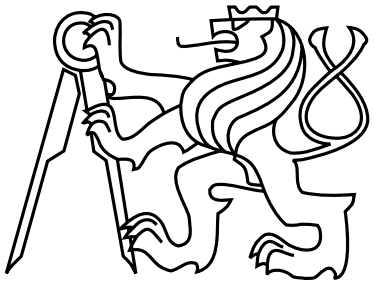
Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Diplomová práca

Veľkoobjemové úložisko emailov

Bc. Patrik Lenárt

Vedúci práce: Ing. Jan Šedivý, CSc.

Študijný program: Otvorená informatika, Magisterský

Odbor: Softwarové inžinierstvo

13. mája 2011

Poďakovanie

Rád by som poďakoval vedúcemu práce pánovi Ing. Janovi Šedivému, CSc. za konzultácie, cenné rady, pripomienky a návrhy, ktoré mi ochotne poskytol počas vypracovávania tejto práce. Tak isto sa chcem poďakovať svojim najbližším, bez ktorých podpory by táto práca nevznikla.

Prehlásenie

Prehlasujem, že som svoju diplomovú prácu vypracoval samostatne a použil som iba podklady uvedené v priloženom zozname.
Nemám závažný dôvod proti užitiu tohto školského diela v zmysle §60 Zákona č. 121/2000 Sb., o autorskom práve, o právach súvisiacich s právom autorským a o zmene niktorých zákonov (autorský zákon).

V Prahe dňa 10. 5. 2011 .....

Abstract

The aim of this thesis is to design an email archiving solution using a distributed database system, commonly known as NoSQL. We made a strong analysis of requirements for the high capacity and fault tolerant storage of email messages, featuring attachment deduplication to provide efficient space saving. We described the basic concepts that are used by distributed database systems. We identified main criteria for high level comparison of modern NoSQL systems, then we chose two possible candidates that comply with requirements. Functions and architectures of Cassandra and HBase systems were described, subjects were thoroughly tested for performance. We modelled and implemented the application prototype using Cassandra.

Abstrakt

Cieľom tejto práce je návrh riešenia pre archiváciu elektronickej pošty s využitím distribuovaných databázových systémov, označovaných pod spoločným názvom NoSQL. Analyzovali sme požiadavky na veľkoobjemový a vysokodostupný archív emailových správ, ktorý bude z dôvodu úspory diskového priestoru využívať deduplikáciu príloh. V práci popisujeme základné koncepty, ktoré sa využívajú pri tvorbe distribuovaných databázových systémov. Stanovili sme základné kritéria na porovnanie NoSQL systémov, z ktorých sme následne vybrali dvoch kandidátov, spĺňajúcich požiadavky systému. Popísali sme funkcionalitu systémov Cassandra a HBase, ktoré sme podrobili výkonnostným testom. Navrhli sme model a implemetovali prototyp aplikácie využivajúci systém Cassandra.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

S neustálym rozvojom informačných technológií súčasne narastá objem informácií, ktoré je potrebné spracúvať. Tento fakt podnietil vznik databázových systémov, ktoré slúžia na organizáciu, uchovávanie a spracovanie dát. V dnešnej dobe existuje veľké množstvo databázových systémov, ktoré sa navzájom líšia napríklad architektúrou, dátovým modelom alebo výrobcom.

Od začiatku sedemdesiatych rokov 20. storočia sú v tejto oblasti dominantou relačné databázové systémy (Relational Database Management Systems). Z dôvodu rapídneho rastu dát v digitálnom univerze [22] začínajú byť tieto systémy nepostačujúce. Medzi hlavné faktory pre výber relačného databázového systému doposiaľ patrili výrobca, cena a pod. Vznikajúce moderné webové aplikácie (napríklad sociálne siete) požadujú od týchto systémov vlastnosti ako vysoká dostupnosť, horizontálna rozšíriteľnosť a schopnosť pracovať s obrovským objemom dát (PB, petabajt[1]). Novo vznikajúce databázové systémy, spĺňajúce tieto požiadavky sa spoločne označujú pod názvom NoSQL (Not Only SQL). Pri ich výbere je dôležité porozumenie architektúry, dátového modelu a dát, s ktorými budú tieto systémy pracovať.

Táto práca si kladie za cieľ viacero úloh a je rozdelená do troch logických častí. V prvej časti popisujeme v kapitole 2, koncepty využívané pri tvorbe distribuovaných databázových systémov, ktoré zabezpečujú vysokú dostupnosť, spoľahlivosť a škálovateľnosť. Kapitola 3 definuje požiadavky na systém, schopný archivovať milióny emailových správ a tieto dáta ďalej spracúvať. Prehľad systémov NoSQL a kritéria pre ich porovnanie popisuje kapitola 4, ktorá v závere doporučuje výber dvoch vhodných kandidátov, systém Cassandra a HBAse. Architektonické princípy, z ktorých tieto systémy vychádzajú a ich vlastnosti sú popísané v kapitole 5 pre systém Cassandra a v kapitole 6 pre HBase. V druhej časti práce v kapitole 7 sme vykonali výkonostné porovnanie týchto dvoch systémov zamerané na operáciu zápisu dát. Návrh modelu systému, výber vhodných nástrojov pre implementáciu prototypu aplikácie emailového úložiska a popisuje kapitola 8.

Záverečná časť, ktorú tvorí kapitola 9 popisuje dosiahnuté výsledky a doporučuje systém pre riešenie danej úlohy.

[1] $1PB = 10^{15}B$

Kapitola 2

Databázové systémy

V tejto časti stručne popíšeme históriu vzniku databázových systémov. Identifikujeme problémy spojené so škálovaním relačných databázových systémov a uvedieme možné spôsoby ich riešenia. Ďalej popíšeme základné koncepty využívané pri tvorbe distribuovaných databázových systémov a techniku MapReduce, ktorá slúži na paralelné spracovanie veľkého objemu dát (PB).

2.1 História

V polovici šesťdesiatych rokov 20. storočia bol spoločnosťou IBM vytvorený informačný systém IMS (Information Management System), využívajúci hierarchichký databázový model. IMS je po rokoch vývoja využívaný dodnes. Po krátkej dobe, v roku 1970, publikoval zamestnanec IBM, Dr. Edger F. Codd článok pod názvom „A Relational Model of Data for Large Shared Data Banks“ [11], ktorým uviedol relačný databázový model. Prvým databázovým systémom implementujúcim tento model bol System R od IBM. Systém používal jazyk pod názvom SEQUEL, ktorý je predchodca dnešného SQL (Structured Query Language) slúžiaceho na manipuláciu a definíciu dát v relačných databázových systémoch. Tento koncept sa stal základom pre relačné databázové systémy, ktoré vďaka širokej škále vlastností (ako napríklad podpora transakcií a dotazovací jazyk SQL) patria v dnešnej dobe medzi najpoužívanejšie riešenia na trhu.

V minulosti boli objem dát, s ktorým tieto systémy pracovali menší a výkon hardvéru mnohonásobne nižší. Dnes napriek tomu, že výkon procesorov a veľkosť pamäťových zariadení rapídne stúpa, je najväčšou slabinou počítačových systémov rýchlosť prenosu dát medzi pevným diskom a operačnou pamäťou. Tento fakt je kritický pre novovznikajúce webové aplikácie ako napríklad sociálne siete alebo *cloudové systémy*, ktoré majú neustále vyššie nároky na spracovávaný objem dát v reálnom čase a vyžadujú podporu škálovania, ktorá zabezpečuje vysokú dostupnosť a spoľahlivosť. Tieto požiadavky sa snažia efektívne riešiť novovznikajúce distribuované systémy pod spoločným názvom NoSQL, ktoré sú popísané v štvrtej kapitole tejto práce.

2.2 Distribuované databázové systémy

Distribuovaný databázový systém je tvorený pomocou viacerých samostatne operujúcich databázových systémov, ktoré nazývame uzly a ich komunikácia sa vykonáva prostredníctvom počítačovej siete. Užívateľovi alebo aplikácii sa javia ako jeden celok [30]. Poďla konfigurácie jednotlivých uzlov v systéme ich ďalej delíme na:

- distribuované databázove systémy typu „master-slave“
- decentralizované distribuované databázové systémy

Systémy master-slave

Táto konfigurácia obsahuje uzol *master*, ktorý plní jedinečnú úlohu a všetky uzly typu *slave* sú na ňom závislé. V prípade jeho havárie je ohrozená funkčnosť celého systému, nazývame ho kritický bod výpadku (SPOF, single point of failure). Túto konfiguráciu uzlov používajú napríklad relačné databázové systémy.

Decentralizované systémy

Pod názvom decentralizácia sa myslí, že každý uzol v distribuovanom databázovom systémy vykonáva tú istú úlohu a je kedykoľvek nahraditeľný. Táto konfigurácia neobsahuje SPOF.

2.3 ACID

Relačné databázové systémy poskytujú veľkú množinu operácií, ktoré je možné vykonávať nad dátami v nich uloženými. Transakcie [27, 28] sú zodpovedné za korektné vykonanie operácií v prípade, že spĺňajú množinu vlastností ACID. Význam jednotlivých vlastností akronymu ACID je následovný:

- *Atomicita* (Atomicity) - zaisťuje, že sa vykonajú všetky operácie reprezentujúce transakciu, čo spôsobí korektný prechod systému do nového stavu. V prípade zlyhania transakcie nemá daná operácia žiaden vplyv na výsledný stav systému a prechod do nového stavu sa nevykoná.
- *Konzistencia* (Consistency) - každá transakcia po svojom úspešnom ukončení garantuje korektnosť svojho výsledku a zabezpečí, že systém prejde z jedného konzistentného stavu do druhého. Konzistentný stav zaručuje, že dáta v systéme odpovedajú požadovanej hodnote. Systém sa musí nachádzať v konzistentnom stave aj v prípade zlyhania transakcie.
- *Izolácia* (Isolation) - operácie, ktoré prebiehajú počas vykonávania jednej transakcie nie sú viditeľné ostatným. Operácie tvoriace transakciu musia mať konzistentný prístup k dátam a to aj v prípade, že u inej transakcie dôjde k jej zlyhaniu.
- *Trvácnosť* (Durability) - v prípade, že bola transakcia úspešne ukončená, systém musí garantovať trvácnosť jej výsledku aj v prípade svojho zlyhania.

Implementácia vlastností ACID, ktoré zaručujú konzistenciu, zvyčajne využíva u relačných databázových systémov metódu zamykania. Transakcia uzamkne dáta pred ich spracovaním a spôsobí ich nedostupnosť až do jej úspešného ukončenia, poprípade zlyhania. Tranzakcie sú vykonávané sekvenčne. Pre databázový systém, od ktorého požadujeme vysokú dostupnosť tento model nie je vyhovujúci. Zámky spôsobujú stavy, kedy ostatné transakcie musia čakať na ich uvoľnenie. Náhradou je mechanizmus s názvom „riadenie súbežného spracovania s viacerými verziami“ (MVCC, Multiversion concurrency control), ktorý umožnuje paralelné vykonávanie operácií nad dátami, jeho popis obsahuje práca od P. Bernsteina a N. Goodmana [6]. Tento mechanizmus je zároveň využívaný systémami NoSQL.

Tranzakcie splňujúce vlastnosti ACID využívajú v distribuovaných databázových systémoch dvojfázový potvrdzovací protokol (Two-phase commit protocol [7]). Systém využívajúci tento protokol, zaručuje konzistentnosť a je schopný odolávať sieťovým prerušeniam (network partitioning) alebo poruchám v systéme. Vlastnosti ACID nekladú žiadnu záruku na dostupnosť systému, naopak nedostupnosť je uprednostnená v prípade operácie, ktorá by mohla spôsobiť nekonzistenciu dát. Takéto systémy sú vhodné pre aplikácie, v ktorých sa vykonávajú platobné operácie a pod. Existuje množstvo aplikácií, u ktorých sa uprednosňuje dostupnosť dát pred ich konzistenciou. Pri tvorbe distribuovaných databázových systémov je preto potrebné upustiť z niektorých ACID vlastností. Riešenie poskytuje model pod názvom BASE 2.5.

2.4 Škálovanie databázového systému

Obecná definícia pojmu *škálovateľnosť* [8] je náročná bez vymedzenia kontextu, ku ktorému sa vzťahuje. V tejto kapitole budeme pojem škálovateľnosť chápať v kontexte webových aplikácií, ktorých dynamický vývoj kladie na databázové systémy viacero požiadaviek. Definujme škálovatelnosť databázového systému ako vlastnosť, vďaka ktorej je systém schopný spracúvať požiadavky webovej aplikácie v definovanom časovom intervale. Medzi hlavné z týchto požiadavkov patrí vysoká dostupnosť, spoľahlivosť a odolnosť systému voči chybám. Typicky sa táto vlasnosť realizuje pridaním nových uzlov do aktuálneho systému s využitím replikácie. Aplikácia týchto mechanizmov má za následok využitie distribuovaného databázového systému. Škálovateľnosť delíme na vertikálnu, horizontálnu a systému dodáva ďalšie z nasledujúcich vlastnosti [30]:

- umožňuje zväčšiť veľkosť celkovej kapacity databáze a táto zmena by mala byť transparentná z pohľadu aplikácie na dáta
- zvyšuje celkové množstvo operácií, pre čítanie a zápis dát, ktoré je systém schopný vykonať v danú časovú jednotku
- v určitých prípadoch môže zaručiť, že systém neobsahuje kritický bod výpadku
- zvyšuje celkovú dostupnosť systému

Vertikálna škálovateľnosť je metóda, ktorá sa aplikuje pomocou zvýšovania výkonnosti hardvéru, do systému sa pridáva operačná pamäť, rychlejšie viacjádrové procesory, zvyšuje sa kapacita diskov. Jednou z nevýhod tohoto riešenia je jeho vysoká cena a možná nedostupnosť

systému v prípade jeho zlyhania. Proces vertikálneho škálovania sa hlavne aplikuje v prípade použitia relačných databázových systémov a obsahuje následujúce kroky:

- zámena hardvéru za výkonnejší
- úprava súborového systému (napr. zrušenie žurnálu a uchovávania informácie o poslednom prístupe k súborom)
- optimalizácia databázových dotazov, indexovanie
- pridanie vrsty pre kešovanie (memcached, EHCache, atď.)
- denormalizácia dát v databáze, porušenie normalizácie

V tomto prípade je možné naraziť na výkonnostné hranice bežne dostupného hardvéru a na rad nastupuje horizontálna škálovateľnosť, ktorá je omnoho komplexnejšia. Horizontálnu škálovateľnosť je možné realizovať pomocou využitia replikácie alebo metódou „rozdeľovania dát“ (sharding).

2.4.1 Replikácia

V distribuovaných systémoch má použitie replikácie za následok, že sa daná informácia nachádza na viacerých uzloch[1] tohto systému. Táto technika zvyšuje dostupnosť, spoľahlivosť a odolnosť systému voči chybám. Replikácia nie je určená pre zálohu dát.

V prípade distribuované databázového systému sa časť informácií uložených v databáze nachádza na viacerých uzloch. Toto usporiadanie môže napríklad zvýšiť výkonnosť operácií, ktoré pristupujú k dátam a to tak, že dochádza k čítaniu dát z databázy paralelne z viacerých uzlov. V systéme obsahujúcom repliku dát nedochádza k strate informácií v prípade poruchy uzlu. Replikácia a propagácia zmien (pridanie alebo odstránenie uzlu s replikou) v systéme sú z pohľadu aplikácie transparentné. Použitie replikácie nezvyšuje pridávaním nových uzlov celkovú kapacitu databázy. Problémom tejto techniky je konzistencia dát. Dáta sa zapisujú na viacero fyzicky oddelených uzlov a zmena sa nemusí prejaviť okamžite vo všetkých replikách. Z pohľadu klienta pristupujúceho k replikovaným dátam, može byť ich obsah nekonzistentný. Medzi metódy pomocou, ktorých je možné zabezpečiť konzistenciu patria:

- *Read one - Write all* - u tejto metódy sa čítanie dát prevedie z ľubovolného uzlu obsahujúceho repliku. Zápis dát sa vykoná na všetky uzly s replikou a až v prípade, že každý z nich potvrdí úspech tejto operácie je výsledok považovaný za korektný. Táto metóda nie je schopná pracovať v prípade ak dôjde k prerušeniu sieťového spojenia medzi uzlami alebo v prípade poruchy jedného z uzlov.
- *metóda kvóra* - viď. **??**

V relačných databázových systémoch sa replikácia rieši pomocou architektúry master - slave. Uzol pod názvom master slúži ako jediný databázový stroj, na ktorom sa vykonáva zápis dát a replika týchto dát je následne distribuovaná na zvyšné uzly pod názvom slave.

[1] Pod pojmom uzol v tomto prípade myslíme samostatný počitačový systém, ktorý je súčasťou distribuovaného systému

Táto metóda umožnuje mnohonásobne zvýšiť počet operácií, ktoré slúžia pre čítanie dát z databazového systému a v prípade zlyhania niektorého zo systémov máme neustále k dispozícii kópiu dát. Slabinou v tomto systéme je uzol v roli master, ktorý nezvyšuje výkonnosť v prípade operácií vykonavajúcich zápis a zároveň jeho porucha môže spôsobiť celkovú nedostupnosť systému.

Druhým možným riešením je technika „multi - master“, kde každý uzol obsahujúci repliku je schopný zápisu dát a následne tieto zmeny preposiela ostatným. Tento mechanizmus predpokladá distribuovanú správu zamykania a vyžaduje algoritmy pre riešenie konfliktov v prípade nekonzistentných dát.

2.4.2 Rozdeľovanie dát

Rozdeľovanie dát (sharding) je metóda založená na princípe, kde dáta obsiahnuté v databáze rozdeľujeme podľa stanovených pravidiel do menších celkov. Tieto celky môžeme následne umiestniť na navzájom rôzne uzly distribuovaného databázového systému. Táto metóda umožňuje zvýšiť výkonnosť operácií pre zápis a čítanie dát a zároveň pridávaním nových uzlov do systému zvyšuje celkovú kapacitu databáze. V prípade, že architektúra distribuovaného databázového systému je navrhnutá s využitím tejto metódy, je zvýšenie výkonu operácií a objem uložených dát sa realizuje automaticky bez nutnosti zásahu do užívateľskej aplikácie.

Techniku rozdeľovania dát môžeme považovať za aplikáciu architektúry známej pod názvom „zdieľanie ničoho“ (shared nothing) [37]. Táto architektúra sa používa pre návrh systémov využívajúcich multiprocesory. V takomto prípade sa medzi procesormi nezdieľa operačná ani disková pamäť. Architektúra zabezpečuje takmer neobmedzenú škálovateľnosť systému a využíva ju mnoho NoSQL systémov ako napríklad Google Bigtable [10], Amazon Dynamo [15] alebo technológia MapReduce [13].

Pri návrhu distribuovaných databázových systémov s využitím tejto techniky patrí medzi kľúčový problém implementácia funkcie spojenia (JOIN) nad dátami, ktorá sa preto neimplementuje. V prípade, že sa, dáta nad ktorými by sme chceli túto operáciu vykonať, nachádzajú na dvoch rozdielnych uzloch prepojených sieťou, takéto spojenie by značne znížilo celkovú výkonnosť systému a viedlo by k zvýšeniu sieťového toku, záťaži systémových zdrojov a možným nekonzistentným výsledkom.

