo pasivní měření. Tento algoritmus však potřebuje udržovat databázi latencí a je tak obtížně škálovatelný.

Adaptivní algoritmus v Cisco Distributed Director používá tři metriky: Inter autonomous system vzdálenost, intra autonomous system vzdálenost a latenci. Latence je měřena aktivně a vytváří tak další provoz v síti.

Existují další proprietární algoritmy používané velkými poskytovateli CDN.

mechanismy

Mechanismy směrování požadavků se mohou rozdělit podle použité technologie:

První možností je použít Global Server Load Balancing (GSLB). Oproti standardnímu loadbalancingu, kde je loadbalancer informován o stavu připojených serverů, v GSLB je každý uzel (cluster) informován také o stavu ostatních uzlů. Díky tomu může za použití autoritativního DNS přesměrovat požadavek na libovolný z lokálních serverů nebo na jiný uzel.

Jiný směrovací mechanismus je založený na DNS. Jednomu doménovému jménu odpovídá více IP adres edge serverů a DNS server z nich vybere a do odpovědi vloží IP adresy serverů zvolených směrovacím algoritmem. Výhodou takového řešení je transparentnost, nevýhodami zjišťování metrik sítě pro klientův lokální DNS server a ne přímo pro klienta, zvýšení latencí při vyhodnocování DNS a sama struktura DNS, kdy odpovědi jsou cachovány a tak dotaz na autoritativní DNS server připadá pouze v 5 % případů [7].

Použití HTTP redirectů pro směrování dotazů má výhody v jednoduchosti, flexibilitě a jemné granularitě. Nevýhody jsou netransparentnost a zvýšení provozu o další dotazy.

Další metodou směrování je URL rewriting, kdy jsou URL v odkazujícím dokumentu přepsána na taková, která odpovídají nejbližšímu vhodnému serveru, respektive clusteru. Přepisování URL může být proaktivní, kdy se URL přepisují při nahrání na origin server, nebo reaktivní, kdy jsou URL upraveny při dotazu klienta na dokument obsahující URL. Výhody tohoto přístupu jsou dosažitelná jemná granularita a odolnost vůči selhání edge serverů, jelikož URL můžou patřit skupině edge serverů. Nevýhodou je zpoždění při parsování a přepisu URL v dokumentu a nemožnost cachování takového dokumentu.

Směrování požadavků pomocí anycastu může být provedeno dvěma způsoby: pomocí IP anycastu [21], kdy je jedna IP adresa přiřazena skupině serverů a a IP routery vždy směrují provoz na nejbližší z nich, nebo pomocí aplikačního anycastu [8], kdy je vytvořena skupina anycast resolverů převádějící anycastová jména na IP adresy. Klient se pak dotazuje na obsah anycastového resolveru, který odpoví adresou vhodného serveru pomocí dat o výkonu a vzdáleností edge serverů. Výhodou oproti IP anycastu je flexibilita, nevýhodou nutnost úpravy serverů a klientů.

2.4 Současný stav

S současné době vývoj na poli akademických CDN prakticky ustal, objevila se však řada úspěšných komerčních implementací CDN. Komerční CDN nezveřejňují implementační detaily. Jsou uvedeni dva zástupci komerčních CDN, avšak jednotlivých poskytovatelů existuje více (CDN77, limelight, MAXCDN, a další)

2.4.1 Komerční CDN

Akamai

CDN společnosti Akamai je v současné době největší na světě. Pro směrování požadavků DNS, kdy vrací adresy dvou nejvhodnějších serverů vzhledem k dotazujícímu se ([25]). Architektura je zřejmě klient - edge -origin. Implementace mechanismů distribuce obsahu není zveřejněna.

Amazon

Amazon nabízí řasu služeb v rámci své značky Amazon Web Services. Amazon Simple Storage Service nabízí služby CDN: Distribuci, replikaci nebo cachování obsahu v závislosti na zvolené službě, přesměrování požadavků, management a monitorovaní vloženého obsahu.

Uživatel si volí do kterého z regionálních redundantních distribuovaných clusterů bude jeho obsah uložen, kdy regionálně je replikován tak, aby byla zaručena úroveň dostupnosti dat. Data jsou uložena vyhledána pomocí slovníkové struktury, kdy obsahu odpovídá unikátní klíč, jde o textový řetězec.

Implementační detaily nejsou zveřejněny. Z vlastností lze odvodit architekturu klient -edge - origin.

Kapitola 3

Architektura CDN Seznamu

Společnost Seznam provozuje vlastní CDN pro výdej audiovizuálních dat na serverech novinky.cz, sport.cz, super.cz, mixer.cz a dalších. Je použito proprietární řešení.

Nová verze systému CDN seznamu je vyvíjena od roku 2012. Mezi požadavky patřila zejména redundance, škálovatelnost, spravovatelnost a použití open source nebo vlastních komponent.

Podle koncepce redundance Seznamu jsou veškeré komponenty provozovány ve dvou kompletních kopiích ve dvou oddělených lokalitách, kdy je zátěž rovnoměrně rozkládána do obou lokalit a v případě výpadku jedné z lokalit je druhá schopna zvládnout kompletní provoz.

U aplikačních a výdejových serverů je toto dosažitelné triviálně, u databázových serverů, které jsou centralizované, je použita některá z metod replikace databází. Systém CDN je v tomto částečnou výjimkou, kdy má být redundantní na úrovni vyšších celků.

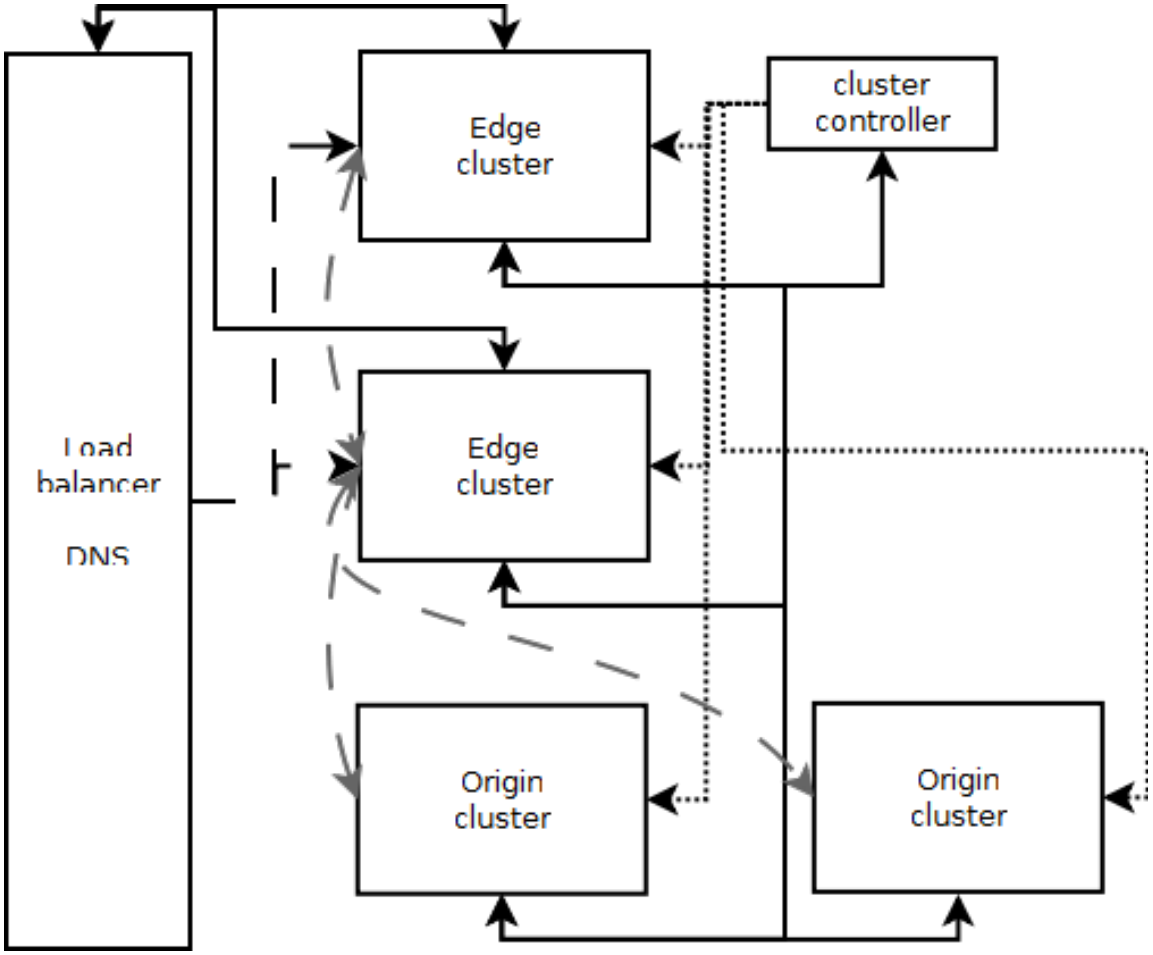
3.1 Přehled

Z hlediska klasifikace v předchozí kapitole jde o CDN druhé generace s push updaty a topologií klient-edge-origin. Inter-cluster vyhledávání dat probíhá directory-based s částečným použitím hashovacích mechanismů, inter-cluster je query-based. Update cache probíhá propagací, požadavky jsou směrovány složením proaktivního přepisu URL (ne ve smyslu přepisu URL při nahrávání do CDN, jelikož CDN není použita k ukládání html dokumentů, ale přiřazení vhodné URL nahrávanému souboru, viz dále) kombinovaného se správou DNS, loadbalancerem a http redirectů.

CDN seznamu se skládá z *clusterů* a *clustercontrolleru*.

Soubory jsou organizovány v hierarchické adresářové struktuře odpovídající URL *služba/hash/dataset/varianta/soubor*, kdy službě je předřazen typ, v současnosti *vod* pro video on demand a *live* pro živé přenosy. Jiné typy obsahu CDN neposkytuje. Živé přenosy vždy používají celou kapacitu CDN, tato práce se nadále bude zabývat pouze typem *vod*.

Každé službě Seznamu používající audiovizuální data (Stream, Novinky, Mixer, ...) je předem přiřazen určitý počet *hash*. *Hash* je zde umělou jednotkou umožňující správu dat v CDN na úrovni granularity mezi službou a datasetem a slouží pro směrování na určené servery v rámci url-rewrite směrování požadavků. Dataset je adresářem obsahujícím všechny soubory k určitému tématu, typicky všechny soubory jednoho videa. Podadresářem datasetu je varianta členěná podle formátu souborů, může obsahovat buď přímo soubory s videem různých kvalit nebo adresáře s jednotlivými chunky videa.



Obrázek 3.1: Schéma logických jednotek CDN. Plné čáry zobrazují databázová spojení, přerušované http požadavky, přerušované šedé http přesměrování a tečkované příkazová spojení. Příkazová spojení jsou stavová, chovají se jako fronta. Databázové propojení edge clusterů a loadbalanceru předává informace o přítomných hashích, databázové propojení mezi clustery slouží k shromažďování dat o zatížení CDN

3.1.1 Cluster controller

Cluster controller je centrálním řídícím prvkem CDN. Probíhá přes něj nahrávání souborů a správa CDN. Skládá se z databáze souborů, databáze stavů nahrávání, databáze topologie CDN, řídících algoritmů nahrávání, centrálního přístupu k záznamům a administračního rozhraní. Tato skupina serverů je jako jediná plně redundantní v rámci lokalit.

Samostatnou součástí cluster controlleru je také *watcher*. Úkolem této komponenty je sbírat informace z jednotlivých *cache serverů* a poskytovat je *mapping controllerům*.

3.1.2 Cluster

Cluster je skupina výdejových a obslužných serverů CDN. Skládá se z více výdejových serverů *cache*, *dispatcheru* pro přesměrování požadavků na data, *download controlleru* řídícího práci se soubory, *Miss controlleru*, *Clean controlleru* a *mapping controleru* pro práci s celky dat.

Clustery se dělí podle typu na *origin* a *edge*. Origin cluster slouží primárně k uchování veškerých platných souborů, edge cluster slouží k výdeji souborů.

Cache

Cache je jednotlivý výdejový stroj. Pro výdej slouží výkonný open source http server nginx [23], nahrávání souborů řeší démon *downloader*, který je řízen objektem *download controller*. Cache je identifikována v CDN jednoznačným číselným identifikátorem, na jeho základě má i veřejné doménové jméno. Na cache směřují také záznamy CNAME[1] ve formátu `hash.service.cdn-domain` pro prvotní směrování požadavků a vyvažování.

Dispatcher

Dispatcher řeší přesměrování na jinou cache, pokud požadovaný soubor není nalezen na *cache*, na kterou byl požadavek prvotně směrován. Údaje jsou získány z databáze souborů příslušné místnímu clusteru, případně je požadavek přesměrován na *dispatcher* jiného, sousedního, clusteru.

Download controller

Řeší nahrávání, update a změny souborů na základě oznámení z *cluster controlleru* a *mapping copntrolleru*. Podle dat lokální databáze mapování souborů a *hash* zadá úkol *downloaderům* jednotlivých *cache*. Vede databázi souborů *clusteru*.

Mapping controller

Mapping controller určuje, na kterých *cache* se mají nacházet které soubory s granularitou na úrovni *hashe* a prací na úrovni clusteru. Účelem této komponenty je vyvažovat a dynamicky optimalizovat využití prostředků jednotlivých *cache*. Vlastní přesuny dat pak obstará *download controller*, po dokončení přesunu dat jsou změny zaznamenány na pobočný DNS server CDN. Tato komponenta je v současnosti nevyužívaná, z provozních důvodů se neuvažuje o jejím využití ani v blízké budoucnosti. Veškeré mapování se bude dít ručně.

Spravuje databázi mapování *hash* - *cache* pro cluster.

