VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Jiří Kozlovský

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

OPTIMALIZACE ČTENÍ DAT Z DISTRIBUOVANÉ DATABÁZE
OPTIMIZATION OF DATA READING FROM A DISTRIBUTED DATABASE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Jiří Kozlovský
AUTHOR

VEDOUCÍ PRÁCE Ing. Tomáš Macho, Ph.D.
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Jiří Kozlovský ***ID:*** 151649
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Optimalizace čtení dat z distribuované databáze

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou NoSQL databází a časově granulovaným přístupem ke konkrétní skupině dat.
2. Proveďte rešerši možných optimalizací pro časově granulovaný přístup ke konkrétní skupině dat v databázi Apache HBase. Jednotlivé metody porovnejte.
3. Zvolte nejvhodnější metodu optimalizace čtení dat z Apache HBase a implementujte ji.
4. Otestujte funkčnost implementace na zvolené množině dat.
5. Vyhodnoťte dosažené výsledky.

DOPORUČENÁ LITERATURA:

Snively, Ben. Combine NoSQL and Massively Parallel Analytics Using Apache HBase and Apache Hive on Amazon EMR. Amazon EMR, 2016.

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 13.5.2019

Vedoucí práce: Ing. Tomáš Macho, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá optimalizací čtení dat z distribuované NoSQL databáze Apache HBase s ohledem na požadovanou granularitu dat. Zadání vzniklo jako produktový požadavek firmy Seznam.cz, a.s. divize Reklamy, nákladového střediska Sklik.cz za účelem vylepšení uživatelské zkušenosti zpřístupněním filtrace agregovaných statistických dat uživatelům inzerentské webové aplikace pro zobrazení historie výkonnosti entit.

KLÍČOVÁ SLOVA

Apache HBase, HBase, Distribuovaná databáze, Datové úložiště, NoSQL, Hadoop, HDFS, Koprocesor, Granularita dat

ABSTRACT

This thesis is focused on optimization of data reading from distributed NoSQL database Apache HBase with regards to the desired data granularity. The assignment was created as a product request from Seznam.cz, a.s. the Reklama division, Sklik.cz cost center to improve user experience by making filtering of aggregated statistical data available to advertiser web application users for the purpose of viewing entity performance history.

KEYWORDS

Apache HBase, HBase, Distributed database, Data storage, NoSQL, Hadoop, HDFS, Coprocessor, Data granularity

KOZLOVSKÝ, Jiří. *Optimalizace čtení dat z distribuované databáze*. Brno, 2019, 72 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: Ing. Tomáš Macho, Ph.D.

Vysázeno pomocí balíčku `thesis` verze 3.03; `http://latex.feec.vutbr.cz`

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Optimalizace čtení dat z distribuované databáze“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Děkuji svému vedoucímu diplomové práce, panu Ing. Tomáši Machovi, Ph.D., za konzultace a podnětné návrhy k práci. Děkuji také panu Ing. Michalu Kuchtovi za odborné vedení a vloženou důvěru při implementaci celého řešení.

Rovněž patří můj dík rodině za podporu při studiu a vytvoření k tomu vhodného zázemí. Děkuji také mé přítelkyni za její trpělivost se mnou nejen při psaní této diplomové práce.

Brno ....

podpis autora

Obsah

Úvod **11**

1 Rozbor zadání **13**

2 NoSQL databáze **16**
2.1 NoSQL vs. SQL . . . 16
2.2 Dělení NoSQL databází . . . 17

3 Apache HBase **18**
3.1 Datový model HBase . . . 18
3.1.1 Namespace . . . 19
3.1.2 Table . . . 19
3.1.3 Column . . . 19
3.1.4 Row . . . 20
3.1.5 Cell . . . 20
3.1.6 Timestamp . . . 20
3.2 Operace datového modelu . . . 21
3.2.1 Get . . . 21
3.2.2 Put . . . 21
3.2.3 Scan . . . 21
3.2.4 Delete . . . 21
3.3 Architektura HBase . . . 22
3.3.1 NameNode . . . 22
3.3.2 DataNode . . . 22
3.3.3 Region a RegionServer . . . 23
3.3.4 Master Server . . . 24
3.3.5 Zookeeper . . . 25
3.3.6 Jak komponenty spolupracují . . . 25
3.3.7 Způsob uložení dat . . . 26
3.3.8 Způsob čtení dat . . . 28
3.4 Aplikační rozhraní HBase . . . 31
3.4.1 Native Java API . . . 31
3.4.2 External API . . . 31
3.4.3 Apache HBase Coprocessors . . . 33

4 Zápis statistických dat **34**
4.1 Granularita dat . . . 34
4.2 Agregace statistických dat . . . 34

4.2.1 Inkrementální agregátory 35
4.2.2 Denní agregátory 36
4.2.3 Měsíční agregátory 37
4.3 Pravidelné spouštění agregací 38

5 Čtení statistických dat 39
5.1 Architektura komponent 39
5.1.1 Frontend-API 40
5.1.2 Statserver-Java 41
5.1.3 Sortserver 42
5.1.4 SortAndFilterEndpoint Coprocessor 43
5.2 Proč Endpoint Coprocessor? 44

6 Volba metody optimalizace čtení dat 45
6.1 Návrh rozhraní mezi frontend a HBase 45
6.2 HBase rozhraní 46
6.3 Optimalizace čtení dat 47

7 Zajištění statistik s týdenní časovou granularitou 48
7.1 Příprava HBase schématu 48
7.2 Implementace týdenního agregátoru 49
7.2.1 Způsob uložení agregovaných dat 49
7.2.2 Volitelné omezení množiny agregovaných dat 50
7.2.3 Proces sčítání denních statistik 51

8 Implementace rozhraní koprocesoru 53
8.1 Definice komunikačního protokolu 53
8.2 Metoda getGranularizedSumStats 54
8.2.1 Inicializace třídy ScanSet 54
8.2.2 Inicializace statistické mapy 55
8.2.3 Skenování dat 56
8.2.4 Tvorba odpovědi 56

9 Implementace rozhraní sortserveru 57
9.1 Implementace granularizedSumStats metody pro všechny druhy entit 57

10 Integrační testy navrženého systému 58
10.1 Formát integračních testů 58
10.2 Kubernetes sandbox 59
10.3 Testovací scénáře 59
10.4 HBase v Dockeru 60

10.4.1 Import schématu a dat 60
10.4.2 Agregace dat 60
10.5 GitLab CI 61

11 Dosažené výsledky **62**
11.1 Vliv změn na existující rozhraní HBase 62
11.2 Vliv změn na existující sortserver rozhraní 63

12 Závěr **64**

Zdroje **65**

Seznam zkratek **70**

Seznam příloh **71**

A Obsah přiloženého paměťového média **72**

Seznam obrázků

1.1 Přehledový graf statistické historie všech kampaní 13
1.2 Hromadný náhled kampaní s aktivními filtry 14
3.1 Nepřímá úměra mezi kardinalitou dotazu a rychlosti jeho zpracování [6] 19
3.2 `HBase` komponenty v master-slave architektuře [17] 22
3.3 `HBase` regiony na `RegionServerech` [17] 23
3.4 `HBase Master Server` v roli `HBase` mastera [17] 24
3.5 Role `Zookeepera` v `HBase` architektuře [17] 25
3.6 Mechanismus ukládání dat v `HBase` [17] 26
3.7 Formát `HFile` souboru [15] 27
3.8 Schéma Read Merge rozhodovacího procesu [15] 28
3.9 Proces `Minor Compaction` pro redukci množství `HFile` souborů [15] . 29
3.10 Proces `Major Compaction` pro redukci množství `HFile` souborů [15] . 30
4.1 Schéma inkrementálního `clicklinker` agregátoru 35
4.2 Schéma denního agregátoru 36
4.3 Schéma měsíčního agregátoru 37
5.1 Schéma komunikace s `Frontend-API` 40
5.2 Schéma komunikace se Statserver-Java 41
5.3 Schéma komunikace se Sortserverem 42
5.4 Schéma klíče řádku v `HBase` tabulkách hromadných náhledů 43
7.1 Schéma k úvaze maximálního počtu týdnů v jednom měsíci 52

Seznam pseudokódů

7.1 Příkaz pro vytvoření týdenních rodin sloupců 49

Úvod

Cílem této práce je navrhnout a implementovat optimální čtení dat z distribuované databáze `Apache HBase` s ohledem na časovou granularitu uložených dat. Požadavek na optimalizaci vznikl ve firmě Seznam.cz a.s. divize Reklamy, nákladového střediska Sklik.cz. Účelem požadavku je přidání možnosti zobrazit inzerentům v grafu relevantní historii jejich statistických dat.

První kapitola je zaměřená na zevrubný popis problému, který je potřeba vyřešit. Kromě toho také definuje základní pojmy používané v reklamním systému.

Další kapitola se zabývá obecným úvodem do NoSQL databází, kde definuje termín NoSQL. Pak pokračuje srovnáním s databázemi SQL a nakonec uvádí příklady NoSQL databází, kde definuje jejich základní dělení a udává, do které kategorie spadá `Apache HBase` datové úložiště.

Třetí kapitola dopodrobna popisuje `Apache HBase`. Začíná jejím datovým modelem, kde se zabývá terminologií abstrakce nad způsobem uložení dat. Poté pokračuje názvoslovím podporovaných operací datového modelu, kde uvedené operace srovnává s jejich SQL protějškem. Pak následuje architektura `HBase`, ve které jsou vysvětleny funkce interních komponent. Kromě toho se tato podkapitola zabývá i způsobem distribuce dat napříč jednotlivými instancemi interních komponent. Uvádí pak, jak uvedené komponenty fungují jako celek. Součástí architektury `HBase` je i fyzický způsob uložení dat, který je vhodný znát k pochopení pozdějších úvah nad rozložením statistických dat pro optimalizaci systému. Stejně tak se podkapitola zabývá i způsobem čtení dat, které přímo souvisejí s později zvolenou optimalizací čtení dat. Na závěr kapitoly je uvedeno aplikační rozhraní `HBase`, pomocí něhož lze přistupovat k uloženým datům. Jsou zde také rozebrány výhody a nevýhody jednotlivých přístupů.

Následující čtvrtá kapitola celkově popisuje, jakým způsobem dochází k vytváření a zápisu statistických dat reklamního systému. Začíná definicí granularity dat, která souvisí s další částí, kterou je agregace statistických dat. V této části jsou ve zjednodušené formě popsány již implementované agregátory časové granularity dat. Závěrem kapitoly je uveden způsob pravidelného spouštění těchto agregátorů.

Pátá kapitola se zabývá procesem čtení statistických dat. Popisuje vzájemnou architekturu komponent, které se podílejí na procesu získávání dat na cestě od klienta reklamního systému až po samotné datové úložiště. Na konci se nachází zdůvodnění, proč existující řešení čtení dat z `HBase` používá konkrétní aplikační rozhraní.

V šesté kapitole je zeširoka popsána zvolená metoda optimalizace čtení dat. Ta především vychází ze vstupů a výstupů webového rozhraní, které se zde definují za účelem možnosti zobrazit statistický graf dle filtrů zadaných uživatelem. Kapitola pokračuje definicí vstupů a výstupů `HBase` rozhraní, což je potřeba k implementaci

fyzického čtení dat z `HBase` tak, aby bylo možné výsledný graf sestavit v požadované podobě. Ze znalostí jednotlivých rozhraní na cestě od inzerenta až po databázi je na konci kapitoly navržen chybějící článek systému, čímž se zajistí optimalizace čtení dat.

V další kapitole je čtenář proveden problematikou zajištění statistik s týdenní časovou granularitou. Toho se týká podkapitola připravení `HBase` schématu, které je nutné pro následnou implementaci týdenního agregátoru. Popis této implementace je rozdělen na způsob uložení agregovaných dat, volitelné omezení množiny agregovaných dat a na proces, jakým probíhá sčítání denních statistik.

Následuje kapitola osmá, která se zabývá provedenou implementací rozhraní koprocesoru, počínaje definicí protokolu určeného ke komunikaci s koprocesorem. Další část se věnuje nové metodě na rozhraní koprocesoru. Je zde popsána inicializace interních struktur potřebných ke správnému přečtení dat uložených v `HBase`. Na to navazuje popis procesu skenování dat s užitím inicializovaných struktur. Závěr kapitoly se zabývá vytvářením odpovědi nové metody na rozhraní koprocesoru.

Devátá kapitola je zaměřena na implementaci rozhraní `sortserveru`, což je jedna z komponent interní architektury reklamního systému.

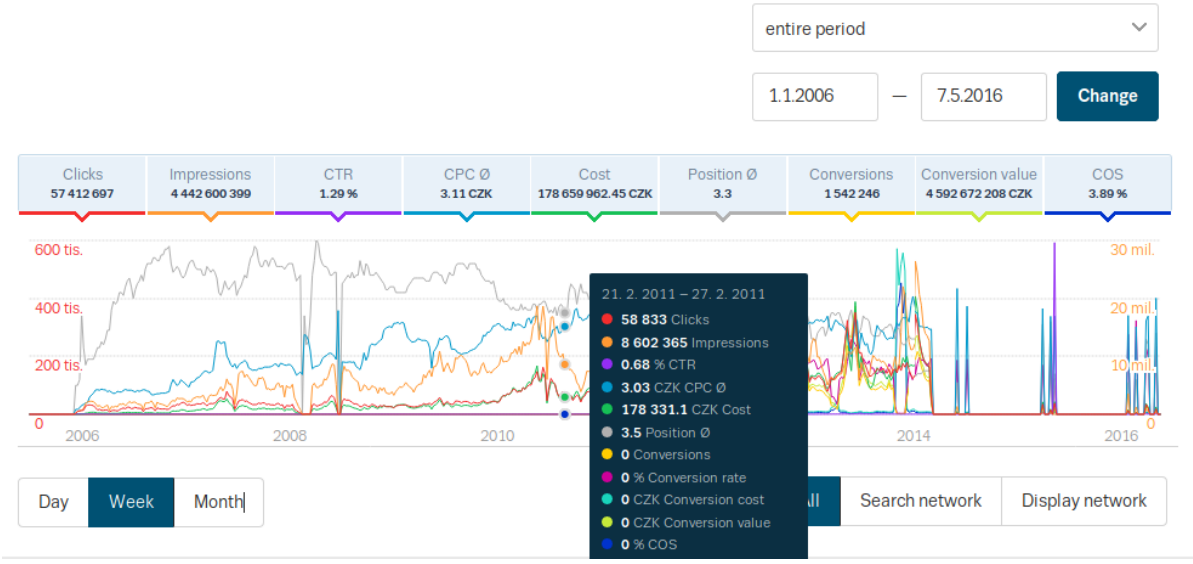
Předposlední desátá kapitola čtenáři představuje formát a implementaci integračních testů celého navrženého systému. Na konci kapitola uvádí, jakým způsobem je existence těchto testů zužitkována při dalším vývoji.

Závěrečná kapitola shrnuje dosažené výsledky provedené implementace. Zhodnocuje vliv změn provedených v `HBase` rozhraní na výkon vybrané již existující metody tohoto rozhraní za pomocí zátěžových testů. Na závěr také uvádí vliv změn na existující `sortserver` rozhraní.

1 Rozbor zadání

Reklamní systém Sklik.cz má inzerentský portál, kde inzerenti obsluhují své účty. Pouštějí zde své reklamní kampaně, editují obsah zobrazovaných inzerátů a provádějí nespočet dalších operací za účelem nakoupení reklamního prostoru.

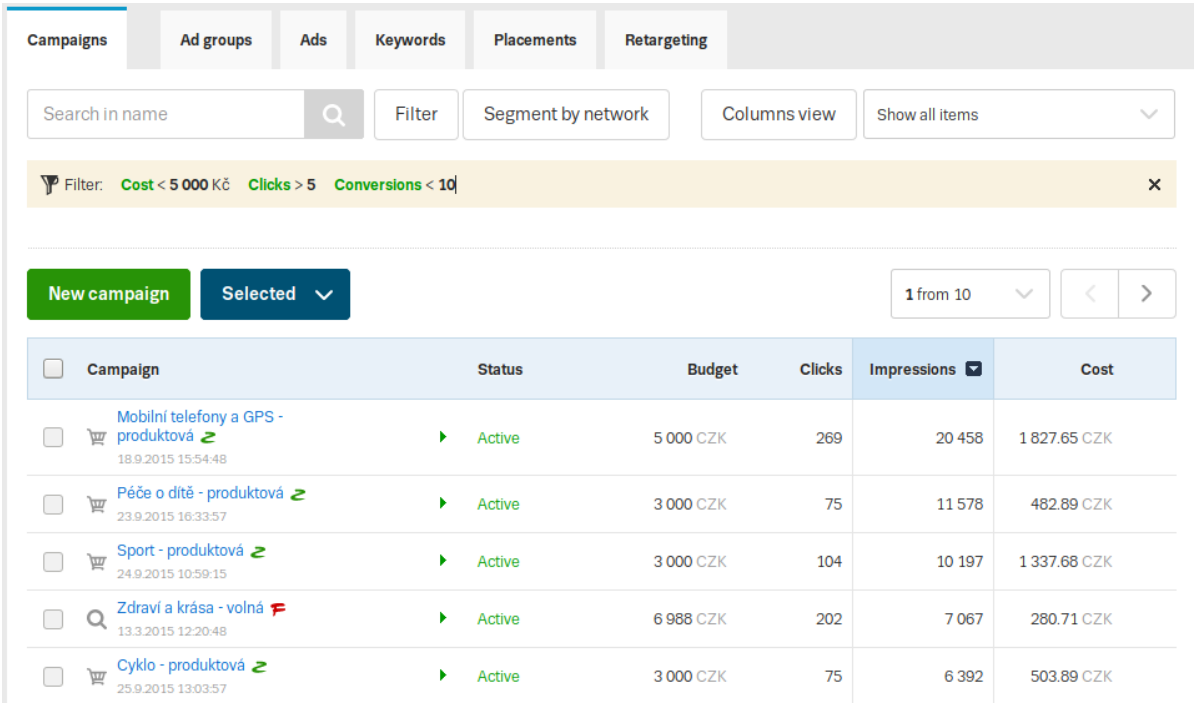
Ihned po přihlášení se inzerentovi zobrazí tabulka se základním přehledem spuštěných kampaní (tzv. hromadný náhled) a jejich statistických údajů, jako je počet kliknutí, či množství peněz, které již lidé proklikali. Nad tabulkou se nalézá přehledový graf reprezentující statistickou historii vývoje všech kampaní, viz obrázek 1.1.



Obr. 1.1: Přehledový graf statistické historie všech kampaní

Tabulka hromadného náhledu obsahuje tlačítko pro výběr filtrů. Uživatel si zde může navolit filtr nad libovolným sloupcem tabulky tak, aby se zobrazily pouze kampaně vyhovující zvolenému filtru. Na obrázku 1.2 je ukázka aktivního filtru tabulky nad třemi sloupci: cena kampaně (menší než 5 000 Kč), počet kliknutí na reklamu (větší než 5) a počet konverzí[1] (menší než 10).

[1]Konverze představuje scénář, kdy uživatel klikl na reklamu a posléze pokračoval inzerentem nadefinovanou sledovanou akcí, jako je nákup zboží, registrace, či obecně zobrazení klíčového obsahu webové stránky.



Obr. 1.2: Hromadný náhled kampaní s aktivními filtry

V tabulce se zobrazují pouze ty kampaně, které vyhovují všem zadaným filtrům. Na zobrazení přehledového grafu z obrázku 1.1 však tento filtr nemá vliv.