Keďže sa dáta nachádzajú na viacerých uzloch systému, hrozí zvýšená pravdepodobnosť hardverového zlyhania, poprípade prerušenie sieťového spojenia a preto sa táto technika často kombinuje s pomocou využitia replikácie.

V prípade použitia tejto techniky v relačných databázach, je nutný zásah do logiky aplikácie. Dáta uložené v tabuľkách relačnej databázy zachytávajú vzájomné relácie a týmto spôsobom dochádza k celkovému narušeniu tohto konceptu. Príkladom môže byť tabuľka obsahujúca zoznam zamestnancov, ktorú rozdelíme na samostatné celky. Každá tabuľka bude reprezentovať mená zamestnancov, ktorých priezvisko začína rovnakým písmenom abecedy a zároveň sa bude nachádzať na samostatnom databázovom systéme. Táto technika so sebou prináša problém, v ktorom je potrebné nájsť vhodný kľúč, podľa ktorého budeme dáta rozdeľovať a zabezpečíme tak rovnomerné zaťaženie uzlov v systéme. Existuje viacero metód, ktoré je možné použiť pre rozdeľovanie dát na úrovni aplikácie [30]:

- segmentácia dát poďla funkcionality - dáta, ktoré je možné popísať spoločnou vlastnosťou ukladáme do samostatných databáz a tieto umiestňujeme na rozdielné uzly systému. Príkladom môže byť samostatný uzol spravujúci databázu pre užívateľov a iný uzol s databázou pre produkty. Túto metódu spracoval Randy Shoup[2][36], architekt spoločnosti eBay.
- rozdeľovanie dát podľa kľúča - v datách identifikujeme kľúč, pomocou ktorého je možné ich rovnomerne rozdeliť. Následne sa na tento kľúč aplikuje hašovacia funkcia a na základe jej výsledku sa tieto dáta umiestňujú na jednotlivé uzly.
- vyhľadávacia tabuľka - jeden uzol v systéme slúži ako katalóg, ktorý určuje, na ktorom uzle sa nachádzajú dané dáta. Tento uzol zároveň spôsobuje zníženie výkonu a v prípade jeho havárie spôsobuje nedostupnosť celého systému (SPOF).

Replikácia a rozdeľovanie dát patria medzi kľúčové vlastnosti využívané v NoSQL systémoch, ktoré popisuje kapitola 4.

2.5 BASE

Akronym BASE bol prykrát použitý v roku 1997 na sympóziu SOSP (ACM Symposium on Operating Systems Principles) [17]. Tento model poľavil na požiadavku zodpovednom za konzistenciu dát, ktorý je garantovaný vlastnosťou ACID. BASE tvoria nasledujúce slovné spojenia:

- *„bežne dostupný“* (Basically Available) - systém je schopný zvládať čiastočné zlyhanie za cenu nižšej komplexity.
- *„zmiernený stav“* (Soft State) - systém nezaručuje trvácnosť dát s cieľom zvýšenia výkonu.
- *„čiastočne konzistentný“* (Eventually Consistent) - je možné na určitú dobu tolerovať nekonzistentnosť dát, ktoré musia byť po uplynutí určitého časového intervalu znovu konzistentné.

Využitím tohto modelu v distribuovanom databázovom systéme sa dosahuje vyššia dostupnosť aj v prípade čiastočného zlyhania alebo sieťového prerušenia. Distribuovaný databázový systém môžeme klasifikovať ako systém spĺňajúci vlasnosti ACID, BASE alebo oboje.

BASE umožnuje horizontálne škálovanie relačných databázových systémov bez nutnosti použitia distribuovaných transakcií. Pre implementáciu tejto techniky môžeme použiť rozdeľovanie dát s metódou segmentácie dát podľa funkcionality [34].

Bankomatový systém je príkladom systému obsahujúceho čiastočnú konzistenciu dát. Po vybraní určitej čiastky z účtu, sa korektná informácia o aktuálnom zostatku môže zobraziť až za niekoľko dní, kdežto transakcia ktorá túto zmenu vykonala musí spĺňať vlasnosti ACID. Medzi webové aplikácie, u ktorých sa nepožadujú všetky vlasnosti ACID patria napríklad nákupný košík spoločnosti Amazon[3], zobrazovanie časovej osi aplikácie Twitter, poprípade systémy spoločnosti Google[4] indexujúce obsah webu. Ich nedostupnosť by zname-

[2] “If you can't split, you cant scale it.” – Randy Shoup, Distinguished architect Ebay
[3] http://www.amazon.com
[4] http://www.google.com

nala obrovské finančné straty (napríklad zlyhanie vyhľadávania pomocou systému Google by znamenalo zobrazenie nižšieho počtu reklám, nedstupnosť nákupného košíka Amazon by spôsobila pokles predaja atp).

Aplikácia vyššie popísaných techník na relačné databázové systémy môže byť netriviálnou úlohou. Relačný model, je spôsob reprezentácie dát, ktorý umožnuje efektívne riešit určité typy úloh, preto snaha prispôsobiť tento model každému problému je nezmyselná. V tomto prípade, môžeme uvažovať alternatívne riešenia, medzi ktoré patria systémy NoSQL.

2.6 CAP

Moderné webové aplikácie kladú na systémy požiadavky, medzi ktoré patrí vysoká dostupnosť, konzistencia dát a schopnosť odolávať chybám. Dr. Brewerer v roku 2000 nastolil myšlienku, dnes známu pod názvom teória CAP [9]. U distribuovaných databázových systémov, ktoré používajú pre vzájomnú komunikáciu sieť musíme predpokládať s prítomnosťou sieťových prerušení. Táto teória tvrdí, že u takýchto systémov je možné súčasne dosiahnúť len dvojicu z vlastností CAP a to CP alebo AP. V roku 2002 platnosť tejto teórie pre asynchrónnu sieť matematicky dokázali Lynch a Gilbert [26]. Modelu asynchrónnej sieťe svojimi vlasnosťami zodpovedá Internet. Akroným CAP tvoria následujúce vlasnosti:

- *Konzistencia* (Consistency) - distribuovaný systém je v konzistentnom stave, ak každý jeho uzol v prípade požiadavku dát vracia tú istú odpoveď.
- *Tolerancia chýb* (Partition Tolerance) - uzly distribuovaného systému navzájom komunikujú pomocou siete, v ktorej hrozí strata správ. V prípade vzniku sieťového prerušenia dané uzly medzi sebou navzájom nedokážu komunikovať. Táto vlasnosť podľa definície (viď. Gilbert a Lynch) tvrdí, že v prípade vzniku zlyhania sieťovej komunikácie medzi niektorými uzlami, musí byť systém schopný naďalej pracovať korektne. V reálnych podmienkach neexistuje distribuovaný systém, ktorého uzly na vzájomnú komunikáciu využívajú sieť a nedochádza pri tom k strate správ, teda k poruchám sieťovej komunikácie.
- *Dostupnosť* (Availability) - distribuovaný systém je dostupný, ak každý jeho uzol, ktorý pracuje korektne, je schopný pri prijatí požiadavku zaslať odpoveď. V spojení s toleranciou chýb, tato vlastnosť hovorí, že v prípade ak nastane sieťový problém[5], každá požiadavka bude vykonaná.

Pravdepodobnosť, že dôjde k zlyhaniu ľubovoľného uzla v distribuovanom systéme, exponenciálne narastá s počtom pribúdajúcich uzlov.

$$P(A) = 1 - P(B)^{\text{počet uzlov}}$$

P(A) - pravdepodobnosť zlyhania ľubovoľného uzlu
P(B) - pravdepodobnosť, že individuálny uzol nezlyhá

[5] týmto sa nemyslí porucha uzla

2.6.1 Konzistencia verzus dostupnosť

V distribuovanom systéme nie je možné súčasne zaručiť vlasnosť konzistencie a dostupnosti. Ako príklad si predstavme distribuovaný systém obsahujúci tri uzly A, B, C, ktorý zaručuje obe vlasnosti aj v prípade sieťového prerušenia. Na všetkých uzloch sa nachádzajú identické (replikované) dáta. Ďalej uvažujme, že došlo k sieťovému prerušeniu, ktoré rozdelilo uzly na dva samostatné celky {A,B} a {C}. V prípade, že uzol C obdrží požiadavku pre zmenu dát má na výber z dvoch možnosti:

1. vykonať zmenu dát čo spôsobí, že sa uzly A a B o tejto zmene dozvedia až vo chvíli ak bude sieťové prerušenie odstranené

2. zamietnuť požiadavok na zmenu dát, z dôvodu že uzly A a B sa o tejto zmene nedozvedia

V prípade výberu možnosti čislo 1 zabezpečíme neustálu dostupnosť systému naopak v prípade možnosti číslo 2 jeho konzistenciu. Nie je možný súčasný výber oboch možností.

CP

Ak od daného systému tolerujúceho sieťové prerušenia požadujeme konzistenciu na úkor dostupnosti jedná sa o alternatívu CP. Takýto systém zabezpečí konzistentnosť operácií pre zápis a čítanie dát a zároveň sa môže stať, že na určité požiadavky nebude schopný reagovať (možnosť čislo 2). Medzi takéto systémy môžeme zaradiť distribuovaný databázový systém využívajúci dvojfázový potvrdzovací protokol.

AP

V prípade, že poľavíme na požiadavku konzistencie tak takýto systém bude vždy dostupný aj napriek sieťovým prerušeniam. V tomto prípade sa jedná o model AP. Je možné, že v takomto systéme bude dochádzať ku konfliktným zápisom alebo operácie čítania budú po určitú dobu vracať nekonzistentné výsledky. Tieto problémy s konzistenciou sa v distribuovaných databázových systémoch riešia napríklad pomocou metódy „vektorových hodin“ (Vector clock) [**?**] alebo na aplikačnej úrovni na strane klienta. Príkladom systému patriaceho do tejto kategórie je Amazon Dynamo.

CA

Ak systém nebude tolerovať sieťové prerušenia, tak bude spĺnovať požiadavok konzistencie a dostupnosti, varianta CA. Jedná sa o nedistribuované systémy pracujúce na jednom fyzickom hardvéri využivajúce databázové transakcie.

Pri výbere distribuovaného databázového systému, môžeme vďaka vyššie popísaným vlastnostiam určiť vhodnosť jeho použitia, na základe požiadavkov aplikácie.

2.7 Čiastočná konzistencia

V distribuovaných systémoch sa pod pojmom konzistencie v ideálnych podmienkách rozumie vlastnosť, ktorá zaručí, že zmena dát (zápis alebo aktualizácia dát) sa prejaví súčasne s rovnakým výsledkom. Konzistencia je zároveň úzko spojená s replikáciou. Väčšina NoSQL systémov poskytuje čiastočnú konzistenciu, poprípade dáva možnosť výberu medzi vlastnosťami CP a AP (napríklad systém Cassandra[6]). V následujúcej časti popíšeme rôzne druhy konzistencie.

V predchádzajúcom texte sme už spomínali, že v dnešne dobe existuje mnoho aplikácií, u ktorých je možné poľaviť na požiadavku konzistencie. Ak sa určitá zmena prejaví s miernym oneskorením funkčnosť systému nebude v tomto prípade ohrozená. Táto konzistencia nie je totožná s konzistenciou definovanou u vlastností ACID, kde ukončenie transakcie zaručuje, že sa systém nachádza v konzistentom stave. Na konzistenciu sa môžeme pozerať z dôch pohľadov. Prvým, je klientský pohľad na strane zadávateľa problému resp. programátora, ktorý rozhodne aká je závažnosť zmien, ktoré sa budú vykonávať v systéme. Druhý pohľad je systemový, zabezpečuje technické riešenie a implementáciu techník zodpovedných za správu konzistencie v distribuovaných databázových systémoch.

2.7.1 Konzistencia z pohľadu klienta

Pre potrebu následujúcich definíc uvažujme distribuovaný databázový systém, ktorý tvorí úložisko dát a tri nezávislé procesy {A, B, C}, ktoré možu v danom systéme zmeniť (vykonať zápis) a načítať hodnotu dátovej jednotky. Na základe toho ako jednotlivé nezávislé procesy pozorujú zmeny v systéme delíme konzistenciu na [40]:

Silná konzistencia (Strong consistency) - proces A vykoná zápis. Po jeho ukončení je nová hodnota dátovej jednotky dostupná všetkým procesom {A, B, C}, ktoré k nej následne pristúpia (vykonajú operáciu čítania). Túto konzistenciu zabezpečujú transakcie s vlastnosťami ACID.

Slabá konzistencia (Weak consistency) - proces A vykoná zápis novej hodnoty do dátovej jednotky. V takomto prípade systém negarantuje, že následne pristupujúce procesy {A, B, C} k tejto jednotke vrátia hodnotu zapísanú procesom A. Definujeme pojem „nekonzistentné okno“, ktoré zabezpečí, že po uplynutí stanovenej časovej doby sa táto nová hodnota dátovej jednotky prejaví vo všetkých procesoch, ktoré k nej pristúpia.

Čiastočná konzistencia (Eventual consistency) - je to špecifická forma slabej konzistencie. V tomto prípade systém garantuje, že ak sa nevykoná žiadná nová zmena hodnoty dátovej jednotky, po určitom čase budu všetky procesy pristupujúce k tejto jednotke schopné vrátiť jej korektnú hodnotu. Tento model ma viacero variacií, niektoré z nich popíšeme v nasledujúcej časti textu.

Model čiastočnej konzistencie ma viacero variácií:

[6] http://cassandra.apache.org

Read-your-write consistency - v prípade, že proces A zapíše novú hodnotu do dátovej jednotky, žiadny z jeho následujúcich prístupov k tejto jednotke nevráti staršiu hodnotu ako naposledy zapísaná.

Session consistency - v tomto prípade pristupuje proces k systému v kontexte relácií. Po dobu trvania relácie platí predchádzajúci typ konzistencie. V prípade zlyhania relácie sa vytvorí nová, v ktorej môže systém vraciať hodnotu dátovej jednotky, zápisanú pred vznikom predchádzajúcej relácie.

Monotonic read consistency - v prípade, že proces načítal hodnotu dátovej jednotky, tak pri každom následujúcom prístupe nemôže vrátiť predchádzajúcu hodnotu dátovej jednotky.

Tieto typy konzistencie je možné navzájom kombinovať a ich hlavným cieľom je zvýšiť dostupnosť distribuovaného systému na úkor toho, že poľavíme na požiadavkoch konzistencie. Príkladom systému s čiastočnou konzistenciou je asynchrónna replikácia v relačnom databázovom systéme využívajúca architektúru master-slave.

2.7.2 Systémová konzistencia

Techniky založené na protokoloch kvóra (Quorum-based protocols [25]) je možné použiť pre zvýšenie dostupnosti a výkonu v distribuovaných databázových systémoch s garanciou silnej alebo čiastočnej konzistencie. Definujme nasledujúcu terminológiu:

- N - počet uzlov, ktoré obsahujú repliku dát
- W - počet uzlov obsahujúcich repliku, na ktorých sa musí vykonať zápis, aby bola zmena úspešne potvrdená
- R - počet uzlov s replikov, ktoré musia vrátiť hodnotu dátového objektu v prípade operácie čitanie

V prípade, že platí $W + R > N$, operácie pre zápis a čítanie dát sa stále prekrývajú minimálne na jednom uzle, ktorý bude vždy obsahovať aktuálnu hodnotu danej operácie. Tento prípad, zabezpečuje silnú konzistenciu v systéme. V prípade $W + R <= N$, môže nastať situácia keď predchádzajúca podmienka neplatí a teda daná operácia je čiastočne konzistentná. Rôzna konfigurácia týchto parametrov zabezpečí rozdielnu dostupnosť a výkonnosť distribuovaného systému. Uvažujme následujúce príklady pre $N = 3$.

1. $R = 1$ a $W = N$, tento prípad zabezpečí, že systém bude optimalizovaný pre operácie čítania dát. Klient číta dáta z ľubovoľnej repliky. Operácie budú konzistentné, pretože uzol z ktorého dáta čítame sa prekrýva s uzlami na ktorých vykonávame zápis. Nevýhodou tohoto modelu je, že nedostupnosti jednej repliky znemožní vykonanie zápisu. V prípade systémov, u ktorych požadujeme aby obsluhovali veľký počet požiadavkov pre čítanie sa môže hodnota N pohybovať v stovkách až tisícoch, závisí to od počtu uzlov v systéme.

2. $W = 1$ a $R = N$, tento prípad je vhodný pre systémy u ktorých požadujeme rýchly zápis. Tento model môže spôsobiť stratu dát v prípade zlyhania uzla s replikou, na ktorú bol vykonaný zápis.

2.8 MapReduce

Nárast diskových kapacít a množstva dát, ktoré na nich ukladáme spôsobuje jeden z ďalších problémov, ktorým je analýza a spracovanie dát. Kapacita pevných diskov sa za posledné roky mnohonásobne zvýšila v porovnaní s dobou prístupu a prenosových rýchlosti pre čítanie a zápis dát na tieto zariadenia.

Pre jednoduchosť uvažujme nasledujúci príklad, v ktorom chceme spracovať pomocou jedného počítačového systému 1 TB dát uložených na lokálnom súborovom systéme, pri priemernej prenosovej rýchlosti diskových zariadení 100 MB/s. Za ideálnych podmienok by čas na prečítanie týchto dát presahoval dve a pol hodiny. V prípade, že by sme 1 TB dát rovnomerne rozdelili na sto počítačov a tieto úseky paralelne spracovali, celková doba by sa znížila za ideálnych podmienok na necelé dve minúty.

Spoločnosť Google v roku 2004 zverejnila programovací model pod názvom MapReduce [13], ktorý slúži na paralelne spracovanie obrovského objemu dát (PB). Model využíva vlastnosti paralelných a distribuovaných systémov, je optimalizovaný pre beh na klastri tvorenom vysokým počtom (tisícky) spotrebných počítačov. Pre programátora sa snaží zastrieť všetky problémy, ktoré prináša paralelizácia výpočtov, poruchovosť systémov, distribúcia dát vzhľadom na ich lokalitu a rovnomerne rozvhovanie záťaže medzi systémami.

Pre použitie tohoto nástroja musí programátor zadefinovať dve funkcie pod názvom *map* a *reduce*. Funkcia map na jednotlivých uzloch systému, transformuje vstupné data na základe zadefinovaného kľúča k a k nemu prináležiacim hodnotám H na medzivýsledok, ktorý obsahuje nové kľúče $g1, ..., g_n$ a k ním odpovedajúce zotriedené hodnoty H. Tieto dáta sa odošlú na uzly vykonávajúce užívateľom definovanú funkciu reduce. Funkcia reduce vykoná nad hodnotami priradenými ku kľúčom $g1, ..., g_n$ požadovanú operáciu, ktorej typickým výsledkom je jedna výsledná hodnota, poprípade viacero hodnôt Z. Tieto operácie je možné popísať následovne:

```
map(k, H) -> list(g, H)
reduce(g, list(H)) -> list(Z)
```

2.8.1 Architektúra

Obrázok

2.8.2 Použitie

MapReduce nie je vhodný na spracovanie dát v reálnom čase. Je optimalizovaný na dávkový beh. Jeho implementácia spoločnosťou Google, ktorá zároveň využíva distribuovaný súborový systém Google File System (GFS) [23] nie je k dispozícii. V rámci hnutia NoSQL vzniklo

open source riešenie pod názvom Hadoop[7], ktoré implementuje tento model na vlastnom distribuovanom súborovom systéme Hadoop Distributed File System (HDFS). K dispozíci sú frameworky HIVE alebo PIG, ktoré sú nadstavbou modelu MapReduce a poskytujú vyššiu abstrakciu vo forme jazyka podobného SQL.

