FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Analýza trendů ve vyhledávání v reálném čase
Student:	Bc. Samuel Butta
Vedoucí:	Ing. Barbora Červenková DiS.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je navrhnout a vytvořit funkční prototyp systému, který počítá a vizualizuje trendy ve fulltextovém vyhledávání uživatelů internetového vyhledávače. Vizualizaci je možno provádět v reálném čase a přepočítáním z historických dat.
Postupujte podle následujících kroků.
- Seznamte se se současnými trendy a nástroji v oblasti distribuovaného výpočtu a zpracování velkých dat v reálném čase (Spark, Kafka, Flink a další).
- Získané poznatky aplikujte při návrhu architektury a následné implementaci prototypu systému.
- Proveďte testování systému na větším množství dat. Data pro výpočet dodá společnost Seznam.cz, a.s., která má pro systém reálné využití.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. prosince 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Analýza trendů ve vyhledávání v reálném čase

Bc. Samuel Butta

Katedra Softwarového inženýrství
Vedoucí práce: Ing. Barbora Červenková DiS.

6. května 2019

Poděkování

Rád bych poděkoval společnosti Seznam.cz, a.s. za poskytnutí dat a výpočetního clusteru. Jmenovitě chci poděkovat své vedoucí práce Barboře Červenkové a Davidu Morávkovi, vedoucímu jednoho z týmů fulltextového vyhledávání, za rady a konzultace.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2019 .....

Odkaz na tuto práci

Butta, Bc. Samuel Butta. *Analýza trendů ve vyhledávání v reálném čase*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Zpracování velkého množství dat je v posledních letech velmi populární a rozvíjející se oblast informatiky. Kromě zpracování velkého množství spíše historických dat, je velmi žádoucí zpracovávat velká data v reálném čase. Tato práce nejprve představuje a popisuje v současnosti používané technologie na zpracování velkých dat a to včetně těch umožnujících zpracování v reálném čase.

Jádrem práce je návrh a implementace prototypu distribuovaného výpočetního systému, který vypočítává dotazy, u kterých došlo k výraznému nárůstu hledanosti jak pro historická data, tak pro data v reálném čase. Součástí práce je také prototyp aplikace, která vizualizuje vypočítané hodnoty.

Klíčová slova Big data, analýza v reálném čase, Apache Hadoop, Apache Spark, Apache Flink, Apache Beam, HDFS, Apache Kafka, Spring Boot, React

Abstract

The processing of Big data has become a very popular and developing area of informatics in the past couple of years. As well as processing a large volume of mostly historical data it is very desirable to process large data in real time. This thesis first introduces and describes the currently used technologies for processing Big data, including those that allow processing in real time.

The core of this thesis is the design and implementation of a distributed computing system prototype, which calculates queries that have registered a significant increase in fulltext web search both for historical data and for data in real time. Included in this thesis is also a prototype of an application, which visualises the computed values.

Keywords Big data, real time analytics, Apache Hadoop, Apache Spark, Apache Flink, Apache Beam, HDFS, Apache Kafka, Spring Boot, React

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Úvod do problematiky

Říká se, že ropou dvacátého prvního století jsou data. Každým okamžikem vzniká ohromné množství nejrůznějších dat, která jsou zaznamenávána a ukládána. S nárůstem počtu lidí využívajících moderní komunikační služby roste i počet dat. Velké množství dat, jako jsou například informace o chování uživatelů, je v současnosti hojně využíváno pro nejrůznější marketingové účely.

V poslední době je velice žádoucí zpracování velkého množství dat v reálném čase. Díky tomu lze například mnohem efektivněji zacílit na uživatele pomocí reklamní kampaně.

Velká data vznikají také při zpracování dotazů na internetový vyhledávač. V Česku mají nezanedbatelný podíl na trhu vyhledávání americká společnost Google a česká společnost Seznam. Je to právě společnost Seznam, která poskytla pro účely této práce vstupní data a výpočetní cluster.

Cíl práce

Cílem práce je vytvořit prototyp distribuovaného výpočetního systému, který dokáže v reálném čase vypočítat, u jakých dotazů došlo k výraznému nárůstu hledanosti. Součástí práce je, mimo samotný výpočetní systém, také tvorba vizualizační části, která zobrazuje vypočítané hodnoty.

Struktura práce

Práce je členěna do sedmi hlavních kapitol. První tři kapitoly jsou věnovány teoretickému úvodu. V první kapitole jsou představeny současné technologie pro zpracování velkých dat.

Ve druhé kapitole jsou podrobněji diskutovány dva druhy zpracování, takzvané batchové a streamové. Ve třetí kapitole je představena a detailněji po-

psána technologie Apache Beam, která byla zvolena pro implementaci distribuovaného systému.

Čtvrtá kapitola se zabývá analýzou a návrhem celého systému, bez implementačních podrobností. V páté kapitole je představena podrobná implementace všech částí systému. Hlavním tématem šesté kapitoly je lokální spuštění a testování. Poslední sedmá kapitola se zabývá nasazením aplikace na cluster a popisuje zajímavé výstupy běhu nad produkčními daty.

Kapitola 1

Technologie velkých dat

V této kapitole představím problematiku spojenou se zpracováním velkých dat neboli *Big data*. Dále popíši fungování modelu MapReduce a představím moderní výpočetní distribuované frameworky, které se v současné době používají.

1.1 Co jsou to data

Informatika je obor, který se z velké části zabývá prací s daty. Pojem data je sice dost obecný, vždy se jedná o určitou informaci, která je nějakým způsobem zaznamenána. Informace může být obsažená na stránce textu, kde je zaznamenána pomocí inkoustu a papíru. V počítačích jsou data uložena v elektronické paměti a jsou reprezentována binárně.

V současné době, tedy v době masivního rozšíření internetu, kdy počet zařízení připojených do globální sítě čítá miliardy osobních počítačů, mobilních telefonů a jiných zařízení, je generováno a ukládáno ohromné množství dat. Objem generovaných dat se podle odhadů každým rokem zdvojnásobí, růst je tedy exponenciální. [9] Tato data mají ve velkém počtu velkou hodnotu. Dají se využít při řešení nejrůznějších analytických a optimalizačních problémů. Hledání zajímavých a skrytých souvislostí v datech se nazývá *Data mining*.

Velká data se také využívají při tvorbě různých modelů technikou strojového učení (anglicky *machine learning*). Všechny tyto oblasti, které jsou v dnešní době velmi populární, dokonce až tak, že se ze zmíněných pojmů staly takzvané *buzzwords*, jsou vzájemně propojeny a tvoří v podstatě jeden celek.

1.2 Velká data

Definice toho, co jsou vlastně *Big data*, není pevně stanovená. Nejčastěji se popisují následujícími vlastnostmi - takzvanými *třemi V*. [10] v různých defi-

nicích se pak přidávají ještě další vlastnosti, tyto tři jsou však základní.

- **Volume** (objem) - Zdrojů pro sběr dat může být mnoho a generují velké objemy.
- **Variety** (různorodost) - Data mohou být v nejrůznějších podobách a jsou z větší části takzvaně nestrukturovaná, to znamená, že důležité informace jako jsou například klíčová slova nebo jména se nevyskytují v předem známých částech. Opakem jsou strukturovaná data, jako příklad lze uvézt klasickou SQL databázovou tabulku.
- **Velocity** (rychlost) - Je požadováno, aby odezva při zpracování byla téměř okamžitě a to ve velké škále.

Obecně by se dalo říci, že množství dat se označuje jako (*Big data*) ve chvíli, kdy dané množství není možné uložit na jedno zařízení. Je tedy patrné, že se tato hranice posouvá s tím, jak se zvyšuje kapacita pamětí počítačů.

Změřit přesně objem uložených dat na světě není možné, odhadem se však jedná o desítky zetabytů (10^{21} bytů). [3]

Velké firmy jako Facebook nebo Google nakládají s ohromným množstvím dat. Je tedy pochopitelné, že technologické hranice posouvají právě takové firmy.

Co se týče firem v České republice, společnost Seznam.cz, a.s. poskytuje vlastní vyhledávač, který je zaměřen především na český trh, přesto však stahuje a ukládá informace k desítkám miliard stránek. To jsou již objemy dat, které mají velikost na úrovni petabytů (10^{15} bytů). Je patrné, že takové objemy dat musí být uloženy na stovkách až tisících počítačích.

1.3 Základní pojmy

1.3.1 Distribuovaný výpočet

Distribuovaný výpočet je jedna úloha, jejíž dílčí části jsou vykonávány na více uzlech. Uzlem se rozumí jeden výpočetní stroj.

1.3.2 Cluster

Cluster je soubor výpočetních uzlů, který tvoří jeden celek. Obvykle jsou propojeny pomocí počítačové sítě.

1.3.3 Škálovatelnost

Jedná se o vlastnost systému, která umožní zvládnutí zvýšené zátěže. To lze zvládnout rozšířením zdrojů, jako jsou například paměť nebo počtu procesorů.

Lze ji rozdělit na dva typy:

- **Horizontální škálování** - přidání více prvků se stejnými zdroji
- **Vertikální škálování** - přidání zdrojů jednomu prvku

Obě dvě metody mají své výhody a nevýhody. V praxi se proto většinou používá horizontální i vertikálí škálování zároveň.

1.3.4 Master/slave architektura

Základní architektonický model, který je společný mnoha distribuovaných systémů, je takzvaný *master/slave* model. Jedná se o komunikační model, kdy jeden prvek, označovaný jako *master*, přebírá kontrolu nad ostatními prvky, které jsou označeny jako pracovní uzly neboli *slave nodes* nebo *worker nodes*.

1.3.5 Webový vyhledávač

Webový vyhledávač (anglicky *web search engine*) je služba, která umožňuje na internetu vyhledávat webové stránky.

1.4 MapReduce model

Technologická firma Google se k problémům spojeným s velkými daty dostávala jako jedna z prvních. Dokladem je to, že již v roce 2003 společnost publikovala článek o vlastním distribuovaném souborovém systému GFS (*Google file system*). [11] O rok později publikovala revoluční článek o výpočetním principu MapReduce. [1]

Ani k jednomu však neposkytla skutečnou implementaci. Proto postupně vznikl open source projekt nazvaný Hadoop.

MapReduce je programovací model pro zpracování a generování velkého množství dat. Myšlenky, na kterých je postavený, jsou staré desítky let a vycházejí z funkcionálního programování. Autoři článku však ukázali, že tento přístup je vhodný při výpočtech nad velkými daty.

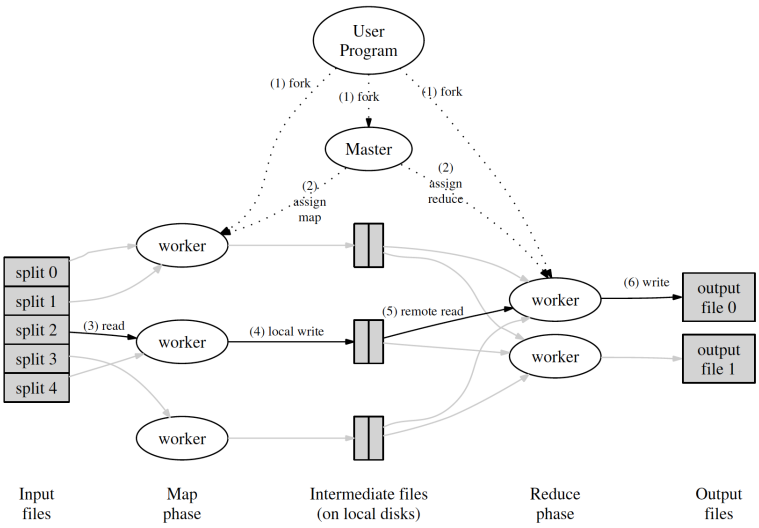
Model je založený na dvou funkcích *map* a *reduce*. Uživatel specifikuje logiku těchto dvou funkcí. První funkce *map* vytvoří data, která jsou ve formátu klíč a jeho hodnota. Tyto dvojice jsou následně zredukovány podle stejného klíče funkcí *reduce*. Příslušný model je natolik silný a obecný, že na něj lze převést mnoho výpočetních úloh. Základní architektura modelu je master/slave.

1.4.1 Podrobný popis průběhu MapReduce

1. Program je rozkopírovaný na uzly v celém clusteru.
2. Vstupní data jsou rozdělena na M částí.
3. Jeden uzel je označen jako master, ostatní jako worker. Worker uzlům master přidělí jednotlivé části dat ke zpracování.

4. Worker si načte příslušná data a aplikuje na ně funkci *map*.
5. Vypočítaná data jsou nejprve v operační paměti, následně jsou po větších kusech ukládána na disk.
6. Informace o uložených datech je předána *master* uzlu. Následně master uzel přiřadí workerům reduce fázi.
7. Reduce fáze začíná seřazením hodnot podle klíče. Následně probíhá iterace přes všechny dvojice. Klíče se stejnou hodnotou jsou spojovány podle funkce reduce.
8. Data po fázi *reduce* jsou zapsána na disky.

Po skončení jsou data typicky zapsána v R souborech. Většinou není nutné spojovat je do jednoho, ani to nemusí být kvůli velikosti možné, protože jsou často vstupem dalšího MapReduce algoritmu. [1]



Obrázek 1.1: Schématické znázornění průběhu MapReduce [1]

Důležitá vlastnost MapReduce frameworku popsaného Googlem je odolnost vůči výpadku uzlu (anglicky *fault tolerance*).

Dále bylo v článku představeno několik úloh, na které lze model aplikovat. Jedná se o následující úlohy. [1]

- Výpočet četnosti slov v daném textu.
- Distribuovaný *grep*, tady hledání částí textu podle určitého vzoru.
- Četnost přístupu na příslušnou url.
- Distribuované řazení.

1.5 Hadoop

Masivně rozšířenou technologií je open source projekt zvaný *Apache Hadoop*, který zahrnuje různé komponenty pro práci s velkými daty.

1.5.1 Historie

Autorem je Doug Cutting, který je také autorem populárního nástroje pro full-textová vyhledávání textu Apache Lucene. Vyvinul se z projektu Apache Nutch, což byl open source *web search engine*, který byl součástí projektu Lucene. Projekt začal v roce 2002. Při vytváření vyhledávače se autoři začali potýkat s problémem, jak ukládat a zpracovávat miliardy stránek. Ve stejné době vyšly oba zmíněné články od společnosti Google, tedy popis distribuovaného souborového systému a popis implementace modelu MapReduce.

Nedlouho poté Nutch vývojáři vytvořili vlastní implementaci. Ukázalo se, že projekt je velice perspektivní a rozšiřitelný na jiné problémy než pouze vyhledávání.

V roce 2006 vznikl tedy na Apache Nutch nezávislý projekt nazvaný Apache Hadoop. Přibližně ve stejné době Cutting nastoupil do společnosti Yahoo!, která začala používat Hadoop ve svém vyhledávacím systému. [3]

1.5.2 Moduly

Základní moduly ekosystému Hadoop jsou následující.

- **Hadoop Common** obsahuje komponenty společné pro více modulů.
- **Hadoop Distributed File System (HDFS™)** je distribuovaný souborový systém.
- **Hadoop YARN** je nástroj pro plánování úloh (anglicky *jobs*) a správce zdrojů clusteru (anglicky *cluster resource management*).
- **Hadoop MapReduce** je framework pro paralelní zpracování, open source implementace modelu MapReduce.

1.5.3 Hadoop Distributed File System

HDFS je distribuovaný souborový systém, který je klíčovým prvkem při práci s velkými daty. Je navržený tak, aby mohl běžet na běžně dostupném a levném hardwaru (anglicky *commodity hardware*). [2]

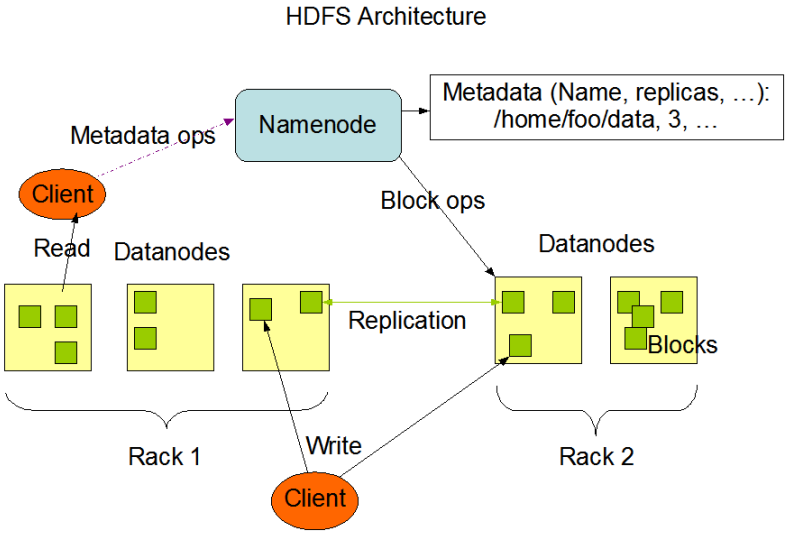
1.5.4 Základní vlastnosti HDFS

- **Odolnost vůči selhání hardwaru** - Vzhledem k tomu, že cluster je tvořen stovkami nebo tisíci uzlů, existuje nemalá pravděpodobnost, že některé uzly vypadnou. Návrh HDFS s tímto problémem počítá.

