ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Čontoš** Jméno: **Pavel** Osobní číslo: **350119**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Využití moderních distribuovaných systémů ukládání dat pro archivaci obrazové informace

Název diplomové práce anglicky:

Usage of modern distributed data storage systems for image information archiving

Pokyny pro vypracování:

Cílem této diplomové práce je provedení analýzy možností moderních distribuovaných úložišť a jejich následné využití pro ukládání diagnostických obrazových dat z laserových systémů v rámci projektu Eli Beamlines. V první řadě je potřeba provést analýzu existujících úložišť a identifikovat jejich výhody a nevýhody, stejně jako jejich další důležité vlastnosti. Hlavní důraz bude kladen na moderní distribuovaná a škálovatelná úložiště vhodná právě pro zpracování velkého množství obrazových dat. V souladu s identifikovanými požadavky bude vybráno nejvhodnější úložiště a to bude následně využito v rámci návrhu a implementace úpravy a/nebo rozšíření naivního úložiště aktuálně používaného v rámci zmíněného projektu. Navržené řešení bude otestováno v rámci zkušebního provozu nad reálnými daty. Na základě provedených experimentů bude ilustrována škálovatelnost a další klíčové vlastnosti navrženého řešení, stejně jako bude provedeno srovnání navrženého řešení vůči stávajícímu stavu.

Seznam doporučené literatury:

Holubová, Irena; Kosek, Jiří; Minařík, Karel; Novák, David: Big Data a NoSQL databáze. ISBN: 978-80-247-5466-6. Grada Publishing, a.s., 2015.

Depardon, Benjamin; Le Mahec, Gaël; Séguin, Cyril. Analysis of six distributed file systems. 2013. <https://hal.inria.fr/hal-00789086>

Weil, Sage A., et al. Ceph: A scalable, high-performance distributed file system. In: Proceedings of the 7th symposium on Operating systems design and implementation. p. 307-320. USENIX Association, 2006.

Marsching, Sebastian: Cassandra PV Archiver Reference Manual. <https://oss.aquenos.com/cassandra-pv-archiver/docs/3.2.5/manual/htmlsingle/>

Jméno a pracoviště vedoucí(ho) diplomové práce:

RNDr. Martin Svoboda, Ph.D., MFF

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **14.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

RNDr. Martin Svoboda, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry v.t. doc. Ing. Jiří Vokřínek, Ph.D.	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Využití moderních distribuovaných systémů ukládání dat pro archivaci obrazové informace

Pavel Čontoš
Otevřená informatika

Květen 2018
Vedoucí práce: RNDr. Martin Svoboda, Ph.D.

Poděkování / Prohlášení

Rád bych touto cestou vyjádřil poděkování RNDr. Martinu Svobodovi, Ph.D. za jeho cenné rady a pomoc při vedení této diplomové práce. Děkuji také Ing. Martinu Mazancovi, Ph.D. a Ing. Bedřichu Himmelovi z Fyzikálního ústavu Akademie Věd ČR, v. v. i. za jejich rady a odborný dohled nad touto diplomovou prací. Rovněž děkuji své rodině za podporu nejen v průběhu mého studia.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. 5. 2018

.....

Abstrakt / Abstract

Distribuované systémy jako Apache Hadoop jsou optimalizovány především pro operace čtení. My generujeme velké množství dat, která musíme efektivně archivovat a dále zpracovat, a proto vyžadujeme zejména stálý výkon operací zápisu, který neklesá s množstvím zapsaných dat. Na výběr máme z řady distribuovaných systémů, jež lze nasadit na komoditní hardware. Tyto systémy nabízejí škálovatelnost nejen zátěže systému, ale také hardwarových zdrojů a úložné kapacity. Samozřejmostí je také dostupnost po celý čas a odolnost vůči selhání síťové komunikace nebo části uzlů. Zaměříme se na popis těchto distribuovaných systémů a porovnání jejich výhod a nevýhod. Navrhneme řešení a vybrané distribuované úložiště propojíme s výchozím systémem archivace experimentálních dat. Získáme tak decentalizovaný systém, který je škálovatelný, dostupný, odolný vůči selhání a který je vhodný nejen pro účely archivace experimentálních dat malého objemu, ale také pro archivaci rozsáhlých obrazových a jiných dat.

Klíčová slova: distribuovaný systém; NoSQL; Apache Cassandra; Cassandra PV Archiver; Ceph; Hadoop; GlusterFS; škálovatelnost; dostupnost; odolnost vůči selhání.

Distributed systems such as Apache Hadoop are optimized primarily for read operations. We generate a large amount of data that we need to efficiently archive and process, and we therefore require a steady performance of write operations that does not drop with the amount of data written. We can choose from a range of distributed systems that can be deployed on commodity hardware. These systems offer scalability not only for system load, but also for hardware resources and storage capabilities. Of course, all-time availability and resilience to network communications or parts of nodes are commonplace. We will focus on describing these distributed systems and comparing their advantages and disadvantages. We will design the solution and link the selected distributed storage to the default archive of experimental data. We gain a decentralized system that is scalable, available, fail-safe, and is not only suitable for archiving small volume experimental data but also for archiving large image and other data.

Keywords: distributed system; NoSQL; Apache Cassandra; Cassandra PV Archiver; Ceph; Hadoop; GlusterFS; scalability; availability; fault tolerance.

Title translation: Usage of modern distributed data storage systems for image information archiving

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Během experimentů v rámci projektu ELI Beamlines[1] generujeme různorodá data, která mají velký objem a která rychle přibývají. Tato data vyžadují jiné formy zpracování, než nám přináší tradiční relační databázové systémy. Pohybujeme se okolo pojmu Big Data [1]. Podobná data nebo časové řady dat nevznikají pouze během sledování našich experimentů, ale jsou generovány např. sociálními médii nebo generovány různými senzory rozmístěnými po celém světě např. za účely sledování a předpovědi vývoje počasí. Čelíme problému, který zahrnuje rychlé generování obrovského množství různorodých dat. Nemusí nám stačit časové řady dat pouze archivovat, protože data mohou mít platnost pouze v okamžiku vzniku. Musíme je v reálném čase vyhodnotit a analyzovat. Nesmíme ani zapomínat na kvalitu dat. Data mohou být poškozena a je otázkou, do jaké míry se pak na taková data můžeme spolehnout.

Cílem diplomové práce je použití vhodného moderního distribuovaného systému pro účely archivace obrazových dat a napojení vybraného distribuovaného systému do stávající infrastruktury, která archivuje skalární nebo vektorová data o objemu v řádu jednotek kilobytů (kB). Vybereme takový distribuovaný systém, který splňuje naše nároky na škálovatelnost, spolehlivost a odolnost vůči selhání a který je vhodný pro archivaci obrazových dat o velikosti v řádu jednotek megabytů. Pro zapojení vybraného distribuovaného systému pak musíme využít některé z poskytovaných API a použít vhodný protokol pro přenos obrazových dat od zdroje, tj. kamer a senzorů uvnitř experimentálního pracoviště, do distribuovaného úložiště.

Tato diplomová práce je členěna do šesti kapitol. První z kapitol je úvodní část, ve které popisujeme naši motivaci pro řešení tohoto problému. V kapitole 2 „Výchozí stav“ popisujeme výchozí stav námi sestavovaného systému pro účely archivace skalárních, vektorových a obrazových dat a určíme požadavky úpravy výchozího systému, které jsou řešeny touto diplomovou prací. V kapitole 3 „Distribuované systémy“ definujeme pojmy, pomocí kterých porovnáme vybrané distribuované systémy. Zaměříme se na architekturu distribuovaných systémů, distribuci dat uvnitř těchto systémů, možnosti škálovatelnosti, dostupnost, odolnost vůči selhání, poskytovaná API a na procesy, často automatizované procesy, které probíhají uvnitř těchto systémů s minimální nutností uživatelského zásahu. V závěru kapitoly vybrané systémy porovnáme a vybereme pro naše účely nejvhodnějšího kandidáta. V kapitole 4 „Archivace obrazových dat“ představíme změny systému, které umožňují archivaci obrazových dat. Nejprve představíme řešení jako celek a následně se zaměříme na jednotlivé pozměněné nebo nově přidané komponenty systému archivace dat. Popíšeme způsob nasazení vybraného distribuovaného úložiště, jeho nastavení a použité prvky nebo funkce vybraného úložiště a také popíšeme implementaci změn výchozího systému tak, aby bylo možné efektivně využívat vybrané distribuované úložiště pro účely archivace obrazových dat. V kapitole 5 „Evaluace“ se zaměříme na vyhodnocení vybraného řešení z pohledu škálovatelnosti a výkonu. V závěrečné kapitole 6 zhodnotíme přínosy námi zvoleného řešení a popíšeme další předpokládaný vývoj a použití systému.

[1] https://www.eli-beams.eu

1.1 Typy úložných systémů

Pro účely archivace dat můžeme použít tradiční *relační databáze* [1] nebo distribuované *NoSQL databáze* [1], které jsou často distribuovanými řešeními. V případě NoSQL databází můžeme dále vybírat mezi těmito druhy: databáze typu klíč-hodnota (angl. *key–value*), dokumentově orientované databáze (angl. *document oriented*), sloupcové databáze (angl. *column based*) nebo grafové databáze.

1.1.1 Relační databáze

Datový model v relačních databázích je založen na ukládání dat v co nejvíc granulované a ploché podobě, čehož je dosaženo pomocí normálních forem (1NF, 2NF, 3NF, BCNF a další) a algoritmy dekompozice nebo syntézy. Roztříštění dat nám umožňuje minimalizovat redundanci dat a optimalizovat operace zápisu na úkor rychlosti operací čtení. Selekce dat je založena na projekcích, spojení a agregací za využití jazyku SQL nebo relační algebry. Počítáme tak nové hodnoty, které nemusí být uloženy v databázi. Dále máme v relačních databázích velkou řadu transakčních modelů, které nám umožňují paralelně realizovat elementární operace a požadavky více uživatelů současně. Prokládání transakcí musí být šetrné a často je realizováno pomocí zámků, tj. pesimistický přístup, aby se zamezilo konfliktům čtení a zápisů dat. Stejně tak je nutné řešit problémy, kdy dlouhotrvající transakce blokují transakce trvající krátkou dobu.

1.1.2 NoSQL databáze

Dnes často převažují operace čtení nad operacemi zápisu. Není nutností data optimalizovat na zápis ani není nutná konzistence dat zaručená transakčními modely. Navíc pracujeme s velkými objemy dat, kde se relační databáze ukazují nevhodné. Potřebujeme technologii založenou na distribuovaném souborovém systému, kde ne všechna data jsou uložena na jediném uzlu. NoSQL databáze jsou alternativou k relačním databázím. Jsou optimalizovány pro operace čtení a umožňují zpracovávat ohromné množství dat v reálném čase. Jedná se o distribuované systémy, které nemají žádné schéma, které umožňují replikaci dat a které poskytují jednoduchá rozhraní.

1.1.3 Databáze typu klíč-hodnota

Jedná se o nejjednodušší model a zástupce NoSQL databází, který je založen na principu dvojice klíče (identifikátoru) a hodnoty (binárního objektu). Na základě klíče se přistupuje k hodnotě, která je černou skříňkou (angl. *black box*) pro databázový systém. Databázový systém neví o obsahu hodnoty a nemůže s ní pracovat. Systém nabízí pouze operace pro práci s klíčem a nelze pracovat s hodnotami na základě jejich vlastností. Tento databázový model je vhodný pro ukládání sezení (angl. *session*) dat u webových aplikací. Databázový server na základě ID (klíče) uloží hodnotu sezení, aniž by systém musel uložené hodnotě rozumět. Naopak se tento model nehodí v případě práce s komplexními daty, nad kterými se potřebujeme dotazovat. Stejně tak nelze zpracovávat operace nad více dvojicemi klíč, hodnota v rámci jediné transakce. Lze zpracovávat pouze atomické operace. Nejčastěji využívanými představiteli databází typu klíč-hodnota jsou systémy Redis[1], MemcachedDB[2] a Riak KV[3].

[1] https://redis.io
[2] https://memcached.org
[3] http://basho.com/riak-kv/

1.1.4 Dokumentově orientované databáze

Jedná se o datový model, který v databázi shromažďuje dokumenty. Podobné dokumenty se organizují v rámci jedné databáze nebo do kolekcí souvisejících dokumentů. Využívá se hierarchické stromové struktury v dokumentech. Podporovány jsou především formáty XML a JSON. Každému dokumentu je přiřazen identifikátor, na jehož základě lze k jednotlivým dokumentům přistupovat a pracovat s nimi. Dokumenty nejsou pro databázový systém černou skříňkou. Jde nad nimi vyhodnocovat dotazy. Tento databázový model je vhodný pro práci se strukturovanými dokumenty, které mají podobné schéma, např. logování událostí, blogy, analytické weby, reklama a jiné. Není nezbytně nutné, aby toto schéma bylo po celou dobu identické. Data jsou v rámci dokumentu uchovávána v agregované podobě (naopak v relačních databázích v granulované podobě). Naopak model není vhodný v případě, kdy chceme realizovat množinové operace, protože nelze manipulovat s více dokumenty současně. Nejčastěji používaným zástupcem dokumentově orientovaných databází je MongoDB [1–2].

1.1.5 Sloupcové databáze

Sloupcové databáze používají předepsané schéma, které je nutné dodat dopředu. Tabulka v tomto modelu je tvořena kolekcemi sloupců a podobných řádků, které nemusí být nutně identické. Každý jednoznačně identifikovaný řádek může obsahovat různou podmnožinu předem definovaných sloupců. Důsledkem toho řádek neobsahuje žádné hodnoty null v případě, že řádek neobsahuje všechny sloupce. Tento model dále umožňuje uchovávat více hodnot stejného sloupce v rámci jednoho řádku. Pro ukládání takovým mnohonásobných hodnot se využívají kolekce typu seznamu, množiny nebo mapy. Sloupcové databáze je vhodné používat v případě, kdy máme strukturovaná a rovná data s podobným schématem. Naopak nejsou vhodné na použití v případě, kdy vyžadujeme ACID [1] transakční přístup nebo komplexní dotazování se nad relacemi v tabulce (agregace, spojování tabulek). Nejčastěji používaným typem jsou Apache Cassandra [3] a Apache HBase[1].

1.1.6 Grafové databáze

Databáze tohoto modelu jsou implementovány jako grafy. Vrcholy grafu představují entity a vztahy mezi entitami jsou vyjádřeny hranami mezi vrcholy. Ke hranám i vrcholům lze přiřadit další atributy, které entity nebo vztahy mezi nimi dále popisují. Grafové databáze dělíme na netransakční a transakční. Netransakční grafové databáze obsahují pouze jediný velký graf. Naopak transakční databáze obsahují větší množství menších grafů, se kterými můžeme manipulovat klasickým transakčním způsobem, např. je můžeme celé přidávat nebo odebírat. Grafové databáze je vhodné použít jako alternativu k relačním databázím v případě, kdy pracujeme s daty, které lze reprezentovat jako graf. Grafové databáze jsou implementačně nejsložitější a nejnáročnější NoSQL databáze (opakem databází typu klíč-hodnota). Pokud je graf příliš veliký, je obtížné graf rozřezat a distribuovat na více uzlů stejně jako je obtížné dotazování se nad ním. U grafu není obtížné přistoupit k nejbližšímu okolí vrcholů, ale pro komplexní zpracování je nevhodný. K dotazování se používají grafové algoritmy. Nejpoužívanějším zástupcem grafových databází je Neo4J [1] obsahujícím pouze jediný obrovský graf, kde se předpokládá netransakční zpracování.

[1] https://hbase.apache.org

Kapitola 2
Výchozí stav

V této kapitole popíšeme výchozí stav systému, který používáme pro archivaci skalárních nebo vektorových dat a který chceme upravit tak, aby archivoval rozsáhlá obrazová data vznikající během experimentů probíhajících v laserových laboratořích. Rozdělíme celý projekt do fází, ve kterých probíhá implementace nebo zapojení jednotlivých částí, popíšeme náš systém a nakonec uvedeme požadavky na komponenty, které chceme přidat a které jsou řešeny touto diplomovou prací.

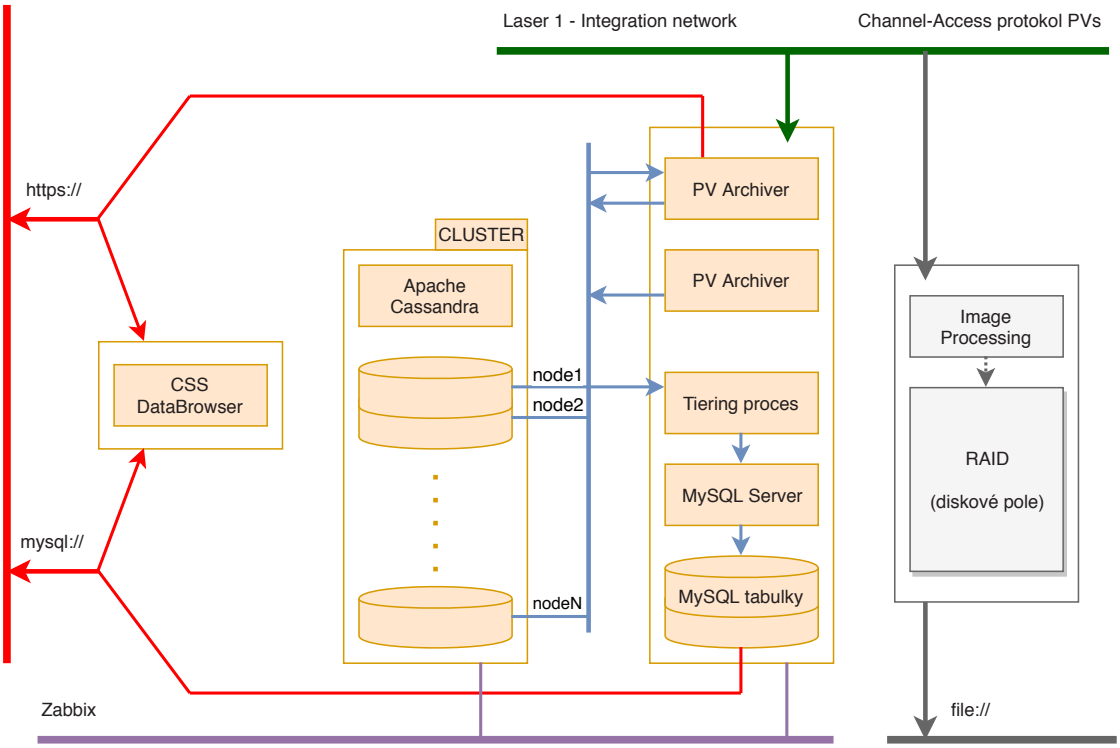
2.1 Fáze projektu

Celý projekt dělíme do dvou fází: krátkodobá a střednědobá fáze. V krátkodobé fázi implementujeme část systému, která umožňuje archivaci skalárních nebo vektorových dat, a část systému, která umožňuje konfiguraci celého systému a uložení nastavení ve vybrané relační databázi. Ve střednědobé fázi pak nasazujeme distribuované úložiště pro rozsáhlejší vektorová nebo obrazová data.

V krátkodobé fázi se zaměřujeme na veškeré procesní proměnné (angl. *process variables*, PV), tj. skalární data laserů, počítačů nebo kontrolních procesů. Chceme, aby tato data a stejně tak uživatelská konfigurace celého systému byla uložena ve strukturovaném úložišti. Ideálně v robustním úložišti, které udržuje datovou konzistenci. Veškerá PV se generují s frekvencí v řádu desítek hertzů (Hz), tudíž pro jejich archivaci využíváme strukturované distribuované úložiště Apache Cassandra, které má kapacitu v jednotkách terabytů (TB). Časové řady PV archivujeme s použitím aplikace Cassandra PV Archiver [4] (CPVA), která je určena pro archivaci právě tohoto typu dat do databáze Apache Cassandra. Tato aplikace je vhodná především díky možnostem škálování. Můžeme nasadit několik instancí CPVA na různé uzly a balancovat zátěž jednotlivých uzlů tak, aby nedocházelo ke ztrátě dat generovaných s vysokou frekvencí. Můžeme snadno přidávat nebo odebírat PV archivovaná skrze konkrétní instance CPVA pouhou změnou nastavení. Databáze Apache Cassandra pak poskytuje redundanci dat. Používáme replikační faktor tři, což zvyšuje odolnost vůči chybám systému, který zajišťuje archivaci skalárních dat.

V dlouhodobé fázi, kterou pokrývá tato diplomová práce, řešíme výběr distribuovaného úložného systému, tj. distribuovaného souborového systému nebo distribuovaného objektového úložiště, pro účely archivace rozsáhlých obrazových dat o velikosti přibližně 1,5 milionů bytů (MiB) generovaných s frekvencí v řádu jednotek Hz.

Obrázek 2.1 ilustruje výchozí stav systému. Z pohledu software máme systém archivující skalární nebo vektorová data malého objemu, tj. část systému řešenou v krátkodobé fázi, vyřešen z velké části, kdy pro archivaci dat využíváme instance CPVA. Zdroje dat komunikují s instancemi CPVA pomocí Channel Access protokolu [5–6], který kromě samotných PV používáme i pro přenos metadat, tj. atributů jako např. PV status nebo hodnoty alarmu přidružené jednotlivým PV. S pomocí tohoto protokolu můžeme využívat různé datové typy pro přenos vektorových nebo skalárních dat, přičemž nejčastěji



Obrázek 2.1. Výchozí stav systému archivace dat.

používáme typy s pohyblivou řádovou čárkou[1] (angl. *floating point*). CPVA pak archivuje data do databáze Apache Cassandra nebo databáze MySQL[2]. Časové řady PV jsou archivovány do prvně jmenované databáze, MySQL pak používáme pro uložení uživatelské konfigurace systému. CPVA umožňuje nejen archivaci dat, ale také jejich prohlížení skrze RESTful API [7] a webové protokoly. K zobrazování dat pak používáme aplikaci Control System Studio[3] (CSS), konkrétně její modul DataBrowser[4]. Tato část systému je v rámci krátkodobé fáze implementována a testována. V rámci střednědobé fáze implementujeme i distribuované datové úložiště pro obrazová data, u kterého předpokládáme jiný protokol pro přenos dat mezi zdrojem obrazových dat a distribuovaným úložištěm. Zároveň předpokládáme možnost prohlížení obrazových dat skrze webovou galerii. Z pohledu hardware i software je tato část ve výchozím stavu pokryta z méně než 5 %.

2.2 Komponenty systému

Systém, ze kterého vycházíme, se skládá z několika komponent: integrační sítě v laserové laboratoři (tzv. *L1 Integration Network*), Channel Access protokolu, CPVA instancí, MySQL serveru, Apache Cassandra clusteru, aplikace CSS DataBrowser a monitorovacího systému Zabbix[5].

Integrační síť v laserové laboratoři je zjednodušeně řečeno vrstvou, která generuje experimentální data. Skládá se z kolekce senzorů a kamer, které zachycují a monito-

1 https://en.wikipedia.org/wiki/Floating-point_arithmetic
2 https://www.mysql.com
3 https://github.com/ControlSystemStudio
4 https://github.com/ControlSystemStudio/cs-studio/wiki/DataBrowser
5 https://www.zabbix.com

rují průběh experimentů. Tyto senzory generují skalární data včetně jim přidružených metadat různého a pro popis systému nepodstatného významu. Kamery pak generují obrazová data o velikosti v řádu jednotek MB s frekvencí např. 4 Hz. Pokud používáme 60 kamer, které generují obrazová data o velikosti 1,5 MiB s frekvencí 4 Hz, pak tato vrstva generuje minimálně 360 MiB dat za sekundu. Tato data je potřeba dále zpracovat a archivovat, přičemž můžeme ovlivňovat nastavení zdrojů dat, tj. např. upravovat frekvenci, se kterou vznikají nová data, nebo můžeme přidávat nebo odebírat senzory a kamery. Lze tedy dosáhnout datového toku, který dosahuje hodnot v řádu GiB za sekundu.

Channel Access protokol slouží pro přenos dat generovaných v laserové laboratoři. Protokol je vhodný pro přenos skalárních nebo vektorových dat v objemu do několika málo MB [5]. Teoreticky je možné používat tento protokol i pro přenos dat v objemu několika GB, ovšem za cenu ztráty výkonu systému.

Vrstva okolo CPVA a úložných systému Apache Cassandra a MySQL má za úkol archivaci experimentálních dat s krátkodobou nebo dlouhodobou platností. Tato vrstva analyzuje a validuje experimentální data a následně je archivuje v některé z uvedených databází. Hodnoty PV jsou archivovány na základě jejich změny (angl. *event based*), nikoliv na základě vzorkování založeném na periodickém sběru dat. PV jsou archivovány spolu s metadaty.

Vzhledem k limitům obou databází, kdy ani MySQL, ani Apache Cassandra nestíhají efektivně zpracovávat a archivovat vektorová obrazová data, tato vrstva není za archivaci takto rozsáhlých dat zodpovědná. Uvažujeme buď o rozšíření této vrstvy přidáním distribuovaného úložiště vhodného pro ukládání dokumentů nebo objektů, nebo o vytvoření nové vrstvy zodpovědné pouze za ukládání obrazových dat.

K datům přistupujeme pomocí vrstvy skládající se z instancí CSS doplněné o modul DataBrowser. Jedná se o modulární nástroj postavený na vývojovém prostředí Eclipse[1], který umožňuje sledovat stav probíhajících experimentů. K datům také přistupujeme napřímo např. pomocí protokolu HTTP[2], kdy využíváme RESTful API systému CPVA nebo můžeme využít rozhraní databáze Apache Cassandra a dotazovat se pomocí jazyku CQL[3].

Pro sledování stavu celého systému pak používáme systém Zabbix, který umožňuje monitorovat různé metriky a stavy jednotlivých komponent, např. propustnost dat, latenci zápisu nebo čtení dat a jiné nejen z pohledu hardware, ale také z pohledu software.

Celý systém pak lze konfigurovat na základě konfiguračních skriptů a XML[4] souborů.

2.3 Definice problému

Použitím stávajících komponent snadno dosahujeme těchto požadavků pro skalární nebo vektorová data menších rozměrů, ale pro archivaci obrazových dat je stávající řešení nedostatečné. Následujícím cílem je rozšíření stávajícího systému o komponenty, které budou realizovat archivaci obrazových dat nebo všeobecně objemných dat, jejichž archivace do stávajících úložišť, tj. MySQL nebo Apache Cassandra, není efektivní z pohledu výkonu celého systému.

Chceme využít dobrých vlastností distribuovaných řešení namísto použití drahých serverů v případě centralizovaných typů úložišť. Spoléháme tak především na použití

[1] http://www.eclipse.org
[2] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[3] http://cassandra.apache.org/doc/latest/cql/
[4] https://www.w3.org/XML/

komoditního hardware a snadné rozšíření úložiště, ideálně za běhu systému bez nutnosti odstávky. Chceme úložiště, které bude vhodné pro ukládání dat s krátkodobou i dlouhodobou platností, kdy platností uvažujeme alespoň 10 dnů u dat s krátkodobou platností a alespoň 90 dnů u dat s dlouhodobou platností. Z pohledu kapacity úložiště vyžadujeme méně než 10 TB úložného prostoru pro archivaci dat s krátkodobou platností, méně než 10 TB prostoru pro archivaci dat s dlouhodobou platností a mnohem více úložného prostoru pro archivaci obrazových dat, pravděpodobně až v řádu stovek TB.

Požadujeme řešení, které má otevřený kód (angl. *open source*) a které umožňuje svobodné použití bez nutnosti pořizování drahé licence.

Kapitola 3
Distribuované systémy

V této kapitole se zaměříme na existující distribuované systémy, které mohou být vhodné pro archivaci časových řad obrazových dat. Nejprve definujeme základní principy a vlastnosti distribuovaných systémů a následně se zaměříme na vybraná distribuovaná řešení a popíšeme tyto systémy na základě definovaných principů a vlastností. Nakonec vybrané systémy vzájemně porovnáme a vybereme systém, kdy je pro náš účel nejvíce vhodný.

3.1 Základní principy a vlastnosti

Distribuované systémy se skládají z mnoha uzlů, které sdílí své hardwarové a systémové prostředky. Sdílené hardwarové prostředky využívají pro zpracování nebo ukládání velkých objemů dat. Jednotlivé uzly jsou spojeny v síť, kde spolu komunikují. Toto spojení s sebou přináší řadu problémů. Uživatel, který používá úložné systémy, očekává, že všechna data jsou vždy přístupná a z jeho pohledu uložena na jednom místě. Úkolem distribuovaných řešení je tato očekávání naplnit a vhodně reagovat na výpadky v síťové komunikaci tak, aby nedocházelo ke ztrátám nebo poškození uložených datových objektů. Distribuované systémy musí:

- Vhodně reagovat na rostoucí objem uložených dat a umožnit nejen škálovatelnost kapacity úložiště, ale také propustnosti dat během operací čtení nebo zápisu.
- Maximalizovat dostupnost služeb a reagovat na uživatelské operace čtení nebo zápisu dat.
- Dynamicky reagovat na chyby v systému a vyřešit je nasazením nástrojů zajišťujících odolnost vůči chybám (angl. *fault tolerant*).
- Ukrýt vlastní složitost před uživateli systému, aby práce s distribuovanými systémy byla stejná jako práce s konvenčními úložnými systémy.

Důležitá je také rychlost vyřízení uživatelských požadavků. Manipulace s objekty by měla být podobně rychlá jako u konvenčních systémů archivujících data.

3.1.1 Architektura

Distribuované systémy z pohledu architektury dělíme na systémy centralizované, distribuované nebo decentralizované.

Centralizované systémy používají centralizovaný prvek, tj. uzel, který spravuje metadata, např. jména a atributy fyzických umístění souborů nebo objektů. Metadata nebo jiná data spravovaná tímto uzlem jsou uložena pouze na tomto místě. Dostupnost celého systému je tak závislá na jediném uzlu a nemůže být garantována. Tento uzel je slabým prvkem (angl. *single point of failure*) celého systému a zároveň úzkým hrdlem (angl. *bottleneck*) z pohledu výkonnosti, pokud se snažíme systém rozšiřovat a přidávat nové uzly. Tento přístup bez použití zálohy centralizovaného prvku je vhodný především pro malé clustery, kde ukládáme pouze dočasná a nekritická data.

Distribuované systémy se nespoléhají pouze na jediný uzel spravující metadata. Naopak metadata rozloží napříč větším množstvím uzlů. Tento přístup je spolehlivější ve srovnání s nasazením jediného centrálního uzlu. Cílem distribuované architektury je zajištění škálovatelnosti s ohledem na zátěž systému. Pokud systém neobsahuje dostatečné množství uzlů spravujících metadata, stává se tento prvek systému úzkým hrdlem z pohledu výkonu. Pokud navíc některý uzel selže, může to vést na kaskádový efekt, protože klienti požadující operace čtení nebo zápisu přesměrují své požadavky na zbývající uzly, které z důvodu přetížení mohou postupně selhávat. Tato architektura je použita např. u systémů MooseFS [8] nebo XtreemFS [9].

Decentralizované systémy jsou založeny na distribuovaném zatížení celého systému. Nepoužívají metadata pro určení umístění objektů. Namísto toho často používají algoritmický přístup k určení umístění objektů např. na základě klíče nebo názvu objektu. Tyto systémy jsou vysoce škálovatelné a mohou ve stejný čas obsluhovat velké množství klientů. Je zde také odstraněn slabý prvek systému, protože se při určení umístění objektu nemusíme dotazovat určené podmnožiny uzlů. Přirozenost škálování těchto systémů zajišťuje také odolnost vůči selhání, protože tyto systémy jsou schopny automaticky detekovat blížící se selhání uzlů a reagovat rebalancováním zasažených bloků dat, aby byl zachován replikační faktor. Zástupcem decentralizovaných systémů je např. Infinit [10].

3.1.2 Distribuce dat

U distribuovaných systémů můžeme použít dvě navzájem nezávislé strategie distribuce dat: sharding nebo replikaci. Obě strategie lze kombinovat.

V případě distribuce dat s použitím technologie sharding chceme jednotlivé objekty z kolekce objektů distribuovat napříč uzly clusteru. Objekty rozdělíme vybraným způsobem do disjunktních podmnožin tak, že různé objekty umístíme na různé uzly clusteru. Přímočarou distribucí tak můžeme škálovat na základě objemu ukládaných objektů. Není ale vidět, jak jsme použitím této strategie schopni navýšit výkonnost systému, tzn. jak systém bude reagovat na požadavky čtení nebo zápisu objektů.
Systémy, které implementují sharding, musí naplňovat tři hlavní požadavky:

- Škálujeme data z hlediska jejich objemu. Pokud jsou uzly clusteru homogenní, předpokládáme, že data budou rozmístěna rovnoměrně. Pokud cluster obsahuje různá disková pole s různou úložnou kapacitou, musíme objekty rozdělit jiným způsobem.
- Distribuce dat musí vést k rozložení zátěže clusteru z hlediska vyhodnocování dotazů nad objekty, tj. z hlediska požadavků na operace čtení nebo zápisu objektů.
- Chceme minimalizovat dobu vyřizování požadavků, tzv. latenci, i s ohledem na geografické umístění jednotlivých uzlů.

Kromě uvedených požadavků musíme určit uzel, který je odpovědný za vyřízení požadavků čtení nebo zápisu objektů. Uživatel distribuovaného systému vždy kontaktuje nějaký uzel v clusteru, který buď tento požadavek vyřídí, nebo uživatele přesměruje na jiný uzel, který může požadavek vyřídit. Můžeme používat různé strategie řešení uživatelských požadavků, např. *master–slave* [1] nebo *peer-to-peer* [1].

