

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

18. března 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 18. března 2015

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2015 Miroslav Hrstka. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.*

### 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** Nahraděte seznamem klíčových slov v češtině oddělených čárkou.

---

# Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

**Keywords** Nahraděte seznamem klíčových slov v angličtině oddělených čárkou.



---

# **Obsah**

<b>Úvod</b>	<b>1</b>
<b>1 Současný stav a použité technologie</b>	<b>3</b>
1.1 Hadoop . . . . .	3
<b>2 Existující řešení optimalizace distribuce dat pro MapReduce</b>	<b>13</b>
2.1 Definice problému . . . . .	13
2.2 MR-part . . . . .	14
<b>3 Návrh řešení pro HBase</b>	<b>19</b>
<b>4 Implementace řešení</b>	<b>21</b>
<b>5 Testování a vyhodnocení měření</b>	<b>23</b>
<b>Závěr</b>	<b>25</b>
<b>Literatura</b>	<b>27</b>
<b>A Seznam použitých zkratek</b>	<b>29</b>
<b>B Obsah přiloženého CD</b>	<b>31</b>



---

# **Seznam obrázků**

1.1	Diagram MapReduce procesu . . . . .	5
1.2	Hadoop Ekosystém . . . . .	6
1.3	Diagram uložení souboru v systému HDFS . . . . .	9
1.4	Datový model HBase . . . . .	10
1.5	Architektura databáze HBase . . . . .	11
2.1	MR-part schéma . . . . .	14
2.2	Pseudokód algoritmu pro Metadata Combination . . . . .	16
2.3	Pseudokód algoritmu pro Repartitioning . . . . .	17



---

# **Úvod**

uvod moji prace :P



# **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 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**

Apache Hadoop je framework který sdružuje projekty vyvíjející software po 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 Jprogramu Java.

### **1.1.1 Základní principy**

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

## **1. SOUČASNÝ STAV A POUŽITÉ TECHNOLOGIE**

---

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 uzelům. Proto Hadoop využívá horizontálního škálování. Díky horizontálnímu škálování se nabízí využití komoditního hardwaru namísto specializovaných výpočetních uzelů. Systém tedy běží na velkém množství samostatných počítačů spojených do clusteru.

### **Řešení selhání hardwaru**

S předchozím bodem úzce souvisí řešení případných výpadků jednotlivých uzelů. Kvůli velkému počtu výpočetních uzelů 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 kopii na různých výpočetních uzelích.

### **Přenášení kodů k datům**

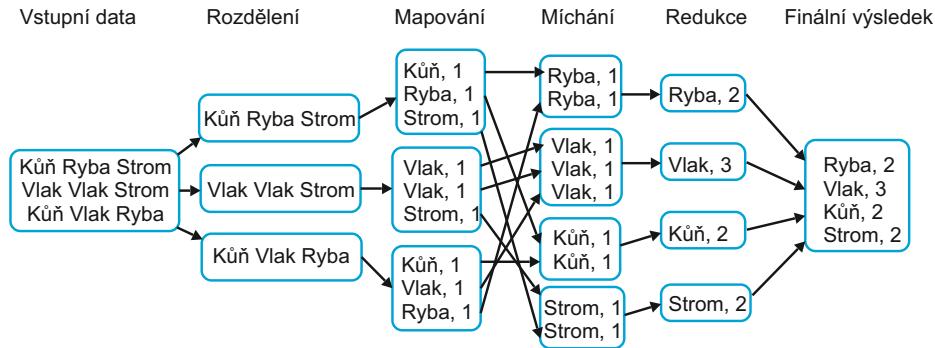
Kvůli velkým objemům dat a poměrně nízké propustnosti sítě spojující jednotlivé výpočetní uzly v clusteru je velmi výhodné namísto rozesílání dat na jednotlivé výpočetní uzly rozesílat na ně pouze kód výpočtu a minimalizovat tak přesuny dat mezi uzly. Na každém uzlu je tak vykonán výpočet pokud možno 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 zpracování. 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 distribuovat 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 páru klíč/hodnota. Tento formát je velmi jednoduchý, přesto se dají téměř všechny údaje takto prezentovat. Tato jednoduchá datová struktura tak umožnuje snadné zpracování dat efektivním způsobem. Klíč a hodnota může být cokoliv. Může se jednat o řetězce, čísla nebo komplexní struktury.