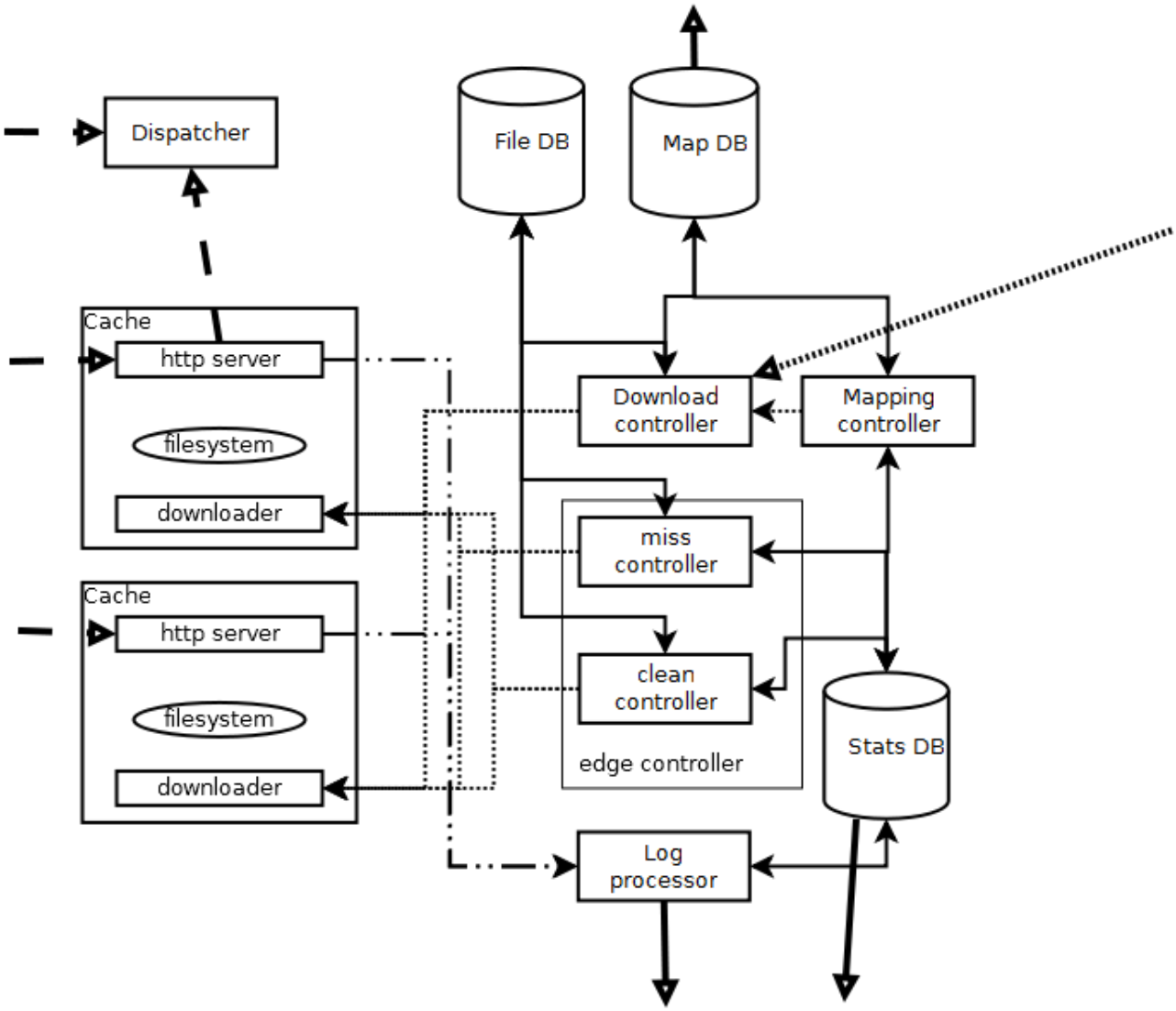
Miss a clean controller

Tyto controllery určují mapování souborů na cache s granularitou na úrovni souborů a pracují na úrovni cachí.

U edge clusterů nahrají na *cache* soubor, který je často z této konkrétní *cache* vyžadován a v případě nedostatku diskového prostoru smažou soubory, které nebyly z této konkrétní *cache* vydány nejdéle. Komponenty předávají příkazy přímo konkrétnímu *downloaderu* s asistencí *downloadcontrolleru*.

U origin *cache* nejsou tyto komponenty zapojeny.

[1] DNS záznam typu CNAME[15] - Canonical name record. Slouží pro přidělení více aliasů jednomu doménovému jménu, zde je použití opačné - alias patří více doménovým jménům



Obrázek 3.2: Schéma logických jednotek clusteru CDN. Plné čáry zobrazují databázová spojení, přerušované http požadavky, čerchované datové roury a tečkované příkazová spojení. Příkazová spojení jsou stavová, chovají se jako fronta. Každá komponenta má redundanci alespoň 2.

3.2 Výdej

Výdej souborů používá protokoly http, https případně jejich nadstavby nebo je používá jako zapouzdření. Šablona uri souboru v CDN je

`hash.služba.cdn.szn.cz/typ/služba/hash/dataset/varianta/nazev`

`hash.služba` je prvotní identifikátor a je použit k první úrovni směrování požadavků pomocí *proaktivního url rewrite*. Pro směrování požadavku na konkrétní *cache* se využívá DNS, kdy `hash.služba` je aliasem několika CNAME doménových jmen *cachí*. Předpokládá se, že dotazovaný DNS server odpoví takovou skupinou CNAME záznamů, která je vhodná, tj. je každý z odpovídajících edge serverů (obecně cache serverů, předpokládá se priorita edge serverů před origin servery) je blízký dotazujícímu se a není přetížený. DNS tak slouží pro loadbalancing. Z konkrétní *cache* je pak požadován soubor a pokud se nachází na dané *cache*, je vydán.

V případě, že se soubor na cache serveru nenachází, odpovědí je http redirect na dispatcher příslušný clusteru dané cache, který vrátí další http redirect podle záznamů v databázi sou borů clusteru. V případě, že je soubor nalezen na daném clusteru (directory-based způsob nalezení dat), vrátí dispatcher http přesměrování na jinou cache v clusteru, která soubor podle záznamů obsahuje.

V případě, že soubor není v clusteru nalezen, dispatcher přesměruje požadavek na dispatcher jiného clusteru, který je považován za vhodný, zaznamená do přesměrované url souboru parametr udávající vlastní cluster, na kterém nebyl soubor nalezen pro předejití cyklům.

V případě, že soubor není nalezen na žádném z vhodných edge clusterů, přesměrovává se na dispatcher origin clusteru, který je autoritativní v oblasti existence souborů. Pokud není soubor nalezen ani zde, CDN jej neobsahuje.

Přesměrování pomocí dispatcheru už míří přímo na doménová jména jednotlivých *cache* `cacheid.cdn-domain`, a algoritmus je bezestavový.

3.3 Nahrávání obsahu

Veškerý obsah je nahráván pomocí *cluster controlleru*, který vede databázi souborů celé CDN včetně zařazení v hierarchické struktuře spolu s časy nahrání a kontrolními součty souborů. Nahrávaný soubor je příslušný službě, má určený dataset a variantu. Ze služby a datasetu je vygenerován hash.

Nahrání souboru znamená zápis nových hodnot do databáze souborů a notifikaci download controllerů všech clusterů. Proces nahrání souboru je ukončen, jakmile je cluster controller notifikován origin clustery o úspěšném ukončení.

Download controller po notifikaci stažení zadává úkol do fronty downloaderů cachí, které mají mapován příslušný hash. Soubor může být nahrán přímo ze zdroje udaného cluster controlleru, nebo z některé z cachí, které obsahují soubor se daným kontrolním součtem. Lze tedy prohlásit, že nahrávání a update cachovaných souborů probíhá metodou propagací updatu.

Které soubory mají být cachovány je určeno dvouúrovňovým mechanismem: Umístění *hashů* určuje mapping controller. Jednotlivé soubory jsou, v případě že *hash* už je na *cache* mapován, nahrány výše zmíněným mechanismem propagací updatu a takové nahrávání souborů lze prohlásit za metodu kooperativního pushování. Obdobným způsobem jsou soubory odstraňovány.

Soubory, které jsou na byly nahrány do CDN před přidáním hashe na cache jsou na cache nahrány až pokud jsou z ní častěji vyžadovány, a to pomocí miss controlleru. Obdobně mohou být soubory, ke kterým již není přistupováno, odstraněny. Toto chování odpovídá nekooperativnímu pull nahrávání.

V případě přidání *hashe* na *origin cache* jsou z jiného origin serveru zkopírovány všechny soubory *hashe* hromadně.

3.4 Požadavky

Vzhledem k architektuře spolu se základními algoritmy CDN Seznamu, které jsou dané, zejména pak postupu vyhledání lokality souboru pomocí DNS, je pro snížení latence a snížení zatížení dispatcherů a interního provozu v CDN vhodné udržovat počet případů, kdy nedojde k nalezení požadovaného souboru na *cache*, která má mapován *hash* konkrétního souboru, minimální, optimálně nulový. V případě, že už takový případ nastane, mělo by být přesměrování lokální v rámci clusteru.

Vzhledem k latenci při přístupu k diskovému poli by soubory měly být vydávány z paměti. Tento požadavek dále komplikuje rozložení dat, je nevhodné mít mapovány soubory, respektive *hash* na více *cache* než je pro výdej nezbytné, přístupy k těmto souborům pak snižují efektivitu vyrovnávacích pamětí.

Zdánlivě přímým protikladem tohoto požadavku je odolnost vůči flash crowds [4] a redundance výdeje.

Jakýkoli algoritmus rozmisťování dat tak musí pracovat primárně na úrovni mapping controlleru. U edge clusterů pak z důvodů optimalizace využití diskového a paměťového prostoru připadá v úvahu funkce rozmisťování souborů také na úrovni miss a clean controllerů, které ovšem vzhledem k tomu, že fungují lokálně k vyvažování v clusteru v rámci jednotlivých *cache*, mají omezené využití pro složitější funkce než udržování právě vydávaných souborů na jednotlivých *cache*. I v případě odstranění tohoto omezení by zásadní změny chování byly v přímém rozporu s požadavkem na nalezení souboru na *cache*, která má *hash* mapován.

Dalším požadavkem je minimalizace dat vydávaných z origin úložiště.

Kapitola 4

Parametry a zatížení CDN

Vzhledem k nemožnosti uveřejnění skutečných dat, logů a podrobných parametrů CDN musely být tyto předzpracovány. Z konfigurace systému jsou uvedeny jen nejnutnější data, logy jsou zpracovány až na úroveň souhrnných statistik.

4.1 Parametry serverů

Pro výkon systému jsou limitující *cache* servery. Existuje celkem 8 origin serverů ve dvou clusterech a 2 edge servery ve dvou clusterch, jejich počet bude ale brzy navýšen. Základní parametry *cache* uvádí tabulka 4.1.

Tabulka 4.1: Tabulka 4.1: Parametry cache

	origin	edge
velikost diskového pole	20 TB	20 TB
velikost paměti	64 GB	64 GB
síť	1 GbE	10 GbE

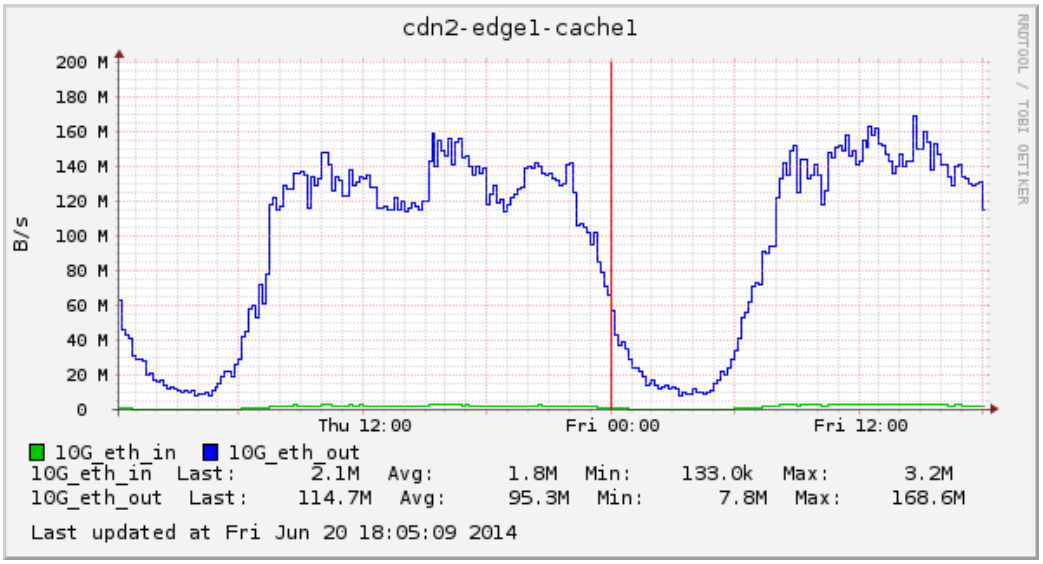
Pro navýšení kapacity výdeje mají 2 cache servery z každého origin clusteru v současnosti 10 gigabitové síťové rozhraní a loadbalancer s DNS jsou konfigurovány tak, aby na ně předávaly požadavky. Toto řešení je dočasné, origin servery nemají být prvními, ze kterých jsou požadována data. Tato síťová rozhraní budou použita pro nové edge servery v odpovídajících clusterech v pozdější fázi zprovoznění CDN, kdy bude množství dat v CDN převyšovat kapacitu origin cache a bude nutné použít nějakou formu distribuce dat.

Počet origin cache pak bude dále postupně navyšován se vzrůstajícími požadavky na úložný prostor, počet edge cache bude navyšován s vzrůstajícími požadavky na propustnost. V nejbližší době půjde zejména o vytvoření dalších edge cache, které zastoupí současné origin cache používané pro výdej.

4.2 Typické zatížení

Typické denní zatížení serverů (edge a origin) výdejem data uvádí obrázky 4.1 a 4.2. Typické zatížení se zásadně neliší v průběhu týdne. Podle záznamů celý systém CDN vydává v současnosti ve špičkách 1.7 GB/s při 153 požadavcích za sekundu. Pro jednotlivé výdejové cache je to pak 362 MB/s a 50 požadavků za sekundu, tyto hodnoty jsou bezpečně nižší než teoretický limit výdeje.

Podrobnější přehled o výdeji dat lze získat ze záznamů jednotlivých edge serverů. U jednotlivých přenosů dat je uveden čas dokončení přenosu, množství přenesených dat, přenášený soubor, cílová IP adresa, velikost přenášeného souboru a číslo http odpovědi. Získaná data byla použita pro zvážení možných optimalizací distribučních algoritmů a k jednomu z testů distribučních algoritmů a jsou zobrazena v grafech 4.4, 4.6, 4.3 a 4.5.



Obrázek 4.1: Typické denní zatížení edge cache. Je pozorovatelné velmi rovnoměrné zatížení mezi 6. a 20. hodinou s útlumem mezi nimi.

Vzhledem latenci výdeje může být zajímavým parametrem, jaká část vydávaných dat se nachází v paměti. Tento údaj není sledován a není ani dohledatelný, lze jej odhadnout z logů pomocí jednoduchého scriptu. Lze předpokládat, že jsou často čtená data z disku udržována v operační paměti mechanismy operačního systému a pro tyto účely je využito až 75 % z celkové kapacity paměti. Pak modelujeme mechanismus udržování vyrovnávací paměti obyčejnou frontou, tedy po naplnění určeného množství vyrovnávací paměti jsou z ní uvolněna data nahraná nejdříve. Tento přístup je zvolen pro svou snadnost implementace, předvídatelnost a jelikož výsledek pak lze považovat za dolní hranici úspěšnosti vyrovnávací paměti.

