Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Shluková analýza datových toků

Diplomová práce

Autor: Tomáš Kratochvíl

Studijní obor: Aplikovaná informatika

Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Hradec Králové Duben 2023

Prohlášení:

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a že jsem v seznamu použité literatury uvedl všechny prameny, z kterých jsem vycházel.

V Hradci Králové dne 26. 4. 2023 .....

Tomáš Kratochvíl

Poděkování

Tímto bych rád poděkoval prof. RNDr. PhDr. Antonínu Slabému, CSc. za vedení této diplomové práce.

Anotace

KRATOCHVÍL, Tomáš. *Shluková analýza datových toků*. Hradec Králové, 2023. Diplomová práce. Univerzita Hradec Králové, Fakulta informatiky a managementu, Katedra informatiky a kvantitativních metod.

Tato diplomová práce se zabývá problematikou shlukové analýzy datových toků. V práci jsou popsány vybrané algoritmy pro shlukovou analýzu datových toků se zaměřením na kategorická data. V práci je navrženo řešení pro shlukovou analýzu e-mailů, využitelné antispamovým systémem, které vychází z vybraných algoritmů. Navržené řešení je testováno na datovém souboru vytvořeném z e-mailů, které byly vyhodnoceny antispamem za jeden den.

Klíčová slova

Shluková analýza, datový tok, aproximace četnosti, Count-Min Sketch, Lossy Counting, StreamCluCD, CSketch, NATS

Annotation

KRATOCHVÍL, Tomáš. *Data Stream Clustering*. Hradec Králové, 2023. Diploma thesis. University of Hradec Králové, Faculty of Informatics and Management, Department of Informatics and Quantitative Methods.

This diploma thesis focuses on the clustering of data streams. The thesis describes selected algorithms for clustering categorical data streams. The thesis proposes a solution for clustering emails based on the mentioned algorithms that can be used in an anti-spam system. The proposed solution is tested on a dataset created from emails evaluated by the anti-spam system in one day.

Keywords

Cluster analysis, data stream, frequency approximation, Count-Min Sketch, Lossy Counting, StreamCluCD, CSketch, NATS

Obsah

Úvod

Čím dál více aplikací produkuje a zpracovává obrovské množství dat, která přicházejí v proměnlivém a často i v těžko předvídatelném tempu. Nejznámější metody shlukové analýzy jsou navrženy především pro statické datové soubory a nepočítají s tímto v podstatě nekonečným modelem dat, jehož velikost přesahuje dostupnou paměť a není kontrola nad pořadím přicházejících objektů. Tento model se nazývá datový tok. Typickým příkladem datového toku je například síťová komunikace, finanční transakce nebo e-mailová komunikace.

E-mailová komunikace je zneužívána pro rozesílání nevyžádaného sdělení. O její zachycení se snaží antispamový systém, který využívá kombinaci různých metod. Jednou z využívaných metod je sledování reputací pro některé hodnoty obsažené v e-mailu a jejich kombinace. Mezi ně patří například IP adresa odesílatele, jeho e-mailová adresa, heš vypočítaný z obsahu e-mailu a další. Rozesílatelé nevyžádané pošty se snaží zachycení své kampaně vyhnout tím, že tyto hodnoty co nejvíce mění. Do textu e-mailu vkládají náhodně vygenerované texty nebo ho do jisté míry pozměňují, aby změnili heš vypočítaný z obsahu e-mailu. I přesto si jsou takto pozměněné e-maily velmi podobné a stále obsahují množství hodnot, které jsou stejné pro většinu e-mailů z dané kampaně nebo od daného rozesílatele. Shluková analýza by mohla být použita k seskupení takových e-mailů z jedné kampaně do jednoho shluku a na jeho ID sledovat reputace.

Navrhnout a zhodnotit řešení shlukové analýzy e-mailů, využitelné antispamovým systémem, je cílem této práce. Aby bylo možné toto řešení reálně použít, je nutné e-mail přiřadit do daného shluku před tím, než je vyhodnocen antispamem jako legitimní nebo nevyžádaná pošta. Toto přiřazení je nutné provést za relativně krátký čas a s omezenou dostupnou pamětí a výpočetními zdroji. Kromě toho je antispam distribuovaný systém složený z mnoha instancí, které běží ve více než jednom datovém centru. Tyto instance paralelně zpracovávají příchozí poštu. Bude tedy nutné navrhnout synchronizaci dat mezi jednotlivými instancemi tak, aby bylo možné provedení shlukové analýzy. Dále je nutné počítat se situacemi, které nastávají při reálném provozu aplikace. Jedná se například o její aktualizace na novější verzi nebo dočasný výpadek některé její komponenty.

První kapitola se věnuje úvodu do problematiky shlukové analýzy datových toků se zaměřením na kategorická data. V této kapitole jsou představeny obecně vyžadované vlastnosti algoritmů pro analýzu datových toků v reálném čase a je zde uveden přehled literatury, která významně ovlivnila návrh řešení v praktické části této práce.

Druhá kapitola se věnuje aproximaci četností hodnot v posloupnosti a jsou v ní popsány vybrané algoritmy pro její určení.

Třetí kapitola je věnována vybraným algoritmům pro shlukovou analýzu datových toků s kategorickými atributy. Tyto algoritmy významně ovlivnily návrh řešení.

Ve čtvrté kapitole je představeno navržené řešení pro shlukovou analýzu e-mailů. Jsou zde uvedeny vybrané atributy, popsán algoritmus pro přiřazování e-mailů do jednotlivých shluků a udržování jejich sumarizací. Je zde popsána synchronizace dat mezi jednotlivými instancemi. Dále se věnuje optimalizaci výpočtu podobnosti a nakonec nejběžnějším situacím, které nastávají v reálném provozu aplikace a jakým způsobem je možné je řešit.

V páté kapitole je navržené řešení testováno na datech z reálného provozu. Nejprve je představen použitý datový soubor, na kterém jsou následně otestovány vybrané algoritmy pro určení aproximovaných relativních četností. Následuje podkapitola věnující se evaluaci shlukování. Poté následují experimenty, ve kterých bude testováno shlukování pomocí vybraných algoritmů a navrženého řešení.

1 Úvod do problematiky

Většina nejznámějších algoritmů byla vytvořena s předpokladem, že kapacita operační paměti počítače postačí pro uložení analyzovaných dat. Z tohoto důvodu se efektivnost těchto algoritmů porovnává podle potřebného počtu elementárních operací. Latence dnešních aplikací ale často pochází především z přenosu a přístupu k datům. Provedení výpočetní operace je daleko rychlejší než přístup k datům. Systémy zpracovávající síťová data jsou distribuované a často velice komplexní. Skládají se z mnoha, mezi sebou komunikujících komponent, které si vyměňují data přes různé druhy síťové komunikace a různé databáze [15].

Právě přístup k datům je zásadní problém u analýzy datového toku. Velikost dostupné operační paměti je, vzhledem k velikosti datového toku, relativně malá. Je proto nutné využívat pouze sumarizaci vhodně vystihující jeho vlastnosti. Kvůli požadavku na rychlost zpracování není možné provádět mnoho dotazů do nějaké databáze. Na zpracování objektu je omezený čas, protože by jinak aplikace nestíhala odbavovat velké množství přicházejících dat. Při zpracovávání objektu dojde k aktualizaci použité sumarizace a poté je objekt algoritmem zapomenut nebo případně uložen do archivní databáze. Archivní databáze slouží především v případech, kdy se řeší různé incidenty nebo počítají různé ad-hoc statistiky [15]. Obecné požadavky na algoritmy pro analýzu datového toku lze shrnout do následujících bodů:

1. Čas na zpracování příchozího objektu je omezený. Algoritmus by měl ideálně garantovat maximální čas potřebný na zpracování jednoho objektu.
2. Dostupná paměť je malá vzhledem k velikosti datového toku. Algoritmus by měl používat omezené množství paměti, které se nezvětšuje s rostoucím počtem příchozích objektů.
3. Přicházející objekty je nutné zpracovávat v pořadí, v jakém přicházejí. Není možný náhodný přístup k jednotlivým objektům do paměti, protože jí není dostatek pro jejich uložení.
4. Data v datovém toku nejsou stacionární a jejich vlastnosti se časem vyvíjí. Algoritmus by měl tento vývoj v čase zohlednit.

Jedny z prvních algoritmů, které byly uvedeny jako vhodné pro shlukovou analýzu datových toků, se na danou problematiku dívají jako na variantu, kdy je potřeba objekty z datového souboru roztřídit v jednom průchodu. Jedná se tedy o jednoprůchodové algoritmy, které zvládají objekty postupně třídit do shluků v pořadí, v jakém přicházejí. Nepředpokládají však, že se vlastnosti přicházejících objektů v čase mění, a že tento proces je v podstatě nekonečný. K přiřazování do shluků používají vlastnosti získané od začátku, které už z časového hlediska nemusí být úplně relevantní [1]. Velikost použité sumarizace se s přicházejícími objekty neustále zvětšuje. To může způsobit nedostatek místa pro běh samotné aplikace a její následné selhání. Mezi takovéto jednoprůchodové algoritmy patří například algoritmus Squeezer představený v práci [21].

Squeezer pro první příchozí objekt vytvoří nový shluk. Pro každý následující příchozí objekt vypočítá jeho podobnost se všemi shluky. Následně vybere shluk, ke kterému je příchozí objekt nejvíce podobný. Přesáhne-li podobnost předem zvolenou prahovou konstantu, je objekt do daného shluku přiřazen. V opačném případě je vytvořen shluk nový. Ke každému shluku jsou udržovány četnosti hodnot vyskytujících se v objektech do něho přiřazených. Ty se používají pro výpočet podobnosti příchozího objektu s daným shlukem. Výhodou tohoto algoritmu je jeho princip přiřazovaní do shluků. Často je obtížné odhadnout pevný počet shluků a odhadnutí minimální podobnosti pro přiřazení do shluku je jednodušší. Nevýhodou je, že jeho nárok na paměť roste s počtem příchozích objektů. Kromě toho může neustále růst i počet shluků a tím dojde ke zvýšení doby potřebné na zpracování příchozího objektu [10].

Provedení shlukové analýzy na objektech obsahující kategorická data může být obtížné, pokud hodnoty jednotlivých atributů nabývají obrovského množství unikátních hodnot. V takovém případě může i sumarizace zabírat velké množství paměti. V případě datového toku je toto velmi častým problémem. Například počítání četností jednotlivých IP adres, kterých může teoreticky být až 2^{128} [2]. Analýza datového toku v reálném čase bývá prováděna především za účelem získání jakéhosi celkového přehledu. Cílem zpravidla není získat úplně přesné četnosti, ale získat k nejčetnějších hodnot nebo aproximaci četnosti těch, které překročily zvolenou konstantu. Pro přesné určení četnosti těchto hodnot je u jednoprůchodového algoritmu zapotřebí množství paměti, které roste lineárně s množstvím unikátních hodnot. Často je možné tolerovat nepřesnosti, pro jejichž velikost máme garantováno, že nepřekročí stanovenou mez. Za cenu těchto nepřesností lze dosáhnout mnohem příznivější paměťové náročnosti. V případě občasné potřeby přesných údajů jsou data ukládána do archivního úložiště [15].

Počítání podobnosti objektu se shlukem pomocí aproximovaných relativních četností využili autoři algoritmu Squeezer a v práci [10] představili nový algoritmus StreamCluCD. StreamCluCD vychází z algoritmu Squeezer a pro aproximaci relativních četností využívá algoritmu Lossy Counting [14]. Ten umožňuje získat aproximaci relativních četností hodnot, které překročily stanovenou konstantu se zvolenou, maximální možnou, relativní chybou. Jeho maximální možná paměťová náročnost roste logaritmicky. V provedených experimentech v práci [14] byla však jeho skutečná náročnost na paměť mnohem menší a zůstávala relativně konstantní. Dosahoval lepších výsledků než algoritmus Frequent [7, 12], jehož teoretická maximální složitost je konstantní. Algoritmus StreamCluCD, stejně jako algoritmus Squeezer, nebere v úvahu časový vývoj a teoretickou nekonečnost datového toku.

Podobné řešení bylo navrženo v práci [2], ve které byl představen algoritmus CSketch. Algoritmus CSketch pracuje s pevným počtem shluků a k získání aproximovaných relativních četností využívá algoritmu Count-Min Sketch [6]. V této práci bylo ukázáno, že pomocí aproximovaných četností lze dosáhnout podobných výsledků jako při použití četností přesných. Tento algoritmus opět nebere v úvahu časový vývoj a teoretickou nekonečnost datového toku. Na rozdíl od algoritmu StreamCluCD je u něj garantováno omezené množství potřebné paměti.

Pro zohlednění časového vývoje dat se nejčastěji používají metody, které do prováděných výpočtů zahrnují pouze nedávná data, spadající například do určitého okna. Tímto oknem je buď konkrétní časový interval nebo zahrnuje určitý počet objektů. Data nespadající do aktuálního okna nejsou zahrnuta do výpočtu a mohou být vymazána [15].

Použití oken v shlukové analýze kategorických dat bylo použito například v práci [4]. V této práci byl představen zobecněný rámec, který využívá již existující algoritmy pro shlukovou analýzu. Na prvním okně, které obsahuje zvolený počet objektů, je provedeno shlukování podle zvoleného, již existujícího, algoritmu. Objekty z následujícího okna jsou postupně přiřazovány do shluků, získaných předchozím shlukováním, podobně, jako v algoritmu Squeezer. Pro příchozí objekt je vypočítána podobnost se všemi shluky pomocí četností. Následně je vybrán shluk s největší podobností. Pokud tato podobnost překročí zvolenou konstantu, je objekt do daného shluku přiřazen. V opačném případě je objekt označen jako odlehlý a není přiřazen do žádného shluku. Na konci okna dojde k porovnání rozdělení shlukování v daném okně s tím, ve kterém byl použit zvolený existující algoritmus. Pokud se rozdělení do shluků výrazně liší, provede se nové shlukování zvoleným algoritmem na aktuálním okně a výsledek z předchozího je zapomenut. Autoři v této práci představili také míru podobnosti mezi shluky z dvou různých shlukování, která je použita pro vizualizaci vývoje shluků. Ačkoli je tento rámec schopen zachytit časový vývoj dat, jsou při novém shlukování zapomenuty všechny shluky, které by ještě mohli být aktuální. Tyto shluky budou sice znovu vytvořeny při novém shlukování, bude ale na ně nahlíženo jako na shluky nové. Použití tohoto rámce na analýzu datového toku má zásadní problém. Kvůli případnému provedení nového shlukování je potřeba si pamatovat všechny objekty z aktuálního okna. Toto okno by mělo zachycovat dostatečně velký časový interval, aby výsledek shlukování na daném okně dával použitelné reprezentativní výsledky. Počet přicházejících objektů v případě datového toku je obrovský a pro udržení těchto objektů nebude k dispozici dostatek paměti.

Kombinace oken a algoritmu využívajícího aproximovaných relativních četností by mohla být vhodným řešením pro shlukovou analýzu datového toku s kategorickými atributy. Následující kapitola proto bude věnována vybraným algoritmům pro určení aproximovaných relativních četností.

2 Aproximace četnosti

V této kapitole budou představeny algoritmy, které lze použít pro určení aproximované četnosti hodnot v posloupnosti, jejichž relativní četnost dosáhla zvolené prahové konstanty s maximální relativní chybou ϵ. Tento problém bude dále nazýván jako určení ϵ-aproximovaných četností.

Přesné řešení lze definovat následovně. Mějme posloupnost n hodnot $x_1, x_2, \ldots, x_n$. Absolutní četnost hodnoty i je rovna číslu $c_i = |\{j|x_j = i\}|$. Nechť $s \in (0, 1)$ je zvolená prahová konstanta. Cílem je získat četnosti všech hodnot i v posloupnosti, pro které platí:

$$c_i \geq s \cdot n. \tag{2.1}$$

Tedy pro relativní četnost hodnoty i rovnou číslu $f_i = \frac{c_i}{n}$ musí platit:

$$f_i \geq s. \tag{2.2}$$

Pro nalezení přesného řešení jednoprůchodovým algoritmem roste množství potřebné paměti lineárně s velikostí dané posloupnosti. Pro aproximativní určení těchto hodnot lze dosáhnout lepší paměťové náročnosti [5].

Problém pro nalezení ϵ-aproximovaných četností lze definovat takto. Mějme posloupnost n hodnot $x_1, x_2, \ldots, x_n$. Nechť $s \in (0, 1)$ je zvolená prahová konstanta a $\epsilon \in (0, 1)$ je zvolená maximální relativní chyba. Nechť f_i je skutečná relativní četnost hodnoty i v dané posloupnosti a $\hat{f}_i$ je její aproximace. Cílem je určit množinu hodnot F z dané posloupnosti, splňující vlastnosti (2.3) a (2.4), a jejich aproximace splňující (2.5).

$$\forall i \in F : f_i \geq (s - \epsilon) \tag{2.3}$$

$$f_i \geq s \implies i \in F \tag{2.4}$$

$$0 \leq f_i - \hat{f}_i \leq \epsilon \tag{2.5}$$

Z takto definovaného problému vyplývá, že získané aproximované relativní četnosti jsou podhodnocené nejvíce o ϵ. Některé algoritmy mohou aproximované četnosti naopak nadhodnocovat. Pro tento případ lze daný problém definovat analogicky.

2.1 Frequent

V práci [5] uvádějí, že tento algoritmus představili na sobě nezávisle v pracích [7, 12] pro nalezení k hodnot, mezi nimiž se nacházejí všechny, jejichž relativní četnost přesáhla hodnotu $\frac{1}{k+1}$. Algoritmus používá k počítadel. Pro hodnotu v posloupnosti se nejprve zjistí, jestli je přiřazena k nějakému počítadlu. Pokud ano, je toto počítadlo inkrementováno o 1. Pokud k dané hodnotě není přiřazeno počítadlo a existuje počítadlo rovné nule, je k tomuto počítadlu hodnota přiřazena. V opačném

Algoritmus: Frequent

vstup:
k: požadovaný počet nejčetnějších hodnot
X: posloupnost hodnot

začátek
$C \leftarrow$ datová struktura s k nulovými počítadly
pro každé $x_i \in X$
pokud $x_i \in C$
$C[x_i] \leftarrow C[x_i] + 1$
jinak pokud existuje $y \in C$ takové, že $C[y] = 0$
$y \leftarrow x_i$
jinak
pro každé $y \in C$
$C[y] \leftarrow C[y] - 1$
konec
konec
konec
konec

Obrázek 1: Algoritmus Frequent

případě jsou všechna počítadla dekrementována o 1. Schéma tohoto algoritmu je zobrazeno na obrázku č. 1.

Tento algoritmus nebyl navržen přímo pro určení ϵ-aproximovaných četností. V práci [3] však ukázali, že pro $\epsilon = \frac{1}{k+1}$ jsou napočítané výskyty na jednotlivých počítadlech nižší nejvíce o $\epsilon \cdot n$. Algoritmus Frequent lze tedy pro určení ϵ-aproximovaných četností použít. Asymptotická složitost na zpracování příchozí hodnoty je, po zvolení vhodných datových struktur, rovna $\mathcal{O}(1)$. Paměťová náročnost tohoto algoritmu je $\mathcal{O}(\frac{1}{\epsilon})$.

2.2 Lossy Counting

Algoritmus Lossy Counting byl představen v práci [14]. Tento algoritmus je navržený pro určení ϵ-aproximovaných četností.

Posloupnost hodnot je rozdělena do oken o velikosti závisející na zvolené, maximální možné, relativní chybě ϵ. Pro příchozí hodnotu v posloupnosti se zkontroluje, jestli pro ni existuje záznam s počítadlem. Pokud ano, je inkrementováno její počítadlo. Pokud ne, je pro ni vytvořen nový záznam s počítadlem. Tento záznam obsahuje počítadlo a index aktuálního okna snížený o 1. Na konci každého okna dojde k vyhodnocení každého uchovávaného záznamu. Pokud napočítaný výskyt konkrétní hodnoty není vyšší než počet oken, ve kterých záznam existuje, je záznam smazán. Schéma tohoto algoritmu je zobrazeno na obrázku č. 2. Jak lze v daném schématu vidět, jednotlivé záznamy jsou trojice $(x, \hat{c}_x, \Delta)$. Hodnota Δ_x v tomto záznamu

Algoritmus: Lossy Counting

vstup:
ϵ: požadovaná maximální relativní chyba
X: posloupnost hodnot

začátek
$w \leftarrow \lceil \frac{1}{\epsilon} \rceil$
$N \leftarrow 0$
$C \leftarrow$ datová struktura uchovávající záznamy $(x, \hat{c}_x, \Delta_x)$
pro každé $x_i \in X$
$N \leftarrow N + 1$
$b \leftarrow \lceil \frac{N}{w} \rceil$
pokud pro x_i existuje záznam v C
$\hat{c}_{x_i} \leftarrow \hat{c}_{x_i} + 1$
jinak
vložit $(x_i, 1, b - 1)$ do C
konec
pokud $N \bmod w = 0$
pro každý záznam $(x, \hat{c}_x, \Delta_x)$ v C
pokud $\hat{c}_x + \Delta_x \leq b$
odebrat $(x, \hat{c}_x, \Delta_x)$ z C
konec
konec
konec
konec
konec

Obrázek 2: Algoritmus Lossy Counting

představuje maximální možný počet výskytů hodnoty x před tím, než pro ní byl vytvořen tento záznam. Ten byl vytvořen v okně s indexem $\Delta_x + 1$. Hodnota $\hat{c}_x$ je počítadlo, které je aproximací absolutní četnosti hodnoty x v dané posloupnosti. Nechť f_x značí skutečnou relativní četnost hodnoty x. Hodnota $\hat{f}_x = \frac{\hat{c}_x}{N}$ pak představuje aproximaci relativní četnosti, která splňuje rovnici (2.5).

Algoritmus pro zvolenou prahovou konstantu $s \in (0, 1)$ vrátí všechny hodnoty, pro které existuje záznam a pro jejich aproximaci relativní četnosti platí:

$$\hat{f}_x \geq (s - \epsilon). \tag{2.6}$$

Autoři algoritmu uvádějí, že paměťová náročnost tohoto algoritmu je v nejhorším případě $\mathcal{O}(\frac{1}{\epsilon} \ln(\epsilon N))$. Na první pohled tedy vypadá nepoužitelně pro analýzu datového toku, protože závisí na počtu příchozích hodnot. Autoři v experimentech nicméně ukázali, že pro hodnoty pocházející z velmi šikmého rozdělení, je počet

uchovávaných záznamů významně nižší. Záznamy pro hodnoty s malou četností bývají rychle mazány a skutečný počet udržovaných záznamů v experimentech je menší než $\frac{1}{\epsilon}$. Složitost zpracování příchozí hodnoty je $\mathcal{O}(1)$, pokud se nejedná o konec okna, kde dochází k vyhodnocení a promazávání záznamů.