Ani jedna ze strategií shardingu není vhodná ať už z hlediska zavádění slabého prvku v clusteru nebo z důvodu výkonu operací zápisu nebo čtení dat. Distribuované systémy tak často neukládají mapování jednotlivých objektů ani nepoužívají centralizované prvky spravující metadata, ale využívají obecných pravidel a algoritmů pro určení místa zápisu nebo čtení objektů. Základem těchto algoritmů je např. *hashování* [1] nebo *range partitioning* [1].

Cílem replikace je distribuce stejných dokumentů, tzn. replik nebo kopií, na různé uzly. Dosáhneme tím zvýšení výkonnosti systému, protože dojde k rozložení zátěže plynoucí z požadavků nad stejným objektem mezi více uzlů ukládajících tento dokument. Stejně tak získáváme možnost vypořádání se s mimořádnými situacemi jako např. selhání úložného zařízení, kdy díky existenci jiné repliky nedochází ke ztrátě dokumentu. Počet replik je určen replikačním faktorem [1]. Samotnou replikaci lze realizovat více způsoby, např. pomocí strategií *master–slave* nebo *peer-to-peer*.

Mnoho distribuovaných systémů kombinuje použití shardingu i replikace a implementují je tak, aby byla minimalizována nutnost zásahu administrátora systému. Ten pouze určí replikační faktor a o vše ostatní se postará distribuovaný systém pomocí automatických procesů.

3.1.3 Škálovatelnost

Jedná se o schopnost systému efektivně využít velkého množství uzlů, které jsou dynamicky a průběžně přidávány do systému nebo naopak odebírány např. v případě selhání. Základem škálovatelnosti systému je použití škálovatelné architektury, která ideálně neobsahuje žádný centralizovaný nebo jiný slabý prvek, na kterém celý cluster závisí nebo který se stává úzkým hrdlem z pohledu výkonu celého systému. Používáme dva základní přístupy pro dosažení škálovatelnosti systému: vertikální nebo horizontální škálování.

Princip vertikálního škálování je také označován jako tzv. *scalling up/down*. Používá se především ve světě relačních databází, kde celý systém může být nasazen na jediném uzlu. Každé navýšení výkonu znamená přidání více hardwarových prostředků tomuto jedinému uzlu. Můžeme použít více paměti RAM, vyměnit procesory za novější nebo rychlejší generaci, rozšířit diskové pole nebo zaměnit jednotlivé disky za výkonnější. Použití jediného uzlu nám umožňuje snadnou implementaci ACID transakcí, které zajišťují silnou konzistenci. Systém skládající se z jediného uzlu je také jednodušší na provoz a údržbu systému, protože neřešíme distribuci ani dostupnost dat. Tento přístup s sebou přináší i řadu nevýhod nebo technických komplikací. Ať už používáme sebelepší a výkonnější uzel, vždy narazíme na výkonnostní limity. Pokud je uzel využíván dostatečně pod limity výkonnosti, systém pracuje dobře a bez zásadních komplikací. V případě, kdy se značně přibližujeme limitům výkonnosti uzlu, může docházet k významnému zpomalování systému. Kromě výkonu jsou dalšími nevýhodami vysoká pořizovací cena, která exponenciálně roste s výkonností systému.

Alternativním principem je horizontální škálování označované také jako tzv. *scalling out/in*. Pokud chceme zvýšit výkon celého systému, přidáme nové uzly do clusteru. Tento princip je založen na použití clusteru, tj. sítě, která sestává z množiny osobních počítačů, a to s možností použití levného komoditního hardware. Nemusí se nutně jednat o desktopový počítač s displejem. Když chceme navýšit výkonnost systému, jednoduše přidáme další uzly. Největší výhodou je poměr cena výkon. Cluster ekvivalentně výkonný se superpočítačem je významně levnější. Navíc získáme flexibilitu systému. Pokud nám přibude 100 000 nových uživatelů, zareagujeme přidáním nových levných uzlů do již stávající infrastruktury. Během takového upgrade ani není nutné systém vypínat, neboť distribuované systémy umožňují rozšíření za běhu a bez přerušení. Odpadá tím tzv. *deployment downtime*.

Princip horizontálního škálování vypadá jako výhodnější ve srovnání s principem vertikálního škálování, ale přesto obsahuje řadu zásadních nevýhod, které mohou bránit jeho použití. Propojení uzlů do sítě představuje kvůli nespolehlivosti sítě značný problém. Nelze přenášet neomezené množství dat mezi jednotlivými uzly (angl. *bandwidth*),

nemáme zaručen čas přenosu zpráv, tj. latenci, a dokonce nemáme zaručen ani přenos samotný.

3.1.4 Dostupnost

Pokud nasadíme funkční uzel v clusteru, tento uzel je dostupný. Uživatel distribuovaného systému je schopen připojit se k uzlu a uzel obsluhuje požadavky na čtení nebo zápis dat. Uzel musí tyto požadavky úspěšně vykonat. Pokud dochází k vykonání požadavků bez výjimek, systém lze označit jako dostupný.

3.1.5 Odolnost vůči selhání

Síť, která propojuje cluster, je zdrojem mnoha technických nebo jiných problémů, kdy dochází např. ke ztrátám zpráv posílaných mezi uzly v clusteru. Stává se, že takový problém rozdělí cluster do několika disjunktních podmnožin, které spolu nemohou komunikovat. Pokud spolu dva uzly nekomunikují, můžeme předpokládat selhání druhého uzlu nebo čekat delší časový úsek na opožděnou zprávu. Pokud zpráva nedorazí ani do předem stanoveného časového limitu, musíme prohlásit, že druhý uzel je nedostupný. Nastává otázka, zda jsme schopni vyřizovat uživatelské požadavky bez nedostupných uzlů.

Systémy odolné vůči selhání jsou odolné vůči ztrátě některých uzlů a stále poskytují služby čtení nebo zápisu dat.

3.2 Apache Hadoop

Apache Hadoop [1, 11–12] je kolekcí open source softwarových frameworků pro distribuované ukládání a zpracování velkého množství dat napříč clusterem složeným z komoditního hardware. Hadoop je implementován ve více variantách, např. HortonWorks[1], cloudera[2], MAPR[3] a další. Ve světě nejvíce používanou implementací je ovšem Apache Hadoop, který je vyvíjen Apache Software Foundation a který je licencován pod Apache licencí 2.0[4].

Apache Hadoop se masivně používá např. v prostředí sociálních sítí, kde je používán k nejrůznějším účelům v různých podobách clusteru. Největší uživatelé provozují cluster, který obsahuje jednotky tisíc heterogenních uzlů. Mezi významné uživatele Apache Hadoop patří Amazon, American Airlines, Google, Hewlett-Packard, IBM, Netflix, Seznam, Spotify Yahoo! a další.
Apache Hadoop se skládá z mnoha modulů. Mezi základní patří:

- Hadoop Common pokrývající základní funkcionalitu, která se používá v ostatních modulech.
- Hadoop Distributed File System (HDFS), který umožňuje čtení nebo zápis rozsáhlých datových bloků a který je optimalizován právě pro velké datové bloky.
- Hadoop Yet Another Resource Negotiator (YARN), který spravuje a koordinuje cluster.
- Hadoop MapReduce, který implementuje model MapReduce a který pro svou práci využívá uvedené moduly.

[1] https://hortonworks.com/apache/hadoop/
[2] https://www.cloudera.com/products/open-source/apache-hadoop.html
[3] https://mapr.com/products/apache-hadoop/
[4] https://www.apache.org/licenses/LICENSE-2.0

Apache Hadoop lze nasadit na libovolný uzel, který provozuje jeden z podporovaných operačních systémů: CentOS[1] 6, 7, RHEL[2] 6, 7, Oracle Linux[3] 6, 7, SUSE[4] 11, Debian[5] 6, 7, Ubuntu[6] nebo Windows Server[7] 2008, 2012.

3.2.1 Historie

Úplným základem tohoto systému je Google File System [13] (GFS) publikovaný v roce 2003. První verze Hadoop je uvolněna o tři roky později. Tato verze zahrnovala jak implementaci souborového systému, tak i framework MapReduce pro třídění objektů. Hadoop se v této době používá především pro indexaci obsahu na webu. Postupně je Hadoop používán především v prostředí sociálních sítí. V toce 2011 dochází k uvolnění multiplatformního distribuovaného úložiště Apache Hadoop implementovaného v jazyce Java, která je v roce 2013 s uvolněním nové stabilní verze rozšířena o systém řízení zdrojů YARN. V prosinci 2017 pak vychází aktuálně poslední stabilní verze 3.0.0.

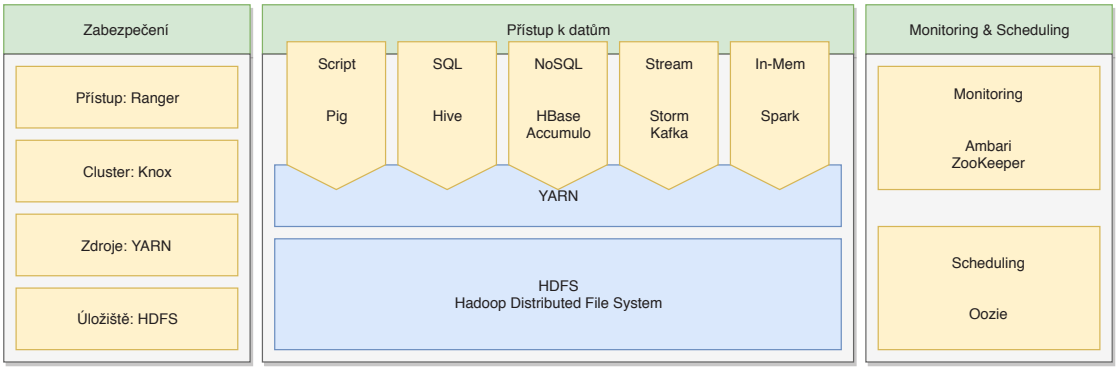
3.2.2 Komponenty

Základem celého distribuovaného systému je souborový systém HDFS, který je implementován v jazyce Java a který poskytuje škálovatelný a vysoce propustný přístup k zapsaným datům. Nad souborovým systémem je postaven framework YARN, který má na starosti správu systémových prostředků a plánování úloh, tzv. *job scheduling*. YARN je následně využíván komponentami určenými pro přístup k datům v HDFS. Mezi tyto komponenty patří např. Pig[8], Hive[9], HBase[10], Accumulo[11], Storm[12], Kafka[13], Solr[14] nebo Spark[15]. Klasifikace komponent je ilustrována na obrázku 3.1. Pro přístup k datům skrze scripty se používá Apache Pig, v infrastrukturách datových skladů s využitím SQL dotazů se používá Apache Hive, pro ukládání velmi velkých tabulek s podporou transakcí se používá Apache HBase nebo Apache Accumulo, proudové zpracování dat umožňují Apache Storm nebo Apache Kafka a pro strojové učení s použitím grafových algoritmů se používá Apache Spark postavený na jazyku Scala.

Pro monitorování stavu Hadoop clusteru se používá Apache Ambari[16], které poskytuje RESTful API a webové uživatelské rozhraní. Apache Zookeeper[17] se používá jako koordinační služba pro distribuované aplikace a služby.

Pro bezpečnostní účely se používají komponenty HDFS, YARN, Apache Knox[18] nebo Apache Ranger[19]. HDFS implementuje vlastnická práva k souborům a adresářům podobně jako známe v Linuxu, YARN spravuje Access Control List (ACL) pro řízení

1 https://www.centos.org
2 https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux
3 https://www.oracle.com/cz/linux/index.html
4 https://www.suse.com
5 https://www.debian.org/index.cs.html
6 https://www.ubuntu.com
7 https://www.microsoft.com/cs-cz/cloud-platform/windows-server
8 http://pig.apache.org
9 http://hive.apache.org
10 http://hbase.apache.org
11 https://accumulo.apache.org
12 http://storm.apache.org
13 https://kafka.apache.org
14 http://lucene.apache.org/solr/
15 http://spark.apache.org
16 http://ambari.apache.org
17 http://zookeeper.apache.org
18 https://knox.apache.org
19 https://ranger.apache.org

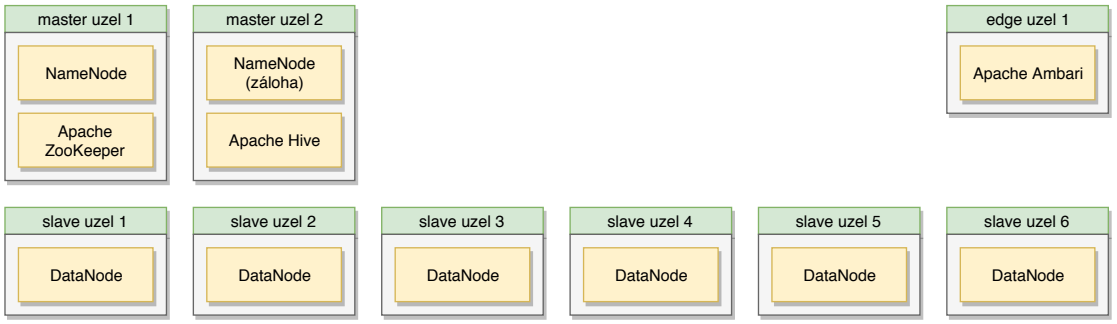


Obrázek 3.1. Hadoop ekosystém.

přístupu ke zdrojům v clusteru, Apache Knox má na starosti autentizaci uživatelů operujících s clusterem a Apache Ranger je bezpečnostní prvek spravující HDFS a aplikace přistupující k datům.

3.2.3 Architektura

HDFS je centralizovaným distribuovaným souborovým systémem postaveným na architektuře *master–slave*. Obsahuje tak pouze jediný *master* uzel (*NameNode*) a velké množství *slave* uzlů (*DataNode*). Cluster lze doplnit o sekundární NameNode, který se vytváří za účelem zálohy hlavního NameNode. Jedná se o persistentní kopii, která umožňuje restart HDFS s aktuální konfigurací v případech selhání hlavního NameNode. Příklad clusteru s více master uzly a několika slave uzly je ilustrován na obrázku 3.2, kde v clusteru kromě HDFS navíc nasazujeme aplikace jako Apache ZooKeeper, Apache Hive nebo Apache Ambari.



Obrázek 3.2. Příklad Hadoop clusteru.

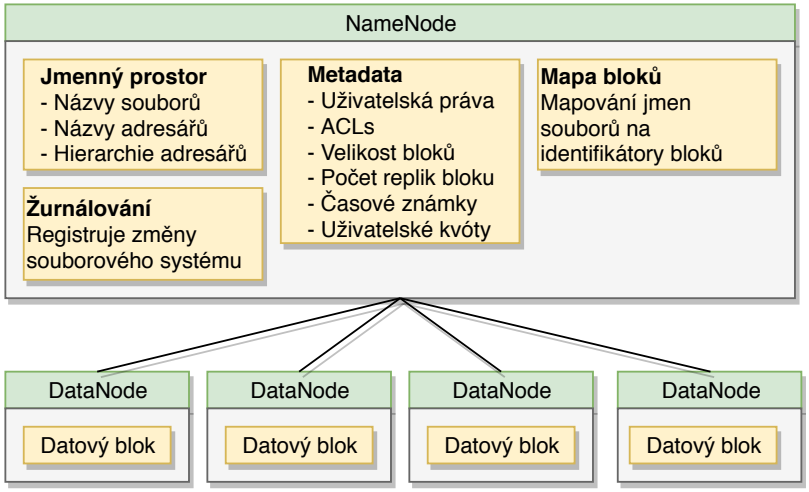
Úkolem NameNode je udržovat jmenný prostor (angl. *namespace*) všech adresářů a souborů použitím *inode*, které obsahují veškerá metadata včetně přístupových práv, diskových kvót, přístupových časů a dalších. NameNode má díky správě jmenného prostoru znalost o struktuře distribuovaného souborového systému. Uživatel tak veškeré požadavky posílá na tento uzel a pouze NameNode je schopen uživatelské požadavky vyřizovat. Současně je NameNode odpovědný za balancování zátěže clusteru, aby vše optimálně pracovalo.

Struktura, která popisuje jmenný prostor a která udržuje datovou strukturu, se nazývá *FsImage*. Tato struktura je uložena v lokálním souborovém systému NameNode uzlu a celá je nahrána v operační paměti (4 GB RAM jsou dostatečné). Její poslední stabilní verze se periodicky zálohují v podobě checkpointů. NameNode take zaznamenává všechny operace (transakce), které byly realizovány. Pro logování těchto událostí

slouží *EditLog*. Když dojde k selhání NameNode nebo během restartování NameNode, vezme se poslední verze FsImage, která je aktualizována o záznamy v EditLogu. Tím dojde k rekonstrukci posledního stavu NameNode.

V případě NameNode se jedná o výkonný prvek clusteru. Naopak v případě DataNode můžeme použít libovolný uzel, např. běžný domácí počítač, na kterém běží Apache Hadoop a na kterém se ukládají datové soubory v rámci lokálního souborového a operačního systému. DateNode nemá znalost o struktuře HDFS. Tato znalost je plně v kompetenci NameNode, který dokonce určuje, které datové bloky konkrétní DataNode uloží nebo odstraní. V důsledku toho se požadavky uživatelů vyřizují kontaktováním NameNode, který následně informuje DataNode o požadavku a veškerá data následně tečou síťovým spojením přímo do vybraného DataNode v případě požadavků na zápis dat nebo z DataNode v případě čtení dat.

Kompetence NameNode a DataNode jsou ilustrovány na obrázku 3.3, kdy hlavní odpovědností NameNode je udržování jmenného prostoru včetně hierarchie adresářů, jmen adresářů a souborů a správa metadat. Odpovědností DataNode je ukládání datových objektů, kdy jednotlivé datové objekty nemusí 1:1 odpovídat datovým souborům.



Obrázek 3.3. Kompetence NameNode a DataNode uzlů v HDFS.

Kromě NameNode a DataNode může být Apache Hadoop doplněn o tzv. *Edge* nebo *Utility* uzly, které se používají pro předání dat službám Hadoopu, které mají na starosti zpracování dat. Pro uzly mimo cluster mohou tyto uzly sloužit jako brána do clusteru.

3.2.4 MapReduce framework

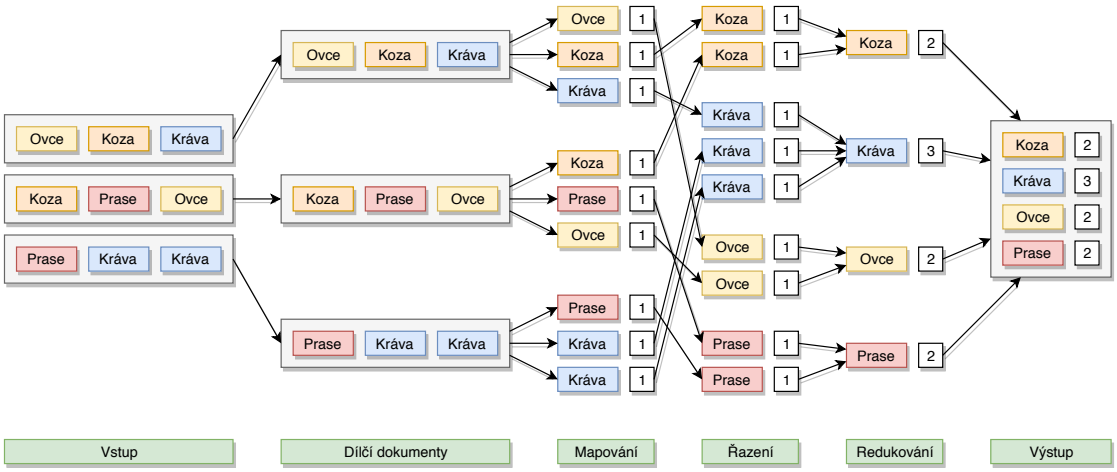
Motivací pro vytvoření frameworku byl PageRank problém [14], kdy Google zpracovával miliardy webových stránek. Idea byla založena na výpočtu důležitosti každé webové stránky. Měli k dispozici data o webových stránkách po celém světě na obrovském množství serverů. Jednalo se o soubory o velikosti TB nebo i větší, které se využívaly především pro čtení a méně pro zápis. Když už docházelo k aktualizacím obsahu souboru, jednalo se o připsání na jeho konec a téměř nikdy o přístup do středu souboru. Navíc se nacházeli v prostředí sítě, kde občas docházelo k výpadku spojení. Celkově se jednalo o problematické prostředí s častými chybami až na úrovni denní rutiny.

Posílání dat mezi jednotlivými uzly za účelem realizace výpočtu bylo a stále je drahou operací, pokud chceme posílat data v objemu řádově TB nebo vyšším. Nechceme přeposílat data, ale naopak přesunout možnosti výpočtu přímo k datům.

Model MapReduce je postaven na programovacím paradigmatu rozděl a panuj (angl. *divide–and–conquer*), kdy máme vyřešit složitou úlohu nad velkými daty (objem řádově TB nebo vyšší). Data rozdělíme do menších bloků, zpracujeme bloky, získáme částečná řešení a ta následně kombinujeme, čímž získáme řešení celého problému. Základem frameworku jsou funkce `map` a funkce `reduce`, jejichž implementaci musí dodat uživatel. O vše ostatní, tj. distribuci vstupních dat, naplánování vykonání jednotlivých úloh, sledování stavu výpočtu, komunikaci mezi uzly nebo o případy selhání uzlů v clusteru, se postará framework.

Úkolem mapovací funkce je rozdělení problému na menší části. Funkce je zavolána nad jedním vstupním dokumentem (nemyslí se soubor), kdy vstupními parametry funkce jsou ID dokumentu a jeho obsah. Cílem této funkce je generování libovolného počtu dvojic klíč, hodnota, čímž rozdělíme původní rozsáhlý problém na množství dílčích problémů.

Cílem funkce reduce je skládání jednoduchých dílčích problémů. Na vstupu funkce dostaneme jeden vygenerovaný klíč a k němu seznam hodnot. Cílem je zredukovat seznam hodnot, které jsou přiřazeny tomuto klíči. Funkce reduce postupně produkuje finální výsledek celé řešené úlohy. Funkce je volána na každý jednotlivý klíč.



Obrázek 3.4. Fáze MapReduce.

Obrázek 3.4 ilustruje jednotlivé fáze MapReduce na úrovni dat. Vstupní dokument se rozdělí na dílčí dokumenty, na které se volá funkce map. Mapovací funkce generuje dvojice klíč, hodnota, které seřadíme podle hodnoty klíče, čímž pro každý klíč získáváme seznam hodnot. Máme tak připraveno vše pro funkci reduce, která se volá opakovaně pro každý klíč, dokud nezískáme finální výsledek.

3.2.5 API

HDFS poskytuje více typů rozhraní pro práci se soubory. Mezi často používaná API patří HDFS Shell, FUSE[1], WebHDFS, libhdfs pro C/C++, Java API a rozhraní pro jazyk Python.

HDFS Shell je rozhraním na příkazové řádce, které umožňuje používat příkazy a volání, např. vytváření, zobrazování, přemísťování nebo odstraňování souborů nebo adresářů a podobně. Vše je prováděno pomocí superuživatele, tzv. HDFS superuživatel, kdy se jedná o analogický root účet, který známe z prostředí Linuxu, nebo administrátorský účet z prostředí operačních systémů Windows.

[1] `https://github.com/libfuse/libfuse`

HDFS také poskytuje API založené na linuxovém kernelovém modulu FUSE. Toto rozhraní umožňuje uživateli pracovat s virtuálním souborovým systémem, který je totožný se vzdáleným fyzickým adresářem (angl. *remote directory*).

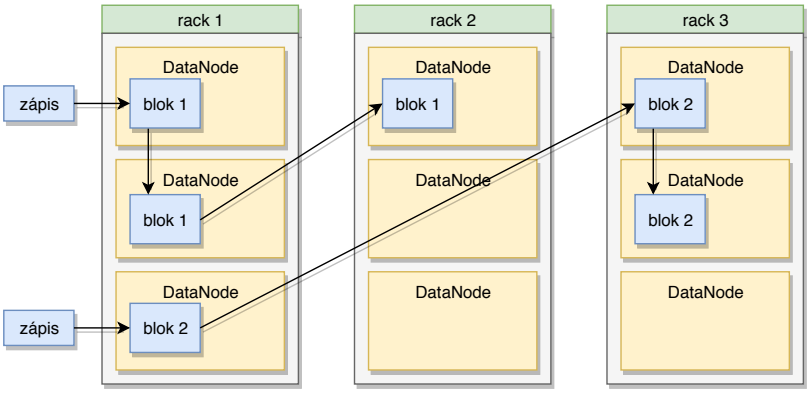
Dalším z řady rozhraní je HTTP rozhraní WebHDFS, které umožňuje procházet jmenný prostor a stahovat obsahy souborů. Toto rozhraní používá pro komunikaci vzdálené volání procedur[1] (RPC) a je vhodné především pro programátory, kteří vytváří webové aplikace. WebHDFS podporuje veškeré administrátorské operace pro práci se soubory, např. čtení nebo zapisování souborů, zobrazování adresářů, přidělování přístupových a vlastnických práv nebo konfiguraci replikačního faktoru.

K dalším API patří knihovny implementované v programovacích jazycích C (libhdfs C), Python a Java (Java API). Tyto knihovny umožňují operace čtení, zápisu nebo mazání souborů nebo vytváření a mazání adresářů.

V případě přístupu k souborům HDFS implementuje *write-once* a *read-many* model. Není povolen souběžný přístup pro zápis do stejného souboru. HDFS také umožňuje čtení ze souboru, který je aktuálně zapisován. Datová konzistence je v tomto případě zajištěna pomocí kontrolního součtu (angl. *checksum*), který je generován pro každý datový blok.

3.2.6 Distribuce dat

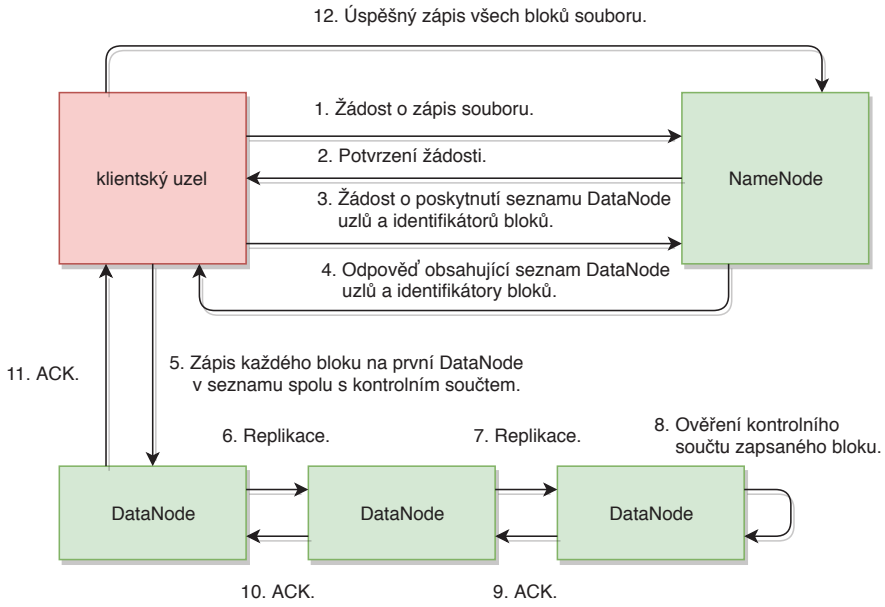
HDFS dělí data do bloků, které následně replikuje a distribuuje napříč mnoha uzly DataNode, přičemž respektuje pravidla pro umístění replik (angl. *placement policy*). Vše se řídí replikačním faktorem a zároveň žádný DataNode nesmí uchovávat více než jednu repliku od stejného bloku a ani žádný rack nesmí uchovávat více než dvě repliky stejného bloku. Obrázek 3.5 ilustruje tato pravidla. Pokud je datový blok uložen vícekrát, než určuje replikační faktor, NameNode určí repliku, která bude odstraněna. Snahou NameNode je odstranění repliky, která je na DataNode s nejmenším volným místem na disku a zároveň se NameNode snaží, aby neklesl počet racků, na kterých jsou repliky stejného bloku uloženy. Pokud je datový blok uložen méněkrát, než určuje replikační faktor, požadavek na vytvoření nové repliky je umístěn do periodicky prohlížené prioritní fronty. Stává se tak např. v případech selhání některého DataNode.



Obrázek 3.5. Pravidla pro umístění replik v HDFS.

Vyšší počet replik nejenže snižuje šanci ztráty dat v případě selhání DataNode, ale současně také zvyšuje propustnost operací čtení dat, protože je možné data číst z více různých DataNode uzlů. Replikační faktor lze konfigurovat. Běžně se používá replikační faktor hodnoty tři, kdy se ke každému bloku vytváří dvě další kopie.

[1] http://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf



Obrázek 3.6. Zápis souborů do HDFS.

Obrázek 3.6 ilustruje zápis souborů do HDFS. Klient nejprve zkontaktuje NameNode a zažádá o zápis souboru do HDFS. NameNode odpoví na základě jména souboru. Klient následně může zažádat NameNode o seznam DataNode uzlů a identifikátory bloků pro každý datový blok, které chce zapsat. NameNode odpoví a odešle seznam identifikátorů bloků a seznam DataNode uzlů. Klient iteruje skrze množinu bloků a každý blok zapíše na první DataNode v seznamu spolu s kontrolním součtem datového bloku. Uvnitř clusteru dochází k replikaci zapsaných bloků, aby bylo dosaženo požadovaného replikačního faktoru, přičemž poslední z DataNode uzlů navíc provede ověření kontrolního součtu. Pokud vše odpovídá, klient obdrží potvrzení (ack). Nakonec klient informuje NameNode o úspěšně provedeném zápisu dat.

3.2.7 Škálovatelnost

HDFS je horizontálně škálovatelné na úroveň tisíců DataNode uzlů v jednom clusteru, které mohou mít souhrnnou kapacitu v řádech petabytů (PB) dat. Neškáluje se pouze kapacita distribuovaného úložného prostoru, ale také možnosti paralelního zpracování dat nebo bandwidth. Obojí se škáluje lineárně s množstvím uzlů v clusteru. Vertikální škálovatelnost je umožněna použitím výkonnějšího hardware na jednotlivých uzlech.

3.2.8 Dostupnost

Základem Apache Hadoop je centralizovaný prvek NameNode, který musí být nahrazen v případě selhání. Pokud cluster obsahuje záložní centralizovaný prvek, lze systém obnovit do posledního známého stabilního stavu za cenu krátké nedostupnosti služeb. Dostupnost je tedy částečná, nicméně k výpadkům NameNode dochází zřídka. V případě selhání DataNode dostupnost replik pro operace čtení i zápisu zajišťuje replikační faktor spolu s dodržováním pravidel pro umístění objektů.

3.2.9 Odolnost vůči selhání

Předpokladem pro každý distribuovaný systém je odolnost vůči selhání. HDFS cluster se může skládat z tisíců uzlů. V tomto prostředí jsou selhání spíše pravidlem než výjimkou. Dochází k selhání síťové komunikace, jednotlivých uzlů, hardwarových komponent

nebo operačních systémů. Některá selhání jsou častá a dopředu se s nimi počítá. HDFS se spoléhá především na redundanci dat, kdy replikační faktor zajišťuje odolnost dat vůči selhání některých uzlů. Ztráta nebo selhání uzlu tak neznamená ztrátu dat, pokud existuje alespoň jedna další replika, která může být použita pro obnovení počtu replik tak, aby byl splněn požadavek daný replikačním faktorem. HDFS dále poskytuje nástroje pro diagnostiku souborového systému, kdy má správce systému možnost sledovat stav a vývoj zdraví clusteru.

Závažná jsou selhání NameNode, která se řeší nasazením záložního NameNode pro potřeby obnovení clusteru v případě selhání primárního NameNode. K obnovení se využije poslední verze FsImage a zaktualizuje se o záznamy v EditLogu, pokud sekundární NameNode plně neodpovídá selhanému NameNode v době krátce předcházející selhání.

3.2.10 Automatické procesy

HDFS cluster se může skládat z tisíců uzlů, kdy každá komponenta může s nenulovou pravděpodobností selhat. Automatické detekce selhání a obnovovací procesy jsou nutností. V rámci automatické detekce selhání každý DataNode v pravidelných intervalech posílá NameNode uzlu zprávu *HeartBeat*, která indikuje, že tento DataNode je funkční a online. V případě, kdy DataNode přestane vysílat HeartBeat zprávu, NameNode nedostane zprávu a ví, že DataNode je nedostupný nebo selhal. V takovém případě NameNode přesměruje všechny požadavky jdoucí do neodpovídajícího DataNode uzlu na jiný DataNode uzel a stejně tak vytvoří dodatečné repliky souborů, které byly tímto DataNode uzlem ukládány, aby byl obnoven replikační faktor. Když DataNode opět nastartuje, vygeneruje seznam všech jeho bloků a odešle *BlockReport* zprávu NameNodu uzlu.

NameNode neposílá data do DataNode přímo. Namísto toho pošle DataNode uzlu instrukce v rámci odpovědi na HeartBeat. Příkladem takové instrukce je: replikuj blok na další uzly, odstraň repliky na lokálním bloku, vypni uzel a další. HeartBeat zprávy mimo jiné poskytují statistické údaje, např. úložnou kapacitu, počet vykonávaných transferů dat a podobně, které NameNode uzlu umožňují rozhodovat se o balancování zátěže.

