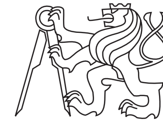
Sem vložte zadání Vaší práce.

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



Diplomová práce

Indexování struktur v grafovém DB stroji Neo4j II

Bc. Jaroslav Ramba

Vedoucí práce: Ing. Michal Valenta, Ph.D.

6. května 2015

Poděkování

Velmi rád bych poděkoval své rodině a přítelkyni za velkou podporu nejen při psaní této diplomové práce, ale i v celém průběhu studia. Taktéž velké díky patří vedoucímu Ing. Michalu Valentovi, Ph.D. a oponentovi práce MSc. Michalu Bachmanovi za konzultace, cenné rady a nasměrování k požadovanému výsledku během celé realizace. A nakonec také spolužáku Bc. Martinu Troupovi za výbornou spolupráci v úvodní části diplomové práce.

Prohlášení

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

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

V Praze dne 6. května 2015 ....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2015 Jaroslav Ramba. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Ramba, Jaroslav. *Indexování struktur v grafovém DB stroji Neo4j II.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Diplomová práce se v první části zabývá analýzou možností indexování grafových vzorů v grafových databázích. Z provedené analýzy se navrhne a implementuje prototyp indexu pro grafové vzory nad zvolenou grafovou databází. Na konci práce bude realizováno kompletní výkonnostní testování několika typických vzorů na vhodných vzorových datech.

Klíčová slova Indexování grafových vzorů, Grafová databáze, Optimalizace databáze

Abstract

This thesis in the first part analyzes indexing graph patterns in graph databases. Subsequently, the analysis, design and implementation of prototype designs of the index with chosen a graph database. At the end of the work is complete performance testing several typical patterns at the appropriate sample data.

Keywords Indexing graph patterns, Graph database, Database optimization

Obsah

Úvod **1**

1 Základní definice **3**
1.1 Grafová teorie . . . 3
1.2 Representational state transfer . . . 7

2 Analýza **11**
2.1 Deklarace záměru . . . 11
2.2 Aktuální stav . . . 11
2.3 Budoucí stav . . . 18
2.4 Požadavky . . . 18

3 Návrh **21**
3.1 Index pro tabulární data . . . 21
3.2 Návrh indexu . . . 22

4 Vyhodnocení dotazů v Neo4j **25**
4.1 Konfigurace testovacího prostředí . . . 25
4.2 Graphaware framework . . . 27
4.3 Erdős–Rényi . . . 28
4.4 Testování výkonu dotazů . . . 28

5 Implementace **37**
5.1 Nástroje a techniky . . . 37
5.2 REST API . . . 37
5.3 Operace INSERT/DELETE . . . 39
5.4 Index . . . 39

6 Testování výkonu indexu **41**
6.1 Databáze Erdős–Rényi . . . 41

6.2 Filmová databáze Cineasts 47
6.3 Databáze transakcí 51

7 Porovnání výsledných řešení **55**
7.1 Databáze Erdős–Rényi 56
7.2 Databáze Cineasts 59
7.3 Databáze transakcí 62

Závěr **65**

Literatura **67**

A Seznam použitých zkratek **73**

B Obsah přiloženého CD **75**

Seznam obrázků

1.1 Příklad orientovaného grafu, který neobsahuje násobné hrany . . . 4
1.2 Ukázkový model pro property graph . . . 5

2.1 Architektura Neo4j . . . 16
2.2 Reprezentace fyzického úložiště Neo4j . . . 17
2.3 Datové úložiště Neo4j . . . 18
2.4 Model případů užití . . . 20

3.1 Distribuce dat v Chronicle-Map . . . 24

4.1 Schéma databáze Erdős–Rényi . . . 28
4.2 Výsledky vyhledávání se zakotvením jednoho uzlu . . . 29
4.3 Výsledky vylepšeného vyhledávání se zakotvením jednoho uzlu . . 30
4.4 Výsledky vyhledávání se zakotvením všech uzlů . . . 31
4.5 Výsledky vyhledávání na podgrafu . . . 32
4.6 Výsledky přehrání jednoho vzoru do externí grafové databáze . . . 33
4.7 Výsledky přehraní všech vzorů do externí grafové databáze . . . 34

5.1 REST API reprezentováno třídou IndexAPI . . . 37
5.2 Ukázka požadavku na vytvoření indexu, získání konkrétních vzorů a smazání indexu . . . 38
5.3 Runtime modul . . . 39
5.4 Grafový index . . . 40

6.1 Výsledky měření na databázi Erdős–Rényi o velikosti 1 000 uzlů a 5 000 hran . . . 42
6.2 Výsledky měření na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran . . . 43
6.3 Výsledky pro přidání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran . . . 44

6.4 Výsledky pro smazání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 44
6.5 Výsledky pro smazání uzlu na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 45
6.6 Výsledky měření na databázi Erdős–Rényi o velikosti 100 000 uzlů a 500 000 hran 46
6.7 Základní schéma filmové databáze Cineasts 47
6.8 Výsledky měření na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 48
6.9 Výsledky pro přidání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 49
6.10 Výsledky pro smazání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 49
6.11 Výsledky pro smazání uzlu na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 50
6.12 Schéma databáze transakcí 51
6.13 Výsledky měření na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 52
6.14 Výsledky pro přidání hrany na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 53
6.15 Výsledky pro smazání hrany na databázi transakcí o velikosti 72 000 uzlů a 106 651 hran 53
6.16 Výsledky pro smazání uzlu na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 54

7.1 Schéma stromového indexu 55
7.2 Porovnání výsledků měření na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 56
7.3 Porovnání výsledků přidání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 57
7.4 Porovnání výsledků smazání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 57
7.5 Porovnání výsledků smazání uzlu na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 58
7.6 Porovnání výsledků měření na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 59
7.7 Porovnání výsledků přidání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 60
7.8 Porovnání výsledků smazání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 60
7.9 Porovnání výsledků smazání uzlu na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 61
7.10 Porovnání výsledků měření na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 62

7.11 Porovnání výsledků přidání hrany na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 63
7.12 Porovnání výsledků smazání hrany na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 63
7.13 Porovnání výsledků smazání uzlu na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 64

Seznam tabulek

1.1 Seznam HTTP metod v rámci webové služby REST 9

3.1 Asymptotické složitosti algoritmů 22
3.2 Příklad struktury indexu pro vzor trojúhelníku 23

4.1 Hardware konfigurace testovacího prostředí 1 25
4.2 Hardware konfigurace testovacího prostředí 2 26
4.3 Konfigurace parametru dbms.pagecache.memory 26
4.4 Konfigurace parametru cache_type 27

6.1 Souhrnné výsledky měření na databázi Erdős–Rényi o velikosti 1 000 uzlů a 5 000 hran 42
6.2 Souhrnné výsledky měření na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 43
6.3 Souhrnné výsledky měření na databázi Erdős–Rényi o velikosti 100 000 uzlů a 500 000 hran 46
6.4 Souhrnné výsledky měření na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 48
6.5 Souhrnné výsledky měření na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 52

7.1 Souhrnné výsledky porovnání na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran 56
7.2 Souhrnné výsledky porovnání na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran 59
7.3 Souhrnné výsledky porovnání na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran 62

Úvod

Více jak 90 % všech dat na světě bylo vytvořeno v posledních dvou letech [1]. Značně se také změnila struktura a zvýšila propojenost dat, které lidstvo generuje, zpracovává a ukládá. Začaly vznikat projekty jako Facebook, Twitter, LinkedIn a mnoho dalších, které mají společnou vlastnost, a tou je sociální aspekt.

Sdílení, doporučování, hodnocení jsou každodenní rutinou uživatelů těchto sociální sítí. Všechny tyto vztahy je potřeba zaznamenávat a vytěžovat z nich maximum znalostí. Příkladem může být doporučení přátelství mezi uživateli, kteří mají společné přátele, čímž existuje určitá pravděpodobnost, že se spolu také znají nebo v budoucnu seznámí a uzavřou trojúhelník přátelství [2].

Také zjišťování komplexity sítě, koeficientu shlukování, celkový počet trojúhelníků a nebo průměrná pravděpodobnost pro nově vznikající vzory (v příkladu vznik hrany značící přátelství) jsou informace, které lze využít například k identifikaci největších komunit sítě [3].

Úložiště, které může být využito pro výše popsané projekty, se mění v čase. Nejedná se o žádný datový snímek, ale o online databázi. Pokud chceme získávat grafové vzory, které se v úložišti vyskytují, je nutné o nich uchovávat určité informace. O jaké informace se jedná, jakým způsobem je ukládat a jak se zpětně pomocí nich odkazovat do úložiště, si objasníme v diplomové práci.

Kapitola **1**

Základní definice

Ještě předtím, než se pustíme do jakékoliv analýzy, testování a průzkumu, seznámíme se s několika základními definicemi a dostupnými technologiemi, které budeme využívat během celé realizace této práce, jak už bylo napovězeno v úvodu. Nejprve si popíšeme základní grafové pojmy, k nimž je vždy doplněna i formální definice. Následně se dostaneme k moderním termínům pro interpretaci grafů a k technologiím pro návrh rozhraní grafového indexu. Konkrétně se bude jednat o architekturu REST (Representational State Transfer), skrze kterou bude grafový index poskytovat rozhraní pro komunikaci s uživatelem a dalšími technologiemi s tím spojenými.

1.1 Grafová teorie

1.1.1 Vrchol (Vertex)

Vrchol (někdy také uzel) je součást grafové struktury, která reprezentuje objekty nebo jejich referenci. Množinu všech uzlů v grafu značíme V.

1.1.2 Hrana (Edge)

Hrana (někdy také vztah) znázorňuje spojení mezi jednotlivými vrcholy [4]. Množinu všech hran v grafu značíme E.

V – neprázdná množina vrcholů (disjunktní s množinou E)

E – množina hran (disjunktní s množinou V)

$|V|$ – počet uzlů množiny V

$|E|$ – počet hran množiny E

1.1.3 Graf

Obecný graf je datová struktura, v níž nemáme začátek ani konec [5]. Definujeme ji v podobě dvojice množin vrcholů (Vertices) a hran (Edges) [6].

Grafy dělíme do dvou skupin, na orientované a neorientované. Orientaci určujeme pomocí hran, které udávají směr z daného vrcholu do dalšího.

Neorientovaný graf G je dvojice (V, E), kde platí $E \subseteq \binom{V}{2}$, neboli množina E je podmnožinou množiny všech dvojic navzájem různých vrcholů [7].

Orientovaný graf $\overrightarrow{G}$ je dvojice (V, E), kde platí, že $E \subseteq (V \times V)$, neboli množina E je podmnožinou kartézského součinu $(V \times V)$, kde prvky E nazýváme orientovanými hranami. [8].

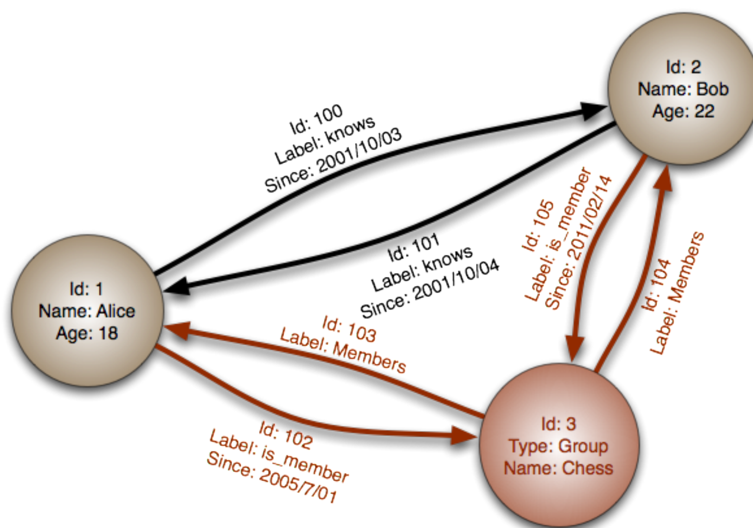
1.1.4 Typy hran [6]

Neorientovaná hrana je neuspořádaná dvojice vrcholů, která nemá definovaný směr průchodu, a hranou lze procházet oběma směry.

Orientovaná hrana je uspořádaná dvojice vrcholů, která má definovaný směr průchodu, a hranou lze procházet pouze vyznačeným směrem.

Násobnými hranami označujeme více hran spojujících stejné vrcholy.

Smyčka je hrana vedoucí do sebe samotné, tedy z vrcholu do téhož vrcholu.



Obrázek 1.1: Příklad orientovaného grafu, který neobsahuje násobné hrany [9]

1.1.5 Podgraf (subgraph)

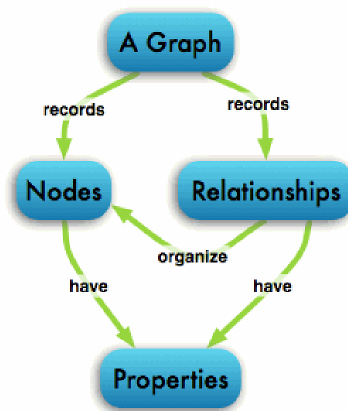
Podgraf $H = (V_H, E_H)$ grafu $G = (V_G, E_G)$ je takový graf, pro který platí následující podmínky [6]:

1. $V_H \subseteq V_G$
2. $E_H \subseteq E_G$
3. Všechny hrany podgrafu H mají krajní uzly obsaženy v množině uzlů podgrafu H (tzv. žádné hrany „nepoletují ve vzduchu“)

1.1.6 Vlastnosti v grafu (property graph)

Výše byl definován obecný graf, který může obsahovat krom uzlů a hran další obohacení. Takto obohacený graf nazýváme **property graph**. Uzly mohou obsahovat neomezený počet atributů, které jsou v podobě klíč–hodnota (key–value). Nadále mohou být k uzlům přiděleny značky (labels), které je zařazují do určité domény.

Hrany obsahují směr, typ, počáteční a koncový uzel. Stejně jako uzly mohou obsahovat neomezený počet atributů, které reprezentují ve většině případů kvantitativní vlastnosti, vzdálenost, hodnocení nebo časový interval [10].



Obrázek 1.2: Ukázkový model [10]

1.1.7 Strom (tree)

Strom je souvislý graf, který neobsahuje kružnice [11]. Po odebrání hrany se poruší souvislost a přidáním kružnice opět vzniká. V teorii grafů je kvůli své hierarchické struktuře přirovnáván k acyklickému grafu s jedním kořenem [6].

1.1.8 Cesta (path)

Cesta v grafu je posloupnost vrcholů a hran $(v_0, e_1, v_1, ..., e_t, v_t)$, kde vrcholy $v_0, ..., v_t$ jsou různé a pro každé $i = 1, 2, ..., t$ je $e_i = \{v_{i-1}, v_i\} \in E$ [12].

1.1.9 Souvislý graf

Souvislost grafu určuje, zda lze graf považovat za jednotný celek. Z každého uzlu se lze dostat nějakou cestou do jiného [12].

Graf G je souvislý, jestliže pro každé jeho dva uzly u a v existuje v G cesta z u do v.

1.1.10 Traverzování

Průchod, jinak řečeno traverzování, umožňuje navštívit určité vrcholy grafu [13]. Můžeme to zjednodušeně popsat jako systematické putování grafovou strukturou po jednotlivých vrcholech a hranách. Jakým způsobem je traverzování realizováno, určují vybrané algoritmy, jakými jsou například prohledávání do hloubky či do šířky [5].

1.1.11 Prohledávání grafu do šířky

Prohledávání do šířky (breadth-first search, zkráceně BFS), je grafový algoritmus, který slouží k postupnému průchodu souvislého grafu. Prohledávání začíná od zvoleného startovního uzlu, u kterého se projdou jeho sousedi, a uloží se do fronty. Na uložené uzly ve FIFO (First In, First Out) frontě se provede stejná operace jako na startovní uzel a takto se postupuje, dokud není fronta prázdná.

Tímto dochází k procházení grafu ve vlnách. Pořadí, v jakém jsou uzly zpracovány, určuje jejich vzdálenost od zvoleného počátečního uzlu. Výsledná asymptotická složitost algoritmu je $O(|V| + |H|)$, jelikož každý uzel a každou hranu projdeme právě jednou. Ovšem za předpokladu, že uložení, výběr z fronty a přístup k potomkům daného uzlu je $O(1)$ [14].

1.1.12 Prohledávání grafu do hloubky

Druhým zmíněným algoritmem je prohledávání do hloubky (depth-first search, zkráceně DFS). Získává vysoké uplatnění, jehož výhod se využívá při zjišťování topologického uspořádání, počtu komponent nebo detekci cyklů [15].

Algoritmus pracuje tak, že si zvolí jakýkoliv uzel, který označí jako otevřený, následně ho zpravuje a rekurzivně zavolá jeho prvního následníka. Toto provede u všech uzlů, které ještě nenavštívil. Pokud ale narazí na vrchol, z kterého už není možné pokračovat (nemá žádné následníky nebo byli všichni navštíveni),

vrací se zpět. Po návratu z rekurze uzel označí jako uzavřený. Tímto způsobem dojde k průchodu všech větví grafu do maximální hloubky.

Pokud použijeme stejný předpoklad jako u předchozího algoritmu, vychází asymptotická složitost na $O(|V|+|H|)$, jelikož opět každý uzel a každou hranu projdeme právě jednou [15].

1.1.13 Automorfismus

Automorfismus grafu $G = (V, E)$ je každý isomorfismus grafu G s G, tj. bijekce $f : V \hookrightarrow V'$ taková, že $\{u, v\} \in E$ právě když $\{f(u), f(v)\} \in E$. Můžeme si všimnout, že čím více je graf symetrický, tím více má automorfismů. Počet odlišných automorfismů grafu můžeme chápat jako jeho míru symetrie. [16]

Graf nazýváme **asymetrický** (strnulý), pokud je jeho jediný automorfismus identické zobrazení (každý vrchol se zobrazí sám na sebe) [17].

1.2 Representational state transfer

Representational state transfer (REST) je architektura rozhraní, která je navržená pro distribuované prostředí. Oproti XML-RPC a SOAP je REST orientován datově, nikoliv procedurálně [18]. Hlavní výhodou REST technologie, je snadný přístup k datovým zdrojům. Definuje způsob, jakým se k datům přistupuje, kdežto ostatní webové služby definují vzdálené procedury a protokol pro jejich volání.

Svou bezstavovostí nadále vychází vstříc trendům vývoje webových aplikací, které jsou založeny na paralelním zpracování distribuovaného obsahu [19], což je další vlastnost, díky které je rozšířenější než konkurenční koncepty.

1.2.1 Limitace

Za REST považujeme pouze architektonický styl, nikoliv exaktně specifikovaný protokol. Každá aplikace využívající architekturu je determinována níže specifikovanými limitacemi [20]. Pouze Code on demand je jediná, která je možná opomenout. Porušením kterékoliv jiné už nelze aplikaci považovat za RESTful.

1.2.1.1 Model klient–server

Rozhraní odděluje klienta od serveru, což má za důsledek separaci zodpovědností. Funkcionality systémů se nepřekrývají a každá z částí řeší pouze úlohy, ke kterým je určená [20]. Server se například stará o persistenci dat a klient o uživatelské rozhraní. Tím získáváme volnost server libovolně škálovat a zároveň možnost jak klientskou, tak serverovou část kdykoliv nahradit odliš-

nou implementací. Ovšem výrok je pravdivý pouze za předpokladu, že bude zachováno původní navržené rozhraní.

1.2.1.2 Bezstavovost

Komunikace mezi klientem a serverem obsahuje veškeré informace pro vyřízení jeho požadavku v rámci relace. Žádné z nich nejsou perzistentně uloženy na serveru, a proto se o další vyvolání relace a rekci na předchozí odpověď musí postarat klient [20]. Zjednodušeně, o každý požadavek se server stará, jako by byl první, který od klienta přijal.

