Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Umístění dat na výpočetní uzly minimalizující datové přenosy v databázi HBase

Bc. Miroslav Hrstka

Vedoucí práce: Ing. Adam Šenk

3. dubna 2015

Poděkování

Doplňte, máte-li komu a za co děkovat. V opačném případě úplně odstraňte tento příkaz.

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 3. dubna 2015 .....

Odkaz na tuto práci

Hrstka, Miroslav. *Umístění dat na výpočetní uzly minimalizující datové přenosy v databázi HBase.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

V několika větách shrňte obsah a přínos této práce v češtině. Po přečtení abstraktu by se čtenář měl mít čtenář dost informací pro rozhodnutí, zda chce Vaši práci číst.

Klíčová slova Nahraďte seznamem klíčových slov v češtině oddělených čárkou.

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

Keywords Nahraďte seznamem klíčových slov v angličtině oddělených čárkou.

Obsah

Úvod **1**

1 Současný stav a použité technologie **3**
1.1 Hadoop Ekosystém 3
1.2 HDFS 7
1.3 HBase 8

2 Existující řešení optimalizace distribuce dat pro MapReduce **13**
2.1 Definice problému 13
2.2 MR-part 14
2.3 Testování a výsledky 16

3 Návrh řešení pro HBase **19**
3.1 Klíčová specifika HBase pro návrh řešení 19
3.2 Proces optimalizase 20

4 Implementace řešení **21**

5 Testování a vyhodnocení měření **23**

Závěr **25**

Literatura **27**

A Seznam použitých zkratek **29**

B Obsah přiloženého CD **31**

Seznam obrázků

1.1 Diagram MapReduce procesu 5
1.2 Hadoop Ekosystém 6
1.3 Diagram uložení souboru v systému HDFS 8
1.4 Datový model HBase 9
1.5 Architektura databáze HBase 9
1.6 Příklad B+ stromu. 10
1.7 Iterativní Merge Multi-Page bloků v LMS stromu 11
1.8 Schéma ukládání dat z HBase v HDFS 11
1.9 Formát souboru HFile 12
1.10 Rozložení Hfile do bloků dat v HDFS 12
1.11 Formát KeyValue struktury 12

2.1 MR-part schéma 14
2.2 Pseudokód algoritmu pro Metadata Combination 15
2.3 Pseudokód algoritmu pro Repartitioning 16

Úvod

uvod moji prace :P

Kapitola 1

Současný stav a použité technologie

V této úvodní kapitole je dán prostor pro seznámení s technologiemi a projekty, se kterými se bude buď přímo pracovat nebo je jejich znalost pro pochopení problematiky nezbytná. Jako první je představen projekt Hadoop™ od firmy Apache™ jako celek. Vše, co bude v této práci představeno, se bude odehrávat v rámci tohoto takzvaného ekosystému. Pro porozumění základní myšlenky projektu Hadoop je také nezbytné vysvětlit programovací model MapReduce. Po uvedení celého Hadoopu jsou detailněji uvedeny produkty, které jsou součástí tohoto ekosystému a se kterými se bude dále pracovat. Jedná se především o databázový systém HBase a také souborový systém HDFS, který je v celém modelu využíván.

1.1 Hadoop Ekosystém

Apache Hadoop je framework který sdružuje projekty vyvíjející software pro spolehlivé, škálovatelné a paralelní zpracování dat na počítačových clusterech. Je založen na dvou stěžejních technologiích pocházejících od firmy Google a to na distribuovaném souborovém systému Google File System (GFS) a na algoritmu MapReduce[1]. Všechny klíčové projekty v systému Hadoop jsou združeny pod Apache Software foundation, která poskytuje podporu pro tyto projekty. Jedná se o open.source software a všechny komponenty jsou psány v programu Java.

1.1.1 Základní principy

Podstata Hadoopu spočívá v uložení dat na velkém množství výpočetních úzlů spojených do clusterů. Většinou se jedná o běžný hardware. Na těchto uzlech jsou data uložená ve vlastním souborovém Systému HDFS. K výpočtům nad clusterem se využívá princip Mapreduce, který bude osvětlen v následující

kapitole. Systém Hadoop je charakteristický především následujícími vlastnostmi, které ho odlišují od klasických databázových systémů.

Horizontální škálovatelnost a komoditní hardware
Pro objemy dat, jimiž by se měl Hadoop při svém zpracovávání primárně zabývat, je poměrně složité a především velmi drahé dosáhnout dostatečné škálovatelnosti pomocí vertikálního škálování, tedy přidávání výkonu a zdrojů ke stávajícím výpočetním uzlům. Proto Hadoop využívá horizontálního škálování. Díky horizontálnímu škálování se nabízí využití komoditního hardwaru namísto specializovaných výpočetních uzlů. Systém tedy běží na velkém množství samostatných počítačů spojených do clusteru.

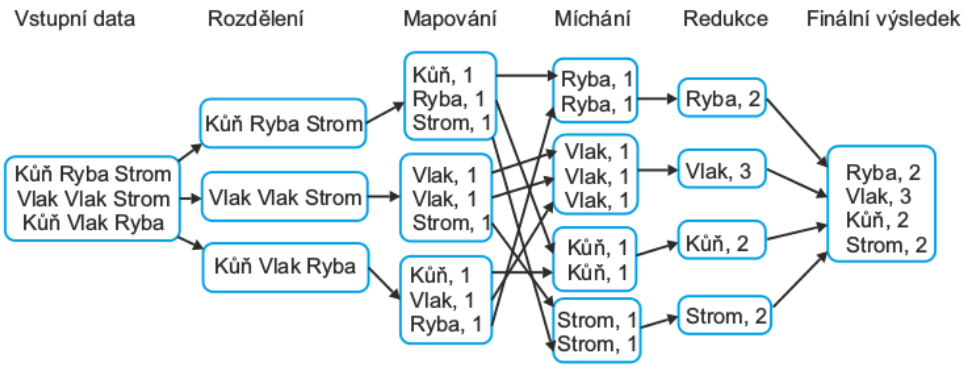
Řešení selhání hardwaru
S předchozím bodem úzce souvisí řešení případných výpadků jednotlivých uzlů. Kvůli velkému počtu výpočetních uzlů a díky použití běžného hardware jsou výpadky poměrně časté. Hadoop je ale navržen tak, že se nesnaží tyto výpadky minimalizovat, ale počítá s nimi. Data jsou dostatečně replikovány a pokud dojde k výpadku při zpracování dat, je přerušená úloha provedena na jiném uzlu obsahující danou replikaci a zároveň je automaticky vytvořena nová záloha dat. Defaultně je zálohovací faktor nastaven na `3` , tedy každý soubor je uložen ve třech kopiích na různých výpočetních uzlech.

Přenášení kodů k datům
Kvůli velkým objemům dat a poměrně nízké propustnosti sítě spojující jednoltivé výpočetní uzly v clusteru je velmi výhodné namísto rozesílání dat na jednotlivé výpočetní uzly rozesílat na ně pouze kód výpočtu a minimalizovat tak přesuny dat mezi uzly. Na každém uzlu je tak vykonán výpočet pokud možno s lokálními daty.