HDFS má snahu o balancování jednotlivých uzlů. Snaží se, aby všechny uzly byly zaplněny přibližně stejným počtem dat podle poměru použitého místa ku celkovému dostupnému místu na uzlu v rozsahu hodnot od 0,00 do 1,00, kdy hodnota 0,00 znamená prázdný uzel a kdy hodnota 1,00 znamená plný uzel.

3.3 Ceph

Ceph [12, 15–16] je open source projekt, který implementuje škálovatelné, dynamicky replikovatelné a distribuované objektové úložiště. Ceph umožňuje škálovatelnost až na úrovni tisíce připojených klientů, kteří přistupují řádově až k exabytům uložených dat. Škálovatelnost je možná jak do výšky, kdy se použije výkonný hardware pro klíčové prvky infrastruktury, tak do šířky, kdy je možné použít levný a běžně dostupný komoditní hardware bez ohledu na výrobce nebo nasazený operační systém. Ceph je možné instalovat na běžně používaných Linuxových systémech jako Debian, Suse, Ubuntu, CentOS a další.

Ceph je možné nasadit nejen jako objektové úložiště (angl. *Ceph Object Storage*), ale také pro účely virtualizace (angl. *Ceph Block Device*) nebo jako souborový systém (angl. *Ceph Filesystem*).

3.3.1 Historie

Ceph vychází z disertační práce Sage A. Weila [17] z roku 2007. Pro účely profesionální podpory Cephu je v roce 2012 založena společnost Inktank Storage. O dva roky později byl Inktank koupen společností Red Hat a v roce 2015 vznikla Ceph Community Advisory Board z přispěvovatelů do Ceph open source projektu: Canonical, CERN, Cisco, Fujitsu, Intel, Red Hat, SanDisk a SUSE.

První stabilní verze vyšla v roce 2012 pod názvem Argonaut. Do roku 2016 následovalo osm dalších verzí. Pro nás je zajímavá až LTS verze Jewel (v10.2.X), která obsahuje první stabilní verzi souborového systému CephFS. V roce 2017 vyšla verze Kraken (v11.2.X) následovaná LTS verzí Luminous [18] (v12.2.X), která zavedla Ceph Manager démona včetně rozhraní pro monitorovací nástroje a formát úložiště BlueStore[1]. BlueStore je od této verze stabilní a doporučen pro použití namísto dosud používaného interního souborového systému.

3.3.2 Architektura

Základem clusteru je Reliable Autonomous Distributed Object Storage [19] (RADOS), tj. spolehlivé, autonomní a distribuované objektové úložiště, které se skládá ze samo zotavujících a inteligentních uzlů. V nejjednodušším typu nasazení se cluster skládá ze dvou typů démonů: Ceph monitoru (`ceph-mon`) a Ceph OSD démonu (`ceph-osd`). Cluster lze také rozšířit o Ceph manager démony (`ceph-mgr`) nebo o Ceph metadata servery (`ceph-mds`) spravující metadata objektů, která jsou využívána např. pro vybudování hierarchické struktury objektů podobné struktuře souborového systému.

`Ceph-mon` je démonem, který monitoruje stav uzlů v clusteru a který udržuje kopii mapy clusteru, tj. seznamu uzlů v clusteru. Pro zajištění dostupnosti v případech selhání některého `ceph-mon` se běžně nasazuje více instancí tohoto démona.

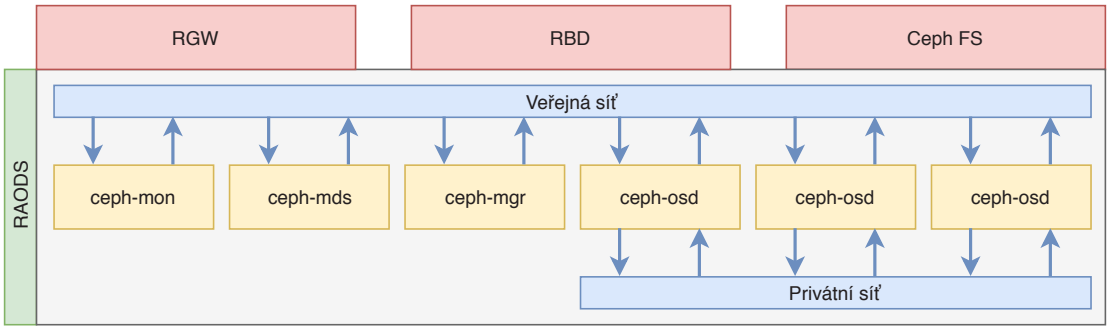
`Ceph-osd` zajišťuje I/O operace na pevných discích. Navíc také zajišťuje kontrolu vlastního stavu a stavu okolních `ceph-osd`. Zjištěné stavy démon odesílá `ceph-mon` v podobě zprávy HeartBeat. `Ceph-osd` jsou mapování na fyzická úložiště v poměru 1:1.

Základní jednotkou pro ukládání objektů jsou logické svazky označované jako pool, které mají nastaveny atributy jako např. vlastník přistupující k uloženým datům, počet tzv. *placement group* (PG, logická podjednotka logického svazku) nebo sadu pravidel CRUSH *ruleset* používanou během výpočtu umístění objektů. Každý logický svazek v clusteru musí být unikátně pojmenován, aby mohl být jednoznačně určen, a rozdělen na předem určený počet PG. Logické svazky se nemapují přímo na `ceph-osd`. Namísto toho se na jednotlivé `ceph-osd` démony dynamicky mapují PG a v případě rebalancování clusteru např. během přidání nebo odebrání `ceph-osd` mohou být PG přemístěny.

Od verze Luminous je součástí clusteru také `ceph-mgr`, který poskytuje webové GUI (*dashboard*) pro účely správy clusteru. `Ceph-mgr` poskytuje také rozhraní pro běžně používané monitorovací nástroje, mezi které patří např. Zabbix, Nagios[2], Prometheus[3] a další.

Pokud chceme používat cluster jako distribuovaný souborový systém, je nutné nasadit alespoň jednu instanci `ceph-mds`. Úkolem `ceph-mds` je správa metadat souborového systému. Příkladem metadat v tomto případě jsou např. označení adresářů, vlastníci souborů, přístupové módy, atributy souborů a podobně. Metadata jsou tak oddělená od objektů a umožňují efektivně provádět běžné operace souborového systému. `Ceph-mds` může běžet jako jediný proces nebo může být distribuován na více fyzických zařízeních,

[1] http://docs.ceph.com/docs/master/rados/configuration/bluestore-config-ref/
[2] https://www.nagios.org
[3] https://prometheus.io



Obrázek 3.7. Architektura Ceph Clusteru.

ať už z důvodu dostupnosti nebo škálovatelnosti pro případy rozsáhlého adresářového stromu.

Na obrázku 3.7 je ilustrována architektura Ceph clusteru, kdy základem je RADOS skládající se z instancí démonů `ceph-mon`, `ceph-osd`, `ceph-mgr` anebo `ceph-mds`. Nad tímto základem je postaveno objektové úložiště, block device pro účely virtualizace nebo distribuovaný souborový systém.

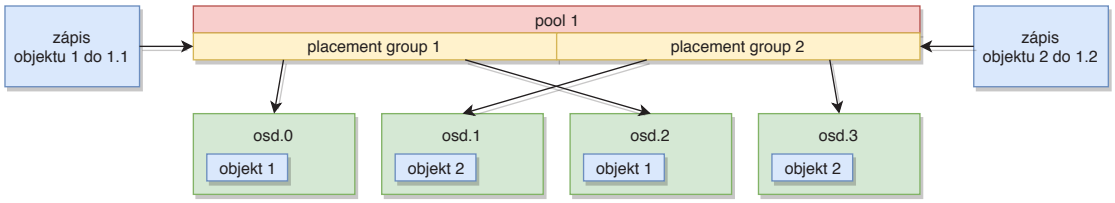
3.3.3 Algoritmus CRUSH

Controlled, Scalable, Decentralized Placement of Replicated Data [20] (CRUSH) je nástrojem, který `ceph-osd` a přistupujícím klientům umožňuje distribuovaný výpočet umístění objektu. Používá se v zejména případech čtení nebo zápisu dat (vyhledávání umístění objektu), rebalancování clusteru nebo dynamického zotavení se z chyb.

Použitím algoritmu odpadá nutnost centrálního prvku v clusteru, který by zaváděl slabý prvek a který by byl zatížený požadavky na umístění objektů ze stran `ceph-osd` i klientů. Výhodou neexistence centrálního prvku a možnost distribuovaného výpočtu umístění objektů nebo jejich replik je možnost významně lepší škálovatelnosti, teoreticky až neomezeně veliké. Algoritmus CRUSH se také stará o určení umístění replik objektů, čímž je zajištěna odolnost dat v případě selhání nebo nedostupnosti některých uzlů v rámci clusteru.

Klient po navázání spojení s clusterem nikdy přímo nezíská konkrétní záznamy o umístění objektů. Získá pouze aktuální kopii mapy clusteru, ze které může určit seznam `ceph-mon`, `ceph-osd` a metadata serverů v clusteru. Umístění objektu musí vypočítat. Ceph klient potřebuje znát dva parametry, aby mohl použít algoritmus CRUSH pro výpočet umístění objektu. Prvním parametrem je identifikátor objektu, tj. klíč objektu, a druhým parametrem je jméno logického svazku, do kterého má být objekt zapsán nebo ze kterého má být objekt čten. Pouze se znalostí dvojice klíč objektu, jméno logického svazku mohou následovat tyto kroky:

1. Klient zadá vstupní parametry identifikátor objektu a jméno logického svazku.
2. Ceph přijme identifikátor objektu a zahashuje jej.
3. Ceph vypočítá hash modulo počet PG, aby získal identifikátor PG.
4. Ceph načte identifikátor logického svazku se jménem, které klient vložil jako jeden ze vstupních parametrů.
5. Ceph vezme oba identifikátory a jako separátor použije znak dot (tečka), čímž vzniká identifikátor ve tvaru `POOL_ID.PG_ID`, např. `17.192`, kde `17` je identifikátorem logického svazku se jménem zadaným na vstupu a `192` je identifikátor PG získaný výpočtem z hashe identifikátoru objektu.



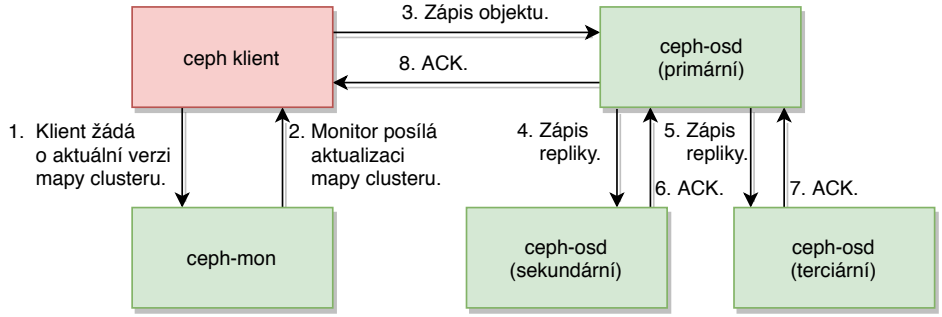
Obrázek 3.8. Průběh mapování objektů.

Výsledek vrácený algoritmem CRUSH umožňuje kontaktování `ceph-osd` pro účely čtení nebo zápisu objektu. Obrázek 3.8 ilustruje použití `PG_ID` pro identifikaci `ceph-osd` během ukládání objektu.

3.3.4 Distribuce a umístění objektů

Ceph umožňuje několik způsobů distribuce dat: klasickou replikaci dat nebo *erasure coding*.

V případě replikace se pro výpočet umístění replik objektů používá algoritmus CRUSH. Tento algoritmus tak používají nejen klienti v případě určení Primárního `ceph-osd`, ale také `ceph-osd` pro určení Sekundárních a následných `ceph-osd` pro ukládání replik objektů. Počet následných `ceph-osd` (následujících po primárním `ceph-osd`) je určen replikačním faktorem. Klient obdrží odpověď od primárního `ceph-osd` až ve chvíli, kdy jsou úspěšně uloženy všechny repliky. `Ceph-osd` zajišťují replikaci dat a tím zároveň i vysokou dostupnost dat a jejich bezpečnost. Obrázek 3.9 ilustruje replikaci objektu, kdy klient požádá o zápis objektu v clusteru a čeká na potvrzení (*ack*) až do chvíle, kdy jsou všechny repliky zapsány v clusteru.



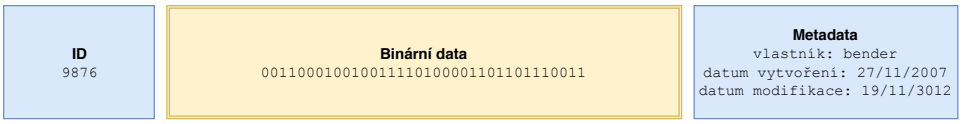
Obrázek 3.9. Zápis objektu v clusteru.

Logický svazek podporující erasure coding ukládá každý objekt v podobě, ve které je původní objektu rozdělen do `K+M` částí. `K` částí odpovídá původnímu obsahu objektu, zatímco `M` částí je rozšířením původního objektu za použití kódování. Všechny části jsou během zápisu umístěny na různé `ceph-osd`. Při čtení i zápisu objektů záleží na pořadí, a proto je ke každé části připojen atribut identifikující pořadí. Pokud chceme získat obsah původního objektu, stačí nám přečíst libovolných `K` z `K+M` částí, které použijeme pro obnovení původního objektu. Jakmile tedy získáme `K` částí nemusíme čekat na zbývající části a zrekonstruujeme původní objekt. Tento přístup je vhodný především pro zapisování objektů, protože zápis objektů je v porovnání s replikací rychlejší. Naopak pro operace čtení je vhodnější použít replikační faktor alespoň o hodnotě 2, protože replikace umožňuje lepší rozložení zátěže clusteru ve srovnání s *erasure coding*.

Aby mohl být objekt uložen, klient musí kontaktovat `ceph-mon` pro získání aktuální kopie mapy clusteru. Teprve poté může být použit algoritmus CRUSH pro výpočet

umístění objektu a kontaktování konkrétního `ceph-osd` za účelem zápisu objektu do příslušného pool. Způsob a místo umístění objektů v Ceph clusteru závisí na hodnotě replikačního faktoru, sadě pravidel CRUSH *ruleset*, velikosti logických svazků a počtu PG v logickém svazku.

Všechna data, která přichází od Ceph klientů do Ceph clusteru, jsou na úložných discích zapsána jako RADOS objekty, které mohou být jiné než původní datové objekty (včetně jejich počtu, kdy jeden datový objekt může být rozdělen na více RADOS objektů). Nezáleží přitom, jestli data přijdou přes objektové úložiště Ceph, blokové úložiště Ceph nebo přes souborový systém CephFS. Každý zapsaný RADOS objekt pak odpovídá jednomu souboru v rámci interního souborového systému (není to samé jako CephFS), ve kterém neexistuje žádná hierarchie adresářů. Používá se plochý jmenný prostor (angl. *flat namespace*).



Obrázek 3.10. Objekt včetně identifikátoru a metadat.

Jak je vidět na obrázku 3.10, kromě vlastního binárního obsahu má každý zapsaný objekt globálně (nejen napříč celým clusterem, ale i v rámci lokálního interního souborového systému konkrétního `ceph-osd`) unikátní identifikátor ID a nepovinně může obsahovat metadata, tj. množinu dvojic jméno a hodnota. Význam (sémantika) takových metadat není pevně stanoven a lze je libovolně definovat nebo používat. Příkladem metadat může být vlastník objektu, přístupová práva, datum vytvoření objektu a podobně.

3.3.5 API

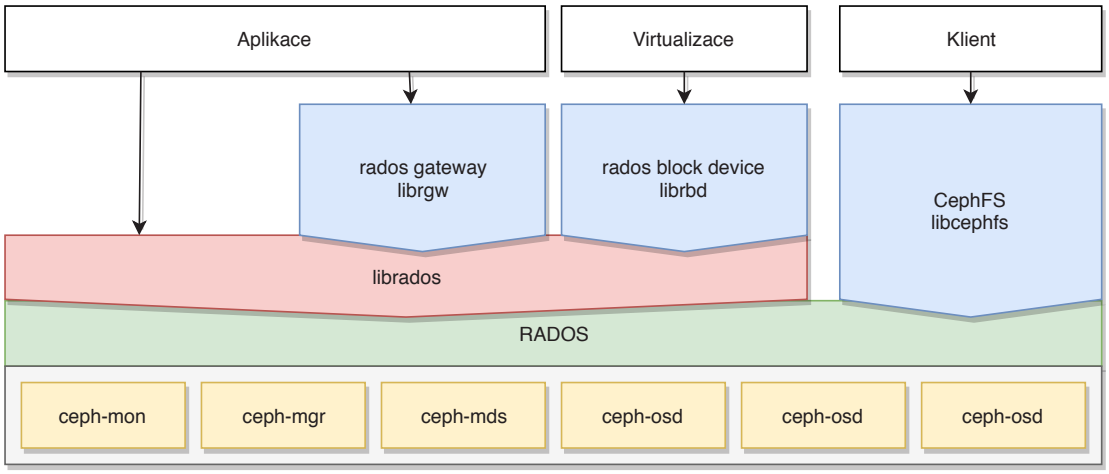
Ceph podporuje různá API, která umožňují cluster používat pro účely virtualizace nebo které s clusterem pracují jako s objektovým úložištěm, případně jako s distribuovaným souborovým systémem.

Obrázek 3.11 ilustruje jednotlivá API a knihovny k tomu používané. Pro komunikaci s clusterem je používán protokol librados, který tvoří základ pro nadstavby librbd, librgw a libcephfs používané pro virtualizaci, objektové úložiště a souborový systém.

Rozhraní Ceph Block Device (= RBD, rados block device) se využívá především pro virtualizaci a *cloud computing*, kde pracujeme s velkými bloky např. virtualizovaných uzlů s nasazenými operačními systémy. RBD pracuje s velikostí bloků a rozdělí je do menších bloků nebo také na více uzlů clusteru, aby byl zajištěn vyšší výkon virtualizovaného uzlu. Ceph podporuje jak kernel objekty (KO), tak i QEMU[1] hypervisor.

Rozhraní Ceph Object Storage (=RGW, rados gateway) poskytuje RESTful API kompatibilní s Amazon S3 nebo OpenStack Swift. Účelem tohoto rozhraní je přímé ukládání objektů a metadat do clusteru. Je nutné poznamenat, že objekty ukládané pomocí S3 nebo Swift rozhraní nutně nemusí 1:1 odpovídat objektům ukládaným v clusteru a že tyto objekty tak nejsou totožné. Může se stát, že původní S3 nebo Swift objekty jsou mapovány na množství interních RADOS objektů. Rozhraní kompatibilní s Amazon S3 a OpenStack Swift lze různě kombinovat. Není nutné používat stejné rozhraní pro čtení i zápis objektu. Můžeme použít jedno rozhraní pro čtení objektů a druhé rozhraní pro zápis objektů.

[1] `https://www.qemu.org`



Obrázek 3.11. Ceph API.

Posledním typem rozhraní je Ceph Filesystem (CephFS), které poskytuje POSIXový[1] souborový systém a které lze použít pomocí příkazu mount nebo jako souborový systém v uživatelském prostoru (FUSE). Adresáře a soubory souborového systému CephFS jsou mapovány na RADOS objekty, které se pak ukládají uvnitř clusteru. Zároveň se o objektech udržují metadata spravovaná `ceph-mds`, která jsou uložena v paměti `ceph-mds` a připravena pro rychlé procházení stromovou strukturou souborového systému. Průchod je pak mnohem rychlejší než potřeba vylistovat všechny objekty uložené v Cephu (znamenalo by obrovskou zátěž na cluster). Prochází se pouze metadata spravovaná `ceph-mds`.

Ceph také umožňuje vytvoření klienta používajícího přímo nativní protokol librados, který poskytuje paralelní přístup k objektům uložených v clusteru. Je možné využívat např. operace s logickými svazky, operace pro čtení nebo zápis objektů nebo operace pro modifikaci atributů objektů.

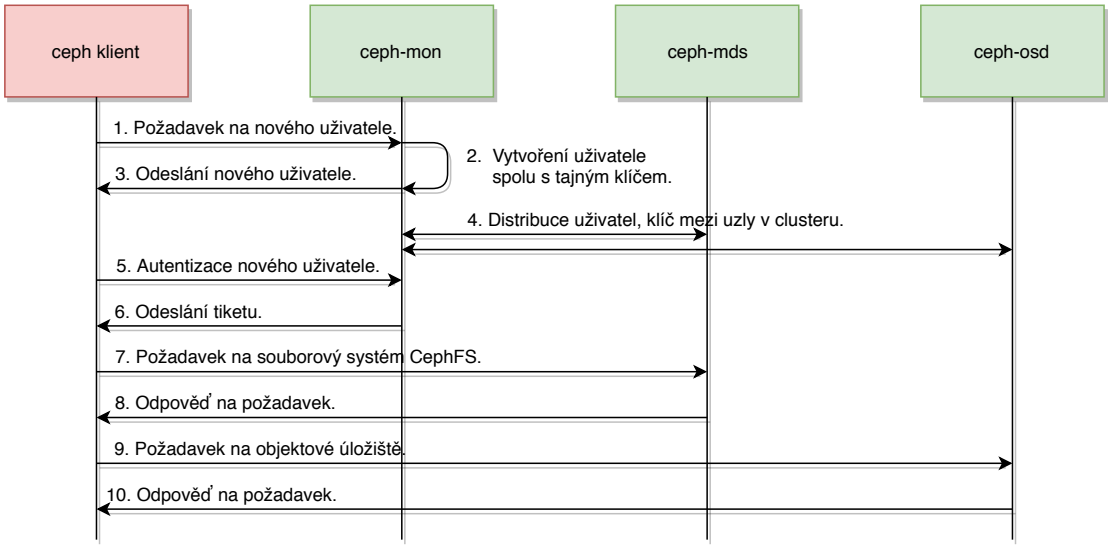
3.3.6 Bezpečnost

Ceph implementuje autentizační protokol cephx[2], který slouží pro autentizaci uživatelů a démonů v rámci clusteru a který přidává protekci proti útokům typu *man-in-the-middle*. Protokol cephx se nestará o šifrování dat během přenosu, zajišťuje pouze autentizaci. Protokol cephx pracuje podobně jako protokol Kerberos[3].

Základem pro použití protokolu cephx je sdílený tajný klíč, který je sdílen mezi uživatelem (klient) a všemi `ceph-mon` v clusteru (decentralizovaná autentizace). Veškerá komunikace mezi klientem a uživatelem tak může být šifrována tímto klíčem a ověřována na obou stranách. Obrázek 3.12 ilustruje použití protokolu cephx. Ceph klient požádá `ceph-mon` o vytvoření nového uživatele. `Ceph-mon` vytvoří nového uživatele a přidělí k němu tajný klíč. Dvojice uživatel, tajný klíč je rozdistribuována mezi `ceph-mon` v clusteru a zároveň odeslána zpět klientovi v zašifrované podobě. O přidělení tajného klíče novému uživateli musí žádat již existující uživatel, např. administrátor, tudíž pro šifrování ve zpětné odpovědi se použije tajný klíč existujícího uživatele. Je žádoucí, aby byl tajný klíč předán uživateli bezpečným způsobem. Jakmile má nově vytvořený uživatel přidělen tajný klíč, může pomocí tajného klíče šifrovat své požadavky a komu-

[1] http://standards.ieee.org/develop/wg/POSIX.html
[2] http://docs.ceph.com/docs/emperor/rados/operations/authentication/
[3] https://web.mit.edu/kerberos/

nikovat s `ceph-mon` v clusteru. Pro komunikaci s ostatními uzly v clusteru (`ceph-mds`, `ceph-osd`) je nutné autentizovat se u `ceph-mon`. Monitor po ověření uživatele vrátí tiket obsahující session klíč s omezenou časovou platností (zabraňuje zneužití tiketu po vypršení platnosti) a zašifruje jej s pomocí tajného klíče. Až tento session klíč lze použít pro komunikaci s `ceph-osd` nebo s `ceph-mds` uvnitř clusteru.



Obrázek 3.12. Komunikace mezi Ceph klientem a Ceph clusterem.

Je důležité, aby nedošlo k prozrazení tajného klíče před vypršením platnosti tajného klíče. Pokud dojde k prozrazení, útočník může zneužít platného tajného klíče k vytváření falešných požadavků pod podvrženou totožností oprávněného a zaregistrovaného uživatele Ceph clusteru.

Protokol cephx autentizuje probíhající komunikaci mezi klientem a servery Ceph clusteru. Každý požadavek, který následuje po počáteční autentizaci, je podepsána pomocí zašifrovaného tiketu, který lze na straně kteréhokoliv uzlu Ceph clusteru (`ceph-mon`, `ceph-osd`, `ceph-mds`) dešifrovat díky znalosti sdíleného tajného klíče a který je na těchto uzlech ověřitelný.

3.3.7 Škálovatelnost

Ceph eliminuje úzká hrdla škálovatelnosti, kdy součástí architektury Cephu není žádné centralizované rozhraní ani žádný centralizovaný prvek zajišťující komunikaci mezi uzly uvnitř clusteru. Naopak každý `ceph-osd` má znalost o dalších `ceph-osd` uvnitř clusteru. Stejnou informaci díky mapě clusteru získávají i Ceph klienti, kteří se ke clusteru připojují a požadují jeho služby. Díky tomuto přístupu je umožněna škálovatelnost na úrovni petabytů až exabytů uložených dat v clusteru, protože veškeré úlohy lze provádět efektivně s využitím všech hardwarových prostředků v rámci clusteru a provádět tyto úlohy distribuovaným způsobem napříč více uzly.

Ceph může spouštět doplňkové instance `ceph-osd`, `ceph-mds` a `ceph-mon` pro zajištění horizontální škálovatelnosti a vysoké dostupnosti. Příkladem škálovatelnosti v reálném nasazení je Ceph cluster nasazený ve výzkumném centru CERN[1], který se skládá z 225 serverů. Každý server má 48 disků s kapacitou 6 TB. Celkem jsou nasazeny 3

[1] https://www.openstack.org/videos/vancouver-2015/ceph-at-cern-a-year-in-the-life-of-a-petabyte-scale-block-storage-service

`ceph-mon`, 3 `ceph-mgr` a 10 800 `ceph-osd`, přičemž `ceph-mon` i `ceph-mgr` jsou nasazeny na uzlech obsahujících i `ceph-osd`.

Běžně dostupná a používaná úložná zařízení (HDD, SSD) jsou limitována z pohledu propustnosti dat, čímž je omezena výkonnost a škálovatelnost maximální rychlostí zápisu dat. Z tohoto důvodu Ceph i jiné úložné systémy podporují ukládání objektů po blocích, tzv. stripované ukládání objektů. Původní objekt se rozdělí na menší bloky, které se paralelně zapisují na více úložných zařízeních, aby se tím navýšila propustnost dat a celkově výkonnost systému.

Nejprve dochází k převodu formátu, který uživatelům poskytuje klient (block device image, RESTful objekty, CephFS souborový systém včetně adresářů), do RADOS objektů, které jsou vhodné pro ukládání do Ceph clusteru. Následně pak ještě dochází ke stripování RADOS objektů tak, aby se zvýšila propustnost dat a výkonnost systému. V případě standardních klientů se stripování vykonává automaticky. Pokud ale implementujeme vlastního klienta s pomocí knihovny librados, musí klient implementovaný na míru vykonat stripování sám.

Vylepšení výkonu s použitím stripování se projeví především u velkých virtuálních obrazů (bloků) nebo velkých objektů (např. video), protože jednotlivé stripy lze zapisovat paralelně do více úložných zařízení. Protože objekty jsou mapovány do různých PG a dále mapovány na různé `ceph-osd`, každý zápis nastává paralelně s použitím maximální rychlosti zápisu.

V případě zápisu na jediný disk je dalším limitujícím faktorem pohyb čtecí hlavy disku a propustnost tohoto zařízení. Pokud se zápis jednoho velkého objektu rozloží do více objektů na různá `ceph-osd` (a tím i různá úložná zařízení), Ceph může zredukovat počet pohybů čtecí hlavy a kombinovat propustnost dat většího množství úložných zařízení, aby dosáhl mnohem větší rychlosti zápisu nebo čtení.

3.3.8 Dostupnost

Ceph je založen na decentralizované architektuře. Ať už se jedná o `ceph-mon`, `ceph-osd`, `ceph-mds` nebo `ceph-mgr`, každý z těchto prvků je veden jako množinu uzlů, kdy v případě selhání některého uzlu nastoupí jiný, který selhaný uzel nahradí a převezme jeho odpovědnost. Tím je dosaženo odstranění slabého článku a zajištěna vysoká dostupnost.

Ceph nepoužívá centralizovaný vstupní prvek pro komunikaci mezi klienty a clusterem. Ceph umožňuje přímou komunikaci mezi Ceph klienty a distribuovanými `ceph-osd`. `Ceph-osd` zajišťují vytváření replik objektů a jsou zodpovědní za jejich umístění na dalších `ceph-osd`. S použitím replik objektů je zajištěna nejen jejich vyšší dostupnost (rozložení zátěže čtení stejného objektu mezi více `ceph-osd`), ale také integrita dat.

Algoritmus CRUSH hraje klíčovou roli v decentralizaci jednotlivých prvků clusteru. S pouhou znalostí mapy clusteru je možné tento algoritmus použít na libovolném Ceph klientu nebo `ceph-osd` pro výpočet umístění objektu (zápis, čtení, rebalancování, umístění repliky).

Než může klient začít zapisovat data, musí vždy kontaktovat `ceph-mon` a získat aktuální kopii mapy clusteru. Tyto požadavky od klientů může zpracovat jediný `ceph-mon`, ale takový přístup zavádí slabý článek do clusteru. Pokud dojde k selhání `ceph-mon`, žádný z Ceph klientů nebude schopen ani zapisovat, ani číst data uložená v clusteru. Je žádoucí, aby Ceph cluster obsahoval více než dvě instance `ceph-mon` (cluster monitorů).

3.3.9 Odolnost vůči selhání

Latence a různé chyby mohou způsobit, že některé `ceph-mon` nebudou disponovat aktuálním stavem clusteru (nebudou mít dostatečně aktuální mapu clusteru). Z tohoto

důvodu je nutná shoda napříč `ceph-mon` ohledně aktuálního stavu clusteru v podobě majoritní většiny za použití PAXOS[1] algoritmu. Většina znamená např. 1, 2:1, 3:2, 4:3 atd. Vzhledem k hledání shody na základě většiny `ceph-mon` je vhodný lichý počet aktivních `ceph-mon` v clusteru. Nemůže pak nastat patová situace, např. 2:2.

3.3.10 Automatické procesy

Uvnitř Ceph clusteru probíhá několik operací, které pro svůj běh vyžadují značné I/O prostředky a které nelze přerušit ani omezit jejich vliv na výkon clusteru. Mezi tyto operace patří peering, rebalancovací nebo obnovovací (angl. *recovery*) procesy a scrubbing. Mimo tyto operace probíhá ještě Heartbeating a samotná replikace dat.

Peering je časově náročným procesem, který probíhá po (re)startování `ceph-osd`. Součástí procesu je skenování celého disku. Pokud je `ceph-osd` umístěno na pomalém disku (HDD) a je zde uložen velký počet souborů, proces může trvat delší dobu. Skenování probíhá, aby se `ceph-osd` mohli shodnout na stavu PG (stav objektů a metadat v rámci PG), které jsou na těchto démonech umístěné. Pokud je během peering procesu odhalen problém nebo selhání, je toto selhání reportováno `ceph-mon`. Běžně se tato selhání vyřeší sama bez asistence správce Ceph clusteru,

Procesy rebalancování a obnovení dat jsou nejnáročnějšími operacemi v Ceph clusteru. Rebalancování clusteru probíhá v případech, kdy dojde ke změně velikosti clusteru, např. během samotného vytvoření clusteru nebo dále přidání, odstranění nebo pád `ceph-osd`. V důsledku změny CRUSH algoritmus přepočítává umístění všech PG. Pokud je nové umístění jiné než původní, dochází k migraci již uložených dat. S větším počtem `ceph-osd` v clusteru ubývá podíl objektů, které je nutné přemístit. Velké množství PG zůstává v původním umístění a nemění se ani nastavení PG. Mění se pouze místo přidělené PG na každém `ceph-osd` (zvětší se).

Scrubbing je proces, který se zaměřuje na zachování konzistence dat uložených v Ceph clusteru. Tento proces je možné vykonávat ve dvou režimech: mělký (angl. *light*) a hluboký (angl. *deep*).

Mělký scrubbing porovnává velikosti objektů a replik objektů a metadata patřící těmto objektům nebo replikám. Tento proces probíhá v základním nastavení jednou denně a jeho cílem je odhalení chyb nebo selhání interního souborového systému. Protože tento proces může být náročný na I/O prostředky, Ceph má snahu spouštět jej ve chvílích, kdy není zatížen. Je také možné určit, kdy a jak často se tento proces provádí. Hluboký scrubbing je těžší variantou, protože se porovnává i konzistence vlastních dat. Provádí se kontrolní součet uložených objektů a jejich replik za použití CRC32 algoritmu [21]. Cílem tohoto výrazně náročnějšího procesu je vyhledávání vadných sektorů na discích. V základním nastavení tento proces probíhá jednou týdně. I to lze konfigurovat.

