Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Matúš Roštár

Dotazování na proudy dat

Katedra softwarového inženýrství

Vedoucí diplomové práce:
Prof. RNDr. Jaroslav Pokorný, CSc.

Studijní program:
Informatika

Na tomto mieste by som rád poďakoval vedúcemu diplomovej práce Prof. RNDr. Jaroslavovi Pokornému, Csc za jeho rady a pripomienky, ktoré pomohli pri vytváraní tejto práce. Ďalej ďakujem rodičom, že ma podporovali po celú dobu štúdia, a všetkým priateľom, s ktorými som túto prácu konzultoval.

Prehlasujem, že som svoju diplomovú prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce.

V Prahe dňa 27. novembra 2007

Matúš Roštár

Obsah

Název práce: Dotazování na proudy dat
Autor: Matúš Roštár
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: Prof. RNDr. Jaroslav Pokorný, CSc.
e-mail vedoucího: Jaroslav.Pokorny@mff.cuni.cz

Abstrakt: V poslednom období vznikajú nové aplikácie, ktoré využívajú dáta nie ako perzistentnú vrstvu, ale ako dočasný prúd dát. Táto práca sa zaoberá problematikou spracovania prúdu dát, vytvárania systému SŘPD a hlavne porovnáva prístupy pri dotazovaní nad prúdmi dát. Cieľom práce je predstaviť nový, zatiaľ nevyskúšaný, prístup spracovávania dát a otestovať presnosť odpovedí pri dlhodobom dotazovaní. Súčasťou práce sú experimenty nad reálnymi dátami, ktoré porovnávajú nový algoritmus s už vyskúšanými algoritmami v minulosti.

Klíčová slova: proud dat, shlukování, posuvné okno, dotaz

Title: Querying Data Streams
Author: Matúš Roštár
Department: Department of Software Engineering
Supervisor: Prof. RNDr. Jaroslav Pokorný, CSc.
Supervisor's e-mail address: Jaroslav.Pokorny@mff.cuni.cz

Abstract: Recently a new class of data-intensive applications has become widely recognized: application in which the data is modeled best not as persistent relations but rather as transient data streams. In this thesis we address to the review of known data stream management systems, and present our own system. The purpose of this thesis is to expose our new algorithm, based on clustering, sliding window and histogram, that processes data and evaluates continuous queries. We also give experiments over real data to prove that our algorithm is more accurate than other algorithms tested out in the past.

Keywords: data stream, clustering, sliding window, query

1 Úvod

V poslednom období vyvstala nová požiadavka na spracovávanie veľkých objemov dát pri zachovaní rýchleho prístupu k nim. Vznikajú nové aplikácie, ktoré využívajú dáta nie ako perzistentnú vrstvu, ale skôr ako dočasný prúd dát. Pre príklady netreba ísť ďaleko, napríklad monitorovanie sietí, finančné aplikácie, senzorové siete, a pod. Dáta pre podobné aplikácie prichádzajú vo viacnásobných, v čase sa meniacich, nepredvídateľných dávkach, ktoré sú súčasťou typicky neohraničeného prúdu. Súčasné aplikácie potrebujú rýchlo a kvalitne spracúvať takýto typ dát. Problémy, ktoré sa vyskytli pri hľadaní riešení pre spomínané aplikácie, sa stali predmetom výskumu v databázovom svete v posledných rokoch.

Pre všetky hore uvedené aplikácie je nevhodné ukladanie dát do tradičných databázových systémov a vykonávanie dotazov nad nimi. Tradičné databázové systémy (SŘBD – systém řízení báze dat) nie sú vhodné na časté vkladanie dát, navyše ak tieto dáta prichádzajú väčšinou po jednej položke. SŘBD takisto priamo nepodporujú dlhotrvajúce dotazy(opakujúce sa dotazy v určitých časových intervaloch), ktoré sú typické pre aplikácie pracujúcimi s prúdom dát. SŘBD produkujú vždy presné, exaktné odpovede na úkor času potrebného na výpočet. Pri spracúvaní prúdu dát a vyhodnocovaní dotazov nad nimi je povolená odchýlka od skutočnej hodnoty, takisto je povolené zaokrúhľovanie výsledných hodnôt. Pri spracovaní prúdu niekedy nie je možné presný výsledok vypočítať. Z tohto hľadiska sú SŘBD na spracúvanie prúdu dát nevhodné.

Z toho dôvodu bol navrhnutý nový systém spracúvania dát SŘPD – systém řízení proudu dat. Tento systém narušuje niektoré tradičné databázové paradigma za účelom efektívneho spracovania prúdu dát. SŘPD pracuje skoro výhradne s operačnou pamäťou (niektoré statické informácie sú ukladané na vonkajšiu pamäť), vďaka čomu dosahuje rýchle odpovede. Na druhej strane je presnosť výsledkov ovplyvnená veľkosťou operačnej pamäte.

Prúd dát je možné definovať ako zoradenú sekvenciu záznamov (záznam je n-tica dát) $x_1,...,x_{n,\ ...}$, ktorá musí byť čítaná v takom poradí ako prichádzajú jednotlivé záznamy x_i a každý záznam x_i môže byť čítaný najviac jeden krát. Ďalšie vlastnosti, ktoré spĺňajú prúdy dát:

- prichádzajú neustále
- nedá sa predpokladať nič o poradí jednotlivých záznamov

- majú všeobecne neobmedzenú veľkosť
- dlhotrvajúce dotazy
- …

Predmetom riešenia tejto práce je navrhnúť systém pre spracovanie prúdu dát (SŘPD), vymedziť skupinu dlhotrvajúcich dotazov, ktoré bude systém schopný spracuvávať a pre tieto dotazy následne spraviť experimenty na reálnych, príp. generovaných dátach. Experimenty vyhodnotiť a vyvodiť závery.

V pôvodnom zadaní diplomovej práce je uvedené, že cieľom práce je implementovať prototyp prostredia pre spracovanie dát. Keďže táto úloha je veľmi obsiahla, rozhodli sme sa, že prácu zameriame na konkrétnejší cieľ, a to zlepšenie jednej triedy algoritmov - algoritmov vyhodnocovania dlhodobých dotazov. Tento algoritmus tvorí jadro systému SŘPD a slúži na zisťovanie hodnôt funkcií na základe znalosti prefixu prúdu dát. Efektivita tohto algoritmu podmieňuje presnosť výsledku dotazu, rýchlosti vyhodnotenia a náročnosť na priestor.

Preto hlavným cieľom práce je implementovať vylepšený algoritmus na vyhodnocovanie dlhodobých dotazov, na vhodných testovacích sadách otestovať presnosť a rýchlosť algoritmu a porovnať výsledky testov so súčasne používaným algoritmom.

Práca je členená na dve veľké podčasti – teoretickú (kapitoly 2 a 3), kde sú vysvetlené teoretické základy práce a praktickú (kapitoly 4, 5 a 6), kde je popísaná implementácia a výsledky testov.

V kapitole 2 je podrobne vysvetlená architektúra systému SŘPD. Je tu popísaná analýza systému SŘPD, jeho podčasti plánovanie a vykonávanie. Detailnejšie sa zaoberá algoritmami na vyhodnocovanie dlhodobých dotazov a špeciálne sa venuje teoretickému pozadie algoritmu zhlukovania.

V kapitole 3 je popísaný algoritmus zhlukovania v kombinácií s technikou posuvného okna. Práve tento algoritmus je novým algoritmom na spracovávanie dotazov, ktorý nebol zatiaľ v praxi otestovaný. Jadrom práce je tento algoritmus otestovať a porovnať výsledky testov s algoritmami, ktoré boli implementované v minulosti.

V kapitole 4 je predstavený systému SŘPD, ktorý bol implementovaný za účelom otestovania algoritmu uvedeného v kapitole 3. Nachádza sa tu ako popis vstupnej časti, tak aj exekučnej časti systému. Dôraz je kladený hlavne na presné zdokumentovanie implementácie

spomínaného algoritmu.

V kapitole 5 je zdôvodnený spôsob testovania. Je tu popísané, aké testovacie dáta boli použité a aké algoritmy boli testované. Kapitola 6 obsahuje výsledky meraní. Kapitola 7 vyhodnocuje výsledky meraní a vyvodzuje závery. V kapitolách 8 a 9 sú načrtnuté možnosti zlepšenia riešenia, prípadne jeho rozšírenia. Nakoniec v kapitole 9 sa nachádza záverečný sumár práce, kde sú zhrnuté všetky dôležité poznatky.

1.1 Základná terminológia

Formalizácia pojmov je prevzatá z [21]. Na definíciu pojmov spojených so SŘPD je dobré začať definíciou postupnosti. *Postupnosť* je usporiadaná množina prvkov (n-tíc, správ), ktorá môže obsahovať duplicitné prvky. V tradičnom chápaní relačného modelu dát nie je poradie prvkov v postupnosti podstatné. Dôležitým pojmom, ktorý je nutný definovať pre ďalšie účely je prúd dát. *Prúd dát* je postupnosť prvkov, ktorej dĺžka nie je obmedzená. Prvky prúdu dát sú zoradené podľa časového razítka, ktoré odpovedá vstúpeniu prvku do prúdu. Prúd dát si možno predstaviť ako reláciu, ktorá je čo do veľkosti neobmedzená, špeciálne usporiadaná.

Pri dotazovaní nad prúdom dát je nutné rozlíšiť tri základné možnosti *modelu prúdu dát*. Prvou je možnosť – vstup: prúd dát, výstup: prúd dát. Druhou možnosťou je – vstup: prúd dát, výstup: relácia a treťou možnosťou je kombinácia predchádzajúcich dvoch.

Pri využívaní SŘPD je možné klásť dotazy na dáta, podobne ako pri bežnom SŘBD. Tieto dotazy nie sú pri SŘPD všetky rovnakého typu, je potrebné rozlíšiť niekoľko kategórií. Prvou z nich sú *ad-hoc dotazy.* Tieto dotazy sú dotazy položené jednorázovo v ľubovolnú dobu. Naproti tomu *dlhotrvajúce(nepretržité, kontinuálne) dotazy* je nutné najprv zaregistrovať a potom sú vyhodnocované neustále v presne určených intervaloch. Dotazy, ktoré využívajú dáta len s jedným spoločným časovým razítkom sú nazývané *one-time (v danom čase) dotazy.* Najväčším problémom pre stavbu optimalizátoru SŘPD predstavujú ad-hoc dotazy.

Na dotazovanie nad SŘPD je nutné zaviesť dotazovací jazyk, ktorý by čo najlepšie využíval možnosti SŘPD a súčasne eliminoval jeho nevýhody. Najvýhodnejšou alternatívou pri tvorbe dotazovacie jazyka je prevzatie jazyka SQL a jeho úprava špeciálnym požiadavkám prúdového spracovania. Do takéhoto jazyka je možné použiť iba tzv. *neblokujúce operátory*, ktoré sú charakteristické tým, že nepotrebujú pre produkovanie výstupu načítať celý vstup. Táto

vlastnosť je dôležitá z pohľadu neobmedzenosti prúdu dát, kde by blokujúci operátor musel čakať až na koniec prúdu pre vyprodukovanie výsledku. *Blokujúci operátor* je operátor, ktorý nie je schopný vyprodukovať prvú n-ticu bez toho, aby nenačítal celý vstup. Hlbšie sa problematike blokujúcich operátorov venuje sekcia 2.4 Blokujúce operátory. Príklad blokujúcich operátorov v jazyku SQL je napríklad NOT EXISTS, NOT IN, agregačné operátory, apod.

Ďalšou dôležitou súčasťou sú synopsie. *Synopsie* sú objekty, ktoré uchovávajú stav operátora a slúžia na uchovávanie dočasných informácií. Každý výskyt operátoru má práve jednu synopsiu. Môžu sa tu uchovávať veľké množstvá dát, preto je niekedy užitočné postaviť nad synopsiou index. Indexy pomáhajú pri rýchlejšom hľadaní dát v synopsiách. Jednoduchšie povedané, synopsie sú dočasné, pomocné metadáta, kde je možné ukladať medzivýsledky pri vykonávaní dotazu.

Taktiež je výhodné zaviesť pojem posuvného okna pre dotazovanie nad prúdmi dát. *Posuvné okno* ohraničuje (vymedzuje) dotazované prvky buď podľa časového princípu (posledných 5 minút) alebo podľa počtu prichádzajúcich prvkov (posledných 100 prvkov). Posuvné okno ponúka niekoľko variant, napríklad okno pevnej dĺžky, alebo okno iba s pohyblivým jedným koncom. Posuvné okno je častou technikou využívanou pri tvorbe aproximovaných odpovedí vďaka svojej jednoduchej implementácií.

Prúdy dát a ich využitie prinášajú nové výzvy aj v oblasti *dolovania dát*. Dolovanie dát je disciplína, ktorá sa zaoberá extrakciou modelov a vzorov z rozsiahlych dát. V praxi je možné rozdeliť metódy dolovania dát do dvoch hlavných skupín – *metódy založených na dátach* a *metódy úlohovo orientované*. Metódy založené na dátach vyberajú časť, podmnožinu dátového prúdu a na základe neho produkujú analýzu. Radia sa sem techniky vzorkovania, skicovania, ktoré sú detailne popísané v sekcií 2.5 Algoritmy procesoru dotazov. Úlohovo orientované metódy sú napríklad aproximačné algoritmy, ktoré predpokladajú, že algoritmus je výpočtovo zložitý a preto pristupujú k aproximáciám pri výpočtoch.

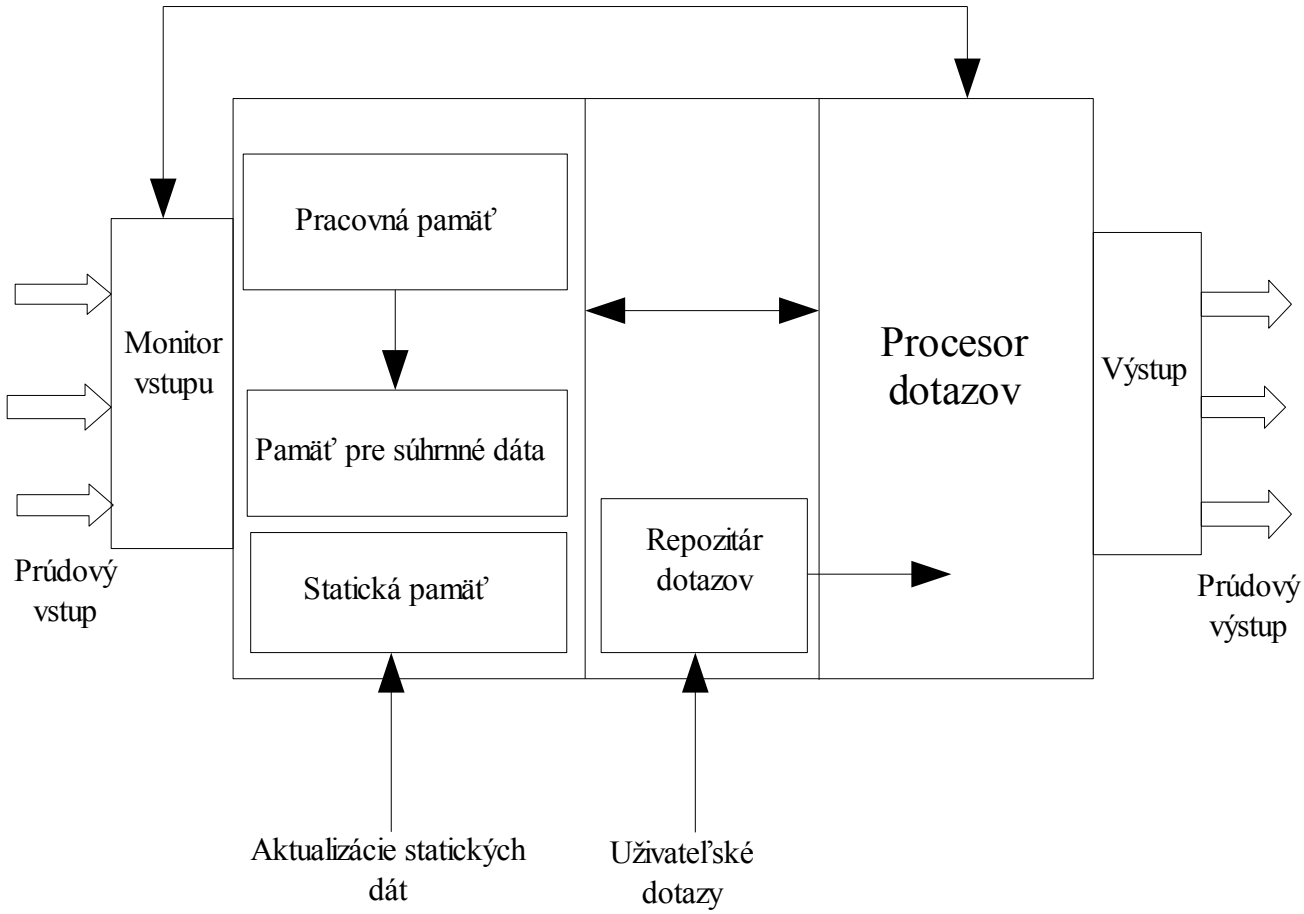
2 Analýza všeobecného systému SŘPD

Pri spracovávaní prúdu dát je nemožné kontrolovať poradie prvkov, takisto je veľmi nevhodné ukladať na vonkajšiu pamäť časti prúdu. Naopak, dlhotrvajúce dotazy bežia stále a produkujú výsledky v určitých intervaloch v závislosti na novo príchodzích dátach. V práci [1] sú definované podmienky kladené na všeobecný SŘPD systém, ktorý bude splňovať hore uvedené nároky.

- dátový model a sémantika dotazu musí povoľovať časové a poradové operácie (napr. definícia posuvného okna pre posledných 5 minút)
- neschopnosť uložiť kompletný prúd dát je predpokladom pre vytváranie štruktúr, ktoré zabezpečia aproximačné odpovede, napr. synopsie. Preto výsledky na niektoré dotazy nemusia byť exaktné, presné
- kvôli výkonnosti, rýchlosti a nárokom na priestor nie je dovolené vracanie sa a čítanie už raz načítaných hodnôt, je dovolené iba jedno čítanie
- vstupný monitor systému musí rýchlo a správne reagovať na nezvyčajné alebo chybné vstupné dáta
- dlhotrvajúce dotazy môžu doznať zmien počas ich života vzhľadom na meniacu sa povahu prúdu dát, ktorý dotazujú (napr. zmena frekvencie príchodzích dát)
- je nutné, aby systém vykonávania dotazov dokázal pre rôzne dlhotrvajúce dotazy pristupovať k prúdom dát ako ku zdieľaným zdrojom

2.1 Architektúra SŘPD

V poslednom období boli navrhnuté architektúry, ktoré by spravovali systém SŘPD. Základom pre všetky dosiaľ navrhnuté architektúry sa stala abstraktná, všeobecná architektúra predstavená v [5].



Obr 2.1: Všeobecná architektúra pre stroje spracúvajúce prúdy dát

Architektúra sa skladá zo siedmych základných častí monitor vstupu, pracovná pamäť, pamäť pre súhrnné dáta, statické dáta, repozitár dotazov, procesor dotazov a výstupný buffer (monitor).

Monitor vstupu slúži na regulovanie vstupu. Vstupom je niekoľko prúdov dát. Úlohou monitoru vstupu je tieto prúdu sledovať a spracovávať do vnútornej štruktúry. Musí takisto dohliadať na frekvenciu vstupu z jednotlivých prúdov a v prípade zvýšenej frekvencie zahadzovať pakety, ktoré sú prebytočné. Taktiež je nutné kontrolovať stav voľnej pamäte a podľa toho regulovať veľkosti posuvných okien apod.

Všetky dáta sú ukladané do jednej z troch pamätí: pracovná pamäť, pamäť pre súhrnné dáta, pamäť pre statické dáta. Pracovná pamäť slúži na uchovávanie aktuálne spracovávaných dát, napr. posuvné okná. Pamäť pre súhrnné dáta je určená pre uloženie dát, ktoré bližšie špecifikujú prúd dát, napríklad značky, synopsie, skice a podobne. Tieto pomocné dáta sú

vypočítavané na základe príchodzieho prúdu dát, ktorý je v pracovnej pamäti. V statickom repozitári sú uložené typicky informácie, ktoré sa v priebehu času nemenia, napríklad umiestenie zdroja prúdu, popis atribútov prúdu a podobne.