Přehledová tabulka také obsahuje záložky, které umožní inzerentu zobrazit hromadný náhled dalších entit, jako je *sestava*[2], *reklama*, *klíčové slovo*[3], *umístění*[4] a *retargeting*[5].

Ani změna entity hromadného náhledu nemá vliv na přehledový graf z obrázku 1.1. Z toho plyne, že přehledový graf není nijak ovlivněn entitami zobrazenými v tabulce hromadného náhledu.

Přehledový graf tedy vždy zobrazuje sumu statistik všech kampaní, nehledě na aplikovaný filtr hromadného náhledu, ani na aktuálně zobrazovanou entitu.

[2]Sestava představuje skupinu reklam, které lze do jisté míry parametrizovat společně.

[3]Klíčové slovo je slovní spojení, které určuje při jakých vyhledávacích dotazech v Seznam vyhledávači se má reklama zobrazit.

[4]Umístění definuje, na jakých stránkách se má reklama zobrazovat.

[5]Retargeting je složitý nástroj na zobrazování reklam těm uživatelům, kteří již s konkrétní reklamou nějakým způsobem přišli do styku.

Zadáním této diplomové práce je připravit `backend`[6] reklamního systému na to, aby přehledový graf respektoval jak aplikované filtry hromadného náhledu, tak aktuálně zobrazovanou entitu. Tedy pokud uživatel například vybere hromadný náhled klíčových slov, zobrazí se v přehledovém grafu (Obr. 1.1) suma statistik všech klíčových slov, které vyhovují všem zadaným filtrům hromadného náhledu.

Zároveň je však potřeba optimalizovat rychlost získávání dat tak, aby se graf vykreslil v řádu maximálně jednotek sekund. Nejedná se o triviální zadání, neboť statistických dat je příliš mnoho na to, aby se k jejich uložení dala použít klasická relační databáze a přitom aby odezva na statistický dotaz se složitým filtrem byla v řádu jednotek sekund.

Proto již existující řešení hromadných náhledů a přehledového grafu používá NoSQL technologie, jejichž výhody vysvětlím v následujících dvou kapitolách.

Současně je žádoucí zachovat již existující funkcionalitu okolo přehledového grafu. Tu zajišťují tlačítka okolo něj. Kupříkladu lze statistiky rozpadnout na dny, týdny a měsíce. To znamená, že například u rozpadu na týdny bude jeden bod v grafu reprezentovat jeden konkrétní týden.

Také lze vykreslené statistiky v grafu omezit dle časového úseku, což na pozadí umožňuje dostatečná časová granularita[7] statistických dat.

Dále lze omezit zobrazené statistiky podle cílení kampaně, které může být fulltextové[8] (tlačítko *Search network*), či kontextové[9] (tlačítko *Display network*). To znamená, že pokud uživatel omezí graf například pouze na kontextové kampaně, tak se mu zobrazí suma statistik pouze těch kampaní, které mají nastavené kontextové cílení.

[6]Backend označuje skupinu softwarových komponent, která se věnuje zpracování požadavků z `frontendu`. Uživatel tak s ní nepřichází přímo do styku. `Frontend` označuje tu skupinu softwarových komponent, se kterou uživatel interaguje přímo (webové rozhraní, veřejné API).

[7]Granularitu dat vysvětlím v podkapitole 4.1.

[8]Fulltextové cílení reklamy zajišťuje její zobrazení na stránce výsledků fulltextového vyhledávání Seznam.cz vyhledávače. Fulltext je termín označující skutečnost, že se prohledává celý obsah zaindexovaných webových stránek.

[9]Kontextové, neboli obsahové cílení reklamy, omezuje zobrazení reklamy na webové články, e-shopy a další weby obecně zajišťující pro uživatele zajímavý obsah. Spadá sem veškerá reklama, která se nezobrazuje ve výsledcích vyhledávače.

2 NoSQL databáze

Termín NoSQL odkazuje na skutečnost, že tradiční relační databáze (SQL databáze) nemusejí být vždy vhodné pro všechna řešení. Zejména pro ty, které zahrnují velké objemy dat. [1]

Termín byl však rozšířen tak, že znamená "Not only SQL" (nejen SQL), což naznačuje podporu pro případná rozhraní založená na SQL, ačkoliv databáze není relační. [1]

NoSQL databáze je vhodné využít na specifické potřeby datového úložiště, kde budou jejich výhody zúročeny. Nejsou určeny jako náhrada SQL databází. V praxi je totiž žádoucí, aby byl softwarový vývojář schopen rozlišit, kdy je vhodné použít SQL a kdy naopak NoSQL databázi, včetně jejího výběru.

2.1 NoSQL vs. SQL

V SQL databázích mají všechna data svou vlastní strukturu. Konvenční databáze, jako je Microsoft SQL Server, MySQL či Oracle Database, používají schéma, tedy formální definici toho, jak budou data vložená do databáze sestavována. [2]

Například daný sloupec v tabulce může být omezen pouze na celá čísla. Výsledkem je, že údaje zaznamenané ve sloupci budou mít vysoký stupeň normalizace. Rigidní schéma databáze SQL také umožňuje relativně snadnou agregaci dat, například pomocí operace `JOIN`. [2]

Naopak u NoSQL datových úložišť mohou být data uložena bez schématu. V podstatě jakákoli data mohou být uložena v libovolném záznamu, protože ačkoliv lze v některých NoSQL databázích schéma definovat, nelze je typově omezit. Data se totiž z výkonnostních důvodů ukládají po bajtech aniž by databáze věděla, jak uloženou informaci interpretovat, neboť tuto funkci obstarává aplikační vrstva. Obecně se dá říct, že jsou NoSQL databáze navrženy tak, aby zvládaly zátěž většího množství dat.

2.2 Dělení NoSQL databází

Pakliže chceme veškeré NoSQL databáze rozdělit do kategorií, čeká nás spousta práce, neboť tato množina zahrnuje všechny databáze, které nejsou omezeny jen na SQL. Neustále vznikají nové druhy NoSQL databází protože požadavky na databáze jsou stále přísnější a konkrétnější.

Z toho důvodu si zde uvedeme jen čtyři nejpoužívanější druhy NoSQL databází a datových úložišť [2]:

- **Úložiště dokumentů** (např. `CouchDB`, `MongoDB`). Data se ukládají ve formátu JSON. Tento druh NoSQL se tedy hodí pro zanořené informace a často se používá ve spojení s `frontend` aplikacemi.
- **Key-Value úložiště** (např. `Redis`, `Riak`). Jedná se o úložiště navržené pro uchování hodnot patřících zvolenému klíči. Účelem je efektivnější vyhledávání hodnot než u SQL. Vyšší rychlost odpovědi je zajištěna zejména skutečností, že hodnoty nikdy nejsou zatíženy vzájemnou relací jako u SQL, a že úložiště má možnost si velmi častá data uchovávat v pracovní paměti RAM.
- **Wide-Column úložiště** (např. `HBase`, `Cassandra`, `BigTable`). Data jsou zde uložena ve sloupcích namísto řádků, jako tomu je u konvenčních SQL systémů. Jakýkoli počet sloupců (a tedy mnoho různých typů dat) lze snadno seskupit anebo agregovat podle potřeby. Další výhodou je, že jsou tato úložiště dobře škálovatelná.
- **Databáze grafů** (např. `Neo4j`). Data jsou reprezentována jako síť nebo graf entit a jejich vztahů, přičemž každý uzel v grafu obsahuje volnou formu dat.

V této práci se dále budu zabývat pouze databází `Apache HBase`, pro níž byl zadán úkol optimalizovat čtení dat s ohledem na jejich časovou granularitu. Podle výše definovaného dělení NoSQL databází patří do skupiny Wide-Column úložišť.

3 Apache HBase

Ve zkratce se jedná o verzovanou distribuovanou open-source databázi. [3]

Skutečnost, že je databáze distribuovaná, znamená, že jsou její data rozložena na několik instancí, čímž se logicky zrychlují vstupně-výstupní operace (oproti databázi s jednou instancí), neboť při zápisu či čtení dat se nemusíme omezovat na zdroje pouze jedné instance, ale paralelně můžeme využít součet zdrojů všech instancí, díky čemuž zároveň dochází k rozložení zátěže. Tato architektura také umožňuje snadnou škálovatelnost.

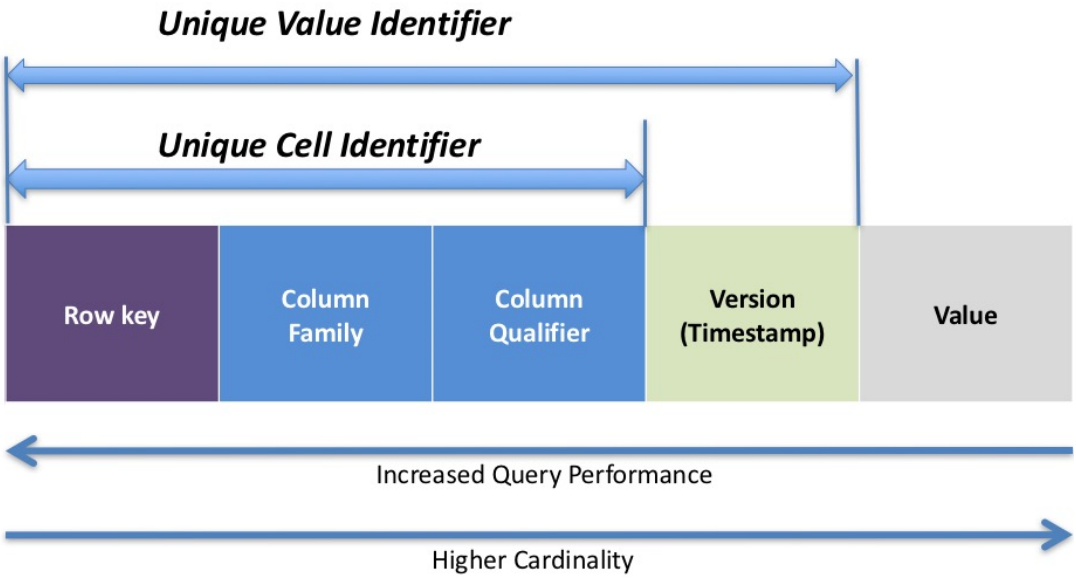
Pilířem `Apache HBase` je HDFS [3], neboli Hadoop Distributed File System. Jak název napovídá, jedná se distribuovaný souborový systém pro `Apache Hadoop` (dále jen `Hadoop`). To je projekt zaměřený na vývoj open-source softwaru pro spolehlivou, škálovatelnou a distribuovanou výpočetní techniku [4]. `Apache HBase` (dále jen `HBase`) tedy vyžaduje mít správně nainstalované a nakonfigurované `Hadoop` prostředí.

3.1 Datový model `HBase`

Abychom pochopili, jak `HBase` ukládá data, potřebujeme definovat základní pojmy. Některé z nich se však vyskytují i v obecně známých relačních databázích, což způsobuje terminologický překryv, který však nepředstavuje vhodnou analogii. Překryv nastává u pojmů *tabulka*, *sloupec* a *řádek*. U `HBase` je vhodné o tabulce smýšlet spíše jako o vícedimenzionální mapě, než jak jsme zvyklí z relačních databází. Při celkovém pohledu na model `HBase` snad bude zřejmé proč. [5]

Celý datový model je složen ze jmenných prostorů, tabulek, řádků, sloupců, rodin sloupců, kvalifikátorů sloupců, buněk a časových známek (z angl. Namespace, Table, Row, Column, Column Family, Column Qualifier, Cell a Timestamp). V tomto pořadí jsou prvky modelu seřazeny od nejvyšší úrovně abstrakce po nejnižší. [5]

Obecně platí, že při dotazování se na data získáváme rychlejší odezvu pokud se dotazujeme na prvky z vyšší úrovně. Naopak vyšší kardinalita dotazu (dotazování se na prvky z nižší úrovně abstrakce) způsobí zpomalení odpovědi [6]. Viz obrázek 3.1.



Obr. 3.1: Nepřímá úměra mezi kardinalitou dotazu a rychlosti jeho zpracování [6]

3.1.1 `Namespace`

Jmenný prostor (z angl. `Namespace`) je logický shluk tabulek, analogický pojmu databáze v relačních databázích, jehož použití je volitelné. Veškeré tabulky, jejichž jmenný prostor není při vytváření definován, budou spadat pod jmenný prostor pod názvem `default`.

Kromě tohoto výchozího jmenného prostoru exituje jmenný prostor pod názvem `hbase`, který ve starších verzích databáze `HBase` obsahoval interní tabulky.

Jmenný prostor je zajímavý zejména z administrativního hlediska, jelikož poskytuje další vrstvu, kde lze řídit kvóty uživatelů, administraci zabezpečení a izolaci dat na konkrétních uzlech. [7]

3.1.2 `Table`

Tabulka (z angl. `Table`) je po jmenném prostoru druhou nejvyšší úrovní abstrakce dělení dat v `HBase`. Každá tabulka může obsahovat vícero řádků a je deklarována jako první při definici schématu. [8]

3.1.3 `Column`

Sloupec (z angl. `Column`) se skládá z rodiny sloupců a kvalifikátoru sloupce. Jeho zápis obsahuje dvojtečku, jež se chová jako oddělovač. Nalevo od ní je rodina sloupců a napravo kvalifikátor sloupce. [9]

`Column Family`

`Column Family` (volně přeloženo jako rodina sloupců) fyzicky sdružuje množinu sloupců a jejich hodnot, zejména z výkonnostních důvodů. Každá rodina sloupců má několikero nastavení úložiště. Například, zda mají být hodnoty uloženy také ve vyrovnávací paměti nebo jakým způsobem a zda vůbec se má provést komprese dat, či informace o enkódování všech obsažených klíčů řádků a další nastavení.

Každý řádek tabulky má stejné rodiny sloupců, nicméně jakýkoliv řádek nemusí mít data v každé z nich.

Fyzicky jsou veškerá data spadající pod jednu rodinu sloupců uložena na jednom uzlu celého `HBase` clusteru[1]. [9]

`Column Qualifier`

Kvalifikátor sloupců byl přidán k rodině sloupců, aby se zavedl index konkrétní skupiny dat v rámci konkrétní rodiny sloupců. Ačkoliv je rodina sloupců pevně definována již při tvorbě tabulky, kvalifikátor sloupců je vytvářen dynamicky až za existence tabulky. [9]

3.1.4 Row

`HBase` řádek se skládá z klíče řádku a jednoho nebo více sloupců. Řádky se při ukládání řadí podle klíče vzestupně lexikograficky. Z tohoto důvodu je návrh klíče řádku velmi důležitý. Cílem je ukládat data tak, aby byly související řádky blízko sebe.

Klíčem řádku jsou nijak neinterpretované bajty. [11]

3.1.5 Cell

Buňka je kombinací řádku, rodiny sloupců a kvalifikátoru sloupců. Obsahuje hodnotu a časovou známku, která reprezentuje verzi uložené hodnoty.

Hodnota taktéž není nijak interpretována a je uložena po bajtech. [12]

3.1.6 Timestamp

Časová známka se zapisuje spolu s každou zapisovanou hodnotou a je identifikátorem pro konkrétní verzi hodnoty. [13]

[1] Cluster v oboru informačních technologií označuje skupinu počítačů, které spolupracují především za účelem paralelizace výpočetního výkonu. [10]

3.2 Operace datového modelu

Nad datovým modelem lze provádět pouze čtyři operace - Get, Put, Scan a Delete. [14]

3.2.1 Get

Tato operace vrátí atributy požadovaného řádku - tedy všechny rodiny sloupců, jejich kvalifikátory a verzované hodnoty [14]. Za její SQL protějšek lze považovat operaci `SELECT` s jednou návratovou hodnotou.

3.2.2 Put

Operace Put vytvoří nový řádek, pokud ještě neexistuje pod dodaným klíčem řádku, jinak jeho hodnotu aktualizuje [14]. SQL protějšek nemá. Nicméně stejného efektu by se dalo docílit SQL procedurou, která nejdříve zjistí, zda entita již existuje a na základě výsledku provede buď `UPDATE`, nebo `INSERT`.

3.2.3 Scan

Na základě specifikovaných atributů vrátí všechny řádky, které splňují veškeré požadované vlastnosti [14]. Z pohledu, co jde na vstup a co dostaneme na výstup, se za SQL ekvivalent dá považovat `SELECT` s několika spojenými tabulkami pomocí `JOIN` přes cizí klíče. Když však budeme brát v potaz vnitřní strukturu dat a proces, jakým se k datům přistupuje, pak `HBase` nelze zjednodušeně srovnávat s SQL databázemi.

Relační databáze byly navrženy pro libovolné množství vazeb mezi entitami, nýbrž `HBase` byla navržena pro obrovské množství dat bez složitých relací [14].

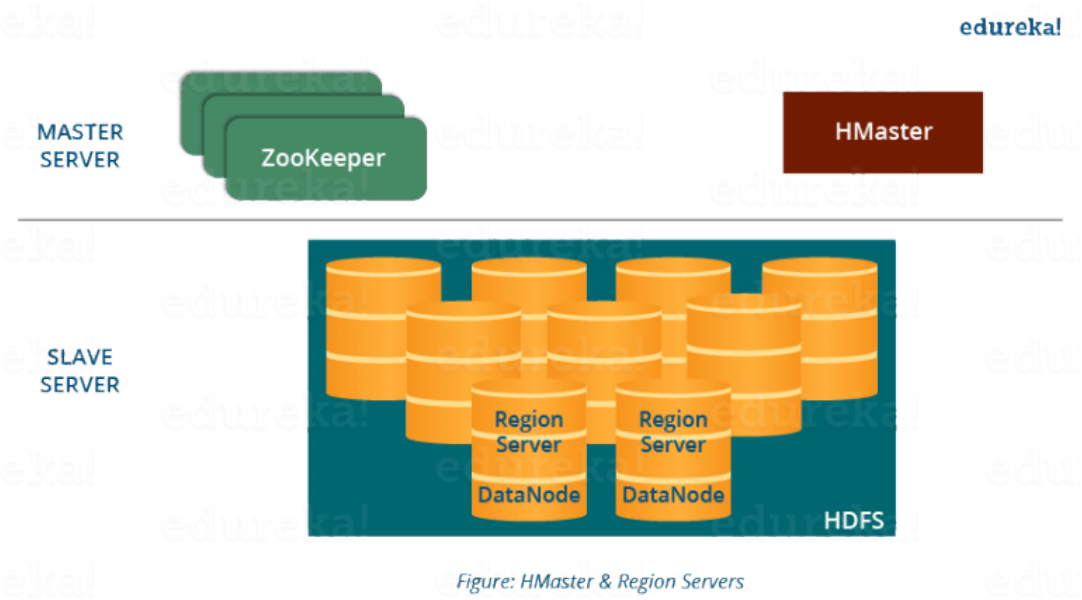
3.2.4 Delete

Touto operací lze požadované řádky připravit na smazání. To se provede tak, že se specifikovaným řádkům přiřadí tzv. *Tombstone markery* (volně přeloženo jako náhrobky).

Tyto řádky se fyzicky vymažou až při `Major Compaction` (volně přeloženo jako hlavní zhutnění), ke kterému se vrátím v podkapitole 3.3.8. [14]

3.3 Architektura HBase

HBase se skládá ze tří komponent v architektuře master-slave. Master roli zastupují Master Server a Zookeeper. Slave roli hraje RegionServer. [15]



Obr. 3.2: HBase komponenty v master-slave architektuře [17]

Důležitou součástí je i HDFS vrstva, která zaštiťuje distribuovaný a replikovaný souborový systém. Součástí této vrstvy jsou dvě základní komponenty, opět v master-slave architektuře. Jedná se o NameNode (master) a DataNode (slave).