Abstrakce od distribuovaných a paralelních aplikací
Hadoop se snaží co nejvíce odstínit vývojáře Hadoop aplikací od řešení zpracování pomocí paralelního a distribučního zpracovaní. Proto poskytuje poměrně jednoduché a dobře definované rozhraní pro jednotlivé komponenty. Při práci s těmito rozhraními tak není nutné řešit, jak se bude kód v clusteru distrubuovat ani další záležitosti spojené s paralelním zpracováním a dovoluje se zaměřit na business logiku aplikace. Cenou za toto zjednodušení je pak právě omezené rozhraní bez možnosti detailnějšího nastavení.

1.1.2 MapReduce

MapReduce je programovací paradigma určené k provádění výpočtů, které by za normálních okolností trvaly značné množství času a to zejména z důvodů



Obrázek 1.1: Diagram MapReduce procesu

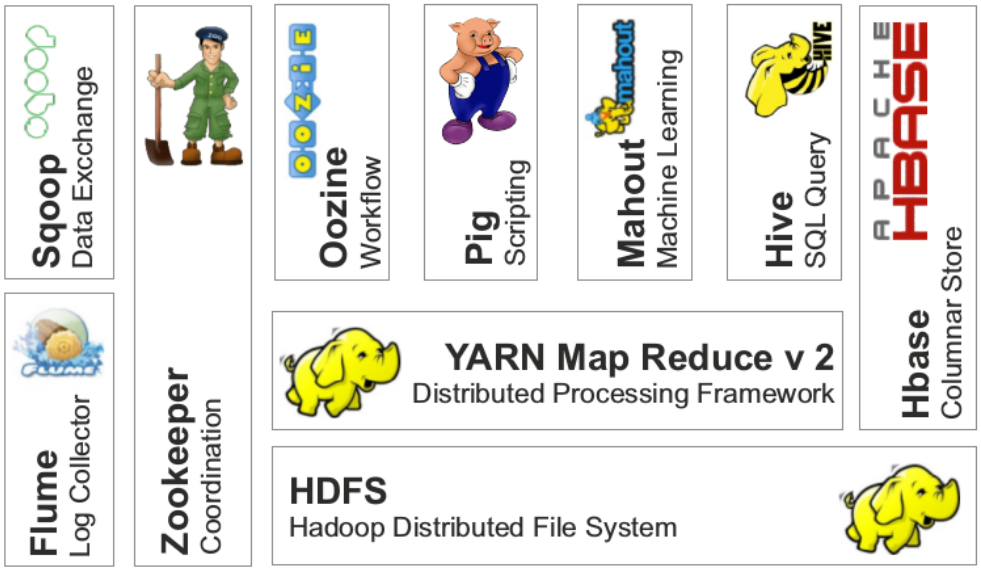
velkého množství dat. Tyto výpočty se pak snaží dokončit v přijatelném časovém horizontu. Princip byl poprvé představen v roce 2004 v práci pro firmu Google jako jedno z opatření pro zvládání obrovského množství dat, se kterým se museli potýkat.

V MapReduce jsou data modelována do párů klíč/hodnota. Tento formát je velmi jednoduchý, přesto se dají téměř všechny údaje takto prezentovat. Tato jednoduchá datová struktura tak umožňuje snadné zpracování dat efektivním způsobem. Klíč a hodnota může být cokoliv. Může se jednat o řetězce, čísla nebo komplexní struktury.

Mapreduce se skládá ze dvou hlavních fází, mapovací fáze a redukční fáze. Nejdříve je spuštěna mapovací funkce, která je dále spočtena nad jednotlivými prvky z množiny uspořádaných dvojic (klíč, hodnota), jež produkuje přechodné klíče a hodnoty.[2] Poté jsou všechny tyto přechodné hodnoty asociované se stejným klíčem seskupeny a následně poslány se do redukční fáze. Redukční fáze přijme na vstupu klíč a související množinu hodnot. Jako výstup je pak množina uspořádaných dvojic, která splňuje zadaná kritéria.

Jak vstupní data, tak mezivýsledky i finální výsledky jsou ve formátu klíč/hodnota. Jak je vidět na obrázku 1.1 probíhají mapovací a redukční fáze paralelně. Z diagramu je také zřejmé, že mezi mapovací fází a fází míchání(shuffling) dochází k přenosu průběžných výsledků mezi jednotlivými výpočetními uzly. Právě optimalizací přesunů v těchto místech se bude zabývat hlavní část této práce. MapReduce je také často popisována funkcí:

$$map : (k_1, v_1) \rightarrow list(k_2, v_2)$$
$$reduce : (k_2, list(v_2)) \rightarrow list(k_3, v_3)$$



Obrázek 1.2: Hadoop Ekosystém

1.1.3 Hadoop Ekosystém

Následující kapitola je určena k seznámení s ekosystémem Hadoop, který kromě základních projektů MapReduce a HDFS obsahuje množství dalších projektů. Složení ekosystému se ale liší v závoslosti na konkrétní distribuci Hadoopu. Je sice možné zvolit si tyto aplikace podle svého výběru a využít přímo zdroje od firmy Apache, ale v praxi se využívají spíše již částečně nakonfigurované distribuce. které navíc nabízejí i možnost placené podpory. Mezi největší hráče na trhu patří distribuce od firem Cloudera, MapR a Hortonworks.[3] Pro účely této práce byla vybrána distribuce Cloudera, protože se jedná o open-source projekt a také kvůli největšímu podílu na trhu.1.2

Nástroje pro vývoj

YARN
: YARN je klíčovým prvkem Hadoop 2. Někdy je také zvaný MapReduce v2. Jedná se o distribuovaný operační systém, který odděluje řízení zdrojů a řízení kapacit od zpracovávající komponenty. To umožňuje podporovat větší škálu přístupů ke zpracování dat a širší pole aplikací.

Hive
: Hive umožňuje dotazování nad velkými datasety uloženými v distribuovaném systému a také jejich řízení. Poskytuje mechanismus k vytvoření struktury nad těmito daty a následně nad daty provádět dotazy v SQL-like jazyku HiveQL. Kromě toho umožňuje také využití klasického map/reduce postupu v případech, kdy není výhodné použít HiveQL.

Pig

Pig poskytuje prostředí pro zpracování jednoduchého skriptovacího jazyka Pig Latin, ve kterém je přeložen na sérii MapReduce úloh. Pig Latin abstrahuje od MapReduce schématu a nabízí dotazování na vyšší úrovni, podobné jako SQL.

Mahout

Mahout je škálovatelná knihovna pro strojové učení. Jsou v ní implementovány algoritmy pro clustering, klasifikaci a kolaborativního filtrování optimalizované pro běh v prostředí Hadoopu.

Ukládání dat a správa metadat

HDFS

Jedná se o distribuovaný souborový systém navržený pro provoz na komoditním hardwaru ve velkých datových skladech, souborový systém HDFS bude detailněji představen v následující kapitole.

HBase

Hbase je sloupcově orientovaný databázový systém, který běží nad HDFS. Nepodporuje strukturovaný dotazovací jazyk a poskytuje prakticky pouze CRUD operace. HBase bude stejně jako HDFS představen detailněji v následujících kapitolách.

Nástroje pro řízení

ZooKeeper

Poskytuje provozní služby pro Hadoop cluster. Jedná se o distribuované konfigurační, synchronizační služby a o jmenné registry pro distribuovaný systém.

Oozie

Aplikace používaná pro plánování Hadoop úloh. Je složena ze dvou hlavních částí. V první části se ukládají a spouštějí různé typy hadoop úloh (Mapreduce, Pig, Hive, atd.) a z části, která koordinuje běh daných úloh na základě předdefinovaných plánů a dostupnosti dat.