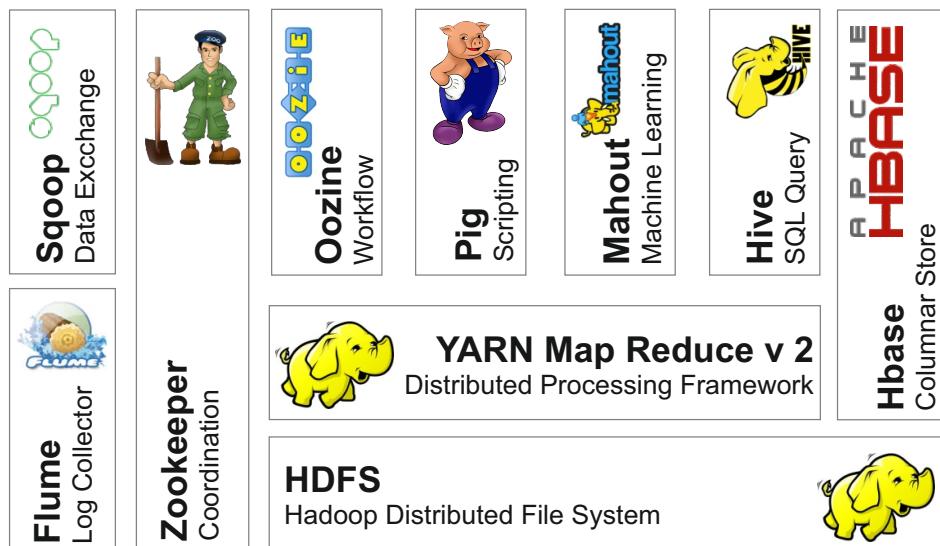
Mapreduce se skládá ze dvou hlavních fází, mapovací fáze a redukční fáze. Nejdříve je spuštěna mapovací funkce, která je dále spočtena nad jednotlivými prvky z množiny uspořádaných dvojic (klíč, hodnota), jež produkuje přechodné klíče a hodnoty.<sup>[2]</sup> Poté se všechny tyto přechodné hodnoty asociovány se stejným klíčem seskupí a pošlou 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 přesuvování(shuffling) dochází k přenosu průběžných výsledků mezi jednotlivými výpočetními uzly. Právě optimalizaci přesunů v těchto místech se bude zabývat hlavní část této práce. MapReduce je také často popisována funkcí:

$$\begin{aligned} \text{map} : (k_1, v_1) &\rightarrow \text{list}(k_2, v_2) \\ \text{reduce} : (k_2, \text{list}(v_2)) &\rightarrow \text{list}(k_3, v_3) \end{aligned}$$

## 1. SOUČASNÝ STAV A POUŽITÉ TECHNOLOGIE

---



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 závislosti 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.

#### 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 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 kolaborativní 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 zložit 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.

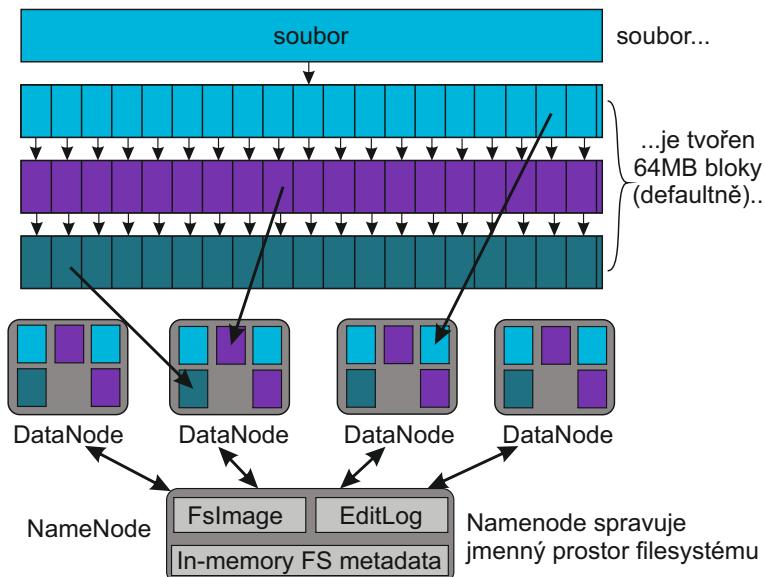
### 1.1.4 HDFS

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 databloku s požadovanými daty. NameNode tedy přípo 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