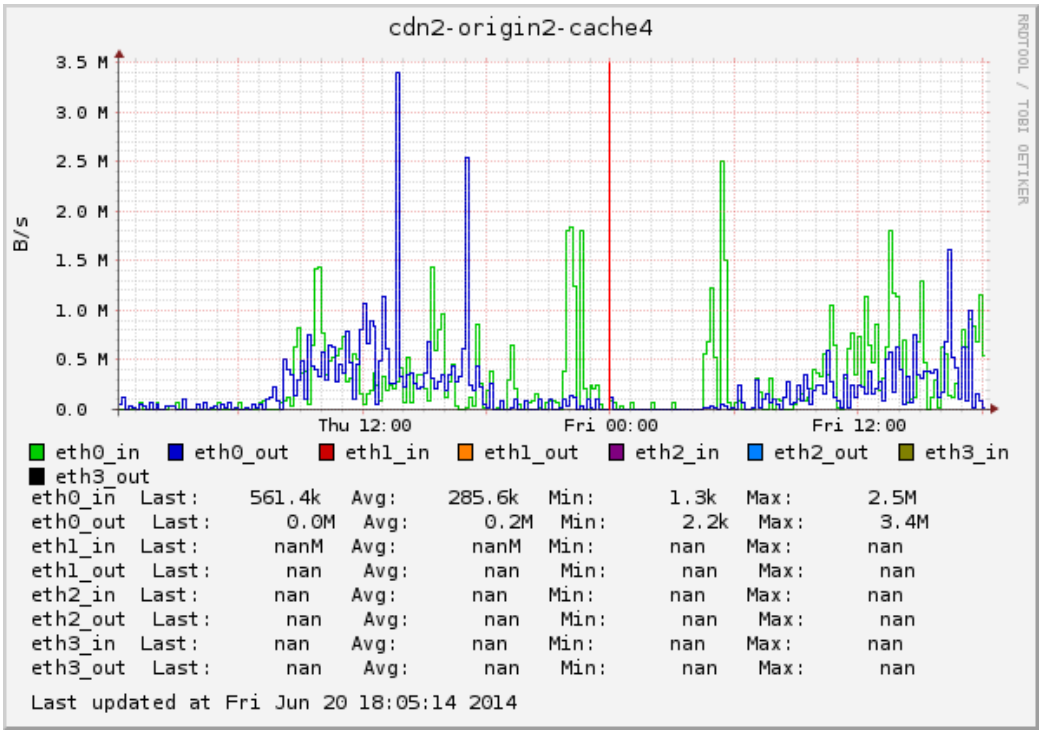
Výsledkem při použití tří čtvrtin paměti jednotlivých cache a správě cache FIFO algoritmem je průměrná úspěšnost 92 % (viz graf 4.7).

Optimální rozložení požadavků je možné odhadnout tak, že vyrovnávací paměť výdejových strojů bude společná. Při optimálním rozložení požadavků na servery, na kterých se požadovaný soubor nachází v paměti, by tak průměrná úspěšnost mohla dosáhnout 99 % (viz graf 4.8). Toto je sice značné zlepšení, ale vzhledem k použité velmi nepřesné metodě odhadu bude reálné zlepšení značně menší.

Na cache je využito cca 80 % diskového prostoru při průměrném denním přírůstku o 0,2 procentního bodu, již brzy tak dojde k vyčerpání základní kapacity systému CDN.

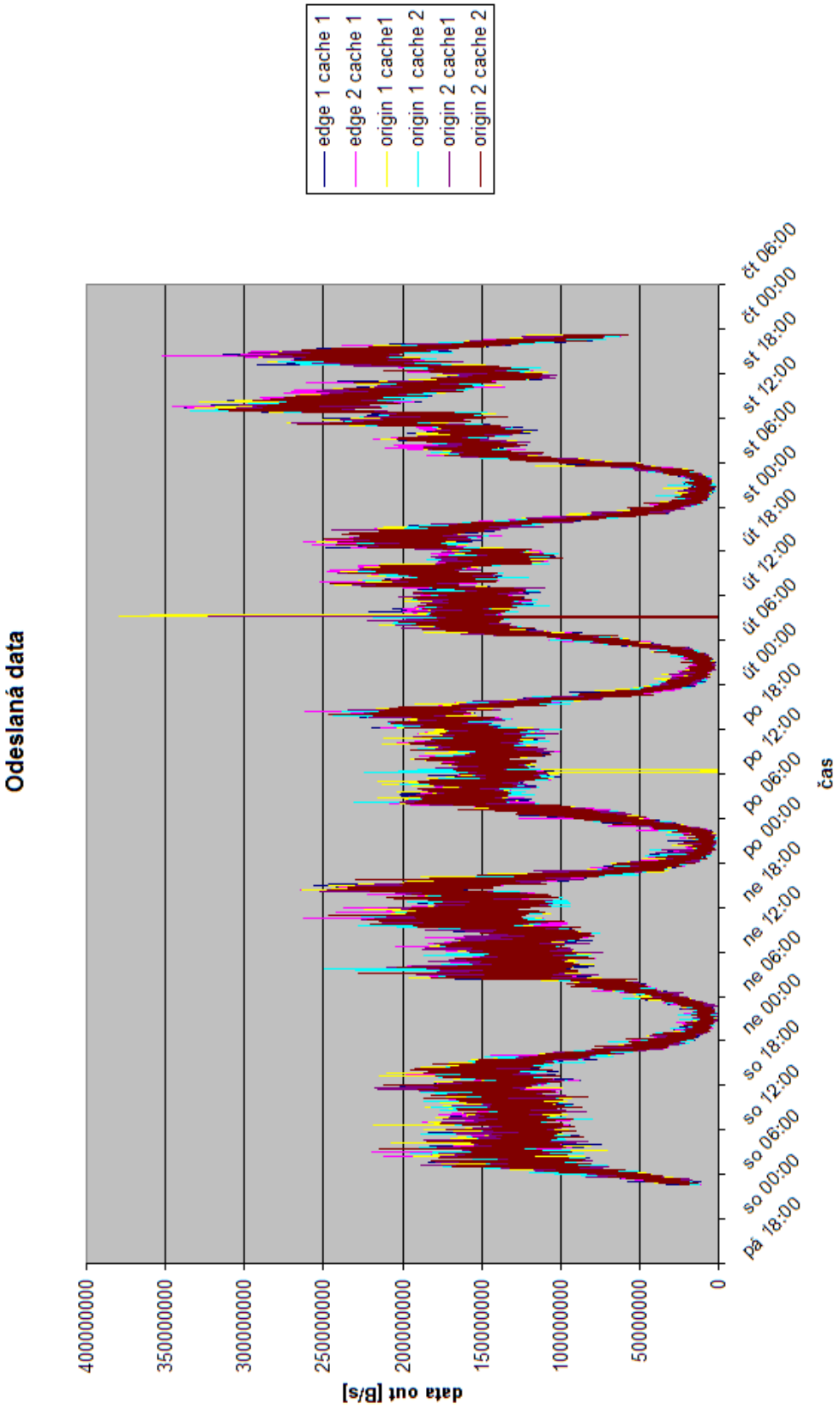
4.3 Limity současného stavu

Výdej je v současnosti limitován prakticky pouze síťovým rozhraním. Nejbližším dalším limitem je velikost RAM a dále propustnost dat do lokality.

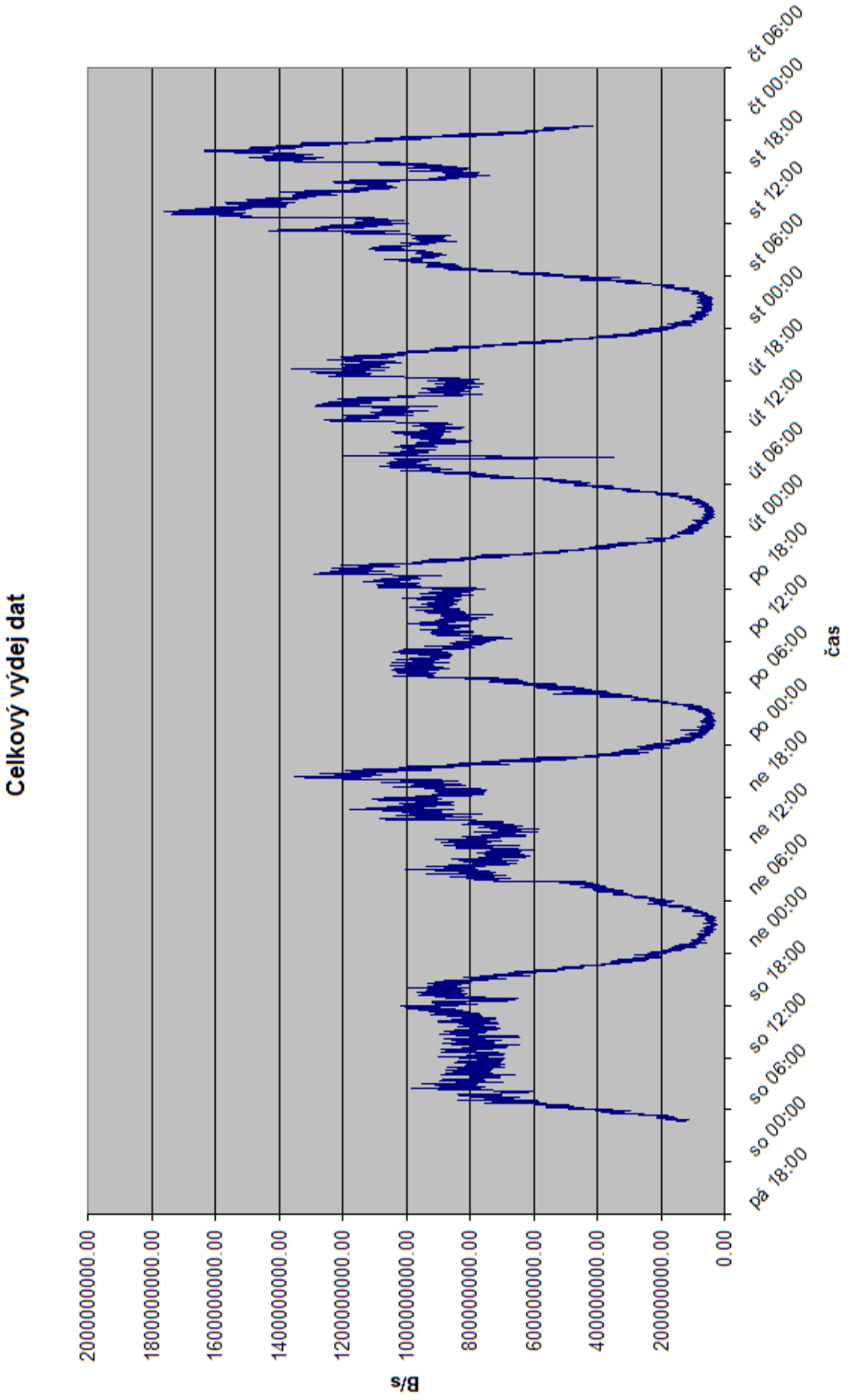


Obrázek 4.2: Typické denní zatížení origin cache. Veškerý provoz souvisí pouze s udržováním konzistence dat.

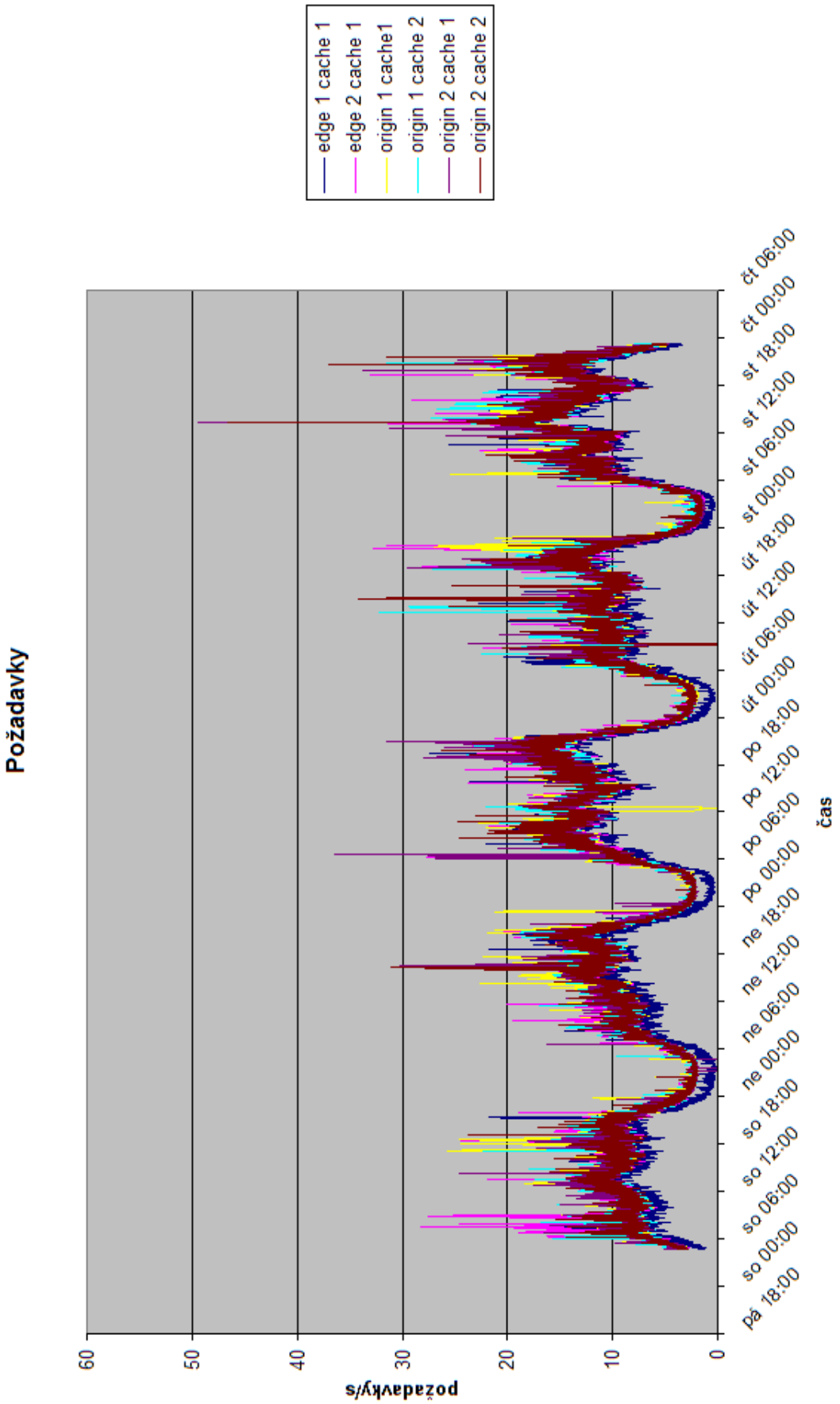
Kapacita je omezena velikostí nejmenší origin cache. Po naplnění budou data na jednotlivé origin cache manuálně rozděleny podle jednotlivých služeb Seznamu.



