Univerzita Hradec Králové
Fakulta informatiky a managementu
katedra informačních technologií

Analýza metod a matematických přístupů pro distribuované výpočty

DIPLOMOVÁ PRÁCE

Autor: Bc. Petr Volf

Vedoucí práce: Mgr. Josef Horálek, Ph.D.

Hradec Králové, 2016

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 9. srpna 2016 Bc. Petr Volf

Poděkování

Děkuji vedoucímu práce Mgr. Josefu Horálkovi, Ph.D. za odborné rady a konzultace. Dále děkuji své rodině a přítelkyni za morální podporu a pochopení při tvorbě této práce.

Anotace

Každou minutu jsou vytvářeny velké objemy dat lidmi, počítači nebo senzory. Všechna vytvořená data je potřeba zpracovat, ale s takto velkým objemem dat se dostáváme do situace, kdy nejsme schopni tato data zpracovávat na běžných strojí. Proto je tedy nutné začít se věnovat novým způsobům zpracování dat, která jsou vytvářena. Diplomová práce Analýza metod a matematických přístupů shrnuje základní informace o problému zpracování velkoobjemových dat (z angl. Big Data), zpracování dat pomocí Distribuovaných Systémů. V praktické části práce pak představujeme námi navržený přístup, jak tyto objemné informace zpracovávat a návrh naší aplikace pro zlepšení a zjednodušení práce uživatelů.

Annotation

Title: Analyses of distributed computing methods and mathematical principles

People, computers or sensors creates huge amount of data every minute. Every piece of data has to be processed but with this huge amount of data we are not able to process data on common hardware. So it is necessary to discover new ways and methods how big data process on common hardware. Analyses of distributed computing methods and mathematical principles summarizes basic information about big data's problem, processing data with distributed systems. At practical part of our work we bring to the attention our idea of big data processing and present blue-prints of our app which should help and make work easier for users.

Obsah

1 Úvod

Informační technologie pronikají do ostatních vědeckých a univerzitních oborů již desítky let, ale za posledních dvacet let můžeme pozorovat pravděpodobně největší rozmach a to i v soukromé podnikatelské sféře. Toto vede především k vytváření velkého objemu dat, které je nutno zpracovávat. Tento rozmach navíc umožňuje využití většího a rychlejšího výpočetního výkonu ve vědeckých a univerzitních oborech, kterými jsou například medicína, chemie, fyzika, matematika či kosmonautika. V těchto vědeckých oborech je využití velkého výpočetního výkonu ohromnou výhodou, jelikož je díky němu možné ověřit teoretické poznatky za mnohem kratší čas, než kdyby výpočet probíhal „ručně“ na papír. Dále můžeme vytvářet modely, které nám dokáží situace přesně nasimulovat. Poté jsme schopni tyto modely opakovat, upravovat, vylepšovat zpravidla v lineárním čase něž dosáhneme správného výsledku. Modelování situací nám umožní předcházet chybným úsudků v reálném světě, které by mohly vést k tragickým událostem.

V oboru biomedicíny či chemie existuje mnoho vzorců pro výpočty chemických sloučenin. K těmto vzorcům se lidé pracující v laboratořích dříve dopracovávali klidně i desítky let, jelikož bylo nutné propočítat stovky, tisíce či miliony kombinací pro různé typy sloučenin. S využitím dnešního výpočetního výkonu se tento proces zkrátil na dny, hodiny či dokonce i minuty, a to i s mnohem větším objemem vstupních dat než při „klasickém ručním“ zpracování. Z těchto důvodů začali vědci využívat nové informační technologie ve velkém počtu. Dnes proto vznikají mnohé kombinace různých skriptů či aplikací vytvořenými vědci na základě jejich potřeby bez ohledu na správné využití daného programovacího jazyku nebo samotného výpočetního výkonu či dalších klíčových prvků v provádění programu, které by mohly zkrátit výpočetní čas daného zkoumaného problému.

Diplomová práce Analýza metod a matematických přístupů pro distribuované výpočty je rozložena do dvou hlavních částí. V první části práce se zabýváme teoretickými základy a v druhé části představujeme námi navrženou

aplikaci a odlišný přístup pro zpracování velkoobjemových dat. Teoretická část práce shrnuje základní informace o zpracování velkoobjemových dat (kapitola 2) a ukazuje na reálných příkladech to, co se dá z těchto dat zjistit. Kapitola 3 shrnuje informace a algoritmy běžně využívané v oblasti distribuovaných výpočtů. V kapitole 4 čtenáře seznamujeme s projektem Apache Hadoop a nastiňujeme jeho architekturu. Poslední část z teoretické práce je věnována OpenSource technologii, jelikož naši aplikaci navrhujeme v rámci otevřeného softwaru.

V praktické části práce (kapitola 7) představujeme náš návrh přístupu a aplikace, které má za úkol odstínit běžné uživatele od nutnosti učit se programovat a psát skripty či aplikace, aby měli možnost využít výpočetního výkonu za pomoci distribuovaných výpočtů.

Cíle této práce jsou:

- seznámit čtenáře se zpracováním velkoobjemových dat, využívání distribuovaných výpočtů pro zpracování velkoobjemových dat
- seznámit čtenáře o našem přístupu pro zpracování velkoobjemových dat
- navrhnout OpenSource aplikaci pro komunikaci mezi uživatelem a výpočetními uzly
  - definovat komunikaci uživatele s aplikací
  - definovat komunikaci aplikace s výpočetními clustery
  - řešení rozložení výpočetního výkonu
  - zjednodušit práci uživatelům s využíváním Hadoop clusterů

2 Big Data a jejich využití

Kapitola Big data a jejich využití shrnuje základní informace o zpracování velkého objemu dat, osvětluje termínu "Big data"a dále obsahuje ukázky využívání zpracování velkého objemu dat v reálném životě.

2.1 Motivace

Podíváme-li se o pár let zpět a to do roku 2009, kdy byl vydán zajímavý článek od společnosti Google v časopise Nature [1], ve kterém je ukázáno, jak je možné pomocí analýzy velkého objemu dat predikovat výskyt chřipky a to v reálném čase. Aby byla společnost Google schopná predikovat výskyt chřipkové epidemie, musí analyzovat milióny dotazů vyhledávaných uživateli pomocí jejich webového vyhledávače. Pomocí správně zvoleného algoritmu a paradigma MapReduce (paradigma MapReduce vysvětlujeme v kapitole 4.3), byla schopna společnost Google predikovat výskyt chřipky téměř v reálném čase. Americká agentura CDC (Centres for Disease Control and Prevention), která je součástí amerického ministerstva zdravotnictví, na rozdíl od společnosti Google musí požádat doktory o zasílání informací a poté tyto informace analyzovat. Získání informací od lékařů byl několika denní až týdenní proces. Pokud se ale jedná o vypuknutí epidemie, tak k tomu může dojít v řádu hodin maximálně dnů a je tedy nutné na tento problém reagovat v co nejkratším čase.

Obdobně o analýze velkého objemu dat začal smýšlet i Oren Etzioni, profesor na Washingtonské univerzitě v Seatlu, který se v roce 2003 začal zabývat problémem koupě nejlevnější letenky u aerolinek v USA. Etzoni vytvořil predikativní model z 12 000 cenových údajů. Tento model ukazoval, jak se ceny mění, ale nevysvětloval proč. Model je založen na pravděpodobnostech, které vyplývají z informací o jiných letech. Model nezahrnuje žádné speciální víkendové nabídky či zbývající místa v letadle. Po založení firmy Farecast, Etzioni domluvil analýzu 200 miliard záznamů o cenách letenek, která měla napomoci zpřesnění

jeho modelu. Po implementaci jeho modelu do vyhledávače Bing od společnosti Microsoft v roce 2012 byl schopen model zákazníkům aerolinek ušetřit až 50 amerických dolarů na jedné letence a poskytoval v 75 procentech správné výsledky.

Společnost Google by však nemohla být schopna predikovat výskyt chřipky v USA a ani Etzioni se svým model pro ceny letenek, kdyby neměli k dispozici velký objem dat, které museli analyzovat. Jelikož se naše práce zabývá zpracováním velkoobjemových dat věnujeme tuto část práce seznámení se s problematikou, základními principy a terminologií.

2.2 Big Data

Každým dnem přibývá více uživatelů používajících Internet a vytvářejících data, které jsou ukládána pro další zpracování. Nárůst velkého objemu dat, který všichni uživatelé elektronických zařízení, a to nejen připojených do sítě Internetu, dennodenně vytváří, se dostává již do stavu, že nejsme schopni tyto data zpracovávat s pomocí běžného softwaru a hardwaru. Pro tyto různorodá a rozsáhlá data vznikl specifický termín „Big Data“.

Big Data neboli problematika velkoobjemových dat či veledat je v dnešní době velmi aktuální téma a to nejen u velkých IT společností, jako jsou Google, IBM, Microsoft, Facebook, Twitter, ale i u menších firem, a to především z důvodů zlepšení, zrychlení a zjednodušení běhu těchto společností. Velkoobjemové informace jsou také generovány dalšími odvětvími jako hydrometeorologie (NOAA, ČHMI), kosmonautika (NASA), státní bezpečnost, medicína, ekonomika, novinařina a podobně. Všechna tato převzatá či vytvořená data musejí být ukládána, zpracovávána a potřebné informace (výstupy) by měli mít uživatelé k dispozici v reálném čase. Než si trochu blíže představíme Big data, tak bychom rádi uvedli základní definici, která je dostupná na internetu i v literatuře. Bohužel tato definice se vyskytuje ve více podobách, ale za nejpřesnější definici považujeme tu od společnosti Gartner:
„Big data je termín aplikovaný na data, která nelze zpracovávat, zachycovat a spravovat běžnými softwarovými i hardwarovými prostředky v rozumném čase.“ [2]
Dále jsou s termínem Big Data spojovány 3 základní termíny takzvané 3V:

- **Objem (Volume)** – objem dat pro zpracování rychle narůstá, což není je-

nom problém pro ukládání dat, ale především pro jejich analýzu a zpracování.

- **Rychlost (Velocity)** – rychlost zde znamená nejen rychlost vytváření dat, ale i rychlost zpracování dat, tak aby byla data připravená dle požadavků uživatele. Pomocí původních metod zpracování dat s jejich nárůstem se úměrně snižuje rychlost jejich zpracování. Je tedy třeba tyto metody pozměnit.
- **Různorodost (Variety)** – při zpracovávání objemových dat nenarazíme pouze na strukturovaná data, která neobsahují chyby, ale většinou se při jejich zpracování budeme potýkat s nestrukturovanými texty obsahující nesrovnalosti.

2.2.1 Chyby v datech

Chyby v datech a jejich ukládání se vyskytují již od prvopočátku vzniku počítačů. Nicméně pro malou množinu dat jsme schopni tyto chyby odstranit. K chybám může docházet nejen při jejich fyzickém ukládání, ale i při transformaci či dokonce při jejich vytváření autorem. Dlouho dobu se vědci snažili zpřesnit jejich měření, aby neobsahovalo chyby a díky tomu se každým dnem posunujeme v chápání a dobývání nejen naší planety, ale dokonce i celého vesmíru. Nicméně při zpracování velkého objemu dat již takové možnosti zatím nemáme a tudíž se budeme muset smířit s tím, že v našich systémech mohou existovat chybná data. V tuto chvíli vyvstává otázka „Je lepší zpracovávat malá, ale bezchybná data a nebo se uchýlit ke zpracovávání velkého objemu dat, i když může obsahovat chybná data?“

Nyní se vrátíme do roku 2000, kdy se autoři kontroly gramatiky Michal Banko a Eric Brill začali zabývat problémem zlepšení tohoto programu. Banko a Brill diskutovali o navrhnutí nového lepšího algoritmu pro kontrolu gramatiky či úpravě již existujících algoritmů. Než se pustili do práce úprav algoritmů, tak se rozhodli, že vyzkoušejí původní metody „nakrmit“ více daty. Tyto metody byly založeny na textových základech, které zahrnovaly maximálně milion slov. Brill a Banko vzali 4 metody, kterým poskytli nejprve 10 milionů, 100 milionů a nakonec miliardu slov. Konečné výsledky byly velmi překvapující. Metody, které na milionu slov poskytovali přibližně 75% až 85% správnosti, se zlepšily po zpracování miliardy slov na přibližně 95% úspěšnosti.

Obdobně jako Brill a Banko uvažovali o pár let později i pracovníci společnosti Google, kteří se rozhodli zaměřit na jazykové překlady. Google se svými možnostmi přístupu a zpracování dat, nechal překladač, aby si prošel miliardy webových stránek s cizojazyčnými překlady a tím se učil nejen slovíčka, ale především souvislosti. Následně si překladač pomocí statistických pravděpodobností vytvořil vazby na daná slovní spojení. Tyto pravděpodobnosti zlepšují překlady, jelikož již nedochází k překladů slovo od slova, ale jsou překládány slovní spojení, celé věty či kratší odstavce textu. Navíc jakýkoli uživatel má možnost dané překlady upravit a tím se pozměňují pravděpodobnosti výskytu daných slovních spojení. I když překladač společnosti Google ze začátku neměl bezchybná vstupní data, tak s nárůstem vstupních dat (chybných i bezchybných) se překlady zlepšují. Samozřejmě je nutné, aby počet správných překladů převažoval počet nesprávných. Dnes Google Translator dokáže překládat přibližně 90 jazyků ať se jedná o přímé překlady či překlady za pomocí dalšího jazyka (zpravidla anglického).

V tuto chvíli bychom rádi odpověděli na otázku, kterou jsme položili na začátku této podkapitoly. S větším počtem dat, i když některá z nich obsahují chyby, jsme schopni lépe určit správný výsledek než s velmi malým bezchybným vzorkem dat, jelikož jsme touto malou množinou dat velmi omezeni.

2.2.2 Korelace Dat

Pro analýzu informací ve světě malých dat využíváme nejčastěji metody založené na hypotézách o tom, jak data ve zvolené množině spolu souvisí (tzv. kauzalita dat). Tyto hypotézy poté zpracováním dat potvrzujeme či vyvracíme a tvoříme další hypotézy, které následně opět dokazujeme. Ve světě malých dat jsme schopni, při dnešních výpočetních výkonech, kauzalitu dat řešit v reálném čase. Přesuneme-li se do světa velkoobjemových dat, tak zkoumání kauzality začíná být velmi náročné a je nutné využívat dalších metod. V článku „The Petabyte Age“ [3], který vyšel v roce 2008, jeho autor Chris Anderson prohlašuje, že metody založené na kauzalitě budou nahrazeny statistickými metodami založených na čistých korelacích. Právě statistické korelační metody se osvědčily, jako výborný nástroj pro získávání potřebných informací o souvislostech dat.

Jedním z příkladů využívání korelační analýzy při zpracování velkoobjemových dat uvedeme obchodní řetězce. Obchodní řetězce každý den sbírají velký objem dat, který vytvářejí zákazníci svými nákupy. Pokud zákazníci navíc po-

užívají platební a zákaznické karty jsou se svými nákupy a potřebami jednoznačně spojováni. Z těchto nasbíraných informací poté mohou společnosti pomocí analýzy dat zjistit informace o potřebách svých zákazníků a tím řídit objednávání, skladování zásob či pozměnit nabídku stávajících produktů.

Jak uvádí Charles Duhigg v [4], tak americká diskontní společnost Target zveřejnila informaci, že umí určit, které zákaznice jsou těhotné pouze podle nákupů aniž by to prozradily v jakémkoli dotazníku. Společnost Target shromažďovala a analyzovala informace o nákupech těhotných žen, které se přihlásily o balíčky pro novorozence. Pomocí analýzy zjistili, že zákaznice kolem 3. měsíce těhotenství začaly ve velké míře nakupovat neparfemované pleťová mléka a o několik týdnů později nakupovaly výživové doplňky s vápníkem, hořčíkem a zinkem. Po ukončení zpracování datového vzorku, analytický tým identifikoval přes dvacet produktů, které pomohly určit skóre předpovědi těhotenství, pro každou jejich zákaznici. Pomocí korelací pak může obchodník předpovědět termín porodu v rozmezí několika dní a díky tomu může zasílat dárkové poukazy nastávajícím maminkám o speciálních nabídkách či slevách na produkty, které jsou určené pro těhotné.

2.2.3 Shrnutí

Jak jsme v této kapitole nastínili, tak data jsou vytvářena po celém světě každou vteřinu a je potřeba je zpracovávat. Jelikož jsou tyto data vytvářena ve velkém měřítku, tak není možné je zpracovávat či zachytávat v reálném čase na jednom počítači. Je třeba tedy začít využívat pro jejich zpracování sofistikovanější systémy zpracování dat. Jedním z těchto systémů je projekt Apache Hadoop, který se zaměřuje na zpracování velkoobjemových dat. Dále je nutné změnit náš pohled na kauzální svět a více se zaměřit na statistické a pravděpodobnostní modely, pomocí kterých jsme schopni určovat vazby mezi daty.

3 Distribuované systémy

V této části práce seznámíme čtenáře se základními definicemi a pojmy v oblasti distribuovaných výpočtů. Dále představíme dnešní nejběžnější využití distribuovaných výpočtů.

S příchodem Internetu v sedmdesátých letech dvacátého století se začal zvětšovat počet aplikacích, které začaly vyžadovat distribuované zpracování a to především proto, že jejich výpočty by na jednom výpočetním stroji trvaly týdny, roky či staletí. Trend vzniku aplikací byl především způsoben novými pokroky v síťových a hardwarových technologií, klesajícími pořizovacími náklady a potřebnými prostředky na jejich provoz. Výše zmiňované body přispěly k tomu, aby se distribuované výpočty staly nákladově efektivní a prakticky využitelné. Příchodem nového tisíciletí, s velkou pomocí World Wide Webu, byl celý svět připojen do sítě Internet. Velkou roli na tom také má snížení cen na potřebné hardwarové prvky a poplatků za připojení do této sítě, které se platí poskytovateli.

3.1 Definice distribuovaného systému

Distribuované systémy lze definovat v informačních technologií více způsoby:

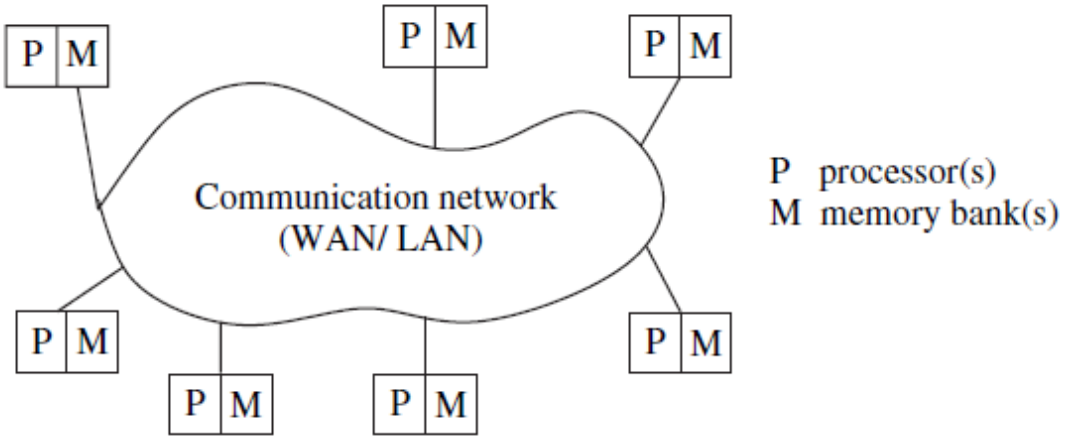
- Kolekce počítačů, které nesdílí paměť nebo společné fyzické hodiny, komunikují pomocí výměny zpráv skrze počítačovou síť a každý počítač má svoji vlastní paměť a svůj operační systém.
- Kolekce nezávislých počítačů, které se uživatelům jeví jako jeden koherentní počítač.
- Termín označuje širokou škálu počítačů - od slabě vázaných počítačových systémů, jako jsou WAN (Wide Area Network), přes pevně vázané počítačové systémy jako LAN (Local Area Network) až po velmi pevně svázané jako jsou multiprocesorové systémy.

Jak je uvedeno v [5], tak distribuovaný systém může být definován jako kolekce nezávislých entit (počítačů, procesorů, skupiny počítačů chovající se jako jeden, paralelních systémů), které spolu spolupracují za účelem vyřešit problém, který není možné vyřešit individuálně, komunikují skrze počítačovou síť a mají následující vlastnosti:

- **Nemají společné fyzické hodiny** - tento předpoklad je velmi důležitý, jelikož zavádí prvek distribuce do systému a vede k asynchronii mezi procesory.
- **Nemají sdílenou paměť** - tento prvek je základní vlastností systému, který vyžaduje komunikaci výměnou zpráv skrze počítačovou síť.
- **Geografická separace** - čím více jsou entity od sebe geograficky dál, tím reprezentativnější distribuovaný systém je. Každý distribuovaný systém nemusí ovšem obsahovat všechny procesory ve WAN a tak postačí, když jsou entity zapojené do systému pouze na LAN.
- **Nezávislost a různorodost** - entity jsou volně vázány a mohou běžet v různých rychlostech či s různými operačními systémy, obvykle nejsou součásti dedikovaných systémů a pouze nabízejí možný výkon jako službu nebo kooperují s ostatními entitami na řešení problému.