1.2.1.3 Cache

S cache, jinak česky řečeno mezipamětí se v informatice můžeme setkat na mnoha místech. Nejen že jde o funkci webového prohlížeče, který využívá cache pro krátkodobé ukládání načtených stránek a jejich částí [21], ale objevuje se i v problematice databází. K rozhodnutí dochází s každou přijatou odpovědí, jestli budou data uložena, či nikoliv. Tím se snažíme předejít skutečnosti, že by se pracovalo se zastaralými nebo jinak nevhodnými daty. Instalace cache spolu s její rozhodovací politikou omezí do určité míry komunikaci mezi klientem a serverem, což ve finále zapříčiní posílení výkonu serveru.

1.2.1.4 Vrstvitelnost systému

Klient nemusí vždy přímo komunikovat se zdrojovým serverem, ale může být spojený skrze zprostředkovaný server, který odlehčuje zátěž zdrojového. Zprostředkovaný server může zlepšit škálovatelnost tím, že umožní vyrovnávání zátěže a zajistí sdílenou cache [20].

1.2.1.5 Code on demand

Server může dočasně rozšířit či přizpůsobit funkcionalitu klienta převodem výkonného kódu. V praxi často využívaným přístupem jsou spustitelné skripty na straně klienta. Převážně se jedná o rozšíření v JavaScriptu.

1.2.1.6 Uniformita rozhraní

Zásadním omezením celého konceptu REST je jednotné a neměnné rozhraní. Odděluje tím server od klienta a jednotlivé části systému mohou být vyvíjeny nezávisle na ostatních. S tím souvisejí patřičné zásady rozhraní, které popisují principy rozhraní REST. Mezi ně patří principy: jak budou jednotlivé zdroje identifikovány, jak bude probíhat manipulace mezi nimi, způsob přechodu klienta mezi dílčími relacemi a způsob, jak je kladen důraz na úplnost zpráv zasílané serverem [20].

Tabulka 1.1: Seznam HTTP metod v rámci webové služby REST

Metody	Použití v REST	Odesílaná data
GET	Získání konkrétní reprezentace zdroje.	Všechna data jsou v URI daného zdroje.
POST	Vytvoření nového objektu v systému.	XML je nejčastější formát v těle zprávy odesílaných dat.
PUT	Změna již existujícího objektu.	Obdobně jako u metody POST.
DELETE	Smazaní objektu ze systému.	Objekt ke smazání je jednoznačně určen svým URI.
HEAD	Získání informace, která popisuje formát zdroje.	Obdobně jako u metody GET.

1.2.2 RESTful webové služby

Předchozí sekce byly věnovány omezujícím podmínkám architektury REST. Za RESTful rozhraní budeme považovat takové, které neporušuje ani jednu z nich, mimo nepovinné (Code on demand). Kromě toho je RESTful webová služba specifická svou implementací využívající HTTP (Hypertext Transfer Protocol) protokolu [20].

Všechny webové služby, které jsou zastoupeny kolekcemi zdrojů, musí splňovat určité vlastnosti. První z nich je výchozí URL (Uniform Resource Locator), tedy např. `http://webservice.com/resources/`, pod kterou je služba reprezentována navenek. Dalším aspektem je podpora konkrétního internetového média, kterým obvykle je XML (Extensible Markup Language), JSON (JavaScript Object Notation) či YAML (Ain't Markup Language) [22]. Nicméně není to nutnou podmínku, ale veškeré operace podporované webovou službou musí využívat metody protokolu HTTP. Zároveň s tím musí být poskytnuté API (Application Programming Interface), tzv. hypertext driven [23].

Tabulka 1.1 obsahuje stručný popis HTTP metod v implementaci RESTful [24].

Kapitola 2

Analýza

Úvod, který pouze naznačil problematiku spojenou s vyhledáváním vzorů v datových úložištích, ale věcného záměru se dotkl jen zlehka, v této kapitole rozebereme podrobněji. Představíme si základní přehled grafových databází a důvod, proč se v posledních letech stávají trendem pro ukládání vysoce propojených dat.

Nabízí se kolem dvou desítek možností, ovšem na zvážení nám postačí níže zmíněná šestice [25]. Po představení si následně popíšeme současný stav indexování ve zvolené grafové databázi a neformálně si shrneme představy, které očekáváme od výsledku diplomové práce, jakmile bude vypracována. Toto neformální deklarování si následně exaktně specifikujeme v podobě funkčních a nefunkčních požadavků. V samotném závěru kapitoly si popíšeme jednotlivé případy užití spolu s obecným dotazováním na grafovou databázi pro celkově lepší představu a pochopení.

2.1 Deklarace záměru

Hlavním cílem práce je analýza možností indexování grafových vzorů v grafových databázích, následný návrh a implementace pro zvolenou grafovou databázi a kompletní výkonnostní testování několika typických vzorů na vhodných vzorových datech.

2.2 Aktuální stav

Žijeme ve světe plném domén, které mají mezi sebou konexe. Nenahlížíme už na informace, jako na izolované kousky dat, a právě proto začaly vznikat grafové databáze, které zahrnují vztahy jako základní aspekt svého datového modelu. Umožňují nám účelně ukládat vysoce propojená data, následně se nad nimi efektivně dotazovat a získávat z nich méně zjevné a netriviální informace.

2.2.1 Grafová databáze

Grafové databáze jsou ideálním úložištěm pro vysoce propojená data, jak již bylo představeno v úvodu kapitoly. Za grafovou databázi lze z obecné definice označit kterýkoliv systém, který poskytuje bezindexovou sousednost, jinak řečeno každý vrchol obsahuje přímé odkazy na své sousedy. Tím odpadá nutnost používání indexů [9].

Grafové databáze nativně ukládají grafové struktury, za pomocí kterých se v nich eliminují ty nepříjemné problémy s uložením grafu do relační databáze, která je pro tyto účely nepřirozená [26], jelikož se neobejde bez výpočetně náročných operací JOIN.

Ideálním řešením jsou grafové databáze také pro mapování na objektovou strukturu u aplikací bez rigidního modelu. To zaručuje volnost schématu se libovolně rozvíjet [22]. Oproti tomu relační databáze přirozeně velmi dobře pracují s tabulárními daty, nad kterými vykonávají agregační a jiné, často se opakující dotazy [5].

Z důvodu progresivity sociálních projektů, u kterých nám velmi záleží na propojenosti mezi daty a následném vytěžování dat a informací, se stávají budoucností úložišť [27]. Jsou naprosto ideálním řešením pro grafové úlohy, jako jsou například nalezení nejkratších cest či vyhledávání komunit v sociální síti [22].

2.2.1.1 Sociogram

Za sociogram v internetovém kontextu považujeme social graph, což je grafické ztvárnění sociálních vztahů [28]. Můžeme například znázorňovat vztahy podle různých kriterií. Mezi ně řadíme sympatie, případně i antipatie, nebo také vztahy vlivu, komunikace aj. [29]. Indexování vzorů v sociogramu, bude jednou z hlavních částí výkonnostního testování, kde půjde o přátelství mezi osobami.

2.2.1.2 Interest graph

Rozdíl oproti sociogramu tkví v tom, že není nutná známost mezi osobami, ale zda tyto osoby mají spolu nějaké společné zájmy [30].

2.2.2 Přehled grafových databází

Už v názvu práce máme řečeno, na které grafové databázi budeme index realizovat. Ovšem krom vybrané existuje mnoho dalších zajímavých grafových databází, pro které by bylo možné index implementovat. Pro přehled si u každé z nich ve zkratce popíšeme její základní vlastnosti a zajímavosti, kterými se liší od ostatních.

2.2.2.1 Titan

Titan je škálovatelná grafová databáze od vývojářského týmu Aurelius, která je optimalizována pro ukládání a dotazování nad grafy, jež jsou ukládány v distribuovaném prostředí. Právě díky této vlastnosti dokáže obhospodařovat tisíce souběžně dotazujících se uživatelů nad grafem v reálném čase. Specifickou vlastností databáze je, že jako interní úložiště dat je využita jedna z databázových technologií Apache Cassandra, Apache HBase nebo Oracle BerkeleyDB [31], ke kterým vývojáři mohou přistupovat skrze jednotné Blueprints grafové API.

Začátkem roku 2015 firma DataStax oznámila, že spolu s týmem Aurelius budou spolupracovat na nové grafové databázi. Celý tým se stěhuje do sídla DataStaxu a budoucnost projektu Titan se stává nejasnou [32].

2.2.2.2 FlockDB

Společnost Twitter vyvíjí FlockDB pod licencí *Apache License, Version 2.0* za účelem analytiky souvisejících vztahů. První stabilní release byl vydán roku 2012 [25]. Na portálu DevWebPro jsou stručně, ale velmi dobře popsány Michaelem Marrem hlavní rysy databáze [33]: „The biggest difference between FlockDB and other graph databases like Neo4j and OrientDB is graph traversal. Twitter's model has no need for traversing the social graph. Instead, Twitter is only concerned about the direct edges (relationships) on a given node (account). For example, Twitter doesn't want to know who follows a person you follow. Instead, it is only interested in the people you follow. By trimming off graph traversal functions, FlockDB is able to allocate resources elsewhere.“ Obratem mu na to bylo odpovězeno, že bez traverzovaní se nejedná o grafovou databázi, ale o tzv. *persisted graph* [34].

Ačkoliv se podle definic jedná o cokoliv, můžeme od FlockDB očekávat velké věci. Profiluje mimo zmíněné další cíle, jako maximální výkon pro operace add/update/remove. Nadále podporuje stránkování výsledků, které obsahují miliony záznamů. Obsahuje množinu aritmetických funkcí, což je další atypičností v prostředí grafových databází. V neposlední řadě poskytuje horizontální škálování s podporou online migrace či replikace [35].

2.2.2.3 Sones GraphDB

Sones GraphDB je vyvíjena německou firmou Sones v jazyku C# [25]. Svou strukturou je grafem s hranami ohodnocenými reálnými čísly, jedná se tedy o vážený graf [36]. Databázi je možné provozovat jak na platformě pod Microsoft .NET frameworkem, ale také Mono, které zajistí její multiplatformnost [36]. I jako SaaS (Software as a Service) lze databázi provozovat na cloudové platformě Amazon S3 nebo Microsoft Azure, což implikuje několikanásobně vyšší výkon [25]. Přístup k datům nám poskytuje uniformní REST

API rozhraní (Representational state transfer Application programming interface), ale také další, jako jsou (např. Java, C#, WebShell, WebDAV) [36].

GraphDB je dostupná pod dvěma licencemi. První je open-source *AGPLv3* (Affero General Public License, version 3) komunitní verze, která je ale omezena určitou funkcionalitou oproti druhé, komerční enterprise verzi [36].

2.2.2.4 AllegroGraph

AllegroGraph je uzavřený produkt tvořený vývojáři Franz, Inc. Databáze je uzpůsobena tak, aby splňovala standardy W3C (World Wide Web Consortium) pro Resource Description Framework, neboli je určena pro manipulaci s propojenými daty a sémantickým webem [25]. Nabízí obsáhlý seznam knihoven pro komunikaci skrze REST API s aplikacemi vyvýjenými například v programovacích v jazycích C#, Perl, Ruby či Scala.

Licenční politika vypovídá o verzích – Free, Developer a Enterprise. U prvních dvou je omezení v počtu uložených záznamů v databázi [37]. Mezi hlavní klienty se řadí velikáni jako jsou Ford, Kodak nebo NASA (National Aeronautics and Space Administration) [25].

2.2.2.5 Neo4j

Databáze Neo4j je nejpopulárnější ve své kategorii. Implementovaná je v Javě [25], o čemž se můžeme přesvědčit na `https://github.com/neo4j`, jelikož se jedná o open-source řešení od společnosti Neo Technology. První release se objevil roku 2007 [38] a od té doby získala velké zastoupení ve vývojářské komunitě, která se zapojuje do běžných diskuzí na `https://groups.google.com/group/neo4j` a `http://stackoverflow.com/questions/tagged/neo4j`.

Grafová struktura Neo4j tvoří vrcholy, hrany a vlastnosti, pro které platí limity v řádech desítek miliard. *Vlastnosti* jsou velkou přidanou hodnotou, kterou může obsahovat každý vrchol nebo hrana ve formě klíč–hodnota. Jedná se o už námi definovaný *property graph database*.

Databáze Neo4j může být použita jako *embedded mode(Java, JAR)* (Java Archive), nebo *standalone mode*, prostřednictvím kterého můžeme k databázi přistupovat skrze REST API [5].

2.2.2.6 InfiniteGraph

Jádro grafové databáze je implementováno v C++. Ostatní části už jsou v jazyce Java. Společnost Objectivity, která už svým názvem odkazuje k databázové struktuře, která je objektově orientovaná, vydala první verzi roku 2010. Sestává se z objektů (v grafové terminologii myšleno vrchol) a hran pro jejich propojení. Bohužel zatím není možné k databázi přistupovat prostřednictvím rozhraní REST API, ale je možné využít pouze rozhraní v Javě [39].

Licence jsou poskytovány ve dvou variantách. Bezplatná verze nese s sebou značné limitace v podobě omezení počtu hran a vrcholů po vypršení 60denní

zkušební verze [40]. Nicméně se Objectivity může pyšnit hlavními klienty, mezi které patří vláda Spojených států amerických a poradenská společnost Deloitte [41].

2.2.2.7 Shrnutí

Už během sepisování zadání jsme si zvolili Neo4j. Hlavním důvodem byl její agilní vývoj a značná komunita v pozadí. V další podsekci si popíšeme její architekturu a systém indexování uzlů a hran, od kterého budeme nadále vycházet při návrhu a specifikaci požadavků.

V následujících sekcích budeme už pracovat s Neo4j databází. Proto změníme ontologický slovník v pojmech vertex a edge a převezmeme terminologii z dokumentace Neo4j, kde se o hraně mluví jako o *relationship* a o uzlu jako *node*.

2.2.3 Architektura Neo4j

Základní pohled na architekturu Neo4j máme rozdělenou do třech hlavních částí. Pro komunikaci a manipulaci s daty máme k dispozici Java API, REST API a další API rozhraní, která jsou vystavená uživatelům databáze [42].

Objekt Cache nám udržuje datové soubory obsahující Java objekty a File System Cache. Ta udržuje části souborového systému pro urychlení I/O operací, u nichž využívá již načtená data místo pomalejšího čtení z pevného disku [42].

Transaction Management nám zajišťuje ACID, neboli atomicitu, konzistenci, izolovanost a trvalost [43]:

Atomicita – Jestliže jakákoliv část transakce selže, tak databáze zůstává nezměněna.

Konzistence – Každá transakce končí v konzistentním stavu.

Izolovanost – Během transakce nemohou být modifikovaná data přístupná ostatním operacím.

Trvalost – Změny, které byly potvrzeny jako úspěšné, jsou uložené v DBMS (Database management systems) a již nemohou být ztraceny.

2.2.3.1 Úložiště datových souborů

Neo4j ukládá data do několika rozdílných datových souborů. Každý obsahuje specifické části grafu jako jsou například uzly, hrany nebo vlastnosti [44]:

neostore.nodestore.db

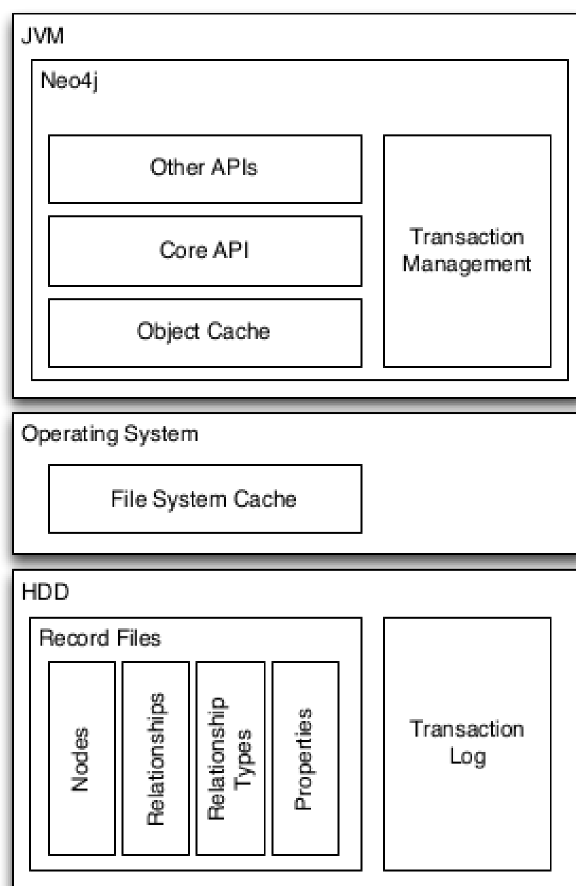
neostore.relationshipstore.db

neostore.propertystore.db

neostore.propertystore.db.index.keys

neostore.propertystore.db.strings

neostore.propertystore.db.arrays



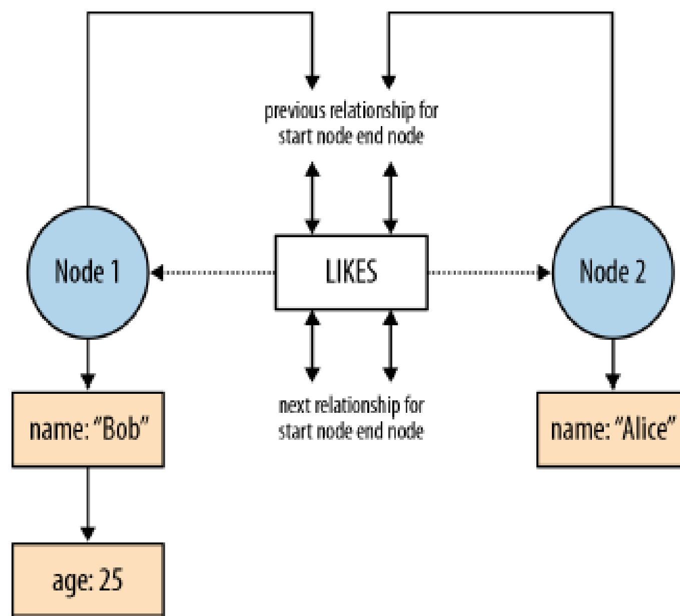
Obrázek 2.1: Architektura Neo4j [45]

2.2.3.2 Úložiště uzlů

Uzly jsou uloženy v souboru neostore.nodestore.db, kde má každý záznam fixní délku 9 bajtů. Pak je možné za pomocí ID (Identification) uzlu vypočíst offset v úložišti v konstantním čase. Tím se přesně dostaneme na danou pozici, na které máme uložené další informace. První z nich je 1 bajtová značka, jestli je daný uzel používaný. Další 4 bajty reprezentují ID první hrany a poslední 4 bajty ID první vlastnosti uzlu [44].

2.2.3.3 Úložiště hran

Stejně tak i hrany mají fixní délku. Každý záznam obsahuje 33 bajtů pro přesné odkázání do úložiště neostore.relationshipstore.db. I zde první bajt značí informaci, zda je daná hrana používaná. Nadále máme ID na počáteční a koncový uzel hrany, ukazatel na typ hrany, ukazatele na předchozí a následující hrany počátečního a koncového uzlu a nakonec odkaz na ID následující vlastnosti.



Obrázek 2.2: Reprezentace fyzického úložiště Neo4j [45]

.

2.2.3.4 Úložiště vlastností

Jak už bylo několikrát zmíněno, vlastnosti jsou reprezentovány dvojicí klíč–hodnota. Klíče jsou Java řetězce a hodnoty mohou nabývat jakéhokoliv primitivního Java typu jako je řetězec, pole, numerická hodnota atd. [45].

I vlastnosti mají fixní délku. Neliší se velikostí ani strukturou v závislosti, jestli se váží k uzlu nebo hraně. Jedná se o 41 bajtový záznam, který obsahuje 36 bitů pro ukazatele na předchozí a následující vlastnost a čtyři 8 bajtové bloky.