Získávání a agregace dat

Sqoop

Nástroj sloužící k efektivnímu přenosu dat z relačních databází do Hadoopu k dalšímu zpracování. Zpracovat tyto data pak může buď MapReduce úloha nebo jiný nástroj (Hive, Pig). Je také možné data vložit do HBase databáze.

Flume
Služba pro efektivní získávání, agregování a přesouvání velkého množství streamovaných dat do HDFS. Typicky se používá k ukládání logů z jednoho zdroje (webové logy, bankovní logy) a agreguje je v HDFS pro pozdější zpracování.

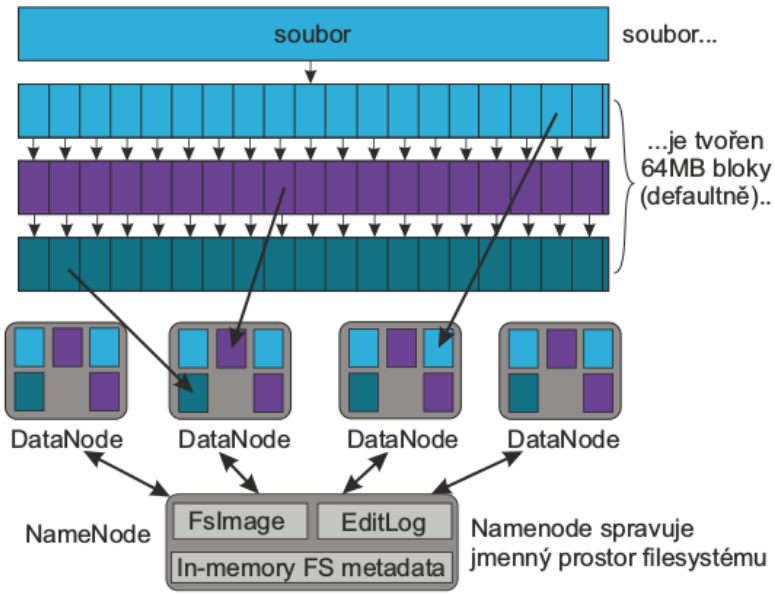
1.2 HDFS

Hadoop Distributed File Systém (HDFS) nabízí způsob skladování velkých souborů na více samostatných počítačích, který je rozdílný oproti klasickému přístupu skladování dat na jednom stroji s dostatečnou diskovou kapacitou. HDFS je navržen na základech Google File Systemu (GFS) a běží na nativním filesystému (Ext3, Ext4,XFS). HDFS je určen pro skladování především velkých souborů (100 MB a více) v menším počtu (řádově miliony) a k ukládání streamovaných dat. Systém dále není vhodný pro soubory, u nichž se očekává časté upravování, a to protože je možné připisovat data pouze na konec souboru. Systém je odolný proti chybám v replikaci a výpadkům v distribuci dat.[4]

HDFS umožňuje, stejně jako většina běžných souborových systémů, operace čtení, zápisu a mazání souborů a vytváření a mazání adresářů. Vždy, když je načten nový soubor do HDFS, je zreplikován do žádoucího počtu, který určuje replikační faktor (defaultní hodnota je 3) a rozdělen do bloků dat o fixní délce (defaultně 64MB). Tyto bloky jsou pak rozdistribuovány a uloženy ve vybraných uzlech clusteru určených pro skladování, tzv. DataNodes viz.1.3 . V HDFS se informace o souborech neukládají společně s daty, ale jsou uloženy na vyhrazeném serveru nazývaném NameNode. Při přístupu k datům klient nejdříve zadá požadavek na data na NameNode, který následně vrátí adresy datablokú s požadovanými daty. NameNode tedy přímo nemanipuluje s daty.

NameNode uchovává a poskytuje strom jmenného prostoru a adresy fyzického umístění bloků ve své operační paměti. Dále ukládá perzistentní záznam těchto adres (kontrolní bod) a registr modifikací (žurnál) pro zotavení z havárie v nativním systému souborů hostitelského počítače. HDFS umožňuje i vytvoření kopie kontrolního bodu a žurnálu na další výpočetní uzel nazývaný SecondaryNameNode. Ten pak slouží jako záloha dat serveru NameNode (nenahrazuje tedy funkci primárního NameNode v případě výpadku, pouze poskytuje data pro jeho obnovu). Ve verzi Hadoop 2+ je už možné mít Standby NameNode, který v případě výpadků může primární NameNode plně a okamžitě nahradit.

Přistupovat k HDFS je možné přímo a to přes nativního klienta nebo pomocí Java nebo C++ API. Dále je možný přístup přes proxy server podporující REST, Thirft a Avro server.



Obrázek 1.3: Diagram uložení souboru v systému HDFS

1.3 HBase

Jedná se o sloupcově orientovanou databázi, někdy označovanou jako Hadoop databáze. HBase podporuje náhodné real-time CRUD operace (narozdíl od HDFS). Je primárně navržená pro uchovávání velkých tabulek o miliardách řádců a milionech sloupcích a jedná se o NoSQL databázi. Nepodporuje tedy přístup založený na SQL jazycích ani relační model. Stejně jako HDFS se vyznačuje jednoduchým klientem a Java API. HBase je založena na projektu Bigtable od Googlu a stejně jako byl Bigtable postaven nad GFS je HBase postavena nad HDFS.[5]

HBase nebyla zavedena za účelem nahrazení klasických RDBMS a ani k tomuto účelu není využívána. HBase je výhodné použít, jak již bylo řečeno, v případě rozsáhlých tabulek. Výborné výsledky vykazuje při vykonávání jednotlivého náhodného výběru z databáze a při vyhledávání dle klíče. Hbase je také vhodným řešením v případě, že jednotlivé řádky tabulky jsou velmi různorodé a v případě řidkých databází, kdy je velký počet sloupců a většina z nich obsahuje nulovou hodnotu. Nevhodné využití je pak právě pro suplování úloh pro tradiční RDBMs jako jsou transakční aplikace nebo relační analýza.[6]

1.3.1 Data model HBase

Data v databázi HBase jsou uložena v tabulkách. Jednotlivé tabulky obsahují řádky. Na každý řádek odkazuje unikátní klíč. Jako hodnota klíče se bere bi-

KEY	Timestamp	JMÉNO (FAMILY)			ADRESA (FAMILY)		
		Titul	Jméno	Příjmení	Adresa	Číslo	PSČ
KEY 001	t 1	Bc.	Franta	Omáčka	Kostomlaty	33	413 01
	t 2	Mgr.					
	t 5						
KEY 002	t 10		Laura	Tlustá	Praha	133	15000
	t 15				Olomouc	480	68300
	t 17			Hubená	Brno	1153	77100