Ceph má tendenci vykonávat mělký a hluboký scrubbing ve chvíli, kdy není Ceph cluster přetížen. Ale jakmile je proces jednou spuštěn, nelze jej přerušit, dokud nedokončí kontrolu.

Důležitým vnitřním procesem je tzv. heartbeating. `Ceph-osd` spolu komunikují a informují sousedy o svém stavu. Pokud jsou `ceph-osd` nedostupní, nemohou o tomto svém stavu informovat `ceph-mon`. Z tohoto důvodu `ceph-osd` v pravidelných časových intervalech posílají `ceph-mon` zprávu *MOSDBeacon*, tj. HeartBeat. Pokud `ceph-mon` neobdrží zprávu v předem určeném časovém úseku, je `ceph-osd` označen jako nedostupný. Stejně tak mají `ceph-osd` schopnost detekovat blížící se selhání sousedních `ceph-osd` a toto zjištění mohou posílat `ceph-mon`.

[1] https://en.wikipedia.org/wiki/Paxos_(computer_science)

3.4 GlusterFS

GlusterFS [12, 22–23] je open source distribuovaný souborový systém vyvinutý týmem Gluster Core pod licencí General Public License (GNU GPL). GlusterFS je založen na elasticitě, škálovatelnosti a paralelismu. Je možné přidávat nebo odstraňovat datové zdroje podle potřeby za běhu systému. Systém je lineárně škálovatelný, kdy zdvojnásobení počtu úložných zařízení také zdvojnásobí výkonnost systému. Škálovatelnosti dosahuje především díky eliminaci metadat [24]. Systém využívá paralelismu k dosažení maximálního výkonu skrze plně distribuovanou architekturu. Distribuované úložiště je řešeno na úrovni software, aby předcházelo vendor lock-in na úrovni hardware. GlusterFS tak lze nasadit na komoditní hardware nebo veřejné cloudové úložné infrastruktury. Jeho použití je výhodné především díky příznivému poměru cena výkon.

3.4.1 Historie

První verze systému Gluster byla vytvořena Anand Babu Periasamym, který založil společnost GLUSTER. V říjnu 2011 je tato společnost koupena Red Hatem, který tak získává distribuované řešení pro souborové systémy a který se stává hlavním správcem systému. GlusterGS je pojmenován jako Red Hat Server Storage. Nicméně v roce 2015 Red Hat získává také Ceph a GlusterFS je přejmenován na Red Hat Gluster Storage.

3.4.2 Architektura

GlusterFS je založen na decentralizované architektuře [25], která má klient-server design a která nepoužívá žádná metadata za účely lokalizace uložených objektů. GlusterFS ukládá data a jejich metadata, např. přístupová práva nebo jiné atributy souborů, ne však metadata určená pro vyhledávání souborů v souborovém systému, na několik různých fyzických serverů v clusteru, tzv. cihly (angl. *brick*), které pak spolu tvoří logické svazky. Na úrovni svazků lze pak konfigurovat stripování dat nebo replikaci dat, kdy jednotlivé stripy dat mohou být různě distribuovány anebo replikovány napříč celým clusterem. Stejně tak na úrovni logických svazků dochází ke většině operací souborového systému.

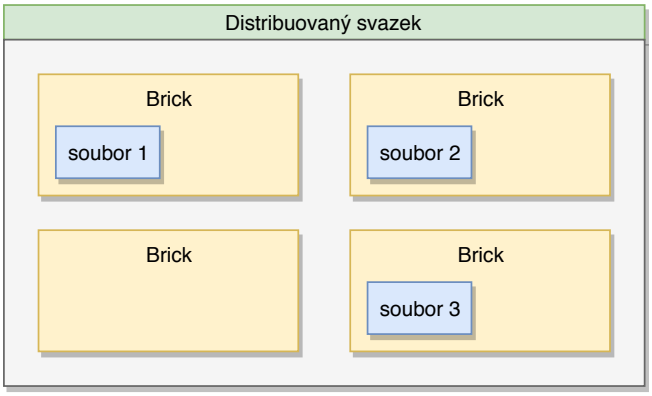
GlusterFS podporuje mnoho typů svazků, které se liší v závislosti na požadavcích uživatele. Některé svazky lze využít pro úpravu výkonnosti celého systému, jiné svazky jsou vhodné pro škálování kapacity úložiště nebo lze do jisté míry kombinovat oba přístupy. GlusterFS podporuje distribuované, replikované nebo stripované svazky a jejich kombinace.

Distribuované svazky jsou základním typem svazků. Každý soubor uvnitř svazku je distribuován na některé a pouze jediné z cihel. Nedochází zde k redundanci dat. Účelem tohoto typu svazku je snadná a levná škálovatelnost kapacity úložiště. Selhání cihly ale vede k úplné ztrátě dat, protože data nejsou replikována. Je tak nutné spoléhat se na hardwarovou úroveň úložiště, abychom předešli ztrátě dat. Obrázek 3.13 ilustruje tento typ svazku, kdy každý soubor je uložen právě na jednom fyzickém úložišti.

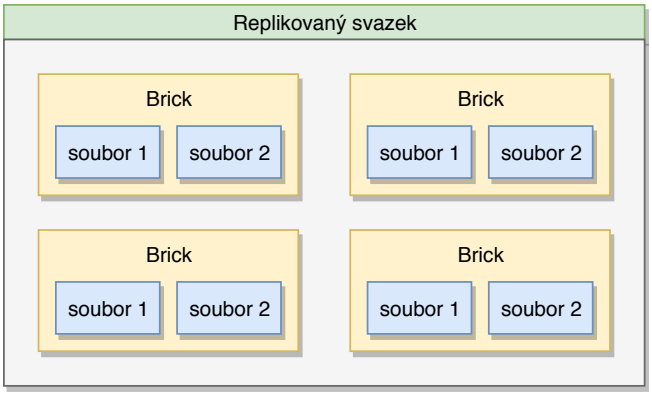
Replikované svazky se snaží zabránit možné ztrátě dat už na softwarové úrovni, kdy se nespoléhají až na hardwarové řešení s použitím vhodného typu RAID[1]. Data ve svazku jsou replikována na všech cihlách. Replikační faktor je nutné určit už při vytváření svazku s tím, že hodnota replikačního faktoru musí být totožná s počtem fyzických úložných zařízení ve svazku. V případě, kdy se svazek skládá alespoň ze dvou cihel, nedochází ke ztrátě dat během selhání některého fyzického úložného zařízení. Výhodou tohoto typu svazku je také větší propustnost dat z pohledu čtení objektů, ale vše za cenu

[1] https://en.wikipedia.org/wiki/RAID

nízké a neškálovatelné úložné kapacity logického svazku a vysoké ceny zápisu dat, kdy je nutné všechny zapisované soubory replikovat na všechny cihly ve svazku. Obrázek 3.14 ilustruje tento typ svazku, kdy každý soubor je uložen na všech cihlách.



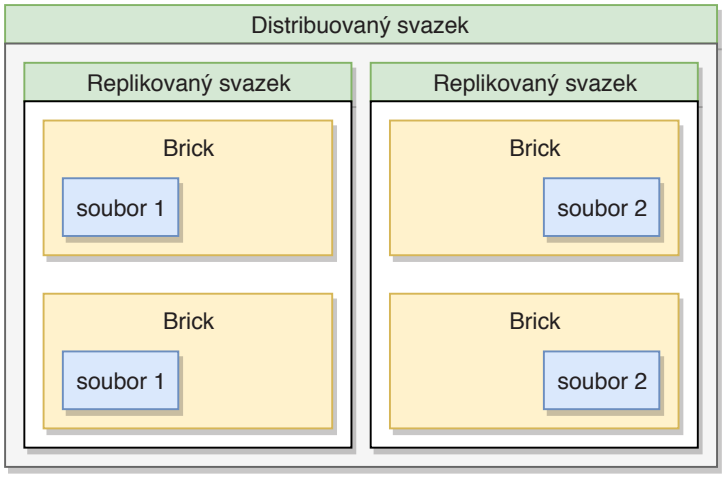
Obrázek 3.13. Distribuovaný svazek.



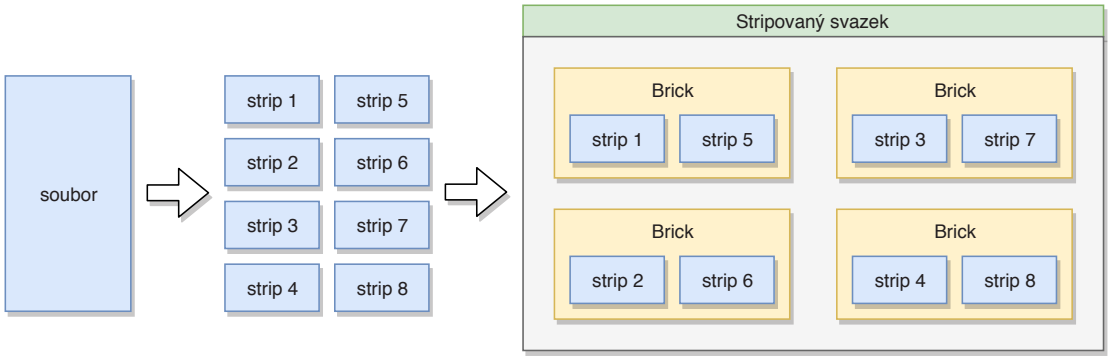
Obrázek 3.14. Replikovaný svazek.

Kombinací vlastností předchozích typů svazků vzniká distribuovaný a replikovaný svazek, kde jsou soubory replikovány napříč podmnožinou všech fyzických úložných zařízení uvnitř svazku. Počet cihel tak musí být násobkem replikačního faktoru, tzn. počet podmnožin krát replikační faktor musí být roven počtu cihel ve svazku. Při určování cihel, které uchovávají repliky, záleží na pořadí cihel ve svazku. Sousední cihly jsou vzájemnými replikami. Tento typ svazku se používá pro vysokou dostupnost dat, kdy ve svazku máme redundantní data díky použití replikace, a zároveň je zajištěna škálovatelná kapacita, protože repliky stejného souboru neukládáme na všechny cihly, ale pouze na jejich podmnožinu. Obrázek 3.15 ilustruje tento typ svazku, který je rozdělen do dvou podmnožin o stejné velikosti a každý soubor je uložen v tomto svazku vícekrát, aby bylo dosaženo požadovaného replikačního faktoru.

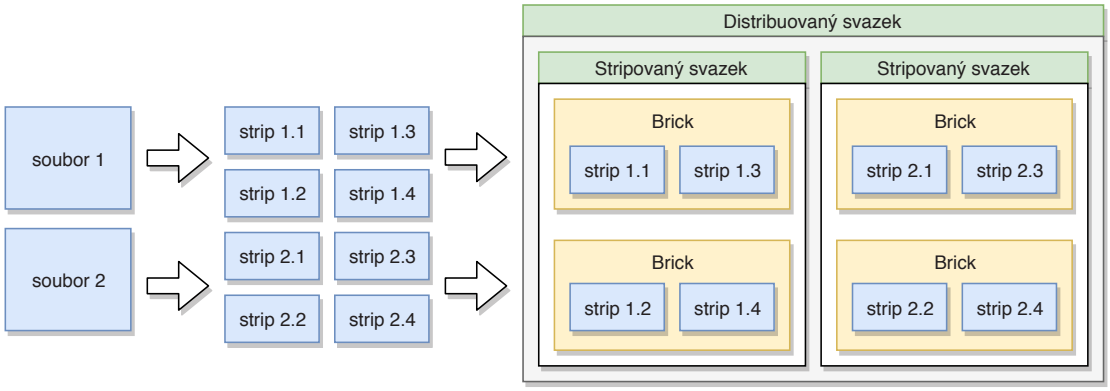
Stripovaný svazek je vhodný pro ukládání objemných souborů, ke kterým přistupuje velké množství klientů ve stejný čas. Velký soubor je rozdělen do množství stripů a jednotlivé stripy jsou umístěny na různých cihlách, čímž dojde k rozložení zátěže mezi více cihel ve svazku a tím také navýšení výkonu celého souborového systému. Tento typ svazku ale neposkytuje žádnou redundanci dat, tudíž ztráta fyzického úložného zařízení může vést k poškození celého souboru. Obrázek 3.16 ilustruje rozdělení souboru do stripů a uložení jednotlivých stripů na různé cihly ve svazku.



Obrázek 3.15. Distribuovaný a replikovaný svazek.



Obrázek 3.16. Stripovaný svazek.



Obrázek 3.17. Distribuovaný stripovaný svazek.

Posledním typem je distribuovaný stripovaný svazek. Velký soubor je také rozdělen do více stripů, ale jednotlivé stripy mohou být uloženy jen na podmnožině cihel. Počet cihel ve svazku tak musí být násobkem počtu stripů, na které se každý soubor dělí. Obrázek 3.17 ilustruje rozdělení souborů do stripů a následné umístění stripů na podmnožinu cihel ve svazku.

3.4.3 Elastic Hash Algorithm

Běžným způsobem lokalizace logického nebo fyzického umístění dat v distribuovaných souborových systémech je použití metadat, která popisují umístění dat. Tento způsob s sebou přináší zavedení centralizovaného prvku, např. serveru spravujícího metadata. Do distribuovaného systému tak může být zaveden slabý prvek, na jehož funkčnosti a výkonu závisí všechny budoucí operace souborového systému. Tento uzel se stává úzkým hrdlem z pohledu výkonu celého úložiště, a to především v případech, kdy roste počet souborů v souborovém systému nebo počet přístupů k těmto souborům. GlusterFS neodděluje metadata od dat. Nepoužívá žádný server spravující metadata, která mohou být použita pro účely lokalizace dat. Namísto toho používá algoritmické řešení Elastic Hash Algorithm (EHA) založené na hashovacím algoritmu Davies-Meyer [25].

Vstupem algoritmu EHA jsou cesta k souboru pathname a jméno souboru filename. Dvojice musí být unikátní v celém distribuovaném systému. Použitím EHA na tento vstup získáme číselný výsledek, např. hodnotu 40. Na základě výsledku následně vybereme logický svazek, kam uložíme data. Každý přistupující klient je tak pouze se znalostí EHA schopen vypočítat umístění dat v souborovém systému GlusterFS.

Pohybujeme se v distribuovaném prostředí, kde často dochází k selhání nebo chybám. Může dojít např. k selhání disků. Stejně tak můžeme na některý logický svazek v porovnání s ostatními svazky ukládat více souborů nebo větší soubory. V takových případech je nutná migrace některých dat v rámci logických svazků, které jsou postiženy selháním fyzických disků nebo nedostatkem úložného prostoru. Na práci EHA tyto skutečnosti nic nemění, protože EHA pracuje na úrovni logických svazků. Logickým svazkům se pouze přidělí více nebo méně fyzických úložných zařízení, čímž je problém vyřešen. Tato vlastnost se označuje jako elasticita a je použita v názvu algoritmu EHA.

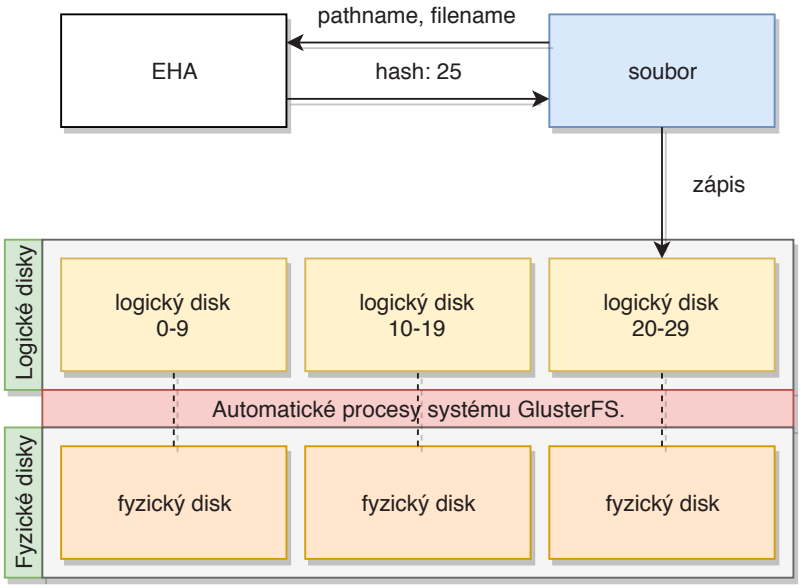
Uložené soubory lze přejmenovat, přičemž nové jméno souboru s sebou přináší i novou hodnotu EHA. Soubor se musí přemístit, aby byl nadále dohledatelný pomocí algoritmu EHA. Pokud se jedná o velký soubor, jeho přesun může být časově náročný, a proto je na novém místě vytvořen ukazatel na původní logické umístění souboru. Po úplném přesunu souboru na nové umístění je odstraněn ukazatel i soubor z původního umístění.

Obrázek 3.18 ilustruje princip EHA. Každému objektu je vypočítán hash na základě vstupu pathname, filename. Na základě tohoto hashe je vybrán logický svazek, kam je objekt následně uložen. Navíc je oddělena logická úložná vrstva od fyzické vrstvy, tudíž systém může pružně reagovat na výpadky uzlů nebo balancovat úroveň volného místa jednotlivých logických svazků v clusteru.

3.4.4 Distribuce dat

GlusterFS nereplikuje jednotlivá data, ale spoléhá se na RAID 1. Vytváří několik kopií celého fyzického úložného zařízení. Kopie umísťuje na některé z dalších fyzických úložišť ve stejném svazku. Ve výsledku za použití např. distribuovaného replikovaného svazku se tak svazek skládá z několika podmnožin fyzických úložišť, kde každá podmnožina obsahuje primární úložné zařízení a několik jeho replik. Tomuto faktu musí odpovídat počet úložných zařízení ve svazku, kterých musí být násobek požadovaného replikačního faktoru.

GlusterFS navíc umožňuje georeplikaci, která poskytuje asynchronní replikaci dat napříč geograficky rozdílnými lokacemi. Na úrovni georeplikace nedochází pouze k replikaci jednotlivých souborů nebo fyzických zařízení, ale replikují se celé svazky jako celek. Georeplikace se používá především jako prostředek zálohování v případech, kdy může dojít např. ztrátě celé lokální infrastruktury.



Obrázek 3.18. Princip EHA.

3.4.5 Škálovatelnost

GlusterFS je navržen tak, aby byl horizontálně škálovatelný pro navýšení výkonu systému i pro navýšení kapacity úložiště z pohledu objemu. Cílem je dosažení lineární škálovatelnosti. Pokud se zdvojnásobí počet úložných zařízení, musí to také znamenat dvojnásobení propustnosti dat za pozorování stejné doby odezvy (angl. *response time*). Přidávání pouze úložných zařízení nestačí. Aby bylo dosaženo škálování výkonu, musí se posílit i výkon procesoru a paměť RAM. Každé větší úložiště je náročnější na správu a vyžaduje lepší procesor a více paměti RAM, aby přidání úložné kapacity nezpůsobilo větší dobu odezvy.

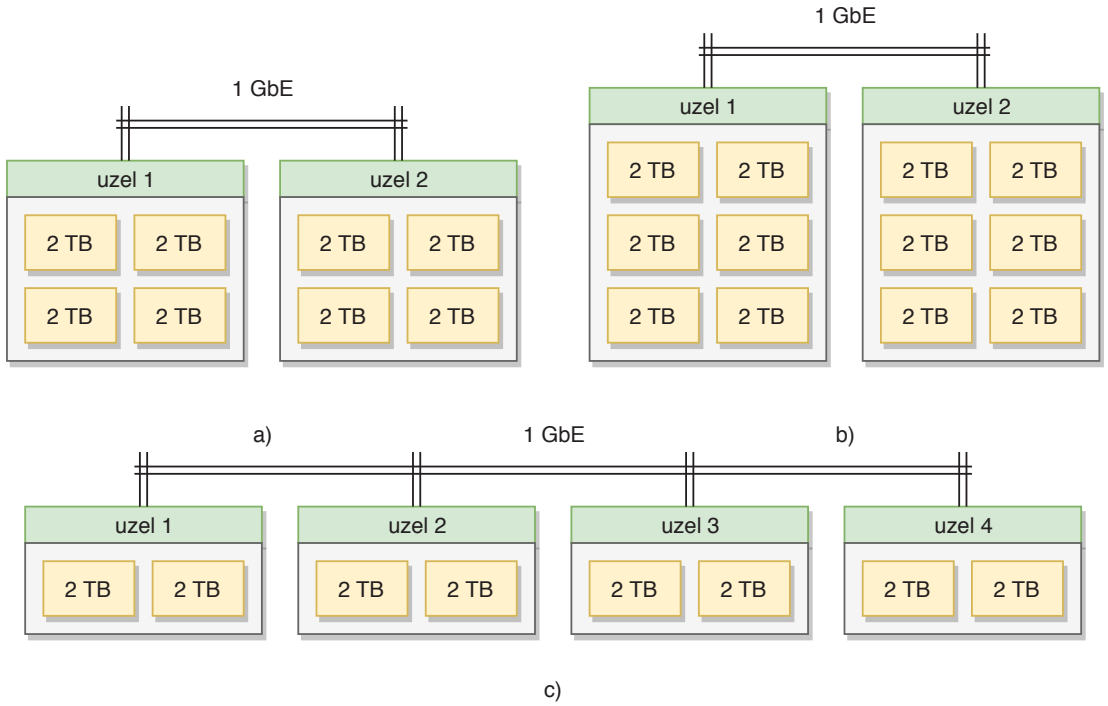
GlusterFS je škálovatelný skrze součet diskových kapacit, výkonu procesorů, paměti a I/O zdrojů velkého počtu komoditních strojů. Přidáním dalších úložných zařízení je možné škálovat kapacitu úložiště nebo přidáním výkonnějšího procesoru a I/O zdrojů je možné škálovat výkonnost systému. Obrázek 3.19 ilustruje tyto možnosti škálovatelnosti. Na obrázku a) vidíme původní systém, který se skládá ze dvou uzlů a čtyř disků na každém uzlu. Na obrázku b) škálujeme kapacitu úložiště přidáním dvou disků na každý uzel. Na obrázku c) škálujeme výkon systému přidáním dvou uzlů, čímž v systému máme více procesorů a I/O umožňující rozložení zátěže, ale také rozdělíme původních osm disků na všechny čtyři uzly, kdy každý uzel má nově pouze dva disky.

3.4.6 Dostupnost

Je vhodné použít replikační faktor alespoň o hodnotě dva, aby byla zaručena dostupnost dat i v případě selhání některého uzlu. Pro zajištění dostupnosti lze kromě redundance dat na úrovni software využít i prostředků na úrovni hardware s použitím RAID 5 nebo RAID 6. Pro posílení dostupnosti v případě lokálního výpadku je možné použít georeplikaci, čímž se zabrání výpadku služeb v případě plošných lokálních výpadků.

3.4.7 Odolnost vůči chybám

Když se uzel v GlusterFS stane nedostupným, je odstraněn ze systému a nahrazen jiným zařízením v rámci balancování volné úložné kapacity svazků. Spolu s tím je doporučeno



Obrázek 3.19. Škálovatelnost souborového systému GlusterFS.

použít replikaci alespoň na úrovni hardware, aby nedocházelo ke ztrátě dat v případě selhání uzlu.

3.4.8 Automatické procesy

GlusterFS implementuje automatické procesy v podobě překladačů (angl. *translator*). Překladače se používají nejen pro řízení distribuce a replikace dat a úpravu zátěže distribuovaného souborového systému, ale také pro úpravu požadavků uživatelů distribuovaného úložiště. Překladače jsou typu *storage*, *debug*, *cluster*, *encryption*, *protocol*, *performance*, *scheduler* nebo *features* [22], přičemž z pohledu výkonu systému jsou nejdůležitější cluster a performance překladače.

Základem pro řízení kapacity a výkonu GlusterFS je překladač *Distributed Hash Table*[1] (DHT), jehož úkolem je umístění dat na místo, kam nejen logicky, ale také fyzicky patří. Překladač DHT k tomu používá hashování, kdy každé cihle je přiřazen 32 bitový rozsah hodnot. Interval hodnot je pak porovnán s hashem přiřazenému ke každému souboru a na základě výsledku porovnání dojde k určení přesného umístění souboru.

Dalším důležitým překladačem je *Automatic File Replication*[2] (AFR), který pracuje nad rozšířenými atributy lokálních souborových systémů a sleduje operace nad soubory. AFR je nejen zodpovědný za replikaci dat napříč jednotlivými cihlami, ale také spravuje konzistenci replikací a umožňuje obnovení dat v případě některého selhání.

Většina procesů GlusterFS včetně replikace, stripování, balancování zátěže, rozvrhování úloh a dalších je implementována v podobě překladačů.

[1] https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3/html/error_message_guide/chap-distributed_hash_table_translator

[2] https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3/html/error_message_guide/chap-automatic_file_replication_translator

3.4.9 API

GlusterFS je souborovým systémem v uživatelském prostoru (angl. *userspace*). Aby souborový systém mohl komunikovat s virtuálním souborovým systémem jádra, tzv. kernel VFS[1], GlusterFS používá File System in Userspace (FUSE), kdy FUSE je kernelovým modulem, který umožňuje interakci mezi kernelem VFS a neprivilegovanými uživatelskými aplikacemi a k jeho API lze přistupovat z uživatelského prostoru.

Kromě mount modulu FUSE je také poskytováno RESTful API. Komunikující klient může číst nebo zapisovat soubory. GlusterFS také podporuje běžně podporované standardy jako NFS nebo SMB.

3.5 Porovnání vybraných distribuovaných systémů

Porovnání vybraných distribuovaných systémů provedeme na základě typu úložiště, architektury, způsobu lokalizace dat, distribuce dat, škálovatelnosti, odolnosti vůči selhání, poskytovaných API, licencí a poplatků za používání těchto systémů.

Apache Hadoop je hierarchickým souborovým úložištěm typu klíč-hodnota, Ceph je objektovým, blokovým nebo souborovým úložištěm, vše implementováno jako úložiště typu klíč-hodnota, a GlusterFS je pouze hierarchickým souborovým systémem. Pro účely archivace obrazových dat je vhodné použití libovolného z těchto poskytovaných přístupů s výjimkou blokového úložiště v případě Cephu, které je vhodné pro virtualizaci systémů a které je optimalizováno pro práci s bloky o velikostí minimálně v řádu GB.

Z pohledu architektury systémů vybíráme mezi centralizovanou architekturou Apache Hadoop, distribuovanou architekturou úložiště Ceph a decentralizovanou architekturou GlusterFS. Naším požadavkem je nepoužívat distribuované systémy, které obsahují centralizovaný prvek, neboť se tento prvek může stát slabým článkem celého systému z pohledu výkonu a dostupnosti. Apache Hadoop nesplňuje tuto podmínku. Úložiště Ceph lze nasadit tak, aby se skládal z více instancí démonů, čímž je distribuována zátěž a zajištěna dostupnost v případě selhání některého z démonů. GlusterFS je plně decentralizovaným systémem.

S architekturou souvisí způsob lokalizace dat. Apache Hadoop udržuje informace o souborovém systému v centralizovaném prvku NameNode. Veškeré požadavky na čtení nebo zápis souborů tak musí projít tímto centralizovaným prvkem. Naproti tomu Ceph i GlusterFS se spoléhají na algoritmický přístup, který umožňuje na základě jména objektu, resp. souboru v případě GlusterFS, určit umístění objektu. Ceph pro účely lokalizace objektů používá algoritmus CRUSH, zatímco GlusterFS se spoléhá na EHA.

Distribuce dat Apache Hadoop je založena na replikaci a stripování. Uložené soubory jsou stripovány do bloků a tyto bloky jsou následně replikovány napříč úložištěm. Replikací je dosažena lepší výkonnost operace čtení. Ceph se spoléhá nejen na replikaci a stripování, ale také na erasure coding. Erasure coding rozdělí původní objekt do bloků, ke kterým je vygenerováno několik kontrolních bloků v závislosti na nastavení erasure coding. Tento přístup zvyšuje výkonnost pro operaci zápisu, ale snižuje výkonnost operace čtení objektů. GlusterFS kromě replikace a stripování navíc umožňuje georeplikaci, která zajišťuje replikaci na úrovni celé infrastruktury na jiném geografickém místě.

Všechny vybrané systémy jsou škálovatelné jak horizontálně, tak vertikálně. Horizontální škálovatelnost je umožněna přidáním dalších uzlů, zatímco vertikální škálovatelnost použitím výkonnějšího hardware. Apache Hadoop, Ceph i GlusterFS škálují výkonnost systému lineárně s přidáním dalšího hardware nebo uzlu.

[1] https://en.wikipedia.org/wiki/Virtual_file_system

Apache Hadoop je pouze částečně odolný vůči selhání. V případě selhání centralizovaného NameNode je nutné jeho obnovení s použitím záložního NameNode, v důsledku čehož je služba krátce nedostupná. K selhání NameNode dochází zřídka, mnohem častější jsou selhání DataNode. DataNode jsou zodpovědná za ukládání souborů, které jsou uloženy na více různých DataNode. Selhání partikulárního DataNode tak pouze dočasně ovlivní výkon čtení objektu a je nutné obnovení replikačního faktoru nad postiženými soubory. Ceph je plně odolný vůči selhání, pokud je každý démon nasazen v dostatečném počtu instancí, tj. minimálně tři instance. Zároveň pro dosažení odolnosti vůči selhání je vhodné nasazovat tyto instance na různé uzly v clusteru. GlusterFS je plně odolný vůči selhání, pokud zvolíme vhodný model distribuce dat spolu s replikací např. na úrovni hardware.

Atribut	Apache Hadoop	Ceph	GlusterFS
typ úložiště	souborové	objektové, blokové, souborové	souborové
architektura	centralizovaná	distribuovaná	decentralizovaná
lokalizace dat	centralizovaná (NameNode)	algoritmická (CRUSH)	algoritmická (EHA)
distribuce dat	replikace, stripování	replikace, erasure coding, stripování	replikace, stripování, georeplikace
škálovatelnost	horizontální, vertikální, lineární	horizontální, vertikální, lineární	horizontální, vertikální, lineární
odolnost vůči selhání	částečná	úplná	úplná
api	FUSE, CLI, WebHDFS, libhdfs, Java API	FUSE, CLI, OpenStack Swift, Amazon S3, librados	FUSE, SMB, NTP
open source	ano	ano	ano
cenový plán	free	free, licence	free, licence

Tabulka 3.1. Porovnání vybraných distribuovaných systémů.

Důležitou roli při výběru distribuovaného systému hrají poskytovaná rozhraní API. Apache Hadoop nabízí pro přístup rozhraní FUSE, nástroje příkazové řádky, webové rozhraní WebHDFS a programovatelná rozhraní libhdfs a Java API. Ceph poskytuje rozhraní FUSE, nástroje příkazové řádky, programovatelná rozhraní kompatibilní s OpenStack Swift a Amazon S3 a knihovnu librados pro přístup k objektům. GlusterFS umožňuje pouze FUSE a příkaz mount. Pro nás jsou velice důležitá programovatelná rozhraní, která používáme skrze volání v upravené instanci CPVA. GlusterFS tak nevyhovuje našim požadavkům.

Vybrané distribuované systémy jsou open source a volně použitelné bez licenčních poplatků. Všechny tyto atributy jsou shrnuty v tabulce 3.1. Našim požadavkům definovaným v kapitole 2 „Výchozí stav“ odpovídá Ceph a vybíráme jej jako úložiště obrazových dat. Apache Hadoop nesplňuje požadavky, protože obsahuje centralizovaný prvek NameNode, který se může stát slabým prvkem v případě zápisu velkého množství dat.

GlusterFS neposkytuje programovatelné API, které vyžadujeme pro účely propojení s aplikací CPVA.

Kapitola 4
Archivace obrazových dat

V této kapitole nejprve představíme řešení jako celek, kdy koncept řešení vychází z požadavků představených v kapitole 2 „Výchozí stav“, a následně se zaměříme na jednotlivé prvky od vybraného distribuovaného úložiště a jeho zapojení do systému až po úpravy původního systému tak, aby vše spolupracovalo podle požadavků.