V práci [5] je uvedeno, že autoři algoritmu Lossy Counting v prezentaci pro svůj článek [14] představily variantu, ve které není nutné v každém záznamu uchovávat hodnotu Δ_x, ale všechny záznamy sdílejí jednu společnou hodnotu Δ. S touto variantou lze opět určit ϵ-aproximované četnosti.

2.3 Space-Saving

Algoritmus Space-Saving, představený v práci [16], je podobný algoritmu Frequent a používá k počítadel. Pro příchozí hodnotu v posloupnosti se nejprve zjistí, jestli je přiřazena k nějakému počítadlu. Pokud ano, je toto počítadlo inkrementováno o 1. Pokud ne, je přiřazena k počítadlu s nejmenší hodnotou a následně je toto počítadlo inkrementováno o 1. Schéma tohoto algoritmu je zobrazeno na obrázku č. 3. Pro velmi četné hodnoty, které jsou uchovávány s počítadlem již od začátku posloupnosti, může být aproximace četnosti velmi přesná, protože nedochází k dekrementaci. Tímto algoritmem lze určit ϵ-aproximované četnosti. Paměťová náročnost tohoto algoritmu je $\mathcal{O}(\frac{1}{\epsilon})$. Po zvolení vhodných datových struktur je asymptotická náročnost na zpracování hodnoty rovna $\mathcal{O}(1)$.

Algoritmus: Space-Saving

vstup:
k: požadovaný počet nejčetnějších hodnot
X: posloupnost hodnot

začátek
    $C \leftarrow$ datová struktura s k nulovými počítadly
    pro každé $x_i \in X$
        pokud $x_i \notin C$
            přiřaď x_i k nejmenšímu počítadlu v C
        konec
        $C[x_i] \leftarrow C[x_i] + 1$
    konec
konec

Obrázek 3: Algoritmus Space-Saving

2.4 Count-Min Sketch

Algoritmus Count-Min Sketch představený v práci [6] je založený na hešování. Algoritmus umožňuje pro příchozí hodnotu získat aproximaci její relativní četnosti s nadhodnocením, které nepřesáhne ϵ s pravděpodobností $1-\delta$. Tento algoritmus sám o sobě, bez dodatečné úpravy, neumožňuje získat výpis hodnot, jejichž aproximovaná četnost přesáhla stanovenou konstantu. Pro získání aproximované četnosti je potřeba mít k dispozici danou hodnotu pro vypočítání potřebných hešů.

Algoritmus: Count-Min Sketch

vstup:
ϵ: relativní chyba
δ: maximální požadovaná pravděpodobnost přesáhnutí chyby ϵ
X: posloupnost hodnot

začátek
    $w \leftarrow \lceil \frac{e}{\epsilon} \rceil$
    $d \leftarrow \lceil ln(\frac{1}{\delta}) \rceil$
    $H \leftarrow$ hešovací funkce $h_1, h_2, \ ... \ , h_d$
    $C \leftarrow (d \times w)$-rozměrné pole počítadel
    $N \leftarrow 0$
    pro každé $x \in X$
        $N \leftarrow N + 1$
        $\hat{c}_x \leftarrow 0$ // odhad absolutní četnosti hodnoty x
        pro každé $h_i \in H$
            $C[i, h_i(x)] \leftarrow C[i, h_i(x)] + 1$
            pokud $\hat{c}_x = 0$ nebo $C[i, h_i(x)] < \hat{c}_x$
                $\hat{c}_x \leftarrow C[i, h_i(x)]$
            konec
        konec
    konec
konec

Obrázek 4: Algoritmus Count-Min Sketch

Count-Min Sketch používá d hešovacích funkcí, které jsou po dvou nezávislé. Ke každé hešovací funkci je přiřazeno w počítadel. Tato počítadla mohou být reprezentována například maticí s d řádky a w sloupci. Čísla w a d určíme dle následujících rovnic:

$$w = \lceil \frac{e}{\epsilon} \rceil, \tag{2.7}$$

$$d = \lceil \ln(\frac{1}{\delta}) \rceil. \tag{2.8}$$

Pro příchozí hodnotu se pomocí hešovací funkce vypočítá heš představující index konkrétního počítadla a toto počítadlo je inkrementováno o 1. Při získávání aproximované četnosti pro danou hodnotu je vybráno počítadlo s nejmenší hodnotou, tedy s nejmenším počtem kolizí. Schéma tohoto algoritmu, s průběžným získáváním aproximace četnosti pro každou příchozí hodnotu, je zobrazeno na obrázku č. 4. Paměťová náročnost tohoto algoritmu je $\mathcal{O}(\frac{1}{\epsilon}ln(\frac{1}{\delta}))$ a časová složitost na zpracování hodnoty je $\mathcal{O}(ln(\frac{1}{\delta}))$.

2.5 ZG2008

V práci [22] představili Linfeng Zhang a Yong Guan jednoprůchodový algoritmus (ZG2008), který nepočítá ϵ-aproximované četnosti přes celou posloupnost, ale přes posouvající se okno o velikosti N.

Tento algoritmus si pro hodnotu udržuje dvě počítadla a záznamy, které při expiraci způsobí dekrementaci nebo smazání těchto počítadel. Při vytvoření prvního počítadla a jeho inkrementaci o 1 je vytvořen záznam, který expiruje po příchodu N hodnot a způsobí smazání prvního počítadla. Pokud toto první počítadlo dosáhne hodnoty $\frac{\epsilon N}{3}$, je druhé počítadlo inkrementováno o 1 a první počítadlo smazáno. Záznam, který měl způsobit smazání prvního počítadla, nyní při expiraci způsobí dekrementaci druhého počítadla o 1. Kromě toho fungují první počítadla jednotlivých hodnot podobně, jako počítadla v algoritmu Frequent. Počet těchto počítadel je omezen na $\frac{3}{\epsilon}$. Pokud pro příchozí hodnotu neexistuje první počítadlo a jejich počet je $\frac{3}{\epsilon}$, dojde k dekrementaci všech prvních počítadel o 1. Počítadla s nulovou hodnotou jsou následně smazány i s jejich záznamy. Pro hodnotu i je ϵ-aproximace absolutní četnosti za posledních N hodnot určena jako:

$$\hat{c}_i = cnto_i + cntc_i \cdot \frac{\epsilon N}{3}. \tag{2.9}$$

Tento algoritmus má asymptotickou paměťovou složitost $\mathcal{O}(\frac{1}{\epsilon})$. Po využití vhodných datových struktur a modulární aritmetiky je asymptotická složitost na zpracování příchozí hodnoty nebo dotazu na ϵ-aproximaci její četnosti rovna $\mathcal{O}(1)$.

3 Vybrané algoritmy

V této kapitole jsou popsány dva vybrané algoritmy, navržené pro shlukovou analýzu datových toků s kategorickými atributy, které významně ovlivnily návrh řešení.

3.1 StreamCluCD

Tento algoritmus byl představen v práci [10] a jedná se modifikaci algoritmu Squeezer [21]. Oproti algoritmu Squeezer používá pro výpočet podobnosti aproximované četnosti hodnot ve shluku, které dosáhly alespoň zvolené konstanty s. Mějme datový tok D složený z objektů $X_1, X_2, \ldots, X_n$, kde každý objekt $X_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ je tvořen m atributy. Podobnost mezi dvěma objekty X a Y je v algoritmu Squeezer definována, jako počet atributů, jejichž hodnoty jsou pro oba objekty stejné. Vypočítána je dle následující rovnice:

$$\mathrm{sim}(X, Y) = \sum_{j=1}^{m} 1 - \delta(x_j, y_j) \;, \tag{3.1}$$

kde

$$\delta(x_j, y_j) = \begin{cases} 0 & x_j = y_j \\ 1 & x_j \neq y_j. \end{cases} \tag{3.2}$$

Podobnost objektu se shlukem je definována jako průměrná podobnost objektu se všemi objekty z daného shluku. Mějme shluk C tvořený objekty $X_1, X_2, \ldots, X_n$. Potom podobnost objektu $X = (x_1, x_2, \ldots, x_m)$ se shlukem C je definována následující rovnicí:

$$\mathrm{Sim}(X, C) = \frac{1}{n} \sum_{X_i \in C} sim(X, X_i). \tag{3.3}$$

Nechť f_i je relativní četnost hodnoty x_i z objektu X ve shluku C. Pak lze podobnost objektu X se shlukem C vypočítat jako součet relativních četností dle následující rovnice:

$$\mathrm{Sim}(X, C) = \sum_{x_i \in X} f_i. \tag{3.4}$$

V algoritmu StreamCluCD se podobnost objektu se shlukem počítá stejně, jen se ve výpočtu použijí ϵ-aproximované relativní četnosti, získané algoritmem Lossy Counting, jejichž hodnota je rovna alespoň $s - \epsilon$.

Algoritmus přiřazuje objekty z datového toku do shluků v pořadí, v jakém přicházejí. Pro příchozí objekt se vypočítá jeho podobnost se všemi hluky. Z nich je vybrán shluk s největší podobností. Pokud podobnost objektu s tímto shlukem přesáhla minimální zvolenou podobnost, je objekt do daného shluku přiřazen. V opačném případě je vytvořen shluk nový. Schéma algoritmu StreamCluCD je zobrazeno na obrázku č. 5.

Algoritmus: StreamCluCD

vstup:
D: datový tok
st: minimální podobnost pro přiřazení do shluku
ϵ: maximální relativní chyba
s: zvolená prahová konstanta aproximované relativní četnosti

začátek
$C \leftarrow$ datová struktura uchovávající shluky a jejich sumarizace

pro každý objekt $X_i \in D$
pro každý shluk $C_j \in$C
výpočet podobnosti Sim(X_i, C_j)
konec

sim$_{max} \leftarrow$ největší vypočítaná podobnost
C$_{max} \leftarrow$ shluk s největší vypočítanou podobností

pokud sim$_{max} \geq st$
přiřazení X_i do shluku C_j a aktualizace sumarizace shluku C_j
udržovaná pomocí Lossy Counting
jinak
Vytvoření nového shluku a jeho sumarizace
konec

konec
konec

Obrázek 5: Algoritmus StreamCluCD

Nechť N označuje počet dosud roztříděných objektů z datového toku D, číslo k označuje aktuální počet shluků a ϵ je maximální možná relativní chyba. Autoři uvádějí, že počet uchovávaných záznamů v algoritmu StreamCluCD nepřekročí hodnotu:

$$\frac{k}{\epsilon} ln(\epsilon) + \frac{k}{\epsilon} ln(\lceil \frac{N}{k} \rceil). \tag{3.5}$$

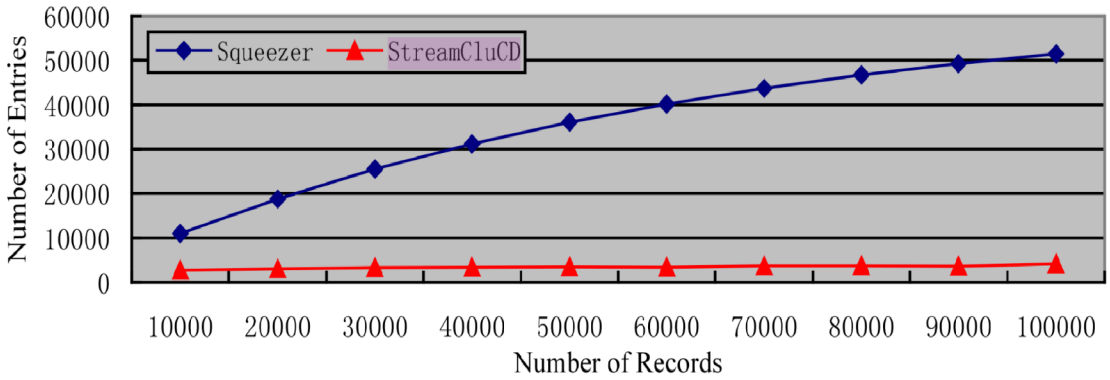
Počet uchovávaných záznamů určitě závisí na počtu atributů m tvořící příchozí objekty. Proto je odhad v rovnici (3.5) nesprávný. Pravděpodobně postačí vynásobení číslem m.

V experimentech autoři otestovali algoritmus StreamCluCD na synteticky vygenerovaném datovém souboru obsahující 4 datové toky s parametry uvedenými v tabulce č. 1.

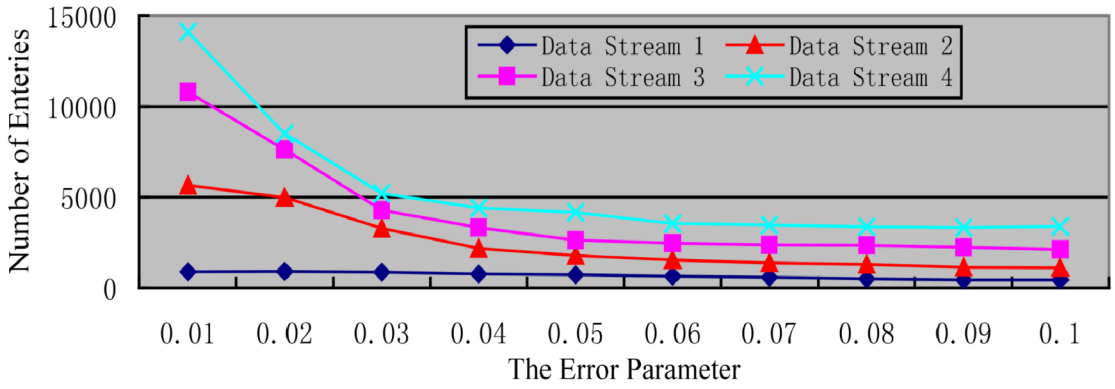
Datový tok	Počet objektů	Počet atributů	Počet tříd
1	100 000	10	10
2	100 000	20	20
3	100 000	30	30
4	100 000	40	40

Tabulka 1: Parametry vygenerovaných datových toků. Převzato a upraveno z [10]

Na obrázku č. 6 je zobrazen rozdíl v počtu uchovávaných záznamů algoritmem Squeezer, používajícím přesné četnosti, a algoritmem StreamCluCD, se zvoleným $\epsilon = 0{,}05$, u vygenerovaného datového toku 4. Na obrázku č. 7 je pak zobrazen vliv parametru ϵ na počet uchovávaných záznamů pro jednotlivé vygenerované datové toky.



Obrázek 6: Uchovávaný počet záznamů algoritmem Squeezer a StreamCluCD v závislosti na počtu roztříděných objektů. Převzato z [10]



Obrázek 7: Uchovávaný počet záznamů algoritmem StreamCLuCD v závislosti na zvolené, maximální možné, relativní chybě. Převzato z [10]

3.2 CSketch

Algoritmus CSketch, představený v práci [2], používá k výpočtu podobnosti aproximované relativní četnosti, získané pomocí algoritmu Count-Min Sketch, a objekty třídí do pevného počtu k shluků. Pro udržování sumarizací se pro každý shluk používá jeden Count-Min Sketch. Všechny mají stejný počet počítadel a používají stejné hešovací funkce, aby se indexy pro získání aproximovaných četností nemusely počítat znovu pro každý shluk.

Pro příchozí objekt se vypočítá podobnost se všemi shluky. Objekt je poté přiřazen do nejvíce podobného shluku. Poté proběhne inkrementace počítadel pro daný shluk. Výpočet indexů pomocí hešovacích funkcí neprobíhá ze samotné hodnoty konkrétního atributu. Může se stát, že některé atributy mohou nabývat stejných hodnot. Například v síťové komunikaci může být IP adresa odesílatele a příjemce považována za dva různé atributy. Ty mohou nabývat stejných hodnot. Z tohoto důvodu jsou heše počítány z textového řetězce, který vznikne spojením názvu nebo indexu atributu a jeho hodnoty. Autoři neuvádějí, jak na začátku algoritmu vytvořit jednotlivé shluky. Schéma algoritmu CSketch je zobrazeno na obrázku č. 8.

Algoritmus: CSketch

vstup:
D: datový tok
k: počet shluků
f, b: parametry určující významnou chybu
γ: s jakou maximální pravděpodobností může nastat (f, b)-významná chyba
N: pro jak velké okno platí pravděpodobnost γ
C: konstanta ovlivňující počet hešovacích funkcí a počet počítadel

začátek
    výpočet w dle rovnice (3.12)
    výpočet h dle rovnice (3.13)
    inicializace h hešovacích funkcí
    inicializace $w \cdot h$ počítadel algoritmu Count-Min Sketch pro každý z k shluků
    pro každý objekt $X_i \in D$
        pro každý shluk C
            výpočet podobnosti X_i s C
        konec
        $C_{max} \leftarrow$ shluk s největší vypočítanou podobností
        Přiřazení X_i do C_{max} a aktualizace jeho počítadel
    konec
konec

Obrázek 8: Algoritmus CSketch

Autoři uvádějí, že je důležité především zachovat uspořádání vypočtených podobností tak, aby byl objekt přiřazen do správného shluku, než velikost jejich chyby. Proto zkoumali, s jakou pravděpodobností je objekt přiřazen do shluku, do kterého by byl přiřazen s využitím přesných četností. Parametry algoritmu Count-Min Sketch se tedy volí podle této pravděpodobnosti, protože se snažíme co nejvíce napodobit algoritmus s přesným výpočtem. Určení těchto parametrů je věnován zbytek této podkapitoly.

Nechť D je datový tok složený z objektů $X_1, X_2, \ldots, X_n$, kde každý objekt X_i je tvořen d atributy. Podobnost mezi objektem $X_i = (x_1, x_2, \ldots, x_d)$ a j-tým shlukem C_j je definovaná, jako součet relativních četností hodnot x_r v daném shluku dle následující rovnice:

$$D^j(X_j) = \sum_{r=1}^{d} f_{rj}. \tag{3.6}$$

Nechť $\hat{f}_{rj}$ je aproximovaná relativní četnost hodnoty x_r ve shluku C_j získaná pomocí algoritmu Count-Min Sketch. Označme D_{ij} podobnost objektu X_i se shlukem C_j vypočítanou pomocí těchto aproximovaných četností, která je vypočtena následovně:

$$D_{ij} = \sum_{r=1}^{d} \hat{f}_{rj}. \tag{3.7}$$

Nechť bylo doposud roztříděno N objektů. Nechť číslo F_p označuje, jakou část z těchto objektů bylo přiřazeno do shluku C_p. Nechť pro vypočtenou podobnost objektu X_i pomocí aproximovaných četností se shluky C_p a C_q platí:

$$D_{ip} \geq D_{iq} + b, \tag{3.8}$$

potom s pravděpodobností alespoň $1 - (d^2/(b \cdot F_p \cdot w))^h$ platí pro přesné podobnosti následující nerovnice:

$$D^p(X_i) \geq D^q(X_i). \tag{3.9}$$

Nechť C_p je shluk s největší vypočtenou podobností pomocí aproximovaných četností a pro všechny ostatní shluky C_q platí:

$$D_{ip} \geq D_{iq} + b_q, \tag{3.10}$$

potom je objekt správně přiřazen do shluku C_p s pravděpodobností alespoň $1 - \sum_{p \neq q}(d^2/(b_q \cdot F_p \cdot w))^h$.

Pro určení počtu hešovacích funkcí h a velikosti pole počítadel w autoři nejprve definovali (f, b)-významnou chybu. Nechť C_q je shluk, do kterého by měl být objekt X_i správně přiřazen. Tato chyba znamená, že objekt byl nesprávně přiřazen do shluku C_p, který obsahuje část f z dosud roztříděných objektů, a zároveň platí rovnice (3.8). Autoři tvrdí, že objekt může být ve skutečnosti podobný s více shluky, proto chyba přiřazení způsobená malými b_q není tak významná. Dále jsou zajímavé

především početnější shluky, které obsahují větší množství objektů, a tedy hlavně do nich nechceme špatně přiřazovat. Pravděpodobnost, že tato chyba nastane při přiřazování objektu do shluku, je rovna číslu $k \cdot (d^2/(b \cdot f \cdot w))^h$. Pravděpodobnost, že po roztřídění N objektů nastala alespoň jedna (f, b)-významná chyba, není vyšší než:

$$\gamma = \frac{N \cdot k}{(b \cdot f \cdot w/d^2)^h}. \tag{3.11}$$

Počet hešovacích funkcí h a velikost pole počítadel w je možné navrhnout tak, aby byly splněny rovnice (3.13) a (3.12) pro zvolené $C > 1$. Potom bude pravděpodobnost, že nenastane ani jedna (f, b)-významná chyba, rovna alespoň číslu $1 - \gamma$.

$$w = \frac{C \cdot d^2}{b \cdot f} \tag{3.12}$$

$$h = \frac{\ln(N) + \ln(k) + \ln(1/\gamma)}{\ln(C)} \tag{3.13}$$

Celkový počet počítadel je roven číslu $M = w \cdot h$. Po dosazení dostaneme následující rovnici pro celkový počet počítadel:

$$M = \frac{C \cdot d^2 \cdot (\ln(N) + \ln(k) + \ln(1/\gamma))}{\ln(C) \cdot b \cdot f}. \tag{3.14}$$

Nejmenšího počtu počítadel je dosaženo pro $C = e$. Zvolením většího C snížíme počet hešovacích funkcí za cenu zvýšeného počtu počítadel. Není vhodné volit $C \in (1, e)$, protože dostaneme více hešovacích funkcí a dohromady více počítadel.

Autoři uvádějí, že N nemusí být nutně počet všech objektů v datovém toku. Může představovat okno, přes které je garantováno, že (f, b)-významná chyba nenastane s pravděpodobností alespoň $1 - \gamma$.

4 Navržené řešení

V této kapitole bude představeno navržené řešení pro shlukovou analýzu využitelné antispamovým systémem. Bude zde popsán výběr proměnných, algoritmus pro přiřazování e-mailů do shluků a jejich udržování včetně synchronizace mezi jednotlivými instancemi paralelně vyhodnocujícími elektronickou poštu. Dále zde bude popsáno, jakým způsobem by bylo možné řešit některé situace, které nastávají při reálném provozu aplikace v produkčním prostředí, jako je například její aktualizace na novější verzi. Toto řešení je navrženo pro antispamový systém, který provozuje společnost Seznam.cz pro svoji freemailovou službu.

Společnost Seznam.cz je největší český poskytovatel freemailové služby. K zachycení nevyžádané pošty využívá vlastní antispamové řešení. Za jeden den je servery zpracováno okolo 70 milionů zpráv, což je průměrně kolem 810 zpráv za sekundu. Zhruba 15 % je antispamem vyhodnocena jako nevyžádaná pošta. V současnosti je antispam provozován ve dvou datových centrech. Skládá se z několika stovek běžících instancí, které paralelně zpracovávají přijaté zprávy, čekající ve frontě na doručení. Antispam je provozován jako kontejnerová aplikace, pomocí technologie Docker [9], která je spravována pomocí orchestračního systému pro správu aplikačních kontejnerů Kubernetes [13].