### 1.1.5 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 řádků 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 řídkých 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]

#### 1.1.5.1 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-

## 1. SOUČASNÝ STAV A POUŽITÉ TECHNOLOGIE

---

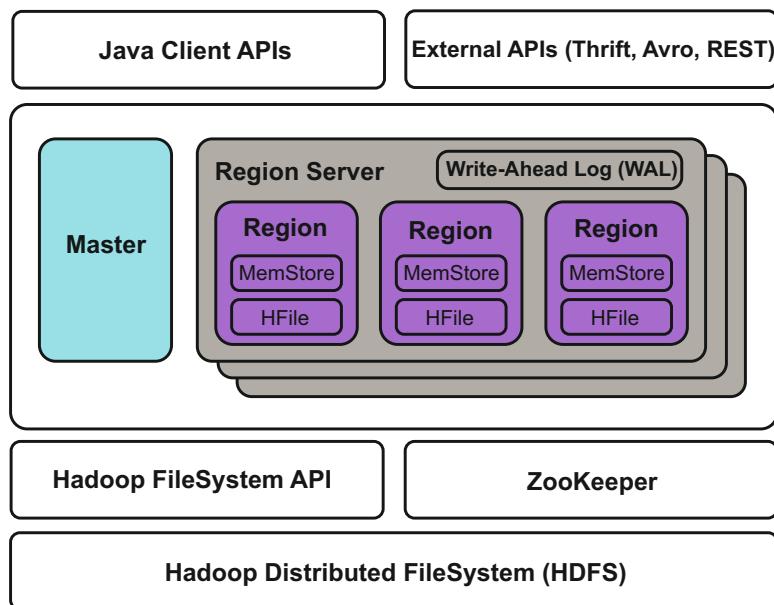
Row Key	Timestamp	Name Family		Address Family	
		First	Last	Apt#	Street
ROW1	T1	Mukund	Patel		
	T5			20	Park Street
	T6			10	Park Street
ROW2	T9	Ranjit	Das		
	T12			44	MG Road
	T13		Dash		

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 narození 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.

**HBase Architektura** HBase je distribuovaná databáze. Proto je i architektura složitější než u databázích 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 pořádku, 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 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 činnosti včetně rozdělování velkých regionů jsou prováděny automaticky bez zásahu uživatele.



Obrázek 1.5: Architektura databáze HBase

#### 1.1.5.2 Fyzické uložení dat

K uložení dat se typicky využívá souborový systém HDFS. Data jsou pak v souboru uložena v souborech nazývaných HFile. HFile má strukturu key-value mapy, kdy klíče jsou uloženy lexikograficky.



---

# 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 Transferred Data in MapReduce [7]. V tomto článku se zaměřují na redukování datových přenosů mezi mapovací a redukční fází ve fázi 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.