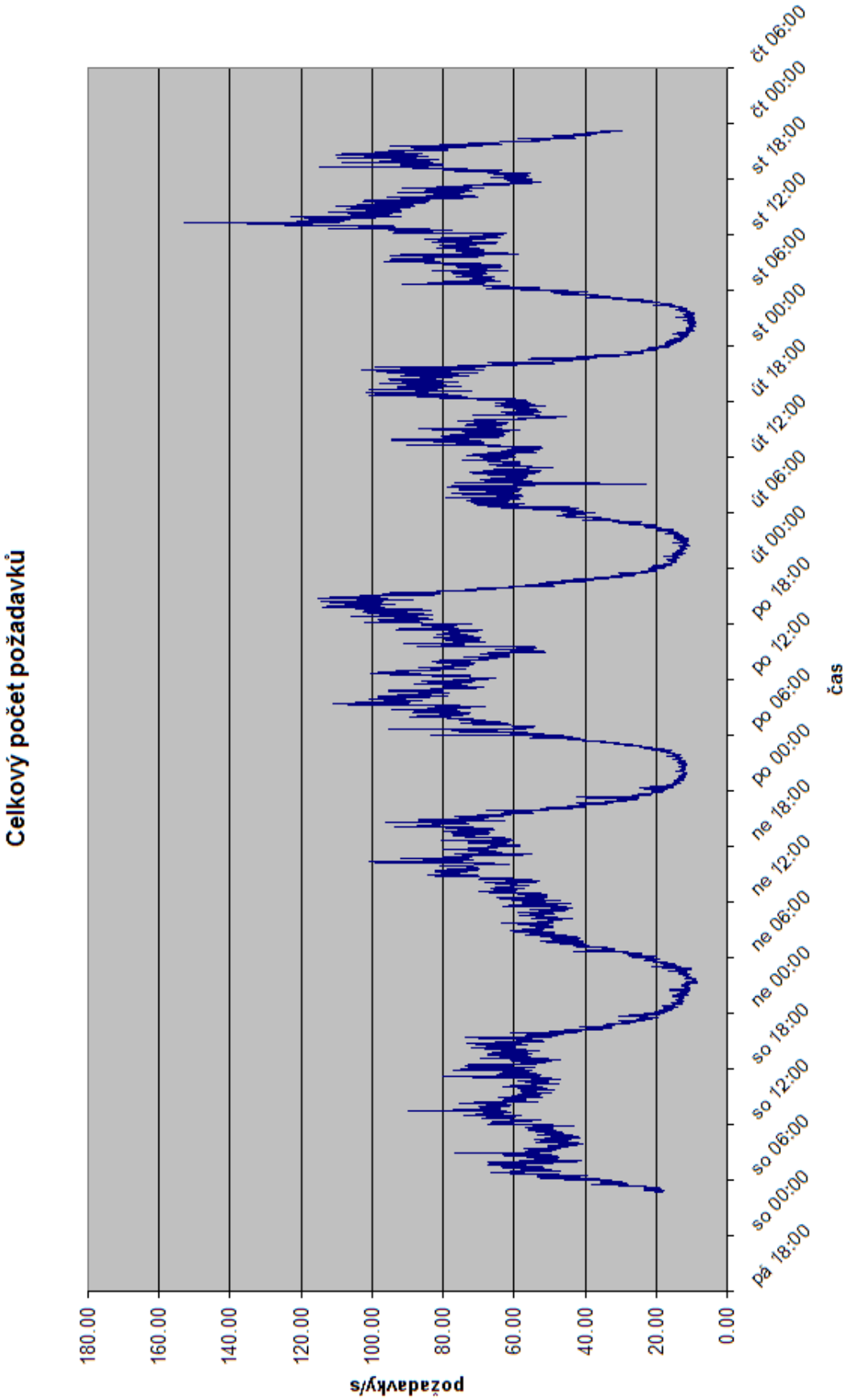
Obrázek 4.3: Vydaná data z jednotlivých výdejových cachí (neděle - čtvrtek), průměrné hodnoty v minutových intervalech



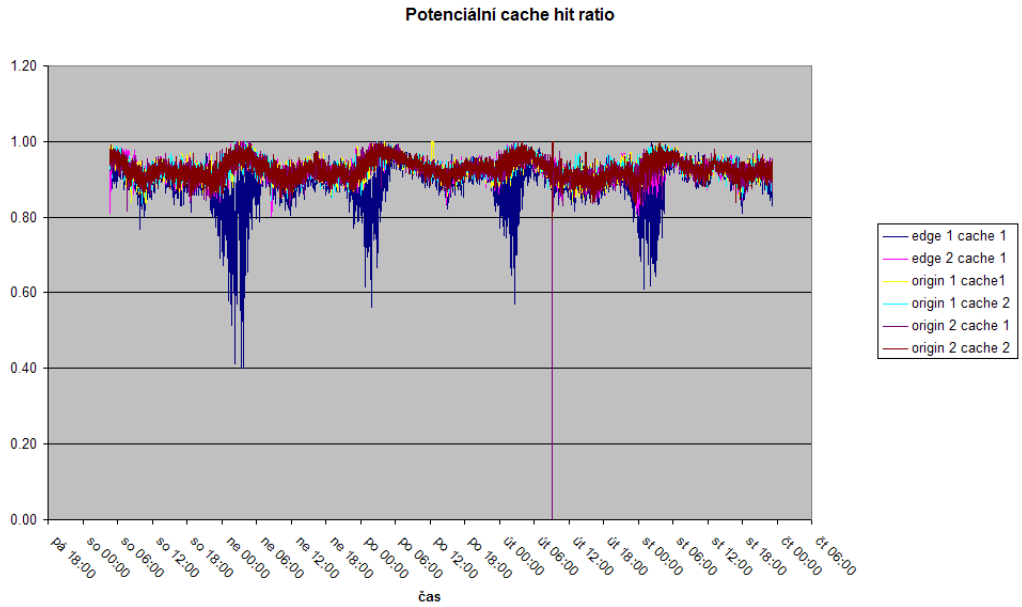
Obrázek 4.4: Vydaná data z celé CDN (neděle - čtvrtek), průměrné hodnoty v minutových intervalech



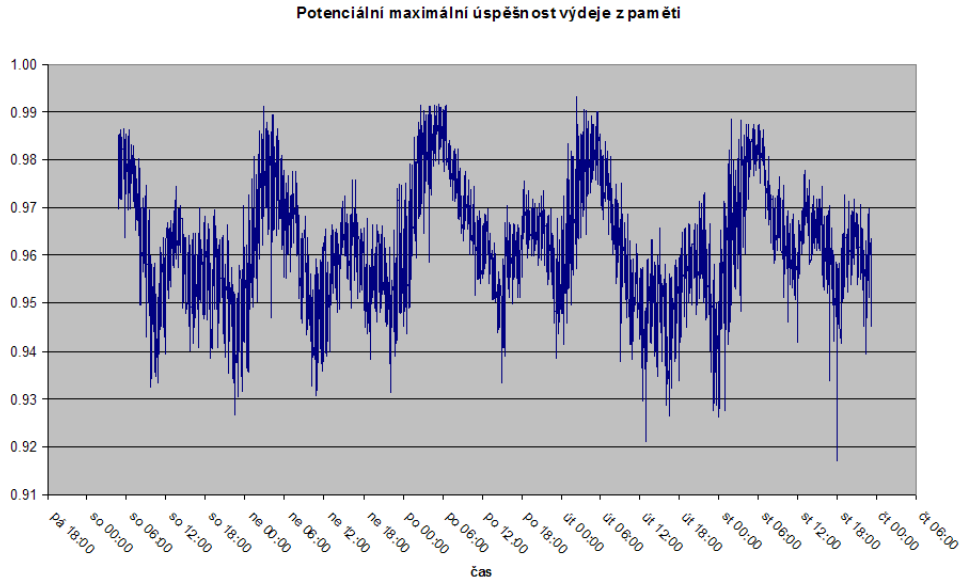
Obrázek 4.5: Požadavky na jednotlivé výdejové cache (neděle - čtvrtek), průměrné hodnoty v minutových intervalech



Obrázek 4.6: Požadavky na celou CDN (neděle - čtvrtek), průměrné hodnoty v minutových intervalech



Obrázek 4.7: Úspěšnost vydání požadavku bez přístupu na disk při využití 75 % RAM pro vyrovnávací paměť čtení dat (neděle - čtvrtek), průměrné hodnoty v minutových intervalech



Obrázek 4.8: Úspěšnost vydání požadavku bez přístupu na disk při využití 75 % RAM pro vyrovnávací paměť čtení dat (neděle - čtvrtek), průměrné hodnoty v minutových intervalech, při společné vyrovnávací paměti odpovídající potenciální horní hranici úspěšnosti při optimálním rozložení požadavků na cache.

Kapitola 5

Simulátor CDN

5.1 Důvody simulace

Přestože existují akademické implementace CDN (CoDeeN, Coral, Globule,...), nejsou vhodné pro experimenty s algoritmy distribuce obsahu, ať už kvůli složitosti změn systémů, nereprodukovatelnosti výsledků ([24]) nebo architektuře vzdálené od Seznam CDN. Seznam CDN pro experimenty nelze použít především z licenčních důvodů.

Bylo navrženo několik simulátorů CDN ([8, 6, 32]), ale pro účely testování mapovacích a cachovacích algoritmů nejsou vhodné. ([24]). Ve zmíněné publikaci navržený CDNsim je pro použití nevhodný: zbytná komplexita, zaměření na globální CDN, paradigma cachování x replikace, především neudržovaný a v současnosti nezkompilovatelný.

Proto byl vytvořen nový, omezený simulátor CDN inspirovaný architekturou Seznam CDN.

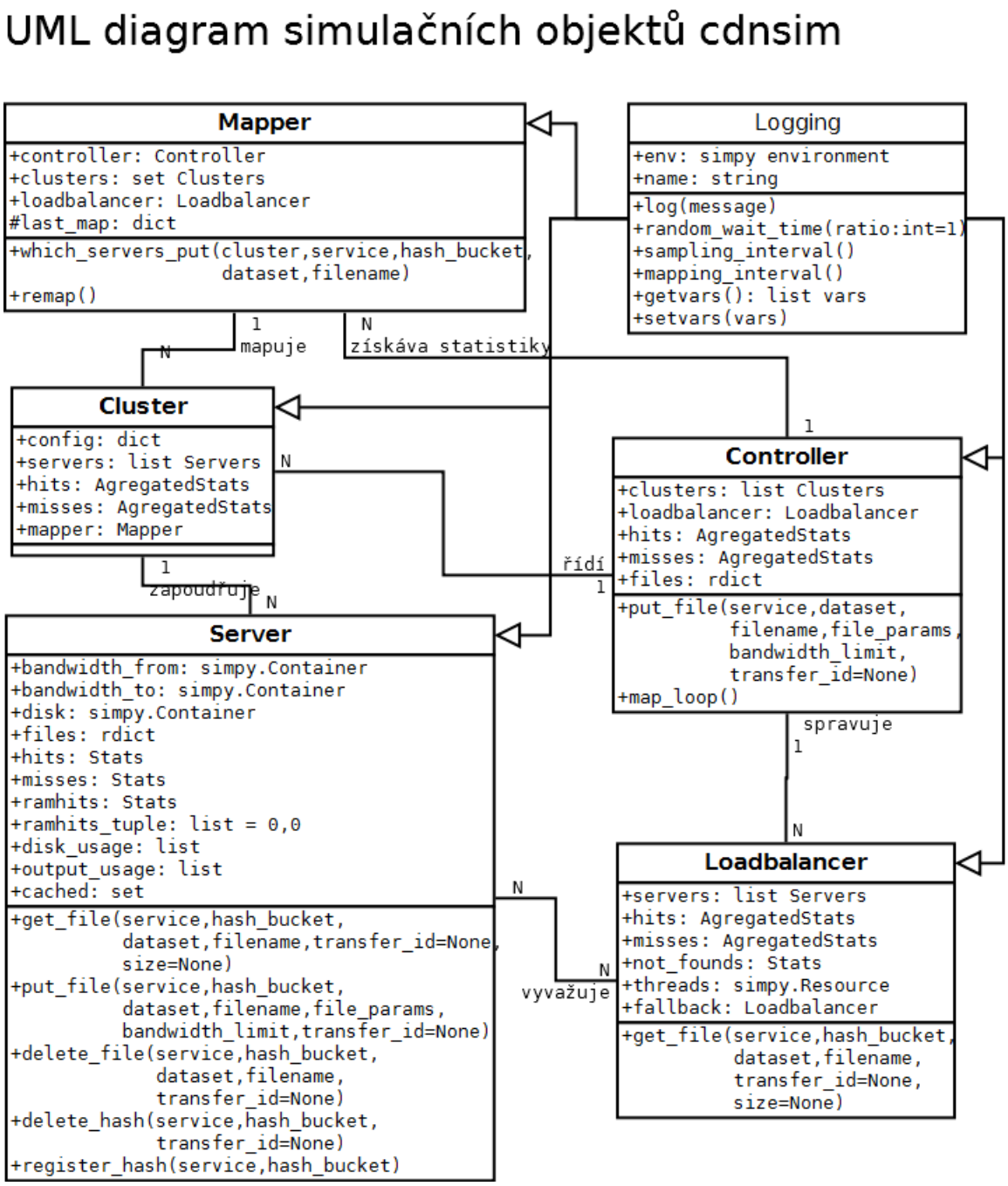
5.2 Provedení

Simulátor CDN je vytvořen v jazyce python pomocí knihovny SimPy [16]. SimPy je knihovna pro diskrétní simulace založená na procesním paradigmatu s next-event řízením. Proces je v SimPy reprezentován generátorovou funkcí. SimPy pak poskytuje modely sdílených prostředků a kalendář. Zápis procesů pomocí generátorů umožňuje vytváření složitých procesů a simulace je tak blízká agentní simulaci.

Pro žádnou část simulace vyjma jádra mapovacího algoritmu není možno z licenčních důvodů použít skutečný kód Seznam CDN. Simulátor není specializovaný na Seznam CDN, byla snaha o obecnější implementaci, avšak vzhledem k nutnosti zachování adresního schématu 3.2 a logiky sdružování souborů do celků 3.1.2 simulátor neumožňuje použít jiné uspořádání souborů, než seznam CDN. Vzhledem k nemožnosti ověřit a změnit síťovou infrastrukturu tato není předmětem simulace. Objekty simulátoru jsou Server, Cluster, Controller, Loadbalancer a Mapper.

5.3 Objekty simulace

Základním objektem simulátoru je Server, zapouzdřující generátory `get_file` a `put_file` odpovídající pokusu o stažení a nahrání souboru. Server obsahuje prostředky odpovídající paměti, disku a dvěma směrům síťové komunikace. Dále server obsahuje slovník souborů, které jsou na něm uloženy a objekty statistiky.



Obrázek 5.1: UML diagram simulačních objektů cdnsim. Všechny objekty dědí společné logovací rozhraní Logging

Mapper je objekt obsahující funkce pro určení umístění nových souborů na konkrétní Server (`which_servers_put`), případně přesunování souborů mezi Servery (`remap`). Funkce tohoto objektu nepoužívají simulační čas.

Několik Serverů, Mapper a statistiky zapouzdřuje Cluster. Slouží především pro organizaci prostředí a dále pro sběr statistik.

Objekt LoadBalancer slouží k simulaci vyvažování požadavků na jednotlivé servery. Má přístup k proměnným serverů a odkaz na případný další LoadBalancer. Metoda `get_file` je generátorem, který simuluje požadavek klienta na soubor, vyhledání serveru a přesměrování na konkrétní server pomocí DNS a loadbalanceru a případné další přesměrování v rámci CDN. Na Serveru obsahujícím požadované údaje spustí LoadBalencer proces stažení souboru.