Docker kontejner umožňuje vytvořit balíček aplikace a pustit ji v izolovaném prostředí. Tyto kontejnery obsahují všechno potřebné pro běh aplikace a nezáleží na tom, jaké závislosti jsou nainstalované na hostitelském počítači. Jedinou podmínkou pro spuštění aplikace je mít nainstalovanou technologii Docker [8].

Kubernetes řídí spouštění kontejnerů a přiděluje jim výpočetní prostředky. Kontejnery jsou spojeny do podů, které jsou základními jednotkami v Kubernetes, a ty se škálují dle požadovaného stavu. Pod může být například tvořen dvěma kontejnery, z nichž jeden odbavuje požadavky a druhý zpracovává logy vytvořené prvním kontejnerem a ukládá je dle potřeby do nějaké databáze. Kubernetes umožňuje vyrovnávat zatížení, sleduje využití prostředků a na základě toho dokáže i dodatečné prostředky přidělovat nebo ubírat. Umožňuje také aplikacím, aby se v případě potřeby samy opravily pomocí automatického restartování jednotlivých kontejnerů nebo celého podu [17].

Při vyhodnocování zprávy antispamem jsou ve stanoveném pořadí pouštěny moduly, na základě jejichž výstupu je zpráva vyhodnocena. Řešení v této práci je navrženo tak, aby jej bylo možné implementovat jako modul do tohoto antispamového systému. Tyto moduly jsou napsány v jazyce Go, proto bude použit pro implementaci navrženého řešení.

Programovací jazyk Go byl vytvořen ve společnosti Google Inc. Jedná se o staticky typovaný jazyk kompilovaný do strojového kódu. Mezi jeho výraznou vlastnost patří snadné paralelní programování částí programů pomocí tzv. goroutines. Tímto jazykem je napsaný zdrojový kód například právě projektů docker a kubernetes.

4.1 Vybrané atributy

Elektronická pošta, neboli e-mail, umožňuje uživateli odeslat zprávu ostatním uživatelům internetu. Elektronická pošta je v některých ohledech podobná jako normální písemná pošta [11]. Skládá se ze tří částí:

1. Obálka (Envelope).
2. Hlavička (Header).
3. Tělo (Body).

Obálka obsahuje potřebné údaje pro doručení e-mailu. Obsahuje jednu nebo více e-mailových adres příjemců, kterým má být zpráva doručena. Dále obsahuje adresu odesílatele, na kterou se doručí případně reporty o nedoručení. Nejedná se však o adresu, kterou vidí příjemce dané zprávy, ta je specifikovaná v hlavičce e-mailu. Příjemci e-mailu je zobrazen "obsah obálky", tedy některé informace specifikované v hlavičce a tělo e-mailu. V hlavičce jsou obsažené položky, z nichž některé jsou povinné. Mezi povinné patří například datum a údaje, od koho je daná zpráva, tj. adresa odesílatele zprávy a primární příjemce zprávy. Mezi volitelné pak patří například předmět zprávy nebo komu má být poslána kopie zprávy. Tělo e-mailu je odesílanou zprávou, která je zobrazena příjemci [11].

Atributy reprezentující objekt mají významný vliv na výsledek shlukové analýzy. Výběr atributů, použitých v navrženém řešení, probíhal expertním odhadem členů týmu provozující antispam. Vybrané atributy jsou zobrazeny a popsány v tabulce č. 2.

U shlukové analýzy s kategorickými atributy je, ve většině případů, objekt definován jako vektor o pevné velikosti. Jeho jednotlivé položky odpovídají hodnotě vybraných atributů a ty nabývají pouze jedné hodnoty. U některých z vybraných atributů, použitých v této práci, se v jednom e-mailu může vyskytovat i vícero hodnot. Také nemusí být přítomná žádná. Proto je objekt v navrženém řešení reprezentován množinou hodnot jednotlivých atributů a ne jako vektor. Velikost této množiny je proměnlivá. To významně ovlivnilo výběr algoritmu pro uchovávání sumarizace, protože některé nelze pro tento případ použít nebo modifikovat.

U některých vybraných atributů se mohou vyskytovat stejné hodnoty. Může se tak stát například u slov z předmětu e-mailu a jeho textu. Je tedy nutné rozlišit, zda se jedná o slova nacházející se v předmětu nebo v samotném textu daného e-mailu. Pro toto rozlišení je použito stejné řešení, jako u algoritmu CSketch. Místo samotných hodnot atributů se spojí název nebo index atributu a jeho hodnoty do jednoho textového řetězce.

Hodnoty některých atributů mohou být celkem dlouhé textové řetězce. Může se jednat například o adresy URL. V takovém případě je možné jejich délku omezit. Můžeme vzít prvních x znaků a ty spojit s hešem vypočítaným ze zbylých znaků. U algoritmu Count-Min Sketch není třeba dlouhé textové řetězce řešit.

Název	Popis
efrom.local	Místní část obálkové adresy odesílatele
efrom.domain	Internetová doména obálkové adresy odesílatele
hfrom.local	Místní část hlavičkové adresy odesílatele
hfrom.domain	Internetová doména hlavičkové adresy odesílatele
sender_name	Jméno odesílatele zobrazené příjemci
subject	Unikátní slova obsažená v předmětu e-mailu
words	Unikátní slova vyskytující se v prvních 150 a posledních 150 bytech těla e-mailu
ip	IP adresa odesílatele
ip_crange	Rozsah IP adresy odesílatele
country	Země, ze které byla zpráva odeslána. Jedná se o dvoupísemný kód dané země podle ISO 3166-1 alpha-2
lang	Použitý jazyk, který je v textu detekován
theme	Téma daného e-mailu přiřazené klasifikátorem v jednom z modulů antispamu (transakční, newsletter, atd ...)
urls	Unikátní adresy URL vyskytující se v e-mailu. Jejich počet je omezen na maximální množství

Tabulka 2: Vybrané atributy

4.2 Přiřazování do shluků

Algoritmus pro třídění příchozích zpráv do jednotlivých shluků vychází z algoritmu StreamCluCD. Nejprve se zvolí konstanty s a ϵ z intervalu $(0, 1)$. Z každé příchozí zprávy se získají hodnoty zvolených atributů reprezentující objekt. K výpočtu podobnosti objektu se shlukem se použijí ϵ-aproximované relativní četnosti hodnot vyskytujících se v daném shluku, definované v kapitole č. 2. Tyto aproximované četnosti se nepočítají ze všech objektů přiřazených do daného shluku. Počítají se pouze ze zvoleného počtu objektů, které byly do daného shluku přiřazeny naposledy. Je tedy definováno okno obsahující zvolený počet naposledy přidaných objektů. Získané aproximované relativní četnosti jsou dále vynásobené vahou. Hodnoty vyskytující se v mnoha shlucích tak budou mít na přiřazení objektu do shluku menší vliv. Počet shluků je omezený na zvolený maximální počet K.

Nechť $x = \{x_1, x_2, \ ... \ , x_n\}$ je množina získaných hodnot z příchozí zprávy. Číslo n udává počet získaných hodnot, který může pro každou zprávu nabývat jiné hodnoty. Nechť $\hat{f}_i$ je aproximovaná relativní četnost hodnoty x_i ve shluku C. Podobnost objektu x se shlukem C se vypočítá dle následující rovnice:

$$\mathrm{sim}(x, C) = \sum_{x_i \in \hat{x}} \hat{f}_i \cdot w_i. \tag{4.1}$$

Množina $\hat{x}$ obsahuje hodnoty z x, pro které je aproximovaná relativní četnost $\hat{f}_i$ rovna alespoň $s - \epsilon$. Váha $w_i \in \langle 0, 1 \rangle$ se určí dle rovnice (4.3). Výpočet této váhy je inspirován metodou TF-IDF, která se používá pro převod textového dokumentu na vektor. V této metodě se určuje váha slova pomocí převrácené četnosti ve všech dokumentech. Jedna z variant výpočtu této váhy je podle následující rovnice:

$$\mathrm{idf} = \ln(\frac{|D|}{d}). \tag{4.2}$$

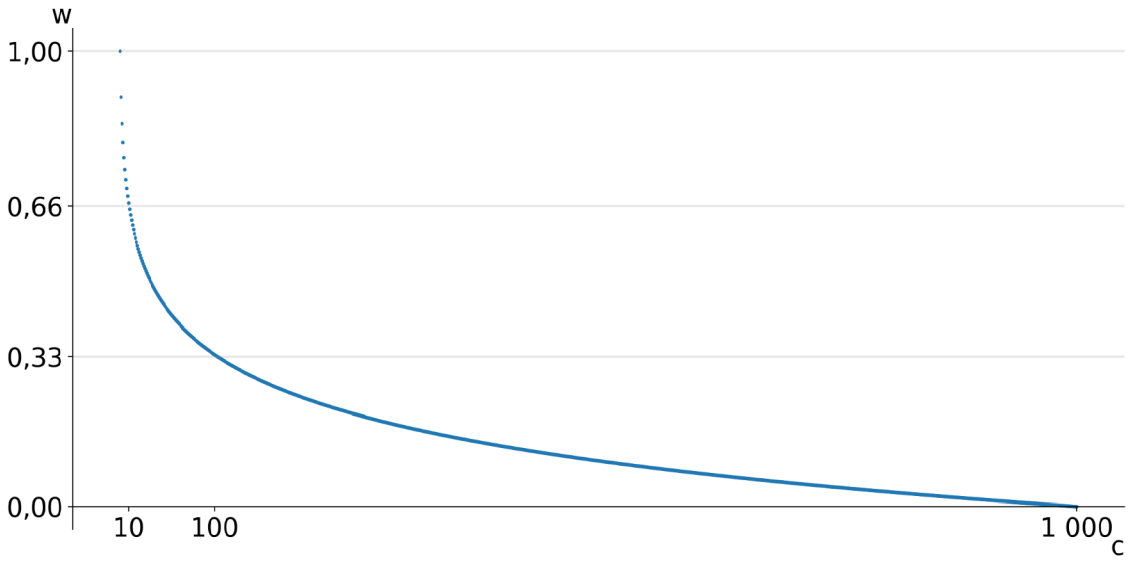
Číslo D představuje celkový počet dokumentů v datovém souboru. Číslo $d > 0$ je počet dokumentů, ve kterém se dané slovo vyskytuje. Hodnota takto vypočítané váhy nabývá hodnot v intervalu $\langle 0, \ln(|D|) \rangle$. Je možné ji normalizovat na interval $\langle 0, 1 \rangle$, pokud je vydělena číslem $\ln(|D|)$.

Nechť k je celkový počet shluků. Číslo $c > 0$ je počet shluků, ve kterém aproximovaná relativní četnost $\hat{f}_i$ hodnoty x_i dosáhla alespoň $s - \epsilon$. Její váha w_i se vypočítá dle následující rovnice:

$$w_i = \frac{\ln(\frac{k}{c})}{\ln(k)} = 1 - \log_k(c). \tag{4.3}$$

Pro $c = 0$ položíme váhu rovnou nule. Takto vypočítaná váha má následující vlastnost. Nechť je aktuální počet shluků k roven například číslu 1000. Pro $c = 1$, je váha w rovná číslu 1. Pokud c vzroste na 10, je váha rovna 0,67. Došlo tedy k poklesu o 0,33. Pokud je $c = 500$ a vzroste o 9 jako v předchozím případě, dojde k výrazně menšímu snížení váhy a to o zhruba 0,003. Vliv na výslednou podobnost je u hodnot s $c = 500$ a $c = 510$ přibližně stejný, zatímco u hodnot s $c = 1$ a $c = 10$ je rozdíl v jejich vlivu na podobnost mnohem větší. Graf funkce pro výpočet váhy je zobrazen na obrázku č. 9.

Pro příchozí objekt se vybere shluk s největší podobností a pokud je dosaženo zvolené minimální podobnosti, je objekt přiřazen do daného shluku. V opačném případě může dojít k vytvoření nového shluku. Pokud je dosažen maximální možný počet shluků, dojde k jejich promazání. Procesu udržování sumarizací, jejich synchronizaci, vytváření nových shluků a mazání starých, se věnuje samostatně podkapitola č. 4.4. V následující podkapitole budou popsány dva navržené způsoby pro udržování sumarizací shluků, tedy pro určení aproximovaných relativních četností hodnot ve shlucích.



Obrázek 9: Graf funkce pro výpočet váhy

4.3 Sumarizace

V kapitole č. 2 byly představeny jednoprůchodové algoritmy, které lze použít pro určení ϵ-aproximovaných četností v posloupnosti hodnot. V případě udržování sumarizací jednotlivých shluků se nejedná o posloupnost hodnot, ale posloupnost objektů. Tyto objekty jsou reprezentované množinou hodnot jednotlivých atributů. Pro určení ϵ-aproximovaných četností v tomto případě nelze použít algoritmy Frequent, Space-Saving a ZG2008. Ty by bylo možné použít v případě, kdy by byl objekt reprezentován vektorem o pevné velikosti, kde by každá složka tohoto vektoru odpovídala hodnotě jednoho konkrétního atributu. Pak by se tyto algoritmy mohly použít zvlášť pro každou složku vektoru.

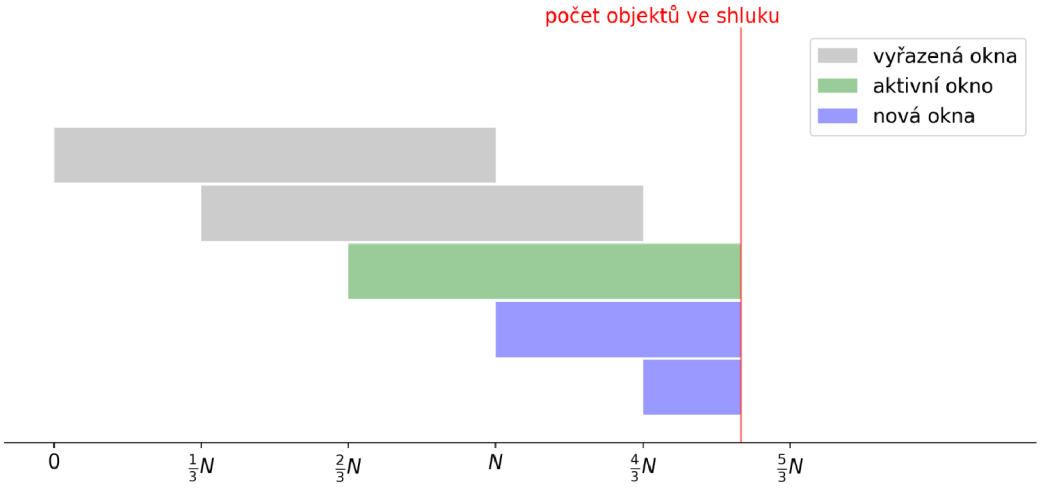
Pro určení ϵ-aproximovaných četností lze v tomto případě použít algoritmy Lossy Counting a Count-Min Sketch. U algoritmu Count-Min Sketch se aproximovaná absolutní četnost, stejně jako v algorimtmu CSketch, vydělí počtem objektů v daném shluku. U algoritmu Lossy Counting neudává velikost okna w počet inkrementací hodnot, ale počet objektů. Stejný způsob je použit v algoritmu StreamCluCD.

Jak již bylo řečeno v podkapitole č. 4.2, aproximované četnosti hodnot se v daném shluku počítají pouze z posledních přidaných objektů. Pro takové počítání jsou navrženy dva způsoby. První způsob používá tzv. posuvná, překrývající se okna (sliding overlapping windows). Druhým způsobem je úprava algoritmu Lossy Counting inspirovaná algoritmem ZG2008. Ta počítá aproximované četnosti hodnot ve shluku z posledních N přidaných objektů. U druhého způsobu není zaručeno, že nedojde k překročení maximální zvolené relativní chyby ϵ. V provedeném experimentu však nebyla nikdy překročena.

4.3.1 Sliding overlapping windows

U prvního způsobu je algoritmus pro počítání ϵ-aproximovaných četností proveden pouze pro okno o velikosti N objektů. Jakmile počet objektů v daném okně dosáhne

čísla N, je sumarizace vzniklá z tohoto okna vyřazena. Aby nedošlo k úplné ztrátě a počítání četností úplně od začátku z dalších příchozích objektů, probíhá výpočet četností pro víc oken zároveň. Mezi těmito okny existuje rovnoměrný rozestup tak, aby bylo dosaženo co nejmenší ztráty po vyřazení nejstaršího okna. Tyto okna se tedy navzájem překrývají a k určení aproximovaných relativních četnosti je vždy použito to nejstarší z nich. Čím víc oken je použito, tím menší ztráty je dosaženo. Například při použití 3 oken se velikost nejstaršího okna pohybuje v rozmezí $\langle \frac{2}{3}N, N \rangle$. S rostoucím počtem oken ale také roste velikost potřebné na paměti. Je možné použít algoritmus Lossy Couting i algoritmus Count-Min Sketch. Tento způsob je zobrazen na obrázku č. 10.



Obrázek 10: Posuvná překrývající se okna

4.3.2 Sliding Lossy Counting

Druhým způsobem je úprava algoritmu Lossy Counting inspirovaná algoritmem ZG2008. Aproximované relativní četnosti jsou udržovány přes posouvající se okno o velikosti N. Stejně jako v algoritmu ZG2008 se pro danou hodnotu udržují dvě počítadla a záznamy, které při expiraci způsobí dekrementaci nebo smazání těchto počítadel.

Posloupnost objektů je rozdělena na části o velikosti $w = \lceil \frac{1}{\epsilon} \rceil$ stejně jako u algoritmu Lossy Counting. N volíme jako m násobek čísla w, tj. $N = m \cdot w$. Na rozdíl od algoritmu Lossy Counting dojde na konci každé části k dekrementaci prvních počítadel. Po přiřazení N objektů do shluku je počítadlo dekrementováno celkem m krát. Pro danou hodnotu je tedy možné udržovat aproximované četnosti za toto okno, pokud její počítadlo nepřekročí hodnotu m. V případě že je tato hodnota překročena, samotné dekrementace nepostačí pro udržování četnosti za dané okno. V takovém případě je první počítadlo vynulováno a druhé počítadlo inkrementováno o 1. Dále je vytvořen záznam, který expiruje po přiřazení N objektů do daného shluku a způsobí dekrementaci druhého počítadla. Aproximovaná absolutní četnost $\hat{c}_i$ je potom určena dle následující rovnice:

$$\hat{c}_i = cnto_i + cntc_i \cdot m. \tag{4.4}$$

Číslo $cnto_i$ je hodnota prvního počítadla a číslo $cntc_i$ hodnota druhého. Pokud budou po dekrementaci obě počítadla pro konkrétní hodnotu rovna nule, dojde ke smazání záznamu pro tuto hodnotu. Schéma tohoto algoritmu je zobrazeno na obrázku č. 11.

4.4 Synchronizace

Pro výpočet podobnosti objektu se shlukem se používají synchronizované sumarizace, které má každá z instancí uložené v paměti RAM. Hodnoty v těchto synchronizovaných sumarizacích nejsou inkrementovány po přiřazení objektu do shluku. Nebylo by totiž možné použít algoritmy pro určení ϵ-aproximované relativní četnosti, pokud bude docházet k inkrementaci na vícero instancí zároveň. Proto je udržování sumarizací jednotlivých shluků rovnoměrně rozděleno mezi všechny běžící instance tak, aby byla sumarizace pro konkrétní shluk udržována pouze na jedné instanci.

Po přiřazení objektu do shluku jsou poslána data pro inkrementaci na instanci, která udržuje sumarizaci pro daný shluk. Ta po splnění určité podmínky rozešle sumarizaci na všechny ostatní instance, která již bude použita pro výpočet podobnosti. Sumarizace použité pro výpočet podobnosti mají menší velikost, protože nemusí obsahovat všechny hodnoty nutné pro jejich udržování.

V případě použití překrývajících se oken stačí použít pouze aktuální okno. Pokud je využito algoritmu Lossy Counting, je poslána sumarizace obsahující jednotlivé hodnoty a jejich relativní četnost. Tato sumarizace obsahuje pouze hodnoty, jejichž relativní četnost dosáhla stanovené prahové konstanty. Většina udržovaných hodnot této prahové konstanty nedosáhne a při výpočtu podobnosti nemají vliv na výslednou podobnost objektu se shlukem. Hodnot, jejichž relativní četnosti dosáhla stanovené prahové konstanty, je mnohem méně a odstraněním těchto málo četných hodnot dojde k výrazné úspoře místa.

Při použití algoritmu Count-min Sketch je nutné poslat všechny počítadla pro aktuální okno. Pro algoritmus Count-min Sketch je použita implementace z [20], která pro inicializaci hešovacích funkcí používá jejich index jako seed. Je tak zaručeno použití stejných hešovacích funkcí napříč všemi instancemi.

Při využití úpravy algoritmu Lossy Counting s použitím expiračních záznamů a dvou počítadel se na všechny instance rozešle sumarizace obsahující hodnoty s jejich relativními četnostmi, které dosáhly stanovené prahové konstanty.

4.4.1 Komunikace mezi instancemi

Komunikace mezi jednotlivými instancemi je zajištěna formou zpráv. Technologie umožňující komunikaci pomocí zasílání zpráv se nazývá Message Oriented Middleware (MOM). Pro zasílání zpráv je použita technologie NATS [18]. NATS je bezplatná technologie poskytována pod Apache-2.0 Open Source licencí, která je již využívána jinými moduly v antispamovém systému, pro který je toto řešení navrženo. Proto bude použita i pro komunikaci mezi instancemi v navrženém řešení.