[7] http://hadoop.apache.org

Kapitola 3

Definícia problému

Množstvo digitálnych informácii, každým rokom prudko narastá. Podľa štatistík spoločnosti IDC[22] v roku 2006 dosahovala kapacita digitálneho univerza veľkosť 161 exabytov[1](EB). Podiel elektronickej pošty (emailov) bez spamu, tvoril 3% z tohoto objemu. Predpoveď na rok 2011[21] predpokláda celkovú kapacitu 1800 EB, čo je viac ako desaťnásobok nárastu pôvodnej kapacity v období piatich rokov. V rozmedzí rokov 1998 až 2006 sa mal počet schránok elektronickej pošty zvýšiť z 253 miliónov na 1.6 miliardy. Predpoveď IDC ďalej uvádzala, že po ukončení roku 2010 tento počet presiahne hodnotu dvoch miliard. Počas obdobia medzi rokmi 1998 až 2006 celkový počet odoslaných správ elektronickej pošty rástol trikrát rýchlejšie ako počet jej užívateľov, dôvodom tohoto prudkého nárastu bola nevyžiadaná elektronická pošta. Odhaduje sa, že až 85% dát z celkového predpokladaného objemu 1800 EB budú spracovávať, prenášať alebo zabezpečovať organizácie. Napriek tejto explózii digitálnych informácii je potrebné správne porozumieť hodnote týchto dát, nájsť vhodné metódy pre ich ukladanie do pamäti počítačových systémov, ich archiváciu a to tak, aby sme ich mohli ďalej spracúvať a efektívne využiť. Táto kapitola práce si kladie za cieľ analýzovať potreby pre archiváciu elektronickej pošty a definovať požiadavky pre systém slúžiaci k archivácii emailov.

3.1 Archivácia elektronickej pošty

Z narástajúcim objemom dát reprezentujúcim elektronickú poštu je potrebné porozumieť tejto štruktúre a následne tieto dáta vhodne spracovať. Cieľom je ukládať emailové správy tak aby sme dosiahli úsporu diskového priestoru, boli sme nad nimi schopný vykonávať operácie ako fultextové vyhľadávanie, zber údajov pre tvorbu štatistík alebo umožnili ich opätovné sprístupnenie. Emaily obsahujú čoraz viac obchodných informácií a iný dôležitý obsah, z tohto dôvodu musia byť organizácie všetkých rozmerov schopné uchovávať tento obsah pomocou vhodných archivačných nástrojov. S problémom archivácie zároveň úzko súvisí problém bezpečnosti. Pod pojmom bezpečnosti v tejto oblasti máme na mysli hlavne ochranu proti nevyžiadanej pošte tj. spam, spyware, malware a phishingu. Na boj proti týmto hrozbám využívajú organizácie anti-spamové a anti-vírusové systémy. Možné dôvody prečo archivovať elektronickú poštu sú následovné [33]:

[1] $1EB = 10^{18}B$

- záloha dát a ich obnova v prípade havárie systému
- vysoká dostupnosť dát
- sprístupnenie dát koncovému užívateľovi
- spĺňanie regulačných noriem a zákonov
- ochrania súkromia a e-Discovery
- vyťazovanie dát (data mining)
- efektívne využitie úložného priestoru (deduplikácia príloh)

3.2 Požiadavky na systém

V následujúcej časti popíšeme požadované vlastnosti systému, ktorý bude slúžiť na archiváciu veľkého objemu emailových správ. Primárnou požiadavkou na systém je jeho neustála dostupnosť, rozšíriteľnosť a nízkonákladová administrácia. Predpokladané množstvo uložených dát v tomto systéme bude dosahovať desiatky až stovky terabajtov[2] (TB). Takúto kapacitu dát nie je možne uchovať na bežne dostupnom hardvéri. Dáta uložené v systéme musia byť replikované, v prípade vzniku havárie niektorej z jeho časti. Nad uloženými emailovými správami je potrebné vykonávať výpočtovo náročné operácie ako generovanie štatistík a fultextové vyhľadávanie v reálnom čase. Tieto požiadavky prirodzene implikujú využitie distribuovaného databázového systému. Medzi hlavných kandidátov, vďaka ktorým sme tieto požiadavky schopní vyriešiť patria NoSQL databázové systémy, ktoré popíšeme v následujúcej kapitole.

3.2.1 Funkcionálne požiadavky

Ukladanie emailov

Základnou jednotkou, ktorú budeme do systému ukladať je emailová správa. Graf na obrázku 3.1 znázorňuje uporiadanie emailov podľa ich veľkosti nad vzorkom približne 1,000,000 emailových správ z reálneho prostredia[3]. Z daných dát vyplýva, že veľkosť cca 80% emailov je do 50 kB. Tieto údaje sú hrubou aproximáciou a závisia na konkrétnych používateľoch.

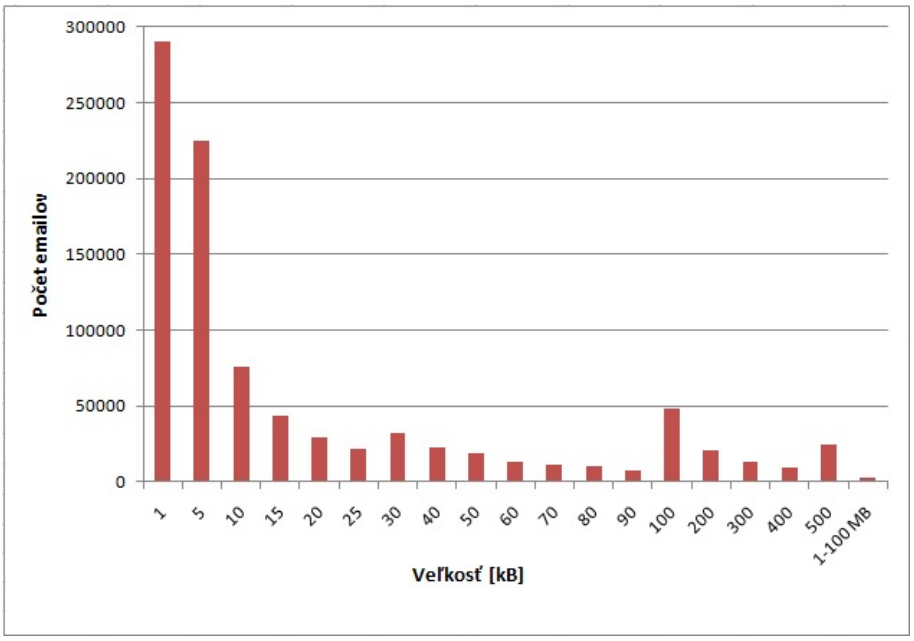
Systém musí umožnovať uloženie emailu bez porušenia jeho integrity. Klúčovým požiadavkom je ukladanie príloh emailov, kde požadujeme aby každá príloha bola jednoznačne identifikovaná a v prípade jej duplicity nebola opakovane uložená v systéme. Cieľom je dosiahnutie úspory diskového priestoru. Ďalším požiadavkom je automatické zmazanie emailov patriacich do danej domény po uplynutí predom špecifikovanej doby.

Export emailov

Systém musí umožnovať prístup k ľubovoľnému uloženému emailu v jeho pôvodnej podobe poprípadne skupine všetkých emailov patriacej danému užívateľovi (inbox).

[2] $1TB = 10^{12}B$

[3] Vzorok emailov pre analýzu bol sprístupnený spoločnosťou Excello.



Obr. 3.1: Približná distribúcia emailových správ. $\mu = 301\,\text{kB}$, $\sigma = 1.3\,\text{MB}$

Vyhľadávanie emailov

Vyhľadávanie je potrebné realizovať nad všetkými emailovými správami uloženými v systéme, jednotlivo nad správami podľa danej domény a nad správami, ktoré prináležia danému uživateľovi. Požadujeme fultextové vyhľadávanie emailov podľa následujúcich údajov:

- príjemca emailovej správy
- odosielateľ emailovej správy
- predmet správy
- dátum obsiahnutý v emailovej správe
- identifikátor emailu (*MessageID*)
- názvy príloh a ich veľkosti
- veľkosť emailu
- vyhľadávanie v tele emailu

Pre administrátorské účely požadujeme vyhľadávanie údajov podľa:

- originálny odosielateľ a príjemca
- IP adresa odosielateľa
- dátum a čas spracovania správy emailovým serverom

Štatistické údaje

Nad uloženými dátami požadujeme výpočet štatistík pomocou využitia MapReduce. Pre emaily patriace do danej domény je potrebné spracovať následujúce štatistické ukazateľe:

- počet emailov označených príznakom spam
- počet emailov bez príznaku spam
- celková veľkosť emailov pre danú doménu
- veľkosť najväčšieho emailu v doméne
- celková dĺžka filtrácie emailov v danej doméne

Nad celým úložiskom je ďalej potrebné spracovať tieto štatistiky:

- počet všetkých emailov
- počet unikátnych domén
- počet unikátnych príloh

3.2.2 Nefunkcionálne požiadavky

Dostupnosť

Systém musí byť neustále dostupný (99,9%), schopný odolávať sieťovým prerušeniam spôsobujúcim nedostupnosť uzlov, úplným zlyhaniam jednotlivých uzlov a umožnovať spracúvať tok pre zápis dát v rozmedzí 10 Mbit až 1 Gbit. Ďalším požiadavkom je aby sa dáta replikovali vo vnútri datacentra na dva uzly a tretia replika bola umiestnená v datacentre, ktoré sa bude nachádzať na geograficky odlišnom mieste. Vyžadujeme aby systém neobsahoval kritický bod výpadku, požadujeme vlastnosť decentralizácie.

Rozšíriteľnosť

Predpokladáme použitie bežne dostupného spotrebného hardvéru (commodity hardware[4]), namiesto superpočítačov. Z dôvodu neustalého nárastu objemu elektronickej pošty, musí systém podporovať horizontálne škálovanie, ktoré bude schopné umožnovať zvýšenie celkovej kapacity dátového úložiska (desiatky petabajtov). Pridávanie nových uzlov do systému umožní zvyšiť celkový vypočetný výkon, ktorý sa využije na spracovanie dát pomocou techniky MapReduce. U distribuovaného databazového systému je nutná podpora replikácie, ktorá zvýši výkonnosť operácií pre čítanie, zápis a vďaka nej nebude potrebné riešiť zalohovanie pomocou externých systémov.

Nízkonákladová administrácia

Prevádzkovanie systému a jeho administrácia by mali byť čo najmenej závislé na zásahu ľudských zdrojov. Detekcia nefunkčných uzlov a automatické rozdeľovanie záťaže sa musí vykonávať automaticky. Pridanie poprípade odobranie nového uzla nesmie ovplyvniť beh celkového systému.

Bezpečnosť

Osoby s oprávnením pre prístup k systému budú schopné manipulovať s jeho celým obsahom. Predpokladáme beh systému v bezpečnom prostredí a nekladieme žiadne požiadavky na uživateľské role v kontexte prístupu k datam.

Implementačné požiadavky

Cieľom je implementácia systému s využitím dostupných open source technologií.

Z analýzy princípov, v predchádzajúcej kapitole, ktoré využívajú relačné databázové syštémy vyplýva, že použitie týchto systémov nie je vhodné pre riešenie zadefinovaného problému. Medzi základné problémy týchto systémov patrí náročné horizontálne škalovanie, čo negatívne ovplyvňuje primárny požiadavok vysokej dostupnosti. V nasledujúcej kapitole sa budeme zaoberať popisom NoSQL systémov a po ich analýze vyberieme vhodného kandidáta, ktorého použijeme k implementácii prototypu, z dôvodu vyskokej komplexnosti riešeného problému.

[4] komponenty sú štandardizované, bežne dostupné a ich cenu neurčuje výrobca (IBM, DELL,...) ale trh.

Kapitola 4

NoSQL

Názov NoSQL bol prvykrát použitý v roku 1998 ako názov relačnej databáze[1], ktorej implementácia bola v interpretovaných programovacích jazykoch a neobsahovala dotazovací jazyk SQL. V druhej polovici roku 2009 sa názov NoSQL začal používať v spojení s databázovými systémami, ktoré nepoužívajú tradičný relačný model, dotazovací jazyk SQL, sú schopné horizontálneho škálovania, pracujú na spotrebných počítačových systémoch, vyznačujú sa vysokou dostupnosťou a používajú bezschémový dátový model.

Pôvodným cieľom hnutia NoSQL bolo vytvoriť koncept, pre tvorbu moderných databáz, ktoré by boli schopné spracúvať požiadavky neustále sa rozvýjajúcich webových aplikácií. Filozofiou týchto systémov je nesnažiť sa za každú cenu prispôsobovať dáta, ktoré do nich ukladáme relačnému databázovému modelu. Cieľom je vybrať systém, ktorý bude čo najvhodnejšie zodpovedať požiadavkom pre uloženie a spracovanie dát. NoSQL nepopisuje žiaden konkrétny databázový systém, namiesto toho je to obecný názov pre nerelačné (non-relational) databázové systémy, ktoré disponujú odlišnými vlastnosťami a umožnujú prácu s rôznymi dátovými modelmi. Medzi ich ďalšie znaky patrí napríklad čiastočná konzistencia, schopnosť spracúvať obrovské objemy dát (PB) a jednoduché programové rozhranie (API). Tieto systémy nepodporujú operáciu databázového spojenia a medzi dátami, ktoré do nich ukladáme je možné vytvárať vzájomné závislosti až na aplikačnej úrovni. Pre tieto databázove systémy ďalej platí, že sú distribuované, automaticky poskytujú replikáciu a rozdeľovanie dát. Jedná sa o relatívne mladé systémy, ktoré sú neustále vo vývoji. Ideou tohto konceptu je riešiť spomínané novo vznikajúce problémy a zároveň koexistovať s relačnými databázovými systémami.

4.1 Dátové modely

NoSQL systémy delíme na dokumentové, grafové, stĺpcové a systémy s dátovým modelom typu „kľúč-hodnota“ (Key-value). V následujúcej časti práce ich stručne popíšeme.

[1] http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql

4.1.1 Relačný model

Relačný databázový model reprezentuje dáta pomocou relácií, ktoré tvoria riadky uložené v tabuľkách. Štruktúra týchto záznamov je normalizovaná aby sa predišlo ich duplikácii. Pre zabezpečenie referenčnej integrity jednotlivých entít sa využívajú cudzie kľúče. Tabuľky s popisom názvov stĺpcov a vzťahy medzi nimi nazývame databázovou schémou. Záznamy sú sekvenčne ukladané na pevný disk. Tento model je vhodný pre systémy, u ktorých sú dominantné operácie vykonávajúce zápis. Relačné databázové systémy sú teda optimalizované pre zápis. Pre efektívny prístup k dátam je možné použiť indexy.

4.1.2 Kľúč-hodnota

Tento model využíva pre ukládanie dát jednoduchý princíp. Blok dát s ľubovoľnou štruktúrou je v databáze uložený pod názvom kľúča, ktorý je často interne reprezentovaný ako poľe bajtov a môže ním byť napríklad textový reťazec. Databázové systémy využívajúce tento dátový model majú jednoduché programové rozhranie, znázorné na obrázku XY.

```
void Put(string kluc, byte[] data);
byte[] Get(string key);
void Remove(string key);
```

Výhodou tohto modelu je, že databázový systém je možné ľahko škálovať. V tomto prípade prácu so štruktúrou uložených dát zabezpečuje klient, čo umožnuje dosahovať vysokú výkonnosť na strane databázového systému.

Jednou z možných nevýhod v porovaní s relačnými databázovými systémami je, že databázový systém nie je schopný medzi uloženými dátami zachytiť ich vzájomné vzťahy (relácie), čo môže byť jedným z požiadavkov pri tvorbe komplexných modelov. Úložiško typu kľúč-hodnota nevyužíva normalizáciu dát, dáta sú často duplikované, vzťahy a integrita medzi nimi sa riešia až na aplikačnej úrovni. Na obsah vkladaných dát a k nim asociovaným kľúčom sa nedefinujú žiadne obmedzenia.

Medzi databázové systémy využívajúce model kľúč-hodnota patria Amazon Dynamo, Tokyo Cabinet [**?**], Voldemort [**?**], Redis [**?**] a iné.

4.1.3 Stĺpcovo orientovaný model

V dnešnej dobe existuje veľký počet aplikácií, u ktorých prevládajú operácie čítania nad zápisom. Patria sem dátové sklady, systémy pre vyťažovanie dát alebo analytické aplikácie pracujúce s obrovským objemom dát. Pre potreby týchto aplikácií a ich reprezentáciu dát je vhodné použiť stĺpcový model [3, 4], ktorý je optimalizovaný pre operácie čítania dát. Data reprezentujúce stĺpce sú uložené na pevnom disku v samostatných a súvislých blokoch. Načítanie dát do pamäti a následná práca s nimi je efektivnejšia ako u riadkových databáz, kde je potrebné načítať celý záznam obsahujúci aj hodnoty stĺpcov, ktoré sú v daný moment irelevantné.

Riadkový model obsahuje v jednom zázname dáta z rôznych domén, čo spôsobuje vyššiu entropiu v porovnaní so stĺpcovým modelom, kde sa predpokláda, že dáta v danom stĺpci

pochádzajú z totožnej domény a sú si preto podobné. Táto vlasnosť umožnuje efektívnu komprimáciu dát, ktorá znižuje počet diskových operácií. Nevýdou tohto modelu je zápis dát, ktorý spôsobuje vyššiu záťaž diskových operácií. Pre optimalizáciu operácií vykonavajúcich zápis sa používa dávkový zápis.

Tento model využívajú databázové systémy ako Google Bigtable, HBase, Hypertable alebo Cassandra.

4.1.4 Dokumentový model

Dokumentové databázy sú založené na modeli typu kľúč-hodnota. Požiadavkom na ukládané dáta je, že musia byť v tvare, ktorý dokáže spracovať databázový systém. Štruktúra vkládaných dát môže byť určená napríklad pomocou XML, JSON, YAML. Schéma týchto systémov následne umožnuje okrem jednoduchého vyhľadávania pomocou modelu kľúč-hodnota vytvárať s využitím indexov zložitejšie dotazy nad dátami, ktoré sú vyhodnocované na strane databázového systému.

Medzi databáze reprezentujúce tento typ úložiska patrí napríklad CouchDB a MongoDB.

4.1.5 Grafový model

Tento typ databáz využíva pre prácu s dátami matematickú štruktúru graf. Dáta sú reprezentované pomocou uzlov, hran a ich atribútov. Základným objektom je uzol. Pomocou hrán medzi uzlami modelujeme závislosti, ktoré sú popísané pomocou atribútov. Nad uzlami a hranami sa využíva model kľúč-hodnota. Medzi hlavné výhody patrí možnosť prechádzania týchto štruktúr s využitím známych grafových algoritmov.