Na obrázku 2.4 je znázorněn detailní obsah druhého bloku. Nejmenší 4 bitová část označená písmenem t zastupuje ukazatel na typ hrany a určuje typ vztahu. Následující 3 bajtový úsek označený k se využívá pro odkaz do úložiště klíčů, kde už jsou jejich řetězcové reprezentace. Poslední 4 bajty označené

písmenem *v* jsou vyhrazeny hodnotu. Pokud je větší jak 4 bajty a menší jak 24 bajtů, tak se uloží do zbývajících bloků. V opačném případě se hodnota uloží do speciálního úložiště pro hodnoty s dynamickou velikostí a do 4 bajtového bloku se uloží odkaz do tohoto úložiště a další bloky se mohou využít pro jiné vlastnosti [45].

Node Record in the Node Store (9 bytes), first bit = inUse flag

next relationship (35 bits)	next property (36 bits)

Relationship Record in the Relationship Store (33 bytes), first bit = inUse flag, second bit unused

first node (35 bits)	second node (35 bits)	type (16 bits)	first node's previous relationship (35 bits)	first node's next relationship (35 bits)	second node's first relationship (35 bits)	second node's next relationship (35 bits)	next property (36 bits)

Property Record in the Property Store (41 bytes)

next property record (36 bits)	previous property record (36 bits)	property block 1 (8 bytes)	v	v	v	v		t	k	k	k	property block 3 (8 bytes)	property block 4 (8 bytes)

Obrázek 2.3: Datové úložiště Neo4j [44]

2.3 Budoucí stav

Během studia grafových databází jsme se přesvědčili o absenci indexu pro grafové vzory. Povětšinou jsou indexované uzly, hrany a jejich vlastnosti. V úvodu jsme si představili několik reálných případů hledání grafových vzorů, kde by jejich indexování bylo obrovských přínosem pro svět grafových databází.

V našem případě jsme si vybrali grafovou databázi Neo4j, pro kterou navrhneme index pro grafové vzory a implementujeme jeho rozhraní skrze webovou službu REST. Rozhraní bude umožňovat vytvoření, mazání indexu a získání vzorů ve formátu JSON. Níže si exaktně specifikujeme funkční a nefunkční požadavky, které chceme, aby rozšíření Neo4j splňovalo.

2.4 Požadavky

2.4.1 Funkční požadavky

Grafový index bude poskytovat rozhraní skrze webovou službu REST.

Rozhraní bude umožňovat vytvoření indexu POST požadavkem na *http://.../indexName/pattern*

Rozhraní bude umožňovat smazání indexu DELETE požadavkem na

http://.../indexName

Rozhraní poskytne grafové vzory ve formátu JSON GET požadavkem na *http://.../indexName/query*

Parametr *pattern* bude definovaný syntaxí dotazovacího jazyka Cypher v podobě *(a)-[d]-(b)-[e]-(c)*. Každý uzel a hrana bude reprezentována proměnnou, což je nutná podmínka pro jednoznačnost.

Parametr *pattern* není možné zadat v nejednoznačné podobě jako např. *()-[d]-()-[]-(c)*, jelikož takto zadaný vzor může znamenat rovnou čáru, trojúhelník atd.

Vzor (parametr *pattern*) bude vždy vytvářen v obecné podobě bez značek a vlastností.

Parametr *query* bude definovaný syntaxí dotazovacího jazyka Cypher v podobě *MATCH <pattern> WHERE <condition> RETURN <expr>*.

Rozhraní bude umožňovat vytvoření indexu POST požadavkem na *http://.../indexName/pattern*

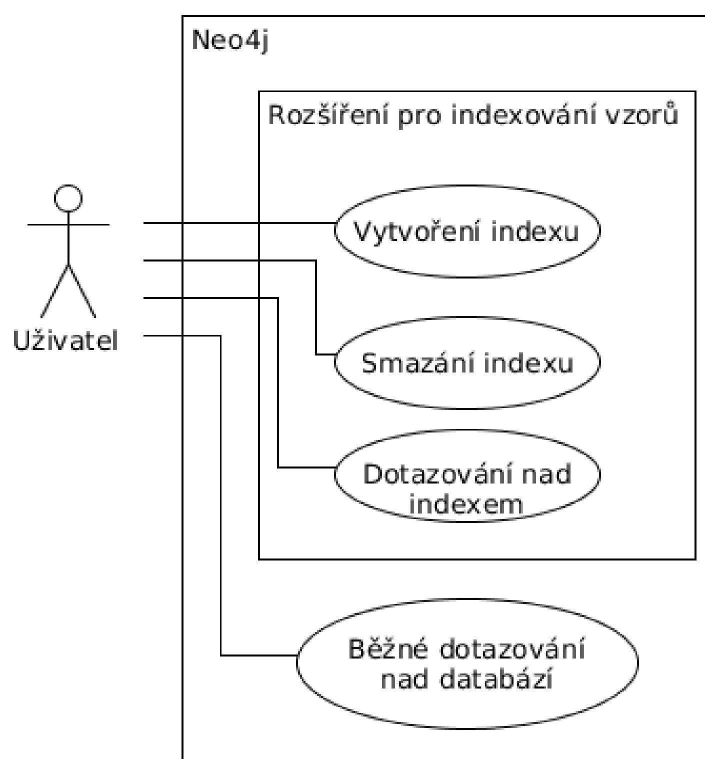
Grafový index nebude v rámci prototypu obsahovat dvoufázový potvrzovací protokol.

2.4.2 Nefunkční požadavky

Rozšíření nijak neovlivní dosavadní výchozí dotazování Neo4j ani jiné fungování grafového úložiště.

Rozšíření bude navrženo abstraktně, aby úložiště indexu bylo možné kdykoliv vyměnit za jiné.

Rozšíření bude využívat GraphAware framework.



Obrázek 2.4: Model případů užití [46]

Kapitola 3

Návrh

Inspirací pro návrh grafového indexu nám budou relační databáze, jejichž vznik sahá až do druhé poloviny minulého století. Možnosti pro indexování záznamů v relačních databázích byly již akademicky dokázány a prakticky dovedeny do výkonnostních maxim. Dnes se s nimi máme možnost setkat v produkčních systémech u komerčních i nekomerčních projektů.

Po získání teoretických základů si navrhneme index pro indexování vzorů v grafových databázích. Určíme si asymptotické meze nynějšího bezindexového vyhledávání grafových vzorů. Následně si popíšeme data, která budeme chtít v indexu uchovávat, a rozhodneme, jaké úložiště by bylo nejvhodnějším řešením.

3.1 Index pro tabulární data

U relačních databází se s indexem setkáme při výběru z jednoho nebo více sloupců, pro které chceme dotazování urychlit. Uložení a chování indexu se i výrazně může lišit podle konkrétního databázového produktu.

Vytvoření zapříčiní vyšší nároky na operační paměť a diskový prostor – s tím, že samotná základní data nejsou nijak ovlivněna. Naopak se změní práce databáze, která musí při jakékoliv aktualizaci nové informace promítnout jak do datových tabulek, tak i do struktury indexu. Jinými slovy, vytvoření indexu nám zapříčiní zpomalení ukládání nových dat, ale značně zrychlí jejich čtení.

Informace uložené v paměťovém prostoru, které určují rozmístění hodnot indexovaných sloupců, jsou povětšinou v nečitelné podobě a jsou předpřipravené pro práci vnitřních algoritmů databáze. Pokud nastane vyvolání dotazu, který se týká indexovaného sloupce, potom filtrovaná hodnota není hledaná podle toho, jak jsou řádky v tabulce uložené za sebou, ale pomocí výše zmíněných informací. Zjednodušeně můžeme index přirovnat k rejstříku knihy, který obsahuje odkaz na příslušnou stránku. V našem případě se jedná o ukazatel na paměťové místo s požadovanými daty.

3.1.1 Implementace indexů

Index může být implementovaný různými strukturami. Nejpopulárnějšími jsou B stromy, B+ stromy a Hash tabulky. Žádná z důvodu optimalizace neuchovává samotná data, ale pouze jejich paměťovou adresu (ukazatel). V tabulce 3.1 si představíme jejich asymptotické složitosti [47] [48] [49].

Tabulka 3.1: Asymptotické složitosti algoritmů

	B strom		B+ strom		Hash tabulka	
	Průměr	Nejhůře	Průměr	Nejhůře	Průměr	Nejhůře
Vyhledání	$O(\log(n)$	$O(\log(n)$	$O(\log_b(n)$	$O(\log_b(n)$	$O(1)$	$O(n)$
Vložení	$O(\log(n)$	$O(\log(n)$	$O(\log_b(n)$	$O(\log_b(n)$	$O(1)$	$O(n)$
Smazání	$O(\log(n)$	$O(\log(n)$	$O(\log_b(n)$	$O(\log_b(n)$	$O(1)$	$O(n)$

* n značí počet záznamů a b je faktor větvení

3.2 Návrh indexu

Z předchozích sekcí jsme si odnesli několik zajímavých poznatků, které využijeme pro návrh indexu grafových vzorů. Nebudeme ukládat veškerá data, ale pouze ID uzlů a hran, která nám zaručí jednoznačný popis vzoru. Postačily by pouze hrany, ale rychlý odkaz do databáze skrze ID je možný v Neo4j pouze z uzlu [50]. Hrana se nepovažuje za startovní prvek pro popis hledaného vzoru, a tudíž musíme do indexu zahrnout i uzly.

Pro ukládání si zvolíme Hash tabulku. Každý zaindexovaný vzor bude reprezentován jednou tabulkou, která bude obsahovat strukturu znázorněnou v ukázce 3.2. Klíč (řádek) Hash tabulky bude reprezentovat zaindexovaný vzor (seznam uzlů a hran). První část klíče jsou ID uzlů seřazeny vzestupně a odděleny jedním podtržítkem. Následuje oddělení dvěma podtržítky mezi seznamem uzlů a seznam hran. Druhá část klíče je seznam ID hran, které jsou stejně jako uzly seřazeny vzestupně.

3.2.0.1 Vyhledávání vzoru v Neo4j

Neo4j nabízí několik možností, pomocí kterých se lze nad grafovou databází dotazovat. Hlavními z nich jsou Cypher a nativní Java API. Rozdíl mezi nimi je v přístupu – první jmenovaný je deklarativní a druhý imperativní.

Ve funkčních požadavcích jsme si určili, že veškeré dotazování nad indexem bude totožné se syntaxí Cypheru (*MATCH <pattern> WHERE <condition> RETURN <expr>*). Hlavním důvodem je dodržení standardů Neo4j a jednoduchost přenesení takovéhoto dotazu skrz námi navržené REST API.

Tabulka 3.2: Příklad struktury indexu pro vzor trojúhelníku

triangle	
key	value
_nodeId11_nodeId27_nodeId34___relId1009_relId4555_relId4565_	
_nodeId18_nodeId24_nodeId65___relId1203_relId2743_relId3001_	
_nodeId48_nodeId52_nodeId87___relId1078_relId2225_relId4563_	
...	
...	
...	
_100_250_290___1001_2500_5252_	
_10_25_850___1011_2200_7252_	
_20_525_695___2201_2500_4252_	

Složitost

Pokud chceme v databázi vyhledat všechny obecné trojúhelníky, tak vykonáme Cypher dotaz:

```
MATCH (a)--(b)--(c)--(a) RETURN a,b,c
```

Složitost takového dotazu je $O(|V| * (|V_P| + |H_P|))$, jelikož z každého uzlu se dohledává vzor algoritmem DFS (ve výchozím nastavení traverzovacího algoritmu) [51].

3.2.1 Úložiště indexu

V této sekci si představíme základní přehled databází, které splňují svou strukturou úložiště, které bude sloužit k ukládání informací o grafových vzorech, které jsme si popsali v tabulce 3.2. Ač nám je nabízeno několik desítek možností, budeme zvažovat pouze níže zmíněnou trojici.

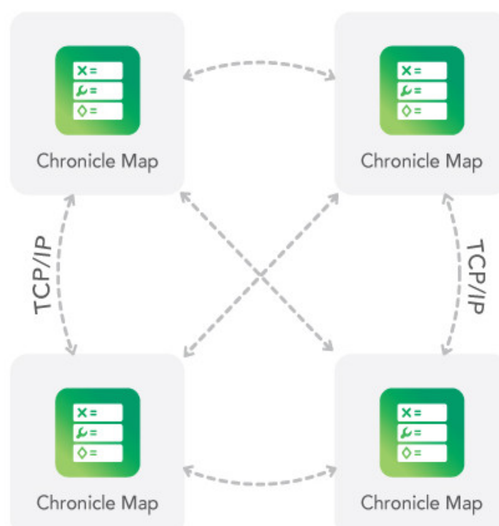
3.2.2 MapDb

MapDB je embedded databáze, která poskytuje přístup ke kolekcím TreeMap a HashMap s možností permanentního uložení na pevný disk. Rychlost, škálovatelnost a snadná použitelnost jsou hlavními charakteristikami, kterými se pyšní společnost CodeFutures, jež sponzoruje vývoj databáze.

Celková velikost rozšíření se pohybuje okolo 250 kB, které kromě zmíněných funkcí obsahuje transakce, serializaci, kompresi dat, šifrování a mnoho dalšího. Kompletní přehled nalezneme na oficiálních stránkách `http://www.mapdb.org/#support`.

3.2.3 Chronicle-Map

Chronicle-Map je open-source projekt publikovaný pod licencí Apache 2.0. Práce s databází je srovnatelná s přístupem ke klasickým Java Map kolekcím. Uložená data jsou automaticky distribuovaná skrze TCP/IP nebo UDP do paměti všech serverů 3.1 (s možností uložení na pevný disk).



Obrázek 3.1: Distribuce dat v Chronicle-Map [52]

3.2.4 Voldemort

Voldemort je distribuovanou databází, kterou využívají velikáni jako je LinkedIn. Struktura je navržena k simulovaní Haspmapy pro ukládání párů klíč–hodnota.

Projekt je stále ve vývoji a vizí projektu je vytvořit velkou, distribuovanou a vůči chybám odolnou perzistentní Hast tabulku. Nikoliv další relační či objektovou databázi splňující vlastnosti ACID.

3.2.5 Shrnutí

Na základě požadavků na strukturu a funkčnost použijeme úložiště MapDb. Svojí jednoduchostí a výkonnostními výsledky `http://www.mapdb.org/benchmarks.html` se jeví jako ideální volba pro úložiště indexu.

U zmíněných alternativ by bylo velmi zajímavé vyzkoušet správu indexu v distribuovaném prostředí. Bohužel jakékoliv hlubší testování alternativních úložišť je nad rámec tématu, ovšem jeví se jako velmi zajímavá studie, která by mohla vycházet z této práce.

Kapitola **4**

Vyhodnocení dotazů v Neo4j

Předtím než přejdeme k implementaci indexu, potřebujeme zjistit, jaké dotazování je nejlepší pro dohledávání zaindexovaných vzorů podle návrhu 3.2. V úvodu kapitoly si představíme testovací prostředí a Graphaware framework, u kterého použijeme modul pro testování `https://github.com/graphaware/neo4j-framework/tree/master/tests`. Následovat bude popis databáze, nad kterou budeme provádět testování dotazů, a nakonec si rozebereme výsledky. Každý dotaz si slovně popíšeme, uvedeme jeho zápis v jazyku Cypher a vysvětlíme si naměřené výsledky, které jsme zanesli do grafů. U všech měřeních jsme vždy měnili typ cache, které si také z počátku podrobně popíšeme.

Jak už jsme zmínili na začátku, otestováním Neo4j dotazů pro dohledávání grafových vzorů s dodatečnou informací, která je popisuje (informace v řádku indexu 3.2), získáme podklady pro následující implementační část práce.

4.1 Konfigurace testovacího prostředí

K testování výkonu Neo4j dotazů a finální výkonu indexu použijeme prostředí 4.1. K porovnání výsledných řešení přejdeme na prostředí s konfigurací v tabulce 4.2.

Tabulka 4.1: Hardware konfigurace testovacího prostředí 1

Parametr	Popis
Výkon procesoru	Intel®Core™2 Duo Processor T8300 2.4 GHz
Paměť RAM	4096 MB 667 MHz Dual-Channel DDR2 SDRAM (2x 2048 MB)
Diskový prostor	1SSD 128 GB
Operační systém	Fedora 14 release x86_64 (Laughlin)

Na uvedených prostředí jsme u každého měření měnili typ cache, které máme k dipozici v Neo4j ve verzi 2.2.

Tabulka 4.2: Hardware konfigurace testovacího prostředí 2

Parametr	Popis
Výkon procesoru	Intel®Core^{TM}i5 2.4 Ghz
Paměť RAM	8 GB 1600 MHz DDR3
Diskový prostor	SSD 256 GB
Operační systém	OS X 10.9.5

4.1.1 Cache v Neo4j

Neo4j poskytuje dva typy cache: *file buffer cache* a *object cache*. *File buffer cache* ukládá do mezipaměti soubory úložiště ve stejném formátu, jakým jsou uloženy na pevný disk. *Object cache* ukládá do mezipaměti uzly, hrany a vlastnosti v optimalizovaném formátu pro traverzování grafem [53]. Veškerá konfigurační nastavení máme možnost změnit v souboru neo4j.properties ve složce conf v domovském adresáři databáze.

V podsekcích si popíšeme jednotlivé parametry pro oba dva typy cache a také pod čím budou prezentovány v jednotlivých typech výkonnostního testování.

4.1.1.1 Konfigurace file buffer cache

Parametr **dbms.pagecache.memory** určuje velikost paměti (v jednotkách bajtů) pro uložení souborů úložiště do paměti. Zvolená velikost je automaticky zaokrouhlena dolů na nejbližší celou stránku.

Tabulka 4.3: Konfigurace parametru dbms.pagecache.memory

Typ testu	Hodnota
NoCache	10k
LowLevelCache	500M
HighLevelCache	500M

4.1.1.2 Konfigurace object cache

Nastavení *object cache* určuje hodnota parametru **cache_type**. Hodnotou může být jakákoliv z následující pětice:

none – Žádné objekty nebudou uloženy do mezipaměti.

soft – Výchozí hodnota v Neo4j Community Edition. Zajišťuje optimální využití paměti. Vhodná pro vysoký výkon traverzování grafem, avšak během toho může nastat problém s GC (Garbage collection), pokud často procházené části grafu se nevejdou do mezipaměti.

weak – Určuje krátkou životnost objektů v mezipaměti. Jedná se o vhodné nastavení pro případy, kde se často přistupuje na větší část grafu, která se nevejde do mezipaměti.

strong – Poskytuje velmi dobrý výkon pro grafy, které se vejdou do mezipaměti. Veškeré údaje, které byly už načteny se v mezipaměti udrží a nebudou z ní uvolněny.

HPC – Tato hodnota nastavení je k dispozici pouze v Neo4j Enterprise Edition. Lze u ní určit konkrétní prostředky pro množství paměti uzlů, hran a vztahů. Pokud se prostředky správně přidělí, tak by měla být nejlepší volbou ze všech uvedených.

Tabulka 4.4: Konfigurace parametru cache_type

Typ testu	Hodnota
NoCache	none
LowLevelCache	none
HighLevelCache	strong

4.2 Graphaware framework

Cílem frameworku je převzetí typických problémů z oblasti grafových databází, čímž se zjednoduší vývoj tak, že architekti a programátoři se mohou soustředit pouze na svá řešení.

Rozděluje se do dvou hlavních částí – GraphAware Server a GraphAware Runtime. GraphAware Server umožní vyvíjet REST API s využitím Spring MVC (Model-view-controller). GraphAware Runtime je runtime prostředí pro embedded i serverový vývoj, které usnadní vývojářům vytvářet další rozšíření (moduly), které obohatí základní funkčnost grafové databáze [54]. Kromě výše framework nabízí další funkce, jakými jsou například:

testování REST API;

rozšíření Unit testů;

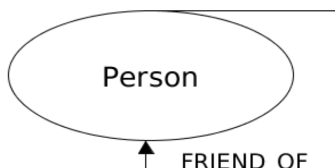
výkonnostní testování dotazů;

API pro správu transakcí;

řízení dávkových operací.