Controller je centrálním objektem správy souborů. Zapouzdřuje Clustery a statistiky. Opakovaně spouští proces `remap` Mapperů jednotlivých Clusterů. Dále metoda `put_file` spouští procesy `put_file` na Serverech spravovaných Clusterů vybraných pomocí Mapperů.

Vztahy jednotlivých objektů ilustruje obrázek 5.1

5.4 Pomocné objekty simulátoru

Prvním pomocným objektem je Logging, který všechny objekty simulace dědí. Objekt vytváří jednotné rozhraní pro záznam událostí dekorovaných simulačním časem a popisem objektu. Objekt také obsahuje centrální proměnnou simulace environment.

Další skupinou objektů je hierarchie tříd Stats sloužící pro zápis jednotlivých událostí na souborech zařazených do stromové struktury výdeje v simulačním čase, přístup ke statistikám záznamu a agregaci těchto záznamů.

Objekt Environment obsahuje kompletní definice simulačního prostředí včetně inicializační funkce prostředí. Umožňuje ukládat a načítat stav simulaci. Ukládání a načítání simulace je však omezeno implementací simpy, které závisí na generátorech pythonu, jejichž stav je obtížně reprodukovatelný a vytvoření obecné struktury generátorů simulačních procesů by bylo nad rozsah této práce. Je tedy možné uložit simulaci pouze ve stavu, kdy neběží žádný proces. Toto je postačující pro uložení výchozího stavu simulace.

5.5 Vztah simulovaných objektů k Seznam CDN

Server odpovídá cache spolu s downloaderem a downloadcontrollerem, Loadbalancer reprezentuje loadbalancer, dns a dispatcher. Cluster je organizační jednotkou odpovídající clusteru u Seznam CDN. Mapper zastupuje mapping controller, miss controller a clean controler.

Controller kromě funkcí cluster controlleru zajišťuje pravidelné spouštění Mapperu, tedy simuluje i určitou část funkčnosti komponent clusteru.

Funkce watcheru a logprocessoru jsou distribuovány po jednotlivých objektech simulace pomocí objektů Stats.

5.6 Příprava dat pro simulátor

Pro přípravu dat pro simulátor existují utility `plan_builder`, `plan_generator` a `pseudonymizer`. `Plan_builder` slouží k vytvoření simulačního plánu ze souborů logu (argumenty jsou název prostředí, název plánu a anonymizované soubory logu). `Plan_generator` slouží k vytváření

náhodných plánů a konečně `pseudonymizer` slouží k odstranění IP adres ze záznamů z provozu a k vytvoření struktury souborů z provozu.

5.7 Požadavky pro běh

Simulátor a pomocné utility pro běh vyžaduje python verze 2.7 (s verzí 3.x není kompatibilní) a dále prostředí simpy v aktuální verzi. Program byl testován na stroji s operačním systémem Debian 7 a balíky python 2.7.5 a simpy 3.0.5.

Kapitola 6

Simulace současného stavu

6.1 Stav

Pro simulaci současného stavu byly zvoleny parametry mírně odlišné od provozního prostředí, respektive předpokládané provozní prostředí po plném oddělení origin a edge serverů.

Testovací prostředí se skládá ze 2 origin a 2 edge clusterů, kde origin clustery mají 4 cache a edge clustery 3 cache. Servery origin clusterů mají všechny síťové rozhraní 1 GbE, edge clustery 10 GbE. Další parametry odpovídají provoznímu prostředí. Požadavky jsou rovnoměrně rozdělovány mezi edge servery, které mají mapován *hash* ve kterém se soubor nachází.

Prostředí je popsáno pomocí funkce `basic` v souboru `environments.py` takto:

```
def basic(initial_time=0):
    origin_bandwidth = 120000000 # Gb/s
    edge_bandwidth = 1200000000 # 10Gb/s
    origin_storage = int(2E13) # cca 20TB
    edge_storage = int(2E13) # cca 20TB
    ram_cache = int(5E10) #cca 50GB

    e = simpy.Environment(initial_time)

    servers = dict(
        origin_a=[
            cdnsim.server.Server("originA1", e, origin_bandwidth,
                                 origin_storage, ram_cache),
            cdnsim.server.Server("originA2", e, origin_bandwidth,
                                 origin_storage, ram_cache),
            cdnsim.server.Server("originA3", e, origin_bandwidth,
                                 origin_storage, ram_cache),
            cdnsim.server.Server("originA4", e, origin_bandwidth,
                                 origin_storage, ram_cache),
            ],
        origin_b=[
            cdnsim.server.Server("originB1", e, origin_bandwidth,
                                 origin_storage, ram_cache),
            cdnsim.server.Server("originB2", e, origin_bandwidth,
```

```
                            origin_storage, ram_cache),
        cdnsim.server.Server("originB3", e, origin_bandwidth,
                            origin_storage, ram_cache),
        cdnsim.server.Server("originB4", e, origin_bandwidth,
                            origin_storage, ram_cache),
        ],
    edge_a=[
        cdnsim.server.Server("edgeA1", e, edge_bandwidth,
                            edge_storage, ram_cache),
        cdnsim.server.Server("edgeA2", e, edge_bandwidth,
                            edge_storage, ram_cache),
        cdnsim.server.Server("edgeA3", e, edge_bandwidth,
                            edge_storage, ram_cache),
        ],
    edge_b=[
        cdnsim.server.Server("edgeB1", e, edge_bandwidth,
                            edge_storage, ram_cache),
        cdnsim.server.Server("edgeB2", e, edge_bandwidth,
                            edge_storage, ram_cache),
        cdnsim.server.Server("edgeB3", e, edge_bandwidth,
                            edge_storage, ram_cache),
        ],
    )
lb = cdnsim.loadbalancer.LoadBalancer("alfa", e,
           servers['edge_a'] + servers['edge_b'], 5)
lb2 = cdnsim.loadbalancer.LoadBalancer("beta", e,
           servers['origin_a'] + servers['origin_b'], 5)
lb.fallback = lb2

cluster_oa = cdnsim.controller.Cluster("originA", e, "origin",
             servers['origin_a'],
             cdnsim.mappers.BasicMapper("originA", e, lb2))
cluster_ob = cdnsim.controller.Cluster("originB", e, "origin",
             servers['origin_b'],
             cdnsim.mappers.BasicMapper("originB", e, lb2))
cluster_ea = cdnsim.controller.Cluster("edgeA", e, "edge",
             servers['edge_a'],
             cdnsim.mappers.BasicMapper("edgeA", e, lb))
cluster_eb = cdnsim.controller.Cluster("edgeB", e, "edge",
             servers['edge_b'],
             cdnsim.mappers.BasicMapper("edgeB", e, lb))
ctrl = cdnsim.controller.Controller("CC", e,
                [cluster_oa, cluster_ob, cluster_ea, cluster_eb],
                (lb, lb2))

return e, ctrl, (lb, lb2), servers,
```

6.2 Vstupy

Pro simulaci současného stavu byla zvolena 1 denní anonymizovaná zátěž, jde o počátek dat získaných z logů v popisu CDN Seznamu (v kapitole 3). Byly použity časy a adresy požadavků, rozdělení na jednotlivé cache je provedeno logikou simulovaného loadbalanceru.

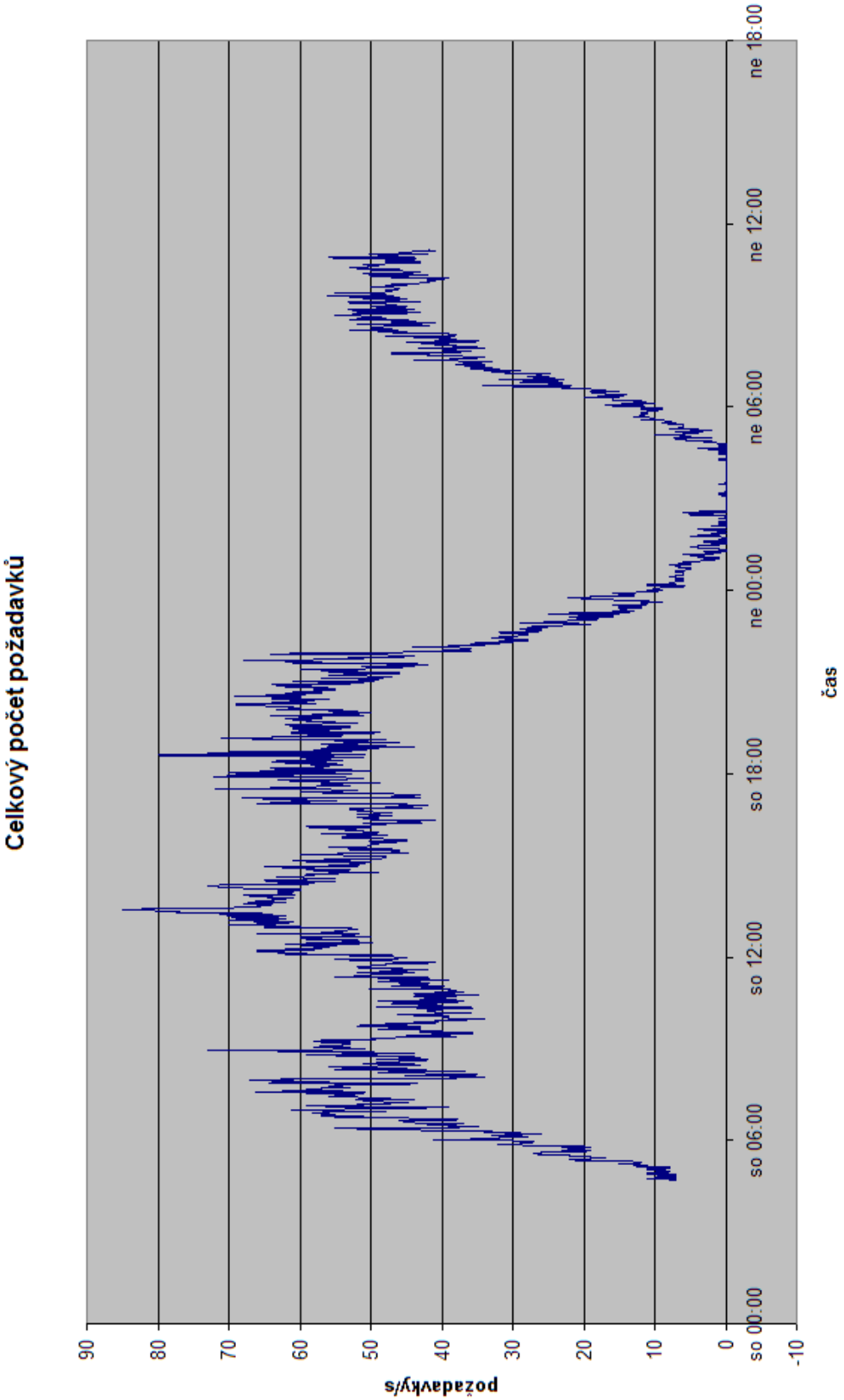
Do simulovaného prostředí byly vloženy pouze soubory, na které bylo přistupováno v daném období. To nemá vliv na výsledky simulace, jejím cílem není testování přesunů dat, pouze zjištění základního rozložení přístupů k souborům v RAM a ověření funkce simulátoru. Soubory byly v souladu se současným stavem vloženy na všechny cache (jak origin, tak edge).

Prostředí s uloženými soubory je uloženo v souboru `days.env`, plán požadavků v souboru `days.pln`.

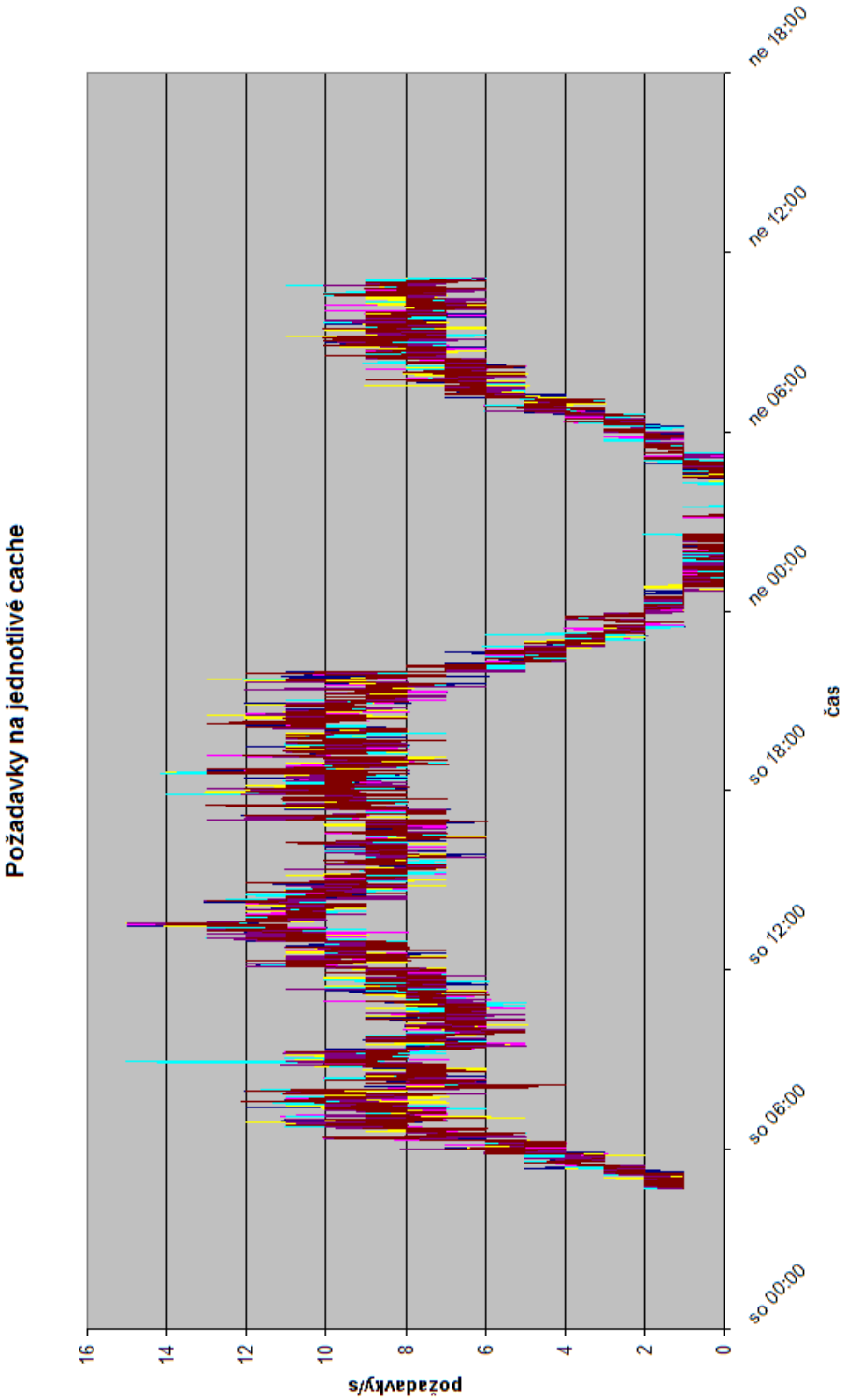
Z uri souboru je pro simulaci použita služba, dataset a filename, varianty a typy simulované prostředí nepoužívá, hash je vypočten metodami simulovaného prostředí.