Tento model sa napríklad využíva v aplikáciach sociálnych sieti alebo pre sémantický web. Patria sem databázové systémy Neo4j alebo FlockDB.

4.2 Porovnanie NoSQL systémov

V dnešnej dobe existuje veľké množstvo NoSQL databázových systémov, ktoré majú odlišné vlasnosti, komplexitu a vďaka tomu ich môžeme použiť pre rôzne účely. Snaha porovnať tieto systémy na globálnej úrovni je nerealizovateľná a často vedie k chybám. Cieľom tejto sekcie je definovať základné body vďaka, ktorým je možné tieto systémy kategorizovať a vrámci danej kategórie porovánať. Tieto zistenia nám následne môžu pomôcť pri výbere vhodného systému, ktorý bude vhodný na riešenie našich požiadavok. Následujúce body patria medzi hlavné kritéria pri porovnávaní týchto systémov:

1. dátový model
2. dotazovací model
3. škálovateľnosť
4. schopnosť odolávať chýbam (failure handling)
5. elastickosť

6. konzistencia dát
7. prostredie behu
8. bezpečnosť

4.2.1 Dátový a dotazovací model

Dátový model definuje štruktúru, ktorá slúži na ukladanie dát v databázovom systéme. Dotazovací model následne definuje obmedzenia a operácie, ktoré je možné nad uloženými dátami vykonávať. V predchádzajúcej sekcii sme popísali základnú kategorizáciu NoSQL systémov poďla dátového modelu. Dátový a dotazovací model do určitej miery popisuje výkonnosť, komplexnosť a vyjadrovaciu silu databázového systému. Dotazovací model určuje programové rozhranie (API).

Pri výbere vhodného dátového modelu pre aplikáciu je dôležité porozumieť štruktúre dát a definovať operácie, ktoré nad týmito dátami budeme vykonávať. Na základe týchto operácii sa navrhne databázová schéma.

4.2.2 Škálovateľnosť a schopnosť odolávať chybám

Tieto vlasnosti kladú na systém požiadavky ako podpora replikácie a rozdeľovanie dát. Distribuované databázové systémy implementujú tieto techniky na systémovej úrovni. V prípade ich podpory, môžeme od systému požadovať:

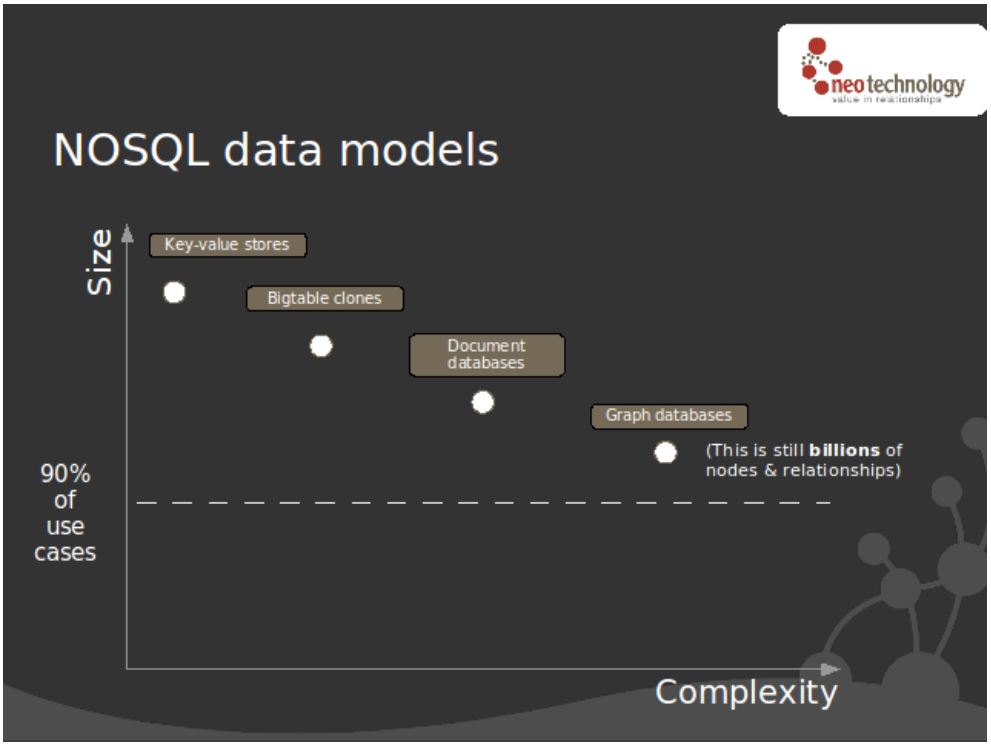
- podporu replikcácie medzi geograficky oddelenými dátovými centrami
- možnosť pridania nového uzlu do distribuovaného databázového systému, bez nutnosti zásahu do aplikácie

Počet uzlov, na ktorých sa vykonáva replika dát a konfigurácia databázového systému, ktorá podporuje geograficky oddelené dátové centrá určujú stupeň odolnosti systému voči chybám, medzi ktoré patria porucha uzla alebo sieťové prerušenie.

Častým požiadavkom webových aplikácií, z dôvodu neustáleho nárastu dát, na databázový systém je podpora škálovania s cieľom zvýšenia veľkosti databáze. S neustálym vývojom nových aplikácií musíme zároveň uvažovať potrebu škálovania aj z pohľadu komplexnosti dát [35]. Výber dátového modelu môže byť ovplyvnený na základe komplexnosti dát. Obrázok 4.2 zachytáva pozíciu dátových modelov z pohľadu škálovania komplexnosti a veľkosti dát.

Dátový model typu kľuč-hodnota a stĺpcovo orientovaný model (Bigtable clones) majú jednoduchú štruktúru, ktorá kladie minimálnu náročnosť v prípade horizontálneho škálovania. Nevýhodou tohto prístupu je naopak to, že všetká práca s dátami a ich štruktúrou sa prenáša do vyšších vrstiev, o ktoré sa musí starať programátor. Naopak dokumentový a grafový model poskytuje bohatšiu štruktúru na prácu s dátami, ktorá spôsobuje komplikovanejšie škálovanie vhľadom k veľkosťi dát. Podľa odhadov spoločnosti Neotechnology[2] až 90% aplikáci, v prípade že sa nejedná o projekty spoločností Google, Amazon atď., spadá do rozmedzia kde sa veľkosť záznamov pohybuje rádovo v miliardách. Za zmienku stojí fakt, že aj napriek tomu, že tieto dátové modely sú si navzájom izomorfné, vhodnosť ich použitia závisí na konkrétnom príklade a požiadavkoch na aplikáciu.

[2] http://neotechnology.com/



Obr. 4.1: Pozícia dátového modelu z pohľadu škálovania podľa veľkosti a komplexnosti. Zdroj: [16]

4.2.3 Elastickosť

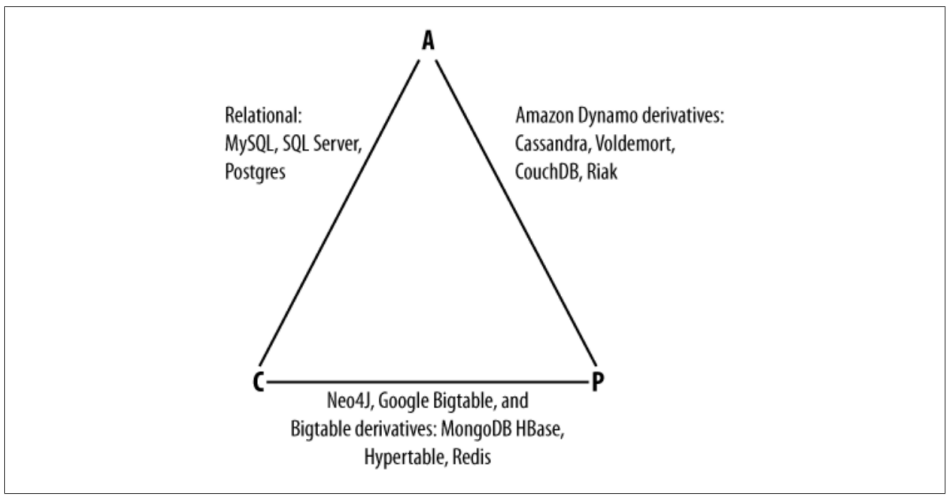
Elastickosť škálovania popisuje ako sa daný systém dokáže vysporiadať s pridaním alebo odobraním uzla. Určuje do akej miery je v tomto prípade potrebný manuálny zásah pre rovnomerné rozloženie záťaže, prípadná potreba reštartovania systému alebo zmena v užívateľskej aplikácii. Ideálne by pridávanie nových uzlov do systému malo zabezpečiť lineárne zvyšovanie výkonu u operácií ako je čítanie alebo zapisovanie dát.

4.2.4 Konzistencia dát

Poďla teórie CAP platí, že v prípade výskytu sieťových prerušení, ktorých prítomnosť v prostredí distribuovaných databázových systémov je prirodzená, nie je možné súčasne zaručiť vlasnosť konzistencie a dostupnosti. NoSQL systémy preto môžeme rozdeliť podľa tohto modelu do dvoch skupín, ktoré znázorňuje obrázok 4.2. Umiestnenie niektorých datázových systémov sa môže meniť na základe ich konfigurácie.

4.2.5 Prostredie behu

NoSQL systémy ďalej delíme podľa prostredia, v ktorom pracujú. Väčšina týchto open source systémov umožnuje ich nasadenie do vlastného prostredia teda privátnych cloudov alebo do verejného cloudového riešenia EC2 od spoločnosti Amazon. Mezdi tieto systémy patria



Obr. 4.2: Rozdelenie databázových systémov podľa CAP

napríklad Cassandra, HBase, Riak, Voldemort. Databázové systémy Amazon SimpleDB, Microsoft Azure SQL, Yahoo! YQL alebo Google App Engine sú poskytované ako komplexné cloudové riešenie. Rozhranie pre prácu s dátami a funkčnosť perzistentného úložiska zabezpečujú poskytovatelia týchto systémov.

4.2.6 Bezpečnosť

Distribuovaný databázový systém je možné nasadiť do cloudu. V prípade, použitia verejných cloudov môže hroziť nebezpečenstvo zneužitia dát treťou stranou a preto je potrebné aplikovať bezpečnostné mechanizmy. Pri nasadení systému do privátneho cloudu zasa môžeme požadovať viacero úrovní ochrany pre prístup k dátam. Systémy NoSQL disponujú minimálnymi prvkami pre zaručenie autentizácie a autorizácie. Mnoho systémov tieto mechanizmy vôbec neimplementuje poprípade ich implementácia je v začiatkoch.

4.3 Výber NoSQL systémov

V predchádzajúcej sekcii sme tieto distribuované databázové systémy rozdelili do štyroch hlavných kategórií na základe ich dátového modelu, ktorý je jedným z kľúčových faktorov pri výbere vhodného kandidáta podľa požiadavkov cieľovej aplikácie. Detailný popis a výkonnostné porovnanie NoSQL systémov, ktoré reprezentujú jednotlivé kategórie by boli nad rámec tejto práce. Paralelne s touto prácou vzniká diplomová práca, ktorá rieši podobný problém s využitím dokumentových databázových systémov [5], preto túto kategóriu pri výbere vynecháme.

Podľa analýzy požiadavkov na našu aplikáciu, ktoré sme popísali v predchádzajúcej kapitole nie je vhodné použitie databázových systémov s grafovým modelom a modelom typu kľúč-hodnota. Systémy s grafovým modelom sú určené na odlišnú úlohu problémov. V prípade použitia systémov s dátovým modelom kľúč-hodnota by sme komplexnosť riešenej

úlohy preniesli na úroveň klienta. Našim požiadavkom najlepšie vyhovuje stĺpcovo orientovaný model, ktorý využijeme v návrhu našej aplikácie. Hlavnou výhodou je, že tento model u systémov NoSQL nestanovuje žiadnu schému na dáta, ktoré budeme do systému vkládať, počet sĺpcov závisí od vstupných dát a ich počet je ľubovoľný. Pre potreby nášho riešenia preto použijeme stĺpcovo orientovaný NoSQL systém.

Stĺpcovo orientované NoSQL systémy

V tejto časti sa zameriame na vzájomné porovnanie systémov, ktoré poskytujú stĺpcovo orientovaný model. Medzi tieto open source systémy patria HBase, Cassandra a Hypertable. Napriek totožnému dátovemu modelu sú tieto systémy založené na rôznych architektonických princípoch a disponujú čiastočne odlišnými vlasnosťami. Tabuľka 4.1 zobrazuje stručný prehľad vlasností, na ktoré sme sa zamerali pri výbere víťaznej dvojice.

Vlastnosť	Databázový systém		
	HBase	Cassandra	Hypertable
Distribuovaný systém	áno	áno	áno
Architektúra	Bigtable	Dynamo	Bigtable
Klient	Thrift, REST	Thrift, Avro	Thrift, C++
Perzistentné úložisko	HDFS, AS3, KFS	LSS[3]	HDFS, KFS, LSS
SPOF	áno	nie	-[4]
Podpora viacerých datacentier	áno	áno	áno
Automatické rozdeľovanie dát	áno	áno	áno
Replikácia	áno	áno	áno
Konzistencia	CP	voliteľná	CP
Kompresia dát	LZO, GZIP	nie	LZO, ZLIB
Programovací jazyk	Java	Java	C++
MapReduce	áno	áno	áno
Indexy	-	+	-
Dokumentácia	+	++	-
Komunita	+	+	-

Tabuľka 4.1: Stručný prehľad vlasností stĺpcovo orientovaných systémov NoSQL

Zo stručného prehľadu v tabuľke je vidieť, že systémy obsahujú množstvo spoločných vlastností. Pri výbere sme zohľadnili aj ich praktické využitie spoločnosťami globálne pôsobiacimi na IT trhu. Napríklad systém Cassandra je používaný spoločnosťou Facebook v aplikácii pre vyhľadávanie a uchovávanie súkromnej pošty. Ďalšími dôležitými požiadavkami pre výber boli podpora komunity a kvalita dokumentácie. U systému Cassandra zohrala pri

výbere hlavnú rolu decentralizácia a možnosť voľby medzi konzistenciou a dostupnosťou. Zo zvolených kandidátov sme vybrali systém HBase a Cassandra. Dôvodom prečo sme zavrhli systém Hypertable je nedostačujúca dokumentácia, málo aktívna komunita a v čase výberu sa nachádzal vo verzii alfa. V následujúcich dvoch kapitolách detailne popíšeme oba systémy.

Počas písania tejto práce bol uvoľnený ďalší open source distribuovaný databázový systém implementujúci model Bigtable pod názvom Cloudata[5], ktorý sme už do naše práce nezahrnuli.

[5] http://www.cloudata.org

Kapitola 5

Cassandra

Distribuovaný databázový systém Cassandra bol vytvorený pre interné účely spoločnosti Facebook v roku 2007. Cassandra slúžila na vyhľadávanie v súkromnej pošte a poskytovala úložisko pre indexy vytvárané nad týmito dátami. Od systému sa predpokládalo, že bude obsluhovať miliardy zápisov denne, podporovať škálovanie podľa narástajúceho počtu používateľov, umožnovať beh na spotrebných počítačoch a podporovať replikáciu medzi geograficky oddelenými dátovými centrami. Ďalším požiadavkom bola vysoká dostupnosť, aby chyba žiadného uzlu nespôsobila celkovú nedostupnosť systému. Cassandra je decentralizovaný systém, kde každý uzol vykonáva tie isté operácie. V roku 2008 bola zverejnená ako open source projekt a je neustále vyvíjaná mnohými spoločnosťami a vývojármi. Tento systém využíva architektonické princípy distribuovaného databázového systému Dynamo [15] od spoločnosti Amazon a dátový model prevzal od distribuovaného databázového systému Bigtable vytvoreného spoločnosťou Google. V následujúcom texte popíšeme hlavne princípy, na ktorých je tento systém založený.

5.1 Dátový model

Cassandra k dátovému modelu systému Bigtable pridala štruktúru pod názvom *„super stĺpec“* (Super column). Základnou jednotkou dátového modelu je stĺpec. Stĺpec je tvorený názvom, hodnotou a časovým odtlačkom, ktorý využíva Cassandra pri riešení konfliktov súbežného zápisu. Skupina stĺpcov je identifikovná pomocou unikátneho kľúča a predstavuje riadok, počet a názvy stĺpcov nie je potrebné vopred definovať. Zoradené riadky podľa hodnoty kľučov a v nich zoradené stĺpce obaľuje štruktúra pod názvom *„rodina stĺpcov“* (Column family). Kľúče interne reprezentované ako reťazec znakov sú zároveň zotriedené. Názvy stĺpcov možu byť viacerých typov ako napríklad *ASCII*, *UTF-8*, bajtové pole a iné, podľa ktorých sú zotriedené. Je možné implementovať vlastnú metódu pre triedenie. Riadky obsiahnuté v jednej rodine stĺpcov sú na pevnom disku fyzicky umiestnené v jednom súbore typu *SSTable*. Je vhodné do rodiny stĺpcov ukládať relevantné záznamy, ku ktorým budeme pristupovať spoločne, čim sa vyhneme nadbytočným diskovým operáciam. V rámci jednej repliky sú operácie nad stĺpcami atomické. Operácie nad riadkom nevyužívajú zamykanie. Voliteľným príznakom, ktorý môžeme u stĺpca nastaviť je parameter TTL (Time to live), ktorý po uplynutí časového intervalu označí dáta príznakom pre zmazanie.

Štruktúra super stĺpec je špecialny typ stĺpca, ktorý je tvorenými obyčajnými stĺpcami. Stĺpec typu super má názov a jeho hodnota je tvorená zoznamom názvov obyčajných stĺpcov. Super stĺpce obaľuje štruktúra pod názvom *„super-rodina stĺpcov“* (Super column family). *Keyspace* združuje rodiny a super-rodiny stĺpcov. Definuje faktor a metódu replikácie, ktorá môže byť závislá poprípade nezávislá na sieťovej topológii. Na keyspace sa môžeme pozerať ako na databázu a rodiny stĺpcov môžeme prirovnať k tabuľkám v relačných databázových systémoch.

Aktuálna verzia Cassandry definuje maximálnu veľkosť dát 2 GB, ktoré je možné uložit do jedného stĺpca a stanovuje limit dve miliardy pre maximálny počet stĺpcov tvoriacih riadok.