- **Velké soubory** - HDFS počítá s ohromnou velikostí souborů, což jsou soubory v řádu desítek gigabytů.
- **Jednoduchý koherentní model zápisu a čtení** - HDFS je postavený na myšlence zapiš jednou, čti opakovaně (anglicky *write-once-read-many access*)
- **Nízká prodleva přístupu k souboru**

1.5.5 Popis HDFS

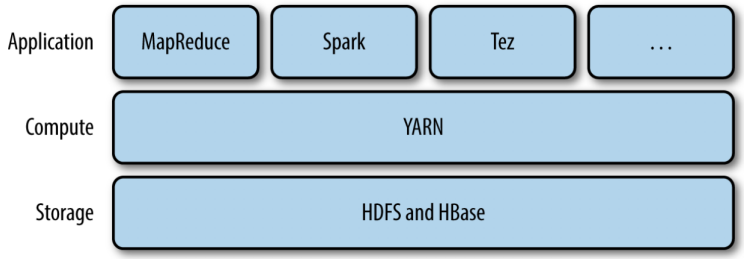
HDFS je postaveno na master/slave architektuře. Master uzel se nazývá *name node* a udržuje informace, na kterých uzlech jsou data uložena, a spravuje přístup klientů k systému. Samotná data jsou uložená na datových uzlech (anglicky *date nodes*). Soubor je rozdělen na bloky. Tyto bloky jsou uloženy na různých datových uzlech.



Obrázek 1.2: Architektura HDFS [2]

1.5.6 Hadoop YARN

Zkratka YARN znamená *yet anothor resource negotiator*. Hlavním důvodem vzniku bylo oddělit správce zdrojů (anglicky *resource manager*) a plánování úloh (anlicky *job scheduling*) do samostatného modulu. Resource manager je komponenta, která má na starosti správu zdrojů v clusteru. Zdroje jsou jmenovitě operační paměť, výpočetní čas na procesoru (anglicky *CPU time*), disk a síť. YARN poskytuje API, které nabízí přístup ke zdrojům. Toto API zpravidla nevolá uživatel, který píše kód aplikace, ale je volán nepřímo skrze distribuovaný framework. [3]



Obrázek 1.3: Umístění YARN v celkové architektuře [3]

1.6 Distribuované výpočetní frameworky

Výpočetní framework neboli *processing framework* a výpočetní jádro neboli *processing engine* jsou dva termíny, které nemají ostře vymezenou hranici a často se zaměňují. Například v kontextu Apache Hadoop je celý samotný Hadoop často nazývaný jako výpočetní framework, zatímco část MapReduce se nazývá výpočetní jádro. Na druhou stranu Apache Spark je nazýván výpočetním frameworkem, ale i výpočetní jádrem. [12] V této práci jsou tedy oba termíny kvůli neostré hranici považovány za totožné a spíše je používán termín výpočetní framework.

Výpočetní framework je program, který dokáže výpočetně operovat nad velkým množstvím dat. Existují dva základní druhy zpracování dat - *batchové* a *streamové* zpracování.

Podrobněji se těmto pojmům věnuje následující kapitola. Pro tuto chvíli stačí poznamenat to, že batchové zpracování je spojeno s výpočty nad uzavřenou množinou dat, dá se říci nad historickými daty. Zatímco streamové zpracování operuje nad potenciálně nekonečnou množinou vstupních dat a je spojeno s výpočty v reálném čase.

Frameworky lze rozdělit do tří kategorií podle typu zpracování.

- **Pouze batchové zpracování** - zástupcem v této kategorii je například Hadoop MapReduce

- **Pouze streamové zpracování** - příkladem v této kategorii je systém Apache Storm nebo Apache Samza
- **Batchové i streamové zpracování** - jako příklad lze uvézt Apache Spark nebo Apache Flink

Streamové zpracování lze rozdělit na dva druhy podle toho, kolik prvků systém naráz zpracovává. [12]

- ***True streaming* neboli opravdu streamové** zpracovává jeden prvek. Příkladem takového systému je Apache Flink.
- ***Micro-batch streaming* neboli streaming po malých dávkách** zpracovává více prvků. Příkladem takového systému je Apache Spark.

1.6.1 Apache Spark

Apache Spark je dnes již vyspělá technologie, která je celosvětově hojně používaná. Jedná se o víceúčelový distribuovaný výpočetní systém, který je implementován v jazyce Scala, ale poskytuje rozhraní pro jazyky Java, Scala, Pyhton a R. Systém vznikl v roce 2009 na univerzitě Berkley, kdy kolektiv autorů vydal článek nazvaný *Spark: Cluster Computing withWorking Sets*. [13]

MapReduce model zaznamenal ohromný úspěch a byl vhodný na řešení mnoha problémů. Existuje však řada problémů, na které příliš vhodný není. Pokud výpočet probíhá ve více iteracích opakovaně, trvá na Hadoop MapReduce velmi dlouho. Primárně je to způsobeno tím, že výstupy jedné fáze výpočtu jsou zapsány na disky a před zahájením další iterace se musí data opět načítat. Příkladem můžou být různé algoritmy strojového učení, které jsou založené na postupném zlepšování nalezeného řešení. Již v úvodním článku autoři uvedli, že Apache Spark je 10x rychlejší pro algoritmy strojového učení. [13] Spark měl také za cíl poskytnout interaktivní rozhraní pro jednorázové analytické operace nad daty takzvané *ad hoc* dotazy.

1.6.1.1 Architektura Apache Spark

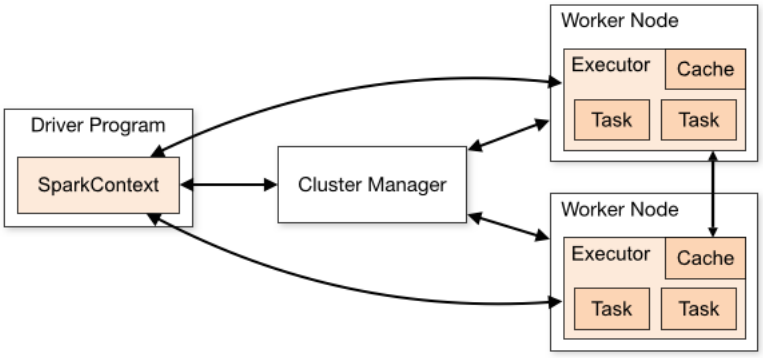
Aplikace pro Spark běží na uzlech clusteru jako nezávislé procesy, které jsou řízeny objektem třídy *SparkContext* reprezentující kontext systému. Klient vytváří takzvaný řídící neboli *driver* program, v rámci kterého vytvoří instanci třídy *SparkContext* a definuje výpočetní logiku. [4]

Průběh výpočtu na Apache Spark vypadá následovně.

1. Instance třídy *SparkContext* se připojí ke správci zdrojů cluster manageru. Může se jednat o různé druhy, například YARN nebo Mesos.
2. Cluster manager přidělí řídícímu programu uzly, které vykonají kód zvané executory.

3. Na executory je odeslán program, který mají vykonat (například JAR soubor).

4. Nakonec *SparkContext* odešle instrukce, jakou operaci a nad jakými daty mají executory vykonat.



Obrázek 1.4: Architektura Apache Spark [4]

1.6.1.2 RDD

Spark přišel s abstrakcí nad daty zvanou *resilient distributed dataset* neboli RDD. Jedná se o distribuovanou množinu dat, která je odolná vůči ztrátě. Odolnost je zajištěna tím, že při ztrátě určité části jsou data automaticky dopočítána a obnovena.

1.6.1.3 Základní pojmy Apache Spark

- **Partition** je část distribuované množiny dat uložená na jednom uzlu.
- **Worker node** je jakýkoliv uzel clusteru, který může spouštět kód uživatele.
- **Executor** je proces spuštěný na uzlu clusteru, který provádí výpočty a ukládá data do paměti nebo na disk. Každá aplikace má vlastní executory.
- **Shuffle** je proces výměny dat mezi jednotlivými výpočetními uzly. Výměna probíhá po síti a z toho důvodu se jedná o relativně pomalou operaci.
- **Task** je nejmenší jednotka práce, která může být poslána na jeden executor.

- **Job** je spojení více tasků.
- **Stage** je to množina tasků, kterou lze vykonat na jednom executoru bez nutnosti výměny dat.

1.6.1.4 Apache Spark streaming

Apache Spark streaming je rozšíření jádra Spark API, které umožňuje škálovatelné zpracování dat v reálném čase. Základní jednotkou abstrakce je takzvaný DStream, který je složený z jednotlivých RDD objektů. [14]

Systém funguje tím způsobem, že vstupní datový tok postupně střádá do malých dávek (anglicky *micro batches*) a po určitém čase tuto dávku zpracuje.

1.6.2 Ostatní frameworky

Na popularitě poslední dobou nabírá framework Apache Flink, který patří do kategorie true stremaing systémů. Existuje celá řada různých distribuovaných výpočetních frameworků, které je dobré pro úplnost zmínit. Jedná se například o frameworky Apache Storm, Apache Apex, Apache Samza nebo Google Dataflow. Pro tuto práci však nejsou zvlášť důležité a proto nejsou podrobněji probírány.

1.7 Ostatní technologie velkých dat

1.7.1 Apache Zookeeper

Apache Zookeeper je distribuovaná koordinační služba (anglicky *coordination service*) pro distribuované aplikace. [15] Vystavuje rozhraní, které mohou distribuované aplikace využívat a odebírá jim samotným zodpovědnost za některé činnosti, jako je například synchronizace. Komponenta je napsaná v jazyce Java a je zde uvedená z toho důvodu, že její přítomnost je nutná pro běh Apache Kafka a jiných distribuovaných aplikací.

1.7.2 Distribuované databáze

Pro tuto práci nejsou distribuované databáze klíčovým pojmem, proto je zmíním pouze okrajově. Při uložení velkého množství dat nejsou klasické relační SQL databáze nejvhodnější, především kvůli nesnadné škálovatelnosti.

Velkou popularitu získala rodina *NoSQL* databází. Do této rodiny spadá také kategorie takzvaných *Big Table* databází, které jsou stavěné k ukládání miliard záznamů. Příkladem takových databázových systémů je Apache HBase nebo Apache Cassandra.

Kapitola 2

Batchové a streamové zpracování

V této kapitole představím rozdělení distribuovaných výpočtů podle vlastnosti vstupních dat. Popíši, v čem se liší a představím způsoby jejich zpracování. Dále představím a blíže popíši transportní technologii Apache Kafka, která se používá při streamovém zpracování.

2.1 Batchové a streamové zpracování

Zpracování dat lze rozdělit do dvou základních typů, podle povahy vstupních dat. První typ je **batchové zpracování** (v českém překladu *"dávkové zpracování"*). Množina vstupních dat je uzavřená.

Druhý typ je **streamové zpracování** (v českém překladu *"proudové zpracování"*), kde počet prvků není ukončen a v čase přibývají nové záznamy. Streamové zpracování je spojeno s analýzou v reálném čase (anglicky *"real time analysis"*).

Analýza v reálném času je z pohledu obchodního využití velice cenná. Pokud je například systém na základě aktuálního, ale i historického chování uživatele schopný rozpoznat, o jaký produkt má právě teď uživatel zájem, může mu být dynamicky nabídnut a tím výrazně zvýšit pravděpodobnost jeho nákupu. Systémy reálného času nejsou samozřejmě spojené pouze s oblastí velkých dat. Velká data do zpracování však vnášejí zajímavé problémy.

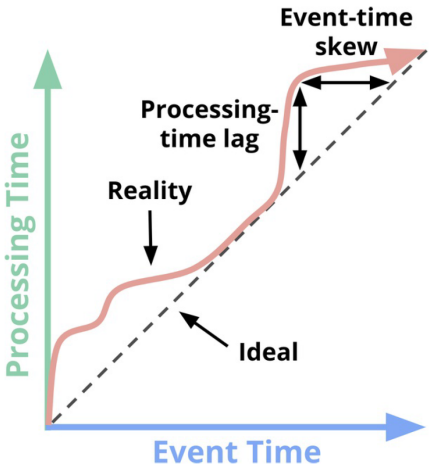
2.1.1 Základní terminologie

Ve světě informačních technologií, je vývoj velmi dynamický. Mnoho pojmů, které se stanou populární a často používané, vlastně nejsou dost dobře definované a každý si pod daným pojmem představí něco jiného. Proto se nejprve pokusím přesně definovat pojmy spojené se streamovým zpracováním. Primárním zdrojem pro jejich definici je kniha *Streaming Systems*. [5]

- **Omezená datová sada** nebo také **datová sada typu bounded** je množina hodnot, která je uzavřená. Tato datová sada je vždy konečná.
- **Nemezená datová sada** nebo také **datová sada typu unbounded**, tedy množina hodnot, která je potenciálně nekončená. Přestože v určitém čase je množina samozřejmě vždy konečná, data do ní stále "přitékají".
- **Streamový systém** je typ výpočetního systému (anglicky *data processing engine*), který dokáže zpracovávat neomezenou datovou sadu.

Při streamovém zpracování je důležité rozlišit dva základní druhy času spojeného s daty.

- **Čas vzniku události** neboli **event time** je čas, kdy se daná událost skutečně stala. A kontextu fulltextového vyhledávání lze uvézt příklad, že se jedná o čas, kdy uživatel zadal nějaký dotaz.
- **Čas zpracování** neboli **processing time** je čas, kdy se systém o události dozví a prvek zpracuje.



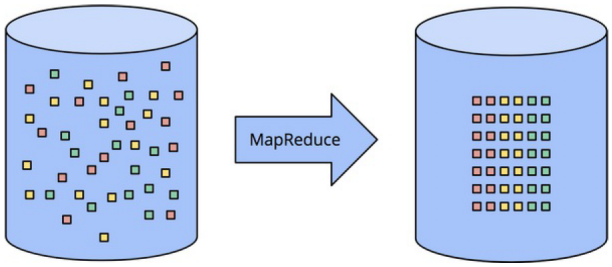
Obrázek 2.1: Rozdíl mezi časem vzniku události a časem zpracování [5]

V ideálním případě by se čas, kdy událost skutečně nastala, a čas, kdy ji systém zpracoval, rovnal. To však kvůli omezením reálného světa není možné a je vždy určitý rozdíl mezi těmito časy. Tento rozdíl může být relativně malý, ale může být i v řádu hodin nebo dní. Například mobilní aplikace, která dočasně nemá připojení k internetu, synchronizuje události až výrazně později. Systém, který data zpracovává, tedy neví, že nějaká událost nastala. Z tohoto plynou určité důsledky pro streamové zpracování. Jedním z důsledků je, že výsledky, které systém poskytuje, nejsou přesné.

2.2 Způsoby zpracování

2.2.1 Batchové zpracování

Batchové zpracování je v celku přímočaré. Na počátku jsou vstupní data, která jsou zpracována nějakým z výpočetních frameworků, například MapReduce, a převedena na výstupní data. Nejznámějším příkladem může být výpočet četnosti slov v textu takzvaný *Wordcount*. Situaci zachycuje následující obrázek.

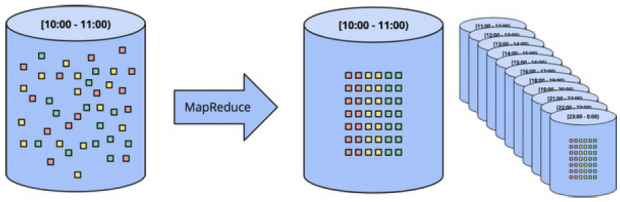


Obrázek 2.2: Znázornění zpracování datové sady typu *bounded* [5]

2.2.2 Unbounded data pomocí batch systémů

Systémy navržené primárně pro batchové zpracování dost dobře nepočítaly s potenciálně nekončeným vstupem dat. Nejpřímějším způsobem, jak datovou sadu typu unbounded zpracovat, je vzít konečnou podmnožinu dat pro určitý časový interval a postupně tento interval posouvat v čase. Tato metoda se nazývá *windowing*.