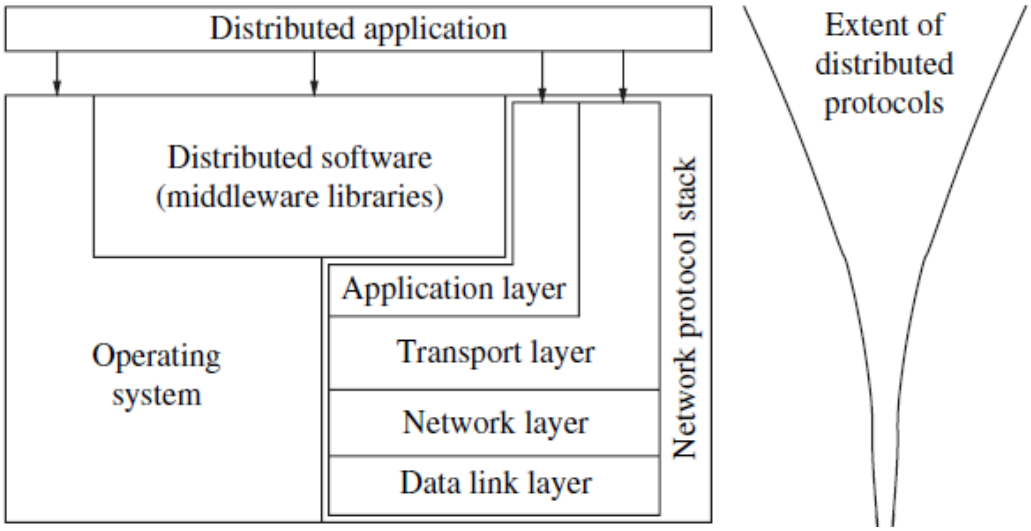
3.2 Vztahy mezi komponentami v systému

Obrázek 3.1 znázorňuje distribuovaný systém, kde každá entita v systému je připojena komunikační sítí, má svůj procesor a paměť. Jak je uvedeno v [5], tak distribuovaný software je také označován jako middleware. Realizace, běh nebo také výpočet v distribuovaném systému znamená provedení procesu, při kterém mají spolupracující entity za úkol dosáhnout společných cílů.



Obrázek 3.1: Klasický distribuovaný systém
Zdroj: [5]

Obrázek 3.2 znázorňuje vztahy mezi softwarovými komponentami běžících na každé entitě (počítači), který využívá vlastního operační systému a síťového protokolového zásobníku.

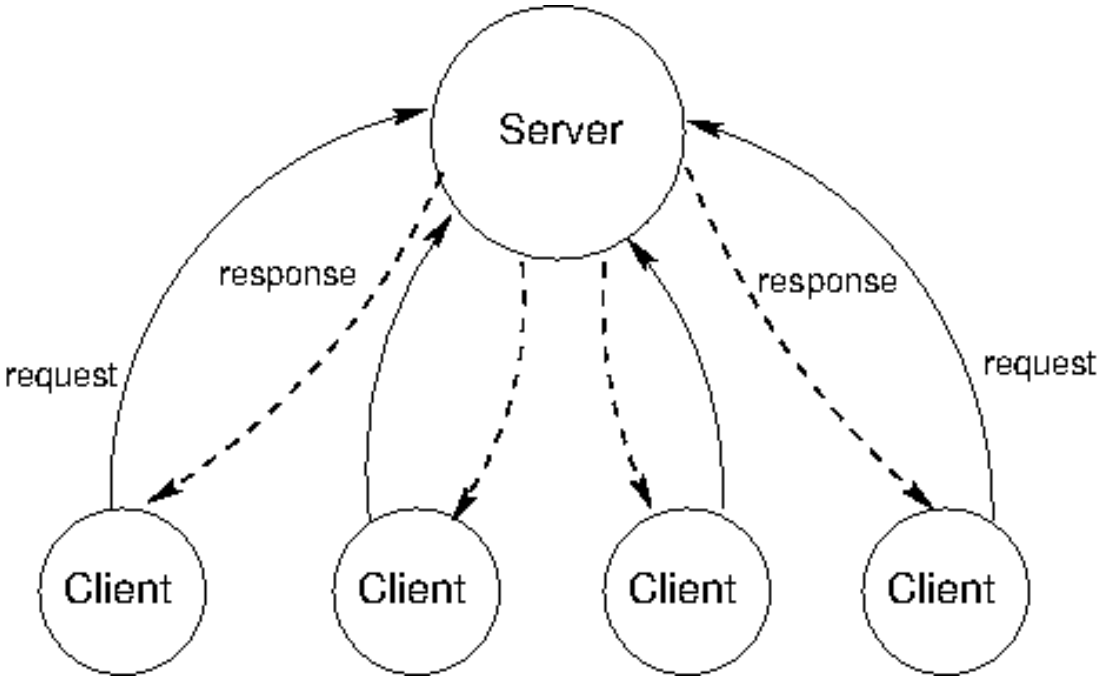


Obrázek 3.2: Interakce komponent ne každé entitě v DS
Zdroj: [5]

3.3 Typy distribuovaných systémů

3.3.1 Klient-server

Jak uvádí autoři [6], tak klient-server model je zvláštním typem distribuovaného systému, který jasně definuje vztah mezi dvěma entitami připojených do systému. Server poskytuje služby (např. zpracování databázových dotazů) a klient tyto služby využívá. U tohoto typu modelu velmi záleží na komunikaci mezi klientem a serverem. Tato komunikace musí být spolehlivá tj. nesmí být žádná data ztracena a na klientskou stanici musí dorazit ve správném pořadí. Pro zajištění komunikace se nejčastěji využívá protokol TCP/IP. Obrázek 3.3 nastiňuje základní komunikaci v systému klient server.



Obrázek 3.3: Návrh cloud computingového systému
Zdroj: [7]

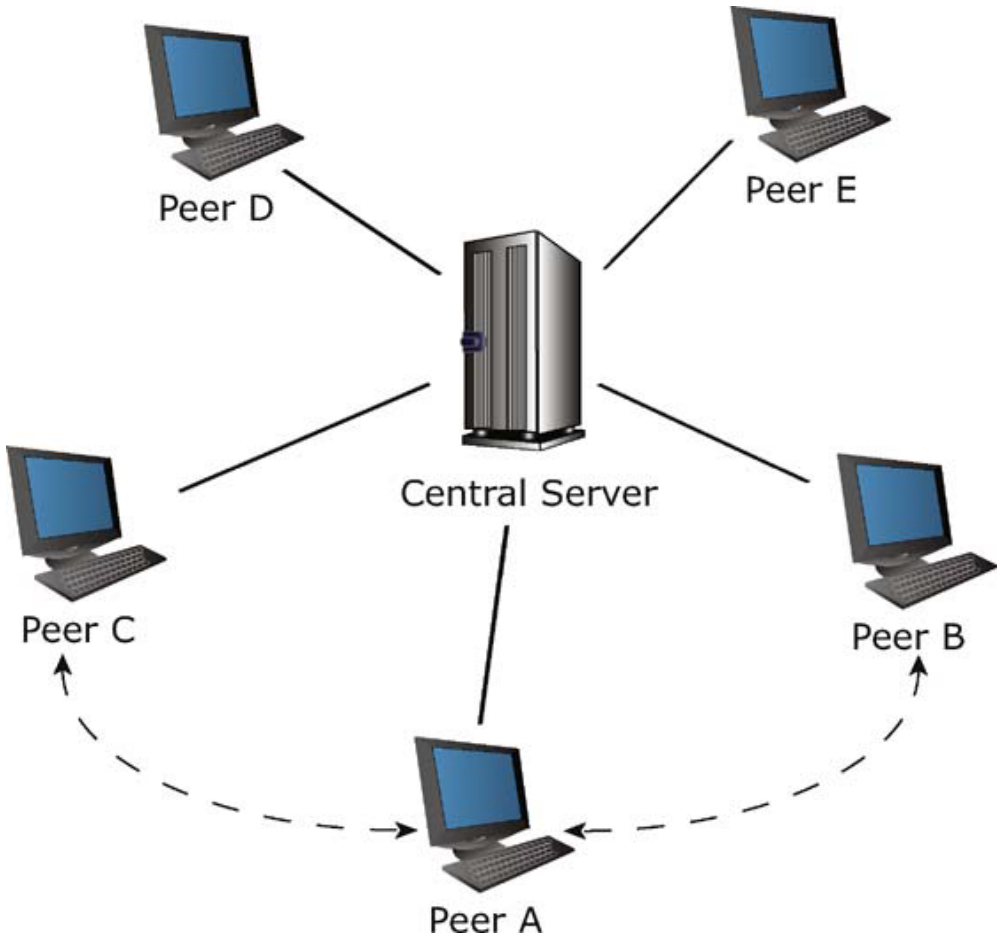
3.3.2 Peer-to-peer

Model peer-to-peer, také označován jako P2P, je navržen tak, že každá entita (tzv. uzel, node či peer) připojená do distribuovaného systému je klientem a zároveň serverem. O proti modelu klient-server P2P model spoléhá na hromadění

aplikací, aby měli přístup k distribuovaným zdrojům decentralizovaným způsobem. Tento model je daleko levnější než klient-server, jelikož obsah a aplikace jsou udržovány individuálně.

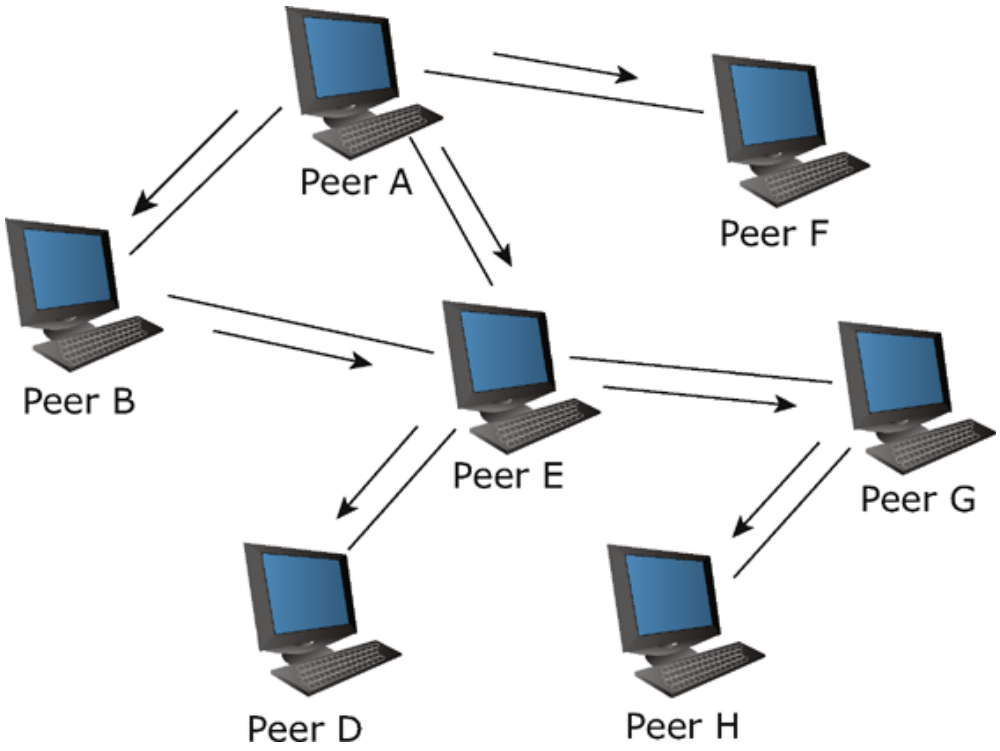
Peer-to-peer můžeme kategorizovat do následujících skupin, které byly definovány například v [6] nebo [8]:

- **Centralizované** - model obsahuje centrální server, který se stará o vykonávání jednoduchých úkonů, jako jsou plánování zátěže či validaci výsledků. Na obrázku 3.4 je vyobrazena P2P centralizovaná síť. Jak je z obrázku patrné, tak narozdíl od typy sítě klient server mohou koncové stanice (peer) komunikovat navzájem bez nutnosti zasílání daného požadavku na server.



Obrázek 3.4: Centralizovaná síť P2P,
Zdroj [8]

- **Decetralizované** - v tomto systému všechny funkce zastávají jednotlivé nody systému bez existence centrálního serveru. Obrázek 3.5 znázorňuje komunikaci v dané síti.

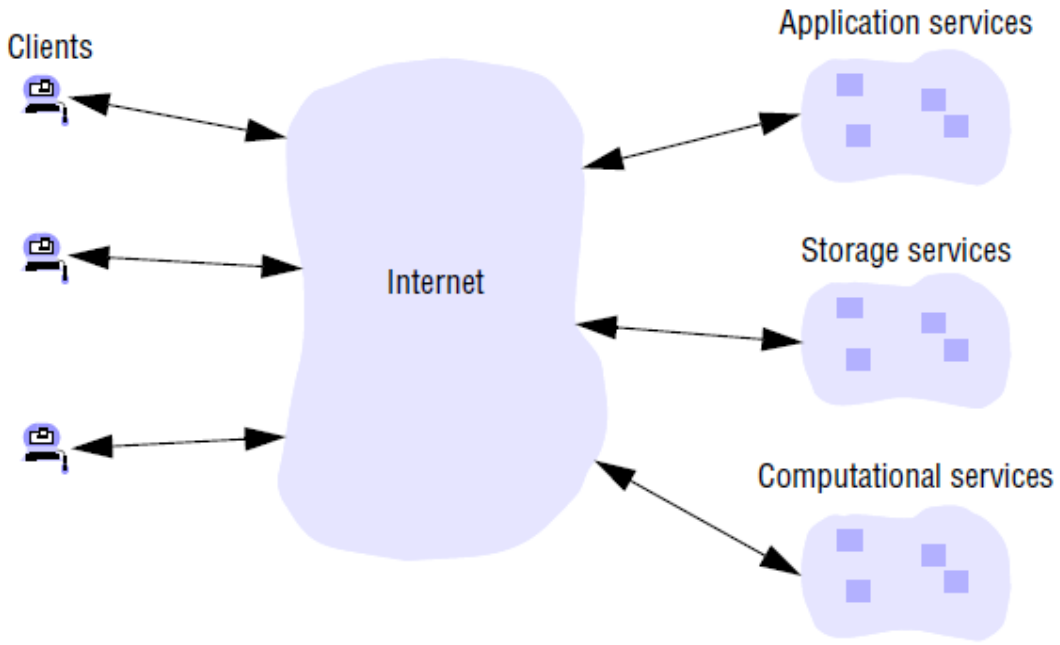


Obrázek 3.5: Centralizovaná síť P2P,
Zdroj [8]

- **Hybridní** - jsou kombinací centralizovaných a decentralizovaných distribuovaných systémů. V modelu existuje tzv. super uzel, který zastává centralizovaný server a je odpovědný za centralizovanou práci. Ostatní uzly pracují decentralizovaně.

3.3.3 Cloud computing

Cloud computing je autory v [9] definován jako soubor internetových aplikací, úložišť a výpočetních služeb, které uspokojují většinu uživatelských potřeb a tak jim umožňuje z větší části pracovat aniž by ukládaly data na jejich lokální zařízení či potřebovali mít instalovaný software potřebný pro jejich práci. Ukázka, jak může vypadat cloud computingový systém je na obrázku 3.6.



Obrázek 3.6: Návrh cloud computingového systému
Zdroj: [9]

Cloud computing vychází ze snižování potřeb uživatelů mít instalované aplikace na jejich strojích a to především v závislosti na navyšování rychlosti, dostupnosti a spolehlivosti počítačové sítě. Množství poskytovaných aplikací a služeb v cloudu se každým dnem navyšuje. Mnoho firem a veřejnosti začíná využívat cloud computing z mnoha důvodů jako jsou například obchodně orientované úkony, zobrazování statistik pro manažery, ukládání dat, jako jsou fotografie, hudba, videa nebo dokumenty apod. Další výhodou jsou nižší hardwarové nároky na klientské stanice, jelikož tyto stanice pouze zobrazují data a samotné aplikace běží na serveru. Na druhou stranu jsou zde i nevýhody, jako je obava o důvěrnost a ochranu osobních údajů není brána jako dostatečně věrohodná, jak uvádí [6].

3.3.4 Grid computing

Jak uvádí autoři v [6], tak pojem grid byl nejprve použit jako metafora k elektrické rozvodné síti. Prvotní myšlenka byla, že přístup k datům a využití výpočtů by mělo být tak jednoduché, jako zapojení jakéhokoli elektrického spotřebiče do sítě. Bohužel není jednoduché najít jednotnou definici pro grid compu-

ting, tak uvádíme ty, které jsou nejčastěji referovány:

- Výpočetní grid (computational grid) je hardwarová a softwarová infrastruktura, které poskytuje spolehlivý, konzistentní a levný přístup ke špičkovým výpočetním možnostem. [10]
- Výpočetní grid je technologie, která umožňuje virtualizaci zdrojů, jejich poskytnutí na požádání a sdílení služeb mezi organizacemi.[11]
- Grid computing má schopnost pomocí sady otevřených standardů a protokolů získat přístup k aplikacím, datům, výpočetnímu výkonu, úložištím a k dalším zdrojům skrze připojení do Internetové sítě. Grid je typ paralelních a distribuovaných systémů, které umožňují sdílení, výběr a agregaci distribuovaných zdrojů napříč různými doménami, které jsou založené na jejich dostupnosti, kapacitě, výkonnosti, ceně a kvalitě služeb požadované uživateli. [12]

3.3.5 Dobrovolnické počítání

V polovině dvacátého století vznikl veliký boj o dobývání vesmíru. Jedním z nově vzniklých projektů byl také SETI (Search for Extraterrestrial Intelligence), který se zaměřoval na hledání života ve vesmíru pomocí mikrovlnných radiových vln. Postupně se do tohoto projektu začala zapojovat i společnost NASA a koncem sedmdesátých let se podařilo projekt SETI rozběhnout. SETI využíval pro zpracování dat superpočítače umístěné přímo v teleskopu, které prováděli převážnou část analýzy zkoumaných dat. Postupem času přestal být tento projekt financován a v roce 1993 byl zrušen kongresem spojených států. Z důvodu nedostatku finančních prostředků se tým vědců z projektu SETI rozhodl začít využívat tehdy málo využívané distribuované výpočty. David Gedey s Craigem Kasnoffem v roce 1995 navrhli, že se radiový výzkum SETI bude provádět na virtuálních počítačích, který se bude skládat z individuálních počítačů připojených pomocí Internetu. Vznikl tedy nový projekt nazvaný SETI@Home. Myšlenka zpracování analýzy dat pro SETI@Home je taková, že namísto drahého superpočítače, který běží dlouhou dobu, je využití stovek, tisící či milionů méně výkonných individuálních počítačů, které obdrží instrukce pro zpracování malého balíku dat. Jakmile je výpočet zpracován data jsou nahrána na server, který s nimi dále pracuje a skládá výsledný výstup pro uživatele. V roce 2004 se projekt SETI@Home integroval do projektu BOINC (Barkeley Open Infrastructure

for Network Computing) a začal vznikat jeden z největších projektů do kterého se pomocí Internetu zapojují miliony dobrovolníků. Dnes existuje mnoho projektů v BOINC, do kterých se mohou dobrovolníci přihlásit a tak půjčit svůj výpočetní výkon pro pomoc vědeckým týmům, univerzitám či soukromým firmám.

Definice dobrovolnického počítání

Dobrovolnické počítání je velmi často řazeno do kategorie grid computing, a to především díky jeho rozsáhlé definice. Nicméně velmi záleží na zvolené definici, jak je uvedeno na oficiálních stránkách BOINC [13].

Podle [13] definujeme dobrovolnické počítání jako uspořádání, ve kterém uživatelé (dobrovolníci) poskytují výpočetní prostředky pro projekty, které si sami zvolili, a tyto poskytnuté výpočetní zdroje jsou využívané pro distribuované výpočty.