4.1 Koncept řešení

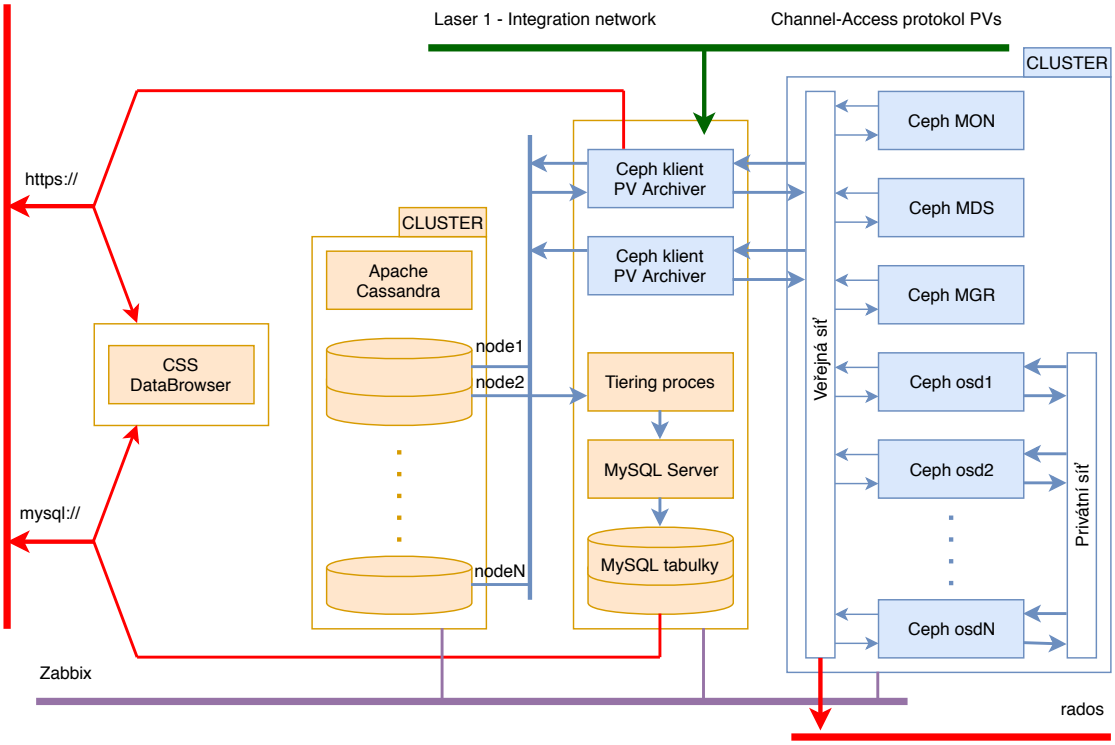
Našim cílem je použít co nejvíce stávajících technologií a nepřidávat nové komponenty, pokud to není nezbytně nutné. Většina systému představeného v kapitole 2 „Výchozí stav“ zůstává beze změny. Původní systém předpokládá s použitím přímé cesty mezi zdrojem dat a datovým úložištěm pro účely archivace obrazových dat a jiné cesty vedoucí do CPVA pro archivaci skalárních nebo vektorových dat. Vzhledem k tomu, že v původním systému je způsob archivace obrazových dat implementován a funkční pouze z 5 %, spojíme obě cesty a nově využijeme Channel Access protokol nejen pro přenos skalárních nebo vektorových dat (PV), ale také pro přenos obrazových dat.

Změna využívání Channel Access protokolu s sebou přináší nutné úpravy CPVA, který přijímá data přenášená tímto protokolem a který je optimalizován pro práci s daty menšího objemu a jejich archivací do databáze Apache Cassandra. Úprava původní verze CPVA zahrnuje nejen napojení na nové distribuované úložiště určené pro archivaci objemných obrazových dat, ale také úpravy zdrojového kódu a implementaci nových funkcionalit musíme:

1. Rozlišovat data určená pro zápis do databáze Apache Cassandra a data pro zápis do objektového úložiště.
2. Využít stávající Channel Access protokol pro posílání obrazových dat a vhodně využít podporovaných datových typů tak, aby nedošlo k poškození obrazových dat kódováním, přenosem nebo příjmem na straně CPVA.
3. Vytvořit klíč používaný pro identifikaci obrazových dat archivovaných v distribuovaném úložišti a uložit tento klíč v databázi Apache Cassandra,
4. Monitorovat stav distribuovaného úložiště, aby nedocházelo k pokusům o archivaci v případech, které ohrožují stav nebo stabilitu distribuovaného úložiště.
5. A zavést nové čítače (angl. *counter*) nebo upravit stávající čítače archivovaných i zahozených dat nejen pro statistické účely.

Nakonec musíme přidat nové distribuované datové úložiště, které odpovídá našim požadavkům na archivaci obrazových dat. Volíme řešení, které není původně plánovaným distribuovaným souborovým systémem, ale řešení, které umožňuje archivaci obrazových dat jako objektů a je objektovým úložištěm typu klíč, hodnota. Práce s objektovým úložištěm se pro naše účely v průběhu výběru distribuovaného řešení ukázala vhodnější variantou oproti původně plánovanému výběru distribuovaného souborového systému. Máme na výběr z více API pro čtení i zápis dat a můžeme objekty identifikovat pomocí jednoznačného a pro uživatele čitelného klíče. Naší volbou objektového úložiště je Ceph.

Zbývající části systému zůstávají beze změny, neboť pro archivaci nebo prohlížení PV se v průběhu používání ukázaly jako vhodné řešení. Celý systém s provedenými změnami je vidět na obrázku 4.1, kde jsou data určená pro archivaci přenášena společnou cestou pomocí protokolu Channel Access do upraveného CPVA. Námi implementované nové prvky CPVA rozhodnou o umístění dat v databázi Apache Cassandra nebo v objektovém úložišti Ceph a dojde k uložení dat nebo částí dat v určeném úložišti. V případě skalárních nebo vektorových dat malého objemu je volena databáze Apache Cassandra, pokud není uživatelsky určeno jinak. V případě obrazových dat se raw data ukládají do Cephu a identifikátor dat (klíč) je uložen do databáze Apache Cassandra. Pro zobrazování a prohlížení dat používáme Control System Studio a jeho plugin DataBrowser. Pro sledování stavu systému a datových úložišť pak používáme systém Zabbix.



Obrázek 4.1. Nový systém pro archivaci dat.

4.2 Předpoklady

Celý systém se skládá z několika komponent s různými nároky na hardwarové nebo softwarové prostředky. Pracujeme ve více vláknovém prostředí, tudíž základním předpokladem je více jádrový procesor s architekturou x86, který může zpracovávat více úloh souběžně. Minimální paměťové nároky jsou na všech uzlech alespoň 8 GB, protože používáme především aplikace vytvořené v programovacím jazyce Java nebo protože musíme mít dostatek operační paměti pro obnovovací a balancovací procesy uvnitř objektového úložiště Ceph. Důležitý je také dostatek volného místa na disku. Především instance `ceph-mon` jsou citlivé na nedostatek volného místa. Pokud zaplnění disku dosáhne hodnoty 95 % kapacity, dochází k výpadku `ceph-mon` a je nutné přidání volného místa. Podobně je tomu v případě `ceph-osd`, kteří slouží pro archivaci dat. V případě nedostatku volného místa u `ceph-osd` může dojít k chování, které ohrožuje stabilitu clusteru a integritu uložených dat. V případě `ceph-osd` se opět jedná o hodnotu 95 % zaplnění disku.

Abychom usnadnili správu celého systému archivace dat, používáme na všech uzlech jednotný operační systém CentOS 7. Pro tento operační systém existují distribuce veškerých používaných komponent, ať už se jedná o tzv. platform independent komponenty vytvořené v programovacím jazyku Java nebo komponenty vytvořené v jazycích C/C++.

V případě uzlů, na kterých běží technologie vytvořené v Javě, používáme Javu 8 SE Java Development Kit (JDK). Použití Java Runtime Environment (JRE) je dostačující, nicméně JDK poskytuje užitečné diagnostické nástroje.

CPVA je optimalizován pro archivaci dat do databáze Apache Cassandra. CPVA navíc musí být nasazen na uzlu `ceph-klient`, aby mohl archivovat data do vybraného objektového úložiště. U klientského uzlu se předpokládá instalace knihovny librados poskytující API pro objektové úložiště a znalost aktuální mapy clusteru. Knihovna librados je vytvořena v jazyku C, tudíž pro volání nativních funkcí v prostředí Java aplikace je nutná instalace knihovny Java Native Access (JNA) a také knihovna rados-java[1]. Knihovnu jna.jar není nutné manuálně přidávat, protože ji přibalujeme jako závislost v námi implementovaném modulu pomocí konfiguračního souboru `pom.xml`. Tento soubor pak tvoří základ pro vytvoření aplikace použitím nástroje Apache Maven[2].

Pro sestavení (angl. *build*) knihovny rados-java, našeho modulu a následně celého upraveného CPVA je nutné mít nainstalovaný a nastavený Apache Maven ve verzi 3.5.3 nebo novější.

Nakonec je důležité vytvořit logický svazek v objektovém úložišti, do kterého archivujeme obrazová data s pomocí upraveného CPVA, a nastavit vhodný počet PG pro tento svazek. V našem případě pojmenováváme pool jednoduše „data“ a používáme 512 PG, což nám téměř zaručuje uniformní distribuci objektů v poolu.

Teprve po splnění těchto předpokladů můžeme nasadit a spustit upravený CPVA a začít archivovat data do objektového úložiště.

4.3 Cassandra PV Archiver

CPVA hraje roli mostu mezi Control System aplikacemi, tj. zdroji dat, a databází Apache Cassandra. Pro účely archivace časových řad skalárních nebo vektorových dat využíváme lineární škálovatelnosti CPVA. Distribuujeme zátěž mezi všechny instance a manuálně přesměrováváme zdroje dat, které komunikují s konkrétní instancí a které generují experimentální data. Zdroje experimentálních dat v terminologii CPVA nazýváme *channels*.

4.3.1 Architektura

Obrázek 4.2 zobrazuje komunikaci mezi CPVA a sousedními komponentami. Systém umožňuje nejen archivaci dat, ale také jejich prohlížení. S použitím klientů jako např. Control System Studio lze k datům ve formátu JSON přistupovat skrze webové protokoly.

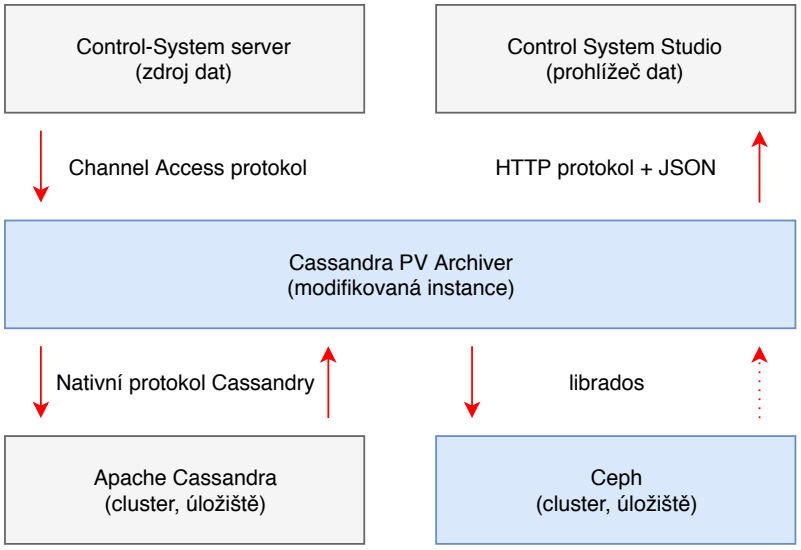
Apache Cassandra archivuje vzorky dat organizované v bucketech o velikosti 100 MB. Tato velikost je vhodné zejména pro skalární nebo vektorové hodnoty o velikosti maximálně jednotek kB. Vzhledem k faktu, že vytváření nového bucketu znamená značný overhead, používání CPVA pro archivaci obrazových dat o velikosti 1,5 MiB nebo větší je tedy nevhodné, neboť tento přístup by vedl k častému vytváření nových bucketů. Sami autoři CPVA nedoporučují použití systému za účely efektivní archivace dat s tokem

[1] https://github.com/ceph/rados-java
[2] https://maven.apache.org

v řádu MB za vteřinu přímo do databáze Apache Cassandra. Musíme implementovat jiné řešení, které je vhodné pro práci s většími obrazovými daty.

V případě obrazových dat neukládáme raw data přímo do databáze Apache Cassandra, ale používáme objektové úložiště, které je vhodné pro tento účel. Apache Cassadra pak slouží pouze pro ukládání identifikátoru dat v objektovém úložišti a metadat spojených s obrazovými daty. Klíče mají velikost řádově desítky B a splňují tak požadavky pro ukládání v databázi Apache Cassandra.

CPVA umožňuje rozšiřování skrze *extensions*. Můžeme implementovat vlastní podporu pro Channel Access protokol. My volíme jinou cestu. Přidáváme vlastní modul spravující archivaci dat v objektovém úložišti a voláme funkce námi implementovaného modulu uvnitř stávající implementace CPVA podle pravidel a potřeb archivace dat. Z pohledu výkonu aplikace je tento postup aktuálně dostačující. V případě, kdy se používání CPVA pro archivaci obrazových dat stane nedostačující, můžeme použít náš modul jako základ nové aplikace, která bude přijímat pouze obrazová data posílaná přes protokol Channel Access.



Obrázek 4.2. Architektura upraveného CPVA.

4.3.2 Nasazení

V minimalistické verzi používáme CPVA nasazený na jediném uzlu spolu s databází Apache Cassandra. Tento způsob využíváme především pro testování nově přidané funkcionality. V produkční verzi máme nasazený cluster skládající se z více instancí CPVA, které jsou synchronizovány pomocí NTP serverů, aby nedocházelo ke korupci dat z důvodu špatné synchronizace času.

Vybrané instance CPVA nasazujeme na ceph-klient uzly za účely propojení s objektovým úložištěm. Tento krok je nutný, aby vybrané instance CPVA měly přístup ke konfiguračním souborům Cephu, tj. k souborům `/etc/ceph/ceph.conf` a `/etc/ceph/ceph.client.admin.keyring`, a měly znalost mapy Ceph clusteru, kterou využívají skrze knihovnu librados při určování konkrétního uzlu v Ceph clusteru zodpovědného za uložení právě archivovaných obrazových nebo jiných dat.

4.3.3 Konfigurace

CPVA je konfigurován vlastním souborem `cassandra-pv-archiver.yaml` ve formátu YAML, do kterého navíc přidáváme atributy pro identifikaci poolu určeného pro

archivaci obrazových dat a atributy umožňující rozlišovat data, která chceme nebo naopak nechceme archivovat v Cephu. Případné umístění konfiguračního souboru `/etc/ceph/ceph.conf` není nutné zadávat do konfigurace CPVA, neboť tento soubor je na každém klientském uzlu umístěn na stejném místě.

CPVA neumožňuje dynamickou změnu konfigurace za běhu aplikace. Veškeré změny v konfiguračním souboru vyžadují restartování instance CPVA.

4.3.4 Zobrazení dat

Data zobrazujeme více různými způsoby. V případě skalárních nebo vektorových dat využíváme existujících webových protokolů nebo používáme klientskou aplikaci Control System Studio a plugin DataBrowser. Data a jejich změny můžeme pozorovat v reálném čase.

4.4 Channel Access protokol

Channel Access protokol používáme pro přenos dat mezi zdroji dat a aplikacemi, které data archivují. Protokol je založen na architektuře klient-server, kdy veškeré zdroje dat jsou serverovou částí a aplikace přijímající data jsou klientskou částí. V případě CPVA se jedná o klientskou část z pohledu Channel Access protokolu.

Protokol Channel Access je implementován v různých programovacích jazycích. V našem případě využíváme implementaci knihovny v programovacím jazyce Java, kdy se jedná o knihovnou EPICS Jackie. Velkou výhodou této implementace je možnost použití na různých platformách, aniž by knihovna obsahovala závislosti na knihovny používané pouze na specifických platformách.

Implementace protokolu nedovoluje vytváření vlastních datových typů pro přenos dat. Vždy musíme použít některý ze sedmi podporovaných datových typů: char, double, enum, float, long, short nebo string. Pro přenos bytových souborů lze využít všech typů, nicméně je nutné jednotlivé byty dat převést do některého z podporovaných datových typů a ideálně využít celé velikosti datového typu, kdy např. char použijeme pro přenos 2 B dat a podobně, aby nedocházelo ke zbytečné zátěži síťových prvků.

Abychom mohli přijímat data pomocí Channel Access protokolu, musíme implementovat rozhraní `ChannelAccessMonitorListener` deklarující dvě metody: `monitorEvent` a `monitorError`. Pro příjem dat používáme především metodu `monitorEvent`, která je volána pokaždé, kdy dojde k aktualizaci hodnoty dat.

Protokol je navržen především pro přenos skalárních nebo vektorových dat o menším objemu. Autoři protokolu v referenčním manuálu udávají maximální velikost přenášených dat, kterým je teoretická hodnota 2 GB. Nicméně tento protokol nedoporučují pro přenos tak velkých datových objektů nebo souborů, protože v rámci zpracování dochází k bufferování a dekódování obsahu zprávy, což je náročné na paměťové prostředky. Samotný přenos informace je navíc náročný na síťové prostředky, tudíž přenos dat takového objemu může znamenat ztrátu výkonu celého systému. Velikost našich obrazových dat ani zdaleka nedosahuje velikosti 2 GB pro každý přenášený objekt. Pracujeme s objekty o velikosti 1,5 MiB, tudíž použití tohoto protokolu je vhodné.

4.5 Distribuované úložiště Ceph

Požadavkům na archivaci obrazových dat více odpovídá objektové úložiště Cephu než původně plánované použití souborového systému, protože objektové úložiště nejen poskytuje různá API pro zápis nebo čtení objektů z aplikace, ale také umožňuje objekty unikátně identifikovat jednoduchým klíčem, tj. člověkem čitelnou sekvencí znaků.

K objektovému úložišti můžeme přistupovat pomocí různých API: librados nebo RESTful API na `ceph-rgw` uzlu kompatibilní s Amazon S3 a OpenStack Swift. Všechna tato rozhraní lze používat pro přímé ukládání objektů a jejich metadat do objektového úložiště. V případě použití knihovny librados se knihovna nestará o rozdělení ukládaných objektů do stripů, abychom dosáhli větší propustnosti během zápisu dat. Naopak Amazon S3 i OpenStack Swift API vykonávají stripování. V důsledku rozdělení objektů do stripů původně ukládané objekty nemusí nutně odpovídat 1:1 objektům, které jsou uloženy v objektovém úložišti. Pro archivaci dat používáme vestavěné API librados volanou skrze knihovnu rados-java, ale máme volnost pro použití některého z dalších API pro čtení objektů.

Ceph používáme pro archivaci obrazových dat, jejichž velikost je v průměru 1,5 MiB. Každému archivovanému objektu uvnitř objektového úložiště jsou přiřazena metadata, která mohou způsobovat overhead v případě, že bychom archivovali objekty o velikosti jednotek kB nebo menší. V takovém případě velikost metadat může být vyšší než velikost archivovaných objektů, tudíž zachováváme funkci CPVA, kdy skalární nebo vektorová data takto malých rozměrů archivujeme přímo do databáze Apache Cassandra.

Tabulka 4.1 zobrazuje démony, ze kterých se skládá náš cluster. Ve výsledku nasazujeme pouze instance `ceph-mon`, `ceph-osd`, `ceph-mgr` a v případě použití Amazon S3 nebo OpenStack Swiftu rozšíříme cluster ještě o `ceph-rgw` instance. Jak napovídá nasazení instancí `ceph-mgr`, používáme poslední stabilní LTS verzi Luminous.

démon	účel démona	počet instancí	otevřené porty
ceph-mon	koordinace clusteru	minimálně 1, alespoň 3 pro odstranění slabého prvku	6789
ceph-osd	úložiště objektů	tak mnoho, kolik máme fyzických úložných zařízení	6800 6801 6802 6803
ceph-mgr	sledování stavu clusteru	minimálně 1, alespoň 3 pro odstranění slabého prvku	7000
rgw	HTTP a REST rozhraní pro objektové úložiště	tak mnoho, kolik jich potřebujeme	fastcgi socket

Tabulka 4.1. Linuxové služby, které jsou nasazeny v našem Ceph clusteru.

4.5.1 Nasazení Ceph clusteru

Základem pro vytvoření clusteru je propojení uzlů do sítě. Na každém uzlu instalujeme jednotný operační systém za účely snadné administrace clusteru. Ceph cluster může být nasazen na uzlech s různými linuxovými distribucemi, nicméně takový přístup navyšuje nároky na údržbu clusteru.

Ceph cluster lze nasadit různými způsoby. Verze Luminous podporuje nástroje ceph-deploy[1], ceph-ansible[2], Crowbar[3], ceph-docker[4] nebo manuální nasazení s pomocí dokumentace [18].

My kombinujeme použití nástroje ceph-deploy spolu s manuálním nasazením některých prvků clusteru. Nejprve vybíráme uzel, který se stane výchozím bodem pro správu clusteru, tj. přidávání nebo odebírání uzlů nebo úpravy konfigurace přes modifikaci konfiguračního souboru `/etc/ceph/ceph.conf` a jeho následnou distribuci napříč clusterem, aby byla zachována konzistence tohoto souboru napříč všemi uzly v clusteru.

Pro případy testování nebo seznámení se tímto distribuovaným systémem postačí jediný virtualizovaný uzel vytvořený např. v programu VirtualBox[5]. Na jediný uzel nasadíme `ceph-mon`, `ceph-osd` instance a podle potřeby démony `ceph-mds` nebo `ceph-mgr`. V produkčním prostředí je používání virtualizovaných uzlů nevhodné a minimálně pro instance `ceph-osd` je vhodné použití plnohodnotných uzlů, aby bylo dosaženo dostatečné propustnosti dat a tím také výkonnosti clusteru.

Tabulka 4.2 popisuje námi navržený a používaný Ceph cluster. Nasazujeme vždy minimálně tři instance stejného typu démona, aby nedocházelo k zavedení slabého článku jako v případě nasazení jediného démona stejného typu. Zároveň je vhodné nasadit lichý počet démonů stejného typu, aby v případech nekonzistencí v clusteru se dala ustanovit většina uzlů, tzv. *kvórum*, která rozhodne o stavu. Minimální doporučený počet instancí stejného typu jsou tři. Na všech uzlech instalujeme jednotný operační systém, kterým je CentOS 7. Pro účely administrace používáme jednotného uživatele, např. deployceph. Použití uživatelského jména ceph není doporučováno, protože je vyhrazeno pro Ceph démony. Pro nasazení Cephu na jednotlivých uzlech je nutné používat `ssh` z administrátorského uzlu, kdy tento uzel používá přístup bez hesla, tzv. *password-less*. Za účely zvýšení bezpečnosti clusteru lze pak uživatele používaného pro administraci clusteru zakázat a povolovat podle potřeby. Je vhodné, aby Ceph cluster používal dvě různé sítě. Cluster používá privátní síť pro komunikaci `ceph-osd` démonů, která slouží především pro rozmístění replik nebo obnovovací procesy uvnitř clusteru, a veřejnou síť, která umožňuje komunikaci mezi klientskými uzly a `ceph-mon` nebo `ceph-osd` pro účely monitorování stavu clusteru a zápisu nebo čtení dat.

	uzel1	uzel2	uzel3	uzel4	uzel5	uzel6
ceph-mon	-	mon.0	mon.1	mon.2	-	-
ceph-mgr	-	mgr.0	mgr.1	-	mgr.2	-
ceph-osd	-	osd.0	osd.1	osd.2	osd.5	osd.8
	-	-	-	osd.3	osd.6	osd.9
	-	-	-	osd.4	osd.7	osd.10
ceph-rgw	ano	-	-	-	-	-
ceph-klient	ano	-	-	-	-	-
veřejná síť	ano	ano	ano	ano	ano	ano
privátní síť	-	ano	ano	ano	ano	ano
CPVA	ano	-	-	-	-	-

Tabulka 4.2. Mapování Ceph uzlů.

[1] https://github.com/ceph/ceph-deploy
[2] https://github.com/ceph/ceph-ansible
[3] http://crowbar.github.io
[4] https://github.com/ceph/ceph-docker
[5] https://www.virtualbox.org

4.5.2 Konfigurace clusteru

Ceph cluster je konfigurován skrze nastavení v souboru `/etc/ceph/ceph.conf`. Tento soubor je distribuován na všech uzlech Ceph clusteru. Změny v tomto souboru se projeví až po opětovném spuštění démona nasazeném na uzlu, kde došlo ke změně konfiguračního souboru. Pokud potřebujeme upravit konfiguraci za běhu clusteru, můžeme změny konfigurace injektovat [18]. Naše konfigurace souboru `/etc/ceph/ceph.conf` vypadá následovně:

```
[global]
fsid = 4af3d6bf-c850-4a65-8466-b35fa0c437e7
public_network = 192.168.6.0/24
cluster_network = 192.168.60.0/24

mon_initial_members = l1imagmon02,l1imagmon03,l1imagosd01
mon_host = 192.168.6.15,192.168.6.16,192.168.6.11

auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx

osd pool default size = 2
osd pool default min size = 2

osd pool default pg num = 32
osd pool default pgp num = 32

osd max object name len = 256
osd max object namespace len = 64

[osd]
osd journal size = 5120
filestore max sync interval = 20
filestore min sync interval = 10
osd crush chooseleaf type = 1

osd recovery max active = 1
osd max backfills = 1
```

Sekce global [18] obsahuje povinný atribut `fsid`, který identifikuje clusteru. Pro nasazení více instancí `ceph-mon` je nutné nastavení veřejné (*public*) a soukromé (*cluster*) sítě. Veřejná síť slouží pro klientský přístup ke clusteru, zatímco veřejná síť se používá pro vnitřní komunikaci a procesy clusteru, např. balancování rozmístění PG na `ceph-osd` nebo distribuci replik. My používáme dvě různé sítě, což s sebou přináší nároky na použití více síťových karet na každém uzlu. Dále tato sekce obsahuje množinu `ceph-mon` instancí, které jsou inicializovány během vytváření Ceph clusteru. Množina je reprezentována nejen *hostname* uzlů, na kterých jsou nasazeny `ceph-mon` instance, ale také jejich IP adresou. Zároveň používáme autentizační protokol *cephx*, který musí být součástí konfiguračního souboru. Atributy `osd pool default size` a `osd pool default min size` se vztahují k použitému replikačnímu faktoru o hodnotě. Říkáme, že chceme použít replikační faktor o hodnotě 2 a zároveň chceme, aby vždy v clusteru byly minimálně 2 repliky objektu. Používáme replikační faktor o této hodnotě, protože nám stačí jediná kopie pro případy selhání některého uzlu a protože neprová-

díme časté operace čtení, které pro vyšší výkon vyžadují vyšší replikační faktor. Zároveň za použití replikačního faktoru 2 zvyšujeme propustnost dat z pohledu počtu zapsaných objektů za sekundu. Zápis dvou replik trvá kratší dobu v porovnání se zápisem třech replik. Každý logický svazek, tj. pool, který vytvoříme v Ceph clusteru, musí mít minimálně 32 PG. Atributy `osd max object name len` a `osd max object namespace len` používáme, protože používáme souborový systém *ext4*[1]. Použití tohoto souborového systému vyžaduje omezení zmíněných atributů.

V sekci osd [18] nastavujeme atributy `ceph-osd` démonů. Používáme žurnálování o velikosti 5120 MB. Objekty v žurnálu se pak zapisují do fyzického úložiště používající technologii Filestore v intervalu mezi 10 a 20 sekundami. Atribut `osd crush chooseleaf type` upravuje strategii umístění replik. Náš cluster se skládá z uzlů, které obsahují jeden nebo více `ceph-osd` démonů, proto volíme toto nastavení. Poslední dva atributy používáme pro omezení počtu vláken procesů, které mohou zatěžovat výkon clusteru. Omezení počtu paralelně vykonávaných procesů tohoto typu sníží zátěž clusteru, nicméně toto nastavení zároveň sníží rychlost procesů, které pomáhají clusteru zotavit se ze stavů selhání.

4.5.3 Ceph-mon

Nasazujeme více instancí `ceph-mon` v plnohodnotné podobě, tj. nepoužíváme virtualizované `ceph-mon`, aby cluster neobsahoval slabý článek. Navíc je žádoucí, aby cluster obsahoval alespoň tři `ceph-mon` již během inicializace. V případě, že nasadíme dodatečné `ceph-mon` s časovým odstupem, může dojít k následujícímu scénáři:

1. Vytvoříme Ceph cluster s jedinou instancí `ceph-mon`, tzv. `initial mon`.
2. Připojíme se s Ceph klientem, čímž také získáme aktuální verzi mapy clusteru obsahující jediný `ceph-mon`.
3. Přidáme další dva `ceph-mon`. Ceph v tuto chvíli obsahuje tři `ceph-mon` a předpokládáme, že jsme vyřešili problém v případě ztráty některé z těchto instancí.
4. Odpojíme `initial mon` nebo dojde k jeho ztrátě z hardwarové nebo softwarové příčiny.
5. Pokusíme se o opětovné připojení se s Ceph klientem. Ceph klient se pokouší připojit k `ceph-mon`, o kterým má informaci v mapě clusteru získané při předchozím připojení se ke clusteru. Ceph klient se nemůže připojit, protože podle mapy clusteru zná pouze `initial mon`, který je v tuto chvíli nedostupný.

Kvůli Paxos algoritmu je důležité, aby byly `ceph-mon` časově synchronizovány. Paxos spoléhá na časové známky map uvnitř clusteru. Interní kontrola ověřuje, že maximální rozdíl mezi běžícími monitory není vyšší než 0,05 sekundy [21]. Tuto hodnotu lze upravit v nastavení. Za účelem synchronizace monitorů je vhodné používat NTP. Pokud nejsou Ceph monitory synchronizovány, objeví se `HEALTH_WARN` clock skew detected on `mon.X`, kdy je časový rozdíl mezi `ceph-mon` vyšší než povolená mez.

Zásadní podmínkou, aby `ceph-mon` pracoval, je dostatek volného místa na uzlu, kde je nasazena instance `ceph-mon`. Jako interní databáze `ceph-mon` se používá LevelDB [21], která je vysoce citlivá na nedostatek místa na disku. V případě, že zbývá 5 % místa na disku nebo méně, dojde k ukončení činnosti monitoru a monitor se vypne. Takto ukončený `ceph-mon` již nelze znovu spustit a musí být zcela nahrazen novou instancí `ceph-mon` na jiném uzlu v clusteru.

[1] https://en.wikipedia.org/wiki/Ext4

4.5.4 Ceph-osd

Veškeré námi fyzické disky používané pro úložiště, tj. spravují je `ceph-osd` démoni, mají stejnou kapacitu, což umožňuje lepší výsledky při dosahování rovnoměrné distribuce archivovaných dat.

Ceph-osd jsou rozmístěni na všech uzlech v clusteru. Používáme mapování jeden `ceph-osd` démon za každé fyzické úložné zařízení na uzlu, tudíž máme nasazeny i tři nebo více `ceph-osd` démonů na uzlu. Tomuto rozložení `ceph-osd` démonů také přizpůsobujeme mapu clusteru a v konfiguračním souboru `/etc/ceph/conf`.

4.5.5 Ceph-mds

Nenasazujeme `ceph-mds` uzly v našem clusteru, protože nepoužíváme Ceph jako souborový systém. `Ceph-mds` slouží pro správu metadat objektů, díky čemuž vzniká hierarchická struktura podobná struktuře v souborovém systému. Tu my nepotřebujeme.

4.5.6 Ceph-rgw a ceph-klient

Námi upravený CPVA vyžaduje pro správný běh nasazení na uzlu, který má znalost mapy clusteru. Ideálním kandidátem je Ceph klient uzel s nainstalovanou knihovnou librados pro přístup, tj. zapisování a čtení objektů, do objektového úložiště Cephu. Alternativním způsobem je použití jiného API než librados, např. Amazon S3 nebo OpenStack Swift. V takovém případě je nutné použití RGW uzlu, který poskytuje rozhraní pro přístup pro objektové úložiště a zároveň podporuje tato API. Pro nás je v tuto chvíli výhodnější používání librados, protože svazujeme instance CPVA s instancemi Ceph klientů, čímž máme volnější ruce pro škálování zátěže. Můžeme zátěž dynamicky balancovat pouhým přesměrováním zdroje obrazových dat na jiný uzel přes grafické nebo RESTful rozhraní CPVA. Kromě tohoto způsobu balancování využíváme librados podpory paralelní a asynchronní komunikace s objektovým úložištěm.

Ceph klienty nepoužíváme pouze pro nasazení upravených CPVA a archivaci dat do objektového úložiště, ale také pro správu, tj. vytváření nebo mazání logických svazků v objektovém úložišti.

Knihovna librados skrze metodu `write(key, value)` umožňuje zápis do objektového úložiště. Návratovým typem této hodnoty je `void`. Metoda nevrací informaci ani o úspěšně, ani o neúspěšně provedeném zápisu objektu. Klíč objektu musíme určit dopředu. Klíč by měl být unikátní napříč clusterem, aby nedošlo k přepsání hodnot. Tento námi definovaný klíč následně uložíme do databáze Apache Cassandra a později jej používáme pro listování archivovanými obrazovými daty. V případě potřeby můžeme archivovaným objektům přiřadit námi definovaná metadata a modifikovat jejich atributy *XATTRs*.

4.5.7 Ceph-mgr a sledování stavu

V clusteru nasazujeme instance `ceph-mgr`, které nám umožňují sledovat stavy zaplnění jednotlivých logických svazků, distribuci PG na jednotlivé `ceph-osd` démony, sledovat zátěž jednotlivých `ceph-osd` démonů během zápisu dat a celkový stav clusteru. Tyto a mnohé jiné metriky lze sledovat pomocí grafického rozhraní přímo `ceph-mgr` nebo lze použít RESTful API `ceph-mgr` instance a tyto metriky sledovat přes systémy určené pro sledování systémů, např. Zabbix.