4.3 Erdős–Rényi

Graf Erdős–Rényi obsahuje uzly, mezi kterými jsou zcela náhodné interakce. Každý uzel se stává sousedem podle určité pravděpodobnosti a limitace na celkový počet vztahů v grafu [55]. Z toho vyplývá, že stupně uzlů nabývají rovnoměrného rozdělení.



Obrázek 4.1: Schéma databáze Erdős–Rényi [46]

4.4 Testování výkonu dotazů

Hledání trojúhelníků je nejtypičtějším příkladem, na kterém si můžeme názorně ukázat celé měření. V běžném životě se s tímto setkáme na sociálních sítích, kde nám je doporučováno přátelství s lidmi, kteří jsou přáteli našich přátel, a tím je zaručena určitá pravděpodobnost, že se s nimi také známe nebo v budoucnu seznámíme a uzavřeme trojúhelník přátelství [2]. Takováto analýza sítě je stavebním kamenem pro jakékoliv doporučující algoritmy, které jsou nad danou sítí aplikovány. Komplexita sítě, koeficient shlukování, celkový počet trojúhelníků a nebo průměrná pravděpodobnost pro nově vznikající vzory (vznik hrany značící přátelství) jsou informace, které lze využít například k identifikaci největších komunit sítě [3].

Pro výkonnostní otestování dotazů jsme si vygenerovali grafovou Erdős–Rényi komunitní síť o velikosti 10 000 uzlů a 50 000 hran. Výsledkem selektivních dotazů, bude vždy stejná množina obsahující všechny trojúhelníky v grafu. Měření vždy několikrát zopakujeme pro získání kvalitnější výsledků a eliminaci okolních vlivů.

4.4.1 Obecné vyhledání

Všechny níže testované dotazy, budeme porovnávat vůči běžnému dotazování Neo4j. Úkolem bude získat všechny vzory ve tvaru trojúhelníku, který bychom zapsali takto:

```
1 MATCH (a)-[d]-(b)-[e]-(c)-[f]-(a) RETURN id(a),id(b),id(c)
```

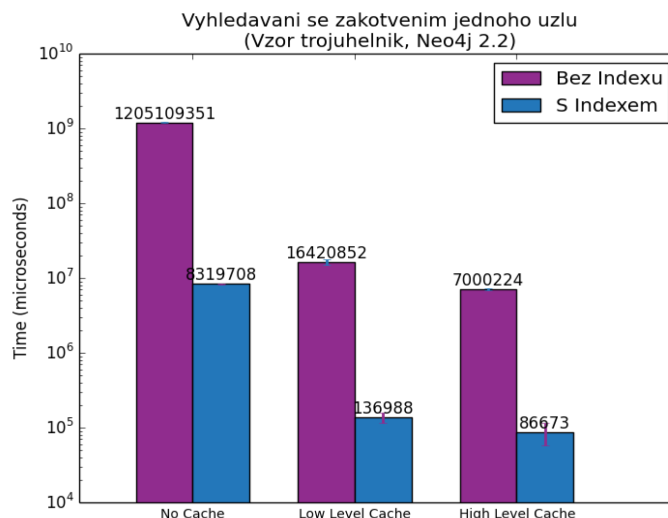
Pokud provedeme dotaz v *neo4j-shell* konzoli, s tím rozdílem, že před *MATCH* umístíme *EXPLAIN*, získáme prováděcí plán, který potvrzuje naše tvrzení z kapitoly 3.2.0.1. Takto obecně zapsaný dotaz projde přes všechny uzly a z každého dohledává trojúhelník.

4.4.2 Vyhledávání se zakotvením jednoho uzlu

Filtrací uzlu v klauzuli *WHERE* se určí počáteční uzel vzoru. U symetrických grafů počátek určit nelze (například trojúhelník), a proto je nutné měnit pozice přes všechny uzly *a, b, c*. Mezivýsledky spojí *UNION* do řešení, které je uloženo do množiny všech výsledků.

```
1 MATCH (a)--(b)--(c)--(a) WHERE id(a)=nodeId11 RETURN id(a),id(b),id(c)
2 UNION
3 MATCH (a)--(b)--(c)--(a) WHERE id(b)=nodeId11 RETURN id(a),id(b),id(c)
4 UNION
5 MATCH (a)--(b)--(c)--(a) WHERE id(c)=nodeId11 RETURN id(a),id(b),id(c)
```

U každého testu budeme v ukázkových dotazech dohledávat trojúhelník pro první záznam v tabulce indexu 3.2. Řešení obsahuje výsledky pro všechny řádky, které jsme iterovali přes celou tabulku a dotaz obdobně opakovali v každém cyklu. Asymptotická složitost dotazu se narozdíl od obecného vyhledávání redukuje na $O(|P|*(|V_P|+|H_P|))$, kde $|P|$ značí počet všech nalezených vzorů (počet řádků v Hash tabulce). Z toho lze jednoduše odvodit, že pokud $|P| \sim |V|$, jedná se o symetrický graf a složitost bude rovna obecnému dotazu. V opačném případě $|P| < |V|$ dochází ke značnému zrychlení 4.2.

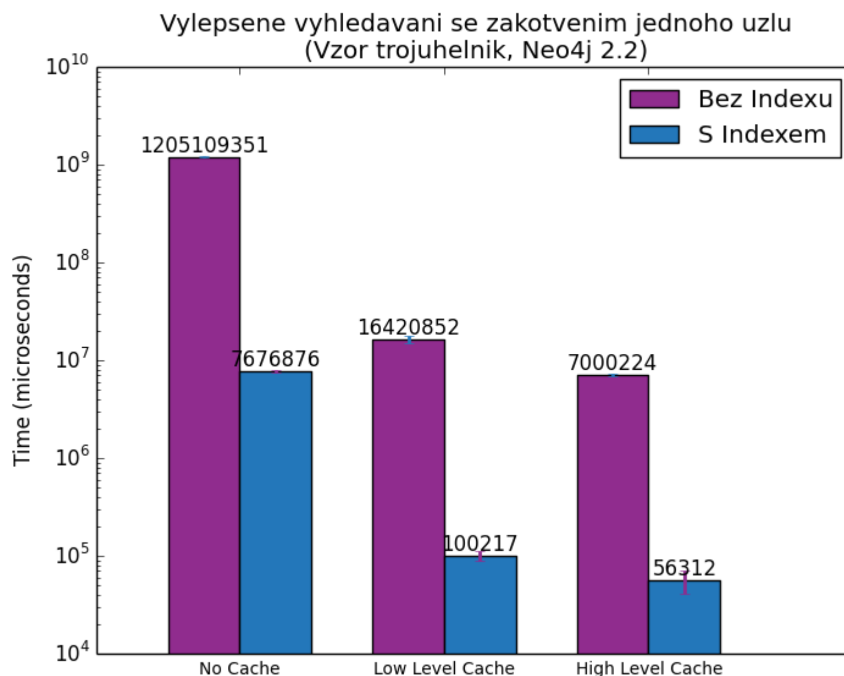


Obrázek 4.2: Výsledky vyhledávání se zakotvením jednoho uzlu

4.4.3 Vylepšené vyhledávání se zakotvením jednoho uzlu

Navržený index skrývá nepříliš viditelnou chytristiku. Uzly a hrany jsou vzestupně seřazeny podle ID. Vylepšení spočívá v tom, že v předchozím dotazu jsme určili počáteční uzel vzoru, první uzel z každé řádky tabulky indexu 3.2. Díky tomu eliminujeme dotazy o uzly, nad kterými jsme se již dotazovali, jelikož mezivýsledky už máme uložené v množině výsledků.

Značné zrychlení můžeme zpozorovat u grafů, které obsahují uzly zastoupené ve více vzorech a zároveň mají nejmenší ID ze všech uzlů ve vzoru.



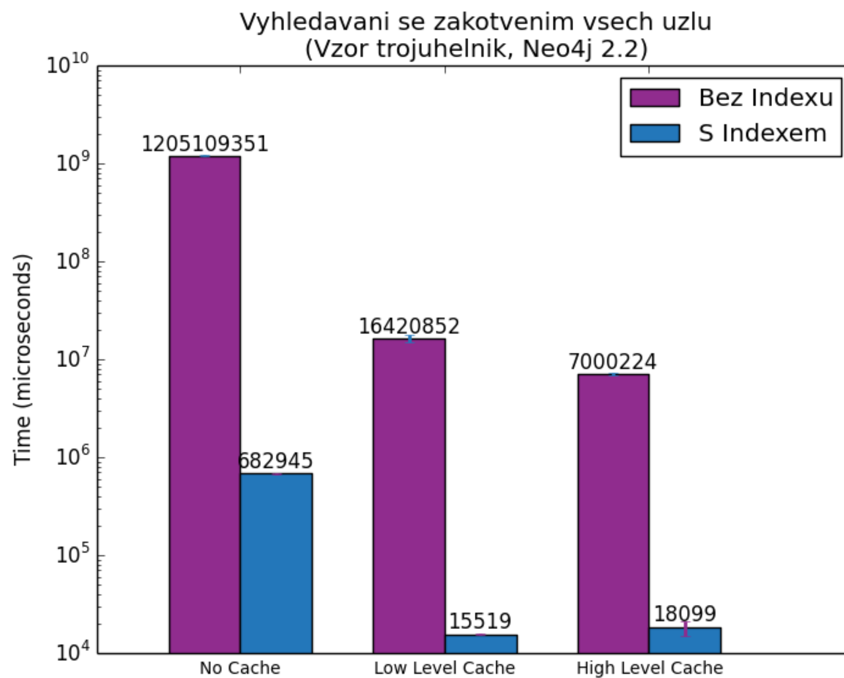
Obrázek 4.3: Výsledky vylepšeného vyhledávání se zakotvením jednoho uzlu

4.4.4 Vyhledávání se zakotvením všech uzlů

První část klíče v tabulce 3.2 je množina ID uzlů vzoru, nad kterou provedeme permutaci. Každý prvek permutace spojíme přes *AND* a mezivýsledky opět pomocí *UNION*.

Permutace nám generuje množinu všech možných uspořádání uzlů, kterých vzor může nabývat. Celkový počet je roven faktoriálu počtu uzlů, což povede k problémům u větších vzorů.

```
1  MATCH (a)--(b)--(c)--(a)
2  WHERE id(a)=nodeId11 AND id(b)=nodeId27 AND id(c)=nodeId34
3  RETURN id(a),id(b),id(c)
4  UNION
5  ...
6  ...
7  ...
8  MATCH (a)--(b)--(c)--(a)
9  WHERE id(a)=nodeId34 AND id(b)=nodeId27 AND id(c)=nodeId11
10 RETURN id(a),id(b),id(c)
```



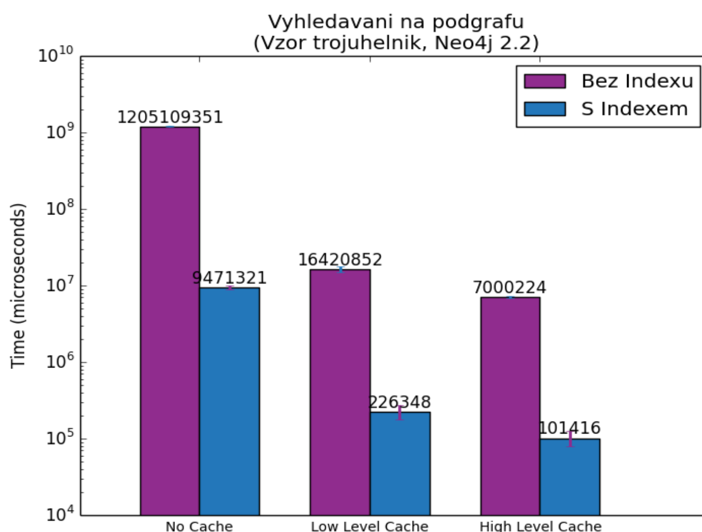
Obrázek 4.4: Výsledky vyhledávání se zakotvením všech uzlů

4.4.5 Vyhledávání na podgrafu

Asymptotická složitost pro odkázání do grafu skrze ID uzlu je $O(1)$. V dotazu se provede tzv. *NodeByIdSeek*, což vychází z architektury Neo4j, kterou jsme si popsali v sekci 2.2.3. I právě proto jsme začali s testováním předchozích dotazů.

Ideálním řešením by bylo, kdyby Neo4j podporovala dotazování na podgrafu, kde bychom definovali uzly a hrany, nad kterými by se provedl příslušný dotaz. Bohužel toto není prozatím podporováno [56], ale je možné zapsat dotaz obdobně v takovéto podobě:

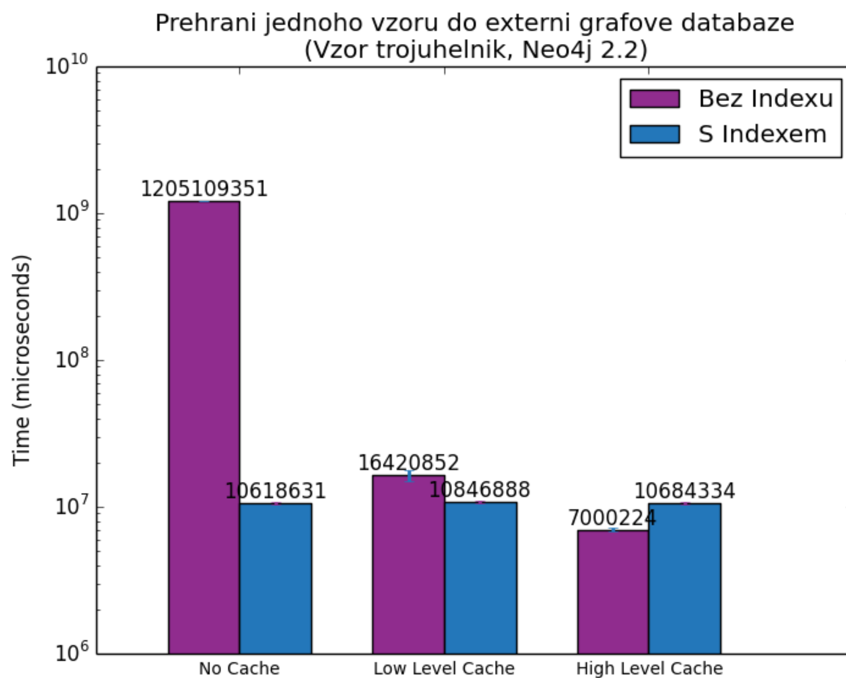
```
1  MATCH triangle=(a)-[d]-(b)-[e]-(c)-[f]-(a) WHERE id(a)=nodeId11
2  AND ALL (n in nodes (triangle) WHERE id(n)
       IN[nodeId11,nodeId27,nodeId34])
3  AND ALL (r in rels (triangle) WHERE id(r) IN[relId1009, relId4555,
       relId4565]) RETURN id(a),id(b),id(c)
4  UNION
5  MATCH triangle=(a)-[d]-(b)-[e]-(c)-[f]-(a) WHERE id(a)=nodeId27
6  AND ALL (n in nodes (triangle) WHERE id(n) IN[nodeId11, nodeId27,
       nodeId34])
7  AND ALL (r in rels (triangle) WHERE id(r) IN[relId1009, relId4555,
       relId4565]) RETURN id(a),id(b),id(c)
8  UNION
9  MATCH triangle=(a)-[d]-(b)-[e]-(c)-[f]-(a) WHERE id(a)=nodeId34
10 AND ALL (r in rels (triangle) WHERE id(r) IN[relId1009, relId4555,
       relId4565]) RETURN id(a),id(b),id(c)
```



Obrázek 4.5: Výsledky vyhledávání na podgrafu

4.4.5.1 Přehrání jednoho vzoru do externí grafové databáze

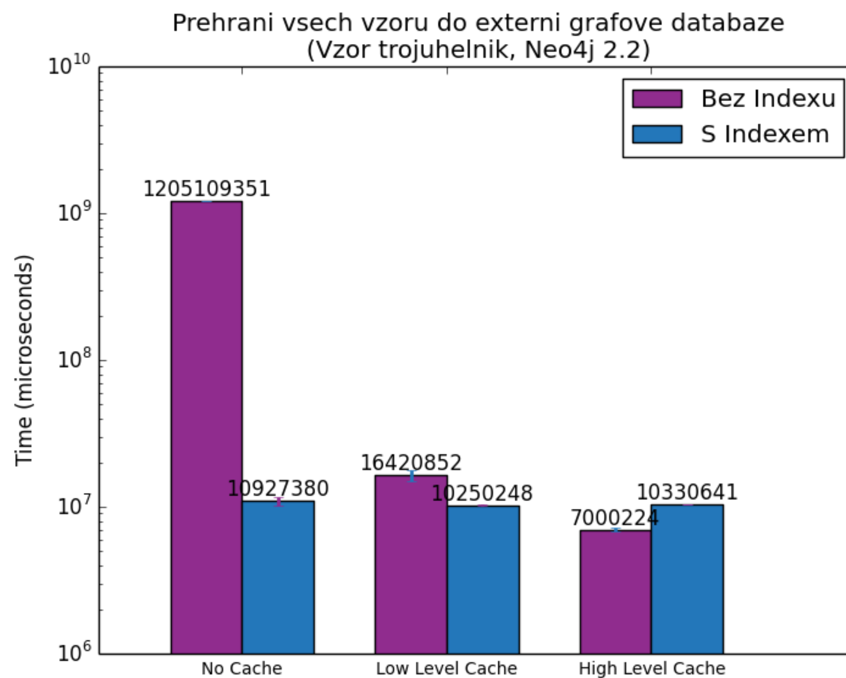
Využití externí databáze vychází z předchozího dotazu, kde jsme chtěli dotazovat na podgrafu. Právě proto se tuto funkcionalitu pokusíme nasimulovat. Každý záznam z indexu přehrajeme pomocí nativního Java API do externí databáze, a tím zkonstruujeme vzor, nad kterým se dotážeme. Následně ho smažeme a tyto dvě operace opakujeme pro všechny záznamy z indexu.



Obrázek 4.6: Výsledky přehrání jednoho vzoru do externí grafové databáze

4.4.5.2 Přehraní všech vzorů do externí grafové databáze

Poslední měření je rozdílné oproti předchozímu v tom, že nepřehráváme vzory postupně, ale přehrajeme všechny najednou. Nově vzniklá externí databáze bude obsahovat pouze uzly a hrany, které náleží nějaké vzoru. Následně se nad nově vzniklou externí databází dotážeme, získáme opět výsledek ekvivalentní obecnému dotazu a nakonec databázi vyprázdníme.



Obrázek 4.7: Výsledky přehraní všech vzorů do externí grafové databáze

4.4.6 Shrnutí

Měření nám ukázalo několik zajímavých poznatků. Přehrávání vzorů do externí databáze trvá přibližně stejně dlouho pro všechny nastavení cache. Veškeré údaje, které přehráváme, jsou unikátní (index neobsahuje automorfní grafy), a proto žádný z typů cache nepomůže. U vyhledávání na podgrafu jsme se přesvědčili, že obecně zadané dotazy Neo4j trvají velmi dlouho. A proto ani vyhledávání na podgrafu nebylo nejlepším způsobem pro dohledávání vzorů.

Nejlepší výsledky jsme shledali u měření se zakotvením všech uzlů, ale zároveň jsme zmínili problém faktoriálního růstu. Právě proto provedeme poslední měření na větším vzoru a porovnáme výsledek s druhým nejlepším – vylepšené vyhledávání se zakotvením jednoho uzlu.

```
1 MATCH
    (a)-[h]-(b)-[i]-(c)-[j]-(a)-[k]-(d)-[l]-(e)-[m]-(a)-[n]-(f)-[o]-(g)-[p]-(a)
    RETURN a,b,c,d,e,f,g
```

Výsledek z měření pro zakotvení všech uzlů jsme nezískali ani po 30 minutách. Vyhledání jednoho vzoru obsahovalo 5040 poddotazů ($|V_P|!$), které jsme spojili klauzulí *UNION*. Oproti tomu měření vylepšeného vyhledávání se zakotvením jednoho uzlů bylo provedlo vždy okolo 1 sekundy.