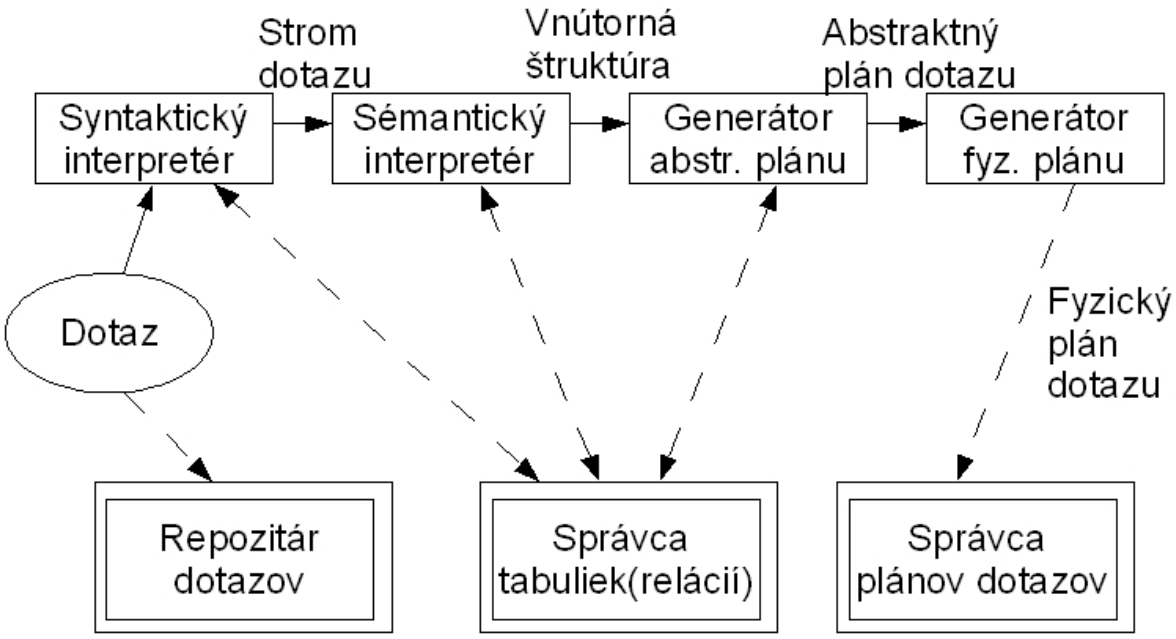
Dlhodobé dotazy sú registrované v repozitári dotazov. Odtiaľto sú vyzdvihnuté a vykonávané. Ad-hoc dotazy môžu byť uložené buď tu alebo sú priamo spracované v procesore dotazov. Je dôležité poznamenať, že dotazy pristupujú k dátam ako ku zdieľaným zdrojom.

Procesor dotazov je hlavnou časťou systému. Má za úlohu plánovanie vykonávania dotazov a ich samotné vykonanie. Komunikuje so vstupným monitorom a môže preplánovať vykonávanie na základe získaných informácií. Výsledky z procesora dotazov sú buď dočasne ukladané, alebo priamo posielané na výstup ako prúd dát. Druhá možnosť je výhodnejšia v tom, že užívatelia môžu pozmeniť svoje dotazy na základe zatiaľ získanej výsledkovej sady.

2.2 Plánovanie

Plánovanie je súčasťou systému, ktorá sa stará o plánovanie vykonávania registrovaných dotazov. Registrované dotazy sú uložené v repozitári dotazov. Sú to jedny z mála dát, ktoré je nutné ukladať na disk, kvôli znovupoužiteľnosti po reštarte systému. V súčastnosti neexistuje jeden abstraktný model plánovania, ale niekoľko prototypov, ktoré sa líšia značným spôsobom. V každom z týchto prístupov je jadrom komponenta *plánovač*. Jedná sa o komponentu, ktorá má za úlohu rozhodovať, v akom poradí sa majú vykonávať elementárne dotazy a v akom poradí sa majú vykonávať operátory použité v dotaze. Musí riadiť tok dát (medzivýsledky) medzi jednotlivými operátormi.

Jednou z možností ako riešiť problém plánovania je už implementovaná v projekte STREAM [22]. Tento projekt je jedným z prvých, ktorý sa pokúsil riešiť komplexne problematiku SŘPD. Bol vyvinutý na Stanfordskej Univerzite a retrospektívne sa ukazuje ako kvalitne navrhnutý spomedzi ostatných konkurenčných projektov. V tejto práci sa bude zväčša vychádzať z poznatkov získaných z tohto projektu. Obrázok 2.2 ukazuje štruktúru plánovacieho systému. Komponenty označené dvojitou čiarou sú prvky, ktoré udržiavajú určitý druh metadát. Komponenty s jednoduchou čiarou slúžia len na výpočtové účely a majú za úlohu transformovať dotaz do konečnej, naplánovanej podoby.



Obr 2.2: Komponenta plánovania v projekte STREAM

Vstupom celého systému plánovania je dotaz zapísaný v jazyku CQL, čo je prispôsobený jazyk SQL pre potreby systému SŘPD vyvinutý špeciálne pre projekt STREAM. Tento dotaz je získaný z repozitáru dotazov. Dotaz je najprv v syntaktickom interpretéri skontrolovaný a transformovaný na strom dotazu. Zároveň syntaktický interpretér načíta informácie o reláciách, ktoré sú použité v dotaze. Tieto informácie sa nachádzajú v komponente Správca tabuliek.

Po vytvorení stromu dotazu ho sémantický interpretér transformuje do vnútornej štruktúry systému (ešte sa nejedná o strom operátorov). V rámci tejto transformácie musí interpretér zistiť súvislosti medzi jednotlivými atribútmi (typy, konverzie, …) pomocou správcu tabuliek, musí doplniť štandardné sémantické skratky jazyka CQL (SQL), ako napr. doplnenie hviezdičky na názvy jednotlivých atribútov a podobne. Musí taktiež konvertovať názvy atribútov na vnútornú reprezentáciu (v prípade STREAM ukazateľ na konkrétne dáta).

Generátor abstraktného plánu je prvkom, ktorý má za úlohu vytvoriť abstraktný plán dotazu. Ide o plán zostavený výhradne z abstraktných operátorov. Abstraktné operátory sú veľmi podobné tradičným operátorom relačnej algebry (projekcia, spojenie, …), ale zároveň sa tu nachádzajú aj vlastné, špeciálne vytvorené pre potreby prúdového spracovania (posuvné okno,

...). Generátor abstraktného plánu má navyše za úlohu:

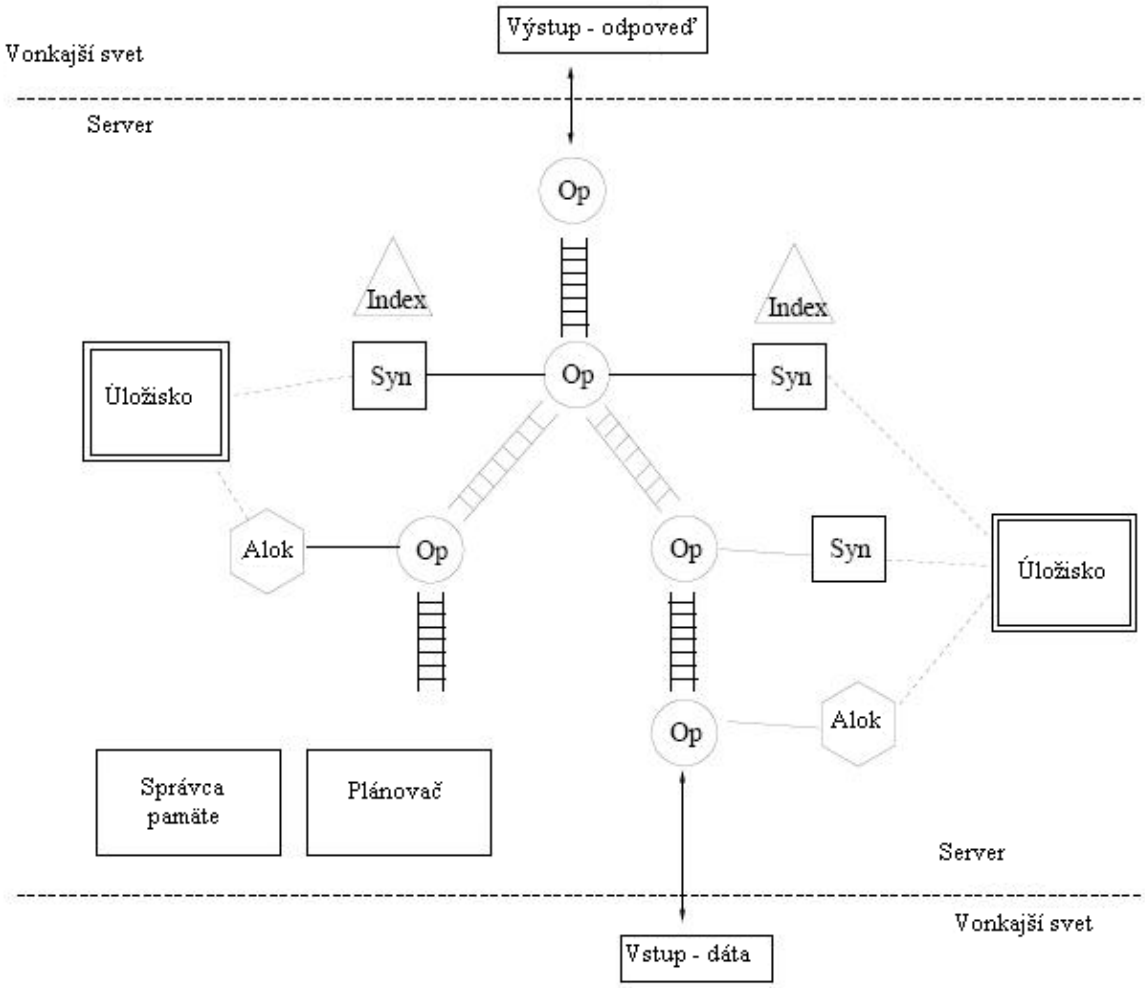
- operátory selekcie vykonávať skôr ako operátory spojenia
- odstraňovať redundantné operátory v rámci jedného prúdu
- odstraňovať redundantné operátory v rámci viacerých prúdov

Výstupom z generátoru abstraktného plánu je abstraktný plán, ktorý je zároveň vstupom pre poslednú komponentu tohto subsystému – generátor fyzického plánu. Táto komponenta už prevádza abstraktný plán na presné fyzické operácie, ktoré korešpondujú s tými, ktoré sú definované vo vykonávacej časti. Abstraktný plán bol vytvorený z toho dôvodu, pretože je jednoduchšie pracovať s abstraktnými operátormi a zjednodušovať dotaz pomocou nich.

V statickej komponente správca plánov dotazov sú udržiavané všetky fyzické plány všetkých registrovaných dotazov.

2.3 Exekučná časť

Vykonanie dotazu, jeho prevedenie vo fyzickom zmysle slova, má na starosti exekučná časť systému. Vykonávací stroj je jedna z najzložitejších komponent systému. Na obrázku 2.3 je zobrazený model, ktorý je použitý v projekte STREAM. Zobrazuje tri základné entity exekučnej časti (operátory, synopsie a alokátory) a ich vzájomnú interakciu.



Obr 2.3: Systém vykonávania dotazov v projekte STREAM

Symbolom Op sú znázornené operátory, Syn synopsie a Alok alokátory miesta. Na obrázku figurujú aj ostatné komponenty statického charakteru – úložisko, správca pamäte a plánovač.

Priebeh práce exekučnej časti je pomerne zložitý a preto budú len načrtnuté hlavné rysy tohto subsystému. V prvom rade je dôležité definovať formát dát, v akom prúdia subsystémom. Jedná sa o päť druhov formátov rozdelených do dvoch podskupín:

- Nízkoúrovňové
  - n-tica hodnôt atribútov - je základnou jednotkou dát. V tradičnom chápaní databáze sa jedná o jeden riadok tabuľky

- element – je n-tica s pridaným špeciálnym príznakom „+“ alebo „–“ a súčasne s časovým razítkom, ktoré vyjadruje čas (vnútorný) vzniku tohto elementu
- heartbeat – je to len časové razítko, ktoré sa využíva na správne časovanie medzi operátormi

- Vysokoúrovňové
  - relácia – ide o skupinu elementov zoradených podľa časového razítka, elementy s príznakom + označuje, že záznam sa má do relácie pridať a elementy s príznakom -, že záznam sa má z relácie odobrať
  - prúd – je skupina elementov, ktoré obsahujú iba príznak +. Ide v podstate o špeciálny prípad relácie

Okrem dát je nutné definovať funkciu operátorov. Operátory sú základnou exekučnou jednotkou, ktorá pracuje nad reláciami a prúdmi (vysokoúrovňovými dátami). Pre operácie nad nízkoúrovňovými dátami sú využívané takzvané evaluátory. Operátor vezme sa svoj vstup prúd alebo reláciu, príp. obe a vyprodukuje prúd alebo reláciu ako svoj výstup.

Každý operátor pracuje v kontinuálnej podobe, to znamená že spracúva dáta tak ako prichádzajú. Akonáhle uvidí operátor elementy s časovým razítkom menším ako T, produkuje svoje výstupné elementy opäť len s časovým razítkom menším ako T. Problém, ktorý sa ponúka je, či a ako presne dokáže operátor, alebo všeobecne nejaká funkcia vypočítať svoju funkčnú hodnotu na základe čítanie iba určitej časti vstupu. Týmto problémom sa zaoberá sekcia 2.5 Algoritmy procesoru dotazov.

2.4 Blokujúce operátory

Blokujúci operátor je operátor dotazu, ktorý nie je schopný vyprodukovať prvý záznam výstupu, pokiaľ neprečítal všetky záznamy vstupu. Triedenie je príkladom blokujúceho operátora, ako aj agregačné operátory MIN, MAX, a AVG. Ak je uvažované o vyhodnocovaní dlhodobých dotazov pri prúde dát tak, ako na tradičné vyhodnocovanie v SŘBD pomocou stromu operátorov, kde vstupné záznamy z prúdu sú načítané do listov a výsledok je dopočítaný

do koreňa, potom by včlenenie blokujúcich operátorov do takéhoto stromu pôsobilo značné problémy. Dlhodobé prúdy môžu byť nekonečné a preto by blokujúci operátor potreboval načítať do listu celý prúd ako jeden zo svojich vstupov, čo by spôsobilo, že by nikdy nevyprodukoval výstup.

Je zrejmé, že blokujúce operátory nie sú veľmi vhodné pri práci s prúdmi dát. Na druhú stranu sú napríklad agregačné funkcie veľmi bežné a používané. Odstránenie blokujúcich operátorov z výpočtového modelu SŘPD by určite používateľom spôsobovalo problémy pri formulovaní dotazov. Vysporiadanie sa s nimi efektívnym spôsobom je jednou z najväčších výziev pri vytváraní modelu SŘPD. Ťažiskom tejto práce je zhodnotiť nový prístup spracovania prúdu dát, kde je možné využívať aj blokujúce operátory. Takéto spracovanie potrebuje využívať špeciálne algoritmy procesoru dotazov, ktoré dokážu spracovávať blokujúce operátory.

Blokujúce operátory, ktoré sú v koreni pomyselného stromu sú jednoduchšie na vyriešenie ako operátory, ktoré sú vo vnútorných uzloch stromu. Vnútorné uzly stromu produkujú medzivýsledky, ktoré sú ďalej predávané ostatným operátorom pre následné spracovanie (napríklad triediaca časť operátoru spojenia alebo použitie agregačného operátoru v poddotaze). V prípade blokujúceho operátoru v koreni stromu, ak tento operátor vyprodukuje na výstup jeden záznam alebo malý počet záznamov, potom výsledok môže byť menený v priebehu času pomocou UPDATE. Ak operátor produkuje väčšie objemy dát, napríklad v prípade, že odpoveď dotazu je relácia, ktorá má byť utriedená, je oveľa praktickejšie spracovávať odpoveď vo vnútornej pamäti a výsledok poskytnúť až neskôr, pretože kontinuálne odpovede by mohli byť nepresné a zavádzajúce. Avšak ani jeden z týchto prístupov nie je vhodný pre operátory vo vnútorných uzloch. Problém je v tom, že výsledky blokujúcich operátorov sa menia v čase vzhľadom na príchodzie dáta z prúdu a preto aj operátory, ktoré spracúvajú tieto výsledky vyššie v strome vyhodnocovania dotazu pracujú s nepresnými dátami a preto robia často nepresné a niekedy úplne chybné rozhodnutia.

Jeden z prístupov ako vyriešiť problém s blokujúcimi operátormi vo vnútorných uzloch, je nahradiť ich neblokujúcimi operátormi, ktoré sú ekvivalentné. Ako príklad môže poslúžiť operátor *juggle* [3], ktorý je neblokujúcim variantom operátoru sort. Funguje tak, že sa zameria na lokálnu časť prúdu a tú poprehadzuje do správneho poradia, tak aby záznamy vstupovali do ďalšieho spracovanie už zotriedené. Otázkou ostáva, ako nájsť podobné ekvivalentné operátory k ostatným blokujúcim operátorom.

V práci [4] sa objavuje iný prístup k riešeniu problému blokujúcich operátorov. Navrhuje doplňovať prúd dát o akési značky alebo poznámky o tom, aké dáta sa môžu v zbytku prúdu vyskytovať. Tieto značky by mali prekladať jednotlivé elementy prúdu dát. Príklad takéhoto typu značky môže byť napríklad pre atribút *deňVmesiaci* - pre všetky budúce záznamy *deňVmesiaci* > 10. Akonáhle agregačný operátor zistí takúto značku, môže všetky dni, ktoré sú menšie ako 10 automaticky zahadzovať. Podobne operátor spojenia môže zahodiť všetky záznamy, ktoré má uložené v pamäti s dňami menšími ako 10 a tak znížiť spotrebu pamäte.

2.5 Algoritmy procesoru dotazov

Za posledných päť až desať rokov sa jedným z hlavných cieľov databázovej komunity stalo vyvinúť algoritmus, ktorý by dokázal efektívne vyhodnocovať dotaz s blokujúcimi operátormi. Dôležitým faktorom pre tento algoritmus je presnosť, s akou dokáže vytvárať odpovede. Druhým faktorom je rýchlosť spracovania dotazu. Na to aby požadovaný algoritmus vedel spracovať blokujúci operátor, potrebuje v prvom rade vedieť vypočítať hodnotu funkcie len na základe znalosti prefixu vstupu tejto funkcie. Presnejšia špecifikácia takéhoto algoritmu by sa dala formulovať nasledovne:

„Algoritmus vezme za svoj vstup postupnosť dátových položiek $x_1, x_2, \ldots x_n, \ldots$ nazvanú prúd dát. Tento prúd dát môže algoritmus prečítať iba jeden krát a to v poradí zvyšujúceho sa indexu. Od algoritmu je požadované, aby vypočítal hodnotu funkcie f, ak prečítal iba prefix prúdu. *Prefixom prúdu* nazveme postupnosť dátových položiek $x_1, x_2, \ldots x_k$, *kde* $k < n$.“

Hlavným problémom pri vytváraní takéhoto algoritmu je náročnosť na pamäť, v druhom rade sú kladené aj podmienky na efektívne vyhodnotenie v čase. Čas a priestor je meraný v závislosti na počte prijatých dátových položiek, označme ho *N*. Hlavným cieľom je zaistiť, aby nároky na priestor boli „malé“. Ideálne riešenie by stavalo nároky na priestor nezávisle od *N*, ktoré môže byť teoreticky neobmedzené. Avšak je ľahké nahliadnuť, že odhad dolnej hranice vylučuje takúto možnosť. Tento odhad nemôže byť lepší ako minimálne závislý od *N*. Problém je vo všeobecnosti považovaný za *dobre vyriešený*, ak je pamäťová náročnosť *O(poly(log N))* a časová náročnosť *O(poly(log N))*. Avšak v niektorých zložitých prípadoch je nemožné dosiahnuť takto kvalitné výsledky.

Presnosť použitého algoritmu je priemerná odchýlka výsledkov algoritmu od skutočnej hodnoty funkcie, ktorú algoritmus počíta. Pre získanie skutočnej hodnoty funkcie, a teda pre správne vyhodnotenie presnosti algoritmu je nutné načítať celý prúd dát, ktorý môže byť nekonečný. Preto pri testoch vo všetkých prácach zaoberajúcimi sa touto problematikou, vrátane tejto, je prúd dát je zhora obmedzený.