6.3 Testy základního algoritmu

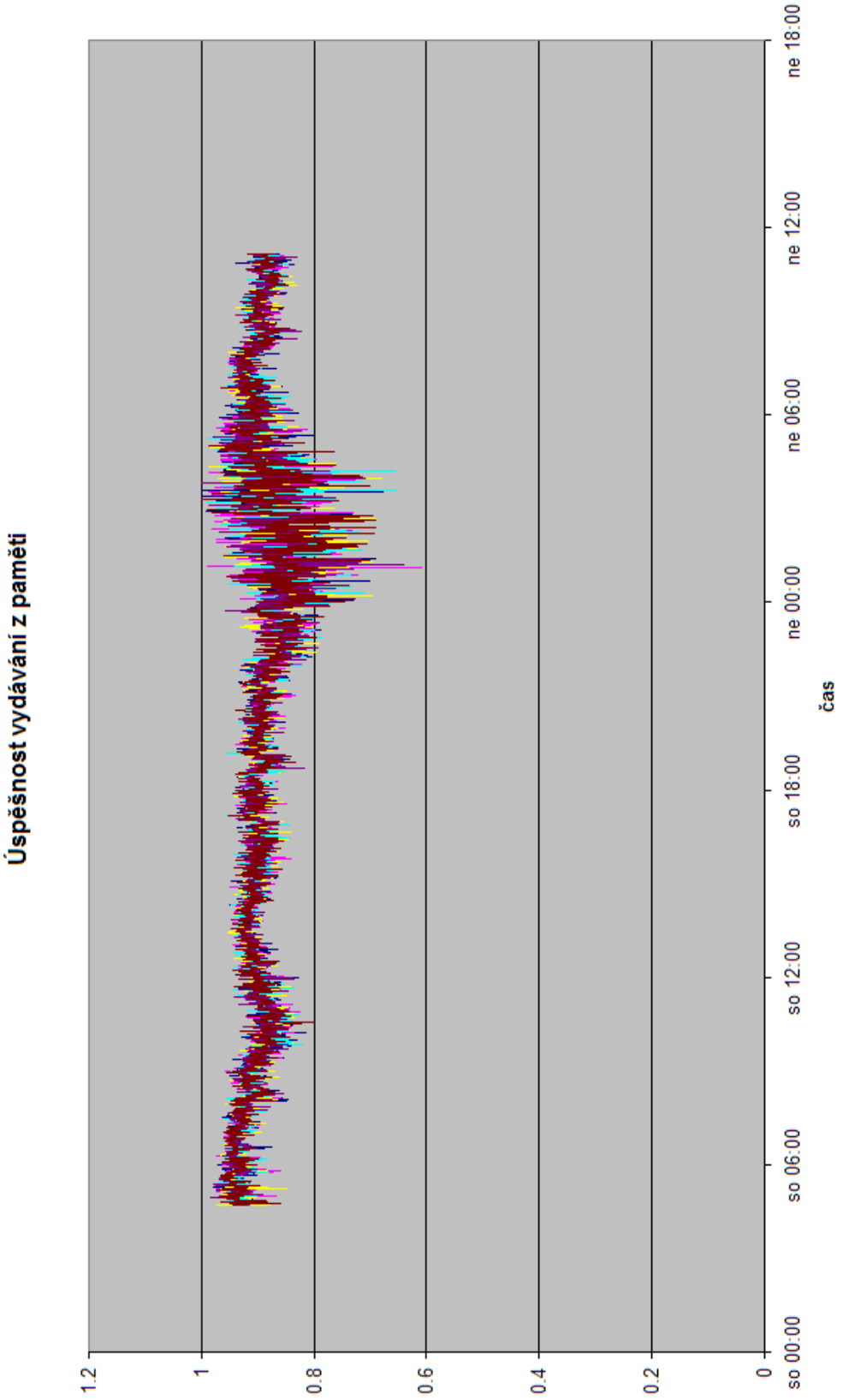
Z výstupů testu byly získány výstupy odpovídající výstupům z provozu (požadavky viz 6.1) s výjimkou lepší práce loadbalanceru, kdy nejsou tak zřetelné výkyvy v zatížení jednotlivých cache (viz obrázek 6.2 oproti 4.5). Míra vydávání z paměti odpovídá předpokládané (obrázek 6.3 oproti 4.7). Data nejsou porovnávána lepší metrikou, jelikož pro ověření simulace podobnost postačuje.



Obrázek 6.1: Simulované denní zatížení CDN. Výstup z zásadě odpovídá výstupu reálného systému (viz 4.6)



Obrázek 6.2: Simulované denní zatížení jednotlivých cache. Oproti skutečnému provozu (viz 4.5) je pozorovatelná lepší funkce loadbalaceru, kdy rozptyl množství požadavků na jednotlivé cache je mnohem menší. Simulátor nemoduluje loadbalacer věrně, používá poněkud idealizovaný model bez zpoždění propagace dat o zatížení jednotlivých cache



Obrázek 6.3: Část požadavků obsloužená při výdeji z paměti cache. Oproti skutečnému stavu opět menší rozptyl způsobený především jiným rozložením zátěže vzhledem k velmi zjednodušenému loadbalanceru.

Kapitola 7

Algoritmy distribuce dat

Cílem algoritmu pro distribuci dat, nebo také mapovacího algoritmu, je určit které soubory mají být vydávány z kterých serverů. To můžeme popsat *plnou mapovací maticí*

$$\mathbf{F} = \begin{matrix} m_{1,1} & \dots & m_{i,1} \\ \vdots & \ddots & \\ m_{1,j} & \dots & m_{i,j} \end{matrix} \tag{7.1}$$

kde $m_{i,j} = 1$, pokud je soubor s indexem i přítomný na cachi označené j. Použití této matice by umožnilo velmi jemné řízení přístupu. Její použití pro řízení rozmístění souborů je ovšem z kapacitních důvodů nevhodné a celkové řízení rozmístění musí být řešeno jinak než centrálně nebo na jiné úrovni granularity než jednotlivé soubory.

Lze například použít *hashovou mapovací matici* $\mathbf{M}$, kde $m_{i,j} = 1$, pokud je *hash* souborů s indexem i přítomný na cache j. Existence jednotlivých souborů *hashe* na cache pak bude určena jinak. Tato kapitola popisuje tento přístup, kdy mapovací matice nebo nějaká její reprezentace udává umístění hashů souborů a existenci jednotlivých souborů řeší lokálně cachovací mechanismy.

Množství dat na jednotlivých cache jako sloupcový vektor pak lze získat (za předpokladu nahrání všech souborů):

$$\mathbf{MS}^T \tag{7.2}$$

kde $\mathbf{S}^T$ je sloupcový vektor velikostí dat obsazených jednotlivými hashi.

Vektor maximální přípustné zátěže hashe získáme:

$$\mathbf{B_{hashmax}} = \mathbf{B_{cache}}^T\mathbf{M} \tag{7.3}$$

kde $\mathbf{B_{hashmax}}$ je maximální možná zátěž hashe jako řádkový vektor, pokud nebudou zatíženy ostatní, a $\mathbf{B_{cache}}^T$ je sloupcový vektor maximálního přípustného zatížení cache.

Průměrné rozložení zátěže hashů na cache získáme:

$$\mathbf{B_{parst}} = mask(\mathbf{B_{cache}}ii(\mathbf{B_{hashmax}}), \mathbf{M}) \tag{7.4}$$

kde $mask$ je operace součinu po jednotlivých prvcích, a ii je funkce, která vytvoří matici, jejíž prvky jsou opačné k původním.

Předpokládané zatížení jednotlivých cache pak, pokud $\mathbf{D}$ označíme řádkový vektor požadované rychlostí přenášení dat z jednotlivých *hashů*.

$$\mathbf{B_{cachep}} = \mathbf{B_{parst}}\mathbf{D} \tag{7.5}$$

7.1 Replikace origin

Řeší redundanci dat pro případ selhání některého ze strojů, cílem je zabránit ztrátě dat, prostředkem udržování určitého počtu kopií všech dat. Problémem je nedělitelnost *hashů* a jejich variabilní velikost v čase i mezi sebou. Kopírování dat mezi jednotlivými *cache* je drahá operace a mělo by být minimalizováno. Technické řešení přesouvání dat za jejich stálé dostupnosti je nad rámec této práce.

Z hlediska optimální *hashové mapovací matice* jsou požadavky jednoduché:

$$\mathbf{IM} >= \mathbf{R} \tag{7.6}$$

$$\mathbf{MS}^T = \mathbf{U} < \mathbf{C} \tag{7.7}$$

$$\mathbf{MS_{pred}}^T = \mathbf{U_{pred}} < \mathbf{C} \tag{7.8}$$

kde $\mathbf{M}$ je hashová mapovací matice, $\mathbf{I}$ jednotkový řádkový vektor velikosti počtu *cache*, $\mathbf{R}$ sloupcový vektor požadované redundance *hashů*, $\mathbf{S}^T$ sloupcový vektor velikostí *hashů*, U vektor dat obsazených *hashi* na *cachi* a $_{pred}$ označuje předvídaný údaj a $\mathbf{C}$ je vektor kapacity *cachí* Předpokládá se takový cachovací mechanimus, že budou využity všechny soubory, takže pro statistiky disku lze použít rovnici 7.2.

Základní kritéria pro optimalizaci rozložení pak jsou:

- rozptyl využití diskového prostoru *cache hashi* $var(\mathbf{K} - \mathbf{U})$
- počet nových umístění *hashů* oproti předchozí hashové mapovací matici $\Sigma p(\mathbf{M} - \mathbf{M_{old}})$, kde p je transformace matice nahrazující záporné prvky matice nulami.

Jelikož složitost přímého výpočtu je vysoká, je nutné použít některou z metod vícekriteriální optimalizace. Případné řešení hrubou silou průchodem celým stavovým prostorem je však v zásadě možné, jelikož stavový prostor lze s úspěchem redukovat zpřísněním požadavku 7.6 na $\mathbf{IM} = \mathbf{C}$.

7.1.1 Metody optimalizace

Nalezení optima v celém stavovém prostoru

Jelikož stavový prostor se spřísněnými požadavky není

Greedy optimalizace

Greedy optimalizace prochází stavový prostor tak, že se vždy přesune do sousedního stavu, který je nejlépe ohodnocen.

Zásadní nevýhodou greedy optimalizace je fakt, že často ukončí běh v lokálním extrému, místo v globálním. Při nasazení na optimalizaci mapovací matice však toto chování může být výhodné, jelikož pak lze pominout kritérium minimálního množství změn matice a spokojit se s prvním nalezeným lokálním optimem.

Genetické algoritmy

Lze také použít některou z metod genetických algoritmů, možný návrh pak může být:

fenotyp M

genotyp vektor čísel - počet nul mezi jedničkami v M, bráno po sloupcích.

převod genotyp - fenotyp do naplnění požadované redundance je sloupec **M** odpovídající *hashi* plněn tak, že pokud požadované pole již není prázdné, naplní se následující prázdné a s dalšími se pokračuje od něj, pokud je za hranicí matice, pokračuje se od prvního pole. Různé genotypy tedy mohou odpovídat stejnému fenotypu.

inicializace náhodné fenotypy splňující kritérium dostatečné redundance (přesně)

křížení libovolné

mutace změna náhodného čísla.

7.1.2 Plánování změn mapování

Změny mapování lze provádět pravidelně, nebo v závislosti na nastalé události.

Reaktivní

Reaktivní plánování spustí algoritmus pro mapování souborů až v okamžiku, kdy se stav rozložení souborů blíží nějakému limitu, u origin replikace to bude docházející diskový prostor na některém ze serverů nebo změna topologie clusteru.

Výhodami toho přístupu je celkové malé množství přesunů dat a možnost dosažení relativně lepšího mapování oproti kontinuálnímu přemapování.

Nevýhodou je nárazové generování mnoha přesunů v krátkém čase a možnost pozdní reakce.

Kontinuální

Kontinuální přemapování pravidelně vyhodnocuje výhodnost provedení změn mapování a v případě nalezení výhodné změny ji provede.

Výhodou je stálé udržování lokálně optimálního mapování po celou dobu běhu, nevýhodou možnost cyklení mezi několika mapováními a tím zbytečná zátěž infrastruktury a také velká pravděpodobnost ustrnutí v pouze lokálním optimu mapování

7.1.3 Další optimalizační kritéria

Připravenost na změny

Rovnoměrné využití i po odebrání cache nebo služby. Případně i možnost navýšení redundance pro případ odebrání cache.

$$I_c M = R + I_h$$

Rovnoměrné využití i v případě nerovnoměrného přibývání dat služeb (nerovnoměrnost přírůstku dat u hashů je však nepravděpodobná).

7.1.4 Předpovědi růstu hashů

Vzhledem k v čase rovnoměrnému růstu velikostí služeb lze předpokládat růst velikosti hashe úměrný jeho současné velikosti, respektive lze použít dlouhodobý průměrný růst.

7.2 Replikace edge

Řeší redundanci dat pro kapacitu výdeje dat, cílem je mít pro veškerá data dostatečnou kapacitu výdeje. Pracuje na úrovni hashů. Problémem je kapacita cache (úložná zařízení, síť, paměť) a rychlost distribuce dat na novou cache.

Reakce edge replikace je kritická při nerovnoměrném sítovém provozu [4]. Přesuny, respektive přidání replikace hashů jsou stále drahou operací, kdy není nutný přesun tak velkého množství dat, ale v době před nahráním všech vydávaných dat na cache dojde k velkému množství přesměrování požadavků. Dále doba reakce systému rozdělování požadavků na cache (Loadbalancer, DNS) může být dlouhá (sekundy až desítky minut)

Vyjma distribuce dat podle zatížení lze uvažovat i distribuci dat podle jiných měřítek, například lokality.

Extrémním řešením je plná replikace v rámci edgeových cachí, kdy je přesouvání dat minimální. Veškerá data pak na edge cache budou spravována cachovacími mechanismy. Takové řešení je podle [24] v některých ohledech optimální.

Problémem tohoto řešení je silně neoptimální využití paměti (z libovolného serveru může být požadován libovolný soubor) a z toho plynoucí vyšší latence výdeje (je nutno načíst data z disku), případně při vyčerpání diskového prostoru až nutnost častého přesměrování na origin cache.

Z hlediska mapovací matice hledáme takovou matici M, pro kterou platí:

$$\mathbf{B_{cachep}} < \mathbf{B_{cache}} \tag{7.9}$$

pro vypočtené podle rovnice 7.5 při novém mapování $\mathbf{M}$ a současném zatížení hashů $\mathbf{D}$ a předpokládaném zatížení hashů $\mathbf{D_{pred}}$, dále při mapováních $\mathbf{M}_{Fj}$ upravených tak, jako by nebyla funkční cache j při $\mathbf{D}$ i $\mathbf{D_{pred}}$ a také při novém mapování $\mathbf{M}$ a zatíženích D_{Fi} a $\mathbf{D_{predFi}}$, která mají zátěž hashe i nastavenu na trojnásobek odhadu, což v případě vyvoření flash crowdu poskytne čas na úpravu mapovaní na změněný stav.