5.2 Rozdeľovanie dát

Kľúčovým požiadavkom systému Cassandra je jeho schopnosť horizontálneho škálovania, čo vyžaduje pridávanie nových uzlov. Tento požiadavok vyžaduje mechanizmus, ktorý zabezpečí automatické rozdeľovanie dát medzi uzlami systému. Uvažujme príklad, kde máme k dispozícii jeden server obsahujúci veľké množstvo objektov, ku ktorým pristupujú klienti. Medzi server a klientov vložíme vrstvu kešovacích systémov, kde každý z týchto systémov bude zodpovedný za prístup k časti objektov nachádzajúcich sa na serveroch. Klient musí byť schopný určiť na základe hľadaného objektu, ku ktorém kešovaciemu systému. Predpokladajme, že klientom zabezpečíme výber jednotlivých kešovacím systémom pomocou lineárnej hašovacej funkcie $y = ax + b(mod\ \ p)$, kde p je počet kešovacích systémov. Pridanie nového kešovacieho systému alebo jeho zlyhanie bude mať katastrofálny dopad na funkčnosť systému. V prípade zmeny hodnoty p, bude každá položka odpovedať novej a zároveň chybnej lokácii. Tento problém rieši elegantne technika pod názvom *„úplné hašovanie“* (Consistent hashing) [31], ktorá sa využíva v distribuovaných systémoch pre určenie pozície pre prístup k distribuovaným hašovacími tabuľkami a využíva ju systém Cassandra.

Výstup hašovacej funkcie MD5 je reprezentovaný pomocou „kruhu“, kde v smere hodinových ručičiek postupujeme od minimálnej hodnoty hašovacej funkcie *0* k maximálnej 2^{127}. Každý uzlol v systéme má pridelenú hodnotu z tohoto rozsahu, ktorá určí jeho jednoznačnú pozíciu. Identifikácia uzlu v systéme, na ktorý sa uložia dáta reprezentované hodnotou kľúča sa vykoná aplikáciou hašovacej funkcie na dáta reprezentujúce kľúč. Na základe tejto hodnoty je jednoznačne určená pozícia v kruhu a v smere hodinových ručiek je vyhľadaný najbližší uzol, ktorému dáta prináležia. Výhodou tejto metódy je, že každý uzol je zodpovedný za hodnotu kľúčov, ktorých poloha sa nachádza medzi ním a jeho predchodcom. V prípade pridania nového uzlu alebo jeho odobratia, sa zmena mapovania kľúčov v kruhu prejaví len u jeho susedov. Táto technika prináša výhody, medzi ktoré patrí rovnomerná distribúcia dát a vyváženie záťaže. Dynamo tento problém rieši spôsobom kde každý uzol je zodpovedný za viacero pozícií na kruhu, takzvané virtuálne pozície. Cassandra využíva vlastné mechanizmy na monitorovanie záťaže uzlov a automaticky presúva ich pozície v kruh. Zároveň je možné explicitne stanoviť polohu každého uzla v kruhu pomocou zadania jeho identifikátora. Tento spôsob je vhodný v prípade, ak vieme predom určiť koľko uzlov bude systém obsahovať. V prípade zvyšovania počtu uzlov je možné tieto identifikátory zmeniť za behu systému. Identifikátor polohy uzlov je možné určiť pomocou následujúceho programu v jazyku Python, kde K je počet uzlov v systéme.

```
def tokens(n):
  r = []
  for x in xrange(n):
    r.append(2**127 / n * x)
  return r

print tokens(K)
```

5.3 Replikácia

S úplným hašovaním úzko súvisý replikácia. Každá dátová jednotka vložená do systému je replikovaná na N uzlov, kde počet N je voliteľne nastaviteľný pre daný keyspace. Každý uzol sa v prípade replikácie N $>$ 1 stáva koordinátorom, ktorý je zodpovedný za replikáciu dát, ktorých kľúč spadá do jeho rozsahu. Počas operácie zápisu koordinátor replikuje dáta na ďalších N - 1 uzlov. Cassandra podporuje viacero spôsobov pre umiestňovanie replík.

Jednoduchá stratégia

Táto stratégia umiestňuje replikú dát bez ohľadu na umiestnenie uzlov (serverov) v datacentre. Primárnu repliku spravuje uzol, do ktorého rozsahu spadajú ukladané dáta a ostanté repliky sú uložené na N - 1 susedov v smere hodinových ručičiek. Z toho vyplýva, že každý uzol je zodpovendný za dáta, ktorých kľúče spadajú do jeho rozsahu a súčasne dáta, ktoré spravuje jeho N predchodcov.

Sieťová stratégia

Pri tejto metóde a úrovni replikácie s hodnotou aspoň tri, je systém schopný zabezpečiť umiestnenie dvoch replík v rozdielnych *„rackoch“*[1] vrámci jedného datacentra. Tretia replika môže byť umiestnená do geograficky oddeleného datacentra. Táto stratégia je výhodna v prípade ak chceme použiť časť serverov na výpočty pomocou MapReduce a zvyšné dve repliky budú slúžiť na obsluhu reálnej prevádzky.

5.4 Členstvo uzlov v systéme

Distribuovaný systém musí byť schopný odolávať chybám ako porucha uzlov alebo sieťové prerušenia. Decentralizácia a detekcia chýb využíva mechanizmy založené na *gosship protokoloch* [20]. Tieto protokoly slúžia pre vzájomnú komunikáciu uzlov vymnieňajúcich si navzájom doležité informácie o svojom stave. Periodicky v sekundových intervaloch každý uzol kontaktuje náhodne vybraný uzol, kde si overí či je tento uzol dostupný. Detekcia nedostupnosti uzla je realizovaná algoritmom s názvom Accrual Failure Detector [29].

Pridávanie novýchu uzlov, presun uzlov v rámci kruhu sa taktiež dejú pomocou Gosship protokolu, ktorý ďalej zabezpečuje, že každý uzol obsahuje informácie o tom, ktorý uzol je

[1] TODO popis rack

zodpovedný za daný rozsah kľúčov v kruhu. Ak sa vykonáva operácia čítania alebo zápisu dát na uzol, ktorý nie je zodpovedný za tento kľúč, dáta sú automaticky preposlané na správny uzol, ktorého výber je zabezpečený s časovou zložitosťou *O(1)*.

5.5 Zápis dát

Tento systém bol primárne navrhnutý tak aby spracúval vysoký tok dát pre zápis. Ak uzol obrží požiadaok pre zápis, dáta sú zapísané do štruktúry pod názvom *Commitlog*, ktorá je uložená na lokálnom súborovom systéme a zabezpečuje trvácnosť dát. Zápis do tejto štruktúry je vykonaný sekvenčne, čo umožnuje dosiahnúť vysokú priepusnosť bez nutnosti vystavovania diskových hlavičiek (disk seek operations). Dáta sú následne zoradené a nahrané do štruktúry pod názvom *Memtable*, ktorej obsah je uložený v operačnej pamati. V prípade, že by tento zápis zlyhal alebo by došlo k neočakávanému pádu inštancie Cassandry obnovenie obsahu týchto štruktúr je možné vďaka Commitlogu. Po dosiahnutí predom stanovej hodnoty, ktorá určuje maximálny počet dát uložených v Memtable sú tieto štruktúry zapísané do súborov pod názvom *SSTable* (Sorted String Tables), ktorých obsah je zoradený podľa hodnoty kľúčov a nie je možné ich modifikovať. Súbory SSTables sa zlievajú (merge sort) v pravidelných intervaloch na pozadí a táto operácia je neblokujúca. Počas zlievania SSTables dochádza k odstraňovaniu dát určených na vymazanie a generovaniu nových indexov. Indexy slúžia na rýchly prístup k dátam uloženým v SSTable. Zároveň sa generujú štruktúry pod názvom *Bloom filters*[2], ktoré sa využívajú pri čítaní dát. Požiadavok pre zápis dát je možné zaslať na ľubovoľný uzol v klastri.

5.6 Čítanie dát

V prípade požiadavku na načitanie dát, sa požadované dáta najprv hľadajú v štruktúrach Memtable. Ak sa dané data nenachádzajú v operačnej pamati, vyhľadávanie sa uskutočnuje podľa kľúča v súboroch SSTable. Kedže snahou systému, je čo najefektívnejšie vyhľadávanie, využívajú sa bloom filtre. Bloom filtre sú nedeterministické algoritmy, ktoré dokážu otestovať či hľadaný prvok patrí do množiny a generujú len falošné pozitíva. Pomocou nich je možné priradiť kľúče uložené v súboroch SSTables do bitových polí, ktoré je možné uchovať v operačnej pamati. Vďaka tomu sa výrazne redukuje prístup na disk. Požiadavok pre čítanie dát možeme zaslať na ľubovoľný uzol.

5.7 Zmazanie dát

Počas vykonania operácie reprezentujúcej zmazanie dát sa tieto dáta nevymažú okamžite. Namiesto toho sa vykoná operácia, ktorá dané dáta označkuje príznakom pod názvom *tombstone*. Po uplynutí doby, ktorá je štandardne nastavená na desať dní, sa tieto dáta odstránia pri procese zlievajúcom súbory SSTables.

[2] http://en.wikipedia.org/wiki/Bloom_filter

5.8 Konzistencia

Konzistencia systému je maximálne konfigurovateľná a využíva princípy techník založených na protokoch kvóra. Klient určuje hodnotu R, ktorá stanovuje počet replík pre čítanie dát. Operácia je úspešná v prípade dostupnosti týchto replík. To samé platí pre zápis dát, kde počet replík je určený parametrom W. Ak je splnený vzťah R + W > N, kde N je počet replík tak výsledok operácie spĺňa definíciu silnej konzistencie, naopak v prípade vzťahu R + W < N, sa jedná o čiastočnú konzistenciu čím zaručíme vysokú dostupnosť.

5.9 Perzistentné úložisko

Cassandra využíva ako perzistentné úložisko dát lokálny súborový systém.

5.10 Bezpečnosť

Implicitne Cassandra nevyužíva žiadne prvky zabezpečujúce bezpečnostné mechanizmy. K dispozícii je modul, ktorý umožnuje nastavenie auntentizácie na úrovni Keyspace-u pomocou textových hesiel alebo ich MD5 odtlačkov. Obmedzovanie prístupu k datám je preto potrebné zabezpečiť na aplikačnej úrovni.

Kapitola 6

HBase

V úvode tejto kapitoly stručne popíšeme distribuovaný súborový systém, ktorý je súčasťou projektu Hadoop a zároveň slúži ako perzistentné úložisko pre distribuovanú databázu HBase. Následne popíšeme základné princípy fungovania tohoto databázového systému.

Hadoop

Hadoop[1] je open source projekt vytvorený v programovacom jazyku Java, ktorý tvorý distribuovaný súborový systém Hadoop Distributed Filesystem (HDFS) a framework MapReduce pre spracúvanie objemu dát v desiatkách PB [38]. Medzi hlavné vlastnosti tohoto systému patria vysoká dostupnosť, škálovateľnosť a distribuovaný výpočet. Architektúra HDFS vychádza z princípov distribuovaného súborového systému Google File System (GFS) [24] a pôvod frameworku MapReduce [14] pochádza taktiež od spoločnosti Google.

Súborový systém využíva architektúru master-slave. HDFS predpokláda prácu so súbormi rádovo v desiatkách gigabajtov, ktoré sú interne reprezentované dátovými blokmi o štandardnej veľkosti 65 MB. Informácie o priradení blokov k súborom a ich umiestnenie na uzloch typu slave sú reprezentované pomocou metadát. Tieto metadáta sú uložené v operačnej pamäti uzla typu master, ktorý sa nazývana *Namenode*. Uzly slave pod názvom *Datanode* slúžia ako fyzické úložisko dátových blokov, ktoré sú zároveň na týchto uzloch replikované. Štandardne je nastavená úroveň replikácie na hodnotu tri, každý blok je v systéme uložený trikrát.

Predtým ako klient vykoná operáciu zápisu alebo čítania dát, tak kontaktuje uzol Namenode, ktorý mu poskytne informácie, na ktorých uzloch sa nachádzajú bloky reprezentujúce súbor a dátová komunikácia následne prebehne medzi klientom a uzlami typu Datanode. HDFS je optimalizovaný pre jednorázový zápis dát a ich následné mnohonásobné čítanie.

Hlavným nedostatkom tejto infraštruktúry je fakt, že uzlol Namenode tvorí kritický bod výpadku, v prípade jeho nedostupnosti nie je možné pracovať s HDFS. Prípadná strata alebo poškodenie dát na tomto uzle spôsobí totálne zlyhanie súborového systému bez možnosti jeho obnovy. Súborový systém nie je vhodný pre ukladanie veľkého počtu malých súborov. Uzol Namenode alokuje pre objekt typu blok a objekt typu súbor 300 B metadát. V prípade

[1] http://hadoop.apache.org/

uloženia súboru, ktorý nepresahuje veľkosť jedného bloku je potrebné alokovať 300 B dát. Ak uložíme 10,000,000 takýchto súborov veľkosť metadát, ktoré udržiava Namenode v operačnej pamati zaberie 3 GB. Celkový počet uložených súborov je obmedzený veľkosťou operačnej pamati RAM, ktorou disponuje uzol Namenode.

Z týchto pozorovaní vyplýva fakt, že distribuovaný súborový systém HDFS nemá praktické využitie ako úložisko dát slúžiace k archivácii emailových správ, pre ktoré sme zadefinovali požiadavky v tretej kapitole.

HBase

Aplikácie HDFS a MapReduce slúžia na dávkové spracúvanie obrovského objemu dát. HBase je open source, distribuovaný, stĺpcovo orientovaný databázový systém, ktorý umožnuje prácu s veľkým objemom dát v reálnom čase. Ako perzistentné úložisko dát využíva distribuovaný súborový systém HDFS, je taktiež implementovaný v Jave a jeho architektornické koncepty vychádzajú z článku Bigtable od spoločnosti Google. HBase bol vytvorený spoločnosťou Powerset na konci roka 2006, pre potreby spracúvania obrovského objemu dát a začiatkom roka 2008 sa stal oficiálnym podprojektom systému Hadoop [43].

6.1 Dátový model

Dátový model je totožný s konceptom Bigtable. Dáta s ktorými pracuje HBase sa ukladajú do tabuliek. Každá tabuľka obsahuje riadky identifikované kľúčom, ktoré môžu byť tvorené ľubovoľným počtom stĺpcov. Klúče sú reprezentované ako bajtové pole. Pre zápis ich hodnoty môžeme použiť ľubovoľný typ dát, napríklad reťazec alebo serializovanú dátovú štruktúru pomocou JSON, XML. Riadky sú radené podľa názvov kľúčov, ktoré sú radené podľa hodnoty jednotlivých bajtov. Stĺpce sú organizované do skupín, ktoré sa nazývajú „rodiny stĺpcov“ (Column families). Obsahu každej bunky, ktorej pozíciu určuje riadok a stĺpec prináleži verzia, ktorá je reprezentovaná časovou značkou a jej obsah je reprezentovaný ako pole bajtov. Časovú značku určuje klient pri zápise dát alebo je automaticky generovaná systémom HBase (reprezentuje ju systémový čas). Obsah buniek tabuľky je možné sprístupniť pomocou kľúča a názvu stĺpca alebo pomocou kľúča, názvu stĺpca a časovej značky. V prípade uloženia viacerých verzií v danej bunke, sú tieto dáta radené od najaktuálnejšej časovej značky po najstaršiu. Región tvorí interval riadkov, kde posledný riadok do daného intervalu nepatrí.

Dôležitým faktom je, že stĺpce, ktoré patria do rovnakej rodiny stĺpcov sú fyzicky uložené na tom istom mieste. Rodiny stĺpcov je potrebné zadefinovať počas vytvárania tabuliek. Ich názvy a počet je potrebné vhodne premyslieť už pri samotnom návrhu databázovej schémy. V prípade, že klaster HBase obsahuje viacero uzlov je na nich potrebné zabezpečiť sychronyzáciu systémového času. V prípade veľkého časového rozdielu na jednotlivých uzloch kastra hrozí, že sa daná inštancia systému HBase sa nespustí.

6.2 Architektúra systému

HBase využíva architektúru master-slave. Uzol v role master sa nazýva *HBaseMaster*, uzly slave *RegionServers* (RS). Pre chod systému je ďalej potrebná služba Zookeeper[2], ktorá zabezpečuje ...

HBaseMaster

Tento uzol v systéme vykonáva priradzovanie regiónov RegionServer-om, detekujem pridanie nového RS, jeho zlyhanie a rovnomerne rozdeľuje záťaž na RS-och v prípade rozdelenia regiónu.

RegionServer

RegionServer sĺúži pre obsluhu klientských požiadavkov, samotný zápis alebo čítanie dát sa vykonáva medzi klientom a RS. Každý RS spravuje niekoľko regiónov, automaticky rozdeľuje regióny a informuje o tom uzol HBase Master. Tieto uzly môžeme v systéme ľubovoľne pridávať alebo odoberať, počas jeho prevádzky.

6.3 Rozdeľovanie dát

Základnou jednotkou, ktorá zabezpečuje rozdeľovanie dát a teda umožnuje horizontálne škálovanie a rovnomerné rozloženie záťaže v klastri je region. Region má náhodne vygenerovaný identifikátor. Tabuľka je tvorená regiónmi, pri jej prvotnom vytvorení ju zvyčajne reprezentuje jeden región. V prípade, že jej veľkosť dosiahne predom stanovenú hranicu (závislé na konfigurácii), dojde k rozdeleniu riadkov do dvoch nových regiónov s podobnou veľkosťou. Tieto regióny sú v klastri distribuované na uzly typu RegionServer. Tento mechanizmus zabezpečuje, že do systému je možné uložiť tabuľku o veľkosti, ktorú by nebolo možné uložiť alebo spracovať pomocou jedného fyzického počítača z dôvodu hardverových limitov. Región je základný element, ktorý zabezpečuje vysokú dostpnosť a rovnomerné rozloženie záťaže.

6.4 Replikácia

Primárnu replikáciu dát, ktorá spĺňa vlastnosti silnej konzistencie a zabraňuje stráte dát zabezpečuje perzistentné úložisko, v tomto prípade HDFS.

HBase podporuje replikáciu v rámci viacerých geograficky oddelených datacentier. Replikácia funguje na rovnakom princípe ako v databázovom systméme MySQL[3], teda master-slave a je asynchrónna. Táto forma replikácie zanáša do distribuovaného systému čiastočnú konzistenciu, pretože negarantuje, že dáta budú zmenené na všetkých uzloch slave v rovnakom čase.

[2] http://zookeeper.apache.org/
[3] http://dev.mysql.com/doc/refman/5.1/en/replication-formats.html

6.5 Perzistentné úložisko

Tento distribuovaný databázový systém je schopný pracovať v lokálnom móde, kde vyššie spomínané komponenty bežia ako samostatné služby na jednom fyzickom uzle a ako úložisko dát sa využíva lokálny súborový systém. V prípade distribuovaného módu rozlišujeme dva typy:

- pseudo-distribuovaný mód, kde všetky komponenty bežia na jednom uzle
- distribuovaný mód, jednotlivé komponenty bežia na samostatných uzloch

Obidve konfigurácie môžu využívať ako perzistentné úložisko distribuovaný súborový systém HDFS, Kosmos Distributed File System (KFS) alebo Amazon S3. Štandardne sa doporučuje použitie HDFS.

6.6 Konzistencia

Systém sa vyznačuje silnou konistenciou. Z modelu CAP splňuje CP, teda uprednosňuje konzistenciu pred dostupnosťou.

6.7 Zápis dát a čítanie dát

V prípade zápisu alebo čítania dát klient kontaktuje Zookeeper, od ktorého obrží informáciu o lokácii tabuľky *-ROOT-* a následne kontaktuje daný RS obsahujúci túto tabuľku. Z tabuľky -ROOT- sa určí RegionServer, ktorý obsahuje tabuľku *.META.*, tieto kroky sa lokálne kešujú na strane klienta. Následne klient kontaktuje daný RegionServer a v tabuľke .META. vyhľadá uzol obsahujúci cieľový región, do ktorého patria hľadané alebo cieľové dáta. V poslednom kroku prebieha celá dátová komunikácia medzi klientom a posledným vyhľadaným uzlom.