Kromě `ceph-mgr` používáme také množství jednoduchých CLI nástrojů, které nám umožňují sledovat stav clusteru. Monitorujeme tak cluster z klientských uzlů např. voláním nástrojů `ceph quorum_status |format json-pretty`, `ceph -s`, `ceph osd tree`, `ceph df` a dalšími. Nástroj `ceph quorum_status` zobrazí stav všech instancí `ceph-mon`

v clusteru. V případě selhání některého monitoru tento stav vidíme a můžeme se k monitoru připojit a pokusit se o manuální opravu minoritních selhání. Nástroj `ceph -s` zobrazuje aktuální stav clusteru. Výpis vypadá např. takto:

```
cluster:
  id:     4af3d6bf-c850-4a65-8466-b35fa0c437e7
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum l1imagosd01,l1imagmon02,l1imagmon03
  mgr: l1imagmon02(active), standbys: l1imagosd02, l1imagmon03
  osd: 11 osds: 11 up, 11 in
  rgw: 1 daemon active

data:
  pools:   5 pools, 640 pgs
  objects: 4118k objects, 5999 GB
  usage:   12023 GB used, 8459 GB / 20483 GB avail
  pgs:     640 active+clean

io:
  client:   125 MB/s wr, 0 op/s rd, 86 op/s wr
```

Zobrazí se nám ID clusteru, stav clusteru a seznam aktivních služeb, tj. informace o `ceph-mon` a kvórum těchto monitorů, seznam uzlů s nasazenými `ceph-mgr` instancemi, počty instancí `ceph-osd` a počty `ceph-rgw` instancí. Dále se zobrazí informace o logických svazcích uvnitř clusteru, kdy používáme čtyři základní svazky a jeden námi definovaný. Pak máme k dispozici informaci o počtu uložených objektů v clusteru a místu, které tyto objekty zabírají. V našem případě objekty zabírají 5999 GB, přičemž replikační faktor 2 způsobuje, že objekty spolu s replikami zabírají více než 12 GB dat. Vidíme také informaci o volném místu v clusteru a stavy PG, které dále rozebírá tabulka 4.3. V závěru výpisu vidíme také informaci o právě probíhajícím zápisu nebo čtení.

Stav	Popis stavu
active	PG je připravena zpracovávat požadavky.
clean	Všechny objekty uložené v PG jsou replikovány v korektním počtu a dostupné. Stav `active + clean` popisuje bezchybný stav PG.
down	PG je nedostupná, protože není dostupný uzel s touto PG.
degraded	Cephu se nepodařilo replikovat některé objekty ukládané v PG v požadovaném počtu (replikační faktor).
inconsistent	Ceph detekuje inkonzistence v jedné nebo více replikách.
peering	Proces peering probíhá v rámci této PG.
recovering	Ceph migruje nebo synchronizuje objekty a jejich repliky.
incomplete	Ceph detekuje, že v PG chybí informace o zápisech, které mohly proběhnout, nebo nemá nějakou zdravou repliku.
stale	PG je v neznámém stavu. Ceph monitory nezískaly update od doby, kdy došlo ke změně mapování PG.

Tabulka 4.3. Tabulka stavů PG. Převzato z [21].

Kromě CLI nástrojů se běžně používají kontroly zahrnuté v dedikovaných softwarech určených pro monitoring jako jsou Zabbix, Nagios, Prometheus nebo jiné. My použí-

váme Zabbix se zapnutým JMX, takže nám Zabbix loguje metriky JVM, tj. samotného CPVA, jako např. garbage collector (GC), haldu (angl. *heap*), vlákna a jiné. Navíc sledujeme stav hardware a software v objektovém úložišti Cephu. Obzvlášť důležitý je monitoring volného místa na discích v clusteru, protože celková kondice a stav clusteru závisí na mnoha databázových (LevelDB) a metadatových operacích, které pro správný běh vyžadují dostatek volného místa na disku.

Instance `ceph-mon` jsou nejdůležitějšími linuxovými službami (démony) pro běh Ceph clusteru. Pokud dojde k nějakému selhání v clusteru, je vhodné začít hledat problém právě od `ceph-mon`. K selhání `ceph-mon`, resp. nemožnosti připojení se klienta k `ceph-mon`, může dojít z různých důvodů, například:

- Máme problémy s konektivitou. Instance `ceph-mon` je schována za firewallem, který nepovoluje port 6789.
- Uzel, na kterém je nasazen `ceph-mon`, má nedostatek volného místa na disku. Databáze LevelDB potřebuje alespoň 5 % volného místa na disku, jinak dojde k vypnutí `ceph-mon`.
- `Ceph-mon` nemusí běžet např. z důvodu selhání.
- Většina instancí `ceph-mon` je ve shodě ohledně stavu clusteru, nicméně některé ve shodě s většinou být nemusí. Pak říkáme, že `ceph-mon` je mimo kvórum.

`Ceph-osd` démoni mohou selhat z různých příčin:

- Selhání úložného zařízení (hard disk), které lze detekovat pomocí systémových zpráv.
- Problémy s konektivitou v síti, které lze odhalit pomocí nástrojů jako ping.
- Nedostatek volného místa na disku. Pokud je disk zaplněn alespoň z 85 %, dojde k varování `HEALTH_WARN`. Při zaplnění některého disku z 95 % dojde ke změně stavu na `HEALTH_ERR` a již není možné zapisovat data.
- Může dojít k nedostatku systémových prostředků. Je nutné mít dostatek systémové paměti, aby bylo možné udržet naživu všechny `ceph-osd` procesy.
- Základní nastavení časových prodlev (angl. *timeout*) komunikace může být nedostatečné a nestíhají se vykonat náročné I/O operace. Pokud jsou nevhodně nastavené HEARTBEAT, může to způsobit až pád instance `ceph-osd`.

Když už je identifikován problém s `ceph-osd` použitím např. nástroje `ceph osd tree`, který nám zobrazí počet `ceph-osd` ve stavu *down* a jednotlivé postižené instance, bývá nutné nahradit selhané `ceph-osd`.

4.5.8 Restart clusteru

Opětovné spuštění clusteru probíhá v několika etapách:

1. Nejprve je nutné obnovit konektivitu v síti. Čeká se na zavedení (angl. *boot*) switchů.
2. Z uzlů Ceph clusteru se nejprve zprovozní uzly, kde jsou nasazeny `ceph-mon`, a čeká se na ustanovení kvóra mezi `ceph-mon`. Tento proces může nějaký čas trvat.
3. Následně je možné spustit `ceph-osd` démony. Peering na jednotlivých `ceph-osd` může trvat delší dobu zvlášť v případě, kdy jsou instance `ceph-osd` umístěny na pomalých discích. V případě, kdy `ceph-osd` selže během stavu peering, je nutné selhané `ceph-osd` znovu spustit.

4.6 Filtr archivovaných dat

Abychom mohli rozlišit data určená pro archivaci do databáze Apache Cassandra od dat archivovaných v objektovém úložišti, vyhodnocujeme jejich atributy. Na základě

vyhodnocení se rozhodujeme, kam data zapíšeme. Pro naše účely jsou vhodné zejména tyto atributy:

1. Pojmenování zdroje dat. V terminologii Channel Access protokolu je zdrojem dat *channel*.
2. Datový typ příchozích dat. Můžeme pracovat pouze s typy, které podporuje Channel Access protokol.
3. Velikost příchozích dat v bytech. Můžeme určit hranici mezi daty ukládanými ještě do databáze Apache Cassandra a daty už ukládanými do objektového úložiště, přičemž tato hodnota se použije ve smyslu minimální velikosti dat ukládaných do objektového úložiště.
4. Hlavička příchozích dat. Využíváme detekce různých datových formátů, čímž můžeme ukládat do objektového úložiště např. všechny soubory typu PNG, BMP, TIFF, PDF a podobně.

Naše implementace umožňuje kombinovat tyto filtry, přičemž mezi všemi kategoriemi platí logický vztah *and*. Můžeme do objektového úložiště archivovat např. data, která jsou datového typu `DBR_TIME_CHAR` a zároveň velikosti alespoň 100 000 bytů a podobně. Navíc naše implementace umožňuje snadné přidání vlastních filtrů na základě dalších atributů.

Jednotlivé filtry zapisujeme do konfiguračního souboru CPVA, kterým je soubor `cassandra-pv-archiver.yaml`. Pro popis filtru používáme čtveřici atributů jméno, datovýTyp, velikostDat a hlavička, které oddělujeme separátorem čtyřtečkou. Na základě typu atributu může být hodnotou řetězec nebo celé číslo nebo znak. V obou případech navíc podporujeme *wildcard* v podobě znaku hvězdičku, který zastupuje libovolný počet libovolných znaků v případě řetězce nebo libovolnou hodnotu v případě celého čísla. Použití wildcard při práci s řetězci může vypadat např. takto:

```
channelName*  jméno zdroje začíná na "channelName" a pak následuje
              libovolně
*channelName* jméno zdroje obsahuje "channelName"
*channelName  jméno zdroje končí na "channelName"
channelName   jméno zdroje je právě "channelName"
*             libovolné jméno zdroje, platí pro všechny řetězce
```

Příklady filtru vypadají takto:

```
alan::DBR_TIME_SHORT::100000::*
jan*::DBR_TIME_CHAR::*::*
*lan*::DBR_TIME_CHAR::*::*
preffix*suffix::*::1000::*
*:ArrayInt::DBR_TIME_INT::500000::*
*::*::*::*
```

Máme šest různých příkladů filtrů, které různými způsoby používají wildcard. Zajímavý je poslední filtr, který označujeme jako anyFilter. Tento filtr je aplikovatelný na veškerá příchozí data ze všech datových zdrojů. V důsledku aplikace anyFilteru způsobí přesměrování veškerých dat do objektového úložiště Ceph.

Naše implementace zavádí omezení nad filtry. Pro každý zdroj dat můžeme použít maximálně jeden filtr. Na každý channel se použije první filtr ze seznamu, který lze použít na jméno zdroje dat. Vycházíme z předpokladu, kdy každý zdroj dat posílá časové řady dat, které pouze mění své hodnoty, ale nikoliv své atributy. Prioritu pravidel tedy určuje jejich pořadí v seznamu. Podobně je např. implementován výběr JDK v OS. V případě, kdy posílaná data mění své atributy, je nutný restart zdroje dat, tudíž na

straně CPVA zareagujeme také změnou filtru následovanou restartem instance CPVA, na které úprava filtru probíhá. Restarty instancí jsou na straně CPVA nutné, neboť úprava filtru probíhá na úrovni konfiguračního souboru. Rozšíření konfiguračního souboru CPVA o atributy popisující archivaci do objektového úložiště, tj. atribut pool, a filtrování vstupní dat, tj. atribut filters, vypadá takto:

```
# YAML values for archiving to Ceph Object Storage.
dfs:
  # Name of Ceph pool where data should be stored.
  pool: "data"
  # Array of strings that describe data filters.
  filters: "*lan::*::100::* jan*::DBR_TIME_CHAR::*::*"
```

Hodnotou atributu filters je pole řetězců oddělených mezerou, kdy každý řetězec zastupuje jeden vzor pro filtr. Rozhodli jsme se pro tento postup, protože ošetření duplicitních filtrů nelze vytvořit bez předchozí znalosti jmen zdrojů dat. Nelze zaručit, že obecné dva „regulární výrazy“ se nikdy neaplikují na stejný řetězec, i když umožníme jen některé případy jako prefix* a *suffix. Příkladem jsou třeba filtry `ab*` a `*ba`, které lze aplikovat na řetězec `aba`.

Řešením může být porovnání všech filtrů vůči všem jménům zdrojů během startu CPVA a vyhodit výjimku v případě, kdy na některý channel jdou aplikovat dva nebo více filtry. V takovém případě se porovnání provede během startu CPVA. Zároveň je nutné kontrolu provádět pokaždé, kdy se přidává nový channel přes grafické uživatelské rozhraní CPVA, protože nesmíme přidat jméno zdroje, které lze aplikovat na více filtrů. Tento přístup je z uživatelského pohledu značně komplikující.

Alternativním řešením je použití více různých filtrů pro jeden zdroj dat, kdy např. použijeme následující filtry pro channel pepicek, tj. `pepicek::*::200000::PNG`, `pepicek::*::100000::*`. Takový přístup považujeme za složitý z pohledu uživatele a následné správy, kdy bychom teoreticky mohli mít desítky filtrů pro každý channel, které se vzájemně prolínají. Především se jedná o filtry s použitím wildcard, kdy filtry `*::*::100000::*` a `pepicek::*::100000::*` dělají totéž, čímž se stává druhý filtr duplicitní.

4.7 Archivovaná data

Pro účely přenosu obrazových dat můžeme využít pouze datových typů podporovaných protokolem Channel Access, přičemž datový typ není pro nás jediným omezením. Vlastnosti protokolu nedovolují za běhu dynamicky reagovat na změnu velikosti posílaných dat, což pro nás v důsledku znamená, že zdroje dat sice mohou posílat data o libovolné velikosti, ale vždy je přenášen stejný počet bytů. Musíme tedy odpovědět na otázku, zda můžeme nějakým způsobem určit velikost původních dat, aby nevznikal overhead na straně úložiště a aby se neukládalo zbytečně velké množství dat.

4.7.1 Datový typ přenášených dat

Našim hlavním požadavkem je přenášení obrazových dat ve formátu PNG, který je bytovým formátem. Ideálním stavem je použití datového typu, který je podporován Channel Access protokolem a který má stejnou velikost jako byte nebo se svojí velikostí co nejvíc přibližuje velikosti typu byte.

Channel Access protokol podporuje pouze datové typy větší velikosti než byte. Mezi podporované typy patří `DBR_TIME_CHAR`, `DBR_TIME_DOUBLE`, `DBR_TIME_ENUM`, `DBR_TIME_FLOAT`, `DBR_TIME_LONG`, `DBR_TIME_SHORT`, `DBR_TIME_STRING`, přičemž

nejmenším typem je DBR_TIME_CHAR o velikosti 2 bytů. Musíme tak vhodně škálovat obrazová data a využít všech bytů datového typu DBR_TIME_CHAR tak, aby nedocházelo k natažení původních dat:

```
0000000   \0  211 \0  P   \0  N   \0  G   \0  \r  \0  \n  \0  032 \0  \n
          000 211 000 120 000 116 000 107 000 015 000 012 000 032 000 012
0000020   \0  \0  \0  \0  \0  \0  \0  \r  \0  I~\0  H   \0  D   \0  R
          000 000 000 000 000 000 000 015 000 111 000 110 000 104 000 122
0000040   \0  \0  \0  \0  \0  004 \0  \0  \0  \0  \0  \0  \0  004 \0  \0
          000 000 000 000 000 004 000 000 000 000 000 000 000 004 000 000
```

Hodnota každého lichého bytu je 0x00, čímž dochází k poškození přenášených dat, na příkladě konkrétně hlavičky formátu PNG. V našem případě je pro přenos dat nejjednodušší použití datového typu DBR_TIME_CHAR, kdy využíváme celé šířky tohoto datového typu. Můžeme použít libovolný datový typ, ale v takovém případě musíme na straně zdroje dat vhodně pracovat s tímto datovým typem a využít jeho celé šířky pro přenos dat. Pokud použijeme např. datový typ DBR_TIME_SHORT, musíme jen využít v celé jeho šířce, tzn. `short = (char << 8) | (char)`.

Channel Access protokol implementuje pro každý datový typ jinu sadu metadat, tudíž v některých případech je použití datového typu jiného než DBR_TIME_CHAR vhodné i pro přenos obrazové informace. Můžeme využít všechny datové typy podporované Channel Access protokolem. Nepoužíváme pouze typ DBR_TIME_STRING, u kterého při rekonstrukci obrazové informace vzniká značný overhead na straně CPVA.

4.7.2 Pevná velikost přenášených dat

Použití jiných datových typů pro přenos obrazových dat než byte také znamená zarovnání velikosti obrázku tak, aby byla násobkem velikosti datového typu, který použijeme pro přenos. V případě, kdy např. použijeme datový typ o velikosti 4 B a velikost obrazové informace jsou 3 B, vzniká natažení původních dat o 1 B z důvodu zarovnání na celou šířku datového typu.

Natažení obrazové informace není jediným zdrojem pro tzv. *trailing zeros* na konci dat. Každý zdroj dat pevně určuje velikost dat posílaných do CPVA, kdy např. skutečný datový soubor má velikost 240 B, ale z důvodů výkonnosti zdroje dat jsou rozměry posílaných bloků určeny na 256 B (nativeCount). Můžeme posílat datové soubory velikostí menší nebo rovny hodnotě nativeCount. V případě posílání souborů větších je nutné změnit tuto hodnotu nebo rozřezat soubory do více bloků, tzv. *chunk*, jinak hrozí poškození dat.

Pokud zdroj dat zaplní bytový buffer datovým souborem do indexu 239, přičemž zbývající indexy 240 až 255 bufferu zůstanou na hodnotě 0x00. Avšak tato velikost nesmí přesáhnout velikost nastavenou na straně CPVA, tj. `maxPayloadReceiveSize` a `maxPayloadSendSize` konfigurovatelné v souboru `cassandra-pv-archiver.yaml`. Spolu s datovým souborem se neposílá skutečná velikost původního datového souboru, tudíž v tuto chvíli nemůžeme bez podrobné znalosti formátu přenášeného datového souboru odstranit přebytečné byty na konci přijatých dat. Mohlo by dojít k poškození datového souboru v případě, že tyto byty mají pro datový soubor nějaký význam.

Vzorová data, která posíláme za účely archivace a která mohou obsahovat platné znaky 0x00, mohou vypadat např. takto:

```
data=np.array([0x00],np.uint8)
data=np.array([0x06,0xde,0xad,0xbe,0xef,0xca,0xfe,0x00],np.uint8)
data=np.array([0x07,0xde,0xad,0xbe,0xef,0xca,0xfe,0x00,0x00],np.uint8)
data=np.array([0x09,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
                0x00,0x00,0x00,0x00,0x00,0x00],np.uint8)
data=np.array([0x0a,0xff,0x00,0xff,0x00,0xff,0x00,0xff,0x00,
                0xff,0x00,0xff,0x00,0xff,0x00,0xff],np.uint8)
```

Stejná data archivovaná v databázi Apache Cassandra, kde jsou data zarovnána na velikost 32 s použitím *trailing zeros*, pak vypadají následovně:

```
{value: 0x09000000000000000000000000000000,
  alarm_severity: 0, alarm_status: 0, units: '', lower_warning_limit: 0,
  upper_warning_limit: 0, lower_alarm_limit: 0, upper_alarm_limit: 0,
  lower_display_limit: 0, upper_display_limit: 0, lower_control_limit: 0,
  upper_control_limit: 0}
{value: 0x0aff00ff00ff00ff00ff00ff00ff00ff,
  alarm_severity: 0, alarm_status: 0, units: '', lower_warning_limit: 0,
  upper_warning_limit: 0, lower_alarm_limit: 0, upper_alarm_limit: 0,
  lower_display_limit: 0, upper_display_limit: 0, lower_control_limit: 0,
  upper_control_limit: 0}
{value: 0x07deadbeefcafe00000000000000000000,
  alarm_severity: 0, alarm_status: 0, units: '', lower_warning_limit: 0,
  upper_warning_limit: 0, lower_alarm_limit: 0, upper_alarm_limit: 0,
  lower_display_limit: 0, upper_display_limit: 0, lower_control_limit: 0,
  upper_control_limit: 0}
{value: 0x06000000000000000000000000000000,
  alarm_severity: 0, alarm_status: 0, units: '', lower_warning_limit: 0,
  upper_warning_limit: 0, lower_alarm_limit: 0, upper_alarm_limit: 0,
  lower_display_limit: 0, upper_display_limit: 0, lower_control_limit: 0,
  upper_control_limit: 0}
{value: 0x00000000000000000000000000000000,
  alarm_severity: 0, alarm_status: 0, units: '', lower_warning_limit: 0,
  upper_warning_limit: 0, lower_alarm_limit: 0, upper_alarm_limit: 0,
  lower_display_limit: 0, upper_display_limit: 0, lower_control_limit: 0,
  upper_control_limit: 0}
```

Nejsme schopni určit původní velikost ani vyhodnotit platnost 0x00. Kontrolní součet původních dat odpovídá kontrolnímu součtu dat, která jsou doplněna o *trailing zeros*. Stejně tak nedochází k poškození takových dat v rámci našeho případu užití.

Vzhledem k Channel Access protokolu a snaze používat stejné a neupravované verze protokolu napříč celým systémem jsme limitováni z pohledu možností, které lze v tomto případě použít. V našem případě zdroje dat posílají obrazová data ve formátu PNG s fixními rozměry. Nejsme nuceni dynamicky reagovat na změnu velikosti příchozích dat a ponecháváme tento stav jako vlastnost implementovaného systému. Volíme možnost posílání dat v jednom bloku a posíláme bloky dat o velikosti 1,5 MiB. Na straně CPVA neřešíme ani pořadí přijatých bloků, ani jejich skládání do původního souboru.

4.7.3 Formát PNG

Jednou z možností řešení problému s *trailing zeros* je využití znalosti datového formátu. V našem případě používáme pro přenos obrazové informace formát PNG, tudíž bychom v rámci této etapy implementace systému archivace podporovali pouze tento datový formát.

V případě formátu PNG můžeme využít struktury datového formátu. Určíme velikost dat na základě součtu velikostí jednotlivých chunků PNG souboru. Tento způsob řešení ovšem naráží na vlastnost PNG formátu, kdy i nulová velikost chunku, který obsahuje data, je platnou velikostí. Museli bychom iterovat přes tyto chunky až do místa, kdy

narazíme na CRC, tedy konec chunku. V nejhorším případě žádný u žádného z chunků není uvedena nenulová velikost, v důsledku čehož musíme iterovat přes všechna platné byty obrázku, což je časově náročné zvlášť v případě, kdy chceme archivovat větší počet paralelně přijímaných obrazových, skalárních nebo vektorových dat. Vzniká zde overhead.

Pokud navíc požadujeme obecné řešení nejen pro formát PNG, musíme pro každý používaný formát vždy dodat implementaci, která počítá velikost souboru konkrétního formátu. Vše směřuje na mechanismus, kde je potřeba posílat velikost dat spolu s daty, např. v podobě hlavičky dat určených k archivaci. Můžeme se inspirovat hlavičkou formátu PNG a vyhradit prvních dvanáct bytů posílaných dat pro naši vlastní hlavičku. Prvních šest bytů použijeme pro identifikaci této naší hlavičky, aby jí bylo možné rozeznat od hlaviček jiných datových formátů. Tento princip nám umožňuje filtrovat data, kterým je a kterým není přidána hlavička s jejich délkou. Následující čtyři byty vyhradíme pro velikost dat, přičemž teoretická maximální velikost dat by dosahovala hodnoty $(2^{31}) - 1$ bytů. Poslední dva byty použijeme pro CRLF nebo jiný oddělovač. Řešení je efektivní na straně příjemce dat, tj. CPVA, ale je náročné na implementaci na straně producenta dat, kterým může být libovolné zařízení s embedded software a minimálními hardwarovými prostředky.

Ani jedno z uvedených řešení pro nás aktuálně není přijatelné, neboť vždy je zaveden overhead buď na straně příjemce dat, nebo na straně zdroje dat. Ponecháváme v platnosti řešení, kdy využíváme statické velikosti posílaných obrazových dat, tudíž jsme dopředu schopni určit ideální velikost tak, abychom minimalizovali počet *trailing zeros* a nenafukovali počet přenášených dat.

4.8 Identifikátor obrazových dat

Pro účely uložení obrazových dat do objektového úložiště Ceph musíme ke každému archivovanému snímku přiřadit dostatečně unikátní klíč, tj. identifikátor objektu. Dojde tím k naplnění podmínky, kdy v jednom logickém svazku clusteru nemohou být dva objekty se stejným klíčem. Pokud se pokusíme uložit více objektů se stejným klíčem, dochází k přepsání dat používajících stejný klíč a v důsledku toho ztrácíme archivovaná data.

4.8.1 Tvorba klíče

Knihovna librados poskytuje metody `write(key, object)` pro zápis dat do objektového úložiště. Tato metoda je přetížená a umožňuje zapisovat různé typy objektů, ale na místě klíče je vždy požadován řetězec typu *String*. Našim cílem je vytvoření dostatečně unikátního klíče, aby během příjmu obrazových dat z paralelně běžících zdrojů nedocházelo k přepisu archivovaných dat. V našem případě máme pro každá archivovaná data k dispozici údaje o časové známce jejich vzniku, ale pouze tento údaj je nedostatečný pro tvorbu unikátního klíče. V jeden čas s přesností na nanosekundy (ns) může vzniknout více snímků na různých zdrojích dat. Musíme časový údaj vzniku obrazových dat identifikovat spolu s dalším údajem, který unikátně identifikuje zdroj dat. Pro tvorbu klíče používáme dvojici `jménoZdroje, časováZnámka`. Předpokládáme, že na jednom zdroji dat v jeden čas s přesností na nanosekundy nemůže vzniknout více než jeden snímek. Když navíc zvážíme velikost obrazových dat, vznik více snímků v jeden čas není z fyzického hlediska reálný. Pokud i přesto dojde k vytvoření dvou snímků ve stejný čas, předpokládáme tento stav za chybu systému a odeslání duplicitních snímků, které budou v objektovém úložišti archivovány pouze jednou.

Chceme identifikátory objektů čitelné pro uživatele, tudíž používáme jméno zdroje dat namísto UUID zdroje dat. Abychom rozlišili část klíče, která je ještě jménem zdroje dat a která už je časovou známkou, používáme oddělovač čtyřtečku. Použití oddělovače čtyřtečky umožňuje použití oddělovače dvojtečky pro účely jmenné konvence zdrojů dat. Výsledný klíč tak má podobu `jménoZdroje::časováZnámka`.

Veškeré klíče jsou pak archivovány v databázi Apache Cassandra spolu s metadaty původních obrazových dat. Nedochází tak ke ztrátě metadat nebo jiných informací, které přichází spolu s obrazovými daty. Tyto klíče lze pak využít pro listování objekty uloženými v objektovém úložišti. Není nutné volat nad objektovým úložištěm příkazy, které zobrazí seznam všech klíčů. Zabraňujeme tak volání příkazu, který může být časově náročný, uvážíme-li, že v jednom logickém svazku můžeme archivovat miliony objektů. V tomto kontextu můžeme použít CLI nástroje pro přístup k uloženým datům:

```
rados -p data ls
rados get channelName::timestamp outputFile.txt --pool=data
rados put channelName::timestamp inputFile.txt --pool=data
```

První příkaz zobrazí všechny klíče k objektům uloženým v poolu se jménem `data`, druhý příkaz načte objekt uložený pod klíčem `channelName::timestamp` v poolu data a uloží objekt do souboru `outputFile.txt` a třetí příkaz uloží obsah souboru `inputFile.txt` do poolu `data` pod klíčem `channelName::timestamp`. Dojde tak k přepisu původně uloženého objektu.

4.8.2 Uložení klíče do Apache Cassandry

Přenos obrazových dat za použití typu String není vhodným přístupem vzhledem k omezením tohoto datového typu. Musíme tak řešit podobu, v jaké ukládáme klíč objektů uložených v Cephu do databáze Apache Cassandra. Vybíráme ze dvou možností řešení problému: buď nahradíme datový typ použitý pro přenos dat typem String a uložíme klíč ve formátu String do databáze Apache Cassandra, nebo použijeme mapování hodnoty klíče na datový typ použitý pro přenos obrazových dat.

V případě první možnosti nahrazujeme původní datový typ za jiný datový typ, tj. String, čímž přicházíme o metadata posílaná spolu s obrazovými daty, neboť různé datové typy obsahují jinou sadu metadat. Klíče uložené v databázi Apache Cassandra jsou stále uživatelsky čitelné bez nutnosti konverze, protože jsou uloženy jako řetězce původních znaků. Na druhou stranu ale přicházíme o metadata spojená s obrazovou informací, protože každý datový typ podporovaný Channel Access protokolem obsahuje jinou sadu metadat.

V druhém klíče mapujeme na datový typ, který používáme pro přenos obrazových dat. Pracujeme se stejným principem jako v případě mapování bytových dat obrazové informace na datový typ podporovaný protokolem Channel Access. Neztrácíme metadata, ale klíč není uložen do databáze Apache Cassandra jako řetězec znaků. Klíč je konvertován do jiného datového typu, čímž přestává být pro uživatele čitelný. Pro jeho použití je nutné nejprve přeložit klíč zpět do typu String. Opět platí vlastnost této konverze, kdy délka klíče po konverzi z pole znaků (String) nemusí odpovídat délce klíče po konverzi krát velikost datového typu a kdy je klíč doplněn o *trailing zeros*. Musíme přidat požadavek na klíče, které nesmí obsahovat platné znaky 0x00 jako součást klíče.

Z uvedených možností implementujeme variantu, u které neztrácíme metadata a řetězce klíčů před uložením do databáze Apache Cassandra překládáme do datového typu použitého pro přenos obrazových dat. Příklady uložených klíčů spolu s metadaty pro datový typ `DBR_TIME_CHAR` mohou vypadat následovně:

```
{value: 0x43532d544553542d524d432d312d494f433a43414d323a3135313035373735
  3339393336353634353237, alarm_severity: 0, alarm_status: 0, units: '',
  lower_warning_limit: 0, upper_warning_limit: 0, lower_alarm_limit: 0,
  upper_alarm_limit: 0, lower_display_limit: 0, upper_display_limit: 0,
  lower_control_limit: 0, upper_control_limit: 0}
{value: 0x43532d544553542d524d432d312d494f433a43414d323a3135313035373831
  3238393231363539303938, alarm_severity: 0, alarm_status: 0, units: '',
  lower_warning_limit: 0, upper_warning_limit: 0, lower_alarm_limit: 0,
  upper_alarm_limit: 0, lower_display_limit: 0, upper_display_limit: 0,
  lower_control_limit: 0, upper_control_limit: 0}
```

4.9 Sledování stavu Ceph clusteru

Testování prototypu upraveného CPVA odhaluje problém během ukládání do objektového úložiště s nedostatkem volného místa na `ceph-osd` démonech. Před dosažením 85 % úložné kapacity `ceph-osd` démona veškerá archivace obrazových dat běží bez problémů. Při překročení 85 % libovolného `ceph-osd` démona se objeví chybový stav `HEALTH_WARN, OSD_BACKFILLFULL: 1 backfillfull osd(s)`, nicméně archivace dat je stále možná a nehrozí náhlé poškození dat. K vážnému problému dochází s překročením hranice 95 % úložné kapacity libovolného `ceph-osd` démona. Zobrazí se nový stav `HEALTH_ERR`, který vypadá např. takto:

```
HEALTH_ERR full flag(s) set; 1 full osd(s)
OSDMAP_FLAGS full flag(s) set
OSD_FULL 1 full osd(s)
osd.9 is full
```

V tomto stavu nejenže dochází k zastavení archivace dat, ale je také způsobeno „zamrznutí“ vláken upraveného CPVA, čímž dochází k zastavení archivace veškerých dat včetně skalárních nebo vektorových dat archivovaných přes postižený uzel CPVA.

Po bližším prozkoumání komunikace mezi upraveným CPVA nasazeným na ceph-klient uzlu a `ceph-osd` démonem zjišťujeme, že ceph-klient nezískává ze strany úložiště ack, které obvykle následuje po úspěšném zápisu všech replik archivovaných obrazových dat. Příčinou nezískání ack je nemožnost uložení některé z replik v důsledku zaplnění některého z `ceph-osd` démona z 95 % jeho úložné kapacity. Knihovna rados-java v tuto chvíli nedetekuje situaci a nevyhodí výjimku.

Problém se ukazuje ještě závažnější, když necháváme Ceph clusteru čas pro běh obnovovacích procesů. Po dvou dnech dochází k úplnému rozbití Ceph clusteru, který tak není schopen úspěšně provádět obnovovací a rebalancovací procesy s minimem volného místa na postižených `ceph-osd` démonech.

Následkem problému tedy není pouze zastavení archivace obrazových dat, ale hrozí eventuální ztráta veškerých dat uložených v Ceph clusteru.

4.9.1 Preventivní řešení

Tento problém můžeme řešit zavedením monitorování stavu Ceph clusteru na straně CPVA a aktivnímu zabránění dosažení stavu `HEALTH_ERR` nebo použitím asynchronních operací pro zápis obrazových dat, kdy CPVA nečeká na ACK.

V případě použití asynchronních operací zápisu dat do objektového úložiště Cephu, které jsou umožněny v knihovně rados-java od její verze 4.0.0, tento problém přetrvává, protože dochází k dosažení stavu `HEALTH_ERR` na straně úložiště, a navíc přicházíme o možnost počítání nebo logování zahozených obrazových dat.

V případě řešení zavedením periodického monitorování stavu Ceph clusteru na straně CPVA aktivně zastavujeme archivaci dat mířících do objektového úložiště Cephu během vybraných stavů `HEALTH_WARN`, abychom zabránili dosažení stavu `HEALTH_ERR`.

Stav `HEALTH_WARN` může být vyvolán z mnoha různých příčin, proto nemonitorujeme pouze stav, ale také příčinu, která vede k `HEALTH_WARN`. Pokud je stav `HEALTH_WARN` vyvolán příčinou `OSD_BACKFILLFULL`, okamžitě pozastavujeme archivování obrazových dat do objektového úložiště Ceph a obrazové informace zahazujeme, aby nedošlo k eskalaci stavu až k `HEALTH_ERR`. Vzhledem k objemu a frekvenci obrazových dat není vhodné, abychom tato data logovali nebo ukládali na záložní úložiště. Pouze v grafickém rozhraní navýšíme hodnotu zahozených vzorků, tj. obrazových dat, podobně jako v případě zahození skalárních nebo vektorových dat, a navíc zaznamenáme chybu do logu na straně CPVA.

Pro zjištění stavu Ceph clusteru používáme volání `ceph -s` s pomocí knihovny librados. Dochází tak ke komunikaci mezi CPVA a `ceph-mon`, které mají informaci o stavu clusteru. Máme možnost upravit četnost dotazování se na stav Ceph clusteru. Můžeme aktualizovat informaci o stavu Ceph clusteru před každým pokusem o zápis obrazových dat nebo můžeme informaci aktualizovat po předem určených časových intervalech.