- Dobrovolníci jsou především členy široké veřejnosti, kteří vlastní počítač s připojením do sítě Internet. Školy či soukromé firmy samozřejmě mohou také dobrovolně poskytnout své výpočetní stroje pro dobrovolnické počítání.
- Dobrovolníci jsou anonymní.
- Dobrovolníci nejsou odpovědní k projektům díky svoji anonymitě.
- Dobrovolník musí projektům důvěřovat, že:
  - aplikace neškodí jeho počítači nebo neproniknou do jeho soukromí.
  - projekt je pravdivý o tom, co dělá a jak dochází ke zpracování výsledků a také, že splňuje všechny bezpečnostní pravidla a praktiky tak, že hackeři nejsou schopni projekt zneužít.
- Projekty, do kterých se mohou dobrovolníci přihlásit, jsou zpravidla výzkumného či akademického zaměření. Nicméně existují i výjimky jako je například projekt GIMPS, který hledá Mersennova prvočísla. GIMPS dne 7. ledna 2016 za pomoci dobrovolnického počítání nalezl zatím nejvyšší Mersennovo prvočíslo.

BOINC wiki [13] také uvádí několik důvodů, proč je dobrovolnické počítání velmi důležité:

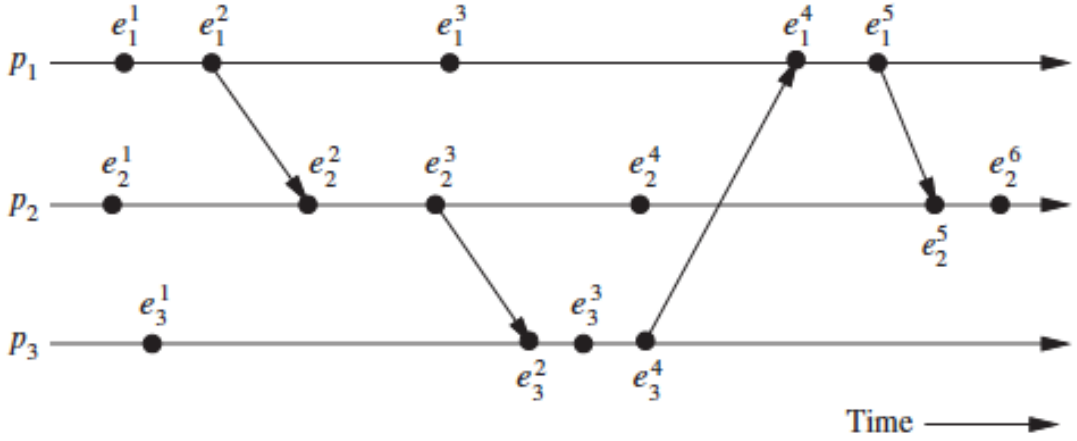
- Vzhledem k obrovskému počtu počítačů na světě, dobrovolnické počítání může poskytnout mnohem větší výpočetní sílu než jakýkoli jiný typ distribuovaných výpočtů. Díky dobrovolníkům se otevírají možnosti pro mnoho vědeckých projektů, které nemají dostatek finančních prostředků, aby mohly vzniknout. S vysokou pravděpodobností, ke stávajícímu ekonomickému růstu, bude přibývat počítačů a herních konzol na kterých lze provádět výpočty a tedy bude i přibývat dobrovolníků poskytující své výpočetní stroje.
- Výkon dobrovolnického počítání nelze koupit, a tedy i projekt s nízkým finančním rozpočtem může mít velký výpočetní výkon, pokud bude v povědomí dobrovolníků.
- Dobrovolnické počítání podporuje zájem veřejnosti o vědu a poskytuje možnost veřejnosti rozhodnout, kam se bude věda uchylovat.

3.4 Distribuovaný program

Program definujeme jako soubor přesně definovaného formátu obsahující posloupnost instrukcí, která mohou být případně sdílená více procesy a data potřebná k provádění daného úkolu. Proces označuje akt provádění programu či instance výpočtu definovanou v programu. Procesy mohou běžet současně na různých procesorech v systému. V tradičních operačních systémech, jako jsou například Unix, Windows, Linux atd., procesy komunikují za pomocí společné globální paměti, globálních hodin, globálního manažera, který se stará o správu procesů jako je například spuštění, odkládání, ukončení.

Autoři [5] uvádí, že distribuovaný program je množina n asynchronních procesů p_1, p_2, ...,p_i, ..., p_n, které spolu komunikují pomocí výměny zpráv skrze počítačovou síť. Uvažme, že procesy běží každý na svém procesoru, nemají žádnou globální paměť a komunikují pouze výměnou zpráv. Nechť C_{ij} značí komunikační kanál mezi p_i a p_j, m_{ij} značí zprávu zaslanou procesem p_i procesu p_j. Komunikační zpoždění je konečné a nepředvídatelné. Jelikož v systému neexistují globální hodiny, tak provedení programu a výměna zpráv musí být asynchronní - proces může být spuštěn kdykoli a proces, který zasílá zprávy, tak nečeká na odpověď o doručení zprávy.

Na obrázku 3.7 je vyobrazeno spouštění procesů a zasílání zpráv v čase. Vo-



Obrázek 3.7: Průběh distribuovaného programu
Zdroj: [5]

dorovné šipky označují čas, body e_i^x označují *x-té* provedení procesu p_i. Šipky spojující body e_i^x označují zaslání zpráv mezi procesy.

Z výše uvedených definicí vyplývá, že hlavními rozdíly mezi prováděním programu v klasických operačních systémech a v distribuovaných systémech je to, že distribuovaný systém neobsahuje žádné společné hodiny, kterými by se mohlo řídit provádění procesů a také neobsahují společnou paměť, která by zajišťovala odkladový prostor pro data, která jsou využívána napříč procesy při provádění daného programu. Při provádění procesu v klasických OS je spouštění procesů řízeno globálním manažerem implementovaným v OS, ale v distribuovaných systémech žádný takovýto manažer neexistuje a proto je využívána metoda zasílání zpráv, pomocí které je možno komunikovat mezi procesy a tím řídit provádění procesů v systému.

4 Apache Hadoop

Cílem kapitoly Apache Hadoop je seznámit čtenáře s historií, součástmi Hadoop a především se základními principy HDFS a MapReduce, které jsou využívány v praktické části této práce.

Než se pustíme do historie Hadoop, tak je nutné vysvětlit co je to Apache Hadoop. Apache Hadoop je framework poskytující podporu pro distribuované zpracování velkoobjemových dat (Big Data) napříč clusterovým řešením za využití jednoduchého programovacího modelu. Pod pojmem cluster rozumíme skupinu dvou a více (zpravidla tisíce) propojených počítačů pomocí počítačové sítě. Propojené počítače mají za úkol chovat se pro okolí jako jeden a poskytnout větší výpočetní výkon a úložný prostor. Hadoop je navržen tak, že umožňuje využívat pro potřeby výpočtů jeden nebo až tisíce počítačů, které poskytují hardwarové zdroje jako jsou výpočetní výkon či úložný prostor. Knihovna Hadoop se sama stará o řešení a ošetření chyb vzniklých na aplikační vrstvě. Jak bude více přiblíženo v kapitole 4.1, tak Hadoop vychází z projektu Lutcene napsaný v jazyce Java, který byl roku 1999 představen Dougem Cuttingem. Apache Hadoop je především určen pro běžně dostupný hardware a tedy zpravidla běží na OS GNU/Linux, ale samozřejmě je možné, aby běžel na jakémkoli stroji podporující jazyk Java, jak je uvedeno v [14].

Projekt Apache Hadoop dnes obsahuje následující základní součásti (můžeme nazývat také moduly či projekty):

- **Hadoop Common** – obecná služba, která je potřebná pro využívání další modulů
- **Hadoop Distributed File System (HDFS)** – Distribuovaný souborový systém, který poskytuje vysokou propustnost (high-throughput) pro přístup k datům. Základní princip HDFS je popsán v kapitole 4.2.
- **Hadoop YARN** – poskytuje správu a schedulování úkolů, která mají být zpracovány.

- **Hadoop MapReduce** – systém založený na YARN (viz. sekce 4.3.1) pro paralelní zpracovávání velkého objemu dat. Tomuto modulu se věnujeme v kapitole 4.3.

Hadoop obsahuje více modulů, kterými jsou např. Ambari, Cassandra, Chukwa, HBase, Hive, Pig či ZooKeeper. Informace o těchto modulech a dalších jsou k dispozici na [14].

4.1 Historie Hadoop

Počátky Hadoop se dají pozorovat už v Apache Nutch, což je webový vyhledávač, který je součástí projektu Apache Lucene. Tento projekt i z něho vycházející Hadoop vytvořili Doug Cutting a Mike Cafarella. Apache Nutch začali využívat již v roce 2002 pro prohledávání webových stránek. S tehdejším trendem rychlého vzniku webových stránek si autoři projektu uvědomovali, že Nutch se nebude moci škálovat na miliardy stránek a proto začali hledat řešení, které by tento problém vyřešilo. V roce 2003 byl publikován dokument [15] popisující architekturu distribuované datové struktury (GFS) společnosti Google. Díky tomuto článku začali autoři pracovat na vlastní open-source implementaci nazvané Nutch Distributed Filesystem (NDFS). V roce 2005 Cafarella a Cutting představili MapReduce implementovaný v Nutch. Původ MapReduce je opět od společnosti Google, která publikovala článek popisující MapReduce v roce v 2004. MapReduce společně s NDFS se staly základem pro projekt nazvaný Hadoop, který vznikl v roce 2006.

Doug Cutting se přibližně ve stejné době připojil k firmě Yahooo!, kde dále pracoval na zlepšování a rozšiřování projektu Hadoop se svým týmem. V lednu 2008 se Hadoop stal top projektem pro Apache.

Rok 2008 přinesl další úspěch pro Hadoop a to v prolomení světového rekordu o nejrychlejší sortovací systém. Hadoop setřídil 1 terabyte dat na 910 nodech clusteru za 209s (3 minuty a 29 sekund), čímž porazil původního vítěze, který setřídil data za 297s (4 minuty a 57 sekund). O rok později Hadoop zvládl setřídit 1TB za 62 sekund. Od této doby se stal Hadoop obecným nástrojem v komerčním sektoru využívaný pro ukládání a analýzu Big Data.

Dnes je Hadoop využívaný stovkami společností. Jako příklad můžeme uvést Yahoo!, New York Times, Google, Facebook, IBM, Last.fm, Microsoft či LinkedIn. Seznam společností využívají Hadoop je k dispozici na oficiálních strán-

kách Apache Hadoop [16].

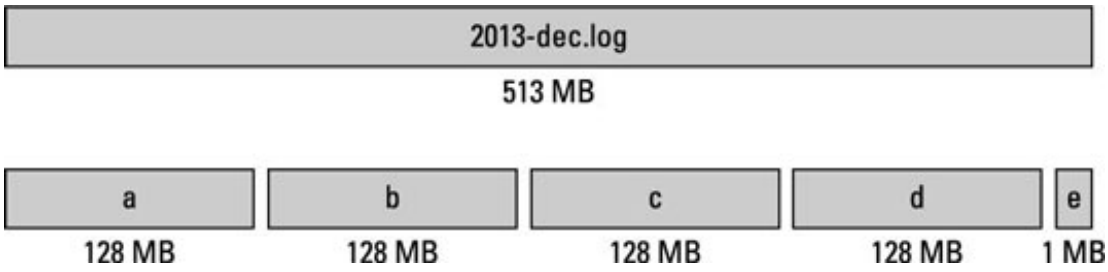
4.2 HDFS

Distribuovaný úložný systém využívaný Hadoopem tzv. Hadoop Distributed File System (zkráceně HDFS) jsme již zmínili v úvodu kapitoly. Tato práce je zaměřena na zpracování velkého objemu dat a proto zde čtenáři vysvětlíme základní princip distribuovaného ukládání Hadoop.

Jak Hadoop, tak i Hadoop Distributed File System jsou především určeny pro běžně a cenově dostupný hardware. HDFS je tedy velmi podobný ostatním distribuovaným souborovým systémům, jako jsou Google Distributed System (GFS), Network File System (NFS), Distributed File System (DFS), avšak obsahuje drobné rozdíly například velikost ukládaných bloků či tvorba replikací. Architektura HDFS je navržena tak, aby bylo možné uložit stovky megabajtů, gigabajtů či terabajtů. Dnes jsou i takové clustery využívající Hadoop, které dokáží ukládat a zpracovávat petabajty dat. [17] [18]

4.2.1 Bloky dat

HDFS má za úkol ukládat a načítat data a autoři Hadoop navrhli využívat bloky dat, jak je tomu i u ostatních filesystémů. Na rozdíl od jiných filesystémů má jeden blok dat v HDFS mnohem větší základní velikost a to 64 MB na jeden blok. Obdobně jako u filesystémů určených pro jeden disk jsou data dělena do pevně zvolených bloků, které se chovají jako samostatné jednotky. Tyto bloky jsou ukládány na jednotlivé uzly clusteru. Obrázek 4.1 zachycuje rozkládání velkého objemu dat do bloků.



Obrázek 4.1: Rozložení souboru do HDFS bloků
Zdroj: [19]

Správné rozložení dat na uzly clustru totiž významně snižuje čas potřebný ke zbytečnému přenosu dat mezi uzly, což zvyšuje rychlost prováděných výpočtů v Hadoop a snižuje objem uložených dat na uzlech clustru. Tedy je nutné nejen vhodné výpočetní paradigma, ale také správné rozdělení dat na nodech, jež zlepšuje výkonnost distribuovaných výpočtů.

4.2.2 Uzly Hadoop

Data se v systému Hadoop skládají z bloků a jsou ukládány na nodech (uzlech), kde jsou dále zpracovány. HDFS podporuje tradiční hierarchický souborový systém a tedy uživatelé i aplikace mohou na uzlech vytvářet složky a ukládat do nich potřebné soubory. HDFS pracuje na principu Master/Slave, a proto má uzly rozděleny na dva typy NameNode a DataNode. Obsahuje-li Hadoop cluster pouze jeden uzel, tak se jedná o tzv. Sigle Node Cluster. Multi Node Cluster nazýváme takový cluster, který obsahuje dva a více uzlů.

NameNode

Mater uzel v HDFS se nazývá NameNode a bývá jediným uzlem v clustru. Základní úkolem NameNode je řízení a správa ukládání dat na uzly v HDFS, kterým se říká DataNodes (viz. DataNode). NameNode dále poskytuje přístup uživatelům, obsahuje uložené informace o souborové struktuře v HDFS a uložených blocích na DataNodech. Tyto tzv. metadata jsou ukládány lokálně na master nodu. Master node je zpravidla využívám pouze pro správu a řízení DataNodu, ale pokud je relativně malý cluster, tak je možné master uzel využít i jako DataNode.

SecondaryNameNode

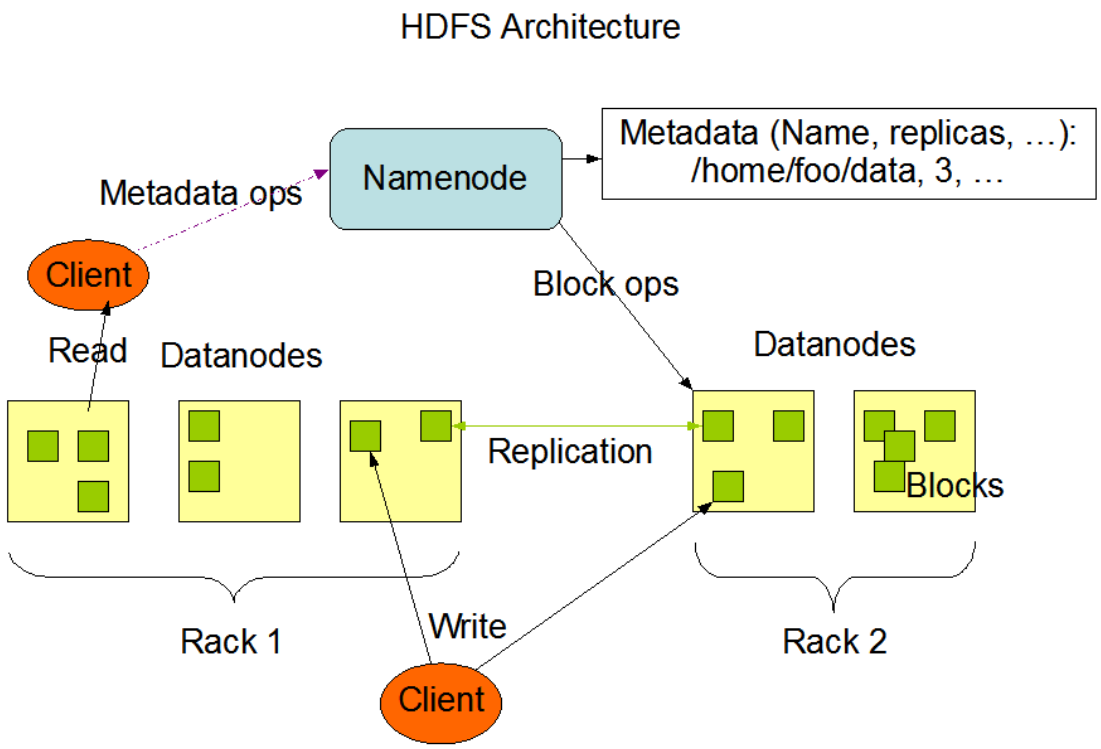
Jelikož jsou metadata a data na NameNodu lokálně ukládány, je velmi vhodné využívat záložního master uzlu nazývaného SecondaryNameNode, pokud by došlo k selhání NameNodu. NameNode označujeme v systému jako bod selhání (z angl. Single Point of Failure) jež znamená, že výpadek NameNodu by mohl způsobit zhroucení celého Hadoop systému. Z tohoto důvodu je doporučováno využívat SecondaryNameNode, který se pravidelně (interval připojení lze definovat v konfiguraci Hadoop) připojuje k NameNode a provádí jeho celkovou zálohu. Dojde-li tedy k výpadku NameNode, tak SecondaryNameNode

je spuštěn a nahradí ho ve správě uzlů HDFS.

DataNode

Z výše uvedeného textu vyplývá, že DataNode je Slave uzel, který obsahuje uložené bloky dat a informace o těchto blocích periodicky zasílá na NameNode. DataNode podporuje čtení, zápis, tvorbu bloků, mazání bloků a jejich replikaci, dle informací získaných od NameNode. DataNode primárně zajišťuje výpočetní výkon pro cluster. Pokud využíváme Single Node Cluster, tak NameNode je zároveň DataNode.

Na obrázku 4.2 je vyobrazena struktura uzlů v HDFS.



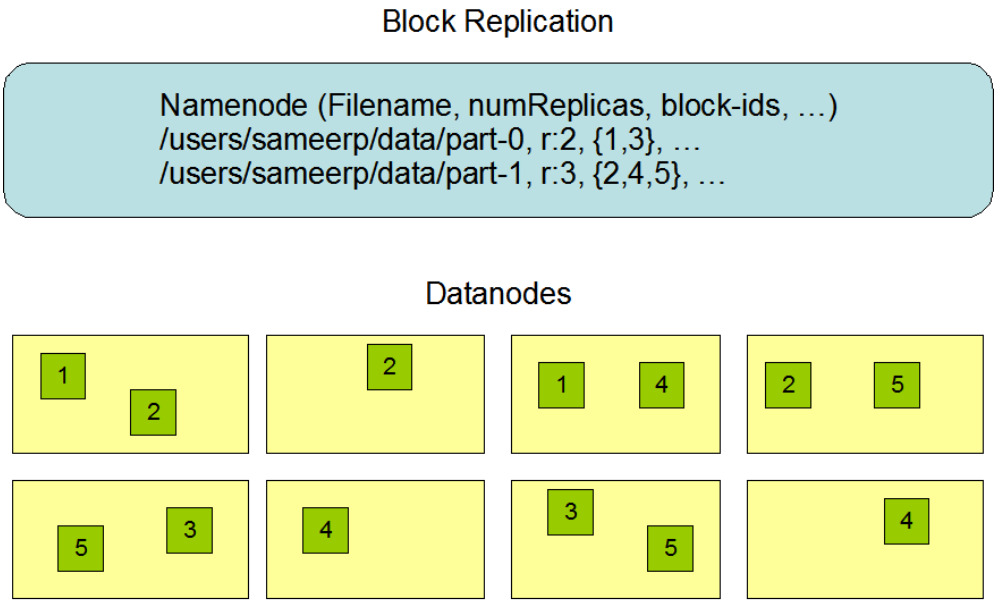
Obrázek 4.2: Architectura HDFS
Zdroj: [20]

4.2.3 Replikace

HDFS je navrženo tak, aby se samo staralo o vzniklé chyby a to i v případě čtení dat. Proto bylo navrženo replikování bloků dat a jejich ukládání na různé uzly

clusterového systému. Replikace jsou také ovšem dobré k tomu, že ne všechny bloky potřebné pro zpracovávání je nutné přenášet, jelikož tyto bloky jsou již na daném uzlu nebo se nacházejí na uzlu, který je blíže pro přenos.