NATS implementuje model publish-subscribe. Odesílatel posílá zprávy na konkrétní předmět. Ty jsou doručeny všem příjemcům, kteří se přihlásili o odběr zpráv z

Algoritmus: Sliding Lossy Counting

vstup:
ϵ: požadovaná maximální relativní chyba
m: násobek, pomocí kterého se určí velikost okna N
X: posloupnost objektů

začátek
$w \leftarrow \lceil \frac{1}{\epsilon} \rceil$
$N \leftarrow m \cdot w$
$S \leftarrow 0$ // počet zpracovaných hodnot
$C \leftarrow$ datová struktura uchovávající záznamy $(x_i, \text{cnto}_i, \text{cntc}_i)$
$P \leftarrow$ datová struktura uchovávající expirační záznamy

pro každý objekt $x \in X$
$S \leftarrow S + 1$
$b \leftarrow \lceil \frac{N}{w} \rceil$

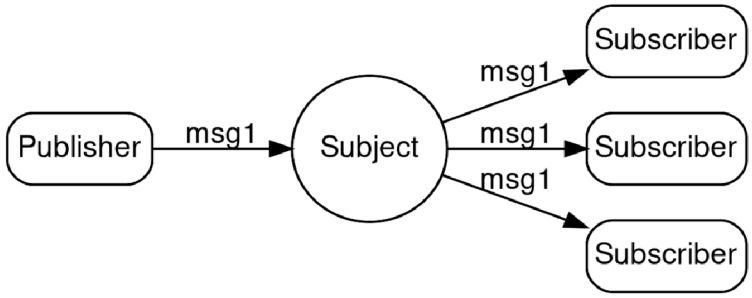
pro každé $x_i \in x$
pokud pro x_i existuje záznam v C
$\text{cnto}_i \leftarrow \text{cnto}_i + 1$
pokud $\text{cnto}_i > m$
$\text{cnto}_i \leftarrow 1$
$\text{cntc}_i \leftarrow \text{cntc}_i + 1$
vložit expirační záznam (x_i, S) do P
jinak
vložit $(x_i, 1, 0)$ do C
konec
konec

dokud existuje záznam v P
$(x_i, \hat{S}) \leftarrow$ nejstarší záznam z P
pokud $S - \hat{S} \geq N$
$\text{cntc}_i \leftarrow \text{cntc}_i - 1$
odebrat tento záznam z P
jinak
přerušit cyklus
konec
konec

pokud $S \bmod w = 0$
pro každý záznam $(x_i, \text{cnto}_i, \text{cntc}_i)$ v C
pokud $\text{cnto}_i > 0$
$\text{cnto}_i \leftarrow \text{cnto}_i - 1$
konec
$\hat{c}_i = \text{cnto}_i + \text{cntc}_i \cdot m$
pokud $\hat{c}_i = 0$
odebrat záznam pro x_i z C
konec
konec
konec
konec
konec

Obrázek 11: Algoritmus Sliding Lossy Counting

daného předmětu. Příjem a doručovaní zpráv zajišťuje NATS server. Toto schéma je zobrazeno na obrázku č. 12. Pro použití ve své aplikaci mohou vývojáři využít klientské knihovny. Tyto knihovny jsou k dispozici například pro programovací jazyky Java, Python, C, Go, Ruby nebo JavaScript. Klientské knihovny posílají zprávy způsobem fire-and-forget. Jakmile je zpráva odeslána na server, odesílatel nečeká na potvrzení o doručení zprávy příjemcům. Nats server garantuje úroveň doručení zpráv označovanou jako „At most once“, což znamená nejvýše jednou. Tato úroveň má velmi malou datovou režii.



Obrázek 12: Model publish-subscribe. Převzato z [19]

Kromě modelu publish-subscribe podporuje NATS i modely request-reply a queue groups. Ty však nejsou pro navržené řešení použity. NATS dále umožňuje použít nadstavbu JetStream, která přidává další funkcionality. Jednou z těchto funkcionalit je použití databáze klíč-hodnota. Tato databáze se v navrženém řešení využívá pro určení instance, na kterou mají být odeslány hodnoty pro aktualizaci sumarizace daného shluku. Databáze klíč-hodnota je dále využívána pro ukládání sumarizací, které si jednotlivé instance stahují. Na základě těchto sumarizací se poté vypočítává podobnost mezi daným objektem a shluky.

V této databázi se vytvářejí segmenty nazývané „buckets“. Do segmentu se ukládají záznamy klíč-hodnota a lze v něm nastavit maximální velikost hodnot nebo jejich expirační doba. Je možné provádět tyto operace:

- **put** – slouží k vytvoření záznamu nebo aktualizaci jeho hodnoty
- **get** – získání hodnoty podle daného klíčem
- **create** – vytvoří záznam, pokud záznam s daným klíčem neexistuje
- **update** – aktualizace hodnoty klíče
- **delete** – odstraní záznam podle daného klíče
- **keys** – slouží k získání všech klíčů nacházejících se v daném segmentu

Kromě těchto standardních operací, běžných u databází klíč-hodnota, je možné sledovat a v reálném čase přijímat změny, které se vyskytnou u konkrétního klíče nebo u všech klíčů v daném segmentu. Při sledování změn je možné nastavit, zda chceme přijímat pouze metadata změněného záznamu nebo také jeho hodnotu.

4.4.2 Určení instance pro udržování konkrétního shluku

Jednotlivé instance antispamového systému běží jako kontejnerová aplikace v technologii Docker, které jsou spravovány pomocí orchestračního systému Kubernetes. Jedna instance této aplikace je v Kubernetes reprezentována jako tzv. pod. Každé této základní jednotce je přiřazen název. Příklad těchto názvů je zobrazen na obrázku č. 13. Tento název je pro každý pod unikátní. Je proto možné ho při komunikaci mezi jednotlivými instancemi použít jako jednoznačný identifikátor. Ke svému názvu může instance přistoupit pomocí proměnného prostředí.

Obrázek 13: Příklad názvů instancí

Každá z instancí bude přijímat data pro inkrementaci hodnot v udržované sumarizaci daného shluku tak, že bude přihlášena pro odběr zpráv na předmětu představující její název. Ten bude dále rozšířen o prefix společný pro všechny instance. Tento prefix bude sloužit pro rozlišení, kdyby se názvy instancí používali také pro jiný účel již existujícími moduly.

Aby bylo možné rozesílat data pro inkrementace na ostatní instance, musí mít každá instance k dispozici názvy všech ostatních. K získání těchto názvů se použije databáze klíč-hodnota, kterou nabízí nadstavba JetStream. V této databázi bude vytvořen segment pro tento účel, do kterého instance vloží klíč představující její název. Hodnota pro tento případ může zůstat prázdná. Každá z instancí bude sledovat změny v tomto segmentu v reálném čase a v pravidelných intervalech aktualizovat svůj klíč. Díky pravidelné aktualizaci bude možné sledovat, zda je daná instance stále aktivní. Každá z instancí si tak bude lokálně udržovat seznam všech aktivních instancí.

Po přiřazení objektu do shluku se předmět, na který budou odeslány hodnoty pro inkrementaci jeho sumarizace, určí následujícím způsobem. Seznam názvů aktivních instancí je seřazen abecedně. Z ID shluku se vypočítá index pomocí hešovací funkce v rozsahu od 0 do $N - 1$, kde N představuje počet aktivních instancí. Hodnoty pro inkrementaci jsou poté odeslány na předmět představující název instance nacházející se v seřazeném seznamu na daném indexu.

Tento způsob předpokládá, že se seznam těchto instancí nebude měnit často. V současném antispamovém řešení dochází k přeškálování počtu instancí spíše výjimečně. K výpadkům jednotlivých instancí, a jejich nahrazení novými, také téměř nedochází. Pokud dojde k výpadku jedné z instancí, a jejímu nahrazení novou nebo k přeškálování, dojde ke změně pořadí v udržovaném seznamu. To bude mít za následek nové rozdělení udržování sumarizací. Další možnou situací, kdy dojde k novému rozdělení, je aktualizace samotné aplikace. To má za následek postupné

nahrazování všech instancí těmi s novou verzí aplikace. Řešením těchto situací se samostatně zabývá část č. 4.6.

4.4.3 Synchronizace sumarizací

K synchronizaci sumarizací bude, stejně jako v případě udržování seznamu aktivních instancí, použita databáze klíč-hodnota nadstavby JetStream. K tomuto účelu bude využit samostatný segment, do kterého budou pod klíčem, představující ID shluku, nahrávány aktualizované sumarizace pro výpočet podobnosti. Každá z instancí bude v tomto segmentu sledovat změny a v reálném čase dostávat informaci o změnách klíčů a jeho hodnot. Po aktualizaci sumarizace obdrží všechny instance zprávu, že došlo ke změně hodnoty asociované ke konkrétnímu klíči. V této zprávě je obsažena nová hodnota konkrétního klíče a instance tak může aktualizovat sumarizaci daného shluku.

Jednotlivé instance budou do databáze posílat aktualizace jednou za specifikovaný interval. Při posílání aktualizace po každé inkrementaci, a jejímu rozesílání na všechny instance, by mohlo dojít k přetížení databáze. Posílání aktualizací by mohlo probíhat i sofistikovanějším způsobem, a to například pouze v případě, pokud by v ní došlo k výraznějším změnám. Nicméně způsob aktualizací je úzce svázán s mazáním starých při dosažení maximálního počtu shluků a návrh takového řešení je velmi obtížný bez testování řešení v reálném provozu aplikace.

4.4.4 Nové shluky

Experimenty, ve kterých byly nové shluky vytvářeny při nedosažení minimální podobnosti, ukázaly, že do téměř poloviny vytvořených shluků nedojde k přiřazení více než 10 e-mailů. E-maily přiřazené do těchto shluků tvoří pouze malou část celkového provozu, protože většina je přiřazována do těch početnějších. Nevyžádaná pošta je rozesílána převážně v mnohem větším množství a proto nejsou takto málo početné shluky, z hlediska sledování reputací, zajímavé a přitom zabírají polovinu použitého místa v paměti. To by mohlo být lépe využito pro sumarizace početnějších shluků, které by díky menšímu množství promazávání déle vydržely. Proto budou objekty přiřazené do takto málo početných shluků brány z hlediska shlukové analýzy jako odlehlé hodnoty a vytváření nových shluků je navrženo takovým způsobem, aby nedocházelo tak často k vytváření málo početných shluků.

Nejprve popíšeme způsob vytváření nových shluků v případě sériového provedení shlukové analýzy. Pokud podobnost příchozího objektu s žádným ze shluků nepřesáhne stanovenou minimální konstantu, není nově vytvořený shluk okamžitě použit pro výpočet podobnosti přicházejících objektů. Tento shluk je nejprve označen jako odlehlý a je uchováván stranou. Pro další nezařazený objekt se vypočítá jeho podobnost s odlehlým shlukem. Pokud tato podobnost přesáhne stanovenou minimální konstantu, je do daného odlehlého shluku přiřazen. Pokud ne, je opět vytvořen odlehlý shluk, který se uchovává stranou. Počet takto uchovávaných odlehlých shluků je omezen na mnohem menší množství, než je maximální počet shluků pro třídění e-mailů. Dojde-li k jeho překročení, je vymazán ten, který nebyl nejdelší dobu aktualizován. Pokud některý z nich stihne před smazáním dosáhnout požadovaný počet

přiřazení, není již tento shluk považován za odlehlý a bude použit pro přiřazování e-mailů do shluků.

Při paralelním provedení shlukové analýzy je zvolen počet instancí, které budou zpracovávat takto nezařazené objekty. Je zvoleno prvních x z abecedně seřazeného seznamu aktivních instancí. Z nich se náhodně jedna vybere a na předmět s jejím názvem se odešle objekt s informací o tom, že se jedná o nezařazený objekt. Ta se pokusí o přiřazení do odlehlých shluků. V případně, že některý dosáhne požadovaného počtu přiřazení, je jeho sumarizace poslána na instanci, která bude udržovat jeho sumarizaci. Ta poté okamžitě uloží jeho sumarizaci určenou pro výpočet podobnosti do databáze, čímž dojde k její rozeslání na všechny instance.

4.4.5 Mazání starých shluků

Jakmile je na instanci dosaženo maximálního počtu shluků, dojde na ní ke smazaní sumarizace nejméně aktuálního shluku. Za nejméně aktuální shluk je možné brát např. ten, u kterého je doba od poslední aktualizace jeho sumarizace největší.

E-mailové kampaně často chodí ve vlnách. Mnohdy bychom chtěli, aby vytvoření shluku o malé velikosti nezpůsobilo smazání shluku velmi početného, i když je u něho největší doba od poslední aktualizace. Pokud bychom měli dva shluky, jeden o velikosti 100 a druhý o velikosti 10 000, tak bychom chtěli, aby doba od poslední aktualizace u většího shluku musela být několikrát větší než u toho menšího, aby došlo k jeho smazání. Pro tento vztah by se dala použít například tato rovnice:

$$d_i = \Delta t_i \cdot w_i, \tag{4.5}$$

kde Δt_i je doba od poslední aktualizace. Číslo w_i je váha, která se snižuje s počtem objektů přidaných do daného shluku. Je smazán shluk s největší hodnotou d_i. Váhu můžeme vypočítat např. pomocí následující rovnice:

$$w_i = \left(\frac{1}{2}\right)^{\log_b(N_i)}. \tag{4.6}$$

Pokud zvolíme za základ logaritmu číslo $b = 10$, tak s každým desetinásobkem dojde ke snížení váhy o její polovinu. Pokud budeme mít dva shluky, jeden o velikosti 100 a druhý o velikosti 10 000, tak doba aktualizace u většího shluku musí být 4 krát větší než u menšího, aby došlo k jeho smazání

Pokud dojde ke smazání určitého shluku, není potřeba vymazat jeho sumarizaci z databáze. V té je nastavená expirační doba, po které dojde ke smazání záznamu.

4.5 Optimalizace výpočtu podobnosti

S rostoucím počtem shluků roste čas výpočtu podobností. U každé hodnoty se nejdříve zjistí její relativní četnost v každém shluku, abychom byli schopni určit její váhu. Následuje výpočet podobností, ve kterém se opět zjišťuje relativní četnost hodnot v každém shluku. V experimentech dosáhl počet shluků, pokud nebyl omezen, i několika tisíc. V takovém případě došlo k výraznému zpomalení výpočtu.

Experimenty ukázaly, že většina hodnot reprezentující daný objekt, se často vyskytuje pouze v několika shlucích. Proto je podobnost s většinou shluků velmi malá. Je možné využít pomocnou datovou strukturu, s jejíž pomocí nebude docházet k iteraci všech shluků. Hodnoty nacházející se v sumarizaci mohou být použity jako klíče v hešovací tabulce. Ke každé hodnotě v této hešovací tabulce bude asociována datová struktura obsahující odkaz na shluky, ve kterých relativní četnost dané hodnoty dosáhla hodnoty $s - \epsilon$.

Je nutné poznamenat, že tuto pomocnou datovou strukturu není možné sestavit, pokud je pro aproximaci relativních četností použit algoritmus Count-Min Sketch. U něho lze zjistit relativní četnost hodnoty pouze pokud ji máme k dispozici z příchozího objektu. Nedokážeme z něj tedy získat seznam četných hodnot.

Tato datová struktura umožňuje získat váhy pro jednotlivé hodnoty rychleji. Když se hledá váha pro konkrétní hodnotu, tak se z hešovací tabulky získá datová struktura obsahující odkazy na shluky, ve kterých se tato hodnota vyskytuje často. Z ní lze jednoduše získat počet shluků a poté přejít k výpočtu váhy. Výpočet podobnosti poté probíhá po složkách. Dochází k iteraci hodnot reprezentující příchozí objekt a vypočítají se složky podobnosti pouze s těmi shluky, u kterých relativní četnost této hodnoty přesáhla zvolenou konstantu.

Pro získání shluku s největší podobností není často potřeba iterovat všechny hodnoty. Některé z hodnot mohou být přítomny ve většině shluků. U e-mailů se může jednat například o jazyk, ve kterém je daný e-mail napsán. Takové hodnoty mají malé váhy a výpočet jejich složek tvoří největší část výpočtu. Přitom bychom je mohli ve většině případů vynechat. Hodnota může do výsledné podobnosti přispět složkou, jejíž velikost je rovna maximálně velikosti její váhy. Výpočet podobnosti probíhá od hodnot s největší vahou po tu s nejmenší. Lze přerušit po výpočtu složek jedné hodnoty, pokud dojde ke splnění jedné z následujících podmínek:

1. Největší vypočítaná podobnost již přesáhla minimální podobnost a rozdíl od druhé největší je větší, než součet vah zbývajících hodnot.

2. Největší podobnost zatím nedosáhla minimální požadované podobnosti pro přiřazení do shluku a již dosáhnout nemůže, protože na to zbylé hodnoty nebudou stačit.

Největšího zrychlení výpočtu je dosaženo pomocí první podmínky. Hodnoty s nejmenší vahou, jejichž složky se počítají až nakonec, tvoří největší část výpočtu podobnosti. Pokud je přeskočíme, dojde k výraznému zrychlení výpočtu. Cenou za zrychlení výpočtu je větší potřebné množství paměti.

U aktualizace pomocné datové struktury nastávají tyto tři situace:

1. Instance přijala sumarizaci nového shluku. V tomto případě se pro každou hodnotu v získané sumarizaci zkontroluje, jestli se již nachází v hešovací tabulce. Pokud ano, je do datové struktury asociované k této hodnotě přidán odkaz na nový shluk. Pokud ne, vytvoří se datová struktura obsahující odkaz na nový shluk a vloží se do hešovací tabulky pod danou hodnotou.

2. Instance přijala aktualizaci sumarizace shluku, který se již používá pro výpočet podobností. V tomto případě mohou pro hodnoty v tomto shluku nastat tři možnosti. Hodnota již byla v předchozí sumarizaci tohoto shluku. Hodnota předtím nebyla v předchozí sumarizaci, protože nepřesáhla $s - \epsilon$ nebo některá hodnota předtím byla a teď už není v této sumarizaci. Pro zjištění, která z těchto možností pro danou hodnotu nastala, postačí porovnat přijatou aktualizaci sumarizace s tou předchozí, která je v daném momentě v paměti instance.

3. Instance přijala sumarizaci nového shluku a došlo k překročení maximálního počtu shluků. Nejprve se určí shluk, který bude smazán. Poté se v pomocné datové struktuře smaže odkaz na tento shluk u všech hodnot, které se nacházejí v jeho sumarizaci. Následně dojde ke smazání celé sumarizace a postupuje se jako v první situaci.

4.6 Provozní situace

Při reálném provozu navrženého řešení je potřeba ošetřit situace, které mohou během provozu nastat. Jinak by mohlo dojít k nesprávnému fungování nebo v horším případě havarování celé aplikace. V této podkapitole jsou uvedeny ty nejpodstatnější.

4.6.1 Inicializace databáze

Nejdříve musí aplikace ověřit, zda v dané databázi existují segmenty pro získávání názvů ostatních instancí a pro ukládání sumarizací. Aplikace nejdřív ověří, zda tyto segmenty existují. Pokud ne, aplikace tyto segmenty vytvoří dle nastavené konfigurace. Mezi hlavní konfigurace pro tyto segmenty patří expirační doba záznamů, maximální velikost uložených hodnot a kolik replikací daného segmentu se má vytvořit. Je nutné počítat s tím, že po vytvoření segmentu se jeho konfigurace nedá změnit. Z tohoto důvodu se při změně této konfigurace musí vytvořit nový segment s jiným názvem. Pokud při spuštění aplikace tyto segmenty již existují, je dobré, aby aplikace nejdříve zkontrolovala, zda konfigurace segmentu odpovídá požadovanému nastavení. V případě neshody zaloguje chybovou hlášku.

4.6.2 První spuštění a aktualizace na novou verzi

Při prvním spuštění nebo aktualizaci aplikace na novou verzi je potřeba se shlukovou analýzou chvíli počkat. V Kubernetes se pouští nové instance, které postupně nahrazují ty staré tak, aby byla služba aplikace stále dostupná. Je proto nutné po spuštění instance se shlukovou analýzou a synchronizací sumarizací chvíli počkat než dojde k nahrazení starých instancí. Tuto dobu je ideální nastavit také tak, aby si instance před spuštěním shlukové analýzy stihly vytvořit aktuální seznam aktivních instancí a z databáze stáhnout sumarizace pro výpočet podobnosti, pokud byly předtím vytvořeny.

V případě, že se v daném segmentu žádné sumarizace nenachází, bude na začátku docházet k většímu počtu odesílání nezařazených objektů na instance, které rozhodují o vytvoření nových shluků. Mohlo by dojít k tomu, že tyto instance nebudou stíhat. To by se dalo řešit tak, že se bude po nějakou dobu shluková analýza provádět pouze na části zpracovávaných e-mailů a tento počet se bude postupně zvyšovat než

dojde k vytvoření dostatečného počtu shluků. Toto zvyšování může být prováděno postupně, nebo může záviset na aktuálním počtu shluků na dané instancí. Tento způsob je možné implementovat například tak, že modul provede shlukovou analýzu s určitou pravděpodobností, která se bude postupně zvyšovat.

4.6.3 Změna seznamu instancí

V průběhu běhu aplikace může dojít ke změně seznamu instancí, který se používá pro určení instance, na kterou se posílají data pro udržování sumarizací a vytváření nových shluků. Může tak dojít například při přeškálování počtu instancí nebo v důsledku výpadku jedné instance a jejímu nahrazení novou. V takovém případě je třeba vyřešit dva problémy.

Prvním je ten, že v důsledku změny pořadí budou chodit nezařazené objekty na jiné instance. Ty si budou nyní uchovávat odlehlé shluky a vytvářet nové. Na těch předchozích by se mělo zajistit, aby došlo k smazání odlehlých shluků. To se může zajistit tak, že si instance při změně seznamu zkontrolují, jestli budou přijímat zprávy s nezařazenými objekty. Pokud ne, tak můžou smazat všechny odlehlé shluky, pokud nějaké uchovávají v paměti.

Druhý problém vzniká u udržování sumarizací jednotlivých shluků. Pokud je detekována změna seznamu instancí, může daná instance postupně rozeslat udržované sumarizace na ty instance, které dle seznamu vychází pro jejich udržování. Toto rozesílání se uskuteční až po uplynutí předem stanovené doby. Pokud totiž dojde ke škálování, je třeba chvíli počkat, než se ustálí seznam všech instancí. Do té doby se také pozastaví posílání dat pro inkrementaci sumarizace. Po odeslání udržované sumarizace dojde na dané instanci k jejímu smazání.

4.6.4 Ztráta udržované sumarizace

Může se také stát, že instance přijme zprávu obsahující hodnoty pro inkrementaci sumarizace shluku, kterou aktuálně neudržuje. K této situaci může dojít třeba při změně seznamu instancí, kdy z nějakého důvodu není na danou instanci sumarizace k udržování doručena. V takovém případě závisí, jaký algoritmus je použit pro udržování sumarizací shluků.

Pokud je použit algoritmus Count-Min Sketch s posuvnými, překrývajícími se okny, tak instance může pokračovat v inkrementaci synchronizované sumarizace. Tu vymaže až tehdy, když druhé nejstarší okno dosáhne požadované velikosti.

V případě použití algoritmu Lossy Counting s posuvnými, překrývajícími se okny, nemáme informaci o tom, kdy může dojít ke smazání existujících záznamů. V takovém případě můžeme pokračovat v inkrementaci jako v předešlém případě s tím, že záznamy přejaté ze synchronizované sumarizace nebudou smazány, dokud nedojde ke smazání daného okna a použití toho následujícího.