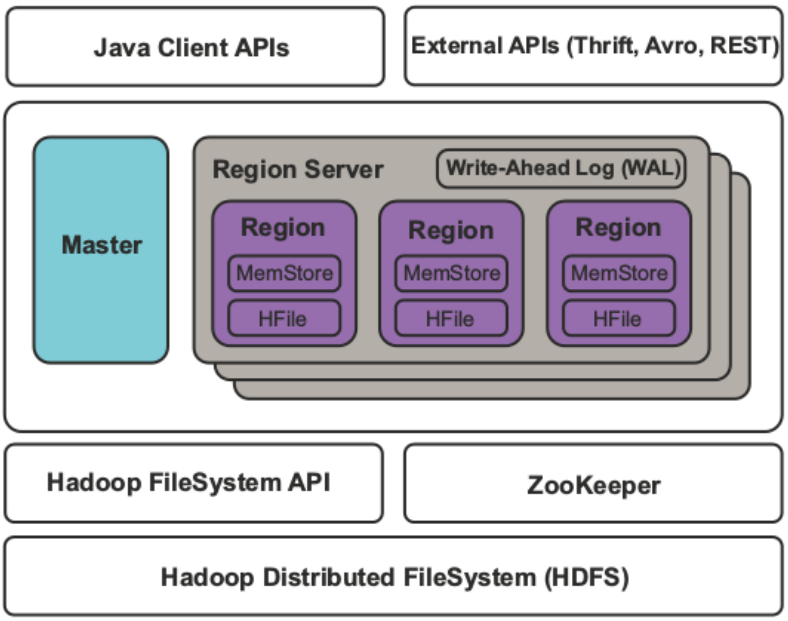
Obrázek 1.4: Datový model HBase

tové pole. Klíč u HBase tedy může být cokoli, string, long nebo vlastní datová struktura. Každý řádek je složen ze sloupců, které jsou sdruženy do rodin (column families). Tyto rodiny sloupců jsou definovány staticky při vytváření databáze narozdíl od samotných sloupců, které se mohou přidávat libovolně. Data jako taková jsou pak uložena v buňkách. Tyto buňky jsou identifikovány pomocí řádku, rodiny, sloupce a časovou značkou(timestamp). Obsah každé buňky je pak uchováván také jako pole bitů. Data v buňkách jsou navíc verzovány. Každá buňka defaultně uchovává poslední tři zadané hodnoty s tím, že pokud není v dotazu specifikována konkrétní verze, vrací vždy tu nejmladší. Řádky jsou v každé tabulce seřazeny lexikograficky podle svého klíče. Příklad takové tabulky je uveden v obrázku 1.4.

1.3.2 HBase Architektura

HBase je distribuovaná databáze. Proto je i architektura složitější než u databází běžících na jednom výpočetním uzlu. HBase musí řešit všechny problémy typické pro distribuované aplikace jako je koordinace a řízení vzdálených procesů, blokování, distribuce dat a příliš velká síťová komunikace. HBase však k tomuto z velké části využívá služeb v rámci Hadoop a Zookeeper. Následující obrázek 1.5 popisuje hlavní architektonické komponenty HBase.

Jednotlivé tabulky jsou složeny z regionů. Region je vždy určitý rozsah řádků uložený pohromadě. Protože jsou řádky v databázi ukládány v lexikografickém pořádí, je nutné počítat s tím, že se velikost těchto rozsahů, tedy regionů, bude v čase měnit. Proto se v případě, kdy velikost regionu překročí stanovenou hranici, rozdělí region na dva přesně v půli podle prostředního klíče. Naopak v případě, kdy se regiony příliš zmenší, dojde k jejich sloučení. Regiony jsou uložené v region serverech. Každý region server může obsahovat jeden a více regionů. Region je však vždy jen na jednom serveru. Master server je zodpovědný za správu region serverů. Pro koordinaci se využívá Zookeeper. Na každém regionu je uložen určitý rozsah klíčů. Rozdělení dat do region serverů umožňuje rychlou obnovu v případě pádu region serveru a také ulehčuje load balancing pokud dochází k přetěžování některých serverů. Všechny tyto



Obrázek 1.5: Architektura databáze HBase

činnosti včetně rozdělování velkých regionů jsou prováděny automaticky bez zásahu uživatele.

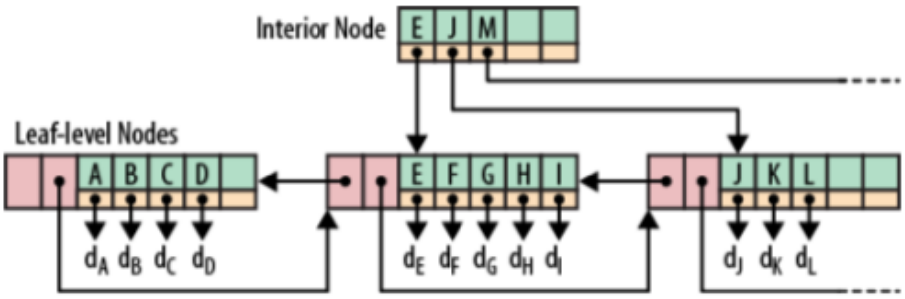
1.3.3 Architektura uložiště RDBMS vs HBase

Dříve než bude uveden detailní pohled na uložení dat v databázi HBase, je na místě popsat základní rozdíl mezi typickou architekturou uložení dat v RDBMS a v HBase. Typické RDBMS ukládá data do struktury B+ Trees oproti tomu HBase a ostatní Big Table architektury využívají Log-Structured Merge Trees[7].

1.3.3.1 B+ Trees

Jedná se o stromovou datovou strukturu, která vychází z B-stromu. Umožňuje rychlé vkládání, vyhledávání a mazání dat. Záznamy v tabulkách jsou identifikovány za pomoci klíčů. Všechna data jsou uložena jen na listech stromu, oproti klíčům, které jsou uloženy i ve vnitřních uzlech. V implementaci těchto stromů se přidává do všech listů kromě vlastních klíčů i odkaz na následujícího sourozence. To umožňuje velmi efektivní sekvenční prohledávání aniž by výrazněji stoupla paměťová náročnost na uložení stromu, odkaz na sourozence je v obrázku znázorněn červenými políčky 1.6.

V B+stromech je data-locality dosažená na úrovni stránek, kde stránky odpovídají blokům v jiných systémech. Stránka tak může vypadat například takto:



Obrázek 1.6: Příklad B+ stromu. Obsahuje data d_1 až d_7

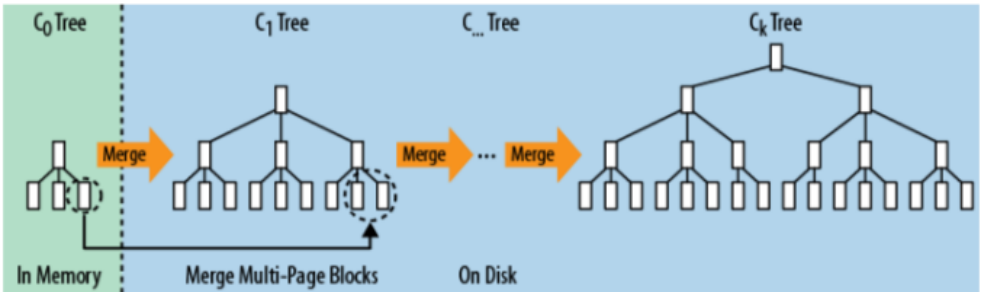
- Odkaz na předchozí stránku
- Odkaz na následující stránku
- key: A $\rightarrow d_A$
- key: B $\rightarrow d_B$
- key: C $\rightarrow d_C$
- key: D $\rightarrow d_D$

Vždy, když se vkládá nový záznam, doplní se daná stránka požadovaným záznamem. Pokud dojde k případu, že stránka už je plná, rozdělí se na 2 poloprázdné stránky a upraví se patřičně rodič těchto stránek. Tímto dochází k fragmentaci dat na disku, když jednotlivé logicky sousedící stránky neleží vedle sebe fyzicky.