Uvažme tedy to, že datové uzly, které obsahují bloky dat potřebné k výpočtu, jsou zaneprázdněny výpočtem jiných úloh. Je tedy nutné tyto bloky přesunout na nevyužívaný datový uzel, aby mohl začít výpočet na tomto uzlu. Tento přesun ovšem stojí čas potřebný ke kopírování kýžených bloků dat a tento fakt významně ovlivňuje výkon celého systému Hadoop. Proto je určitě vhodné využívat replikaci dat, která přinese zlepšení výkonu Hadoop běžícím na clusteru. Musíme, ale brát v potaz to, že s větším počtem replikovaných dat roste nárok na velikost lokálních disků, které jsou na data nodech.

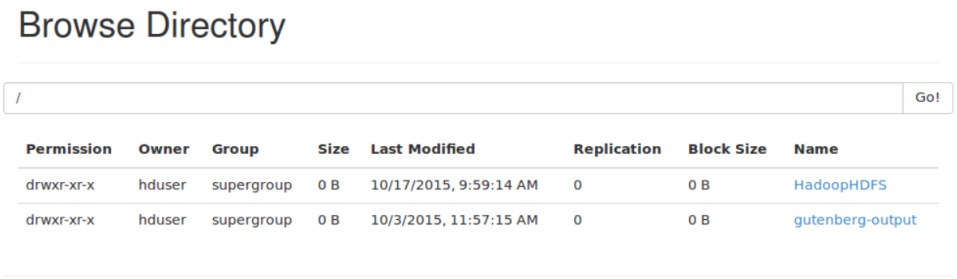


Obrázek 4.3: Replikace v HDFS
Zdroj: [20]

Obrázek 4.3 ukazuje možnou replikaci bloků dat na uzly clusteru. Jak je z obrázku zřejmé, jedná se o replikaci pro bloky dat s názvem 1, 2, 3, 4, 5. Pro data part-0 je rozdělení do bloků 1 a 3 a je použita hodnota 2 pro počet replikací. Data part-0 jsou tedy uloženy ve 2 kopiích. Data part-1 jsou rozděleny do bloků 2, 4, 5 a je nastaven počet replikací na hodnotu 3. Ve spodní části obrázku je znázorněné rozdělení bloků dat na 8 datových uzlů včetně jejich replikací.

4.2.4 Ovládání HDFS

Tato podkapitola přináší zmínku o základní orientaci v souborovém sytému Hadoop. Jak jsme v úvodu kapitoly zmínili, tak HDFS je velmi obdobný jiným souborovým systémům. HDFS lze spravovat i pomocí grafického rozhraní, které ovšem je zpřístupněno především v komerčních distribucích Hadoop. Hadoop samotný umožňuje také přístup skrze webové rozhraní, avšak zde je uveden pouze přehled informací o aktuálním stavu uzlů v Hadoop clusteru. Pomocí webového rozhraní lze samozřejmě procházet HDFS. Na obrázku 4.4 je náhled na správu HDFS skrze webové rozhraní.



Obrázek 4.4: Správa HDFS pomocí webového rozhraní
Zdroj: vlastní tvorba

Standardní přístup správy HDFS je ovšem přes příkazový řádek (jak je mnoho uživatelů zvyklých z Unixových distribucí OS). Příkazy jsou velmi podobné jako systémové, akorát před tyto příkazy vždy musíme přidat prefix „hadoop fs“. Příkladem uvádíme příkaz „-ls“, který slouží zobrazení seznamu souborů/složek nad spouštěnou složkou obdobně, jako v Unixovém systému. Základní příkaz a jeho možnosti vypadají následovně:

$$hadoop fs - ls[-d][-h][-R][-t][-S][-r][-u] < args >$$

a příklad použití nad složkou dir1:

$$hadoop fs - ls/user/hadoop/dir1$$

Výsledkem tohoto příkazu je seznam složek a souborů nalézajících se ve složce dir1. Úplný seznam i s příklady použití naleznete na stránce File System Shell Guide [21].

4.3 MapReduce

V této části práce seznámíme čtenáře s programovacím paradigmatem MapReduce a se správcem úloh MapReduce2 neboli YARN, který nahradil původního správce úloh MapReduce.

MapReduce je programovací model pro zpracovávání dat a využívá se především pro velkoobjemová data. MapReduce se skládá ze dvou základních fází průběhu procesu, jak je vidět z názvu funkce, fáze Map a fáze Reduce. Každá tato fáze má párové vstupy a výstupy. Tyto vstupy jsou volitelné programátorem stejně tak jsou programovatelné i fáze Map a Reduce. My zde uvádíme upravenou verzi paradigma MapReduce, která je rozšířena o tzv. Combiner, který funguje tak, že předpřipraví výstupní data z funkce Map a vstupující data do funkce Reduce.

Uvažme velký objem dat, který chceme zpracovat pomocí paradigma MapReduce. Do Map funkce (tzv. Mappru) vkládáme velký počet dat, který je mezi sebou mapován do zvolené struktury. Tato struktura je navržena programátorem (např. slovo a číslo). Výsledné dvojice Mappru jsou předány funkci Combiner, který upraví výsledek Mappru a připraví jej tak pro funkci Reduce (tzv. Reducer), která má za úkol spojit a následně vytvořit koncový výstup z modelu MapReduce. Průběh MapReduce můžeme znázornit následovně:

$$(input) < k1, v1 > -> map -> < k2, v2 > -> combine -> < k2, v2 > -> reduce -> < k3, v3 > (output)$$

Pro názornost jsme zvolili jednoduchý příklad, na kterém ukážeme základní princip modelu MapReduce. Tento příklad je k dispozici na [14] s rozšířenějším výkladem a dalšími příklady, kde jsou funkce Map a Reduce upraveny pro dosažení jiných výsledků. Příklad je nazýván WordCount a funguje tak, že program WordCount si z HDFS načte zdrojová data, v našem případě textové soubory a spočte, kolikrát se dané slovo vyskytne v námi nahraných datových souborech.

V našem příkladu budeme uvažovat dva vstupní textové soubory. První soubor – text1 – obsahuje tyto údaje: „Hello World Bye World“ a druhý – text2 – „Hello Hadoop Goodbye Hadoop“. Mapper zde funguje tak, že jeho vstupem je $<< slovo >, 1 >$, kde $< slovo >$ je nahrazeno jedním slovem z našeho vstupu a číslo 1 je ohodnocení, které zůstává stejné pro všechny slova. Výsledný výstup funkce Map popisuje tabulka 4.1.

soubor text1	soubor text2
<Hello, 1>	<Hello, 1>
<World, 1>	<Hadoop, 1>
<Bye, 1>	<GoodBye, 1>
<World, 1>	<Hadoop, 1>

Tabulka 4.1: Mapování vstupních souborů
Zdroj: vlastní tvorba

Dále proběhne nad každým výsledkem Mappru tzv. Combiner (probíhá lokálně). Funkci Combiner ponecháme v našem případě stejnou, jako Reducer (obecně toto lze nastavit v konfiguraci pro dané zpracovávání úloh). Vzniklé mapy jsou popsány tabulkou 4.2.

Map pro text1	Map pro text2
<Hello, 1>	<Hello, 1>
<World, 2>	<Hadoop, 2>
<Bye, 1>	<GoodBye, 1>

Tabulka 4.2: Výstup z Combineru
Zdroj: vlastní tvorba

Po ukončení všech lokálních Mapprů a Combinerů, proběhne program Reduce nad všemi úlohami a vznikne nám výsledný výstup, který je vyobrazen v tabulce 4.3.

output
<Bye, 1>
<Goodbye, 2>
<Hello, 2>
<Hadoop, 2>
<World, 2>

Tabulka 4.3: Výstup z Reduceru
Zdroj: vlastní tvorba

Z výše uvedeného příkladu je vidět, jak WordCount z k vstupních souborů

vytvoří jediný výstupní soubor pomocí MapReduce, který obsahuje slova a jejich počet výskytů z k vstupních souborů.

4.3.1 YARN (MapReduce2)

Úvodem podkapitoly 4.3 jsme čtenáři nastínili základní fungování paradigmatu MapReduce, ale neuvedli jsme bližší informace o zpracování úloh napříč clusteru. O fungování a spouštění úloh se dříve staral MapReduce verze 1 (původně navržený MapReduce z roku 2005). Hadoop verze 2 a novější dostali nového správce s názvem YARN (také uváděn jako MapReduce2), který byl navržen především z důvodu navyšování výpočetních uzlů v clusteru. Původní MapReduce verze 1 totiž při počtu uzlů nad 4 000 nebo nad 40 000 spuštěných úloh začínal mít problémy se škálovatelností. YARN je navržen až pro 10 000 nodů a více jak 100 000 spuštěných úloh.

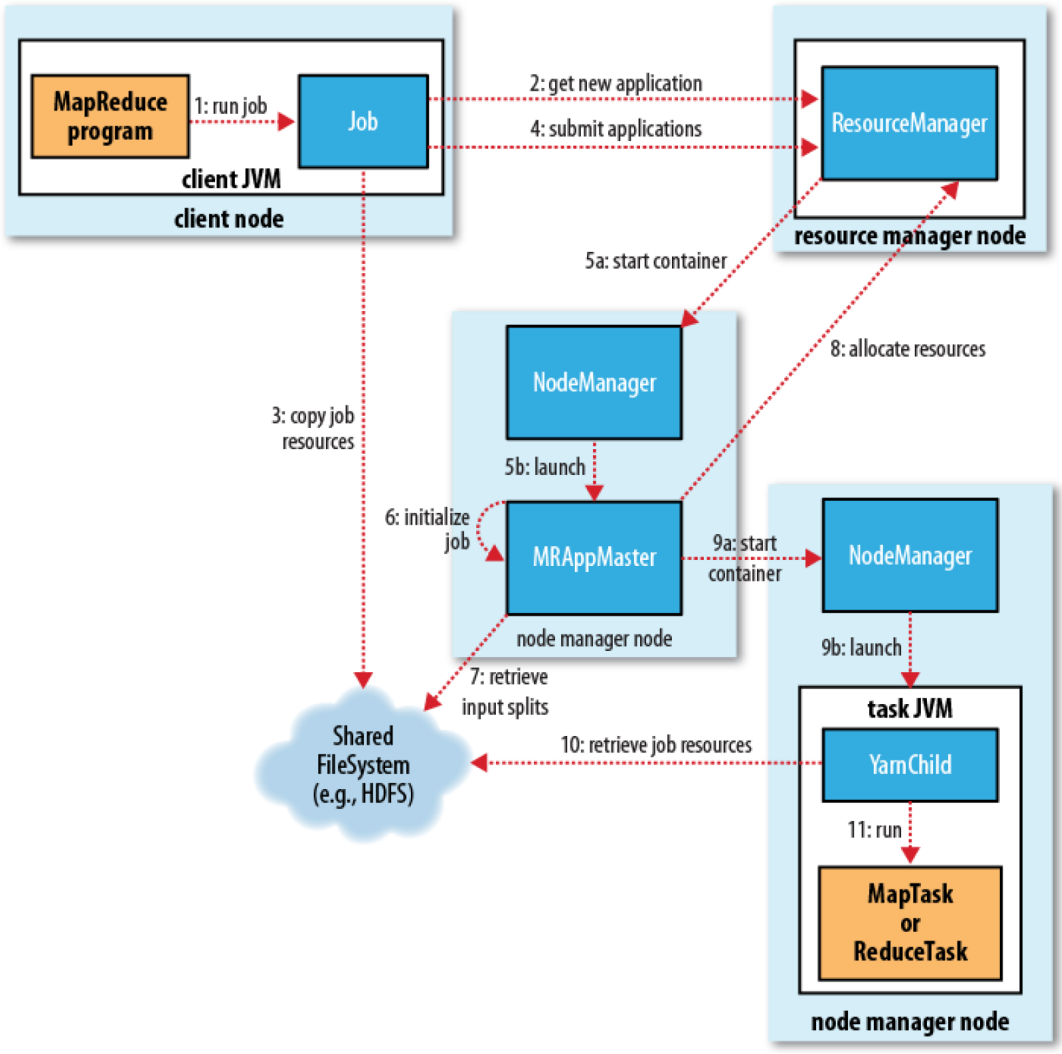
MapReduce2 dnes především znám jako YARN („Yet Another Resource Negotiator“) má za úkol správu prováděných úloh na uzlech a přidělování zdrojů (procesory, paměť, atd.). YARN je umístěn mezi aplikační vrstvou a vrstvou datového úložiště. V našem případě je tedy YARN umístěn mezi MapReduce aplikace a HDFS.

Zpracování MapReduce úlohy v YARN

YARN spravuje úlohy v Hadoop a princip zpracování MapReduce aplikace za pomocí tohoto frameworku je nastíněn na obrázku 4.5. YARN framework se skládá z:

- **Klienstkého uzlu** (client Node) – spouští MapReduce úlohy.
- **Správce zdrojů** (ResourceManger node) – koordinuje alokaci zdrojů v clusteru.
- **YARN node manager** (Manager Node) – spouští a monitoruje kontejnery na uzlech clusteru.
- **Hlavního uzlu aplikace MapReduce** (MRAppMaster) – koordinuje běžící úlohy v MapReduce programu. Spouští kontejnery s úlohami, které jsou schedulované ResourceManagerem.

- **Distribuovaný souborový systém** (HDFS) – využíván pro sdílení souborových zdrojů napříč celého clusteru.



Obrázek 4.5: Architectura YARN
Zdroj: [17]

Uživatelem spuštěná úloha probíhá v následujících krocích (kroky se odkazují na obrázek 4.5):

Spuštění programu

Nahrání programu MapReduce (*krok 1*). Po nahrání programu uživatelem je vygenerované ID, které je přiřazeno ke spuštěnému programu. ID je vygenero-

váno ResourceManagerem (*krok 2*). Poté jsou zdroje potřebné programem soubory JAR, konfigurace a dělící mechanizmy nahrány do HDF (*krok 3*). Následně je program nahrán do ResourceManagera (*krok 4*).

Inicializace programu

Po přesunutí programu do ResourceManagera je volán scheduler, který alokuje kontejner a ResourceManager spustí hlavní proces aplikace, který běží na Manager nodu (*krok 5a* a *krok 5b*). Dále jsou spuštěny úlohy aplikace, které běží ve správě MRAppMaster (*krok 6*). Po inicializaci úloh jsou získány z HDFS potřebné datové zdroje žádané úlohami (*krok 7*). Po dokončení získání potřebných datových zdrojů, je zjištěna informace, zda se bude provádět výpočet na více uzlech a nebo v již běžícím JVM MRAppMaster uzlu. Pokud je rozhodnuto, že úloha poběží v MRAppMaster uzlu, je tato úloha nazvána uber neboli malá úloha. Úloha uber (malá úloha) musí splňovat tyto požadavky:

- obsahovat maximálně 10 mapprů,
- pouze jeden reducer,
- velikost vstupních dat musí být menší než jeden HDFS blok.

Výše zmíněné podmínky jsou základním nastavením, které je možné přenastavit, dle požadované konfigurace. Ještě je nutné zmínit, že než je jakákoli úloha spuštěna, tak je vytvořena složka, kam se budou ukládat výstupy.

Přiřazení úloh

V případě, že se nejedná o úlohu typu uber jsou pro všechny mappry a reducery úlohy vyžádány alokace zdrojů od ResourceManagera, který jim přidělí potřebný kontejner pro jejich spuštění.

Provedení úloh

Jakmile je úloze přidělen kontejner od ResourceManagera (*krok 8*) je úloha spuštěna MRAppMaster uzlem. Master uzel kontaktuje uzel, na kterém má úloha být spuštěna (*krok 9a*) a následně je tato úloha inicializována na daném uzlu (*krok 9b*). V rámci inicializace úlohy jsou staženy datové soubory, JAR soubory a konfigurace z HDFS (*krok 10*). V tuto chvíli již nic nebrání spuštění úlohy (*krok 11*).

Informace o průběhu zpracování úlohy

Každá úloha vytváří informace o svém průběhu zpracování, které jsou odesílány do uzlu MRAppMaster. Program spuštěný uživatelem se na tento status dotazuje každou vteřinu (je možné konfigurovat) a stav průběhu zpracování úlohy je mu vrácen z MRAppMasteru uzlu.

Kompletace úloh

Obdobně, jak se klient dotazuje na progresi úloh, tak se každých pět vteřin (je konfigurovatelné) dotazuje, zda program již ukončil svoji činnost. Ukončíli program výpočet, tak je proveden úklid již nepoužívaných kontejnerů, jsou archivovány logy o úlohách a uživatel je informován pomocí zpětného volání (tzv. Callback), které je vyvoláno MRAppMaster uzlem.

4.4 Shrnutí

Cílem této kapitoly bylo seznámit čtenáře s frameworkem Hadoop od společnosti Apache a okolnostmi jeho vzniku. Dále byl čtenář seznámen s fungováním HDFS. U HDFS je důležité mít na paměti to, že při implementaci HDFS na Hadoop cluster je nutné vhodně rozložit lokální zdroje a také nastavit potřebnou konfiguraci pro HDFS, aby nedocházelo ke zbytečnému nárůstu času na výpočet úloh MapReduce. Tento problém může vzniknout především při špatném nastavení replikací. Dále je vhodné využívat záložního uzlu nazývaného SecondaryNameNode, který zabrání nefunkčnosti Hadoop clusteru, pokud dojde k selhání hlavního uzlu spravující HDFS tzv. NameNodu. V poslední části kapitoly jsme nastínili základní fungování MapReduce paradigma a také zpracování úloh typu MapReduce v YARN, který je správcem pro operace probíhající v Hadoop.

5 Open Source software

Kapitola Open Source software má za úkol seznámit čtenáře s vývojem softwaru v Open Source návrhu, jelikož naší aplikaci navrhujeme v rámci open source. Definice pro Open Source software byla vytvořena společností Open Source Initiative (OSI), které byla založena v roce 1998. OSI je celosvětově uznávaná nezisková organizace, která má za úkol vzdělávat, propagovat výhody open source softwaru, sjednocovat rozdíly vznikající v open source komunitě a také udělovat Open Source licence. Základní definice otevřeného softwaru je k nalezení na stránkách OSI [22]. Open source neznamená pouze dostat přístup ke zdrojovým kódům aplikace, ale musí také splňovat další následující kritéria:

- **Volné rozšiřování** – licence k programu nesmí nikoho omezovat v prodeji či další distribuci tohoto programu jako součást celku obsahující programy z různých zdrojů. Licence nesmí vyžadovat poplatek ze takovýto prodej.
- **Zdrojový kód** – program musí obsahovat zdrojový kód a musí být umožněna jeho distribuce jak ve zdrojovém kódu tak v kompilovaném stavu. Pokud není zdrojový kód dodáván k programu, je nutné aby byly zdrojové kódy snadno získatelné za přiměřenou cenu na náklady na vytvoření jejich kopií nebo například stažitelné ze sítě Internet. Není dovoleno úmyslné zmatení kódu a nejsou ani povoleny meziprodukty jako je výstup z překladače či preprocesoru.
- **Odvozené práce** – licence musí povolovat modifikace a odvozená díla a musí také dovolit jejich distribuci za stejných podmínek jako originální software.
- **Integrita autorova zdrojového kódu** – licence může omezit distribuci modifikovaného zdrojového kódu pouze tehdy, jestliže umožňuje distribuci balíčků (patchů) spolu se zdrojovým kódem za účelem modifikace programu při jeho kompilaci. Licence musí výslovně dovolovat distribuci

softwaru vytvořeného z modifikovaného zdrojového kódu. Licence může vyžadovat, aby odvozená díla nesla odlišný název než původní produkt nebo jiné číslo verze.

- **Diskriminace vůči osobám nebo skupinám** – licence nesmí diskriminovat žádnou osobu či skupinu osob.
- **Diskriminace sfér použití** – licence nesmí nikoho omezovat v používání programu při konkrétním snažení.
- **Šíření licence** – práva spojená s programem se musí vztahovat na každého, komu byl program redistribuován bez nutnosti akceptovat další dodatečné licence.
- **Licence nesmí záviset na programovém produktu** – práva spojená s programem nesmí záviset na tom, zda je program částí určité softwarové distribuce.
- **Licence nesmí ovlivňovat ostatní programy** – licence nesmí stanovit omezení na jiný software distribuovaný společně s licencovaným softwarem.
- **Licence musí být technologicky neutrální** – žádné ustanovení licence nesmí být na jakékoli individuální technologii či stylu rozhraní.