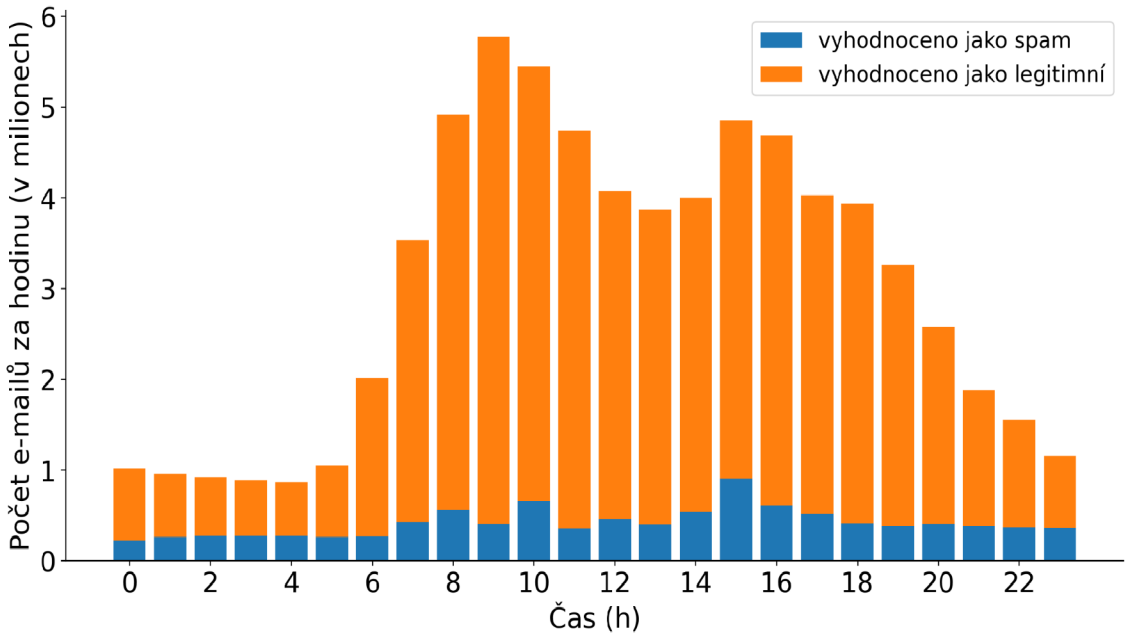
U algoritmu Sliding Lossy Counting můžeme na začátku vytvořit potřebný počet expiračních záznamů tak, aby po přiřazení dalších N objektů do daného shluku došlo k jejich promazání.

5 Experiment

V této kapitole je testováno navržené řešení na datech z reálného provozu. Nejprve je představen použitý datový soubor, na kterém jsou následně otestovány vybrané algoritmy pro určení aproximovaných relativních četnosti. Následuje podkapitola věnující se evaluaci shlukování. Poté následují experimenty, ve kterých bude zhodnocen algoritmus StreamCluCD a navržené řešení.

5.1 Datový soubor

Experiment je proveden na datovém souboru vytvořeném z e-mailů vyhodnocených antispamem za jeden den. Z každého e-mailu se získaly hodnoty vybraných atributů. Každý řádek datového souboru reprezentuje jeden e-mail. Řádek obsahuje čas, ve kterém byl antispamem vyhodnocen. Dále obsahuje informaci, zda byl daný e-mail vyhodnocen jako legitimní nebo jako spam. Nakonec obsahuje hodnoty vybraných atributů. Řádky jsou chronologicky seřazeny. Datový soubor obsahuje celkem okolo 72 milionů e-mailů, z nichž je 10 milionů vyhodnoceno jako spam. Na obrázku č. 14 je možné vidět rozložení e-mailů v průběhu dne podle hodin.



Obrázek 14: Rozložení e-mailů nacházejících se v datovém souboru během dne podle jednotlivých hodin

Pro vybrané atributy je průměrně v příchozím e-mailu přítomno okolo 50 unikátních hodnot. Průměrný počet hodnot a celkový počet unikátních hodnot, pro jednotlivé atributy, je zobrazen v tabulce č. 3.

Název	Unikátních hodnot	Průměrně hodnot
theme	16	0,9
lang	92	1
country	203	1
ip_crange	89 690	1
efrom.domain	351 496	1
ip	730 813	1
hfrom.domain	741 369	1
sender_name	1 656 294	0,9
hfrom.local	2 659 525	1
urls	4 139 288	2,8
subject	4 684 524	7
efrom.local	27 966 631	1
words	97 233 291	30,4
celkem	140 283 472	49,9

Tabulka 3: Unikátní počet hodnot vybraných atributů a jejich průměrný počet v jednom e-mailu

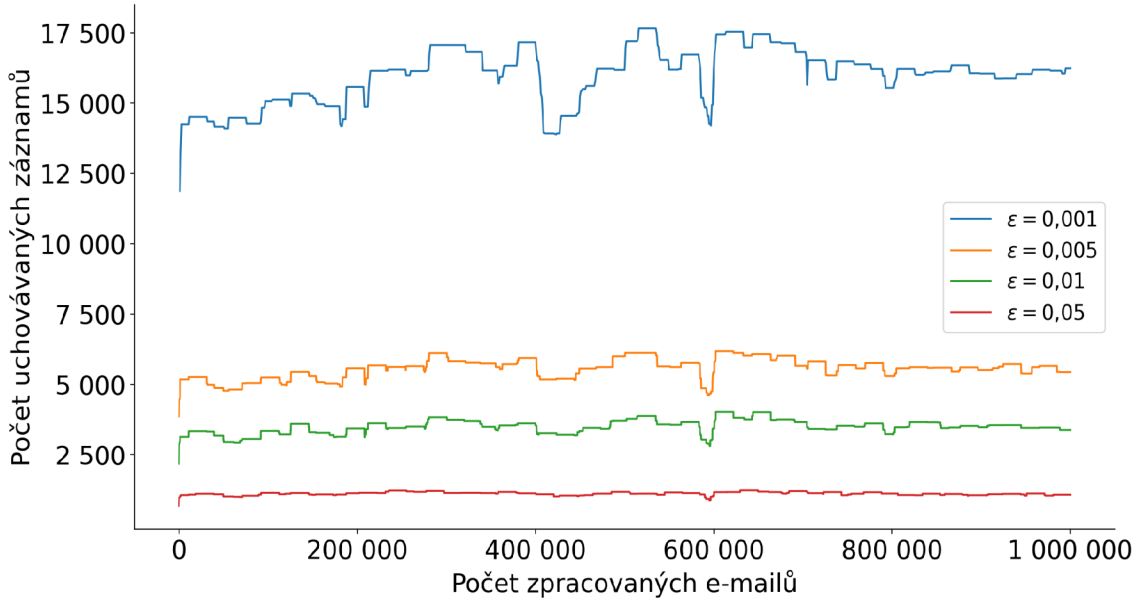
5.2 Testování aproximace četnosti

Na datovém souboru byly nejdříve otestovány algoritmy Lossy Counting, Count-Min Sketch a Sliding Lossy Counting. K tomuto účelu bylo použito prvních milion e-mailů z datového souboru. Algoritmy byly použity na všech hodnotách vybraných atributů.

Jako první byl otestován algoritmus Lossy Counting. Cílem bylo zjistit, kolik je uchováváno záznamů pro různě zvolenou maximální relativní chybu ϵ. Počet záznamů byl zaznamenáván před promazáním na konci každého okna o velikosti $w = \lceil \frac{1}{\epsilon} \rceil$. Pro zvolené $\epsilon = 0{,}01$, se počet uchovávaných záznamů pohyboval zhruba v rozmezí 1 000–3 500. Počet uchovávaných záznamů pro jednotlivá ϵ v průběhu experimentu je zobrazen na obrázku č. 15. Hodnoty uchovávaných záznamů hodně kolísaly, proto se pro zobrazení použilo klouzavých maxim.

Při navrhování počtu počítadel v algoritmu Count-Min Sketch je nutné počítat s tím, že se v tomto případě nejedná o posloupnost hodnot, ale posloupnost objektů složených z množiny vybraných hodnot. Počet počítadel w v případě posloupnosti hodnot určujeme podle zvolené relativní chyby ϵ rovnicí (2.7). Zvolená relativní chyba je pak překročena s pravděpodobností rovnou maximálně číslu δ.

Po N inkrementacích je absolutní chyba rovna maximálně $\epsilon \cdot N$ s pravděpodobností $1-\delta$. V tomto experimentu je N rovno počtu objektů. Nechť číslo m udává průměrný počet hodnot v jednom objektu. Dostaneme celkový počet inkrementů roven číslu $m \cdot N$. Absolutní chyba je tedy rovna maximálně číslu $\epsilon \cdot m \cdot N$ s pravděpodobností $1 - \delta$. Číslo ϵ v tomto případě udává relativní chybu vzhledem k celkovému počtu inkrementů $m \cdot N$. Relativní chyba vzhledem k počtu objektů je rovna číslu $\hat{\epsilon} = \epsilon \cdot m$. Proto pro návrh použijeme přepočítané ϵ dle následující rovnice:



Obrázek 15: Počet uchovávaných záznamů algoritmem Lossy Counting pro různě zvolené maximální relativní chyby

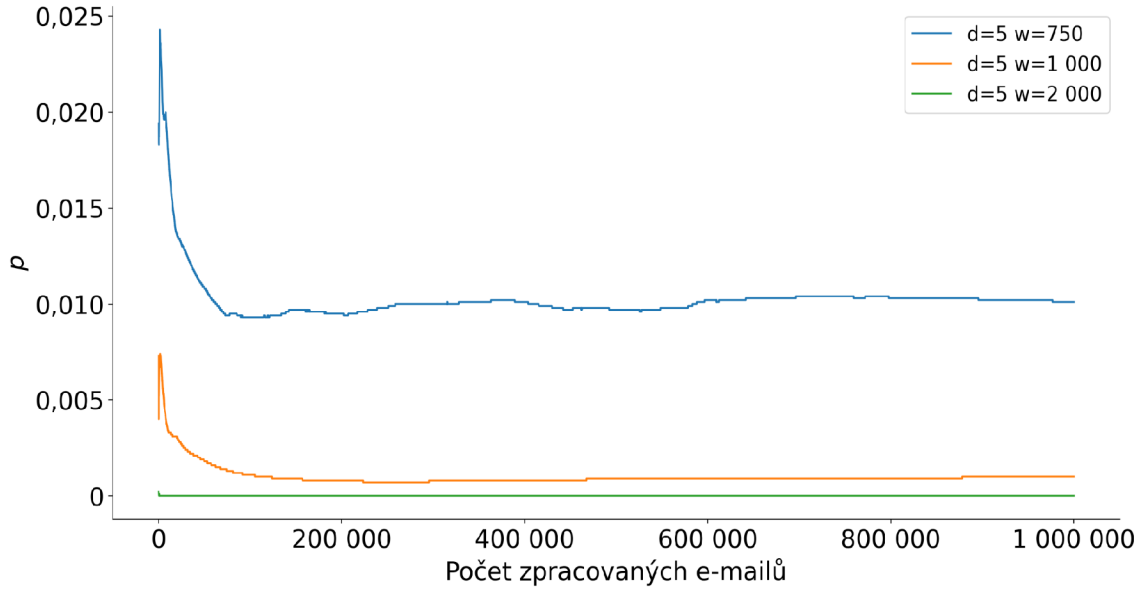
$$\epsilon = \frac{\hat{\epsilon}}{m}, \tag{5.1}$$

kde $\hat{\epsilon}$ je požadovaná relativní chyba vzhledem k počtu objektů. Pro požadované $\hat{\epsilon} = 0{,}05$ a $m = 50$ dostaneme $w = 2\ 719$. Pro zvolené $\delta = 0{,}01$ dostaneme počet hešovacích funkcí $h = 5$. V provedeném experimentu byla měřena pravděpodobnost, s jakou byla zvolená relativní chyba překročena. Ta byla překročena s mnohem menší pravděpodobností, proto byl experiment proveden také s nižšími hodnotami parametru w. Naměřené pravděpodobnosti překročení zvolené relativní chyby lze vidět v tabulce č. 4. Je v ní zobrazena také průměrná relativní chyba.

w	p	Průměrná relativní chyba
2 882	0	0,0041
2 000	0	0,0067
1 000	0,001	0,0172
750	0,0101	0,0255
500	0,2373	0,0434

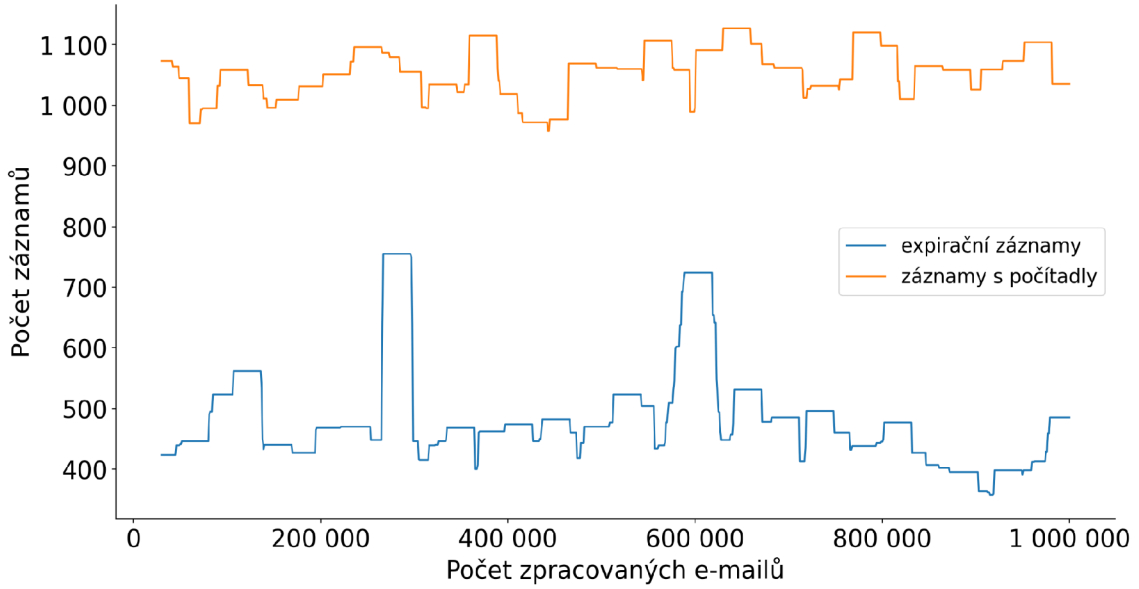
Tabulka 4: Pravděpodobnost p překročení zvolené relativní chyby ϵ =0,05 pro zvolené hodnoty parametru w

Pravděpodobnosti přibližně 0,01 bylo dosaženo s $w = 750$. Jedná se o pravděpodobnost za celý experiment. Na začátku experimentu dosáhla hodnoty 0,025. Je tedy vhodnější zvolit $w = 1\ 000$. Pro tuto hodnotu parametru pravděpodobnost nepřekročila hodnotu 0,01. Vývoj pravděpodobnosti překročení zvolené chyby v průběhu experimentu je zobrazen obrázku č. 16.



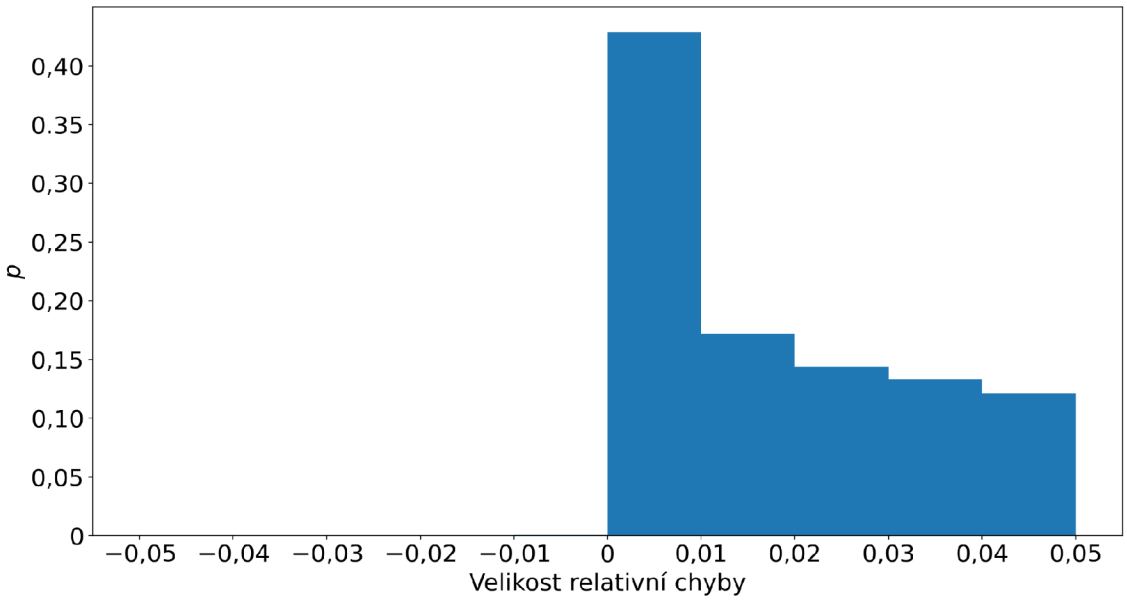
Obrázek 16: Vývoj pravděpodobnosti p překročení zvolené relativní chyby $\epsilon = 0{,}05$ v průběhu experimentu

Dále byl experiment proveden pro algoritmus Sliding Lossy Counting se zvolenými parametry $\epsilon = 0{,}05$ a $N = 1\,000$ ($m = 50$). Počet uchovávaných záznamů s počítadly kolísal v rozmezí 200–1 100. Počet expiračních záznamů kolísal většinou mezi 200 až 600. Počet uchovávaných záznamů a expiračních záznamů je zobrazen na obrázku č. 17. Pro zobrazení se kvůli kolísání použila klouzavá maxima.



Obrázek 17: Počet uchovávaných záznamů a expiračních záznamů algoritmem Sliding Lossy Counting pro zvolené $\epsilon = 0{,}05$ a $N = 1\,000$

Relativní chyba během experimentu nepřekročila hodnotu 0,05. Je zajímavé, že pro dané rozdělení dat došlo k nadhodnocení pouze v 0,1 % případů. Rozdělení velikosti relativních chyb je možné vidět na obrázku č. 18.



Obrázek 18: Naměřená pravděpodobnost p velikostí relativních chyb. Kladné hodnoty znamenají podhodnocení relativní četnosti a záporné nadhodnocení relativní četnosti

5.3 Evaluace kvality shlukování

V pracích [10, 2] vyzkoušeli shlukování na datových souborech, jejichž objekty měly přiřazenou správnou kategorii. V použitém datovém souboru je k dispozici údaj, zda byl e-mail vyhodnocen antispamem jako legitimní nebo jako spam. Vyskytují se však případy, kdy spam není zachycen nebo je naopak legitimní pošta označena jako spam. Ruční roztřídění takto rozsáhlého datového souboru by nebylo možné provést za rozumnou dobu. Konkrétní e-mail může být pro jednoho uživatele spam a pro druhého legitimní e-mail. Ukazatel, zda algoritmus roztřiďuje e-maily do shluků tak, že e-maily z jednoho shluku jsou převážně vyhodnoceny jako legitimní nebo jako spam, může být velmi užitečný. Podle tohoto ukazatele můžeme rozhodnout o tom, zda ve vývoji navrženého řešení dále pokračovat a testovat jeho přínos v reálném provozu.

Algoritmus StreamCluCD testovali v práci [10] na datovém souboru obsahující údaje o různých houbách. U nich byl přítomen údaj, jestli jsou jedlé nebo jedovaté. Autoři se pokusili vyhodnotit, zda algoritmus roztřiďuje do shluků tak, že jsou jednotlivé shluky tvořené převážně jedlými nebo nejedlými houbami. K měření přesnosti použili následující ukazatel přesnosti:

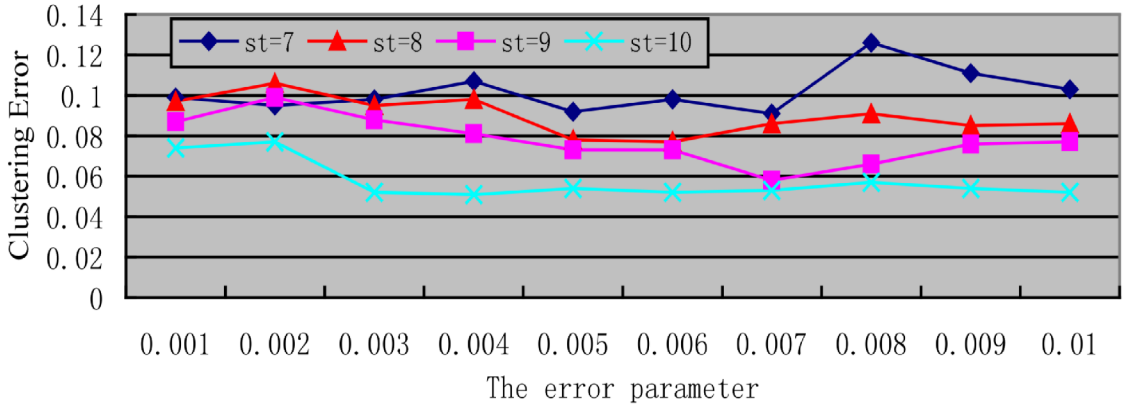
$$r = \frac{\sum_{i=1}^{k} a_i}{n}, \tag{5.2}$$

kde n je počet objektů v datovém souboru. Číslo k je celkový počet shluků. Číslo

a_i udává počet objektů ve shluku C_i v převažující kategorii v daném shluku. Chybu lze pak definovat následující rovnicí:

$$e = 1 - r. \tag{5.3}$$

Na obrázku č. 19 je zobrazena velikost chyby vypočtená předchozí rovnicí, kterou naměřili v dané práci. Obrázek zobrazuje její závislost na zvolené, maximální možné relativní chybě a minimální podobnosti pro přiřazení objektu do shluku. Chyba se dle zvolené maximální relativní chyby výrazně neměnila. Závisela především na zvolené minimální podobnosti.



Obrázek 19: Chyba shlukování podle zvolené maximální relativní chyby a minimální podobnosti *st* pro přiřazení do shluku. Převzato z [10]

Algoritmus CSketch testovali v práci [2] na datových souborech určených pro detekci narušení (intrusion detection) od společnosti IBM. Záznamy v těchto datových souborech byly vytvořeny senzory nacházející se na různých geografických místech a představovaly výstrahu, která byla vytvořena daným senzorem. Těchto typů výstrah bylo několik stovek. Autoři hodnotili kvalitu shlukování podobně jako autoři algoritmu StreamCluCD. Testovali, zda třídí do shluků tak, že jsou jednotlivé shluky tvořené převážně jedním typem výstrahy.

Nechť číslo s označuje celkový počet kategorií. Nechť $p_1, p_2, \ldots, p_s$ označují relativní četnost výskytu těchto kategorií ve shluku C_i. Jako ukazatel kvality separace ve shluku C_i použili jeho entropii vypočtenou dle následující rovnice:

$$E(C_i) = 1 - \sum_{j=1}^{s} p_j^2. \tag{5.4}$$

Hodnota entropie $E(C_i)$ je blízká nule, pokud je shluk tvořený převážně jednou kategorií. Na druhou stranu, čím více je rovnoměrně tvořen více kategoriemi, tím více je tato hodnota bližší jedné. Ukazatel kvality separace shlukování je definován jako vážený průměr entropií všech shluků dle následující rovnice:

$$E = \frac{\sum_{i=1}^{k} |C_i| \cdot E(C_i)}{\sum_{i=1}^{k} |C_i|}, \tag{5.5}$$

kde k je celkový počet shluků.