Dáta sú zapísané do štruktúry *HLog* (typu *Write-Ahead-Log*), ktorá je uložená v súborovom systéme HDFS. Po potvrdení o úspešnom zápise sú data nahrané do štruktúry *MemStore*, ktorá je uschovaná v operačnej pamäti. Štruktúry MemStore sú zapisované obdobne ako u systému Cassandra do zoradených súborov typu *HFile*, ktorých štruktúra je podobná ako u súborov SSTable. V prípade, že RegionServer obrdží požiadavok na čítanie dát, dáta sú načítane buď zo štruktúry MemStore alebo HFile. Pre zvýšenie výkonnosti je možné použiť Bloom filtre.

Zápis alebo čítanie dát na úrovni riadku identifikovaného pomocou kľúča je atomická operácia.

6.8 Zmazanie dát

Pri operácií zmazania dát je možné určiť vymazávané data konkrétnou časovou značkou, poprípade je možné vymazanie všetkých verzií dát, ktoré sú staršie ako zadaná časová značka.

Zmena sa nevykoná okamžite z dôvodu, že obsah štruktúr HFile nie je možné modifikovať (jedným z dôvodov je aby sa zabránilo vykonávaniu nadbytočných diskových operácií). Namiesto toho sa vykoná operácia, ktorá dané dáta označkuje príznakom pod názvom *tombstone*. Tieto dáta sa odstránia obdobne ako v systéme Cassandra, pri procese zlievajúcom súbory HFile.

6.9 Bezpečnosť

Hadoop a HBase aktuálne poskytujú prvok autentizácie pomocou služby Kerberos. Možnosť pridania autorizácie na úrovni tabuliek a rodiny stĺpcov je neustále vo vývoji [42].

Kapitola 7

Testovanie výkonnosti

Výkonové porovnanie NoSQL systémov je zložitá úloha, neexistuje univerzálny nástroj, ktorým by bolo možné tieto systémy navzájom porovnať. Každý z týchto systémov sa vyznačuje špecifickými vlastnosťami medzi ktoré patria typ konzistencie, optimalizácia pre zápis alebo čítanie a ich výber závisí na konkrétnom prípade použitia. Všeobecný nástroj pre ich porovnanie by preto nemal žiadne opodstatnenie. Aktuálne nie sú k dispozíci žiadne všeobecné nástroje, ktorými by bolo možné testovať napríklad konzistenciu, spoľahlivosť a iné vlastnosti. Výkon týchto systémov ovplyvňuje hodnota replikácie, spôsob rozdeľovania dát a typ konzistencie. Veľmi častou a zároveň časovo náročnou metódou, ktorá slúži na porovnávanie týchto systémov je implementácia daného riešenia s využitím všetkých porovnávaných systémov. V tejto kapitole sa zameriame na popis testov, ktoré sme vykonali v reálnych podmienkach.

7.1 Testovacie prostredie

Pre výkonnostné testovanie sme mali k dispozícii 9 počítačov s rovnakou hardvérovou a softvérovou konfiguráciou, ktoré boli navzájom prepojené pomocou 10 Gbit smerovača a komunikovali po 1 Gbit linke. Konkrétnu softvarovú konfiguráciu testovaných aplikácií popíšeme jednotlivo v nasledujúcich podkapitolách.

Hardvérová konfigurácia

- 4 jádrový procesor Intel, 5506@2.13Ghz
- 4 GB RAM
- 5 pevných diskov (SATA, 7200RPM) o veľkosti 1TB zapojených v poli RAID0
- 1 Gbit sieťová karta

Softvérová konfigurácia

Každý uzol obsahoval inštaláciu operačného systém Debian GNU/Linux Lenny x64 a aplikáciu Sun Java JDK 1.6.0_+88. Na každom uzle bol deaktivovaný odkladací priestor (swap). Za účelom monitorovania bol použitý softvér Zabbix a nástroje VisualVM, htop, iostat a dstat.

Sieťová konfigurácia

Hodnota maximálnej reálnej sieťovej priepustnosti medzi dvoma uzlami bola zmeraná pomocou aplikácie nuttcp[1] s výslednou hodnotou 940 Mbit.

7.2 Popis testovacej metodológie

Nad oboma distribuovanými databázovými systémami sme vykonali testy na základe, ktorých sme pozorovali ako tieto systémy ovplyvňuje rôzna hodnota replikácie, typ konzistencie, pozorovali sme ich schopnosť horizontálneho škálovania a chovanie sa pod záťažou. Testy boli zamerané na operácie zápisu dát, ktorý je kritickým prvkom pre potreby našej aplikácie.

7.2.1 Testovací klient

Testovacím klientom bola aplikácia využivajúca vlákna, založená na princípe producent - konzument, kde konzumenti predstavovali jednotlivé vlákna vykonávajúce zápis alebo čítanie dát. Optimálny počet paralelne zapisujúcich vlákien bol stanovený na hodnotu 50, pri ich navýšení sa zvyšovala hodnota latencie a nedošlo k zvýšeniu dátového toku pre zápis. Pre čítanie dát bolo použitých 250 vlákien. Dôležitým bodom bolo zabezpečiť rovnaké zaťaženie každého uzla v klastri, počas celého priebehu jednotlivých testov. Detailný popis splňujúci tento bod je obsiahnutý v časti popisujúcej test konkrétneho databázového systému.

7.2.2 Testovací prípad pre zápis dát

V tomto testovacom prípade sme postupne do klastra obsahujúceho jeden, tri a šesť uzlov zapisovali 8,000,000 riadkov. Každý riadok obsahoval jeden stĺpec o veľkosti 1000 B, ktorého obsah tvorili náhodne vygenerované dáta. Tento počet zapisovaných riadkov sme zvolili z dôvodu, aby dochádzalo k zápisu štruktúr Memtable a Memstore na pevný disk. Vďaka týmto operáciam sa chovanie klastra viac priblížilo reálnym podmienkám.

7.2.3 Testovací prípad pre čítanie dát

Dôvodom tohto testovacieho prípadu bolo určiť priepustnosť pri čítaní dát z databázového systému. Táto hodnota bude mať výrazný vplyv na celkovú dobu trvania výpočtov štatistík, pomocou MapReduce. V tomto testovacom prípade sme náhodne načítali 1,000,000 riadkov z databázového systému o veľkosti 1000 B. Počet uzlov tvoriacích klaster bol jeden, tri a šesť.

[1] http://www.wcisd.hpc.mil/nuttcp/

7.2.4 Zaťažovací test

Cieľom bolo zistiť stabilitu klastra v prípade, keď bude pod sústavným zápisom dát, budú v ňom prebiehať operácie pre zlievanie (compaction) štruktúr SSTable a HTable na pevnom disku. Tento test sme vykonali po dobu piatich hodín. Náhodne generované dáta o rôznej veľkosti boli zapisované do jedného stĺpca. Počas niektorých testovacích prípadov sme použili dvoch klientov z dôvodu aby sme zaručili maximálnu saturáciu šírky prenosového pásma a vylúčili úzke hrdlo na strane klienta (1 Gbit linka umožňuje maximálny teoretický dátový tok 125 MB/s).

7.3 HDFS

Nad distribuovaným súborovým systémom HDFS sme vykonali testy určujúce maximálnu hodnotu priepustnosti pri zápise dát, z dôvodu aby sme vylúčili možné úzke hrdlo v jeho prepojení s databázovým systémom HBase. Pre účely testovania sme použili verziu Hadoop-0.20.2, veľkosť haldy pre JVM (Java Virtual Machine) bola nastavená na hodnotu 1 GB.

Meranie sme vykonali v troch konfiguráciach. Každá konfigurácia obsahovala jeden uzol v role master, na ktorom boli spustené služby Namenode a JobTracker. Na zvyšných uzloch typu slave bežali služby Datanode a Tasktracker. Konfigurácia klastra bola následovná:

- A - tri uzly slave s faktorom replikácie jedna
- B - tri uzly slave s faktorom replikácie tri
- C - šesť uzlov slave s faktorom replikácie tri

Testy boli vykonané nástrojom TestDFSIO, ktorý je súčasťou zdrojových kódov systému Hadoop. Pomocou techniky MapReduce, boli dáta do súborového systému zapisované jednou funkciou typu map. Počas jednotlivých testov sme na HDFS zapisovali tri rôzne veľkosti súborov a bola zachovaná štandardná veľkosť bloku 65 MB. Každý test bol vykonnaný trikrát a výsledná hodnota bola určená ako aritmetický priemer. Výsledky testu, ktoré zobrazuje tabuľka 7.1, reprezentujú maximálny tok pre zápis dát v klastri.

~~Veľkosť súboru [MB]~~	Klaster		
	A	B	C
65	287 MB/s	102 MB/s	190 MB/s
512	371 MB/s	85 MB/s	162 MB/s
2048	433 MB/s	85 MB/s	163 MB/s

Tabuľka 7.1: Priepustnosť pri zápise dát na HDFS

Na základe týchto hodnôt pozorujeme, že zvýšenie hodnoty replikácie ma zásadný negatívny vplyv na celkový výkon distribuovaného súborového systému. Dôležitý fakt, ktorý vyplynul z výsledkov testovania je, že v prípade ak zvýšime dvojnásobne počet uzlov v klastri (prípad klastrov v konfigurácii B a C) jeho výkonnosť vzrastie lineárne, čo potvrdzuje vysokú škálovateľnosť systému.

7.4 HBase

Pre test distribuovaného databázového systému sme nainštalovali systém Hadoop 0.20.2 a HBase 0.90.1. Na jednom fyzickom uzle bežali následujúce služby:

- HBase Master
- Zookeeper
- Namenode

Tieto služby sú v oboch systémoch súčasťou uzla master. Na zvyšných uzloch boli spustené služby RegionServer a Datanode. Veľkosť haldy pre JVM sme v prípade systému HDFS nastavili na 1 GB operačnej pamäti a v prípade systému HBase na 2 GB.

Prázdna tabuľka je po vytvorení v systéme HBase reprezentovaná jedným regiónom. Tento región je uložený na jednom uzle. K rozdeleniu tohto regiónu dochádza v prípade ak objem dát zapisaných v tabuľke prekročí štandardne nastavenú hranicu s hodnotou 256 MB. Prázdnu tabuľku sme vytvorili pomocou programového rozhrania (API) systému HBase a to tak, že sme ju predrozdelili na počet regiónov, ktorý odpovedal počtu uzlov typu slave v klastri. Názvy kľúčov sme generovali pomocou náhodného generátora. Vďaka tejto metóde sme dosiahli rovnomerné zaťaženie všetkých uzlov po celú dobu testovania. Tabuľka 7.2 zobrazuje výsledky meraní podľa testovacieho prípadu pre zápis dát.

Počet uzlov	Replikácia	Čas	Riadok/sek	Priepustnosť [MB/s]
1	1	551	14519	14
3	1	202	39613	39
3	3	317	25110	25
6	3	211	37864	37

Tabuľka 7.2: Hbase: zápis riadkov o veľkosti 1000 B

Škálovateľnosť

Z daných meraní vidíme, že systém je maximálne škálovateľný a dvojnásobne zvýšenie počtu uzlov zvýši priepusnosť o cca 50%.

Replikácia

Zvýšenie hodnoty replikácie (riadky 3,4) spôsobilo zníženie prenosovej rýchlosti o cca 37%.

Konzistencia

HBase vyžaduje silnú konzistenciu pre operácie čítania a zápisu dát, na úkor dostupnosti. Tento fakt sme zaznamenali počas testovania, keď v určitých intervaloch došlo k zlyhaniu operácie zápisu, ktorá bola následne zopakovaná.

Čítanie dát

Pri vytváraní tabuľky v systéme HBase chýba automatická podpora Bloom filtrov, ktoré sú neusale vo vývoji. Aktiváciu týchto filtrov je potrebné vykonať z príkazového interpreta, ktorý slúži pre manipuláciu so systémom HBase. Bloom filtre boli počas testu aktivované. Výsledky testovacieho prípadu pre čítanie dát zaznamenáva tabuľka 7.3.

Počet uzlov	Replikácia	Čas	Riadok/sek	Priepustnosť [MB/s]
1	1	246	4069	4
3	1	117	8561	8
3	3	127	7936	8
6	3	190	11904	12

Tabuľka 7.3: HBase: čítanie riadkov o veľkosti 1000 B

Zaťažovací test

V tabuľke 7.4 sú znázornené výsledky zaťazovacieho testu.

	Počet uzlov			
Veľkost riadku	3	4	5	6
1 KB	32	35	36	38
10 KB	31	37	41	49
100 KB	35	43	52	55
512 KB	25	40	51	63
1 MB	35	47	53	68

Tabuľka 7.4: Hbase: maximálna priepustnosť klastru v MB/s

7.5 Cassandra

Pri testovaní klastru bol použitý distribuovaný databázový systém Cassandra verzie 0.7.3. Adresár obsahujúci súbory typu commitlog bol na samostatnom fyzickom disku, dátový adresár bol na diskoch zapojených v poli RAID0. Veľkosť štruktúry Memtable bola nastavená na hodnotu 120 MB. Každý uzol zaberal na hašovacom kruhu rovnaký, predom nastavený úsek.

Tabuľka 7.5 obsahuje výsledky z viacerých meraní, podľa testovacieho prípadu pre zápis dát. Ako hodnoty kľúčov boli pre jednotlivé zapisované riadky použité prirodzené čísla z rozsahu nula až celkový počet riadkov. Cassandru sme nastavili tak, aby boli jednotlivé kľúče a k ním prinaležiace dáta, náhodne zapisované na jednotlivé uzly systému, bol použitý *RandomPartitioner*. Toto nastavenie zabezpečilo rovnomernú záťaž každého uzla počas celkovej doby zápisu. Výsledok hašovacej funkcie MD5 aplikovaný na hodnotu kľúča určil uzol, do ktorého boli zapísané dáta.

Škálovateľnosť

Z výsledkov meraní je vidieť, že tento distribuovaný databázový systém je maximálne škálovateľný z pohľadu rýchlosti zápisu. V prípade, keď sme zdvojnásobili počet uzlov z troch na šesť (riadok 3,5) vzrástla priepusnosť zápisu o cca 50%.

Replikácia

V prípade zvýšenia hodnoty replikácie z 1 na 3 sa automaticky znížila rýchlosť zápisu o jednu tretinu.

Konzistencia

Počas zápisu s konzistenciou kvóra, ktorá zabezpečuje silnú konzistenciu databázového systému, sa rýchlosť znížila podľa očakávaní. V tomto prípade, aby klient obdržal odpoveď o úspešnom zápise museli byt dáta zapísané na celkový počet replík $N/2 + 1$, kde N označuje počet uzlov v klastri. Počas zápisu v prípade konzistencie One, klient obdržal potvrdenie o úspešnosti po zápise na jeden uzol.

Čítanie dát

Výsledky testovacieho prípadu pre čítanie dát sú zaznamenané v tabuľke 7.6. Riadky boli čítané v náhodnom poradí a kešovanie kľúčov a riadkov, ktoré Cassandra podporuje bolo vypnuté. Počas tejto operácie boli automaticky aplikované Bloom filtre.

Zaťažovací test

Výsledky zaťažovacieho testu zobrazuje tabuľka 7.7, kde jednotlivé položky zobrazujú priemerný dátový tok počas doby testovania v MB/s. Počas testu sme klaster monitorovali a zistil viacero závažných dôsledkov. Na všetkých uzloch prebiehali veľmi časté GC kolekcie (Garbage collections) z dôvodu častého zápisu štruktúr Memtable na pevný disk. Podľa hardverovej špecifikácie pre systém Cassandra všetky uzly disponovali minimálnou veľkosťou operačnej pamäti 4 GB. Z dôvodu zabezpečenia stability systému bola horná hodnota pri ktorej sa zapisuje Memtable z operačnej pamäti na disk 120 MB. Následkom tohto nastavenia vznikalo veľké množstvo SSTable súborov na pevnom disku, ktoré Cassandra zlievala na pozadí (compactions), čo spôsobovalo záťaž vstupno výstupných operácií (I/O wait). V prípade, takto zaťaženého systému a veľkého množstva SSTable súborov by bola operácia čítania veľmi pomalá, pretože dáta patriace do jednej rodiny stĺpcov by boli uložené vo veľkom množstve samostatných súborov a ich načítanie by vyžadovalo zvýšené množstvo diskových operácií (seek).

Z tohoto testu ďalej vyplynulo pozorovanie, že v prípade zápisu malých súborov rádovo v kB, je hlavným úzkym hrdlom systému CPU, kdežto v prípade zápisu veľkých blokov dát sú to V/V diskové operácie.

Počet uzlov	Replikácia	Konzistencia	Čas	Riadok/sek	Priepustnosť [MB/s]
1	1	One	338	23669	23
3	1	One	207	38647	38
3	3	One	311	25723	25
3	3	Quorum	351	22792	22
6	3	One	202	39604	39
6	3	Quorum	263	30418	30

Tabuľka 7.5: Cassandra: Zápis riadkov o veľkosti 1000 B

Počet uzlov	Replikácia	Konzistencia	Čas	Riadok/sek	Priepustnosť [MB/s]
1	1	One	383	261	2.5
3	1	One	211	4745	4.6
3	3	One	51	19630	19
3	3	Quorum	103	9750	10
6	3	One	32	30903	30
6	3	Quorum	59	16924	17

Tabuľka 7.6: Cassandra: Čítanie riadkov o veľkosti 1000 B

	Počet uzlov			
Veľkost riadku	3	4	5	6
1 KB	18	28	30	33
10 KB	63	77	93	118
100 KB	71	92	111	134
512 KB	67	87	109	127
1 MB	62	92	100	126

Tabuľka 7.7: Cassandra: maximálna priepustnosť klastru v MB/s

7.6 Voľba databázového systému

Primárnym požiadavkom aplikácie je zvládať vysoký tok pre zápis a nízkonákladová administrácia. V oboch týchto prípadoch prevládajú vlastnosti systému Cassandra nad systémom HBase. Systém HBase má okrem iného zložitú infraštruktúru, ktorá je závislá na externých službách ako HDFS a Zookeeper. Naproti tomu decentralizácia systému Cassandra prináša vďaka svojej jednoduchosti mnoho výhod. Aj napriek tomu, že systém HBase neobsahuje kritický bod výpadku, čo zabezpečuje služba Zookeeper, problematickým článkom je perzistentné úložisko HDFS, ktoré tento požiadavok nespĺňa. Ďalšou výhodou Cassandry je podpora indexov, ktoré sa vytvárajú asynchrónne na pozadí a nie je potrebné na aplikačnej úrovni zabezpečovať ich správu pomocou samostatnej rodiny stĺpcov.

Cassandra dosahovala vyššiu priepustnosť pre zápis dát, ktorú sme zmerali v záťažových testoch s obmedzeným počtom uzlov, ktoré sme mali k dispozícii. V prípade použia šiestich uzlov sme prekročili hranicu 125 MB/s, teda sme boli schopný zápisu šírky pásma 1 Gbit, čo splňuje požiadavku na náš systém.