V níže uvedeném výčtu zdůrazňujeme hlavní důvody proč jsme se rozhodli pro volbu navrhnout naší aplikaci, jako open source software jsou následující:

- Uživatel může zdrojový kód upravit dle svých potřeb.
- Otevřený kód větší komunitě znamená možnost jeho zlepšení v různých směrech jako je zjednodušení kódu či zlepšení výkonnosti algoritmů.
- Otevřený kód může také pomoci v dalším vývoji či rozvoji aplikací nebo jejich autorů.

6 Shrnutí teoretické části

Data jsou vytvářena každou minutu a to nejenom lidmi, ale velkou část tohoto objemu vytvářejí samotné počítače, kamery, senzory a další zařízení, která data zaznamenávají a ukládají je pro další zpracování. Vytvořená data je třeba zpracovat podle potřeb uživatele, které jsou zpravidla odlišné i když se jedná o stejná data. Například lidé v obchodním centru starající se o nákup produktů od dodavatelů chtějí mít informace o počtu prodaných/skladovaných kusů a informace o cenách, které nabízejí dodavatele, aby mohli řešit doplnění zboží v čas. Zatímco lidé starající se o reklamní kampaně prodeje produktů na stejných datech chtějí znát informace o prodaných produktech, jejich umístění v obchodě či probíhajících reklamních kampaní pro dané produkty, aby mohli zajistit vyšší prodej daných produktů.

Objem zaznamenávaných dat začíná narůstat do té míry, že již nejsme schopni data zpracovávat v reálném čase na daných počítačích či zařízeních, která data zaznamenávají, a proto je potřeba hledat další možnosti zpracování dat. Jednou z možností zpracování velkoobjemových dat jsou distribuované výpočty, které umožňují propojit větší výpočetní výkon napříč celým světem pomocí sítě Internet. Bližší informace o distribuovaných výpočtech uvádíme v kapitole 3. Jeden z typů distribuovaných výpočtů je i dobrovolnické počítání, kde se uživatelé dobrovolně přihlásí do projektů, kterým chtějí poskytnout svůj výpočetní výkon. Zpravidla se jedná o vědecké projekty, které chtějí dobrovolníci podpořit. Vzhledem k velkému počtu počítačů, které mají výpočetní výkon nevyužitý, například když jdou jejich uživatelé na oběd, sledují videa, poslouchají hudbu či vykonávají kancelářskou práci. Právě tento nevyužitý výpočetní výkon je možné poskytnout na podporu různých projektů. Dobrovolnické výpočty mají výhodu v tom, že i projekty s malým finančním rozpočtem mohou získat velký výpočetní výkon, jelikož dobrovolnický výkon nelze koupit, ale záleží pouze na dobrovolnících jakému projektu poskytnout svůj počítač.

Právě jedním z osvědčených nástrojů na zpracování velkoobjemových dat je Apache Hadoop, který je navržený a optimalizovaný na to, aby zvládl velmi

rychle zpracovat tato data a uživateli vrátit potřebné informace. Hadoop poskytuje velmi dobrou škálovatelnost a to především po přechodu na novější verzi MapReduce2 často označovanou také jako YARN, která přinesla lepší správu prováděných úloh v Hadoop clusteru. S Apache Hadoop byl čtenář seznámen v kapitole 4, kde detailněji popisujeme jeho architekturu a navržené paradigma MapReduce a naznačujeme práci s HDFS.

7 Návrh přístupu a aplikace

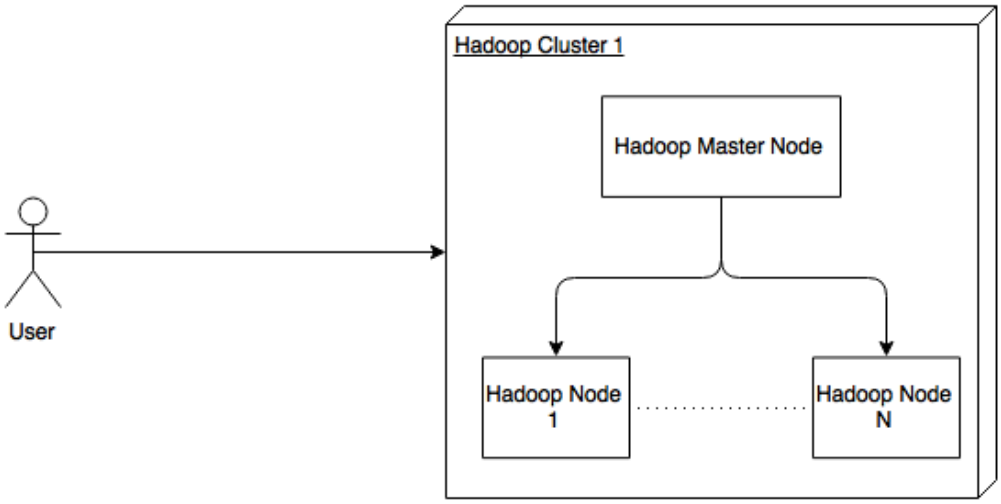
Cílem praktické práce je především popsat návrh našeho přístupu pro zpracování velkoobjemových dat, vytvořit návrh aplikace, která má zlepšit práci uživatelům tím, že zautomatizuje a zjednoduší komunikaci mezi uživatelem a Hadoop Clustery, poskytne možnost většího výpočetního výkonu a rozložení zpracovávání dat na více clusterů. Dále v návrhu aplikace nastiňujeme jak by mohlo vypadat rozhraní pro uživatele, základní funkčnosti a metody potřebné pro zpracovávání dat.

7.1 Návrh přístupu

V teoretické části jsme nastínili, proč je důležité se zabývat zpracování velkoobjemových dat a přiblížili jsme dnes využívané možnosti. Nejčastějším přístupem zpracování velkoobjemových dat je jeden cluster obsahující stovky či tisíce uzlů, které zpracovávají všechny požadavky uživatelů. Jsou-li uzly využity naplno, je možné tento cluster rozšířit o další uzly. Využíváme-li Hadoop, tak jeden cluster má omezení na maximum 10 000 uzlů a 100 000 spuštěných úloh, jak jsme zmiňovali v kapitole 4.3.1.

Obrázek 7.1 zachycuje tradiční přístup jednoho uživatele ke clusteru. Jakékoli výpočty, které probíhají na clusteru jsou řízeny managerem, jež je zodpovědný za jejich správu, jejich spuštění a ukončení. Jakmile je program ukončen, tak výstupní data jsou uložena na HDFS a klient je informován o skončení výpočtu. K danému Hadoop clusteru ovšem nepřistupuje pouze jeden uživatel, ale desítky, stovky, tisíce či miliony zároveň a to v závislosti na daném využívaní clusteru.

S větším počtem uživatelů bude růst doba potřebná pro zpracování všech úloh. Navíc v tomto případě je nutné, aby klient dodal nejen data potřebná ke zpracování, ale také program napsaný v podporovaném jazyce, jak se mají data zpracovávat a co má být výstupem. Dnes různé firmy, jako je například Clou-



Obrázek 7.1: Klasický přístup uživatele k Hadoop Clusteru
Zdroj: vlastní tvorba

dera (viz. [23]) poskytují možnosti pro firemní řešení, které je založeno na využívání Hadoop clusteru.

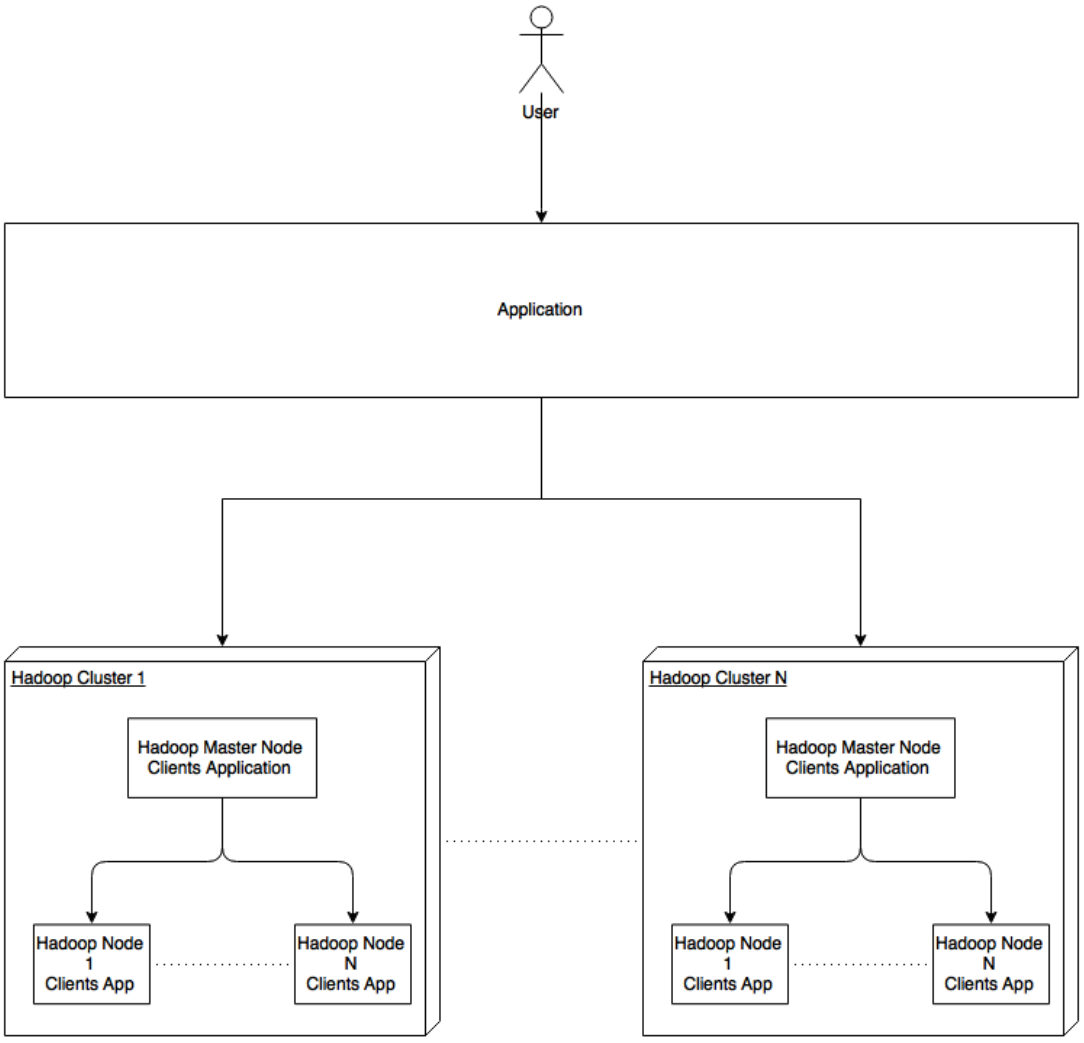
Mnoho uživatelů využívající sílu distribuovaných výpočetních center jsou vědečtí pracovníci, kterým jde především o správnost koncového výsledku a už tolik neřeší správnost kódu, nadměrné podmínky, cykly či připojovací skripty. Všechny tyto nedostatky mohou vést k navyšování doby nutné pro zpracování úloh. Nevhodně implementované podmínky či cykly nejen mohou navyšovat čas pro zpracování, ale také může docházet k chybným výpočtům, či dokonce může dojít k pádu celého clusteru.

Námi navržený přístup, jak zpracovávat velkoobjemová data spočívá v propojení technologií distribuovaných výpočtů, a to modelu dobrovolnického počítání (kapitola 3.3.5), a Apache Hadoop (kapitola 4). U dobrovolnických výpočtů se uživatelé zapojují jako výpočetní uzly, které dodávají potřebný výkon jež je využíván pro náročné výpočty. Uzlem definujeme, jakékoli zařízení (stolní počítač, notebook, tablet apod.), které je schopné provádět potřebné výpočty, má hardwarovou výbavu pro zapojení do počítačové sítě, aby bylo možné jej připojit do clusteru a jež umí pracovat s Apache Hadoop tj. obsahuje podporovaný OS. Apache Hadoop zatím podporuje operační systémy Linux a Windows, a jak

autoři Apache Hadoop uvádí v [24], tak i BSD, Mac OS/X a OpenSolaris se dají využít pro Apache Hadoop.

Všechny tyto uzly se přihlašují do většího celku, ze kterého vzniká jeden Hadoop cluster. Přihlášení do clusteru probíhá za pomoci klientské aplikace, která je nainstalovaná v potřebném zařízení obdobně, jak je tomu u dobrovolnického počítání. Každý Hadoop cluster si poté spravuje uzly a využívá jejich výpočetního výkonu. Přidání uzlů do clusteru může probíhat dynamicky za běhu clusteru bez nutnosti jeho restartování či vypínání jak je popsáno v [24]. Tato možnost nám poskytuje přidávat uzly do clusteru v případech, kdy nejsou zařízení používána a mohou tedy zpracovávat data pouze v určitých intervalech, kdy je uživatel aktivně nepoužívá (specificky definované intervaly jako je například doba oběda).

Všechny Hadoop clustery ať jsou sestavovány dynamicky nebo mají pevně danou strukturu se přihlásí do námi navržené aplikace. Aplikace slouží jako mezičlánek mezi uživatelem a Hadoop clustery, pro připojování uzlů do Hadoop clusterů za pomoci klientské aplikace a pro správu přihlášených Hadoop clusterů. Umístění aplikace, Hadoop clusterů a nodů v návrhu je zobrazena na obrázku 7.2.



Obrázek 7.2: Návrh rozšířeného přístupu pro zpracování dat
Zdroj: vlastní tvorba

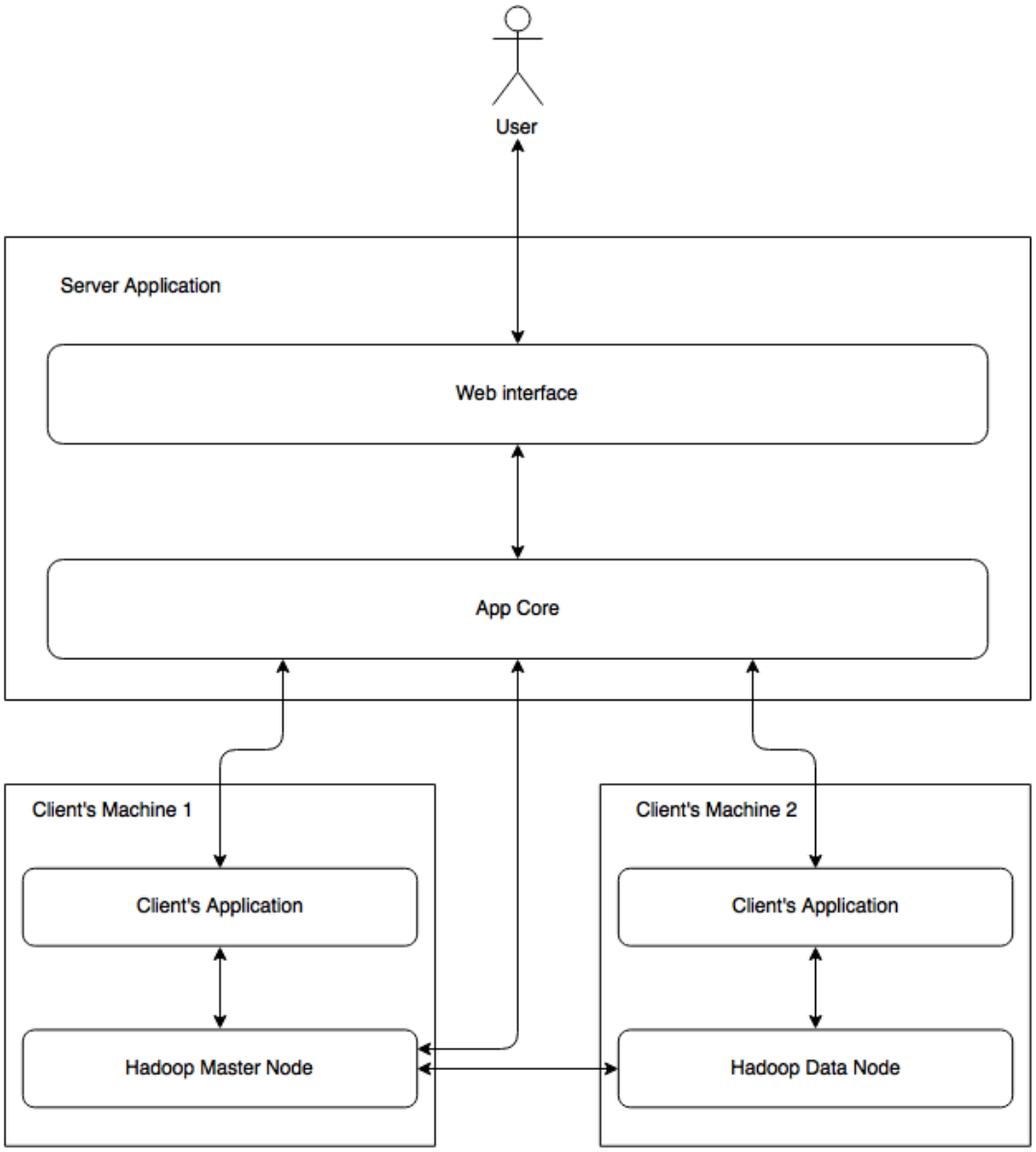
Než se pustíme do hlubšího seznámení s návrhem aplikace, tak bychom chtěli uvést reálné situace využití našeho přístupu pro zpracování velkoobjemových dat za pomoci distribuovaných výpočtů. Vezmeme příkladem naší univerzitu, která má k dispozici 11 budov, kde každá budova může fungovat jako jeden Hadoop cluster. V každé budově je mnoho zařízení, které jsou vzájemně propojeny pomocí počítačové sítě. Tyto zařízení nejsou nikdy využívána na 100 %, ale přesto jsou v provozu skoro 24 hodin denně 5 dnů v týdnu - uvážíme-li, že v sobotu a v neděli jsou budovy bez provozu a počítače jsou vypnuté. Především ve večerních hodinách již na zmiňovaných počítačích nikdo nepracuje

a tak je možné je využívat jako výpočetní uzly v rámci Hadoop Clusteru. Právě Hadoop je velmi vhodný pro využívání na běžném hardwaru, jelikož už od vzniku si jeho autoři kladli toto jako cíl. Všechny budovy/clustery univerzity jsou připojeny na server pomocí počítačové sítě, kde běží námi navržená aplikace, které řídí správu těchto clusterů. Tento výpočetní výkon bude poté možné využívat pro univerzitní účely, jako jsou náročné výpočty v bakalářských, diplomových či výzkumných prací nebo pro nabídnutí tohoto výkonu jako komerční využití pro externí firmy. Další výhodou je to, že se nemusí investovat velký obnos peněz, jelikož výpočetní uzly, master uzel Hadoop clusteru již univerzita vlastní protože se jedná o běžné počítače, které jsou pomalu na každém stole v budově.

Obdobný vzor můžeme aplikovat například na nemocnice. V mnoha fakultních nemocnicích jsou oddělení, které využívají pro svoji vědeckou práci náročné výpočty. Tyto výpočty je potřeba zpracovat, a proto jsou využívány různé výpočetní střediska, které si nechávají za požadovaný výkon zaplatit. Pokud využijeme námi navržený přístup, tak každá z nemocnic se bude chovat jako jeden Hadoop cluster a pomocí sítě Internet budou zapojeny do aplikace, která provádí správu nad clustery a tím vznikne obrovský výpočetní výkon, který může částečně či úplně nahradit výkon výpočetních středisek dodávaných externími firmami.

7.2 Popis aplikace a základních funkčností

Z výše uvedeného návrhu našeho přístupu pro zpracování velkoobjemových dat plyne, že využíváme dvě základní aplikace. Hlavní aplikace bude běžet na serveru tzn. serverová aplikace. Pro registrované uživatele do serverové aplikace bude možné si stáhnou příslušnou klientskou aplikaci, která bude instalována na uzly jež se budou připojovat do Hadoop clusterů spravovaných touto serverovou aplikací.



Obrázek 7.3: Návrh komunikace mezi aplikacemi a klientskými stanicemi
Zdroj: vlastní tvorba

Na obrázku 7.3 je vyobrazen základní návrh komunikace mezi serverovou alikací („Server Application“), uživatelem („User“), klientskou aplikací („Client's Application“) a Hadoop Clustery („Hadoop Master Node“ či „Hadoop Data Node“). Serverovou aplikace („Server Application“) se skládá ze dvou základních pilířů a to Webového rozhraní („Web interface“) a jádra aplikace

(„App Core“).

Jak je vidět na obrázku 7.3, tak uživatel („User“) komunikuje pouze se serverovou aplikací prostřednictvím webového rozhraní, které mu poskytuje potřebné informace. Uživatel totiž nepotřebuje a často ani nechce znát informace, kolik strojů, kde jsou výpočty zpracovávány či kdo zpracovává jeho výpočty pokud je zajištěno dostatečné bezpečí dat, ale především ho zajímají informace, zda zpracovávané úlohy doběhly či došlo k chybě, jaké úlohy byly zpracovávány, jak dlouho jejich zpracování trvalo a podobně. Z těchto důvodů jsme připravili základní návrh webového rozhraní v podkapitole 7.2.2.