Databázová komunita sa v poslednom čase snaží vyvinúť vhodný algoritmus, ktorý by riešil horeuvedený problém s čo najlepšími výsledkami s ohľadom na presnosť a súčasne na rýchlosť. Vzniklo niekoľko rôznych základných techník, na ktorých je možné požadovaný algoritmus založiť. Od jednoduchších ako *vzorkovanie, posuvné okno* alebo *skicovanie* k zložitejším *histogramom* a *vlnkám* alebo *zhlukovaniu.*

2.5.1 Vzorkovanie

Vzorkovanie je staršia metóda, ktorú je možné využiť v prípade, že je možné vytvoriť malú, jednoduchú vzorku zo základných údajov, ktoré obsahuje prúd dát. Na vytváranie vzorky sa využívajú rôzne pravdepodobnostné a štatistické metódy. Jedná sa zrejme o najjednoduchšiu formu sumarizácie dát v SŘPD a ďalšie algoritmy môžu využívať vzorkovanie ako svoju podčasť. Vzorkovacie algoritmy, ktoré boli navrhnuté je možné rozdeliť do viacerých skupín:

- doménové vzorky
- všeobecné vzorky
- úložiskové vzorky
- nezávislé vzorky
- atď.

Príklad vzorkovania, ktorý bol navrhnutý v [6], je veľmi jednoduchý a je základom pre všetky ostatné typy vzorkovania. Hlavná myšlienka algoritmu je opakovane generovať číslo S, ktoré vyjadruje počet preskočených záznamov. Prvý nepreskočený záznam je uložený do úložiska. V úložisku je uchovávaných N kandidátov. Títo kandidáti sú reprezentanti prúdu.

```
Algoritmus náhodného vzorkovania
while not „koniec prúdu“ do
begin
     Vygeneruj náhodné číslo S(n, t); // závislé na čase t a
                                          počte načítaných záznamov
     ZAHOĎ_ZÁZNAMY(S); // nespracuje(preskočí) S záznamov
     if not „koniec prúdu“ then
          begin
               ULOŽ_ZÁZNAM(t);
               ČÍTAJ_ĎALŠÍ_ZÁZNAM()
          end
     t := t + S + 1;
end,
```

2.5.2 Skicovanie

Skicovanie je metóda založená na náhodnom premietaní podmnožiny vlastností. Je to proces vertikálneho vzorkovania dátového prúdu. Nech $S = x_1,...,x_N$ je postupnosť elementov, kde každý x_i patrí do domény $D=\{1,...,d\}$. Nech frekvencia je daná ako $m_i = |\{j|\ x_j = i\}|$ a určuje počet výskytov hodnoty i v postupnosti S. Pre $k \geq 0$, nazveme k-tym frekvenčným momentom S

$F_k = \sum_{i=1}^{d} m_i^k$. Potom je ľahké nahliadnuť, že F_0 je počet rôznych hodnôt v sekvencií, F_1 je dĺžka sekvencie. Pre všetky hodnoty F je bohužiaľ potrebný lineárny priestor. Skicovanie je metóda, ktorá sa snaží odhadovať uvedené hodnoty s nižšou pamäťovou náročnosťou.

Skicovanie je možné použiť pre porovnanie dátových prúdov a pre agregačné dotazy. Hlavnou nevýhodou skicovania je nepresnosť.

2.5.3 Histogramy

Histogramy sú bežne využívané štruktúry na uchovávanie metadát pomocou vytvárania stručných sumárov o distribúcií hodnôt v dátovej množine (stĺpec, tabuľka, ..). Sú používané na rôzne úlohy, cez odhady dotazov a približných výsledkov na dotazy až po dolovanie dát z

prúdových zariadení. Existuje niekoľko druhov histogramov, z ktorých najzaujímavejšie sa javia V-optimal histogramy, equi-width histogramy a end-biased histogramy.

V práci [7] je popísaný algoritmus V-optimal histogramu. Tento algoritmus očakáva na svojom vstupe zotriedenú postupnosť. Na jej základe pomocou metódy dynamického programovania dokáže vytvoriť histogramy v čase $O(N^2B)$ a pamäťovej zložitosti O(N), kde N je veľkosť vstupu a B je počet priehradiek. Priehradka je interval, ktorý ohraničuje hodnoty. Zjednotenie všetkých priehradiek dá vo výsledku celkovú dátovú sadu. Samozrejme, že presnosť výsledku operátoru pri používaní histogramu závisí od presnosti aproximácie, ktorá je podmienená dvomi faktormi. Technikou delenia hodnôt medzi jednotlivými priehradkami a aproximačnou technikou používanou v rámci každej priehradky. V neskoršej práci [8] bola odstránená podmienka na zotriedenú postupnosť a namiesto toho bol využitý algoritmus skicovania.

Druhým typom histogramu je equi-width histogram. Tento typ histogramu rozdeľuje hodnoty do priehradiek alikvotne. Inými slovami, počet hodnôt spadajúcich do jednotlivých priehradiek je rovnaký pre každú priehradku. V práci [9] bol predstavený algoritmus, ktorý využíva jeden prechod dátami.

End-biased histogramy sú úzko spojené s problematikou tzv. iceberg query – dotazy na špičke ľadovca. Iceberg query je agregačný dotaz, ktorého zaujímajú len výsledky nad nejakým určeným prahom. Takéto dotazy sú časté v rôznych aplikáciách, zahrňujúcich dolovanie dát, zhlukovanie, získavanie informácií apod. V práci [10] je popísaný algoritmus nad perzistentnými dátami na disku, ktorý využíva viac prechodov a rieši problém iceberg query. V neskoršej práci [11], ktorá vychádzala z tejto práce, sú prezentované náhodné a deterministické algoritmy pre počítanie frekvencie údajov a počítanie iceberg query pre dátové prúdy s jedným prechodom. Náhodný algoritmus používa vzorkovanie a hlavná myšlienka je, že ak sa položka nachádza v nejakej frakcií všetkých položiek, potom je vysoko pravdepodobné, že bude časťou jednotného vzorku o veľkosti 1/e. Deterministický algoritmus spracúva vzorky rozdielnych položiek a ich frekvencie. Kedykoľvek je pridaná nová položka, ktorá už existuje vo vzorku, tak je zvýšená jej frekvencia. Periodicky je zoznam prechádzaný a vzorky s malou frekvenciou sú mazané. End-biased histogramy zobrazujú stĺpce s frekvenciou väčšou ako je prah presne, zvyšok je aproximovaný do jedného stĺpca.

2.5.4 Vlnky

V*lnka* je nástroj na hierarchické rozdelenie funkcií/prvkov/signálov podľa zvolených kritérií. V praxi je využívaná najmä *Haarova vlnka*, čo je veľmi jednoduchá funkcia rozdelujúca spektrum do troch skupín(-1, 0, 1). Vlnky sú často používané synopsie napríklad pri tvorbe histogramov alebo zhlukovaní.

V nedávnych výskumných prácach bola ukázaná vysoká efektivita vlniek pre rozličné úlohy, ako odhady selekcie, aproximácie dát, alebo počítanie multi-dimenzionálnych agregátov. Všetky práce[12] dokazujú, že odhady získané pomocou algoritmov vlniek sú presnejšie ako tie, ktoré sú získané za pomoci histogramov pri použití tej istej pamäte. Tak isto ako v iných algoritmoch, aj pri algoritme vlnky boli najprv vytvorené metódy pre statické dáta [12]. Až v neskoršom období bola táto metóda rozšírená o spracovanie prúdu dát. Technika(popísaná v [13]) aproximácie najlepšieho dvojčlenného intervalu, ktorý najviac znižuje chybu, dala vzniknúť jednoduchému hltavému algoritmu, ktorý nájde najlepšiu vlnkovú reprezentáciu.

2.5.5 Posuvné okno

Ďalšou technikou pre vyhodnocovanie približných odpovedí je posuvné okno. Vyhodnocuje dotaz na základe najposlednejších záznamov a nie na základe celej histórie. Napríklad, iba dáta za posledný týždeň sú zahrnuté do výpočtu, staršie dáta sú zahadzované.

Implementácia posuvného okna pre prúdy dát je prirodzenou formou aproximácie , ktorá má niektoré významné výhody. Je dobre definovaná a ľahko pochopiteľná. Užívateľ, ktorý využíva posuvné okná vie, čo môže od výsledku očakávať a podľa toho bude svoj dotaz formulovať. Je deterministická, takže nevzniká žiadne riziko vzniknutia chyby spôsobenej nesprávnym výberom náhodných hodnôt (vzorkovanie, skicovanie, …). A čo je najdôležitejšie, pracuje s najnovšími dátami, o ktoré je v drvivej väčšine aplikácií najväčší záujem. Posuvné okná a ich využitie pri práci s prúdmi dát sú predmetom výskumu v posledných rokoch.

Hlavnou motiváciou tejto práce je spojenie metódy posuvného okna s niektorým aproximačným algoritmom. V práci [15] bolo prvýkrát predvedené takéto spojenie so skicovaním. Táto technika potrebovala pamäťovú zložitosť $O(\frac{1}{e.logN})$, kde N je veľkosť

posuvného okna a *e* je parameter presnosti. Takisto tu boli predstavené niektoré dolné obmedzenia pre problémy s posuvnými oknami.

Stále neboli preskúmané efektivity niektorých algoritmov v kombinácií s posuvným oknom. Jedná sa napríklad o zhlukovanie, vlnky, … apod. Hlavným cieľom tejto práce je preskúmať a implementovať algoritmus zhlukovania na posuvné okná.

2.6 Zhlukovanie

Zhlukovanie, ako nástroj pri dátovej analýze, je algoritmus, ktorý rieši problém nájdenia častí dátovej množiny takých, aby boli podobné záznamy v rovnakej časti a rôzne v rôznych častiach. Jedným z prípadov zhlukovania, ktorým sa zaoberá aj táto práca, je úloha nájdenia k-mediánov, t.j. nájdenia k centier takých, že suma vzdialeností každého bodu od najbližšieho centra bude minimalizovaná.

2.6.1 Teoretická analýza zhlukovania

Pre najprirodzenejšiu funkciu zhlukovania je riešenie takejto funkcie optimalizačným problémom z množiny NP–ťažkých. Preto existuje veľa teoretických štúdií, ktoré sa snažia navrhnúť aproximačné algoritmy, ktoré by garantovali riešenie funkcií v rámci nejakej pevnej odchýlky od optimálneho riešenia.

Hľadanie k-mediánov, alebo hľadanie *k*-centier, je jedným z týchto aproximačných algoritmov. Riešenie pozostáva z *k* záznamov dátovej množiny, *k* je je počet požadovaných centier (vstupný parameter). Tieto záznamy sú vybrané ako centrá, každý iný záznam z dátovej množiny je priradený k jednému z týchto centier a je mu priradená vzdialenosť k tomuto centru. Jadro problému teda spočíva v nájdení presne *k* centier. *Vzdialenosťou* tu rozumieme dištančnú funkciu metrického priestoru, odkiaľ záznamy pochádzajú.

Rozdiel medzi hľadaním *k*-mediánov a *k*-centier je, že pri hľadaní *k*-centier je minimalizovaná najväčšia vzdialenosť bodu od najbližšieho centra, zatiaľčo pri hľadaní *k*-mediánov je minimalizovaná suma vzdialeností. K-centrá majú problémy pri osamelých záznamoch mimo zhluky.

Definícia 1. **k-Medián problém**

Nech je daná množina *S* prvkov metrického priestoru M s metrikou d. Nech je navyše dané číslo k, ktoré udáva počet hľadaných mediánov. Úlohou je nájsť množinu *C*, obsahujúcu k prvkov $c_1,..c_k$ z množiny *S* takú, aby funkcia

$f(S, c_1, \ldots, c_k) = \sum_{x \in S} \min_{1 \leq i \leq k} d(x, c_i)$ bola minimalizovaná.

V práci [17] je predstavený jednopriechodový algoritmus pre riešenie problému k-centier v čase *O(nk logk)* a *O(k)* priestore. Pre problém k-medián v čase *O(nk)* a $O(n^a)$ pre $0 < a < 1$.

2.6.2 Návrh algoritmu Small-Space

Algoritmy pracujúce s dátovými prúdmi nemôžu mať vysoké požiadavky na veľkosť pamäte. Takže je dôležité ukázať, že zhlukovanie môže byť implementované s využitím malého priestoru $O(n^e)$ pre n záznamov a $0 < e < 1$. Najprv budú preskúmané algoritmy, ktoré využívajú metódu postupného spracovania. Špeciálne sa jedná o metódu rozdeľuj a panuj, a algoritmus small-space, ktorý rozdelí dáta na disjunktné časti, vyrobí zhluky pre tieto časti (centrá) a potom znovu vyrobí zhluky pre centrá, ktoré získal (pre každé centrum je uchovávaná váha, ktorá vyjadruje počet záznamov priradených tomuto centru). Algoritmus je prevzatý z [16].

```
Algoritmus Small-Space(S)
```

1. Rozdeľ S na g disjunktných častí $X_1,...X_l$
2. Pre každé i, nájdi O(k) centier v X_i. Priraď každý bod v X_i k svojmu najbližšiemu centru.
3. Nech X´ je O(gk) centier získaných v 2., kde každé centrum je vážené počtom bodov, ktoré sú k nemu priradené.
4. Použi algoritmus zhlukovania pre X´ a nájdi k centier.

Požadujeme, aby sme zhlukovali do priestorov, ktoré budú menšie ako vyhradený priestor v operačnej pamäti. Preto by sme mali voliť *g* opatrne, tak aby sa aj *S* aj *X´* zmestilo do hlavnej pamäte. Ak bude *S* príliš veľké, žiadne *g* neexistuje – tento problém bude vyriešený neskôr.

Definícia 2: **aproximačný algoritmus s konštantným faktorom**

Problém v triede NP má aproximačný algoritmus s konštantným faktorom, ak existuje

efektívny polynomiálny algoritmus, ktorý nájde riešenie v rámci nejakej konštantnej odchýlky od optimálneho riešenia. Aproximačný algoritmus nazveme *c-aproximačný* pre konštantu c, ak je možné dokázať, že riešenie, ktoré nájde tento algoritmus je najviac *c-krát* horšie ako optimálne riešenie. Konštantu c nazývame *aproximačný faktor.*

V [16] je dokázané, že algoritmus Small-Space má aproximačný faktor 2c(1+2b) + 2b, kde v kroku 2 využijeme b-aproximačný algoritmus a c-aproximačný algoritmus v kroku 4.

2.6.3 Algoritmus Smaller-Space

Bolo povedané, že algoritmus Small-Space dokáže produkovať prijateľné výsledky v polynomiálnom čase. Zovšeobecnením tohto algoritmu o rekurzívne volanie na čoraz menšiu množinu vážených centier je možné získať algoritmus Smaller-Space. Konštanta *i* vyjadruje počet rekurzívnych volaní.

```
Algoritmus Smaller-Space(S,i)
```

1. Ak i<1 skonči
2. Rozdeľ S na l disjunktných častí X_1,..X_l
3. Pre každé h z 1,...,l nájdi O(k) centier v X_h. Priraď každý bod v X_h k svojmu najbližšiemu centru.
4. Nech X´ je množina O(lk) centier získaných v kroku 3, kde každé centrum c je vážené počtom bodov, ktoré je mu priradené.
5. Zavolaj Smaller-Space(X´,i-1)

Pre konštantu *i* je algoritmus Smaller-Space(S,i) aproximačným algoritmom s konštantným faktorom pre k-Medián problém. Dôkaz tohto tvrdenia je možné nájsť v literatúre [16].

Keďže X´ sa musí uložiť do pamäte, nastáva problém s volením čísla l. V prípade, že nie je možné nájsť také l, je nutné vytvoriť hierarchickú schému rozumnejšie a tým získame algoritmus pre dátové prúdy.

2.6.4 Zhlukovanie pre dátové prúdy

V modeli dátového prúdu, je celková výpočtová kapacita obmedzená veľkosťou pamäte M a dáta môžu byť prechádzané iba jedenkrát. V tejto časti bude predstavený algoritmus, ktorý pracuje s prúdovými dátami. Tento algoritmus je jednopriechodový a O(1)-aproximačný. Algoritmus predpokladá, že M nie je príliš malá, konkrétne veľkosti n^e, kde $0 < e < 1$ a n je veľkosť prúdu.

Algoritmus zhlukovania pre dátové prúdy:

1. Zo vstupu načítaj m hodnôt (bodov). Použi k-aproximačný algoritmus popísaný vyššie na zredukovanie na O(k) bodov. Tento algoritmus pracuje v čase $O(m^2)$.
2. Opakuj bod 1, pokiaľ nie je načítaných $m^2 / (2k)$ originálnych dátových bodov. V tomto momente existuje presne *m* mediánov úrovne 1.
3. Využi zhlukovanie pre m mediánov úrovne 1 na získanie 2k mediánov úrovne 2.
4. Opakuj bod 3, všeobecne – z najviac *m* mediánov úrovne i generuj 2k mediánov úrovne i+1, s váhami vypočítanými ako suma váh mediánov, ktoré boli priradené novému mediánu.
5. Po načítaní celého vstupu, zhlukni všetky mediány najvyššej úrovne na k výsledných mediánov.

Počet úrovní potrebných pre tento algoritmus je najviac $O(\frac{(\log(\frac{n}{m}))}{(\log(\frac{m}{k}))})$. Za predpokladu k<<m a $m=O(n^e)$ pre $e < 1$, je tento algoritmus O(1)-aproximačný. Malé m je možné voliť ako √M.

Riešenie k-medián problému na prúde dát je možný v čase $O(n^{1+e})$ a priestore $O(n^e)$ s aproximačným faktorom najviac 2O(1/e). Dôkaz opäť k nájdeniu v [16].

3 Zhlukovanie v kombinácií s posuvným oknom

Na základe všeobecného modelu popísaného v sekciách 2.1, 2.2 a 2.3 bol vytvorený prototyp systému SŘPD, ktorý spracováva prúdy dát a vyhodnocuje dotazy nad nimi. Keďže úloha vytvorenia komplexného SŘPD je veľmi rozsiahla, boli vytvorené určité obmedzenia na spracovávané prúdy ako aj na dotazy nad nimi. Prototyp nemá za úlohu verne simulovať celú funkčnosť systému SŘPD, ale jeho úlohou je ukázať nevyužité možnosti zhlukovania v kombinácií s posuvným oknom pre získanie presnejších dát pre agregačné dotazy. Ako architektonický základ pre prototyp bol použitý model z obrázku 2.1, ktorý sa skladá z viacerých modulov. Ako programovací jazyk bola zvolená Java.

Z teoretickej časti boli vybrané prístupy zhlukovania, posuvné okno a histogramy. Tieto tri komponenty ešte neboli spolu v praxi odskúšané, pričom sa ich kombinácia by mohla priniesť zvýšenú presnosť a relevantnosť odpovedí na dlhotrvajúce dotazy.

V kapitole 4 je teoreticky vysvetlený princíp zhlukovania a navrhnutý spôsob jeho riešenia. S problémom zhlukovania je úzko spojený SWKM[1] problém, ktorý je definovaný ďalej. Po vyriešení problému SWKM je jednoduchšie riešiť problém zhlukovania pre situáciu s posuvnými oknami. Výhoda zhlukovania je v tom, že prvky, ktoré zhlukovanie vyberie (mediány) sú ideálni reprezentanti pre agregačné dotazy. Je vysoko pravdepodobné, že odpoveď na tradičné agregačné dotazy bude veľmi blízko odpovedi, ktorá by bola generovaná z celého prúdu. Samozrejme je nutné zohľadniť veľkosť okna a čas t, v ktorom bol dotaz vyhodnocovaný. Je zrejmé, že ak bol dotaz vyhodnocovaný v malom čase t (nebolo načítané ani cca 10 % prúdových dát), potom záleží na rozmiestnení dátových hodnôt v spektre a od odchýlky prúdu.

V práci [18] je navrhnutý algoritmus založený na algoritme [16] popísanom v odstavci 2.6.4 Zhlukovanie pre dátové prúdy, ktorý naviac využíva histogramy na ukladanie informácií o situácií v posuvnom okne. V tejto práci je však na zhlukovanie použitý algoritmus [19] s využitím [20] ako podprocedúrou. Celkový algoritmus nebol ešte v praxi implementovaný a otestovaný na experimentoch. Cieľom tejto práce je implementovať spomínaný algoritmus, otestovať ho na experimentoch, vyhodnotiť ich a navrhnúť možnosti zlepšenia.

1 Semantic Web Knowledge Middleware

Pre ďalšie uvažovanie nech je hodnota záznamu, pre zjednodušenie, iba typu číslo. Po definovaní správnych metrík je možné rozšíriť tento model o ďalšie typy. Navrhnutá architektúra SŘPD dovoľuje implementovať ďalšie metódy, ktoré môžu určovať metriku pre dátové typy, napr. reťazec, dátum, ...