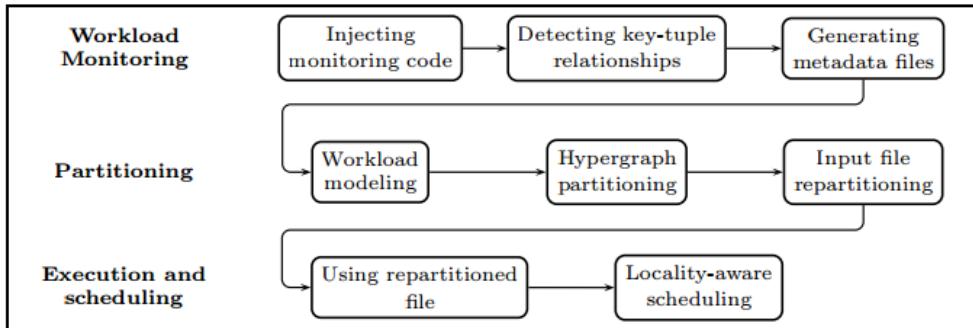
## 2.1 Definice problému

Mějme 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 mezivýsledky (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 ve fázi míchání. Při tomto rozdělování se nebere v úvahu plánování mapovacích a redukčních vází, ale pouze inteligentní rozdělení intermediate klíčů mezi jednotlivé redukční uzly.

Definujme daný problém formálně. Mějme vstupní data pro MapReduce úlohu  $job_\alpha$  složená z jednotlivých souborů  $D = \{d_1, \dots, d_n\}$ , které jsou rozděleny do množiny bloků (chunks)  $C = \{c_1, \dots, c_p\}$ . Funkce  $loc : D \rightarrow C$  přiřazuje data do bloků. Nechť  $job_\alpha$  je složen z  $M_\alpha = \{m_1, \dots, m_p\}$  mapovacích úloh a  $R_\alpha = \{r_1, \dots, r_q\}$  jsou redukční úlohy. Předpokládejme, že každá mapovací úloha  $m_i$  zpracuje blok  $c_i$ . Nechť  $N_\alpha = \{n_1, \dots, n_s\}$  je množina výpočetních uzlů použitých pro provedení úlohy.  $node(t)$  reprezentuje výpočetní uzel, kde

## 2. EXISTUJÍCÍ ŘEŠENÍ OPTIMALIZACE DISTRIBUCE DAT PRO MAPREDUCE



Obrázek 2.1: MR-part schéma

se vykonává úloha  $t$ .

Nechť  $I_\alpha = \{i_1, \dots, i_m\}$  je množina intermediate páru klíč-hodnota produkováných mapovací fází jako je  $map(d_j) = \{i_{j_1}, \dots, i_{j_t}\}$ .  $k(i_j)$  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 páru 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 páru přiřazených k redukční úloze  $r_i$ . Funkce  $part : k(I_\alpha) \rightarrow R$  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  $P_{ij} \in \{0, 1\}$  je proměnná, která se rovná 0 pokud intermediate pár  $i_j$  je vyprodukovan na stejném výpočetním uzlu jako je následně zpracováván v redukční části a 1 v opačném případě.

Nechť  $W = job_1, \dots, job_w$  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 zadанého úkolu 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, a to z Workload Monitoring, Partitioning a Execution and scheduling, tak jak je vidět z obrázku 2.1. V první fázi se shromažďují informace o vykonávání MapReduce úloh, které jsou 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 vhodné 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 dat se využívá třída `RecordReader`<sup>1</sup>, která je rozšířena o monitorovací funkci, která unikátně identifikuje vstupní páry klíč-hodnota a jejich pozici ve vstupních datech. Pro každou mapovací úlohu se pak 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 když je načten vstupní páár, inkrementuje se counter o 1. Poté pokud dojde k vytvoření vstupního páru, je vygenerován páár ( $k, \{rc_1, \dots, rc_n\}$ ).

**Combination** Následující fáze již neběží zároveň s jinými úlohami, ale pustí je uživatel ideálně v čase, kdy systém není vytízen 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 podmožin. Parametr  $k$  je nastaven podle počtu bloků ve vstupním souboru. Po provedení tohoto algoritmu by měli být v jednotlivých vygenerovaných podmožinách seskupeny uzly generující stejný klíč. Následně se použijí tyto podmožiny 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

---

<sup>1</sup>RecordReader je třída, která parsuje vstupní soubor a generuje vstupní páry. Každý datový formát má jiný RecordReader. Soubory tedy obvykle používají stále stejný.