Právě za uživatele všechnu komunikaci s Hadoop Clusterem zajišťuje jádro aplikace, které nahrazuje původně pro běžné uživatele náročný postup nahrávání dat, spouštění úloh či získávání informací o průběhu zpracovávaných úloh. Jádro aplikace navíc zajišťuje směrování úloh na clustery, které nejsou plně využívány, aby úlohy byly zpracovávány v co nejkratší době.

Pokud by tohoto chtěl uživatel dosáhnout v původním návrhu viz. obrázek 7.1, tak by musel na všech Hadoop clusterech zjistit potřebné informace o průběhu zpracovávání úloh na clusterech a podle těchto informací zvolit cluster, který bude jeho data zpracovávat. Další výhodou přístupu uživatelů ke clusterům pomocí aplikace je to, že stovky či tisíce uživatelů nezahlcuje Hadoop Cluster dotazy na zpracovávání jejich úloh, ale tyto dotazy jsou prováděny skrze serverovou aplikaci, která je na to připravena. Jelikož Hadoop Master Node je připravován pro běžný hardware, tak není vždy zaručeno, že mnoho dotazů na jednou bude zvládat.

Serverová aplikace dále komunikuje s klientskými stanicemi („Client's Machine“) a to pomocí klientské aplikace či přímo s Hadoop Clusterem (Hadoop Master Nodem), který poskytuje potřebné informace o průběhu zpracovávání úloh. Pokud by došlo k selhání Hadoop Master Nodu a bude existovat Hadoop Secondary Master Node, tak serverová aplikace začne komunikovat s tímto uzlem, jako by byl Master Node. Komunikace mezi klientskou aplikací a serverovou je blíže rozebrána v kapitole 7.2.1. Jádro aplikace komunikuje s Hadoop a klientskou aplikaci pomoci výměny XML/JSON objektu, které jsou zasílány skrze internetovou sít pomoci protokolu HTTP/HTTPS, který je běžně využívat pro komunikaci skrze internet.

7.2.1 Klientská aplikace

Aplikace pro klientská zařízení má za úkol správu připojení daného zařízení, na kterém je nainstalován Hadoop, do clusteru. Tento klient po instalaci zjistí zda uživatelské zařízení, má instalovaný Hadoop a nabídne uživateli možnosti nastavení v grafickém rozhraní. Pokud uživatel nemá nainstalován Hadoop, tak aplikace mu nabídne možnost instalace. Pokud uživatel zvolí instalaci Hadoop, aplikace zařídí stažení potřebných souboru ze serveru a provede uživatele instalační průvodcem.

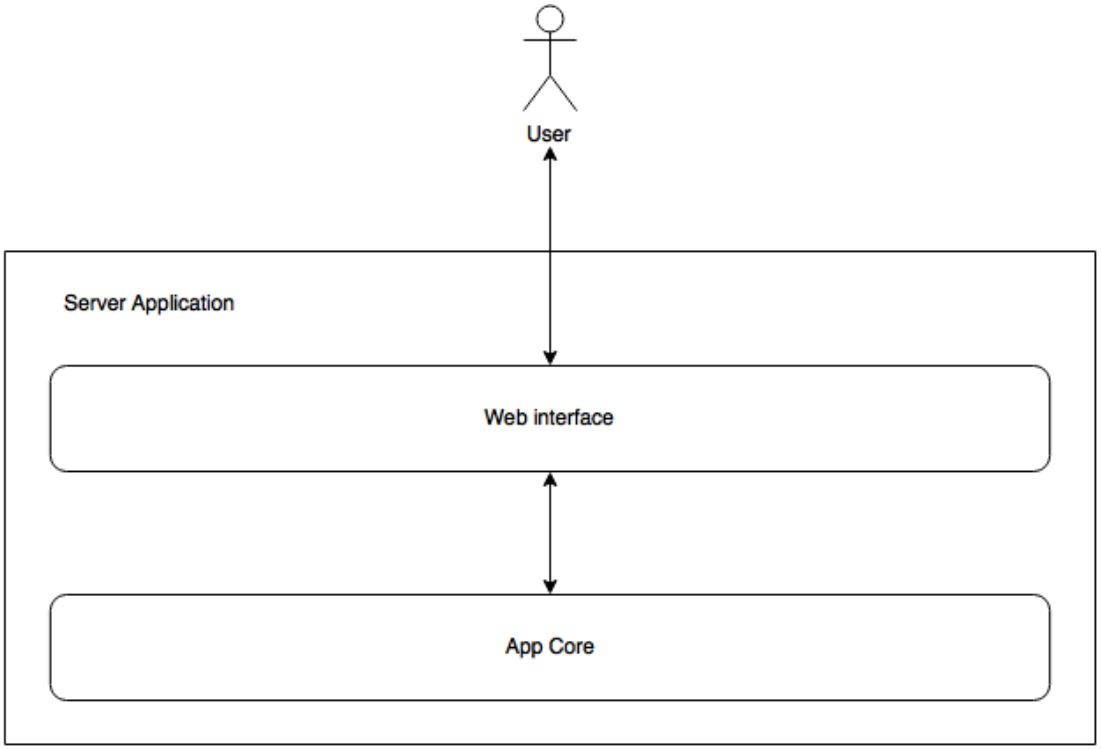
Po ověření existence či v průběhu instalace Hadoop je uživateli zobrazeno nastavení Hadoop clusteru. Uživatel má možnost si zvolit preferovaný Hadoop cluster (jeden cluster připojený do serverové aplikace), ke kterému se bude zařízení připojovat, zvolit automatické připojení nebo zastávat pozici NameNodu tj. stát se Hadoop clusterem. Pokud uživatel zvolí možnost preferovaného Hadoop clusteru zařízení se připojí vždy do tohoto clusteru. Uživateli je při prvním spuštění aplikace nabídnut seznam dostupných Hadoop clusterů, ke kterým se může připojit. Jakmile uživatel potvrdí volbu tohoto clusteru, tak klientská aplikace zařídí potřebné připojení. Při opětovném přihlášení do zvoleného clusteru již není nutné kontaktovat serverovou aplikaci a připojení daného DataNode do clusteru je prostřednictvím klientské aplikace. Dojde-li při připojování k chybě - cluster obsahuje maximální počet uzlů, NameNode není v provozu apod., tak je tato informace odeslána na server a uživateli je daný problém oznámen a aplikace mu nabídne seznam dostupných Hadoop clusterů, ke kterým je možné se přihlásit.

Zvolí-li uživatel automatické připojení, tak při každém připojení klientská aplikace kontaktuje serverovou aplikaci, která vybere vhodný Hadoop cluster, předá tyto informace na klienta a ta zařídí připojení výpočetního uzlu obdobně, jak je tomu při volbě preferovaného clusteru. Výhodu automatického zvolení clusteru je především vyvažování zátěže na clusterech, další roli hraje vzdálenost připojeného uzlu od NameNodu, který komunikuje v pravidelných intervalech s nově připojeným uzlem a také jsou mezi nimi vyměňovány soubory potřebné pro jejich zpracování. Právě automatické nastavení je preferovanějším způsobem připojení uzlu a to především pro mobilní zařízení jako jsou notebooky či tablety. Poslední možností je stát se NameNodem ke kterému se budou připojovat další DataNody a tím vznikne nový Hadoop cluster. Dále může uživatel specifikovat maximální využití operační paměti, CPU a datového úložiště.

Tyto zmiňované informace se propisují po potvrzení do konfigurace Hadoop pro daný uzel. U zařízeních, které jsou využívány přes den k jiným aktivitám než jsou výpočty prováděné Hadoopem, tak je možné nastavit intervaly pro připojení do takto vytvořené sítě.

7.2.2 Serverová aplikace

Aplikace bežící na serveru má za úkol nejen vyvážení zátěže a správu Hadoop clusterů - zprostředkovává připojení nových clusterů, DataNodů (výpočetních uzlů) do existujících clusterů tj. jádro aplikace - ale také má zjednodušit práci uživateli - webové rozhraní umožňuje správu a využití aplikace. Obrázek 7.4 vyobrazuje základní strukturu serverové aplikace.



Obrázek 7.4: Návrh serverové komunikace
Zdroj: vlastní tvorba

Jádro aplikace je základem serverové aplikace a jeho úkolem je obstarávat požadavky uživatelů, notifikace uživatelů o ukončení úloh, spravovat Hadoop clustery, zajišťovat vyvažování zátěže a řešit ošetření chyb vzniklých v celém systému. Bližší popis jádra aplikace rozebíráme v podkapitolách Základní funk-

ce (kap. 7.2.3), Databáze (kap. 7.2.4), Směrovací algoritmy a vyvažování zátěže (kap. 7.2.5) a Chybovost a validování dat (kap. 7.2.6). Podkapitoly poskytují základní návrh toho, co by jádro aplikace mělo umožňovat a jak by mělo pracovat po implementaci v daném jazyce. Kapitola Možné implementace (kap. 7.2.7) navrhuje konkrétní jazyky a frameworky ve kterých doporučujeme zpracovat uváděný návrh.

Pomocí webového rozhraní je možné se přihlásit do aplikace pod dvěma základními typy účtů, a to uživatelským a administrátorským. Prvním typem účtu je uživatelský, který umožňuje uživateli správu nahraných programů a zobrazování statistik jeho prováděných úloh. Než je možné začít tento účet využít je potřeba, aby se uživatel zaregistroval na webovém rozhraní aplikace (je-li tato možnost administrátorem aplikace povolena) nebo mu byl založen účet administrátorem. Po vytvoření účtu je uživateli zaslán informační email s potřebnými údaji. Po přihlášení uživatele do systému mu je zobrazen portál, který má následující strukturu a jsou mu umožněny tyto akce:

- **Přehled** - toto je úvodní obrazovka, kterou uživatel vidí po přihlášení. Na této obrazovce jsou zobrazeny statistiky probíhajících úloh a dokončených úloh včetně jejich úspěšného/neúspěšného dokončení. Uživatel má možnost rozkliknout každou probíhající úlohu, aby získal podrobnější informace o jejím průběhu zpracovávání. U úspěšně dokončených úloh je po rozkliknutí uživateli zobrazena ikona pro stažení výsledných souborů a datum dokončení prováděné úlohy. Zobrazení detailnějších informací u neúspěšně dokončených úlohách obsahuje informaci o chybě.

- **Moje programy** - tato obrazovka obsahuje seznam MapReduce programů, které byli uživatelem nahrány. Také, jak je tomu na úvodní stránce, je uživateli zobrazen pouze stručný seznam obsahující název, verzi programu, zda je program aktivní. Po kliknutí na zvolený program zobrazí detailnější informace. Detailnější informace obsahují popis úlohy, popis struktury vstupního souboru, datum nahrání a informaci o reprezentaci výstupních dat. Nad seznamem uživatelských programů je odkaz, který otevře průvodce pro nahrání nového programu - viz. „Průvodce nahrání programu“.

- **Průvodce nahrání programu** - nahrání nového programu probíhá v několika krocích.

1. Průvodce vyzve uživatele, aby zvolil program MapReduce a vstupní data. Po úspěšném zvolení souborů uživatel může pokračovat pomocí tlačítka „další“.
2. Uživatel doplní název programu, popis funkčnosti programu, popis vstupního souboru a také jak správně interpretovat výsledný soubor. Po vyplnění povinných informací může uživatel pokračovat na další krok skrze tlačítko „další“.
3. Na poslední obrazovce průvodce nahrání programu je uživateli zobrazena rekapitulace vyplněných informací. Pokud uživatel vyhodnotí, že jsou informace v pořádku, může pomocí tlačítka „dokončit“ průvodce ukončit.
4. Po dokončení průvodce uživatelem, jsou nahrané soubory přesunuty z uživatelského PC do úložiště aplikace. Uživateli je zobrazena informace o úspěšném nahrání souborů. Do databáze je vytvořen potřebný záznam o daném programu a tento program je zařazen do fronty čekajících programů na ověření. V tuto chvíli již uživatel v seznamu jeho nahraných programů tento nově nahraný program vidí a u tohoto programu je doplněna informace „čeká na ověření“. Ověření programu probíhá na speciálním Hadoop clusteru (zpravidla postačí pouze DataNode), který je speciálně určen pro tyto účely. Ověřování programu jsme do návrhu přidali, abychom předešli nefunkčnosti programu či pádům clusterů na kterých probíhá zpracování úloh. Pokud spuštění, průběh a vytvoření výsledného souboru proběhne v pořádku, tak je stav programu z „čeká na ověření“ změněn na „ověřen“. Je-li stav programu „ověřen“ má uživatel možnost svůj program publikovat pro ostatní uživatele. Dojde-li k nějaké chybě je programu nastaven stav „chybující“ a do detailních informací je propsána chyba ke které došlo.

V každém kroku (kromě prvního kroku) se uživatel může vrátit do předchozího kroku pomocí tlačítka „zpět“. Samozřejmě může uživatel ukončit průvodce nahrávání programu pomocí tlačítka „ukončit“.

Druhým typem účtu je administrátorský, který je přidělován pouze lidem, kteří spravují, upravují nastavení aplikace a řeší chybové události vzniklé v aplikacích. Administrátor může ve webovém prohlížeči spravovat Hadoop clustery

a také běžící úlohy na těchto clusterech. Dále vidí vytížení všech clusterů.

7.2.3 Základní funkce (Metody)

Tato podkapitola obsahuje seznam základních metod a popis jejich funkčnosti. Vybrané metody jsou doporučené pro implementaci našeho návrhu. Seznam není striktně daný a je možné metody přidávat či modifikovat dle potřeb implementace a tím rozšiřovat funkčnost aplikace.

- Směrování úloh - metoda pro určení clusteru, který bude danou úlohu zpracovávat. Algoritmus pro zpracování je uveden v 7.2.5.
- Navázání komunikace s Hadoop clustery - jakmile je rozhodnuto, na kterém cluster bude úloha zpracovávána, tak tato metoda zajistí navázání spojení a nahrání potřebných dat na cluster.
- Nahrávání a zpracování souborů - soubory obsahující zdrojové kódy a data, které mají být zpracovávány, jsou nahrávány skrze webový prohlížeč do aplikace.
- Logování a zpracování chyb - dojde-li k jakékoli chybě v průběhu zpracování výpočtů, jsou tyto informace zapsány do logu na daném clusteru. Tato metoda slouží pro získání dat o chybě z clusteru a informování uživatele s popisem chyby, ke které došlo.
- Založení clusteru - metoda vytvoří potřebné záznamy do databáze, aby bylo možné daný cluster využívat pro výpočty.
- Odstranění clusteru - metoda zajistí ukončení prováděných operací na daném clusteru. Existují-li úlohy ve stavu čekající, tak jsou tyto úlohy přesunuty na aktivní clustery nebo existuje-li úloha ve stavu zpracovávání, tak je jí nastaven stav chybovaná a po určitém čase je spuštěna na jiném aktivní clusteru. Metoda dále zařídí odstranění (trvalé smazání informací) nebo deaktivaci (dočasné odstranění) clusteru.
- Přidání výpočetního uzlu - přidání uzlu provede potřebné nastavení v databázi a do hlavního uzlu Hadoop clusteru nastaví potřebnou konfiguraci. Výběr Hadoop clusteru může být pevně daný či automatický. V případě automatického je volána metoda "Zvolit Hadoop cluster", která vybere cluster pro připojení

- Odstranění uzlu - provede odstranění uzlu z konfigurace NameNodu a promítne potřebné změny do databáze.
- Zvolit Hadoop cluster - metoda provede výběr vhodného clusteru do kterého se má nový výpočetní uzel připojit. Kritéria pro výběr jsou popsány v kapitole 7.2.5.
- Získání informací o vytížení clusteru - metoda získá informace z Resource Manager API pro daný cluster.

7.2.4 Databáze

Databáze (v textu také označujeme „db“ či „DB“) se využívají pro ukládání dat již více jak čtyři desetiletí a proto pro námi navrženou aplikaci také využijeme tento způsob ukládání dat. Obecně databáze dělíme na dva hlavní směry, a to SQL a NoSQL. Každý z těchto směrů má svoje výhody i nevýhody. V krátkosti čtenáři představíme oba dva směry a především se zaměříme na NoSQL, které využívají dokumentového způsobu ukládání dat, jelikož tento typ databáze využíváme pro uložení dat v aplikaci. Dále v této části nastíníme návrh databáze pro aplikaci a jaká data se budou ukládat.

SQL nebo také relační databáze (RDB) se především vyznačují tím, že udržují data v relacích. Relace jsou často také označovány jako tabulky. Každá tabulka je rozdělena na sloupce a řádky. Sloupce definují strukturu tabulky a řádky jsou data, která byla do tabulky vložena. Soubor tabulek, které mají společné charakteristiky označujeme jako Schéma. Pro získání dat z relačních databázích byl navržen speciální jazyk SQL (Structure Query Language). SQL má mnoho dobrých vlastností, ale my se zaměříme spíše na nevýhody, které v návrhu frameworku jsou pro nás rozhodující pro volbu typu databáze. Jednou z hlavních nevýhod je rozšiřování struktur tabulek a jejich vztahů. Chceme-li do již existujícího schématu ukládat dodatečné informace, tak je nutné vytvořit další tabulky, které řeší vztahy mezi řádky v tabulkách nebo rozšířit strukturu tabulek. Tím samozřejmě narůstá komplexnost databáze. Další nevýhodou RDB je rychlost získávání dat z databáze a to především pokud jsou tabulky normalizovány. Normalizace dat je využívána především proto, abychom při ukládání neduplikovali data, či tabulky nenabývaly velkého počtu sloupců, a to nás nutí vytvářet další tabulky, rozšiřovat již existující tabulky a vytvářet složitější dotazy do DB, čímž narůstá čas potřebný pro získání dotazovaných dat.

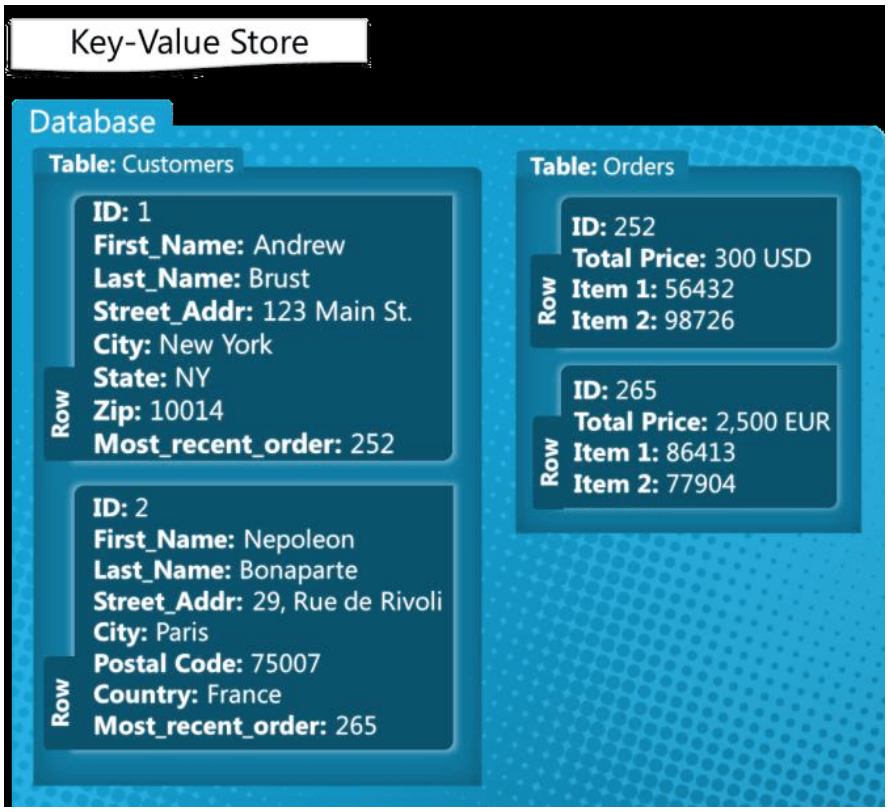
Tabulka 7.1 obsahuje porovnání základních funkčností SQL a NoSQL databází.