Základní kritéria pro optimalizaci mapování pak mohou být:

- Jednotlivé *hashe* jsou vydávány z nejmenšího počtu cachí, neboli počet *hashů* na *cachi* je co možná nejmenší, neboli součet všech hodnot $\mathbf{M}$ je minimální

$$\mathbf{I_c M I_h}$$

- minimální počet přesunů hashů.

$$\Sigma\left(\mathbf{M} - \mathbf{M_{old}}\right)$$

Vzhledem k složitosti přímého algoritmického řešení (třída NP) je nutné prohledávat stavový prostor. Použití hrubé síly není možné vzhledem k velikosti stavového prostoru a nemožnosti jeho omezení.

7.2.1 Metody optimalizace

Nalezení optima v celém stavovém prostoru

Vzhledem k velikosti stavového prostoru je nepraktické.

Greedy

V zásadě má stejné výhody a nevýhody jako u replikace origin.

Genetické algoritmy

fenotyp odpovídá $\mathbf{M}$

genotyp vektor reálných čísel $< 0, 1 > \mathbf{M_{vect}}$, kdy každé odpovídá hodnotě z matice $\mathbf{M}$.

převod genotyp - fenotyp zaokrouhlením.

inicializace náhodné genotypy a genotyp převedený ze současného mapování.

křížení křížení reálných čísel.

mutace změna náhodného čísla.

Toto mapování v principu nebude dávat výsledky splňující základní požadavek 7.9. Částečným řešením (které zrychlí konvergenci k řešení vyhovujícímu požadavku, ale sníží pravděpodobnost nalezení optimálního mapování) je změna inicializace z náhodných fenotypů na takové náhodné fenotypy, které tuto podmínku splní.

7.2.2 Plánování změn mapování

Reaktivní

Reakce na docházející zátěž *cache* blížící se limitu, případně velkou změnu zátěže *hashe* nebo změnu topologie clusteru.

Výhody a nevýhody jsou obdobné reaktivnímu plánování změn origin replikace.

Kontinuální

Pro edge replikaci může mít větší smysl kontinuální přemapování, jelikož přesuny jednotlivých hashů jsou zde méně náročné (přesouvají se pouze používaná data) a globální optimum pravděpodobně vzhledem k proměnlivé zátěži není praktické dosáhnout

7.2.3 Předpovídání zátěže hashů

Budoucí zátěž jednotlivých *hashů* lze nejjednodušeji odhadnout lineární extrapolací z posledních dat zátěže. Lze volit i jiné extrapolace, ale vzhledem k velké dynamice zátěže jejich předpovědi nebudou zásadně přesnější.

7.3 Cache

Řeší omezení diskovým prostorem cache (serveru). Řešení vyrovnávací paměti je mimo téma této práce, i když s ním blízce souvisí. V optimálním případě by na cachi (serveru) byly nahrány pouze soubory, které jsou z něj aktivně vydávány, a tedy jsou nahrány v paměti.

Úkolem cachovacího mechanismu je vyžádat soubory, které jsou z cache požadovány, z origin serverů a odstranit z disku soubory, které již nadále nejsou vydávány. Při extrémní replikaci na edge serverech se mechanismus diskové replikace musí částečně týkat i replikace obsahu, kdy upřednostňuje časté přesměrování na okolní edge servery před nahráním dalších dat. Vzhledem k jemné granularitě musí být mechanismus udržování cache nenáročný na paměťové prostředky a velmi jednoduchý. Důležitá je i rychlost reakce.

7.3.1 LRU

Least recently used - z cache jsou odstraněny soubory, ke kterým nebylo nejdéle přistupováno. Vyžaduje udržování fronty souborů podle času posledního přístupu. Je možné zjednodušení, je tato fronta udržována pouze pro určitý časový interval a přednostně jsou odstraňovány náhodné soubory, které se ve frontě nenacházejí.

7.3.2 LFU

Least frequently used, obdoba LRU, avšak je v brána v potaz frekvence přístupu k souborům. Z implementačního hlediska je velmi vhodné udržovat data pro jednotlivé soubory pouze v určitém intervalu.

Kapitola 8

Výběr algoritmů, návrh zlepšení

8.1 Návrh

Pro testování a simulaci byly zvoleny v první fázi jak pro origin, tak pro edge jednoduché algoritmy s kontinuálním přemapováním. Optimalizace mapování pak probíhá greedy přístupem.

Pro edge mapování je dále navržen algoritmus používající hashovou mapovací matici, která je optimalizována pomocí genetických algoritmů.

Vzhledem k omezené velikosti CDN Seznamu není třeba používat explicitní algoritmy pro přesun vyžadovaných dat do lokalit, kde jsou vyžadovány. Tuto funkcionalitu však hashová mapovací matice obsahuje implicitně v případě, že jednotlivé clustery se nachází v samostatných lokalitách.

8.2 Implementace

Nejjednodušším algoritmem je pro origin mapování iterativní greedy kontinuální přemapování. Pracuje tak, že najde v clusteru origin server, který má nejmenší podíl volného diskového prostoru a případě, že je tento podíl menší než nastavená hranice, provede přesun náhodně zvolených hashů na servery, které tyto hashe neobsahují a mají největší podíl volného místa. Tato implementace je nejjednodušším funkčním přístupem. Výhodou je nenáročnost na systémové prostředky, předvídatelnost a při nastavení hranice volného místa pro přesun i vyhýbání se zacyklení. Nevýhodou je stav stálých přesunů a možnost překmitů, kdy vzhledem k ignorování velikosti přesouvaných hashů je možné, že bude přesunuto zbytečné množství dat.

Nejjednodušším algoritmem pro edge mapování je obdoba jednoduchého algoritmu origin mapování. Sledované veličiny však musí být dvě, a to celkový výdej dat cache a výdej dat jednotlivých hashů.

V první fázi je pro cluster vyhodnocen výdej dat jednotlivých hashů. Ten je normalizován počtem cache, na kterých se daný hash nachází, a seřazením vznikne fronta hashů pro distribuci na další cache. Metrikou v této fázi je počet požadavků na hash.

Tato je rozdělena na tři části podle rozptylu: hashe, jejichž zatížení je menší než polovina mediánu zatížení jsou přiřazeny k odebrání, hashe, jejichž zatížení je větší než n-násobek mediánu jsou přiřazeny k n-násobnému přidání.

V konečné fázi je naplánována změna mapování. Jednotlivé hashe ze skupiny k odebrání jsou odebírány z té cache (na které se nachází), která byla zatížena nejvíce požadavky

a odpovídajícím způsobem je upraven model zatížení cache. Pokud je hash mapován na poslední cache, není odebrán.

Obdobným způsobem je změněno mapování hashů ze skupiny k přidání, kdy jsou přidány na maximálně N nejméně zatížených cache.

V případě, že se vyskytují cache, jejichž modelové zatížení je více než dvojnásobkem průměrného zatížení, jsou z nich přesunuty na nejméně zatížené cache náhodně vybrané hashe ze skupiny zbylých, dokud není jejich modelové zatížení menší než 1,75 průměrného zatížení.

Dále jsou na nejméně zatíženou cache registrovány hashe, které se v clusteru nenachází a byly požadovány.

Kapitola 9

Testy, simulace a vyhodnocení

Simulace proběhla v upraveném prostředí, ve kterém byl testován simulátor na shodu se skutečným stavem. Prostředí bylo upraveno změnou velikosti diskového prostoru jednotlivých cache pro otestování mapovacího algoritmu.

9.1 Základní

Základní test proběhl s objekty `Mapper` jednotlivých simulovaných clusterů s následujícím nastavením: pro origin clustery je použit `OriginMapper` a počtem replik souborů 2, pro edge clustery `EdgeMapper`. Byly provedeny následující testy: Test nahrávání souborů a test mapování souborů při běžném zatížení.

9.2 Vyhodnocení

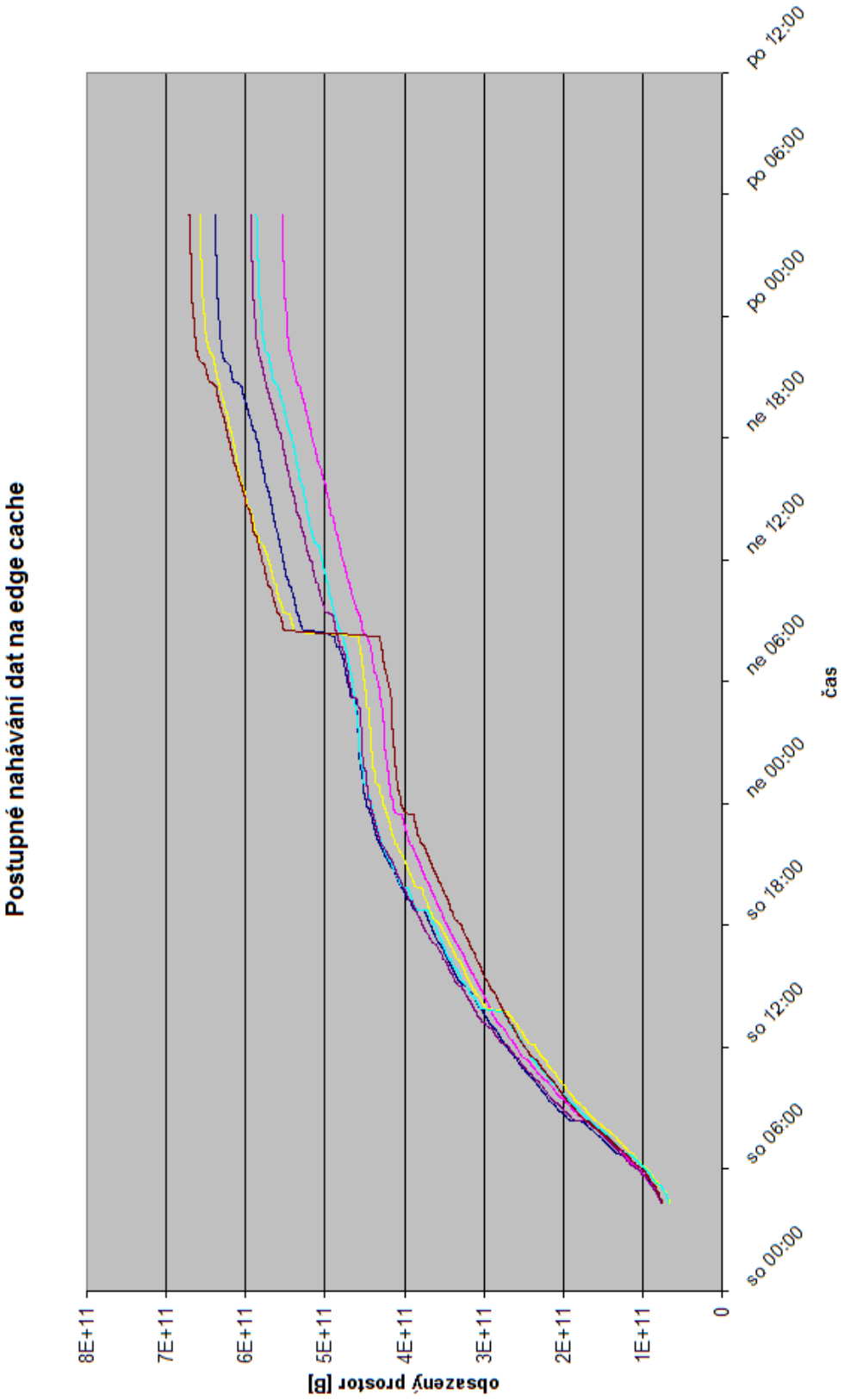
9.2.1 Nahrávání souborů

Postupné nahrávání souborů na edge cache je znázorněno na obrázku 9.1. Na jednotlivých cahce se nachází pouze právě vydávané soubory, povšimněte si skoků při změně mapování jednotlivých cache.

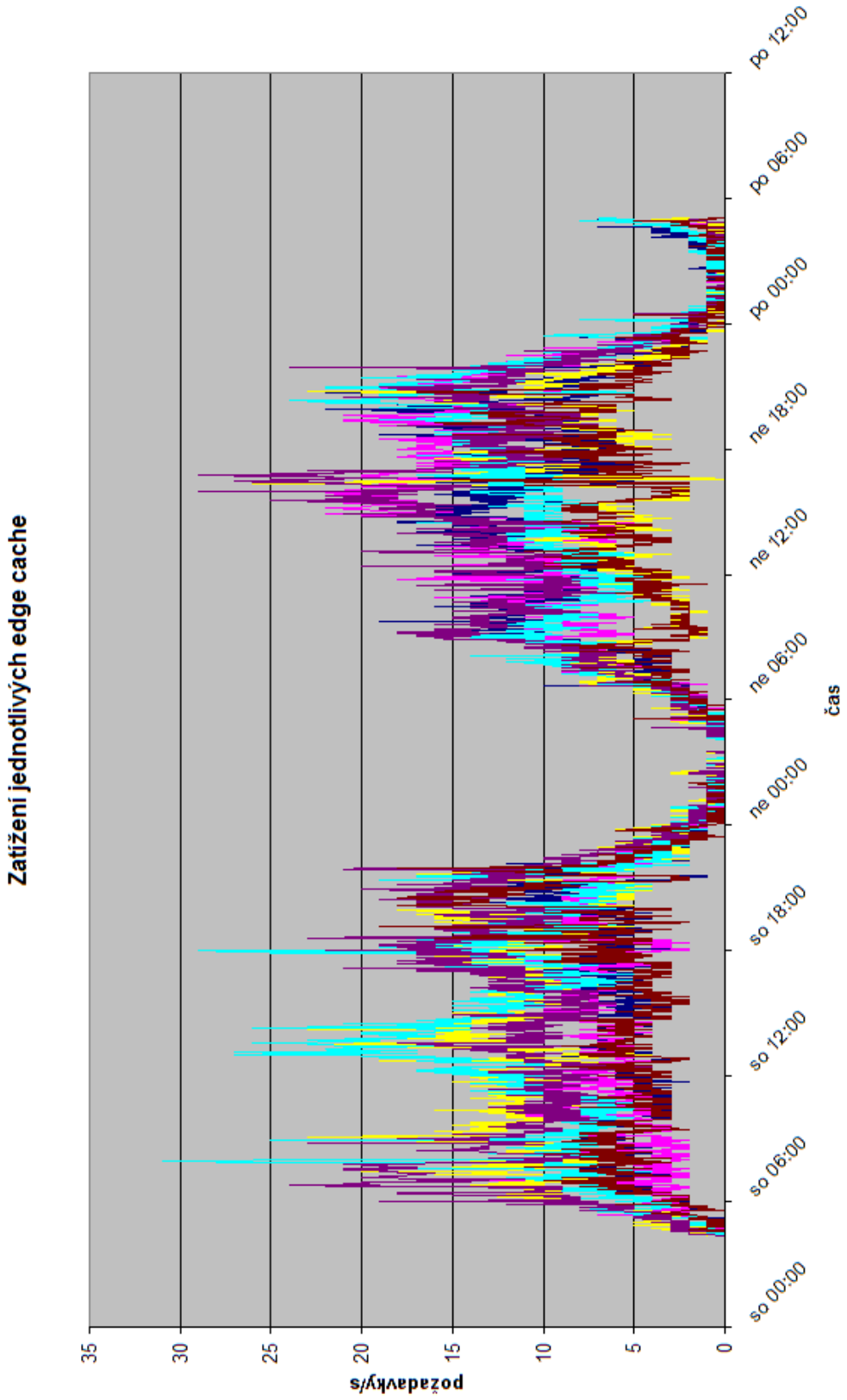
9.2.2 Běžné zatížení

Při běžném zatížení je pozorovatelný větší rozptyl zatížení jednotlivý cache způsobený tím, že soubory nejsou rozmístěny rovnoměrně a loadbalancer tak má méně prostoru k vyrovnávání zatížení.