V případě, kdy zjišťujeme stav clusteru před každým uložením obrazové informace, může vznikat overhead na straně `ceph-mon`, které musí reagovat na velké množství požadavků v jeden čas. Pokud navíc předpokládáme, že se náš cluster skládá z minimálního možného počtu `ceph-mon` tak, aby ještě monitory slabým prvkem, může dojít k situaci, že tři instance `ceph-mon` budou muset obsluhovat požadavky volání na stav clusteru s každým pokusem o zápis obrazových nebo jiných dat.

Pokud nezjišťujeme stav clusteru před každým pokusem o zapsání dat, ale pouze jednou za předem zvolený časový interval, významně snižujeme komunikaci mezi ceph-klient uzly s nasazenými modifikovanými CPVA. V tomto případě musíme časový interval nastavit tak, aby během tohoto intervalu nemohlo dojít ke změnám stavu `HEALTH_OK` na `HEALTH_WARN` a následně na `HEALTH_ERR`.

Námi implementovaná varianta periodicky sleduje stav Ceph clusteru, kdy zjišťujeme stav clusteru v minutových intervalech.

4.9.2 Sledované stavy

Během aktivního sledování stavů můžeme zjistit jeden ze tří stavů: `HEALTH_OK`, `HEALTH_WARN` nebo `HEALTH_ERR`. Při zjištění těchto stavů následuje:

- Pokud zjistíme stav `HEALTH_OK`, pokračujeme v archivaci obrazových dat. Tento stav je zcela v pořádku a neindikuje žádný problém.
- Pokud zjistíme stav `HEALTH_WARN`, dále zjišťujeme příčinu, která k tomuto stavu vede. Stav `HEALTH_WARN` může vyvolat např. `CLOCK_SKEW` mezi `ceph-mon` nebo např. `REQUEST_SLOW` vyvolaný pomalým zápisem dat na úložná zařízení `ceph-osd` démonů. V uvedených případech nechceme zastavit archivaci obrazových dat. Pokud zjistíme `HEALTH_WARN` např. z důvodu překročení 85 % úložné kapacity některého z `ceph-osd`, archivaci zastavujeme a zahazujeme obrazová data, aby nedošlo k eskalaci problému až do stavu `HEALTH_ERR`.
- Pokud zjistíme stav `HEALTH_ERR`, okamžitě zastavujeme archivaci dat a zahazujeme obrazová data.

Volání `ceph -s` nám umožňuje zjistit nejen stav Ceph clusteru, ale také příčinu, která tento stav vyvolává. O zastavení archivace se rozhodujeme na základě obou těchto údajů. Zjištěné stavy Ceph clusteru jiné než `HEALTH_OK` zároveň zapisujeme do logu CPVA.

4.10 Čítače

CPVA v původní variantě umožňuje počítání archivovaných i zahozených dat. Abychom řádně využívali tyto čítače, musíme je inkrementovat i na úrovni naší implementace.

4.10.1 Zahozená data

CPVA počítá data, která se z některé příčiny nepodařilo archivovat. Tento čítač nazývá *dropped samples*. Naše implementace inkrementuje tento čítač, pokud se nepodaří archivovat obrazová data v objektovém úložišti Ceph nebo pokud je Ceph cluster ve stavu, který může ohrozit stabilitu clusteru, a my se preventivně rozhodneme zastavit archivaci dat.

Čítač zahozených dat implementujeme jako *counter per thread*, kdy pro každý channel máme vlastní čítač. V případě, kdy chceme znát celkové hodnoty, tj. součet hodnot všech čítačů, známe pouze orientační hodnotu s přesností na jednotky vzorků.

4.10.2 Archivovaná data

Naše implementace umožňuje počítat archivované vzorky stejným způsobem jako původní CPVA. Samotné zápisy do Ceph clusteru nepočítáme. Počítáme až následné zápisy klíčů do Apache Cassandry, což za nás dělá čítač, který je součástí CPVA. Pokud používáme naši knihovnu samostatně bez CPVA, máme implementovaný counter per thread, kdy každý channel má vlastní čítač úspěšně zapsaných vzorků v objektovém úložišti Ceph. Tento čítač pracuje podobně jako čítač zahozených dat.

4.10.3 Čítače pro sledování výkonu

V rámci archivace do Ceph clusteru nás nezajímá pouze počet archivovaných nebo zahozených dat, ale chceme znát také čas, po který archivace vzorku trvala. Sledujeme tedy také dobu od začátku zápisu až po úspěšné ukončení zápisu do Ceph clusteru. Tyto čítače jsou v produkční verzi vypnuty, nicméně je vhodné použít je pro testování výkonnosti v testovacím prostředí spolu s logováním na úrovni `DEBUG`.

Kapitola 5
Evaluace

V této kapitole představíme metody použité pro testování výkonu našeho testovacího clusteru, popíšeme konfiguraci testovacího clusteru jak z pohledu hardware, tak i z pohledu software, ukážeme škálovatelnost našeho systému a vyhodnotíme výkon systému archivace obrazových dat.

5.1 Nástroje a data

Pro vyhodnocení výkonu Ceph clusteru používáme benchmarkový nástroj rados bench[1]. Jedná se o nástroj spustitelný v příkazové řádce (CLI), který lze konfigurovat pomocí parametrů jako např. počet vláken provádějících operace čtení nebo zápisu, velikost čteného nebo zapisovaného objektu v jednotkách bytů (B), délku měření v sekundách a jiné. Výstup měření pomocí tohoto nástroje vypadá např. následovně:

sec	ops	started	finished	avg MB/s	cur MB/s	last lat(s)	avg lat(s)
40	16	5508	5492	201.091	57.1289	0.0313413	0.113392
41	16	5530	5514	196.972	32.2266	0.0305342	0.114652
42	16	5541	5525	192.666	16.1133	1.06476	0.116871
43	16	5675	5659	192.75	196.289	0.0584037	0.121063
44	16	5816	5800	193.063	206.543	0.0439375	0.121142
45	16	5974	5958	193.915	231.445	0.0332082	0.120462
46	16	6133	6117	194.762	232.91	0.0779536	0.120221
47	16	6274	6258	195.012	206.543	0.0583192	0.119748
48	16	6405	6389	194.946	191.895	0.254009	0.11982
49	16	6544	6528	195.122	203.613	0.0542624	0.119884
50	16	6696	6680	195.672	222.656	0.0898487	0.119503
51	16	6845	6829	196.114	218.262	0.265921	0.119274
52	16	7000	6984	196.709	227.051	0.0634673	0.118924
53	16	7156	7140	197.308	228.516	0.0620207	0.11837
54	16	7311	7295	197.858	227.051	0.0757357	0.118201
55	16	7454	7438	198.069	209.473	0.191914	0.118073
56	16	7589	7573	198.062	197.754	0.0648419	0.117945
57	16	7727	7711	198.133	202.148	0.0524941	0.117945
58	16	7859	7843	198.051	193.359	0.0671682	0.118018
59	16	8003	7987	198.268	210.938	0.285248	0.118035

Vidíme zde čas, po který měření trvá, aktuální počet zapisujících vláken, počet započatých zápisů a úspěšně ukončených zápisů objektu do této chvíle, průměrnou propustnost dat za celou dobu, propustnost dat za poslední sekundu měření, průměrnou latenci zápisu za poslední vteřinu měření a průměrnou latenci zápisu za celé měření. Obdobně vypadá výstup, pokud sekvenčně čteme objekty uložené v úložišti.

Nástroj rados bench používáme nejen pro ověření škálovatelnosti clusteru, ale také pro vyhodnocení chování clusteru během dlouho trvajících měření, tj. takových měření,

[1] http://docs.ceph.com/docs/giant/man/8/rados/

během kterých se snažíme zaplnit Ceph cluster zapisovanými objekty do maximální kapacity, která ještě neohrožuje stabilitu clusteru.

V případě ověřování škálovatelnosti vyhodnocujeme cluster z pohledu operací sekvenčního čtení nebo zápisu objektů. Měříme a vyhodnocujeme parametry jako např. počet přečtených nebo zapsaných objektů za daný časový úsek (objekt/s), propustnost dat v jednotkách MB za sekundu (MB/s), počet operací na logickém disku za sekundu (IOPS) a latenci operací čtení nebo zápisu v jednotkách sekund. Pro měření používáme různý počet vláken, která souběžně vykonávají buď operace čtení, nebo operace zápisu, a stejně tak různé velikosti objektů. Počty vláken se pohybují v rozmezí od 1 do 32, zatímco pracujeme s objekty o velikosti od 1 do 16 MB. Délka měření, při kterém ověřujeme škálovatelnost, je 300 sekund. Z časových důvodů se nesnažíme o zaplnění kapacity celého úložiště při tomto druhu měření.

Pokud vykonáváme dlouhodobé měření zápisu dat, používáme pro tento účel 16 souběžně zapisujících vláken, která zapisují objekty o velikosti 1,5 MiB až po zaplnění některého z ceph-osd z 85 % jeho kapacity, čímž dosáhneme stavu `HEALTH_WARN`. Napodobujeme tak zátěž clusteru, během které 16 zdrojů dat posílá experimentální data s maximální možnou frekvencí, kterou ještě Ceph clusteru stíhá obsluhovat. Takový benchmark provádíme po 3600 sekundových úsecích, které navazují na sebe téměř bez přerušení. Kromě zmíněných atributů navíc sledujeme extrémy maximální latence, tzn. zajímají nás časové úseky, během nichž clusteru vykazuje problematické a potenciálně stabilitu ohrožující chování.

Pro vyhodnocení výkonu CPVA používáme vlastní čítače výkonnosti. Pro daný časový úsek sledujeme počty zapsaných objektů, průměrnou a maximální latenci zápisu. Data sbíráme každých 60 sekund, přičemž průměrnou latenci počítáme nejen pro právě uplynulých 60 sekund, ale také za celou dobu od začátku měření. Tato statistická data sbíráme nad všemi zdroji archivujícími data do Ceph clusteru a sledujeme také celkové statistiky za každou minutu archivace. Záznam v logu CPVA pak vypadá např. následovně:

```
     channel-name               writes maxlat lat(s) alltmlat(s)
  1. CS-TEST-RMC-1-IOC:CAM1        300  0.659  0.077       0.077
  2. CS-TEST-RMC-10-IOC:CAM1       299  0.638  0.068       0.067
  3. CS-TEST-RMC-11-IOC:CAM1       301  1.120  0.069       0.067
  4. CS-TEST-RMC-12-IOC:CAM1       299  1.010  0.080       0.082
  5. CS-TEST-RMC-13-IOC:CAM1       299  0.449  0.064       0.069
  6. CS-TEST-RMC-14-IOC:CAM1       298  0.713  0.080       0.078
  7. CS-TEST-RMC-15-IOC:CAM1       298  0.887  0.082       0.073
  8. CS-TEST-RMC-16-IOC:CAM1       299  0.611  0.069       0.069
  9. CS-TEST-RMC-2-IOC:CAM1        300  0.847  0.062       0.068
 10. CS-TEST-RMC-3-IOC:CAM1        300  0.854  0.075       0.070
 11. CS-TEST-RMC-4-IOC:CAM1        299  0.820  0.082       0.069
 12. CS-TEST-RMC-5-IOC:CAM1        300  0.659  0.082       0.080
 13. CS-TEST-RMC-6-IOC:CAM1        298  0.653  0.077       0.081
 14. CS-TEST-RMC-7-IOC:CAM1        300  0.940  0.059       0.059
 15. CS-TEST-RMC-8-IOC:CAM1        299  1.008  0.073       0.068
 16. CS-TEST-RMC-9-IOC:CAM1        299  0.681  0.071       0.069
 ------------------------------------------------------------------
 Time: 12 min                     4788  1.120  0.073       0.074
```

V případě CPVA nás zajímají všechny zaznamenávané metriky. Měření získaná pomocí nástroje rados bench nám poskytují horní limit výkonnosti Ceph clusteru. Máme tak možnost ověřit, zda je archivace pomocí CPVA dostatečně výkonná a efektivní.

Veškerá data z měření získaná nástrojem rados bench nebo zaznamenaná čítači CPVA jsou součástí elektronické přílohy této diplomové práce.

5.2 Experimentální sestava

Testovací sestava se skládá ze šesti uzlů. Ceph cluster je nasazen za pěti z nich, zatímco poslední uzel je vyhrazen pro `ceph-klient` spolu s instancí upraveného CPVA, který archivuje obrazová data do objektového úložiště.

Ceph cluster je nasazen na uzlech node1, node2, node3, node4 a node5. Uzly node1 a node2 jsou osazeny osmi jádrovým procesorem Intel Xeon CPU E3-1241 v3 @3.50 GHz s 32 kB L1 cache, 256 kB L2 cache a 8 MB L3 cache. Uzly node3, node4 a node5 jsou osazeny šesti jádrovým procesorem Intel Xeon CPU E5-2620 v3 @2.40GHz s 32 kB L1 cache, 256 kB L2 cache a 15 MB L3 cache. Všechny tyto uzly mají 32 GB DDR4 2133 MT/s RAM (PC4-2133). Každý uzel, ze kterého se skládá Ceph cluster, obsahuje ATA SanDisk SD7SB3Q2 SATA 3 SSD s kapacitou 256 GiB a mají navíc datové SATA 3 disky WD RED ATA WDC WD20EFRX-68E 5400RPM s kapacitou 2 TiB v počtu 1 ks v případě node1, node2 a v počtu 3 ks v případě node3, node4 a node5. Každý uzel navíc obsahuje dvě síťové karty 1 GbE, které se používají pro účely veřejné nebo privátní clusterové sítě.

Na uzlu node6 je nasazena nejen upravená varianta CPVA, ale také Apache Cassandra. Pro testovací účely nenasazujeme celý cluster, ale pouze jediný uzel této distribuované databáze. Node6 je osazen osmi jádrovým procesorem Intel Xeon CPU E5620 @2.40GHz s 32 kB L1 cache, 256 kB L2 cache a 12 MB L3 cache, 48 GB DDR4 2133 MT/s RAM (PC4-2133). Disková úložiště jsou na tomto uzlu stejná jako v případě node1 nebo node2, tzn. ATA SanDisk SD7SB3Q2 SATA 3 SSD s kapacitou 256 GiB a jediný datový disk WD RED ATA WDC WD20EFRX-68E 5400RPM s kapacitou 2 TiB.

Na každém uzlu je instalován jednotný operační systém CentOS 7. Na uzlech, ze kterých se skládá Ceph cluster, nasazujeme tři instance `ceph-mon`, tři instance ceph-mgr a celkem jedenáct `ceph-osd` démonů. V případě `ceph-klient` uzlu instalujeme navíc Java JDK 8.

Konfigurace Ceph clusteru, tj. obsah souboru `/etc/ceph/ceph.conf`, vypadá následovně:

```
[global]
fsid = 4af3d6bf-c850-4a65-8466-b35fa0c437e7
public_network = 192.168.6.0/24
cluster_network = 192.168.60.0/24

mon_initial_members = l1imagmon02,l1imagmon03,l1imagosd01
mon_host = 192.168.6.15,192.168.6.16,192.168.6.11

auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx

osd pool default size = 2
osd pool default min size = 2
osd pool default pg num = 32
osd pool default pgp num = 32
```

```
osd max object name len = 256
osd max object namespace len = 64

[osd]
osd journal size = 5120
filestore max sync interval = 20
filestore min sync interval = 10
osd crush chooseleaf type = 1

[mgr]
mgr_modules = dashboard
```

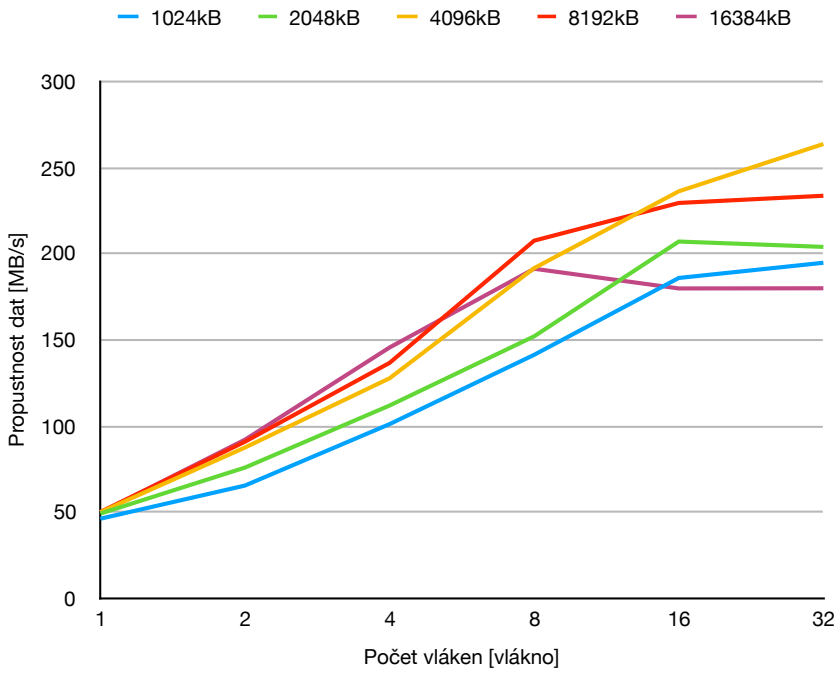
Významy a důvody použití jednotlivých atributů jsou popsány v kapitole 4 „Archivace obrazových dat“.

5.3 Výsledky měření

Nejprve představíme možnosti škálovatelnosti výkonu našeho systému. Provedeme měření výkonosti operací zápisu a čtení s použitím 1 až 32 klientů (tj. vláken), kteří zapisují nebo čtou objekty o velikosti 1 až 16 MB. Všechna tato měření probíhala po dobu 300 sekund. Následně představíme výsledky dlouhodobých měření 16 souběžně zapisujících klientů, kteří zapisují objekty o velikosti 1,5 MiB po dobu 16 hodin, tj. po dosažení stavu `HEALTH_WARN`. Nakonec představíme podobné měření z pohledu CPVA.

5.3.1 Měření zápisu objektů

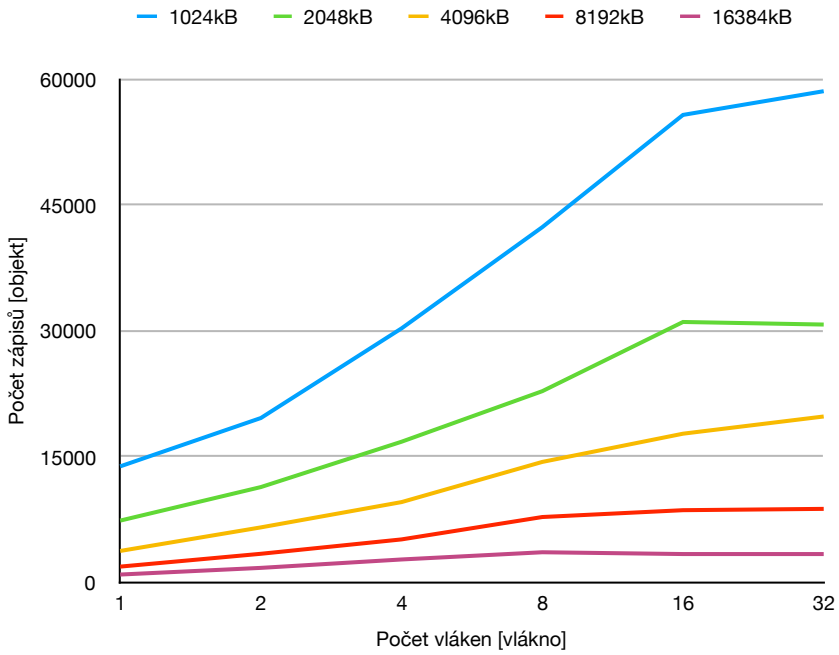
Následující hodnoty pochází z měření, které probíhalo po dobu 300 sekund a které proběhlo nad Ceph clusterem se 100 % volné kapacity na počátku měření.



Obrázek 5.1. Škálovatelnost zápisu objektů - propustnost dat.

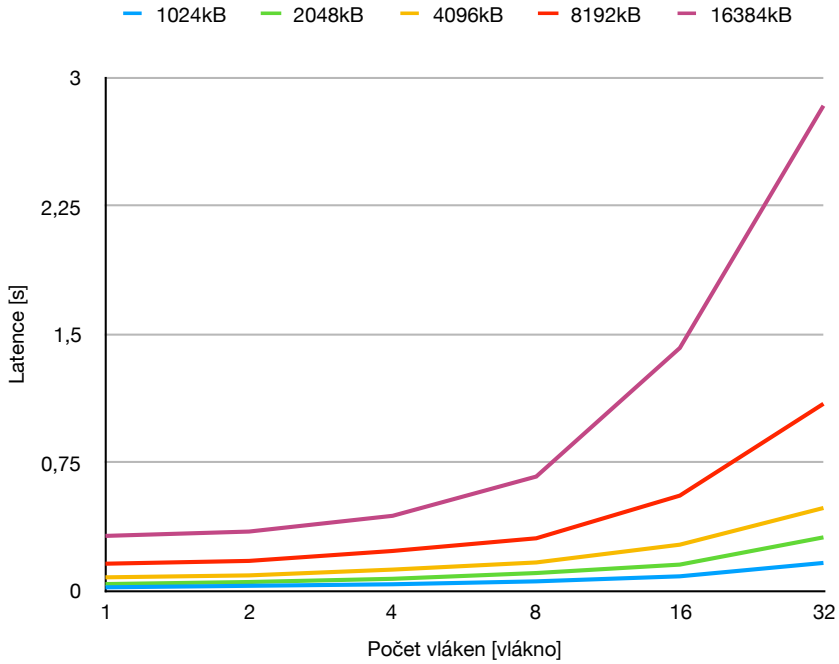
Obrázek 5.1 zobrazuje naměřené hodnoty odpovídající průměrné propustnosti dat v jednotkách MB za sekundu. Systém je lineárně škálovatelný až po dosažení limitů

hardware. Zmíněné limity představuje součet propustností operací zápisu všech disků, na které jsou mapovány instance `ceph-osd`.



Obrázek 5.2. Škálovatelnost zápisu objektů - počet objektů.

Obrázek 5.2 zobrazuje naměřené hodnoty odpovídající celkovému počtu zapsaných objektů. Systém je z pohledu zapsaných objektů lineárně škálovatelný až po hodnoty saturace propustnosti dat, viz obrázek 5.1.



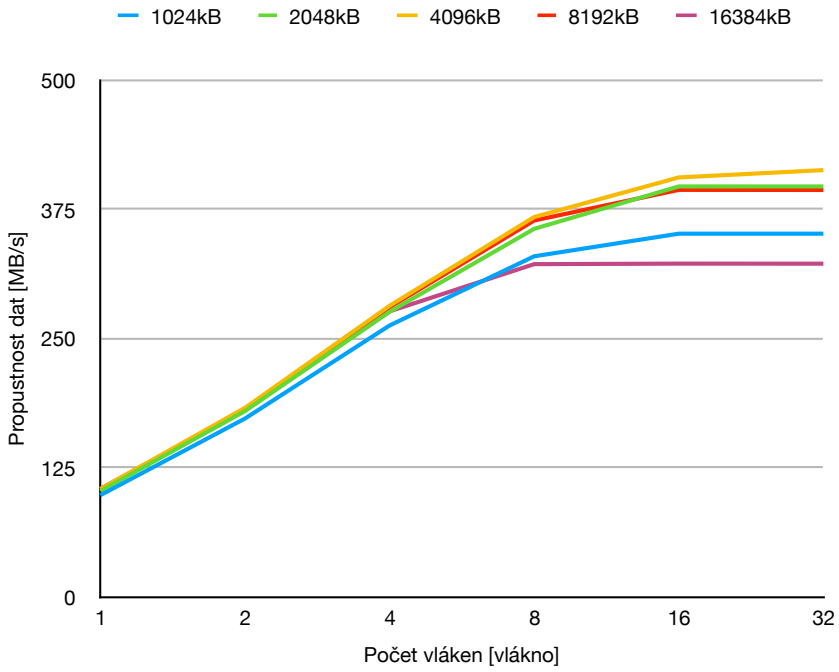
Obrázek 5.3. Škálovatelnost zápisu objektů - latence.

Obrázek 5.3 zobrazuje naměřené hodnoty odpovídající průměrné latenci. Latenci ovlivňuje nejen velikost objektu, ale také počet souběžně zapisujících klientů. S velikostí objektu přímo úměrně roste latence zápisu objektu. Mírný nárůst latence zaznamenáváme i v případě použití více klientů, kteří souběžně zapisují data do objektového

úložiště. Zvlášť znatelný nárůst latence pak je v případě, ve kterém souběžně zapisujeme objekty o velikosti 16 MB z 32 klientů. Tento nárůst je ovlivněn také vlastnosti úložných zařízení, konkrétně součtem propustnosti operací zápisu pevných disků použitých v clusteru. Celková propustnost dat je nedostačující pro zápis objektů o velikosti 16 MB za použití 32 souběžně zapisujících vláken.

5.3.2 Měření čtení objektů

Následující hodnoty pochází z měření, které probíhalo po dobu 300 sekund a které proběhlo nad Ceph clusterem, ve kterém bylo zapsáno 300 000 objektů o velikosti, ke které se vztahuje měření.

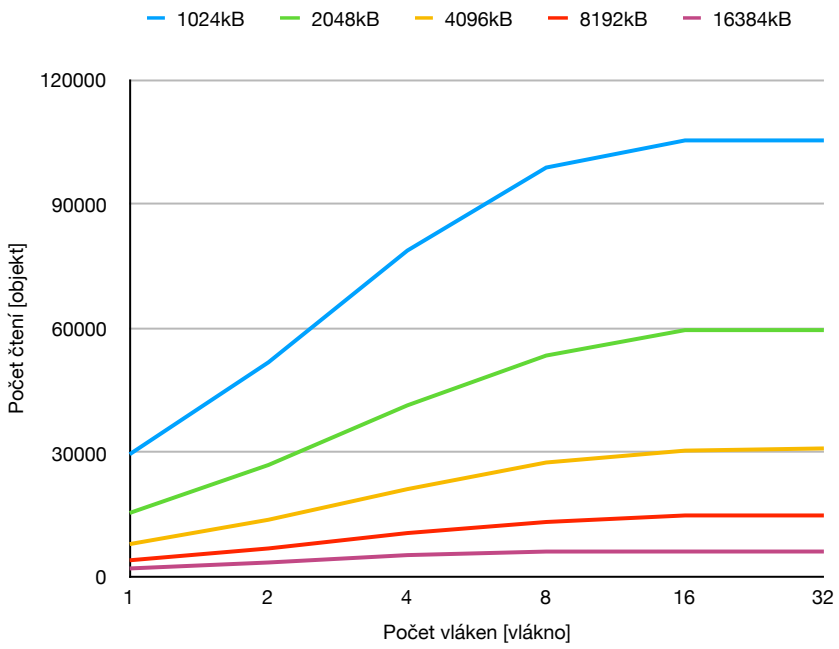


Obrázek 5.4. Škálovatelnost čtení objektů - propustnost dat.

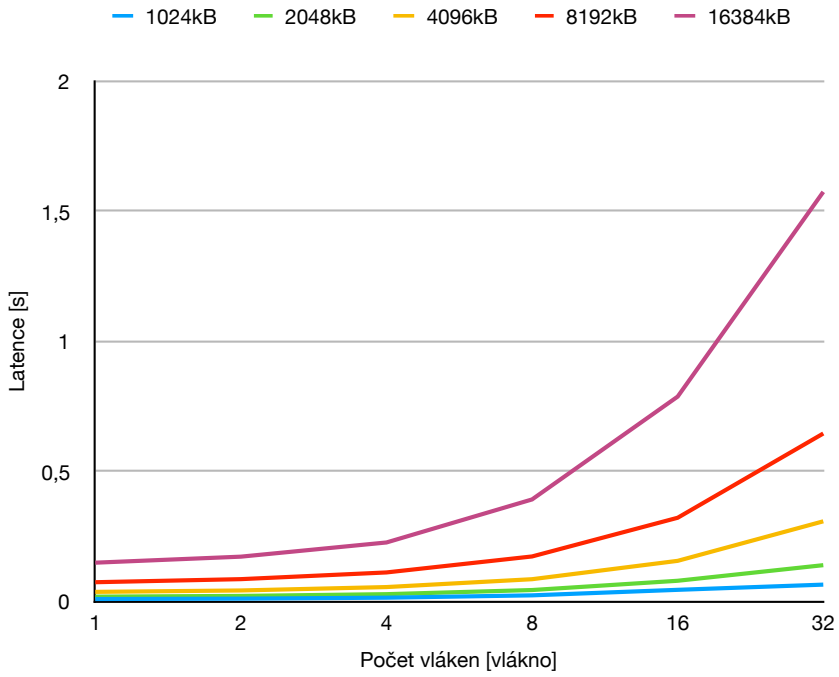
Obrázek 5.4 zobrazuje naměřené hodnoty odpovídající průměrné propustnosti dat v jednotkách MB za sekundu. Systém je lineárně škálovatelný až po dosažení limitů hardware. Zmíněné limity představuje součet propustností operací čtení všech disků, na které jsou mapovány instance `ceph-osd`.

Obrázek 5.5 zobrazuje naměřené hodnoty odpovídající celkovému počtu přečtených objektů. Počet přečtených objektů je nepřímo úměrný velikosti čteného objektu. Počet klientů, kteří čtou data, lineárně zvyšuje počet přečtených objektů až po dosažení limitů hardware z pohledu propustnosti operace čtení dat. Systém je lineárně škálovatelný z pohledu operace čtení objektů.

Obrázek 5.6 zobrazuje naměřené hodnoty odpovídající průměrné latenci operace čtení. Latenci ovlivňuje nejen velikost objektu, ale také počet klientů, kteří souběžně čtou objekty uložené v Ceph clusteru. Latence roste přímo úměrně s velikostí objektu. Mírný nárůst latence zaznamenáváme i v případě použití více klientů, kteří souběžně čtou data z objektového úložiště. Latenci během čtení objektů můžeme snížit nejen přidáním více fyzických úložných zařízení nebo zvýšením propustnosti dat, ale také použitím vyššího replikačního faktoru. Zvýšení replikačního faktoru na druhou stranu způsobí pomalejší operace zápisu objektů, jelikož vyšší hodnota replikačního faktoru vede na vytvoření většího počtu replik.



Obrázek 5.5. Škálovatelnost čtení objektů - počet objektů.

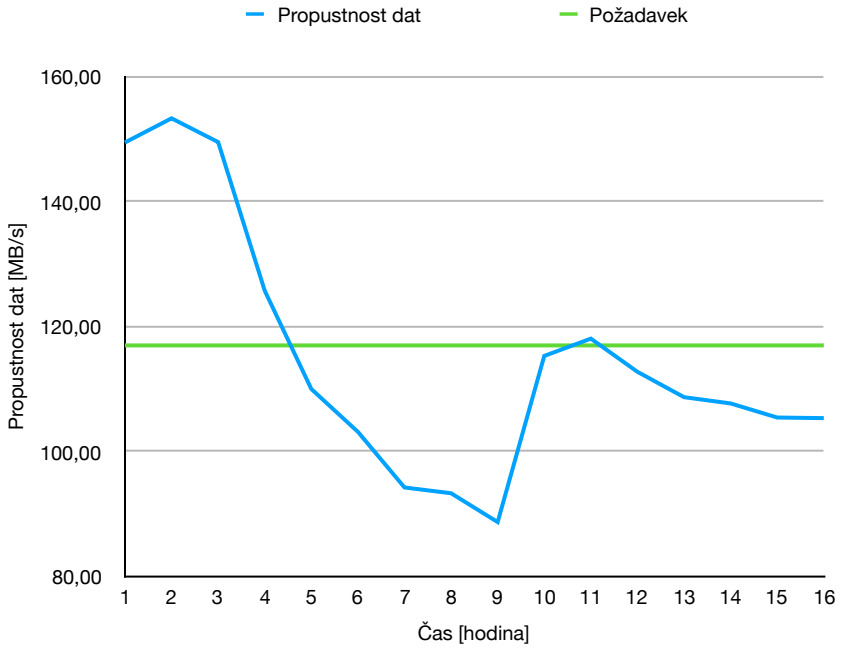


Obrázek 5.6. Škálovatelnost čtení objektů - latence.

5.3.3 Dlouhodobé měření zápisu objektů

Následující hodnoty pochází z měření, které začalo nad Ceph clusterem se 100 % volné kapacity a skončilo při dosažení stavu `HEALTH_WARN`, pokud tento stav byl způsoben zaplněním některého ceph-osd alespoň z 85 % jeho kapacity. Pro zápis bylo použito 16 souběžně zapisujících klientů zapisujících objekty o velikosti 1,5 MiB, tj. velikosti skutečných obrazových dat. Celková doba trvání měření činila 16 hodin.