Domnievame sa, že jedným z možných obmedzení výkonnosti systému HBase môže byť fakt, že štruktúra HLog, ktorá zabezpečuje trvácnosť dát sa spolu so štruktúrami MemStore zapisuje do súborového systému HDFS, čo spôsobuje vyššiu záťaž na strane diskových operácií. Záťaž fyzického disku je dvojnásobná v porovnaní so systémom Cassandra. Cassandra tento problém rieši efektívne, pretože štruktúra commitlog, zaručujúca trvácnosť dát, môže byť uložená na fyzicky odlišnom pevnom disku, ako disk na ktorý sa následne zapisujú súbory SSTable. Systém HBase sa tento nedostatok snaží riešiť napríklad spôsobom, kde je možné pomocou programového rozhrania pri zápise dát nastaviť aby tieto dáta neboli zapisované do štruktúry HLog. V tomto prípade hrozí strata dát a systém nezaručuje vlastnosť trvácnosti (durability).

TODO: z dovodu odlahcenia = zataze read s CL . one

Kapitola 8

Návrh systému

V tejto kapitole popíšeme návrh systému, ktorý bude slúžiť na archiváciu emailov a spĺňať požiadavky, ktoré sme pre tento systém zadefinovali. V prvej časti sa zameriame na popis štruktúry a zberu dát, ďalej navrhneme databázovú schému. V druhej časti popíšeme výber vhodných open source nástrojov, ktoré využijeme pre implementáciu porototypu. V závere popíšeme dosiahnuté výsledky v testovacom prostredí, ktoré preukážu vhodnosť využitia NoSQL systému Cassandra pre riešenie tejto úlohy.

8.1 Zdroj dát

Základným prvkom, ktorý budeme v našom systéme archivovať je emailový objekt, ktorý definuje dokument RFC 2821 [2]. Tento objekt pozostáva z SMTP obálky a emailovej správy. Obálka obsahuje informácie, ktoré sú potrebné pre korektné doručenie správy pomocou emailového servera a patria tam napríklad odosielateľ emailového objektu a jeden alebo viacerý príjemcovia. Emailová správa predstavuje semištrukturovaný dokument [39] v textovej podobe, ktorého syntax popisuje štandard RFC 2822 [1] z roku 2001 pod názvom Formát Internetovej správy (Internet Message Format). Dokument RFC 2822 nahradzuje a upravuje pôvodné RFC 822 pod názvom Štandard pre formát Internetových textových správ ARPA z roku 1982 (Standard for the Fromat of ARPA Internet Text Messages). Obsah emailovej správy delíme na hlavičku a telo, ktoré sú od seba oddelené znakom reprezentujúcim prázdny riadok. Telo správy nie je povinné. Štruktúru tela správy a polia v hlavičke rožšírujú štandardy, pod názvom MIME (Multipurpose Internet Mail Extensions), RFC 2045, RFC 2046 [18, 19], RFC 2047, RFC 2048 a RFC 2049. Tieto štandardy pridávajú možnosť použitia iných znakových sád (štandardná sada US-ASCII), ďalej umožnujú štruktúrovať telo správy (vnorené správy rfc822), definujú formát a typy pre zasielanie príloh atď.

Zber emailových objektov budeme realizovať na unixových serveroch, ktoré používajú emailový server Qmail[1]. Tento server zároveň realizuje antispamovú kontrolu pomocou modulu Qmail-scanner[2], ktorý je naprogramovaný v jazyku Perl[3]. Po doručení emailového objektu na server je emailový objekt spracovaný programom Qmail, ktorý volá obslužný modul

[1] http://cr.yp.to/qmail.html
[2] http://qmail-scanner.sourceforge.net
[3] http://www.perl.org

`qmail-scanner.pl` a následne dokončí doručenie správy. Tento modul sme vhodne modifikovali pre potreby našej aplikácie. Výstupom je súbor s príponou *.envelope*, ktorý obsahuje viacero štatistických údajov znázornených na obrázku 8.1. Detailný popis tejto štruktúry sa nachádza na webovej adrese http://qmail-scanner.sourceforge.net/. Výstupom aplikácie Qmail je, v prípade úspešného doručenia emailu, textový súbor reprezentujúci emailovú správu, ku ktorej je jednoznačne priradený súbor s príponou *.envelope*. Oba súbory sú spracúvané analyzátorom, ktorý popíšeme v následujúcej časti. Súčasťou prílohy tejto práce je modifikovaný súbor `qmail-scanner.pl`.

```
Tue, 15 Mar 2011 10:12:09 CET Clear:RC:1(88.208.65.55):SA:1 0.007811 9508
odosielatel@server prijemca@server2 predmet <13001802281039145460aq>
1300180329.16836-0.forid1:5987 priloha1:134
```

Obr. 8.1: Obsah obálky z programu Qmail-scanner

Zo súboru s príponou *.envelope* sme použili následujúce údaje:

- dátum a čas prijatia emailového objektu serverom Qmail
- pole SA:1, kde hodnota 1 reprezentuje spam, inak nula
- čas spracovania antispamovým filtrom
- veľkosť emailu
- polia príjemnca a odosielateľ

8.2 Analýza dát

Jedným z hlavých požiadavkov systému je deduplikácia príloh emailových správ z dôvodu úspory diskovej kapacity. Hlavička s názvom *Content-Type*, ktorú definuje RFC 2045 špecifikuje typ dát v tele MIME správy. Jej hodnota je tvorená z dvoch častí a to názov typu média (media type) a bližšie špecifikovaný podtyp, napríklad *„image/gif“*. Norma definuje päť základných typov médií a to text, image, audio, video a application. V prípade, našej aplikácie má zmysel využiť deduplikáciu na všetky tieto typ s výnimkou typu *„text/plain“*, kde predpokladáme, že sa jedná o bežnú textovú správu napísanú uživateľom.

Program pre analýzu a deduplikáciu emailovej správy bol napísaný v programovacom jazyku Python[4]. Tento program dodržiava špecifikáciu RFC 2822 a RFC2045. Medzi povinné polia hlavičky emailu patria pole *From* a *Date*. Aj napriek tomu, že tieto polia sú definované už od roku 1982 v RFC 1982 analýza nášho datasetu preukázala prítomnosť správ, ktoré tento požiadavok nespĺňali, jednalo sa hlavne o správy typu spam. Z množiny, ktorá obsahovala 980,000 emailových sprav bolo 0.022% emailov, ktoré nespĺňali štruktúru definovanú normou RFC 2045. Metódu deduplikácie sme riešili nasledujúcim postupom:

- programom pre analýzou emailu sme určili časti, v ktorých sa nachádzajú prílohy

[4] http://python.org

- H je výstup hašovacej funkcie SHA2 nad dátami reprezentujúcimi prílohu, ktorý sme použili ako unikátny identifikátor prílohy
- dáta reprezentujúce prílohu v emaile sme nahradili značkou v tvare MARK:H
- dáta reprezentujúce prílohu sme do databáze uložili pod kľúčom H

TODO>: ukazka emailu s prilohou + vysledok po deduplikacii

8.3 Databázová schéma

Databázovú schému sme navrhli s ohľadom na to aké operácie nad danými dátami budeme vykonávať a pri návhru sme využili poznatky získané štúdiom architektúry databázového systému Cassandra. Schéma je tvorená pomocou štyroch rodín stĺpcov a to:

- *messagesMetaData* - obsahuje meta informácie identifikované v obálke emailového objektu a emailovej správy, nad ktorými budeme vykonávať štatistické výpočty pomocou metódy MapReduce
- *messagesContent* - obsahuje obálku, hlavičku a telo správy
- *messagesAttachment* - slúži na ukladanie deduplikovaných príloh emailov
- *lastInbox* - v chronologickom časovom poradí, podľa hodnoty poľa *Date* v hlavičke emailu, zaznamenáva správy daného užívateľa

Tradičné techniky pre popis databázových schémat nie je možné aplikovať na databázové systémy, ktoré vychádzajú z konceptov Bigtable alebo Dynamo. Jedným z dôvodom je, že na tieto schémy sa aplikuje denormalizácia, duplikácia dát a klúče sú často komplexného charakteru. Dodnes neexistuje, žiadny štandard, ktorý by definoval popis týchto schémat. Článok pod názvom: „Techniky pre definíciu štruktúr pomocou diagramov v cloude a návrhové vzory“ (Cloud data structure diagramming techniques and design patterns [12]) definuje stereotypy pre diagramy v jazyku UML a obsahuje vzory pre popis štruktúry týchto dát. Obrázok 8.2 znázorňuje databázovú schému nášho modelu a využíva techniky popísané v spomínanom článku.

Ako jedinečný identifikátor emailovej správy v databáze využivame nasledujúcu schému:

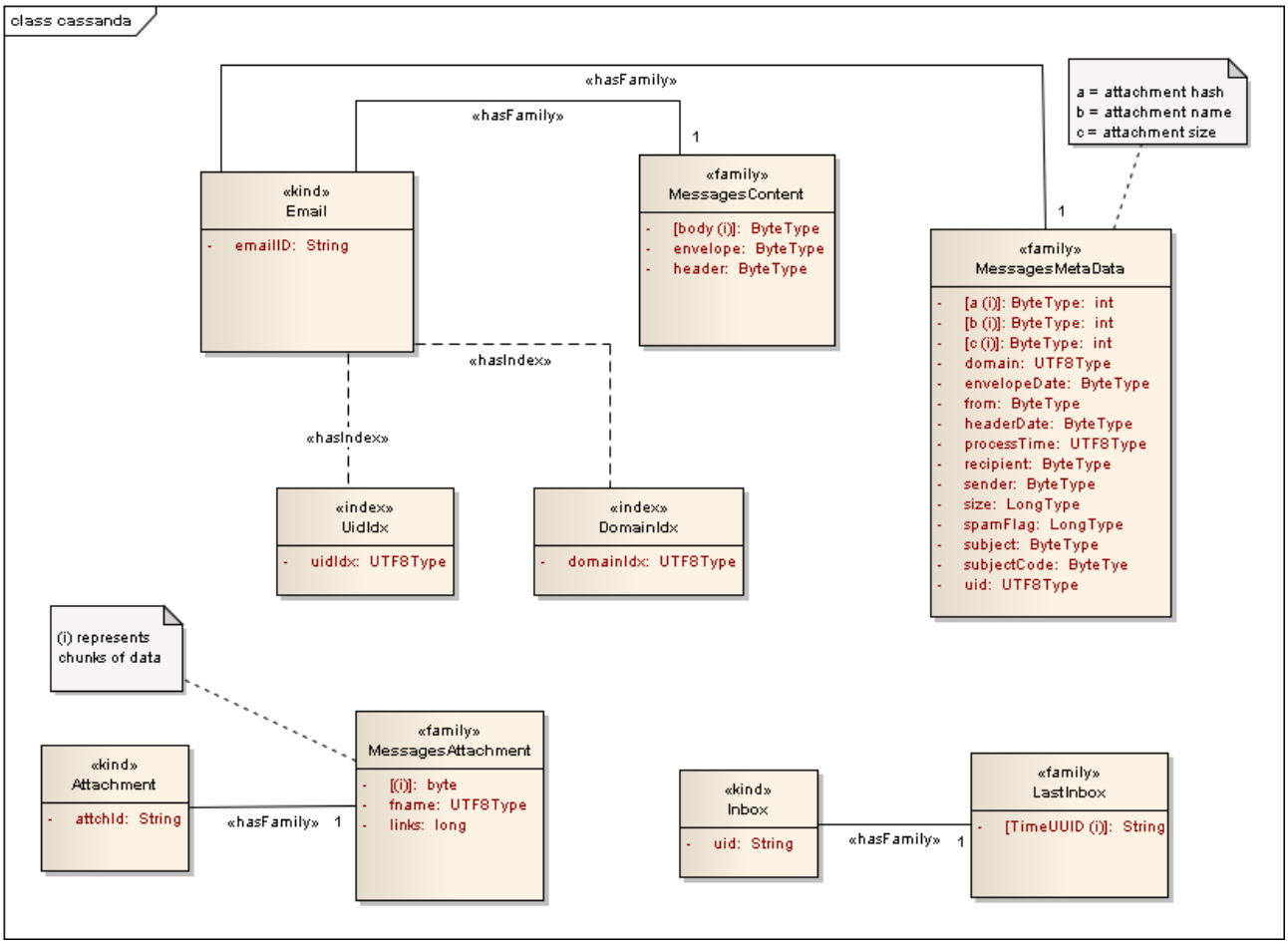
```
emailID = sha256(uid + MessageId + date)
```

V tejto schéme reprezentuje:

- *uid* emailovú adresu príjemcu, v tvare jan@mak.com
- *MessageID* je ide identifikátor správy z hlavičky daného emailu
- *date* je časová značka reprezentujúca čas kedy bol email prijatý emailovým serverom (formát: rok, mesiac, deň, hodina, minúta, sekunda)

- *emailID* je výstup hašovacej funkcie SHA2 v hexadecimálnom tvare

V predchádzajúcej kapitole sme zistili, že Cassandra nie je optimalizovaná pre zápis blokov dát (blob), ktorých veľkosť presahuje 1 MB avšak optimálne výsledky pre zápis dosahuje pri veľkosti blokov 512 kB. V prípade ak dáta reprezentujúce telo emailovej správy alebo objekt s prílohou presahujú veľkosť 1 MB, tak ich zapisujeme do samostatných stĺpcov o veľkosti 512 kB. Toto rozdeľovanie dát na menšie bloky vykonávame na aplikačnej úrovni na strane klienta zapisujúceho dáta do databázového systému. Názvy stĺpcov číslujeme vzostupne v intervale *0-N*, kde *N* je počet blokov. Spätnú rekoštrukciu dát vykonáva klient.



Obr. 8.2: Databázová schéma

Výsledný návrh databázovej schémy ma viacero výhod. Metadáta, nad ktorými budeme vykonávať výpočet štatistík (messagesMetaData) budú uložené v jednom súbore, čo zabezpečí ich efektívne spracovanie. V tejto rodine stĺpcov ďalej využívame index s názvom DomainIdx, vďaka ktorému sme schopný získať všetky emaily uložené v systéme prináležiace určitej doméne, v tvare napríklad „cvut.cz“.

Všetky emailové správy, ktoré prináležia užívateľovi, ktorý je identifikovaný pomocou svojej emailovej adresy budú uložené v jednom riadku v rodine stĺpcov lastInbox. Názvy

stĺpcov budú reprezentované pomocou TimeUUID, čo zabezpečí ich zoradenie podľa času. Túto rodinu stĺpcov môžeme použiť v aplikácii pre sprístupnenie obsahu emailového adresára koncovým užívateľom.

8.4 Fultextové vyhľadávanie

Fultextové vyhľadávanie realizujeme pomocou samostatného NoSQL systému Elasticsearch[5]. Hlavný index, ktorý obsahuje všetky zaindexované dáta ma názov *emailArchive*, a delíme ho na dva typy s názvom *email* a *envelope*. Schéma týchto typov obsahuje polia poďla, ktorých chceme v emailovom archíve vyhľadávať a jej reprezentáciu zapísanú vo formáte JSON znázorňuje obrázok 8.3.

```
mappingsEmail = {
  "inbox": {"type": "string"},
  "from": {"type": "string"},
  "subject": {"type": "string"},
  "date" :{"type": "date"},
  "messageID" :{"type": "string", "index": "not_analyzed"},
  "attachments":{"type": "string"},
  "size": {"type": "long", "index": "not_analyzed"},
  "body": {"type": "string"}
}

mappingsEnvelope = {
  "sender": {"type": "string"},
  "recipient": {"type": "string"},
  "ip": {"type": "ip"},
  "date": {"type": "date"}
}
```

Obr. 8.3: JSON schéma pre fultextové vyhľadávanie

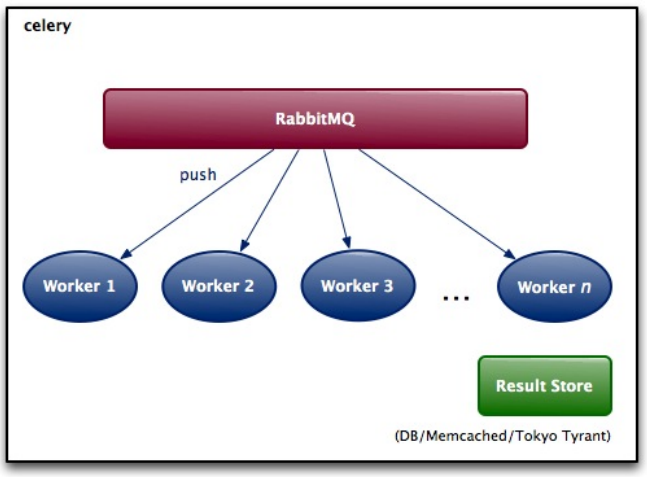
[5] http://elasticsearch.org

8.5 Implementácia

V programovacom jazyku Python sme implementovali klienta pre zápis dát do databáze Cassandra a ElasticSearch. Jednou z najdoležitejších vlastností týchto klientov je voľba úrovne konzistencie pri zápise. Našou prioritou je integrita dát a od databáze požadujeme silnú konzistenciu. Zvolili sme mód kvóra (quorum), ktorý zabezpečí zápis dát na $N/2+1$ replík a klient následne obdrží potvrdenie o úspešnosti zápisu, inak sa zápis zopakuje. Tieto vlastnosti nám zabezpečujú, v prípade použitia hodnoty replikácie tri (dáta sa v databázovom systéme nachádzajú trikrát), silnú úroveň konzistencie na strane klienta a databázového systému. Analýza emailovej správy a jej deduplikácia spotrebúva hlavne CPU zdroje. Moderné procesory obsahuju viacero jadier, tento fakt môžeme využiť pre paralelné spracúvanie emailov, teda každé jadro CPU bude spracúvať súčasne jednu emailovú správu.

Celery

Paralelizáciu našej aplikácie sme zabezpečili pomocou využitia asynchrónnej fronty úloh pod názvom Celery[6], ktorá využíva architektúru distribuovaného predávania správ (distributed message passing). Architektúru znázorňuje obrázok 8.4. „Pracovníci“ (workers) reprezentujú samostatné procesy v našom prípade proces pre analýzu a deduplikáciu emailu, ktoré môžu bežať paralelne. Ako sprostredkovateľ (broker) je použitá aplikácia RabbitMQ[7]. Sprostredkovateľ obdrží od klienta správu a uloží ju do fronty. Správa obsahuje identifikátor emailovej správy, v tomto prípade cestu na lokálnom súborovom systéme k súboru reprezentujúcom email.