Z posledního měření vychází nejlépe vylepšené vyhledávání se zakotvením jednoho uzlu, které implementujeme do grafového indexu pro dohledávání grafových vzorů.

Kapitola 5

Implementace

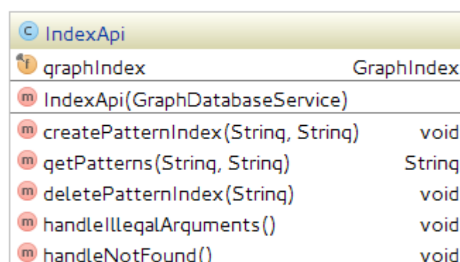
Na základě návrhu a testování přejdeme k implementaci rozšíření grafového indexu. Představíme si technologie a techniky, které byly během implementace využity, a popíšeme si jednotlivé části rozšíření.

5.1 Nástroje a techniky

Java 8 a Neo4j ve verzi 2.2 byly základními prvky, které jsme k vývoji využili. Stejně tak tomu bylo u GraphAware framework, o kterém jsme se už zmínili v kapitole o návrhu. Maven 3.2.5 jsme použili k získání závislostí, které tak byly do projektu automaticky navázány. JUnit testy jsme otestovali funkčnost jednotlivých částí systému a nyní si ji rozebereme.

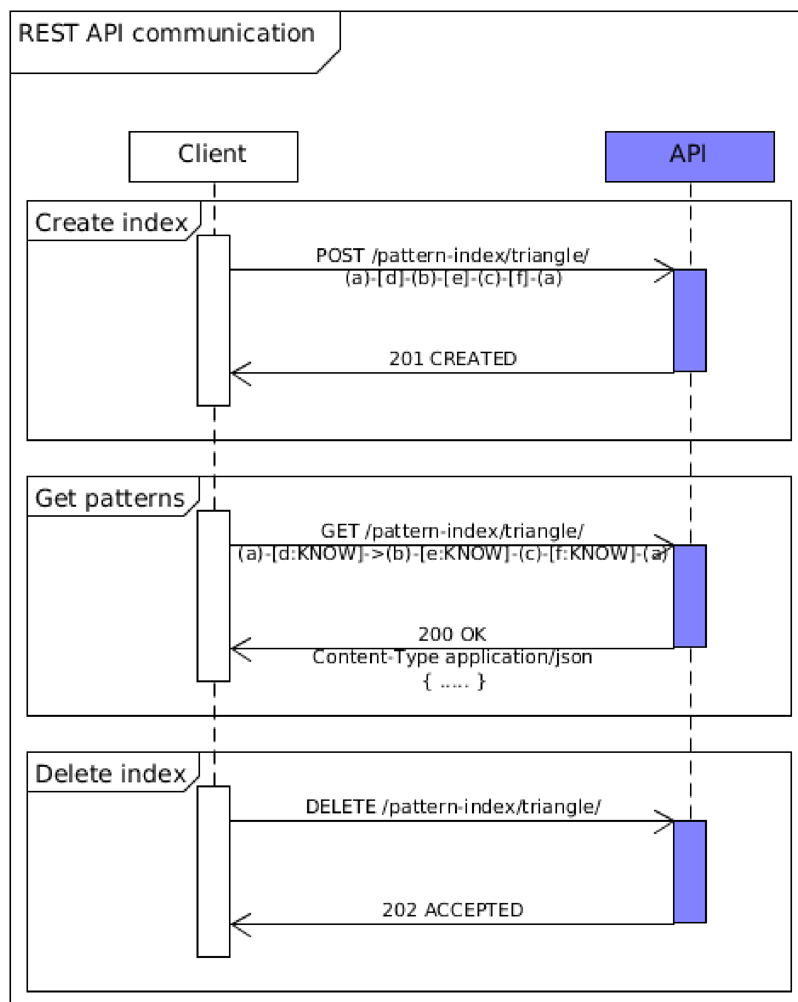
5.2 REST API

REST API je reprezentováno třídou *IndexAPI* 5.1, která poskytuje rozhraní ke komunikaci s indexem. Hlavními metodami jsou získání vzorů a vytvoření a smazání indexu. Metody jsou navěšeny na příslušné zdroje (*resources*), a pokud v nich nastane chyba, je vyvolaná výjimka *IllegalArgumentException*, která vrací odpověď s informací o chybném požadavku.



Obrázek 5.1: REST API reprezentováno třídou IndexAPI [46]

Komunikaci REST API si ukážeme na sekvenčním diagramu 5.2. První požadavek vytvoří index pro vzor trojúhelník. Následně se nad ním dotážeme konkrétnějším vzorem a získáme JSON odpověď, která obsahuje data s požadovanými výsledky za *RETURN* klauzulí. Poslední požadavek smaže index, který jsme prvním požadavkem vytvořili.



Obrázek 5.2: Ukázka požadavku na vytvoření indexu, získání konkrétních vzorů a smazání indexu [57]

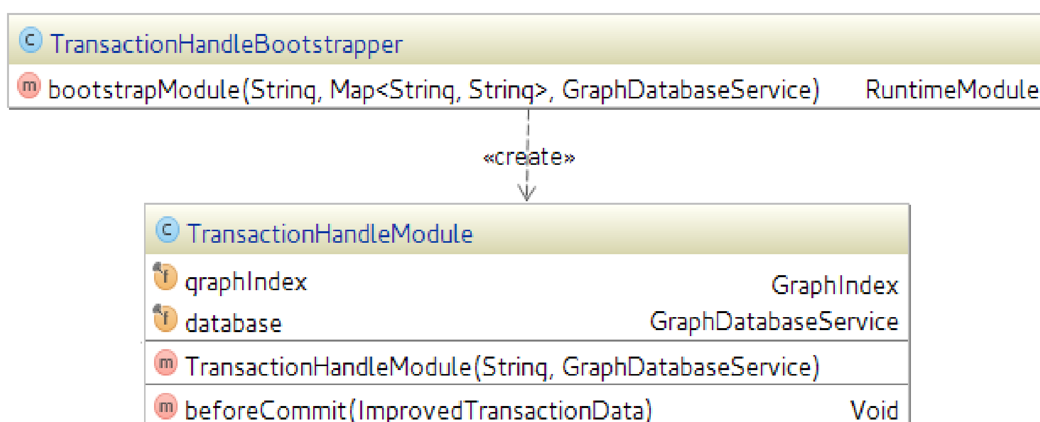
5.3 Operace INSERT/DELETE

Vkládání a mazání nových hran nám může způsobit vznik, či zánik vzoru. Před každým *commitem* odchytíme transakci, z které získáme všechny nově vzniklé hrany. U každé z nich si zjistíme startovní uzel, z kterého se následně dohledávají všechny zaindexované vzory. Pokud vznikne přidáním nové hrany nový vzor, přidá se do příslušné Hash tabulky zaindexovaného vzoru.

V opačném případě, při odebráním hrany, se sekvenčně projdou všechny Hash tabulky zaindexovaných vzorů a hledá se, jestli počáteční uzel je obsažen v některém z klíčů (reprezentace vzoru).

GraphAware Runtime modul sváže index s Neo4j databází, která mu předává transakce, z kterých jsou získány výše popsané hrany 5.3. Pro zaregistrování modulu je nutné o něm přidat informace do konfiguračního souboru *neo4j.properties*:

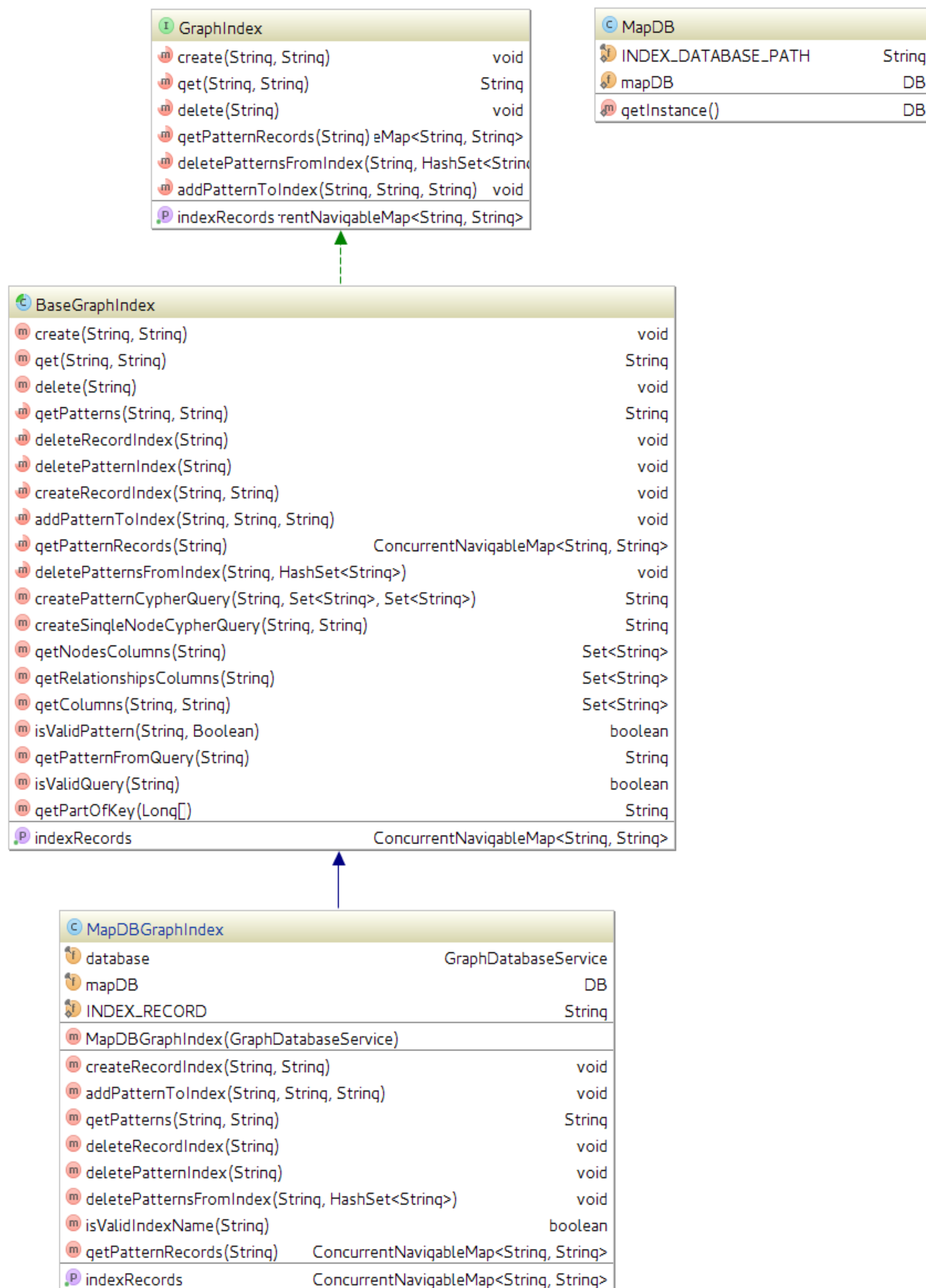
```
1 com.graphaware.module.PATTERNINDEX.2=
2 com.rambajar.graphaware.transactionHandle.TransactionHandleBootstrapper
```



Obrázek 5.3: Runtime modul [57]

5.4 Index

Hlavní částí je *MapDBGraphIndex*, který se stará o veškerou funkcionalitu indexování vzorů. Z REST API přijímá parametry, kontroluje jejich validitu a následně je zpracovává. *MapDB* je singleton vracející instanci *MapDB*, se kterou se pracuje v metodách *MapDBGraphIndexu*. *BaseGraphIndex* je abstraktní třída, která mu definuje rozhraní a obsahuje obecné metody, jež nejsou závislé na vybraném úložišti indexu (MapDB).



Obrázek 5.4: Grafový index [57]

Kapitola 6

Testování výkonu indexu

Abychom mohli v další části porovnat výsledky se souběžnou diplomovou prací, provedeme výkonnostní testování vlastní implementace. Na databázích Erdős–Rényi, Cineasts a transakcí změříme dobu indexace, velikost indexu a čas vyhodnocení dotazů. Zároveň provedeme benchmark DML (Data Manipulation Language) operací indexu oproti DML operacím bez indexu. Výsledky měření na konci vyhodnotíme, sepíšeme veškeré postřehy a poznatky, které jsme během měření získali.

6.1 Databáze Erdős–Rényi

Komunitní síť Erdős–Rényi ve velikosti 10 000 uzlů a 50 000 hran jsme použili k měření času dotazů. Pro další testování jsme vytvořili menší databázi, která obsahuje 1 000 uzlů a 5 000 hran a jednu větší obsahující 100 000 uzlů a 500 000 hran. Stejně jako v předchozím měření jsme vyhledávali trojúhelníky a výsledky porovnávali s dotazováním bez indexu.

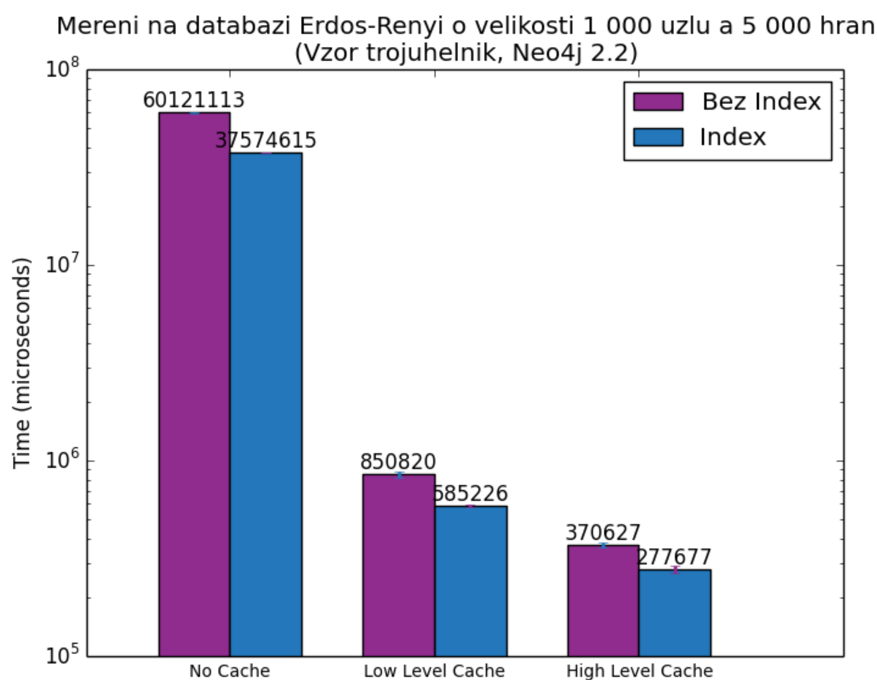
U každého měření uvedeme velikost indexu, který je oddělen od databáze, délku jeho vytvoření a velikost redukce počtu dotazů, které budou na databázích provedeny. Všechna měření zaneseme do tabulky, kromě výsledků měření času dotazů, které opět znázorníme do grafu pro všechny druhy cache.

Měřený dotaz:

```
1 MATCH (a)--(b)--(c)--(a) RETURN a,b,c
```

6.1.1 1 000 uzlů a 5 000 hran

U nejmenší databáze jsme shledali nepatrné zrychlení pro všechny druhy cache. Počet dotazů jsme zredukovali celkem o 28 a velikost indexu na pevném disku zabrala 73 kB.



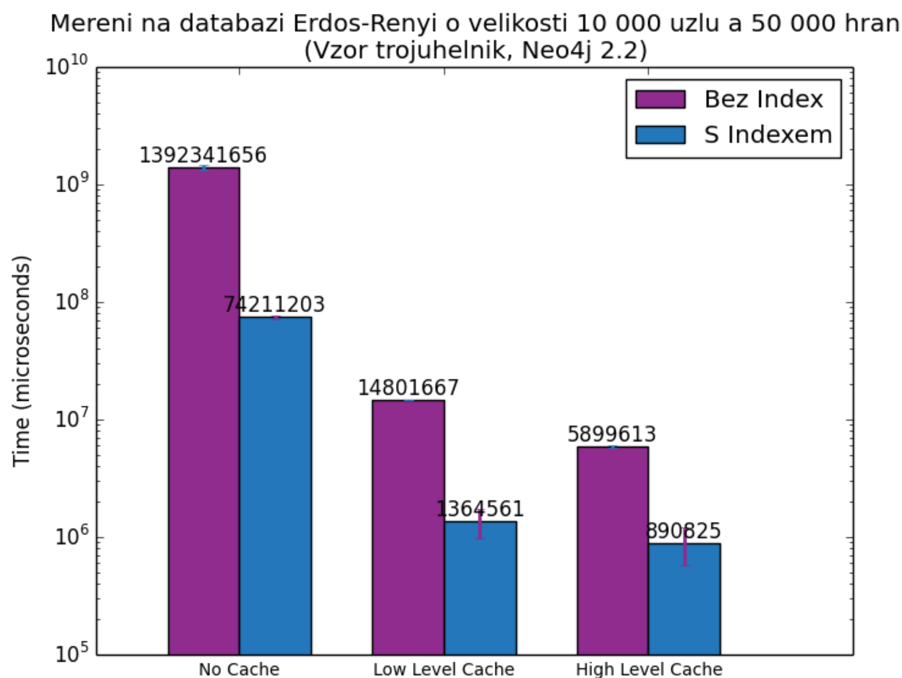
Obrázek 6.1: Výsledky měření na databázi Erdős–Rényi o velikosti 1 000 uzlů a 5 000 hran

Tabulka 6.1: Souhrnné výsledky měření na databázi Erdős–Rényi o velikosti 1 000 uzlů a 5 000 hran

Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
6,71 s	7,43 s	78,18 s	168	140	73,0 kB

6.1.2 10 000 uzlů a 50 000 hran

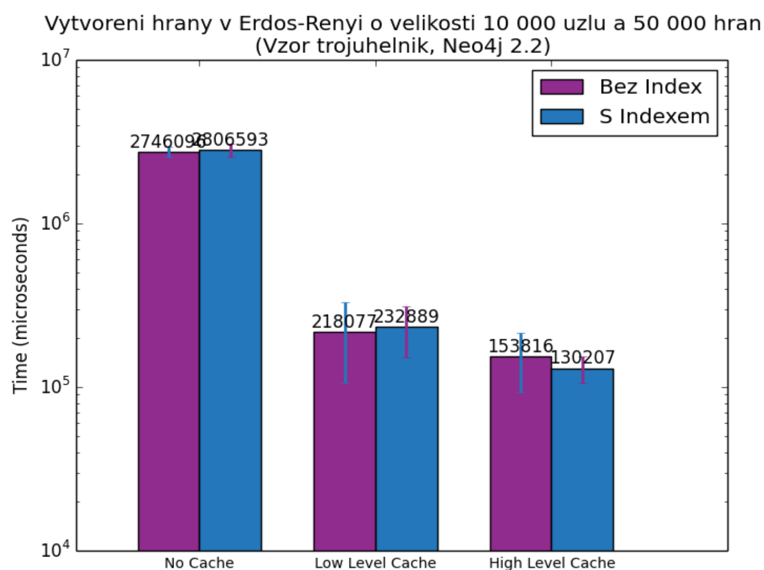
U databáze se střední velikostí jsme shledali značné zrychlení pro všechny druhy cache. U každé jsme dosáhli řádového zrychlení. Počet dotazů jsme zredukovali o čtyři a index na pevném disku zabral 77,1 kB. Zároveň jsme měřili operace vytvoření hrany, s kterou vznikne nový vzor. Smazání hrany, s kterou zanikne vzor a smazání uzlu, který náleží do zaindexovaného vzoru. U žádné z těchto operací, jsme neshledali zpomalení oproti výchozím operacím Neo4j.