5.4 Testování shlukování

Skripty pro testování shlukování jsou napsané v jazyce Go. Všechny výpočty byly prováděné na počítači s 40 x Intel Xeon CPU E5-2630L v4 @1,80 GH a 360 GB RAM pamětí.

Jako první bylo provedeno shlukování algoritmem Squeezer. Ten používá pro výpočet podobnosti objektu se shlukem přesné relativní četnosti. Po přiřazení e-mailu do shluku bylo zaznamenáno ID shluku, do kterého byl přiřazen. Jednou za 1 000 roztříděných e-mailů byl zaznamenán počet uchovávaných záznamů, počet shluků a množství alokované paměti. Měřil se také čas, za který je objekt přiřazen do shluku. Zaznamenávána byla jeho průměrná hodnota za posledních 1 000 roztříděných e-mailů. Tento experiment byl proveden na prvním 1 milionu e-mailů z datového souboru. Shlukování bylo testováno pro různé hodnoty minimální podobnosti.

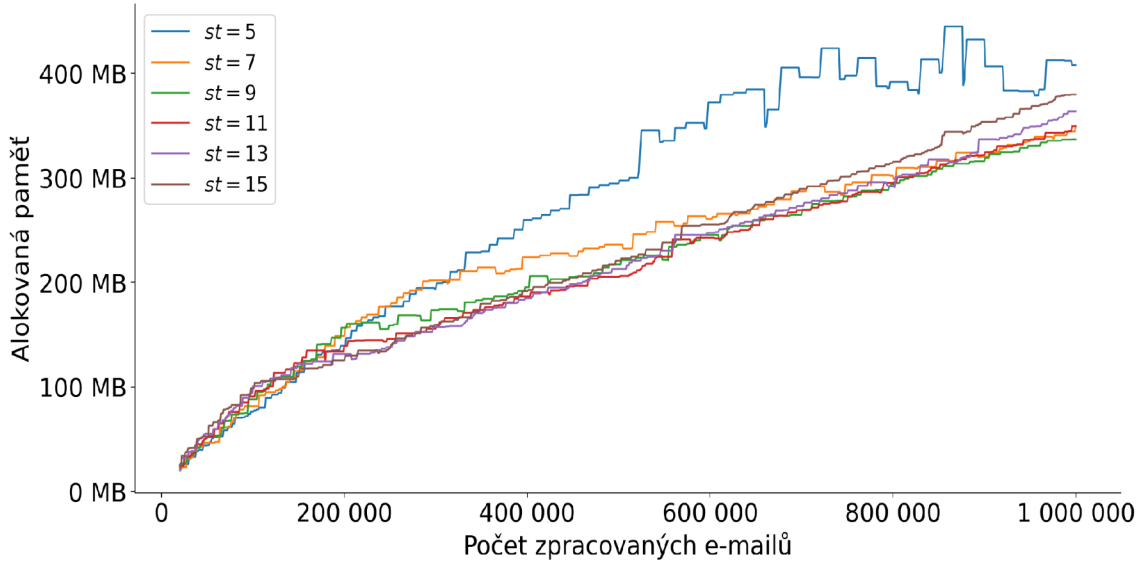
S rostoucí minimální podobností rostl výsledný počet shluků. Důsledkem toho rostl čas na přiřazení objektu do shluku a celkový počet uchovávaných záznamů. Pro minimální podobnost rovnou číslu 5 byl počet shluků roven 1 801. Průměrný čas, za který je objekt přiřazen do shluku, byl na konci experimentu kolem 14 milisekund. Celkový počet uchovávaných záznamů činil 3,485 milionu. S minimální podobností rovnou 15 vzrostl počet shluků na 43 301 a s tím průměrný čas pro přiřazení na 180 milisekund. Počet záznamů vzrostl na 4,382 milionu. Tabulka č. 5 zobrazuje naměřené hodnoty v tomto experimentu.

st	k	n (v milionech)
5	1 801	3,485
6	3 355	3,559
7	5 140	3,630
8	7 129	3,690
9	9 506	3,766
10	13 302	3,840
11	19 244	3,942
12	25 485	4,049
13	30 835	4,153
14	36 833	4,261
15	43 301	4,383

Tabulka 5: Naměřené hodnoty pro algoritmus squeezer. Číslo st představuje zvolenou minimální podobnost. Číslo k je výsledný počet shluků a číslo n je počet uchovávaných záznamů na konci experimentu

Množství alokované paměti v průběhu experimentu kolísalo, protože jazyk Go používá automatickou správu paměti. Zajímavým zjištěním bylo, že s minimální podobností rovnou číslu 5, bylo alokováno více paměti než pro vyšší hodnoty minimální podobnosti. Možnou příčinnou by mohl být růst hešovacích tabulek, použitých pro počítání četnosti hodnot. Hešovací tabulky po dosažení použitelného počtu klíčů zdvojnásobí svoji velikost. Díky menšímu počtu shluků pravděpodobně vznikaly velmi početné hešovací tabulky s velkým množstvím alokované paměti. Pro zobrazení byla použita klouzavá maxima. Velikost těchto maxim se pohybovala v rozmezí

300–400 MB na konci experimentu. Vývoj množství alokované paměti je zobrazen na obrázku č. 20.



Obrázek 20: Množství alokované paměti algoritmem squeezer v průběhu experimentu pro různě zvolené minimální podobnosti

Pro minimální podobnost rovnou 5 nepřesáhl počet objektů v polovině shlucích čísla 14. U vyšších hodnot minimální podobnosti se jednalo dokonce o více než 90 % shluků. Kvantily velikostí pro zvolené minimální podobnosti je možné vidět v tabulce č. 6.

st \ q	0,25	0,50	0,75	0,90	0,99	0,999
5	3	14	106	614	7 787	45 120
6	2	6	39	250	3 974	33 998
7	2	5	24	134	2 618	27 856
8	1	4	17	88	1 883	19 056
9	1	4	13	62	1 457	15 935
10	1	3	9	37	894	11 176
11	1	2	6	23	556	6 890
12	1	1	4	16	376	5 550
13	1	1	4	12	300	4 723
14	1	1	3	10	236	3 739
15	1	1	2	8	202	3 091

Tabulka 6: Kvantily q počtu objektů ve shlucích pro zvolené minimální podobnosti

Pro evaluaci kvality shlukování byl použit ukazatel přesnosti r vypočítaný dle rovnice (5.2). Ten se pohyboval v rozmezí 0,95 až 0,98. S rostoucí minimální podobností se zvětšovala jeho hodnota. Hodnoty pro jednotlivé minimální podobnosti jsou zobrazené v tabulce č. 7.

st	5	6	7	8	9	10	11	12	13	14	15
r	0,949	0,944	0,949	0,951	0,96	0,968	0,972	0,979	0,979	0,981	0,981

Tabulka 7: Hodnoty ukazatele r naměřené pro algoritmus Squeezer pro zvolené minimální podobnosti

5.4.1 StreamCluCD

V dalším experimentu bylo provedeno shlukování s algoritmem StreamClucD a zaznamenávaly se stejné hodnoty jako v předchozím experimentu. Konstanta s se zvolila 0,2. Stejně jako v práci [10] se ukázalo, že zvolená maximální relativní chyba ϵ má vliv na výslednou hodnotu ukazatele. Nedošlo však k jeho výrazné změně pro různá ϵ. Výsledné hodnoty jsou zobrazeny v tabulce č. 8.

st ϵ	0	0,001	0,005	0,01	0,05
5	0,946	0,948	0,945	0,947	0,952
10	0,966	0,970	0,970	0,970	0,972
15	0,985	0,985	0,985	0,985	0,985

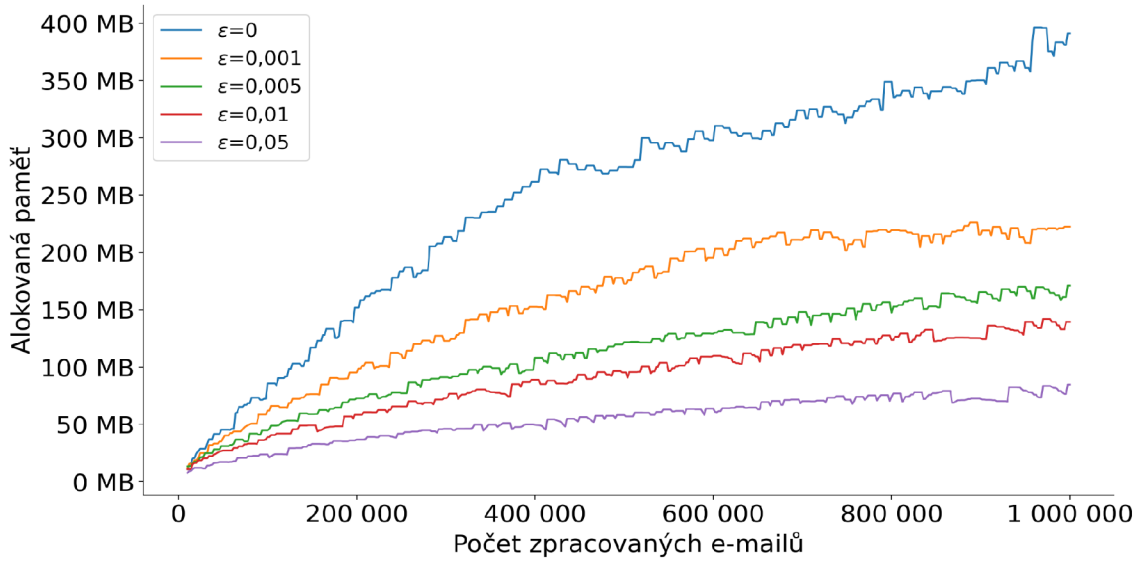
Tabulka 8: Velikost ukazatele kvality shlukování r pro různě zvolené minimální podobnosti st a maximální možnou relativní chybu ϵ

Zatímco nedošlo k výrazné změně ukazatele přesnosti r, zvolená maximální relativní chyba měla významný vliv na celkový počet uchovávaný záznamů a množství alokované paměti. S vyšší zvolenou podobností však docházelo k čím dál menší úspoře, protože se zvyšovalo množství shluků. Počet uchovávaných záznamů na konci experimentu je zobrazen v tabulce č. 9

st ϵ	0	0,001	0,005	0,01	0,05
5	3 613 140	1 700 989	1 012 225	808 066	432 748
10	3 941 557	2 458 152	1 958 785	1 748 735	1 290 495
15	4 426 714	3 051 098	2 621 437	2 444 074	2 097 608

Tabulka 9: Počet uchovávaných záznamů podle zvolené minimální podobnosti st a maximální relativní chyby ϵ

Pokud se počítaly přesné relativní četnosti, alokovaná paměť byla na konci experimentu 400 MB pro minimální podobnost rovnou 5. Pří zvolení $\epsilon = 0{,}001$ klesla velikost alokované paměti skoro o polovinu. Pro $\epsilon = 0{,}05$ se na konci experimentu pohybovala okolo 60 MB. Množství alokované paměti v průběhu experimentu je pro minimální podobnost rovnou 5 zobrazeno na obrázku č. 21. Pro zobrazení jsou opět použita klouzavá maxima.



Obrázek 21: Velikost alokované paměti pro $st = 5$ v průběhu experimentu podle zvolené maximální relativní chyby ϵ

5.4.2 Navržené řešení – sériové shlukování

Navržené řešení bylo nejdříve testováno v sériovém provedení, aby šlo posoudit vliv některých parametrů u paralelního shlukování. Shlukování bylo prováděno na prvním 1 milionů e-mailů z datového souboru. Po přiřazení e-mailu do shluku bylo zaznamenáno ID shluku, do kterého byl přiřazen. Jednou za 1 000 roztříděných e-mailů byl zaznamenán počet uchovávaných záznamů, počet shluků a množství alokované paměti. Měřil se také čas, za který je objekt přiřazen do shluku. Zaznamenávána byla jeho průměrná hodnota za posledních 1 000 roztříděných e-mailů. Pro všechny experimenty bylo zvoleno $s = 0{,}2$ a $\epsilon = 0{,}05$.

První experiment proběhl bez omezení počtu uchovávaných shluků s uchováváním sumarizací pomocí algoritmu Sliding Lossy Counting s $N = 1\ 000$. Nové shluky vznikaly okamžitě, aby vznikalo větší množství shluků a bylo možné otestovat, jak rychle probíhá shlukování při vyšším počtu shluků. Pro stejné hodnoty minimální podobnosti se, oproti algoritmu StreamCluCD, počet shluků výrazně zvýšil. Toto zvýšení je pravděpodobně způsobeno důsledkem použití vah, díky kterým dojde ke snížení vypočtené podobnosti objektu se shlukem. Pro minimální podobnost rovnou 5 došlo k nárůstu počtu shluků z 1 801 na 11 091. Také došlo ke zvýšení hodnoty ukazatele přesnosti r, který pro danou minimální podobnost zvýšil z 0,952 na 0,98. Výsledný počet shluků a ukazatel r pro zvolené hodnoty minimální podobnosti je zobrazen v tabulce č. 10.

st	3	4	5	6	7	8	9	10
k	11 091	21 170	28 978	38 884	52 296	59 036	64 874	73 902
r	0,967	0,976	0,980	0,982	0,984	0,986	0,987	0,988

Tabulka 10: Počet shluků k a velikost ukazatele r pro zvolené minimální podobnosti u sériového provedení navrženého řešení. K udržování sumarizací byl použit algoritmus Sliding Lossy Counting s $N = 1\ 000$

Při použití optimalizovaného provedení výpočtu s algoritmem Lossy Counting nebo Sliding Lossy Counting došlo k výraznému zrychlení. Průměrná doba na přiřazení objektu do shluku se při dosažení 5 000 uchovávaných shluků držela pod 1 milisekundou. S tímto počtem shluků tak bylo možné roztřídit okolo 1 000 objektů do shluků za sekundu. Při použití neoptimalizovaného výpočtu dosahovala průměrná doba na přiřazení okolo 20–30 milisekund. Průměrná doba na přiřazení objektu do shluku, dle počtu shluků a použité varianty navrženého řešení, je zobrazena v tabulce č. 11.

k	LC	SLC	CMS	LC OPT	SLC OPT
500	1,9ms	1,5ms	1,2ms	230ns	210ns
1 000	5,3ms	4,3ms	4,5ms	240ns	245ns
2 000	11ms	9,5ms	12ms	360ns	322ns
5 000	27ms	23,4ms	29ms	960ns	860ns
10 000	58ms	49ms	61ms	1,9ms	1,6ms
20 000	108ms	96ms	109ms	2,6ms	2ms

Tabulka 11: Průměrný čas, za který je objekt přiřazen do shluku, podle zvolené varianty navrženého řešení

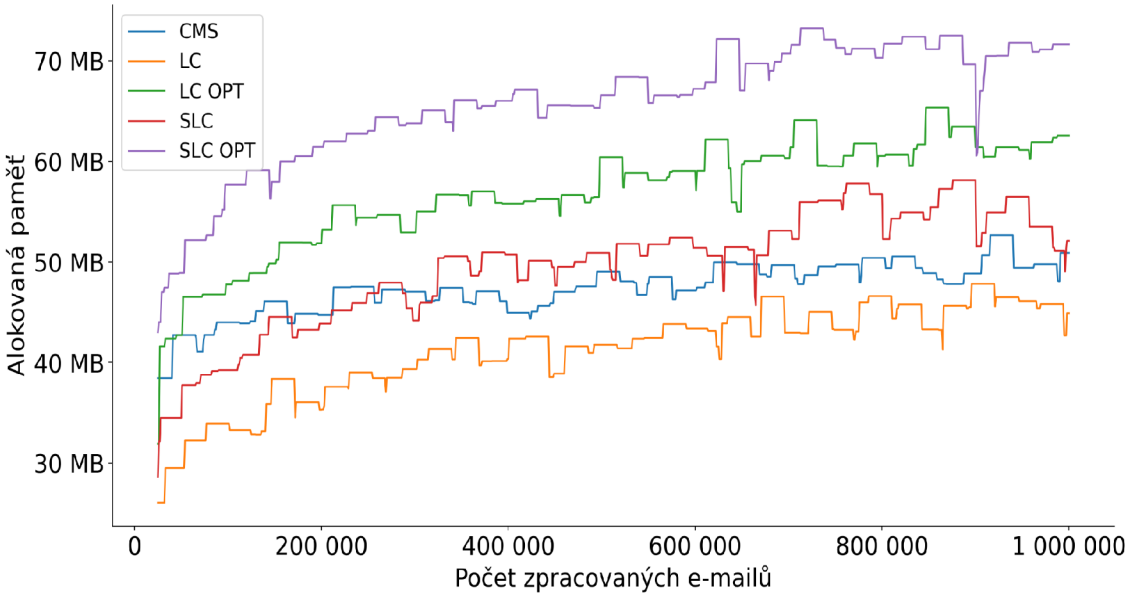
V dalším experimentu se omezil počet uchovávaných shluků a hodnotil se nárok na paměť u jednotlivých variant. Množství alokované paměti opět kolísalo, proto se opět sledovala klouzavá maxima. Maximální počet shluků byl omezen na 2 000. Algoritmy Lossy Counting a Count-min Sketch se použily se 3 překrývajícími se okny o počtu 1 000 objektů. Algoritmus Sliding Lossy Counting s $N = 1\,000$.

Nejmenšího množství alokované paměti se dosáhlo použitím algoritmu Lossy Counting přes překrývající se okna v neoptimalizované verzi. V této variantě bylo množství alokované paměti rovno nejvíce zhruba 45 MB. Varianty s optimalizovaným výpočtem dle očekávání zabíraly větší množství místa. Nicméně, vzhledem k výrazně rychlejšímu výpočtu, se jeví jako vhodnější řešení. Alokovanou paměť u jednotlivých variant je možné vidět na obrázku č. 22.

Varianta s použitím algoritmu Lossy Counting přes překrývající se okna zabírala méně místa, než varianta s algoritmem Sliding Lossy Counting. Ukazatel přesnosti nabýval pro tyto dva algoritmy téměř identické hodnoty. Použití okna o stálé velikosti N tedy nepřineslo výhodu oproti oknu o proměnlivé velikosti $\langle \frac{2}{3}N, N \rangle$, jehož použití má menší nároky na paměť. Ukazatel byl u všech variant velmi blízko hodnotě 0,98. Velikost ukazatele r, pro jednotlivé varianty, je zobrazena v tabulce č. 12.

Varianta	LC	SLC	CMS	LCOPT	SLCOPT
r	0,980	0,977	0,980	0,979	0,980

Tabulka 12: Velikost ukazatele r pro jednotlivé varianty navrženého řešení



Obrázek 22: Množství alokované paměti pro jednotlivé varianty navrženého řešení v sériovém provedení shlukování

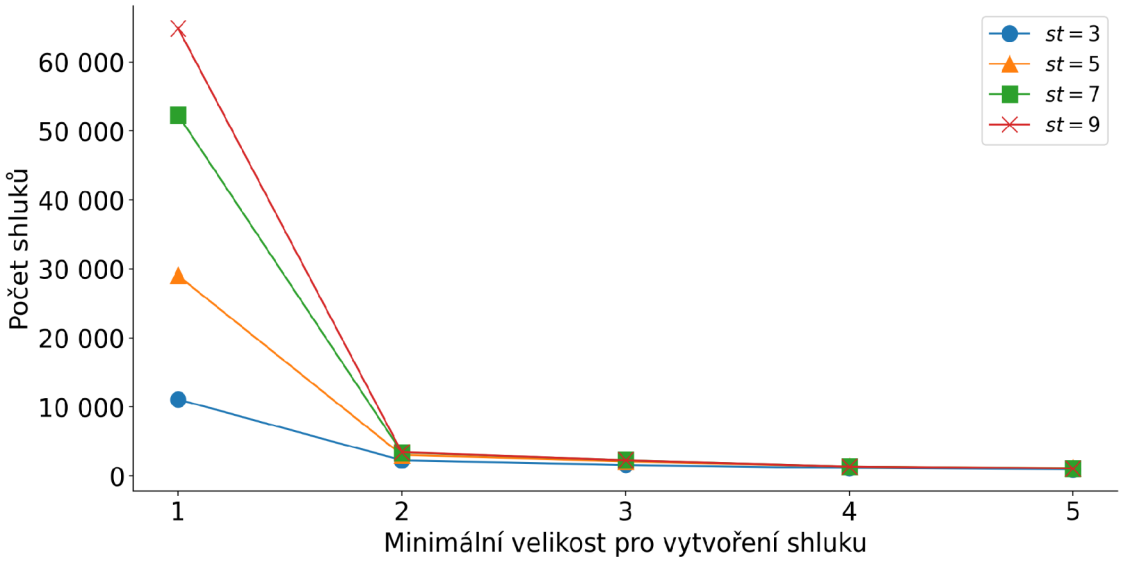
Dále byly provedeny experimenty, kterými se zjišťovalo, jaké je rozložení velikosti uchovávaných shluků v paměti a jejich stáří. Provedlo se shlukování pro různě zvolený základ logaritmu b v rovnici (4.6). Ta se využívá při určování shluku, který má být nahrazen při přesáhnutí zvoleného maximálního počtu uchovávaných shluků. Dále se měnila minimální velikost shluku, kterou musel shluk dosáhnout, aby nebyl označen jako odlehlý. Počet odlehlých shluků byl omezen na 50.

Nejdříve byl experiment proveden bez omezení počtu uchovávaných shluků, aby se zjistilo, jaký budou mít tyto parametry vliv na výsledný počet shluků. Zjistilo se, že základ logaritmu b na výsledný počet shluků neměl vliv. Významný vliv měla minimální velikost shluku, jak je možné vidět na obrázku č. 23. Pro minimální velikost rovnou dvěma došlo ke snížení počtu shluků z 64 870 na 3 437 pro minimální podobnost rovnou 9. Výsledný počet shluků byl pro všechny zvolené hodnoty minimální podobnosti velmi podobný.