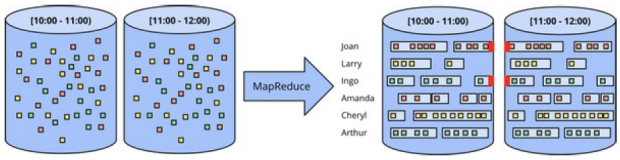
Jeden z typů strategie windowing je pomocí fixního okénka neboli *fixed window*. V pravidelném intervalu je tedy spuštěn batchový výpočet. Pro mnoho případů je tento způsob vhodný. Pokud se však data opožďují a nejsou k dispozici při zahájení výpočtu, dochází k velkým nepřesnostem.



Obrázek 2.3: Znázornění zpracování datové sady typu *unbounded* pomocí strategie *fixed window* [5]

Pokročilejší strategie, která se snaží udržovat kontext dat mezi jednotlivými intervaly, se nazývá *session windowing*. Jako příklad lze uvézt výpo-

čet, který měří aktivitu, například počet prokliknutých produktů na webové stránce pro daného klienta. Při použití strategie fixed windowing se kontext uživatele přeruší, přestože stránku neopustil. Proto je dobré předat vypočtenou informaci i do dalšího běhu.



Obrázek 2.4: Znázornění zpracování datové sady typu *unbounded* pomocí strategie *session window* [5]

2.2.3 Zpracování unbounded datasetu pomocí streaming systémů

V současné době existují distribuované systémy, které již byly navrhované primárně pro zpracování neomezené datové sady. Jako příklad lze uvézt Apache Storm nebo Apache Flink. Tyto systémy si musí být schopné poradit nejenom s neomezeností vstupních dat, ale také s následujícími problémy.

- Přicházející data jsou obecně silně neuspořádaná.
- Funkce popisující rozdíl mezi časem vzniku a časem zpracování není lineární. Nelze tedy s jistotou předpokládat, že za určitou dobu již bude mít systém všechna data.

Existují určité přístupy, které se s výše uvedenými problémy snaží vypořádat.

První přístup je takzvaný **časově agnostický** (anglicky *time-agnosting*), což znamená, že na čase během zpracování vůbec nezáleží. Tím odpadá mnoho starostí a výše zmíněné zpracování pomocí batchového systému je naprosto dostačující. Příkladem může být jednoduchá filtrace. Například filtrace logů pro daného uživatele.

Druhým přístupem jsou **aproximační algoritmy**. Jedná se o přibližné algoritmy. Například nejčetnějších N prvků (anglicky *approximate top-N*).

2.2.4 Windowing

Přístup zvaný *windowing* již byl částečně popsaný výše. Jde o techniku, kdy se vstupní data rozdělí na části podle času. Lze zvolit jak čas vzniku události tak čas zpracování systémem.

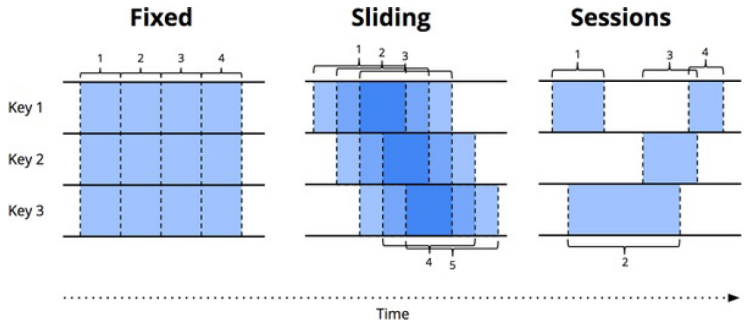
Strategii *windowing* lze rozdělit do tří základních typů.

- *fixed windowing*
- *sliding windowing*
- *session windowing*

Fixed windowing a *session windowing* byl diskutován výše, proto je zde popsát pouze *sliding windowing*.

2.2.5 Sliding window

Posuvné okénko je v podstatě zobecněním fixního okénka. Velikost okénka a doba, o kterou se okénko posouvá, nemusí být stejná. Výpočet je například spouštěný každých patnáct minut, ovšem velikost okénka je nastavená na jednu hodinu. Z toho vyplývá, že se data, která jsou zpracovávána, se překrývají. Posuvné okénko však umožňuje získávat přesnější výsledky.



Obrázek 2.5: Znázornění různých druhů strategie typu *windowing* [5]

V této sekci jsem představil obecné principy, které jsou více méně společné všem výpočetním frameworkům. V následující části se zaměřím na podporu streamového zpracování u konkrétních technologií.

2.3 Apache Kafka

Apache Kafka je důležitá technologie při streamovém zpracování dat. Jedná se o distribuovanou streamovou platformu (anglicky *distributed streaming platform*), která je založená na modelu *Publish/Subscribe* [7] a v principu je podobný datové struktuře fronta a na systému zasílání zpráv (anglicky *messaging system*). Jedná se o tedy o transportní vrstvu. Apache Kafka se používá pro dva základní typy problémů.

- Tvorba takzvaných *real-time pipelines*, tedy komponenty pro přenos dat v reálním čase, mezi různými systémy a aplikacemi.
- Tvorbu takzvaných *real-time streaming applications*, tedy aplikací v reálním čase, které trasformují nebo jiným způsobem reagují na tok dat.

Apache Kafka beží na clusteru, který se skládá z jednoho a více uzlů, které mohou být na různých místech, tedy v různých datových centrech.

Kafka ukládá tok záznamů (anglicky *stream of records*). Záznamy (anglicky *records* nebo *messages*) jsou vždy vztažené k právě jedné kategorii, která se nazývá *topic*. Jeden záznam se skládá z klíče, hodnoty a časové známky (anglicky *timestamp*). Záznamy nemají z pohledu systému žádný význam, protože na ně nahlíží pouze jako na pole bytů. Všechny záznamy jsou trvale ukládány na disky (anglicky *durably persists*), parametry ukládání jako je například doba, po kterou jsou data uložena, jsou konfigurovatelné.

2.3.1 Základní pojmy Apache Kafka

2.3.2 Topic

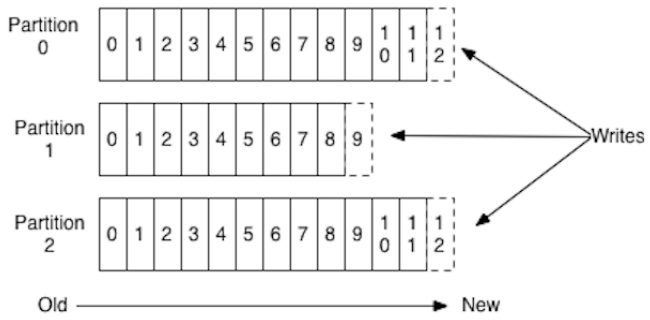
Topic je kategorie, do které je záznam zařazen. Jako příklad lze uvést topic *error-log-topic*, který shromažďuje chybové logy z aplikace. Topic je takzvaně *multi-subscribe*, to znamená, že může mít více odběratelů.

2.3.3 Partition

Topic je rozdělen do oddílů (anglicky *partitions*). Oddíl je uložen na jednom fyzickém stroji. Aby bylo zachováno uložení dat i při výpadku jednoho uzlu, jsou tyto oddíly uloženy na více strojích. Každý oddíl je uspořádaná, neměnná (anglicky *immutable*) posloupnost záznamů, které postupně přibývají. Každý záznam v rámci jednoho oddílu je jednoznačně identifikován svým posunem (anglicky *offset*).

2.3.4 Broker

Apache Kafka běží na clusteru. Jeden uzel clusteru se nazývá *broker*.

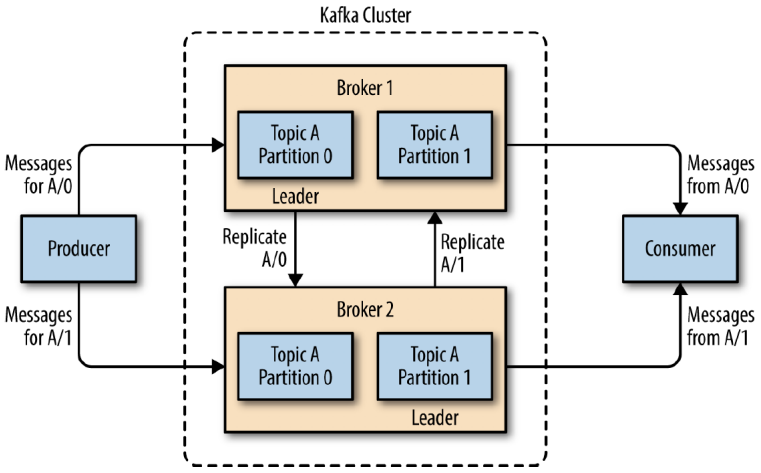


Obrázek 2.6: Apache Kafka topic rozdělený do oddílů [6]

2.3.5 Producent a Konzument

Kafka rozlišuje dva typy uživatelů - producenty a konzumenty. [7] Producent zapisuje data do Kafky, zatímco konzument data čte. Konzumentem může být distribuovaná aplikace, která provádí výpočty a analýzu nad příslušnými daty. [6]

Jediná informace, kterou si ke konzumentovi Kafka pamatuje, je jeho aktuální offset, tedy pozice, na které data čte. Offset je jinak plně kontrolovaný konzumentem, může tedy číst data téměř v libovolném pořadí. Z tohoto důvodu nejsou konzumenti pro systém nijak zvlášť nároční.



Obrázek 2.7: Schéma Kafka clusteru [7]

Kapitola 3

Apache Beam

V této kapitole podrobněji představím klíčovou technologii pro tuto práci, protože jsem ji zvolil jako nástroj při implementaci. Jedná se o poměrně novou záležitost na poli distribuovaných výpočtů, jejíž první verze vyšla v roce 2016. [16] Apache Beam je open source nástroj, který vytváří jednotný model a rozhraní pro různé distribuované frameworky, jedná se o vyšší vrstvu abstrakce. Apache Beam nabízí jednotný model jak pro batchové, tak streamové zpracování.

SDK Apache Beam má momentálně podporu pro jazyky Java, Go a Python. Program napsaný v Beam SDK lze spustit na libovolném distribuovaném frameworku, který je podporovaný. Jedná se například o Apache Apex, Apache Flink, Apache Spark, nebo Google Cloud Dataflow. [17]

3.1 Základní koncepty

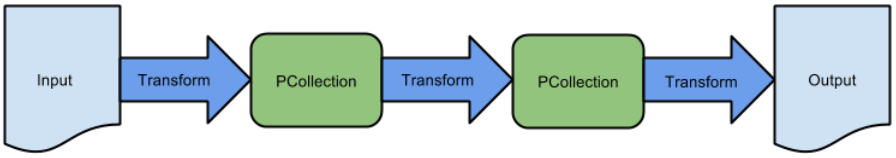
Program v Apache Beam, který se vytváří v jednom z jeho SDK, se nazývá *driver program*. Abstrakce je celkem intuitivní a blízká reálnému světu. Jedná se o potrubí, kterým protékají data. Tato data se v průběhu toku mění. Základní pojmy jsou *Pipeline*, *PCollection*, *PTransformation*. [8]

- **Pipeline** je objekt, který zapouzdřuje celý paralelní výpočet.
- **PCollection** neboli *paralelní kolekce* reprezentuje distribuovanou generickou množinu dat. Dalo by se říci, že tyto kolekce "protékají" definovaným "potrubím".
- **PTransform** je operace, která zpracovává data. Jedná se o jeden krok v datové *pipeline*.
- **Runner** je objekt, pomocí něhož se definuje, pod jakým frameworkem aplikace běží.

Typický driver program se skládá z následujících kroků.

1. Vytvoření instance třídy *Pipeline*.
2. Definování počáteční kolekce, kterou lze načíst z různých zdrojů jako je například externí distribuovaný souborový systém.
3. Vytvoření transformací, tedy jednotlivých kroků.
4. Zapsání výsledku (výsledná požadovaná kolekce může být opět zapsána například na souborový systém)
5. Spuštění programu nad výpočetním frameworkem pomocí konkrétní třídy *Runner*

Jednotlivé transformace si lze představit jako uzly v grafu. Výsledný graf může být velice komplexní.



Obrázek 3.1: Ukázka lineární Apache Beam pipeline [8]

3.1.1 Vytvoření Pipeline

Nejprve je nutné vytvořit instanci třídy *Pipeline*. Program je zpravidla konfigurovatelný různými parametry. Tyto parametry zastřešuje instance třídy *PipelineOptions*, která je předána jako parametr konstruktoru při vytvoření objektu *Pipeline*.

Instance třídy *PipelineOption* může obsahovat například informaci, pod jakým distribuovaným frameworkem se má program spustit, nebo cestu k souborům se vstupními daty. Apache Beam nabízí možnost nastavit tyto hodnoty pomocí parametrů příkazové řádky.

```
1 PipelineOptions options = PipelineOptionsFactory.create();
2
3 Pipeline p = Pipeline.create(options);
```

3.1.2 PCollection neboli paralelní kolekce

PCollection představuje distribuovanou kolekci. Celou *Pipeline* protékají paralelní kolekce různých typů. Ve chvíli, kdy je vytvořena *Pipeline*, je nutné načíst data a převést je na *PCollection*, tedy objekt, kterému Beam rozumí. Beam SDK poskytuje relativně velkou podporu pro načítání dat z různých zdrojů. Jedním ze základních zdrojů je textový soubor uložený na distribuovaném souborovém systému. Tento zdroj je použit v následující ukázce.

```
1 PCollection<String> lines =
    p.apply(TextIO.read().from("path/to/inputData.txt"));
```

Je možné také vytvořit PCollection v paměti (anglicky *in-memory data*), což je vhodné k jednoduchému testování.

3.1.2.1 Základní vlastnosti PCollection

- Kolekce může být omezená (anglicky *bounded*) i neomezená (anglicky *unbounded*)
- Kolekce patří právě jednomu objektu *Pipeline*, není ji tedy možné sdílet.
- Datový typ kolekce může být libovolný, je však nutné, aby mohl být zakódovaný do pole bytů (anglicky *byte string*), kvůli posílání jednotlivým uzlům.
- PCollection je neměnná (anglicky *immutable*). To znamená, že jednou vytvořená kolekce již nemůže být modifikována. Při transformacích tedy vznikají vždy nové kolekce.

Apache Beam používá techniku *windowing* pro zpracování neomezeného datového vstupu. Každý prvek v kolekci má svojí časovou známku (anglicky *timestamp*), která je přiřazena při vytváření kolekce zdrojem.

3.1.3 Transformace

Transformace definují operace nad kolekcemi. Z matematického pohledu je vhodnou abstrakcí transformace funkce respektive zobrazení. Jedná se o generické objekty, které jsou parametrizovány funkcemi, které vytváří klient (anligky *user code*).

3.1.4 Aplikování transformace

Instance třídy *PCollection* poskytuje metodu *apply*, jejíž parametr je objekt *PTransform*. Metodu *apply* lze zřetězit. Tato metoda vrací novou *PCollection*, jejíž typ předepisuje transfornace.

```
[Final Output PCollection] = [Initial Input PCollection].apply([First
    Transform])
.apply([Second Transform])
.apply([Third Transform])
```

Vzhledem k tomu, že vstupní kolekce nejsou nijak modifikované, ale vždy vznikají nové, je možné na data aplikovat různé transformace a tím graf rozvětvit.

3.1.5 Základní transformace

Apache Beam poskytuje šest základních transformací. Mimo ně poskytuje i velkou škálu specifickým, často používaných transformací, které jsou ovšem na základních transformacích postaveny.

3.1.5.1 ParDo

Je obecná paralelní operace. Má blízko k *map* fázi v MapReduce modelu. *ParDo* aplikuje na každý prvek v paralelní kolekci určitou funkci definovanou uživatelem. Výstupní hodnotu přidává do výstupní kolekce. Může emitovat nula až n hodnot. Typickým využitím *ParDo* transformace je filtrování hodnot, formátování nebo typová konverze.

Transformace přijímá jako parametr instanci třídy *DoFn*. Jedná se o třídu Beam SDK, která definuje distribuovanou funkci.