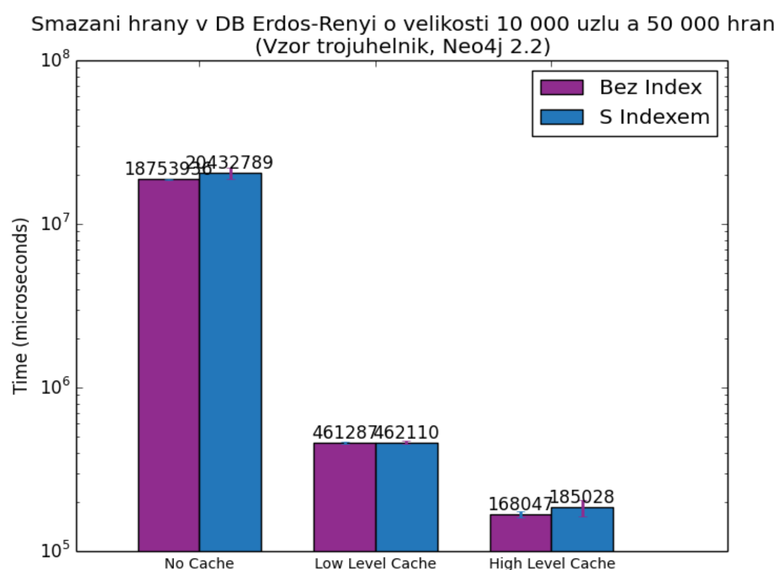
Obrázek 6.2: Výsledky měření na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran

Tabulka 6.2: Souhrnné výsledky měření na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran

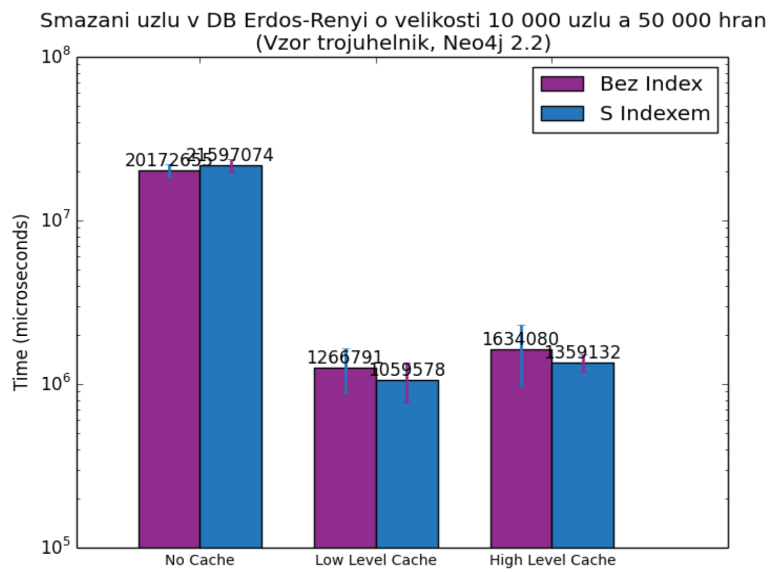
Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
20,93 s	21,11 s	1 338,94 s	183	179	77,1 kB



Obrázek 6.3: Výsledky pro přidání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran



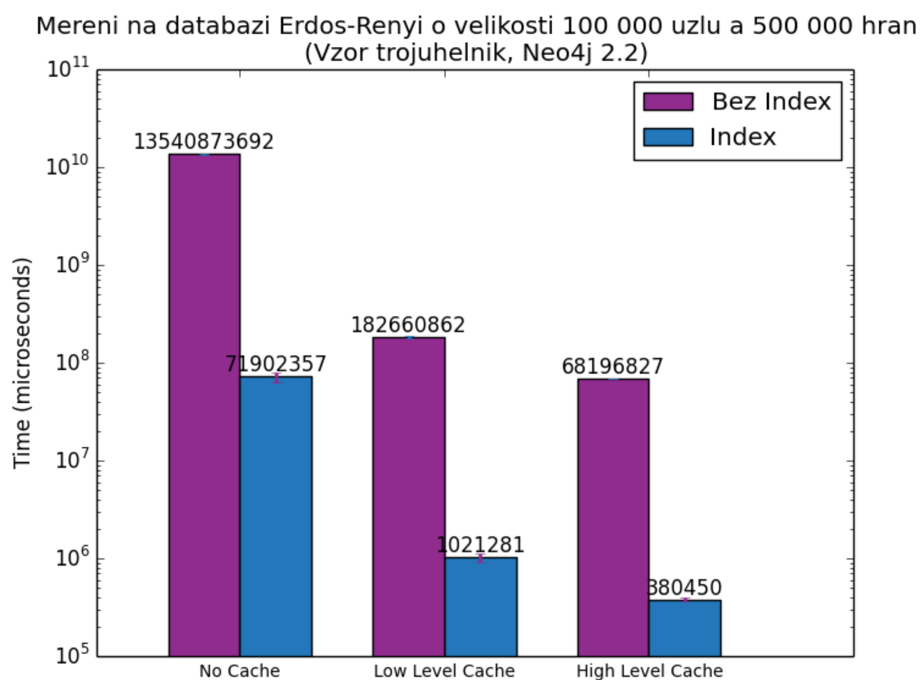
Obrázek 6.4: Výsledky pro smazání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran



Obrázek 6.5: Výsledky pro smazání uzlu na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran

6.1.3 100 000 uzlů a 500 000 hran

Na největší databázi jsme shledali nejlepší výsledky u všech druhů cache. Na každé jsme dosáhli zrychlení o minimálně 2 řády. Počet dotazů jsme zredukovali o pouhý jeden a velikost indexu na pevném disku byla 77,7 kB.



Obrázek 6.6: Výsledky měření na databázi Erdős–Rényi o velikosti 100 000 uzlů a 500 000 hran

Tabulka 6.3: Souhrnné výsledky měření na databázi Erdős–Rényi o velikosti 100 000 uzlů a 500 000 hran

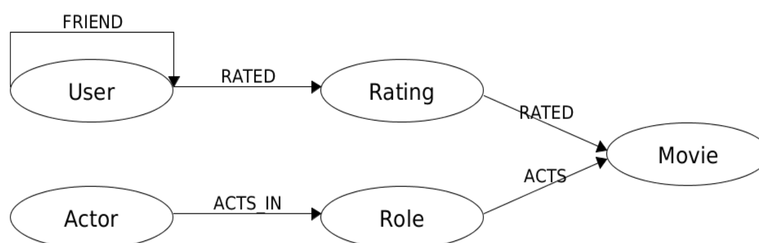
Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
78,03 s	166,92 s	13 807,62 s	157	156	77,7 kB

6.2 Filmová databáze Cineasts

Cineasts je filmová databáze se sociálním aspektem. Velikosti databáze je 72 000 uzlů a 106 651 hran. Obsahuje fanoušky (uživatele), kteří hodnotí filmy, udělují hodnocení a sdílí je se svými přáteli. Pochopitelně zahrnuje i filmy a herce, kteří byli obsazeni do nějakého filmu. Schéma databáze je znázorněno na obrázku 6.7.

Měřený dotaz:

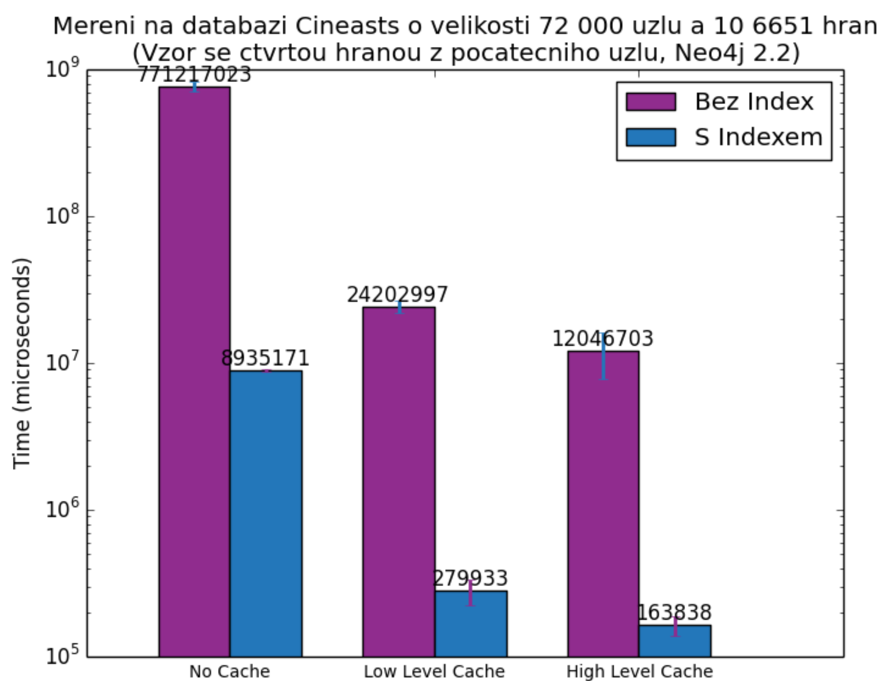
```
1 MATCH (a)--(b)--(c)--(a)--(a) RETURN a,b,c,d
```



Obrázek 6.7: Základní schéma filmové databáze Cineasts [46]

U databáze Cineasts jsme dosáhli zrychlení pro všechny druhy cache. Hlavním důvodem byla velká redukce vzorů z 86 na pouhých 6. Velikost indexu na pevném disku byla 73 kB.

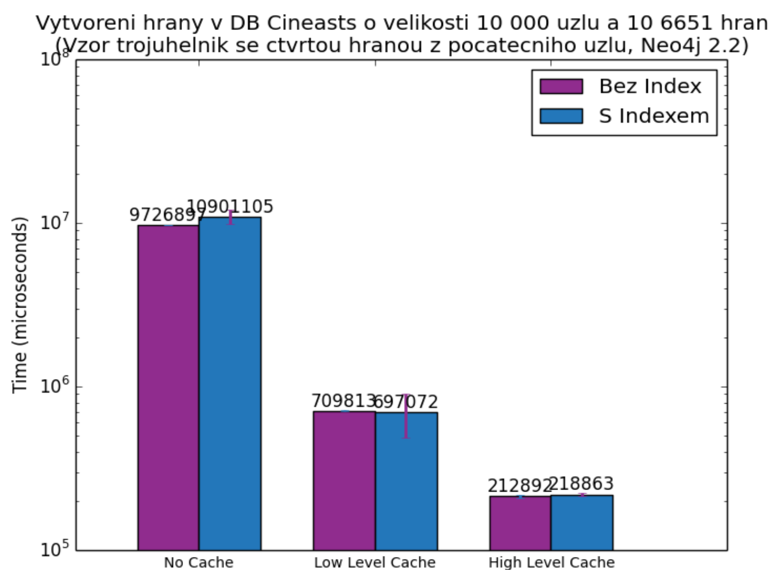
Ani u tohoto měření jsme neshledali zpomalení operací přidání hrany, smazání hrany a smazání uzlu oproti výchozím operacím Neo4j.



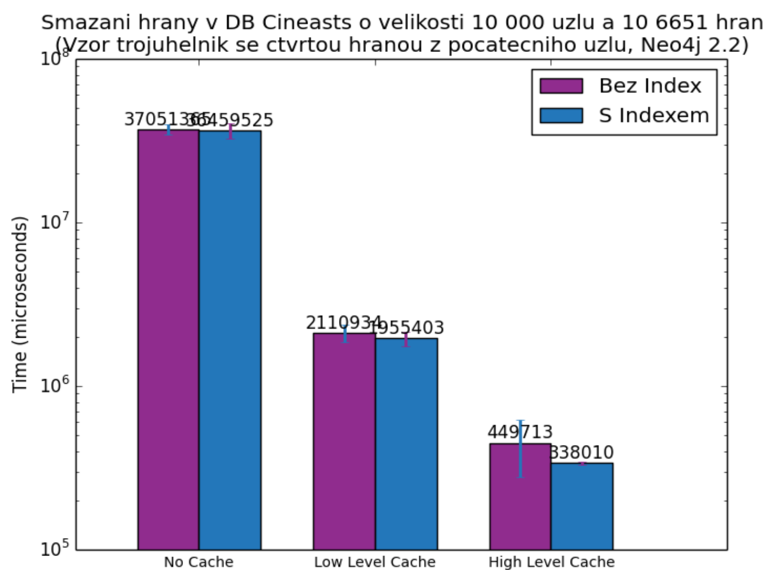
Obrázek 6.8: Výsledky měření na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran

Tabulka 6.4: Souhrnné výsledky měření na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran

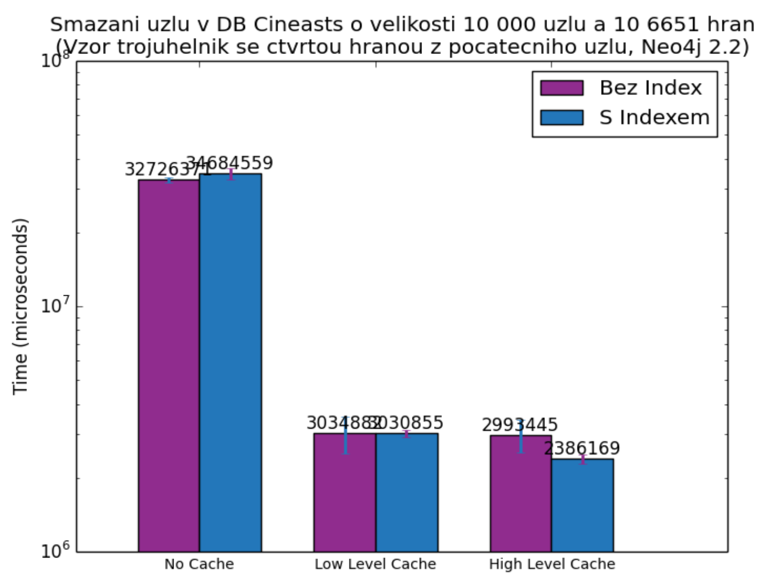
Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
17,36 s	26,86 s	834,38 s	86	6	48,3 kB



Obrázek 6.9: Výsledky pro přidání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran



Obrázek 6.10: Výsledky pro smazání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran



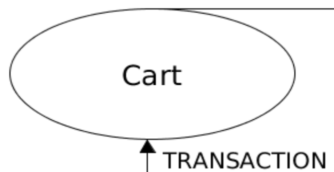
Obrázek 6.11: Výsledky pro smazání uzlu na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran

6.3 Databáze transakcí

Databáze transakcí je zcela náhodná síť. Velikosti databáze je 10 000 uzlů a 99 970 hran. Obsahuje uzly reprezentující kreditní karty a hrany transakce mezi nimi. Každá hrana je náhodně vytvořená s předem určenou pravděpodobností. Schéma databáze je na obrázku 6.12.

Měřený dotaz:

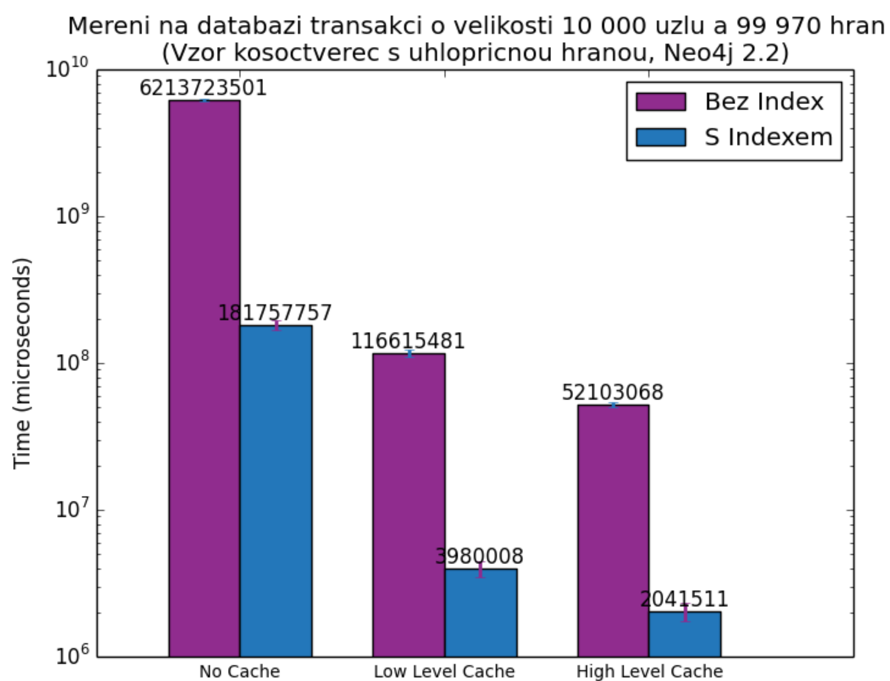
```
1 MATCH (a)-[e]-(b)-[f]-(c)-[g]-(a)-[h]-(d)-[i]-(b) RETURN a,b,c,d
```



Obrázek 6.12: Schéma databáze transakcí [46]

U databáze transakcí jsme docílili zrychlení pro všechny druhy cache. Na každé jsme dosáhli zrychlení přibližně o 2 řády. Počet dotazů jsme zredukovali o čtyři a velikost indexu na pevném disku byla 74,2 kB.

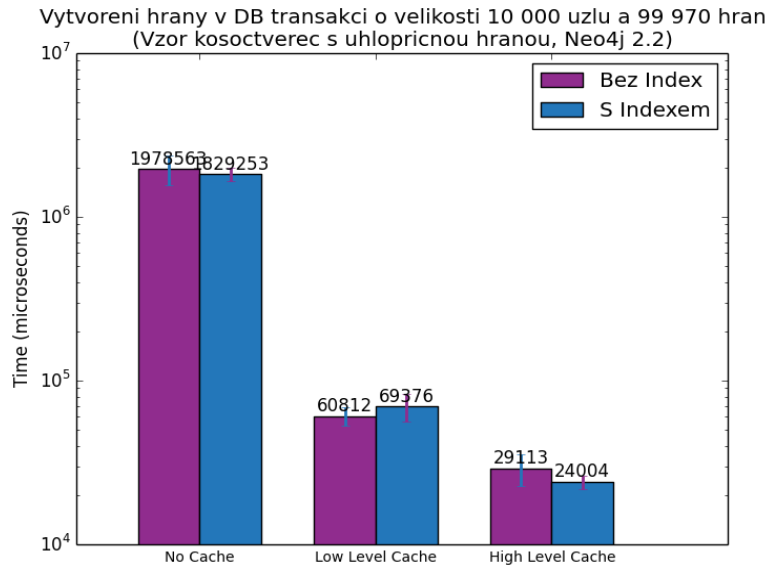
Pouze na měření smazání hrany jsme shledali zpomalení u *LowLevelCache*, ostatní operace byly srovnatelné s výchozími dotazy Neo4j.