Vlastnost/DB	SQL	NoSQL (dokumentové)
CRUD operace	Ano	Ano
Transakční	Ano	závisí na db
Uchování dat	databáze relace (tabulka) řádek sloupec	databáze kolekce dokument pole
Způsob dotazování	SQL jazyk	závisí na db
Škálovatelnost	ne/obtížně	ano/jednoduše
Komplexnost	musí vznikat nové tabulky pro nehodící se data	data jsou uložena podle potřeby

Tabulka 7.1: Porovnání SQL a NoSQL databázových systémů

V aplikaci nám jde především o škálovatelnost, rychlost získávání/ukládání dat, práci s velkým objemem dat a jejich různorodost s obdobnými vlastnostmi. Právě z těchto důvodů jsme se rozhodli využít NoSQL databáze. NoSQL databáze mají mnohem volnější strukturu než relační databáze, což právě přináší lepší škálovatelnost a výkonnostní podmínky pro databázi. Obecně jsou uznávané čtyři druhy NoSQL databází, které se dělí podle toho, jak ukládají data.

- **Klíč-hodnota** (Key-value stores) je nejjednodušším typem. Obrázek 7.5 znázorňuje uložení dat v databázi. Data jsou v kolekcích (tabulce viz. obrázek) pomocí dvojice klíč-hodnota. V tomto typu nevznikají duplicity a díky přístupu podle klíče přes hash tabulku je tento typ úložiště velmi rychlý. Databáze hodnoty pouze uchovává a to v typu BLOB a zpracování obsahu hodnot je pouze na aplikaci. S tímto typem ukládáním se můžeme setkat například u databází Dynamo, Oracle NoSQL, Redis, Riak či Azure Table Storage.

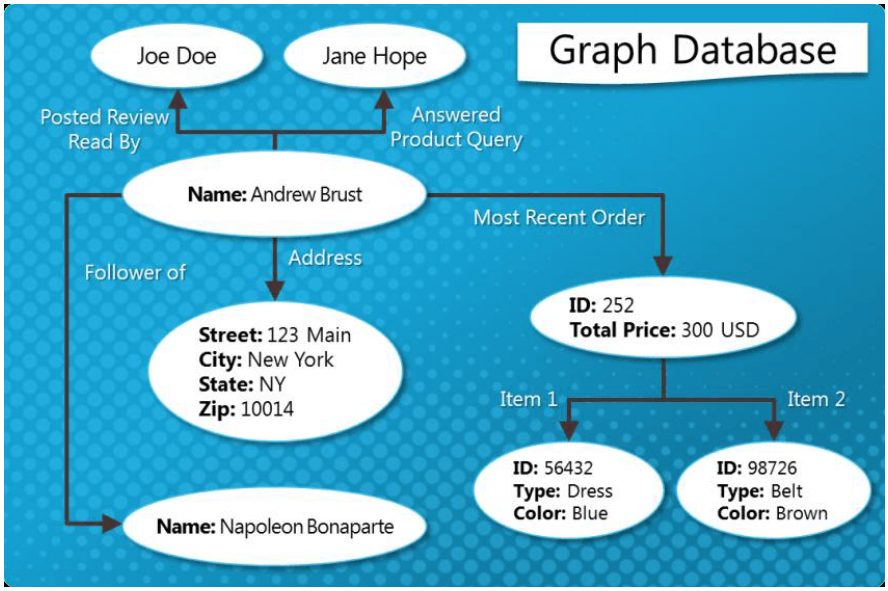


Obrázek 7.5: Uložení dat v NoSQL pomocí klíč-hodnota
Zdroj: [25]

Na obrázku 7.5 jsou dvě tabulky, které jsou uloženy v databázi. Tabulka „Customers“ obsahuje dva záznamy, které můžeme označit jako řádky (z angl. „row“). Ke každému řádku přistupujeme pomocí unikátní hodnoty, která je v našem případě „ID“. V našem případě dotážeme-li se na klíč „ID = 1“, tak jsou nám vráceny hodnoty pro řádek „ID: 1“. Bohužel při vkládání dat do databáze klíč-hodnota je potřeba data předem připravit pro vložení. Dále je složitější dotazovaní pro získání potřebných dat. Námi zvolené technologie především komunikují pomocí výměny JSON objektů a proto tento typ úložiště nevyužijeme i když má velký potenciál a to především díky rychlosti získávání dat.

- **Grafové úložiště** (Graph stores) především vynikají v práci s propojenými daty. Grafové databáze obsahují hrany a uzly. Hrany propojují uzly a tím vyjadřují jejich vztah. Jak hrany, tak uzly obsahují informace uložené jako klíč-hodnota, což je vidět i na obrázku 7.6. Obecně u grafových

úložišť je omezená škálovatelnost a to především tím, že je zapotřebí mít všechna data na jednom stroji. Databáze využívající tento typ úložišť jsou například Allegro, InfiniteGraph či Noe4J. Jak zmiňujeme výše, tak gra-

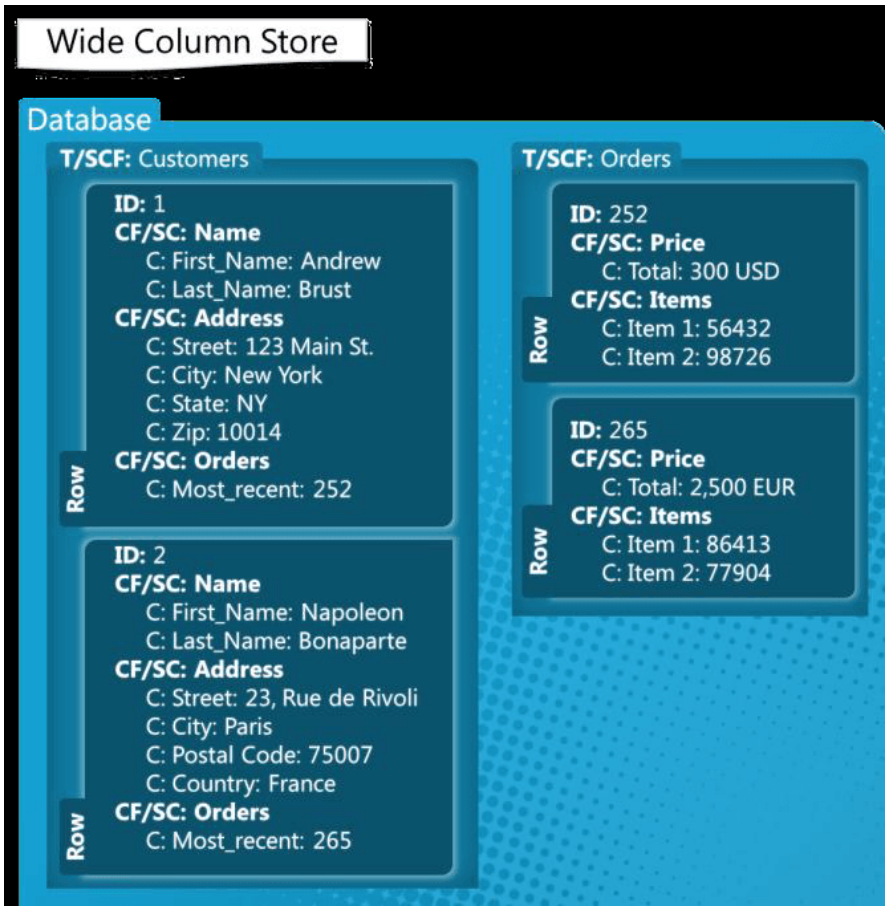


Obrázek 7.6: Uložení dat v NoSQL pomocí grafu
Zdroj: [25]

fové úložiště není vhodné pro dynamické změny, je omezena škálovatelnost a data je nutné ukládat na jednom stroji pro lepší výkonnost. Právě reakce na změny a škálovatelnost je pro nás v návrhu aplikace velmi důležitá, a proto grafové úložiště není vhodné pro naší aplikaci.

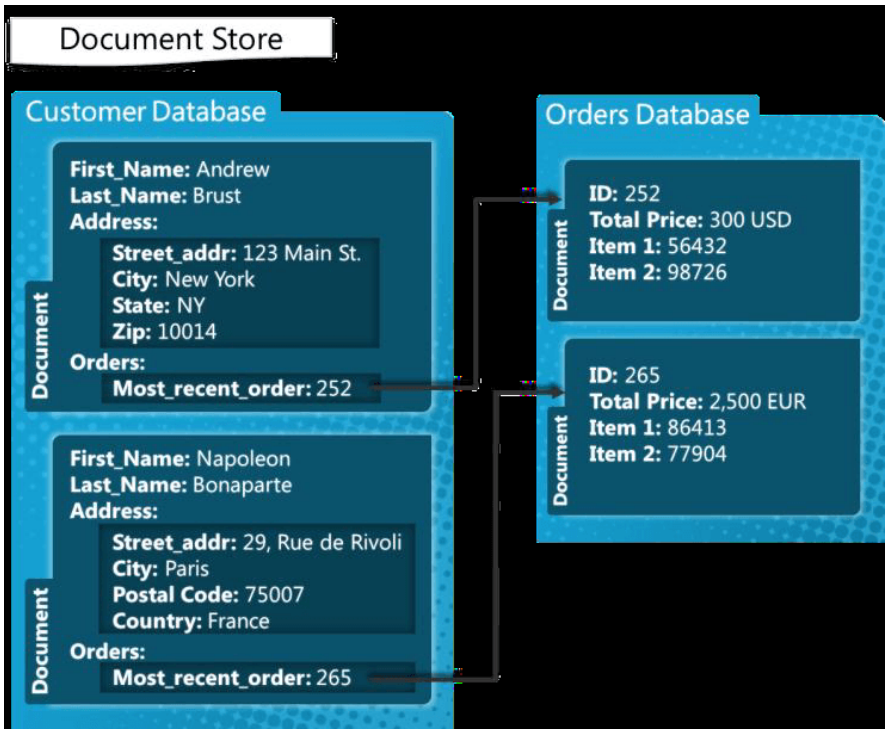
- **Sloupcové úložiště** (Column stores) má řádky velmi podobné jako u relačních databázích. U řádku je kolekce klíč-hodnota, kde klíč je název sloupce. Hlavním rozdílem mezi relačním a sloupcovou db je to, že ve sloupcové db mohou mít řádky odlišné klíče (sloupce). Obrázek 7.7 popisuje uložení dat ve sloupcové databázi. Jak je vidět z výše uvedeného obrázku, tak každý řádek v tabulce „Customers“ obsahuje jedinečný identifikátor „ID“. Další atributy „Name“, „Address“ a „Orders“ můžeme označit „skupina sloupců“ (z angl. Column Family), které shlukují sloupce podobného charakteru. V našem ukázkovém příkladu skupina sloupců označená „Name“ obsahuje sloupce „First_Name“ a „Last_Name“, které obsahují data vložená uživatelem. Příkladem tohoto typu databází jsou

třeba Accumulo, Cassandra, Druid, Big Table či HBase. Databáze HBase je produktem Apache a je využívána v Apache Hodoop.



Obrázek 7.7: Uložení dat v NoSQL pomocí sloupců
Zdroj: [25]

- **Dokumentové úložiště** (Document stores) jsou velmi podobné jako úložiště key-value, avšak hodnota je zpravidla XML/JSON nebo objekt. Dále umožňuje reference na jiné záznamy, vnořování struktur a kolekce. Dotazování do databáze zvládá i složitější než jen přes hodnotu. Ukázka uložení dat pomocí dokumentů je znázorněn na obrázku 7.8 Příklady databází jsou Apache CouchDB, Coachbase, OrientDB, Azure documentDB či MongoDB.



Obrázek 7.8: Uložení dat v NoSQL pomocí sloupců
Zdroj: [25]

Tabulka 7.2 obsahuje porovnání klíčových vlastností NoSQL databázových úložišť. Jak je z tabulky vidět, tak Dokumentový typ úložiště je velmi dobře hodnocen.

Typ úložiště	Klíč-hodnota	Sloupcové	Grafové	Dokumentové
Výkon	vysoký	vysoký	variabilní	vysoký
Škálovatelnost	vysoká	vysoká	variabilní	variabilní (vysoká)
Flexibilita	vysoká	průměrná	vysoká	vysoká
Komplexnost	žádná	nízká	vysoká	nízká
Funkcionalita	variabilní (žádná)	minimální	teorie grafů	variabilní (nízká)

Tabulka 7.2: Porovnání klíčových atributů NoSQL úložišť

Námi navržená aplikace bude do databáze ukládat data pro svoje vnitřní řízení a jelikož bude pracovat s velkým množstvím dat, tak jsme se rozhodli vy-

užít NoSQL databázi s dokumentovým typem úložiště. Nejen již zmiňovanými výhodami NoSQL je naše rozhodnutí podpořeno tím, že komunikace v rámci aplikace a Hadoop clusterů probíhá výměnou XML nebo JSON objektů, které tak budou moci být ukládány do databáze bez dalších větších transformací.

Z výše uvedených příkladů dokumentového typu NoSQL databáze jsme se rozhodli využívat open source databázi MongoDB. Dalším důvodem, proč jsme se rozhodli využívat MongoDB ve spolupráci s Hadoop je to, že již mnoho velkých firem tuto kombinaci využívá. Firmy, které využívají Hadoop pro analýzu dat a MongoDB pro ukládání jsou například Ebay, Orbitz či Pearson. První verze MongoDB byla uveřejněna již v roce 2009 společností 10gen. Díky velkému zájmu o MongoDB se společnost 10gen v roce 2013 přejmenovala na MongoDB,Inc. MongoDB je v dnešní době velmi populární při volbě NoSQL databáze.

Návrh několika základních kolekcí, které budou využívány pro ukládání dat v aplikaci:

- **Users** - základní informace o uživatelích, kteří se do systému přihlašují.
- **Clusters** - v této kolekci jsou uloženy potřebné informace o clusterech jako je adresa clusteru, celkový počet uzlů, počet aktivních uzlů či informace o uzlech pro daný cluster, informace jsou získávány z resourceManager REST API, které je dostupné z [26].
- **ClustersScheduler** - informace o probíhajících úlohách na clusteru (odkazuje se do kolekce Cluster), využívané pro směrovací algoritmus, tyto informace jsou v pravidelných intervalech a při typických událostech (dokončení výpočtu na clusteru) aktualizovány.
- **Programs** - obsahuje informace o programech, které byly nahrány. Obsahuje informaci, zda kdo program nahrál, popis programu, název programu, zda je program privátní či veřejný, počet jeho spuštění, průměrný čas potřebný na zpracování 1 MB programem pro každý cluster.

7.2.5 Směrovací algoritmy a vyvažování zátěže

Hlavním záměrem naší aplikace je především práce s větším počtem clusterů a je tedy potřeba zajistit správné směrování úloh na volné Hadoop clustery,

na kterých budou prováděny samotné výpočty. Směrování úloh na clustery je zajištěno podle následujících pravidel:

- Informace o počtu volných uzlů Hadoop clusterů a jejich volné paměti, tyto informace jsou získávány z Resource Manager API, pokud neexistuje žádný volný uzel nebo je vyčerpána všechna dostupná paměť clusteru, tak je daný cluster vyřazen z provádění dalších výpočtů.
- Základní výpočet pro vhodný Hadoop Cluster na kterém má být úloha spuštěna je doba potřebná na zpracování objemu dat, které jsou nahrané uživatelem a informací získaných opakovaným spouštěním daných úloh na clusteru:

$$V \cdot (\frac{1}{v_{avg_n}} + t_{avg_n}) \tag{7.1}$$

  - V - Objem zpracovávaných dat v MB.
  - n - Zvolený Hadoop Cluster.
  - v_{avg_n} - Celková průměrná rychlost (MB/s) přenosu dat mezi serverovou aplikací a Hadoop clusterem (n) ve vzdálenosti l je vypočítána:

$$v_{avg_n} = \frac{v_1 + v_2 + ... + v_l - 1 + v_l}{l} \tag{7.2}$$

  - l - Počet propojení mezi aplikací a Hadoop Clusterem.
  - v_l - Průměrná rychlost přenosu mezi uzly *l-1* a l, rychlost je vypočítána při přihlášení uzlu do aplikace v MB/s.
  - t_{avg_n} - Průměrný čas potřebný na zpracování 1 MB na zvoleném clusteru pro zvolenou úlohu:

$$t_{avg_n} = \frac{\frac{t_1+t_2+...+t_{j-1}+t_j}{j}}{V_1 + V_2 + ... + V_{j-1} + V_j} \tag{7.3}$$

    Pokud nebyl zatím daný program na clusteru spouštěn, tak t_{avg_n} je nastaveno na průměrnou hodnotu provádění na ostatních clusterech, nebyl-li program zatím spuštěn na žádném Hadoop clusteru, tak t_{avg_n} je nastaveno na hodnotu, která byla naměřena při ověřování funkčnosti po nahrání programu uživatelem. Toto nastavení je z důvodu abychom neeliminovali či nezvýhodňovali clustery na kterých zatím daný program nebyl spuštěn, jelikož by byl průměrný čas 0, a to by značně ovlivnilo pořadí clusteru v seznamu.

- t_j - Čas potřebný pro provedení úlohy na clusteru. Tento čas je získán rozdílem ukončení a začátku provádění úlohy z API Hadoop Clusteru pro úlohy, které byly úspěšně ukončeny (status nastaven na „SUCCEEDED“).
- j - Počet spuštěných úloh v minulosti.

Výsledkem výše uvedeného výpočtu je seznam možných clusterů, na kterých může výpočet proběhnout. Seznam je seřazen od nejnižších hodnot po nejvyšší. Z tohoto seznamu je vybrán první cluster v pořadí a pro něj jsou spuštěny potřebné operace pro přenos dat a spuštění úloh.

Pro zjištění nejkratších vzdáleností mezi clustery a serverem aplikace či mezi uzly v clusteru využijeme jeden z běžných algoritmů pro výpočet minimální vzdáleností, jako jsou Dijkstra či Flojd Warshall. Potřebné informace o clusteru a uzlech jsou zjištěny pomocí Resource Manager Api, které poskytuje potřebné funkce a je dostupné z [26]. Zjištěné informace jsou uloženy do databáze a dále využívány pro potřeby aplikace.

Jak jsme zmiňovali v kapitole 7.2.1, tak jedno z možných připojení datových uzlů do clusterů může probíhat automaticky. Pro tento typ přidávání datových uzlů do clusteru jsou zohledněny následující kritéria:

- Počet uzlů v clusteru, kde se porovná hodnota s průměrným počtem uzlů a pokud cluster obsahuje podprůměrný počet uzlů, tak je vložen do seznamu kandidátů na přidání uzlu
- a vzdálenost/datová prostupnost mezi datovým uzlem a clusterem, do kterého bude daný uzel připojen.

Po vyhodnocení kriterií je sestaven seznam možných clusterů, do kterých je možné DataNode připojit a vybere se první z tohoto seznamu.

7.2.6 Chybovost a validování dat

V této podkapitole se věnujeme chybovosti a validování dat. Validování dat dělíme na dvě skupiny: data pro zpracování zvoleným programem a vstupy uživatelské. Validování/očištění dat vzniklé při zpracovávaní zvoleným programem necháváme čistě na programátorovi daného programu. Pro takovéto potřeby má aplikace metodu, která zobrazí potřebné informace o vzniklé validační

chybě uživateli a zároveň vytvoří potřebný záznam o chybě do textového souboru, který je vyhrazen pro tento účel. Validování vstupů, které zadává uživatel do webového prohlížeče, je řešeno dvěma způsoby a to validacemi na klientovi a validacemi na serveru. Validování na klientovi je pro povinná pole, správné typy hodnot ve formulářích tj. validování pole formuláře pro číslo, správný formát data, emailu, apod. Druhotné validování probíhá na serveru, kde jsou využity náročnější způsoby validování. Pokud nastane problém při validování, je toto ohlášeno uživateli pomocí hlášky na webovém prohlížeči.