3.3.1 NameNode

NameNode je centrální server HDFS, který udržuje informace o metadatech pro všechny fyzické datové bloky, které obsahují soubory. Zároveň řídí přístup k těmto souborům a kromě toho zařizuje replikaci datových bloků. [18]

3.3.2 DataNode

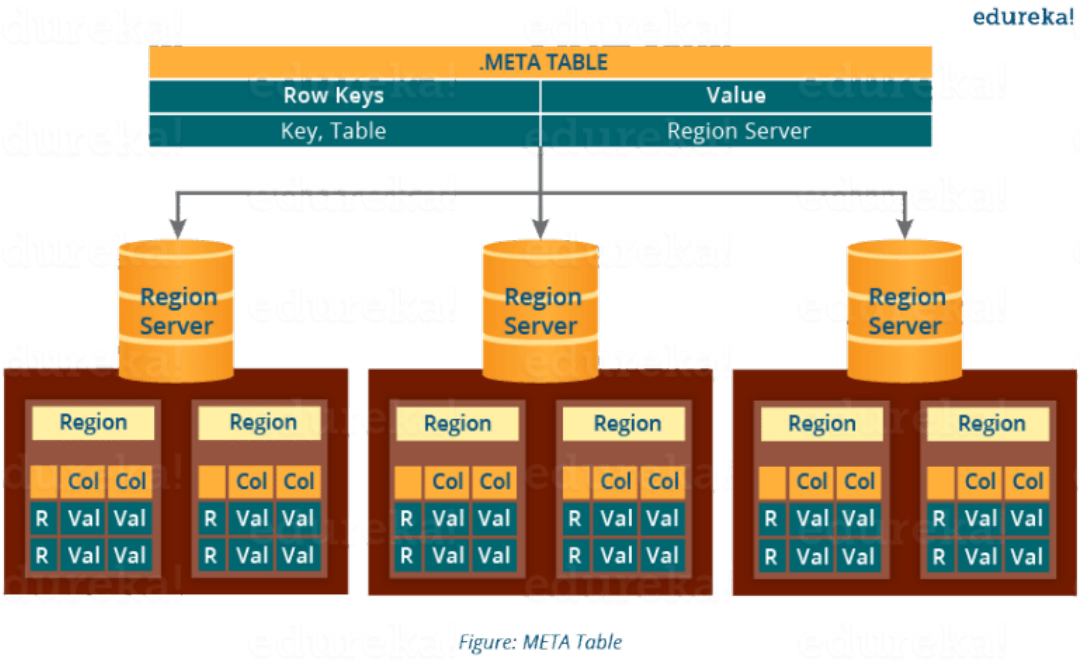
DataNode zodpovídá za uložení dat v distribuovaném úložišti HDFS [18]. V souvislosti s HBase ji poskytuje rozhraní pro uložení jejích dat.

3.3.3 Region a RegionServer

`HBase` tabulky se rozdělují dle klíče řádku do regionů. Jeden region vždy obsahuje všechny řádky tabulky v rozpětí od počátečního klíče až po koncový klíč regionu. Tyto hranice klíčů se mohou nastavit při vytváření tabulky. Pokud se tak nestane, pak si `HBase` sama postupně s přibývajícími daty posouvá i tyto hranice tak, aby byly regiony podobně velké. [19]

Jednotlivé regiony jsou přiřazeny konkrétním uzlům v clusteru, viz obrázek 3.3. Tyto uzly se nazývají `RegionServery`[2] a starají se o čtení a zápis dat v regionech. Jeden `RegionServer` může obsahovat až tisíce regionů. [15]

`RegionServer` je zároveň komponentou, která přímo komunikuje s klienty za účelem přístupu k uloženým datům. [15] [19] [20]



Obr. 3.3: `HBase` regiony na `RegionServerech` [17]

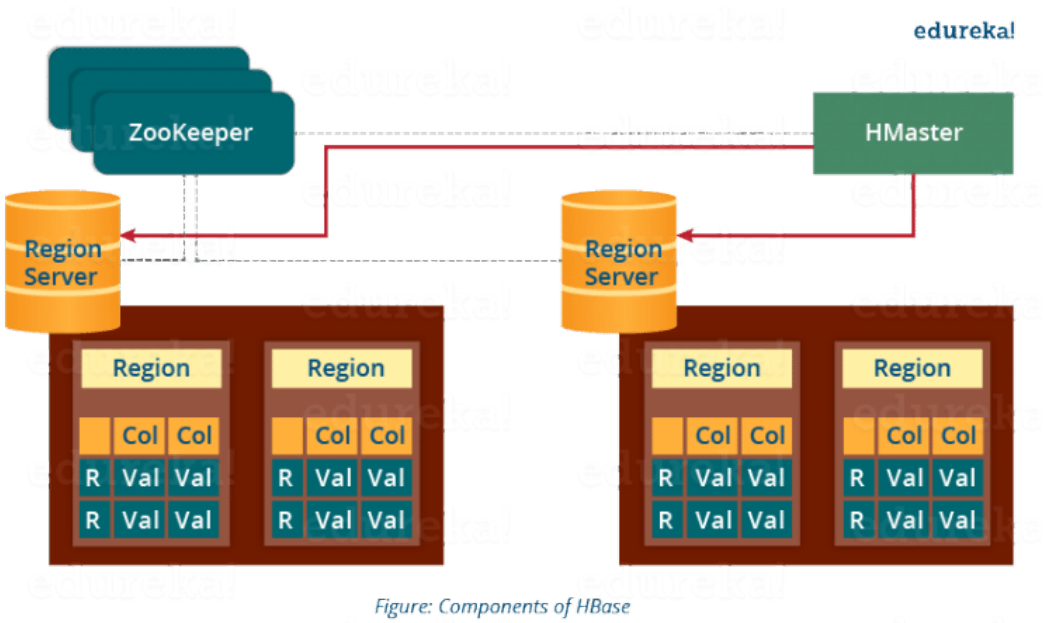
[2] Rozmístění regionů je definováno v tzv. META tabulce, která na místě klíče obsahuje název tabulky, začátek klíče, region a na místě hodnoty konkrétní `RegionServer`. Při čtení dat si pak klient nejdříve musí z META tabulky přečíst, kterého `RegionServeru` se má na data zeptat. [20]

3.3.4 Master Server

`HBase Master Server` je zodpovědný za přiřazení regionů v clusteru jednotlivým `RegionServerům` při startu `HBase` a poté i za přemísťování regionů mezi `RegionservERY` za účelem rozložení zátěže. [21]

Monitoruje také stav všech `RegionServerů` v clusteru a podniká nápravné akce v případě selhání některého `RegionServeru`. Dále má na starosti spouštění DDL[3] operací. [15]

Pak také dodává externí rozhraní. Jedno běží na http protokolu za účelem snadného sledování stavu celého `HBase` clusteru. Druhé rozhraní je typu RPC (Remote Procedure Call), které používá známý `Google Protocol Buffer`, což je protokol navržený skupinou Google Developers[4]. Účelem tohoto protokolu je jednak komunikace s `RegionServerem`, ale i možnost připojit klientskou aplikaci založenou na RPC. [22]



Obr. 3.4: `HBase Master Server` v roli `HBase` mastera [17]

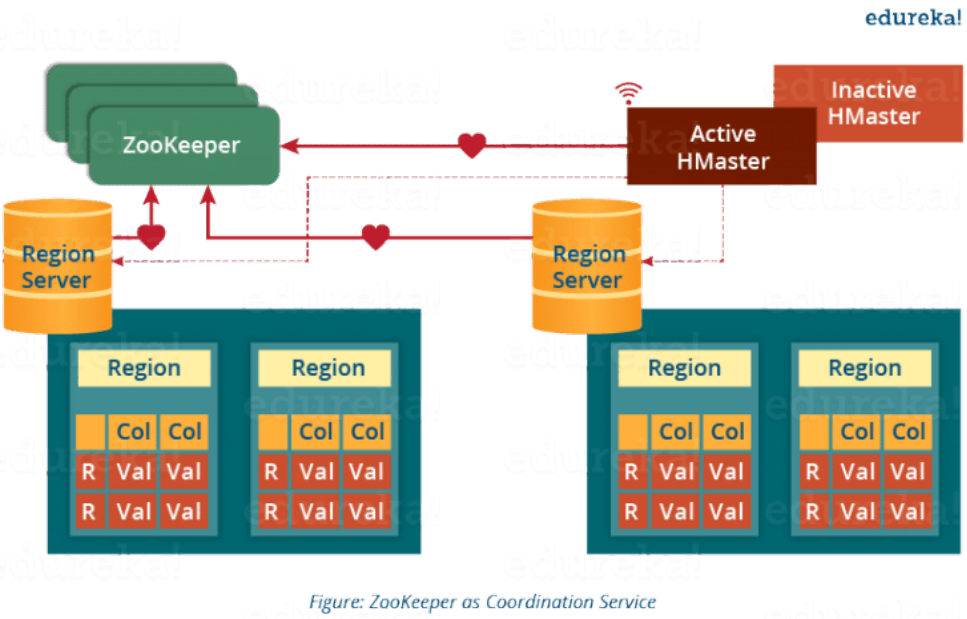
[3] *Data Definition Language* (DDL) se používá k definici struktury databáze, tedy vytvoření, či odstranění tabulek

[4] Pro zajímavost, architektura tohoto protokolu je velice podobná architektuře ROS zpráv určených primárně pro roboty [37] [38].

3.3.5 Zookeeper

`Zookeeper` je součástí `HDFS` a je používán jako distribuovaná koordinační služba k údržbě stavu celého datového úložiště. `Zookeeper` hlídá, které služby jsou dostupné a při výpadku notifikuje administrátory systému. [15] [17]

`Zookeeper` běží v několika instancích. Aby zajistil správnost vyhodnoceného stavu clusteru, používá konsenzus. Pro konkrétní stav se tedy musí shodnout nadpoloviční většina `Zookeeperů`. Z toho důvodu se Zookeper provozuje ve třech, či pěti instancích. [15]



Obr. 3.5: Role `Zookeepera` v `HBase` architektuře [17]

3.3.6 Jak komponenty spolupracují

Ústřední komponentou je `Zookeeper`. Všechny `RegionServery` a pouze aktivní `Master Server` s ním mají vytvořené aktivní sezení, které používají k aktivnímu pravidelnému ohlašování vlastní použitelnosti. Tento kontinuální proces se nazývá HeartBeat (volně přeloženo jako tlukot srdce). Komunikace je znázorněna na obrázku 3.5.

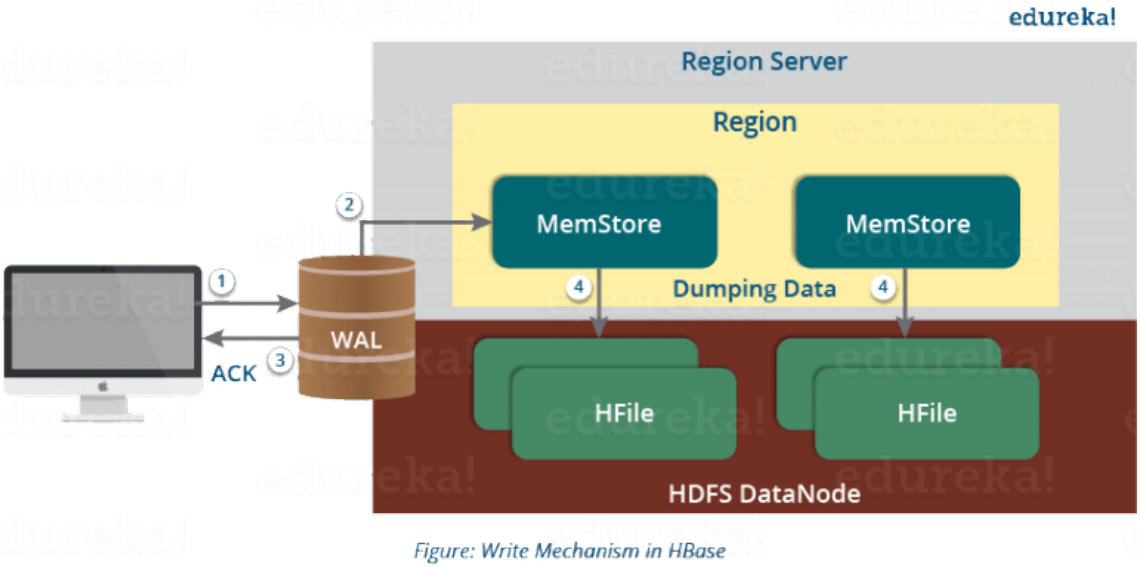
Při startu `HBase Zookeeper` prohlásí první `Master Server`, na který narazí, za aktivní. Neaktivní `Master Server` pak čeká na selhání aktivního, což mu pak je oznámeno `Zookeeperem` a tím je prohlášen za aktivního.

Aktivní `Master Server` také čeká na chybu `RegionServeru`, kterou zjistí tak, že `Zookeeper` za nakonfigurovaný časový limit od něj nepřijal heartbeat a je tak prohlášený za mrtvého. Díky výchozí replikaci `HDFS` úložiště je `Master Server` schopen

obnovit regiony patřící uhynulému `RegionServeru` a přiřadit je jiným. [15] [17]

3.3.7 Způsob uložení dat

Jak již bylo zmíněno, o čtení a zápis dat se stará `RegionServer`, který je zapisuje do `HDFS` datového úložiště. Samotné uložení však sestává z několika kroků.



Obr. 3.6: Mechanismus ukládání dat v `HBase` [17]

Write Ahead Log

Úplně první operaci, kterou `RegionServer` provede, je zápis na konec WAL (Write Ahead Log) souboru v distribuovaném souborovém systému. Používá se k obnově nejnověji zapsaných dat, které se nestihly perzistentně zapsat na permanentní úložiště tabulky před výpadkem `RegionServeru`. [23]

MemStore

Po zápisu do WAL se nová data zapíší do tzv. MemStore, což je vyrovnávací paměť, neboli cache, která sbírá zatím neuložená data za účelem prvotního seřazení před zápisem na disk. Ihned po zapsání nových dat do MemStore se vrátí klientovi odpověď o úspěšné operaci Put.

Pro každou rodinu sloupců existuje jeden MemStore. [24]

HFile

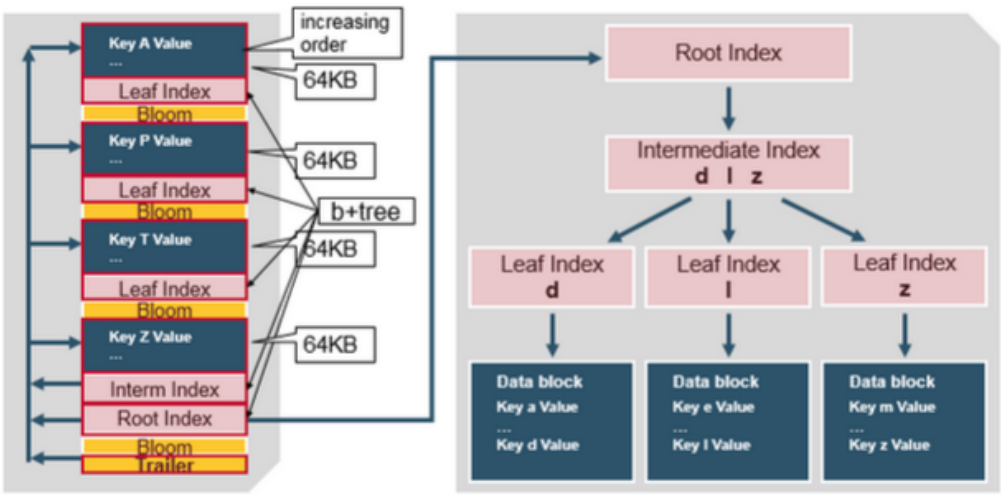
Když MemStore nakumuluje dostatečné množství dat, všechna v něm seřazená data se zapíší do nového `HFile` v `HDFS`. Tato automatická operace se nazývá Region Flush.

Každá rodina sloupců tak může obsahovat až několik `HFile`. Díky tomu, že je zápis `HFile` na disk sekvenční, probíhá čtení těchto dat velice rychle i na pevných discích, neboť se vyhýbají zbytečnému pohybu jejich zápisové hlavice.

`HFile` navíc obsahuje několikavrstvé indexy, díky čemuž `HBase` umožňuje vyhledat data bez nutnosti číst celý soubor.

Několikavrstvé indexy v `HFile` jsou podobné binárnímu stromu. Uložené hodnoty jsou totiž seřazeny podle klíče řádku.

Klíče řádku odkazují na svá data po blocích velkých 64 KB. Každý takový blok má zároveň svůj vlastní Leaf Index. Poslední klíč řádku každého bloku je pak uložen do jednoho Intermediate Indexu. Na ten pak odkazuje Root Index, který je přečten vždy na začátku `HFile` souboru, takže pevný disk pak provádí čtení vždy pouze v jednom směru, čímž se přirozeně zrychluje oproti tomu, kdyby se na disku skákalo z místa na místo. [15] [25] [26]



Obr. 3.7: Formát `HFile` souboru [15]

3.3.8 Způsob čtení dat

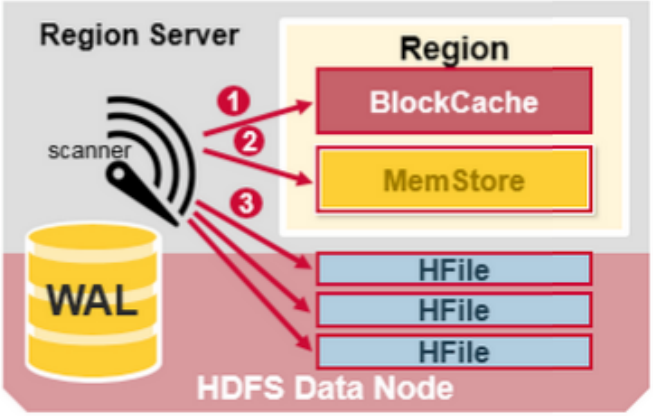
Čtení dat z `HBase` provádí `RegionServer`. Přitom používá vyrovnávací paměť pod názvem `BlockCache`, kam si ukládá naposledy přečtená data. Když se naplní, jsou z ní odstraněna data, jejichž poslední použití bylo nejdříve, tedy tzv. LRU (Least Recently Used) data. [27]

Z toho však plyne, že disponibilní data se mohou nacházet na více místech. Konkrétní buňka může být již v `HFile`, zároveň se však může novější verze nacházet v MemStore, a ještě k tomu se může nalézat mezi naposledy přečtenými buňkami v `BlockCache`. Když se tedy čte řádek, je potřeba rozhodnout, odkud se vrátí jeho hodnota. [15]

Read Merge

Zmíněný rozhodovací proces se nazývá Read Merge (volně přeloženo jako sjednocené čtení). Nejdříve čtecí skener `RegionServeru` hledá buňku řádku v `BlockCache`, tedy čtecí vyrovnávací paměti. Pokud ji zde nenajde, jde ji hledat do MemStore, tedy zápisové vyrovnávací paměti. Pakliže ji nenajde ani zde, začne řádek hledat v `HFile` souborech, které nejsou v operační paměti, ale na disku. [15] [17]

Vzhledem ke způsobu zápisu `HBase` dat se však může stát, že k jednomu MemStore náleží vícero `HFile` souborů. Tím pádem musí `HBase` průběžně otevírat všechny tyto soubory, dokud nenajde nejnovější hledaný řádek.