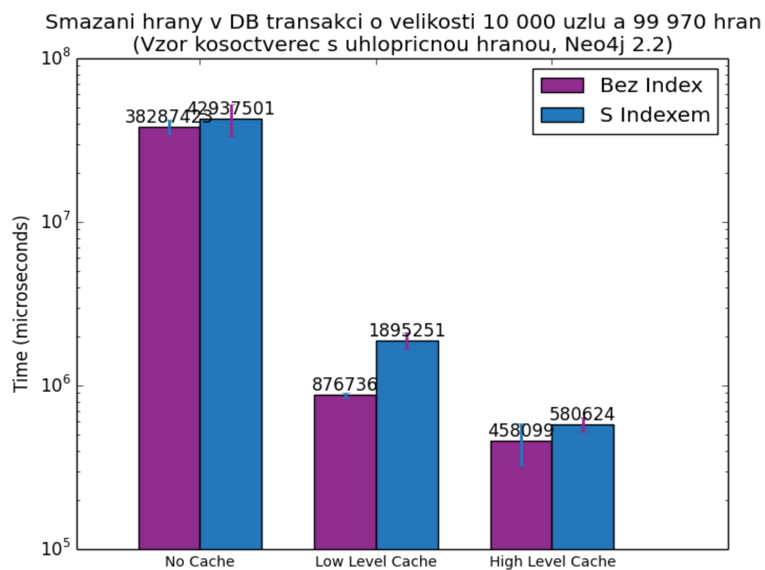
Obrázek 6.13: Výsledky měření na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran

Tabulka 6.5: Souhrnné výsledky měření na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran

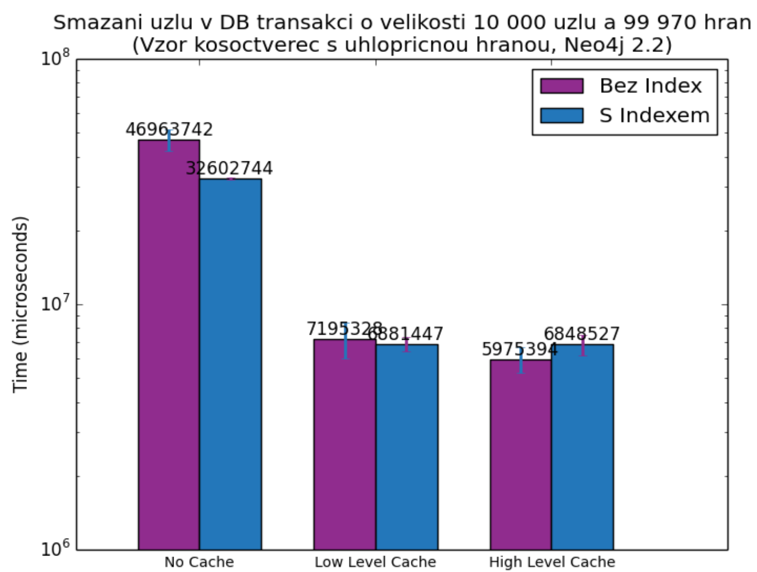
Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
73,96 s	129,55 s	6219,054 s	70	66	74,2 kB



Obrázek 6.14: Výsledky pro přidání hrany na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran



Obrázek 6.15: Výsledky pro smazání hrany na databázi transakcí o velikosti 72 000 uzlů a 106 651 hran



Obrázek 6.16: Výsledky pro smazání uzlu na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran

Kapitola **7**

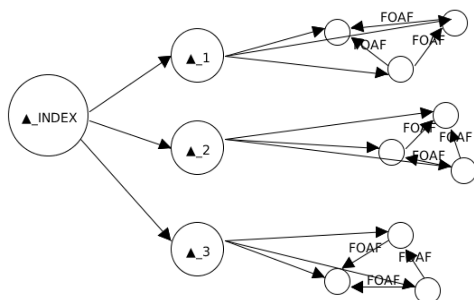
Porovnání výsledných řešení

Souběžně s touto prací byla vypracována diplomová práce, která měla za úkol vytvořit index, který bude udržován v Neo4j databázi. Na začátku této kapitoly si popíšeme algoritmus a schéma indexu, které bylo navrženo kolegou Bc. Martinem Troupem. Následovat budou jednotlivá měření na databázích, které jsme už použili během testování výkonu indexu. Taktéž budeme opět měřit získání všech vzorů, smazání hrany, přidání hrany a smazání uzlu.

U každého měření porovnáme a vyhodnotíme výsledky, zaneseme je do grafů a souhrnných tabulek s dodatečnými informacemi o měření.

7.0.1 Popis algoritmu

Algoritmus je postaven na stromové struktuře. Veškeré informace o indexu jsou uloženy přímo v Neo4j databázi. Každý index má svůj kořen, který má vazby na jednotky indexovaných vzorů. Obsahuje jméno a popis vzoru. Z jednotek vedou hrany na konkrétní datové uzly a zároveň mají uloženy seznam všech vzorů sdílejících dané datové uzly jako množinu ID hran.

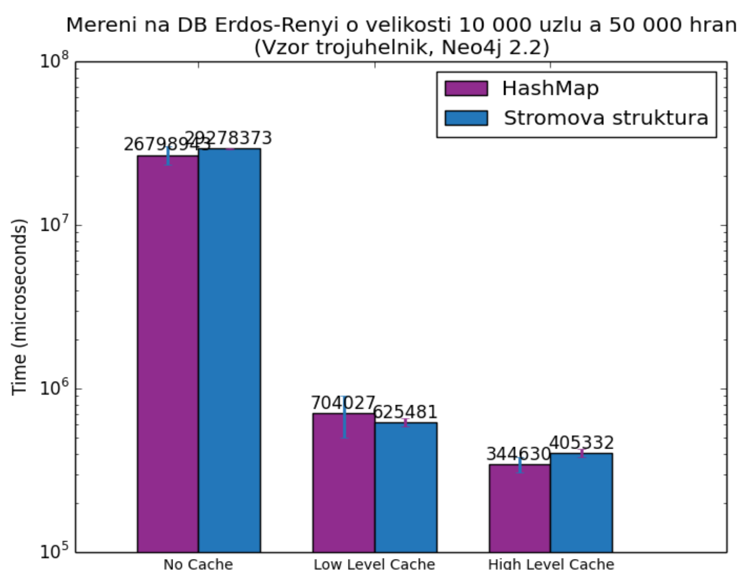


Obrázek 7.1: Schéma stromového indexu [46]

7.1 Databáze Erdős–Rényi

U databáze Erdős–Rényi jsme dosáhli u měření dotazování nad indexem o trochu lepších výsledků u indexu uloženého v Hash tabulce. A to i přesto, že počet vzorů po redukci byl větší než počet vzorů po redukci u indexu uloženého přímo v databázi. Zkonstruování indexu vyšlo kromě *NoCache* velmi podobně.

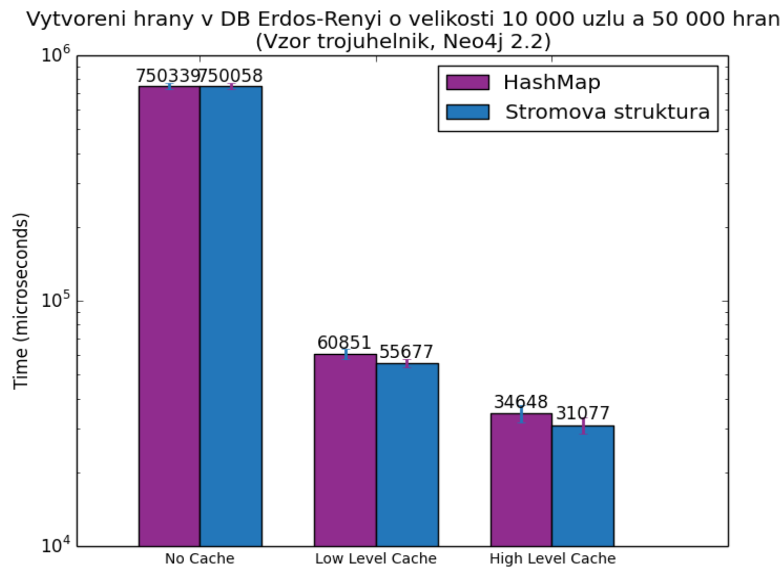
Smazání hrany a vytvoření hrany vyšlo taktéž velmi podobně. Ovšem u smazání uzlu se projevilo značné zrychlení indexu uloženého přímo v databázi, kde se ukázala výhoda rychlého přístupu k *META* hranám zaindexovaného vzoru.



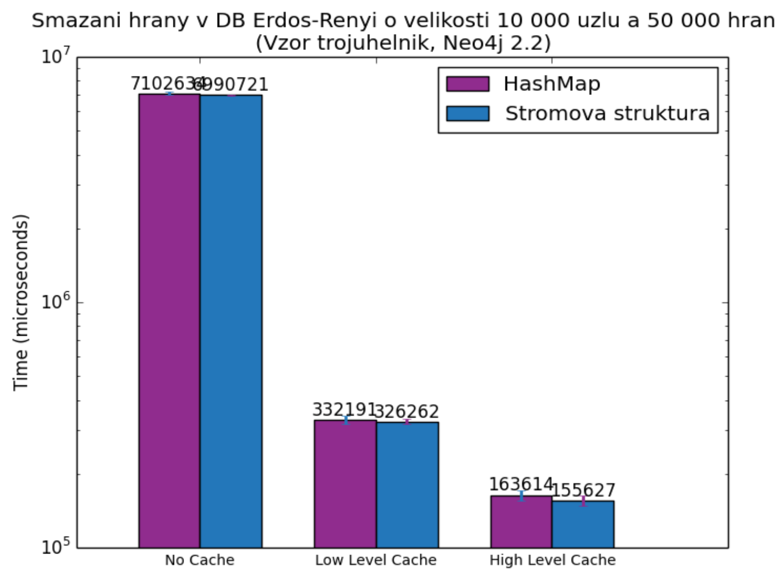
Obrázek 7.2: Porovnání výsledků měření na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran

Tabulka 7.1: Souhrnné výsledky porovnání na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran

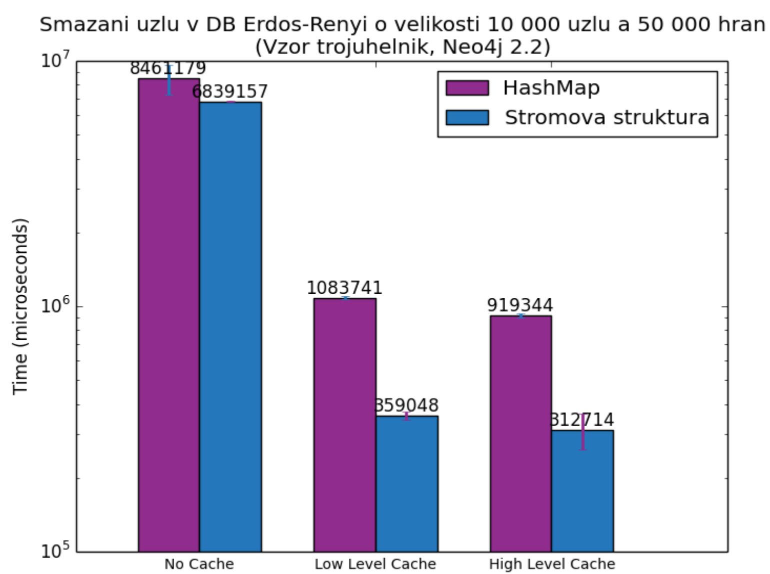
Typ indexu	Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
HashMap	4,98s	8,49 s	334,56 s	183	179	70 kB
Stromová struktura	5,24 s	8,76 s	425,8 s	183	166	3 MB



Obrázek 7.3: Porovnání výsledků přidání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran



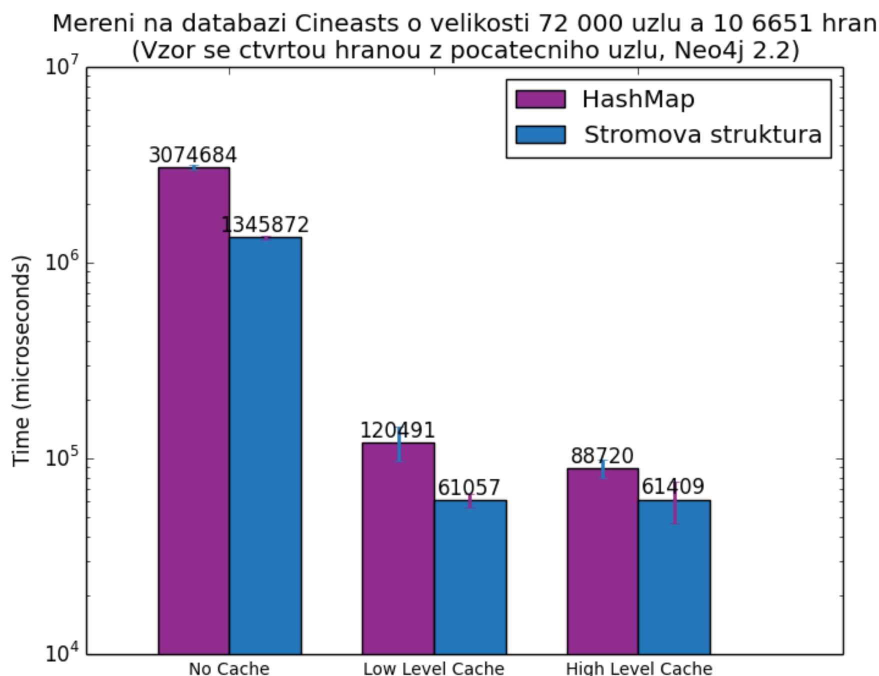
Obrázek 7.4: Porovnání výsledků smazání hrany na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran



Obrázek 7.5: Porovnání výsledků smazání uzlu na databázi Erdős–Rényi o velikosti 10 000 uzlů a 50 000 hran

7.2 Databáze Cineasts

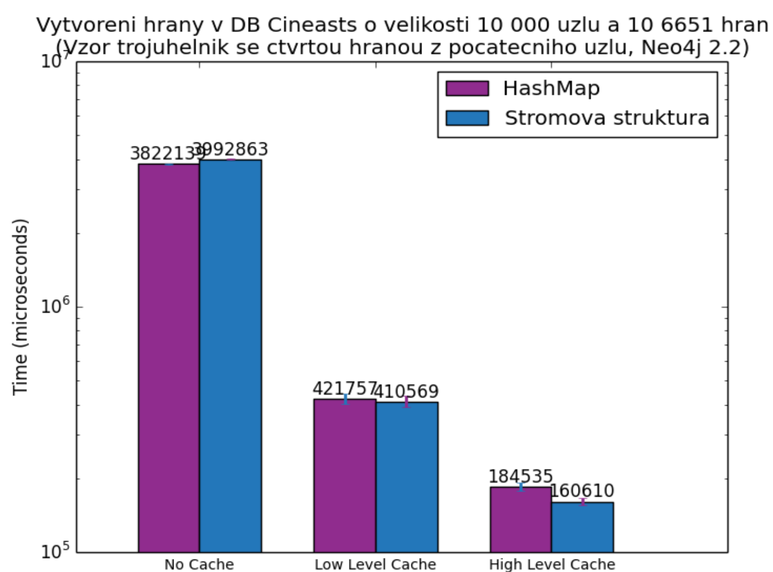
U databáze Cineasts jsme shledali, že redukce pro strukturu vyšla poloviční oproti Hash tabulce. Od toho se odvíjí výsledky dotazování, u kterých index postavený nad Hash tabulkou vyšel hůře než index uložený přímo v databázi. Taktéž jako u předchozího měření se projevil rozdíl u operace smazání uzlu.



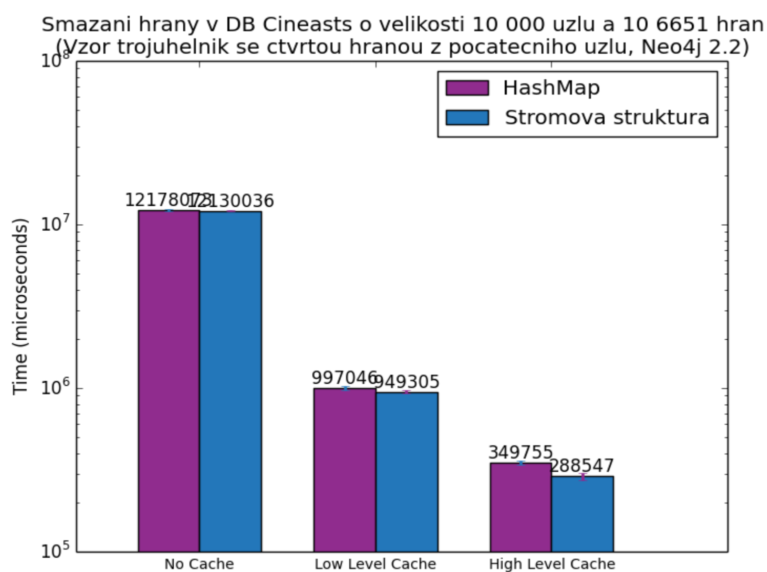
Obrázek 7.6: Porovnání výsledků měření na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran

Tabulka 7.2: Souhrnné výsledky porovnání na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran

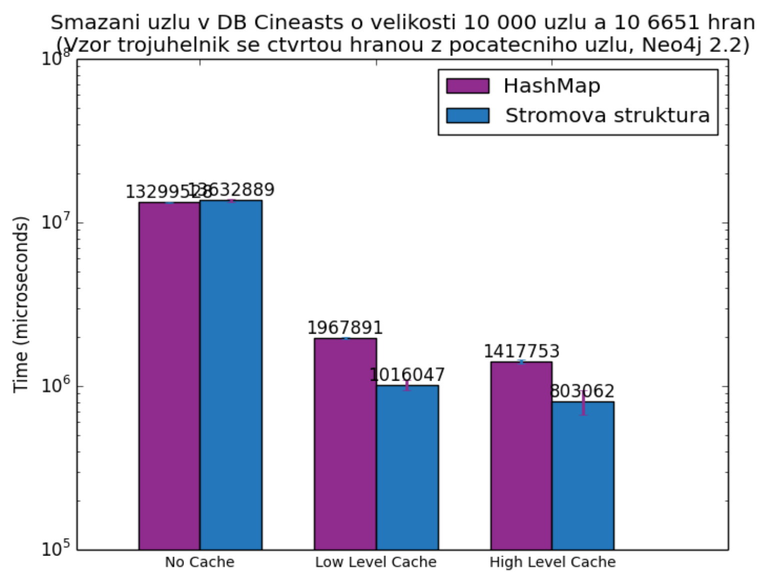
Typ indexu	Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
HashMap	11,64 s	15,18 s	387,93 s	86	6	98 kB
Stromová struktura	11,81 s	12,7 s	325,13 s	86	3	100 kB



Obrázek 7.7: Porovnání výsledků přidání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran



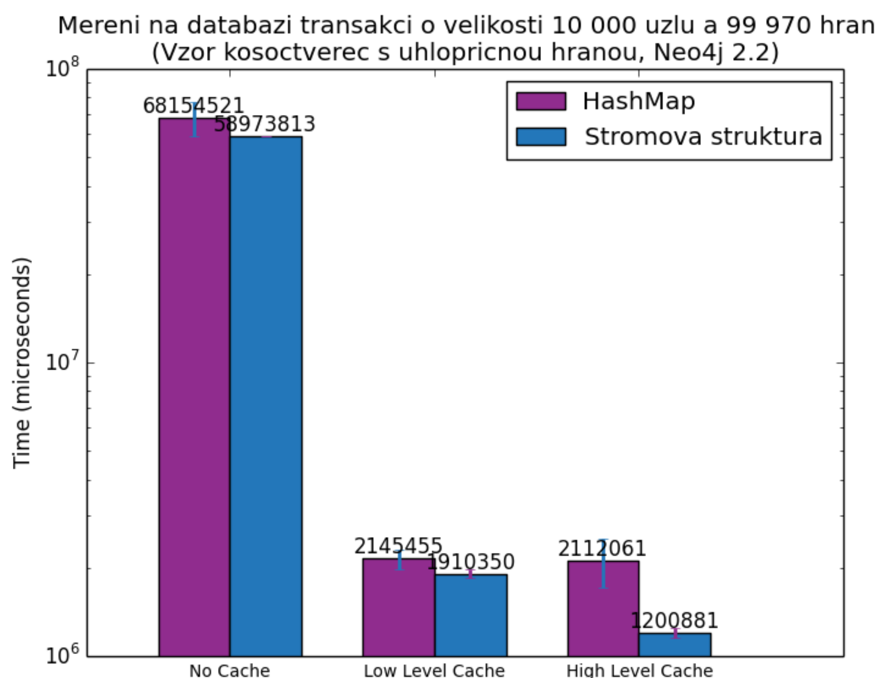
Obrázek 7.8: Porovnání výsledků smazání hrany na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran



Obrázek 7.9: Porovnání výsledků smazání uzlu na databázi Cineasts o velikosti 72 000 uzlů a 106 651 hran

7.3 Databáze transakcí

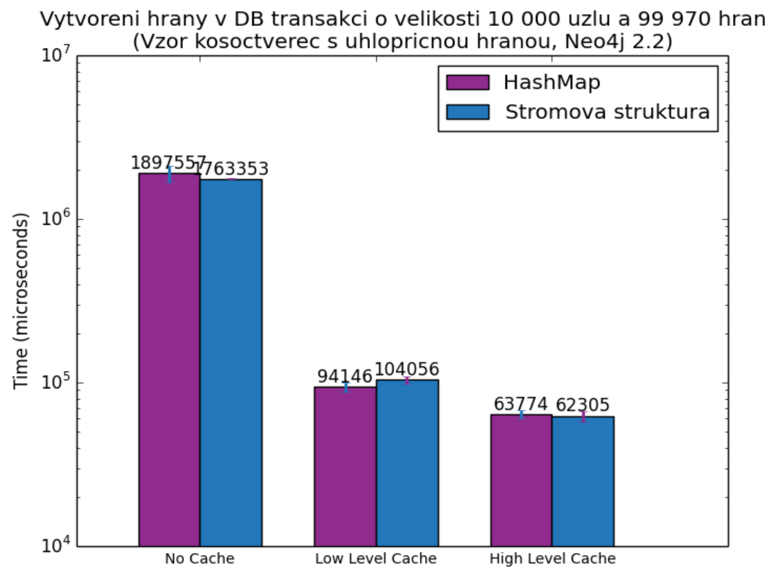
U databáze transakcí se projevil největší rozdíl na *HighLevelCache*, pří měření získání všech zaindexovaných vzorů. Ostatní měření nám neukázaly další rozdíly, až na operaci smazání uzlu, kterou index uložený přímo v databázi provedl i u ostatních testů mnohem lépe, než index uložený v Hash tabulce.