1.3.3.2 Log-Structured Merge Trees

Log-Structured Merge Trees využívají způsob odlišný od B+ stromů. Všechny příchozí data jsou nejprve ukládány v logovacích souborech a to kompletně sekvenčně. Jakmile jsou informace uložené v logu, uloží se data v in-memory uložišti, kde jsou uloženy naposledy upravené záznamy. Vždy, když je k dispozici dostatečný počet záznamů, dojde k zapsání už seřazených dat do uložiště souborů. Po zapsání dat na disk je log soubor smazán.

Datové soubory jsou pak strukturovány podobně jako B stromy, s tím že jsou optimalizovány pro sekvenční přístup na disku. Všechny uzly stromu jsou zaplněny a uloženy v sigle-page nebo multi-page bloku. Při přidávání dat se uložená data na disku v multi-page blocích spojí s příchozími in-memory daty. Tento proces je znázorněn na obrázku 1.7. Vždy, když data využijí celou kapacitu bloků, dojde k vytvoření nového. Postupně se tak zvyšuje počet vytvořených souborů. Tyto soubory jsou pak spojovány do větších celků. Všechny tyto soubory jsou uloženy sekvenčně za sebou a tak je umožněn optimální



Obrázek 1.7: Iterativní Merge Multi-Page bloků v LMS stromu

sekvenční přístup k datům. Strom může být také rozdělen v případě, kdy je potřeba vložit nová data s klíči mezi ostatními. Mazání záznamů se provádí dávkově. Každý záznam, který je určený ke smazání je označen jako smazaný, a při nahlížení do dat se ignoruje. Při přepisování stránek pak dojde k odstranění těchto záznamů ze stromu.

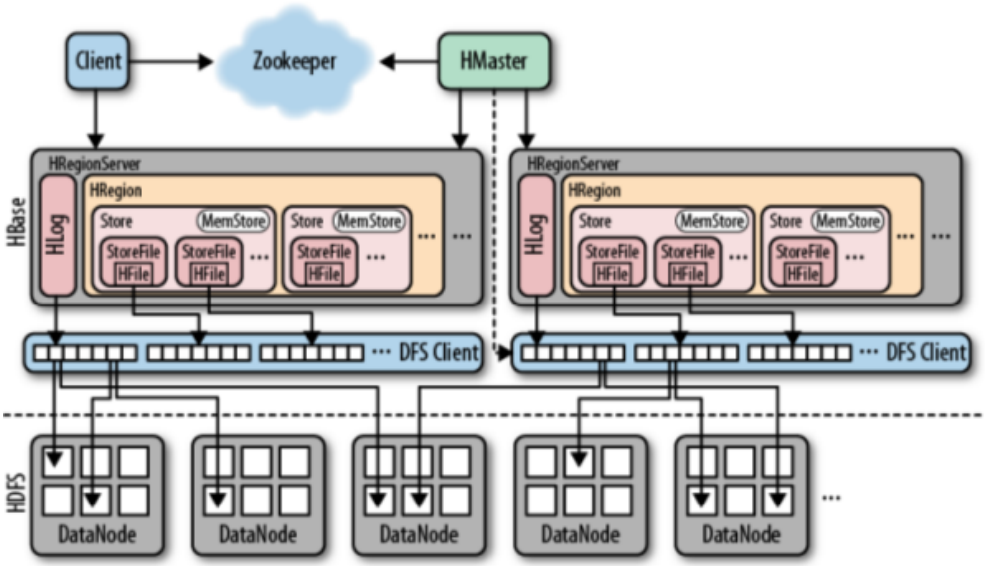
Z těchto rozdílných přístupů k uložení dat vyplývá, že B+ stromy jsou určeny pro úlohy, kde se očekává častá modifikace již vložených dat, zatímco LSM stromy jsou určeny k ukládání velkého množství dat a jeho následovnému čtení.

1.3.4 Fyzické uložení dat

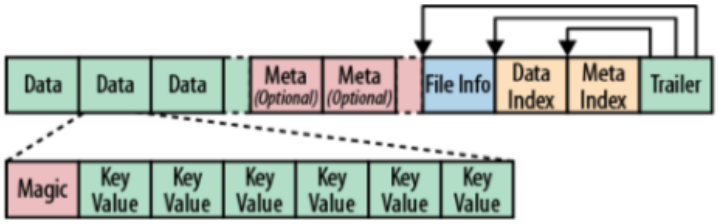
Pro většinu uživatelů je forma fyzického uložení dat v HBase zcela zkrytá a k plnohodnotnému využívání databáze prakticky nevýznamná. Avšak pro potřeby této práce, je nezbytné tuto část osvítit více, než je zapotřebí v běžném využití. Na obrázku 1.8 je znázorněno schéma uložení dat, které zobrazu je uložení dat v souborovém systému HDFS. Hbase pracuje především s dvěma hlavními typy souborů. se soubory HLog reprezentující write-ahead log (WAL) a HFile pro uložení dat.[8]

1.3.4.1 formát HFile

Formát souboru HFile je navržen tak, aby ukládal data co nejefektivněji. Je založen na formátu souboru TFile z Hadoopu. Soubor obsahuje několik bloků,jak je vidět n obrázku 1.9, z nichž jsou fixní pouze info a trailer bloky. Tyto bloky jsou uloženy na konci souboru a ukončují tak daný soubor, který se stává neměnným. V index blocích jsou uloženy offsety data a metadata bloků. Jak metadata bloky tak i data bloky jsou nepovinné, nicméně už z podstaty věci jsou data bloky téměř vždy součástí Hfile souboru. Velikost jednotlivých bloků je defaulrtně definována na 64 KB. Je možné tuto velikost změnit v závislosti na očekávané struktuře dotazů na databázi. Čím větší bloky, tím efektivnější bude sekvenční prohledávání databáze, naopak náhodný přístup bude efektivní méně. V jednotlivých blocích se sekvenčně ukládají KeyValue instance



Obrázek 1.8: Schéma ukládání dat z HBase v HDFS

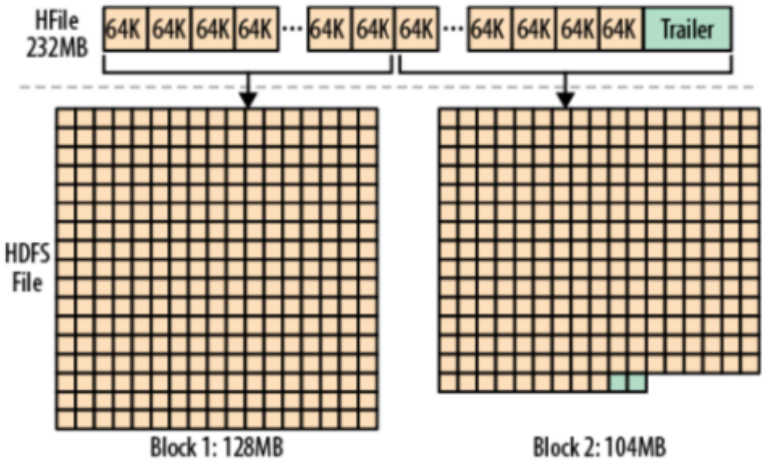


Obrázek 1.9: Formát souboru HFile

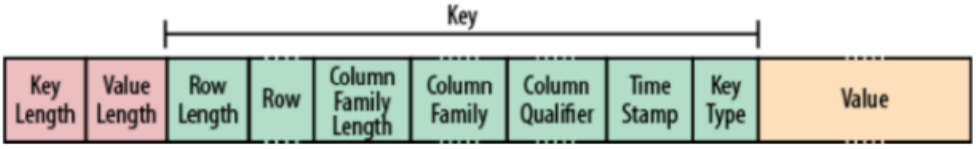
v KeyValue formátu. Velikost bloků není fixní, odvíjí se od velikosti poslední vložené key value instance, která se do bloku uloží celá a pak teprve dojde k uzavření bloku. V praxi tedy boky bývají o něco větší než je definovaná hodnota. Je možné také použít libovolný komprimační algoritmus na ukládané data, což ale neovlivní celý proces, protože checksum je dopočítáván až po přidání poslední key value instance.