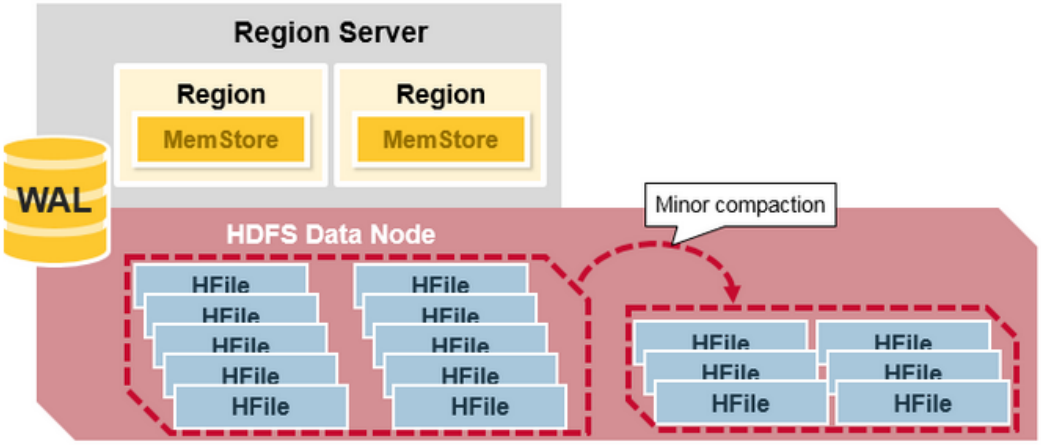
Obr. 3.8: Schéma Read Merge rozhodovacího procesu [15]

Compaction

Hledání konkrétního řádku ve více `HFile` souborech, jejichž data nejsou navzájem seřazena, negativně ovlivňuje rychlost hledání řádku a nadměrně přitom zatěžuje vstupně-výstupní operace `RegionServeru`.

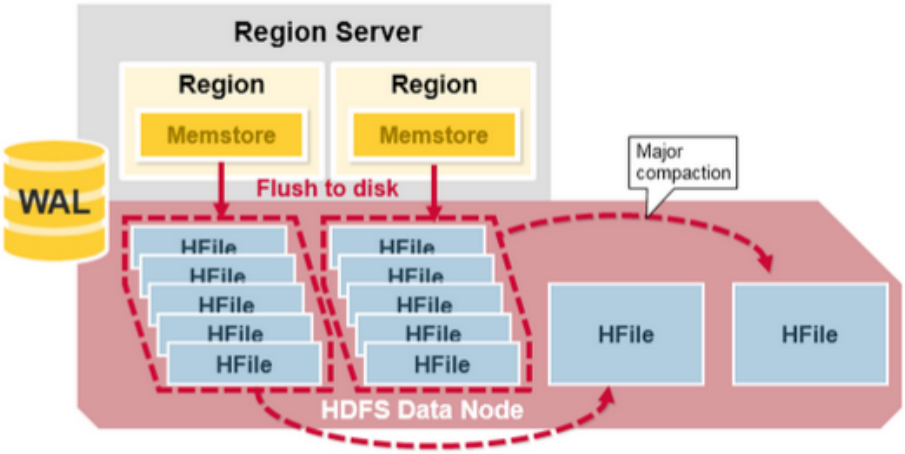
Proto `HBase` automaticky spustí `Minor Compaction` (volně přeloženo jako drobné zhutnění) ihned poté, co se vytvoří předem nakonfigurovaný počet `HFile` souborů.

Tento proces si klade za cíl sjednotit velké množství malých `HFile` souborů do menšího množství větších `HFile` souborů proto, aby se zredukoval počet přeskoků čtecí hlavy pevného disku při čtení `HFile` souborů, ale i proto, aby se veškeré řádky sjednocovaných `HFile` souborů oproti sobě seřadily a bylo tak snazší vyhledat ty potřebné. Proces je znázorněný na obrázku 3.9. [15] [28]



Obr. 3.9: Proces `Minor Compaction` pro redukci množství `HFile` souborů [15]

Jednou za nakonfigurovanou dobu se pak spouští `Major Compaction` (volně přeloženo jako hlavní zhutnění). Tím se zajišťuje vytvoření jediného `HFile` pro každou rodinu sloupců. Přitom se rovnou zahazují expirované buňky, včetně těch určených k vymazání. Tento proces znázorňuje obrázek 3.10. [28]



Obr. 3.10: Proces `Major Compaction` pro redukci množství `HFile` souborů [15]

Procesem `Major Compaction` se zajistí výrazné zrychlení čtecího procesu. Bohužel se však u tohoto procesu přepisují všechny `HBase` soubory, což zahlcuje vstupně výstupní operace disků. Také se může stát, že se redukcí obsazeného místa regionem `Master Server` rozhodne ho přesunout na jiný `RegionServer`, což zase může vytížit síť. Z důvodu vytížení `HBase` během `Major Compaction` se typicky tento proces pouští o víkendech či večerech. [15]

Filtrace čtených dat

Při operaci Scan, či Get lze zadat tzv. `HBase` filtr, který omezí výsledky. Jak jsou výsledky omezeny, záleží na druhu použitého filtru. Pro filtraci na základě identifikátoru řádku se používá `RowFilter`. Pro filtraci na základě hodnoty sloupce je k dispozici skupina filtrů pod názvem `Column Value Comparators`, do níž patří například `BinaryComparator`, který porovnává dodanou hodnotu binárně s hodnotou řádku, nebo `SubstringComparator`, který porovnává dodanou hodnotu filtru jako řetězec s hodnotou řádku, ke které se chová také jako k řetězci namísto jako k poli bajtů.

Hlavním filtrem je strukturální `FilterList`, který umožňuje spojovat více `HBase` filtrů. Lze zadat, aby dva spojené filtry uspěly oba, nebo alespoň jeden z nich. Sám `FilterList` může být jedním ze spojených filtrů, takže se takto mohou tvořit složitější logické operace napříč různými filtry. [16]

Při použití nativního `HBase` API je možné filtry, jež jsou součástí `HBase`, rozšířit a naimplementovat si tak vlastní logiku filtrace čtených dat.

3.4 Aplikační rozhraní `HBase`

Přístup k datům v `HBase` je možno realizovat buďto využitím nativního Java API (Application Programming Interface), nebo externího API, které vytváří další vrstvu abstrakce za účelem propojení s jinými programovacími jazyky.

3.4.1 Native Java API

Toto API poskytuje knihovny, jež `HBase` používá interně. Výhodou jeho použití je skutečnost, že mezi `HBase` a implementovanou aplikací není žádná další vrstva, která by mohla představovat zbytečnou režii. [29]

Velkou nevýhodou nativního Java API je nutnost při potenciálním povýšení verze `HBase` přepsat kód z předchozí verze tak, aby byl funkční v nové verzi. Aktualizace stabilních komponent implementujících čtení dat za použití nativního Java API tak může být drahá záležitost v závislosti na složitosti problému, který řeší.

Nicméně pokud je rychlost čtení dat nejvyšší prioritou, pak je použití této varianty implementace na místě.

Momentální řešení pro čtení statistických dat v reklamním systému využívá právě toto API.

3.4.2 External API

Pod Externí API se řadí veškerá aplikační rozhraní, která nejsou interní. Slouží především k možnosti napojení `HBase` na aplikace napsané v jiném programovacím jazyce. Kromě toho však také poskytují stabilnější rozhraní oproti nativnímu Java API. [30]

REST

Representational State Transfer, tedy REST, byl představen v roce 2000 v doktorské disertaci Roye Fieldinga, jednoho z hlavních autorů specifikace HTTP. Obecně umožňuje interakci klient-server prostřednictvím API rozhraní, které je svázáno s určitou URL adresou. [31]

Naimplementovaný REST server je již součástí `HBase`. Pro jeho použití stačí po instalaci `HBase` tento server vhodně nakonfigurovat a zapnout. Na rozhraní vystavuje tabulky, řádky, buňky a metadata `HBase`. [31]

Apache Thrift

Softwarový framework `Apache Thrift` (dále jen `Thrift`) je určen pro vývoj škálovatelných služeb ve všech mainstreamových programovacích jazycích. Kombinuje

soubor softwarových knihoven s generátorem zdrojového kódu pro vytváření služeb, které efektivně a bezproblémově fungují napříč jazyky C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml, Delphi a dalšími. [32]

`Thrift` umožňuje definovat typy dat a rozhraní v jednoduchém definičním souboru. Tento soubor pak slouží jako vstup do kompilátoru, jenž vygeneruje kód, který lze použít pro snadné vytváření RPC klientů a serverů, které plynule komunikují napříč programovacími jazyky. [32]

Ušetří se tak mnoho času. Namísto programování serializace a transportu objektů na obou stranách komunikace se programátoři mohou soustředit na to, co je podstatné.

Z hlediska `HBase` je `Thrift` zajímavý zejména díky podpoře tzv. `Filter Language`. Ten dovoluje provádět filtraci dat na straně serveru na základě požadavku klienta. Tím se docílí toho, že jsou po síti přenášeny pouze ta data, která klient opravdu potřebuje. Výsledkem je tedy ušetření nejen síťového provozu, ale i výpočetního výkonu na straně klientské aplikace. [33]

Scala

Scala je jak objektově orientovaný, tak funkcionální staticky typovaný programovací jazyk [34]. Jeho výhodou je, že také běží na JVM (Java Virtual Machine), tedy Java virtuálním stroji, stejně jako zkompilovaný zdrojový kód Javy [35]. Sdílení virtuálního stroje umožňuje Scale používat nativní Java `HBase` API, ačkoliv Scala samotná na Javě založena není.

3.4.3 Apache HBase Coprocessors

Velkou nevýhodou externího API je nutnost před aplikačním zpracováním získávaných informací data poslat po síti do klientské aplikace, kde se teprve zpracují. Tento problém řeší právě koprocesory, což jsou aplikace používající nativní `HBase` API, které běží přímo na `HBase` serveru. Jejich úkolem je podle naprogramovaných pravidel shlukovat data do nižších úrovní granularity přímo na serveru.

Existují celkem dva druhy `HBase` koprocesorů - `Observer` a `Endpoint` [36].

Observer Coprocessor

Jeho použití lze srovnat s použitím `TRIGGER` u SQL databází. Funguje tak, že při každé operaci Get nebo Put nad ní vykonává naprogramovanou logiku [36].

Z hlediska řešení optimalizace čtení dat by mohl posloužit jako jednoduchý agregátor vysoce granulovaných dat.

Endpoint Coprocessor

Tento koprocesor se typicky používá pro jednoduché statistické výpočty na straně serveru, například sumu či průměr konkrétní dimenze dat. Ve srovnání s Observerem, který má transparentní životní cyklus, tento koprocesor poběží pouze tehdy, když jej klient explicitně zavolá. [36]

Pro komunikaci používá již zmiňovaný `Google Protocol Buffers`.

4 Zápis statistických dat

V této kapitole se budu věnovat způsobu, jakým v reklamním systému zapisujeme statistická data, která generují návštěvníci reklamních prostor. Denně zaevidujeme desítky milionů událostí, které mohou být zajímavé pro zpětnou vazbu inzerenta za účelem neustálého vylepšování svých marketingových postupů při nakupování reklamy.

Kromě toho jsou však tato data návštěvníků důležitá pro reklamní systém i z důvodu fakturace inzerentů. Podle uložených statistik jsme schopni z virtuálního účtu inzerenta strhnout přesnou sumu peněz a vystavit mu za to fakturu.

Jistě si lze snadno představit, že množství dat tekoucí dovnitř reklamního systému je enormní. Nyní tedy vysvětlím základní terminologii a přiblížím postup, jakým způsobem příchozí data evidujeme tak, abychom neměli zahlcený systém a zároveň byli schopni na požádání inzerenta vrátit statistiky jeho entit s odezvou v řádu maximálně jednotek sekund, ideálně však ve zlomku vteřiny.

4.1 Granularita dat

Granularita je v oboru informačních technologií míra detailu uložených informací [39] [40]. Vyšší úroveň granularity dat označuje skutečnost, že jsou data uložena ve vysokém detailu, který je například pro účely zobrazování statistik prakticky nepoužitelný.

Z toho důvodu se vysoce granulovaná data agregují do nižších úrovní, odkud lze uživateli zobrazit přínosnější informace.

Obecně se má za to, že je nevhodné ukládat data na úrovni maximální dosažitelné granularity. Je to z několika důvodů. Jeden z nich je, že data uložená tímto způsobem zabírají obrovské množství úložné kapacity datového úložiště, ale také, že je pak nutné vynaložit velké množství výpočetní kapacity pokaždé, když uživatel systému vyžaduje zobrazit souhrn informací založený na takto granulovaných datech.

4.2 Agregace statistických dat

Základní statistickou jednotkou reklamního systému je kliknutí na reklamu, tedy tzv. *klik*. Nastane, když návštěvník internetových stránek, na kterých se zobrazuje reklama, na ni klikne. V ten moment se zapíše do logu informace o návštěvníkovi a jeho kliknutí na odkaz.

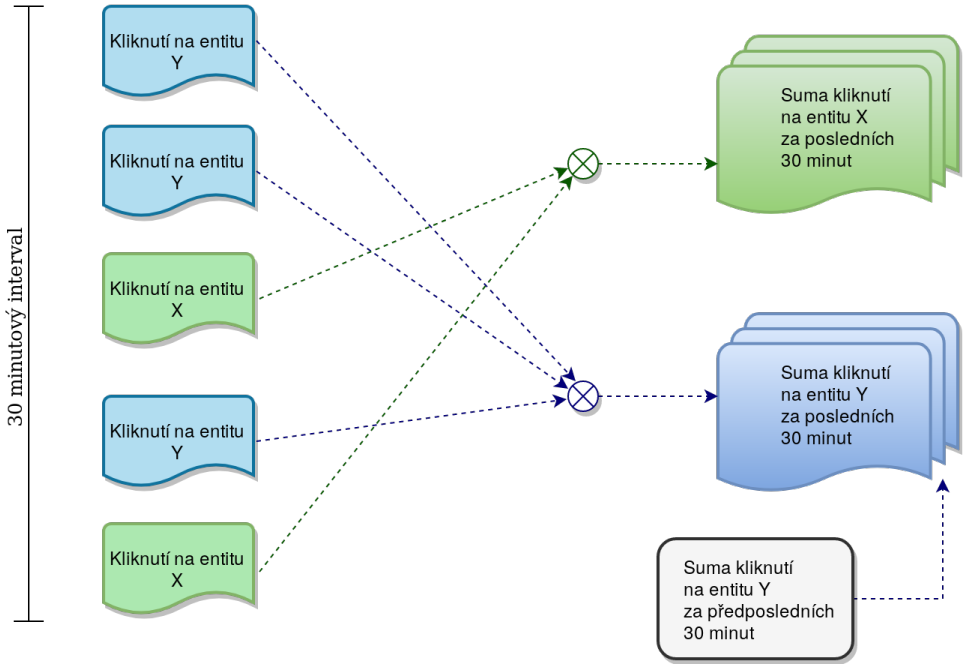
Tato informace je však vysoce granulovaná a seznam těchto informací by pro inzerenta neměl vypovídající hodnotu, aniž by si je sám nějakým způsobem zpracoval.

Z toho důvodu se vstupní statistická data o sledovaných entitách agregují do nižší úrovně granularity za použití tzv. inkrementálního agregátoru[1].

4.2.1 Inkrementální agregátory

Inkrementální agregátory slouží ke zpracování nejvíce granulovaných dat a pouštějí se automaticky každou půlhodinu.

Například agregátor všech kliknutí jednotlivých návštěvníků se nazývá *clicklinker*. Ten prochází logy všech kliknutí za posledních 30 minut a informace o kliknutí agreguje do `HBase` tabulky patřící entitě, které se kliknutí týká. Tento proces tedy prakticky zajišťuje snižování granularity dat[2].



Obr. 4.1: Schéma inkrementálního `clicklinker` agregátoru

Sledovaných akcí spjatých s konkrétními entitami je však mnohem více, než jen zmíněné kliknutí na reklamu návštěvníkem. Tomu také odpovídá velké množství specifických inkrementálních agregátorů, které se průběžně starají o to, aby byla granularita jednotlivých druhů dat redukována.

Každý inkrementální agregátor zapisuje data v tabulce entity do rodiny sloupců, která je určená výhradně inkrementálním statistikám. Do stejné tabulky zapisují

[1] Ve statistické části reklamního systému Sklik.cz se agregace dat provádí téměř výhradně v časové dimenzi. Nedochází tedy k redukci granularity u ostatních datových dimenzí a proto se často pod pojmem granularita myslí časová granularita.

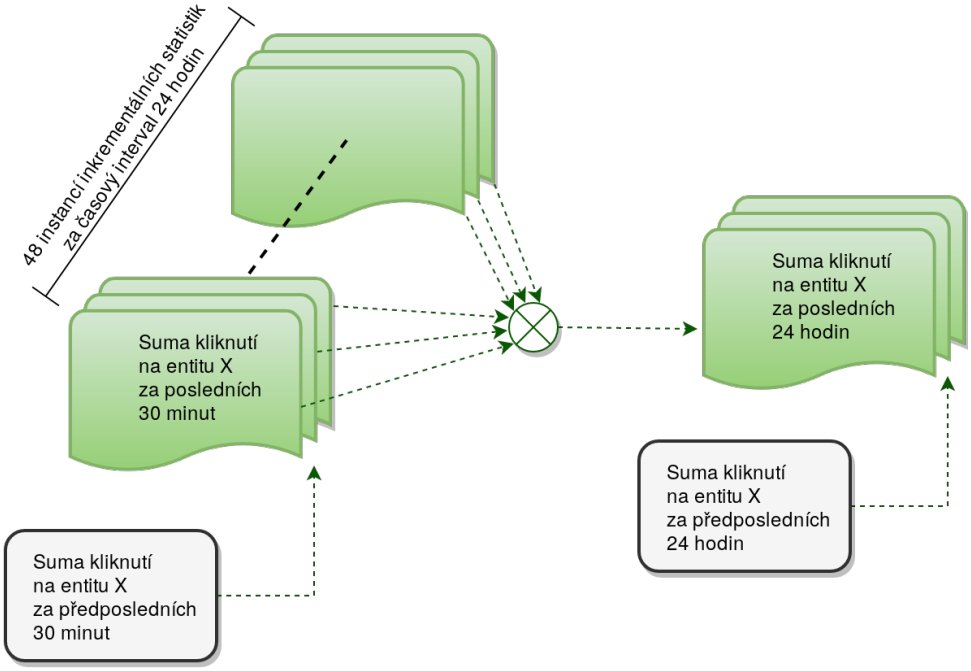
[2] Snížení granularity je docíleno tím, že se neuloží každé kliknutí zvlášť, ale pouze množství kliknutí za určité období.

i ostatní agregátory založené na redukci časové granularity. Každý z nich však má v tabulce svou vlastní rodinu sloupců, do které zapisuje[3].

Všechny rodiny sloupců, do kterých inkrementální agregátory zapisují, mají nastavenou délku života svých buněk na 36 hodin. To znamená, že po této době expiruje jejich platnost a při `Major Compaction` se pak vymažou. Do té doby se však inkrementálně agregovaná data zužitkují s pomocí denních agregátorů.

4.2.2 Denní agregátory

Účelem denních agregátorů je jednou denně zapisovat data z rodiny sloupců inkrementálních agregátorů za 24 hodin předchozího dne do stejné `HBase` tabulky hromadných náhledů pod jinou rodinu sloupců. Tato cílová rodina sloupců je určena výhradně pro statistiky s jednodenní časovou granularitou. Denní statistiky se již mohou zobrazovat inzerentům a proto má rodina sloupců pro denní agregátory neomezenou délku života svých buněk.