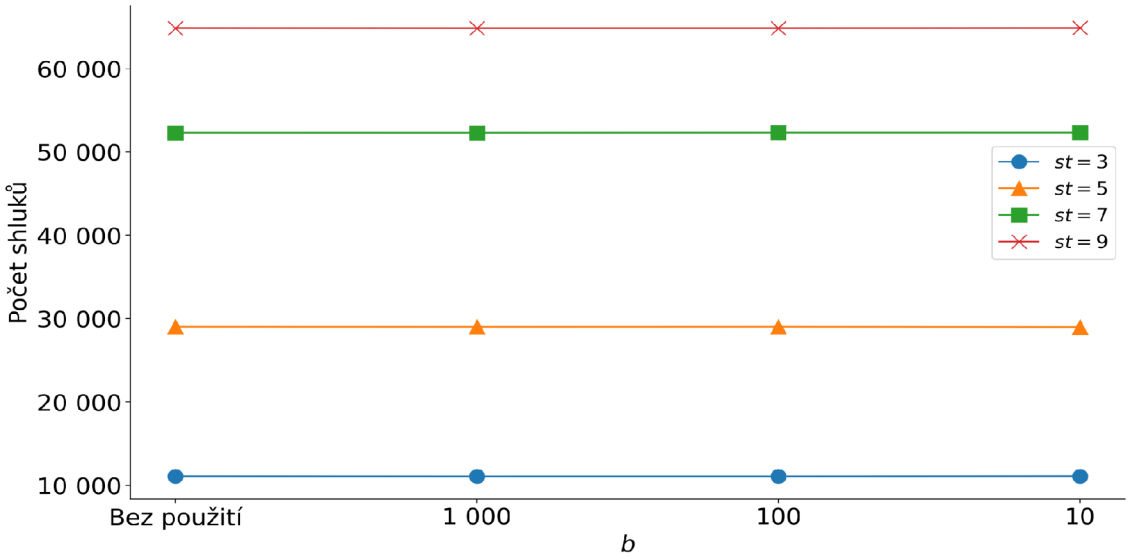
Následoval experiment, kterým bylo zkoumáno rozložení velikosti shluků uchovávaných v paměti a jejich stáří. Počet shluků se pro tento účel omezil na 1 000. Následně se provedlo shlukování pro všechny kombinace minimální velikosti shluku a hodnot parametru b z předchozího experimentu. Po roztřídění 1 milionu e-mailů se zaznamenala velikost všech shluků uchovávaných v paměti a jejich stáří.

Nejprve byly nové shluky vytvářeny okamžitě. Do 75 % uchovávaných shluků nebylo přiřazeno více než 11 e-mailů a 75 % shluků nebylo starší víc, než 89 sekund. Nově vzniklé shluky tak velmi rychle nahrazovaly ty, které bychom chtěli uchovat déle.

Při zvolení parametru $b = 10$ došlo ke zlepšení a horní kvartil velikosti shluků se zvýšil na 201. Nicméně medián velikosti shluků stále nabýval velmi malé hodnoty. Do poloviny shluků uchovávaných v paměti nebylo přiřazeno více, než 4 e-maily. Po-



Obrázek 23: Výsledný počet shluků v závislosti na zvolené minimální velikosti pro vytvoření nového shluku



Obrázek 24: Výsledný počet shluků podle v závislosti na parametru b

lovina shluků také nebyla starší, než 30 sekund. Mnohem výraznější vliv na rozdělení velikosti a stáří měla minimální velikost shluku.

Při nastavení minimální velikosti shluku na 2 došlo k výraznému zlepšení. Polovina shluků dosáhla stáří 28 minut a horní kvartil velikosti shluků se zvýšil z 11 na 258. Dolní kvartily, mediány a horní kvartily pro jednotlivé kombinace zvolených parametrů jsou zobrazeny v tabulkách č 13 a č. 14. Dále v tabulce č. 15 je zobrazeno, kolik procent e-mailů bylo označeno jako odlehlých. Můžeme pozorovat, že pro minimální velikost rovnou 2, bylo okolo 10 % e-mailů označeno jako odlehlých. Při navýšení počtu shluků na 5 000 dojde k poklesu na 8,5 %. Pro všechny zvolené parametry se ukazatel přesnosti pohyboval mezi 0,97 a 0,98. Přesné hodnoty jsou zobrazeny v tabulce č. 16.

		Dolní kvartil				Medián				Horní kvartil			
m	b	—	1 000	100	10	—	1 000	100	10	—	1 000	100	10
1		1	1	1	1	2	3	3	4	11	24	43	201
2		13	15	16	23	62	76	79	98	258	286	307	333
3		19	22	24	30	73	83	87	94	278	298	324	336
4		24	27	28	32	80	89	90	97	304	326	327	342
5		23	25	25	27	74	83	82	82	290	321	324	324

Tabulka 13: Kvartily velikostí shluků uchovávaných v paměti na konci experimentu bez zvýhodňování početnějších shluků (—) a pro kombinaci parametru b a minimální velikosti pro vytvoření shluku m

		Dolní kvartil				Medián				Horní kvartil			
m	b	—	1 000	100	10	—	1 000	100	10	—	1 000	100	10
1		0	0	0	0	1	5	7	30	89	196	306	1 615
2		277	328	361	478	1 490	1 734	1 908	2 219	3 027	3 085	3 102	3 146
3		515	592	598	718	1 879	2 062	2 070	2 265	2 968	2 969	2 975	2 992
4		847	898	858	886	2 064	2 073	2 058	2 102	2 854	2 854	2 856	2 863
5		679	662	659	592	1 807	1 828	1 807	1 810	2 729	2 759	2 739	2 745

Tabulka 14: Kvartily stáří shluků uchovávaných v paměti na konci experimentu bez zvýhodňování početnějších shluků (—) a pro kombinaci parametru b a minimální velikosti pro vytvoření shluku m

m	b	—	1 000	100	10
1		0	0	0	0
2		8,9	8,8	8,8	8,8
3		10,5	10,5	10,6	10,6
4		12,5	12,5	12,6	12,5
5		14,0	13,7	14,0	13,9

Tabulka 15: Počet e-mailů označených jako odlehlé hodnoty v procentech bez zvýhodňování početnějších shluků (—) a pro kombinaci parametru b a minimální velikosti pro vytvoření shluku m

m	b	—	1 000	100	10
1		0,977	0,977	0,975	0,974
2		0,970	0,971	0,970	0,971
3		0,971	0,971	0,971	0,971
4		0,976	0,977	0,977	0,977
5		0,970	0,970	0,977	0,976

Tabulka 16: Velikost ukazatele r bez zvýhodňování početnějších shluků (—) a kombinaci parametru b a minimální velikosti vytvoření shluku m.

5.4.3 Navržené řešení – paralelní shlukování

Cílem testování paralelního shlukování bylo zjistit, jaký má vliv velikost intervalu pro rozesílání aktualizací sumarizací a zvolený počet instancí na shlukování. Zda bude mít způsob vytváření nových shluků a mazání těch nejméně aktuálních podobný vliv na rozložení velikosti uchovávaných shluků v paměti a jejich stáří. Také bylo cílem určit nároky na paměť.

Pro testovaní paralelního shlukování byly jednotlivé instance simulovány použitím gorutin. Pro komunikaci mezi jednotlivými instancemi bylo použito takzvaných kanálů (channels), které lze využít jak k synchronizaci, tak i k realizaci front se zprávami. V hlavním vláknu programu probíhalo čtení jednotlivých řádků datového souboru. Z řádku se získal čas, kdy začal antispam e-mail zpracovávat, zda byl vyhodnocen jako spam a jednotlivé hodnoty reprezentující objekt. Toto vlákno fungovalo jako producent, který tyto údaje zapisoval do kanálu. Ten sloužil jako fronta se zprávami. Jednotlivé instance pak fungovaly jako konzumenti, kteří čekali na údaje zapsané do kanálu.

Hlavní vlákno dále určovalo, kdy má být provedena synchronizace sumarizací mezi jednotlivými instancemi. Čas se určoval z přečtených řádků. Jakmile od poslední synchronizace uběhl specifikovaný interval, pozastavilo se čtení řádků, dokud neproběhla synchronizace. Důvodem pro takovéto provedení bylo to, aby se pro účel experimentu zajistila synchronizace za stanovený interval.

Nejprve se pro komunikaci mezi instancemi v experimentu použil navrhovaný NATS server. V experimentu však v některých okamžicích plynul čas mnohem rychleji. K doručení sumarizace nového shluku, nebo aktualizací stávajících, docházelo se zpožděním až 2 minuty. Proto se pro posílání aktualizací využilo kanálů, se kterými bylo možné dosáhnout kontroly nad doručením aktualizací.

Množství alokované paměti jednou instancí se odhadlo tím, že se celkový počet alokované paměti vydělil počtem gorutin. Nedosáhlo se tím úplně přesných podmínek pro určení náročnosti algoritmu na pamět, protože automatická správa paměti (Garbage Collector) byla provozována pro všechny instance zároveň. Tento údaj byl však považován za dostatečný odhad.

Pro všechny Experimenty se zvolila varianta s optimalizovanou verzí výpočtu a algoritmem Lossy Counting přes posuvná překrývající se okna. Počet posuvných oken byl zvolen 3 a jejich velikost 1 000. Pro všechny experimenty bylo zvoleno $\epsilon = 0{,}05$ a $s = 0{,}2$. Maximální počet odlehlých shluků byl zvolen 50. Testovalo se na prvním 1 milionu e-mailů z datového souboru.

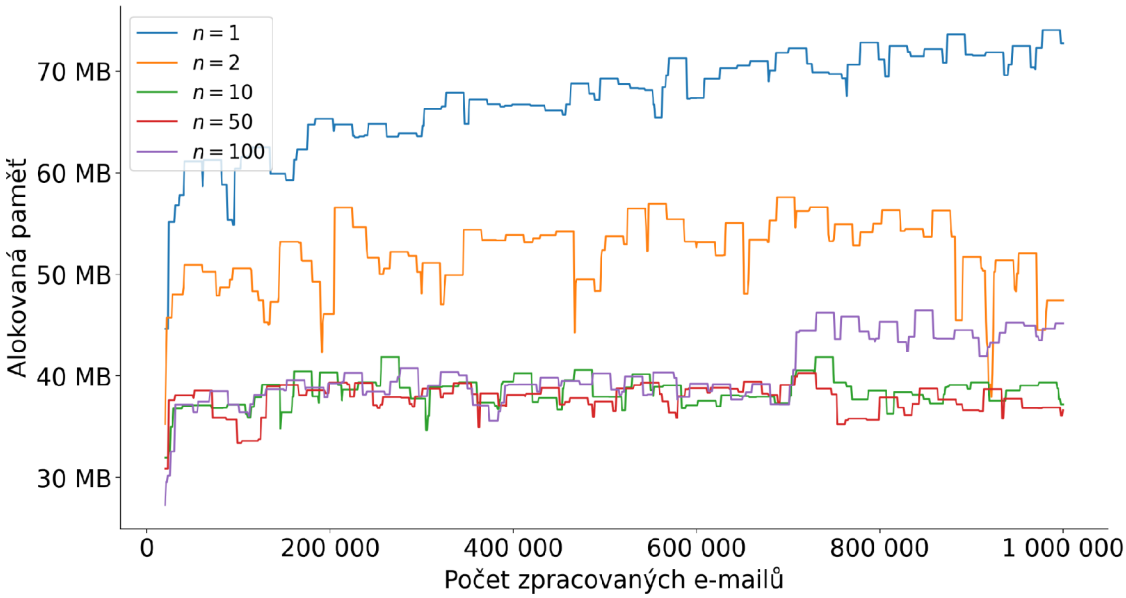
Prvním experimentem se testovalo, jaký má vliv počet instancí a velikost intervalu, za který se budou posílat sumarizace, na ukazatel přesnosti r. Pro tento experiment se omezil maximální počet shluků na 2 000. Odlehlé shluky zpracovávala pouze jedna instance a minimální velikost pro vytvoření nového shluku se zvolila 1. Interval posílání aktualizací byl měněn v rozsahu od jedné sekundy do jedné minuty. Počet instancí byl nastaven na 2, 10, 50 a 100. Hodnota ukazatele r pro kombinace takto nastavených parametrů nevykazovala velkou změnu a pohybovala se v rozmezí 0,976–

0,98. Hodnoty pro jednotlivé kombinace parametrů jsou zobrazeny v tabulce č. 17.

Δt \ n	2	10	50	100
1	0,976	0,977	0,977	0,977
5	0,976	0,977	0,976	0,977
10	0,976	0,977	0,977	0,977
30	0,976	0,977	0,977	0,978
60	0,976	0,977	0,977	0,980

Tabulka 17: Velikost ukazatele r v závislosti na zvoleném počtu instancí n a velikosti intervalu Δt v sekundách, za který proběhla synchronizace sumarizací

S rostoucím počtem instancí se snižoval odhad množství alokované paměti. Uchovávání sumarizací, které zabírají více místa, bylo rozděleno mezi instance. Na obrázku č. 25 jsou zobrazena klouzavá maxima odhadu alokované paměti podle počtu instancí. Při použití jedné instance bylo množství alokované paměti okolo 75 MB. Pro dvě instance kleslo na zhruba 55 MB a pro 10, 50 se pohybovalo okolo 40 MB. Pro 100 instancí se pohybovalo také okolo 40 MB a v závěru experimentu stouplo na 45 MB.

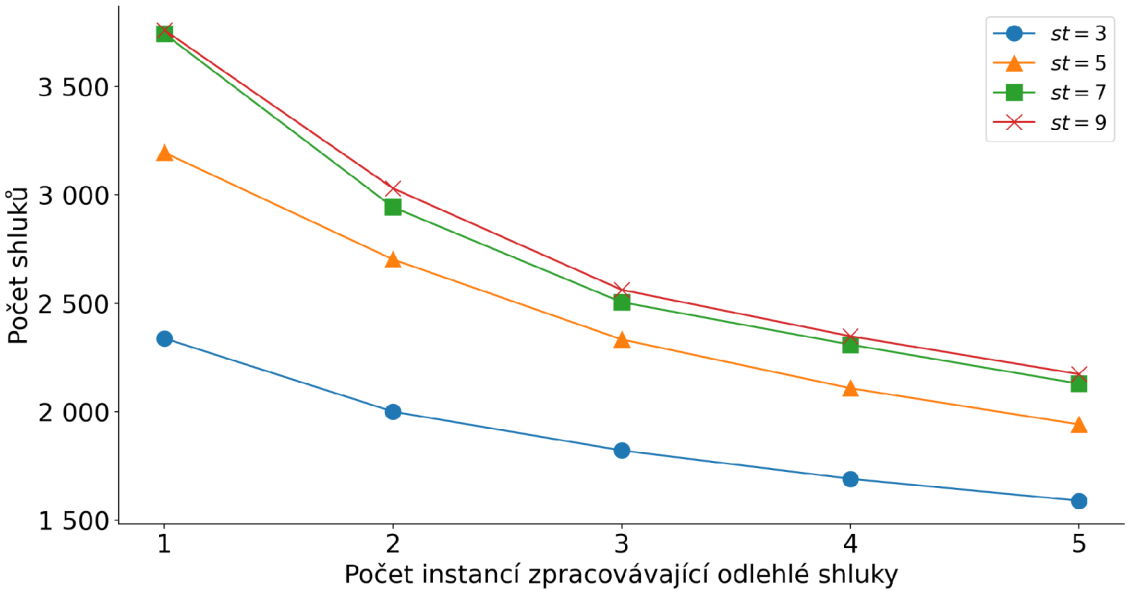


Obrázek 25: Množství alokované paměti podle zvoleného počtu instancí n

Dalším experimentem se zjišťovalo, zda je rozložení velikosti uchovávaných shluků v paměti a jejich stáří podobné jako v sériovém provedení. Minimální velikost pro vytvoření shluku byla volena 1 a 2. Zvyšoval se počet instancí zpracovávajících odlehlé shluky a měnil se základ logaritmu b v rovnici (4.6).

Počet shluků zůstal nejdříve neomezený. Rozdíl mezi minimální velikostí rovnou 1 a 2 byl podobný, jako v sériovém případě. Pro minimální podobnosti rovnou 9 klesl počet shluků ze zhruba 60 000 na 3 750. Se zvyšujícím se počtem instancí zpra-

covávajících odlehlé shluky se snižoval počet shluků. Na obrázku č. 26 je zobrazen počet shluků v závislosti na počtu instancí zpracovávajících odlehlé shluky.



Obrázek 26: Výsledný počet shluků v závislosti na počtu instancí zpracovávajících odlehlé shluky. Minimální velikost pro vznik shluku je rovna 2

Následoval experiment, u kterého se omezil počet uchovávaných shluků na 1 000 a zkoumalo se rozložení velikosti shluků uchovávaných v paměti a jejich stáří. Následně se provedlo shlukování pro všechny kombinace hodnot parametru b a počtu instancí zpracovávajících odlehlé shluky. Po roztřídění 1 milionu e-mailů se zaznamenala velikost všech shluků uchovávaných v paměti a jejich stáří.

Pro všechny zvolené parametry se ukazatel přesnosti opět pohyboval mezi 0,97 a 0,98. Přesné hodnoty jsou zobrazeny v tabulce č. 16. Naměřené výsledky jsou velmi podobné těm při provedení sériového shlukování. Při okamžitém vytváření nových shluků nebylo do více než 75 % všech uchovávaných shluků přiřazeno více než 8 e-mailů a 75 % shluků nebylo starší než 42 sekund. Při zvolení parametru $b = 10$ došlo stejně jako v sériovém provedení k mírnému zlepšení, ale k nejvýraznějšímu zlepšení došlo opět po zvolení minimální velikosti shluků na 2. Naměřené výsledky jsou velmi podobné těm při provedení sériového shlukování a jsou zobrazeny v tabulkách č. 18 a č. 19.

Stejně jako u sériového shlukování bylo e-mailů označených jako odlehlé hodnoty okolo 10 %. Při navýšení počtu instancí však nedochází k tak výraznému růstu počtu odlehlých hodnot, jako při zvyšování minimální velikosti pro vytvoření nového shluku. Počet odlehlých hodnot je zobrazen v tabulce č. 21.

			Dolní kvartil				Medián				Horní kvartil			
n	m	b	—	1 000	100	10	—	1 000	100	10	—	1 000	100	10
1	1		1	1	1	1	1	1	2	3	8	11	18	118
1	2		12	14	15	22	63	73	79	96	270	313	320	345
2	2		16	17	23	29	74	84	90	103	303	320	325	354
3	2		20	24	27	36	77	88	90	110	305	335	331	365
4	2		24	27	27	35	81	90	98	102	308	331	354	355
5	2		22	29	32	36	82	92	94	104	331	346	359	375

Tabulka 18: Kvartily velikostí shluků uchovávaných v paměti na konci experimentu bez zvýhodňování početnějších shluků (—), pro kombinaci parametru b, minimální velikosti pro vytvoření shluku m a počtu instancí n zpracovávajících odlehlé shluky

			Dolní kvartil				Medián				Horní kvartil			
n	m	b	—	1 000	100	10	—	1 000	100	10	—	1 000	100	10
1	1		0	0	0	0	0	0	1	5	42	66	106	779
1	2		250	320	335	447	1 386	1 734	1 792	2 194	3 036	3 096	3 103	3 142
2	2		464	482	527	682	1 903	2 070	2 155	2 300	3 010	3 054	3 041	3 104
3	2		540	630	686	840	2 114	2 196	2 192	2 395	3 029	3 053	3 028	3 073
4	2		719	778	724	885	2 126	2 270	2 302	2 410	3 008	3 040	3 008	3 031
5	2		652	802	837	914	2 184	2 207	2 273	2 338	3 010	3 008	3 030	3 030

Tabulka 19: Kvartily stáří shluků uchovávaných v paměti na konci experimentu bez zvýhodňování početnějších shluků (—), pro kombinaci parametru b, minimální velikosti pro vytvoření shluku m a počtu instancí n zpracovávajících odlehlé shluky

n	m	b	—	1 000	100	10
1	1		0,977	0,977	0,977	0,974
1	2		0,976	0,975	0,976	0,975
2	2		0,971	0,975	0,975	0,974
3	2		0,976	0,974	0,970	0,971
4	2		0,978	0,974	0,975	0,974
5	2		0,973	0,975	0,970	0,975

Tabulka 20: Velikost ukazatele r bez zvýhodňování početnějších shluků (—), kombinaci parametru b, minimální velikosti vytvoření shluku m počtu instancí n zpracovávajících odlehlé shluky

n	m	b	—	1 000	100	10
1	1		8,4	8,3	8,3	8,0
1	2		9,5	9,5	9,5	9,5
2	2		9,7	9,6	9,7	9,7
3	2		10,0	9,7	10,0	9,7
4	2		10,3	10,2	10,3	10,3
5	2		10,4	10,3	10,3	10,4

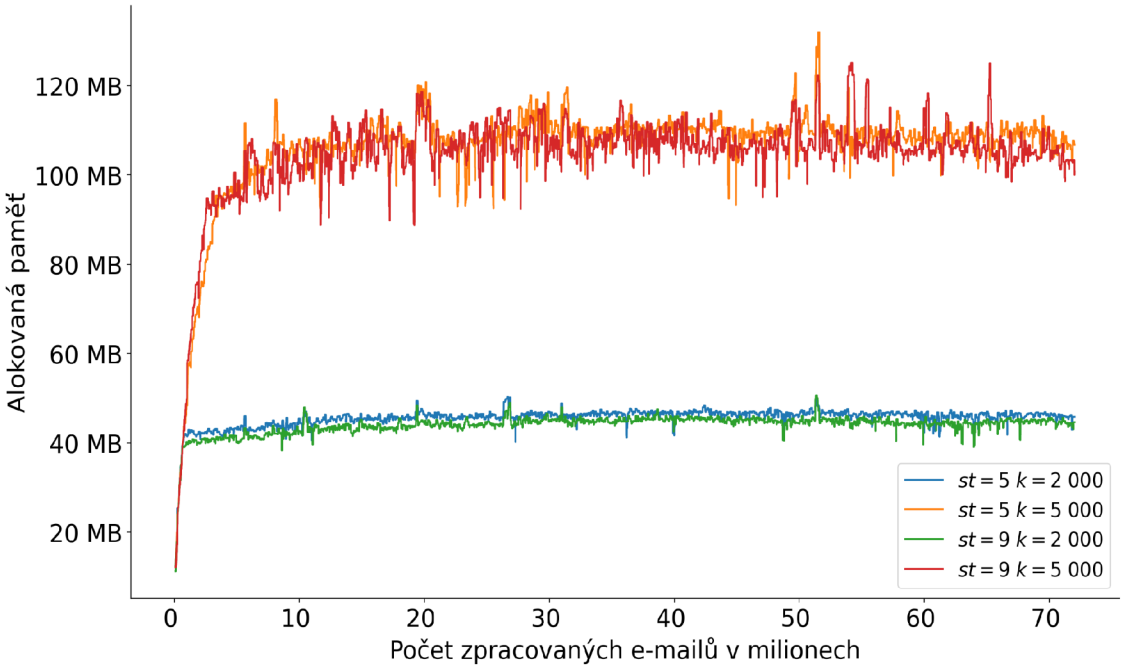
Tabulka 21: Počet e-mailů označených jako odlehlé bez zvýhodňování početnějších shluků (—), kombinaci parametru b, minimální velikosti vytvoření shluku m počtu instancí n zpracovávajících odlehlé shluky

5.4.4 Finální experiment

Po otestování navrženého řešení na části datového souboru a testování vlivu některých parametrů byl proveden finální experiment na celém datovém souboru. Cílem tohoto experimentu bylo otestovat, zda si navržené řešení udrží stabilní paměťovou náročnost a zda třídí do shluků tvořených převážně poštou vyhodnocenou jako legitimní nebo jako spam. Také se zjišťovala velikost shluků uchovávaných v paměti na konci experimentu a jejich stáří.