Ačkoli se nabízí myšlenka, že velikost bloků v HFile nastavená na 64KB souvisí z velikostí souborů v HDFS, který je nastaven na 64 MB, není mezi těmito hodnotami žádná spojitost. HFile jsou v HBase ukládány jako běžné soubory a HDFS je vnímá pouze jako binární data. Na obrázku 1.10 je zobrazeno schéma ukládání HFile souborů do HDFS.

Formát KeyValue KeyValue je struktura, která reprezentuje fyzické uložení jedné buňky z tabulky. Z obrázku1.11 vyplývá, že jako část klíče se po-



Obrázek 1.10: Rozložení Hfile do bloků dat v HDFS



Obrázek 1.11: Formát KeyValue struktury

užívá mimo jiné název tabulky, family a sloupce. Tento klíč se vyskytuje u každé buňky a proto se doporučuje, mít vše pojménováno co nejkratšími identifikátory.

1.3.4.2 Write-Ahead Log

Region servery ukládají příchozí data do paměti, do doby něž se jich nahromadí dostatečné množství a poté je najednou zapíše na disk. Ve fázi, kdy se data hromadí v paměti, jsou ale zranitelná a v případě výpadku dojde k jejich ztrátě. Právě pro zamezení tohoto problému se používá WAL. Jeho funkce tedy je omezit dopady výpadku. Do logu se proto zapisují všechny změny dat, a až v případě, že jsou úspěšně uloženy na disk, je klient informován o úspěšném uložení dat.

Kapitola 2

Existující řešení optimalizace distribuce dat pro MapReduce

V této kapitole bude představeno řešení, které poskytl M. Liroz-Gistau et al. ve svém článku Data Partitioning for Minimizing Transfered Data in MapReduce [9]. V tomto článku se zaměřují na redukování datových přenosů mezi mapovací a redukční fází. V tomto místě přichází na řadu fáze, kde probíhá míchání přechodných klíčů(shuffle phase). Od tohoto řešení se bude poté odvíjet návrh řešení pro databázi HBase. Protože v předchozích kapitolách už byl vysvětlen princip zpracování dat pomocí Map Reduce, může tato kapitola plynule navázat na tyto poznatky.

2.1 Definice problému

Nejdříve je zapotřebí fommálně definovat, jaký problém chceme řešit. Mějme tedy sadu MapReduce úloh, které reprezentují typické zatížení systému a sadu vstupních dat. Předpokládejme, že budoucí MapReduce úlohy budou vykonávat podobné úlohy na podobných datech a budou generovat podobné intermediate key (předpokládá se, že v praxi se vykonávají pořád stejné úlohy, jen dochází například ke zvětšování datasetu o nově zapsané data).

Cílem navrhovaného systému je automatické rozdělení vstupních dat tak, aby u budoucího vykonávání MapReduce úloh byl minimalizován přenos dat mezi jednotlivými uzly v shuffle fázi. Při tomto rozdělování se nebere v úvahu plánování mapovacích a redukčních fází, ale pouze inteligentní rozdělení intermediate klíčů mezi jednotlivé redukční uzly.

Definujme daný problém formálně. Mějme vstupní data pro MapReduce úlohu job_α složená z jednotlivých souborů $D = \{d_1, ..., d_n\}$, které jsou rozděleny do množiny bloků (chunks) $C = \{c_1, ..., c_p\}$. Funkce $loc : D \to C$ přiřazuje data do bloků. Nechť job_α je složen z $M_\alpha = \{m_1, ..., m_p\}$ mapovacích úloh a $R_\alpha = \{r_1, ..., r_q\}$ jsou redukční úlohy. Předpokládejme, že každá mapovací

úloha m_i zpracuje blok c_i. Nechť $N_\alpha = \{n_1, ..., n_s\}$ je množina výpočetních uzlů použitých pro provedení úlohy. $node(t)$ reprezentuje výpočetní uzel, kde se vykonává úloha t.

Nechť $I_\alpha = \{i_1, ..., i_m\}$ je množina intermediate párů klíč-hodnota produkovaných mapovací fází jako je $map(d_j) = \{i_{j_1}, ..., i_{j_t}\}$. $k(i_j)$ reprezentuje klíč z intermediate páru i_j a $size(i_j)$ reprezentuje celkovou velikost v bytech. Definujeme $output(m_i) \subseteq I_\alpha$ jako množinu intermediate párů produkovaných mapovací úlohou m_i, tedy $output(m_i) = \bigcup_{d_j \in c_i} map(d_j)$. Dále definujeme $input(r_i) \subseteq I_\alpha$ jako množinu intermediate párů přiřazených k redukční úloze r_i. Funkce $part : k(I_\alpha) \to R$ přiřazuje intermediate klíč k redukční úloze.

Nechť i_j je intermediate klíč-hodnota pár, pak $i_j \in output(m)$ a $i_j \in input(r)$. Nechť $P_{i_j} \in 0, 1$ je proměnná, která se rovná 0 pokud intermediate pár i_j je vyprodukován na stejném výpočetním uzlu jako je následně zpracováván v redukční části a 1 v opačném případě.

Nechť $W = job_1, ..., job_w$ je množina všech úloh. Cílem je pak najít optimální loc a $part$ funkce tak aby $\sum_{job_\alpha \in W} \sum_{i_j \in I_\alpha} size(i_j)P(i_j)$ bylo minimální.

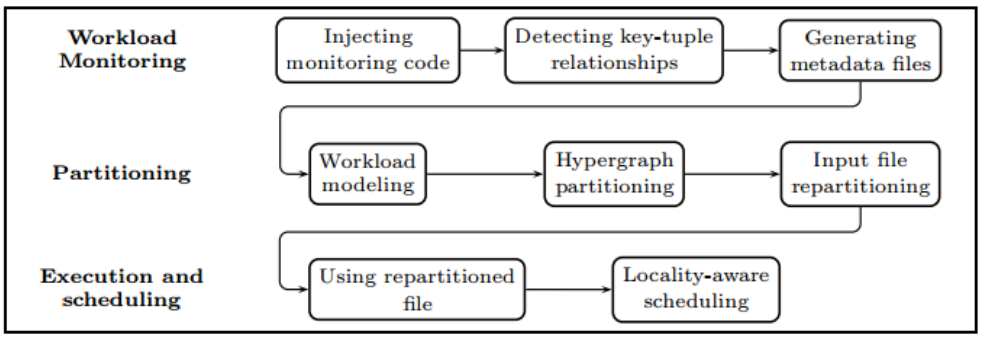
2.2 MR-part

Pro vyřešení zadaného problému byla navržena technika pojmenovaná MR-Part. Tato technika, za pomocí automatického dělení vstupních souborů, dovoluje využití maximální výhody data-locality při plánování redukčních úloh a výrazně snižuje množství dat, které je potřeba přesunout v shuffle fázi. MR-part se skládá ze tří hlavních fází. Jedná se o fáze Workload Monitoring, Partitioning a Execution and scheduling, tak jak je vidět z obrázku 2.1. V první fázi se shromáždují informace o vykonávání jednotlivých MapReduce úloh, které jsou následně zkombinovány. Z těchto informací je vytvořen model zatížení pomocí hypergrafu. V druhé fázi se na vytvořený hypergraf aplikuje dělící algoritmus, který rozdělí data na požadovaný počet bloků a následně jsou vstupní soubory upraveny na základě tohoto rozdělení. V poslední fázi se využije upravených vstupních souborů a za pomoci optimalizace přiřazování redukčních úloh se dosáhne minimalizace přenosu dat v shuffle fázi.