## 2. EXISTUJÍCÍ ŘEŠENÍ OPTIMALIZACE DISTRIBUCE DAT PRO MAPREDUCE

---

**Data:**  $F$ : Input file;  $W$ : Set of jobs composing the workload  
**Result:**  $H = (H_V, H_E)$ : Hypergraph modeling the workload

```

begin
     $H_E \leftarrow \emptyset; H_V \leftarrow \emptyset$ 
    foreach  $job \in |W|$  do
         $T \leftarrow \emptyset; K \leftarrow \emptyset$ 
        foreach  $m_i \in M_{job}$  do
             $md_i \leftarrow getMetadata(m_i)$ 
            if  $F = getFile(md_i)$  then
                foreach  $\langle k, \{rc_1, \dots, rc_n\} \rangle \in md_i$  do
                     $\{t_1.id, \dots, t_n.id\} \leftarrow generateTupleID(c_i, \{rc_1, \dots, rc_n\})$ 
                     $T[k] \leftarrow T[k] \cup \{t_1.id, \dots, t_n.id\}; K \leftarrow K \cup \{k\}$ 
            foreach intermediate key  $k \in K$  do
                 $H_V \leftarrow H_V \cup T[k]; H_E \leftarrow H_E \cup \{T[k]\}$ 

```

Obrázek 2.2: Pseudokód algoritmu pro Metadata Combination

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**<sup>2</sup>

### 2.2.3 Třetí fáze - Execution and scheduling

K tomu, abychom mohli plně využít výhody získané přeskupením záznamů v předchozích fázích, je zapotřebí maximalizovat data locality při plánování redukčních úloh. K tomuto účelu byl upraven algoritmus poskytnutý

We have adapted the algorithm proposed in [7], in which each (key,node) pair is given a fairness-locality score representing the ratio between the imbalance in reducers input and data locality when key is assigned to a reducer. Each key is processed independently in a greedy algorithm. For each key, candidate nodes are sorted by their key frequency in descending order (nodes with higher key frequencies have better data locality). But instead of selecting the node with the maximum frequency, further nodes are considered if they have a better fairness-locality score. The aim of this strategy is to balance reduce inputs as much as possible. On the whole, we made the following modifications in the MapReduce framework: – The partitioning function is changed to assign a unique partition for each intermediate key. – Map tasks, when finished, send

---

<sup>2</sup><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
     $H_V \leftarrow H_V \cup t_{virtual}$ 
     $\{P_1, \dots, P_k\} \leftarrow mincut(H, k)$ 
    for  $i \in (1, \dots, k)$  do
        [ create  $tempf_i$ 
        foreach  $c_i \in F$  do
            initialize( $RR, c_i$ );  $rc \leftarrow 0$ 
            while  $t.data \leftarrow RR.next()$  do
                [  $t.id \leftarrow generateTupleID(c_i, rc)$ 
                 $p \leftarrow getPartition(t.id, \{P_1, \dots, P_k\})$ 
                 $RW.write(tempf_p, t.data)$ 
                 $rc \leftarrow rc + 1$ 
            ]
        ]
         $(j_1, \dots, j_k) \leftarrow reorder(tempf_1, \dots, tempf_k)$ 
        for  $j \in (j_1, \dots, j_k)$  do
            [ write  $tempf_i$  in  $F'$ 
        ]
    ]

```

Obrázek 2.3: Pseudokód algoritmu pro Repartitioning

to the master a list with the generated intermediate keys and their frequencies. This information is included in the Heartbeat message that is sent at task completion. – The master assigns intermediate keys to the reduce tasks relying on this information in order to maximize data locality and to achieve load balancing.



# KAPITOLA 3

---

## Návrh řešení pro HBase

### 3.1 Klíčová specifika HBase pro návrh řešení

#### 3.1.1 Řazení záznamů v databázi

#### 3.1.2 Automatický split a merge Hregionů

#### 3.1.3 Fyzické uložení family column

### 3.2 Proces optimalizace

#### 3.2.1 Monitoring

##### 3.2.1.1 RecordReader Class

##### 3.2.1.2 TableInputClass 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 Repartitioning 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] et al, M. L.-G.: *Data Partitioning for Minimizing Transferred Data in MapReduce*. INRIA & LIRMM, Montpellier, France.



## **Seznam použitých zkrátek**

**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



## 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
```