Použití je patrné z následující ukázky, která zachycuje příklad, kdy transformace z počáteční kolekce slov počítá délku každého slova.

```
PCollection<String> words = ...;

static class ComputeWordLengthFn extends DoFn<String, Integer> { ... }

PCollection<Integer> wordLengths = words.apply(
    ParDo
    .of(new ComputeWordLengthFn()));
```

Uživatelem definovaná funkce rozšiřuje třídu *DoFn*. Třída má dva typové parametry, které učují typ vstupní a výstupní hodnoty. Samotná logika se děje v metodě označené anotací *ProcessElement*.

```
static class ComputeWordLengthFn extends DoFn<String, Integer> {
  @ProcessElement
  public void processElement(@Element String word,
      OutputReceiver<Integer> out) {
 output element.
    out.output(word.length());
  }
}
```

3.1.5.2 GroupByKey

GroupByKey je transformace, která se používá při zpracováni dvojic klíč hodnota. Je podobná fázi *reduce* v modelu MapReduce. Transformaci je vhodné použít pro agregaci dat, které mají něco společného. Klíčem může být například URL webové stránky a hodnotu počet přístupů na stránku. Pro spočítání všech přístupů na danou stránku je použití této transformace vhodné. Vstupem je kolekce klíčů a jejich hodnot, výstupem je unikátní klíč a všechny hodnoty shluknuté do jednoho objektu typu *Iterable*.

3.1.5.3 CoGroupByKey

Transformace *CoGroupByKey* se používá ve chvíli, kdy je potřeba sloučit dvě kolekce typu klíč hodnota, které mají stejné klíče a různé hodnoty.

3.1.5.4 Combine

Combine je transformace, která podle nadefinované funkce určitým způsobem zkombinuje hodnoty. *Combine* lze použít na celou kolekci, případně na kolekci dvojic klíč hodnota.

Jako příklad použití lze uvézt situaci, po provedení transformace *GroupByKey* vznikla kolekce unikátních URL adres, které jsou klíče a hodnota je kolekce (typu *Iterable*) obsahuje počet uživatelů pro daný den. Trasformaci Combine lze například využít při výpočtu průměrné denní návštěvnosti.

Apache Beam má mnoho *Combine* funkcí, které jsou často používány, ve svém SDK naimplementované.

3.1.5.5 Flatten

Transformace *Flatten* umožňuje spojit více kolekcí. Slučované kolekce musí být stejného typu. Použití demonstruje následující ukázka, v níž dojde ke spojení dvou kolekcí typu String (textový řetězec), může se jednat například o jména uživatelů.

```
1 PCollection<String> names1 = ...;
2 PCollection<String> names2 = ...;
3 PCollectionList<String> collections =
      PCollectionList.of(names1).and(names2);
4
5 PCollection<String> merged =
      collections.apply(Flatten.<String>pCollections());
```

Tímto způsobem lze spojit více kolekcí než pouze dvě zřetězením metody *and*.

3.1.5.6 Partition

Opačnou transformací k transformaci *Flatten* je operace *Partition*. *Partition* umožňuje rozdělit kolekci na více menších kolekcí. Rozdělení se řídí funkcí, která je předána jako parametr tranformace.

3.2 Batchové a streamové zpracování v Apache Beam

Apache Beam je navržený tak, aby sjednocoval pohled na batchové a streamové zpracování. To lze odvodit i z názvu, protože se jedná o spojení těchto dvou slov: *Beam = Batch + Stream*.

Logika výpočtu zůstává stejná, ovšem pro správné fungování streamového zpracování je nutné správně nastavit dvě vlastnosti.

- **Nastavit velikost okna**, protože ve výchozím stavu je nastaveno takzvané globální okno. To znamená, že existuje pouze jedno okno, které obsahuje všechny prvky.
- **Nastavit správnou hodnotu času, kdy k události došlo**, protože výchozí hodnotu, kterou systém přiřadí, je čas, kdy systém prvek zpracoval, a nikoliv skutečný čas, kdy k události došlo.

3.2.1 Windowing

Rozdělení podle časové známky do určitých skupin se nazývá *windowing*. Vzhledem k tomu, že datové transformace jako *GroupBy* nebo *Combine* agregují prvky kolekce a v případě potenciálně nekonečné datové sady není možné mít všechny prvky, využívá Beam pro zpracování metodu windowing, jejíž parametry lze různým způsobem nastavit.

3.2.2 Trigger

Pomocí objektu *trigger* neboli spouštěče se definuje, kdy má dojít k emisi dat.

Trigger lze nastavit mnoha způsoby. Výchozí nastavení je takové, že k emisi dat dojde ve chvíli, kdy hodnota funkce zvané *wattermark* překročí čas konce okna.

Kapitola 4

Analýza a návrh systému

V této kapitole nejprve představím samotnou doménu, kterou je vyhledávání na internetu pomocí webových vyhledávačů. Dále definuji funkční a nefunkční požadavky, které má implementovaný systém splňovat. V poslední části této kapitoly se zaměřím na obecný návrh celé aplikace.

4.1 Úvod do problematiky

Internet, tedy celosvětová síť propojených počítačů, umožnil vznik *World wide webu*. Přestože jsou běžnou veřejností tyto pojmy zaměňovány, nejsou stejné.

WWW je v podstatě síť dokumentů, které jsou vzájemně propojeny. Formát dokumentu je *Hypertext Markup Language* neboli HTML. Propojení dokumentů je zajištěno pomocí hypertextových odkazů.

V terminologii teorie grafů by dokument reprezentoval uzel a orientované hrany by byly odkazy. Dokument respektive stránka musí mít jednoznačný identifikátor, aby k ní mohl uživatel přistoupit. Tímto identifikátorem je veřejná IP adresa. IP adresa je poměrně dlouhé číslo, které lze těžko zapamatovat. Z toho důvodu vznikly slovní identifikátory neboli domény.

Při průchodu webem je tedy nutné znát domény. Nejprve vnikaly různé katalogy stránek, které obsahovaly seznamy odkazů. Obsah stránek se však mohl měnit a uživatel se chtěl přímo dostat k nějakému dokumentu s jemu užitečnou informací. Logickým požadavkem proto bylo přímo vyhledávání v dokumentech.

4.1.0.1 Fulltextové vyhledávání

Fulltextové vyhledávání je hledání v dokumentu podle hledané fráze napříč celým textem, jak napovídá překlad.

Pro malé dokumenty není obtížné projít celý text slovo po slově podle hledané fráze. Pro větší dokumenty již tento jednoduchý postup kvůli výpočetní náročnosti není možný.

Dokument je proto předzpracován tzv. *zaindexován*. Tím vznikne speciální struktura zvaná *index*, v které lze velice rychle vyhledávat.

Funkcí internetového vyhledávače je poskytnout fulltextové vyhledávání nad webovými stránkami ideálně na celém internetu.

4.1.1 Internetové vyhledávače

Za první vyhledávač lze považovat Archie Search engine, který lze zařadit ještě do kategorie *pre-web*, tedy před představením WWW. Archie byl založen na protokolu FTP. [18]

Od té doby postupně vznikla řada různých firem. V současné době má téměř dominantní postavení na globálním trhu vyhledávání společnosti Google. Česká republika je jedna z mála zemí, která má svojí lokální alternativu v podobě vyhledávače od společnosti Seznam.cz, který udržuje relativně velké procentu v celkovém počtu hledání v České Republice. [19]

4.2 Základní funkce systému

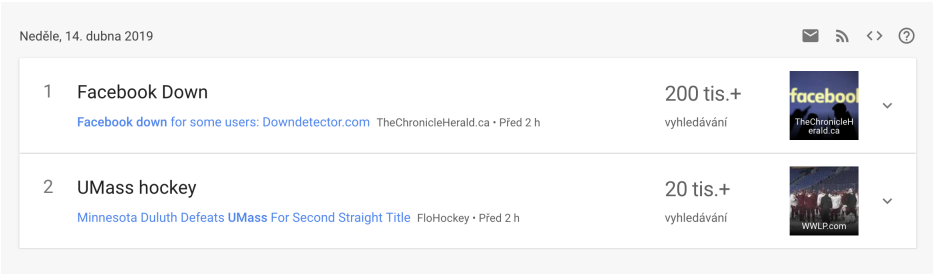
Internetový vyhledávač Seznam každou sekundu zpracovává stovky až tisíce nejrůznějších dotazů. Tato data nesou mnohé informace. Systém, který je jádrem této práce, umožňuje v reálném čase získávat informace o tom, které dotazy jsou takzvaně *trendující*. **Trendující dotazy** jsou dotazy, u kterých došlo k výraznému nárůstu hledanosti.

4.3 Analýza současných systémů

4.3.1 Google Trends

Google provozuje již od roku 2006 stránku http://trends.google.com. Stránka obsahuje velké množství různých grafů a porovnání hledaných slov.

Nabízí například sekci *Nedávné trendy*, která zobrazuje témata, u kterých za posledních 24 hodin výrazně vzrostla hledanost. Tato data jsou aktualizována každou hodinu.



Obrázek 4.1: Ukázka denních trendů v aplikaci Google Trends

V kontextu této práce je zajímavé, že obsahuje také sekci *Trendy ve vyhledávání v reálném čase*. V popisu této sekce je přímo napsáno: *"Trendy ve vyhledávání v reálném čase se zaměřují na články, které byly ve službách Google oblíbené za posledních 24 hodin a jsou aktualizovány v reálném čase. Tyto články představují sbírku tvořenou tématy z Diagramu znalostí, zájmem ve Vyhledávání, populárními videi YouTube a články Zpráv Google, které najdou naše algoritmy."* [20] Tato služba ovšem není podporována ve všech zemích a Česká republika je jednou z těch zemí, kde k dubnu roku 2019 není podporována.

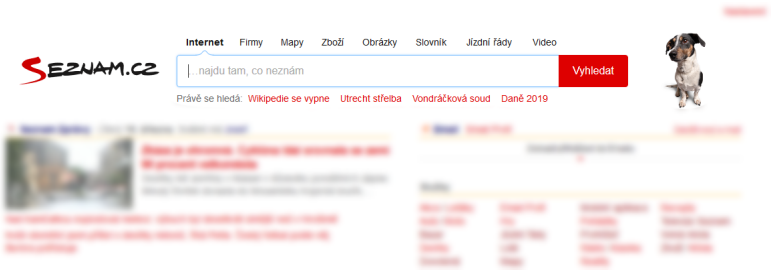
4.3.2 Seznam skokani

Seznam provozuje pod subdoménou *skokani.seznam.cz* webovou aplikaci, která obsahuje žebříčky nejhledanějších dotazů. Na stránce se nacházejí tři sekce: *Aktuální skokani*, *Měsíční skokani za kraje* a *Roční skokani*. Bohužel se zdá, že stránka není dlouhodobě aktualizovaná. K dubnu roku 2019 jsou poslední data ze září 2018.

Obrázek 4.2: Ukázka aplikace Skokani Seznam

4.3.3 Trendující témata na hlavní stránce Seznam.cz

V březnu 2019 spustil Seznam v testovacím režimu zobrazení trendujících témat přímo na hlavní stránce. Témata, o která je zvýšený zájem, jsou zobrazena přímo pod vyhledávacím polem. [21]



Obrázek 4.3: Ukázka trendujích témat na hlavní stránce Seznam.cz

4.4 Analýza požadavků

Systém má být podle zadání prototyp a není požadována zcela produkční kvalita. Jedná se spíše o takzvaný PoC tedy *Proof of concept*. Přesto je vhodné stanovit funkční a nefunkční požadavky, které má systém splňovat.

4.5 Funkční požadavky

Funkční požadavky definují, jaké konkrétní funkcionality má systém umožňovat a které aktivity či procesy má pokrývat. Funkční požadavky odpovídají na otázku, *co by měl systém umět*.

4.5.1 FRQ01 - Výpočet trendujících dotazů

Klíčovou funkcí systému je, že dokáže pro určitý časový úsek vypočítat, jaké dotazy jsou trendující. Vrací seznam dotazů s dalšími informacemi, u kterých došlo k nárůstu hledanosti.

4.5.2 FRQ02 - Batchové zpracování

Aplikace umožňuje vypočítat trendující dotazy pro uzavřenou datovou sadu. Lze vypočítat historická data. Například v rozmezí dní nebo měsíců.

4.5.3 FRQ03 - Streamové zpracování

Aplikace umožňuje počítat trendující dotazy v reálném čase. Pracuje nad potenciálně nekonečnou datovou sadou. Doba, za kterou jsou data počítána, je parametrizovatelná.

4.5.4 FRQ04 - Možnost konfigurace při spuštění

Při spuštění aplikace je možné zadávat různé parametry, podle kterých výpočet probíhá. Jedná se například o hodnotu parametru, který určuje, v jaké chvíli je dotaz označen jako trendující.

4.5.5 FRQ05 - Vizualizace dat

Výstupní hodnoty jsou zobrazeny ve webové aplikaci. Data se automaticky aktualizují.

4.5.6 FRQ06 - Podpora pro různé formáty dat

Systém umožňuje načítat různé formáty dat. Podpora je pro následující formáty: CSV, JSON a Parquet.

4.6 Nefunkční požadavky

Nefunkční požadavky odpovídají na otázku, *jaký by měl systém být*. Jde o požadavky, které uživatel úplně nevidí, ale přesto můžou mít kritický vliv na celou aplikaci.

4.6.1 NRQ01 - Distribuovaná aplikace

Aplikace musí být distribuovaná, běží na výpočetním clusteru a musí být možné ji masivně škálovat.

4.6.2 NRQ02 - Podpora pro Hadoop cluster

Výpočetní část systému lze spustit na *Hadoop clusteru*, na kterém je jako správce zdrojů použita služba YARN, a který běží na operačním systému Linux.

4.6.3 NRQ03 - Podpora pro více distribuovaných výpočetních frameworků

Výpočetní část systému není závislá na jednom konkrétním výpočetněním frameworku.

4.6.4 NRQ04 - Snadná rozšiřitelnost

Aplikace je navržena a implementována tak, že je relativně snadné rozšířit ji o další funkce. Mělo by být snadné přidat podporu pro další vstupní i výstupní formát dat.

Dále je jednoduché rozšířit samotnou výpočetní část a rozšířit ji o další datové transformace.

4.7 Role v systému

Existují pouze dva typy uživatelů, kteří budou se systémem pracovat. Jedním bude administrátor a druhým bude programátor.

4.7.1 Administrátor

Administrátor je člověk, který nutně nepotřebuje mít technické znalosti o tom, jak výpočet probíhá. Přistupuje do aplikace skrze webové rozhraní. To, co je pro něho klíčové, jsou vypočítané hodnoty, z kterých analyzuje dotazy a hledá ty nejvýznamnější a nejrelevantnější.

4.7.2 Programátor

Programátor je člověk, který má technické znalosti o tom, jak výpočet probíhá. Umí ho pomocí připraveného rozhraní konfigurovat a spouštět na clusteru.

4.8 Návrh systému

V této části navrhnu a popíši architekturu celého systému, která není závislá na konkrétní technologii.

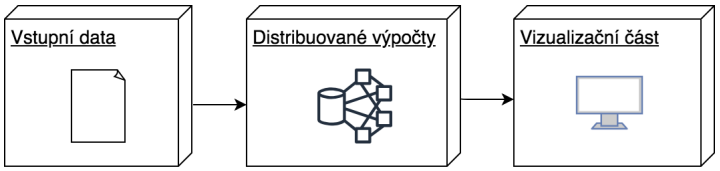
4.8.1 High-level architektura

Podle návrhu je ve velkém měřítku celková architektura systému rozdělena do tří základních částí. Na každou část lze nahlížet jako na černou skříňku, která přijímá určité vstupy a poskytuje nějaké výstupy.

První část systému udržuje vstupní data. Tuto vrstvu není nutné nijak implementovat, pouze se k ní připojit a načíst data v potřebném formátu. Může se jednat o souborový systém, databázi nebo jakoukoliv službu, která ukládá data.

Druhá část provádí samotné distribuované výpočty. Lze na ni nahlížet jako na funkci, jejíž vstupem jsou informace z hledání, které poskytuje první část, a výstupem data, důležitá pro uživatele, tedy seznam dotazů, které jsou označené jako trendující dotazy. Vstupních formátů může být více a mohou se případně měnit nebo rozšiřovat. Z toho důvodu je na tento předpoklad nutné myslet při návrhu této části systému.

Vypočítaná data jsou předána dále do části, která má na starosti vizualizaci. Třetí část má za úkol udržovat vypočítané informace a poskytovat je uživateli.



Obrázek 4.4: High-level schéma systému nezávislé na implementaci

4.8.2 Metodika vývoje

Před samotnou implementací neproběhl žádný složitý proces detailního návrhu celé aplikace, jako je například úplný diagram tříd, vše vznikalo agilně během vývoje v několika iteracích.