Obr. 4.2: Schéma denního agregátoru

Časový interval jednoho dne je nejvyšší úroveň časové granularity, jakou je reklamní systém Sklik.cz schopen nabídnout svým inzerentům. Naproti tomu napří-

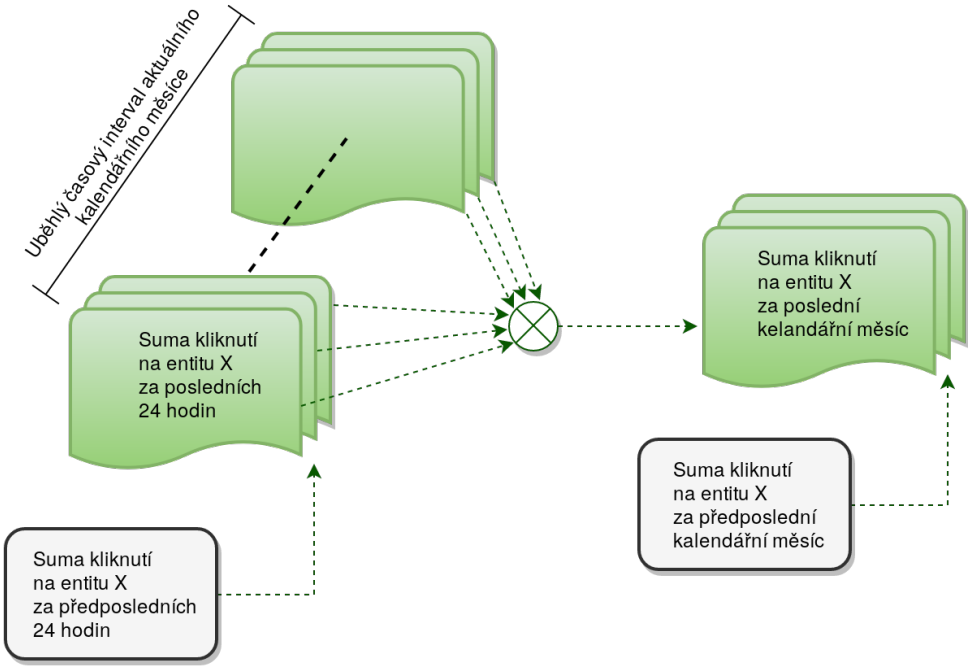
[3] `HBase` tabulce určené k zápisu agregátorů se přezdívá tabulka hromadných náhledů. To bohužel vytváří terminologický překryv s tabulkou hromadných náhledů na webovém rozhraní, proto význam tohoto slovního spojení vždy záleží na kontextu tématu.

klad konkurence Google Ads umožňuje svým inzerentům zobrazit statistiky až s hodinovou granularitou[4].

4.2.3 Měsíční agregátory

Měsíční agregátory fungují na velmi podobném principu jako denní. Jednoduše sumují denní statistiky entit do rodiny sloupců pro měsíční časovou granularitu s neomezenou délkou života svých buněk.

Sice sumují statistiky za aktuální kalendářní měsíc, ale ani přesto nejsou spouštěny jednou měsíčně, nýbrž každý den. Důvodem je nutnost zvládnout zobrazit sumované statistiky i za aktuální kalendářní měsíc s měsíční časovou granularitou. Kdybychom například měsíční agregaci spouštěli jednou za měsíc, tak bychom museli sumární statistiky počítat z denních statistik při každém požadavku zobrazit statistiky posledního měsíce v měsíční granularitě. Systémovější řešení je tedy spouštět měsíční agregaci každý den ihned po dokončení denní agregace, čímž vždy budou připraveny sumární statistiky za aktuální kalendářní měsíc.



Obr. 4.3: Schéma měsíčního agregátoru

[4] Na inzerentském portálu Google Ads při pokusu o zobrazení statistik za celou dobu existence účtu (asi 9 let) s hodinovou granularitou jsem se bohužel nedočkal a po chvíli čekání jsem dostal chybu. Pravděpodobně distribuované databázi, kterou Google Ads používá pro uchovávání statistik, trvalo příliš dlouho nasbírat tak enormní kvantum dat a někde po cestě od front-end části až k databázi nastalo vypršení časového limitu spojení.

4.3 Pravidelné spouštění agregací

Zpravidla se k plánovanému pouštění úloh používá unixový `Crontab`, do kterého se definuje, ve které časy se má spustit příslušná úloha. Nicméně `Crontab` má své nedostatky. Například nedisponuje webovým rozhraním, kde by autorizovaná osoba měla možnost živě sledovat standardní výstupy aktuálně probíhajících úloh. Nebo by měla možnost spustit jednotlivé úlohy mimo naplánovanou dobu s vlastními parametry, či si zobrazit jednoduchý přehled několika posledních spuštění, včetně inspekce logů proběhlých úloh. A to vše aniž by musela mít vzdálený přístup k příkazové řádce serveru, což by `Crontab` vyžadoval.

Všechny tyto potenciální úkony však umožňuje vykonávat software vyvinutý společností LinkedIn pod názvem `Azkaban Scheduler`. Právě tento software se v reklamním systému používá k zajištění a monitorování pravidelných běhů výše zmiňovaných agregátorů.

5 Čtení statistických dat

Teď, když je zřejmé, jak funguje zápis statistických dat, se lze přesunout na popis opačné operace, tedy čtení těchto zapsaných dat.

V této kapitole bude přiblížena komunikace mezi komponentami na cestě od klienta reklamního systému až po samotné datové úložiště. Budou také popsány komunikační protokoly používané na jednotlivých vrstvách. Kromě toho se také přiblíží problematika optimalizace čtení dat, která probíhá na více vrstvách. Nakonec bude zdůvodněna volba konkrétního aplikačního rozhraní na straně `HBase`.

Většina informací sice bude popisovat stav reklamního systému před implementací diplomové práce, ale na závěr bude navrženo řešení, které umožňuje filtraci všemi dostupnými datovými rozměry a zároveň nikterak neplýtvá dostupnými systémovými prostředky, nýbrž je efektivně využívá.

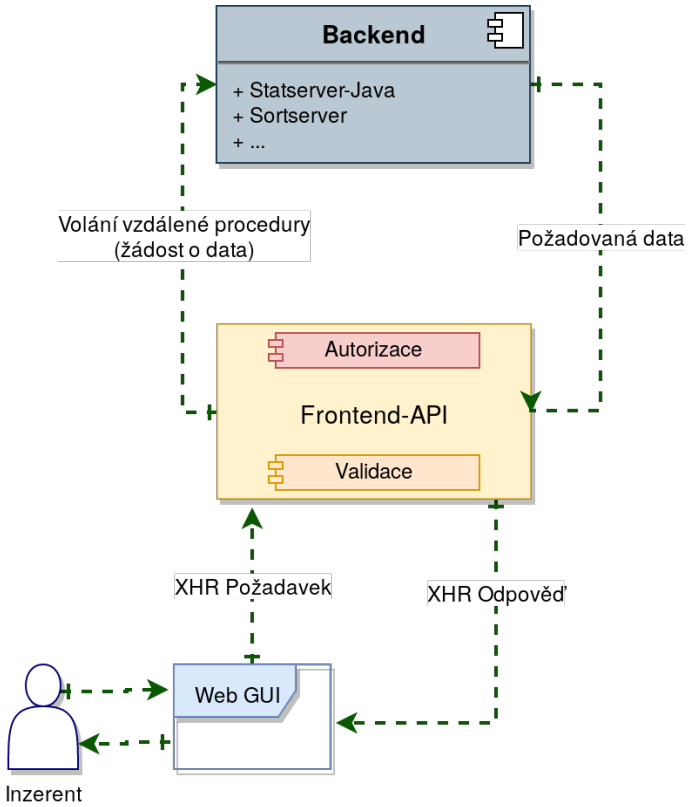
5.1 Architektura komponent

Komponenty reklamního systému se v zásadě dělí do dvou skupin, kterými jsou `frontend` a `backend`. Tyto skupiny mají své specializované vývojáře, kteří se zaměřují na vývoj komponent patřících do jejich skupiny. V zásadě platí, že `frontend` je ta část systému, se kterou přímo interaguje uživatel systému. Naopak `backend` se typicky zabývá databázemi, implementací rozhraní pro předem domluvené neveřejné služby, zpracováním dat, údržbou `backendových` komponent a v podstatě všemi programatickými úkony, se kterými uživatelé systému nepřicházejí do styku, ani jejich existenci nijak přímo nevnímají. Zjednodušeně se dá říct, že `backend` poskytuje služby `frontendu` a `frontend` zase uživatelům.

Autor této diplomové práce patří do skupiny `backendu` a proto se v této práci ani nebude zabývat implementací, která je potřeba udělat na straně `frontendu`.

5.1.1 Frontend-API

Celý proces získávání dat z reklamního systému začíná u inzerenta, který si zobrazí webové rozhraní pro inzerenty. Tím jeho prohlížeč na základě statických definic webové stránky pošle XHR požadavek v JSON formátu na veřejně dostupné rozhraní `frontendové` komponenty zvané `frontend-api`. Viz obrázek 5.1.

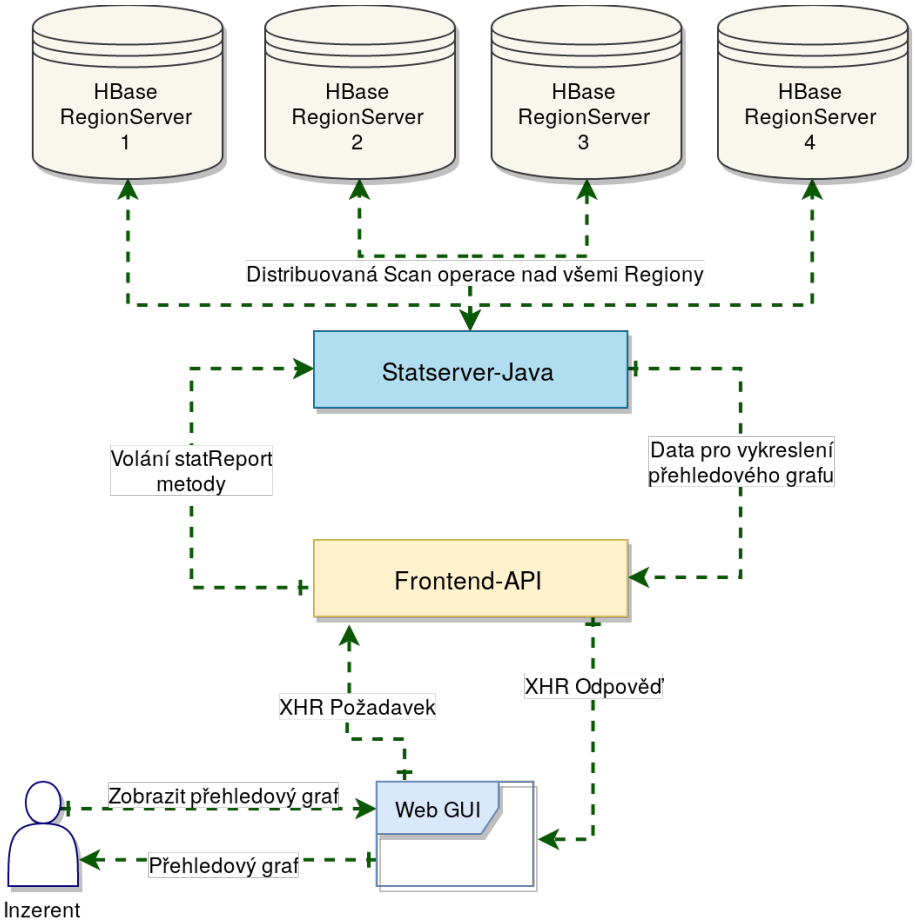


Obr. 5.1: Schéma komunikace s `Frontend-API`

Tato komponenta příchozí požadavky validuje, autorizuje a přeposílá dále na `backend`. Z jejího pohledu se na `backend` posílá dotaz typu '*Chci data do grafu, či tabulky za toto období pro entity vyhovující tomuto filtru*' a očekává odpověď typu '*Za toto období tento uživatel měl ve filtrovaných entitách tolik kliků, zobrazení, ...*' . Odpověď pak znova transformuje do JSON odpovědi, kterou předá uživatelově prohlížeči, který ji dle statických pravidel (CSS, JS, XHTML) vykreslí uživateli.

5.1.2 Statserver-Java

V případě, že inzerent, tedy uživatel reklamního systému, požaduje zobrazit graf se statistickou historií entit (viz obrázek 1.1), pak `frontend-api` s pomocí HTTP JSON API protokolu zavolá metodu `statReport` služby `statserver-java`, která na vstupu očekává ID uživatele, požadovanou časovou granularitu, jaké cílení se má použít a časový interval, pro který se má graf vykreslit.



Obr. 5.2: Schéma komunikace se Statserver-Java

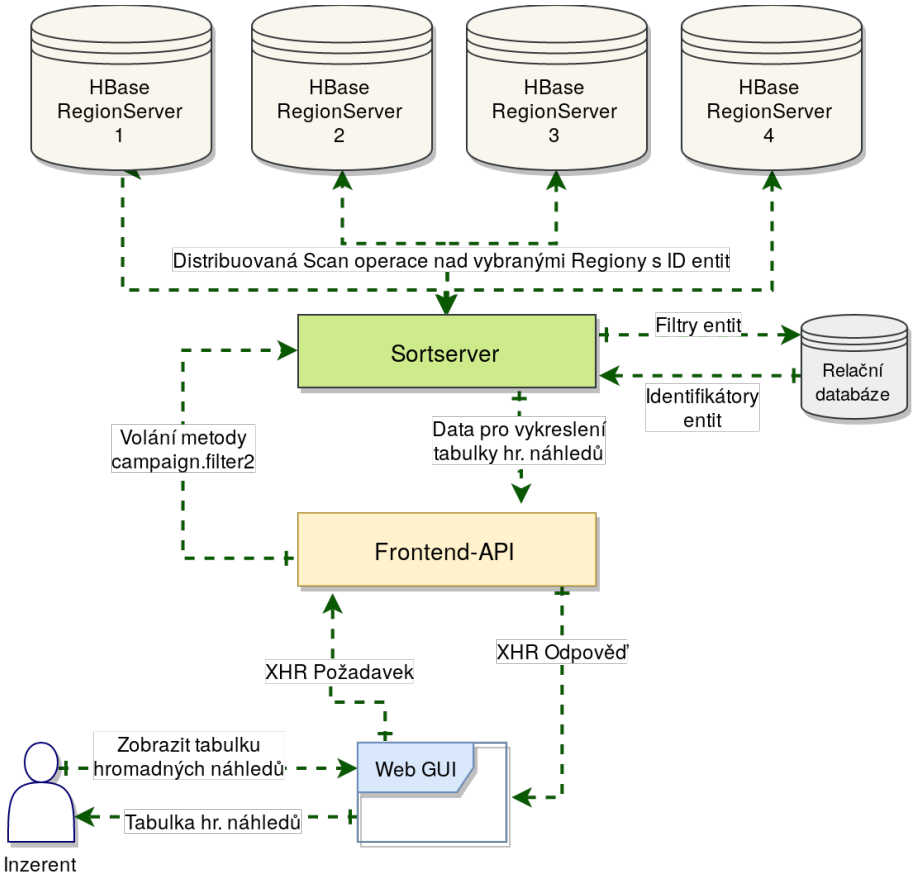
Na výstupu pak `statserver-java` vrací pole asociativních polí reprezentujících jeden bod v grafu. Klíč tohoto asociativního pole představuje statistickou datovou dimezi, jako je například kliknutí na reklamu. Hodnota přiřazená klíči pak určuje velikost dimenze za dané období.

`Statserver-java` získává data tak, že spustí operaci Scan s aktivními filtry na časové období nad `HBase` tabulkou hromadných náhledů v rodině sloupců odpovídající požadované časové granularitě dat. `HBase` filtry na časové období jsou již součástí implementované knihovny reklamního systému.

5.1.3 Sortserver

V případě, že inzerent požaduje na webovém rozhraní zobrazit tabulku hromadných náhledů (viz obrázek 1.2) například pro kampaně, pak `frontend-api` s pomocí FastRPC[1] protokolu zavolá metodu `campaign.filter2` služby `sortserver`, která na vstupu umožňuje specifikovat stovky parametrů. Těmi pro nás zajímavými však jsou ID uživatele, časové rozpětí, časová granularita, cílení reklamy a filtry nad datovými dimenzemi zobrazovaných entit. Kromě toho lze na vstupu požádat rovněž o seřazení entit podle jedné dimenze.

Na výstupu také vrací pole asociativních polí, neboli map. Každá mapa obsahuje ID entity a všechny s druhem entity spjaté statistické dimenze, včetně jejich velikosti.



Obr. 5.3: Schéma komunikace se Sortserverem

Filtrace entit dle hodnot jejich dimenzí probíhá tak, že se `sortserver` zeptá relační databáze na seznam těch identifikátorů entit, které filtru vyhovují. V relační

[1] FastRPC je protokol založený na XML-RPC. Byl vyvinut a označen za open-source společností Seznam.cz. Jeho zdrojový kód si lze stáhnout či prohlédnout na URL adrese: `https://github.com/seznam/fastrpc`

databázi jsou totiž uloženy celkové sumy všech dimenzí všech entit za celé časové období jejich existence. Jedná se tedy o data s nejnižší možnou časovou granularitou, a proto je možné, ba naopak i vhodné, tato data ukládat právě v relační databázi.

Za použití RPC volání s užitím Google Procotol Buffers se pak zašle distribuovaný požadavek na `HBase SortAndFilterEndpoint Coprocessor` metody `getSortedIds` všech `RegionServerů`, které drží data o požadovaných entitách. Všechny jejich odpovědi jsou pak sloučeny do výsledné odpovědi `sortserveru` popsané výše.

5.1.4 `SortAndFilterEndpoint Coprocessor`

`SortAndFilterEndpoint Coprocessor` Je implementací `HBase` aplikačního rozhraní `Endpoint Coprocessor` pro RPC komunikaci za použití `Google Protocol Buffer`. Z volání `sortserveru` přijme ID uživatele, seznam identifikátorů entit, jejichž statistiky se mají zobrazit, požadované řazení entit, filtry velikostí datových dimenzí entity, požadovanou časovou granularitu a časový interval, pro který má vrátit statistiky.

Dle druhu entity si pak odvodí název `HBase` tabulky, ve které bude hledat statistiky entit. Dále si dle dodané časové granularity odvodí název rodiny sloupců. Pak si ze seznamu identifikátorů entit odvodí počáteční a koncový identifikátor entity. Vzhledem ke konstrukci klíče řádku ve všech rodinách sloupců `HBase` tabulek hromadných náhledů je pak schopen přesně říct, jaký je nejmenší a největší možný klíč řádku, kde by mohl najít to, co hledá.

Klíč řádku se totiž skládá se série bajtů, kde první bajt představuje identifikátor uzlu, další čtyři bajty patří identifikátoru uživatele, další bajt je rok, další měsíc a posledních pět bajtů je identifikátor entity.

	Uzel	Identifikátor uživatele	Rok	Měsíc	Identifikátor entity
Velikost	1B	4B	1B	1B	5B

Obr. 5.4: Schéma klíče řádku v `HBase` tabulkách hromadných náhledů

Tímto rozložením bajtů v klíči řádku je zajištěna rychlost čtení uživatelových dat, neboť se díky implicitnímu řazení dat v `HBase` nacházejí blízko sebe.

5.2 Proč Endpoint Coprocessor?

Hlavní výhodou, proč se používá Endpoint Coprocessor namísto ostatních aplikačních rozhraní `HBase` je skutečnost, že je napsaný v nativním Java API, jež `HBase` používá, čímž se zbavuje zbytečné režie, která by nastala v případě volby jiného aplikačního rozhraní vlivem transformací a serializací dat.