Pro všechny experimenty se zvolila varianta s optimalizovanou variantou výpočtu používající algoritmus Lossy Counting přes posuvná překrývající se okna. Počet posuvných oken byl zvolen 3 a jejich velikost 1 000. Pro všechny experimenty bylo zvoleno $\epsilon = 0{,}05$ a $s = 0{,}02$. Maximální počet odlehlých shluků byl zvolen 50 a počet instancí 50. Odlehlé shluky zpracovávaly 3 instance, minimální velikost pro vytvoření nového shluku byla 2 a parametr $b = 10$. Experiment se provedl pro 4 varianty, ve kterých se kombinoval maximální počet shluků rovný 2 000 a 5 000 s minimální podobností rovnou číslu 5 a 9.

S omezeným počtem shluků na 2 000 se klouzavá maxima alokované paměti na jednu instanci pohybovala něco málo přes 40 MB. S omezeným počtem na 5 000 se pohybovala okolo 110 MB. Průběh těchto klouzavých maxim zobrazuje obrázek č. 27. Zde je vidět, že maximální velikost alokované paměti byla v průběhu experimentu celkem stabilní.



Obrázek 27: Množství alokované paměti pro jednotlivé varianty během provádění shlukové analýzy na celém datovém souboru

Na konci experimentů přesáhly největší shluky uchovávané v paměti 1 milion přiřazených objektů. Zhruba polovina shluků uchovávaných v paměti byla starší než 7 hodin a 30 minut. Percentily velikostí a stáří uchovávaných shluků je možné vidět v tabulce č. 22 a č. 23.

		$st = 5$		$st = 9$	
p	k	2 000	5 000	2 000	5 000
10		23	19	12	10
20		121	88	66	40
30		588	307	289	142
40		1 734	604	1 014	392
50		3 133	988	2 078	700
60		4 903	1 649	3 454	1 200
70		7 870	2 964	6 080	2 367
80		15 320	6 068	11 900	5 095
90		47 668	18 860	32 959	15 688
100		2 193 456	2 120 416	1 570 100	1 535 791

Tabulka 22: Percentily velikostí shluků uchovávaných v paměti na konci experimentu

		$st = 5$		$st = 9$	
p	k	2 000	5 000	2 000	5 000
10		8m 57s	14m 5s	2m 21s	1m 9s
20		47m	1h 23m	21m	24m
30		2h 52m	4h 24m	1h 27m	1h 58m
40		5h 59m	7h 51m	4h 38m	5h 2m
50		9h 54m	12h 15m	7h 38m	8h 34m
60		14h 57m	15h 10m	13h 31m	13h 28m
70		17h 30m	17h 9m	16h 30m	15h 54m
80		22h 35m	21h 34m	20h 54m	19h 13m
90		23h 54m	23h 39m	23h 50m	23h 28m
100		24h	24h	24h	24h

Tabulka 23: Percentily stáří shluků uchovávaných v paměti na konci experimentu

Výsledná hodnota ukazatele přesnosti r a počet e-mailů označených jako odlehlé hodnoty, jsou pro jednotlivé varianty zobrazeny v tabulce č. 24. V tabulce č. 25 jsou zobrazeny vybrané shluky s konkrétní velikostí, které jsou tvořené převážně legitimní poštou. V tabulce č. 26 jsou zobrazeny ty, které jsou tvořené převážně spamem.

st	k	r	Odlehlé hodnoty (v %)
5	2 000	0,981	9,3
5	5 000	0,982	8,3
9	2 000	0,987	13,8
9	5 000	0,986	12,7

Tabulka 24: Velikost ukazatele r a množství e-mailů z datového souboru, které jsou označeny jako odlehlé hodnoty

Velikost	Spam (%)	Velikost	Spam (%)	Velikost	Spam (%)	Velikost	Spam (%)
2 120 413	0,08	29 892	0,03	9 964	0,07	1 000	0,00
741 099	3,16	29 813	0,00	9 932	0,07	1 000	0,10
617 676	0,03	29 665	1,86	9 900	0,02	1 000	0,00
567 509	0,00	29 326	0,44	9 871	0,00	1 000	0,10
514 069	0,00	29 320	0,02	9 848	0,03	999	0,00
459 876	0,02	29 276	0,01	9 815	0,00	998	0,50
446 710	0,90	29 050	0,00	9 712	0,55	998	0,00
445 739	0,01	28 979	0,02	9 696	0,00	998	0,00
432 145	0,01	28 842	0,00	9 672	0,00	997	0,40
425 612	0,01	28 817	0,01	9 647	0,00	997	0,00
404 260	0,02	28 599	0,00	9 643	0,00	997	0,10
402 500	0,04	28 572	0,05	9 637	0,00	997	0,20
396 901	0,00	28 546	0,04	9 627	0,05	996	0,30
392 903	0,03	28 483	0,02	9 624	0,21	995	0,00
390 448	0,00	28 306	0,00	9 559	0,97	995	1,41
389 562	0,02	28 263	0,00	9 531	0,24	994	0,00
346 245	0,06	28 259	0,04	9 524	0,02	992	0,00
344 175	0,69	28 239	1,30	9 503	0,07	992	1,71
343 338	0,05	28 073	32,81	9 475	0,02	992	1,81
340 962	0,02	28 003	0,02	9 447	0,02	992	0,00
328 263	0,02	27 946	0,06	9 444	3,88	991	0,00
312 014	0,00	27 934	0,00	9 440	0,00	991	0,20
301 629	0,01	27 888	0,00	9 424	0,01	991	2,12
299 167	0,00	27 732	0,00	9 410	20,24	990	0,00
281 367	0,00	27 575	0,00	9 402	0,00	990	2,02
279 524	0,06	27 496	0,03	9 399	0,15	989	31,14
278 745	0,02	27 447	0,02	9 396	0,04	989	0,10
276 482	0,01	27 222	0,00	9 379	0,13	989	0,00
267 616	0,07	26 983	0,00	9 374	0,91	986	1,62
266 485	0,00	26 699	0,03	9 370	0,00	986	0,20
264 941	0,01	26 644	0,03	9 334	0,40	986	0,00
261 482	0,04	26 625	0,09	9 327	0,00	985	0,00
259 147	26,00	26 612	0,08	9 325	0,17	985	14,21
250 715	0,00	26 400	0,08	9 309	0,09	985	0,30
247 903	0,03	26 133	0,00	9 290	7,45	985	0,20
239 872	0,05	26 115	0,00	9 286	0,10	983	0,31
227 870	0,00	26 031	0,06	9 282	0,06	981	0,10
226 252	0,03	25 969	0,06	9 275	0,01	981	0,00
225 081	0,00	25 941	0,02	9 273	0,27	979	0,00
224 020	0,56	25 807	0,16	9 257	47,01	979	0,00
222 683	0,01	25 802	0,00	9 256	0,02	978	0,00
219 408	0,00	25 701	0,00	9 249	0,00	978	0,20
214 786	0,30	25 578	0,02	9 247	0,01	977	45,14
212 873	1,46	25 508	0,22	9 242	0,01	977	0,00
208 047	2,69	25 095	1,09	9 232	0,00	976	0,41
204 725	0,04	24 900	0,00	9 224	0,00	974	0,21
201 083	0,00	24 878	0,00	9 224	2,87	974	0,21
197 288	0,01	24 832	0,01	9 213	0,01	973	0,31
194 172	0,00	24 759	0,00	9 175	0,00	968	0,10
187 054	7,35	24 673	0,00	9 154	2,16	968	0,10

Tabulka 25: Vybrané shluky tvořené převážně legitimní poštou

Velikost	Spam (%)	Velikost	Spam (%)	Velikost	Spam (%)	Velikost	Spam (%)
375 060	99,98	29 980	99,89	9 768	99,42	996	94,58
283 798	76,27	29 699	99,75	9 757	71,61	996	72,49
252 460	99,95	29 677	99,96	9 754	98,83	996	99,50
242 922	98,40	29 430	99,96	9 687	97,69	992	100,00
236 581	99,97	28 102	99,13	9 687	99,68	986	95,64
191 324	80,06	28 012	99,85	9 682	99,99	978	73,21
185 944	99,95	27 199	78,76	9 569	95,39	968	99,69
180 584	99,96	27 050	99,92	9 459	83,50	967	60,81
180 573	99,94	26 309	99,96	9 339	99,60	964	70,85
176 034	99,96	26 131	95,08	9 279	99,88	962	62,27
157 809	99,99	25 527	96,72	9 263	69,33	959	64,75
138 323	93,88	25 152	100,00	9 152	65,81	954	55,35
128 025	93,11	24 849	97,88	8 950	93,06	952	85,40
117 210	99,17	24 665	99,27	8 849	88,25	952	100,00
116 616	99,96	23 713	65,19	8 638	98,24	945	53,02
111 856	99,95	23 415	98,63	8 459	69,13	945	99,05
109 354	99,39	22 915	98,06	8 426	99,94	944	100,00
104 266	99,98	21 870	94,64	8 401	99,89	935	100,00
97 933	67,60	21 859	99,96	8 358	98,86	934	100,00
82 338	99,88	21 781	99,99	8 348	99,98	931	70,46
80 995	99,87	21 707	99,66	8 335	99,90	929	99,78
79 630	99,81	21 663	100,00	8 288	99,78	928	55,28
78 695	68,76	21 507	99,99	8 215	100,00	913	76,56
77 354	98,77	21 397	95,39	8 193	85,57	912	99,89
73 778	54,85	21 320	99,10	8 178	99,90	909	80,20
73 408	99,10	21 208	77,15	8 133	98,08	907	72,33
73 209	98,92	21 187	99,97	8 125	99,99	906	51,32
71 484	89,95	20 998	99,29	8 071	99,72	900	90,00
67 647	96,43	20 891	89,75	8 058	97,62	896	99,78
66 316	99,97	20 356	100,00	8 021	99,84	884	98,19
64 924	57,92	20 106	99,99	7 989	98,69	877	65,56
59 670	99,94	19 867	98,59	7 952	97,25	876	95,21
59 003	99,87	19 789	96,79	7 879	91,86	871	99,89
55 159	100,00	19 708	99,96	7 860	59,25	869	88,26
53 841	98,66	19 463	95,80	7 819	95,75	868	98,27
53 356	93,07	19 458	100,00	7 435	99,88	866	93,07
50 035	99,97	19 381	99,83	7 398	99,09	863	74,74
48 215	99,94	19 160	99,79	7 355	99,55	862	78,54
48 190	98,22	19 032	99,82	7 284	99,89	860	62,79
47 892	99,97	18 935	82,10	7 267	99,94	853	85,93
47 453	99,97	18 863	100,00	7 261	99,99	849	100,00
47 429	93,68	18 856	99,53	7 258	99,88	846	82,98
47 170	94,15	18 834	96,45	7 251	99,92	840	75,83
46 211	72,72	18 800	99,98	7 244	99,86	840	98,21
45 829	97,59	18 657	99,19	7 240	99,75	839	100,00
41 762	99,67	18 645	99,94	7 237	99,94	838	96,30
39 486	90,87	18 412	99,97	7 232	99,94	837	97,73
39 048	69,89	18 189	99,82	7 232	91,77	836	99,64
38 621	100,00	18 084	99,49	7 230	99,97	833	100,00
38 530	99,94	17 983	68,20	7 226	99,96	831	99,76

Tabulka 26: Vybrané shluky tvořené převážně spamem

6 Závěr a shrnutí

V teoretické části této práce byla představena problematika analýzy datových toků, byly zde popsány algoritmy pro určení aproximovaných relativních četností a nakonec algoritmy StreamCluCD a CSketch, které jsou určeny pro shlukovou analýzu datových toků s kategorickými atributy. Tyto algoritmy jsou jednoprůchodové a pro výpočet podobnosti objektu se shlukem používají aproximované relativní četnosti. Za cenu nepřesností v relativní četnosti dosahují velmi příznivé paměťové náročnosti.

Hlavním cílem této práce bylo navrhnout řešení shlukové analýzy e-mailů, které by mohlo být využitelné antispamovým systémem. Na ID jednotlivých shluků by se sledovaly reputace, podle kterých by navržené řešení přispívalo určitou vahou do celkového skóre e-mailu stanoveného antispamem. Po překročení určité hranice je e-mail označen jako spam. Všechna data pro provedení shlukové analýzy a přiřazení e-mailu do shluku je potřeba držet v paměti RAM, protože čas na roztřídění příchozího e-mailu je omezený a provádění velkého množství dotazů do databáze by velmi zpomalilo zpracování daného e-mailu. Dále bylo nutné navrhnout synchronizaci dat mezi jednotlivými instancemi.

Navržené řešení z velké části vychází z představených algoritmů StreamCluCD a CSketch. K výpočtu podobnosti objektu se shlukem používá aproximované relativní četnosti hodnot v daném shluku za určitý počet naposledy přidaných objektů, aby se zohlednil časový vývoj vlastností v datovém toku.

Navržené řešení dosahuje příznivé paměťové náročnosti, jejíž velikost závisí na zvoleném maximálním počtu shluků. Pro 2 000 shluků se maximální alokovaná velikost paměti pohybuje něco málo přes 40 MB, pro 5 000 je to pak okolo 110 MB. Navržené řešení dokáže třídit e-maily do shluků dostatečně rychle. Průměrný čas na přiřazení e-mailu do shluku se pro zvolené parametry pohyboval do 1 milisekundy. Ukázalo se, že sériové provedení shlukování a paralelní provedení shlukování se synchronizací dat produkují velmi podobné výsledky.

V experimentech bylo navržené řešení testováno na datovém souboru obsahujícím e-maily zpracované antispamem za jeden den. Tento datový soubor obsahuje okolo 72 milionů e-mailů z nichž 10 milionů je vyhodnoceno antispamem jako spam. Navržené řešení třídí e-maily do shluků tak, že shluky jsou převážně tvořené poštou vyhodnocenou antispamem jako legitimní nebo jako spam. Zhruba 98 % e-mailů přiřazených do shluku se nachází v tom, ve kterém převažují ty se stejným vyhodnocením a z velké části tak kopíruje existující antispamové řešení.

Podle dosažených výsledků bylo rozhodnuto, že se bude pokračovat ve vývoji navrženého řešení a bude testováno v provozu. Cíl této práce lze tak považovat za splněný. Navržené řešení označuje přibližně 10 % e-mailů jako odlehlé hodnoty. Bylo by dobré jej v tomto vylepšit, aby se tento počet snížil. Při testování v provozu by se mohla zjistit příčina, proč je těchto 10 % e-mailů označeno jako odlehlé hodnoty. Toto číslo by se mohlo vylepšit např. výběrem vhodnějších atributů pro provedení shlukování nebo zvolením vhodnějších parametrů. Dále by bylo vhodné zkoumat, zda budou výsledky shlukové analýzy srovnatelné po nepřetržitém provozu

řešení po několik dní. Také by bylo vhodné zkoumat, zda nedochází u shluků obsahujících převážně legitimní poštu k takovému vývoji, že se stanou shlukem obsahující převážně spam a naopak.

Seznam použité literatury

[1] AGGARWAL, C. C.; YU, P. S.; HAN, Jiawei; WANG, Jianyong. A Framework for Clustering Evolving Data Streams. In *Proceedings 2003 VLDB Conference*. 1. Berlin, Germany : Morgan Kaufmann, 2003, 81–92. ISBN 978-0-12-722442-8. Dostupné z: `https://www.vldb.org/conf/2003/papers/S04P02.pdf`.

[2] AGGARWAL, Charu. A Framework for Clustering Massive-Domain Data Streams. In *2009 IEEE 25th International Conference on Data Engineering*. Shangai, China : IEEE, 2009, 102–113. ISBN 978-1-4244-3422-0. Dostupné z: `http://charuaggarwal.net/cskrevise.pdf`.

[3] BOSE, Prosenjit; KRANAKIS, Evangelos; MORIN, Pat; TANG, Yihui. Bounds for Frequency Estimation of Packet Streams. In *SIROCCO 10: Proceedings of the 10th Internaltional Colloquium on Structural Information Complexity*. 2003, 33–42. Dostupné z: `https://cglab.ca/~morin/publications/traffic/streaming-sirocco.pdf`.

[4] CAO, Fuyuan; HUANG, J. Z.; LIANG, Jiye. Trend analysis of categorical data streams with a concept change method. *Information Sciences*. 2014, vol. 276, 160–173. ISSN 0020-0255. Dostupné z: `http://cicip.sxu.edu.cn/docs/2019-04/7a87d5966402426ea041a2808df163c7.pdf`.

[5] CORMODE, Graham; HADJIELEFTHERIOU, Marios. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*. 2008, vol. 1, no. 2, 1530–1541. ISSN 2150-8097. Dostupné z: `http://hadjieleftheriou.com/papers/vldb08-2.pdf`.

[6] CORMODE, Graham; MUTHUKRISHNAN, S. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. In *LATIN 2004: Theoretical Informatics*. Springer Berlin Heidelberg, 2004, 29–38. ISBN 978-3-540-21258-4. Dostupné z: `https://dsf.berkeley.edu/cs286/papers/countmin-latin2004.pdf`.

[7] DEMAINE, E. D.; LÓPEZ-ORTIZ, A.; MUNRO, J. I. Frequency Estimation of Internet Packet Streams with Limited Space. In *Algorithms - ESA 2002*. Rome, Italy : Springer Berlin Heidelberg, 2002, 348–360. ISBN 978-3-540-44180-9.

[8] Docker Inc. *Docker docs* [online]. c2023 [cit. 26. 3. 2023]. Get started, Part 1: Overview. Dostupné z: `https://docs.docker.com/get-started/`.

[9] Docker Inc. *Docker* [software]. Verze 20.10.21. c2023 [cit. 27. 3. 2023]. Dostupné z: `https://www.docker.com/`.

[10] HE, Zengyou; XU, Xiaofei; DENG, Shengchun; HUANG, J. Z. Clustering Categorical Data Streams. *arXiv preprint arXiv:cs/0412058v1 [cs.DB]* [online]. 13. 12. 2004 [cit. 19. 2. 2023]. Dostupné z: `http://arxiv.org/abs/cs/0412058`.

[11] ITSKAWAL2000; ANSHITAAGARWAL. E-Mail Format. In *GeeksForGeeks* [online]. Poslední aktualizace 20. 10. 2020 [cit. 26. 3. 2023]. Dostupné z: `https://www.geeksforgeeks.org/e-mail-format/`.

[12] KARP, Richard M.; SHENKER, Scott; PAPADIMITRIOU, Christos H. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*. 2003, vol. 28, no. 1, 51–55. ISSN 0362-5915.

[13] Kubernetes Authors. *Kubernetes* [software]. Verze 1.27. c2023 [cit. 27. 3. 2023]. Dostupné z: `https://kubernetes.io/`.

[14] MANKU, G. S.; MOTWANI, Rajeev. Approximate Frequency Counts over Data Streams. In *Proceedings 2002 VLDB Conference*. 1. Hong Kong SAR, China : Morgan Kaufmann, 2002, 346–357. ISBN 978-1-55860-869-6. Dostupné z: `https://www.vldb.org/conf/2002/S10P03.pdf`.

[15] MEDJEDOVIC, Dzejla; TAHIROVIC, Emin. *Algorithms and data structures for massive datasets*. Manning Publications Co, 2022. ISBN 978-1-61729-803-5.

[16] METWALLY, Ahmed; AGRAWAL, Divyakant; AMR, E. A. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Database Theory - ICDT 2005*. Edinburg, UK : Springer Berlin Heidelberg, 2005, 398–412. ISBN 978-3-540-24288-8. Dostupné z: `https://www.cse.ust.hk/~raywong/comp5331/References/EfficientComputationOfFrequentAndTop-kElementsInDataStreams.pdf`.

[17] Microsoft. *Azure* [online]. c2023 [cit. 26. 3. 2023]. Co je Kubernetes?. Dostupné z: `https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-kubernetes/`.

[18] NATS authors. *NATS* [software]. Verze 2.9.10. c2023 [cit. 27. 3. 2023]. Dostupné z: `https://nats.io/`.

[19] Nats authors. *NATS Docs* [online]. Poslední aktualizace 16. 1. 2022 [cit. 27. 3. 2023]. Core Nats, Publish-Subscribe. Dostupné z: `https://docs.nats.io/nats-concepts/core-nats/pubsub`.

[20] SHEN, Wei. *An implementation of Count-Min Sketch in Golang* [software]. Commit b4482ac. Poslední aktualizace 19. 5. 2016 [cit. 27. 3. 2023]. Dostupné z: `https://github.com/shenwei356/countminsketch`.

[21] ZENGYOU, He; XIAOFEI, Xu; SHENGCHUN, Deng. Squeezer: An efficient algorithm for clustering categorical data. *Journal of Computer Science and Technology*. 2002, vol. 17, no. 5, 611–624. ISSN 1000-9000.

[22] ZHANG, Linfeng; GUAN, Yong. Frequency Estimation over Sliding Windows. In *2008 IEEE 24th International Conference on Data Engineering*. Cancun, Mexico : IEEE, 2008, 1385–1387. ISBN 978-1-4244-1836-7. Dostupné z: `https://www.researchgate.net/publication/220966535_Frequency_Estimation_over_Sliding_Windows`.

Seznam obrázků

Seznam tabulek

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Tomáš Kratochvíl**
Studium:	I2000791
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Shluková analýza datových toků**
Název diplomové práce AJ:	Datastream clustering

Cíl, metody, literatura, předpoklady:

Cílem práce je popsat problematiku analýzy datových toků, navrhnout a zhodnotit řešení pro shlukovou analýzu e-mailů využitelnou antispamovým systémem

Rámcová osnova:

- Úvod do problematiky analýzy datových toků (pojem datový tok a problémy, které přináší analýza datových toků)
- Algoritmy a datové struktury pro analýzu datových toků se zaměřením na kategorická data
- Návrh řešení pro shlukovou analýzu datových toků v distribuovaném systému
- Implementace řešení
- Analýza a porovnání dosažených výsledků
- Závěr a shrnutí výsledků

Literatura : Dzejla Medjedovic, Emin Tahirovic, and Ines Dedovic Algorithms and Data Structures for Massive Datasets Alaettin Zubaroglu and Volkan Atalay. Data stream clustering: a review. Artificial Intelligence Review, 54(2):1201–1236, February 2021 Charu Aggarwal. A Framework for Clustering Massive-Domain DataStreams. In2009 IEEE 25th International Conference on Data Engi-neering, pages 102–113, Shanghai, China, March 2009. IEEE. ISSN:1084-4627.

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	prof. RNDr. PhDr. Antonín Slabý, CSc.
Datum zadání závěrečné práce:	26.1.2021