Chyby dělíme na chyby při přenosu dat a chyby při zpracování v rámci Hadoop clusterů. Chyby při přenosu dat mohou nastat při přenosu dat od uživatele na servery aplikace a také při přenosu potřebných souborů z aplikace na dané clustery, kde bude docházet k jejich zpracování. Jelikož pro uložení dat využíváme již existující protokoly, které jsou http/https nebo ftp/ftps, tak využíváme základní chybové kódy navrácené těmito protokoly. Aplikace reaguje na chyby oznámením uživateli a zapsáním potřebných informací do speciálního logu pro chyby. Řešení druhého typu chyb vzniklých při zpracování úloh na Hadoop clusterech je ponecháno na Hadoop, který tyto problémy řeší v rámci svého návrhu. Jak je uvedeno v [17], tak Hadoop dělí chyby do několika kategorií:

- **Chyby úloh** - nejčastější příčinou vzniku těchto chyb je vyhození chyby za běhu programu (z angl. „runtime exception“). Pokud k tomuto problému dojde, tak jej JVM (Java Virtual Machine) reportuje hlavní aplikaci dané úlohy a poté je úloha ukončena. Při vzniky chyby je tato informace zapsána do uživatelského logu na Hadoop clusteru. Hlavní aplikace označí pokus zpracování úlohy za neúspěšnou (z angl. „failed“) uvolní kontejner, aby jej mohl využít manažer zdrojů pro další úlohy. Dalším typem chyb je neočekávaný pád JVM například při chybě v JVM, která byla zapříčiněna kódem programátora. V tomto případě manažer uzlů tento problém začne řešit tím, že informuje hlavní aplikaci, aby úlohu označila jako chybovanou (z angl. „failed“). Zaseknuté, zacyklené či visící úlohy (z angl. „hanging tasks“) jsou odhaleny tím, že hlavní aplikace neobdrží informaci o progresi úlohy po delší dobu a tím je úloha označena jako chybová a je ukončena. Základní nastavení pro změnu stavu úlohy na chybovanou je 10 minut. Tato doba je nastavitelná. V případě, že na clusteru je nastavena hodnota vypršení časového limitu na nula, tak zaseknuté úlohy

nejsou nikdy ukončeny převodem do stavu chybovaná. Toto způsobuje, že nejsou uvolněny kontejnery pro zpracování programů a tím pádem může dojít až k ukončení celého clusteru. Jakmile hlavní aplikace zjistí, že daná úloha skončila chybně, tak je provádění této úlohy přeplánováno (z angl. „reschedule“). Hlavní aplikace se pokusí naplánovat provádění úlohy na odlišný uzel, kde úloha neskončila chybou. Počet pokusů pro přeplánování úloh je v základním nastavení clusteru nastaven na čtyři pokusy. Tuto hodnotu je možné nastavit na clusteru. Pokud čtyřikrát dojde k chybovému ukončení úlohy, tak je celý program označen jako chybný.

- **Chyby hlavní aplikace** (Application Master Failure) - obdobně, tak jako u úloh prováděné MapReduce procedury je hlavní aplikaci poskytnuto více pokusů na úspěšné provedení programu. V základním nastavení je maximální počet pokusů nastaven na dva, ale tuto hodnotu je možné změnit. Pokud tedy v základním nastavení Hadoop clusteru dojde dvakrát k chybovému ukončení hlavní aplikace, tak celý program skončí chybou. O této skutečnosti je informován uživatel například na základě dotazování na ukončení daného programu. Způsob obnovení probíhá následovně. Hlavní aplikace zasílá periodický oznamovací signál (z angl. „heartbeat“) manažeru zdrojů. Jakmile dojde k chybě na hlavním aplikaci, tak manažer zdrojů detekuje tuto skutečnost a zahájí novou instanci hlavní aplikace v novém kontejneru, který je spravován manažerem uzlů. Dále podle historie programu následuje nastartování úloh, které ještě nebyly dokončeny a jsou přeskočeny již úlohy, které skončili chybou (z angl. „failed“). Jelikož se klient dotazuje pravidelně na průběh provádění programu Hadoop zajistí přesměrování dotazů na novou instanci.

- **Chyby manažera uzlů** (Node Manager Failure) - vznikne-li chyba způsobená pádem manažera uzlů nebo manažer uzlů běží velmi pomalu, tak přestane zasílat pravidelný oznamovací signál (z angl. „hearbeat“) manažerovi zdrojů. Manažer zdrojů zpozoruje, že mu manažer uzlů nezasílá pravidelný signál, pokud jej ani jednou za 10 minut nepřijme. Hodnota 10 minut je základním nastavením a je možné tuto hodnotu nastavit. Pokud manažer zdrojů zjistí, že nedostává pravidelný signál, tak odstraní program spuštěný na manažerovi uzlů z naplánovaného seznamu provádění úloh. Jakákoli úloha či hlavní aplikace běžící na chybovém manažeru uzlů bude obnoveno (z angl. „recovered“) pomocí mechanizmů pro zotavení

uvedené v sekcích chyby úloh a chyby hlavní aplikace.

- **Chyby manažera zdrojů** (Resource Manager Failure) - chybovost manažera zdrojů je velmi závažná, jelikož bez manažera zdrojů nejsou spuštěny ani programy ani přidělování kontejnerů pro úlohy. V základním nastavení je manažer zdrojů bodem, který může způsobit výpadek systému (z angl. „single point of failure“) či dokonce problém nefunkčnosti celého clusteru jelikož všechny programy končí chybou a nemohou být obnoveny. Pro dosažení vysoké dostupnosti (z angl. „high availability“) je proto důležité mít dvojici manažerů zdrojů, která využívá jednoho aktivního manažera a jednoho v pohotovostním stavu. Pokud dojde k selhání primárního manažera zdrojů, tak dojde k aktivaci sekundárního manažera, který je v pohotovostním stavu. Tímto přepnutím se o selhání primárního manažera zdrojů zpravidla klient ani nedozví. Díky nastavení vysoké dostupnosti jsou ukládány informace ohledně všech běžících aplikacích do úložiště jako je HDFS. Manažer, který přešel z pohotovostního do aktivního stavu, je schopen díky uloženým informací obnovit správnou funkčnost selhaného primárního manažera zdrojů. Dále je nutné zmínit, že pokud je využita dvojice manažerů zdrojů je nutné správně nakonfigurovat klienta a také manažera uzlů.

- **Chyby HDFS** (HDFS Failure) - řešení tohoto problému jsme již nastínili v kapitole 4.2.2. Replikace jsou jednou z možností, jak se vyhnout ztrátě dat, ale NameNode je stále nejslabším článkem, jelikož jsou potřebné konfigurace ukládány lokálně a když tento uzel selže, dojde k nemožnosti provádět jakékoli úlohy či komunikovat s clusterem nebo uživatelem. Pro zotavení z pádu NameNodu je nutné aby systémový administrátor nastartoval nový primární NameNode a nastavil potřebnou konfiguraci. Z těchto důvodů se využívá záložního uzlu nazvaný SecondaryNameNode.

7.2.7 Možné implementace

Tato podkapitola shrnuje základní požadavky na funkčnost aplikace. Na základě těchto požadavků doporučíme, jaký zvolit programovací jazyk. Podkapitola je rozdělena do třech základních částí Jádro aplikace, Webové rozhraní a Klientské aplikace.

Jádro aplikace

Ve výše uvedených kapitolách jsme čtenáře seznámili se základním návrhem a funkčnostmi jádra aplikace. Jádro aplikace je základem celého systému a je tedy nutné zvolit programovací jazyk, který zvládne dané operace. Dnes skoro všechny jazyky nebo frameworky na nich založené podporují matematické výpočty, práci s textem a připojení do databáze. Jádro aplikace také připravuje data pro webové rozhraní a tak doporučujeme, aby zvolený jazyk podporoval REST API například pomocí nějakého framework pro daný jazyk.

REST je zkratka pro „REpresentational State Transfer“. REST API je technologie, které zpravidla pro komunikaci využívá HTTP. V oblasti vývoje, kde se autoři pohybují je toto API velmi oceňováno především pro jeho jednoduchost. Komunikace skrze HTTP probíhá za pomoci výměny např. JSON, XML objektů či prostého textu (z angl. „plain text“). REST API je využíváno i v samotném Apache Hadoop, ke kterému přistupujeme pro získávání informací z clusterů.

Pro implementaci můžeme využít mnoho REST frameworků splňující potřebná kritéria. Tyto frameworky rozšiřují zvolený jazyk pro vývoj o potřebné funkčnosti. V dnešní době se nejčastěji používají pro implementaci jazyky Java či .Net. Pro jazyk Java existují například tyto frameworky s REST API - Jersey, Spring, Spark a další. Pro .Net můžeme využít například framework ASP.Net.

Ve výše uvedených frameworcích i v dalších je možné napsat jádro aplikace. My pro implementaci volíme Spring framework, který byl navržen pro vývoj firemních aplikací v jazyce Java. Spring umožňuje vývojářům vytvářet velmi výkonná, jednoduše testovatelná a znovupoužitelná řešení pro Java aplikace. Spring převzal mnoho návrhových vzorů a postupů, které se během let staly nejlepší pro praxi. Spring framework v základním balíku poskytuje například:

- Dependency Injection - technika pro vkládání závislostí mezi komponentami tak, aby neměly reference jedna na druhou v době sestavování.
- Aspektově Orientované Programování (AOP) - má za cíl zvýšit modularitu programů.
- Spring MVC webové aplikace a RESTful web - používá se pro zjednodušení práce s webovými komponentami a jejich komunikaci.
- Základní podpora pro různé druhy konektorů, jako jsou JDBC (Java Database Connectivity) a JPA (Java Persistence API) - obě pro práci s data-

bázovými objekty, JMS (Java Message Service) - používá se pro zasílání zpráv.

Spring využívá modulů, které se dají postupně přidávat pro rozšíření funkčností Springu. Například můžeme uvést moduly pro bezpečnost - vhodný pro nastavování uživatelských účtů a omezování práv, konektory pro data - rozšiřují základní sadu konektorů a zjednodušují práci s datovými objekty. Z výše uvedených vlastností a díky autorovým zkušenostem doporučujeme zvolit Spring framework pro implementaci jádra serverové aplikace.

Doporučujeme také před začátkem implementace hlouběji prozkoumat možnosti jazyku Scala a jeho frameworků, které jej rozšiřují o další funkčnosti, jako je například REST API či konektory do databáze. Příkladem můžeme uvést knihovny Sprey, Unfiltered či Spring framework, který je tvořen jako komunitní projekt. Autoři jazyka Scala přišli s propojením objektového a funkcionálního přístupu. Scala běží na JVM (Java Virtual Machine) a může být jednoduše kombinována s jazykem Java. Scala poskytuje lepší čitelnost a jednoduchost kódu vůči Javě.

Webové rozhraní

Základní požadavky na webové rozhraní jsme definovali v úvodu podkapitoly 7.2.2. Webové rozhraní slouží především pro komunikaci mezi uživatelem a serverovou aplikací a pro zobrazování statistik. Právě pro komunikaci, zobrazování statistik a grafů je vhodné využít jazyk, kterému jádro aplikace poskytne potřebná data pro zobrazení a jazyk pro webové rozhraní daná data zobrazí.

Pro implementaci webového rozhraní můžeme využít například framework Angular 2 či knihovnu React. React i Angular 2 jsou postavené nad jazykem JavaScript a slouží především pro zobrazování dat uživatelům ve webovém prohlížeči. Obě technologie přinášejí robustní množinu nástrojů pro škálovatelné a reaktivní webové aplikace. Stačí tedy, aby jádro aplikace připravilo potřebná data, která mají být zobrazována a React či Angular 2 se postarají o jejich zobrazení na klientské stanici.

Pro naši implementaci doporučujeme zvolit knihovnu React a to z důvodu, že Angular 2 je v tuto chvíli pouze v beta verzích a není stoprocentně jasné jak to bude s přenositelností komponent ze starších verzí do nové.

Klientská aplikace

Návrh klientské aplikace jsme rozebírali v 7.2.1, kde jsme definovali základní funkčnosti. Tyto funkčnosti vymezují výběr jazyka pro implementaci. Je tedy nutné, aby zvolený jazyk podporoval:

- Multiplatformnost - Apache Hadoop je možné instalovat na různé operační systémy a je tedy potřebné, aby i klientská aplikace dokázala pracovat na různých operačních systémech. Zatím Hadoop podporuje Windows a operační systémy založených na Linuxovém jádře.
- Grafické Uživatelské Rozhraní (z angl. „Graphical User Interface“ GUI) - grafické rozhraní je dnes základem pro komunikaci mezi běžným uživatelem a aplikací.

Pro implementaci klientské aplikace můžeme využít libovolný jazyk, který splňuje výše uvedené požadavky. Například lze využít již zmiňované jazyky nebo jejich frameworky Java, .Net, Scala. Samozřejmě je možné zvolit i další jazyky, které budou splňovat výše uvedené požadavky a umět komunikovat s Hadoop a serverovou aplikací. Z důvodu využití jazyka Java pro serverovou aplikaci a také autorových zkušenostech s tímto jazykem, tak Javu doporučujeme pro implementaci klientské aplikace.

7.3 Shrnutí

Již několikrát jsme zmiňovali, že je Apache Hadoop velmi oblíbený a používaný v mnoha velkých institucích pro práci s velkoobjemovými daty a to především pro jeho dobrou funkčnost a rychlost zpracování dat. Jak jsme uvedli v podkapitole 7.1, tak je ve firmách nejčastěji využíván klasický přístup uživatele k Hadoop clusteru. Tohoto přístupu využívají i další společnosti, jako je například Claudera, která poskytuje upravenou distribuci Hadoop jako firemní řešení.

Hlavními novými přístupy či odlišnosti našeho návrhu od klasického je hned několik. Zaprvé využíváme shlukování více Hadoop Clusterů, které mají za úkol dosáhnout většího výkonu. Zadruhé stavíme náš návrh na využívání strojů, které jsou již v daných institucích a tím není potřeba investovat velký kapitál pro vystavení clusterů.

Mnoho vědeckých institucí využívá speciálních data center, která poskytují zpracování velkých dat. Tato data centra využívají profesionální hardware jehož

pořizovací cena je velmi vysoká a to se poté samozřejmě odráží na ceně využívání těchto služeb. Dále zjednodušujeme práci uživatelům, aby pro ně byla aplikace přínosná, jak v efektivitě zpracovávání dat, tak i z hlediska prezentování výsledků. Pokud lidé nevyužívají nějaké modifikace Hadoop, tak prezentace výsledných dat je bude stát opět čas, aby tyto data dále zpracovávali. Jelikož aplikace obsahuje větší výpočetní výkon shlukováním Hadoop clusterů, tak může být využívána širší komunitou lidí, kteří si mohou sdílet navzájem jejich MapReduce programy a tudíž není potřeba vymýšlet stejný program několikrát. Jednou z posledních odlišností je postavení návrhu a aplikace na myšlenkách Open Source. Otevření kódu komunitě přináší možnosti zlepšení algoritmizace či návrhu aplikace.

Než ukončíme tuto kapitolu, tak bychom zde chtěli navrhnout možné rozšíření návrhu, které se nám nevešlo do rozsahu práce. Zaměřit se na návrh možnosti sestavování MapReduce programů uživatelem přímo z webového rozhraní. Jednalo by se především o přípravu šablon, podle kterých by byl vygenerován kód MapReduce programu a šablon pro interpretaci výsledných dat. Uživatel tedy nemusí rozumět jazyku kódu, ve kterém je daný program implementován, ale pouze specifikuje postup provádění a chování programu. Tato funkčnost by velmi urychlila implementaci programů a zjednodušila práci uživatelů. Navíc bychom měli možnost vytvořit šablony nezávislé na interpretovaném jazyku, který je podporován Hadoop.

8 Závěr

Cílem práce Analýza metod a matematických přístupů pro distribuované výpočty bylo seznámit uživatele se zpracováním velkoobjemových dat za pomoci distribuovaných výpočtů, návrh nového řešení zpracovávání tohoto typy dat a navržení aplikace, které umožní tato data zpracovávat.

Čtenáře jsme o zpracování velkoobjemových dat seznámili postupně v kapitolách 2, 3 a v 4, kde jsme představili jednu z možných technologií vhodnou pro zpracování Big data. Shrnutí informací z teoretické části a naše důvody, proč jsme zvolili tuto strukturu jsme popsali v kapitole 6.

Druhým hlavním cílem této práce bylo prozkoumat možnosti zpracovávání velkoobjemových dat a navrhnout nový přístup zpracování těchto dat. Náš návrh nového přístupu, kombinace využití myšlenky dobrovolnického počítání a technologie Apache Hadoop, popisujeme v kapitole 7. Ve zmiňované části dále představujeme návrh aplikace a jejich funkčností. V podkapitole 7.3 jsme shrnuli náš návrh, aplikaci a také jsme doporučili jedno z možných rozšíření, které se nám již nevešlo do rozsahu práce. Tato dodatečná funkcionalita by měla ještě více zjednodušit práci uživatelům při zpracovávání velkého objemu dat a zefektivnit navrhování jejich programů pro práci s daty.

Přínos práce spatřujeme především díky novému přístupu, který využívá dnes trochu přehlíženého výkonu běžného hardwaru namísto využívání drahých výpočetních center. Nesmíme samozřejmě přehlédnout přínos pro uživatele využívající tuto aplikaci, které má webové rozhraní navržené na to, aby zajistila a zjednodušila komunikaci mezi uživatelem a Hadoop clustery a také zlepšila prezentaci výsledků jejich práce. Jedním z posledních přínosů této práce vidíme navržení aplikace v myšlenkách Open Source. Právě otevření kódu větší komunitě umožňuje zlepšení výkonu aplikace či dokonce celého systému.

Literatura

[1] M. H., Patel R. S., Brammer L., Smolinski M. S., Brilliant L. Ginsberg, J., Mohebbi. Detecting influenza epidemics using search engine query data. Nature, 457(7232):1012–1014, 02 2009.

[2] Gartner says solving big data challenge involves more than just managing volumes of data. [Online]. Dostupné z WWW: `http://www.gartner.com/newsroom/id/1731916`.

[3] Chris Anderson. The End of Theory: The Data Deluge Makes the Scientific Method Obsolete. Wired, 16(07), June 2008.

[4] C. Duhigg. The power of habit: why we do what we do in life and business. Random House, 2012.

[5] M. Kshemkalyani, A. D., Singhal. Distributed computing: principles, algorithms, and systems. Cambridge University Press, 2008.

[6] N., Machado F., Lopes J. Neves, R., Mestre. Parallel and distributed computing boinc grid implementation. [Online]. Dostupné z WWW: `http://www.deei.fct.ualg.pt/%7Ea27981/spd/boinc-grid.pdf`.

[7] Barkley: Distributed and Parallel Computing. [Online]. Dostupné z WWW: `http://wla.berkeley.edu/~cs61a/fa11/lectures/communication.html`.

[8] L., Beng Chin O. Quang Hieu, V., Mihai. Peer-to-peer computing: principles and applications. Springer, 2010.

[9] J., Kindberg T., Blair G. Coulouris, G., Dollimore. Distributed Systems: Concepts and Design. Addison-Wesley Publishing Company, USA, 5th edition, 2011.

[10] C. Foster, I., Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Advanced computing. Computer systems design. Morgan Kaufmann Publishers, 1999.

[11] R. Plaszczak, P., Wellner. Grid Computing: The Savvy Manager's Guide. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[12] IBM Solutions Grid for Business Partners: Helping IBM Business Partners to Grid-enable applications for next phase of e-business on demand, 2002.

[13] BOINC. [Online]. Dostupné z WWW: `http://boinc.berkeley.edu/trac/wiki`.

[14] Welcome to apache™ hadoop®! [Online]. Dostupné z WWW: `https://hadoop.apache.org/`.

[15] Google research publication: The google file system. [Online]. Dostupné z WWW: `http://research.google.com/archive/gfs.html`.

[16] Poweredby. [Online]. Dostupné z WWW: `http://wiki.apache.org/hadoop/poweredby`.

[17] T. White. Hadoop: the definitive guide. O'Reilly Media, fourth edition, 2015.

[18] Scaling hadoop to 4000 nodes at yahoo! [Online]. Dostupné z WWW: `https://developer.yahoo.com/blogs/hadoop/scaling-hadoop-4000-nodes-yahoo-410.html`.

[19] Data blocks in the hadoop distributed file system (hdfs). [Online]. Dostupné z WWW: `http://www.dummies.com/how-to/content/data-blocks-in-the-hadoop-distributed-file-system-.html`.

[20] Hdfs architecture. [Online]. Dostupné z WWW: `http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfsdesign.html`.

[21] Apache hadoop hdfs. [Online]. Dostupné z WWW: `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/filesystemshell.html`.

[22] The open source definition. [Online]. Dostupné z WWW: https://opensource.org/docs/osd.

[23] Cloudera. [Online]. Dostupné z WWW: http://www.cloudera.com/.

[24] Hadoop wiki - faq. [Online]. Dostupné z WWW: http://wiki.apache.org/hadoop/faq.

[25] Types of nosql databases. [Online]. Dostupné z WWW: http://www.jamesserra.com/archive/2015/04/types-of-nosql-databases/, 2015.

[26] Hadoop yarn - introduction to the web services rest api's. [Online]. Dostupné z WWW: https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/webservicesintro.html.

[27] J. Ginsberg. Detecting influenza epidemics using search engine query data. Google, Inc., 2009.

[28] Apache hadoop. [Online]. Dostupné z WWW: https://developer.yahoo.com/hadoop/tutorial/module2.html#basics.

Seznam obrázků

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. Petr Volf**

Studium: I1300493

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Analýza metod a matematických přístupů pro distrubované výpočty**

Název diplomové práce AJ: Analyses of distributed computing methods and mathematical principles

Garantující pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: Mgr. Josef Horálek, Ph.D.

Oponent: Ing. Jakub Pavlík

Datum zadání závěrečné práce: 1.12.2013