Dalším pozitivem je minimalizace síťového provozu, neboť je aplikační vrstva přímo na uzlu, kde se nachází i požadovaná data. Díky tomu se navenek tato implementace jeví jako součást `HBase`, kterou je možno využít odkudkoliv. Po síti tedy putují pouze data agregovaná do požadované podoby v minimální možné granularitě. Poslední výhodou je skutečnost, že optimalizovaná implementace je jednoduše znovupoužitelná dalšími klientskými aplikacemi.

6 Volba metody optimalizace čtení dat

V této kapitole bude proveden návrh rozhraní, pomocí něhož `frontend` získá data potřebná k zobrazení přehledového statistického grafu, který respektuje nastavené filtry u tabulky hromadných náhledů. Dále bude proveden návrh `HBase` rozhraní, které vrátí statistické dimenze entit hromadných náhledů sečtené pro každou instanci zvolené časové granularity. Na závěr této kapitoly bude provedena volba metody optimalizace čtení dat.

6.1 Návrh rozhraní mezi `frontend` a `HBase`

`Statserver-java` bohužel nepodporuje filtraci entit, ze kterých se mají spočítat statistiky. Ani nelze zadat požadavek, aby byly vráceny statistiky pro jiné entity než kampaně. `Sortserver` na druhou stranu sice podporuje filtraci entit, včetně specifikace druhu entity, ale vrací data v příliš vysoké granularitě, což je pro účel zobrazení sumárních statistik v grafu zbytečné. Z toho plyne, že žádná z uvedených komponent nemá takové rozhraní, které by dle zadání umožnilo na vstupu omezit entity dle velikosti jejich dimenzí a zároveň na výstupu skutečně zobrazit jen ta data, která jsou potřeba, aby se nemrhalo výpočetním výkonem.

Pro získání statistik s požadovanou granularitou je tedy potřeba vytvořit vlastní metody. Je žádoucí, aby komponenta, kde se budou vytvářet tyto nové metody, byla napsána v kompilovaném jazyce, čímž se zajistí vyšší rychlost zpracování než u interpretovaných programovacích jazyků. Za účelem implementace nových metod se nabízí napsat novou komponentu. To by však bylo velice časově náročné, a proto je potřeba k implementaci nových metod vybrat již existující komponentu. Na výběr tedy zůstávají již zmiňované komponenty `statserver-java` a `sortserver`. Obě již mají implementovanou konektivitu do `HBase`. Nicméně pouze `sortserver` má navíc schopnost filtrovat entity a zároveň není omezen jen na kampaně. Ze všech zde uvedených důvodů vyplývá, že nejvhodnější komponentou pro implementaci nových metod je `sortserver`.

Každá entita hromadného náhledu má jiné statistické dimenze, ve kterých lze filtrovat. Jelikož však filtry budou součástí vstupních parametrů, musí se vstupní parametry nových metod lišit napříč entitami. Proto bude každá entita mít svou vlastní metodu, která na vstupu akceptuje filtry konkrétní entity.

Nové metody `sortserveru` budou mít následující názvy:

- campaign.granularizedSumStats
- ad.granularizedSumStats
- group.granularizedSumStats
- keyword.granularizedSumStats
- retargeting.granularizedSumStats
- interest.granularizedSumStats
- intend.granularizedSumStats
- sitelink.granularizedSumStats
- theme.granularizedSumStats
- pattern.granularizedSumStats
- productSet.granularizedSumStats

Na vstupu budou nové metody očekávat stejné parametry, jako `filter2` metody jednotlivých entit hromadných náhledů s tím rozdílem, že nebudou podporovat řazení dle dimenze entity, neboť to není u statistického grafu potřeba. Metody však na vstupu budou vyžadovat časovou granularitu dat. Volitelně také bude možné nastavit, zda je potřeba spočítat celkovou sumu vrácených statistik. Na výstupu bude pole map, kde každá mapa bude reprezentovat bod v grafu. V mapě budou sumy jednotlivých datových dimenzí entit za časové období definované časovou granularitou a zároveň každá mapa bude mít volitelné ohraničení začátku a konce časového období ve vlastním klíči.

Součástí nové implementace budou také integrační testy nových metod `sortserveru`. Tomuto tématu se ale budu věnovat podrobně až v kapitole 10.

6.2 `HBase` rozhraní

Zatím neexistuje takové `HBase` rozhraní, které by umožnilo novým metodám `sortserveru` získávat data hromadných náhledů agregovaných do zvolené časové granularity. Sice existuje takové `HBase` rozhraní, které by těmto metodám umožnilo získat požadovaná data, nicméně by nebyla sečtena skrze jednotlivé instance zvolené časové granularity, což by vyústilo ve zbytečně nadměrné zatížení sítě a nutnost veškerá data sčítat na straně `sortserveru`. Proto je potřeba vytvořit `HBase` rozhraní, které umožní `sortserveru` získat data ve tvaru nejnižší možné granularity dat s ohledem na požadovanou časovou granularitu.

Pro komunikaci se `sortserverem` byl již před psaním této diplomové práce zvolen Endpoint Coprocessor z důvodů uvedených v podkapitole 5.2. Uvedené důvody přetrvávají, a proto může `HBase` rozhraním pro nové `sortserver` metody zůstat

Endpoint Coprocessor. Nicméně do tohoto rozhraní je potřeba přidat novou metodu tak, aby splňovala požadavky nových `sortserver` metod.

Skutečnost, že nové metody `sortserveru` mají velmi podobné vstupní parametry jako původní `filter2` metody vypovídá o možnosti, že vstupní parametry `SortAndFilterEndpoint` metody `getSortedIds` budou mít do velké míry pro novou Endpoint Coprocessor metodu znovupoužitelné vstupní parametry. Máme-li nyní k dispozici tento poznatek, pak nejenže můžeme předejít duplicitám v definičním souboru `Google Protocol Buffer` pro `SortAndFilterEndpoint` implementaci `HBase` rozhraní, ale také využijeme již existující ověřenou funkční definici, která je již léty používání odladěná. Novou metodu tedy zakomponujeme do `SortAndFilterEndpoint` Coprocessor implementace a bude se jmenovat `getGranularizedSumStats`.

6.3 Optimalizace čtení dat

Z tlačítek okolo existujícího přehledového grafu statistické historie (obrázek 1.1) lze zjistit, že se uživatelům nabízí tyto statistiky celkem ve třech časových granularitách. Jsou jimi denní, týdenní a měsíční granularity. Pro denní a měsíční granularity již existují specializované rodiny sloupců pro všechny `HBase` tabulky hromadných náhledů, kam se průběžně agregují statistiky v konkrétní časové granularitě.

Pro týdenní časovou granularitu však žádné takové rodiny sloupců neexistují. Proto se musí při každém požadavku inzerenta o souhrn jeho týdenních statistik tyto statistiky vypočítat z dat denní časové granularity. Z toho důvodu je potřeba vytvořit rodiny sloupců určené pro týdenní statistiky a naimplementovat týdenní agregátor včetně zajištění jeho pravidelného běhu. Dále je nutné u nové metody `HBase` rozhraní naimplementovat čtení dat z těchto nových rodin sloupců. Příprava týdenních statistik v `HBase` je tedy prerekvizitou implementace nové metody `HBase` rozhraní. Zároveň nová metoda `HBase` rozhraní má být používána novými metodami `sortserveru`, z čehož vyplývá, že je implementace nové metody `HBase` rozhraní prerekvizitou nových metod `sortserveru`.

První v pořadí tedy bude implementace týdenního agregátoru včetně přidání potřebných rodin sloupců, na což bude navazovat vytvoření nové metody `HBase` rozhraní a nakonec bude doplněn poslední chybějící článek, tedy nové metody `sortserveru`.

7 Zajištění statistik s týdenní časovou granularitou

V této kapitole se budu věnovat popisu implementace týdenního agregátoru včetně přípravy schématu `HBase` tak, aby agregátor měl kam ukládat výstupní data. Dále se budu věnovat způsobu uložení agregovaných statistik, na což budou navazovat implementované možnosti omezit množinu agregovaných dat. Na závěr kapitoly popíšu, jakým způsobem proběhla optimalizace velikosti používané operační paměti při běhu agregátoru.

Veškerá implementace této kapitoly je k nahlédnutí v příloze v adresáři `/weekly-data-aggregation`.

7.1 Příprava `HBase` schématu

V kapitole 5 bylo uvedeno, že každý agregátor statistik hromadných náhledů založený na redukci časové granularity má svou odpovídající rodinu sloupců, kam zapisuje svá výstupní data.

Z toho důvodu jsem pro nový týdenní agregátor přidal vlastní rodiny sloupců do tabulky každé entity hromadných náhledů. Po vzoru ostatních rodin sloupců byl zvolen název `cw` pro statistiky kontextového cílení a `fw` pro statistiky fulltextového cílení reklamy. První písmeno označuje cílení (z angl. context a fulltext) a druhé vypovídá o týdenní granularitě (z angl. weekly). Doba života hodnot jejich buněk je neomezená, neboť inzerent bude moci požádat o libovolné časové období z intervalu existence jeho účtu.

Zde uvádím kompletní seznam `HBase` tabulek entit hromadných náhledů, do kterých jsem přidal příslušné rodiny sloupců:

- ad_stats_mod24
- ad_group_stats_mod4
- automatic_location_group_stats_mod24
- automatic_location_group_group_stats_mod4
- campaign_stats_mod24
- campaign_stats_mod4
- group_stats_mod24
- intend_stats_mod24
- interest_stats_mod24
- keyword_group_stats_mod4
- keyword_stats_mod24

- pattern_group_stats_mod4
- pattern_stats_mod24
- product_set_stats_mod24
- retargeting_group_stats_mod4
- sitelink_stats_mod24
- theme_stats_mod24

Samotná definice nové změny schématu je k nalezení v příloze pod názvem `/weekly-data-aggregation/weekly-column-family.txt`. Pro vytvoření nadefinovaných rodin sloupců se obsah tohoto textového souboru dodává na standardní vstup `hbase shell` příkazu na jednom z `RegionServerů`.

Pseudokód 7.1: Příkaz pro vytvoření týdenních rodin sloupců

```
$ cat weekly-column-family.txt | hbase shell
```

7.2 Implementace týdenního agregátoru

Abych mohl využít nativní Java `HBase` API a zajistit tak vyšší rychlost agregace dat, napsal jsem agregátor v jazyce Java. K tomuto rozhodnutí také přispěla existence dostupných knihoven reklamního systému pro serializaci a deserializaci statistického objektu do pole bajtů, které se fyzicky ukládá jako hodnota jednotlivých buněk v `HBase`.

7.2.1 Způsob uložení agregovaných dat

Zmíněný statistický objekt se nazývá `HBaseCtxStatProto` a umí reprezentovat hodnoty všech známých statistických dimenzí reklamního systému. Pomocí tohoto objektu se také ukládají denní statistiky, které vytváří denní agregátor, uvedený v kapitole 4.

Vytvořený týdenní agregátor čte hodnoty zvolených rodin sloupců denní časové granularity a serializuje je do `HBaseCtxStatProto` objektu. Za použití nově implementované třídy `WeekStatSummarizer` pak přečtené hodnoty statistických dimenzí `HBaseCtxStatProto` objektů sčítá do nového `HBaseCtxStatProto` objektu, který již reprezentuje statistiky týdenní časové granularity. Tento nový `HBaseCtxStatProto` objekt se pak v deserializované podobě uloží do odpovídající týdenní rodiny sloupců pod klíčem řádku, který se vytvoří dle obrázku 5.4 s tím rozdílem, že místo čísla měsíce se uloží číslo týdne, do kterého patří dny navzájem sečtených statistik.

Číslo týdne je odvozeno od mezinárodní normy pro zápis data a času ISO 8601, která říká, že první týden v roce je ten, který jako první obsahuje čtvrtek. Jiná interpretace tohoto pravidla zní, že se do nového roku počítá týden započatý v roce předchozím jen tehdy, když v novém roce obsahuje více dní, než v roce, ve kterém započal.

Kvalifikátor týdenních rodin sloupců má vždy hodnotu rovnu jedné, stejně jako tomu je u měsíčních rodin sloupců.

Hodnoty statistických dimenzí se sčítají zvlášť pro kontextové a fulltextové cílení reklamy. Pokud je tedy například zdrojová rodina sloupců (denní časové granularity) kontextového cílení, tak cílová rodina sloupců (týdenní časové granularity) bude také kontextového cílení, tedy `cw`, jak je uvedeno v podkapitole 7.1.

7.2.2 Volitelné omezení množiny agregovaných dat

Agregátor vyžaduje jeden poziční parametr. Je jím název entity, díky němuž pak za pomocí existující knihovny zjistí název `HBase` tabulky hromadných náhledů, ve které bude agregovat.

Pokud se agregátor spustí, aniž by se mu zadaly volitelné vstupní parametry, provede se agregace za celé období existence všech denních statistik dodané entity, čehož se využije pouze při prvním nasazení týdenního agregátoru do produkčního prostředí.

Vstupními parametry lze například zajistit omezení vstupních dat časovým intervalem denních statistik. Toho je možné dosáhnout díky složení bajtů v klíči řádku uvedeného na obrázku 5.4. Šestý a sedmý bajt každého řádku totiž dohromady určuje rok a měsíc, ke kterému řádek obsahuje statistická data. Informace o dni je pak obsažena ve kvalifikátoru sloupce. Za použití existujících knihoven je pak možné vytvořit takovou Scan operaci, která omezuje čtení dat na konkrétní časový rozsah.

Do vstupních parametrů agregátoru lze také zadat identifikátor uživatele, jehož denní statistiky chceme agregovat do týdenních. Toho lze využít například po opravě chybně naagregovaných denních statistik poškozeného uživatele, aby se tato oprava projevila i v jeho týdenních statistikách. Pro omezení Scan operace se přitom opět využívá složení klíče řádku denních statistik, kde uživatelský identifikátor zabírá druhý až pátý bajt.

Kromě toho agregátor také umožňuje agregovat pouze určité cílení reklamy. Zde dochází k omezení Scan operace jednoduše tím, že se specifikuje pouze rodina sloupců konkrétního cílení.

7.2.3 Proces sčítání denních statistik

O proces sčítání denních statistik do týdenních se stará nově naimplementovaná třída `WeekStatSummarizer`. Ta řeší uchovávání statistik v operační paměti pro každého uživatele, každou entitu a každý týden zvlášť pomocí instance třídy `StatSummarizer`. Tato třída definuje všechny známé statistické dimenze a implementuje jejich vzájemné sčítání s ohledem na datový typ těchto dimenzí. Kromě toho také umí inicializovat sebe sama z `HBaseCtxStatProto` objektu a naopak i vytvořit `HBaseCtxStatProto` objekt z vlastní instance.

Rozlišení entity, uživatele a týdne je implementováno tak, že se v poli o velikosti počtu agregovaných týdnů ukládá namapování uživatele na namapování statistik jeho entit za týden specifikovaný indexem v poli.

`WeekStatSummarizer` pak dodává metodu `getOrCreate`, která přijímá celkem tři parametry. Jedná se o index týdne, identifikátor uživatele a identifikátor entity. Tato metoda se pokusí v mapovacím poli najít vytvořenou `StatSummarizer` instanci, kterou vrátí na svůj výstup. Pokud ji nenajde, tak ji v mapovacím poli vytvoří a pak ji vrátí na výstup. Tuto metodu používá týdenní agregátor k získání potřebné instance `StatSummarizer`, nad níž pak zavolá metodu `add`, která do této instance přidá hodnoty jednolivých dimenzí ze serializovaného objektu denních statistik.

Před hledáním vytvořené instance třídy `StatSummarizer` se `WeekStatSummarizer` pokusí vymazat ty instance, které již zaručeně nebudou potřeba. Důvodem je skutečnost, že lze uvnitř mapovacího pole uchovat až tolik `StatSummarizer` instancí, kolik je násobek mezi celkovým počtem jednoho druhu entit a počtem agregovaných týdnů. Kupříkladu, kdyby měla proběhnout agregace jen za posledních 8 týdnů v tabulce entity klíčových slov, kterých evidujeme několik miliónů, tak by v mapovacím poli bez průběžného mazání vzniklo až několik desítek miliónů instancí `StatSummarizer` třídy. Při sto kilobajtové velikosti jedné instance `StatSummarizer` by tedy celková velikost mapovacího pole v operační paměti mohla dosáhnout až tisíce gigabajtů, tedy jednotky terabajtů, což je nepřípustné.

Které instance `StatSummarizer` lze bezpečně smazat?

Čtení dat probíhá sekvenčně po řádcích, kde každý řádek denních statistik obsahuje všechny dny měsíce[1]. Jelikož však jeden týden může začínat v jednom měsíci a končit v tom dalším, je potřeba si pamatovat alespoň tolik týdnů nazpět, kolik jich může alespoň z části být v jednom měsíci. Při maximální délce měsíce 31 dní může poslední

[1] Den měsíce je uložen v kvalifikátoru sloupce a proto jsou při přečtení jednoho řádku denních statistik přečteny statistiky všech dnů měsíce.

den prvního týdne být prvním dnem měsíce, jehož poslední den může být druhým dnem šestého týdne.

Týden č. 1				Týden č. 2								Týden č. 3								Týden č. 4								Týden č. 5								Týden č. 6			
St	Čt	Pá	So	Ne	Po	Út	St	Čt	Pá	So	Ne	Po	Út	St	Čt	Pá	So	Ne	Po	Út	St	Čt	Pá	So	Ne	Po	Út	St	Čt	Pá	So	Ne	Po	Út	St	Čt	Pá
27	28	29	30	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3
Měsíc č. 1				Měsíc č. 2																																	

Obr. 7.1: Schéma k úvaze maximálního počtu týdnů v jednom měsíci [2]

Dle této úvahy může do jednoho měsíce zasahovat až 6 týdnů, viz obrázek 7.1. Proto je bezpečné mazat z mapovacího pole celé týdny aktuálně zpracovávaného uživatele, které jsou starší než 5 týdnů nazpět.

Díky umístění identifikátoru uživatele v klíči řádku denních statistik je také jisté, že pokud již čteme statistiky jiného uživatele, tak už statistiky předchozího uživatele nebudou potřeba, a tudíž se mohou všechny smazat z paměti bez ohledu na aktuální týden.

Je sice pravda, že stejný uživatel může být uložený na více uzlech, a jelikož je číslo uzlu v klíči řádku prvním bajtem, je možné, že znovu začneme číst jeho denní statistiky v jedné iteraci. Číslo uzlu se však odvozuje od identifikátoru entity, takže je zaručené, že statistiky každé jeho entity jsou vždy pouze na jednom uzlu. Tudíž statistiky z aktuálního uzlu pak už nebudou potřeba.

[2] Zvolená čísla týdnů a měsíců jsou zde jen pro lepší představu maximálního počtu týdnů zasahujících do jednoho měsíce (zvolené číslování ani nemůže představovat čísla kalendářních měsíců ani kalendářních týdnů, neboť by například poslední týden prvního kalendářního měsíce nemohl být prvním kalendářním týdnem roku, nicméně ani počet celkových dní prvního a druhého měsíce by nesouhlasil s počtem prvního ani druhého kalendářního měsíce).

8 Implementace rozhraní koprocesoru

V této kapitole uvedu komunikační protokol, který ke komunikaci s novou metodou `HBase` rozhraní využívají nové metody `sortserveru`, jejichž popis následuje v kapitole 9. Posléze se budu zabývat inicializací interních struktur, potřebných ke správnému přečtení dat uložených v `HBase`. Pak bude následovat popis procesu skenování dat, kdy se k tomuto účelu používají nainicializované interní struktury. Na závěr popíšu, jakým způsobem se vytváří odpověď nové metody `getGranularizedSumStats`.