Kapitola 5

Implementace systému

V této kapitole popíši implementaci celého systému. Nejprve představím systém jako celek, a poté popíši jednotlivé části, z kterých se skládá. U každé části popíši technologii, kterou jsem použil, popíši samotnou strukturu a implementační detaily.

5.1 Základní popis

Podle návrhu je aplikace rozdělena do tří samostatných částí. Jak již bylo zmíněno, první část, tedy uložiště dat, není nutné implementovat. Data jsou uložená v souborech na distribuovaném souborovém systému HDFS a v systému Apache Kafka. Zbývají tedy dvě části, výpočetní a vizualizační. Vizualizační část je rozdělena do dvou samostatných aplikací.

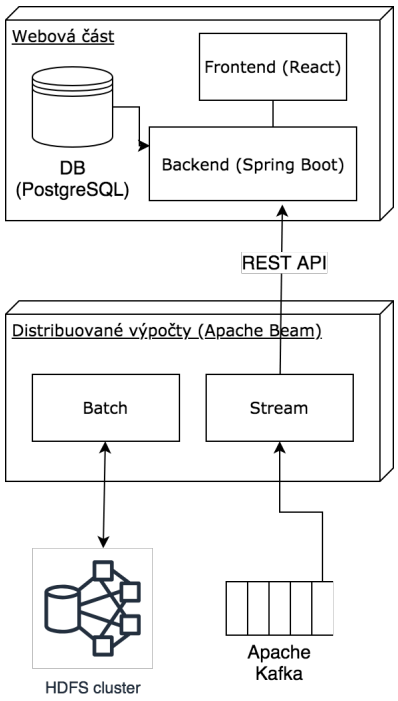
Celá aplikace se skládá ze tří samostatných komponent. První a nejdůležitější komponenta se nazývá **Search trends Core** (dále jen **Core**) a jde o distribuovaný výpočet. Aplikace je napsaná ve frameworku Apache Beam v jazyce Java. Jeho funkcí je vypočítávat trendující dotazy ze vstupních dat.

Další dvě části tvoří jeden celek a to webovou aplikaci, která vypočítaná data ukládá a vizualizuje. Aplikace je rozdělena na zcela oddělenou backend a frontend část. Rozdělení webové aplikace na backend a frontend je u moderních webových aplikací časté. Backend poskytuje rozhraní, ve formě REST API. Frontendová část data pomocí rozhraní přijímá a vykresluje je uživateli.

Backend aplikace se nazývá **Search trends Web Backend** (dále jen **Web backend**) a frontend část **Search trends Web Frontend** (dále jen **Web Frontend**).

Backend je postavený ve frameworku Spring Boot a napsaný v jazyce Kotlin. Frontend stojí na frameworku React a jazyce JavaScript.

Schéma celého systému je uvedeno na následujícím obrázku.



Obrázek 5.1: High-level schéma systému s konkrétní technologií

5.2 Search trends Core

5.2.1 Výběr jazyka

Aplikace je napsaná v Jazyce Java. Java se spouští na virtuálním stroji zvaném *Java virtual machine* neboli JVM. Tento proces zajišťuje to, že je jazyk multiplatformní.

Na JVM mohou běžet i jiné jazyky. Zajímavou alternativou k Javě je jazyk Scala, který spojuje funkcionální a objektový styl programování. V tomto jazyce je z velké části implementován framework Apache Spark. Jiná moderní a populární technologie je jazyk Kotlin, který vyvíjí společnost JetBrains. Tento jazyk také běží na JVM.

5.2.2 Výběr výpočetního frameworku

Systém je postavený na frameworku Apache Beam. Důvody, proč jsem tuto technologii použil jsou prakticky totožné s tím, co uvádí samotní autoři. Primárně tedy proto, že se jedná o vyšší míru abstrakce, která není závislá na konkrétním výpočetním frameworku. Kromě toho jsem ji použil také z toho důvodu, že se používá ve společnosti Seznam.cz.

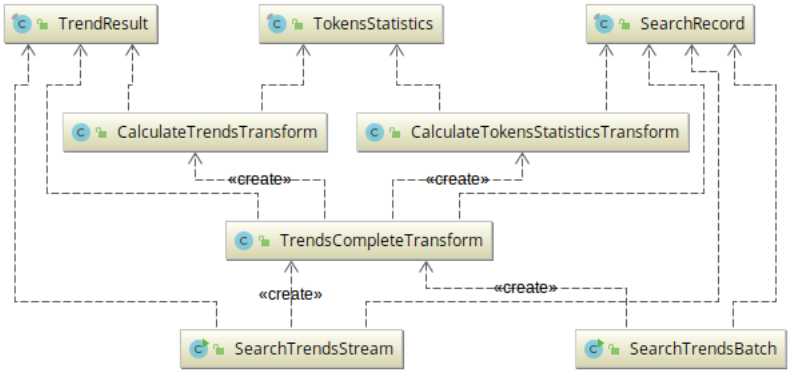
Vývojářům v Seznamu se podařilo na konci roku 2018 integrovat svůj vlastní projekt, jehož filozofie je prakticky totožná s filozofií Apache Beam, na-

zvaný Euphoria [22] přímo do Apach Beam a to formou rozšíření. [23]

5.2.3 Struktura projektu

Projekt dodržuje standardní strukturu *Maven* projektu. V kořenovém adresáři projektu nalezneme soubor *pom.xml*. Tento soubor mimo jiné předepisuje, jak má být projekt sestaven a obsahuje závislosti, které projekt potřebuje ke svému běhu. Dále obsahuje adresář *src* a *script*. Adresář *script* obsahuje pomocné užitečné skripty v jazyce *Bash* a *src* obsahuje samotné zdrojové kódy aplikace.

Všechny zdojové třídy patří do balíčků (anglicky *packages*), které začínají prefixem *cz.fit.cvut.searchtrends*. V kořeni tohoto balíčku jsou dvě spouštěcí třídy, které obsahují hlavní spouštěcí metodu *main*.



Obrázek 5.2: Diagram tříd části aplikace Search trends Core

- ***SearchTrendsBatch*** je třída, která spouští batchový výpočet. Vstupní data mohou být ve formátu JSON, CSV a Parquet.
- ***SearchTrendsStream*** je třída, která spouští streamový výpočet. Data jsou načítána z Apache Kafka, jednotlivý záznam je ve formátu JSON.

Jak je patrné z diagramu tříd, obě třídy sdílejí stejnou logiku co se týče výpočtu. Liší se primárně ve způsobu načítání dat a v zápisu výstupu.

5.2.4 Vstupní data

Při vyhledání dotazu, který je zadán uživatelem, jsou informace o hledání uloženy, takzvaně *zalogovány*. Jeden záznam se nazývá *log*.

Logy jsou ukládány do distribuované transportní vrstvy Apache Kafka.

Základní vstupní hodnoty pro výpočet jsou následující:

- Tokeny dotazu

- Identifikátor klienta
- Čas hledání

5.2.4.1 CSV

Comma-separated values neboli CSV je jednoduchý textový formát dat. Jednotlivé hodnoty jsou uloženy na jeden řádek a odděleny speciálním znakem. Často se jedná o tabulátor případně středník. Tato reprezentace je vhodná spíše pro testovací a ladící účely, protože formát je dobře čitelný pro člověka.

Core umí načíst tento soubor s předpokládanou strukturou, že hodnoty jsou oddělené středníkem v pořadí: čas události, Identifikátor klienta a celý dotaz. Dotaz je až jako poslední hodnota, protože může obsahovat oddělovací znak. Samotná logika parsování probíhá ve třídě *CsvLogParser*.

```
1.551175962025E9;#NDEuMjMwL;vysavac protool
1.551177506217E9;#MTc2Ljk3L;csob uvodni stranka
1.551178534898E9;#ODUuMjA3L;facebook
1.551176014795E9;#MmESi3xM;benzina
1.551176203594E9;#MmEwMDoxM;ikea
```

5.2.4.2 JSON

Jednotlivá hledání jsou do systému Kafka ukládána ve formátu JSON. *JavaScript Object Notation* neboli JSON je textový strukturovaný formát, který je dobře lidsky i strojově čitelný. Při zpracování velkých dat se však kvůli jeho velikosti nejedná o nejlepší formát na dlouhodobé skladování dat.

Tento JSON obsahuje informace, které vznikly na různých místech při zpracování požadavku. Tyto informace jsou například: Původní dotaz, identifikátor uživatele, čas, kdy uživatel dotaz zadal, nebo základní informace o uživateli jako je operační systém, typ zařízení či webový prohlížeč.

Pro tuto práci je důležitá část JSON dokumentu zvaná *metasearchInfo*. Tato sekce obsahuje informace o dotazu, například i to, zda se jedná o pornografický dotaz, díky této informaci lze dotazy kvůli korektnosti filtrovat. Nejdůležitější však je, že obsahuje *tokeny* dotazu. Tokenizace je proces, během něhož je text rozdělen na nejmenší logické celky. V nejjednodušším případě se může jednat o tokenizaci podle mezer a jiných neviditelných znaků (anglicky *whitespace*).

V následující ukázce je uveden příklad části JSONu pro hledané slovo "jihlava".

```
1     ...
2     "metasearchInfo": {
3       "pornFilterLevel": "noporn",
4       "query": {
5         "correctionUsed": false,
```

```
      "languages": [
        {
          "langCode": "cs",
          "langProbability": 0.998667072504
        }
      ],
      "naviness": 1,
      "query": "jihlava",
      "tokens": {
          "derived": [],
          "original": {
            "tag": "k1gFnSc1wM",
            "token": "jihlava",
            "lang": "cs:0.975956, ",
            "lemma": "jihlava",
            "order": 0,
            "orig": "jihlava",
            "paddingLeft": " ",
            "paddingRight": ""
          }
        }
      ]
    },
    "status": 200,
    "statusMessage": "OK",
    "docsFound": 8126,
    "foreigness": 0.138163805008
  }
  ...
```


Jak již bylo zmíněno, kvůli tomu, že textová reprezentace není vzhledem k objemu dat pro dlouhodobé skladování vhodná a systém Apache Kafka není stavěný na ukládání velkého množství dat po dlouhou dobu, jsou v pravidelných intervalech data z Kafky načtena a uložena v jiném úspornějším formátu na HDFS. Formát, v kterém jsou data dlouhodobě uchována, je sloupcově orientovaný formát *Apache Parquet*.

5.2.4.3 Apache Parquet

Parquet je sloupcově orientovaný formát dat. Tato orientace je výhodná především při dotazování nad daty v případě, že jeden řádek obsahuje mnoho sloupců. Častěji je totiž potřeba vybrat jen některé sloupce všech záznamů. Díky sloupcové orientaci není nutné načítat do paměti celý záznam a až poté vybírat příslušný sloupec.

Logika načítání formátu parquet je ve třídě *ParseParquetTransform*. Apache Beam poskytuje rozhraní pro načítání parquet souboru. Jelikož jsou interně jednotlivé záznamy uloženy ve formátu Avro, je potřeba při načítání poskyt-

nout i schéma. Toto schéma je uloženo přímo v projektu jako statický zdroj v adresáři *resource*.

5.2.5 Tokenizace

Jak již bylo zmníněno, proces tokenizace vytvoří z celého dotazu seznam tokenů. Tato informace je dostupná pouze pro data ve formátu JSON, které jsou čteny z Kafky. Ostatní formáty tuto informaci bohužel neobsahují.

Bylo nutné tedy vytvořit vlastní tokenizaci. V projektu se nachází rozhraní *Tokenizer*, které definuje metodu *tokenize*, která přijímá jako parametr textový řetězec a vrací seznam tokenů. Díky definici rozhraní je jednoduché přidat vlastní implementaci tokenizátoru.

Projekt obsahuje jednoduchou implementaci v podobě třídy *SimpleTokenizer*, která rozdělí slova podle bílých znaků a podle určitých speciálních znaků, jako jsou tečky nebo čárky. Tato tokenizace je použita v CSV a parquet parserech.

5.2.6 Konfigurace parametrů

Apache Beam nabízí možnost konfigurace instance třídy *Pipeline* pomocí rozhraní *PipelineOptions*. Spouštěcí třídy pro batchové i streamové zpracování mají některé konfigurační hodnoty společné. Ty jsou definované ve společném rozhraní *SearchTrendsCommonOptions*. Unikátní parametry jsou pak definovány pro každou třídu zvlášť v rozhraních *SearchTrendsBatchOptions* a *SearchTrendsStreamOptions*, které *SearchTrendsCommonOptions* rozšiřují.

Tabulka 5.1: Společné parametry pro batchový i streamový výpočet

Název parametru	**Popis**
debugEnabled	Boolean hodnota určuje, zda se jedná o ladící mód.
debugOutputPath	Definice cesty, na kterou se uloží CSV soubory s vypočtenými hodnotami. Vhodné pro ladění.
removeDuplicateQueriesByClient	Boolean příznak, který říká, zda se má více hledání stejného dotazu jedním klientem započítat pouze jednou.
trendingRatioThreshold	Koeficient, při jehož překročení je dotaz označen jako trendující.
restUrl	URL adresa, na které je vystaveno REST API, pomocí něhož aplikace zapisuje vypočtené hodnoty. Výchozí hodnota je nastavena na *http://localhost:8080*.

Tabulka 5.2: Parametry pouze pro batchový výpočet

Název parametru	**Popis**
inputFile	Cesta ke vstupnímu souboru. Souborů může být více než jeden, v takovém případě je pro zápis použit znak hvězdičky, zvaný *wildcard*. Zápis může vypadat například následovně */input/data/*.csv*.
inputFormat	Formát vstupních dat. Může nabývat hodnot *csv*, *json*, *parquet*.
pastFromDateTime	Začátek prvního, tedy staršího časového intervalu.
pastToDateTime	Konec prvního, tedy staršího časového intervalu.
nowFromDateTime	Začátek druhého, tedy novějšího časového intervalu.
nowToDateTime	Konec druhého, tedy novějšího časového intervalu.

Tabulka 5.3: Parametry pouze pro streamový výpočet

Název parametru	**Popis**
bootstrapServers	Adresy Kafka brokers, tedy uzlů Kafka clusteru, ke kterým se může klient připojit. Jedná se o seznam hodnot oddělených středníkem ve formátu *host:port*.
topic	Název Apache Kafka topic.
jdbcEnabled	Příznak určující, zda má být výstup zapsán pomocí JDBC rozhraní. Výchozí hodnota je *false*.
windowSizeInSeconds	Velikost okna v sekundách.
resultsEverySeconds	Hodnota určuje, kdy má dojít k zápisu výsledků.
newPartSizeInSeconds	Velikost části časového intervalu, v které jsou hodnoty považovány za novější.

5.2.7 Výpočet

Výpočetní logika se nachází v balíčku *cz.fit.cvut.searchtrends.transform*.

Jednotlivé datové transformace lze vnořovat do sebe, z toho důvodu jsou rozděleny do podbalíčků *partial* a *complete*. Zapouzdření celé logiky výpo-

čtu trendů se nachází ve třídě *TrendsCompleteTransform*. Zapouzdření celého procesu do jedné transformace, vede k usnadnění testování.

Vstupem do transformace je paralelní kolekce objektů *TrendResult*. Výstupem je opět paralelní kolekce vypočítaných trendů, což jsou instance třídy *TrendResult*.

Kroky datové transformace jsou následující.

1. Filtrace záznamů s neprázdnými tokeny a případně jiné filtrace hodnot.
2. Přemapování vstupních objektů na strukturu klíč-hodnota. Klíčem jsou tokeny a hodnotou jsou zbylá data.
3. Aplikování transformace *GroupByKey*. Po této transformaci jsou získány unikátní tokeny a s ním seznam klíčů s dodatečnými informacemi.
4. Následuje výpočet statistik. Statistky obsahují četnosti hledání za určité časové intervaly. Výpočet statistik je umístěn ve třídě *CalculateTokensStatisticsTransform*. Je implementován pomocí transformace *ParDo*.
5. Dojde k výpočtu samotých trendů pomocí statistik. K tomu dochází ve třídě *CalculateTrendsTransform*.

5.2.7.1 Koecifient trendovosti

Logika určující, které dotazy jsou na základě vypočítané statistiky trendující, se nachází v balíčku *transform.logic*. Při návrhu byl opět kladen důraz na jednoduchou rozšiřitelnost, proto se zde nachází rozhraní *TrendResultDivider*. Rozhraní definuje metodu *process*, která rozhoduje, zda je daný dotaz trendující. Metoda obsahuje následující parametry: seznam tokenů, stará četnost, nová četnost a hodnotu *trendingRatioThreshold*.