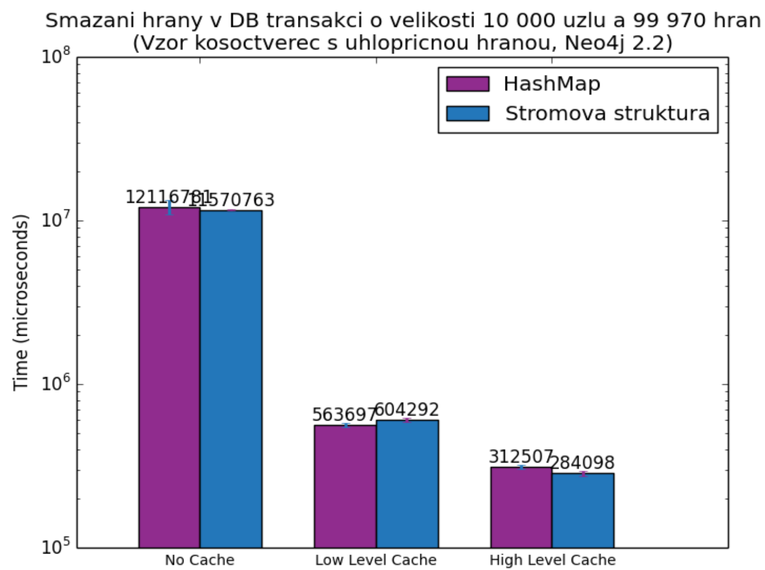
Obrázek 7.10: Porovnání výsledků měření na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran

Tabulka 7.3: Souhrnné výsledky porovnání na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran

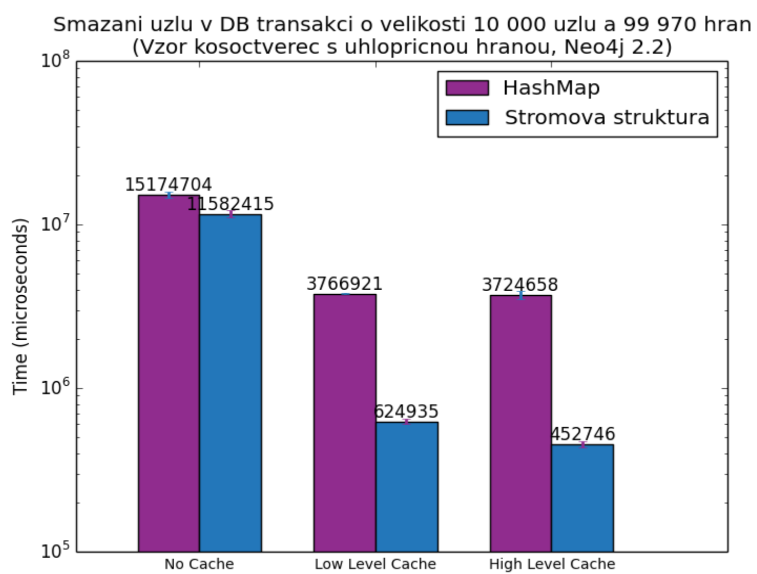
Typ indexu	Zkonstruování indexu s HighLevelCache	Zkonstruování indexu s LowLevelCache	Zkonstruování indexu s NoCache	Celkem vzorů	Celkem vzorů po redukci	Velikost indexu
HashMap	47,62 s	71,88 s	více jak 1 h	70	66	74,2 kB
Stromová struktura	44,68 s	68,25 s	více jak 1 h	70	64	100 kB



Obrázek 7.11: Porovnání výsledků přidání hrany na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran



Obrázek 7.12: Porovnání výsledků smazání hrany na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran



Obrázek 7.13: Porovnání výsledků smazání uzlu na databázi transakcí o velikosti 10 000 uzlů a 99 970 hran

Závěr

Výsledek práce můžeme považovat za zdařilý, navrhli jsme prototyp pro indexování vzorů v Neo4j databázi, provedli jsme otestování Neo4j dotazů k dohledávání zaindexovaných vzorů a následně implementovali navržený prototyp.

Na námi vybraných databázích jsme ukázali příklady dotazování, ke kterým by byl navržený index v reálném použití vhodný. Výsledky nám potvrdily, že pro případy, které jsme navrhli, by index velmi urychlil dotazování a nepříliš ovlivnil operace vkládání/mazání uzlů a hran do grafové databáze.

Také jsme navržený prototyp indexu porovnali s prototypem souběžné diplomové práce. Zjistili jsme, že index uložený přímo v databázi méně ovlivní operaci smazání uzlu, než námi navržený index. V dohledávání vzorů pomocí indexu se prototypy časově výrazně nelišily. Pokud ale došlo k nějakým odchylkám u dotazování nad indexem, rozhodoval vždy počet vzorů po redukci, který index dohledával. Efektivnost redukce prototypu pokaždé záležela na konkrétním obsahu databáze, a proto nemůžeme exaktně říci, která je lepší či horší.

V požadavcích, které jsme si vydefinovali na začátku práce, jsme uvedli, že v prototypu akceptujeme absenci dvoufázového potvrzovací protokolu, který by byl pro produkční nasazení nutnou podmínkou.

Na prototypu mám motivaci nadále pracovat a pokusím se prosadit funkcionalitu indexování vzorů do budoucí verze Neo4j. Závěrem věřím, že práce bude přínosem pro ty, kteří se setkají s problematikou indexování vzorů v souvislosti s grafovými databázemi. Také i pro ty, kteří se zajímají o grafovou teorii a chtějí si rozšířit obzory v novém trendu databázových úložišť.

Literatura

[1] IBM ČESKÁ REPUBLIKA, SPOL. S R.O.: *What is big data? [online].* [cit. 2015-04-29]. Dostupné z: http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

[2] EASLEY, D.; KLEINBERG, J.: Networks, Crowds, and Markets: Reasoning about a Highly Connected World. Draft version, [cit. 2015-15-04]. Dostupné z: https://www.cs.cornell.edu/home/kleinber/networks-book/networks-book.pdf

[3] SERROURA, B.; ARENASB, A.; GÓMEZ, S.: Detecting communities of triangles in complex networks using spectral optimization. [cit. 2015-22-04]. Dostupné z: http://arxiv.org/pdf/1005.3658.pdf

[4] ADAMCHIK, V.: Graph Theory. University Lecture, 2005, [cit. 2015-05-04]. Dostupné z: http://www.cs.cmu.edu/~adamchik/21-127/lectures/graphs_1_print.pdf

[5] HOLÝ, J.: Grafové databáze a neo4j [video]. Prosinec 2012, [cit. 2015-04-05]. Dostupné z: https://www.youtube.com/watch?v=lLuqIcYvWUQ

[6] HORDĚJČUK, V.: *Teorie grafů [online].* [cit. 2015-04-05]. Dostupné z: http://voho.cz/wiki/matematika/graf/

[7] JIROVSKÝ, L.: *Matematická definice grafu [online].* [cit. 2015-03-26]. Dostupné z: http://teorie-grafu.cz/zakladni-pojmy/matematicka-definice-grafu.php

[8] JIROVSKÝ, L.: *Orientované grafy [online].* [cit. 2015-03-11]. Dostupné z: http://teorie-grafu.cz/zakladni-pojmy/orientovane-grafy.php

[9] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *Graph database [online].* Duben 2015, [cit. 2015-03-13]. Dostupné z: http://en.wikipedia.org/wiki/Graph_database

[10] NEO TECHNOLOGY, INC.: *What is a Graph Database? [online].* 2015, [cit. 2015-03-18]. Dostupné z: http://neo4j.com/developer/graph-database/

[11] JIROVSKÝ, L.: *Stromy [online].* [cit. 2015-03-17]. Dostupné z: http://teorie-grafu.cz/zakladni-pojmy/stromy.php

[12] JIROVSKÝ, L.: *Cesta a souvislost v grafu [online].* [cit. 2015-03-26]. Dostupné z: http://teorie-grafu.cz/zakladni-pojmy/cesta-a-souvislost-grafu.php

[13] WITTMAN, T.: Lecture 18: Tree Traversals. University Lecture, [cit. 2015-02-04]. Dostupné z: http://www.math.ucla.edu/~wittman/10b.1.10w/Lectures/Lec18.pdf

[14] MIČKA, P.: *Prohledávání do šířky [online].* [cit. 2015-04-15]. Dostupné z: http://www.algoritmy.net/article/1399/Prohledavani-do-sirky

[15] MIČKA, P.: *Prohledávání do hloubky [online].* [cit. 2015-04-15]. Dostupné z: http://www.algoritmy.net/article/1378/Prohledavani-do-hloubky

[16] KOVÁŘ, P.: Teorie grafů. Projekt Matematika pro inženýry 21. století, [cit. 2015-18-04]. Dostupné z: http://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu_kindle.pdf

[17] MATOUŠEK, J.; NEŠETŘIL, J.: *Kapitoly z diskrétní matematiky.* nakladatelství Karolinum, 2000, ISBN 80-246-0084-6.

[18] MALÝ, M.: *REST: architektura pro webové API [online].* Srpen 2009, [cit. 2015-03-18]. Dostupné z: http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[19] PAŽOUREK, T.: *Webové služby v architektuře REST na platformě .NET.* Bakalářská práce, Fakulta informatiky Masarykovy univerzity, 2012.

[20] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *Representational state transfer [online].* Březen 2015, [cit. 2015-03-18]. Dostupné z: http://en.wikipedia.org/wiki/Representational_state_transfer

[21] FIO BANKA: *Cache v prohlížečích [online].* 2008, [cit. 2015-04-10]. Dostupné z: http://www.fio.cz/navody/cache/

[22] MAŠEK, J.: *Aplikace pro zpracování a ukládání sociálních vztahů.* Diplomová práce, Fakulta informačních technologií ČVUT v Praze, 2012.

[23] FIELDING, R.: *REST APIs must be hypertext-driven [online].* Říjen 2008, [cit. 2015-04-22]. Dostupné z: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[24] KADLEC, J.: *REST a webové služby v jazyce JAVA*. Diplomová práce, Fakulta informatiky Masarykovy univerzity, 2010.

[25] KLINT, F.: *5 Graph Databases to Consider [online]*. Duben 2011, [cit. 2015-03-16]. Dostupné z: http://readwrite.com/2011/04/20/5-graph-databases-to-consider

[26] MUDRUŇKA, T.: *Netabulkové databáze NoSQL [online]*. Květen 2010, [cit. 2015-03-11]. Dostupné z: https://github.com/Harvie/Kyberia/blob/master/dokuwiki/pages/netabulkove_databaze_nosql.txt

[27] NEO TECHNOLOGY: Intro to Neo4j [video]. Leden 2013, [cit. 2015-03-11]. Dostupné z: http://www.youtube.com/watch?v=7Fsfa5DP9sE

[28] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *Social graph [online]*. Listopad 2014, [cit. 2015-03-25]. Dostupné z: http://en.wikipedia.org/wiki/Social_graph

[29] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *Sociogram [online]*. Září 2014, [cit. 2015-03-25]. Dostupné z: http://en.wikipedia.org/wiki/Sociogram

[30] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *Interest Graph [online]*. Leden 2015, [cit. 2015-03-20]. Dostupné z: http://en.wikipedia.org/wiki/Interest_Graph

[31] DATASTAX AURELIUS: *TITAN [online]*. Únor 2014, [cit. 2015-04-20]. Dostupné z: https://github.com/thinkaurelius/titan/wiki

[32] WOLPE, T.: *DataStax snaps up Aurelius and its Titan team to build new graph database [online]*. Únor 2015, [cit. 2015-03-11]. Dostupné z: http://www.zdnet.com/article/datastax-snaps-up-aurelius-and-its-titan-team-to-build-new-graph-database/

[33] MARR, M.: *Exploration of NoSQL: FlockDB [online]*. Březen 2011, [cit. 2015-03-25]. Dostupné z: http://www.devwebpro.com/exploration-of-nosql-flockdb/

[34] POPESCU, A.: *FlockDB and Graph Databases [online]*. Duben 2011, [cit. 2015-04-05]. Dostupné z: http://nosql.mypopescu.com/post/4423840717/flockdb-and-graph-databases

[35] AVITAL, B.: *FlockDB [online]*. Září 2011, [cit. 2015-04-16]. Dostupné z: https://github.com/twitter/flockdb

[36] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *Sones GraphDB [online]*. Leden 2015, [cit. 2015-04-12]. Dostupné z: http://en.wikipedia.org/wiki/Sones_GraphDB

[37] FRANZ INC.: *AllegroGraph Editions [online].* 2013, [cit. 2015-03-18]. Dostupné z: http://www.franz.com/agraph/allegrograph/ag_commercial_edition.lhtml

[38] FINDTHEBEST: *What was the initial release date of Neo4j? [online].* 2013, [cit. 2015-03-16]. Dostupné z: http://nosql.findthebest.com/q/12/3860/What-was-the-initial-release-date-of-Neo4j

[39] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *InfiniteGraph [online].* Říjen 2014, [cit. 2015-03-24]. Dostupné z: http://en.wikipedia.org/wiki/InfiniteGraph

[40] OBJECTIVITY INC.: *Achieve greater functionality, ease of use and even faster performance with infinitegraph 3.3 [online].* Červen 2014, [cit. 2015-03-17]. Dostupné z: http://www.objectivity.com/InfiniteGraph3.3_Release

[41] OBJECTIVITY INC.: *InfiniteGraph 3.0 Brings Unrivaled Performance and Scalability to the Only Distributed Graph Database(TM) for Big Data Analytics [online].* Říjen 2012, [cit. 2015-03-17]. Dostupné z: http://www.objectivity.com/news-archive/pr121001/

[42] BACHMAN, M.: Modelling Data as Graphs (Neo4j). Leden 2014, modelling Data in Neo4j for beginners, common mistakes, frequently asked questions, hardware sizing and a few extra tips. Dostupné z: http://www.slideshare.net/bachmanm/modelling-data-as-graphs-neo4j

[43] NEO TECHNOLOGY, INC.: *Chapter 18. Transaction Management.* [cit. 2015-03-20]. Dostupné z: http://neo4j.com/docs/stable/transactions.html

[44] RAZA, A.: Neo4j Graph Storage. Říjen 2013, describes neo4j which is a graph database. how it stores data, neo4j as emerging scalable graph db and its benchmarks. Dostupné z: http://www.slideshare.net/aliraza995/neo4j-graph-storage-27104408

[45] BACHMAN, M.: *GraphAware: Towards Online Analytical Processing in Graph Databases.* Diplomová práce, Imperial College London, 2013.

[46] UMLet Team: UMLet. [software]. Verze 13.2., 2015 [cit. 2015-26-04]. Dostupné z: http://www.umlet.com/

[47] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *B-tree [online].* Březen 2015, [cit. 2015-04-24]. Dostupné z: http://en.wikipedia.org/wiki/B-tree

[48] WIKIPEDIA: THE FREE ENCYCLOPEDIA: *Hash table [online].* Duben 2015, [cit. 2015-04-25]. Dostupné z: http://en.wikipedia.org/wiki/Hash_table

[49] BEREZOVSKÝ, M.; MAŘÍK, R.: Search trees, 2-3-4 tree, B+tree. 2012. Dostupné z: https://cw.fel.cvut.cz/wiki/_media/courses/a4m33pal/paska12x.pdf

[50] LEISHMAN, C.: *Matching relationship by ID is not optimized [online].* Září 2014, [cit. 2015-04-24]. Dostupné z: https://github.com/neo4j/neo4j/issues/3064

[51] FREEMAN, W.: Optimizing Cypher Queries in Neo4j. Březen 2014, mark and Wes will talk about Cypher optimization techniques based on real queries as well as the theoretical underlying processes. They will start from the basics of what not to do, and how to take advantage of indexes, and continue to the subtle ways of ordering MATCH/WHERE/WITH clauses for optimal performance as of the 2.0.0 release. Dostupné z: http://www.slideshare.net/neo4j/optimizing-cypher-32550605

[52] AUSTION, R.: *Chronicle Map [online].* Duben 2015, [cit. 2015-04-28]. Dostupné z: https://github.com/OpenHFT/Chronicle-Map

[53] NEO TECHNOLOGY: *Caches in Neo4j [online].* 2015, [cit. 2015-04-29]. Dostupné z: http://neo4j.com/docs/stable/configuration-caches.html

[54] BACHMAN, M.: *Introducing GraphAware Neo4j Framework [online].* Květen 2014, [cit. 2015-04-22]. Dostupné z: http://graphaware.com/neo4j/2014/05/28/graph-aware-neo4j-framework.html

[55] BACHMAN, M.: *Random Graph Models (Part I) [online].* Červenec 2014, [cit. 2015-04-30]. Dostupné z: http://graphaware.com/graph/theory/2014/07/16/random-graphs-part-one.html

[56] HUNGER, M.: *Matching exact graph pattern [online].* Březen 2015, [cit. 2015-04-24]. Dostupné z: https://groups.google.com/forum/#!searchin/neo4j/troup/neo4j/gTeFMZZ3Zuc/oHvFlrXePV0J

[57] JETBRAINS S.R.O.: IntelliJ IDEA. [software]. Verze 14.1.2., 2015 [cit. 2015-20-04]. Dostupné z: https://www.jetbrains.com/idea/

Příloha **A**

Seznam použitých zkratek

ACID Atomicity, Consistency, Isolation, Durability

AGPL Affero General Public License

API Application Programming Interface

BFS Breadth-first search

DBMS Database management systems

DFS Depth-first search

DML Data manipulation language

FIFO First In, First Out

GC Garbage collection

HTTP Hypertext Transfer Protocol

ID Identification

JSON JavaScript Object Notation

MVC Model-view-controller

NASA National Aeronautics and Space Administration

REST Representational State Transfer

SaaS Software as a Service

SOAP Simple Object Access Protocol

SSD Solid-state drive

URL Uniform Resource Locator

W3C World Wide Web Consortium

XML Extensible Markup Language

XML–RPC Extensible Markup Language – Remote procedure call

YAML Ain't Markup Language

Příloha B

Obsah přiloženého CD

```
readme.txt ................................ stručný popis obsahu CD
src
  impl .................................... zdrojové kódy implementace
  thesis .................................. zdrojová forma práce ve formátu LaTeX
  measurement ............................. výsledky měření
text ...................................... text práce
  thesis.pdf .............................. text práce ve formátu PDF
  thesis.ps ............................... text práce ve formátu PS
```