Obr. 8.4: Architektúra Celery, Zdroj: [41]

Táto správa je následne zaslaná ľubovoľnému pracovníkovy (v našom prípade klient vykonávajúci analýzu a deduplikáciu emailu), ktorý ju spracuje. Táto architektúra je plne

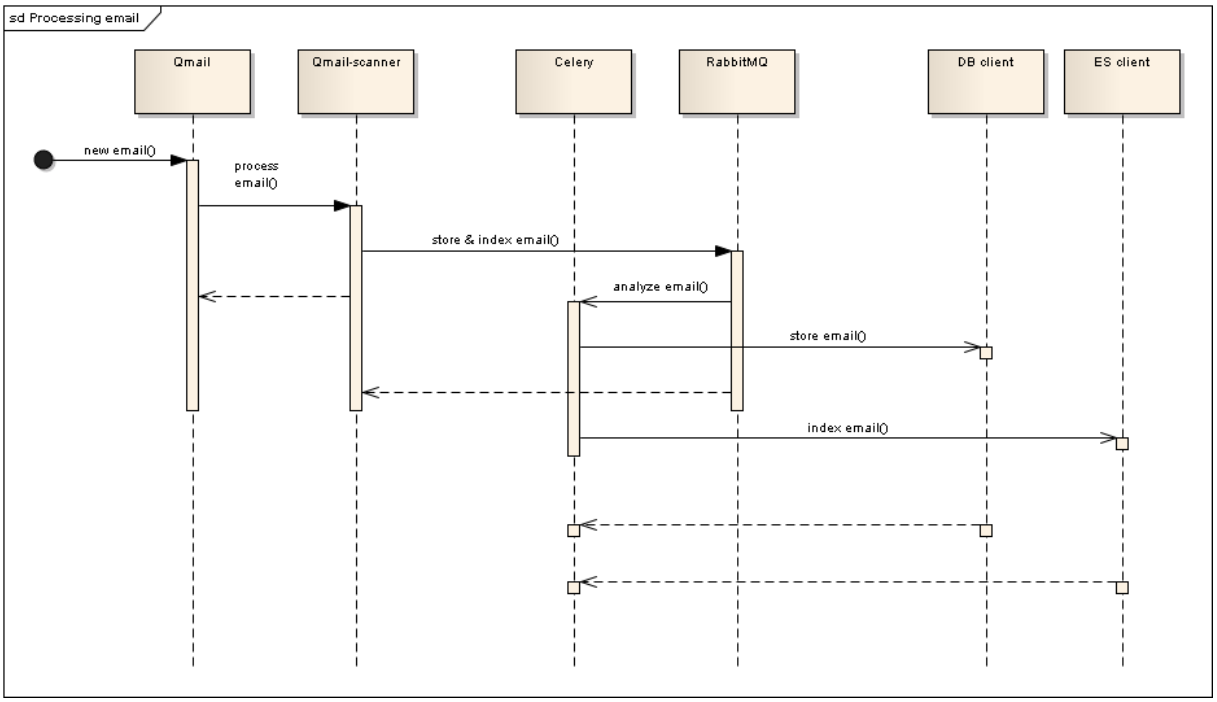
[6] http://celeryproject.org
[7] http://www.rabbitmq.com

distribuovaná, dokáže odolávať chybám (napr. v prípade výpadku elektrickej energie správy nadalej pretrvávajú vo fronte).

Frontu RabbitMQ sme podrobili výkonnostnému testu, zvládala XY operácií pre zápis a XY pre načitanie správ.

8.5.1 Klient

Proces spracovania nového emailu je znázornený pomocou sekvenčného diagramu na obrázku 8.5. Pri príchode nového emailu, ktorý je spracovaný emailovým serverom Qmail, je vytvorená nová úloha pomocou aplikácie Celery. Táto úloha uloží do sprostredkovateľa identifikátor emailu. V prípade, že je v daný okamžik k dispozícií ľubovoľný pracovník, je emailová správa spracovaná pomocou analyzátora a následne zapísaná do databáze Cassandra a fultextového systému Elasticsearch. Na testovacie účely sme nemali k dispozícii reálny dátový tok emailov. Vrstvu reprezentujúcu Qmail sme nahradili modulom, vytvárajúcim nové úlohy prechádzaním lokálneho súborového systému, ktorý obsahoval testovaciu množinu emailových správ.



Obr. 8.5: Spracovanie emailu

Pomocou klienta pyccassa[8] sme implementovali aplikáciu pre prístup k dátam uloženým v databázovom systéme. Klient umožnuje následovné operácie:

- načítanie pôvodnej štruktúry emailu, ktorý sme uložili do systému, prípadná deduplikácia je nahradená dátami reprezentujúcimi prílohu

[8] http://pycassa.github.com/pycassa/

- načitanie voliteľného počtu emailov pre daného užívateľa, ktoré sú zoradené v chronologicky od najstaršieho
- náhodný výpis definovaného počtu emailov pre daného užívateľa
- výpis stručných informácií reprezentujúcich email, konkrétne polia: odosielateľ, predmet, dátum, veľkosť emailu, počet príloh

TODO Návrh možnej realizácie daného riešenia je zachytený pomocou diagramu nasadenia na obrázku B.1. V tomto prípade uvažujeme 2 geo datacen ... + hadoop... popisat

8.5.2 Výpočet štatistík

Cassandra spolupracuje so systémom Hadoop, čo nám dáva do rúk mocný nástroj na masívne paralelné spracovanie dát pomocou techniky MapReduce. V tomto prípade je potrebné na uzly, na ktorých je Cassandra nainsťalovať aplikácie HDFS a Hadoop. Tvorba aplikácií v tomto frameworku prebieha v Jave, je náročná a okrem toho programový model MapReduce obsahuje viaceo problémov. Model napríklad neobsahuje primitíva na filtrovanie, agregáciu, spájanie dát a je potrebná ich vlastná implementácia. Tieto nedostatky rieši nástroj Pig [32] vďaka, ktorému sme boli schopný efektívne spracúvať metadáta uložené v databáze. Obrázok 8.6 zobrazuje programovú ukážku, ktorá slúži na výpočet veľkosti najväčsieho emailu pre domény, ktorých emaily archivujeme. Pre jednoduchosť sme vynechali časti, ktoré slúžia na načítanie dát z databáze a obsahujú uživateľsky definovanú funkciu v programovacom jazyku Java, ktorá slúži na predprípravu vstupných dát. Podpora uživateľom definovaných funkcií je jednou z ďalších výhod nástroja Pig.

```
notSpam = FILTER grp BY group.spam == 1;
maxSize = foreach grp {
  size = rows.size;
  generate group, MAX(size);
};
STORE maxSize into 'biggestEmailPerDomainDomain' using PigStorage(',');
```

Obr. 8.6: Programová ukážka v jazyku Pig

8.5.3 Webové rozhranie

Vďaka tomu, že klient pre prístup k dátam bol vytvorený v programovacom jazyku Python sme pomocou Django vytvorili jednoduchú prezenčnú vrstu, ktorou sme boli schopný pristupovať k uloženým emailovým správam pomocou webového rozhrania.

8.6 Overenie návrhu

Pomocou vyšie popísaného návrhu a implementovaných nástrojov sme overili funkčnosť nami navrhovaného modelu. Vhodná voľba daných verzií u aplikácií Celery a RabbiMQ bola určená

počas písania a ladenia aplikácie. Všetky tieto aplikácie sú neustále vo vývoji, to isté platí pre databázu Cassandra a systém Hadoop. Počas písania tejto práce prebehlo viacero rozhovorov s autormi týchto aplikácii. Konkrétne databáza Cassandra na začiatku práce neobsahovala takmer žiadnu ucelenú dokumentáciu, počas začiatkov experimentov sme začínali s verziou 0.7.0. Počas ukončovania tejto práce je k dispozícii aktuálna verzia 0.7.5 a medzitým vznikla kvalitná online dokumentácia od spoločnosti Datastax[9].

Konfigurácia

Hardverová konfigurácia bola totožná s konfiguráciou z kapitoly 7. Na šiestich serveroch bola nainštalovaná databáza Cassandra 0.7.3, Hadoop 0.20.2, dvojica serverov obsahovala klientskú aplikáciu pre zápis dát, ktorú používala aplikácia Celery 2.6. V úlohe sprostredkovateľa bola použitá aplikácia RabbitMQ 2.1.1, ktorá bola nainštalovaná na samostatnom serveri.

Overenie integrity dát

Databázový kluster sme naplnili testovacími dátami obsahujúcimi emaily o objeme cca 300 GB. Tie isté dáta sme zároveň spracovali a zapísali do systému ElasticSearch. Následne sme simulovali prípad obnovy dát z archívu, kde sme všetky emaily v náhodnom poradí z databázy načítali, zostavili ich do pôvodného tvaru klientskou aplikáciou a porovnali sme ich odtlačok pomocou hašovacej funkcie MD5 s odtlačkom pôvodných dát uložených na pevnom disku. Tento test prebehol bez akejkoľvek chyby. Klient, ktorý slúžil na čítanie dát z databáze využíval mód kvóra, z dôvodu požiadavku na integritu dát.

Dosiahnuté výsledky

Doležitým pozorovaním bol fakt, že z celkového objemu emailových správ cca 300 GB sa po deduplikácií príloh tento objem znížil na cca 99 GB, teda došlo k 67% úspore diskovej kapacity. Štruktúry do ktorých sme ukladali dáta pre potrebu štatistík zaberali 0,4% z celkového objemu dát, čo je zanedbateľná položka.

ElasticSearch rychlost pre fultextové vyhladavanie bola do XY ms, indexy nad objemom dát 300 GB tvorili XY GB.

[9] https://datastax.com

Kapitola 9

Záver

Literatúra

[1] Internet message format, 2001.

[2] Simple mail transfer protocol, 2001.

[3] D. Abadi, P. Boncz, and S. Harizopoulos. Column-oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665, 2009.

[4] D. Abadi, P. Boncz, and S. Harizopoulos. Column-oriented database systems, VLDB Tutorial. 2009.
`www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf`.

[5] T. Barina. ???, máj 2011.

[6] P. Bernstein and N. Goodman. Concurrency control in distributed database systems. *ACM Computing Surveys (CSUR)*, 13(2):185–221, 1981.

[7] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*, volume 5. Addison-wesley New York, 1987.

[8] A. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203. ACM, 2000.

[9] E. Brewer. Towards robust distributed systems. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, volume 19, pages 7–10, 2000.

[10] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

[11] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387, June 1970.

[12] David Salmen, Tatiana Malyuta, Rhonda Fetters, Normert antunes. Cloud Data Structure Diagramming Techniques and Design Patterns, 2010.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[14] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[15] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[16] E. Eifrem. Neo4j the benefits of graph databases. `http://wiki.neo4j.org/content/Presentations`, stav z 3. 1. 2011.

[17] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gauthier. Cluster-based scalable network services. In *ACM SIGOPS Operating Systems Review*, volume 31, pages 78–91. ACM, 1997.

[18] N. Freed and N. Borenstein. Multipurpose internet mail extensions (mime) part one: Format of internet message bodies, 1996.

[19] N. Freed and N. Borenstein. Multipurpose internet mail extensions (mime) part two: Media types, 1996.

[20] A. Ganesh, A. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE transactions on computers*, pages 139–149, 2003.

[21] J. F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting, and A. Toncheva. The diverse and exploding digital universe. *IDC White Paper*, 2, 2008.

[22] J. F. Gantz, J. Mcarthur, and S. Minton. The expanding digital universe. *Director*, 285(6), 2007.

[23] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[24] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[25] D. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, pages 150–162. ACM, 1979.

[26] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33:51–59, June 2002.

[27] J. Gray et al. The transaction concept: Virtues and limitations. In *Proceedings of the Very Large Database Conference*, pages 144–154. Citeseer, 1981.

[28] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983.

[29] N. Hayashibara, X. Defago, R. Yared, and T. Katayama. The ϕ accrual failure detector. In *Symposium on Reliable Distributed Systems (SRDS'2004)*, pages 66–78. Citeseer, 2004.

[30] E. Hewitt. *Cassandra: the definitive guide*. O'Reilly Media, Inc., 2010.

[31] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[32] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[33] A. O. R. W. Paper. Why Cloud-Based Security and Archiving Make Sense, March 2010.
`www.google.com/postini/pdf/why_cloud_based_wp.pdf`, stav z 28. 2. 2011.

[34] D. Pritchett. Base: An acid alternative. *Queue*, 6:48–55, May 2008.

[35] T. Segaran and J. Hammerbacher. *Beautiful data: the stories behind elegant data solutions*. O'Reilly Media, 2009.

[36] R. Shoup. The eBay Architecture, Striking a balance between site stability, feature velocity, performance, and cost, November 2006.
`www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf`, stav z 28. 2. 2011.

[37] M. Stonebraker. The case for shared nothing. *Database Engineering Bulletin*, 9(1):4–9, 1986.

[38] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 1013–1020, New York, NY, USA, 2010. ACM.

[39] J. Udell. *Practical Internet GroupWare*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 1999.

[40] W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.

[41] Celery introduction.
`http://ask.github.com/celery/getting-started/introduction.html`, stav z 3. 2. 2011.

[42] Discretionary access control.
`https://issues.apache.org/jira/browse/HBASE-1697`, stav z 3. 5. 2011.

[43] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

Dodatok A

Zoznam použitých skratiek

API -

GC

GFS

HDFS

JVM

KFS

NoSQL

PB

RFC

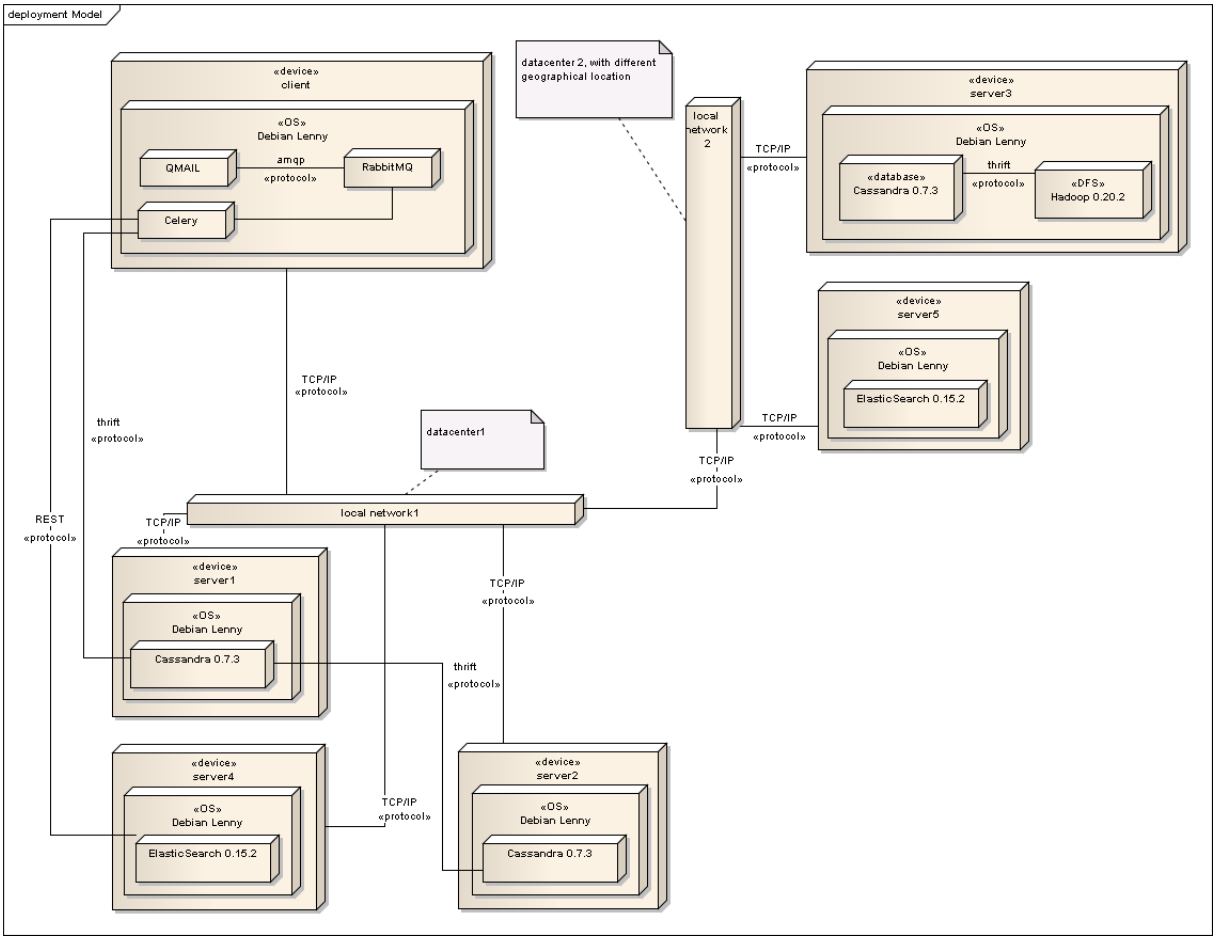
SPOF

TTL

⋮

Dodatok B

UML diagramy



Obr. B.1: Diagram nasadenia aplikácie

Dodatok C

Inštalačná a užívateľská príručka

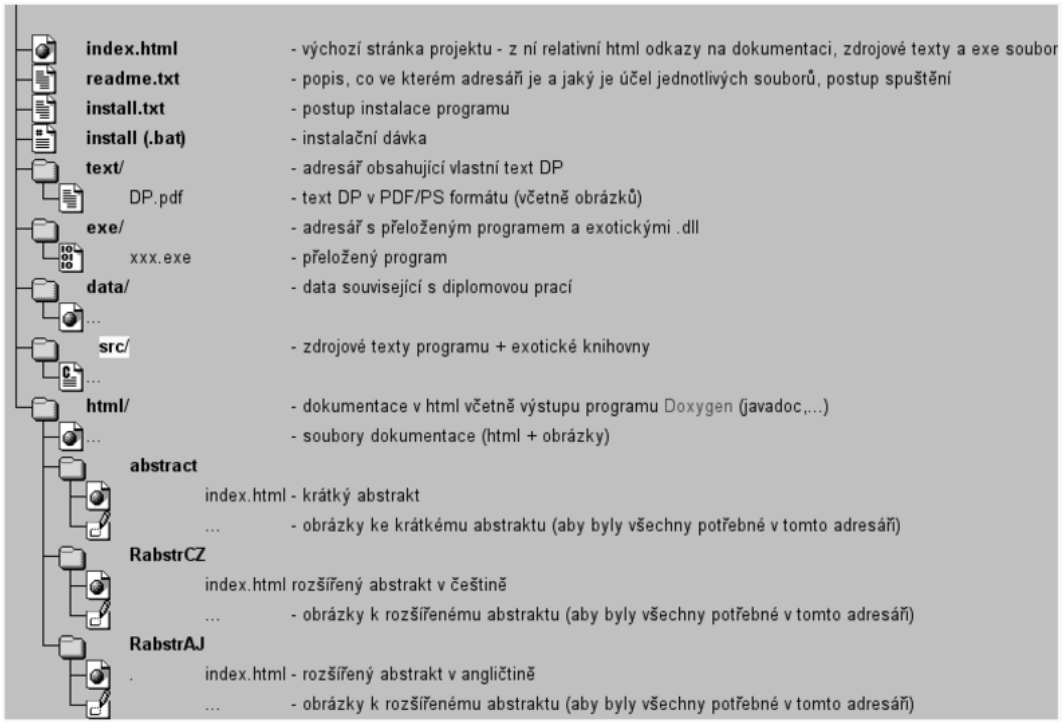
Tato příloha velmi žádoucí zejména u softwarových implementačních prací.

Dodatok D

Obsah priloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [**?**]):



Obr. D.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.