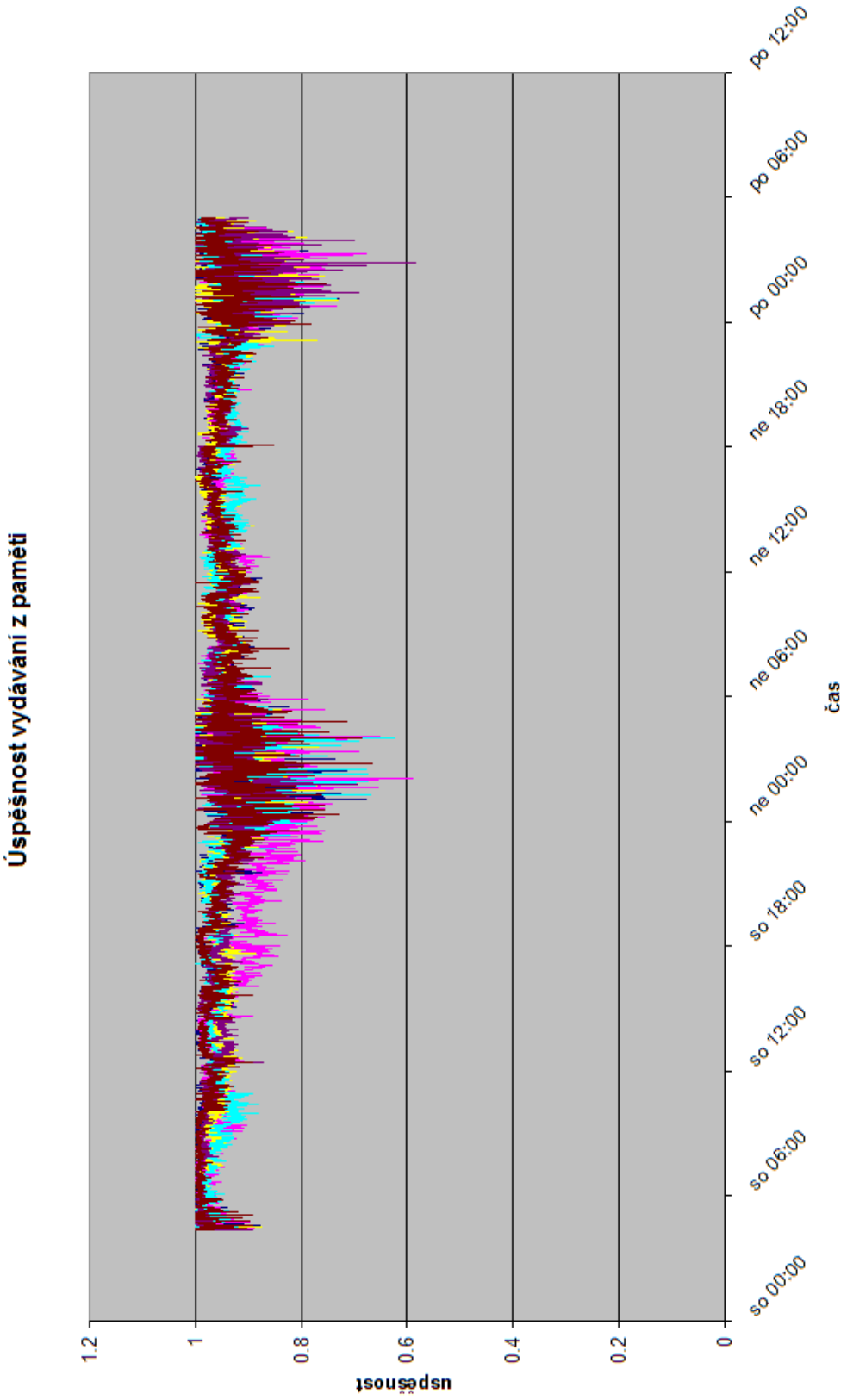
Oproti tomu úspěšnost výdeje z paměti značně stoupla. Je to způsobeno menším množstvím dat na cache a tedy větší pravděpodobností, že se vyžadovaný soubor nachází v paměti.



Obrázek 9.1: Simulované nahrávání do CDN edge cache. Povšimněte si skoků při přemapování hashů)



Obrázek 9.2: Simulované denní zatížení CDN. Zátěž jednotlivých cache není vyrovnaná jako u algoritmu všechno všude, ale je rovnoměrnější



Obrázek 9.3: Simulované denní zatížení CDN. Úspěšnost výdeje z RAM je zásadně větší než u základního algoritmu všechno všude.

Kapitola 10

Závěr

V této práci byla na základě rešerše o architektuře a algoritmech CDN vypracována *hashové distribuční matice* pro řízení mapování souborů určený pro CDN společnosti Seznam. Dále vzhledem nemožnosti nasazení algoritmů distribuce souborů do testovacího provozu ve společnosti Seznam byl vytvořen vysokoúrovňový simulátor vnitřní části CDN, na kterém byly provedeny základní experimenty bez distribučních algoritmů a se základními distribučními algoritmy. Vzhledem k dostatečnosti základních distribučních algoritmů pro určené účely byly pokročilé algoritmy ponechány pouze ve fázi návrhu. Prozkoumání výhod pokročilých algoritmů založených na maticovém modelu zůstává předmětem budoucí práce stejně jako test všech algoritmů ve skutečném prostředí, který však byl z provozních důvodů odložen na dobu neurčitou. Hlavním přínosem práce tak zůstává maticový model mapování dat (hashová distribuční matice), simulátor CDN a náhled na architekturu a některé algoritmy CDN druhé generace společnosti Seznam.

Literatura

[1] Aggarwal, A.; Rabinovich, M.: Performance of dynamic replication schemes for an Internet hosting service. *AT & T Labs, Tech. Rep*, 1998.

[2] Andrews, M.; Shepherd, B.; Srinivasan, A.; aj.: Clustering and server selection using passive monitoring. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, ročník 3, IEEE, 2002, s. 1717–1725.

[3] Ardaiz, O.; Freitag, F.; Navarro, L.: Improving the service time of web clients using server redirection. *ACM SIGMETRICS Performance Evaluation Review*, ročník 29, č. 2, 2001: s. 39–44.

[4] Arlitt, M.; Jin, T.: A workload characterization study of the 1998 world cup web site. *Network, IEEE*, ročník 14, č. 3, 2000: s. 30–37.

[5] Bartolini, N.; Casalicchio, E.; Tucci, S.: A walk through content delivery networks. In *Performance Tools and Applications to Networked Systems*, Springer, 2004, s. 1–25.

[6] Bent, L.; Rabinovich, M.; Voelker, G. M.; aj.: Characterization of a large web site population with implications for content delivery. *World Wide Web*, ročník 9, č. 4, 2006: s. 505–536.

[7] Cardellini, V.; Casalicchio, E.; Colajanni, M.; aj.: The state of the art in locally distributed Web-server systems. *ACM Computing Surveys (CSUR)*, ročník 34, č. 2, 2002: s. 263–311.

[8] Chen, Y.; Qiu, L.; Chen, W.; aj.: Efficient and adaptive Web replication using content clustering. *Selected Areas in Communications, IEEE Journal on*, ročník 21, č. 6, 2003: s. 979–994.

[9] Cieslak, M.; Foster, D.; Tiwana, G.; aj.: Web cache coordination protocol v2. 0. *At URL: http://www. web-cache. com/writings/internet-drafts/draft-wilson-wrec-wccp-v2-00. txt*, 2000.

[10] Day, M.; Cain, B.; Tomlinson, G.; aj.: A model for content internetworking (CDI). *Work in Progress*, 2003.

[11] Delgadillo, K.: Cisco distributeddirector. *Cisco White Paper*, 1999.

[12] Kangasharju, J.; Roberts, J.; Ross, K. W.: Object replication strategies in content distribution networks. *Computer Communications*, ročník 25, č. 4, 2002: s. 376–383.

[13] Karger, D.; Sherman, A.; Berkheimer, A.; aj.: Web caching with consistent hashing. *Computer Networks*, ročník 31, č. 11, 1999: s. 1203–1213.

[14] Lazar, I.; Terrill, W.: Exploring content delivery networking. *IT Professional*, ročník 3, č. 4, 2001: s. 47–49.

[15] Mockapetris, P.: Rfc 1034: Domain names-concepts and facilities, 1987. *URL: ftp://ftp. isi. edu/in-notes/rfc1034. txt.*

[16] Muller, K.; Vignaux, T.: SimPy: Simulating Systems in Python. *ONLamp. com Python Devcenter*, 2003.

[17] Ni, J.; Tsang, D. H.: Large-scale cooperative caching and application-level multicast in multimedia content delivery networks. *Communications Magazine, IEEE*, ročník 43, č. 5, 2005: s. 98–105.

[18] Ni, J.; Tsang, D. H.; Yeung, I. S.; aj.: Hierarchical content routing in large-scale multimedia content delivery network. In *Communications, 2003. ICC'03. IEEE International Conference on*, ročník 2, IEEE, 2003, s. 854–859.

[19] Pallis, G.; Stamos, K.; Vakali, A.; aj.: Replication based on objects load under a content distribution network. In *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, IEEE, 2006, s. 53–53.

[20] Pallis, G.; Vakali, A.; Stamos, K.; aj.: A latency-based object placement approach in content distribution networks. In *Web Congress, 2005. LA-WEB 2005. Third Latin American*, IEEE, 2005, s. 8–pp.

[21] Partridge, C.; Mendez, T.; Milliken, W.: Host anycasting service. Technická zpráva, RFC 1546, November, 1993.

[22] Pathan, M.; Byuya, R.; Vakali, A.: *Content Delivery Networks*, kapitola Content Delivery Networks: State of the Art, Insights and Imperatives. Springer, 2008, ISBN 978-3-540-77886-8, s. 3–32.

[23] Reese, W.: Nginx: the high-performance web server and reverse proxy. *Linux Journal*, ročník 2008, č. 173, 2008: str. 2.

[24] Stamos, K.; Pallis, G.; Vakali, A.: *Content Delivery Networks*, kapitola Caching Techniques on CDN in Simulated Frameworks. Springer, 2008, ISBN 978-3-540-77886-8, s. 127–153.

[25] Su, A.-J.; Choffnes, D. R.; Kuzmanovic, A.; aj.: Drafting behind Akamai: inferring network conditions based on CDN redirections. *IEEE/ACM Transactions on Networking (TON)*, ročník 17, č. 6, 2009: s. 1752–1765.

[26] Sysoev, I.: Nginx (2002).

[27] Szymaniak, M.; Pierre, G.; Van Steen, M.: Netairt: A Flexible Redirection System for Apache. In *ICWI*, Citeseer, 2003, s. 435–442.

[28] Tse, S. S.: Approximate algorithms for document placement in distributed web servers. *Parallel and Distributed Systems, IEEE Transactions on*, ročník 16, č. 6, 2005: s. 489–496.

[29] Vakali, A.; Pallis, G.: Content delivery networks: Status and trends. *Internet Computing, IEEE*, ročník 7, č. 6, 2003: s. 68–74.

[30] Valloppillil, V.; Ross, K. W.: Cache array routing protocol v1. 1998.

[31] Verma, D. C.: *Content distribution networks: an engineering approach*. John Wiley & Sons, 2003.

[32] Wang, L.; Pai, V.; Peterson, L.: The effectiveness of request redirection on CDN robustness. *ACM SIGOPS Operating Systems Review*, ročník 36, č. SI, 2002: s. 345–360.

[33] Wessels, D.: Application of internet cache protocol (ICP), version 2. 1997.

Příloha A

Stručný návod simulátoru

Program je psán ve skriptovacím jazyce python, instalace spočívá ve zkopírování adresářové struktury `prog` na místo, kde má uživatel právo k zápisu.

Simulátor je python modul `cdnsim`. Jeho použití je znázorněno v souboru `test01.py`. Soubor se spouští s argumenty název uloženého prostředí a název plánu. Uložené prostředí lze vytvořit pomocí utility `prepare`. Veškerá uložená prostředí musí být vytovořena pomocí funkce pro vytváření prostředí, kdy dvě základní se nacházejí v modulu `simpy.Environment`.

Výsledky simulace jsou přístupné jednak ve výstupním prostředí simulace (je pojmenované šablonou `prostředí_plán_timestamp.env` a v záznamu simulace v adresáří `logs`. V souboru prostředí jsou zajímavé zejména přístupové statistiky. Každý z objektů simulátoru má několik atributů statistik: `Server` například `misses`, `hits` a `ramhits`, udávající po řadě neúspěšné pokusy o stažení, úspěšné pokusy o stažení a úspěšné pokusy o stažení uskutečněné z paměti pro každý ze souborů. Statistiky lze získat pomocí metod `last_interval`, `num_in_last_interval`, `interval`, `num_in_interval` a `list` vracející časy přístupů za posledních interval sekund, počet přístupů, obdobně pro interval vymezený pomocí počátku a konce a výpis odpovídající ls. Prvním argumentem je tuple cesty k souboru (i části), druhou v případě last_interval funkcí délka intervalu v sekundách, p případě interval funkcí je druhým a třetím argumentem počátek a konec intervalu v simulačním čase.

Záznam simulace lze do formátu csv převést pomocí utility `statparser` s argumenty název záznamu, délka intervalu pro agregaci statistik a název výstupního souboru.

Příloha B

Obsah DVD

adresář	obsah
input_logs	vstupní data s provozu CDN v Seznamu
parsed_logs	vstupní data zpracovaná na soubory CSV
prog	adresář s programem
sim_parsed_logs	zpracovaný výstup simulace
tex	soubory pro sazbu této práce

dále DVD obsahuje pdf s textem práce

obsah adresáře prog

jméno	popis
cdnsim	adresář simulátoru
logs	adresář pro výstup simulátoru
*py	spouštěcí soubory pro simulátor a pomocné soubory
files.pickle	údaje o souborech CDN
days.env	základní testovací prostředí
mapdays.env	testovací prostředí s mapováním
days.pln	testovací plán
prog	adresář s programem
sim_parsed_logs	zpracovaný výstup simulace