Definícia 3: **Odchýlka**

Nech je daný prúd dát, v každom okamžiku definujme odchýlku posledných N záznamov ako $VAR = \sum_{i=1}^{N} (x_i - u)^2, kde\ u = \frac{1}{N} \sum_{i=1}^{N} x_i$.

Definícia 4: **SWKM problém**

Nech je daný prúd dátových bodov z metrického priestoru M s dištančnou funkciou l, veľkosť okna N a parameter k. Problém SWKM je nájsť množinu mediánov $C(x) = \{c_1, c_2, ..., c_k\} \in M$, *pre ktoré je* $\sum_{x \in X_t} l(x, C(x))$ *minimálna.* X_t *je množina N* najposlednejších bodov v čase t.

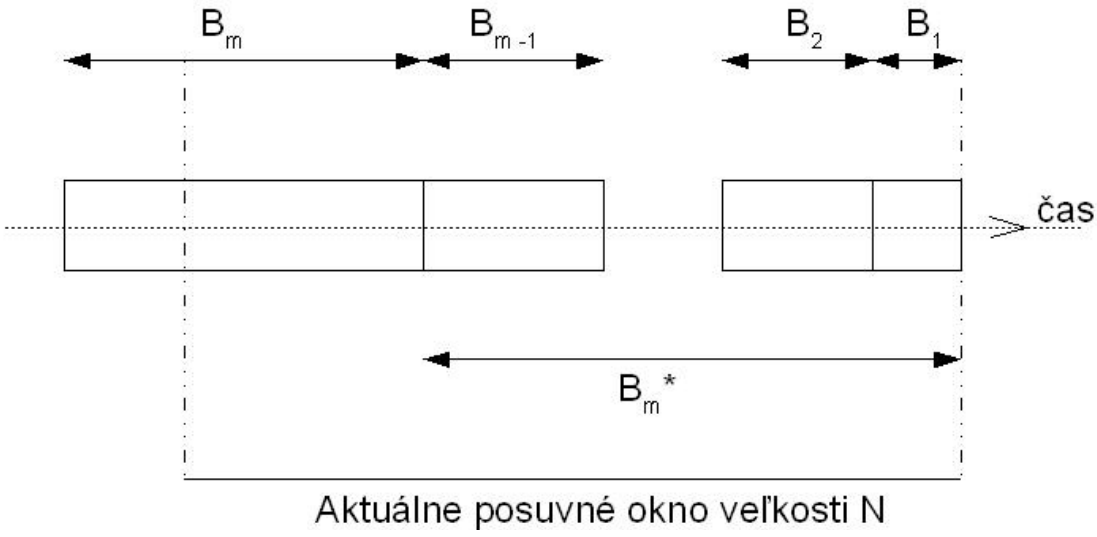
V práci [16] je navrhnutý algoritmus zhlukovania na prúde dát (nie pre posuvné okno). Algoritmus spracúva mediány rôznych úrovní. Originálne vstupné hodnoty označí ako mediány úrovne 0. Po zhliadnutí N^t mediánov úrovne 0 (t je volená konštanta, ktorá určuje veľkosť jednej úrovne), ich zhlukuje s použitím aproximačného algoritmu na O(k) centier, ktoré predstavujú mediány úrovne 1. Akonáhle je naakumulovaných N^t mediánov úrovne 1, sú opäť zhlukované na mediány úrovne 2 atď. Všeobecne, ak je naakumulovaných N^t mediánov úrovne i, sú zhlukované na mediány úrovne (i+1). Je teda zrejmé, že v ľubovolnom momente existuje na každej úrovni najviac N^t mediánov. Keďže sú zhlukované skupiny o veľkosti N^t, je zrejmé že strom vytvorený po viacnásobnom zhlukovaní bude mať hĺbku nanajvýš 1/t, kde N je počet originálnych vstupných bodov (hodnôt).

Algoritmus hľadania k-mediánov pre situáciu pre posuvné okná využíva algoritmus popísaný v odstavci vyššie. Základná myšlienka algoritmu je, že posuvné okno je rozdelené do priehradiek (bucketov) a o každej priehradke sú vedené popisné informácie. Mediány sa vyhľadávajú lokálne pre každú priehradku zvlášť a v niektorých prípadoch sa dve priehradky pomocou špeciálneho algoritmu zlejú do jednej. Ďalej budú popísané dátové štruktúry, čiastkové podprocedúry a nakoniec bude predstavený záverečný algoritmus.

3.1 Dátová štruktúra

Vstupný monitor prototypu SŘPD načítava prvky prúdu v takom poradí, ako prichádzajú. Procesor dotazov potom tieto prvky spracúva a ukladá ich do operačnej pamäte do špeciálnej dátovej štruktúry. Pre tieto účely je uchovávaná histogram pre aktívne dátové elementy z dátového prúdu. Histogram sa skladá z m *priehradiek* (viď 2.5.2). Pre každú priehradku sa vedú súhrnné informácie. Dátové elementy sú do priehradiek ukladané podľa časového razítka (v poradí v akom prichádzajú). Pre každú priehradku je ukladaná informácia (časové razítko) o najnovšom zázname v tejto priehradke. Akonáhle časové razítko dosiahne hodnoty N + 1 (N je veľkosť posuvného okna), sú všetky hodnoty v tejto priehradke expirované a celú priehradku je možné vysypať a uvoľniť pamäť. Priehradky sú číslované B_1, …, B_m, začínajúc od najnovšieho B_1, po najstarší B_m. Hodnoty t_1, …,t_m vyjadrujú časové razítka jednotlivých priehradiek. Je možné, že priehradka popri aktívnych elementoch obsahuje aj elementy, ktoré sú expirované. Táto skutočnosť závisí na algoritme vkladania záznamov do priehradiek.

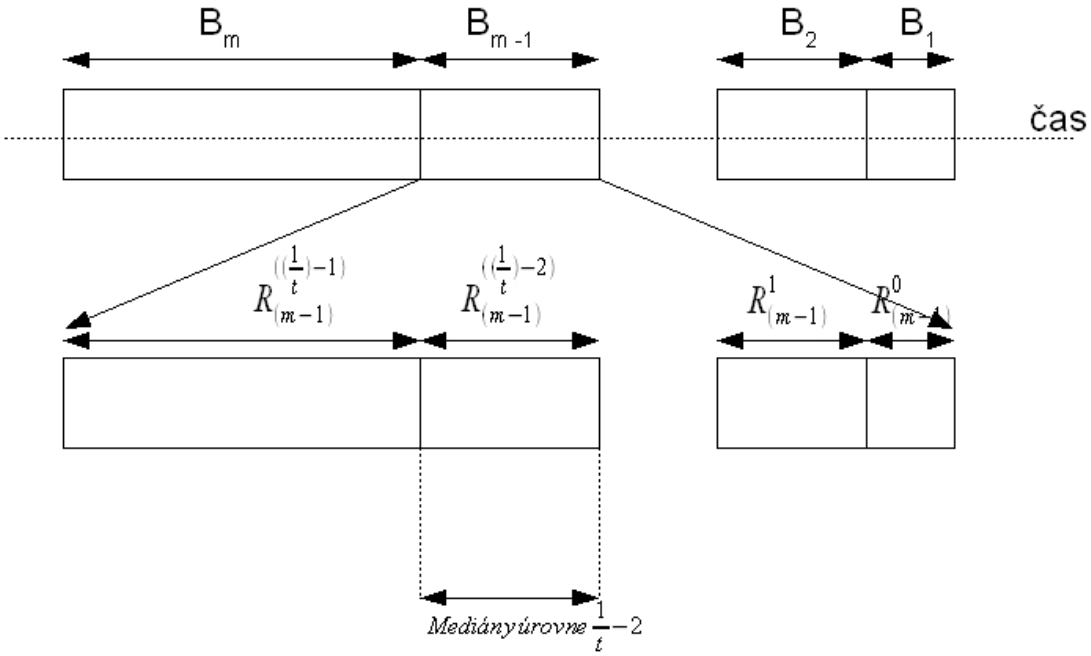
Alternatívne sú definované priehradky B_1*, …, B_j*, ktoré definujú „prípony“ dátového prúdu. Priehradka B_i* obsahuje všetky záznamy, ktoré sú novšie ako časové razítko priehradky B_i. Formálne zapísané : $B_i^* = \bigcup_{l=1}^{i-1} B_l$. Výnimku tvorí priehradka *Bm**, ktorá nie je udržovaná, pretože je vytváraná dočasne v priebehu algoritmu.



Obr 3.1: Ilustrácia histogramu s použitím priehradiek

Priehradka obsahuje časové razítko najnovšieho záznamu, ktorý obsahuje a zoznam mediánov (originálne dátové body sú mediány úrovne 0). Každý medián je reprezentovaný trojicou (p(x), w(x), c(x)). Hodnota p(x) je identifikátor hodnoty x. Môže sa jednať o koordináty v Euklidovskom priestore, ale v praxi sa používa časové razítko ako jednoznačný identifikátor. Hodnota w(x) je váha mediánu, t.j. počet bodov v priestore, ktoré tento medián reprezentuje. Podobne ako v [16], ak je x úrovne 0 potom w(x) = 1, Ak je x úrovne i, potom w(x) je suma váh mediánov úrovne (i-1), ktoré boli priradené x po prebehnutí zhlukovania úrovne i. Hodnota c(x) je odhadovaná cena(x), t.j. odhadovaná suma cien l(x,y). Ak je x úrovne 0, potom c(x) = 0. Ak je x úrovne 1, potom c(x) = suma vzdialeností bodov priradených x. Ak je úroveň x > 1, potom c(x) je suma cez všetky priradené body y k x, c(y) + w(y).l(x,y).

Keďže kedykoľvek je N^t mediánov jednej úrovne naakumulovaných, sú zhlukované, je aj každá priehradka B_i rozdelená na 1/t skupín R_i^0, …, $R_i^{1/t-1}$, kde R_i^j obsahuje mediány úrovne j patriace do priehradky B_i. Priehradka obsahuje navyše ešte odhadovanú cenu priehradky, ktorá je vysvetlená nižšie.



Obr 3.2: Ilustrácia histogramu priehradiek spolu s úrovňami mediánov

3.2 Výpočet ceny priehradky

Pre každú B_i nech $Y_i = \bigcup_{j=0}^{1/t-1} R_i^j$ je množina mediánov v priehradke. Každý z mediánov má priradenú hodnotu c(x), tak ako je popísané vyššie. Po prevedení zhlukovania Y_i na k mediánov $c_1, \ldots, c_k$ je možné definovať cenu priehradky ako:

$$f(B_i) = \sum_{x \in Y_i} c(x) + w(x).l(x, C(x)),\ kde\ C(x) \in \{c_1, \ldots, c_k\}\ je\ medián\ najbližší\ k\ x.$$

3.3 Zlučovanie priehradiek

Nech B_i a B_j sú dve (zvyčajne susedné) priehradky, ktoré sa majú zlúčiť do jednej výslednej priehradky $B_{i,j}$ a nech $R_i^0, \ldots, R_i^{1/t-1}$ a $R_j^0, \ldots, R_j^{1/t-1}$ sú skupiny mediánov v týchto priehradkách. Potom $R_{i,j}^0 = R_i^0 \cup R_j^0$. Ak $|R_{i,j}^0| > N^t$, potom je nutné previesť zhlukovanie pre hodnoty z $R_{i,j}^0$ a nastaviť $R_{i,j}^0$ ako prázdnu. Nech C_0 označuje množinu O(k) mediánov získaných zo zhlukovania $R_{i,j}^0$, alebo prázdnu množinu, ak zhlukovanie nebolo potrebné. Potom $R_{i,j}^1 = R_i^1 \cup R_j^1 \cup C_0$. Tak ako predtým, ak $R_{i,j}^1$ presiahne hodnotu N^t, je nutné zhlukovanie a pomocou neho je získaná množina C_1 s ktorou je nakladané analogicky ako s C_0. Tento postup je opakovaný pre všetky skupiny $R_i^0, \ldots, R_i^{1/t-1}$ a $R_j^0, \ldots, R_j^{1/t-1}$.

3.4 Odhad zhlukovania

Ak je dotaz položený v čase t, je nutné previesť zhlukovanie nad aktuálnymi elementami. Na odhad zhlukovania je navrhnutý nasledujúci algoritmus:

1. Z priehradiek $B_1, \ldots, B_{m-1}$ (každá obsahuje najviac $\frac{1}{t} N^t$ mediánov) vyber všetky mediány a zhlukovaním získaj k mediánov.
2. Z B_m získaj zhlukovaním ďalších k mediánov
3. *Return* 2k mediánov ako výsledok

3.5 Algoritmus vkladania nového prvku

```
    x_t := aktuálne prijatá hodnota z dátového prúdu s časovým
razítkom t
    if   B_1 obsahuje menej ako k mediánov úrovne 0 then
         pridaj x_t do B_1 ako medián úrovne 0
    else
         vytvor novú priehradku B_1, ktorá bude obsahovať x_t a
         prečísluj ostatné priehradky.
    if priehradka B_m má časové razítko vyššie ako N then
         zmaž priehradku B_m.
    for every B_i , i > 2 do
         if f(B_i, i-1) ≤ 2f(B_i-1*) then    // cena priehradiek, viď
                                        3.2 Výpočet ceny priehradky
              zlúč_priehradky(B_i, B_i-1) // zlúčenie viď 3.3
                                          Zlučovanie priehradiek
              break for
         endif
    endfor
```

4 Architektúra vlastného SŘPD

Prototyp systému SŘPD je možné rozdeliť na dva logické celky, vstupnú časť a exekučnú časť. Pri štarte systému sú obidve časti spustené súčasne a nezávisle na sebe v separátnych vláknach.

4.1 Vstupná časť

Vstupná časť reprezentuje podľa architektúry 2.1, z ktorej bolo vychádzané, časti monitor vstupu, pracovnú pamäť, pamäť pre súhrnné dáta a statickú pamäť. Úlohou vstupnej časti je načítavať zaregistrované prúdy dát a udržiavať synopsie a iné metadáta pre účely neskoršieho spracovania.

4.1.1 Štruktúra vstupnej časti

Vstupná časť je, ako už bolo spomínané, spúšťaná v separátnom vlákne. V prvom rade je dôležité, aby si monitor vstupu zistil všetky zaregistrované prúdy dát, ktoré je nutné spracovávať. Na tieto účely slúži statická pamäť. S jej pomocou si monitor vstupu načíta všetky prúdy dát, ktoré je nutné spracovať. Tieto prúdy dát musia byť najprv zaregistrované.

Registrovanie prúdu dát je možné pomocou vloženia riadku do konfiguračného súboru. Formát tohto súboru je demonštrovaný ukážkou.

```
Streams.txt

Test1 dat//stream1.dat Attr1,Int,1
Test2 dat//stream2.dat Attr1,Int,1
Test3 dat//stream3.dat Attr1,Int,2
Test4 dat//stream4.dat Attr1,Int,0
```

Každý riadok reprezentuje jeden prúd dát. Prvá položka je názov prúdu (napr. Test1). Druhou položkou je umiestnenie prúdu. Umiestnenie môže byť napríklad veľký súbor dát, ktorý je nepretržite čítaný. Zvyšné položky sú definície atribútov v prúde dát. Definícia atribútu je trojica (položky v rámci trojice sú oddelené čiarkou) - názov atribútu, typ atribútu, príznak zhlukovania. Príznak zhlukovania môže nadobúdať tri hodnoty 0, 1 a 2 (podľa typu algoritmu viď 5.2 Zvolenie testovaných algoritmov).

Načítanie informácií o registrovaných prúdoch má na starosti statická pamäť. V riešení prototypu je predstavovaná triedou *StaticStore.*

Trieda *StaticStore* je implementovaná podľa návrhového vzoru Singleton. Existuje jediná inštancia tejto triedy po celú dobu trvania programu. Prúd dát je načítaný do štruktúry tvorenej triedou *Stream.* Ide o obalovú triedu, ktorá zabezpečuje manipuláciu s prúdom dát.

Pre každý prúd dát musí byť definované jedinečné meno, umiestnenie a zoznam atribútov. Zoznam atribútov je implementovaný ako hashmapa, ktorá mapuje názov atribútu na triedu *Attribute*. Táto trieda obaľuje atribút potrebnými informáciami.

Trieda *Stream* obsahuje okrem iného aj objekt typu *SlidingWindow*, ktorý reprezentuje posuvné okno pre tento prúd.

4.1.2 Sliding window

Pre správne fungovanie protoypu SŘPD je kľúčová trieda *SlidingWindow*, ktorá zabezpečuje správne ukladanie príchodzích dát do štruktúry posuvného okna. Táto trieda využíva techniky, ktoré sú popísané v teoretickej časti, viď 3.1 Dátová štruktúra. Pre ukladanie atribútov vyžadujúcich zhlukovanie je v *SlidingWindow* implementovaný histogram s využitím priehradiek. Pre uchovávanie priehradiek je vytvorená trieda *Bucket*.

Táto trieda v sebe obsahuje aj zoznam objektov typu triedy *Median*. Táto trieda slúži na zapúzdrenie informácií o mediáne. Okrem štandartných položiek obsahuje trieda Median aj odkaz na iný objekt typu Median, ktorý reprezentuje jeho centrum. Je dobré si uvedomiť, že aj vstupné záznamy sú typu Median. Konkrétne ide o Median úrovne 0. Podobne má každý medián odkaz na všetky objekty Median, pre ktoré je on sám centrom. Tieto informácie sú udržiavané kvôli zvýšeniu rýchlosti algoritmu.

Trieda *SlidingWindow* obsahuje kľúčovú metódu *getNext(),* pomocou ktorej monitor vstupu vkladá do posuvného okna nové záznamy. Samotná metóda *getNext()* sama rozpozná, či sa jedná o atribút vyžadujúci zhlukovanie alebo o obyčajný atribút. Podľa toho zvolí prístup ukladania záznamu do vnútornej pamäte.

V prípade, že sa jedná o obyčajný atribút, je záznam vložený do dátovej štruktúry pre obyčajné atribúty – fronty. Veľkosť fronty je pritom obmedzená veľkosťou posuvného okna. Táto veľkosť je pevná a je daná konštantou v triede *SlidingWindow.*

Na druhú stranu, ak je spracovávaný atribút s príznakom zhlukovania, je vyvolaná metóda *insert()*, ktorá simuluje správanie tak, ako je popísanie v sekcií 3.5 Algoritmus vkladania nového prvku. Pri vykonávanie tejto metódy dochádza k využitiu zhlukovania. Dochádza k nemu práve tak, ako je popísane vyššie v teoretickej časti 3.1 Dátová štruktúra. Rutiny zhlukovania sú implementované pomocou triedy *Clustering*.

4.1.3 Činnosť vstupnej časti

Po spustení vstupnej časti si monitor vstupu načíta všetky zaregistrované prúdy dát, ktoré je nutné spracovať, tak ako je popísané v predchádzajúcich kapitolách. Pre každý z týchto prúdov vytvorí zvláštne vlákno, ktoré sa bude starať o svoj prúd dát. Pre tieto účely slúži trieda *ExecuteStream.* Táto trieda si vezme za svoj vstup prúd dát a spracúva ho, pokiaľ nie je ukončený beh systému alebo je ukončený prúd dát.

Trieda *ExecuteStream* v nekonečnej slučke číta riadky súboru, ktorý je uvedený ako zdroj prúdu dát, ktorý sa má spracovávať. Každý riadok je vnímaný ako nový záznam. Atribúty záznamu musia byť v súbore uvedené v rovnakom poradí ako sú uvedené v registračnom súbore streams.txt. Pre každý takýto riadok je vyvolaná metóda *getNext*() pre triedu *SlidingWindow* príslušného prúdu dát.

4.2 Exekučná časť

Druhou logickou časťou systému je exekučná časť. Táto časť je spúšťaná v samostatnom

vlákne. Podľa abstraktného modelu SŘPD obsahuje exekučná časť moduly procesor, prúdový výstup, statickú pamäť a synopsie (pracovnú pamäť).

4.2.1 Statická pamäť

V statickej pamäti sú uložené informácie, ktoré je nutné uchovať aj po reštarte systému. Sú to hlavne informácie o zaregistrovaných prúdoch dát a informácie o zaregistrovaných dlhodobých dotazoch. O prácu so statickou pamäťou sa stará trieda StaticStore, ktorá umožňuje čítanie informácií zo statickej pamäte.

Formát, akým je registrovaný prúd dát je popísaný v odstavci 4.1.1 Štruktúra vstupnej časti. Dotazy sú registrované v konfiguračnom súbore. Dotaz je kvôli jednoduchosti obmedzený na možnosť dotazovať sa len na jeden atribút jedného prúdu. Ťažisko práce je venované problému zhlukovania a preto je umožnené v dotaze využiť agregačný operátor. Spomedzi agregačných operátorov bol vybratý operátor „priemer“. Zhlukovanie vyberá z dátovej množiny množinu reprezentantov a pri operátore priemeru je najmarkantnejší rozdiel pri práci so zhlukovaním a bez neho. Syntax registrovania dotazu je ukázaná na nasledujúcom výňatku z konfiguračného súboru :

```
Attr1 Test
AVG Attr2 Test2
```

Jeden riadok súbor predstavuje jeden dotaz. Dotaz sa skladá z dvoch identifikátorov. Prvý z nich označuje atribút, ktorý požadujeme a druhý identifikuje prúd dát, kde sa atribút nachádza. Ak sa na prvom mieste riadku nachádza kľúčové slovo AVG, bude ako výsledok dotazu vyhodnocovaný priemer dotazovaného atribútu.

4.2.2 Modul procesor

Modul procesor má na starosti správu dlhodobých dotazov. Pracuje v oddelenom vlákne kvôli zvýšenej efektívnosti. Algoritmus pre prácu procesoru je jednoduchý. Najprv si zo statickej pamäte pomocou obslužnej rutiny zistí všetky zaregistrované dlhodobé dotazy. Následne pre každý dotaz vytvorí vlákno, ktoré sa stará o vykonávanie konkrétneho dotazu.

Vo všeobecnosti je vykonávanie dotazu zložitá procedúra, ktorá v sebe zahrňuje okrem

iného stavanie vyhodnocovacieho stromu, optimalizácie operátorov, apod. V tejto práci bol vzhľadom na náročnosť prijatý model, ktorý dovoľuje len dotazy v tvare, ktorý je popísaný v 4.2.1 Statická pamäť. Vyhodnocovanie dotazu prebieha v závislosti na type atribútu – buď sa jedná o atribút vyžadujúci zhlukovanie alebo o obyčajný. Procesor musí rozlíšiť tieto dva prípady a v prípade zhlukovaného atribútu vrátiť odpoveď na základe vybudovaného histogramu, ktorý je popísaný v sekcií 4.1.2 Sliding window.

4.2.3 Prúdový výstup

Výstupom systému sú odpovede na dlhodobé registrované dotazy v čase. Tieto odpovede sa priebežne ukladajú do súboru out.txt.

4.3 Implementácia zhlukovania

Z naštudovanej literatúry sa bolo potrebné rozhodnúť medzi dvomi prístupmi implementácie zhlukovania. Prvý z nich [20] využíva na hľadanie *k* mediánov riešenie facility location problému(viď Definícia 5. Facility location problém (hľadanie stredísk)). Pre túto alternatívu je potrebná znalosť lineárneho programovania, pretože je najprv vyriešená duálna úloha a až potom spätne dopočítané riešenie pôvodnej. Druhý prístup [16] (ďalej len algoritmus 2) vychádza zo základného frameworku zhlukovania Smaller-space(viď 2.6.3 Algoritmus Smaller-Space) a ukazuje jeho použitie pri hľadaní k-Medián.

V prvej fáze práce bola implementovaná prvá alternatíva [20] (ďalej algoritmus 1), preto je dôležité ju podrobne popísať (4.3.1 Algoritmus 1). Až po určitej dobe riešenia tejto práce bol naštudovaný algoritmus 2, ktorý je vylepšením algoritmu 1. Keďže sa jedná o algoritmus, ktorý pracuje rýchlejšie a presnejšie, bol implementovaný algoritmus 2. Všetky metódy uvedené v tejto sekcií sú implementované v triede *Clustering*. Metódy, ktoré sú využívané iba v algoritme 1 sú označené ako „deprecated“.

4.3.1 Algoritmus 1

Algoritmus 1 je aproximačný algoritmus, ktorý rieši problém hľadania stredísk (centier), ako aj k-medián problém. Existuje jednoduchý lokálny hltavý algoritmus pre hľadanie stredísk [20], ktorý je 2,414-aproximačný a pracuje v čase $O(n^2)$. Zároveň je v [20] ukázaný algoritmus pre k-medián problém, ktorý je 4-aproximačný a pracuje v čase $O(n^3)$ a využíva riešenie problému hľadania stredísk.

Pre pokračovanie je nutné definovať problém hľadania stredísk [20].

Definícia 5. **Facility location problém (hľadanie stredísk)**

Nech je daná množina N bodov v metrickom priestore s dištančnou funkciou d: N x N -> R, a parameter z. Pre akúkoľvek množinu $C=\{c_1, c_2, ..., c_k\} \subset N$ k centier, definuj rozdelenie N do k centier N_1, N_2, …, N_k tak, že N_i obsahuje všetky body z N, ktoré sú bližšie k c_i ako ku ktorémukoľvek inému centru. Úlohou je nájsť hodnotu k a množinu C tak, aby bola minimalizovaná stredisková cena FC:

$$FC(N, C) = z|C| + \sum_{i=1}^{k} \sum_{x \in N_i} d(x, c_i)$$

Alternatívna definícia:

Nech je daný graf s metrikou hrán c(), cenu f_i otvorenia jednotlivých centier i, Úlohou je minimalizovať cenu vybraných centier (stredísk - facility) F plus sumu priradzovacích cien každého vrcholu grafu k najbližšiemu centru C. Priradzovacia cena vrcholu j k centru i je $d_j c_{ij}$, kde c_{ij} určuje vzdialenosť medzi i a j. Konštanta d_j určuje požiadavku vrcholu j.

Algoritmus: **Facility location – lokálne hľadanie a hladový algoritmus**

Nech F je stredisková cena a C nech je celková priradzovacia cena riešenia tak ako sú definované v defnícií problému. Úlohou je teda minimalizovať F + C. Algoritmus začne z nejakého začiatočného riešenia a opakovane sa snaží zlepšovať F + C vykonávaním lokálnych zmien.

Hľadanie počiatočného riešenia

Počiatočné riešenie je vybrané takto. Centrá (strediská) zotrieďme podľa rastúcej ceny f_i. To bude stáť O(n log n) času. Následne budeme počítať možné riešenia inkrementačne vzhľadom na f_i. Idea je v tom, že medzi riešením pozostávajucim z i stredísk a i + 1 stredísk je malý rozdiel. Tento rozdiel je možný spočítať v čase O(n).

Uvažujme včlenenie (i + 1) strediska do riešenia zloženého z i stredísk. Pre niektoré vrcholy môže byť výhodnejšie pripojiť sa k i+1 stredisku ako to ku ktorému sú aktuálne pripojené. Prejdeme teda v O(n) všetky vrcholy a otestujeme nové stredisko.

Predchádzajúcu procedúru budeme robiť pre všetky potenciálne strediská, ktorých je O(n). Celkovo teda hľadanie počiatočného riešenia zaberie $O(n^2)$ času.

Nech S je množina stredísk, ktoré boli vybrané do počiatočného riešenia. Uvažujme stredisko i. Pokúsime sa vylepšiť aktuálne riešenie S pridaním vrcholu i a prípadným odobratím niektorých vrcholov z S.

Funkcia zisk (určuje či pridaním vrcholu i do riešenia znížime celkovú cenu)

Niektoré vrcholy môžu byť bližšie k i ako k aktuálne priradenému vrcholu. Všetky takéto vrcholy sú preradené k i. Navyše je možné niektoré stredisko z riešenia vylúčiť. Ak stredisko i´ vylúčime, potom všetky vrcholy, ktoré boli pripojené k i´ sú teraz pripojené k i. Zisk z tejto operácie vyjadruje najvyšší možný pokles F + C. Ak je zisk kladné číslo je i zahrnuté do riešenia, v opačnom prípade je odmietnuté.

Algoritmus vyberá náhodné vrcholy i a opakuje funkciu zisk popísanú vyššie. Po určitom počte opakovaní (v literatúre [20] je odporúčaný O(log N)) je algoritmus ukončený a množina S je predložená ako správny výstup.

Pozorovania

- Počiatočné riešenie je vybrané v čase $O(n^2)$
- Funkcia zisk – ktorá sa opakuje – je vypočítaná v čase O(n)

Horeuvedený algoritmus nerieši priamo problém k-mediánu, ale môže byť využitý ako podprocedúra pri jeho riešení. Nech každé f_i je rovné konštante *z*. Na začiatku zvolíme náhodne *z* a následne ho pomocou polenia intervalov usmerňujeme tak, aby nám algoritmus vydal presne *k* stredísk. Pre každú hodnotu *z* teda voláme algoritmus 1.

4.3.2 Algoritmus 2

Algoritmus 2 vychádza z poznatkov Algoritmu 1, ale rieši problém k-Mediánu trochu inak. V prvom rade vylepšuje hľadanie počiatočného riešenia.

Algoritmus **Počiatočného riešenia**:

vstup: množina bodov N (x_1,..,x_n), stredisková cena z

výstup: množina stredísk S (s_1,..s_q) // q je spočítané v algoritme

```
1. Náhodne prerovnaj prvky N
2. s1 := x1  // prvý bod v poradí zvoľ ako centrum
3. for každý ďalší bod xi do
      d := vzdialenosť bodu xi od najbližšieho centra z S
      if  pravdepodobnosť d / z then
          sk := xi // vytvor z bodu xi nové centrum
      else pridaj xi do poľa pôsobnosti jeho najbližšieho
           centra sj
4. endfor
5. return S
```

Oproti algoritmu 1 postačuje tomuto algoritmu len 1 priechod oproti log n priechodom algoritmu 1.

Plný algoritmus 2 taktiež využíva riešenie facility location ako svoju subrutinu. Riešenie facility location pre algoritmus 2.

Algoritmus FL

vstup:

- N – množina bodov,
- d(.,.) - dištančná funkcia,
- z – stredisková cena,
- e – parameter konvergencie,
- (I, a) – počiatočné riešenie, *I* je množina stredísk a *a* je je priraďovacia funkcia N -> I

výstup :

- riešenie (I`, a`)

```
1. (I`, a`) := (I, a)
2. C := cena aktuálneho riešenia pre N
3. M := množina náhodných bodov z N ktoré nie sú centrá
4. for každý bod y z M do
5. if gain(y) > 0 then sprav všetky potrebné zmeny v riešení
   (I`,a`) (gain funkcia je totožná s funkciou zisk z
   algoritmu 1)
6. endfor
7. C´ := cena riešenia (I`, a`)
8. if C´ < (1 – e)C then krok 2
9. else return (I`, a`)
```

Plný algoritmus pre hľadanie k mediánov v metrickom priestore. Využíva oba vyššie uvedené algoritmy ako subrutiny.

Algoritmus 2:

vstup:

- N – množina bodov
- d(.,.) - dištančná fcia
- k – počet mediánov
- e - parameter konvergencie

výstup :

- F – množina mediánov

1. z_{min} := 0
2. z_{max} := $\sum_{x \in N} d(x, x_0)$, x_0 je ľubovolný bod z N
3. z := (z_{max} + z_{min}) / 2
4. (I, a) := Počiatočné_riešenie(N, z)
5. **while** #mediánov <> k **a** z_{min} < (1 – e^n)z_{max}
6. (F,g) := aktuálne riešenie
7. (F`,g`) := algortimus FL (N, d, e, (F, g))

```
8. if (k <= |F`| <= 2k) then endwhile
9. if |F`| > 2k then z_min := z a z := (z_max + z_min) / 2
10.else if |F`| < k then z_max = z a z = (z_max + z_min) / 2
11.loop
12.return F´
```

Počiatočná hodnota z_{max} je vybraná ako triviálny horný odhad (suma priradzovacích cien) hodnoty z. Algoritmus 2 pracuje v čase O(nm + nklog(k)), kde m je počet centier vybraných počíatočným riešením. Jedná sa teda o zrejmé vylepšenie algoritmu 1.

5 Spôsob testovania

Úlohou zhodnotenia experimentálnych výsledkov je ukázať, že algoritmus spracovávania prvkov prúdu dát pomocou zhlukovania je presnejší ako doteraz používaná štandardná technika.

V kapitolách 5 a 6 sa nachádzajú grafy meraní. Väčšina grafov má rovnaké osi. Vodorovná os (x-ová) znázorňuje počet načítaných záznamov a zvislá (y-ová) os znázorňuje hodnotu záznamu – číslo od 0 do 100 000. Osi sú popísané pri každom grafe osobitne.

Testovanie prebiehalo na počítači osadenom procesorom Pentium IV, 2,6 Ghz. Testované boli tri algoritmy spracovania dát v modeli SŘPD. Tieto algoritmy boli testované na troch rôznych testovacích sadách. Výber testovacích sád, ako aj testovaných algoritmov, je zdôvodnený v sekciách 5.1 Zvolenie testovacích dát a 5.2 Zvolenie testovaných algoritmov.

Hlavným testovaným údajom bola presnosť odpovedí v čase na dotaz priemeru. Dotaz priemeru bol vybraný, pretože pri ňom je možné využívať metódu zhlukovania. Samozrejme je možné použiť aj iné agregačné dotazy, ktoré využívajú blokujúce operátory. Je dôležité si uvedomiť, že práve pri priemere je vidieť výhody zhlukovania. Zhlukovanie vytvára množiny mediánov, tým pádom je priemer oveľa presnejší so zhlukovaním, zatiaľčo napríklad pri maxime alebo count(počet záznamov) nebude rozdiel v presnosti tak veľký.

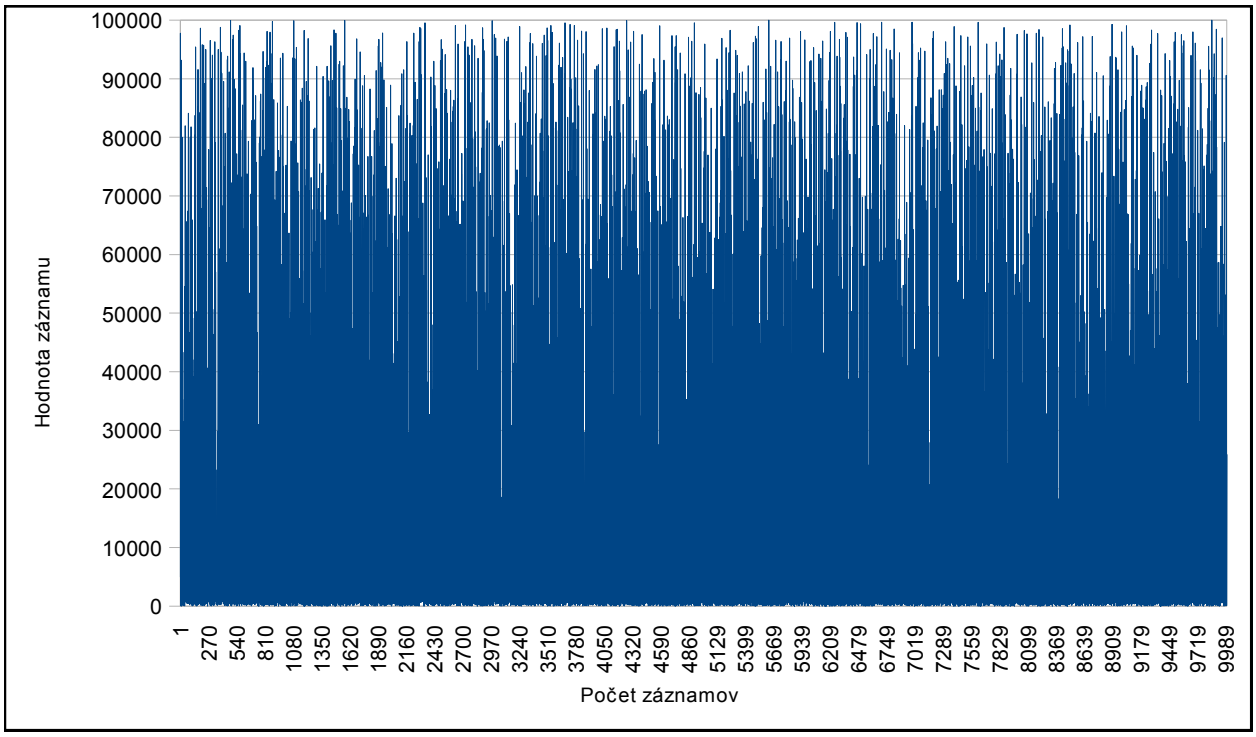
Sekundárnym testovaným atribútom bola rýchlosť odpovedí, ktoré systém produkoval pri zapnutom, resp. vypnutom zhlukovaní.

5.1 Zvolenie testovacích dát

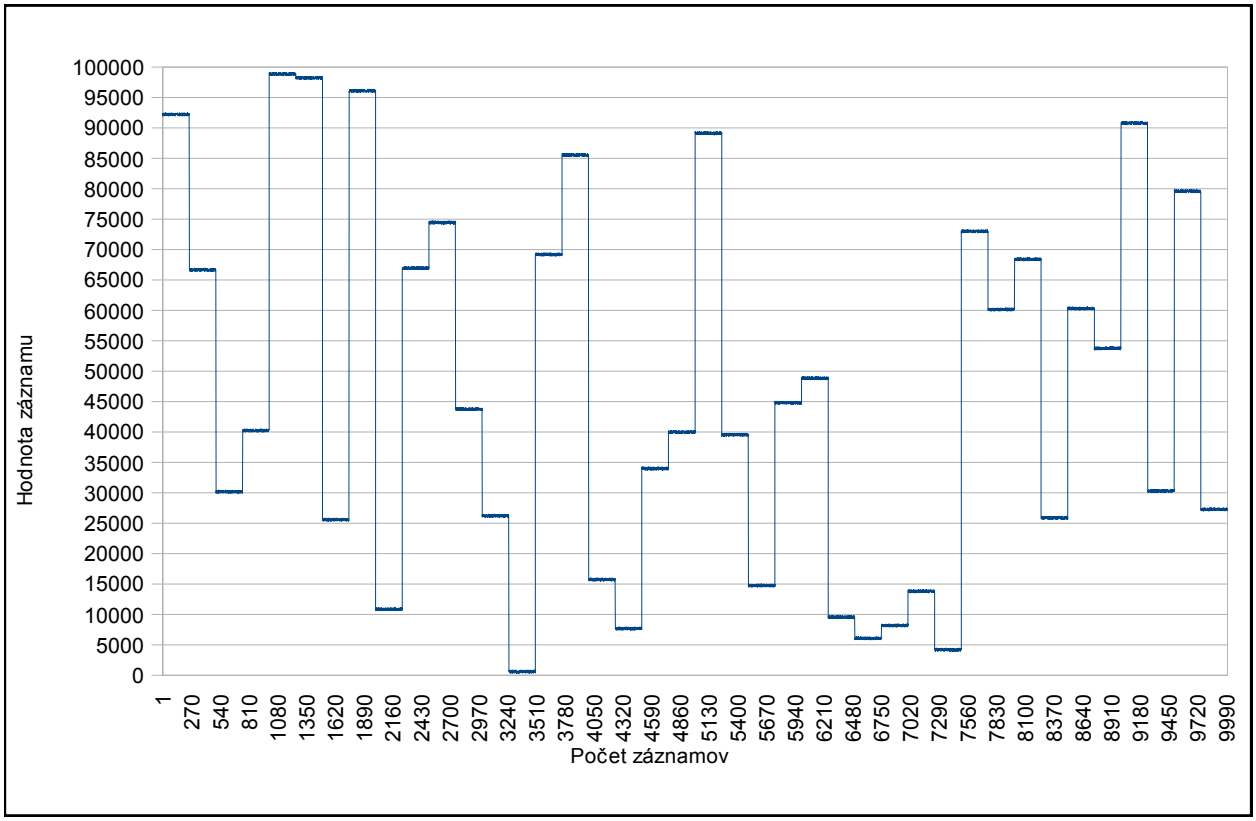
Na testovanie boli použité tri testovacie sady. Každá testovacia sada mala inú povahu, aby boli výsledky merania čo najobjektívnejšie. Všetky testovacie sady obsahovali prúd dát s jediným atribútom. Typom tohto atribútu bolo celé nezáporné číslo v rozmedzí od 0 po 100 000.

Prvá testovacia sada obsahuje prvky pseudonáhodne rozmiestnené v spektre 0 až 100 000, kde pseudonáhodnosť je zabezpečená funkciou random() z jazyka Java.

Graf 5.1: Testovacia sada 1

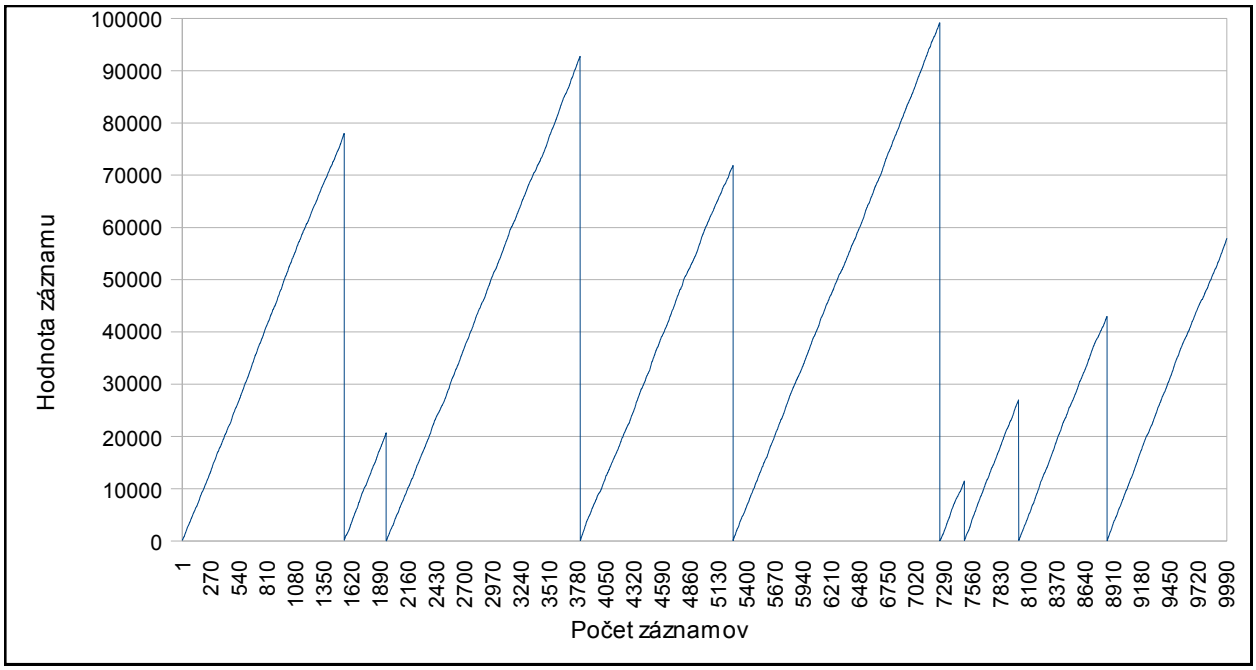


Graf 5.2: Testovacia sada 2



Druhá testovacia sada obsahuje prvky tvoriace dlhšie zhluky v jednej oblasti. Tieto zhluky sa nepravidelne striedajú. Úlohou tejto sady je otestovať, ako si jednotlivé algoritmy poradia s veľa záznamami idúcimi za sebou s približne rovnakou hodnotou.

Graf 5.3: Testovacia sada 3



Tretia testovacia sada obsahuje náhodné vzrastajúce postupnosti, pri ktorých je zložité predikovať priemer.

5.2 Zvolenie testovaných algoritmov

Do experimentov boli zahrnuté tri algoritmy. Každý z nich reprezentuje iný prístup k spracovaniu dátových prúdov. Všetky algoritmy mali k dispozícii rovnaký dátový priestor v operačnej pamäti.

Prvý algoritmus je dnes najčastejšie používaný v doterajších prototypoch SŘPD. Jedná sa o jednoduché použitie posuvného okna, kde sa uchováva posledných N záznamov, kde N je veľkosť posuvného okna. Výhodou tohto algoritmu je bezosporu jednoduchosť naprogramovania a určite aj rýchlosť spracovania prvkov, pretože nie je žiadna réžia udržiavania okrem vymazania najstaršieho prvku. Pri jednotlivých experimentoch bude uvedené, aká veľkosť posuvného okna

bola zvolená. Algoritmus je vo všetkých grafoch označovaný modrou farbou a v legende vystupuje pod názvom „Posuvné okno“.

Druhý algoritmus využíva taktiež metódu posuvného okna, avšak pre udržiavanie relevantnejších prvkov využíva zhlukovanie. Algoritmus pracuje tak, že po načítaný N záznamov spustí algoritmus zhlukovania, ktorý z N záznamov vyberie N / 2 reprezentantov, ktorých uloží namiesto pôvodných N záznamov. Po načítaní zvyšných N / 2 záznamov sa rutina opakuje. Výhodou tohto algoritmu oproti pôvodnému je vyššia relevancia záznamov pri dlhotrvajúcich dotazoch. Cenou za to je vyšší čas potrebný k spracovaniu jedného záznamu. Obzvlášť veľkou nevýhodou je, že algoritmus zhlukovania nepočíta s váhami jednotlivých prvkov. Algoritmus bude ďalej označovaný ako „Posuvné okno so zhlukovaním“.

Tretí algoritmus, ktorý je detailne opisovaný v kapitole 3 a v sekcií 4.3, využíva posuvné okno veľkosti N, nad ktorým je postavený histogram. Tento histogram dobre odráža relevanciu záznamov (v prvých priehradkách sú posledné záznamy) a súčasne rieši problém predchádzajúceho algoritmu s váhami mediánov. Algoritmus bude ďalej označovaný ako „Posuvné okno so zhlukovaním a histogramom“.

6 Výsledky meraní

Výsledky meraní sú rozdelené na tri podčasti. V prvej sa zameriame na sledovanie presnosti, v druhej na sledovanie rýchlosti a v poslednej na sledovanie záťaže.

6.1 Meranie presnosti

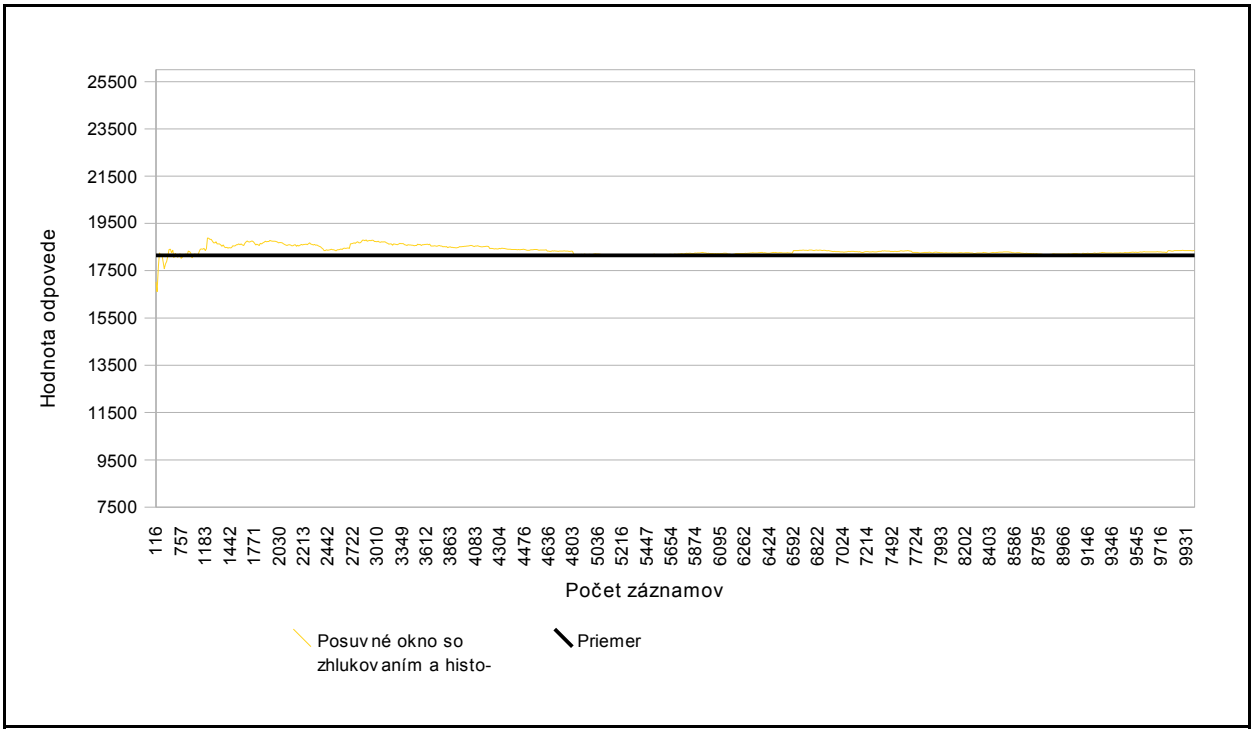
Presnosť je kľúčový atribút pri vyhodnocovaní dlhotrvajúcich dotazov. Aj keď je možné vracať odpovede s určitou odchýlkou, je samozrejmé že čím presnejšia odpoveď, tým je algoritmus považovaný za lepší.

Grafy ukazujú ako jednotlivé algoritmy v čase odpovedali na dotaz priemeru. Všetky tri algoritmy sa snažili predikovať na základe už prečítaných dát celkový priemer dátového prúdu. Postupne bude rozobraté správanie všetkých troch algoritmov vzhľadom ku každej testovacej sade.

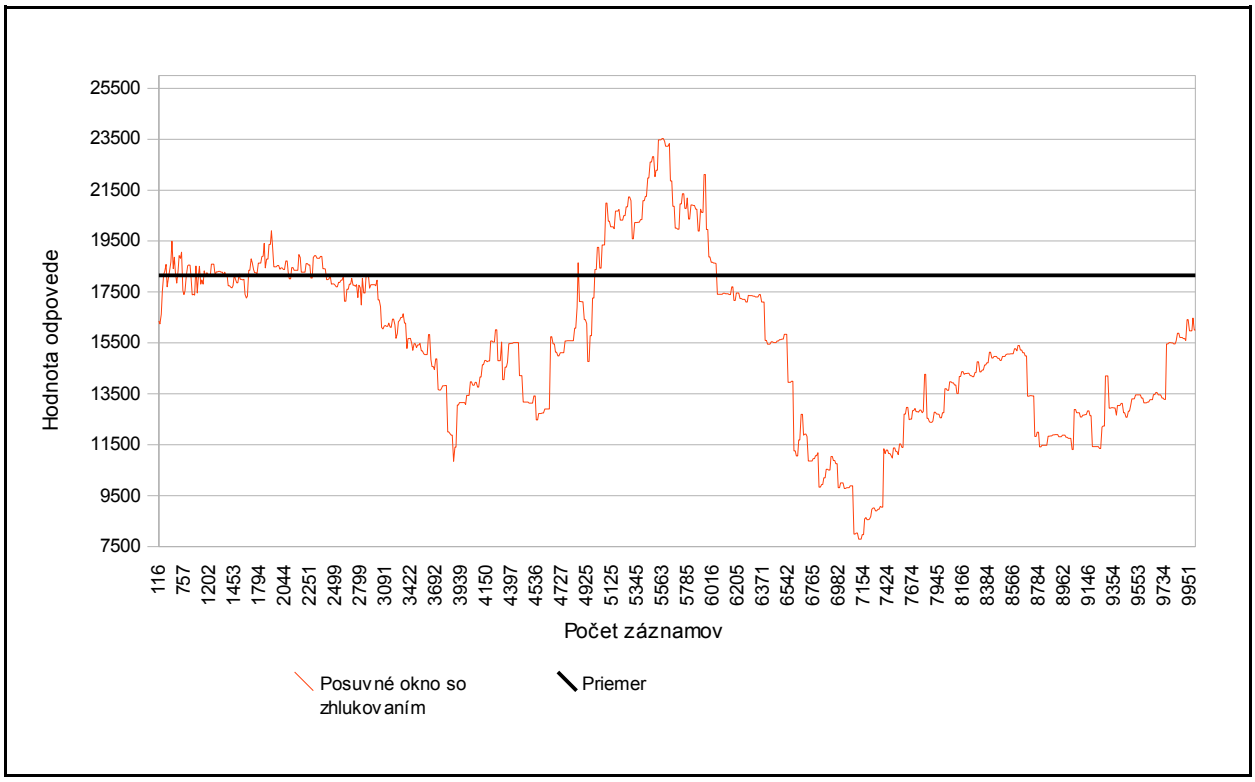
6.1.1 Testovacia sada 1

Výsledky meraní všetkých algoritmov na testovacej sade číslo 1 sú znázornené na nasledujúcich grafoch.

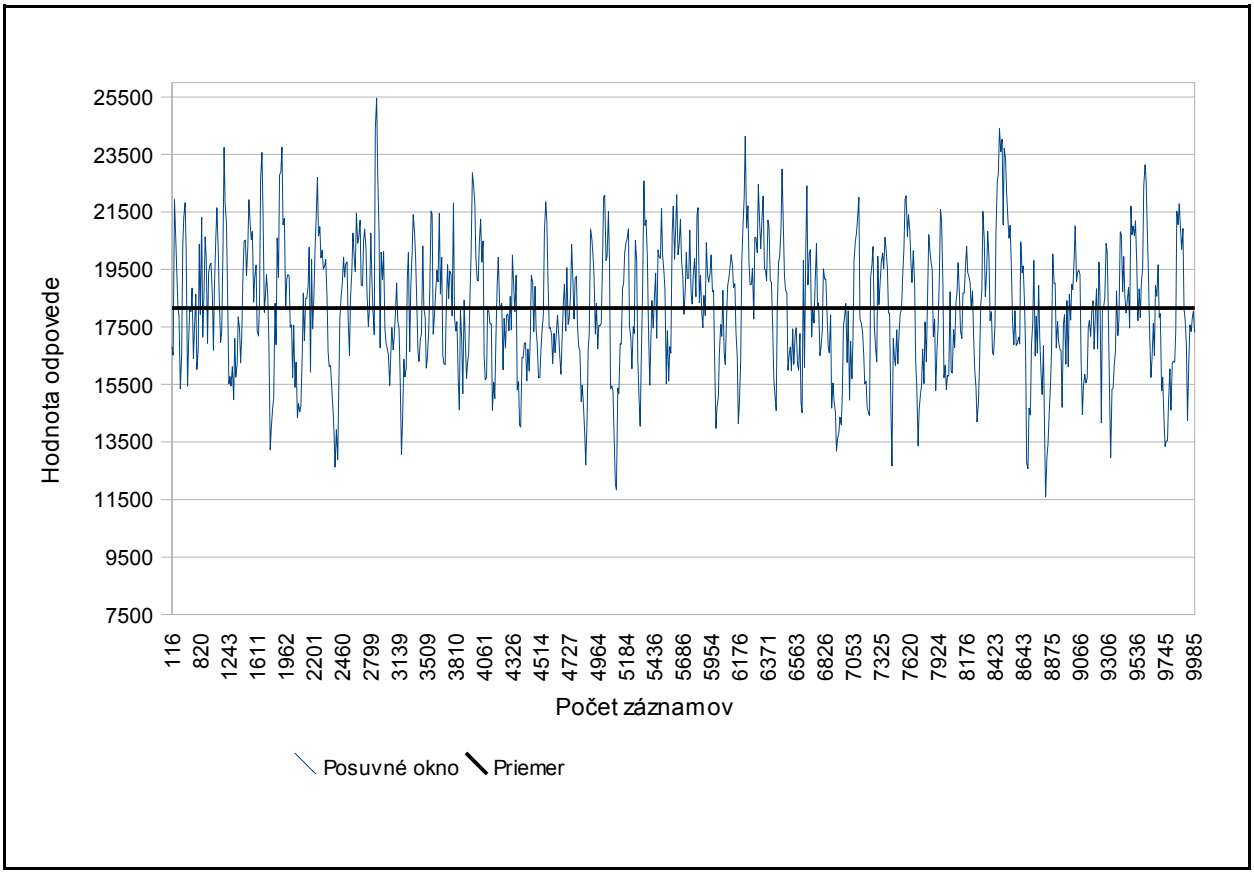
Graf 6.1: Odpovede algoritmu „Posuvné okno so zhlukovaním a histogramom“ na sadu 1



Graf 6.2: Odpovede algoritmu „Posuvné okno so zhlukovaním“ na sadu 1



Graf 6.3: Odpovede algoritmu „Posuvné okno“ na sadu 1



Algoritmus „Posuvné okno“, ktorý počíta s jednoduchou logikou posuvného okna má veľké odchýlky odpovede od skutočného priemeru. Tieto odpovede sa v čase nepribližujú k tomuto priemeru. Odpovede sa lokálne menia veľmi prudko, čo je veľmi nepríjemné pri dlhotrvajúcich dotazoch, ktoré očakávajú stabilné odpovede. Od priemeru vykazuje druhú najvyššiu priemernú odchýlku, ako ukazuje tabuľka 1. Z povahy algoritmu a testovacej sady je zrejmé, že algoritmus reaguje len na posledných N prvkov a podľa toho produkuje výsledky. Akonáhle sú príchodzie záznamy náhodne rozmiestnené v spektre stráca algoritmus „Posuvné okno“ požadovanú presnosť a veľmi sa vychyľuje.

Algoritmus „Posuvné okno so zhlukovaním“ vykazuje najvyššiu priemernú odchýlku spomedzi testovaných algoritmov. Na druhú stranu z Tabuľka 6.1 a z grafu Graf 6.2 je možné vyčítať, že jeho odpovede sa v čase nemenia tak prudko a zachovávajú približne svoju povahu. Odpovede sa približujú a vzďaľujú od priemeru na základe toho, ako boli záznamy v tom ktorom

momente spracované zhlukovaním. Tu sa ukazuje najväčšia nevýhoda zhlukovania, ktorý neprikladá dôležitosť váham jednotlivých záznamov.

Algoritmus „Posuvné okno so zhlukovaním a histogramom“ je na prvý pohľad najpresnejší vzhľadom na predchádzajúce dva algoritmy. Spočiatku síce osciluje okolo priemeru vo väčších odchýlkach, ale zhruba po načítaní 20 % záznamov sa ustáli a vykazuje veľmi presné odpovede. Odpovede sa v čase skoro vôbec nemenia (po načítaní spomínaných 20 %).

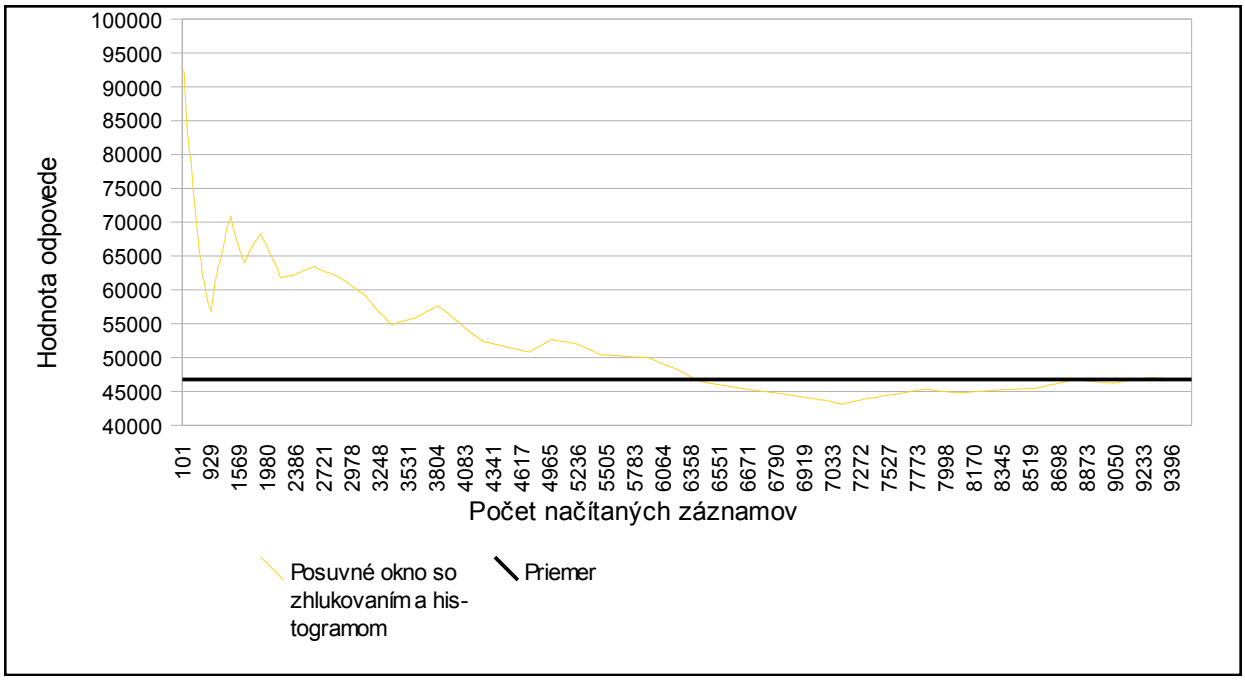
Tabuľka 6.1: Odchýlky algoritmov od priemeru pre testovaciu sadu 1

Algoritmus	***Priemerná odchýlka***	***% k testovacej sade***
„Posuvné okno“	2263,19	2,26319 %
„Posuvné okno so zhlukovaním“	4189,77	4,18977 %
„Posuvné okno so zhlukovaním a histogramom“	273,24	0,27324 %

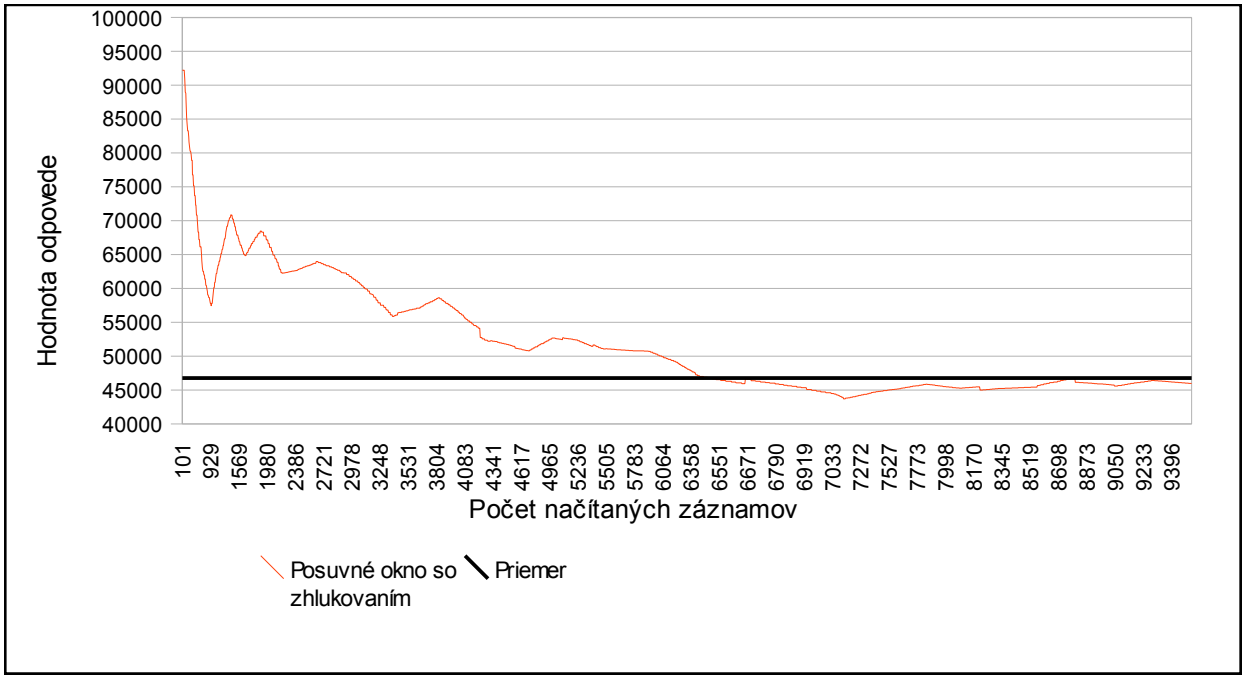
6.1.2 Testovacia sada 2

Graf Graf 6.6 ukazuje správanie všetkých troch algoritmov pre dáta z testovacej sady 2.

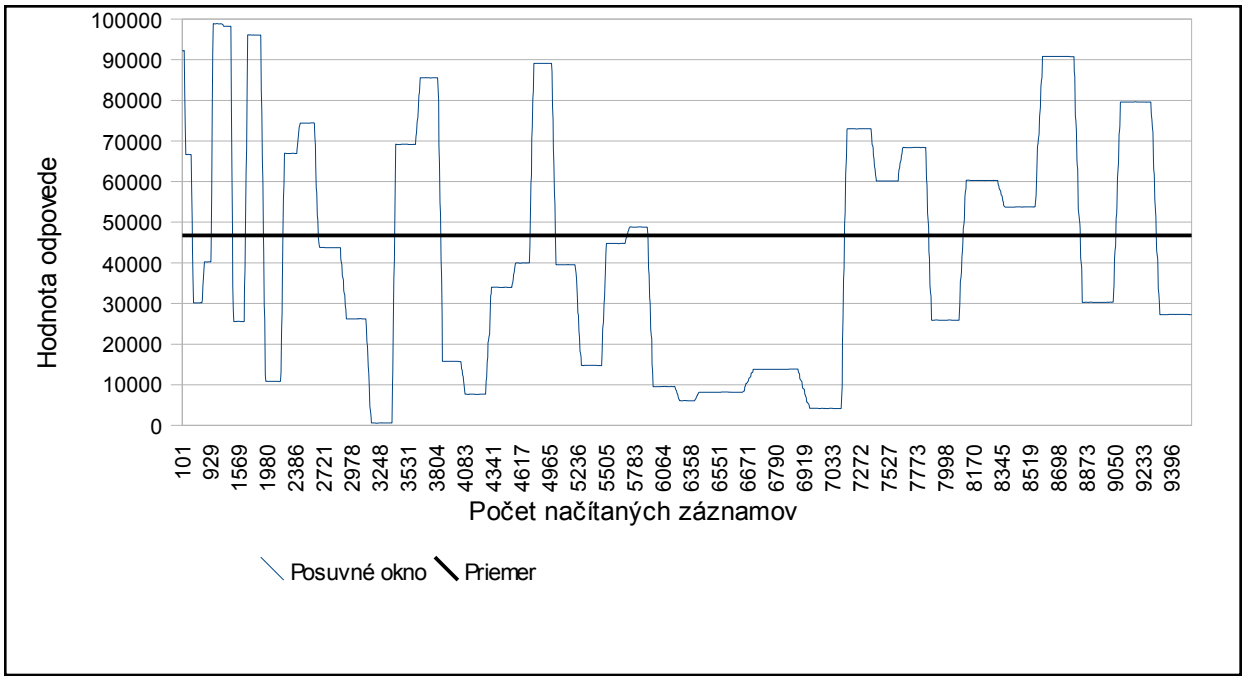
Graf 6.4: Odpovede algoritmu „Posuvné okno so zhlukovaním a histogramom“ na sadu 2



Graf 6.5: Odpovede algoritmu „Posuvné okno so zhlukovaním“ na sadu 2



Graf 6.6: Odpovede algoritmu „Posuvné okno“ na sadu 2



Algoritmus „Posuvné okno“ má pre túto testovaciu sadu najväčšiu odchýlku (viď tabuľka Tabuľka 6.2). Testovacia sada obsahuje zhluky približne rovnakej hodnoty. Algoritmus po načítaní časti zhluku zaplní celé posuvné okno záznamami z tejto oblasti a vykazuje odpoveď, ktorá je priemerom len tejto lokálnej oblasti. Z tohto dôvodu sú odpovede algoritmu „Posuvné okno“ tak nepresné. Nepríjemný je aj fakt, že algoritmus prudko mení odpoveď – v relatívne malom čase sa odpoveď zmení o 80 % spektra.

Algoritmus „Posuvné okno so zhlukovaním“ si viedol v tomto teste o poznanie lepšie ako pri testovacej sade 1. Spočiatku algoritmus vydáva nepresné odpovede, v závislosti na tom, že zhluky testovacej sady nie sú rovnomerne rozložené v spektre. Po načítaní približne 50 % záznamov však vydáva algoritmus veľmi presné odpovede.

Algoritmus „Posuvné okno so zhlukovaním a histogramom“ má veľmi podobné odpovede ako predchádzajúci algoritmus. Je to spôsobené tým, že je zrejmé, že každý zhluk má jednoznačného reprezentanta, ktorý má jednoznačnú váhu. Pri zvážení faktu, že všetky zhluky sú rovnako veľké, je zrejmé, že nie je dôležité prihliadať na váhy jednotlivých prvkov. Z tohto dôvodu je medzi poslednými dvomi algoritmami taký malý rozdiel.

Graf Graf 6.6 a tabuľka Tabuľka 6.2 ukazujú, že algoritmus „Posuvné okno“ má približne trikrát horšie odpovede ako zvyšné dva algoritmy, ktoré pracujú zhruba s rovnakou presnosťou odpovedí.

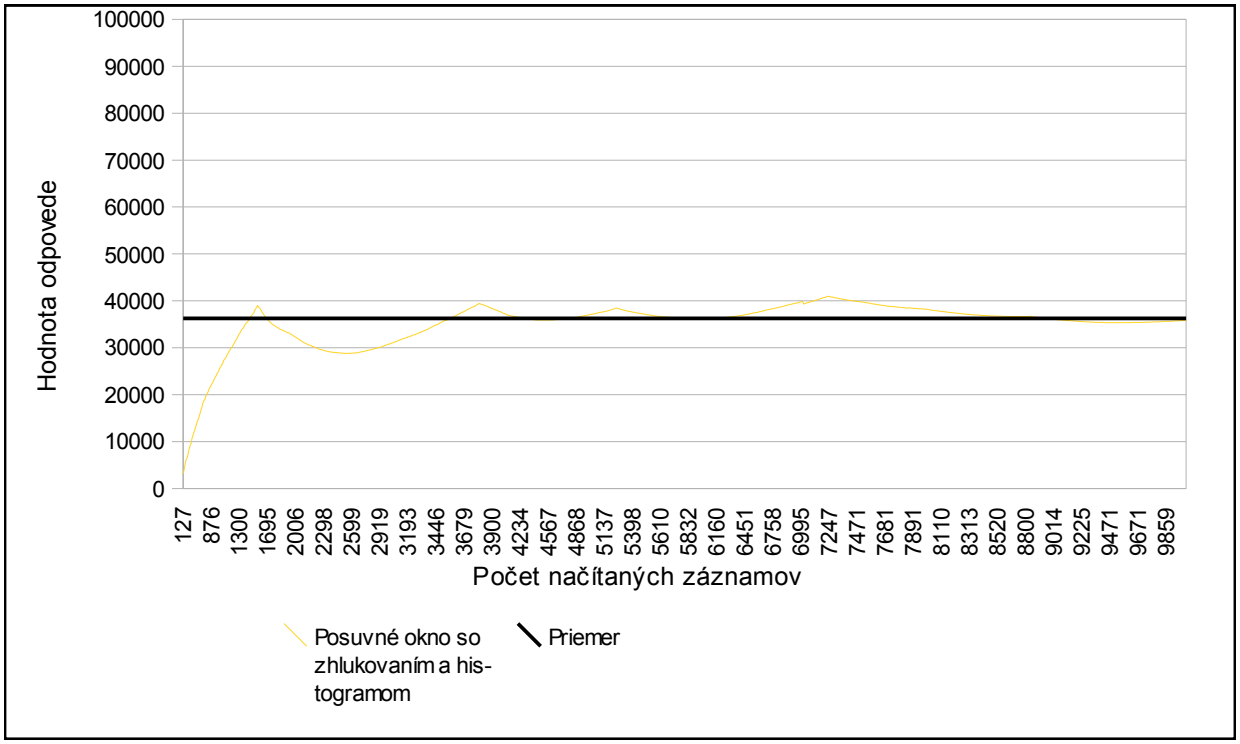
Tabuľka 6.2: Odchýlky pre všetky algoritmy pre testovaciu sadu 2

Algoritmus	***Priemerná odchýlka***	***% k testovacej sade***
„Posuvné okno“	28766,13	28,76613 %
„Posuvné okno so zhlukovaním“	9356,68	9,35668 %
„Posuvné okno so zhlukovaním a histogramom“	9003,78	9,00378 %

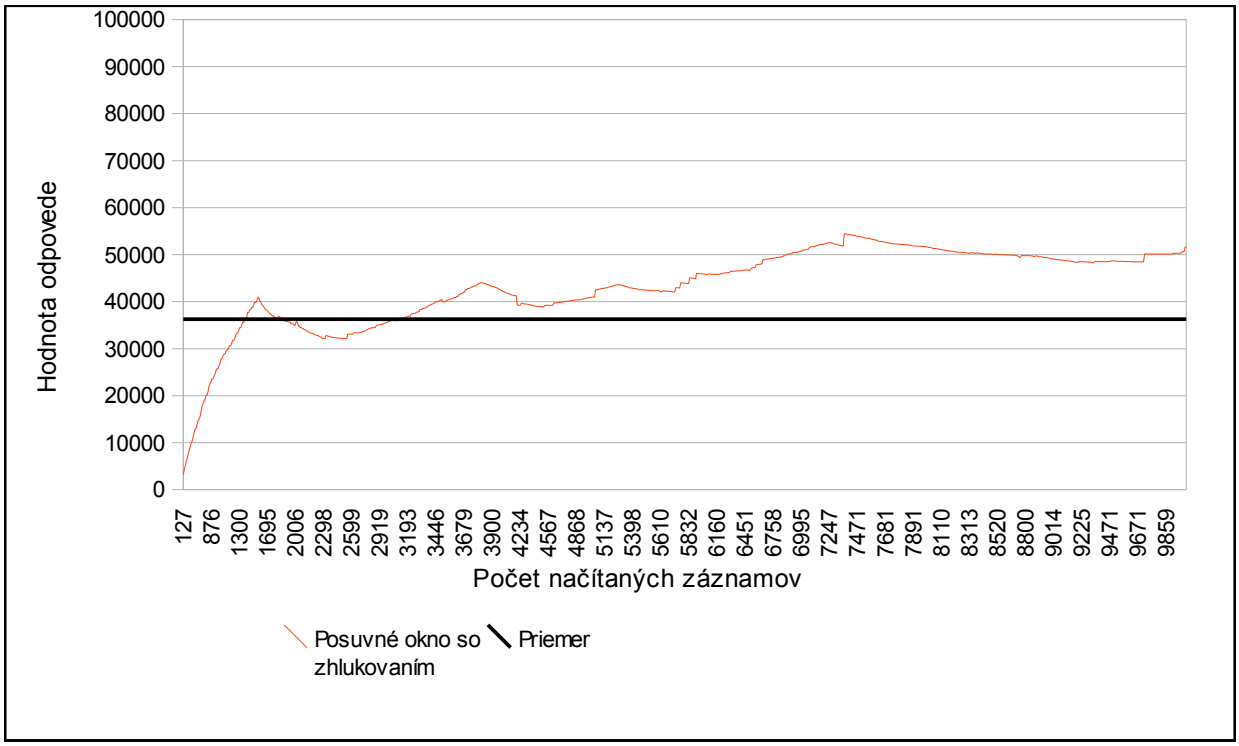
6.1.3 Testovacia sada 3

Tretia testovacia sada je zameraná na otestovanie schopnosti algoritmu predvídať vývoj sady pri dlhodobých monotónnych funkciách. Schopnosť predikovať správny priemer a neoscilovať vo veľkých odchýlkach.

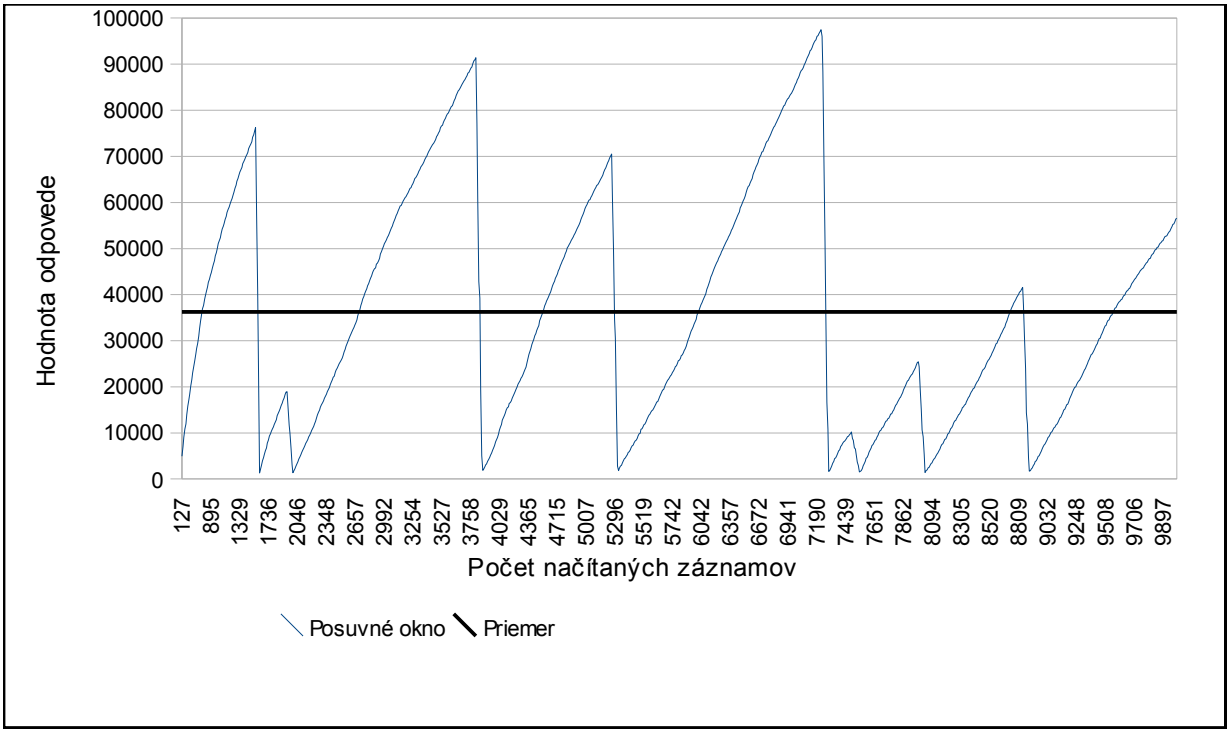
Graf 6.7: Odpovede algoritmu „Posuvné okno so zhlukovaním a histogramom“ na sadu 3



Graf 6.8: Odpovede algoritmu „Posuvné okno so zhlukovaním“ na sadu 3



Graf 6.9: Odpovede algoritmu „Posuvné okno“ na sadu 3



Algoritmus „Posuvné okno“ tak ako v predchádzajúcom prípade, kopíruje povahu načítaných záznamov. Opäť sú u tohto algoritmu znateľné výkyvy v odpovediach. Tento algoritmus podľa tabuľky Tabuľka 6.3 vykazuje znovu najhoršie odpovede.

Algoritmus „Posuvné okno so zhlukovaním“ po načítaní určitého množstva vstupných dát (cca 15 %) odpovedal na dotazy oveľa presnejšie ako predchádzajúci algoritmus a odpovede sú stabilné (v malom čase je zaručená malá zmena odpovede).

Algoritmus „Posuvné okno so zhlukovaním s histogramom“ znovu odpovedá s najvyššou presnosťou. Po prečítaní 10 % prúdu vcelku spoľahlivo a presne odpovedá na dotaz priemeru. Odpovede sú stabilné.

V tabuľke Tabuľka 6.3 sú zapísané vypočítané hodnoty, ktoré ukazujú, že algoritmus „Posuvné okno“ mal pri testovacej sade najhoršiu presnosť, kde priemerná odchýlka sa pohybuje na úrovni 25 %. Naproti tomu algoritmus „Posuvné okno so zhlukovaním“ dokázal znížiť chybovosť na 10 %, aj keď je vidieť, že ku koncu sady nekonverguje k správnemu výsledku. A nakoniec algoritmus „Posuvné okno so zhlukovaním a histogramom“ pracoval v príjemnej hladine 5 %.

Tabuľka 6.3: Odchýlky pre všetky algoritmy pre testovaciu sadu 3

Algoritmus	***Priemerná odchýlka***	***% k testovacej sade***
„Posuvné okno“	25383,36	25,38336 %
„Posuvné okno so zhlukovaním“	10984,15	10,98415 %
„Posuvné okno so zhlukovaním a histogramom“	5000,29	5,00029 %

6.2 Meranie rýchlosti

Druhým atribútom, ktorý je dôležitý pri spracovávaní dát je priemerná rýchlosť, akou dokáže algoritmus spracovať jeden záznam. Rýchlosť bola meraná opäť na troch testovacích sadách.

6.2.1 Testovacia sada 1

Testovacia sada 1 má náhodne rozmiestnené prvky v spektre. Nasledujúca tabuľka ukazuje časy, aké dosiahli testované algoritmy pri spracovaní 10000 záznamov.

Tabuľka 6.4: Rýchlosť algoritmov pre testovaciu sadu 1

Algoritmus	***Priemerný čas na 1 záznam***
„Posuvné okno“	0,0000094 sekundy
„Posuvné okno so zhlukovaním“	0,0017656 sekundy
„Posuvné okno so zhlukovaním a histogramom“	0,3277157 sekundy

Algoritmus „Posuvné okno“ nemá skoro žiadnu réžiu na udržiavanie štruktúry. Preto aj čas, ktorý dosiahol je v podstate čas potrebný na čítanie z disku 10000 riadkov. Hlavne kvôli vysokej rýchlosti je v súčasnosti často využívaný.

Algoritmus „Posuvné okno so zhlukovaním“ sa oproti predchádzajúcemu algoritmu líši tým, že vždy keď sa naplní posuvné okno spúšťa zhlukovanie pre nájdenie N / 2 mediánov. Čas potrebný na spracovanie všetkých záznamov je potom závislý na počte vykonaných zhlukovacích metód. Tento počet sa dá odhadnúť ako 2 * (počet záznamov) / N – 1.

Algoritmus „Posuvné okno so zhlukovaním a histogramom“ vykonáva zhlukovanie skoro pre každý vkladaný prvok a v približne 75 % vykonáva zhlukovanie viac krát pre jeden záznam. Preto je čas potrebný na spracovanie 10000 záznamov tak vysoký.

6.2.2 Testovacia sada 2

Pri testovacej sade 2 bola testovaná rýchlosť algoritmov pri načítavaní dlhých sekvencií rovnakých dát. Testovacia sada obsahovala zhluky približne rovnakej hodnoty. Výsledky meraní je možné nájsť v tabuľke Tabuľka 6.5**.**

Tabuľka 6.5: Rýchlosť algoritmov pre testovaciu sadu 2

Algoritmus	***Priemerný čas na 1 záznam***
„Posuvné okno“	0,0000094 sekundy
„Posuvné okno so zhlukovaním“	0,0023937 sekundy
„Posuvné okno so zhlukovaním a histogramom“	0,5461046 sekundy

Algoritmus „Posuvné okno“ tak ako pri predchádzajúcej testovacej sade dokázal spracovať záznamy v rovnakom čase. Pre tento algoritmus nie je rozhodujúca povaha dát. Algoritmus spracúva dáta v posuvnom okne nezávisle od hodnoty príchodzieho záznamu.

Algoritmus „Posuvné okno so zhlukovaním“ si poradil s testovacou sadou 2 o málo horšie ako s predchádzajúcou.

Algoritmus „Posuvné okno so zhlukovaním a histogramom“ skončil podľa tabuľky Tabuľka 6.5 opäť znateľne najhoršie spomedzi všetkých algoritmov. Jeho čas je dokonca ešte o 0,3s / záznam horší ako pri predchádzajúcej sade. Je to spôsobené tým, že algoritmus zhlukovania hľadá dlhšie sadu najlepších mediánov spomedzi podobných prvkov ako z diametrálne odlišných.

6.2.3 Testovacia sada 3

Tretia testovacia sada otestuje rýchlosť algoritmov pre vzrastajúce postupnosti. Výsledky nameraných údajov je možné nájsť v tabuľke Tabuľka 6.6.

Tabuľka 6.6: Rýchlosť algoritmov pre testovaciu sadu 3

Algoritmus	***Priemerný čas na 1 záznam***
„Posuvné okno“	0,0000094 sekundy
„Posuvné okno so zhlukovaním“	0,0020781 sekundy
„Posuvné okno so zhlukovaním a histogramom“	0,4418047 sekundy

Algoritmus „Posuvné okno“ preukázal svoju rýchlosť z už spomínaných dôvodov – je

nezávislý na povahe dát. Opäť dosiahol najrýchlejší čas spomedzi ostatných algoritmov, rovnako ako vo všetkých predchádzajúcich prípadoch.

Algoritmus „Posuvné okno so zhlukovaním“ mal pri tejto testovacej sade horší výsledok ako pri prvej testovacej sade, ale na druhú stranu lepšie ako v prípade testovacej sady 2.

Algoritmus „Posuvné okno so zhlukovaním a histogramom“

7 Vyhodnotenie výsledkov

Výsledky meraní z kapitoly 6 budú zhodnotené v troch oddelených kapitolách vzhľadom k jednotlivým testovacím sadám a v záverečnej podkapitole bude vyvodený všeobecný záver, ktorý platí pre všetky testované sady a bude hlavným výstupom testovaných algoritmov ako aj celej práce.

Dva testované atribúty boli rýchlosť a presnosť. Výsledným merítkom, ktorý určuje vhodnosť algoritmu na nasadenia do praxe je zrejme pomer týchto dvoch atribútov. Je dôležité si hneď na začiatku uvedomiť, na ktorý z týchto atribútov bude kladený väčší dôraz a v akom pomere. Toto rozhodnutie bude zrejme závisieť od konkrétneho nasadenia do praxe. Algoritmus zhlukovania tak, ako je implementovaný, od začiatku kládol vysoký dôraz na presnosť, pričom rýchlosť bola považovaná ako sekundárny meraný atribút.

7.1 Testovacia sada 1

V nasledujúcich odstavcoch budú zhodnotené všeobecné vlastnosti testovaných algoritmov na základe výsledkov, ktoré dosiahli pri použití testovacej sady 1. Pre vyvodenie záverov sú dôležité tabuľky Tabuľka 6.1: Odchýlky algoritmov od priemeru pre testovaciu sadu 1 a Tabuľka 6.4: Rýchlosť algoritmov pre testovaciu sadu 1. V týchto tabuľkách sú vyjadrené presnosti a rýchlosti jednotlivých algoritmov pre testovaciu sadu 1.

Je zrejmé, že algoritmus „Posuvné okno so zhlukovaním a histogramom“ je výrazne presnejší ako zvyšné dva. Naproti tomu jeho rýchlosť je nízka – dokáže spracovať priemerne tri záznamy za sekundu.

Najlepší pomer rýchlosť a presnosť dosahuje triviálny algoritmus „Posuvné okno“, hlavne vďaka svojej rýchlosti. Priemerná odchýlka 2,26 % je pre väčšinu aplikácií dostatočne nízka.

Najhoršie pri dátach z testovacej sady dopadol algoritmus „Posuvné okno so zhlukovaním“. Tento algoritmus nedokázal udržať priemernú odchýlka pod 4 % a súčasne pracoval oveľa dlhšie ako algoritmus „Posuvné okno“.

7.2 Testovacia sada 2

Druhá testovacia sada bola pre všetky algoritmy najťažšia na spracovanie, čo sa týka času aj presnosti. Je zrejmé, že zhluky v prúde dát pôsobia značné odchýlky vo výpočtoch a súčasne je pri nich nutné vyšší počet zhlukovaní. Algoritmus „Posuvné okno“ je približne trikrát nepresnejší ako zvyšné dva algoritmy, ale na druhú stranu je 250 krát rýchlejší ako algoritmus „Posuvné okno so zhlukovaním“ a 60 000 krát rýchlejší ako algoritmus „Posuvné okno so zhlukovaním a histogramom“.

7.3 Testovacia sada 3

V tabuľkách Tabuľka 6.3: Odchýlky pre všetky algoritmy pre testovaciu sadu 3 a Tabuľka 6.6: Rýchlosť algoritmov pre testovaciu sadu 3 sú uvedené výsledky algoritmov pre testovaciu sadu 3. Z týchto výsledkov je možné usudzovať, že algoritmus „Posuvné okno so zhlukovaním a histogramom“ bol opäť najpresnejší.

7.4 Zhrnutie výsledkov

Na základe výsledkov všetkých testov vychádza ako najlepší algoritmus triviálny algoritmus „Posuvné okno“. Je to hlavne vďaka jeho vysokej rýchlosti. Aj keď na druhú stranu je nutné poznamenať, že v niektorých prípadoch dosahovala priemerná odchýlka až 25 percent, čo je v nasadeniach vyžadujúcich presnosť neprijateľné. Práve pomer presnosti a rýchlosti, ktorý sa líši v závislosti na tom, kde bude systém nasadený, je kľúčom pri výbere správneho algoritmu.

Príklad nasadenia algoritmu „Posuvné okno so zhlukovaním a histogramom“ je napríklad pri spracovávaní prúdu dát z videokamier v budovách, kde je požiadavka približne na 1 záznam za sekundu. Algoritmus „Posuvné okno so zhlukovaním“ pracoval pre oba pozorované atribúty veľmi zle. Nebol ani rýchly a ani presný. Jeho nasadenie preto nie je odporúčané v takmer

žiadnych podmienkach.

Je zaujímavé zaoberať sa myšlienkou, keby sa testy vykonávali na oveľa väčších dátach. V tomto prípade je dôvod sa domnievať, že rýchlosť všetkých troch testovaných algoritmov by sa nezmenila. Spracovanie jedného záznamu by trvalo približne rovnako. Rozdiel by bol pozorovateľný určite v prípade, že by sa rozšírila veľkosť posuvného okna. Potom by algoritmus „Posuvné okno so zhlukovaním a histogramom“ pracoval ešte pomalšie, čo by bolo už veľmi nepríjemné. Avšak na druhú stranu sa dá podľa vývoja grafov predpokladať, že presnosť algoritmu „Posuvné okno so zhlukovaním a histogramom“ by sa od načítania určitého počtu záznamov ustálila na veľmi dobrej hodnote.

Všeobecne sa dá konštatovať, že algoritmus „Posuvné okno so zhlukovaním a histogramom“ sa chovalo, čo sa týka presnosti, veľmi dobre až po načítaní určitého počtu záznamov. Ak by sme odhliadli od prvých 20 % prúdu a spočítali odchýlku dostaneme zaujímavú tabuľku.

Tabuľka 7.1: Priemerná odchýlka pre testovaciu sadu 3 pri odhliadnutí od prvých 20 % prúdu

Algoritmus	Priemerná odchýlka	% k testovacej sade
„Posuvné okno“	26234,67	26,23467 %
„Posuvné okno so zhlukovaním“	11296,31	11,29631 %
„Posuvné okno so zhlukovaním a histogramom“	2005,72	2,00572 %

Pri porovnaní s tabuľkou Tabuľka 6.3: Odchýlky pre všetky algoritmy pre testovaciu sadu 3 vidíme, že jediný algoritmus „Posuvné okno so zhlukovaním a histogramom“sa dokázal zlepšiť – a to dosť znateľne. Podobné testy pri testovacích sadách 1 a 2 ukázali rovnakú tendenciu.

Pretože jednotlivé testované algoritmy sú vhodné do rôznych prostredí, ponúka sa myšlienka ponechať rozhodnutie na užívateľovi a vytvoriť systém podobný tomu ktorý bol implementovaný pre účely tejto práce. Tento systém by nechával užívateľa zvoliť, ktorý z algoritmov zvoliť. Napríklad pri prúde, kde je očakávaná vysoká frekvencia príchodzích prvkov by užívateľ zvolil obyčajný algoritmus „Posuvné okno“. Zatiaľčo pri dátových prúdoch, kde od prípadných dotazov budú vyžadované presné odpovede, môže užívateľ zvoliť algoritmus „Posuvné okno so zhlukovaním a histogramom“.

8 Záver

Cieľom práce bolo navrhnúť a implementovať prototyp prostredia na spracovávanie prúdu dát. Ťažiskom práce sa stalo vytvoriť algoritmus spracovávania dotazov, ktorý by odpovedal na položené dlhotrvajúce dotazy presnejšie, než doteraz využívané algoritmy.

Práca je rozdelená na dve časti. Prvá sa zaoberá teoretickým skúmaním problému vytvorenia prostredia SŘPD, ako aj niektorými konkrétnymi návrhmi algoritmov na spracovávanie dotazov. Druhá časť potom opisuje náš nový prístup k problému. Podrobne opisuje navrhnutý prototyp SŘPD, implementovaný algoritmus a vyvodzuje závery z výsledkov testovania.

Teoretická časť sa na začiatku zaoberá popisom základných pojmov. Nasleduje predstavenie všeobecnej architektúry systémov SŘPD s detailným popisom hlavných modulov. V závere teoretickej časti je dôraz kladený hlavne na ukázanie možných prístupov spracovávania dlhotrvajúcich dotazov. Jednotlivé prístupy sú dôkladne rozobrané, najmä prístup zhlukovania, ktorý bol vybratý ako hlavný prístup pre nami implementovaný algoritmus.

Navrhnutý algoritmus kombinuje tri prístupy spracovávania dotazov – zhlukovanie, kvôli zaisteniu zvýšenej presnosti, posuvné okno, kvôli zvýšeniu relevantnosti záznamov a histogram, kvôli vhodnej dátovej štruktúre ukladania záznamov. Očakávaný prínos tohto algoritmu bolo podstatné zvýšenie presnosti odpovedí na dlhotrvajúce dotazy pri spracovávaní prúdu dát.

Praktická časť práce popisuje implementáciu navrhnutého systému SŘPD so zameraním sa na testovaný algoritmus. Testovanie prebiehalo nad vygenerovanými testovacími dátami, ktoré boli zvolené štandardne vzhľadom k problematike.

Výsledky testov neurčili jednoznačne najvhodnejší algoritmus na spracovávanie dát. Avšak boli vyvodené závery na základe, ktorých je možné konštatovať, že využite algoritmu závisí na type nasadenia a na povahe spracovávaných dát. V prípadoch, kde je požiadavka na vysokú rýchlosť spracovania, je najvhodnejší prvý algoritmus („Posuvné okno“) s minimálnou vnútornou logikou. Naproti tomu pre nasadenia, vyžadujúce vysokú presnosť na úkor rýchlosti spracovania, je náš navrhnutý algoritmus („Posuvné okno so zhlukovaním a histogramom“) najlepšie riešenie. Najvhodnejšou cestou je zrejme ponechať výber na užívateľovi, ktorý by mal

z povahy dát a nasadenia určiť, ktorý algoritmus použiť. Prostredie SŘPD by teda malo obsahovať nástroj, pomocou ktorého by sa užívateľ rozhodoval medzi viacerými algoritmami. Náš navrhnutý prototyp prostredia SŘPD podobný nástroj obsahuje v podobe voľby, či má byť daný atribút prúdu dát zhlukovaný alebo nie.

Prínos tejto práce je v navrhnutí a implementovaní nového algoritmu pre spracovávanie dát, hlavne potom v jeho otestovaní a porovnaní s už existujúcimi algoritmami. Sekundárny prínos je prehľad prístupov na spracovávanie dát a uvedenie do problematiky SŘPD v slovenskom jazyku.

Hlavným nedostatkom práce je pomalosť navrhnutého algoritmu. Ako už bolo spomínané, táto pomalosť nemusí byť nevýhodou v niektorých nasadeniach, kde nie je potrebné rýchlo spracúvať záznamy. Rozšírenie riešenia by sa mohlo zaoberať otázkou zrýchlenia algoritmu, napríklad vhodnejším výberom začiatočných centier, alebo uspokojením sa s nižšou presnosťou (testovanie rôznych hodnôt parametrov konvergencie), čím by sa algoritmus zrýchlil. Predmetom skúmania ostáva, či by zrýchlenie bolo dostatočnou kompenzáciou presnosti.

9 Literatúra

[1] B.Babcock, S.Babu, M. Datar, R. Motwani, and J. Widom. Model and issues in data stream systems. In PODS, 2002

[2] Yan-Nei Law, Haixun Wang, Carlo Zaniolo. Query Languages and Data Models for Database Sequences and Data Streams

[3] V. Raman, B. Raman nad J. Hellerstein. Online dynamic reordering for interactive data processing. In Proc. Of the 1999 Intl. Conf. On Very Large Data bases, 1999

[4] P. Tucker, D. Maier, T. Sheard and L. Fegaras. Enhacing relational operators for querying over punctuated data streams. Manusript, 2002

[5] L. Golab, M. Tamer Ozsu. Data Stream management isuues - A survey. 2003

[6] J. Vitter. Random sampling with a reservoir. ACM Trans. On Mathematical Software, 1985

[7] H. Jagadish, N.Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik and T. Suel. Optimal histograms with quality guarantees. In Proc. Of the 1998 Intl. Conf. On Very Data Bases, 1998

[8] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S/ Muthukrishnan and M. Strauss. Fast, small-space algorithms for apprroximate histogram maintenance. In proc. Of the 2002 Annual ACM Symp. On Theory of Computing, 2002

[9] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In Proc. Of the 2001 ACM SIGMOD Intl. Conf. On Management of Data, 2001

[10] M. Fang, N.Shivakumar, H. Garcia-Molina, R. Motwani, J.D.Ullman. Computing iceberg queries efficiently, 1998

[11] G. Manku, R.Motwani. Approximate frequency counts over streaming data, 2002

[12] K. Chakrabarti, M. N. Garofalakis, R. Rastogi and K. Shim. Approximate query processing using wavelets. 2000

[13] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan nad M. Strauss. Fast, small-space algorithms for approximate histogram maintenance, 2002

[14] A. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss. Surfing wavelets on streams: One

pass summaries for approximate aggregate queries, 2001

[15] M.Datar, A. Gionis, P. Indyk and R. Motwani. Maintaining stream statistics over sliding windows. In Proc. Of the 2002 Annual ACM-SIAM Symp. On Discrete Algorithms, 2002

[16] S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan. Clustering Data Streams: Theory and Practice, 2003

[17] M. Charikar, S. Chekuri, T. Feder and R. Motwani. Incremental clustering and dynamic information retrieval, 1997

[18] B.Babcock, M. Datar, R. Motwani, L. O'Callaghan. Mainaining variance and k-median over data Stream Windows

[19] P. Indyk. Sublinear time algorithms for metric space problems, In Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999

[20] M. Charikar nad S. Guha. Improved Combinatorial Algorithms for the facility location and k-median problems, 1999

[21] J. Pokorný, V. Snášel. Zpracování proudu dat, 2006

[22] The STREAM Group. STREAM: The Stanford Stream Data Manager (short overview paper) IEEE Data Engineering Bulletin, March 2003

Príloha A

Obsah CD

Priložené CD obsahuje okrem kompletného textu diplomovej práce aj zdrojové súbory prototypu implementovaného SŘPD. Taktiež obsahuje testovacie dáta, pre ktoré boli robené testy. Štruktúra adresárov na CD je nasledujúca.

- **/text** – text diplomovej práce vo formáte pdf
- **/grafy** – obsahuje namerané testované hodnoty aj s vygenerovanými grafmi vo formáte openoffice
- **/clustering/src** – zdrojové súbory prototypu
- **/clustering/doc** – vygenerovaná dokumentácia Javadoc
- **/clustering/dat** – vstupné testovacie dáta použité pre testovacie účely
- **/clustering/static** – konfiguračné súbory programu
- **/clustering/bin** – preložené zdrojové súbory

Príloha B

Užívateľská príručka prototypu

Program je spúšťaný bez parametrov. Všetky konfigurácie sú riešené pomocou externých súborov. Po spustení si program načíta spomínané externé súbory a podľa nich začne vykonávať činnosť. Pracuje až do chvíle, kým všetky prúdy dát neskončia.

Vstupné dáta

Pre bezproblémové spustenie programu je nutné nakonfigurovať súbory „static/streams.txt“ a „static/queries.txt“. Spôsob, ako má vyzerať formát súboru je popísaný v odstavcoch 4.1.1 Štruktúra vstupnej časti a 4.2.1 Statická pamäť. Pre upresnenie významu príznaku zhlukovania slúži nasledujúca tabuľka.

Príznak	Význam
0	Algoritmus „Posuvné okno“
1	Algoritmus „Posuvné okno so zhlukovaním“
2	Algoritmus „Posuvné okno so zhlukovaním a histogramom“

Formát súboru nie je kontrolovaný, preto je dôležité pri registrovaní nových prúdov dbať na správnosť a korektnosť vkladaných údajov.

Spustenie prototypu

Prototyp je naprogramovaný v jazyku Java. Spúšťa sa štandardne pomocou príkazovej riadky, napr. : java -cp bin org.clustering.Main.

Výstupné dáta

Program zapisuje informácie na konzolu a do výstupného prúdu. Na konzolu je zapísaný

čas a informácia o tom, aký prúd sa začal spracovávať, príp skončil spracovávať. Do výstupného prúdu sú potom zaznamenávané odpovede na dlhodobé dotazy. Výstupný prúd v je prototype implicitne nastavený na súbor output.txt. V súbore output.txt sú položky oddelené stredníkom. Prvá položka určuje čas dotazu. Druhá položka označuje prúd dát, nad ktorým sa vykonával dotaz. Tretia položka označuje atribút prúdu, ktorý bol dotazovaný. Štvrtá položka je samotný výsledok dotazu, v prípade priemeru predchádza prefix AVG, v prípade dátovej položky je priamo uložená hodnota.