Výstupem metody je objekt *TrendResultWrapper*, který zapouzdřuje informaci o tom, zda byl dotaz vyhodnocen jako trendující, spolu se samotnou hodnotou.

Core obsahuje implementaci tohoto rozhraní v podobě třídy *SimpleTrendResultDivider*. Na základě četnosti z dvou různých intervalů se spočítá podíl novější hodnoty vůči starší. Za trendující dotaz je označen ten, který překročí předem stanovenou hranici zvanou *trendingRatioThreshold*. Četnost staršího intervalu je označena jako *pastFrequency* a novějšího jako *nowFrequency*. Ve třídě je definována hranice, kterou musí nová četnost překročit, pokud je hodnota staré četnosti nulová.

$$\frac{nowFrequency}{pastFrequency} >= trendingRatioThreshold$$

V této části je prostor pro rozšíření o složitější implementaci, který by mohla obsahovat zajímavou nelineární funkci.

5.2.7.2 Streamové zpracování

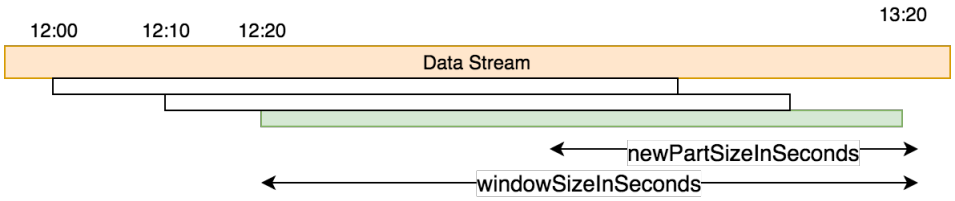
Při streamovém zpracování je nejprve nutné připojit se k systému Apache Kafka a začít konzumovat záznamy. To je zajištěno třídou *KafkaIO*, kterou poskytuje Beam SDK.

Již při čtení záznamů je vhodné provádět deserializaci hodnot. Deserializaci provádí třída *SearchRecordJsonDeserializer*. Nejdůležitějším úkonem, bez kterého by streamové zpracování nefungovalo správně, je nutnost přiřadit záznamům časovou známku. To se děje pomocí vlastní implementace rozhraní *TimestampPolicyFactory*, kterou je třída *SearchRecordTimestampPolicyFactory*. Načítání dat je ukázáno v následující ukázce.

```
1   KafkaIO.Read<Long, SearchRecord> kafkaIOReader =
2       KafkaIO.<Long, SearchRecord>read()
3           .withBootstrapServers(options.getBootstrapServers())
4           .withTopic(options.getTopic())
5           .withKeyDeserializer(LongDeserializer.class)
6           .withValueDeserializer(SearchRecordJsonDeserializer.class)
7           .withTimestampPolicyFactory(new
                SearchRecordTimestampPolicyFactory())
8           .updateConsumerProperties(
9               ImmutableMap.of(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
                    "latest"));
```

Po načtení hodnot je potřeba nastavit strategii streamování tím že se definuje velikost okna a časový interval, kdy dojde k emisi výstupů. Nastavení probíhá pomocí transformace, jak je vidět z následující ukázky.

```
1   ...
2   .apply(
3       Window.into(
4           SlidingWindows.of(
5               Duration.standardSeconds(options.getWindowSizeInSeconds()))
6           .every(
7               Duration.standardSeconds(options.getResultsEverySeconds())))
8               );
```



Obrázek 5.3: Grafické znázornění strategie streamového výpočtu

5.2.8 Výstup

Aplikace podporuje dvě možnosti, jak vypočítaná data přenést dále. První možností je zapsat data přímo do SQL databáze. Druhou možností je odeslat výstupy pomocí REST API. Výstupy jsou opět určité druhy transformací, není tedy problém s jejich rozšířením.

5.2.8.1 JDBC

V rámci projektu je implementovaná podpora pro databázi *PostgreSQL*. Pro zápis do databáze je nutné nastavit parametry: ovladač databáze (anglicky *driver*), který musí být přítomný na *classpath*, tedy zahrnutý v projektu nebo poskytnut prostředím, na kterém aplikace běží.

5.2.8.2 REST

Representational State Transfer Application Interface neboli REST API je typ rozhraní, které umožňuje komunikaci s aplikací. REST je postaven na protokolu HTTP.

Samostatný REST klient, tedy komponenta, která rozhraní volá, je ve třídě *TrendResultsRestClient*. Existuje celá řada REST klientů, které lze využít. V projektu je použita knihovna *Unirest* [24] a to především kvůli jednoduchosti použití a malé velikosti.

5.3 Search trends Web backend

5.3.1 Výběr jazyka

Pro implementaci backend části jsem zvolil jazyk Kotlin. Mohl jsem zvolit samozřejmě Javu, ale chtěl jsem si vyzkoušet modernější a populární jazyk, který ovšem těží z veškerého Java ekosystému jako jsou knihovny a frameworky.

5.3.2 Technologie

Backend je postavený na frameworku *Spring Boot*. Tuto technologii jsem zvolil především z toho důvodu, že jsem s ní měl největší zkušenost. Spring framework je velmi používaný, léty prověřený a má kolem sebe velkou komunitu vývojářů.

Jako buildovací nástroj jsem pro změnu použil Gradle. Jedná se o alternativu k Apache Mavenu. Na rozdíl od Mavenu je syntaxe oproti XML úspornější, protože je v jazyce Groovy, případně přímo v Kotlinu.

5.3.3 Struktura projektu

Přestože je aplikace malá, skládá se ze dvou samostatných modulů nazvaných *web* a *persistence*.

5.3.3.1 Persistence modul

Modul spravuje persistentní neboli databázovou vrstvu. Spravuje entity, které díky objektovému relačnímu mapování (ORM) představují i databázové tabulky. Entita je třída s anotací *Entity*.

Pro manipulaci s entitou slouží repozitáře. Projekt Spring Data v rámci Spring frameworku nabízí elegantní způsob, jak tyto třídy vytvořit.

Definují se pouze rozhraní, která rozšiřují rozhraní *JpaRepository*, toto rozhraní přijímá dva typové parametry. Prvním parametrem je typ entity, nad kterou repozitář operuje, a druhým parametrem je typ primárního klíče.

Již tato implementace poskytuje základní CRUD operace nad entitami. Pro složitější operace je potřeba nadefinovat funkce v rozhraní. Na základě názvu metody framework sám za běhu aplikace vytvoří třídy včetně implementace. Díky tomu se minimalizuje prostor pro chybu, ušetří se čas při implementaci a zachová se jednotná konvence při pojmenování metod.

```
1 @RepositoryRestResource
2 interface SearchTrendRepository : JpaRepository<SearchTrend, Long> {
3
4     fun findAllByOrderByCreatedAtDesc(): List<SearchTrend>
5
6     ...
```

V předchozí ukázce stojí za povšimnutí anotace *RepositoryRestResource*, díky které jsou za běhu aplikace vygenerovány základní REST zdroje pro CRUD operace nad entitami. Při prototypování je tato anotace velice užitečná.

5.3.3.2 Web modul

Web modul obsahuje základní spouštěcí třídu celé aplikace. V nejjednodušším případě vypadá tato třída následovně.

```
1 @SpringBootApplication
2 class Application
3
4 fun main(args: Array<String>) {
5     runApplication<Application>(*args)
6 }
```

Modul dále obsahuje konfigurační třídy a REST kontrolery (anglicky *Controllers*). Kontrolery jsou třídy, které vystavují REST API metody a děje se v nich složitější logika, která není obsažená ve vygenerovaných třídách.

V adresáři *resources* jsou uložené konfigurační soubory, které jsou ve formátu YAML.

5.3.3.3 REST API

Aplikace poskytuje následující REST operace.

Tabulka 5.4: Přehled REST metod poskytovaných aplikací Search trends Core

Url	HTTP metoda	Popis
/api/search-trends/last-hour	GET	Vrací seznam nejvýznamnějších trendujících dotazů za poslední hodinu, unikátních podle jména a seřazené sestupně podle koeficientu trendovosti.
/api/search-trends/top/last-hour	GET	Vrací seznam deseti nejvýznamnějších trendujících dotazů za poslední hodinu, které jsou unikátní podle jména a seřazené sestupně podle koeficientu trendovosti.
/api/search-trends/all	GET	Vrací seznam všech trendujících dotazů seřazených sestupně podle času vzniku.
/api/search-trends/history	GET	Pro dotaz, který je uvedený jako parametr volání, vrací seznam všech historických hodnot.
/api/jpa/search-trends	GET, POST, PUT, DELATE	Standardní základní CRUD operace nad entitou.

5.4 Search trends Web frontend

5.4.1 Výběr Jazyka

Při vývoji webového rozhraní kromě klasických technologií jako je HTML a CSS, dominuje jazyk JavaScript. Pro implementaci byl tedy zvolen JavaScript konkrétně verze ES6.

5.4.2 Výběr technologie

Smyslem práce nebylo vytvořit rozsáhlou webovou aplikaci se spoustou možností, ale pouze data poměrně jednoduchým způsobem vizualizovat. Z tohoto důvodu jsem se rozhodl použít volně dostupnou šablonu pod licencí MIT

a upravit ji tak, aby odpovídala potřebám aplikace. Šablona se jmenuje CoreUI a jejím hlavním autorem je Łukasz Holeczek. [25] Šablona je ve frameworku React. Samotnou datovou logiku aplikace však bylo nutné naprogramovat.

5.4.2.1 React framework

React je JavaScriptová knihovna pro tvorbu uživatelského rozhraní. Technologii vyvinula společnost Facebook a následně uvolnila jako open source projekt. [26] Knihovna je založená na konceptu komponent. Komponenty lze skládat dohromady z různých komponent. Komponenta udržuje svůj stav, tedy například hodnoty, které má vykreslit. Při změně stavu framework sám velmi efektivně překreslí změněné hodnoty.

5.5 Vývoj

5.5.1 Vývojové prostředí

Při vývoji jsem používal vývojové prostředí IntelliJ IDEA od společnosti JetBrains. Použil jsem komunitní verzi, která je dostupná zdarma. Pro vývoj frontendu v jazyce JavaScript jsem používal vývojové prostředí Visual Studio Code od společnosti Microsoft, které je také zcela zdarma, a open source.

5.5.2 Verzování

Všechny tři projekty jsou verzovány pomocí nástroje GIT a jedná se o samostatné repozitáře, které jsou uložené na serverech *github.com*.

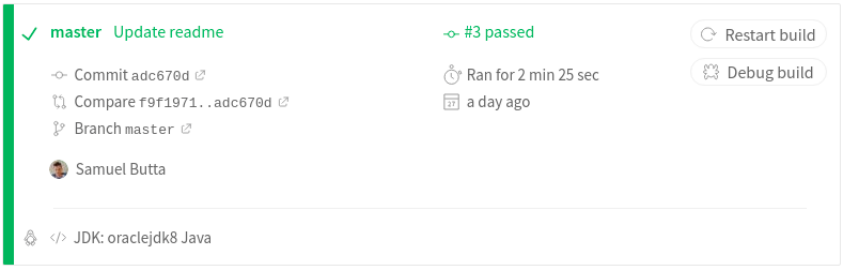
Vzhledem k tomu, že jsem na vývoji pracoval sám, udržoval jsem pouze jednu větev a to větev *master*, pod tímto názvem bývá označena hlavní a nejdůležitější větev.

Při práci ve více lidech není tento způsob vývoje vhodný. Často má práva takzvaně *zamergovat* změnu, tedy přidat změnu do hlavní větve pouze omezená skupinka správců projektu (anglicky *maintainer*). Při vývoji se běžně z hlavní větve vytvoří větev s novou funkcionalitou, která je po kontrole správcem do hlavní větve přidána.

5.5.3 CI

Continous integration neboli CI je snaha integrovat změny od všech vývojářů co nejrychleji, tak aby se urychlil vývoj, zamezilo se případným konfliktům a chyby se objevily co nejdříve. Hlavní součástí tohoto procesu je integrační server, na kterém dochází k pravidelnému sestavení aplikace a spuštění testů. V rámci CI se také často provádí automatické či poloautomatické nasazení aplikace na vývojové nebo dokonce produkční prostředí.

V rámci Core projektu byl nastaven integrační server pomocí služby Travis CI. [27] Integrace byla nastavená tak, že při každé změně v hlavní větvi dojde k sestavení a spuštění aplikace. Konfigurace služby se nachází v souboru *.travis.yml*.



Obrázek 5.4: Výstup úspěšného sestavení pomocí služby Travis CI

5.5.4 Dokumentace

Všechny třídy projektu a většina metod obsahuje dokumentaci ve formátu *Javadoc*. V kořenovém adresáři každého projektu se nachází soubor *README.md*, ve kterém je popsaný postup, jak aplikaci sestavit a spustit.

5.6 Možnosti rozšíření

Aplikace je implementovaná jako funkční prototyp. Při vývoji byl kladen důraz na snadnou rozšířitelnost. Zajímavé by bylo rozšířit aplikaci tak, aby umožňovala počítat hodnoty pro různé intervaly najednou.

Jedním z problémů současné implementace je, že dotazy, které jsou podobné nebo mají zcela totožný význam, ale liší se jejich zápis, tak jsou brány jako rozdílené dotazy. Řešit tuto úlohu není triviální, a proto přesahuje rámec této práce, jedná se však o logické rozšíření.

Dalším možným rozšířením by byla filtrace dotazů, které nejsou pokládány člověkem, ale automatickým webovým robotem. Opět se jedná o poměrně nesnadný úkol, který je nad rámec této práce.

V prezentační části je velký prostor pro rozšíření. Může se jednat o filtrace záznamů, řazení výsledků nebo vyhledávání nad dotazy. Tato rozšíření se spíše týkají samotného vývoje webové aplikace a jejího rozhraní. Cílem aplikace nebylo vytvořit co nejvíce možností zobrazení, ale pouze demonstrovat funkčnost.

Kapitola 6

Lokální spuštění a testování

V této kapitole nejprve popíši, jak lze všechny části systému zkompilovat do spustitelné podoby, a představím způsoby, jakými je aplikace testována.

Nakonec ukáži, jak lze celý výpočetní proces spustit na jednom lokálním počítači.

6.1 Sestavení aplikace

6.1.1 Search trends core

Jako sestavovací nástroj je použit nástroj Apache Maven. Maven je program, který lze mít nainstalovaný přímo v systému. Kvůli kompatibilitě verzí se však tento program přidává přímo do projektu v podobě zvané Maven Wrapper.

Maven definuje jednotlivé *tasky*, což jsou operace probíhající nad projektem. Celou aplikaci lze sestavit taskem *package*, který zkompiluje zdrojový kód, spustí všechny testy a sestaví aplikaci do spustitelného souboru JAR. Task *clean*, který je použit v následující ukázce, pouze zajistí smazání souborů, které vznikly při předchozím sestavení.

```
1 ./mvnw clean package
```

Díky profilování je možné zkompilovat výslednou aplikaci pro konkrétní distribuovaný výpočetní framework.

Definováním parametru profilu dojde při sestavení výslednému JAR souboru k přidání všech knihoven potřebných pro spuštění na konkrétní výpočetní platformě.

```
1 ./mvnw clean package -P spark-runner
```

Po vykonání příkazu se v adresáři *target* vytvoří dva JAR soubory. Jedním z nich je takzvaný *fat* JAR, což je soubor, který obsahuje všechny potřebné

závislosti a je spustitelný. Tento soubor má příponu *bundled*, například *search-trends-core-bundled-0.1.jar*.

6.1.2 Search trends Web backend

Aplikaci lze sestavit pomocí nástroje Gradle. Gradle je program, který je principiálně dost podobný jako Maven. Samotný Gradle program je opět jako Gradle Wrapper součástí projektu.

```
./gradlew clean build
```

Pro vytvoření spustitelného JAR souboru lze použít task *bootJar*. Tento task vytvoří v adresáři *web/build/libs* soubor *web-0.1-SNAPSHOT.jar*. Vygenerovaný soubor obsahuje všechny potřebné závislosti a je spustitelný.

```
./gradlew bootJar
```

6.1.3 Search trends Web frontend

Pro sestavení frontend části je použit nástroj *Node package manager* neboli NPM. Pro lokální vývoj je vhodné použít příkaz *npm start*, který aplikaci spustí a to konkrétně na portu 3000. Výhodné je, že při tomto použití se jakákoliv změna v kódu okamžitě promítne bez nutnosti znovu načíst stránku.