Obrázek 5.7 zobrazuje naměřené hodnoty odpovídající průměrné propustnosti dat v jednotkách MB za sekundu. Pozorujeme vysokou propustnost dat na počátku mě-



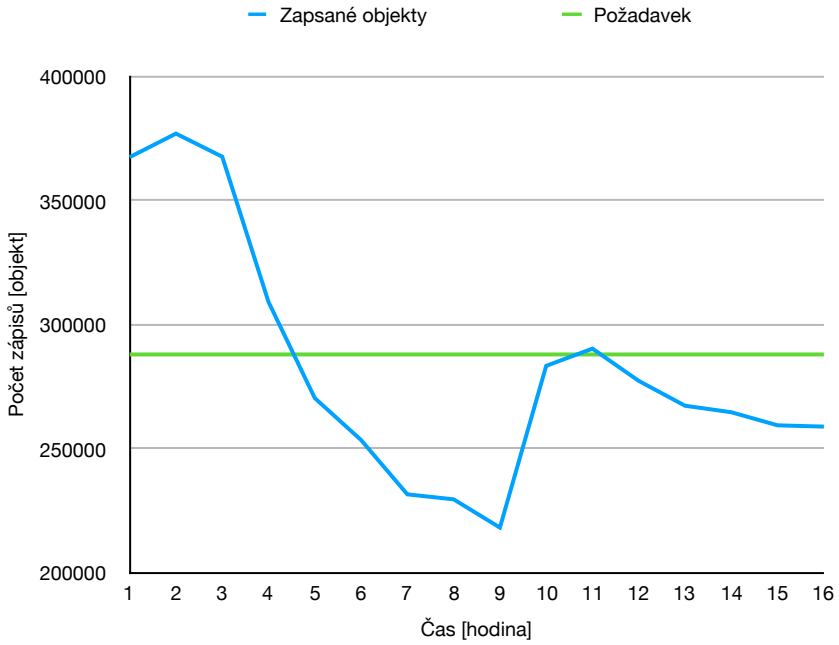
Obrázek 5.7. Dlouhodobé měření zápisu objektů - propustnost dat.

ření, kdy průměrné hodnoty během prvních třech hodin přesahují 150 MB/s. Výrazný pokles propustnosti dat nastává během čtvrté hodiny, kdy je v objektovém úložišti zapsáno přibližně 1,2 milionů objektů. Propustnost dat padá k hodnotám okolo požadované hodnoty 122 MB/s a dále klesá až do deváté hodiny měření, kdy je zapsáno přibližně 2,5 milionů objektů. Nejnižší průměrná pozorovaná hodnota dosahuje 88,7 MB/s. Během desáté hodiny se propustnost dat náhle zlepšuje a vzrůstá na hodnotu 115,3 MB/s. Toto zlepšení pozorujeme nejen během vyhodnocení výkonnosti s použitím nástroje rados bench, ale i během měření archivace obrazových dat skrze upravený CPVA. V následujících hodinách propustnost dat klesá a ustaluje se okolo hodnoty 105,2 MB/s.

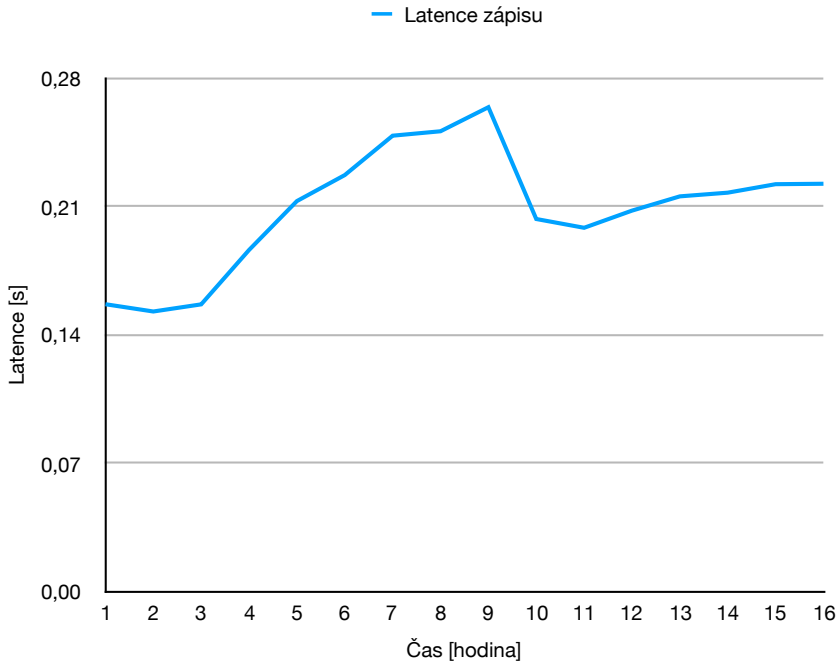
Propustnost dat je omezena limity hardware. Používáme jedenáct `ceph-osd` instancí, které jsou mapovány na jedenáct pevných disků o souhrnné kapacitě 22 TiB. Toto měření ukazuje, že ani zdaleka nedosahujeme maximální propustnosti dat během měření rados bench. Musíme tak dále hledat lepší konfiguraci Ceph clusteru, která nám umožní vyšší propustnost dat i po třetí hodině běhu tohoto měření, abychom dosáhli propustnosti dat požadované objemem obrazových dat.

Obrázek 5.8 zobrazuje naměřené hodnoty odpovídající počtu zapsaných objektů za hodinu měření. Pozorujeme vysoký počet zapsaných objektů na počátku měření. Počty zapsaných objektů během prvních třech hodin přesahují hodnotu 350 000 objektů. Během čtvrté hodiny pozorujeme pokles zapsaných objektů pod úroveň 310 000 objektů. Požadujeme archivaci 80 snímků za minutu, tj. hodnotu 288 000 zapsaných objektů za hodinu. Hodnota 310 000 tak představuje dostatečný počet zapsaných objektů. Hranice našeho požadavku je prolomena během páté hodiny a počet zapsaných objektů se dále snižuje až do deváté hodiny, kdy zapíšeme pouze necelých 220 000 objektů. Během desáté hodiny pozorujeme zlepšení a vracíme se k hodnotám okolo požadovaných 288 000 objektů, nicméně dále pokračuje mírný pokles a počty zapsaných objektů se ustálí okolo hodnoty 260 000 objektů za hodinu.

Obrázek 5.9 zobrazuje naměřené hodnoty odpovídající průměrné latenci zápisu v jednotkách sekund. Pozorujeme nízkou latenci zápisu objektů na počátku měření. Prů-



Obrázek 5.8. Dlouhodobé měření zápisu objektů - počet objektů.

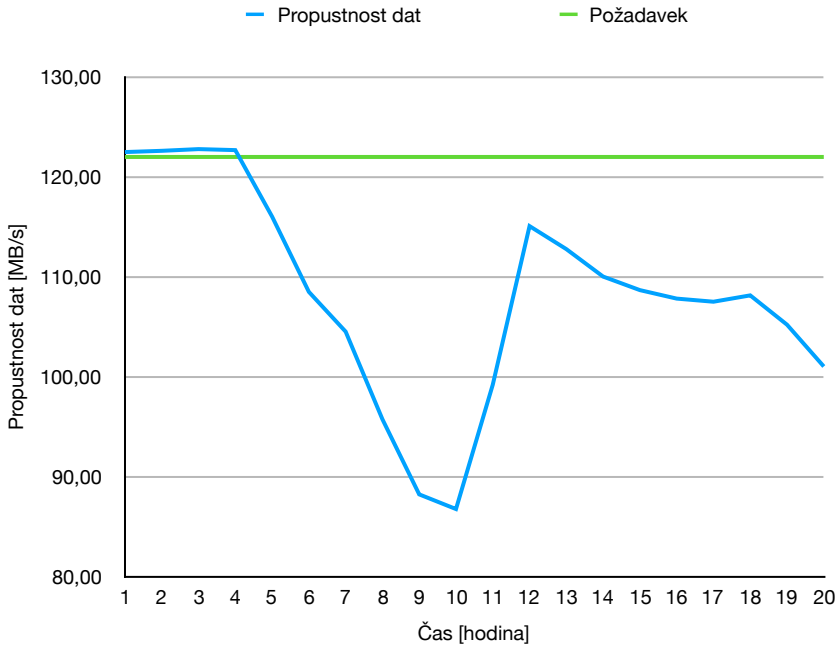


Obrázek 5.9. Dlouhodobé měření zápisu objektů - latence.

měrné hodnoty latence jsou na úrovni 0,15 s. Od čtvrté hodiny až do deváté hodiny průměrná latence zápisu vzrůstá až k hodnotám 0,26 s. Během desáté hodiny pozorujeme zlepšení k hodnotám 0,20 s následované mírnou korekcí k hodnotám okolo 0,22 s.

5.3.4 Měření výkonu CPVA

Následující hodnoty pochází z měření, které začalo nad Ceph clusterem se 100 % volné kapacity a skončilo při dosažení stavu `HEALTH_WARN`, pokud tento stav byl způsoben zaplněním některého ceph-osd alespoň z 85 % jeho kapacity. Pro zápis bylo použito 16



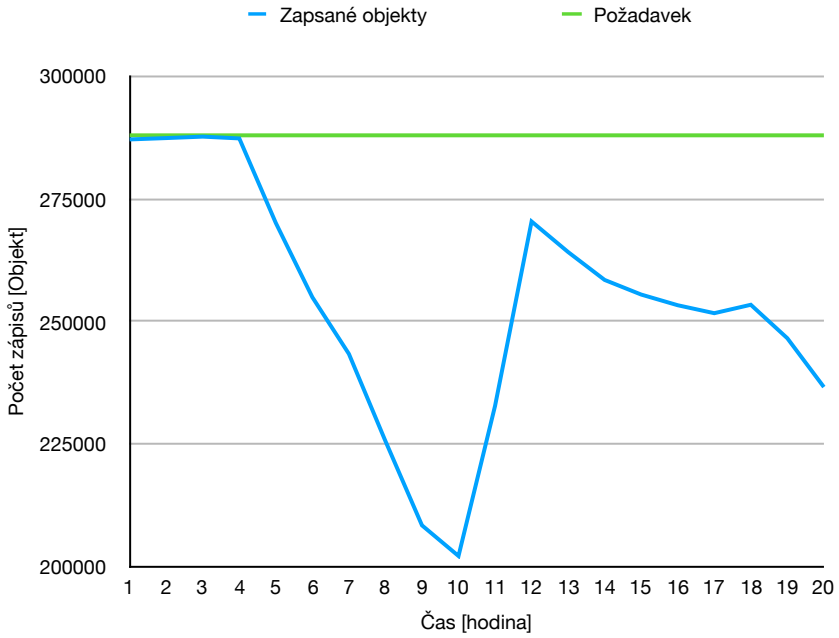
Obrázek 5.10. Dlouhodobé měření zápisu objektů aplikací CPVA - propustnost dat.

zdrojů dat, které zapisovaly data s frekvencí 5 Hz. Bylo tak zapisováno až 80 snímků o velikosti 1,5 MiB každou sekundu měření. Celková doba měření trvala 20 hodin.

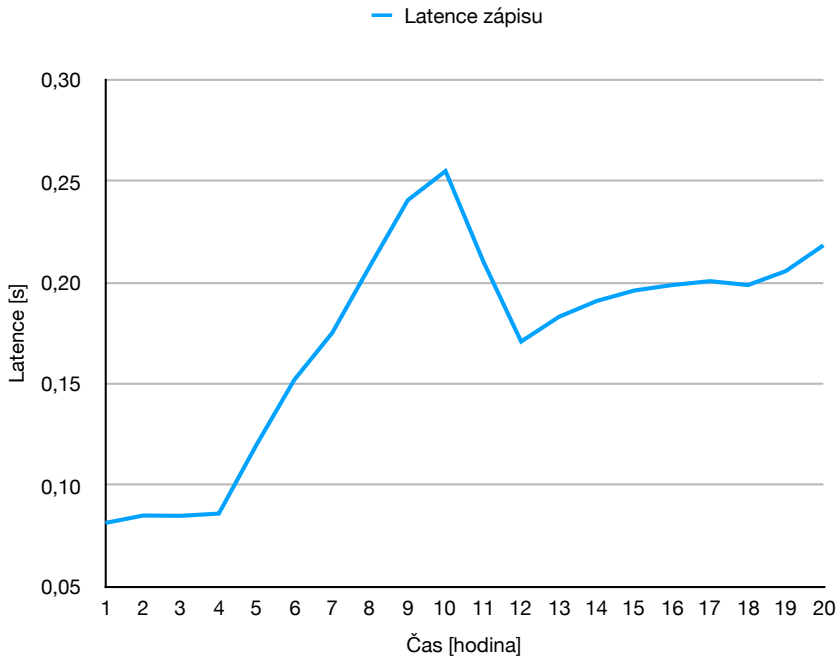
Obrázek 5.10 zobrazuje naměřené hodnoty odpovídající průměrné propustnosti dat v jednotkách MB za sekundu. Během prvních čtyř hodin měření pozorujeme propustnost dat 122,5 MB/s, která odpovídá zápisu všech objektů v úložišti. V průběhu této doby zapíšeme přibližně 1,2 milionů objektů. Během následujících několika hodin opět dochází k propadu propustnosti dat stejně jako v případě měření pomocí nástroje rados bench. Nejhorší propustnost dat zaznamenáváme během desáté hodiny, kdy je dosaženo hodnoty 2,5 milionů zapsaných objektů a kdy dosahujeme průměrné propustnosti dat 86,7 MB/s. Následuje zlepšení pozorovatelné i v průběhu dalších opakování měření. Během dvanácté hodiny se vracíme k průměrné propustnosti dat 115,1 MB/s, nicméně tato hodnota je hluboko pod optimální propustností dat. Nedochází tak k zápisu veškerých obrazových dat do objektového úložiště. V průběhu následujících hodin průměrná propustnost dat opět klesá, až se dosahujeme hodnoty průměrné hodnoty 101,2 MB/s v poslední hodině měření. Na začátku dvacáté první hodiny dosahujeme stavu `HEALTH_WARN` po zápisu 5,1 milionů objektů a ukončujeme měření.

Obrázek 5.11 zobrazuje naměřené hodnoty odpovídající počtu zapsaných objektů za hodinu měření. První čtyři hodiny měření pozorujeme optimální počet zapsaných objektů, tj. přibližně 288 000 zapsaných objektů za každou hodinu. Mezi pátou a desátou hodinou dochází k prudkému propadu množství zapsaných objektů. Nejnižší počet zapsaných objektů, tj. 202 000, zaznamenáváme během desáté hodiny měření. Následující dvě hodiny pozorujeme zlepšení, kdy zapisujeme až 270 000 objektů za hodinu. Nicméně tyto hodnoty jsou již pod optimální hodnotou 288 000 objektů za hodinu. V průběhu třinácté hodiny až po ukončení měření opět zaznamenáváme mírný propad zapsaných objektů.

Obrázek 5.12 zobrazuje naměřené hodnoty odpovídající průměrné latenci zápisu v jednotkách sekund. Pozorujeme nízkou latenci zápisu objektů na počátku měření. Průměrné hodnoty latence jsou na úrovni 0,09 s. Od páté hodiny až do dosažení desáté hodiny měření průměrná latence zápisu vzrůstá až k hodnotám 0,26 s. Během jedenácté



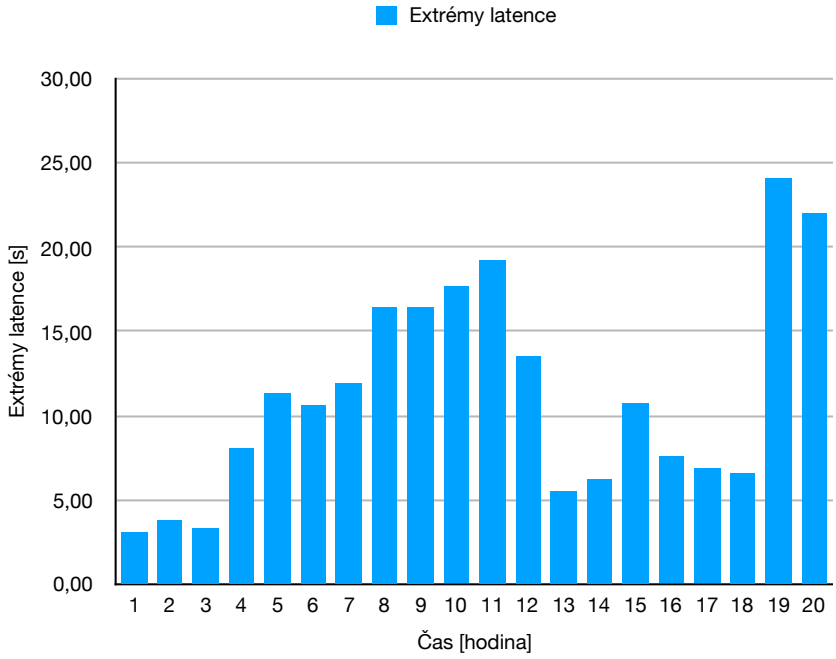
Obrázek 5.11. Dlouhodobé měření zápisu objektů aplikací CPVA - počet objektů.



Obrázek 5.12. Dlouhodobé měření zápisu objektů aplikací CPVA - latence.

a dvanácté hodiny pozorujeme zlepšení k hodnotám 0,17 s následované mírnou korekcí k hodnotám okolo 0,21 s.

Obrázek 5.13 zobrazuje naměřené hodnoty odpovídající extrémům latence zápisu v jednotkách sekund během jednotlivých hodin měření. Pozorujeme nízké extrémy pod 3 sekundy během prvních hodin třech měření. S rostoucím počtem objektů zapsaných v úložišti postupně roste i hodnota extrémů, kdy na zápis objektu během jedenácté hodiny čekáme až téměř 20 sekund. Po tuto dobu dochází k výpadku archivace obrazových dat, protože vlákno aplikace CPVA vykonávající archivaci musí čekat na *ack* od `ceph-osd`. Vzniká tak problém, při kterém nenávratně přicházíme o data. Hodnoty



Obrázek 5.13. Dlouhodobé měření zápisu objektů aplikací CPVA - extrémy latence.

extrémů se následně snižují, ovšem při téměř úplném zaplnění objektového úložiště opět opakovaně dosahují hodnot nad 20 sekund.

5.4 Závěry měření

Měření škálovatelnosti ověřuje lineární škálovatelnost objektového úložiště. Větší počet vláken, která souběžně čtou nebo zapisují data, škáluje výkon systému až po dosažení limitů hardware použitého pro sestavení úložiště.

Měření, která trvají 12 nebo více hodin, odhalují nedostatek testované sestavy. Přibližně po zápisu 1,2 milionů objektů dochází ke ztrátě výkonu dokonce pod námi požadovanou minimální hodnotu propustnosti dat. Tento problém odhalují nejen měření prováděná pomocí nástroje rados bench, ale také měření prováděná skrze upravený CPVA, který archivoval vzorová obrazová data o velikosti 1,5 MiB. Zároveň dochází k extrémním hodnotám latence během druhé a třetí třetiny doby měření, kdy na zápis objektů čekáme i více než 20 sekund. Toto čekání blokuje archivaci dat, v důsledku čehož ztrácíme data bez možnosti jejich obnovy.
Jako řešení odhaleného problému navrhujeme tyto varianty:

- Použití erasure coding namísto replikačního faktoru, čímž snížíme počet replik každého zapsaného objektu. S použitím *erasure coding* `4+2` zapíšeme objem dat odpovídající jeden a půl násobku původního objemu dat, zatímco replikační faktor 2 zapisuje objem dat odpovídající dvou násobku původního objemu dat. Na druhou stranu tím snížíme propustnost operace čtení, což nám nevadí, protože my data především archivujeme. K operacím čtení dochází minimálně.
- Upravení nastavení `/etc/ceph/ceph.conf` tak, abychom se pokusili narovnat výkon celého systému po celou dobu. Nabízí se použití vyrovnávací vrstvy, která se bude skládat z rychlých SSD, čímž můžeme odstranit extrémy latence a celkově navýšit výkon systému.
- Propustnost dat pak můžeme navýšit větším množstvím úložných zařízení. V tuto chvíli je hodnota propustnosti dat součtem jedenácti pevných disků. Přidáním do-

statečného počtu dalších pevných disků navýšíme propustnost dat na hodnoty požadované pro zápis alespoň 80 objektů o velikosti 1,5 MiB za sekundu.

- Propustnost dat můžeme zvýšit také nahrazením pomalých disků za rychlejší disky, např. SSD, ale toto řešení je významně finančně náročnější než předchozí varianty.

Kapitola 6
Závěr

Systém archivace časových řad obrazových dat rozšiřuje archivaci skalárních nebo vektorových dat, přičemž je založen na stejných technologiích a principech jako původní systém s výjimkou přidání nového úložiště, které je vhodné pro archivaci rozměrných dat. V systému jsou implementovány nové možnosti nastavení, jež uživatelů umožňují filtrovat data z pohledu jejich zdroje, datového typu, objemu nebo formátu na základě hlavičky dat a určit, kam mají být tato konkrétní data archivována. Systém je tak vhodný nejen pro účely archivace obrazových dat do objektového úložiště, ale také pro archivaci libovolných dat, která lze popsat těmito filtry.

Rozšíření CPVA je vytvořeno jako samostatný modul, který lze snadným zásahem upravit tak, abychom nebyli vázáni na použití úložného systému Ceph. Volání knihovny librados lze nahradit za jiné API nejen distribuovaného systému Ceph, ale i za API jiných systémů a v důsledku tak použít jiné úložné řešení, např. Apache Hadoop, GlusterFS, Infinit a další.

Během této diplomové práce jsme neřešili pouze rozšíření systému archivace dat. Měli jsme možnost vyzkoušet si různá distribuovaná řešení ještě před rozhodnutím, který úložný systém použijeme pro rozšíření původního systému archivace dat. Ve virtualizovaném prostředí jsme nasadili systémy Apache Hadoop, GlusterFS, Infinit, Ceph a další. Vyzkoušeli jsme různá API poskytovaná těmito systémy a propojili tak tato úložiště s prototypem upraveného CPVA. Měli jsme možnost sledovat, jak tyto systémy reagují minimálně na úrovni virtualizace. Během experimentů s těmito systémy jsme si také vyzkoušeli různé scénáře selhání, které nám pomohly pochopit chování distribuovaných systémů vice do hloubky a které nás naučily vhodně reagovat na takové situace.

Nejtěžší částí rozšíření původního systému není ani výběr distribuovaného řešení, ani propojení vybraného řešení se systémem archivace dat. Naopak vůbec nejtěžší částí je konfigurace systému a sestavování clusteru tak, abychom dosáhli požadovaného výkonu. Rozšíření našeho systému zasahuje do více úrovní. Na úrovni hardware musíme vybrat vhodné komponenty:

- Výkonné více jádrové procesory, které nám umožní paralelní zpracování velkého množství archivovaných dat.
- Dostatečné množství paměti RAM, protože pracujeme se systémy, které jsou citlivé na nedostatek paměti z důvodu např. interních procesů.
- Rychlé disky pro práci systému a vhodné disky pro archivaci velkého množství dat. Zvlášť propustnost operace zápisu u disků pro archivaci dat se v průběhu experimentů ukazuje jako klíčová, abychom dosáhli požadovaného výkonu.
- Celý systém je založen na síťové komunikaci. Běžně dostupné 1 GbE síťové prvky nejsou dostatečné pro přenos velkého množství dat po síti a musíme z pohledu propustnosti volit výkonnější 10 GbE prvky.

Náš problém vyžaduje řešení i na úrovni software, kde musíme vybrat vhodný a ideálně jednotný operační systém. Nasazení distribuovaného systému jako např. Ceph vyžaduje znalost linuxových operačních systémů na velmi dobré úrovni.

Vzhledem k tomu, že se pohybujeme v distribuovaném prostředí s velkým množstvím pevných disků, které mají netriviální pravděpodobnost selhání, musíme vhodně reagovat na veškeré chyby a selhání komponent, ze kterých se náš systém skládá. Musíme pravidelně sledovat a vyhodnocovat stav systému. Abychom zmírnili dopad případných selhání, spoléháme se na systémy s decentralizovanou nebo distribuovanou architekturou, čímž odpadá slabé místo v našem systému. Jakékoliv lokální selhání tak nevede na výpadek celého systému.

6.1 Budoucí práce

Veškerá práce a vyhodnocení systému proběhlo nad testovací sestavou, která více do hloubky odhalila vlastnosti vybraného úložiště. V první řade rozšíříme stávající systém o další fyzická úložná zařízení, abychom navýšili nejen kapacitu úložiště, ale i propustnost operace zápisu do cílové hodnoty, která nám umožní archivovat požadovaný počet snímků po celou dobu archivace a která zajistí minimální ztrátovost obrazových dat. Tuto sestavu ovšem nepoužijeme v produkčním prostředí ani po rozšíření. Vytvoříme dvě sestavy clusteru, první pro účely testování a druhou nasazenou v produkčním prostředí. Veškeré naše experimenty budeme schopni provádět nad rozšířenou experimentální sestavou, aniž by hrozila ztráta produkčních dat. Zároveň tato experimentální sestava bude sloužit pro replikaci chyb v produkční sestavě a pro hledání jejich řešení.

V tuto chvíli používáme Ceph cluster nasazený v podobě objektového úložiště pro archivaci obrazových dat. V následujících měsících přidáme instance ceph-mds a budeme používat Ceph cluster také jako distribuovaný souborový systém pro účely jiné než archivace obrazových dat.

V dlouhodobém časovém úseku zavedeme do systému klasifikaci obrazových dat na základě jejich vlastností, což nám umožní efektivnější analýzu nad jednotlivými snímky a zároveň sníží objem archivovaných dat. Popis dat na základě vybraných atributů plně obsahově nahradí rozsáhlá data. Úložiště pak bude využíváno pouze jako dočasné úložiště rozsáhlých dat mezi jejich vytvořením a klasifikací.

Literatura

[1] Irena Holubová, Jiří Kosek, Karel Minařík a David Novák. *Big Data a NoSQL databáze*. Grada Publishing, a.s., 2015. ISBN 978-80-247-5466-6.

[2] MongoDB. *The MongoDB 3.6 Manual*. 2018. https://docs.mongodb.com.

[3] The Apache Software Foundation. *Apache Cassandra Documentation v4.0*. 2018. http://cassandra.apache.org/doc/latest/.

[4] Sebastian Marsching. *Cassandra PV Archiver Reference Manual*. 2017. https://oss.aquenos.com/cassandra-pv-archiver/docs/3.2.5/manual/htmlsingle.

[5] Sebastian Marsching. *EPICS Jackie Reference Manual*. 2018. https://oss.aquenos.com/epics/jackie/docs/2.0.0/manual/.

[6] Jeffrey O. Hill a Ralph Lange. *EPICS R3.15 Channel Access Reference Manual*. 2009. https://epics.anl.gov/base/R3-15/5-docs/CAref.html.

[7] Mark Massé. *REST API Design Rulebook*. O'Reilly Media, Inc., 2012. ISBN 978-1-449-31050-9.

[8] Piotr Robert Konopelko. *MooseFS 3.0 User's Manual*. 2017. https://moosefs.com/Content/Downloads/moosefs-3-0-users-manual.pdf.

[9] Björn Kolbeck, Jan Stender, Michael Berlin, Christoph Kleineweber, Matthias Noack, Paul Seiferth, Felix Langner, NEC HPC Europe, Felix Hupfeld, Juan Gonzales, Patrick Schäfer, Lukas Kairies, Jens V. Fischer, Johannes Dillmann a Robert Schmidtke. *The XtreemFS Installation and User Guide*. 2015. http://www.xtreemfs.org/xtfs-guide-1.5.1.pdf.

[10] Infinit. *Infinit Documentation*. 2018. https://infinit.sh/documentation/reference.

[11] The Apache Software Foundation. *Apache Hadoop 3.1.0*. 2018. http://hadoop.apache.org/docs/current/.

[12] Benjamin Depardon, Gaël Le Mahec a Cyril Séguin. *Analysis of six distributed file systems*. 2013. https://hal.inria.fr/hal-00789086/.

[13] Sanjay Ghemawat, Howard Gobioff a Shun-Tak Leung. *The Google File System*. 2003. https://dl.acm.org/citation.cfm?id=945450.

[14] Lawrence Page, Sergey Brin, Rajeev Motwani a Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web*. 1998. http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf.

[15] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long a Carlos Maltzahn. *Ceph: A scalable, high-performance distributed file system*. In: *Proceedings of the 7th symposium on Operating systems design and implementation*. 2006. 307–320.

[16] Karan Singh. *Ceph Cookbook*. Packt Publishing Ltd, 2016. ISBN 9781784393502.

[17] Sage A Weil. *Ceph: reliable, scalable, and high-performance distributed storage*. 2007.

[18] Red Hat Inc. *Ceph Documentation*. 2018. http://docs.ceph.com/docs/master/.

[19] Sage A Weil, Andrew W Leung, Scott A Brandt a Carlos Maltzahn. *Rados: a scalable, reliable storage service for petabyte-scale storage clusters.* In: *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07*. 2007. 35–44.

[20] Sage A Weil, Scott A Brandt, Ethan L Miller a Carlos Maltzahn. *CRUSH: Controlled, scalable, decentralized placement of replicated data.* In: *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. 2006. 122.

[21] Christian Huebner, Pawel Stefanski, Kostiantyn Danilov a Igor Fedotov. *Ceph Best Practices Manual*. 2016. https://docs.mirantis.com/openstack/fuel/fuel-9.1/assets/CephBestPractices.pdf.

[22] Gluster Inc. *GlusterFS Documentation*. 2018. https://docs.gluster.org/en/latest/.

[23] Gluster Inc. *Gluster: Performance in a Gluster System*. 2011. https://s3.amazonaws.com/aws001/guided_trek/Performance_in_a_Gluster_Systemv6F.pdf.

[24] Alex Davies a Alessandro Orsaria. Scale out with GlusterFS. *Linux Journal*. 2013, 2013 (235),

[25] Gluster Inc. *Gluster: An Introduction to Gluster Architecture*. 2011. http://moo.nac.uci.edu/~hjm/fs/An_Introduction_To_Gluster_ArchitectureV7_110708.pdf.

Příloha A
Slovníček zkratek

ACL	Access Control List
AFR	Automatic File Replication
API	Application Programming Interface
BCNF	Boyce–Codd Normal Form
BMP	Windows Bitmap
CERN	Conseil Européen pour la Recherche Nucléaire
CLI	Command Line Interface
CPU	Central Processing Unit
CPVA	Cassandra PV Archiver
CQL	Cassandra Query Language
CRC	Cyclic Redundancy Check
CRUSH	Controlled, Scalable, Decentralized Placement of Replicated Data
CSS	Control System Studio
DFS	Distributed File System
DHT	Distributed Hash Table
EHA	Elastic Hash Algorithm
EPICS	The Experimental Physics and Industrial Control System
FUSE	Filesystem in Userspace
GC	Garbage Collector
GFS	Google File System
GNU	GNU's Not Unix
GPL	General Public License
GUI	Graphical User Interface
HDD	Hard Disk Drive
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IOPS	Input/Output Operations per Second
JDK	Java Development Kit
JMX	Java Management Extensions
JNA	Java Native Access
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KO	Kernel Object
LTS	Long Term Support
MDS	Metadata Server
MON	Monitor
NFS	Network File System
NTP	Network Time Protocol
OS	Operating System

OSD	Object Storage Device
PDF	Portable Document Format
PG	Placement Group
PNG	Portable Network Graphics
POSIX	Portable Operating System Interface
PV	Process Variable
QEMU	Quick EMUlator
RADOS	Reliable Autonomous Distributed Object Storage
RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
RBD	Rados Block Device
REST	Representational State Transfer
RGW	Rados Gateway
RHEL	Red Hat Enterprise Linux
RPC	Remote Procedure Call
SMB	Server Message Block
SQL	Structured Query Language
SSD	Solid State Drive
TIFF	Tagged Image File Format
UUID	Universally Unique Identifier
VFS	Virtual File System
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language
YARN	Yet Another Resource Negotiator
1NF	First Normal Form
2NF	Second Normal Form
3NF	Third Normal Form

Příloha B
Obsah přiloženého DVD

`./measurements`	Složka, která obsahuje výsledky měření a soubory pro konfiguraci systému archivace dat, tj. CPVA a Ceph clusteru.
`./measurements/2018*.zip`	Archivy, které obsahují záznamy jednotlivých měření. Jedná se o záznamy běhů buď nástroje rados bench, nebo CPVA.
`./measurements/ceph.conf`	Konfigurační soubor Ceph clusteru.
`./measurements/script.sh`	Script, který umožní spustit přednastavený rados bench v módu zápisu dat.
`./measurements/scriptSeqRead.sh`	Script, který umožní spustit přednastavený rados bench v módu sekvenčního čtení dat.
`./measurements/scriptWrite*.sh`	Přednastavené skripty, které umožní spustit měření zápisu dat pomocí nástroje rados bench za použití 16 vláken zapisujících souborvy o velikosti 1500 kiB po dobu 60, 300 nebo 3600 sekund.
`./sources`	Složka, která obsahuje zdrojové kódy.
`./sources/ceph-plugin`	Složka, která obsahuje zdrojové kódy námi implementovaného rozšíření CPVA. Složka obsahuje celý projekt, který lze sestavit pomocí nástroje Apache Maven.
`./sources/ceph-tools`	Složka, která obsahuje zdrojové kódy námi implementovaného naivního nástroje pro zápis objemných dat do úložiště Ceph.
`./thesis-pdf`	Složka, která obsahuje tuto diplomovou práci ve formátu PDF.
`./thesis-pdf/thesis.pdf`	Tato diplomová práce ve formátu PDF.
`./thesis-tex`	Složka, která obsahuje zdrojový formát této diplomové práce včetně obrázků ve formátu PDF.
`./thesis-tex/images`	Složka, která obsahuje obrázky.
`./thesis-tex/Untitled.tex`	Zdrojový formát této diplomové práce.
`./thesis-tex/*.*`	Soubory, které jsou nutné pro vytvoření PDF verze této diplomové práce.
`./readme.txt`	Soubor readme.txt.