Veškerá implementace této kapitoly je k nahlédnutí v příloze v adresáři `/endpoint-coprocessor`. Veškeré změny provedené na této komponentě jsou k nalezení v rozdílovém souboru `/endpoint-coprocessor/coprocessor.diff`. Celkem bylo pro implementaci rozhraní pouze v `java` souborech přidáno či odebráno 2053 řádků.

8.1 Definice komunikačního protokolu

V podkapitole 6.2 bylo řečeno, že vstupní parametry nové metody `getGranularizedSumStats` budou velice podobné vstupním parametrům již existující metody `getSortedIds`. Tato metoda totiž na vstupu očekává pole identifikátorů entit, požadované cílení statistik a možnost přidat filtry dle hodnot datových dimenzí entity. Tyto vyjmenované vstupní parametry jsou podmnožinou vstupních parametrů, které umí zpracovávat nová metoda `getGranularizedSumStats`. Kromě nich totiž podporuje i specifikaci požadované časové granularity statistických dat a časový rozsah statistik. Jinak se vstupní parametry těchto metod neliší.

Naopak obě tyto metody mají velmi rozdílné výstupní parametry. Proto pro výstup nové metody již nelze využít existujících objektů z předchozí definice `Google Protocol Buffer`, což je důvod, proč byla definice `Google Protocol Buffer` pro `SortAndFilterEndpoint` upravena tak, aby umožnila na výstupu `getGranularizedSumStats` metody vracet nový objekt pod názvem `GranularizedSumStatsResponse`. Ten dovoluje posílat časově granulované statistiky včetně jejich rozdělení podle cílení reklamy. Kromě toho může v závislosti na vstupních parametrech obsahovat i sumy granulovaných statistik za celé období. Důvodem je, že přehledový graf kromě granulovaných statistik zobrazuje i celkovou sumu za celé období. Sice by tento výpočet mohl dělat uživatelův prohlížeč, nicméně je z hlediska rychlosti načtení grafu rychlejší tyto sumy vypočíst už při čtení dat, které je distribuované. Dále objekt `GranularizedSumStatsResponse` obsahuje již existující `Status` objekt, který říká, zda operace čtení dat z `HBase` uspěla, či nikoliv.

Výsledná definice `Google Protocol Buffer` komunikačního protokolu `SortAndFilterEndpoint Coprocessor` rozhraní je k nahlédnutí v příloze v souboru `/endpoint-coprocessor/SortAndFilter.proto`.

8.2 Metoda `getGranularizedSumStats`

V předchozí podkapitole byly definovány vstupní parametry této metody. Ty jsou na počátku běhu této metody použity ke konstrukci instance třídy `ScanSet`. Tato třída v pozdější části běhu zajišťuje vytvoření takových Scan operací nad `HBase` tabulkou hromadných náhledů, které respektují vstupní parametry metody.

8.2.1 Inicializace třídy `ScanSet`

V první části se ze seznamu identifikátorů entit vytvoří `HBase` filtr pod názvem `LongIdFilter`, který filtruje čtená data pouze na zadané entity právě díky složení klíče řádku, ve kterém se nachází identifikátor entity. Poté dojde k vytvoření filtru s názvem `DateFilter`, který zajistí ignoraci těch statistik, které se nenacházejí v zadaném časovém intervalu. Následuje prosté uložení informace o uživatelském identifikátoru, o žádané časové granularitě dat a o cílení reklamy.

Ohled na požadovanou časovou granularitu

Na konci inicializace třídy `ScanSet` následuje v metodě `addScanShard` složitý rozhodovací proces, jehož úkolem je připravit na skenování tabulky dostatečně velkou časovou granularitu k tomu, aby byly vráceny statistiky přesně vyhovující zadanému časovému intervalu. Například, pokud inzerent zadal omezení časovým intervalem 24. 4. 2019 až 4. 6. 2019 a přitom chce zobrazit měsíční časovou granularitu svých statistik, tak nelze k celému výpočtu použít pouze měsíční rodiny sloupců, neboť ty mají statistiky naagregované vždy od prvního až po poslední den v měsíci. Z toho důvodu je potřeba v uvedeném příkladu část dat přečíst z denních rodin sloupců a sečíst je tak, aby vznikly sumy v každém měsíci. V tomto příkladě třída `ScanSet` vytvoří celkem tři `Scan` objekty. První bude číst z rodin sloupců s denní časovou granularitou v časovém rozmezí 24. 4. 2019 až 30. 4. 2019. Druhý bude číst z rodin sloupců ve zvolené časové granularitě, tedy měsíční, v časovém rozmezí 1. 5. 2019 až 31. 5. 2019. Třetí `Scan` objekt bude číst z rodin sloupců opět denní časové granularity, tentokrát v časovém rozmezí 1. 6. 2019 až 4. 6. 2019. V případě prvního a třetího Scan objektu musí koprocesor zajistit sečtení přečtených statistik tak, aby tato data dokázal interpretovat v měsíční časové granularitě a ve správném formátu je vrátit na výstupu metody `getGranularizedSumStats`. Celý algoritmus rozhodovacího procesu je následující:

Pokud alespoň část ze žádaného časového intervalu vyplní velikost jednotky požadované časové granularity[1], tak se vytvoří `Scan` objekt, který pro tuto část časového intervalu použije rodinu sloupců odpovídající časové granularity, neboť již agregátory vypočetly sumu za všechny možné jednotky časové granularity. Pokud počátek časového intervalu není dnem, kterým začíná alespoň jedna jednotka požadované časové granularity, vytvoří se `Scan` objekt, který od tohoto dne použije rodinu sloupců denní časové granularity. Denní granularita se v tomto případě použije až do dne, kterým začíná alespoň jedna jednotka požadované časové granularity, a to pouze tehdy, když tento den je před koncem požadovaného časového intervalu, jinak se denní granularita použije na celý požadovaný časový interval. Poslední pravidlo říká, že pokud konec a začátek požadovaného časového intervalu nejsou ve stejné jednotce požadované časové granularity a zároveň dnem konce žádaného časového intervalu nekončí alespoň jedna jednotka požadované časové granularity, vytvoří se `Scan` objekt, který začíná prvním dnem v měsíci a končí posledním dnem požadovaného časového intervalu, přičemž se použije rodina sloupců denní časové granularity.

8.2.2 Inicializace statistické mapy

Metoda `getGranularizedSumStats` pokračuje inicializací statistické mapy, která je definována pomocí Java třídy `HashMap`, kde jejím klíčem je instance třídy `Number` a hodnotou instance třídy `SplittableStats`, která se v reklamním systému používá k přenosu statistických dat mimo `HBase`.

V klíči statistické mapy je uložen identifikátor přiřazený jednotce časové granularity. Pro denní granularitu má formát[2] `YYYYmmdd`, pro týdenní `YYYYww`, pro měsíční `YYYYmm`, pro čtvrtletní `YYYYmm` a roční jen `YYYY`. V hodnotě statistické mapy je statistický objekt reprezentující celková data za období definované konkrétní jednotkou časové granularity.

Inicializace celé mapy probíhá v metodě `initGranularityStatsMap` třídy `SortAndFilterEndpoint` ještě před Scan operací v `HBase`. Na základě požadovaného časového intervalu a časové granularity se pro vytvoření pole všech potřebných identifikátorů jednotek granularit zavolá granularitě příslušná metoda v třídě `GranularityCalendar`. Tím koprocesor získal seznam identifikátorů, které se později dostanou až k inzerentovi v podobě bodů v grafu.

Každému z těchto identifikátorů přiřadí prázdnou instanci `SplittableStats`, čímž je inicializace statistické mapy hotová.

[1] Jednotka časové granularity například pro týdenní granularitu je jeden týden, pro měsíční granularitu je to měsíc a pro denní časovou granularitu je jednotka jeden den.

[2] `YYYY` představuje rok, `mm` kalendářní měsíc v roce, `ww` kalendářní týden v roce dle ISO 8601 a `dd` kalendářní den v měsíci.

8.2.3 Skenování dat

Po inicializaci statistické mapy se na základě časové granularity pokračuje nastavením všech potřebných rodin sloupců. K tomu dochází v metodě `addColumnFamiliesByGranularity` třídy `ScanSet`. Pokud požadovaný časový interval končí dnešním dnem, tak se do připravených Scan objektů přidají rodiny sloupců inkrementálních statistik, aby inzerent měl graf sestavený z nejnovějších statistik aktuálního dne. Poté se vždy přidají rodiny sloupců denních statistik nezávisle na zadané časové granularitě pro speciální případy popsané v podkapitole 8.2.1. Pokud je časová granularita týdenní, přidají se týdenní rodiny sloupců. Pokud časová granularita je měsíční, čtvrtletní, roční, či není zadána vůbec (tím je zajištěna zpětná kompatibilita metod, které nepoužívají granularitu), přidají se měsíční rodiny sloupců.

Pak následuje samotné skenování dat pomocí metody `scanIntoStatsMap` v každém `Scan` objektu přichystaném v inicializaci `ScanSet` třídy, popsané v podkapitole 8.2.1. V této metodě se nachystá `HBase` skener, který prochází jednotlivé buňky vyhovující všem nastaveným `HBase` filtrům.

U každé buňky si koprocesor z klíče řádku zjistí rok, měsíc nebo týden a identifikátor entity. Pokud se jedná o denní rodinu sloupců, zjistí si i den podle kvalifikátoru sloupce. Pak serializuje hodnotu buňky do `HBaseCtxStatProto` objektu. Z identifikátoru entity, data aktuálních statistik a aktuální rodiny sloupců pak sestaví identifikátor klíče do statistické mapy, kde získá odpovídající instanci třídy `SplittableStats`, do níž přidá statistiky z `HBaseCtxStatProto` objektu. Tímto procesem se postupně naplní statistická mapa.

8.2.4 Tvorba odpovědi

Po dokončení skenování dat se pro každou hodnotu statistické mapy vytvoří `GranularizedSumStats` instance, která je očekávána v `Google Protocol Buffer` odpovědi. Nastaví se jí tzv. identifikátor granularity, což je aktuální klíč statistické mapy, tedy identifikátor jednotky časové granularity. `GranularizedSumStats` instanci se nastaví fulltextová a kontextová hodnota časově granulované statistiky. Pokud byl na vstupu `getGranularizedSumStats` metody specifikován příznak pro zahrnutí celkový sumy, tak se jednotlivé časové granulované statistiky navzájem sečtou do celkové sumy. Výsledný objekt se deserializuje a odešle zpět na `sortserver`.

9 Implementace rozhraní `sortserveru`

V této kapitole se budu věnovat popisu implementace nových metod existující `sortserver` komponenty.

Veškerá implementace této kapitoly je k nahlédnutí v příloze v adresáři `/sortserver`. Veškeré změny provedené v `java` souborech této komponenty jsou k nalezení v rozdílovém souboru `/sortserver/sortserver-only-java.diff`. Celkem bylo pro implementaci rozhraní pouze v `java` souborech přidáno či odebráno 4845 řádků.

9.1 Implementace `granularizedSumStats` metody pro všechny druhy entit

V podkapitole 6.1 bylo již uvedeno, že u vstupních parametrů nových metod `granularizedSumStats` bude rozdíl od vstupních parametrů již existujících `filter2` metod spočívat v absenci možnosti řazení, neboť to pro účely zobrazení přehledového grafu není potřeba. Kromě toho ale nové metody budou oproti `filter2` metodám navíc akceptovat požadovanou granularitu dat. Výstupní parametry všech nových metod `sortserveru` již přesně nadefinovala zmíněná podkapitola.

Všechny nové metody `sortserveru` jsou napsány tak, aby využívaly stejnou logickou strukturu kódu a aby se jejich implementace navzájem lišily jen minimálně. Kromě vstupních parametrů jejich vzájemná odlišnost spočívá pouze ve způsobu inicializace třídy `UserFilter2`, která zajišťuje filtraci entit dle dodaných filtrů statistických dimenzí druhu entity. Její inicializace je vždy jiná v parametru třídy `EntityDefinition`. Tato třída pro každý druh entity definuje vlastní pravidla filtrace. Její instance je statickou součástí rozhraní každého druhu entity, což usnadňuje celý inicializační proces `UserFilter2` třídy.

Následuje volání nové metody `granularizedSumFiltered` třídy `FilterAndSort`. Tato metoda prvně z relačních databází díky instanci `UserFilter2` třídy získá seznam identifikátorů entit. Poté na základě identifikátorů entit spočítá, kterých `HBase`uzlů se bude ptát na data. Pak na všechny takto zjištěné uzly paralelně pošle `Google Protocol Buffer` zprávu žádající `HBase`rozhraní o data.

Jakmile všechny uzly vrátí odpověď, `granularizedSumFiltered` metoda pokračuje sloučením odpovědí všech uzlů voláním metody `mergeCoprocessorGranularizedSumStatsResponses` třídy `FilterAndSort`. Nakonec vrátí sloučenou odpověď `granularizedSumStats` metodě, která ji vloží do instance objektu určeného k definici odpovědi. Tato instance je pak vrácena na výstupní rozhraní `sortserveru`, čímž `frontend` získává požadovaná data.

10 Integrační testy navrženého systému

V této kapitole se budu věnovat provedení integračních testů navrženého systému. Nejdříve popíši formát zápisu integračních testů a způsob jejich spouštění. Pak představím obsah integračních testů vytvořených pro nově navržený systém čtení dat z `HBase`

Také se v této kapitole budu zabývat vytvořením `HBase` kontejneru potřebného k testování. Veškeré testovací scénáře jsou k nalezení v adresáři `/integration-tests/scenarios` . Implementace k tomu potřebné `HBase` v `Dockeru` je pak v adresáři `/integration-tests/hbase-Docker` .

Veškeré změny provedené ve vytváření `HBase Dockeru` jsou k nalezení v rozdílovém souboru `/integraion-tests/hbase-Docker/hbase-Docker.diff` . Celkem bylo pro implementaci `HBase` v `Dockeru` přidáno či odebráno 3144 řádků.

10.1 Formát integračních testů

Na `backendu` reklamního systému platí úzus, že se scénáře integračních testů definují v `YAML` souboru. V každém `YAML` objektu se nachází celkem čtyři hlavní klíče. První klíč je `method`, který definuje metodu, která se má scénářem zavolat. Druhý klíč je `parameters`, do kterého se vkládá libovolná struktura dat. Konkrétní podoba této struktury závisí na vstupních parametrech provolávané metody. Třetí klíč je `response`, ve kterém je opět libovolná struktura dat. Tato struktura slouží k definici očekávaného výstupu metody. Posledním klíčem je `skipped`, jehož hodnota je booleovského typu, která říká, zda se má scénář nadefinovaný tímto `YAML` objektem přeskočit.

Výsledné testovací scénáře pak jdou na vstup `rpctest` skriptu napsaného v interpretovaném jazyce `Python`[1]. Tomuto skriptu se při jeho spuštění zadá cílová URL adresa služby, na které bude pouštět nadefinované scénáře. Pokud alespoň jedna testovaná metoda vrátí jinou odpověď, než jakou odpovídající testovací scénář očekává, pak skript vypíše, jaký byl očekávaný výstup metody, jaký výstup dostal a pro rychlou orientaci vypíše i rozdíl mezi těmito výstupy. Následuje ukončení skriptu s nenulovou návratovou hodnotou, což signalizuje neúspěch testů. Pokud však všechny testovací scénáře uspějí, pak je návratová hodnota skriptu nula, tedy úspěch.

[1] Testovací skript `rpctest` je také k nalezení v příloze. Je pro něj vytvořený adresář `/integration-tests/rpctest` .

10.2 Kubernetes sandbox

Pro integrační testování komponent `backendu` reklamního systému se používá Kubernetes cluster. Kubernetes je open-source platforma pro správu kontejnerizovaných služeb[2]. [41]

Sandbox je termín, který se v reklamním systému používá pro označení jednorázového prostředí určeného na testování komponent. Před zahájením testování je vždy vytvořena nová instance sandboxu. Jakmile je testování dokončeno, je tato instance sandboxu zahozena. K dynamickému vytváření a zahazování těchto jednorázových testovacích prostředí se využívá Kubernetes, který zajišťuje alokaci potřebných hardwarových a síťových prostředků.

Použití sandboxového prostředí tak ale kvůli charakteru Kubernetu vyžaduje, aby testované komponenty byly kontejnerizovanými službami. V reklamním systému se pro kontejnerizaci používá `Docker` což je jedna z mnoha implementací kontejnerizovaných systémů. [**?**]

10.3 Testovací scénáře

Testovací scénáře se vždy pouštějí nad komponentami v čistém sandboxovém prostředí. Z toho důvodů je potřeba ještě před spuštěním integračních testů inicializovat jednotlivé entity, o jejichž existenci se tím pádem bude moci `sortserver` dozvědět při zjišťování seznamu identifikátorů entit, pomocí kterého se zeptá `HBase` rozhraní na statistická data.

Tuto inicializaci zajišťuje první `YAML` definice integračních testů, která prakticky není testovacím scénářem, neboť pouze využívá schopnosti `rpctest` skriptu převést `YAML` objekt na volání konkrétní metody určité komponenty za účelem vytvoření potřebných entit v relačních databázích, kde je pak `sortserver` očekává. Název této `YAML` definice je `001-rpc_adminserver-init.yml`.

Po inicializaci potřebných entit následují očekávané testovací scénáře. První v pořadí je `002-daily.yml`, který testuje denní časovou granularitu. Dalším je `003-weekly.yml`, za nímž následuje `004-monthly.yml`, pak `005-quarterly.yml` a nakonec `006-yearly.yml`. Z anglických názvů je zřejmé, že se jimi testují týdenní, měsíční, čtvrtletní a nakonec roční časová granularita.

Vyjmenované `YAML` definice se aplikují na všechny naimplementované metody `sortserveru`, čímž se otestuje nejen implementace `sortserveru`, ale i koprocesorů a týdenních agregátorů.

[2] Kontejnerizovaná služba je taková služba, která běží v kontejneru. Kontejner je standardizovaná softwarová jednotka, která obaluje dodaný kód, včetně všech jeho závislostí tak, aby navržená aplikace běžela rychle a spolehlivě. [**?**]

10.4 HBase v Dockeru

Jelikož integrační testy očekávají přesně stanovenou podobu dat, musí se pro ně použít taková instance HBase která obsahuje stejně definovaná data. To však nelze zajistit jinak, než její kontejnerizací a zakomponováním do sandboxového prostředí, kde se nová instance HBase spustí na požádání s nachystanými daty v očekávané podobě.

Aby nemuselo při každém startu kontejneru docházet k importu schématu a požadovaných dat, je žádoucí tato schémata a data zapsat přímo do výsledného Docker obrazu. Pak totiž při jeho spuštění budou všechna data připravená k přečtení.

10.4.1 Import schématu a dat

Import schématu lze provést pouze tehdy, když HBase zrovna běží. Za tímto účelem tedy vznikl automatizační skript, který při stavění Docker obrazu zařídí potřebnou sérii úkonů. Nejdříve nakonfiguruje HBase a HDFS Poté HBase spustí a počká, až Master Server začne vystavovat webové rozhraní. Mezitím automatizační skript začne stahovat HbDC[3] nástroj pro import dat. Tento nástroj později skript zužitkuje.

Pokud se HBase nespustí do stanoveného časového limitu, celý stavící proces se ukončí s chybou. Pokud však spuštění uspěje, naimportují se všechna reklamnímu systému známá HBase schémata. Jen tento proces importu trvá téměř deset minut. Pokud byl úspěšný, pak automatizační skript přechází na import dat. Ten provádí tak, že najde všechny JSON soubory v adresáři data a dodá je na vstup HbDC nástroje. Tento nástroj zařídí jejich přidání do příslušných HBase tabulek s denní granularitou. Pokud byl i import dat úspěšný, pak skript přechází na agregaci dat.

10.4.2 Agregace dat

V této fázi jsou staženy aktuální verze všech knihoven potřebných k provedení agregace, včetně týdenního a měsíčního agregátoru. Následuje spuštění týdenní agregace pro každou tabulku hromadných náhledů. Pak se spustí měsíční agregace pro každou tabulku hromadných náhledů.

Pokud byla agregace úspěšná, HBase instance se zastaví a smažou se všechny dodatečně stažené knihovny a nástroje, aby nemohlo dojít k jejich nechtěnému načtení do JVM při pouštění hotového kontejneru. Stav kontejneru je v této fázi uložen do Docker obrazu, který kvůli závislostem HBase a Hadoopu zabírá přibližně

[3] HbDC je interní knihovna pro import dat do HBase ve formátu již dříve zmíněného objektu HBaseCtxStatProto z JSON souboru, takže i člověk může snadno definovat hodnoty jednotlivých statistických dimenzí entity přímo do rodiny sloupců denních statistik.

3 GB místa. V této podobě je pak `Docker` obraz odeslán do interního úložiště `Docker` obrazů, odkud jej může použít kdokoliv s přístupem do sítě.

10.5 GitLab CI

`GitLab` je git[4] repozitář, který zároveň umožňuje používat CI (Continuous Integration) proces. Ve zjednodušené formě to je proces, který poskytuje nástroje ke zlepšení vývoje automatizací. [43]

V této práci je tento proces využit k automatickému spouštění integračních testů `sortserveru` pokaždé, když dojde ke změně jeho kódu. Autor těchto změn pak v případě, že tyto testy selžou, dostane jemu preferovaným kanálem informaci o tomto selhání. To velmi urychluje proces vývoje, neboť se programátor nemusí zabývat spouštěním testů a kontrolou, zda uspěly, ale může se zaměřit na řešení jiného problému.

[4] `Git` je nástroj primárně určený na verzování zdrojových kódů. [44]

11 Dosažené výsledky

Výstupem této práce je funkční `backend` rozhraní připravené pro tým `frontend` tak, aby jeho členové byli schopni za pomoci vstupních parametrů získat data potřebná k vykreslení přehledového statistického grafu, zobrazeného na obrázku 1.1. Přitom jsem se zaměřil na optimalizaci čtení dat, aby čtení proběhlo v co možná nejmenším časovém intervalu. Mojí snahou zároveň bylo napsat stabilní implementaci, k čemuž slouží integrační a unit testy jednotlivých komponent. Všechny napsané testy byly úspěšné.

Nová implementace se dotkla již existujícího řešení, zejména na straně `HBase` rozhraní, kde byly interní funkce a metody přepsány do takového stavu, aby byla zachována původní funkcionalita systému, ale zároveň se umožnila napsat nová. Proto bylo na místě porovnat výkonnost již existujících metod před a po provedené implementaci. Vzhledem k tomu, že při úpravách existujících metod bylo prioritní optimalizovat čtecí proces, tak se výkonnost již existujících metod na `HBase` rozhraní dokonce zlepšila. Vyplývá to ze zátěžových testů, uvedených v následující podkapitole 11.1.

11.1 Vliv změn na existující rozhraní `HBase`

Za účelem zjištění, zda provedenými změnami nedošlo k negativnímu ovlivnění existujících metod na `HBase` rozhraní, byly provedeny zátěžové testy.

V první fázi testování byla na vývojovém `HBase` clusteru nainstalována nová verze koprocesoru dodávající novou implementaci již existujícího `HBase` rozhraní. Následovalo spuštění série dvaceti volání metody `getSortedIds` na rozhraní koprocesoru. V každém volání bylo do vstupních parametrů dodáno časové období dvou let uživatele se statistikami za celé toto období a na výstupu byla žádána jedna statistická dimenze uložených dat. Čas vykonávání každého volání byl uložen do prvního souboru. Pak byla nainstalována stabilní verze koprocesoru, jak byla implementována před úpravami provedenými v této práci. Znovu byla spuštěna série stejných dvaceti volání rozhraní koprocesoru a časy vykonávání jednotlivých volání byly uloženy do druhého souboru.

Druhá fáze proběhla po třiceti minutách, kdyby bylo stejné volání puštěno celkem šedesátkrát nad původní implementací `HBase` rozhraní. Doba běhu jednotlivých volání byla přidána na konec druhého souboru. Pak byla nainstalována opět nová verze koprocesoru, na což opět navazovalo spuštění šedesáti stejných volání. Časové intervaly mezi zavoláním a vrácením výsledků metod byly přidány na konec prvního souboru.

Dle uložených výsledků se ukázalo, že průměrný čas trvání vykonávání metody v nové verzi je `1,04265` sekund, zatímco ve staré stabilní verzi je `1,23293` sekund.

Došlo tedy ke zrychlení odpovědi metody, jejíž optimalizace ani nebyla cílem této práce. Při přepisu interních funkcí a metod totiž byly prováděny i velmi malé optimalizace, díky kterým se zefektivnil rozhodovací proces pouštěný pro každou buňku, kterou koprocesor určitým způsobem zpracovává. Jelikož je tento proces používaný také již existujícími metodami, rovněž jejich čtecí proces logiky se tím pádem také zlepšil.

Mezi přiloženými soubory je k nalezení také rozdílový soubor obsahující veškeré změny provedené na straně koprocesoru pod názvem `/endpoint-coprocessor/coprocessor.diff`. Mezi těmito změnami lze jednotlivé optimalizace rozhodovacího procesu spatřit v rozdílové části třídy `Scan`. Optimalizace však z velké části proběhla také v třídě `DateRange`.

11.2 Vliv změn na existující `sortserver` rozhraní

Vzhledem k tomu, že při zavádění nového `sortserver` rozhraní nebylo potřeba měnit interní procesy, ale stačilo jen přidat nové rozhraní, které nemá vliv na způsob vykonávání již existujících metod, nebylo potřeba spouštět zátěžové testy.

Nicméně i přesto byly provedeny. Sice ne v takovém rozsahu, jako na straně koprocesoru, ale jejich výsledkem bylo zlepšení v řádu stovek milisekund.

12 Závěr

Cílem této práce bylo navrhnout a implementovat optimální čtení dat z distribuované databáze `Apache HBase`. Podrobně jsem popsal problém, kterého se tento cíl týká, aby čtenáři v průběhu práce bylo zřejmé, proč podnikám jednotlivá rozhodnutí. Na to jsem navázal obecným úvodem do NoSQL databází, po čemž následovalo srovnání s databázemi SQL a nakonec příklady NoSQL databází, abych kategorizoval používanou `Apache HBase` databázi. Pokračoval jsem úvodem do `Apache HBase`, ve kterém jsem popsal její datový model, na který navazuji definicí jeho operací. Dále jsem zevrubně popsal architekturu `HBase`, aby čtenář získal informace, na které se později v textu odkazuji. V této architektonické části jsem se kromě jiného zabýval fyzickým uložením dat, včetně procesu jejich čtení, neboť jsem vyhodnotil, že znalost těchto procesů je klíčová k pochopení zvolené optimalizace čtení dat. Poté jsem uvedl existující `HBase` rozhraní a navzájem srovnal jejich výhody a nevýhody, aby se tyto úvahy později zužitkovaly.

Pokračoval jsem popisem způsobu vytváření a zápisu statistických dat v reklamním systému. Zavedl jsem pojem granularity dat, což mi pak umožnilo zdůvodnit, proč na každý druh dat existuje vlastní agregátor. Dále jsem popsal proces čtení statistických dat tak, jak již byl implementovaný, aby se tato existující architektura později dala použít k návrhu řešení nového způsobu čtení dat. Na to jsem navázal popisem zvolené implementace k dosažení optimálního čtení dat. Zabýval jsem se zde i definicí použitých protokolů a formátu vstupně-výstupních dat mezi jednotlivými vrstvami architektury reklamního systému.

V další kapitole jsem se věnoval problematice zajištění statistik s týdenní časovou granularitou, což bylo podstatou zvolené optimalizace. Rozebral jsem zde připravení `HBase` schématu, jakožto prerekvizity samotné implementace týdenního agregátoru. Způsob optimalizace této implementace jsem pak zdůvodnil. Následoval popis implementovaného rozhraní koprocesoru, ve kterém jsem nadefinoval komunikační protokol a přiblížil čtenáři algoritmus čtení časově granulovaných statistických dat přímo z `HBase`. Po dokončení nové metody koprocesoru jsem pokračoval implementací nových metod `sortserveru`, jakožto jedné z komponent interní architektury reklamního systému. Abych však zajistil jistou stabilitu implementovaného systému, pokračoval jsem zajištěním integračních testů v sandboxovém prostředí Kubernetu za použití kontejnerů. Pouštění těchto testů jsem pak zakomponoval do vývojového procesu reklamního systému.

Cíle této práce bylo dosaženo. Pro `frontend` tým bylo připraveno rozhraní, které umožňuje inzerentům v grafu zobrazovat relevantní historii jejich statistických dat. V průběhu zhodnocování jsem také došel k závěru, že dokonce došlo ke zrychlení metody, do které jsem zasahoval pouze nepřímo.

Zdroje

[1] Pivotal software. *Understanding NoSQL* [online] Pivotal software, Spring.io, 2019. [cit. 2. ledna 2019] Dostupné z URL: <https://spring.io/understanding/NoSQL>

[2] Yegulalp, Serdar. *What is NoSQL? NoSQL databases explained* [online] InfoWorld.com, 2017. [cit. 2. ledna 2019] Dostupné z URL: <https://www.infoworld.com/article/3240644/nosql/what-is-nosql-nosql-databases-explained.html>

[3] Apache HBase Team. *Apache HBase - Apache HBase ™ Home* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/>

[4] Borthakur, Dhruba. *HDFS Architecture Guide* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>

[5] Apache HBase Team. *Apache HBase ™ Reference Guide - Data model* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#datamodel>

[6] Snively, Ben. *Combine NoSQL and Massively Parallel Analytics Using Apache HBase and Apache Hive on Amazon EMR* [online] Amazon EMR, 2016. [cit. 2. ledna 2019] Dostupné z URL: <https://aws.amazon.com/blogs/big-data/combine-nosql-and-massively-parallel-analytics-using-apache-hbase-and-apache-hive-on-amazon-emr/>

[7] Apache HBase Team. *Apache HBase ™ Reference Guide - 23. Namespace* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#_namespace>

[8] Apache HBase Team. *Apache HBase ™ Reference Guide - 24. Table* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#_table>

[9] Apache HBase Team. *Apache HBase ™ Reference Guide - 26. Column Family* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#columnfamily>

[10] Buyya, Rajkumar. *High Performance Cluster Computing: Architectures and Systems* Prentice Hall. [cit. 11. května 2019]

[11] Apache HBase Team. *Apache HBase ™ Reference Guide - 25. Row* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#_row>

[12] Apache HBase Team. *Apache HBase ™ Reference Guide - 27. Cells* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#_cells>

[13] Apache HBase Team. *Apache HBase ™ Reference Guide - 29. Versions* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#versions>

[14] Apache HBase Team. *Apache HBase ™ Reference Guide - 28. Data Model Operations* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#_data_model_operations>

[15] McDonald, Carol. *An In-Depth Look at the HBase Architecture* [online] MapR Technologies, Inc. [cit. 9. května 2019] Dostupné z URL: <https://mapr.com/blog/in-depth-look-hbase-architecture/>

[16] Apache HBase Team. *Apache HBase ™ Reference Guide - 69. Client Request Filters* [online] The Apache Software Foundation. [cit. 12. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#client.filter>

[17] Sinha, Shubham. *HBase Architecture: HBase Data Model & HBase Read/Write Mechanism* [online] Brain4ce Education Solutions Pvt. Ltd. [cit. 9. května 2019] Dostupné z URL: <https://www.edureka.co/blog/hbase-architecture/>

[18] Borthakur, Dhruba. *HDFS Architecture Guide - NameNode and DataNodes* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>

[19] Apache HBase Team. *Apache HBase ™ Reference Guide - 72. Regions* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#regions.arch>

[20] Bertozzi, Matteo. *HBase - Who needs a Master?* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://blogs.apache.org/hbase/entry/hbase_who_needs_a_master>

[21] Apache HBase Team. *Apache HBase ™ Reference Guide - 70. Master* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#architecture.master>

[22] Zahoor, Mohamed J. *HBase HMaster Architecture* [online] Zahoor, Mohamed J. [cit. 9. května 2019] Dostupné z URL: <http://blog.zahoor.in/2012/08/hbase-hmaster-architecture/>

[23] Apache HBase Team. *Apache HBase ™ Reference Guide - 71.7. Write Ahead Log (WAL)* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#wal>

[24] Apache HBase Team. *Apache HBase ™ Reference Guide - 72.7.1. MemStore* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#store.memstore>

[25] Apache HBase Team. *Apache HBase ™ Reference Guide - 72.7.4. StoreFile (HFile)* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#hfile>

[26] Zhang, Schubert. *HFile: A Block-Indexed File Format to Store Sorted Key-Value Pairs* [online] cloudepr.blogspot.com. [cit. 9. května 2019] Dostupné z URL: <http://cloudepr.blogspot.com/2009/09/hfile-block-indexed-file-format-to.html>

[27] Apache HBase Team. *Apache HBase ™ Reference Guide - 71.4. Block Cache* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#block.cache>

[28] Apache HBase Team. *Apache HBase ™ Reference Guide - 72.7.7. Compaction* [online] The Apache Software Foundation. [cit. 9. května 2019] Dostupné z URL: <https://hbase.apache.org/book.html#compaction>

[29] Apache HBase Team. *Apache HBase ™ Reference Guide - Apache HBase APIs* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#hbase_apis>

[30] Apache HBase Team. *Apache HBase ™ Reference Guide - Apache HBase External APIs* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#external_apis>

[31] Apache HBase Team. *Apache HBase ™ Reference Guide - 97. REST* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#_rest>

[32] Apache Software Foundation. *Apache Thrift - Home* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://thrift.apache.org/>

[33] Apache HBase Team. *Apache HBase ™ Reference Guide - Thrift API and Filter Language* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#thrift>

[34] Scala Community. *Introduction | Scala Documentation* [online], scala-lang.org. [cit. 2. ledna 2019] Dostupné z URL: <https://docs.scala-lang.org/tour/tour-of-scala.html>

[35] Bahadoor, Nadim. *Learn what is Scala programming language* [online], 2018. [cit. 2. ledna 2019] Dostupné z URL: <https://allaboutscala.com/tutorials/scala-introduction/learn-scala-programming-language/>

[36] Apache HBase Team. *Apache HBase ™ Reference Guide - 109. Types of Coprocessors* [online] The Apache Software Foundation. [cit. 2. ledna 2019] Dostupné z URL: <https://hbase.apache.org/book.html#_types_of_coprocessors>

[37] Open Source Robotics Foundation. *msg - ROS Wiki* [online] Open Source Robotics Foundation, 2017. [cit. 2. ledna 2019] Dostupné z URL: <https://wiki.ros.org/msg>

[38] Google Developers. *Developer Guide | Protocol Buffers | Google Developers* [online] Google Developers, 2018. [cit. 2. ledna 2019] Dostupné z URL: <https://developers.google.com/protocol-buffers/docs/overview>

[39] Elmore, Ryan. *What is the granularity of data?* [online] Quora.com, 2017. [cit. 2. ledna 2019] Dostupné z URL: <https://www.quora.com/What-is-the-granularity-of-data/answer/Ryan-Elmore-2>

[40] Cuthbert, Guy. *What is granularity in data warehouse?* [online] Quora.com, 2017. [cit. 2. ledna 2019] Dostupné z URL: <https://www.quora.com/What-is-granularity-in-data-warehouse/answer/Guy-Cuthbert>

[41] The Kubernetes Authors. *What is Kubernetes* [online] The Kubernetes Authors, 2019. [cit. 13. května 2019] Dostupné z URL: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

[42] Docker Inc. *What is a Container?* [online] Docker Inc, 2019. [cit. 13. května 2019] Dostupné z URL: <https://www.docker.com/resources/what-container/>

[43] GitLab Inc. *What is GitLab?* [online] GitLab Inc, 2019. [cit. 13. května 2019] Dostupné z URL: <https://about.gitlab.com/what-is-gitlab/>

[44] The Git Project. *1.3 Getting Started - What is Git?* [online] The Git Project, 2019. [cit. 13. května 2019] Dostupné z URL: <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git>

Seznam zkratek

PoC	Proof of Concept
SQL	Structured Query Language
NoSQL	Not only SQL
DB	databáze
HDFS	Hadoop Distributed File System
DDL	Data Definition Language
API	Application Programming Interface
REST	Representational State Transfer
JVM	Java Virtual Machine
WAL	Write Ahead Log
LRU	Least Recently Used
RPC	Remote Procedure Call
CI	Continuous Integration

Seznam příloh

A Obsah přiloženého paměťového média **72**

A Obsah přiloženého paměťového média

```
/
├── xkozlo04-DP.pdf ..... písemná zpráva diplomové práce
├── thesis-sources/ ..... zdrojové soubory písemné zprávy diplomové práce
├── weekly-data-aggregation/
│   ├── weekly-aggregator.diff ..... rozdílový soubor všech změn agregátorů
│   ├── weekly-column-family.txt ..... definice HBase schématu
│   ├── WeekAggregator.java ..... týdenní agregátor
│   └── lib/ ..... knihovny agregátoru
├── endpoint-coprocessor/
│   ├── coprocessor.diff ..... rozdílový soubor všech změn koprocesoru
│   ├── SortAndFilter.proto ..... definiční soubor Google Protocol Buffer
│   ├── SortAndFilterEndpoint.java ..... implementace SortAndFilterEndpoint
│   ├── lib/ ..... knihovny koprocesoru
│   └── test/ ..... unit testy koprocesoru
├── sortserver/
│   ├── sortserver-only-java.diff ..... rozdílový soubor změn sortserveru
│   ├── handlers/ ..... adresář implementací jednotlivých metod sortserveru
│   └── lib/ ..... knihovny sortserveru
└── integration-tests/
    ├── hbase-docker/ ..... adresář s definicemi HBase v Dockeru
    │   ├── hbase-docker.diff ..... rozdílový soubor změn HBase v Dockeru
    │   └── src/ ..... zdrojové soubory pro sestavení Docker obrazu
    ├── rpctest/ ..... implementace skriptu pro přehrávání testovacích scénářů
    └── scenarios/ ..... scénáře integračních testů celého systému
        ├── sortserver-only-tests.diff ..... rozdílový soubor všech změn testů
        ├── 014-campaign.granularizedSumStats/ ..... scénáře kampaní
        ├── 015-ad.granularizedSumStats/ ..... scénáře reklam
        ├── 016-group.granularizedSumStats/ ..... scénáře sestav
        ├── 017-keyword.granularizedSumStats/ ..... scénáře klíčových slov
        ├── 018-retargeting.granularizedSumStats/ ..... retargeting scénáře
        ├── 019-interest.granularizedSumStats/ ..... scénáře zájmů
        ├── 020-intend.granularizedSumStats/ ..... scénáře úmyslů
        ├── 021-sitelink.granularizedSumStats/ ..... sitelink scénáře
        ├── 022-theme.granularizedSumStats/ ..... scénáře témat
        ├── 023-pattern.granularizedSumStats/ ..... scénáře šablon
        └── 024-productSet.granularizedSumStats/ ..... productSet scénáře
```