Pro vytvoření produkčně nasaditelných souborů lze použít příkaz *run build*, který vytvoří adresář *build*, v kterém se nachází potřebné soubory. Jediné, co stačí, je v prohlížeči otevřít soubor *index.html*.

```
npm run build
```

Aplikace komunikuje přes REST API, jehož identifikátor se pro různá prostředí liší. Není tedy vhodné mít tuto definici přímo v kódu aplikace. Standardním způsobem je nastavení proměnných prostředí, jejichž hodnoty si při sestavení aplikace načte a nastaví. Lokální spuštění s nasazením hodnoty proměnné prostředí by vypadalo následovně.

```
REACT_APP_TRENDS_REST_URL="http://localhost:8080" npm start
```

Příkaz pro sestavení produkčních souborů je následující.

```
REACT_APP_TRENDS_REST_URL="http://srch.trendsdev.com:8080" npm run build
```

6.2 Lokální spuštění aplikace

Celý soubor aplikací lze spustit na lokálním počítači. Pro spuštění streamového výpočtu je nejprve nutné spustit lokální systém Apache Kafka a vytvořit

příslušný topic, do kterého se data budou ukládat. Pro batchové spuštění výpočtu stačí mít vstupní data v souboru.

6.2.1 Nastavení Apache Kafka

Nejprve je nutné mít stažený adresář se spouštěcími soubory. V adresáři *bin* Apache Kafka se nachází skripty pro spuštění. Kafka potřebuje pro svůj běh komponentu Zookeeper, kterou je nejprve nutné nastartovat.

```
1 bin/zookeeper-server-start.sh config/zookeeper.properties
```

Následuje spuštění samotné. Kafka běží ve výchozím nastavení na portu 9092.

```
1 bin/kafka-server-start.sh config/server.properties
```

Pomocí kafka-topics.sh skriptu lze vytvořit topic. V následující ukázce je nazvaný *trends-topic*.

```
1 bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --topic
    trends-topic
```

Dále je potřeba do systému odesílat nějaká ověřovací data, aby měl výpočet s čím pracovat. K tomuto účelu slouží třída *TrendsKafkaProducer* nacházející se v *test* adresáři projektu Core v balíčku *cz.fit.cvut.searchtrends.mockdata*.

Třída obsahuje spouštěcí metodu *main*, ve které se nachází nekonečná smyčka, která v nepravidelných intervalech odesílá testovací hodnoty do příslušného Kafka topic.

```
1 bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
    trends-topic
```

6.2.2 Spuštění Search trends Core

Pro lokální spuštění je nejjednodušší použít *DirectRunner*. Tato hodnota je výchozí, což znamená, že je pod ním aplikace spuštěna i přímo z vývojového prostředí. Tento runner je vhodný pro testování. Přestože se snaží co nejvíce simulovat skutečný distribuovaný framework, je lepší aplikaci spustit přímo v něm. Pro tyto účely je v projektu adrešář *script/run*.

Spouštěcí skript pro framework Spark vypadá zhruba následovně. Velice podobně vypadají i spouštěcí skripty pro jiné výpočetní frameworky.

```
1 #!/usr/bin/env bash
2
3 TIME_STAMP=‘date "+%Y%m%d-%H%M%S"‘
4 SPARK_SUBMIT=/opt/spark-2.4.0-bin-hadoop2.7/bin/spark-submit
```

```
MAIN_CLASS=cz.fit.cvut.searchtrends.SearchTrendsBatch
JAR_PATH=target/search-trends-core-bundled-0.1.jar

${SPARK_SUBMIT} \
 --class ${MAIN_CLASS} \
 --master local ${JAR_PATH} \
 --runner=SparkRunner \
 --debugEnabled=true \
 --removeDuplicateQueriesByClient=true \
 --inputFormat="csv" \
 --inputFile="/tmp/data/csv/file.csv" \
 --debugOutputPath="/tmp/output/batch-$TIME_STAMP-" \
 --pastFromDateTime="2019-01-01T12:00:00Z" \
 --pastToDateTime="2019-02-01T12:00:00Z" \
 --nowFromDateTime="2019-02-01T12:00:00Z" \
 --nowToDateTime="2019-02-28T12:00:00Z" \
 --trendingRatioThreshold=2.5
```

6.2.3 Spark streaming

Pokud je na lokálním počítači správně nastavený systém Kafka, je možné spustit lokálně i streamový výpočet. K tomu slouží skript *spark-stream-local.sh*.

6.2.4 Flink

Pro spuštění pod technologií Flink je nutné, aby výsledný JAR soubor obsahoval speciální závislosti pro tento framework. Je nutné program sestavit pod profilem *flink-runner*.

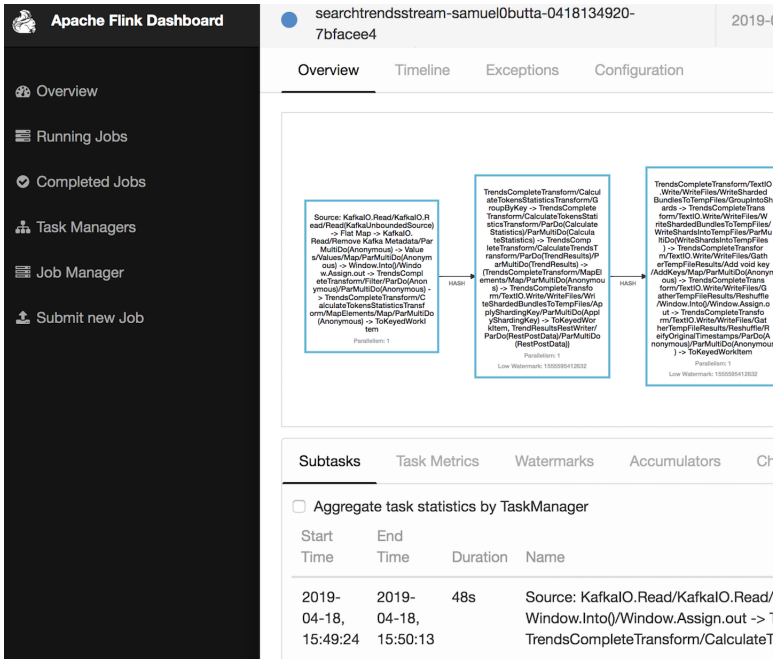
```
mvn clean package -P flink-runner
```

Dále je potřebné mít spuštěný Flink jako službu, aby se k ní mohla aplikace připojit. Spouštěcí skript se liší ve spuštění samotného Flinku, ostatní parametry zůstávají stejné.

```
...
FLINK_PATH=/Users/samuel.butta/Programy/flink-1.8.0/bin/flink
MAIN_CLASS=cz.fit.cvut.searchtrends.SearchTrendsStream
JAR_PATH=target/search-trends-core-bundled-0.1.jar

${FLINK_PATH} run -c ${MAIN_CLASS} ${JAR_PATH} \
 --runner=FlinkRunner \
 ...
```

Na portu 8081 běží administrační aplikace. V ní je možné vidět a spravovat běžící programy.



Obrázek 6.1: Rozhraní administrační webové aplikace Apache Flink

6.3 Testování

Testování aplikace je možné různými způsoby. Projekt Search trends Core je pokryt testy a testován na následujících úrovních. Tyto úrovně vycházejí z možností, které jsou k dispozici ve frameworku Apache Beam.

- Jednotkové testování (anglicky *Unit tests*)
- Integrační testy
- Verifikace při lokálním spuštění
- Verifikace při spuštění na clusteru

6.3.1 Jednotkové testování

Jednotkové testy neboli Unit testy slouží k otestování právě jedné funkcionality, tedy třídy. Pro testování byl použit nástroj JUnit. Jako příklad jednotkových testů, které jsou přítomny v projektu, lze uvézt třídy *TokenizerTest*, *GsonSearchLogParserTest* nebo *SimpleTrendResultDividerTest*.

6.3.2 Integrační testování

Integrační testy testují více propojených jednotek. Apache Beam umožňuje vytvořit testovací objekt, který zapouzdřuje více transformací, zvaný *TestPipeline*.

Díky testovacím třídám z balíčku *org.apache.beam.sdk.testing* lze ověřit správnost výstupu. Ukázku testu logiky transformace, která je ve třídě *CalculateTrendsTransform*, lze vidět na následující ukázce.

```
1  @Rule public TestPipeline p = TestPipeline.create();
2
3  private final List<TokensStatistics> INPUT = ...
4  private final List<TrendResult> EXPECTED_OUTPUT = ...
5
6  @Test
7  public void testCalculateTrendsTransform() {
8    ...
9
10   PCollection<TokensStatistics> input = p.apply(Create.of(INPUT));
11   PCollection<TrendResult> output = input.apply(new
         CalculateTrendsTransform(1.2));
12   PAssert.that(output).containsInAnyOrder(EXPECTED_OUTPUT);
13
14   p.run();
15 }
```

6.3.3 Verifikace při lokálním spuštění a na clusteru

Při lokálním spuštění je vhodné sledovat výstupy aplikace v čitelné formě. V aplikaci Core k tomuto účelu slouží nastavení parametrů *debugEnabled* a *debugOutputPath*. Pokud je první zmíněný příznak nastavení na hodnotu *true*, pak jsou ladící výstupy zapisovány v čitelném textovém formátu na příslušnou cestu uvedenou jako druhý parametr.

Obdobným způsobem lze vypisovat příslušné hodnoty i na testovacím clusteru. Samotné spuštění na clusteru je popsáno v následující kapitole.

Kapitola 7

Nasazení na cluster a výstupy aplikace

V této kapitole popíši proces nasazování aplikace na cluster. Zmíním některé zajímavé problémy, které se při nasazení objevily. Nakonec představím zajímavé výstupy aplikace, která běžela nad skutečnými daty z produkčního hledání.

7.1 Spuštění na clusteru

Společnost Seznam má dvě datová centra, která obsahují tisíce strojů. [28] Mimo produkční clustery má k dispozici různě velké testovací clustery. Pro účely této práce jsem mohl využít jeden z menších clusterů. Jeho výpočetní síla se pro tento druh výpočtu a objem dat ukázala jako dostatečná.

7.1.1 Parametry clusteru

Tabulka 7.1: Základní parametry testovacího clusteru

Správce zdrojů	YARN
Počet fyzických strojů	8
Celková paměť clusteru	352 GB
Počet virtuálních jader	176

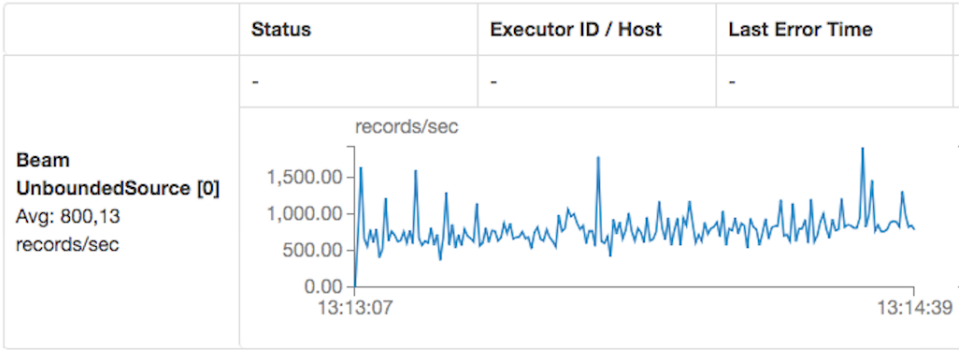
Na clusteru je nainstalovaný pouze výpočetní framework Apache Spark konkrétně verze 2.3. Z toho důvodu byl tento framework využit pro běh distribuované aplikace.

7.1.2 Batchový výpočet

Nejprve byla funkčnost programu ověřena batchovým výpočtem, jehož vstupní data byla uložená na distribuovaném souborovém systému HDFS. Distribuovaný výpočet proběhl nad daty, který byly nasbírány za větší časové intervaly, řádově hodiny a dny. Spuštění bylo pouze s jinými parametry dost podobné jako běh na lokálním počítači, pouze bylo při ladění běhu potřeba správně nastavit velikost paměti.

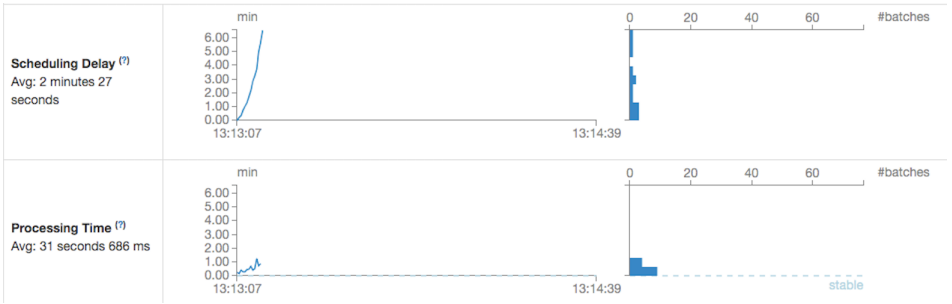
7.1.3 Streamový výpočet

Streamový výpočet se uskutečnil pod frameworkem Spark Streaming. Testovací cluster sice neměl přímo přístup do produkční Kafky, ale pouze do testovací. Z produkční Kafky jsou však téměř okamžitě data přeposílána do té testovací. To znamená, že výpočet měl k dispozici celý objem produkčních dat v reálném čase.



Obrázek 7.1: Graf ukazující vstupní datový tok z Apache Kafka

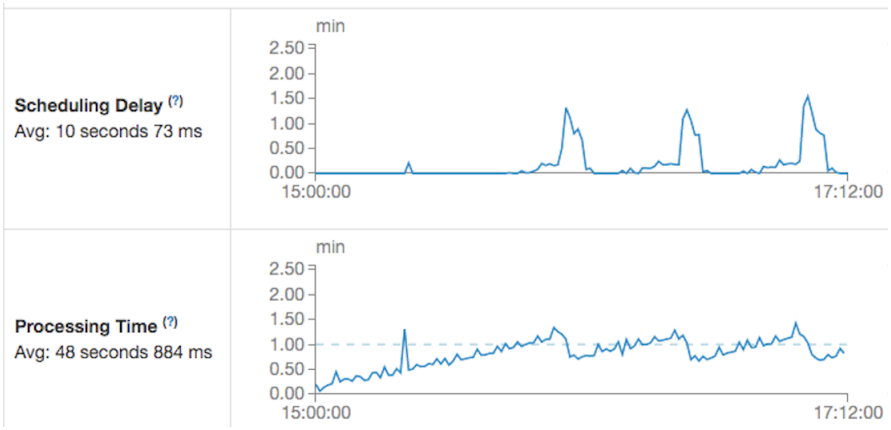
Při jednom z prvních spuštění se vyskytl zajímavý problém, kdy zpracování po krátké době začalo nabírat velké zpoždění. Z následujícího grafu je patrné, že nárůst zpoždění byl téměř od začátku dost velký.



Obrázek 7.2: Graf ukazující nárůst zpoždění výpočtu

Zpoždění bylo způsobeno tím, že velikost jedné malé dávky byla v základní konfiguraci nastavená na půl sekundy, což systém při paměti a výpočetní síle, kterou měl k dispozici, nestíhal spočítat.

Řešením bylo zvětšit velikost jedné malé dávky pomocí parametru *batchIntervalMillis*, který je uveden ve spouštěcím skriptu. Zvětšení velikosti dávky znamená, že se potenciálně zvyšuje časová prodleva o velikost dávky. Ovšem v kontextu této aplikace se jedná o zanedbatelné zpoždění. Po nastavení velikosti jedné malé dávky na hodnotu 30 a více sekund běžel výpočet stabilně. Graf zpoždění a výpočetního času vypadal následovně.



Obrázek 7.3: Graf ukazující zpoždění a dobu zpracování streamového výpočtu

7.2 Vypočtené hodnoty

Existuje mnoho možností, jak nastavit výpočetní parametry aplikace. Při výpočtech nad skutečnými daty se ukázalo, že čím kratší byla velikost okna, za kterou byly hodnoty počítány, tím více dotazů, které jsou často hledané prakticky neustále, bylo označeno jako trendující. Jako příklad takového dotazu lze uvézt *facebook*. To lze vysvětlit tím, že čím je interval kratší, tím je náchylnější na šum.