2.2.1 První fáze - Workload Characterization

Pro zajištění minimalizace přenosů mezi výpočetními uzly při přechodu z mapovací do redukční fáze je nejdříve zapotřebí zjistit, jaké páry hodnot se generují pro vstupní data a následně je vhodně seskupit. K tomu dochází v monitorovací a kombinační části první fáze.

Monitoring Nejprve je zapotřebí získat potřebná data z typických MapReduce úloh, u kterých se očekává jejich častější vykonávání. K zachycení těchto



Obrázek 2.1: MR-part schéma

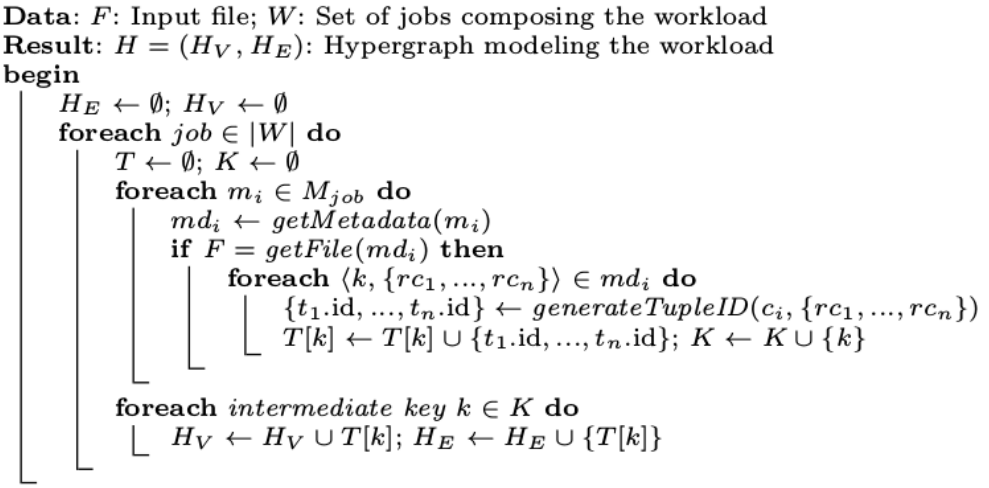
dat se využívá třída `RecordReader` [1] , která je rozšířená o monitorovací funkci. Monitorovací funkce unikátně identifikuje vstupní páry klíč-hodnota a jejich pozici ve vstupních datech. Pro každou mapovací úlohu se tek vytvoří soubor s metadaty. Vždy, když je načten nový blok s daty, je zároveň vytvořen i nový soubor, obsahující informace o bloku. Následně je iniciován record counter(rc). Pokaždé kdy je načten vstupní pár, inkrementuje se counter o 1. Poté pokud dojde k vytvoření vstupního páru, je vygenerován pár (k, rc). Po dokončení zpracování bloku dat jsou takto vygenerované páry uloženy do již vytvořeného souboru ve formátu $\langle k, \{rc_1, ..., rc_n\}\rangle$.

Combination Následující fáze již neběží zároveň s jinými úlohami, ale pustí je uživatel ideálně v čase, kdy systém není vytížen jinými výpočty. V kombinační fázi se shromáždí a zkombinují metadata z monitorování a na jejich základě se vygeneruje pro každý vstupní soubor hypergraf. Hypergraf $H = (H_V, H_E)$ je graf, kde každá hyperhrana $e \subseteq H_E$ může propojovat více jak dva vrcholy $v \subseteq H_V$. Po zpracování metadat se pak do tohoto hypergrafu uloží každý zpracovávaný prvek (vygenerovaný unikátní identifikátor reprezentující typicky řádek ve vstupním souboru). Poté se přidá hyperhrana, reprezentující klíč a propojí vrcholy, které tento klíč vygenerovaly. Detailní popis algoritmu v pseudokódu je zobrazen na obrázku. 2.2

2.2.2 Druhá fáze - Repartitioning

Nyní, když je vygenerován hypergraf modelující rozložení dat v jednotlivých souborech, je na každý hypergraf aplikován min-cut k-way dělící algoritmus. Tento algoritmus má jako vstup hodnotu k a hypergraf, ze kterého následně vygeneruje k disjunktních podmožin vrcholů tak, aby byla minimalizována suma hran mezi uzly rozdílných podmnožin. Parametr k je nastaven podle

[1] `RecordReader` je třída, která parsuje vstupní soubor a generuje vstupní páry. Každý datový formát má jiný `RecordReader`. Soubory tedy obvykle používají stále stejný.



Obrázek 2.2: Pseudokód algoritmu pro Metadata Combination

počtu bloků ve vstupním souboru. Po provedení tohoto algoritmu by měli být v jednotlivých vygenerovaných podmnožinách seskupeny uzly generující stejný klíč. Následně se použijí tyto podmnožiny k vygenerování nových vstupních souborů, kde už jsou data seřazena tak, aby řádky generující stejný klíč byly maximálně seskupeny. Tímto nově vzniklým souborem je následně nahrazen starý vstupní soubor, který je smazán. Pseudokód algoritmu je uveden na obrázku. 2.3 V algoritmu je uvedená funkce *RR*, která reprezentuje funkci třídy *RecordReader* použitou pro parsování vstupních souborů. Dále se v kódu oběvuje funkce *RW* znamenající *RecordWriter*. Její funkce je inverzní k funkci *RecordReader*. V této části je výpočetně nejsložitější vykonání min-cut algoritmu. Min-cut algoritmus spadá do skupiny NP-Complete problémů. Existuje však několik aproximačních algoritmů, které byly navrženy k řešení tohoto problému. V tomto případě byl použit algoritmus `PATOH` [2]

2.2.3 Třetí fáze - Execution and scheduling

K tomu, aby bylo možné plně využít výhody získané přeskupením záznamů v předchozích fázích, je zapotřebí maximalizovat data locality při plánování redukčních úloh. K tomuto účelu byl upraven algoritmus fairness-locality poskytnutý v [10], který pro každý pár key-value vypočítá skóre reprezentující poměr mezi vyvážeností vstupů do redukční fáze a lokací dat. Každý klíč je zpracován nezávisle pomocí greedy algoritmu. Pro každý klíč jsou pak možné uzly seřazeny podle jejich frekvence výskytu v sestupném pořadí (uzly s vyššími frekvencemi mají lepší data locality). Avšak namísto vybrání uzlu s ma-

[2] `http://bmi.osu.edu/~:umit/software.html`

```
Data: F: Input file; H = (H_V, H_E): Hypergraph modeling the workload; k: Number of
      partitions
Result: F': The repartitioned file
begin
    H_V ← H_V ∪ t_virtual
    {P_1, ..., P_k} ← mincut(H, k)
    for i ∈ (1, ..., k) do
        create tempf_i
    foreach c_i ∈ F do
        initialize(RR, c_i); rc ← 0
        while t.data ← RR.next() do
            t.id ← generateTupleID(c_i, rc)
            p ← getPartition(t.id, {P_1, ..., P_k})
            RW.write(tempf_p, t.data)
            rc ← rc + 1
    (j_1, ..., j_k) ← reorder(tempf_1, ..., tempf_k)
    for j ∈ (j_1, ..., j_k) do
        write tempf_i in F'
```

Obrázek 2.3: Pseudokód algoritmu pro Repartitioning

ximální frekvencí se upřednostní uzel s lepším fairness-locality skore. To má za následek maximální vyvážení vstupů do redukčních fází.

V MapReduce frameworku je tedy nutné provést následující modifikace:

- Upravení partitioning funkce tak, aby každému intermediate klíči přiřadila unikátní partition.
- Po dokončení mapovací části odeslat do master uzlu seznam s vygenerovanými intermediate klíči a jejich frekvencemi. Tato informace je přiložena k Heartbeat zpávě, která je odesílána po dokončení mapovací úlohy.
- Master přiřadí intermediate klíč k redukční úloze na základě dodaných informací a zajístí tak maximální vyvážení a data-locality v redukční části výpočtu.

2.3 Testování a výsledky

MR-Part byl implementován v Hodoop-1.0.4 a otestován na `Grid5000` [3]. Jedná se o rozsáhlou infrastrukturu složenou s rozdílných sítí s několika clustery výpočetních uzlů. Nutno podotknout, že tato infrastruktura není typická pro nasazení Hadoopu a vykonávání MapReduce úloh, především právě kvůli nesourodosti clusterů v síti, jejich fyzické vzdálenosti a tím i kapacitně omezeném

[3] https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

propojení. Částečně i proto bylo zrychlení zprácování tak znatelné. K testování byly využity datasety z `TPC-H` [4].

2.3.1 Výsledky

Množství přenesených dat pro různé typy dotazů
Při spustění několika různých úloh `TPC-H` [5]. na původních datasetech bylo v shuffle vázi přeneseno cca. 80% dat. Po provedení dotazů na již upravených datech pak docházelo k přenosu dat nižším než 10% pro všechny provedené dotazy.

Množství přenesených dat v závislosti na velikosti clusteru
Při provedení vybraného dotazu nad konfiguracemi systému s různým počtem clusterů nedocházelo k žádné výrazné změně v podílu přenesených dat. Jednalo se o konfigurace o 5 až 25 uzly v clusteru.

Časová náročnost dotazů
Jak již bylo řečeno, časová náročnost je velmi vázaná na propustnost sítě. Bylo provedeno vykonání dotazů na různých šířkách pásma a výsledek byl očekávaný. Tedy čím pomalejší síť mezi výpočetními uzly, tím výraznější zrychlení při vykonávání dotazů.

[4] http://www.tpc.org/tpch/www.grid5000.fr/mediawiki/index.php/Grid5000:Home
[5] Implementace použitých dotazů: http://www.cs.duke.edu/starfish/mr-apps.html

Kapitola 3

Návrh řešení pro HBase

Při návrhu řešení pro optimalizaci umístění dat na výpočetní uzly minimalizující datové přenosy V databázi HBase jsem vycházel z řešení pro souborový systém HDFS, které jsem popsal v předchozí kapitole. Nebudu proto uvádět znovu celý návrh řešení, ale pojmu tuto kapitolu jako výčet změn, oproti návrhu pro implementaci v HDFS.

3.1 Klíčová specifika HBase pro návrh řešení

3.1.1 Řazení záznamů v databázi

Jeden z největších změn při návrhu optimalizace pro databázi HBase bylo fyzické uložení jednotlivých řádků databáze. Nativní funkcí HBase je ukládání záznamů v lexikografickém pořadí. Tato funkce je pro HBase klíčová a je charakteristickým prvkem, kterým se odlišuje od jiných databází. Právě tato vlastnost dává HBase velkou sílu při sekvenčním prohledávání ať už celého datasetu nebo určité "výseče"z dat. Tento fakt se ale přímo rozchází se základní myšlenkou optimalizace pro HDFS, kdy je nezbytné změnit fyzické pořadí a tím i uložení záznamů. Bylo tedy nutné navrhnout možná řešení, jak docílit toho, aby bylo možné změnit pořadí záznamů v závislosti na požadavcích optimalizačního algoritmu. V úvahu přicházely dva možné varianty jak se s tímto omezením vyrovnat. Návrhy byly následující:

Změna klíče řádku podle potřeb optimalizačního algoritmu
: Toto řešení se nabízí jako nejjednoduší. Naráží však na výrazný problém. Při návrhu logického modelu pro databáze v HBase se typicky jako klíč pro jednotlivé záznamy používá klíč nesoucí určitou informační hodnotu (například část klíče odpovídá času pořízení daného záznamu). Změnu klíčů by jsme si tedy v tomto případě nemohli dovolit, pro aplikování tohoto přístupu by se nabízely tabulky, kde je klíč automaticky generován a nemá žadnou spojitost s daty které se nachází v řádku, který reprezentuje.

Přidání prefixu k již existujícímu klíči
Druhým možným řešením je přidání prefixu k již existujícímu

3.1.2 Automatický split a merge Hregionů

3.1.3 Fyzické uložení family column

3.2 Proces optimalizase

3.2.1 Monitoring

3.2.1.1 RecordReader Class

3.2.1.2 TableImputClass Class

3.2.1.3 Metadata file

3.2.2 Repartitioning

3.2.2.1 HyperGraph Class

3.2.2.2 PATOH Algoritmus

3.2.2.3 Repartioning Class

Kapitola 4

Implementace řešení

Kapitola 5

Testování a vyhodnocení měření

Závěr

Literatura

[1] deRoos, D.: *Hadoop For Dummies*. O'Reilly Media, Inc., 2014, ISBN 978-1-118-60755-8.

[2] White, T.: *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., třetí vydání, 2012, ISBN 0596521979, 9780596521974.

[3] Cloudera trainings. 2015. Dostupné z: `http://cloudera.com/content/cloudera/en/training/library.html`

[4] HDFS User Guide. 2014. Dostupné z: `http://hadoop.apache.org/docs/r2.3.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html`

[5] George, L.: *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., první vydání, 2011, ISBN 978-1-4493-9610-7.

[6] Team, A. H.: *Apache HBase Reference Guide*. 2015. Dostupné z: `http://hbase.apache.org/book.html`

[7] George, L.: *HBase: The Definitive Guide*. O'Reilly Media, první vydání, 2011, ISBN 1449396100.

[8] HBaseCon. 2015. Dostupné z: `http://hbasecon.com/archive.html`

[9] et al, M. L.-G.: *Data Partitioning for Minimizing Transferred Data in MapReduce*. INRIA & LIRMM, Montpellier, France.

[10] Ibrahim, S.; Hai, J.; Lu, L.; aj.: Locality/fairness-aware key partitioning for mapreduce in the cloud. *CloudCom 2010*, 2010: s. 17–24.

Příloha A

Seznam použitých zkratek

HDFS Hadoop Distributed File System

REST Representational State Transfer

CRUD Create Read Update Delete

API Application Programming Interface

SQL Structured Query Language

NoSQL Not only SQL

RDBMS Relational DataBase Management System

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS