VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

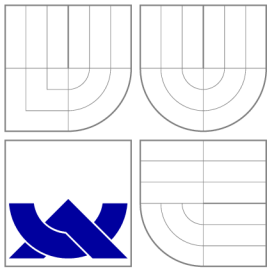
DOLOVÁNÍ V PROSTŘEDNÍ MS SQL POMOCÍ INKREMENTÁLNÍCH ALGORITMŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

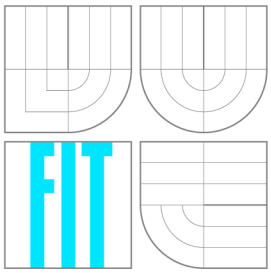
AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ DAVID

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DOLOVÁNÍ V PROSTŘEDNÍ MS SQL POMOCÍ INKREMENTÁLNÍCH ALGORITMŮ

DATAMINING IN MS SQL USING INCREMENTAL ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ DAVID

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠEBEK

BRNO 2012

Abstrakt

Tato práce pojednává o problematice dolování v proudu dat, která patří v dnešní době k velmi dynamickým oblastem informačních technologií. V práci jsou popsány obecné principy dolování v datech a principy dolování v proudových datech. Podrobně je rozebrán implementovaný algoritmus CluStream. V rámci praktické části bylo navrženo a implementováno řešení zpracování proudových dat v technologii MSSQL za použití výše uvedeného algoritmu. Funkčnost algoritmu pak byla ověřena za pomocí vlastního generátoru proudu dat.

Abstract

This work deals with issues in data streams mining which nowadays is a very dynamic area in information technology. The thesis describes the general principles of data mining. There are also the principles of data mining in the data streams. Special attention is given to the implemented algorithm CluStream. In the practical part the data stream processing solution was designed and implemented by the MSSQL technology using the above algorithm. The functionality of the algorithm was verified using own data stream generator.

Klíčová slova

Dolování v datech, proud dat, shlukování, shluková analýza, CluStream, MS SQL, CLR Integration, .NET framework, Microsoft StreamInsight, MOA

Keywords

Data mining, data stream, clustering, cluster analysis, CluStream, MS SQL, CLR Integration, .NET framework, Microsoft StreamInsight, MOA

Citace

Lukáš David: Dolování v prostřední MS SQL pomocí inkrementálních algoritmů, diplomová práce, Brno, FIT VUT v Brně, 2012

Dolování v prostřední MS SQL pomocí inkrementálních algoritmů

Prohlášení

Prohlašuji, že jsem tento diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Šebka.

.....
Lukáš David
22. května 2012

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu Ing. Michalu Šebkovi za cenné rady při práci na diplomovém projektu.

Obsah

Kapitola 1

Úvod

Oblast získávání znalostí z databází prošla v posledních desetiletích obrovským vývojem. Vděčí za to zejména obrovskému rozmachu internetu a informačních technologií, kdy stále více aktivit spojených s každodenním životem probíhá skrze internet, případně jiné informační subjekty. Jako příklad si lze uvést internetové bankovnictví, nakupování, sociální sítě, provoz na síti atp. Tyto aktivity pak tvoří nekonečný proud dat, která je potřeba zpracovávat a získávat z nich potřebné informace (znalosti). Může se jednat například o detekci neobvyklé sekvence transakcí na bankovních účtech, jenž může znamenat potenciální hrozbu korupce atp. Cílem práce je uvést čtenáře do problematiky získávání znalostí z databází a zpracování proudů dat. Dále jsou zde představeny možnosti zpracování proudu dat nad MS SQL serverem a popis návrhu a implementace vybraného algoritmu v tomto prostředí. Výsledek této práce pak bude dále využit v některých projektech na Fakultě informačních technologií VUT v Brně.

První kapitola je věnována úvodu. Tato kapitola má za cíl uvést čtenáře do problematiky rozebírané v rámci celé práce. Na tuto kapitolu navazuje druhá kapitola, ve které je uveden obecný úvod k získávání znalostí z databází a jeho spojení s oblastí strojového učení, potažmo umělé inteligence. Dále je zde uveden stručný popis historie dolování z dat a nejběžnější typy dolovacích úloh.

Třetí kapitola je celá věnována jednomu typu dolovací úlohy, a to konkrétně shlukové analýze, neboť pro vlastní implementaci v rámci diplomové práce byl vybrán shlukovací algoritmus CluStream. V kapitole jsou nejdříve rozebrány obecné požadavky na shlukové algoritmy, posléze typy dat se kterými tyto algoritmy pracují. Dále je zde uvedeno základní rozdělení shlukových metod spolu se stručným popisem každé z nich.

Vzhledem k zaměření práce na proudová data je další kapitola věnována problematice proudů dat. Jsou zde uvedeny největší výzvy související s tímto typem dat. Jako příklad lze uvést potenciální nekonečnost těchto dat. Neméně zajímavou částí kapitoly je oblast porovnání dolování z klasických a z proudových dat. Dále jsou zde zmíněny některé speciální postupy (algoritmy) předzpracování dat, jejichž návrh přímo souvisí s potenciální nekonečností těchto dat. Poslední část kapitoly obsahuje rozbor jednotlivých typů dolovacích úloh vztažený právě k proudovým datům spolu s uvedením největších problémů, které u těchto úloh musíme řešit.

Pátá kapitola se věnuje podrobně vybranému algoritmu CluStream. Nejdříve je zde rozebrán nutný teoretický základ pro pochopení funkčnosti vlastního algoritmu. Dále je zde podrobně rozebrán postup zpracování vstupních dat, jejich ukládání a další zpracování.

V šesté kapitole je popsán vlastní návrh řešení. Nejdříve jsou zde rozebrány použité technologie, na které navazuje rozbor navrženého řešení v podobě diagramu balíčků a popis

funkčnosti jednotlivých balíčků. Součástí tohoto rozboru je i diskuze nad dalšími možnostmi řešení daného problému spolu s odůvodněním, proč bylo vybráno právě implementované řešení. V poslední části kapitoly jsou detailněji rozebrány jednotlivé balíčky v podobě diagramů tříd a jejich popisu.

Sedmá kapitola pojednává o výsledné implementaci. Jsou zde rozebrány některé významné části implementace jednotlivých balíčků. Dále jsou zde uvedeny ukázky vlastního uživatelského rozhraní výsledné aplikace.

V osmé kapitole je rozebrán postup testování implementovaného algoritmu. Nejdříve je zde rozebrána implementace generátoru proudu dat, jenž se využívá při vlastním testování. V další části kapitoly jsou ukázky jednotlivých testů a jejich výsledků. Poslední část kapitoly se pak věnuje porovnání výsledné implementace algoritmu s implementací v rámci frameworku MOA. Na závěr je zde uvedeno zhodnocení dosažených výsledků.

V závěru jsou potom shrnuty veškeré poznatky z celé diplomové práce.

Kapitola 2

Získávání znalostí z databází

V dnešní době jsme doslova zahlceni obrovským množstvím různorodých informací. Jedná se o např. o podnikové databáze, bankovní transakce, prodeje zboží atp. Proces získávání znalostí nazývaný také *dolování z dat* je proces, který se snaží v těchto datech vydolovat nějakou netriviální, zajímavou a pro naše účely užitečnou informaci, případně znalost. Netriviální informací je myšlena taková informace, kterou nelze získat například pouhým SQL dotazem, nicméně je potřeba použít nějaký sofistikovaný přístup. Jako příklad si můžeme uvést situaci, kdy banka eviduje určitá data (výše příjmů, zaměstnání, věk atp.) o svých klientech, kterým poskytla půjčku. Dále banka eviduje, jak tito klienti splácí danou půjčku, tj. jestli vše platí v termínech či nikoliv. Na základě těchto informací může chtít banka vytvořit model pro klasifikaci nově příchozích klientů do různých tříd dle rizika, že daní klienti nebudou splácet svoji půjčku. Tuto informaci posléze může použít v rozhodování, zdali novému klientovi poskytne půjčku či nikoliv. Při tvorbě takovéhoto modelu dochází k strojovému učení bankovního počítače. Strojové učení je podrobně rozebráno v následující podkapitole.

2.1 Strojové učení

K řešení problému na počítači potřebujeme algoritmus, jenž je posloupnost instrukcí transformující vstup na výstup. V praxi máme nicméně spoustu problému, které nelze vyřešit pouhým napsáním takovéhoto jednoduchého algoritmu. Jako příklad lze uvést rozpoznávání ručně psaného textu, hledání skrytých vzorů v datech atp. V těchto případech je nutné použít speciální algoritmy, jenž implementují určité modely jako například neuronovou síť, rozhodovací strom atp.

Strojové učení je tedy vědecká disciplína zabývající se programováním počítačů za účelem optimalizace výkonnostních kritérií na základě vstupních dat, případně předchozích zkušeností. Dochází zde k programovému nadefinování určitého modelu a jeho atributů. V průběhu běhu programu dochází k úpravě těchto parametrů a modelu za použití trénovacích dat nebo předchozích zkušeností. Takováto úprava je chápána jako učení daného modelu. Existují nicméně i modely, jenž nevyužívají žádná trénovací data, neboť daný typ úlohy takováto data neposkytuje. Při psaní tohoto odstavce jsem čerpal informace z [1].

Algoritmy (modely) strojového učení lze tedy rozdělit dle způsobu učení do následujících kategorií [2]:

- *Strojové učení bez učitele* – jedná se o učení, kdy hledáme skrytou informaci (strukturu) v datech, aniž bychom měli nějaká trénovací data. Do tohoto typu učení můžeme

zařadit například shlukovou analýzu.

- *Strojové učení s učitelem* – zde máme určitou množinu trénovacích dat (dvojic - vstup, očekávaný výstup). Na základě této množiny vytváříme funkci, která může být diskrétní, nebo spojitá. Diskrétní funkce se nazývá klasifikátor, spojitá funkce se nazývá regrese. Do tohoto typu učení spadá například klasifikace.

- *Učení posilováním* – jedná se o model učení softwarového agenta, který se snaží určit své ideální chování v rámci daného kontextu tak, aby maximalizoval jeho výkon. Zde je požadovaná zpětná vazba (odměna) agentovi, která dopomáhá jeho učení. Model tohoto typu učení je poněkud komplikovanější, více lze nalézt na [2].

Algoritmy strojového učení lze také dělit dle zpracování trénovací množiny, a to do dvou skupin:

- *Dávkové algoritmy* – tyto algoritmy zpracovávají veškerá data na začátku činnosti (dávkově).

- *Inkrementální algoritmy* – jedná se o algoritmy, které zpracovávají vstupní data postupně. Díky této vlastnosti jsou tyto algoritmy využívány pro zpracování proudu dat, neboť zde nemáme všechna data na začátku činnosti. Proudy dat a některé inkrementální algoritmy jsou rozebrány v kapitole 4.

2.2 Historie získávání znalostí

Historie dolování dat je úzce spjata s historií databázových systémů. Je to dáno skutečností, že výhradním zdrojem dat pro dolování jsou právě databázové systémy. Počátky databázových systémů se datují do 60. let 20. století. V této době se používaly výhradně síťové a hierarchické databázové systémy. Tyto modely byly v 70. letech nahrazeny relačním modelem dat, jehož implementací je systém řízení báze dat. Tento systém byl v následujících letech dále vyvíjen a zdokonalován, vznikly pokročilé databázové modely jako např. deduktivní databáze.

V 90. letech již byly tyto systémy natolik vyvinuty, že vznikal tlak pro podporu pokročilejších operací (operace pro podporu rozhodování, získávání znalostí, analýzu dat atp.). V tomto období vznikají technologie datových skladů, algoritmy pro dolování v datech. Počátkem 21. století došlo k vývoji dolovacích algoritmů i nad jinými typy dat, jako jsou např. proudy dat.

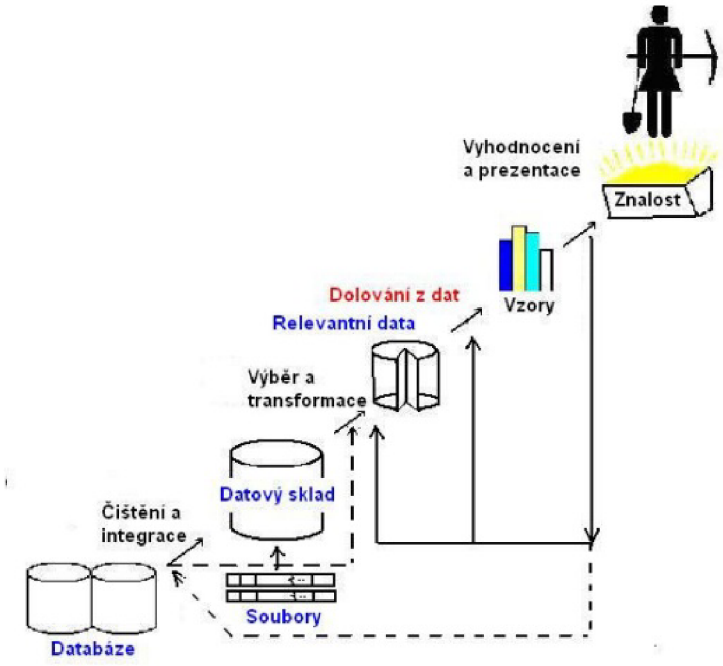
2.3 Proces získávání znalostí

Proces získávání znalostí(dolování dat) se skládá z několika fází, jenž jsou vidět na obrázku 2.1. Jedná se o fáze [3]:

1. **Čistění dat** – data pro dolování mohou být různým způsobem špatná. Může se jednat o chybějící, nesmyslné a nekonzistentní hodnoty. Cílem této fáze je vypořádat se s těmito chybami a upravit vstupní data do správné podoby.

2. **Integrace dat** – data pro dolování můžeme získat z několika zdrojů. Cílem této fáze je z těchto datových zdrojů vytvořit jeden koherentní zdroj.

3. **Výběr dat** – cílem této fáze je vybrat data relevantní pro námi požadovanou dolovací úlohu.
4. **Transformace dat** – cílem této fáze je transformovat data do podoby vhodné pro dolování.
5. **Dolování dat** – hlavní fáze procesu získávání znalostí. Cílem je pomocí zvolené metody získat z dat potřebné informace (vzory, modely).
6. **Vyhodnocení modelů a vzorů** – cílem fáze je identifikace zajímavých vzorů a modelů z pohledu dané úlohy.
7. **Prezentace znalostí** – zobrazení informace v podobě vhodné pro uživatele.



Obrázek 2.1: Proces získání znalostí z databází, převzato z [3]

2.4 Dolovací úlohy

Pro dolování dat máme k dispozici několik typů úloh a algoritmů k jejich realizaci. Obecně se dolovací metody dělí na dvě základní skupiny *deskriptivní* a *prediktivní*. Deskriptivní metody hledají obecné vlastnosti analyzovaných dat, naproti tomu prediktivní metody analyzují stávající data za účelem vytvoření modelu pro predikci budoucích dat. V následujících podkapitolách bude uveden stručný úvod k základním typům dolovacích úloh. Více k některým dolovacím metodám, vztaženo na zpracování proudových dat, bude uvedeno v podkapitole 4.3. Při popisu jednotlivých typů úloh jsem čerpal informace z [3], [4].

2.4.1 Popis konceptu/třídy

Jedna ze základních dolovacích úloh. Data jsou rozdělena do jednotlivých tříd (konceptů), které jsou dále popsány. Třídy jsou popsány buď pomocí *charakterizace* dat či *diskriminace* dat. *Charakterizací* dat je myšlena sumarizace obecných vlastností analyzované třídy. *Diskriminace* dat popisuje vlastnosti třídy pomocí vymezení vztahu vůči vlastnostem ostatních rozdílových tříd.

2.4.2 Frekventované vzory a asociační pravidla

Tento typ úlohy patří do skupiny deskriptivních úloh. Frekventované vzory jsou takové vzory, které se vyskytují v analyzovaných datech poměrně často. Cílem této dolovací úlohy je nalézt množinu frekventovaných vzorů ve vstupních datech a z nich posléze vytvořit příslušná asociační pravidla. K popisu asociačních pravidel používáme dvě metriky, a to *podporu* a *spolehlivost*.

Podpora určuje frekvenci daného vzoru ve vstupních datech, spolehlivost určuje závislost dat (atributů) v rámci těchto vzorů. Pakliže asociační pravidlo splňuje vlastnosti minimální podpory a spolehlivosti, je označeno za zajímavé pro příslušnou analýzu. Typickou aplikací této dolovací úlohy je analýza nákupního košíku, kdy máme obrovské množství nákupních transakcí a snažíme se z nich získat zajímavé vzory (trendy) při nákupu zákazníků, které posléze mohou být použity např. pro reklamu.

2.4.3 Analýza odlehlých hodnot

Jedná se o jakýsi protipól oproti frekventovaným vzorům. Při této analýze hledáme právě takové hodnoty (vzory), které se v daných datech objevují zcela mimořádně. Tato analýza se často používá pro odhalení určitých podvodných aktiv a různých neobvyklých akcí.

2.4.4 Klasifikace, predikce

Klasifikace a predikce patří mezi typické představitele prediktivních dolovacích úloh. *Klasifikace* se skládá ze dvou kroků. Prvním krokem je učení, kdy na základě trénovací množiny dat vytváříme model jednotlivých tříd dat, schopný klasifikovat budoucí neznámá data do těchto tříd. Druhým krokem je vlastní klasifikace nových dat tj. jejich zařazení do jednotlivých tříd. *Predikce* značí proces předpovědi nějakého chybějícího atributu nového záznamu na základě vzniklého modelu dat. Klasifikace je jedním z druhů predikce, kdy předpovídáme atribut představující třídu daného objektu.

2.4.5 Shlukování

Vzhledem k tomu, že tématem práce je implementace shlukového algoritmu, je tento typ úlohy rozebrán podrobněji v následující kapitole.

Kapitola 3

Shluková analýza

Shlukování rozděluje analyzovaná data do jednotlivých tříd na základě jejich podobnosti. Na rozdíl od klasifikace vytváří tyto třídy až v průběhu zpracování dat. Požadovanou vlastností shlukování je, aby byly objekty v rámci jedné třídy hodně podobné a zároveň co nejvíce odlišné od objektů jiných tříd. Podobnost objektů je určena hodnotami jejich atributů a vzdálenostní funkcí. Volba vzdálenostní funkce se odvíjí od typu příslušných atributů. Více k jednotlivým typům vzdálenostních funkcí bude uvedeno dále v kapitole. Z hlediska využitelnosti jednotlivých shlukových metod nás zajímají pouze takové metody, jenž jsou schopny zpracovávat rozsáhlé databáze. Každá shluková metoda má určité vlastnosti, z kterých vyplývá její použití. Typicky jsou na shlukové metody kladeny následující požadavky [3]:

- **Škálovatelnost** – požadavek, aby si algoritmy poradily i s velkými objemy dat. Některé algoritmy pracují dobře pouze s menšími objemy dat, což se dá vyřešit vzorkováním dat z databáze, nicméně to sebou nese riziko zkreslení výsledků. Proto je potřeba mít vysoce škálovatelné algoritmy.
- **Schopnost zpracovávat různé druhy atributů** – pro většinu algoritmů je typické zpracovávání pouze numerických dat, nicméně některé aplikace vyžadují zpracování i jiných druhý atributů např. binárních, ordinálních atp.
- **Vytvářet shluky různého tvaru** – většina algoritmů vytváří shluky kulovitého tvaru, což vždy nemusí odpovídat skutečným třídám v datech.
- **Minimální požadavky na znalost problému při určování parametrů** – u některých algoritmů je nutné zadat některé vstupní parametry před vlastní analýzou dat. Nejběžnějším takovým parametrem je požadovaný počet shluků. Takto zvolené parametry mají pak velký vliv na vlastní shlukování.
- **Schopnost vyrovnat se s daty obsahující šum** – potřeba vyrovnat se s určitým procentem chybějících, chybných nebo neznámých dat.
- **Schopnost zpracovávat vysokodimenzionální data** – některé algoritmy zpracovávají dobře data o nízkém počtu dimenzí (2-3), nicméně více významné jsou algoritmy zpracovávající více dimenzionální data.
- Mezi další požadavky patří také **schopnost shlukování na základě omezování**, vytváření **interpretovatelných a použitelných shluků** a **necitlivost na pořadí vstupních záznamů**.

3.1 Typy dat

V rámci této podkapitoly jsem čerpal informace z [3], [4]. Vzhledem k tomu, že shluková analýza stojí na určování podobnosti dat a jejich přiřazování do shluků, vzniká zde otázka, jak určit podobnost takovýchto dat. Podobnost je určena vzdálenostní funkcí. Vzdálenost objektů i a j označíme jako *d(i,j)*. Obecně *d(i,j)* je nezáporné číslo blížící se 0, pokud jsou si objekty i a j velmi podobné. Naopak čím více jsou objekty odlišné, tím větší hodnoty nabývá vzdálenostní funkce. Určování vzdálenosti je závislé na konkrétních typech dat jednotlivých atributů, jenž budou rozebrány v následujícím textu.

3.1.1 Intervalové proměnné

Intervalové proměnné jsou spojité hodnoty s lineárním rozložením. Jedná se například o hodnoty jako jsou rychlost, dráha, objem, věk atp. Tyto hodnoty mohou nabývat obecně nekonečného množství hodnot. Problémem zde je, že různé jednotky jednotlivých atributů mohou výrazně ovlivnit shlukovou analýzu. Jako příklad lze uvést převod jednotek vzdálenosti z kilometrů na metry atp. Z těchto důvodů se zde používá standardizace atributů, která se snaží všem atributům přiřadit stejnou váhu. Standardizace se provádí tak, že původní hodnoty převedeme na bezjednotkové proměnné. Jedním z možných způsobů jsou:

1. **Výpočet střední odchylky** *(mean standard deviation)*

$$s_f = \frac{1}{n}\left(|x_{1f} - m_f| + |x_{2f} - m_f| + ... + |x_{nf} - m_f|\right), \tag{3.1}$$

   kde $x_{1f}, x_{2f}, ..., x_{nf}$ jsou hodnoty proměnné f a m_f je střední hodnota f.

2. **Výpočet z-score**

$$z_{if} = \frac{x_{if} - m_f}{s_f} \tag{3.2}$$

Standardizace nemusí být ve všech případech užitečná, proto je její použití vždy nutné zvážit dle konkrétního případu. Pro výpočet vzdáleností mezi objekty i a j dimenze p, kde $i = (x_{i1}, x_{i2}, ..., x_{ip})$ a $j = (x_{j1}, x_{j2}, ..., x_{jp})$ se nejčastěji používají následující funkce:

- **Euklidovská vzdálenost**

$$d(i,j) = \sqrt{|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + ... + |x_{ip} - x_{jp}|^2} \tag{3.3}$$

- **Manhattanovská vzdálenost**

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + ... + |x_{ip} - x_{jp}| \tag{3.4}$$

- **Minkowského vzdálenost**

$$d(i,j) = (|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + ... + |x_{ip} - x_{jp}|^q)^{\frac{1}{q}} \tag{3.5}$$

3.1.2 Binární proměnné

Jedná se o proměnné, jenž nabývají pouze hodnot 0 nebo 1. Jako příklad si lze uvést proměnnou vyjadřující, zdali člověk kouří či nekouří. Binární proměnné se dělí na dva typy [3]:

- **Symetrické binární proměnné** – u tohoto typu proměnných mají oba stavy (0 nebo 1) stejnou váhu. Může se jednat například o již zmíněný atribut, který rozděluje osoby na kuřáky a nekuřáky. Pro určení podobnosti dvou objektů na základě těchto proměnných se používá koeficient shody:

$$d(i,j) = \frac{r+s}{q+r+s+t}, \tag{3.6}$$

  kde r značí počet proměnných, které mají hodnotu *1* pro objekt j a hodnotu *0* pro objekt i. Naproti tomu s značí počet proměnných, které mají hodnotu *0* pro objekt j a hodnotu *1* pro objekt i. Hodnota q značí počet proměnných s hodnotou *1* pro objekt oba objekty i a j, naproti tomu hodnota t značí počet proměnných s hodnotou *0* pro oba objekty i a j. Stejné označení bude použito i u druhého typu binárních proměnných.

- **Asymetrické binární proměnné** – u tohoto typu proměnných nejsou stavy *0* a *1* stejně významné. Jedná se o takové případy, kdy většina zkoumaných objektů má stejnou hodnotu dané proměnné a jen určitá část má hodnotu rozdílnou. Jako příklad lze uvést aktuální stav zaměstnanosti, kdy většina lidí je zaměstnaná a pouze určitá část je nezaměstnaná. Informace o nezaměstnanosti je pak pro nás významnější než informace, že daný člověk je zaměstnán. Významnější výsledek se obvykle označuje hodnotou *1*. Pro určení podobnosti *asymetrických binárních proměnných* se většinou používá *Jaccardův koeficient* dle rovnice 3.7, která se od předchozí rovnice liší v tom, že neuvažuje počet negativních shod t.

$$d(i,j) = \frac{r+s}{q+r+s} \tag{3.7}$$

3.1.3 Nominální proměnné

Nominální proměnné představují zobecnění binárních proměnných a mohou nabývat více než dvou hodnot. Příkladem takové proměnné může být typ zaměstnání (soustružník, ekonom, právník atp.). Vzdálenost dvou objektů s takovýmito proměnnými pak určujeme dle koeficientu shody:

$$d(i,j) = \frac{p-m}{p}, \tag{3.8}$$

kde m je počet shod, tj. počet proměnných, kde mají objekty i a j stejnou hodnotu a p je celkový počet proměnných.

3.1.4 Ordinální proměnné

Tento typ proměnných vychází z nominálních proměnných. Jediným rozdílem je, že hodnoty, kterých nabývají jednotlivé proměnné objektu, jsou uspořádány v určitém pořadí. Jako příklad si lze uvést proměnnou určující obtížnost kurzu (začátečník, středně pokročilý, pokročilý atp.). Pro určení podobnosti dvou objektů popsaných tímto typem proměnných

se používá faktu, že ordinální proměnné lze zpracovávat jako intervalové proměnné. Pakliže proměnná f nabývá hodnot, jejichž počet je roven hodnotě M_f, lze tyto hodnoty nahradit číselnými hodnotami r_{if} z intervalu <1;M>. Vzhledem k tomu, že M bude pro většinu proměnných různé, využívá se zde transformace rozsahu jednotlivých proměnných do intervalu <0;1>dle vztahu:

$$z_{if} = \frac{r_{if} - 1}{M_f - 1} \tag{3.9}$$

Po této transformaci můžeme s proměnnými vyjádřenými pomocí z_{if} pracovat jako s intervalovými proměnnými a využít některou ze vzdálenostních funkcí.

3.1.5 Poměrové proměnné

Mezi tento typ proměnných řadíme proměnné měřené na nelineární stupnici, jako je například exponenciální stupnice přibližně podle vzorce Ae^{Bt} nebo Ae^{-Bt}, kde A i B jsou kladné konstanty a t typicky reprezentuje čas [4]. Jako příklad si lze uvést růst populace bakterií nebo rozpad radioaktivního prvku. Existuje několik způsobů zpracování takovýchto proměnných. Jedním z nich je pracovat s nimi jako s intervalovými proměnnými, nicméně to sebou nese riziko zkreslení informací. Dalším možným způsobem je použití logaritmické transformace proměnných a s transformovanou hodnotou pracovat opět jako s intervalovou.

3.1.6 Proměnné různého typu

Pro zpracování objektů s proměnnými různých typů můžeme použít techniku, kdy zpracováváme odděleně vždy pouze proměnné stejného typu pomocí metod uvedených v předchozích kapitolách. Takovéto analýzy nicméně obvykle dávají různé výsledky, proto je lepším řešením zpracovat všechny atributy najednou a provést jednu shlukovou analýzu přes všechny proměnné. Máme-li tedy data popsány p atributy různých typů, můžeme vzdálenost objektů i a j určit následovně [3]:

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}, \tag{3.10}$$

kde koeficient $\delta_{ij}^{(f)} = 0$ pokud nastane jeden z následujících dvou případů:

1. Hodnota x_{if} nebo x_{jf}
2. $x_{if} = x_{jf} = 0$ a f je binární asymetrická proměnná

V ostatních případech je $\delta_{ij}^{(f)} = 0$. Příspěvek $d_{ij}^{(f)}$ proměnné f k celkové vzdálenosti objektů i a j se pak počítá podle typů proměnné f takto:

- V případě, že f je binární nebo nominální proměnná pak: $d_{ij}^{(f)} = 0$ jestliže $x_{if} = x_{jf}$, v ostatních případech $d_{ij}^{(f)} = 1$.
- V případě, že f je ordinální nebo poměrová proměnná, pak vypočítáme hodnoty r_{if} a z_{if} a hodnotu z_{if} zpracujeme jako intervalovou proměnnou.
- V případě, že f je intervalová proměnná pak: $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_h\ x_{hf} - min_h\ x_{hf}}$, kde h jde přes všechny hodnoty proměnné f pro jednotlivé objekty.

3.2 Metody shlukování

Shlukové metody můžeme rozdělit do několika skupin na základě použitého algoritmu pro vytváření jednotlivých tříd. Volba dané metody záleží na typu dat, které chceme analyzovat, a také na typu dané úlohy. Jednotlivé skupiny jsou rozebrány v následujících kapitolách.

3.2.1 Metody založené na rozdělování

Metody rozdělují analyzovaná data do určitého předem daného počtu tříd. Jedná se o iterativní metody, kdy jsou data v prvním kroku rozdělena do příslušného počtu tříd. Posléze jsou iterativně tyto třídy upravovány tak, aby podobnost objektů v rámci jednotlivých tříd rostla a zároveň podobnost objektů různých tříd klesala.

Mezi nejpoužívanější metody, založené na rozdělování, patří metoda založená na centrálním bodu (*k-means method*) a metoda založená na reprezentujícím objektu (*k-medoids method*). Vzhledem k tomu, že metoda *k-means* je významnou měrou používána ve výsledné práci, rozeberu ji zde podrobněji.

K-means

Metoda má jako vstupní parametr počet požadovaných shluků q a pracuje na principu rozdělení množiny n objektů právě do q shluků (tříd). V rámci každého takovéhoto shluku (třídy) jsou si objekty podobné, naproti tomu mezi objekty různých shluků je podobnost minimální. Každý shluk je reprezentován pomocí fiktivního centrálního bodu, jehož atributy jsou určeny jako střední hodnota hodnot atributů objektů, které byly přiřazeny do daného shluku (třídy). Příslušnost objektů do takovýchto shluků je pak dána vzdáleností od jejich fiktivního centrálního bodu (bod přísluší do shluku s nejmenší vzdáleností). Metoda pak pracuje iterativně. V rámci jednotlivých iterací se přeskupují body mezi jednotlivými shluky, přičemž tyto iterace se končí, pokud funkce použitá jako kritérium začne konvergovat. V praxi se však používá postup, že iterace se ukončí tehdy, nedojde-li už v dané iteraci k žádnému přesunu objektu. Tento algoritmus funguje dobře, pokud v datech existují kompaktní shluky. Často se stává, že metoda nenajde nejoptimálnější řešení, ale uvízne v lokálním minimu. Existuje několik různých variant této metody, které se liší zejména inicializací, případně způsobem výpočtů centrálních bodů. Jedna s takovýchto modifikací je použita ve vlastním algoritmu, jenž je hlavním tématem práce. Tato modifikovaná metoda k-means je rozebrána podrobněji v podkapitole 5.2.

3.2.2 Hierarchické metody

Jedná se o metody založené na tvorbě hierarchické struktury jednotlivých shluků. Dle toho, jak vytváříme danou strukturu shluků, rozdělujeme tyto metody na *shlukující hierarchické metody* a *rozdělující hierarchické metody*. Existuje několik takových metod například AGNES (shlukující hierarchická metoda) a DIANA (rozdělující hierarchická metoda).

3.2.3 Metody založené na hustotě

Tyto metody vytváří shluky tam, kde je velká hustota jednotlivých objektů. Množství objektů, které je požadováno, aby byla oblast označena za hustou, je dáno parametrem příslušné úlohy, který se odvíjí od konkretní úlohy. Mezi nejznámější metody, pracujícími na tomto principu, patří DBSCAN a DENCLUE.

3.2.4 Metody založené na mřížce

Tyto metody rozdělí prostor objektů na konečný počet buněk, které posléze tvoří mřížkovou strukturu. Shlukování je následně prováděno nad touto mřížkovou strukturou. Tento princip je využíván metodami WaveCluster a STING.

3.2.5 Metody založené na modelech

Tyto metody se snaží najít shodu mezi datovou množinou a matematickým modelem. Metody pak vytváří shluky, jejichž tvar odpovídá co nejvíce daným modelům. Mezi tyto metody patří například metoda konceptuálního shlukování a Expectation - Maximization.

3.2.6 Metody pro shlukování vysoce-dimenzionálních dat

Tyto metody jsou založené na principu úpravy dat před vlastním shlukováním. Je to dáno tím, že velké množství dimenzí v sobě skýtá velké množství potenciálních problémů. Například pouze určité dimenze jsou relevantní pro jednotlivé shluky, velký počet dimenzí způsobuje větší rozptýlení dat, tedy těžší hledání shluků atp. Řešením je tedy předzpracování takovýchto dat. Používá se metoda *transformace rysů*, které transformují vysoce-dimenzionální prostor do prostoru s menším počtem dimenzí a metoda *výběr atributů*, které vybírají pouze relevantní atributy pro danou úlohu. Tyto principy jsou využívané například metodami CIIQUE a PROCLUS.

Kapitola 4

Proudy dat

V posledních letech došlo k obrovskému rozvoji hardwarových a softwarových technologií a k obrovskému rozmachu internetu, mobilních telefonů atp. Toto sebou nese neustále se zvětšující tlak na zpracování proudových dat. *Proudem dat* chápeme potenciálně nekonečný datový tok, který do systému přichází a zároveň z něj může odcházet. Rychlost tohoto toku se v čase může měnit. Jako příklad můžeme uvést prohlížení webových stránek, kreditní operace, provoz na síti, vyhledávaná slovních spojení ve vyhledávačích atp.

Vzrůstá tedy neustále množství proudových dat a s tím se zákonitě zvětšuje i tlak na jejich efektivní zpracování a dolování zajímavých a relevantních informací z těchto dat. Dolování v proudu dat je v dnešní době díky své důležitosti obrovskou výzvou pro mnoho výzkumných skupin.

4.1 Dolování v proudu dat

Dolování v proudu dat se od klasického dolování nad relační databází liší v několika směrech:

- Některá, případně všechna data, nejsou dostupná pro náhodné čtení z pevného disku či paměti, ale přichází z jednoho čí více datových proudů.
- Datový proud má potenciálně neomezenou velikost.
- Systém nemá kontrolu nad pořadím příchozích datových elementů proudu.
- Rychlost datového proudu se v čase mění a může dosahovat místy extrémních hodnot.
- Elementy datového proudu jsou dostupné pouze v čase příchodu.

4.1.1 Omezení

Zpracování a dolování v proudových datech sebou nese několik omezení plynoucích z výše uvedených vlastností. Jednotlivá omezení [5]:

- Při zpracování většího množství dat není možné tato data zpracovávat víceprůchodově. Většina algoritmů by tedy měla datový tok zpracovávat jednoprůchodově.
- Vzhledem k tomu, že zpracováváme data, která se postupně vyvíjí v čase, musí být algoritmy navrženy tak, aby byly schopné zpracovávat takto se vyvíjející data. Čas je přirozenou součástí většiny dolovacích algoritmů u proudů dat.

- Algoritmy musí respektovat hardwarové limity zařízení, jenž provádí zpracování datového toku. Dochází zde ke kompromisu mezi přesností a hardwarovou náročností.

4.2 Předzpracování proudu dat pro dolování

Podobně jako u dolování v klasických datech se i v dolování v proudových datech používají některé techniky předzpracování, které data upravují do podoby vhodné pro dolování. Tyto techniky se dají obecně rozdělit do dvou skupin, a to na *techniky založené na datech* a *techniky založené na úkolech* [6].

4.2.1 Techniky založené na datech

Jedná se o techniky, které pracující na principu snížení velikosti zpracovávaného toku dat, proto je vhodné tyto techniky použít při zpracování vysoce-rychlostního proudu dat. Využívají k tomu techniky sampling, load shedding, agregation a sketches. Tyto techniky jsou rozebrány v následujících podkapitolách.

Sampling (vzorkování)

Technika vybírá náhodně vzorky z daného proudu dat, které jsou posléze zpracovány. Tím je výrazně usnadněno zpracování i velmi rychlých datových toků. Technika má i své nevýhody a omezení. Hlavním problémem vzorkování proudových dat je jejich proměnlivá velikost v čase. Dále tato technika není vhodná pro předzpracování proudů dat, jehož hodnoty značně kolísají, neboť pak dochází při vzorkování k velkým chybám.

Load Shedding

Technika podobná vzorkování. Z datového toku odstraňuje vždy určité části toku. Je vhodná pro systémy, které jsou náchylné na dramatické výkyvy v množství dat, která se mají zpracovávat. Patří sem například systémy zpracovávající HTTP požadavky, analyzující síťový přenos atp.

Sketches

Sketches je technika, která se používá pro sumarizaci vlastností datového toky za použití malého množství paměti. Na rozdíl od předchozích metod bere v úvahu všechny vzorky datového proudu. Používá se při aproximaci frekvenčního momentu vstupní množiny dat S. Frekvenční moment sekvence elementů S, jenž nabývají hodnot z univerza $U = \{1, 2, .., n\}$, je dán vztahem 4.1 [4].

$$F_k = \sum_{i=1}^{n} \left(m_i^k \right), \tag{4.1}$$

kde m_i značí počet prvků množiny S, které nabývají hodnoty i z univerza U a $k \geq 0$. F_0 je tedy rovno počtu odlišných prvků v sekvenci, F_1 je roven délce sekvence, F_2 se nazývá *Giniho index homogenity.*
Frekvenční momenty mají tu vlastnost, že velmi dobře indikují stupeň zkreslení v datech dané sekvence, což je jeden z hlavních požadavků paralelních databázových aplikací.

Hlavním problémem této metody je velikost domény n. Pokud bychom nezavedli žádnou aproximaci, je nutné si pro každou hodnotu $i \in n$ pamatovat hodnotu m_i. Nicméně víme, že při zpracování proudových dat máme výrazné omezení v množství použitelné paměti. Je-li tedy množství paměti menší než velikost domény n, je nutné použít tzv. přehledy, kterým se říká v oblasti frekvenčních momentů *sketches*. Tyto vytváří souhrny pro tzv. distribuční vektor za použití náhodné lineární projekce nad základními vektory. Sketches poskytují garanci kvality aproximované odpovědi např. odpověď na daný dotaz je 12 ± 1. Máme-li N prvků vstupní posloupnosti nabývajících hodnot n hodnot, pak tato technika dokáže aproximovat F_0, F_1 a F_2 do prostoru o složitosti $O = (log\ N + log\ n)$.

Agregace

Tato technika předzpracovává data pomocí agregačních funkcí např. medián, průměr atp. Tato data jsou pak použita pro vlastní dolovací algoritmy. Technika nicméně není vhodná pro všechny druhy proudových dat, problémy má zejména při zpracování neseřazených dat, kdy výrazně roste doba jejich zpracování, proto není vhodná pro proudy dat s kolísavými hodnotami.

4.2.2 Techniky založené na úkolech

Jedná se o techniky, které modifikují stávající techniky, případně se jedná o úplně nové metody. Cílem těchto metod je řešit výpočetní problémy spojené s zpracováním proudu dat.

Posuvné okno

Posuvné okno (angl. sliding window) je metoda, která nezpracovává veškerá data, případně jejich vzorek, ale zpracovává pouze data určité historie. Pakliže se v čase t objeví nový element, tak tento element ztrácí platnost po čase w, kde w je velikost *sliding window*. Používá se například u síťových senzorů, kde jsou nedávná data pro dolování nejdůležitější.

4.3 Algoritmy pro dolování v proudu dat

V následujících podkapitolách je uveden stručný rozbor jednotlivých dolovacích úloh vztažených k proudům dat. Jsou zde popsány hlavní rozdíly oproti metodám pracujícím nad klasickými daty.

4.3.1 Klasifikace, predikce

V podkapitole 2.4.4 byl uveden stručný popis klasifikace a predikce nad obecnými daty. Jak již bylo řečeno, klasifikace se sestává ze dvou kroků a to učení, kdy vytváříme výsledný model dat schopný klasifikovat neznámá data, a testování, kdy dochází k testování takto vytvořeného modelu.

Klasifikace nad proudem dat se liší od klasifikace nad statickými daty výhradně v prvním kroku. Metody, navržené pro statická data, pracují mnohdy víceprůchodově, proto není možné tyto metody použít nad proudem dat. Algoritmy, používané pro klasifikaci a predikci nad proudy dat, jsou tedy jednoprůchodové. Mezi tyto algoritmy patří například *Hoeffding Tree*, *Very Fast Decision Tree* a LWClass.

4.3.2 Shlukování

Při shlukování je nutné vypořádat se s obdobnými problémy jako při klasifikaci. Shlukování nad proudy dat nám oproti klasickému shlukování přináší tyto výzvy [7]:

- Množství uložených dat musí být redukováno.
- Příchozí data musí být inkrementálně shlukována.
- Změny u existujících shluků je třeba rychle identifikovat. Současně může být vytvořen nový shluk, což sebou nese nutnost odstranění některého ze stávajících shluků.
- Předchozí data musí být vhodně sumarizována tak, aby usnadnila shlukovací mechanismus.

Mezi algoritmy pro shlukování nad proudy dat patří:

- **BIRCH** – tento algoritmus byl navržen pro shlukování nad klasickými daty, nicméně je stavěný pro shlukování velkého množství dat. Tato vlastnost umožňuje jeho využití i ve shlukování proudu dat. U tohoto algoritmu se poprvé objevily dva nové koncepty *micro clustering* a *macro clustering*. Tyto pojmy budou podrobněji rozebrány v podkapitolách 5.1.1 a 5.2. Algoritmus pracuje ve dvou krocích. V prvním kroku algoritmus projde databázi a sestaví datový strom, který obsahuje informace o jednotlivých shlucích. V druhém kroku odstraní uzly stromu, jenž reprezentují odlehlé shluky. Hlavní nevýhodou algoritmu je omezená kapacita jednotlivých uzlů a také to, že algoritmus pracuje dobře pouze u dat, jejichž shluky mají kulovitý tvar, neboť pro výpočet hranic shluku používá poloměr a průměr. Informace pro popis tohoto algoritmu jsem čerpal z [8].
- **STREAM** – jedná se o jednoprůchodový shlukovací algoritmus využívající techniku *K-median*. Algoritmus začíná shlukováním jednotlivých částí proudu, jejichž velikost je dána množstvím dostupné paměti, do $2k$ bodů. Posléze, až má dostatek takovýchto bodů na druhé úrovni, algoritmus provede shlukování do $2k$ bodů třetí úrovně. Tento proces se opakuje dle požadovaného počtu úrovní. Na závěr dojde ke shlukování $2k$ shluků do k shluků. Informace o algoritmu jsem čerpal z [6].
- **CluStream** – algoritmus je založen na spojení ideí z algoritmů STREAM a BIRCH [8]. Algoritmus bude podrobně popsán v kapitole 5.

Volba algoritmu

Pří výběru algoritmu pro řešení vlastní práce rozhodovaly již známé výsledky experimentování s těmito algoritmy. Algoritmus CluStream dosahoval nejlepších výsledků z výše uvedených algoritmů, proto byl vybrán pro řešení práce. Výhody tohoto algoritmu jsou [9]:

- Možnost vytvářet vysokoúrovňové shluky pro historii dat libovolné délky.
- Dosahuje nejvyšší přesnosti a rychlosti shlukování z uvedených algoritmů.

4.3.3 Frekventované vzory

V podkapitole 2.4.2 byl uveden stručný popis frekventovaných vzorů, zejména pak pojmů *podpora* a *spolehlivost*. V oblasti proudů dat předkládá tento typ dolovací úlohy ze všech uvedených nejvíce problémů (výzev). Důvody jsou [5]:

- Při hledání frekventovaných vzorů je potřeba prohledávat prostor s exponenciálním počtem vzorů. Důsledkem toho může být množina výsledných frekventovaných vzorů velmi rozsáhlá. Dále také generování takovéto množiny může spotřebovat obrovské množství paměti. Je proto tedy nutné, aby algoritmy byly při práci s pamětí co nejvíce efektivní.
- Zjišťování, zdali daný vzor je frekventovaný, je výpočetně velmi náročné. Při rychlejším proudu dat je tedy nutné udržet tempo s tímto proudem.

Vzhledem k výše uvedenému zde hraje významnou roli přesnost aproximace skutečných výsledků. Více než u ostatních úloh se zde projevuje kompromis mezi požadovanou přesností a potřebnou pamětí, případně výpočetním výkonem. Zvětšení přesnosti sebou nese výrazné zvýšení těchto nároků. Na základě toho musí algoritmy pro dolování frekventovaných vzorů poskytnou uživateli možnost nastavit požadovanou přesnost. Mezi tyto algoritmy patří například *KPS algoritmus* a *Lossy Counting algoritmus*.

Kapitola 5

Algoritmus CluStream

Algoritmus CluStream je inkrementální algoritmus pro shlukování nad proudy dat. Proces vlastního shlukování se rozděluje do dvou částí *online Micro-clustering* a *offline Macro-Clustering*, které budou vysvětleny v následujících podkapitolách. Dále zde bude uveden podrobný popis algoritmu, spojený s vysvětlením ostatních pojmů, potřebných pro pochopení práce algoritmu.

5.1 Online část

Tato část algoritmu se stará o sledování, ukládání a zpracování proudů dat tak, aby výsledky mohla použít *offline část*.

Definice 5.1.1. [5] *Micro-cluster* pro množinu *d-dimensionálních* bodů X_{i_1}, X_{i_2}, .. , X_{i_n} s časovými razítky T_{i_1}, T_{i_2}, .., T_{i_n} je $(2 * d + 3)$ n-tice $\left(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n\right)$, kde $\overline{CF2^x}$ a $\overline{CF1^x}$ odpovídají vektoru d záznamů. Definice každé z těchto položek je následující:

- $\overline{CF2^x}$ – jedná se o vektor sum druhých mocnin hodnot jednotlivých dimenzí všech vektorů, kde p-tý prvek prvek je tedy roven $\sum_{j=1}^{n} \left(x_{i_j}^p\right)^2$.
- $\overline{CF1^x}$ – jedná se o vektor sum hodnot jednotlivých dimenzí všech vektorů, kde p-tý prvek prvek je tedy roven $\sum_{j=1}^{n} x_{i_j}^p$.
- $\overline{CF2^t}$ – jedná se o sumu druhých mocnin hodnot všech časových razítek T_1,T_2, .., T_n.
- $\overline{CF1^t}$ – jedná se o sumu hodnot všech časových razítek T_1,T_2, .., T_n.
- n – počet zpracovávaných vektorů.

5.1.1 Uložení Micro-clusterů

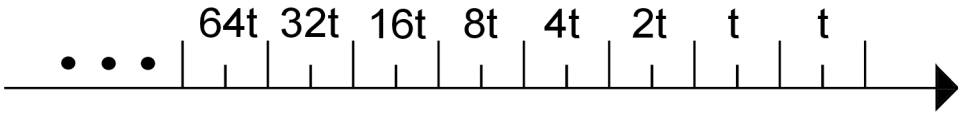
Micro-clustery jsou ukládány v podobě snímků vázaných na určitý čas. Při ukládání těchto snímků vychází volba časového intervalu mezi jednotlivými snímky z předpokladu, že macro-clustering komponenta, která je součástí offline analýzy, a bude popsána v podkapitole 5.2, vždy bude vyžadovat tvorbu vysoce-úrovňových shluků na základě historie předem dané délky a zároveň pro starší data nebude potřeba taková přesnost. Uvažujme například, že aktuální čas je t_c a uživatel chce nalézt shluky na základě historie délky h. Komponenta

macro-clustering tedy vezme snímky v čase t_c a $(t_c - h)$ a využije aditivních vlastností micro-clusterů pro vytvoření vysoce-úrovňových shluků na základě dané historie h. Nicméně zde existuje problém, že není možné ukládat snímky v každém časovém okamžiku, existuje proto několik způsobů jak snímky ukládat.

1. **Přirozené časové rámce** – časová osa rozdělena na několik rámců, různých úrovní granularity dle přirozeného výběru tj. výběru dle příslušné aplikace např. hodiny, týdny, měsíce atp. Ukázka je uvedena na obr. 5.1.

Obrázek 5.1: Přirozené snímkování [5]

2. **Logaritmické časové rámce** – časová osa opět rozdělena na několik rámců, nicméně úroveň granularity v jednotlivých rámcích je určena logaritmickým měřítkem. Ukázka na obr. 5.2.



Obrázek 5.2: Logaritmické snímkování [5]

3. **Pyramidové časové rámce** – tato metoda odstraňuje nevýhodu předchozí metody, kterou je velký skok mezi po sobě jdoucími úrovněmi granularity. Metoda sice potřebuje pro uložení snímků více paměťového prostoru než předchozí metoda, nabízí nicméně mnohem lepší výsledky co se týče úrovně aproximace skutečnému výsledku.

   V této metodě jsou jednotlivé snímky opět uloženy do několika úrovní granularity. Jednotlivé snímky jsou klasifikovány do tzv. *pořadí*, které nabývá celočíselných hodnot od 0 do $log(T)$, kde T je uběhlý čas od začátku sledování daného proudu dat. Toto pořadí pak určuje, do které úrovně granularity patří daný snímek. Jednotlivé snímky jsou v rámci konkrétního *pořadí* uchovávány dle následujícího klíče:

   - Snímky i-tého pořadí jsou ukládány v časových intervalech α^i, kde α je celé číslo a $\alpha \geq 1$. Jedná se tedy o snímky, jejichž čas je dělitelný α^i bezezbytku.
   - V každém pořadí je uloženo posledních $\alpha + 1$ snímků.

   Důležitou vlastností této metody je, že pro libovolnou historii délky h v čase t_c existuje minimálně jeden snímek v čase t_s kde platí, že t_s náleží intervalu $(t_c - 2 * h, t_c - h)$. Platí tedy nerovnice 5.1.

$$t_c - t_s \leq 2 * h \tag{5.1}$$

   Pokud chceme zvýšit přesnost výše uvedené metody, bude v každém rámci uloženo $\alpha^l + 1$ snímků, kde $l > 1$. Pak platí nerovnice 5.2.

$$t_c - t_s \leq (1 + \frac{1}{\alpha^{l-1}}) * h \tag{5.2}$$

   V tabulce 5.1 je uvedena ukázka uložení jednotlivých snímků pro parametry $\alpha = 2$, $l = 2$ a $t_c = 44$.

Pořadí snímků	Čas snímků [%]
0	44 43 42 41 40
1	44 42 40 38 36
2	44 40 36 32 28
3	40 32 24 16 8
4	32 16
5	32

Tabulka 5.1: Příklad uložení snímků

4. **Geometrický časový rámec** Tato metoda je obdobná předchozí metodě. V této metodě jsou jednotlivé snímky klasifikovány do tzv. číselných rámců, jejichž hodnoty jsou v rozsahu 0 až $log_2(T)$, kde T je maximální délka proudu dat. Zde je klíč pro ukládání jednotlivých snímků následující:

   - Snímky i-tého rámce jsou ukládány v časových intervalech 2^i.
   - Maximální počet snímků v jednotlivých rámcích je dán hodnotou *max_kapacity*.

   Máme-li uživatelem definovanou velikost časového okna h v čase t_c, potom, je-li hodnota *max_kapacity* ≥ 2, existuje snímek v čase t_s takový, že platí nerovnice 5.3.

$$\frac{h}{2} \leq t_c - t_s \leq 2 * h \tag{5.3}$$

   Z rovnice 5.3 je vidět, že metoda má podobnou přesnost, jako předchozí metoda. Hlavním rozdílem zde je výrazné snížení redundance v rámci jednotlivých rámců. Zde nedochází k situaci, kdy je jeden snímek uložen ve více rámcích. Pravidla pro ukládání snímků t (v čase t) do jednotlivých rámců jsou následující:

   (a) Pokud $t \ mod \ 2^i = 0$ a $t \ mod \ 2^{i+1} \neq 0$, pak snímek t je vložen do rámce čísla i.

   (b) Chceme-li vložit snímek t do rámce i, jehož počet snímků je roven hodnotě *max_kapacity*, pak odstraníme nejstarší snímek a vložíme snímek t.

   V tabulce 5.2 je uvedena ukázka uložení snímků pro parametry $t_c = 52$ a *max_kapacity* $= 3$.

Číslo rámce	Čas snímků [%]
0	51 49 47
1	50 46 42
2	52 44 36
3	40 24
4	48 16
5	32

Tabulka 5.2: Příklad uložení snímků

5.1.2 Udržování Micro-clusterů

Cílem této části algoritmu je udržování informací s vysokou úrovní granularity a přesnosti, které budou posléze použity v off-line analýze. Tato fáze je nezávislá na uživatelském vstupu (požadovaná přesnost, granularita atp.). Jedná se o iterativní část algoritmu, která udržuje určitý počet micro-clusterů po celou dobu analýzy proudu dat. Počet těchto micro-clusterů je dán konstantou q, která je odvozena od množství paměti, které je k dispozici pro danou analýzu. Typicky je tato hodnota větší než skutečný počet shluků v proudu dat a zároveň výrazně menší než množství objektů v daném proudu dat za určitou dobu. Algoritmus tedy udržuje neustále micro-clustery $\mathcal{M}_1$,$\mathcal{M}_2$, ... , $\mathcal{M}_q$. Každý takovýto micro-cluster má své unikátní *id* číslo. Takto udržované micro-clustery potom tvoří jednotlivé snímky, které jsou v časech α^i $(i \geq 0)$ ukládány na disk pro off-line analýzu. Dále v tento čas dochází také k mazání starých micro-clusterů z disku. Vlastní algoritmus pro udržování micro-clusterů pracuje v několika krocích:

- Na začátku dojde k inicializaci q micro-clusterů. Tohoto je docíleno tak, že se ukládá určitý počet objektů ze začátku proudu na disk. Posléze, až dosáhne tento počet určité hodnoty, inicializuje se q micro-clusterů pomocí metody *k-means*.

- Po vlastní inicializaci dochází již k udržování a aktualizaci jednotlivých micro-clusterů na základě příchozích dat v proudu. Objeví-li se tedy nový bod $\overline{X_{i_k}}$, musí dojít k absorpci tohoto bodu do některého z micro-clusterů, případně k vytvoření nového micro-clusteru, určeného právě tímto příchozím bodem.

- Pro každý micro-cluster $\mathcal{M}_j$ je posléze spočítána vzdálenost jeho centrálního bodu od právě příchozího bodu $\overline{X_{i_k}}$. Dojde tedy k nalezení micro-clusteru $\mathcal{M}_p$, jenž má nejmenší vzdálenost od dobu $\overline{X_{i_k}}$. Dosti často se nicméně stává, že bod $\overline{X_{i_k}}$ nepřísluší do micro-clusteru $\mathcal{M}_p$. Důvody mohou být dva:

  1. Bod $\overline{X_{i_k}}$ je vně hranic nalezeného micro-clusteru $\mathcal{M}_p$.
  2. Bod $\overline{X_{i_k}}$ odpovídá začátku novému shluku z pohledu vývoje daného proudu dat.

- Výše uvedené body nelze bezpečně rozhodnout pouze na základě aktuálního bodu $\overline{X_{i_k}}$, jsou nutné další body proudu. Tento postup ovšem není možný, neboť je nutné se rozhodnout nyní, zdali daný bod $\overline{X_{i_k}}$ náleží micro-clusteru $\mathcal{M}_p$ či nikoliv. Toto se dělá tak, že se zjistí, zdali bod $\overline{X_{i_k}}$ leží uvnitř *maximální hranice* daného micro-clusteru $\mathcal{M}_p$. Pakliže ano, je daný bod přidán do micro-clusteru $\mathcal{M}$. Neleží-li daný bod $\overline{X_{i_k}}$ uvnitř *maximální hranice*, dojde k vytvoření nového micro-clusteru.

- Hodnota *maximální hranice* micro-clusteru, který obsahuje více jak jeden bod, je dána hodnotou směrodatné odchylkou bodů v micro-clusteru. Směrodatná odchylka pro náhodnou veličinu X se počítá dle rovnice 5.4.

$$\sigma = \sqrt{E\left(\left(X - E\left(X\right)\right)^2\right)} = \sqrt{E\left(X\right)^2 - \left(E\left(X\right)\right)^2} \tag{5.4}$$

  Z této rovnice je patrné, proč je struktura micro-clusteru volena tak, jak bylo popsáno v definici 5.1.1. Tato struktura totiž umožňuje vypočítat směrodatnou odchylku pomocí hodnot uchovávaných v micro-clusteru. Podrobnější realizace je uvedena v podkapitole 7.4.1. Obsahuje-li micro-cluster pouze jeden bod, nelze u něj počítat směrodatnou odchylku. V tomto případě je hodnota hranice určena vzdáleností od středu nejbližšího micro-clusteru.

- Vzhledem k tomu, že je nutné udržovat konstantní počet micro-clusterů, při vytvoření nového micro-clusteru musí dojít ke zrušení jednoho ze stávajících micro-clusterů, případně ke spojení dvou micro-clusterů do jednoho.

- Algoritmus nejdříve zjistí, zda-li je možné nějaký micro-cluster odstranit. Pokud ano, dojde ke zrušení tohoto micro-clusteru. Zjištění, zdali je možné odstranit nějaký micro-cluster, probíhá v několika krocích:

  1. V ideálním případě by tento proces fungoval tak, že by algoritmus zjistil průměr časů posledních m bodů jednotlivých micro-clusterů a odstranil by ten s nejnižším průměrem (nejméně aktuální micro-cluster). Toto řešení má nicméně za následek nutnost pamatovat si všechny časy posledních m prvků a tím i větší nároky na paměť. Proto se tento způsob nedá použít. Algoritmus tedy pracuje tak, že pro každý micro-cluster $\mathcal{M}$ spočítá průměr a směrodatnou odchylku časů jednotlivých bodů micro-clusteru $\mathcal{M}$ dle rovnic 5.5 a 5.6.

$$\mu\mathcal{M} = \frac{CF1^t}{n} \tag{5.5}$$

$$\sigma\mathcal{M} = \sqrt{\frac{CF2^t}{n} - \left(\frac{CF1^t}{n}\right)^2} \tag{5.6}$$

  2. Posléze se hledá čas příchodu bodu odpovídajícímu *m/(2*n)-tému* percentilu bodů v $\mathcal{M}$ za předpokladu, že časy jednotlivých příchodů mají normální rozložení.
  3. Tento nalezený čas příchodu potom značí *relevantní razítko* daného clusteru $\mathcal{M}$.
  4. Následně se vybere micro-cluster s nejmenší *relevantním razítkem*. Je-li jeho hodnota nižší jak uživatelem definovaný práh δ, pak je možné tento mikro-shluk odstranit.

- Pokud nelze ani jeden micro-cluster odstranit, dochází ke spojení dvou nejbližších micro-clusterů. Dále se k takto vytvořenému micro-clusteru přiřadí seznam *ids*, obsahující *id* hodnoty spojených micro-clusterů.

5.2 Off-line část

Tato část je založena na tvorbě *macro-clusterů*. Jedná se o vysokoúrovňové shluky, které jsou lépe čitelné uživateli než micro-clustery. Macro-cluster algoritmus využívá pro tvorbu shluků uložené micro-clustery, nepoužívá tedy objemná data z vlastního proudu dat. Toto umožňuje použít pro tvorbu macro-shluků víceprůchodové algoritmy. Do algoritmu pro tvorbu macro-clusterů již vstupuje uživatel, který definuje požadovanou historii h a počet shluků, které chce vytvořit. Princip algoritmu je následující:

- Předpokládejme, že uživatel v aktuálním čase t_c chce vytvořit vysokoúrovňové shluky na základě historie délky h. Algoritmus tedy potřebuje najít micro-clustery uložené v rámci snímku v čase $(t_c - h)$. Jak již bylo řečeno v podkapitole 5.1.1, vždy existuje uložený snímek v čase $(t_c - h')$, kde $h' \doteq h$ v rámci určité tolerance, jenž je daná parametry pyramidového časového rámce.

- Posléze pro každý micro-cluster $\mathcal{M}_{t_c}$, náležící do množiny micro-clusterů v čase t_c, nalezneme seznam *ids*. Posléze se pokusí pro každý prvek seznamu ids nalézt odpovídající micro-cluster $\mathcal{M}_{t_c-h'}$, náležící do množiny micro-clusterů v čase $(t_c - h')$. Pakliže takovýto micro-cluster nalezneme, odečteme *CF* vektory micro-clusteru $\mathcal{M}_{t_c}$ a $\mathcal{M}_{t_c-h'}$. Tímto se zajistí, že micro-clustery vytvořené dříve, než je požadovaná historie, nebudou dominovat ve výsledku shlukování. Takto vytvořenou množinu micro-clusterů označíme $\mathcal{N}(t_c, h')$.

- Nad množinou $\mathcal{N}(t_c, h')$ je potom spuštěn modifikovaný algoritmus *k-means*, jenž vytvoří požadované vysokoúrovňové shluky. Modifikace algoritmu *k-means* spočívá v těchto odlišnostech [9]:

  - Vzhledem k tomu, že tvorba shluků je prováděna nad micro-clustery, kde každý obsahuje určitý počet bodů, je potřeba tuto skutečnost zohlednit při výběru inicializačních micro-clusterů. Ty jsou tedy vzorkovány s pravděpodobností úměrnou počtu bodů v daném micro-clusteru. Micro-cluster s větším počtem bodů bude tedy vybrán jako inicializační s větší pravděpodobností než micro-cluster s menším počtem bodů.
  - Vzdálenost mezi fiktivním centrálním bodem shluků a micro-clustery je počítána jako vzdálenost tohoto bodu od středu micro-clusteru.
  - Při výpočtu nových fiktivních centrálních bodů je použit výpočet váhových středů daných micro-clusterů.

Kapitola 6

Návrh řešení

Kapitola se věnuje vlastnímu návrhu řešení implementace algoritmu *CluStream* v prostředí MS SQL Serveru. V první části jsou popsány použité technologie potřebné pro vlastní realizaci. V další části je rozebrán vlastní návrh výsledné aplikace v podobě diagramu balíčků a tříd.

6.1 Použité technologie

V následujících podkapitolách budou popsány jednotlivé technologie, které jsou důležité pro pochopení návrhu řešení.

6.1.1 .NET Framework

Jedná se o softwarový framework určený pro OS Microsoft Windows sloužící pro vývoj webových a desktopových aplikací. Vyšel již v několika verzích počínaje verzí .NET framework 1.0 až po verzi .NET Framework 4.5. Všechny tyto verze jsou zpětně kompatibilní, což znamená, že v případě vytvoření aplikace ve verzi 1.0 je zajištěno, že nám pojede i s využitím vyšších verzí.

.NET podporuje řadu programovacích jazyků jako C#, VB .Net, C++, J#, Pascal a podobně. Proces kompilace námi napsaného programu (knihovny) je obdobný Javě. Program je pomocí překladače přeložen do mezikódu IL (obdoba Byte kódu v Javě). Výsledkem tohoto překladu je buď spustitelný *exe* soubor nebo knihovna *dll*. Tomuto souboru se také říká Assembly a je to minimální jednotka v .NET, která může být dále používaná v jiných programech v .NET. Takto zkompilovaný soubor (knihovna) je potom přenositelný na jakýkoliv počítač, kde je nainstalován .NET framework (v tom spočívá jeho nezávislost na hardware) a zde může být spuštěn pomocí CLR, který jej přeloží do strojového jazyka příslušného počítače a vykoná.

6.1.2 Microsoft SQL Server

Jedná se o relační databázový server vytvořený společností Microsoft. Obdobně jako u všech relačních databázových serverů je i zde hlavním cílem uchovávání informací a poskytnutí rozhraní pro přístup k nim pomocí dotazů z různých aplikací. Zde se používá jako primární dotazovací jazyk T-SQL. Microsoft SQL Server umožňuje ukládat data ze strukturovaných dokumentů, ale i nestrukturovaných jako jsou například obrázky a multimediální soubory.

Tento produkt také nabízí spoustu dalších služeb. Lze zde uvést například možnosti synchronizace, vyhledávání, generování sestav a vytváření analýz. Historie tohoto nástroje padá až do roku 1989, kdy vznikla první verze tohoto systému. V současné době je nejaktuálnější verze z roku 2010 *SQL Server 2008 R2*.

CLR Integration

Jedná se o prostředí pro běh .NET aplikací, které je hostované na Microsoft SQL Serveru. Můžeme tak přímo definovat databázové procedury, triggery atp. přímo v *managed kódu* běžícím v rámci CLR. Managed kód se liší od klasického mezikódu tím, že využívá CAS (Code Access Security), který zabraňuje jednotlivým assemblies vykonání některých operací plynoucí z toho, že kód běží na databázovém serveru.

Tato technologie se využívá zejména v případech, kde by kód v T-SQL byl nepřehledný, případně nerealizovatelný. Této technologie bude dále využito pří návrhu řešení vlastní práce.

6.1.3 Microsoft StreamInsight

Tato technologie není přímo využita ve výsledné aplikaci, neboť se v průběhu implementace ukázala oproti prvotním předpokladům jako nevhodná, nicméně je na ni několik odkazů v textu, a proto ji zde ve stručnosti popíši. Jedná se o platformu, která slouží k vývoji aplikací, jenž zpracovávají proudová data z mnoha vstupů, dělají nad nimi potřebné operace a výsledky dále poskytují. Platforma je založená na .NET frameworku a poskytuje nám rozhraní pro tvorbu aplikací pro efektivní zpracování událostí (proudů dat). Zdrojem událostí (dat) mohou být databázové systémy, různé senzory, webové aplikace atp. Filtrace a výběr požadovaných událostí, spojování více proudu atp. je zde prováděno pomocí jazyka *LINQ*. Syntaxe tohoto jazyka je velmi podobná jazyku SQL. Více o této platformě můžete nalézt na [10].

6.2 Návrh řešení

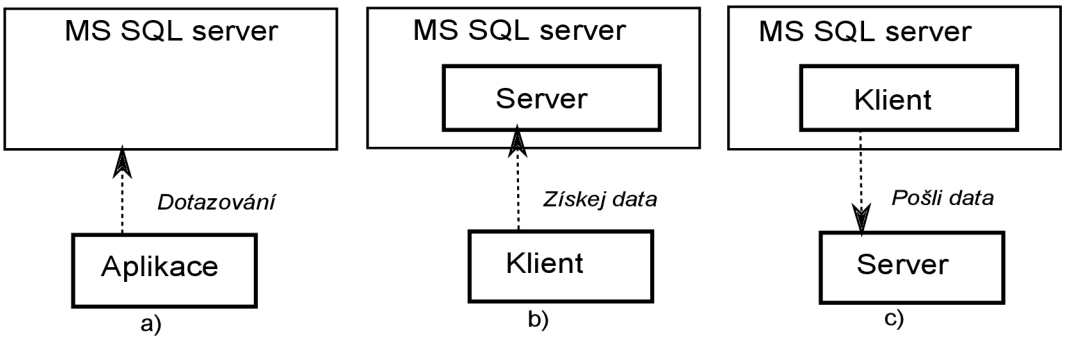
Návrh řešení vychází z předpokladu, že proudem dat nad MS SQL jsou příchozí transakce, které vkládají nová data do jednotlivých tabulek. Zpracování takovéhoto proudu dat pro účely implementace algoritmu *CluStream* lze třemi způsoby, jenž jsou uvedeny na obrázku 6.1. Popis jednotlivých možností (výhody, nevýhody) je uveden zde:

- **Varianta a** – Tato varianta jako jediná nevyužívá prostředí CLR Integration. Aplikace pracuje na principu dotazování v určitém intervalu nad konkrétní tabulkou (pohledem), nad jehož daty probíhá vlastní algoritmus. Výhodou tohoto řešení je, že není potřeba nasazovat na MS SQL server žádný kód, který by hlídal příchozí transakce, čímž se nezpomalí jejich vykonávání. Dále pak fakt, že pro pouhé dotazování nad danou tabulkou (pohledem) jsou většinou potřeba menší oprávnění, než pro následující dvě varianty, jenž využívají *assembly* nasazovaných na MS SQL server. Nicméně tato varianta má jednu obrovskou nevýhodu, a to předem neznámou rychlost příchodu transakcí (proudu dat). Není tedy možné předem určit množství transakcí, jenž proběhly od posledního dotazu. Musíme tedy vždy přenést veškeré informace z dané tabulky (pohledu) a uvnitř aplikace poznat pouze ty nové nebo doplnit do databáze sloupec, jenž by označoval již zpracované transakce (řádky). První varianta

je neakceptovatelná při běžných velikostech tabulek, které obsahují tisíce řádků, neboť by bylo přenášeno velké množství informace. Druhá varianta sebou nese nutnost změny struktury databáze za běhu, což je také velmi nevýhodné. Vzhledem k těmto problémům nebyla tato varianta vybrána pro výslednou implementaci.

- **Varianta b** – Z uvedených varianta tato varianta využívá prostředí CLR Integration nejvíce. Jedná se o architekturu, kde většina logiky algoritmu CluStream pracuje přímo v prostředí MS SQL serveru. Klientem je v tomto případě pouze GUI aplikace, která dotazováním získává z tohoto serveru potřebná data, která vyžaduje uživatel. Výhodou tohoto řešení je možnost práce algoritmu *off-line*, bez potřeby přenosu dat po sítí a také nižší výkonové nároky na straně klienta. Nevýhodou je zde nicméně přenesení veškeré výpočetní náročnosti na stranu MS SQL serveru.

- **Varianta c** – Jedná se o variantu, jenž je kompromisem mezi předchozími dvěma variantami. Odstraňuje jejich hlavní nevýhody (potřeba dotazování, velká zátěž na straně serveru), a proto byla vybrána pro výslednou implementaci. Klientská část zde běží na serveru a posílá data serverové části, jenž běží lokálně na konkrétním počítači. Detailní návrh tohoto řešení je uveden v následující podkapitole.



Obrázek 6.1: Možná řešení

6.2.1 Klientská část

Klientská část je realizována pomocí CLR triggeru, jenž je vytvořen nad konkrétní tabulkou nebo pohledem a který bude odesílat požadovaná data serverové části. Vzhledem k tomu, že algoritmus *CluStream* je určen ke zpracování číselných dat, probíhá zde již první filtrace a na server jsou posílána pouze číselná data. Realizace a sestavení takovéhoto triggeru nad konkrétní tabulkou (pohledem) bude více rozebráno v kapitole 7.2.2.

6.2.2 Serverová část

Tato část tvoří jádro celé aplikace. Struktura této části je uvedena v podobě diagramu balíčků na obrázku 6.2. Popis diagramu je uveden zde:

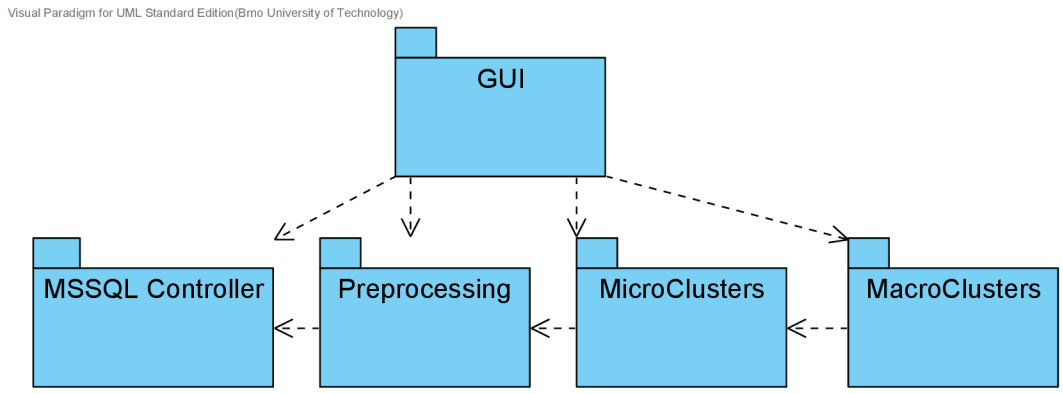
- *MSSQL Controller* – balíček reprezentující třídy, které zajišťují komunikaci s klientskou částí. Jedná se o zpracování vstupních dat a tvorbu CLR triggeru nad konkrétní tabulkou (pohledem).

- *Preprocessing* – třídy balíčku jsou využity k předzpracování vstupních dat dle uživatelského nastavení. V rámci této části aplikace bylo původním záměrem využít také frameworku *StreamInsight Serveru*, jenž je určen pro tvorbu aplikací zpracovávající proudy dat pomocí dotazovacího jazyka LINQ. Toto řešení se nicméně v průběhu implementace ukázalo jako nevhodné. Více je k tomuto problému uvedeno v kapitole 7.3.

- *MicroClusters* – tento balíček tvoří jádro celé aplikace. Balíček obsahuje třídy implementující hlavní část algoritmu CluStream, a to je správa micro-clusterů a jejich ukládání do jednotlivých snímků pomocí pyramidového časového rámce.

- *MacroClusters* – balíček, jehož součástí jsou třídy, které se starají o off-line analýzu dat.

- *GUI* – balíček obsahující třídy uživatelského rozhraní. Slouží k ovládání aplikace a k vizualizaci výsledků shlukování. Umožňuje nastavení všech ostatních součástí aplikace a také jejich ovládání.



Obrázek 6.2: Diagram balíčků

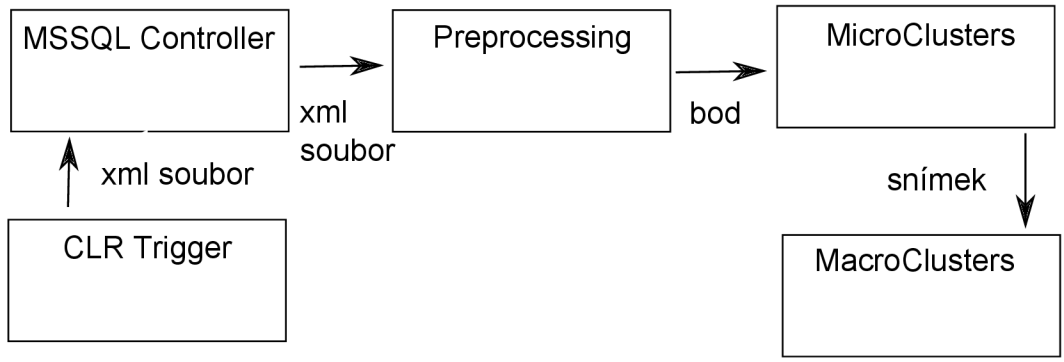
Na obrázku 6.3 je vidět forma jednotlivých dat, která jsou předávána mezi jednotlivými komponentami. Jak je vidět, balíček *MS SQL Controller* nemění strukturu dat, pouze přeposílá příchozí xml soubory balíčku *Preprocessing*. Ten již upravuje tato data do podoby n-dimenzionálního pole hodnot typu double. Tato data jsou zpracovávána balíčkem *MicroClusters*, který pak poskytuje balíčku *MacroClusters* jednotlivé snímky.

6.2.3 Diagramy tříd

V následujících podkapitolách jsou detailněji rozebrány jednotlivé balíčky v podobě diagramů tříd.

6.2.4 Balíček MSSQL Controller

Obrázek 6.4 zobrazuje třídy tohoto balíčku. Balíček má dvě hlavní třídy, a to třídu DatabaseWorker a třídu ControllerServer. Obě tyto třídy využívají rozhraní ListenerMessage, jenž slouží pro komunikaci s vrstvou GUI a pro zobrazení chybových a informačních zpráv uživateli. Toto řešení umožňuje nechat na uživateli tohoto balíčku (knihovny), jak se budou zobrazovat dané zprávy. Třída DatabaseWorker slouží k práci s konkrétní databází na



Obrázek 6.3: Forma komunikace

daném MS SQL serveru. Obsahuje tedy metody pro vyzkoušení spojení s databází, nahrání tabulek a pohledů z dané databáze atp. Druhou hlavní třídou je třída ControllerServer. Tato třída se stará o vlastní zpracování příchozího bodu. K tomu využívá třídu ClientWorker, jenž vždy obslouží daný požadavek v samostatném vláknu. Pro komunikaci s vyšší vrstvou zde byl použit návrhový vzor Observer. Na rozdíl od Javy, která obsahuje přímo definici třídy Observerable případně rozhraní Observer, v .Net nic takového není. Proto je tento návrhový vzor implementován pomocí tříd Customers, Subject a rozhraní Observer.

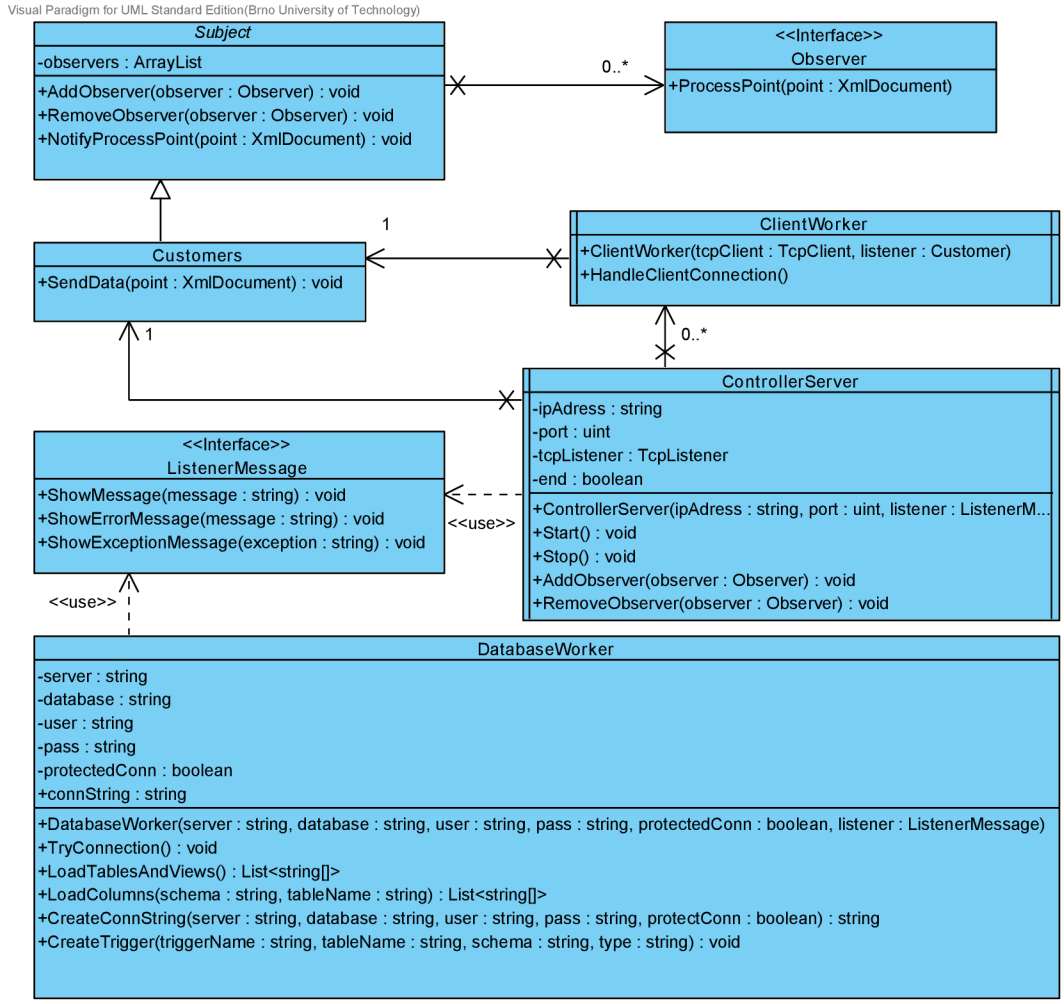
6.2.5 Balíček Preprocessing

Na obrázku 6.5 je zobrazen diagram tříd tohoto balíčku. Hlavní třídou balíčku je třída DataProcessor. Tato třída implementuje rozhraní Observer s balíčku MSSQL Controller a zpracovává tak příchozí body z této vrstvy. Dále tato třída pracuje jako vstupní bod pro komunikaci s balíčkem GUI, jenž implementuje vlastní možnosti nastavení předzpracování. Tento balíček se stará o jednoduché předzpracování dat na základě nastavení od uživatele výsledné aplikace. Využívá k tomu třídu Settings, která v sobě zahrnuje nastavení pro jednotlivé dimenze. V oblasti předzpracování byla implementována možnost odstranění bodů s extrémními hodnotami a možnost zpracovávat data s hodnotami *null*. Proto třída Settings poskytuje metody pro nastavení minima, maxima, zpracování hodnot null a funkci pro resetovaní veškerého nastavení.

V tomto balíčku probíhá také kontrola přijímaných dat, neboť navržený model komunikace umožňuje nasazení triggeru nad libovolnou tabulku a může tedy dojít k situaci, kdybychom přijímali data z jiné tabulky, než která je v tuto chvíli cílem naší analýzy. Dále také může nastat situace, kdy některé požadované sloupce budou v přijímaných datech chybět z důvodu špatného nastavení triggeru. Všechny tyto situace je potřeba odfiltrovat a balíčku MicroClusters posílat pouze kompletní data. Pro komunikaci s tímto balíčkem byl opět použit návrhový vzor Observer, jenž je implementován pomocí třídy Subject a rozhraní Observer.

6.2.6 Balíček MicroClusters

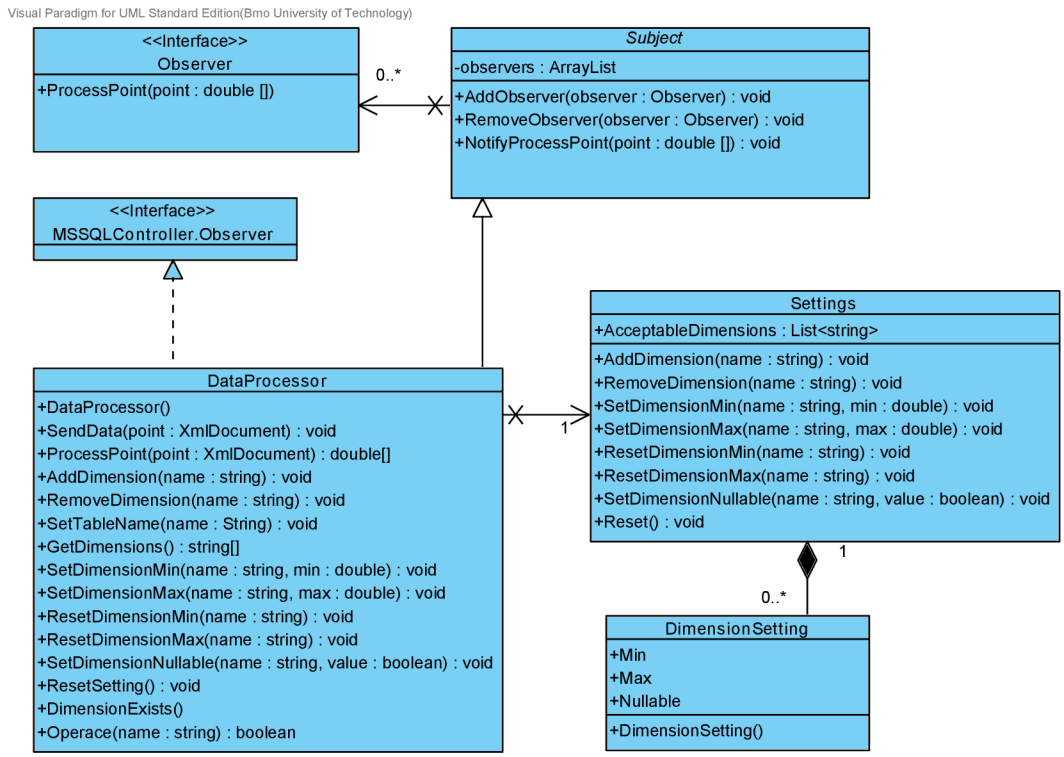
Třídy patřící do balíčku MicroClusters jsou uvedeny na obrázku 6.6. Vzhledem k tomu, že tento balíček tvoří jádro vlastní aplikace, bude zde rozebrán detailněji. Vstupním bodem je třída PointPreprocessor(), jenž přijímá body z balíčku Preprocessing díky implementaci rozhraní Observer. Tyto body předává třídě MicroClustering, jenž implementuje hlavní logiku uchovávání Micro-clusterů. Tato třída má několik důležitých parametrů, jenž jsou stěžejní



Obrázek 6.4: Třídy balíčku MSSQL Controller

pro vlastní funkci algoritmu. Tyto parametry jsou nastaveny v konstruktoru třídy a nemohou být dále měněny. Toto odpovídá logice algoritmu, kdy jsou parametry nastaveny na začátku a nemohou být v průběhu měněny. Volba názvů těchto parametrů vychází z popisu algoritmu v kapitole 5.1.2.

Hlavní metodou třídy MicroClustering je metoda ProcessPoint(). Zde je implementována veškerá logika zpracování nově příchozího bodu dle vlastního algoritmu. Dochází zde tedy k počátečnímu sběru určitého počtu inicializačních bodů, nad nimiž je za využití algoritmu *k-means* (pomocí metody Kmeans()) vytvořeno q inicializačních micro-clusterů. Posléze po příchodu dalšího bodu dochází buď k jeho přiřazení již existujícímu micro-clusteru, nebo k vytvoření nového micro-clusteru. Celý kód této metody je synchronizován pomocí monitorů, neboť se jedná o kritickou sekci implementovaného algoritmu a nemůže dojít ke zpracování dvou bodů zároveň. Vzhledem k tomu, že po inicializaci je nutné již ukládat jednotlivé snímky pro off-line analýzu, je zde metoda Stop(), která explicitně ukončí toto ukládání, neboť tento proces běží za využití časovače ve vlastním vlákně. Poslední metodou této třídy je metoda GetSnapshots(), která získá potřebné snímky pro off-line analýzu na základě historie dané délky. Pro výpočet vzdáleností dvou n-dimenzionálních bodů je zde abstraktní třída Distance, jenž má dvě konkrétní implementace EuklidDistance a ManhattanDistance,



Obrázek 6.5: Třídy balíčku Preprocessing

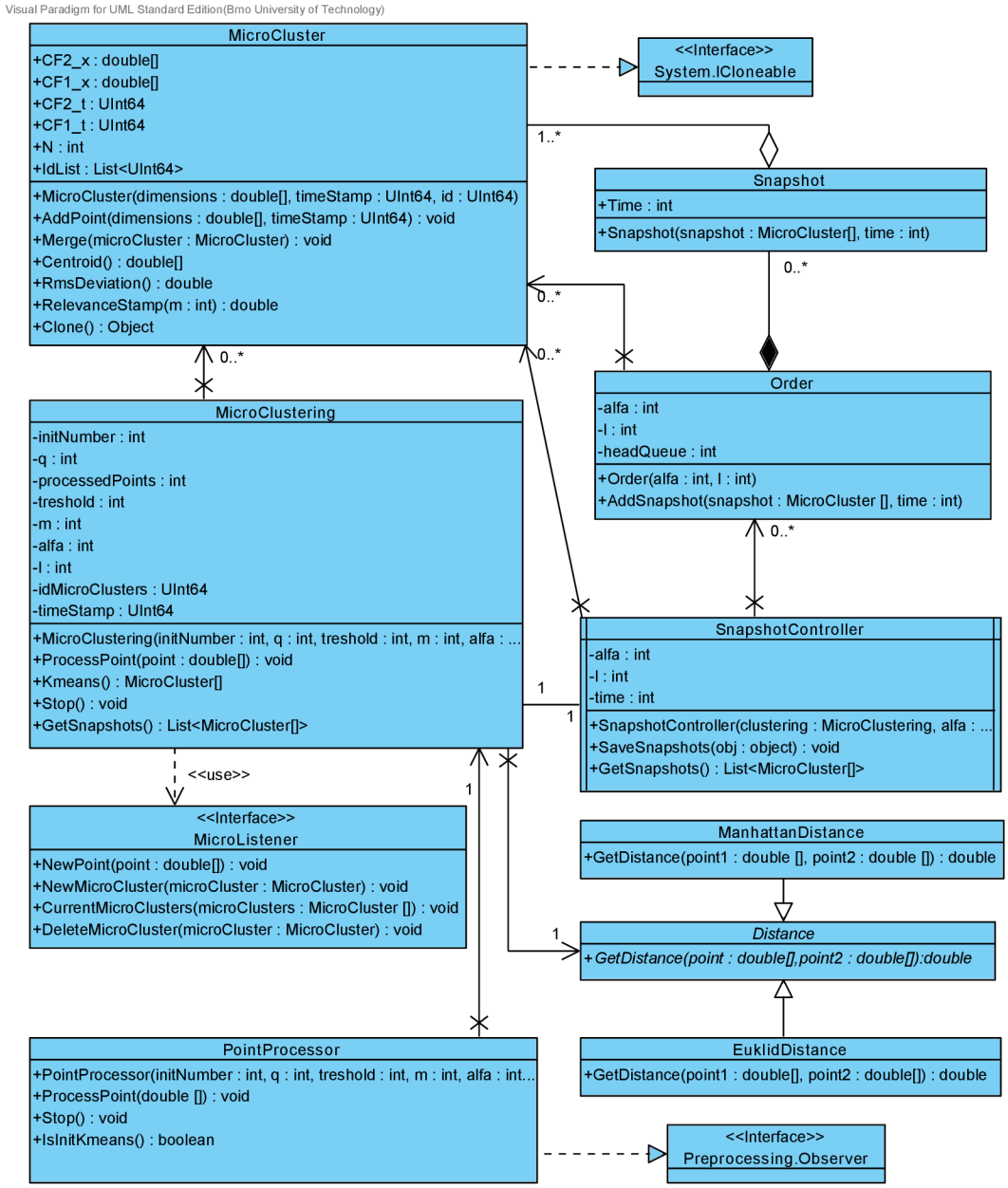
které implementují výpočet Euklidovské a Manhattanovské vzdálenosti. Tento návrh také umožňuje rozšíření aplikace o další možné vzdálenostní funkce.

Pro komunikaci s balíčkem GUI je zde použit koncept rozhraní, které je pak na úrovni tohoto balíčku implementováno. Jedná se o rozhraní MicroListener, které definuje metody, jenž informují o příchodu nového bodu, vytvoření nového micro-shluku atp. Tento návrh umožňuje snadnou použitelnost balíčků (knihoven) algoritmu i v jiných aplikacích (konsolové aplikace atp.), neboť stačí implementovat pouze toto rozhraní na vyšší vrstvě a předat v konstruktoru tuto implementaci. Podporována je i možnost neimplementovat dané rozhraní a předat v konstruktoru hodnotu *null*, nicméně toto použití dává smysl pouze v případě, že nás zajímá pouze off-line analýza a nechceme zkoumat tvorbu a udržování jednotlivých micro-clusterů za běhu algoritmu.

Jak již bylo řečeno, pro off-line analýzu je nutné ukládat jednotlivé snímky v pravidelných časových intervalech. O tuto funkčnost se stará třída SnapshotController, která je spouštěna ve vlastním vlákně pomocí časovače každou vteřinu. Tato třída implementuje logiku ukládání snímků pomocí techniky *pyramidového časového rámce*. Využívá k tomu třídu Order, jenž reprezentuje jednotlivá pořadí. Vzhledem k tomu, že počet snímků v daném pořadí je konstantní, využil jsem k jejímu ukládání implementaci fronty pomocí pole dané velikosti. Třída Snapshot pak reprezentuje jednotlivé snímky. Skládá se tedy z kolekce micro-clusterů a dané časové značky. Jak plyne z výše uvedeného, snímky jsou v průběhu algoritmu ukládány a zpracovávány pouze v paměti programu. Nepoužil jsem zde ukládání na disk do souborů, neboť snímků není díky použité technice mnoho dokonce i při velmi dlouho trvající analýze. Navíc ukládání na disk by sebou neslo mnohem delší dobu zpracování a větší systémové nároky (nový snímek každou vteřinu).

Poslední neméně důležitou třídou balíčku je třída MicroCluster. Jak již plyne z ná-

zvu, reprezentuje jeden konkrétní Micro-cluster. Třída obsahuje metody pro výpočet středu micro-clusteru, přidání bodu, spojení s jiným Micro-clusterem a další metody, jenž vyplývají z vlastního algoritmu. Vzhledem k nutnosti ukládání snímků (kopií stávajícího stavu), třída MicroCluster implementuje rozhraní System.ICloneable.

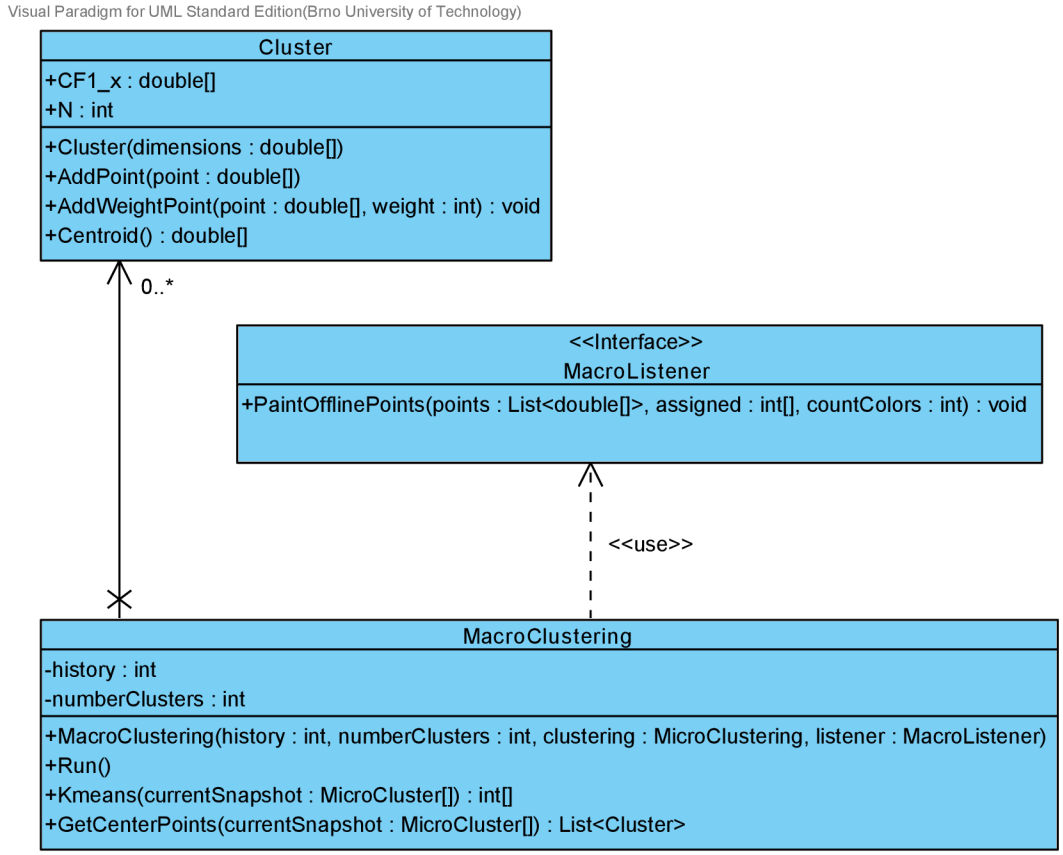


Obrázek 6.6: Třídy balíčku MicroClusters

6.2.7 Balíček MacroCluster

Jak je vidět na obrázku 6.7, balíček se skládá ze dvou tříd a jednoho rozhraní. Hlavni třída má název MacroClustering. Tato třída implementuje vše potřebné pro off-line analýzu. V konstruktoru třídy jsou nastaveny požadované parametry off-line analýzy. Metoda Run()

pak provede vlastní analýzu dle nastavených parametrů. Dále tato třída obsahuje metodu Kmeans(), která implementuje modifikovaný algoritmus k-means tak, jak byl popsán v kapitole 5.2. Třída Cluster pak reprezentuje jeden konkrétní shluk. Obsahuje tedy body (středy micro-clusterů) náležící do daného shluku a metody pro přidání bodu, váhového bodu a výpočet středu. Pro komunikaci a vizualizaci výsledků off-line analýzy byl opět použit koncept rozhraní, jenž je vyšší vrstvou implementováno a předáno v konstruktoru při zahájení off-line analýzy. V případě, že je předána hodnota *null*, k předání informace nedojde.



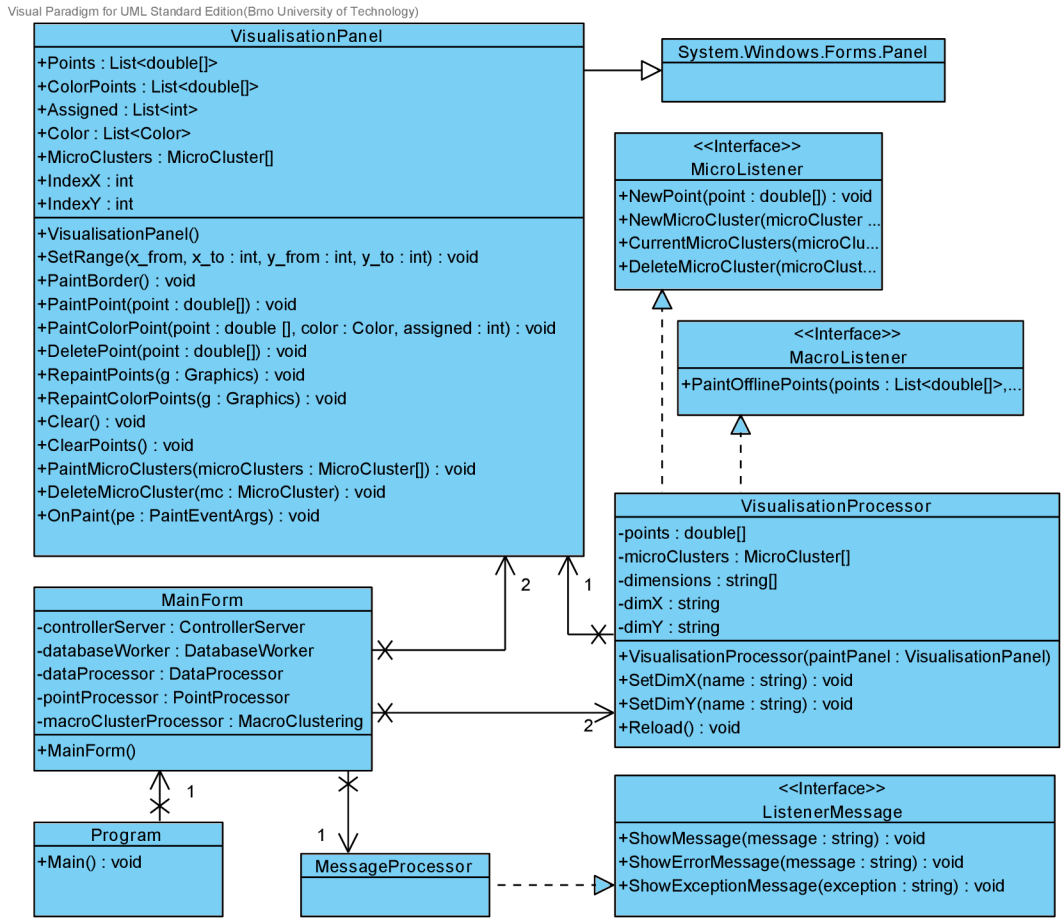
Obrázek 6.7: Třídy balíčku MacroClusters

6.2.8 Balíček GUI

Posledním balíčkem aplikace je balíček GUI. Jedná se o řídící balíček, z něhož je ovládána celá aplikace. Diagram tříd je uveden na obrázku 6.8. V diagramu jsou uvedeny pouze stěžejní třídy balíčku, pro zjednodušení diagramu zde neuvádím třídy grafických prvků jako tlačítka, textová pole apod. Vstupním bodem balíčku a celé aplikace je metoda Main() třídy Program. V této metodě je pouze vytvořena instance třídy MainForm. Veškeré ovládání aplikace je pak prováděno z této třídy. Vyjma konstruktoru jsem u této třídy neuvedl žádné metody, neboť jich je velké množství a jedná se o metody obsluhující události vzniklé nad prvky formuláře (stisk tlačítka atp.) + validační metody. Mým cílem bylo také poskytnout uživateli aplikace nástroj pro vizualizaci výsledků algoritmu. K tomuto účelu slouží třída VisualisationPanel, která dědí od třídy System.Windows.Forms.Panel. V rámci této třídy byly implementovány metody pro nastavení rozsahu, hranic, zobrazení a mazání příchozích bodů,

zobrazení a mazání MicroClusterů atp.

Třída VisualisationProcessor pak implementuje rozhraní MicroListener a MacroListener a zpracovává tak informace z nižších vrstev, které dále předává příslušnému panelu. Poslední třídou je třída MessageProcessor, která slouží pro zpracování všech druhů zpráv z balíčku MSSQLController.



Obrázek 6.8: Zjednoduššený diagram tříd balíčku GUI

Kapitola 7

Implementace

Tato kapitola se věnuje popisu vlastní implementace. Jsou zde uvedeny některé stěžejní části implementace vztažené k jednotlivým komponentám (balíčkům) z návrhu a také ukázky výsledného GUI. Jako programovací jazyk byl vybrán jazyk *C#*, a to z toho důvodu, že pracujeme v prostředí MSSQL, kde jde využít *dll assembly*. Dalším důvodem pro tuto volbu bylo, že původním plánem návrhu bylo použít k předzpracování frameworku *Microsoft StreamInsight*, jenž se skládá z mnoha dll knihoven, jejichž použití v programu postaveném na .Net frameworku je bezpečnější a přehlednější, než používat tyto knihovny v jiném programovacím jazyce. Jako vývojové prostředí jsem zvolil Microsoft Visual Studio.

7.1 Komunikace klient-server

Jak již bylo uvedeno v části návrhu, komunikace mezi klientskou a serverovou částí je realizována skrze XML soubory. Formát takovéhoto souboru je uveden na obrázku 7.1. Přenáší se tedy pouze data o zdrojové tabulce a hodnoty jednotlivých sloupců, jejichž typ patří do množiny číselných typů (double, float, int atp.). Vzhledem k jednoduchosti přenášených dat jsem také zvažoval posílat tato data jako holý text, kdy by byly jednotlivé hodnoty od sebe odděleny znakem středníku. Tímto by došlo k redukci počtu bajtů přenášených po síti, nicméně vzrostla by obtížnost zpracování takovýchto dat na straně serveru (oblast předzpracování), proto jsem nakonec zvolil formát jednoduchého XML souboru.

```
<?xml version="1.0"?>
<data tableName="ContrShort">
    <ICO>1122874</ICO>
    <ROKVYS>1997</ROKVYS>
    <MESVYS>12</MESVYS>
    <VALUE>854,125</VALUE>
</data>
```

Obrázek 7.1: Formát komunikačního XML souboru

7.2 MSSQL Controller

Komponenta MSSQL Controller zastává ve výsledné aplikaci dvě zásadní funkce, a to zpracování vstupních dat a práce s databází.

7.2.1 Zpracování vstupních dat

Při příchodu nových dat se vždy vytvoří nové vlákno, které tato data zpracuje. Toto vlákno pak provádí logiku veškerého zpracování tj. od předzpracování až po samotnou vizualizaci. Proto je nezbytné některé části kódu synchronizovat (zejména v oblasti udržování micro-clusterů). Při zpracování může dojít k mnoha výjimkám, které jsou případně předány komponentě GUI a zobrazeny uživateli.

7.2.2 Práce s databází

Pro práci s databází jsou implementovány funkce pro připojení k databázi, zjištění všech tabulek, pohledů, sloupců daných tabulek (pohledů) apod. V této oblasti bylo nejobtížnější, jak vyřešit implementaci CLR triggeru, který běží nad danou tabulkou (pohledem) konkrétní databáze. Zde jsem implementoval dvě řešení, která zde budou porovnána.

- **Samostatný projekt** – Jedná se o úplně oddělené řešení od zde rozebírané aplikace. Microsoft Visual Studio umožňuje vytvořit *CLR Database Project* přímo nad konkrétní databází. V rámci takovéhoto projektu jsem tedy vytvořil CLR trigger, který implementoval potřebnou funkčnost. Toto řešení (samostatný projekt) bude také součástí odevzdaných aplikací. Výhodou zde oproti druhému řešení je, že lze přímo nastavit konkrétní data z tabulky, která budou dál posílána. Je to dáno koncepcí triggeru. Trigger je volán pro událost FOR INSERT v případě nasazení nad tabulkou. V případech nasazení nad pohledem se jedná o událost INSTEAD OF INSERT. V obou případech je tedy kód spuštěn tedy před vlastním vložením dat do tabulky (pohledu). Posléze vybere požadované sloupce z právě vkládaných dat pomocí sql dotazu jako např. SELECT jmeno, prijmeni, plat, vyska FROM INSERTED. A právě tato data jsou dále předmětem tvorby posílaného XML souboru. Máme-li tedy tabulku s mnoha sloupci, můžeme takto výrazně redukovat množství dat posílaného po síti. Další výhodou je možnost nastavení požadovaného portu a serveru, na který se budou data posílat. Toto u druhého řešení není možné.

- **Univerzální přístup** – Tento přístup vychází z myšlenky poskytnout uživateli příslušný komfort tvorby CLR triggeru za cenu menší efektivity. Zde dochází k tvorbě přímo v aplikaci. Tvorba je založena na tom, že byl vytvořen *univerzální* CLR trigger, jehož assembly (dll) je pak použito k vytvoření ASSEMBLY na MSSQL serveru. Pomocí tohoto ASSEMBLY je pak vytvořen trigger, pracující nad příslušnou tabulkou (pohledem) dle uživatelského nastavení. Univerzálnosti je zde dosaženo tím, že se posílají veškerá data pomocí dotazu SELECT * FROM INSERTED a na stejný server a port.

  Při takovémto vytváření nového triggeru musí být odstraněn původní. Je to dáno tím, že konkrétní ASSEMBLY (vytvořené z daného dll souboru) může být vytvořeno na serveru pouze v jedné kopii. Při pokusech vytvořit více ASSEMBLY vytvořených z jednoho dll souboru pod různými jmény, server to pozná dle identifikátoru MVID a ohlásí chybu. Pro ujasnění je postup vytvoření CLR triggeru uveden v následujícím pseudokódu:

  Nasazení CLR triggeru na server

  – **Vstup**: jméno tabulky, jméno schématu, jméno triggeru a typ (trigger nad pohledem/tabulkou)

1. Nalezení původního triggeru, který využívá dané ASSEMBLY. Pokud takový trigger existuje, je nutné jej odstranit.
2. Pokud existuje dané ASSEMBLY, je potřeba jej odstranit.
3. Vytvoříme nové ASSEMBLY z aktuálního dll souboru.
4. Nad takto vytvořeným ASSEMBLY vytvoříme trigger s příslušným jménem a nad příslušnou tabulkou (pohledem).

V rámci této části implementace jsem zkoumal, jaký vliv má počet zpracovávaných dimenzí na dobu provádění triggeru. Chtěl jsem zjistit, zdali se vyplatí využívat neuniverzálního řešení, kdy si vyberu pouze požadované dimenze ze všech možných v dané tabulce. V rámci neuniverzálním přístupu byly pokaždé posílány z celkového počtu dimenzí pouze dvě dimenze. Jak je vidět z hodnot uvedených v tabulce 7.1, vliv je zde minimální. Rychlost se tím nijak výrazně nezvýšila. Toto je dáno tím, že nejvíce náročnou částí běhu triggeru je navázání spojení TCP spojení s aplikací. Z tohoto měření lze usoudit, že využití univerzálního řešení nemá zvláštní dopad na efektivitu na straně serveru a může jej použít i v případě vícedimenzionálních tabulek.

Počet dimenzí	Univ. přístup (μs)	Neuniv. přístup (μs)
2	12,9	13
5	13,8	14,1
10	14,7	13,8
15	16,3	14,4
20	13,3	12,6

Tabulka 7.1: Naměřené hodnoty trvání triggeru

7.3 Preprocessing

V oblasti předzpracování jsem při prvotním návrhu počítal s využitím frameworku Microsoft StreamInsight, jehož funkce byla stručně uvedena v podkapitole 6.1.3. Tento framework poskytuje možnosti zpracovávat proudy dat z více zdrojů, tyto proudy různým způsobem kombinovat a poskytnout na výstup požadovaná data. Cílem bylo pak porovnat moji implementaci předzpracování (pomocí klasických konstrukcí jako jsou podmínky atp.) s předzpracováním, implementovaným pomocí této technologie za využití dotazovacího jazyka LINQ. Tato cesta se však při vlastní implementaci ukázala jako špatná z těchto důvodů:

- Parametry předzpracování musí být známy již při kompilaci aplikace, nelze tedy generovat dotazy jazyka LINQ dynamicky pomocí nastavení v GUI.
- Rychlost zpracování je několikanásobně pomalejší, než u klasického řešení. Při testování nad tabulkou s 5ti sloupci a stejném nastavení předzpracování mé vlastní řešení trvalo průměrně 0,002 ms, kdežto řešení pomocí tohoto frameworku průměrně 1 ms. Toto je dáno tím, že vstupní data jsou pomocí vstupního adaptéru nejdříve převedena na tzv. *vstupní události*, nad kterými je provedena filtrace pomocí LINQ. Výsledek je pak převeden na výstupní událost a pomocí výstupního adaptéru poslán na výstup.

Vzhledem k výše uvedeným důvodům jsem od této implementace upustil a naprogramované adaptéry a převodní logika není součástí aplikace.

7.4 MicroClusters

Jak již bylo řečeno, komponenta MicroClusters tvoří jádro aplikace, a proto zde bude její implementace rozebrána podrobněji.

7.4.1 Zpracování příchozího bodu

Veškerá logika zpracování příchozího bodu je obsažena ve třídě MicroClustering a v její metodě ProcessPoint(). Parametry zpracování jsou nastaveny již při vytvoření instance této třídy. Zpracování bodu pracuje v několika fázích:

- Nejdříve nastává inicializační část, kdy jsou příchozí body pouze ukládány do bufferu a dále nejsou zpracovávány.
- Po inicializační části je zavolána metoda Kmeans(). Ta provede počáteční inicializaci micro-clusterů. Po této inicializaci je nutné spustit ukládání snímků proudu dat. Toto je provedeno pomocí delegátu TimerCallback, jemuž je předána v konstruktoru metoda SaveSnapshots() třídy SnapshotsController. Tento delegát je pak použit pro inicializaci třídy Timer, která každou vteřinu pustí metodu delegáta.
- Nyní již nastává vlastní aktualizační část algoritmu. Nejdříve je tedy nalezen nejbližší micro-cluster k právě přijatému bodu. Zde je využita abstraktní třída Distance a její metoda GetDistance(). Tato třída má dvě konkrétní implementace pro výpočet Euklidovské a Manhattanovské vzdálenosti. Uživatel si tak může na začátku zvolit, která metoda bude použita. Hranice nejbližšího micro-clusteru je dána směrodatnou odchylkou bodů v něm obsažených. Pro tento výpočet je určena metoda rmsDeviation(), která je podrobněji popsána v 7.4.1. V rámci zpracování bodu může dojít k situaci, kdy je potřeba odstranit některý micro-cluster dle jeho *relevantního razítka*. Výpočet této hodnoty provádí metoda RelevanceStamp(), jejíž implementace je rozebrána podrobněji v 7.4.1. Poslední situací, která může nastat, je spojení dvou micro-clusterů do jednoho. Tato funkčnost je implementováno v metodě Merge().

Hranice micro-clusteru

Jak již bylo řečeno, hranice micro-clusteru je daná hodnotou směrodatné odchylky bodů obsažených v micro-clusteru. Jistou zvláštností zde je, že se pohybujeme v n-dimenzionálním prostoru. Proto je postup výpočtu poněkud komplikovanější a je uveden na následujícím pseudokódu 1. Dochází tedy nejdříve k výpočtu rozptylů postupně nad všemi dimenzemi bodů micro-clusteru a následně k jejich sumarizaci. Posléze je spočítána hodnota průměrného rozptylu, z něhož je vypočítána výsledná směrodatná odchylka. Výsledná hodnota se pak ještě násobí zvolenou konstantou τ. Tento algoritmus vychází z obdobného algoritmu, uvedeného v [11].

Algoritmus 1 Algoritmus pro výpočet hranic micro-clusteru

$r_j^2 = \overline{CF2^x}_j/n - \overline{CF1^x}_j * \overline{CF1^x}_j/n^2$
$R = \sum_{j=1}^{n} r_j^2$
$R = \sqrt{R/n}$
return $\tau * R$;

Relevantní razítko

Jak již bylo uvedeno v podkapitole 5.1.2, pro výpočet relevantního razítka micro-clusteru používáme průměr a směrodatnou odchylku časů příchodu jednotlivých bodů micro-clusteru. Posléze je potřeba najít hodnotu *m/(2*n)-tého* percentilu za předpokladu, že časy jednotlivých příchodů mají normální rozložení. Implementace tohoto zdánlivě banálního problému se ukázala jako poněkud komplikovaná. Snažil jsem se nalézt nějakou funkci, která by na vstup dostala tyto dvě hodnoty, jenž popisují normální rozložení a vrátila hodnotu požadovaného percentilu. Pro implementaci tohoto problému jsem byl nucen použít řešení s využitím *standardní normální distribuční tabulky* (SNDT), jejíž určitá část je v aplikaci použita ve formě statického pole. S pomocí hodnot této tabulky a rovnice 7.1 jsme schopni aproximovat skutečnou hodnotu odpovídající požadovanému percentilu. Hodnota tedy není nikdy úplně přesná, nicméně v oblastech, ve kterých hledáme tj. od percentilu 0,0025 a menší, jsou již rozdíly mezi percentily tak malé, že se to nijak výrazně neovlivní na přesnosti funkce. Výpočet tedy probíhá tak, že se v SNDT hledá oblast, do které spadá hledaný percentil. Získáme tak z-score, pomocí něhož je dopočítána požadovaná hodnota.

$$z = \frac{x - \mu}{\sigma}, \tag{7.1}$$

kde z je hodnota nalezeného z-score, μ je průměr, σ je směrodatná odchylka a x značí hledanou hodnotu.

7.4.2 Ukládání snímků

Ukládání snímků je prováděno každou vteřinu v metodě SaveSnapshots(). Zde jsem implementoval rozšířenou metodu *pyramidového časového rámce*, kde se dle parametrů α a l ukládá v každém pořadí $\alpha^l + 1$. Tímto je dosaženo velmi přesné aproximace s přijatelnými paměťovými nároky. Pro vytváření snímku tj. kopií micro-clusterů jsem využil realizaci rozhraní System.ICloneable. Klonování je zde prováděno manuálně, tedy vytvořením nového objektu MicroCluster a kopií jeho atributů. Toto řešení jsem použil proto, že pravděpodobnost změny struktury objektu je velmi malá, proto využití univerzálních metod jako serializace, deserializace apod. ztrácí význam. Navíc je tato metoda nejrychlejší za cenu své neuniverzálnosti, která nám ale nevadí.

7.5 MacroClusters

V oblasti off-line analýzy jsem implementoval funkci pro tvorbu vysokoúrovňových shluků na základě historie dané délky. O tuto funkčnost se stará metoda Run() třídy MacroClustering. Metoda tedy nejdříve získá dva snímky (aktuální a historický) pomocí metody GetSnapshots() třídy SnapshotsController balíčku MicroClusters. Posléze je provedeno odečtení micro-clusterů historického snímku od micro-clusterů aktuálního snímku. Nad výsledným seznamem zbylých micro-clusterů je provedeno shlukování dle modifikovaného algoritmu *k-means*.

Tato oblast aplikace v sobě skrývá největší možnosti pro rozšíření. Vzhledem k charakteru algoritmu, kdy jádrem je udržování struktury micro-clusterů v průběhu analýzy a jejich snímkování, lze zde doimplementovat některé další varianty evoluční analýzy proudu dat. Jako příklad lze uvést:

- Porovnávání dvou časových oblastí proudu dat.

- Analýza, zdali jsou v čase t_2 nějaké nové micro-clustery oproti času t_1.
- Analýza, zdali došlo k posunu některých micro-clusterů v časovém intervalu (t_1,t_2).

Variant je nicméně mnohem více a jejich doimplementování záleží na konkrétní aplikaci a použití. Toto bylo jedním z důvodů oddělení *on-line* části algoritmu od *off-line* části. Lze tak využít pouze balíček (knihovnu) MicroCluster a přilinkovat ji do libovolné aplikace (prostředí) a nad ní postavit off-line analýzu dle vlastních požadavků.

7.6 Uživatelské rozhraní

Při návrhu a implementaci uživatelského rozhraní jsem si kladl za cíl vytvořit nástroj, který uživateli usnadní veškeré potřebné činnosti související s implementací algoritmu. Jedná se konkrétně o tyto tři hlavní skupiny:

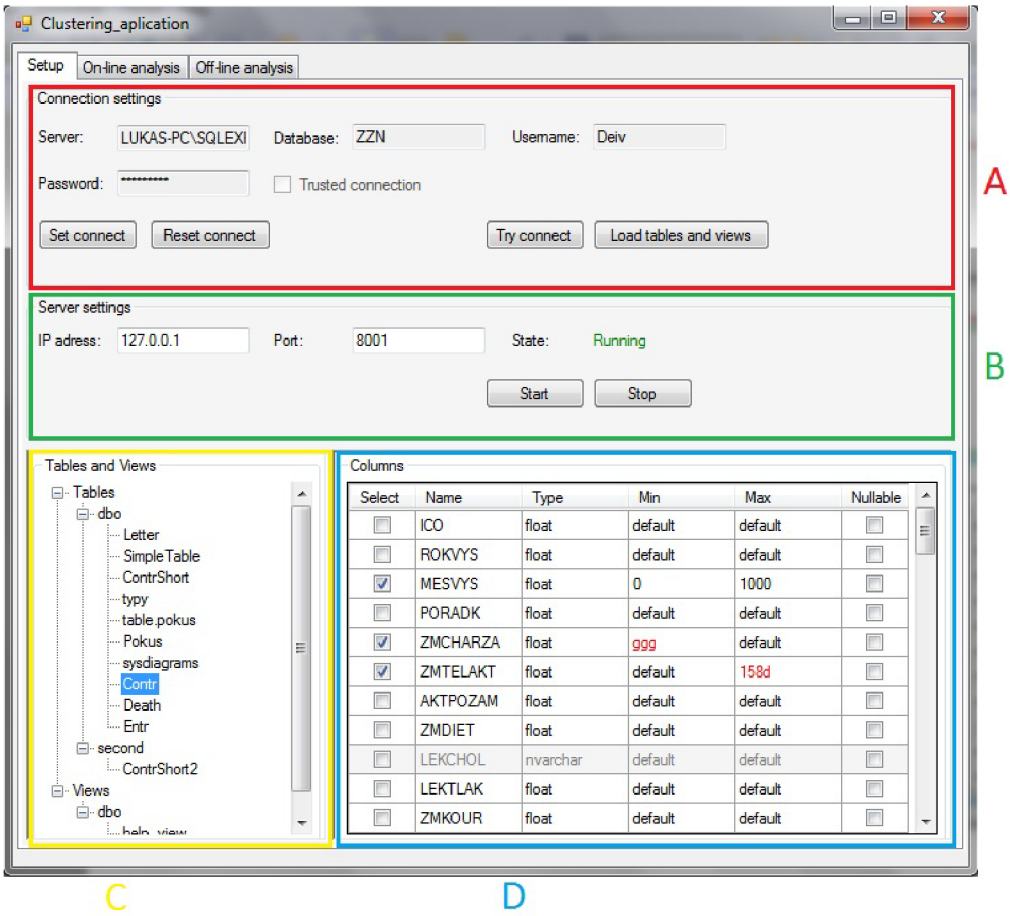
- Nástroj pro nastavení serveru, připojení k databázi, vytvoření CLR triggeru a nastavení parametrů předzpracování.
- Nástroj pro on-line analýzu proudu dat a reprezentaci aktuálních micro-clusterů.
- Nástroj pro off-line analýzu tj. poskytnout nástroj pro analýzy uvedenou v předchozí kapitole.

Uživatelské rozhraní má tedy tři hlavní obrazovky, z nichž na každé jsou prvky vycházející ze stanovených cílů.

7.6.1 Nastavení analýzy

Jak je vidět na obrázku 7.2, obrazovka pro nastavení analýzy je rozdělená na 4 části:

- **Část A** – zde jsem naimplementoval funkčnost pro vyzkoušení databázového připojení. Dále pak zde můžeme toto připojení nastavit. Po nastavení již nelze měnit parametry připojení. Toto je z toho důvodu, že tyto parametry jsou dále využívány při další práci s databází a jejich nechtěné změnění v průběhu práce s aplikací může vyvolat problémy.
- **Část B** – část pro nastavení IP adresy a čísla portu, na kterém bude běžet aplikace (server).
- **Část C** – obsahuje přehled tabulek a pohledů dané databáze. Dále zde lze vytvořit stiskem pravého tlačítka nad příslušnou tabulkou (pohledem) CLR trigger pro posílání dat.
- **Část D** – zde je možné nad konkrétní tabulkou nastavit požadované sloupce, jenž budou předmětem analýzy. Pro implementaci možnosti nastavení byl použit grafický prvek DataGridView. Jak je vidět na obrázku, prvek implementuje i validaci zadaných hodnot pro předzpracování. Pokud jsou hodnoty vyplněny špatně, nelze spustit vlastní analýzu.



Obrázek 7.2: Vzhled okna aplikace pro nastavení připojení: A. parametry připojení k databázi, B. Nastavení serveru, C. Tabulky a pohledy databáze, D. Sloupce tabulky/pohledu a jejich nastavení.

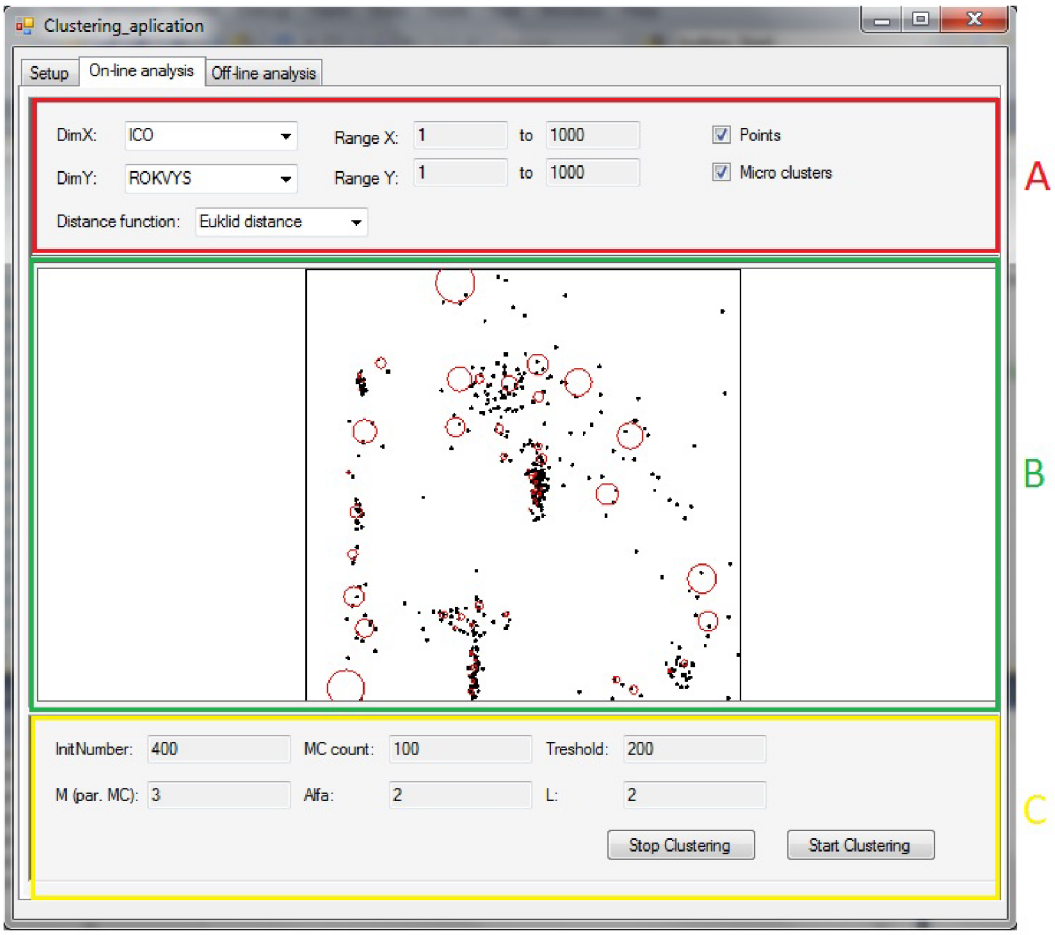
7.6.2 On-line analýza

Na obrázku 7.3 lze vidět ukázku obrazovky pro on-line analýzu. Obrazovka je rozdělena na 3 části:

- **Část A** – zde lze nastavit parametry vizualizace. Při vizualizaci se používá projekce n-dimenzionálního prostoru do dvou dimenzí. Tyto dimenze je potřeba zvolit a mohou být změněny v průběhu vizualizace. Vzhledem k tomu, že dochází k vizualizaci proudu dat, je nutné nastavit i rozsah hodnot, které chceme vidět v jednotlivých osách. Není totiž dopředu nikdy známo, jaký bude skutečný rozsah hodnot. Zde jsem experimentoval s možností implementace vizualizace dynamicky ve smyslu, že by se rozsah vizualizačního panelu dynamicky přenastavoval dle rozsahu příchozích bodů. Toto ovšem dávalo nepřehledné výsledky. Je to dáno tím, že při tomto řešení stačí jeden odlehlý bod a hned se vizualizace stane nepřehlednou. Proto jsem zvolil řešení, kdy se nastaví rozsah před vlastní vizualizací a sleduje se tak pouze určitá část prostoru.

- **Část B** – tato část obsahuje vizualizační panel pro vlastní vykreslování výsledků. Implementaci tohoto panelu je věnována podkapitola 7.6.2.

- **Část C** – zde se nastavují parametry vlastního algoritmu. Jedná se o parametry počtu inicializačních bodů, požadovaný počet micro-clusterů atp. Dále je zde tlačítko pro vlastní zahájení analýzy a její ukončení.



Obrázek 7.3: Vzhled okna aplikace pro vizualizaci on-line analýzy: A. Parametry vizualizace, B. Vizualizační panel, C. Parametry algoritmu.

Vizualizační panel

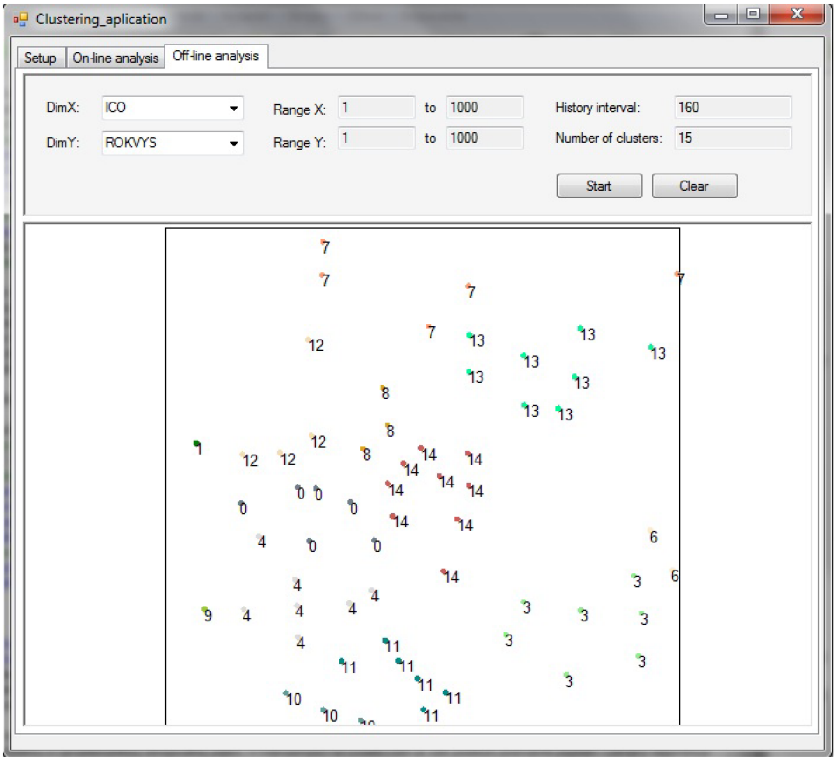
V rámci on-line a off-line části je nutné v průběhu analýzy provádět vizualizace. Pro tyto účely jsem implementoval vlastní panel, který je v aplikaci reprezentován třídou VisualisationPanel. Tento panel umožňuje vizualizaci bodů, micro-clusterů, přiblížení, posun atp. Je tak možné při větší koncentraci bodu v jednom místě si tyto body přiblížit a zobrazit si tak detailnější pohled na danou oblast. Toto se hodí zejména v aplikacích, kde se pracuje s hodnotami o velkém rozsahu, neboť limitem pro velikost panelu je velikost obrazovky.

Při vlastní implementaci jsem musel zvolit kompromis mezi dokonalou vizualizací a rychlostí vizualizace. Problém je zde v tom, že při příchodu nového bodu dochází vždy při vlastní analýze k rušení (vytváření), případně posunu některého z micro-clusterů. Toto sebou nese potřebu neustále překreslovat celý panel v rámci vizualizace. Zde se ale objevil problém blikání panelu. V některých případech pomůže nastavení *bufferování* panelu na hodnotu true, kdy je překreslování *bufferováno*. Nicméně při rychlejším proudu dat ani

toto opatření nezabránilo blikání a vizualizace se tak stala nepřehlednou. Proto jsem zvolil řešení, kdy nedochází k překreslení vždy celého panelu, ale pouze se smaže stávající rušený (posouvaný) micro-cluster a zobrazí se nový micro-cluster. Rušení zde probíhá překreslením daného micro-clusteru bílou barvou. Uvedené řešení má ale tu nevýhodu, že může dojít k přemazání i jisté části jiného micro-clusteru, čímž není vizualizace úplně dokonalá. V této podobě je nicméně použitelnější, než kdyby bylo použito bufferování a překreslování celého panelu. Dalším opatřením, které jsem implementoval, je to, že se vždy zobrazuje v panelu posledních 500 bodů. Je to z toho důvodu, že při delší analýze (tisíce bodů) je již panel přehlcen starými body a nejsou patrné nově příchozí body. Tuto konstantu jsem zvolil na základě subjektivního pocitu z jednotlivých vizualizací.

7.6.3 Off-line analýza

Na obrázku 7.4 je vidět výstup off-line analýzy. Uživatel si zde může opět nastavit osy, ve kterých mu budou zobrazeny výsledky. Jako body jsou zde použity středy jednotlivých micro-clusterů, které spadají do dané historie proudu dat. Při vlastním používání této analýzy hraje významnou roli volba počtu shluků, které chceme vytvořit. Vzhledem k tomu, že tento údaj není dopředu znám, řešil jsem zde problém, jak vizualizovat výsledné shluky. Jedná se o problém obarvení bodů jednotlivých shluků. Zde jsem použil implementaci statického pole, reprezentujícího všechny dostupné barvy. Posléze se přiřazení barev jednotlivým shlukům určí na základě vygenerování pseudonáhodného čísla v rozsahu všech dostupných barev. Nicméně se zde často stává, že některé barvy nelze od sebe dobře rozlišit např. dva podobné odstíny zelené, proto je ke každému bodu napsáno vždy číslo shluku.



Obrázek 7.4: Vzhled okna aplikace pro vizualizaci off-line analýzy

Kapitola 8

Testování a zhodnocení výsledků

V rámci testování výsledného algoritmu a k ověření jeho funkčnosti jsem potřeboval vhodný vzorek dat. Testovací data pro algoritmus *CluStream* jsou specifická tím, že se jedná o n-dimenzionální data pouze s číselnými atributy. Abych mohl otestovat funkčnost algoritmu, potřeboval jsem proudová data, která obsahují alespoň nějaké shluky. Vzhledem k tomu, že nemám přístup k reálným datům, jenž bych mohl otestovat, musel jsem si vytvořit vlastní generátor proudu dat. V následující podkapitole je uveden princip tohoto generátoru a v dalších podkapitolách jsou uvedeny jednotlivé testy a jejich výsledky. Na závěr kapitoly je uvedeno závěrečné zhodnocení výsledků.

8.1 Generátor dat

Pro tvorbu testovacího proudu dat jsem si vytvořil vlastní generátor. Princip tohoto generátoru vychází z myšlenky, že potřebuji náhodná data, nicméně tato data musí obsahovat alespoň nějaké shluky tak, aby bylo co testovat. Princip generátoru je zde popsán v několika krocích:

1. Vstupními parametry generátoru je příslušná tabulka, do které budou data generována. Dále pak požadovaný počet shluků Q a požadovaný počet generovaných bodů K. Posledním parametrem je rozsah, v jakém mají být generované hodnoty. Tento rozsah je stejný pro všechny dimenze. Vzhledem k tomu, že jde o testování, generátor předpokládá v tabulce pouze sloupce číselných datových typů (negeneruje řetězce atp.).

2. Generátor zjistí počet dimenzí (sloupců) dané tabulky. Počet dimenzí bude dále značen jako N.

3. Na základě požadovaného počtu shluků Q generátor vygeneruje Q náhodných referenčních bodů X_q v N-dimenzionálním prostoru. Tyto body budou brány jako referenční body pro jednotlivé shluky.

4. Příslušnost generovaných K bodů do jednotlivých shluků musí být také náhodná, aby shluky obsahovaly různý počet bodů. Generátor tedy vygeneruje Q náhodných hodnot P_q pro které platí, že jejich suma je rovna 1.

5. Následujícím požadavkem je, aby jednotlivé shluky měly různý tvar. Zde je využito *normálního rozložení*. Řešení je takové, že se pro každou dimenzi každého referenčního

bodu X_q vygeneruje náhodná hodnota rozptylu R_n v rozsahu <0,25;1/10* rozsah dané dimenze>. Tohoto rozptylu je pak použito při určování posunu nového bodu vůči referenčnímu bodu příslušného shluku.

6. Nyní následuje již generování K náhodných bodů. Generování nového bodu probíhá v těchto krocích:

   - Aby bylo dosaženo iluze skutečného proudu dat (občas odlehlé hodnoty), je zde implementován doplněk, že s pravděpodobností 5% dojde k vygenerování náhodného bodu, který nepatří do žádného shluku. Pokud nastane tato situace, generátor nepokračuje v dalších krocích generování bodu, vygeneruje náhodný bod, pošle jej do tabulky a vrátí se zpět ke generování dalšího bodu. Jinak se pokračuje následujícím krokem.
   - Nejdříve se zjistí na základě již vygenerovaných Q hodnot P_q, do kterého shluku bude patřit aktuální bod.
   - Po nalezení příslušného shluku (jeho referenčního bodu X_q) je nutné provést posunutí v jednotlivých dimenzích od tohoto bodu. Posunutí v jednotlivých dimenzích je dáno pomocí náhodné hodnoty s normálním rozložení, jehož průměr je roven 0 a rozptyl R_n je roven hodnotě, která byla vygenerována pro daný shluk a danou dimenzi v jednom z předchozích kroků.
   - Takto vytvořený bod je pak poslán do příslušné tabulky.

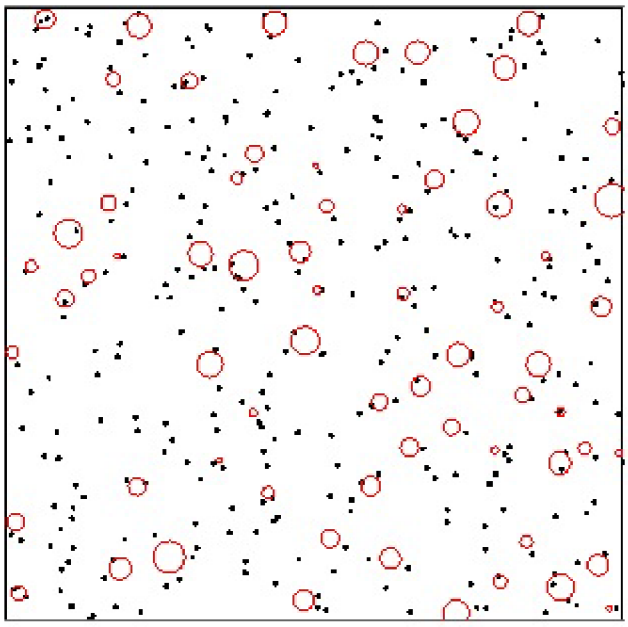
8.2 Testování s různými parametry

V následujících podkapitolách budou ukázány příklady výstupů některých testů na základě různých vstupních parametrů algoritmu a generátoru. Volba parametrů je pro běh algoritmu *CluStream* zásadní. Při špatně zvolených parametrech nelze výsledky algoritmu v praxi rozumně použít.

8.2.1 Náhodný šum

Jak je vidět na obrázku 8.1, algoritmus pracuje i nad čistě náhodnými daty. Nad těmito daty jeho použitelnost nicméně rapidně klesá. Jedná se zde o fakt, že v průběhu běhu algoritmu dochází ve většině případů k rušení některého ze stávajících micro-clusterů a tvorbě nového micro-clusteru, neboť dimenze nově příchozího bodu jsou náhodné. Tímto je velká část micro-clusterů tvořena pouze jedním bodem, což při udržování konstantního počtu micro-clusterů v průběhu analýzy snižuje schopnost algoritmu poskytnout potřebná data pro off-line analýzu.

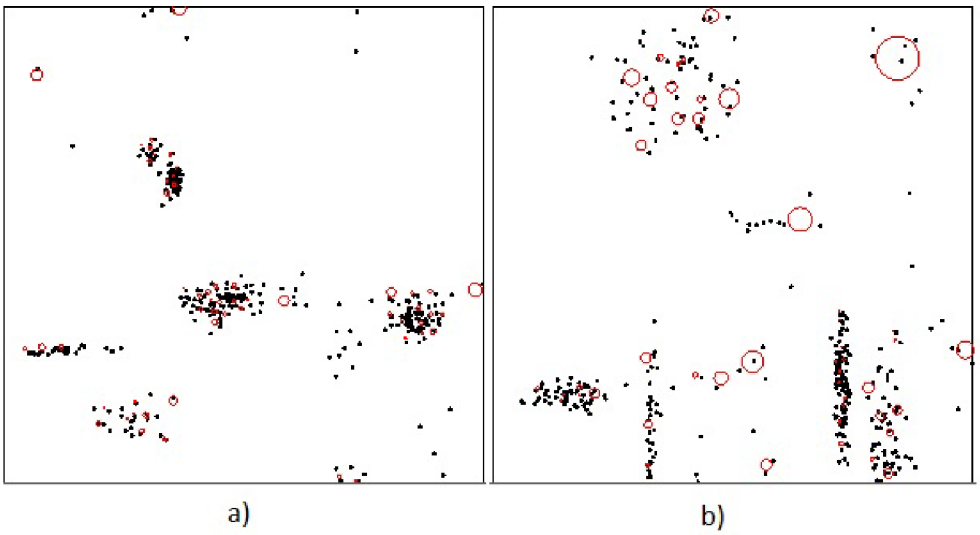
Jako příklad si lze uvést požadavek na analýzu proudu dat za *delší* časové období. Je zde předpoklad, že v rámci tohoto období bylo zpracováno velké množství bodů. Protože ale algoritmus neustále rušil stávající micro-clustery a tvořil nové, v každém micro-clusteru je pouze informace o pár bodech (většinou jednom bodu), tímto je rozsah vstupu pro off-line analýzu výrazně omezen. To znamená, že spousta bodů, jenž byla zpracována v tomto období, není v aktuálním snímku, z čehož vyplývá, že nikterak neovlivní off-line analýzu. Tímto jsme ztratili výraznou část informace.



Obrázek 8.1: Analýza náhodného šumu při nastavení počtu micro-clusterů na hodnotu 100

8.2.2 Data obsahující shluky

Na obrázku 8.2 je uvedena ukázka dvou analýz, kde již byl použit generátor popsaný v předchozí části kapitoly. Na levém obrázku lze vidět shluky, jenž mají menší rozsah (dáno nastavením generátoru při této analýze). Výsledné micro-clustery pak lze hůře rozeznat pouhým okem, proto aplikace umožňuje jejich přiblížení. Naproti tomu na pravém obrázku jsou shluky s větším rozsahem. Je možné si všimnout, že u takovýchto shluků mají micro-clustery větší hranice. Toto je dáno tím, že obsahují body, které jsou od sebe více vzdálené.



Obrázek 8.2: Ukázka dvou analýz se stejnými parametry, ale rozdílnými rozsahy jednotlivých shluků.

8.3 Rychlost zpracování nového bodu

Po ověření funkčnosti algoritmu jsem se zaměřil na změření rychlosti výsledné implementace. Zde jsem také porovnal svoji implementaci s již existující implementací v rámci frameworku MOA [1]. Jednou z částí frameworku je i implementace algoritmu CluStream. Abych mohl porovnat řešení vytvořené v rámci práce s řešením v MOA, musel jsem si stáhnout tuto aplikaci a upravit její zdrojové kódy pro účely testování rychlosti. Při testování jsem se zaměřil pouze na část zpracování nově příchozího bodu (on-line část algoritmu), neboť tato část je jádrem celé implementované aplikace. V následujících dvou podkapitolách jsou uvedeny popisky jednotlivých testů spolu s naměřenými hodnotami. Hodnoty je nicméně potřeba brát s jistou rezervou, neboť každá aplikace je postavena na jiné platformě (MOA je implementována v Javě), dále pak výsledky měření nejsou deterministické při stejné konfiguraci generátoru. Toto je dáno tím, že při každém testu generátor generuje nový proud dat, který je vždy jedinečný (shluky v jiných oblastech atp.) a jeho současná konfigurace může mít vliv několika procent na rychlost. Proto jsem všechna měření opakoval vícekrát a použil vždy průměrnou hodnotu.

8.3.1 Počet micro-clusterů

Prvním testem bylo zjištění závislosti doby zpracování jednoho bodu na počtu micro-clusterů, které jsou udržovány v průběhu analýzy. Testování bylo provedeno nad čtyřdimenzionálními body tak, aby měření nebylo ovlivněno faktorem počtu dimenzí. Jak je patrné z grafu na obrázku 8.3, tato závislost má vzrůstající tendenci (pozvolně narůstající směrnice přímky). Dále je z grafu patrné, že rychlosti obou implementací jsou velmi podobné. Mé řešení dosahovalo v průměru o zlomek lepších časů, nicméně je nutné mít pořád na paměti, že se porovnává implementace postavená na .NET frameworku s implementací postavenou na Javě, dále každá implementace má svůj generátor atp., proto bych obě implementace označil v podstatě za stejně rychlé.

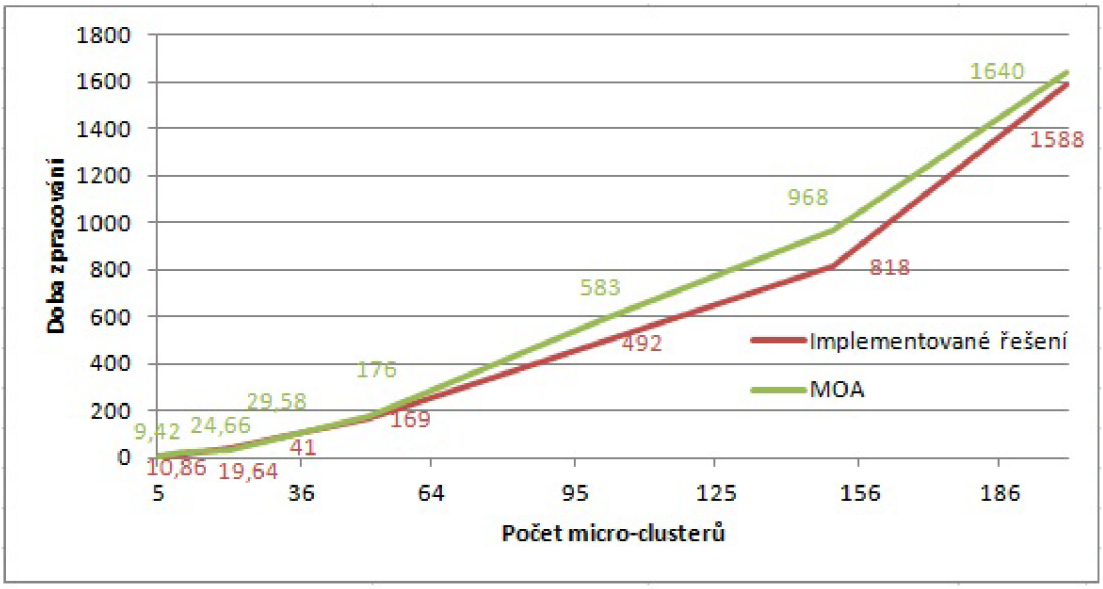
8.3.2 Počet dimenzí

Druhým předmětem zkoumání bylo zjištění závislosti rychlosti zpracování jednoho bodu na počtu jeho dimenzí. Jak je vidět z grafu na obrázku 8.4, je tato závislost téměř lineární, což odpovídá předpokládanému výsledku. Při měření jsem použil konstantní počet udržovaných micro-clusterů (200) a počet generovaných shluků (70). Dále je z grafu patrné, že při stejné konfiguraci je doba zpracování v mém řešení opět v průměru o něco menší, než doba zpracování v frameworku MOA. Zkoušel jsem i vícedimenzionální body (20 a více dimenzí), nicméně zde již implementace v MOA způsobovala výjimky, proto již nešlo tyto hodnoty dále porovnat.

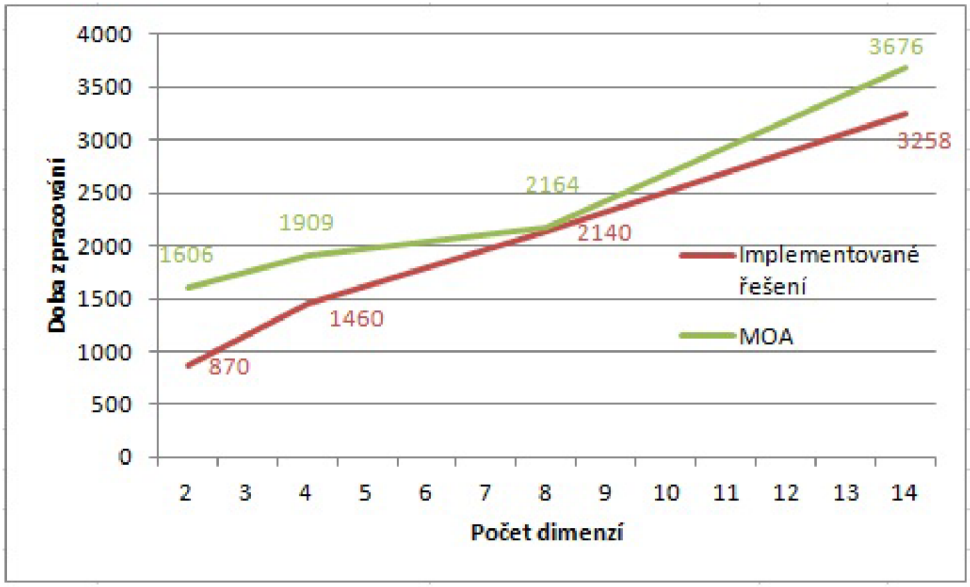
8.4 Zhodnocení

Pro testování algoritmu jsem naimplementoval vlastní generátor proudu dat. S pomocí tohoto generátoru, možnosti debuggingu v prostředí MS Visual Studia a vizualizace výsledků jsem ověřil správnou funkčnost algoritmu. Při porovnání s implementací algoritmu v frameworku MOA jsem také ověřil, že mé řešení je o zlomek rychlejší, než řešení v rámci tohoto frameworku. Pro další zrychlování algoritmu zde již není prostor neboť, jak plyne

[1] Více o frameworku lze nalézt na *http://moa.cs.waikato.ac.nz/*



Obrázek 8.3: Graf závislosti doby trvání zpracování jednoho bodu (v mikrosekundách) na požadovaném počtu micro-clusterů



Obrázek 8.4: Graf závislosti doby trvání zpracování jednoho bodu (v mikrosekundách) na počtu dimenzí

z kapitoly 5, definice algoritmu je poměrně striktní a není zde již další prostor pro vlastní invenci. Aplikace jako celek pak byla navržena tak, aby bylo možné implementaci algoritmu dále využít jako samostatnou knihovnu v jiných projektech na fakultě FIT VUT v Brně.

Kapitola 9

Závěr

Cílem diplomové práce bylo seznámit se s problematikou získávání znalostí z databází se zaměřením na problematiku dolování v proudu dat, dále pak navrhnout a implementovat vybraný algoritmus pro dolování v proudu dat v prostředím MS SQL serveru. Po dohodě s vedoucím jsem se zaměřil na oblast shlukové analýzy. Pro implementaci byl vybrán algoritmu CluStream. Práce navazuje na semestrální projekt, kde jsem se seznámil s teorií potřebnou k vlastní implementaci.

V textu práce je tedy nejprve uveden stručný úvod k problematice získávání znalostí z databází a strojového učení. Jsou zde rozebrány jednotlivé typy úloh s podrobnějším popisem shlukové analýzy. V rámci této analýzy byly popsány jednotlivé typy dat a základní rozdělení shlukových metod. V další části textu je podrobněji rozebrána problematika proudů dat a dolování v proudu dat.

Po rozboru teorie následuje oblast návrhu a implementace řešeného algoritmu CluStream v prostředí MSSQL. Nejprve byly v práci stručně rozebrány použité technologie. Na tuto část navazuje rozboru možných řešení, kde je uvedeno odůvodnění vybraného typu řešení. Dále zde byl popsán návrh architektury aplikace v podobě diagramů balíčku a detailněji rozebrány jednotlivé balíčky. V sekci popisujícím implementaci jsou rozebrány detailněji některé zajímavé části implementace. V této části práce jsou také uvedeny důvody, proč jsem k vlastní realizaci nakonec nevyužil frameworku Microsoft StreamInsight, který je určen pro zpracování proudových dat, a který se jevil ze začátku jako velmi přínosný. Jedná se konkrétně o nutnost rekompilace při každém nastavení aplikace a rychlost, která je výrazně pomalejší než u mého vlastního řešení. Nad výslednou realizací algoritmu jsem také implementoval jednoduché grafické uživatelské rozhraní pro vizualizaci a ovládání algoritmu. Celá aplikace byla nicméně navržena tak, aby bylo možné jednoduše využít implementace daného algoritmu i v rámci jiných aplikací.

Funkčnost výsledného algoritmu pak byla ověřena za použití vlastního generátoru proudu dat. Ukázky a výsledky tohoto testování byly předvedeny v předposlední kapitole. V rámci testování jsem se také zaměřil na dosaženou rychlost algoritmu. Tuto jsem pak porovnal s již existujícím řešením v rámci frameworku MOA. Porovnání ukázalo, že se mi podařilo dosáhnout o pár procent efektivnější realizace daného algoritmu. V závěrečné části práce pak bylo uvedeno stručné zhodnocení dosažených výsledků.

Co se týče dalšího vývoje aplikace, zde je největší prostor v oblasti předzpracování dat a offline analýzy. Mohly by být doimplementovány některé speciální metody předzpracování pro proudy dat, případně jiné druhy offline analýzy plynoucí z použití algoritmu v dané úloze. Implementace algoritmu jako takového již neskýtá možnosti dalšího rozšíření, neboť jeho funkčnost je striktně vymezena a kompletně naimplementována.

Literatura

[1] ALPAYDIN, E. *Introduction to Machine Learning.* London: The MIT Press, 2010. ISBN 978-0-262-01243-0.

[2] ZBOŘIL, F.; AJ.. *Základy umělé inteligence – studijní opora.* Brno: FIT VUT v Brně, 2006.

[3] ZENDULKA, J.; AJ.. *Získávání znalostí z databází – studijní opora.* Brno: FIT VUT v Brně, 2009.

[4] HAN, J.; MICHELINE, K.. *Data Mining: Concept and Techniques.* San Francisco: Elsevier, 2006. ISBN 13: 978-1-55860-901-3.

[5] AGGARWAL, C. C.. *Data Streams: Models and Algorithms.* New York: Springer Science+Business Media, 2007. ISBN-10: 0-387-28759-0.

[6] GABER, M.M.; ZASLAVSKY, A.; KRISHNASWAMY, S.. Mining Data Streams: A Review. *SIGMOD Rec.* červen 2005, roč. 34, č. 2. S. 18–26. ISSN 0163-5808.

[7] ELASMAR, M.; THIRUVENGADACHARI, P.; MARTIN, J.S.. *Clustering Data Streams.* Dostupné na: <http://www.cc.gatech.edu/projects/disl/Courses/cs4440/07Fall/project/proposals/Team5Proposal_final.pdf>.

[8] KHALILIAN, M.; MUSTAPHA, P.. *Data Stream Clustering: Challenges and Issues.* March 2010. Dostupné na: <http://arxiv.org/ftp/arxiv/papers/1006/1006.5261.pdf>.

[9] AGGARWAL, C. C., HAN, J., WANG, J. et al. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases - Volume 29.* 2003. S. 81–92. VLDB '03. ISBN 0-12-722442-4.

[10] MSDN. *Microsoft StreamInsight 2.0* [online]. @ 2012 [cit. 2011-12-29]. Dostupné na: <http://msdn.microsoft.com/en-us/library/hh750619%28v=sql.10%29.aspx>.

[11] AGGARWAL, C. C., HAN, J., WANG, J. et al. A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30.* 2004. S. 852–863. VLDB '04. ISBN 0-12-088469-0.

Seznam příloh

- **A** Obsah CD

Příloha A

Obsah CD

Adresářová struktura přiloženého CD:

- **docs** – programová dokumentace implementace algoritmu a aplikace
- **thesis**
  - **src** – zdrojové texty technické zprávy
  - **dp-xdavid00.pdf** – technická zpráva
- **aplications**
  - **clustream** – zdrojové kódy aplikace
  - **clr** – zdrojové kódy CLR triggeru