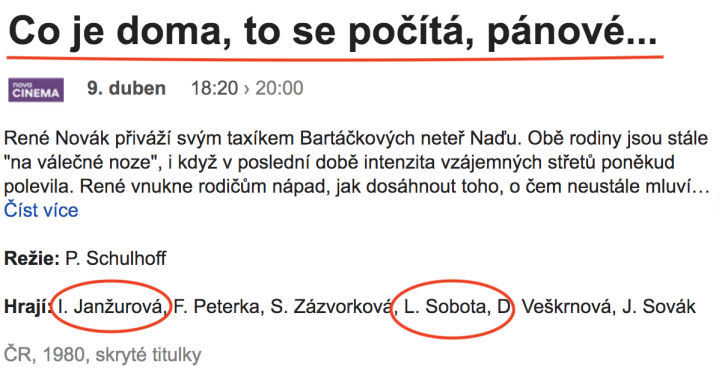
Jako celkem vhodné řešení se ukázalo počítat hodnoty za poslední hodinu s tím, že výstup je generován každých 15 nebo 30 minut. To znamená, že se hodinové okno vždy o patnáct minut posune. Zmenšuje se tím riziko, že se nepodaří odhalit důležité dotazy s nárůstem hledanosti, jak by se mohlo stát při posouvání okna o celou hodinu.

Následující tabulka obsahuje data vypočítaná v rámci dvou odpoledních hodin. Jednalo se o výpočet nad daty z 9. 4. 2019.

Zajímavé je, že zvýšený zájem koresponduje s tím, jaké pořady právě běží v televizi, a s herci, kteří ve filmech nebo seriálech hrají.

Tabulka 7.2: Vypočítané hodnoty v rámci dvou odpoledních hodin

Dotaz	Stará četnost	Nová četnost
miloš kopecký	26	71
hodonínský fotbal	2	14
co je doma to se počítá	1	23
poštovní spořitelna přihlášení	3	18
luděk sobota	3	17
iva janžurová	2	14



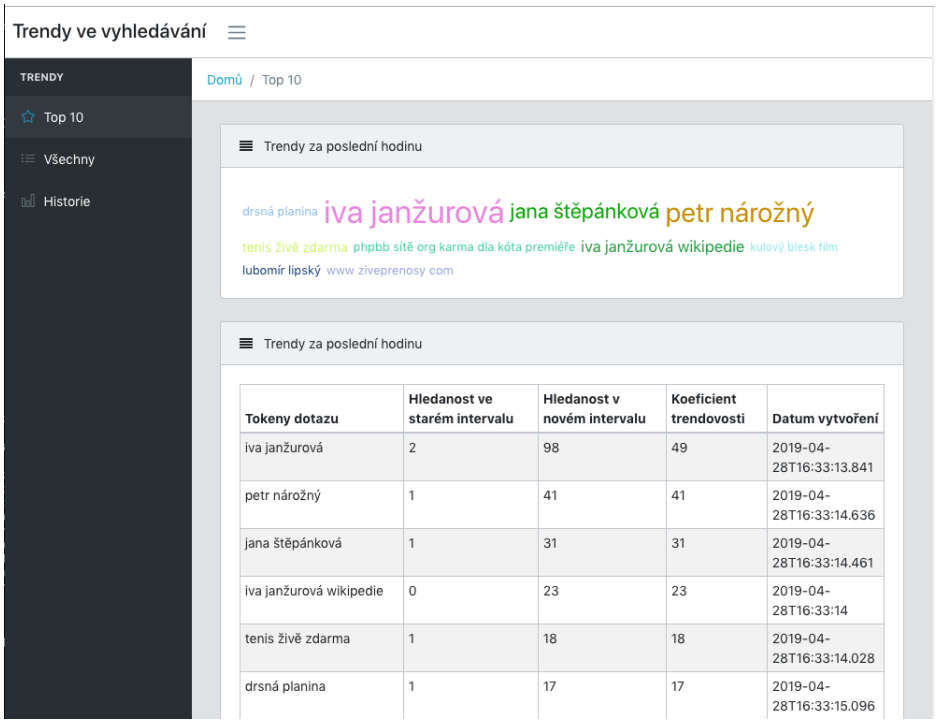
Obrázek 7.4: Výstřižek z televizního programu obsahující několik hodinových trendujících dotazů

7.3 Výstup aplikace Search trends Web frontend

Aplikace Web frontend má tři obrazovky. První obrazovka nazvaná *Trendy - Top 10* obsahuje seznam deseti dotazů za poslední hodinu, u kterých došlo k nejvyššímu nárůstu hledanosti. Kromě seznamu obsahuje také vizualizaci v podobě komponenty *tag cloud*. Tato komponenta vykresluje barevně dotazy s různou velikostí podle toho, jak jsou významné.

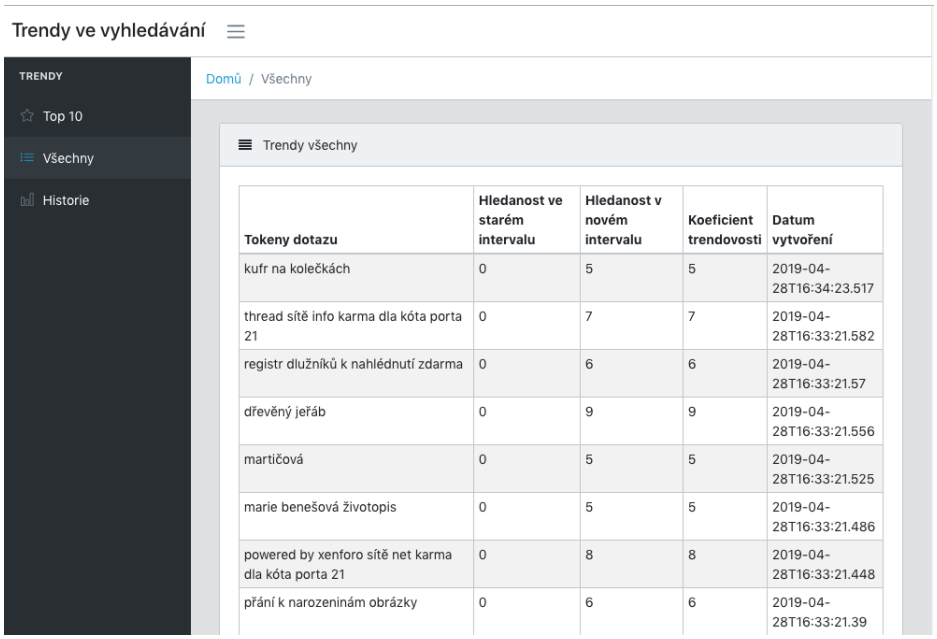
Druhá obrazovka nazvaná *Trendy - Všechny* zobrazí všechny záznamy, které byly označeny jako trendující dotazy. Seznam dotazů je seřazen sestupně podle času vzniku.

Třetí obrazovka nazvaná *Trendy - Historie* umožňuje filtrovat záznamy podle konkrétního dotazu a zobrazit, jak se jeho hodnoty vyvíjely v čase.

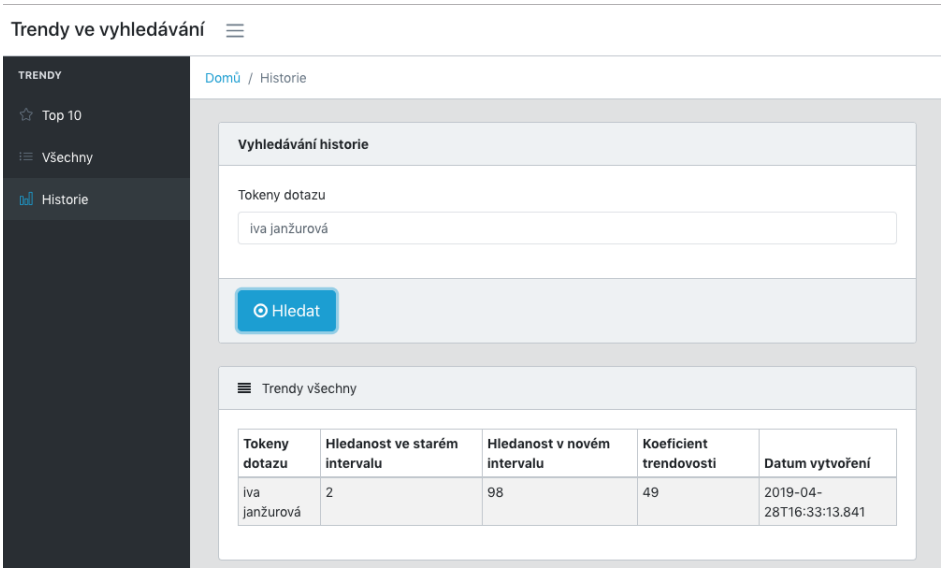


Tokeny dotazu	Hledanost ve starém intervalu	Hledanost v novém intervalu	Koeficient trendovosti	Datum vytvoření
iva janžurová	2	98	49	2019-04-28T16:33:13.841
petr nárožný	1	41	41	2019-04-28T16:33:14.636
jana štěpánková	1	31	31	2019-04-28T16:33:14.461
iva janžurová wikipedie	0	23	23	2019-04-28T16:33:14
tenis živě zdarma	1	18	18	2019-04-28T16:33:14.028
drsná planina	1	17	17	2019-04-28T16:33:15.096

Obrázek 7.5: První obrazovka aplikace Web frontend



Obrázek 7.6: Druhá obrazovka aplikace Web frontend



Obrázek 7.7: Třetí obrazovka aplikace Web frontend

Závěr

Téma závěrečné práce jsem si zvolil ze zájmu o streamové zpracování nad velkými daty. V první části práce jsem se seznámil se současnými technologiemi na zpracování velkých dat a podrobněji se zaměřil na zpracování velkých dat v reálném čase.

Dále jsem navrhl a implementoval výpočetní systém, skládající se ze tří samostatných aplikací, který načítá záznamy z fulltextového hledání společnosti Seznam.cz, a.s. a v reálném čase vypočítává dotazy, u kterých došlo k výraznému nárůstu hledanosti.

První částí je distribuovaný a škálovatelný program, který provádí výpočet nad daty z distribuovaného souborového systému nebo z transportního systému Apache Kafka. Aplikace není závislá na jednom distribuovaném výpočetním frameworku, ale lze pro její běh využít různé moderní technologie jako je Apache Spark, Apache Flink nebo Google Dataflow. Aplikace umožňuje načítání dat ve formátu Parquet, JSON a CSV a poskytuje mnoho konfiguračních možností. Přestože jejím hlavním přínosem je to, že počítá hodnoty v reálném čase, dokáže také zpracovat velké množství historických dat.

Druhá část systému je backend aplikace postavená na frameworku Spring Boot, do které jsou vypočítané hodnoty přenášeny pomocí REST API.

Třetí aplikace je frontendové webové rozhraní, sloužící k vizualizaci vypočítaných dat, které je postavené na frameworku React.

Všechny aplikace byly otestovány a nasazeny. Výpočetní část byla spuštěna na clusteru nad skutečnými produkčními daty z fulltextového hledání v reálném čase. Na základě výstupů jsem hledal a modifikoval interní parametry systému tak, aby byl výpočet co nejpřesnější.

Literatura

[1] Dean, J.; Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. 2004.

[2] Apache Hadoop: *Apache Hadoop 2.9.2 - HDFS Architecture [online]*. [cit. 2019-03-09]. Dostupné z: http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html/

[3] White, T.: *Hadoop: the definitive guide*. Sebastopol: O'Reilly, třetí vydání, ISBN 978-1-449-31152-0.

[4] Apache Spark: *Cluster Mode Overview - Spark 2.4.0 Documentation [online]*. [cit. 2019-03-23]. Dostupné z: https://spark.apache.org/docs/latest/cluster-overview.html

[5] Akidau, T.; Chernyak, S.; Lax, R.: *Streaming systems: the what, where, when, and how of large-scale data processing*. OReilly Media, Inc., 2018.

[6] Apache Kafka: *Apache Kafka Introduction [online]*. [cit. 2019-03-03]. Dostupné z: https://kafka.apache.org/intro

[7] Neha NARKHEDE, T. P., Gwen SHAPIRA: *Kafka: the definitive guide : real-time data and stream processing at scale*. Sebastopol: O'Reilly, ISBN 1491936169.

[8] Apache Beam: *Beam Programming Guide [online]*. [cit. 2019-03-16]. Dostupné z: https://beam.apache.org/documentation/programming-guide/

[9] *21 Big Data Statistics Predictions on the Future of Big Data [online]*. [cit. 2019-05-04]. Dostupné z: https://www.newgenapps.com/blog/big-data-statistics-predictions-on-the-future-of-big-data

[10] Oracle: *What is Big data [online]*. [cit. 2019-03-30]. Dostupné z: https://www.oracle.com/big-data/guide/what-is-big-data.html

[11] Sanjay Ghemawat, H. G.; Leung, S.-T.: The Google File System. 2003.

[12] Digital Ocean: *Hadoop, Storm, Samza, Spark, and Flink: Big Data Frameworks Compared [online]*. [cit. 2019-04-19]. Dostupné z: https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared

[13] Matei Zaharia, M. J. S. S. I. S., Mosharaf Chowdhury: Spark: Cluster Computing withWorking Sets. 2010.

[14] *Spark Streaming Tutorial – Sentiment Analysis Using Apache Spark [online]*. [cit. 2019-04-28]. Dostupné z: https://www.edureka.co/blog/spark-streaming

[15] Apache: *ZooKeeper: Because Coordinating Distributed Systems is a Zoo [online]*. [cit. 2019-04-18]. Dostupné z: https://zookeeper.apache.org/doc/current/zookeeperOver.html

[16] *The first release of Apache Beam! [online]*. [cit. 2019-04-28]. Dostupné z: https://beam.apache.org/beam/release/2016/06/15/first-release.html

[17] Apache Beam: *Apache Beam Overview [online]*. [cit. 2019-03-15]. Dostupné z: https://beam.apache.org/get-started/beam-overview/

[18] Aaron Wall: *Search Engine History [online]*. [cit. 2019-03-30]. Dostupné z: http://www.searchenginehistory.com/

[19] *Google vs Seznam: Jaký je podíl vyhledávačů v roce 2018 [online]*. [cit. 2019-04-20]. Dostupné z: https://www.martindomes.cz/google-vs-seznam-jaky-je-podil-vyhledavacu-v-roce-2018

[20] Google: *Trendy Google [online]*. [cit. 2019-04-14]. Dostupné z: https://trends.google.com/trends/trendingsearches/daily

[21] Seznam.cz: *Trendující témata Vyhledávání - Blog Seznam.cz [online]*. [cit. 2019-04-17]. Dostupné z: https://blog.seznam.cz/2019/03/trendujici-temata-vyhledavani/

[22] Seznam.cz: *Euphoria [online]*. [cit. 2019-04-12]. Dostupné z: https://github.com/seznam/euphoria

[23] Apache Beam: *Euphoria Java 8 DSL [online]*. [cit. 2019-04-12]. Dostupné z: https://beam.apache.org/documentation/sdks/java/euphoria/

[24] *Unirest for Java - Simplified, lightweight HTTP Request Library [online]*. [cit. 2019-04-13]. Dostupné z: http://unirest.io/java.html

[25] *Github.com - CoreUI [online].* [cit. 2019-04-20]. Dostupné z: https://github.com/coreui/coreui-free-react-admin-template

[26] *Github.com - React [online].* [cit. 2019-04-20]. Dostupné z: https://github.com/facebook/react

[27] *Travis CI - Test and Deploy with Confidence [online].* [cit. 2019-04-20]. Dostupné z: https://travis-ci.com

[28] *Na čem běží Seznam.cz: Běžný standard už nestačí, přechází na vlastní cloud i servery – Connect.cz [online].* [cit. 2019-04-17]. Dostupné z: https://connect.zive.cz/clanky/na-cem-bezi-seznamcz-bezny-standard-uz-nestaci-prechazi-na-vlastni-cloud-i-servery/sc-320-a-196137/default.aspx

Příloha A

Seznam použitých zkratek

GFS Google file system

HDFS Hadoop Distributed File System

CPU Central processing unit

API Application programming interface

JAR Hadoop Distributed File System

RDD Resilient Distributed Dataset

SQL Structured Query Language

SDK Software development kit

URL Uniform Resource Locator

WWW World Wide Web

HTML Hypertext Markup Language

XML Extensible Markup Language

IP Internet Protocol

FTP File Transfer Protocol

PoC Proof of Concept

JVM Java virtual machine

CSV Comma-separated values

JSON JavaScript Object Notation

HTTP Hypertext Transfer Protocol

REST Representational State Transfer

ORM Object-relational mapping

CRUD Create, read, update and delete

CI Continuous integration

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `search-trends-core` ..... zdrojové kódy části Core
  - `search-trends-web-backend` ..... zdrojové kódy části Web backend
  - `search-trends-web-frontend` ..... zdrojové kódy části Web frontend
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF