FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Visual content search from video analysis
Student:	Bc. Lukáš Molnár
Supervisor:	Ing. František Fait
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2019/20

Instructions

Quantasoft has a proprietary video analysis system. The system focuses on tracking of car parks and public areas that generate variable metadata from each major image.

The subject of the thesis is the research and selection of technologies for storing large volumes of metadata with emphasis on visual content search (VCS-VA). VCS-VA must support semantic object search - based on textual descriptions or visual contents, i.e. searching for a particular object or person according to an image template.

1. Design and implement a versatile time-series based metadata storage for video analytics.
2. Design and implement .NET Core 2.1 compliant APIs for storing metadata from video analytics.
3. Design and implement .NET Core 2.1 compliant APIs for querying metadata from video analytics.
4. Test and evaluate performance of your solution.

References

[1] S.Feng, Z. Li, Y. Xu and J. Sun, "Compact scalable hash from deep learning features aggregation for content de-duplication," *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, Luton, 2017, pp. 1-5. doi: 10.1109/MMSP.2017.8122286

[2] Douze, M., Johnson, J.,& Jegou, H. (2017, March29). Faiss: A library for efficient similarity search. Retrieved from *https://code.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/*

[3] Stein, Benno & Potthast, Martin. (2018). Applying Hash-based Indexing in Text-based Information Retrieval. Retrieved from https://www.researchgate.net/publication/228543039_Applying_Hash-based_Indexing_in_Text-based_Information_Retrieval

[4] Cao, Y., & Guo, X.(2016, February 24). Online Security Analytics on Large Scale Video Surveillance System. Retrieved from *https://www.slideshare.net/SparkSummit/online-security-analytics-on-large-scale-video-surveillance-system-by-yu-cao-and-xiaoyan-guo*

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague September 13, 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Vyhľadávanie vizuálneho obsahu z analýzy videa

Bc. Lukáš Molnár

Katedra softwarového inženýrství
Vedúci práce: Ing. František Fait

28. júna 2019

Poďakovanie

Najväčšia vďaka patrí mojej rodine menovite manželke Eve a synovi Adamovi, že boli trpezliví a dali mi priestor na dokončenie záverečnej práce. Ďalej by som chcel poďakovať firme Quantasoft s.r.o. a jej pracovníkom, že mi umožnili využiť ich výpočtové a dátové zdroje a boli nápomocní pri vzájomných konzultáciách.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. Ďalej prehlasujem, že som s ČVUT uzavrel dohodu, na základe ktorej sa ČVUT vzdalo práva na uzavrenie licenčnej zmluvy o používaní tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona. Táto skutočnosť nemá vplyv na ust. § 47b zákona č. 111/1998 Sb. o vysokých školách.

V Prahe 28. júna 2019 .....

Odkaz na túto prácu

Molnár, Lukáš. *Vyhľadávanie vizuálneho obsahu z analýzy videa.* Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Cieľom tejto diplomovej práce je analýza, návrh a implementácia jednotného úložiska pre metadáta z analýzy vizuálneho obsahu. Okrem náhodného prístupu podporuje úložisko vyhľadávanie na základe podobnosti vizuálnych objektov. Práca sa zaoberá aj návrhom a implementáciou API pre toto úložisko, návrhom manažovacej databázy pre poskytovateľov metadát a návrhom formátu metadát ukladaných v úložisku. Výsledky tejto práce sú nasadené v prostredí spoločnosti Quantasoft.

Klíčová slova počítačové videnie, inteligentná analýza videa, vyhľadávanie objektov podľa podobnosti, embedding, časové postupnosti

Abstract

The aim of this thesis is to analyze, design and implement a unified repository for metadata from visual content analysis. In addition to random access, the repository supports search based on visual object similarity. The thesis also deals with the design and implementation of API for this repository, the design of database management for metadata providers and the design of the metadata format stored in the repository. The results of this work are deployed in Quantasoft.

Keywords computer vision, intelligent video analysis, similarity object search, embedding, time series

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

S masívnym rozmachom výpočtových kapacít za prijateľnú cenu vhodných na spracovanie obrovských objemov dát, grafických kariet, multiprocesorových počítačov na jednej strane a na druhej strane cloudových služieb ako AWS, Azure či Google Cloud, prišiel ruka v ruke obrovský boom vo výskume a možnostiach spracovania obrazu, počítačovom videní alebo anglicky *computer vision* (CV).

Spoločnosť Quantasoft má vo svojom portfóliu sadu nezávislých analytických procesov týkajúcich sa počítačového videnia zameraných na rôzne segmenty trhu. Spoločným znakom procesov je, že výsledok popisuje procesovaný snímok alebo obrázok. Spomínané popisy sa donedávna líšili v závislosti na určení. Mnohé popisy sa využili iba dočasne, napríklad na vizualizáciu priamo do videa, ale v zápätí sa vyhodili. Iné pracovníci ukladali do rozličných binárnych a textových formátov alebo si popisy vymieňali v JSON formáte cez HTTP protokol. Skrátka, v spoločnosti panovala nejednotnosť.

Cieľ práce

Cieľom tejto práce je navrhnúť a implementovať jednotné úložisko pre metadáta z analýzy vizuálneho obsahu a následné vyhľadávanie v ňom. Pri riešení týchto úloh sme postupovali následovne:

1. Analýza problému
2. Prieskum trhu
3. Výber technológie
4. Návrh riešenia
5. Implementácia riešenia
6. Testovanie riešenia

Súčasťou implementácie bol aj návrh aplikačných rozhraní v niekoľkých programovacích jazykoch.

Tomuto postupu odpovedá rozvrhnutie tejto práce do jednotlivých kapitol a sekcií.

Okrem splnenia cieľa práce sme navrhli jednotnú podobu metadát z rôznorodých analytických procesov. Pri plnení cieľov práce sme takisto navrhli a implementovali manažovacú databázu, ktorá slúži na evidenciu zdrojov médií, poskytovateľov metadát a známych subjektov. Navrhli sme aj spoločný formát na reprezentáciu metadát z analýzy obrazu.

Formátovanie

V celej záverečnej práci je použitý nasledujúci štýl formátovania textu:
`Rezom písma s rovnakým odstupom` budu označené názvy tried, funkcií, databázových tabuliek, procedúr, stĺpcov a identifikátorov.
Kurzívou budú označené cudzie výrazy alebo odborné skratky.

Kapitola 1

Problematika analýzy videa

Analýza videa je proces, pri ktorom sa obrazová informácia pomocou širokej škály algoritmov spracuváva do podoby metadát. Analýza videa spadá pod disciplínu počítačového videnia, anglicky *computer vision*.

> Cieľom počítačového videnia je vyťaženie užitočných informácií z obrázkov. Toto sa ukázalo ako nečakane náročná úloha; za uplynulé štyri dekády zamestnala tisíce inteligentných a tvorivých myslí, a napriek tomu sme stále ďaleko od vytvorenia univerzálneho "vidiaceho stroja".
>
> — Strana 16, Computer Vision: Models, Learning, and Inference, 2012.

V konečnom dôsledku pri analýze videa ide o spracovanie obrázkov, nakoľko video je sekvencia obrazových snímkov, ktoré sú video dekóderom skonvertované a uložené do kompatibilného kontajnera. Kontajnery podporujú dvojaký spôsob transportu záznamu, buď vo forme súborov na ľubovoľnom fyzickom médiu alebo ako tok dát, tzv. *video stream*. Spôsob šírenia obrazového záznamu má zásadný vplyv na možnosti spracovania videa, vymedzuje minimálne hardvérové, softvérové a časové nároky na implementované riešenie.

V prostredí spoločnosti Quantasoft je na video stream alebo video súbor nazerané ako na sekvenciu dvojíc x_t, kde snímok x je RGBA bitmapa s časovou značkou t. Časová značka môže reprezentovať poradové číslo snímku s počiatkom v 1. (`FrameId`) alebo aktuálny čas snímku v dobe záznamu (`FrameTsUtc`).

1.1 Spracovanie v reálnom čase

Magickou hranicou alebo obvyklým orientačným bodom je spracovanie v reálnom čase. Pod týmto pojmom si môžeme predstaviť spracovanie videa počas jeho prehrávania v jeho originálnej kvalite, tj. pri zachovaní pôvodnej šírky a výšky obrazu, obrazového rozlíšenia a pri zachovaní počtu prehrávaných snímkov za sekundu (*fps*).

1.1.1 Príklad spracovania v reálnom čase

Majme video vo *FullHD* rozlíšení a priemerne 30 snímkov za sekundu bez prekladávania (angl. *progressive scan*). Pre jednoduchosť a približnú predstavu uvažujme nad RGBA pixelmi, kde jedna farebná zložka v nekomprimovanom stave zaberá 8 b. Za jednu sekundu cez spracovanie videa pretečie prihližne 1,85 GiB surového videa. Jeden snímok má okolo 63,3 MiB a jeho spracovanie v reálnom čase musí analytický nástroj stihnúť do 33,3 milisekundy a do tejto doby musí stihnúť výsledky analýzy zobraziť alebo uložiť.

1.2 Algoritmy analýzy videa

Inteligentná analýza videa (IVA) pre daný snámok x_t je odkázaná na algoritmy detekcie objektov. Tieto typy algoritmov sú trénované na obrovských dátových množinách s vysvetlenými preddefinovanými objektami. Na týchto množinách sú natrénované hlboké neurónové siete (anglicky *deep neural networks*), takzvaný *model*. Jeho výstupom sú detekované objekty a ich pozície v rámci x_t.

1.3 Kroky analýzy videa

V nasledujúcich sekciách sú popísané kroky analýzy videa, ktoré sa štandardne používajú v analytickom procese. Kroky sa môžu mierne líšiť podľa segmentu. Všetky metódy spracovania sú všeobecne nezávislé na okolitých snímkoch s výnimkou trackingu a reidentifikácie.

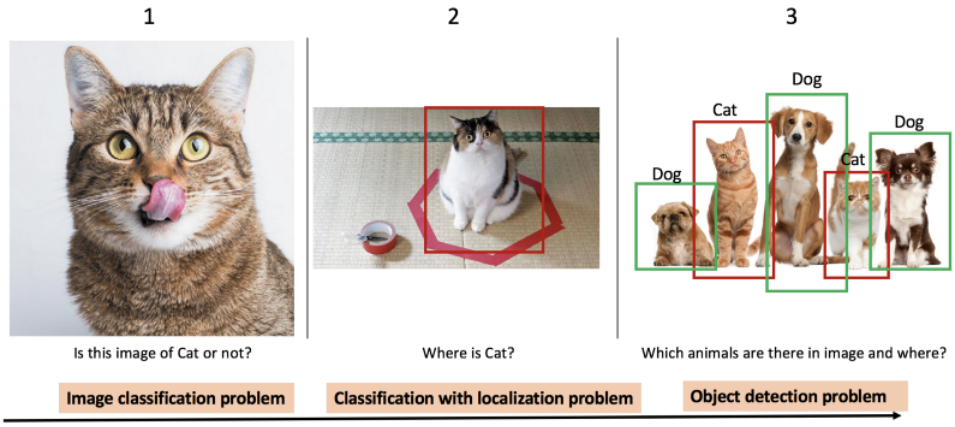
1.3.1 Dekódovanie

Dekódovanie je operácia, pri ktorej sa vstupné súbory alebo streamy prevedú do surovej podoby, matice pixelov. Tento krok je možné realizovať pomocou rozličných softvérových nástrojov, napr. open source knižnicou OpenCV.

1.3.2 Predspracovanie

Predspracovanie je voliteľný krok, v ktorom sa môže so snímkom vykonať jedna alebo niekoľko z nasledujúcich operácií:

- aplikácia kalibrácie obrazu z video kamier
- aplikácia transformačných operácií ako zmena veľkosti alebo rotácia
- orezanie oblasti záujmu
- aplikácia farebných filtrov
- rozbalenie obrazu kamery s rybím okom (angl. *dewarping*)



Obr. 1.1: Porovnanie klasifikácie, lokalizácie a detekcie, prevzaté z [1]

1.3.3 Klasifikácia, lokalizácia a detekcia objektov

Klasifikácia obrázka — určenie, či sa na obrázku nachádza známy typ objektu.

Klasifikácia s lokalizáciou — označenie obdĺžnikovou oblasťou, tzv. *bounding box*, kde sa na snímku nachádza známy typ objektu

Detekcia objektov — klasifikácia a lokalizácia všetkých známych typov objektov

Vizuálne porovnanie je zobrazené na obrázku 1.1. Medzi najbežnejšie využitia detekčných algoritmov patrí

Klasifikácia v medicíne — diagnostika chorôb z medicínskych snímkov

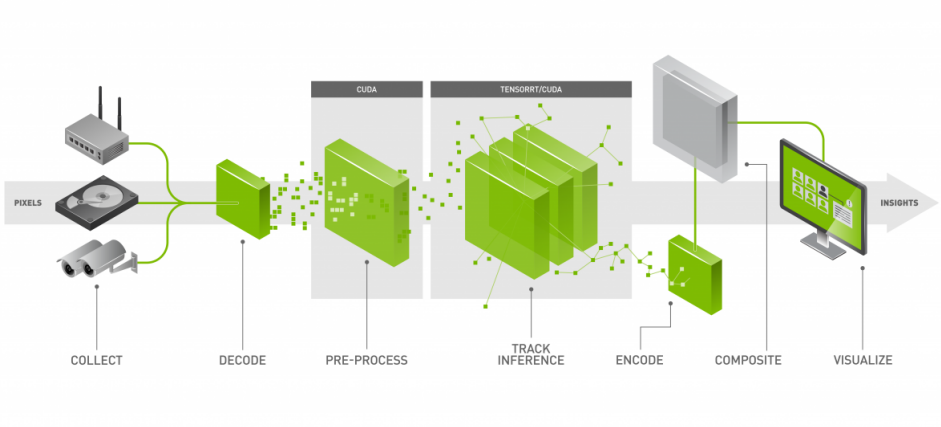
Detekcia tvárí — tváre osôb obdržia identifikačný vektor, *embedding*

Detekcia vozidiel — dochádza aj ku extrakcii registračnej značky (LPR), identifikátora vozidla

Segmentácia — spresnenie detekcie, presnejšie hranice objektu sa určia vo forme polygónov

1.4 NVIDIA DeepStream SDK

NVIDIA DeepStream SDK slúži na rýchle a efektívne spracovanie streamov videa na špecializovanom hardvéri. Podporuje rozšíriteľnosť o moduly napísané



Obr. 1.2: Diagram pipeline NVIDIA DeepStream SDK, prevzaté z [2]

v jazyku C/C++. V jednej *pipeline* vyobrazenej na obrázku 1.2, ktorá približne odpovedá analytickému procesu v spoločnosti Quantasoft, postupne prebiehajú nasledujúce kroky:

1. sťahovanie a dekódovanie videa
2. predspracovanie videa
3. spúšťanie sekvencie detekcií
4. sledovanie objektov
5. vizualizácia metadát
6. zakódovanie spracovaného vstupu do súboru alebo streamu

1.5 Vizuálny obsah

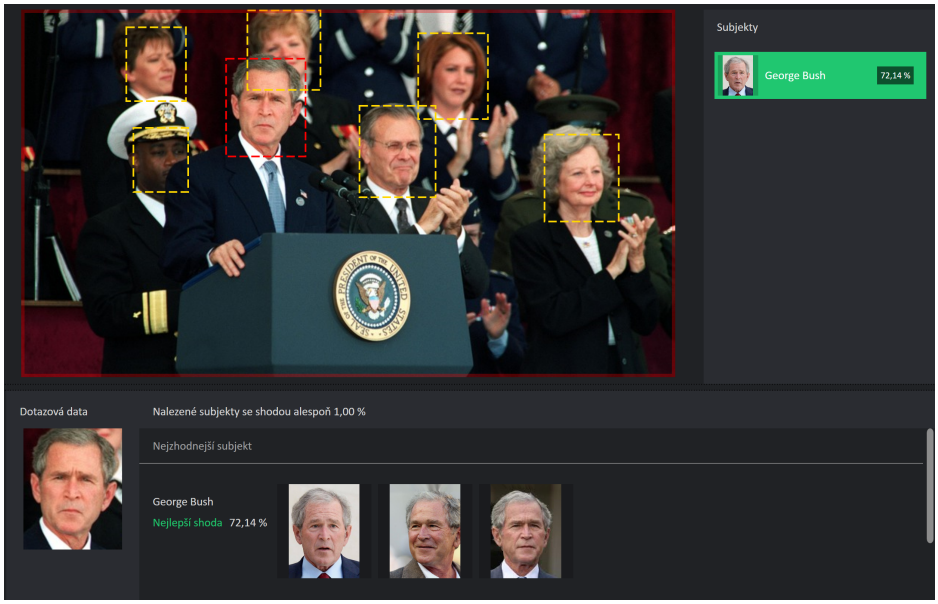
Predmety a osoby zachytené na video nahrávke alebo prenose označujeme spoločným výrazom *subjekt*. Tvoria pre nás vizuálny obsah. Obrázky 1.3, 1.4 a 1.5 zobrazujú rôzne subjekty, príklady vizuálneho obsahu.

1.6 Výstupy z analytického procesu

Po spracovaní každého snímku môžu byť podľa typu objektu na výstupe nasledujúce informácie:

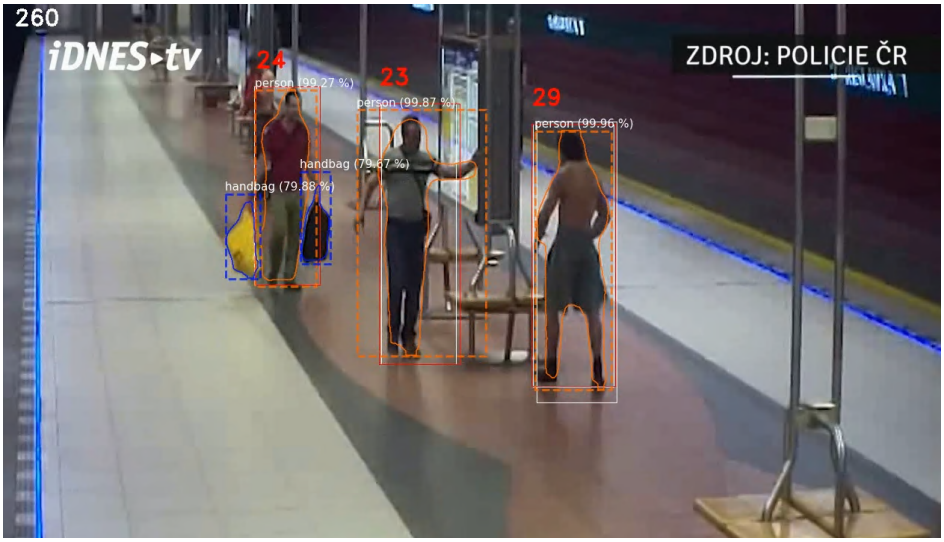
Klasifikácia — príslušnosť objektu do niektorej z tried

Bounding box — horizontálne a vertikálne ohraničenie objektu



Obr. 1.3: Vizualizácia metadát v aplikácii rozponávania tvárí

Obr. 1.4: Vizualizácia metadát z parkoviska



Obr. 1.5: Ukážka vizualizácie metadát z bezpečnostnej kamery

Embedding — identifikátor alebo vektor z metrického priestoru, detailne popísaný v kapitole 6.5

Feature — hodnota s odhadom pravdepodobnosti zhody s realitou, ako napríklad odhad veku a pohlavia osoby alebo textová reprezentácia registračnej značky vozidla (LPR)

Atribút — hodnota, ktorá je jednoznačne určená, ako napr. histogram alebo farba objektu

Kapitola 2

Analýza a návrh

2.1 Požiadavky na úložisko metadát a manažovaciu databázu

Požiadavky na vlastnosti úložiska vznikli na základe konzultácií s firmou Quantasoft. Požiadavky sú rozdelené na funkčné a nefunkčné.

2.1.1 Funkčné a výkonnostné požiadavky na úložisko metadát

2.1.1.1 F1 Analytické schopnosti

Úložisko umožní spúšťať základné agregačné funkcie a analytické dopyty.

2.1.1.2 F2 Programovateľnosť

Úložisko umožní prácu s dátami pomocou skriptovacieho programovacieho jazyka.

2.1.1.3 F3 Náhodný prístup

Úložisko umožní prístup k ľubovoľnému snímku v ľubovoľnom čase.

2.1.1.4 V1 Škálovateľnosť úložiska

Kapacita úložiska bude rozšíriteľná pomocou pridania diskov.

2.1.1.5 V2 Škálovateľnosť do šírky

Priepustnosť úložiska bude môcť byť navýšená zvýšením počtu inštancií. Pozn.: jedná sa o horizontálnu škálovateľnoť, anglicky *scale-out*.

2.1.1.6 V3 Škálovateľnosť do výšky

Priepustnosť úložiska bude úmerná výkonu inštancie. Pozn.: jedná sa o vertikálnu škálovateľnosť, anglicky *scale-up*.

2.1.2 Nefunkčné požiadavky na úložisko metadát

2.1.2.1 NF1 Nasadenie

Úložisko bude možné prevádzkovať v prostredí operačného systému Microsoft Windows Server 2019 alebo Linux (Ubuntu Server 18.04 LTS).

2.1.2.2 NF2 Multiframeworková implementácia API

API bude implementované pre tieto programovacie jazyky:

a) **C#** — na .NET Core 2.2

b) **Python** — minimálna podporovaná verzia 3.5

c) **C++** — minimálna podporovaná verzia štandardu C++14

2.1.3 Funkčné požiadavky na manažovaciu databázu

2.1.3.1 F4 Správa paralelného prístupu

Manažovacia databáza musí zamedziť vzniku konfliktov v prípade viacpoužívateľského paralelného prístupu pri aktualizácií rovnakých záznamov.

2.2 Nefunkčné požiadavky na manažovaciu databázu

2.2.1 NF3 Nasadenie na Microsoft SQL Server 2017

Manažovacia databáza musí podporovať Microsoft SQL Server 2017.

2.3 Formy vyhľadávania a prístupu k metadátam

2.3.1 Náhodný prístup

Pod pojmom náhodný prístup k metadátam rozumieme dopyty, ktoré vedú k prístupu k ľúbovoľnému snímku alebo rozsahu snímkov. Analytický proces sa skladá zo závislých a nezávislých krokov. Závislé kroky požadujú výsledky z iných krokov procesu, ktoré si dohľadajú v úložisku metadát. Náhodný prístup k ľubovoľnému snímku v ľubovoľnom čase umožní neblokujúce reťazenie analytických krokov a uľahčí distribúciu na samostatné výpočtové uzly.

Náhodný prístup sa bude často využívať v analytickom procese a pri vizualizácii metadát. Nech pracujeme s video streamom alebo so súbormi, tak sa v celom známom rozsahu videí, archíve, potrebujeme jednoznačne odkazovať na konkrétny snímok.

2.3.1.1 Video súbory

So súbormi je situácia prehľadná. Video súbor má jasne daný začiatok a koniec, čo umožňuje snímok jednoznačne identifikovať jeho poradovým číslom `FrameId` alebo `SequenceId`.

2.3.1.2 Video streamy

Pri streame nevieme, kedy sa začal a kedy skončí, preto predpokladáme, že teoreticky môže byť nekonečný. Mohli by sme snímky so streamu označovať poradovým číslom rovnako ako sa to deje pri súboroch, ale museli by sme autoritatívne určiť, kde je jeho začiatok. Snímok na streame z videa budeme identikovať pomocou koordinovaného univerzálneho času UTC.

2.3.2 Podľa predlohy

Hlavným záujmom záverečnej práce je vyvinúť spôsob, ako v obrovskej množine predpripravených metadát nájsť odpovedajúce záznamy aj bez toho, aby sme mali k dispozícii zdroj médií. V praxi to bude znamenať, že úložisko metadát bude separované od video archívu a bude komunikovať cez pripravené aplikačné rozhranie. V obsahu metadát musí byť uložená reprezentácia objektu, aby ho bolo možné porovnávať s predlohou, ktorá bola pripravená rovnakou metódou. V tomto procese sa musí uplatniť `Confidence`, miera podobnosti, ktorá určí hranicu, kedy dve reprezantácie považujeme za podobné.

2.3.3 Podľa popisu

Aplikačné rozhranie by malo poskytnúť prostriedky, ako dostatočne robustne popísať vyhľadávaný objekt. V jednej variante by sa popis zúžitkoval pri filtrovaní výsledkov na základe typu objektu alebo aj hodnôt atribútov. V druhej variante by sa z popisu vygenerovala predloha, približná obrazová reprezentácia objektu a aplikovalo by sa vyhľadávanie podľa predlohy s primerane nastavenou hodnotou `Confidence`.

2.4 Manažovacia databáza

V zadaní diplomovej práce sa zmienka o evidencii zdrojov, subjektov a poskytovateľov nevyskytuje, ale pri navrhovaní systému sa táto časť ukázala ako kľúčová kvôli korektnému zaobchádzaniu s údajmi rozličného pôvodu a

formátu. Evidencia sa uchováva v OLTP systéme, zvyčajne v relačnej databáze. V implementácii bola zvolená Microsoft SQL Server 2017, pretože v spoločnosti Quantasoft je to dlhodobo preferovaná databáza. Na operačnom systéme Linux do úvahy prichádza aj PostgreSQL alebo MariaDB.

Funkčná požiadavka F4 na správu paralelného prístupu zo subsekcie 2.1.3.1 v zapisovacích procedúrach bola splnená kontrolou hodnoty `DataVersion` zo vstupu a voči hodnote v databáze. Ak sú hodnoty rovnaké, teda volanie je realizované nad aktuálnou verziou údajov, volanie procedúry umožní zápis, ak nie, volanie skončí s chybou a zabráni sa tak zápisu dát s konfliktnou verziou. Hodnota `DataVersion` pochádza z rovnomenného stĺpca tabuliek dátoveho typu `RowVersion`. Hodnota je generovaná automaticky pri každej zmene riadku, v tabuľke nadobudne vyššie číslo [3]. Spolu s vhodnou úrovňou izolácie transakcie dochádza k optimistickej správe paralelného prístupu. Riešenie chybových stavov bolo pre vyššiu flexibilitu presunuté na stranu klienta. Podľa potreby má možnost zopakovať operáciu s aktuálnou verziou alebo chybový stav odignorovať. Nepredpokladá sa, že by chybové stavy nastávali často, pretože volania procedúr nebudú nasadené vo vysoko konkurenčnom prostredí.

2.4.1 Evidencia subjektov

Ku známym subjektom v tabuľke `[Subject]` si môžeme uchovávať informácie ako

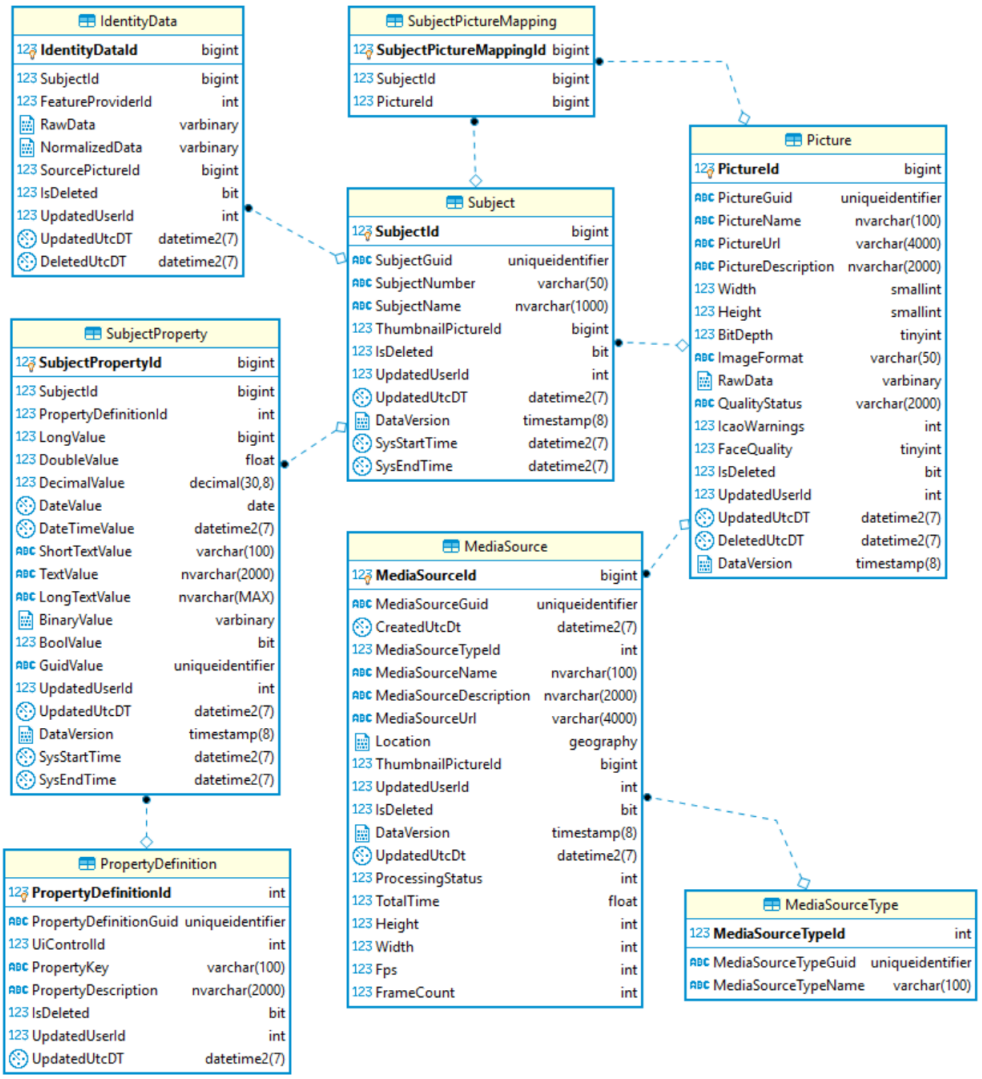
- embedding alebo identifikačné vektory, tabuľka `IdentityData`
- náhľadové a identifikačné fotografie, tabuľka `Picture`
- mená, názvy a poznámky
- dynamicky definovateľné vlastnosti subjektu, tabuľky `SubjectProperty` a `SubjectPropertyDefinition`

2.4.2 Evidencia zdrojov médií

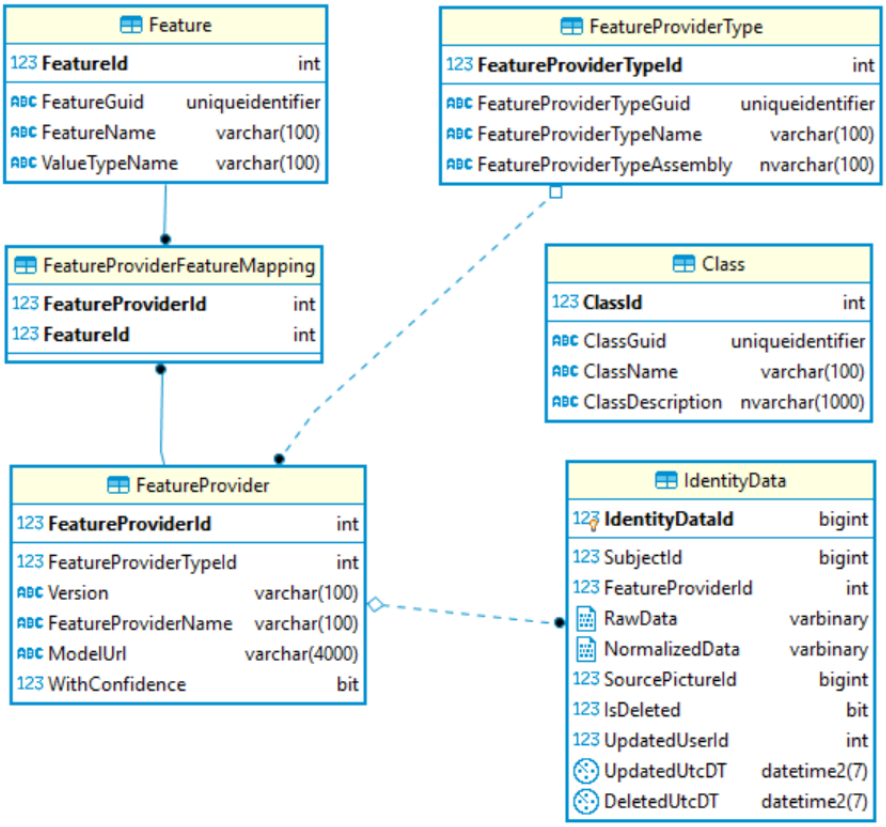
Evidencia zdrojov médií je zoznam všetkých dostupných obrázkov, video súborov alebo video streamov. V evidencii, tabuľke `MediaSource`, je najdôležitejšia dvojica hodnôt `MediaSourceId` a `MediaSourceUrl`, (identifikátor zdroja a adresa umiestnenia zdroja). Identifikátor je kľúčový v úložisku metadát. Tabuľka obsahuje fyzické vlastnosti mediálneho zdroja ako rozlíšenie bodov, dĺžka, počet snímkov za sekundu alebo geografická poloha. Mediálny zdroj môže mať priradený náhľadový obrázok.

2.4.3 Evidencia poskytovateľov metadát

Poskytovateľ metadát je z pravidla súčasťou analytického procesu a využíva definované rozhranie v podobe .NET knižníc. Evidencia poskytovateľov metadát je dôležitá pre korektný pohľad na generované a spracovávané metadáta



Obr. 2.1: Zjednodušený databázový diagram evidencie subjektov a zdrojov médií



Obr. 2.2: Databázový diagram evidencie poskytovateľov metadát

reprezentované ako rozhrania s dynamickým obsahom, ktoré sú popísané v sekcii 2.5, a podsekciách 2.5.3, 2.5.1 a 2.5.5. Evidencia obsahuje tieto tabuľky

`Class` — zoznam známych tried objektov

`FeatureProvider` — zoznam poskytovateľov metadát

`FeatureProviderType` — dátové typy a .NET knižnice rozhraní poskytovateľov metadát

`Feature` — zoznam poskytovaných vlastností a hodnôt

`FeatureProviderTypeMapping` — uchováva aké vlastnosti a hodnoty daný poskytovateľ poskytuje

2.5 Návrh rozhraní metadát

Schéma metadát je definovaná ako množina tried a rozhraní v jazyku C#. Jej podstatnú časť zobrazuje 2.3. Rozhrania sú podkladom pre generátor serializačného a deserializčného kódu v jazyku C# a Python. Teoreticky je možné na základe rozhraní vygenerovať kód pre akýkoľvek programovací jazyk. Generátor nie je predmetom záujmu záverečnej práce.

2.5.1 Rozhranie IClassification

`IClassification` reprezentuje klasifikáciu vizuálneho objektu alebo celého snímku. Pojem klasifikácia vnímame ako príslušnosť k vizuálnej triede. Klasifikácia sa priradzuje s percentuálnou mierou určitosti `Confidence` ako float v rozmedzí 0 až 1. Zoznam všetkých známych tried sa nachádza v evidenčnej databáze.

2.5.2 Rozhranie IPolygon

Polygón je ohraničený ako zoznam bodov, `IPoint`, absolútnych pozícií vzhľadom k pôvodnej veľkosti snímku. Od implementácie sa požaduje, aby programátor definoval metódy na získanie význačných bodov.

2.5.3 Rozhranie IImageMetadata

`IImageMetadata` reprezentuje popis jedného snímku alebo obrázku. Snímku ako celku môžu byť priradené viaceré klasifikácie a môže obsahovať viaceré detekované hodnoty a atribúty. Snímok môže obsahovať kolekciu vizuálnych objektov *IObject*.

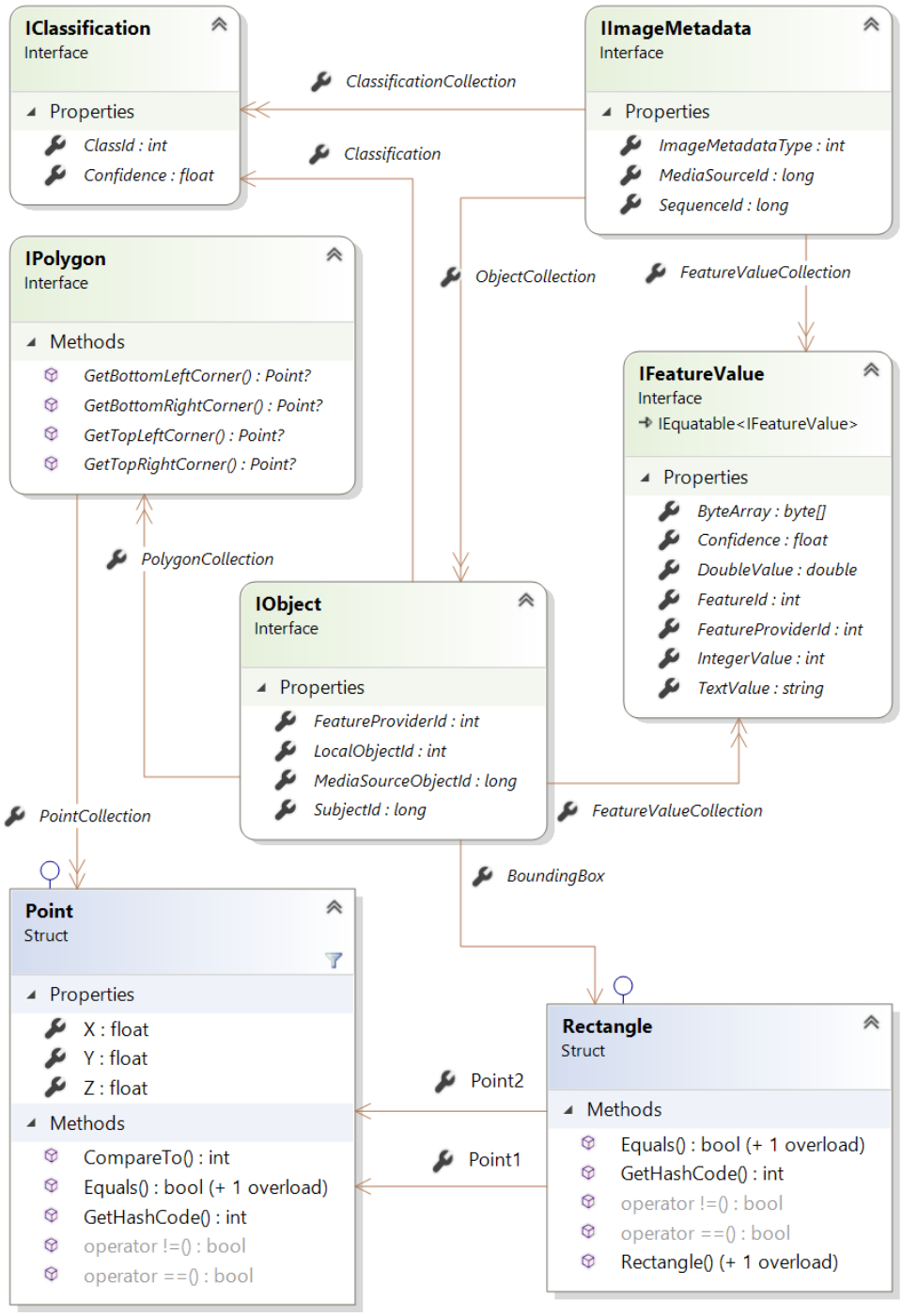
2.5.4 Rozhranie IObject

Popis jedného vizuálneho objektu, ktorý bol vygenerovaný konkrétnym poskytovateľom metadát identifikovaným hodnotou `FeatureProviderId`. Objekt má vygenerovanú klasifikáciu. Vždy obsahuje referenciu na zdroj média a sekvenciu v danom médiu. Môže už obsahovať referenciu na stotožnený subjekt v evidencii, `SubjectId`. Ďalšie dôležité hodnoty tvoria:

`LocalObjectId` — poradové číslo objektu v rámci snímku

`MediaSourceObjectId` — poradové číslo v rámci celého média

`PolygonCollection` — nepovinná segmentácia objektu ako kolekcia polygónov



Obr. 2.3: Diagram rozhraní a štruktúr IImageMetadata

2.5.5 Rozhranie IFeatureValue

`IFeatureValue` slúži na uchovávanie detekovaných hodnôt, atribútov rozmanitého charakteru. Detekovaná hodnota môže byť priradená s percentuálnou mierou určitosti. Identifikátor `FeatureProviderId` sa odkazuje na poskytovateľa metadát, ktorý hodnotu vytvoril. Poskytovateľ hodnotu ukladá buď ako celé číslo, číslo s pohyblivou rádovou čiarkou, textový reťazec alebo ako bajtové pole. Spôsob reprezentácie a význam hodnoty je možné zistiť v evidencii poskytovaľov metadát pomocou identifikátora `FeatureId` v tabuľke `Feature`.

2.5.5.1 Druhy hodnôt

Rozhranie umožňuje ukladať tieto druhy hodnôt:

`TextValue` — textová hodnota, napr. pre LPR

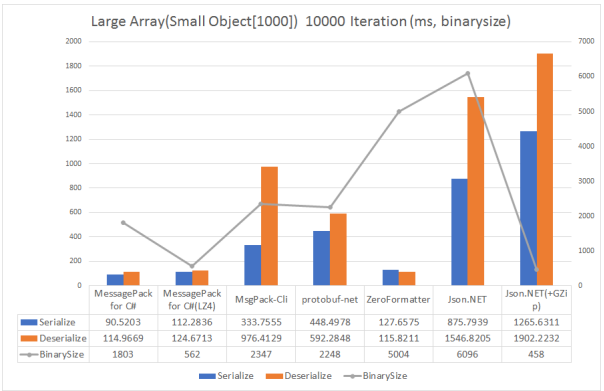
`DoubleValue` — číselná hodnota s pohyblivou rádovou čiarkou v dvojitej presnosti

`IntegerValue` — celočíselná hodnota, vhodná aj pre booleovskú hodnotu

`ByteArray` — bajtové pole, reprezentácia akýchkoľvek hodnôt ľubovoľnej dĺžky

2.6 Formát metadát

MessagePack je multiplatformový binárny serializačný formát podobný ako JSON, ale rýchlejší a kompaktnejší [4]. Pre *MessagePack* ako transportný a perzistetný formát sme sa rozhodli na základe porovnania implementácií iných serializačných formátov pre .NET [5]. V spolupráci s komprimovacím formátom L4Z môžme očakávať skvelý pomer medzi rýchlosťou a nárokmi na priestor, ako si môžeme všimnúť na grafickom porovnaní 2.4.



Obr. 2.4: Porovnanie serializačných formátov pre .NET, prevzaté z [5]

Kapitola 3

Analýza úložiska metadát

Táto kapitola sa zaoberá prieskumom, porovnaním a výberom vhodných technológií

3.1 Prieskum trhu

Prieskum sme realizovali na základe viacerých aspektov:

1. online prieskum kandidátov
2. konzultácie s kolegami
3. prieskum podobných riešení na trhu

3.2 Výber technológií

V tejto sekcii sa zaoberáme prehľadom databázových a komunikačných technológií vhodných pre úložisko metadát. Okrem funkčných a nefunkčných požiadaviek zo sekcie 2.1.1 sme sa pri výbere orientovali aj potrebou nasadenia riešenia offline prípadne na edge zariadeniach s menším výkonom.

Na základe skúseností z minulých projektov sme z výberu vylúčili technológie založené na *Java Virtual Machine* kvôli nekompatibilite s prostredím nasadenia.

3.2.1 Online prieskum kandidátov

Pri výbere kandidátov bola použitá webová stránka DB Engines a jeho rebríček databázových systémov[1].

[1] `https://db-engines.com/en/ranking`

3.2.1.1 Key-value úložiská

Key-value úložiská (KV) ukladajú dvojice, ktoré sa skladajú z jedinečného kľúča a hodnoty. Pre potreby úložiska metadát by kľúčom v KV bolo vo vhodnej forme reprezentované rozhranie `IFrameKey` a hodnotou `IImageMetadata` v binárnom formáte. Implementácie ako open source Redis[2] alebo komerčné Amazon DynamoDB[3] svoje dáta primárne ukladajú do operačnej pamäti, vďaka čomu majú nižšiu odozvu než bežné relačné databázy. Oba produkty majú vlastnosti vhodné na nasadenie v komerčnom prostredí ako zákaznícka podpora, škálovateľnosť a vysoká dostupnosť. V prospech DynamoDB hovorí aj extrémne nízka odozva a vysoký počet paralelných transakcií.

3.2.1.2 Wide-column databázy

Najpoužívanejším reprezentatom wide-column databáz je Apache Cassandra, s podporou dobrej škálovateľnosti, vysokej dostupnosti a korporátnej podpory. Podľa meraní uvedených v [6] na optimálny výkon je nutné Apache Cassandra spúšťať na desiatkach uzlov. Nemá ani podporu skriptovania na strane servera.

3.2.1.3 Časové série

Databázové riešenia nad časovými sériami (angl. *time series*) sú optimalizované na prácu s dátami, ktoré majú časovú známku (angl. *timestamp*) ako napríklad hodnoty z meraní zo senzorov alebo dáta z obchodovania na trhoch s vysokou frevenciou (angl. *high frequency stock trading*). Timeseries databázy zvládajú spracovávať vysoký počet transakcií. Populárnou open source implementáciou je InfluxDB, ktorá spĺňa spomínané kritéria. Kandidátom sa stalo aj rozšírenie do PostgreSQL TimescaleDB.

Na základe pozitívnych ohlasov od kolegov a článkov, ktoré porovnávajú výkonnosť spomínaných riešení s podporou časových sérii [7] a odôvodnením, prečo je SQL je vhodnejšia cesta než NoSQL [8], sme sa rozhodli úložisko metadát postaviť na PostgreSQL a TimescaleDB.

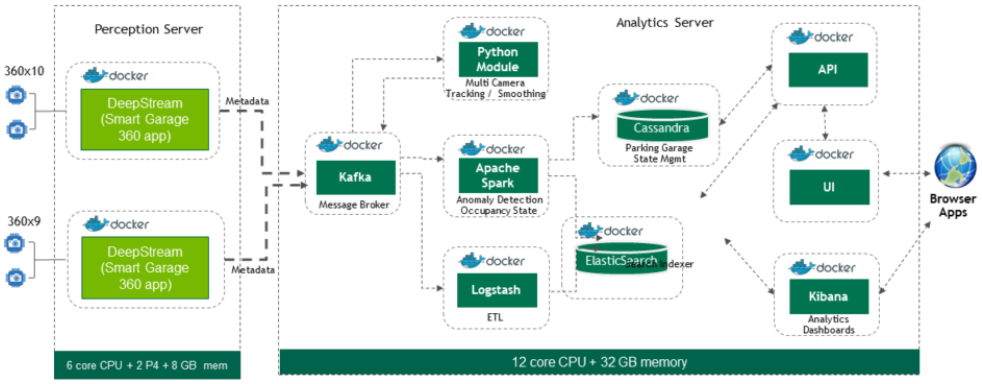
Do budúcna plánujeme zapojiť do riešenia message brokera (ako napr. Redis, RabbitMQ alebo ZeroMQ), ktorého úlohou bude koordinovať komunikáciu medzi oddelenými súčasťami systému ako napr. medzi klientom a serverom alebo medzi serverami navzájom.

3.2.2 Prieskum podobných riešení na trhu

Pozreli sme sa na podobné riešenia a systémy, ktoré obsahujú podporu ukladania a vyhľadávania metadát z analýzy videa. V oboch prípadoch sa jedná o omnoho komplexnejšie riešenia, než vyžaduje zadanie.

[2] `https://redis.io/`

[3] `https://aws.amazon.com/dynamodb/`



Obr. 3.1: Diagram IVA na NVIDIA DeepStream SDK.

3.2.2.1 Deep Video Analytics

Do prieskumu podobných riešení sa dostalo Deep Video Analytics, čo je rozsiahla platforma na ukladanie, analýzu a zdieľanie vizuálnych dát[4]. Platforma je zostavená z open source technológií ako PostgreSQL, Redis, RabbitMQ a Django. Prístup do systému je poskytovaný cez webové rozhranie.

3.2.2.2 NVIDIA DeepStream SDK

Obrázok 3.1, zachytáva referenčnú implementáciu inteligentnej analýzy videa. Obrázok je prevzatý z webu[5].

3.3 Dopytovanie

V tejto sekcii prezentujeme dôvody, prečo je vhodné používať uložené procedúry alebo funkcie. V ďalšom texte bude použitý zastrešujúci pojem *procedúry*.

3.3.1 Výkonnosť

Procedúry sú na tom z výkonnostného pohľadu oproti ORM lepšie, pretože vývojár má širšie možnosti, aby napísal rýchly, spoľahlivý a efektívny SQL príkaz. Príkaz si vývojár má možnosť profilovať, vyhodnotiť jeho exekučný plán. Na zefektívnenie plánu môže využiť triky, pretože pozná povahu dát, s ktorými pracuje. O ORM sa to s istotou povedať nedá. Dotaz na procedúru má v dobe, keď sa procedúra bude spúšťať, exekučný plán pripravený, takže databázový engine nemusí drahocenné milisekundy venovať jeho príprave. Pri adhoc dopytoch sa exekučný plán generuje vždy, ak sa práve

[4] `https://www.deepvideoanalytics.com/`
[5] `https://devblogs.nvidia.com/multi-camera-large-scale-iva-deepstream-sdk/`

nenachádza vo vyrovnávacej pamäti, cache. Cache je ale obmedzená, takže nepojme ľubovoľné množstvo plánov.

Na PostgreSQL serveri sme výkonnosť ovplyvnili aj absenciou cudzích kľúčov, referencií na iné tabuľky. Pri každom vložení a dátovej úprave tabuľky s cudzím kľúčom by sa kontrolovala referenčná integrita, čo môže v závislosti na kľúči spomaliť dopyt.

3.3.2 Bezpečnosť

Pokiaľ sa v procedúre nevykonávajú dynamické dopyty, tak volanie procedúr s parametrami úplne eliminuje výskyt SQL injection. Jedná sa o rozšírený spôsob útoky na systém, kedy volanie API so špecifickými hodnotami zapríčiní volanie dopytu, ktorý nebol pri realizácií systému plánovaný. Záškodnícke volanie môže spôsobiť únik alebo stratu citlivých informácií, nestabilitu a obmedzenie funkčnosti systému alebo dokonca sa útočník môže zmocniť systému. Eliminácia rizika spočíva v používaní parametrov pri volaní dopytov s premennými. Dotaz nikdy nesmie vzniknúť dynamicky či na strane klienta alebo servera. Volanie procedúr umožňuje aj prehľadnejšie a jednoduchšie riadenie úrovní používateľského prístupu k údajom.

3.4 GiST verzus vlastné indexovanie

Všeobecný vyhľadávací strom GiST (*Generalized Search Tree*), podľa prorockých slov jeho autora spred dvadsiatich rokov je stále hojne využívaný v PostgreSQL inštanciách.

> Pre akýkoľvek databázový systém sú rozhodujúce efektívne prístupové metódy nad vyhľadávacími stromami. Tradičné relačné manažment systémy si vystačia s B+ stromami nad štandardnými datovými typmi z SQL. Dnešné rozšíriteľné objektovo-relačné databázové manažment systémy (ORDBMS) sú nasadzované, aby podporovali aplikácie ako dynamické webové servery, geografické informačné systémy, CAD nástroje, knižnice multimédií a dokumentov, sekvenčné databázy, systémy na identifikáciu pomocou odtlačkov prstov, biochemické databázy atď. Nové typy prístupových metód sú nutné na tieto typy aplikácií. — Marcel Kornacker, High-Performance Generalized Search Trees, Proc. 24th Int'l Conf. on Very Large Data Bases, Edinburgh, Škótsko, september 1999.

PostgreSQL obsahuje rozšírenie *cube* [9] na prácu s poliami. Nad takýmto poľom sa dá vytvoriť GiST index. Z dôvodu maximálneho obmedzenia na 100 prvkov, `CUBE_MAX_DIM`[6], nebolo možné bez ďalšej implementácie medzivrstvy aplikovať GiST index. Napokon sme sa rozhodli odskúšať indexovanie v metrických systémoch, detaily v sekcii 6.5.2.

[6] `https://github.com/postgres/postgres/blob/master/contrib/cube/cubedata.h`

Kapitola 4

Návrh rozhraní API

Táto kapitola popisuje návrh aplikačného rozhrania pre úložisko metadát v troch programovacích jazykoch C#, Python a C++. Každá varianta v spomínaných jazykoch sa skladá z troch základných celkov:

Manažment spojení — konfigurácia a správa spojení s databázou

Vkladanie — vkladanie alebo aktualizácia záznamov

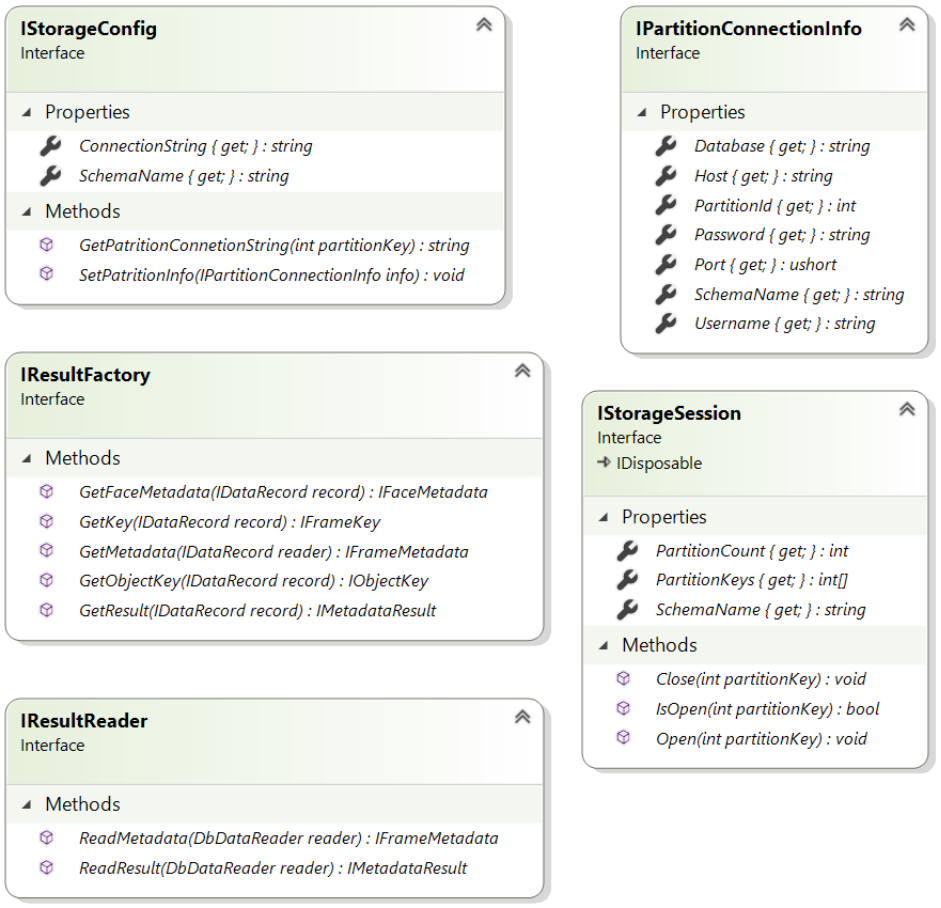
Dopytovanie — možnosti vyhľadávania a dopytovania sa na existujúce záznamy

Vnútorná reprezentácia času vo frameworkoch a databáze nie je rovnaká. Pri prenose medzi frameworkom a databázou alebo medzi frameworkami navzájom môže dôjsť k drobným odchýlkam, popr. strate presnosti. Riešenie tejto nepresnosti sme prenechali na používateľa API s odporúčaním, aby pri ukladaní metadát s časovou informáciou nastavoval aj `FrameId` tak, aby hodnota v čase bola monotónne rastúca. Túto nepresnosť nie je možné vyriešiť na strane servera.

Rôzni klienti (napr. kamery) nemusia mať zosynchronizovaný čas. Rôzne súčasti analytického procesu pracujú s rôznymi časmi podla toho, či klienti poskytujú alebo neposkytujú časovú informáciu (napr. timestamp vo videu). V prípade, že poskytujú, narážame na problém uvedený vyššie. V prípade, že neposkytujú, analytický proces nemusí pracovať so správnou časovou značkou, protože čas medzi vznikom snímku a jeho spracovaním sa môže nedeterministicky líšiť.

4.1 .NET Core

Rozhrania sú definované v .NET Standard 2.0 knižnici, ktorá je podporovaná .NET Core 2.2, čo je v súlade s NF2 v sekcii 2.1.2.2.



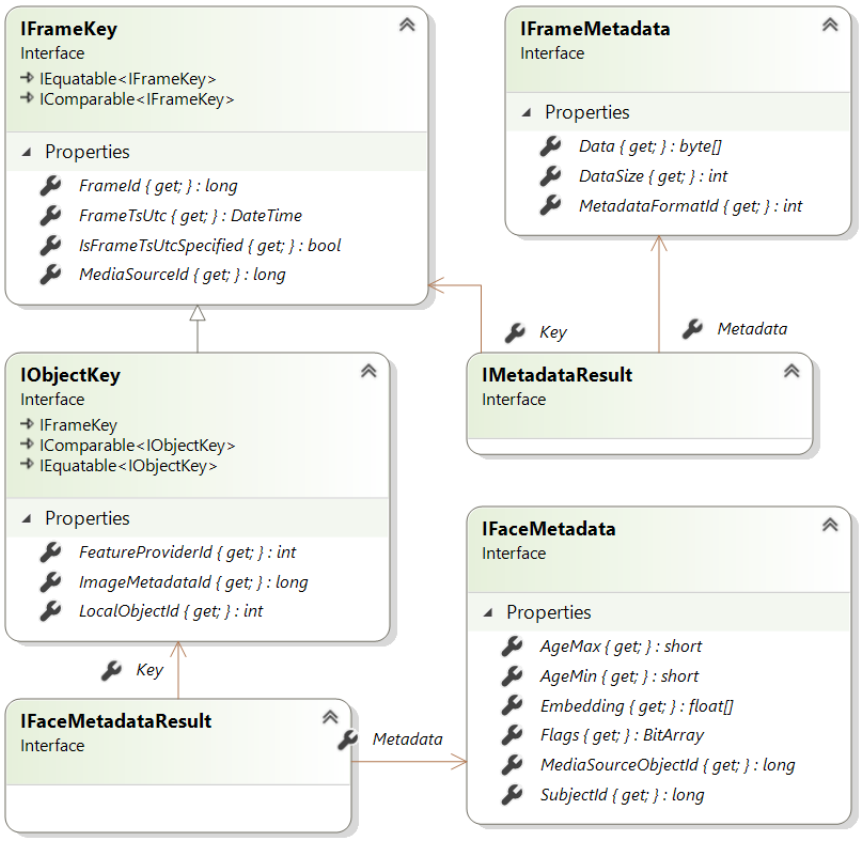
Obr. 4.1: Diagram rozhraní a nastavení rozhraní API.

4.1.1 Manažment spojení

Navrhované riešenie ponúka rozhrania na uchovávanie nastavení pre každé separátne spojenie `IPartitionConnectionInfo` a rozhranie pre správu sedenia `IStorageSession`.

4.1.2 Dopytovanie

Aktívne sedenie použijeme na vytvorenie dopytovacieho objektu. Rozhranie `IMetadataReader` umožňuje získavať dáta konkréntneho snímku alebo vyhľadávať podľa dopytovacieho parametra typu`IMetadataQuery`. Rozhranie `IFaceReader` budú implementovať triedy, ktoré sprístupňujú vyhľadávanie objektov typu ľudská tvár.



Obr. 4.2: Diagram datových rozhraní API.

4.1.3 Vkladanie

Rozhranie `IMetadataWriter` definuje metódy na vkladanie alebo vkladanie s prepisovaním.

4.2 Python

Jazyk Python a knižnice založené na tomto jazyku sú široko používané v komunite vývojarov strojového učenia (ML) a umelej inteligenicie (AI). V spoločnosti Quantasoft sa používa na vývoj prototypov a testovanie, preto nesmie chýbať podpora vkladania a dopytovania sa na výsledky z analýzy videa.

Napriek faktu, že Python je dynamický skriptovací jazyk, tak pomocou anotácií umoňuje vytvárať abstraktné triedy. Vkladanie aj dopytovanie majú na starosti triedy, ktoré dedia od abstraktnej triedy `MetadataStore`. Tieto triedy musia implementovať nasledujúce metódy:

`insert` — vkladanie, pokus o vloženie duplikátneho záznamu musí končiť

Obr. 4.3: Diagram dopytovacích rozhraní API

chybou

`set` — vkladanie, pokus o vloženie duplikátneho záznamu prepíše originál

`get` — dopyt na jeden konkrétny snímok

`select` — dopyt na rozsah snímkov

4.3 C++14

Skompilovaný kód z jazyka C++ má najlepšie predpoklady, že bude bežať čo najrýchlejšie pre akúkoľvek procesorovú architektúru vďaka prekladu priamo do strojového kódu s možnosťou optimalizácie kódu na konkrétnu inštrukčnú sadu. Oproti jazykom C# a Python má výhodu, že reverse-engineeringom nie je triviálne získať pôvodný zdrojový kód.

4.3.1 Vkladanie

V dobe písania tohoto textu rozhranie podporuje iba zápis metadát, pretože sa očakáva jeho využitie v spolupáci s DeepStream SDK, kde je predmetom

záujmu iba vkladanie vygenerovaných metadát.

4.3.2 Manažment spojení

Manažment spojení je riešený konfiguračne. Na koncových zariadeniach ako NVIDIA Jetson Xavier alebo Jetson Nano jedna škatuľka zvládne jednotky vstupných video streamov, ktorých počet sa v čase nebude dynamicky meniť.

Kapitola 5

Implementácia API k metadátam

Táto kapitola popisuje implementáciu aplikačného rozhrania pre prístup k metadátam, ktorá je používateľovi sprostredkovaná vo forme knižníc. Čitateľ si bude môcť porovnať, ako vyzerá zápis operácie vkladania metadát do navrhovaného úložiska v jazykoch C#, Python a C++.

5.1 .NET Core

Projekty pre Visual Studio 2017 týkajúce sa API k metadátam produkujú interoperabilné knižnice typu .NET Standard 2.0. Ako programovací jazyk je zvolený C#. Je to primárny jazyk implementácií programovateľných riešení v spoločnosti Quantasoft.

```
1  string connectionString =
2  "user=user; password=pass; host=ip_or_host; dbname=db";
3  var key = new c.FrameKey(2,1,DateTime.UtcNow);
4  var metadata = new c.FrameMetadata(1, new byte[]{0});
5  var config = new tsdb.StorageConfig(connectionString);
6  var factory = new tsdb.MetadataFactory();
7  using (var sesion = new tsdb.Session(config))
8  using (var writer = new tsdb.MetadataWriter(sesion))
9  {
10     writer.Insert(key, metadata); // z kamery
11     writer.Insert(new c.FrameKey(1,1), metadata);
12 }
```

Listing 5.1: Ukážka vkladania metadát pomocou API pre .NET Core

5.1.1 Projekt Quantasoft.Iva.MetadataStorage

Obsahuje výhradne dátové, komunikačné a konfiguračné rozhrania. Jedná sa o abstraktnú vrstvu úložiska nezávislej na komunikačných protokoloch a ani na výbere databázy.

5.1.2 Projekt Quantasoft.Iva.MetadataStorage.Common

Implementácia spoločných tried úložiska metadát nezávislých na realizácii databázy.

5.1.3 Projekt Quantasoft.Iva.MetadataStorage.TimescaleDb

Implementácia projektu je zavislá na Npgsql knižnici a realizácii na PostgreSQL databáze. V knižnici je implementovaná podpora prerozdeľovania spojení na konkrétnu inštanciu podľa `MediaSourceId`, čoho výsledkom je rozloženie dát a záťaže na samostatné uzly. Pri inicializácii spojenia sa databázove príkazy predpripravia, čo zaručí, že volané funkcie v databáze skutočne existujú a samotné volania budú o čosi rýchlejšie spracované. Parametre príkazov sa takisto pripravia dopredu, čo zníži potrebu alokácie objektov v halde. Nevýhodou je nutnosť udržiavať objekty pre každé spojenie separátne.

Pri inicializácii sedenia sa stiahne zoznam inštancií a zoznam referencií, ku ktorej inštancii zdroje médií patria. Dopyty so známym `MediaSourceId` a vkladanie metadát prebehne na jednom spojení. Ostatné dopyty sa odošlú naprieč všetkými inštanciami, zozbierajú sa výsledky a pospájajú sa do jedného celku.

5.2 Python

Implementácia v jazyku Python je závislá na knižniciach:

asyncpg — realizuje asynchrónne volania procedúr

psycopg2 — realizuje synchrónne volania procedúr

```
1 import iva_metastore.pgsql.pgsqlmeta as pg
2 import iva_metastore.pgsql.connection as c
3 import iva_metastore.metastore as m
4 from datetime import datetime
5
6 config = pg.PgsqlMetaStoreConfig(
7   password='pass', username= 'user',
8   host='ip_or_host', database='dbname', execute_sp=True)
9 store = pg.PgsqlMetaStore(config)
10 meta = m.Metadata(b'0', 1)
```

```
key = m.FrameKey(2, frameId=1, utc=datetime.utcnow())
store.insert(m.FrameKey(1, frameId=1), meta)
store.insert(key, meta)
```

Listing 5.2: Ukážka vkladania metadát v jazyku Python

5.2.1 Modul iva_metastore.pgsql.asyncpgsqlmeta

Python modul `asyncpgsqlmeta` implementuje abstratné funkcie pomocou knižnice asyncpg.

5.2.2 Modul iva_metastore.pgsql.pgsqlmeta

Python modul `pgsqlmeta` implementuje abstratné funkcie pomocou knižnice psycopg2.

5.3 C++

Implementácia v C++ s databázou komunikuje cez knižicu libpqxx[7], ktorá v dobe implementácie patrila k najpoužívanejším z voľne dostupných komunikačných knižníc s PostgreSQL.

```
#include <qsmeta_pqxx.hpp>
#include <chrono>
#include <cstring>
using namespace std::chrono;
using namespace qsmeta;
milliseconds utcnow(){
return duration_cast<milliseconds>(
system_clock::now().time_since_epoch()
);}

auto conn ="user=usr password=pass host=ip dbname=db";
const char* data = "0";
auto meta = FrameMetadata(4, data, strlen(data));
PqxxStoreSession session(conn);
session.open();
session.insert(FrameKey(1, 1), meta);
session.insert(FrameKey(utcnow().count(), 2, 1), meta);
session.close();
```

Listing 5.3: Ukážka vkladania metadát v C++

[7] http://pqxx.org/development/libpqxx/

Kapitola 6

Návrh a implementácia úložiska metadát

Táto kapitola sa bude zaoberať databázovým strojom PostgreSQL, jeho rozšírením TimescaleDB, návrhom klastra, popisom tabuliek a používateľských funkcií a typov.

6.1 Databáza PostgreSQL

Po zvolení PostgreSQL ako úložiška metadát sme zvažovali a overovali verzie:

9.6 na operačnom systéme Ubuntu 16.04

10.4 na operačnom systéme Ubuntu 16.04 aj Windows Server 2016

11.3 na operačnom systéme Ubuntu 18.04 aj Windows Server 2019

Nakoniec sme sa rozhodli pre verziu 11.3, ako najnovšiu stabilnú verziu v dobe písania tohoto textu, z dôvodu najlepšej podpory lokálnych paralelných dopytov a najširšej palety funkcionality[8].

6.1.1 TimescaleDB

Na všetkých inštanciách je nainštalovaná verzia 1.3. Na každej z nich je aplikovaná optimalizácia nastavení databázy podľa aktuálnej hardvérovej konfigurácie pomocou príkazu `timescledb-tune`[9].

[8] `https://www.postgresql.org/about/featurematrix/`

[9] `https://blog.timescale.com/better-database-performance-using-timescaledb-tune-fbd7ae7016fa/`

6.1.1.1 Hypertabuľky

Hypertabuľka (*hypertable*) je primárnym bodom interakcie nad dátami z časových postupností v TimescaleDB. V úložisku metadát sú vytvorené dve `ImageMetadataUtc` detaily v podsekcii 6.3.1 a `FaceMetadataUtc` detaily v podsekcii 6.3.3. Volanie funkcie `create_hypertable` je jednou z možností, ako ju vytvoriť. Funkcia sa volá s parametrami názov tabuľky a

`FrameTsUtc` — časový kľúč

`MediaSourceId` — doplňujúci deliaci kľúč (partitioning key).

6.2 Klaster

Klaster pozostáva z dvoch párov serverov. Jeden pár je nasadený na operačnom systéme Ubuntu 18.04, druhý pár na Windows Server 2019 Standard. Kluster je konfigurovaný tak, že každá inštancia dokáže fungovať samostatne bez ohľadu na stav ostatných inštancií.

Funkčnú požiadavku je možné naplniť tak, že diskový priestor sa bude zväčšovať buď rozširovaním partície alebo pridávaním samostatných diskov. Pridávanie samostatných diskov bude znamenať, že databázu bude nutné rozšíriť o nový tabuľkový priestor [10] a nastaviť rozdeľovanie dát (*partitioning*) [11]. Zároveň sa môže zlepšiť výkonnosť dopytovania.

6.2.1 Rozdeľovanie dát

Rozdeľovanie dát (*partitioning*) je implementované formou *share nothing* riešenia, kedy každá inštancia úložiska obsahuje snímky z disjunktnej podmnožiny zdrojov médií. Najväčšou výhodou je možnosť spúšťať na nezávislých dátach pararalelné dopyty. Výhodou je tiež nižšia náročnosť zostavovania konzistentných dopytov a všetky inštancie majú spoločnú rovnakú sadu dopytov. Nevýhoda je v konfiguračnej náročnosti a vyššej zložitosti implementácie API na dotazovacie funkcie. Na klientskej strane sa musia zlučovať výsledky a uchováť spojenia so všetkými inštanciami, čo má za následok rastúcu pamäťovú náročnosť s rastúcim počtom inštancii. Dôraz bol ale kladený na rýchlosť dotazov. Detaily implementácie rozdeľovania dát v API sa nachádzajú v sekcii 5.1.3.

6.2.2 Vysoká dostupnosť

Podpora vysokej dostupnosti (HA) momentálne nie je implementovaná, avšak pri návrhu sme počítali s budúcim rozšírením. Vďaka širokej palete možností, ktoré ponúka PostgreSQL, ako zabezpečiť zotavenie sa z výpadku, vieme pokryť podstatu problému. Možnosti HA začínajú pri použití spoločného disku prípadne replikácie súborového systému, pokračujú doručovaním transakčných

denníkov, logickou replikáciou, preposielaním dopytov, spracovaním na základe spúšťačov (*trigger-based*) a končiac v asynchronnej replikácii [12].

Uvažujeme, že každá inštancia bude mať svojho dvojníka v multimaster zapojení, čo by mohlo viesť k zrýchleniu vyhľadavacích dopytov.

6.3 Tabuľky

V tejto sekcií budú popísané najsignifikantnejšie tabuľky z úložiska metadát. Graficky sú zobrazené na diagrame na obrázku 6.1. Väzby znázornené bodkovanou čiarou sú imaginárnymi odkazmi, pretože kvôli rýchlosti vkladania metadát neboli implementované.

6.3.1 Tabuľka ImageMetadataUtc

Tabuľka `ImageMetadataUtc` je určená na ukladanie metadát zo streamov videa alebo mediálnych zdrojov, ktoré majú definovaný čas vzniku snímku. Čas vzniku sa ukladá do stĺpca `FrameTsUtc`. Na základe tohto stĺpca primárny kľúč hypertabuľka nemá. Okrem toho má identifikátor metadát platný pre inštanciu PostgreSQL servera, stĺpec `ImageMetadataId`, ktorého hodnota sa získava zo sekvencie `ImageMetadata_ImageMetadataId_seq`.

6.3.2 Tabuľka ImageMetadataFi

Tabuľka je určená na ukladanie metadát z video súborov alebo mediálnych zdrojov, kde je jednoznačne možné určiť poradové číslo snímku. Primárny kľúč tabuľky pozostáva zo stĺpcov `MediaSourceId` a `FrameId`. `ImageMetadataId`, stĺpec, získaný zo sekvencie `ImageMetadata_ImageMetadataId_seq` a tým pádom máme vytvorený spoločný identifikátor metadát na jednej inštancii úložiska metadát.

6.3.3 Tabuľka FaceMetadataUtc

Do tabuľky `FaceMetadataUtc` sa ukladajú rozbalené metadáta popisujúce tváre subjektov z tabuľky `ImageMetadataUtc` . Primárny kľúč kvôli rozšíreniu TimescaleDB tabuľka nemá. Na detekciu duplikátov sa používa jedinečný index `ix_facemetadatautc_frame`, ktorý pozostáva zo stĺpcov:

`MediaSourceId` — identifikátor mediálneho zdroja

`FrameTsUtc` — časová známka snímku

`LocalObjectId` — poradové číslo objektu v rámci snímku

`FeatureProviderId` — identifkátor poskytovateľa metadát

6.3.4 Tabuľka FaceMetadataFi

Do tabuľky sa ukladajú rozbalené metadáta popisujúce tváre subjektov z tabuľky `ImageMetadataFi` . Primárny kľúč pozostáva zo stĺpcov:

`ImageMetadataId` — identifikátor z tabuľky `ImageMetadataFi`

`LocalObjectId` — poradové číslo objektu v rámci snímku

`FeatureProviderId` — identifkátor poskytovateľa metadát

Obe tabuľky s prefixom `FaceMetadata` majú spoločný identifikačný vektor subjektu `Embedding`, atribúty a vlastnosti `AgeMin AgeMax` a `Flags`. Atribúty a vlastnosti sa v súčasnej dobe dátami nenaplňujú, ale sú pripravené na vyhľadávanie podľa popisu.

6.3.5 Objektové tabuľky

Kvôli podpore vyhľadávania rôznorodých vizuálnych objektov bez rozbaľovania ImageMetadata bude musieť vzniknúť dvojica objektových tabuliek pre streamy a video súbory. Tie by uchovávali rozbalený obsah metadát z každého snímku, kde by jeden záznam okupoval jeden objekt z konkrétneho snímku. Ak by spoločné vlastnosti objektov obsiahnuté v objektových tabuľkách neboli postačujúce na popis a vyhľadávanie, pre každú triedu zo špecifickými vlastnosťami a atribútmi by musela vzniknúť separátna dvojica tabuliek, podobne ako majú tváre `FaceMetadata`. Pre každú dvojicu by museli vzniknúť prístupové funkcie a samostatné rozhrania v API. Zložitosť riešenia by sa dala znížiť automatickým genrátorom zdrojového kódu, ktorý by vygeneroval funkcie aj API na základe schémy tabuliek.

6.4 Kompozitné typy

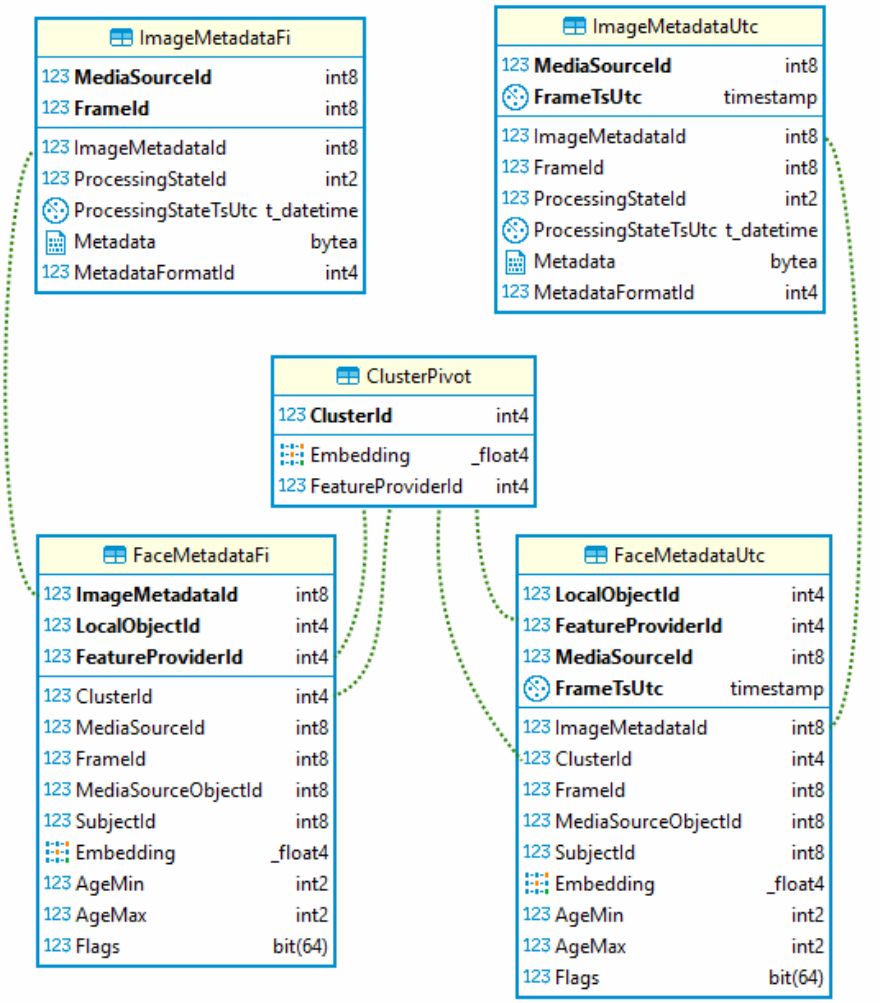
Používateľom definované kompozitné typy [13] (anglicky *composite types*) sa používajú na definíciu výstupnej množiny dotazovacích funkci alebo ako dátový typ vstupného parametra funkcií. V úložisku metadát sú definované nasledujúce typy:

`face_result` — výsledok dopytu na tváre z image metadata

`framekey_result` — výsledok dopytu na identifikátor snímku

`metadata_result` — výsledok dopytu na metadáta zo snímkov

`partitionconnection_result` — výsledok dopytu na zostavenie pripojovacieho reťazca do databázy



Obr. 6.1: Databázový diagram metadát v PostgreSQL

6.5 Vyhľadávanie objektov

V tejto sekcii sa budeme venovať teoretickým princípom, ktoré súvisia s vyhľadávaním v metrických systémoch. Z pohľadu spracovania dát z výstupu analytických procesov je embedding kľúčovou hodnotou pri identifikácii objektov, pričom embedding je prvkom metrického systému.

6.5.1 Embedding ako zložka metrického prietoru

Definícia 6.5.1 *Vektorový priestor* $V \subset R^d$, *kde* $d = 512$.

d je empiricky zvolená hodnota na základe nepublikovaných experimentov pracovníkov z Quantasoftu na datasete VGG Face 2 [14] pomocou internej

Lecko metriky 6.6.1. V literature sa tiež udávaju hodnoty d 64, 128 alebo 256 [15].

Tabuľka 6.1 porovnáva priemerný čas výpočtov vzdialenosti Lecko metriky s kosínusovou, euklidovskou a štvorcovou euklidovskou.

Embedding, identifikátor tváre $v \in V$, nad V je definovaný metrický priestor D, kde každá zložka je uložená ako 32-bitové číslo s pohyblivou rádovou čiarkou, *single*. Jeden vektor zaberá presne 2 KiB. V operačnej pamäti vektor reprezentujeme ako 512 prvkové pole.

6.5.2 MSQL clustering

Pri budovaní rýchlejšieho vyhľadávania na množine V v PostgreSQL databáze sme sa inšpirovali prácou [16]. Nad testovacou dátovou množinou sme vybudovali klustering, formu indexácie.

Množina V bola rozdelená na podmnožiny $V_x, x \in \{1, \cdots, p\}$, kde v každej V_x sme zvolil reprezentanta, *pivota*, vektor v ideálnom prípade s centralizovanou pozíciou v rámci podmnožiny. Reprezentantov ukladáme do samostatnej tabuľky `ClusterPivot` s umelým primárnym kľúčom `ClusterId` typu integer. Centralizovaná poloha pivota je výhodná kvôli priblíženiu sa ku rovnomernému rozdeleniu príslušníkov V_x.

Vektory ukladáme do tabuliek `FaceMetadataFi` alebo `FaceMetadataUtc` v závisloti na type videa. S vektorom ukladáme aj jeho príslušnosť k podmnožine. Príslušnosť odvodzujeme od vzdialenosti od pivota. Čím je vzdialenosť menšia, tým sú vektory podobnejšie a aj príslušnosť jednoznačnejšia. Nakoniec si zvolíme `ClusterId`, pivota s minimálnou vzdialenosťou a identifikátor uložíme do tabuľky.

Pri dopytovaní sa na k-najbližších susedov opäť prechádzame všetkých reprezentantov a vyberáme si najpodobnejšiu podmnožinu V_x, odkiaľ vyberieme k-najbližších susedov, ale nedopytujeme sa už na ostatné podmnožiny, čím ušetríme operácie porovnávania. Predpokladáme, že $k \ll \| V_x \|$, čím minimalizujeme rizko, že niektorí z k-najbližších budú patriť do inej podmnožiny.

Optimalizáciu na dopyty, keď vyhľadávame vektory v maximálnej vzdialenosti, sme ponechali na budúcu implementáciu.

6.5.3 Voľba pivotov

Voľba navýhodnejšej množiny pivotov je NP-úplná úloha. Z dôvodu výpočtovej náročnosti sme zvolili nasledujúci heuristický algoritmus voľby pivotov z testovacích dát:

1. Z každého mediálneho zdroja sme zvolili náhodnú vzorku 100 kandidátov.

2. Medzi všetkými dvojicami zo vzoriek sme vypočítali vzdialenosť pomocou funkcie `distance_lecko`, ktorá je popísaná v sekcii 6.6.1.

3. Z dvojíc s čo najväčšími vzájomnými vzdialenosťami sme náhodne vybrali kandidátov na pivota.

4. Porovnávali sme vzdialenosti medzi všetkými kandidátmi. Z výberu sme vyradili tých kandidátov, ktorí boli od ľubovoľného iného kandidáta vo vzdialenosti menšej ako 0,7.

Na konci voľby nám zostalo 10 kandidátov, ktorích sme zvolili za pivotov. V prílohe v súbore compare_embeddings.ods je možné nájsť podkladové dáta. Pre zaujímavosť, pri voľbe sme použli 32 000 jedinečných párov.

V produkčnom prostredí by výber pivota musel vyzerať odlišne. Proces nemôže byť jednorázového charakteru a nemôže byť manuálny vzhľadom na neustále pribúdajúce dáta. Musel by vzniknúť automatický algoritmus, ktorý by bežal opakovane.

6.6 Používateľsky definované funkcie

Používateľsky definované funkcie a procedúry (UDF) je skriptovateľné rozšírenie logiky databázy. Funkcie je možné definovať ako čisté SQL dopyty [17], v procedurálnom jazyku PL/PgSQL [18], ako volania natívnych knižníc [19], prípadne ako skripty v jazykoch Python, Perl, Lua a iných [20]. Z dôvodu optimalizácie dopytov sa odporúča deklarovať funkcie s vhodným a čo najstriktnejším označením volatility [21]. Funkcie je možné preťažovať [22], čo v implementácii využívame.

Úložisko metadát sprostredkováva funkcie ako jediný komunikačný kanál pre API, čo znamená, že API smie prístupovať do databázy iba prostredníctvom funkcií či procedúr.

6.6.1 Vzdialenostné funkcie

Vzdialenostné funkcie sú kľúčové pri porovnávaní float vektorov, ako je popísané v sekcii 6.5. Pri hľadaní existujúcich rozšírení do PostgreSQL, ktoré by podporovali výpočet vzdialenosti medzi float vektormi, sme narazili na niekoľko problémov. Buď mali podporu iba textových stĺpcov v module pg_similarity [23] alebo polí s nedostatočnou dĺžkou v module cube. Jediným riešením bolo definovať si vlastné funkcie. Rozhodli sme sa použiť populárnu implementáciu z Python knižnice *scipy* a jej module na výpočty vzdialeností [24].

Definícia 6.6.1 *Majme definovaný skalárny súčin nad vektormi z* R^d*, potom* $u \cdot v = \sum_{i=1}^{d} u_i v_i$ *a euklidovskou normu* $||u||_2 = \sqrt{u \cdot u}$

V úložisku metadát sú implementované nasledovné funkcie:

distance_cosine — kosínusová alebo uhlová vzdialenosť $1 - \frac{u \cdot v}{||u||_2 ||v||_2}$

distance_euclidean — euklidovská vzdialenosť $\sqrt{(u-v)\cdot(u-v)}$

distance_sqeuclidean — kvadrát euklidovskej vzdialenosti $(u-v)\cdot(u-v)$

distance_lecko — interná metrika založená na skalárnom súčine $1-u\cdot v$, za predpokladu normovaných vektorov u, v

Tabuľka 6.1 porovnáva priemerné časy funkcií potrebných na výpočet vzdialenosti prepočítané na porovnanie jednej dvojice. Merania prebehli na inštancii **O1**, popis v sekcii 7.1.

lecko	euclidean	cosine	sqeuclidean
0.157082	0.162036	0.175355	0.177590

Tabuľka 6.1: Porovnanie priemerných časov funkcií na výpočet vzdialenosti

6.6.2 Funkcie na vkladanie dát

Funkcie na vkladanie dát sú implementované dvojakého typu. Funkcie, ktorých názov začína prefixom `sp_insert` vkladajú dáta, ale v prípade kolízie kľúča či indexu volanie skončí chybou. Funkcie s prefixom `sp_upsert` v názve v prípade kolízie dáta prepíšu. Na rozdiel od evidencie metadát, sa z výkonnostných dôvodov kontrola paralelného prístupu nevykonáva. O validitu záznamov sa musí postarať analytický proces sám.

6.6.3 Dotazovacie funkcie

V subsekciách budú popísané funkcie určené k dotazovaniu sa na metadáta.

6.6.3.1 Náhodný prístup

Funkcie na dotazovanie s podporou náhodného prístupu sú implementované separátne pre nasledujúce funkcie:

`sp_get_metadata` — množina preťažených funkcií na vyhľadanie jedného snímku

`sp_getrange_metadata` — množina preťažených funkcií na vyhľadanie rozsahu snímkov

6.6.3.2 Dopytovanie podľa predlohy

Dopytovanie podľa predlohy je implementované na vyhľadanie tvárí cez funkciu `sp_find_faces`. Jedná sa o variantu algoritmu vyhľadávania k-najbližších susedov. Parameter `embedding` je predloha, ktorá vznikla z obrázka vyhľadávanej tváre. Parameter `kcount` limituje maximálny počet vrátených výsledkov.

Kapitola 7

Testovanie výkonu

Výkonnosť riešenia sme overovali testami na rozličných inštanciách, aby sme potvrdili splnenie požiadaviek V1, V2, V3 a V4 zo sekcie 2.1.1.

7.1 Testovacie inštancie

Merania prebiehali na štyroch typoch inštancií, ktoré majú tieto spoločné vlastnosti:

- Virtuálny stroj hostovaný na Hyper-V s rezervovaným výkonom na 90%.
- Procesor Intel Xeon Gold 5120 @ 2,2 GHz
- Dátový disk so 100 GiB na SSD oddelený od systémového
- Operačný systém Ubuntu 18.04 LTS

Popis jednotlivých inštancií:

S1 — 1 vCPU s 8 GiB RAM

S2 — 1 vCPU s 4 GiB RAM a obmedzenín na 50% výkonu procesora

O1 — 8 vCPU s 48 GiB RAM

C1 — ako O1, ale zapojená vo dvojici

Testy boli spúšťané z virtuálneho stroja na Windows Server 2019 zapojeného v rovnakej sieti, ale z iného hostiteľského stroja.

7.2 Testovacie dáta a skripty

7.2.1 Testovacie dáta z kamier

Dáta na testovanie výkonu boli vygenerované z nahrávok z video kamier a mediálnych zdrojov. Jedná sa o video záznam v H.264 kódovaní, kde veľkosť obrazu je FullHD s rôznym *fps*. Nahrávky zaznamenávajú pohyb ľudí, kde sú zreteľné aj ich tváre. Tabuľka 7.1 obsahuje štatistiky mediálnych zdrojov. Zo súborov boli vygenerované dáta do tabuľky `ImageMetadataFi` a hypertabuľky `ImageMetadataUtc`. V hypertabuľke je hodnota `MediaSourceId` navýšená o 1000 než udáva tabuľka 7.1. Počiatočná hodnota `FrameTsUtc` je *2019-06-01 10:12:15*. Každý ďalší naimportovaný snímok bude niesť hodnotu oproti predošlému navýšenú o prírastok z tabuľky 7.1. Súbory s metadátami sú uložené ako sekvencie ImageMetadata vo formáte MessagePack a nachádzajú sa v prílohe B.

MediaSourceId	**#snímkov**	**#tvárí**	**prírastok [ms]**	**[MiB]**
224	4500	75300	200,0	53,0
225	1892	21186	200,4	17,1
226	10756	90552	40,0	105,0
230	667	4767	33,3	3,4
231	5481	31822	190,0	34,3
232	135	424	100,0	0,5
233	161	6002	180,0	6,2
234	201	7026	180,8	7,3
Spolu:	**23793**	**237079**		**226,8**

Tabuľka 7.1: Zoznam mediálnych zdrojov

7.2.2 Testovací skript

Testovanie výkonu je implementované v .NET Core 2.2 ako volania API úložiska metadát. Testovanie je riadené konfiguračným súborom a parametrami príkazozového riadku. Vhodnou kombináciou parametrov a konfigurácie je možné spúšťať testovacie operácie na ľubovoľnej inštancii s nastaviteľnou úrovňou simulovaného paralelného prístupu do databázy. Simulovaný paralelizmus je implementovaný ako spúšťanie rovnakých dopytov na samostatných vláknach, separátnych spojeniach s úložiskom a riadený bariérovou synchronizáciou. V grafoch a tabuľkách v tejto kapitole sú testovacie prípady označené ako:

p1 — jedno vlákno

p2 — dve vlákna

p3 — tri vlákna.

Testovací skript Run-Tests2.ps1 napísaný v jazyku PowerShell, ktorý spúšťa testovacie prípady, sa nachádza v prílohe.

7.3 Vkladanie

Vkladanie záznamov je špecifické v tom, že rýchlosť zápisu sa nemusí výrazne líšiť na testovacích inštanciách s rozličným počtom jadier CPU v prípade použitia jedného databázového spojenia. Jedno databázové spojenie nemusí naplno vyťažiť výpočtové zdroje na inštancii. Z tohto dôvodu je potrebné spúšťať paralelné merania na jednej inštancii.

Na grafe 7.1 a v tabuľke 7.2 pre `ImageMetadataFi` a na grafe 7.2 a v tabuľke 7.3 pre `ImageMetadataUtc` je vidieť:

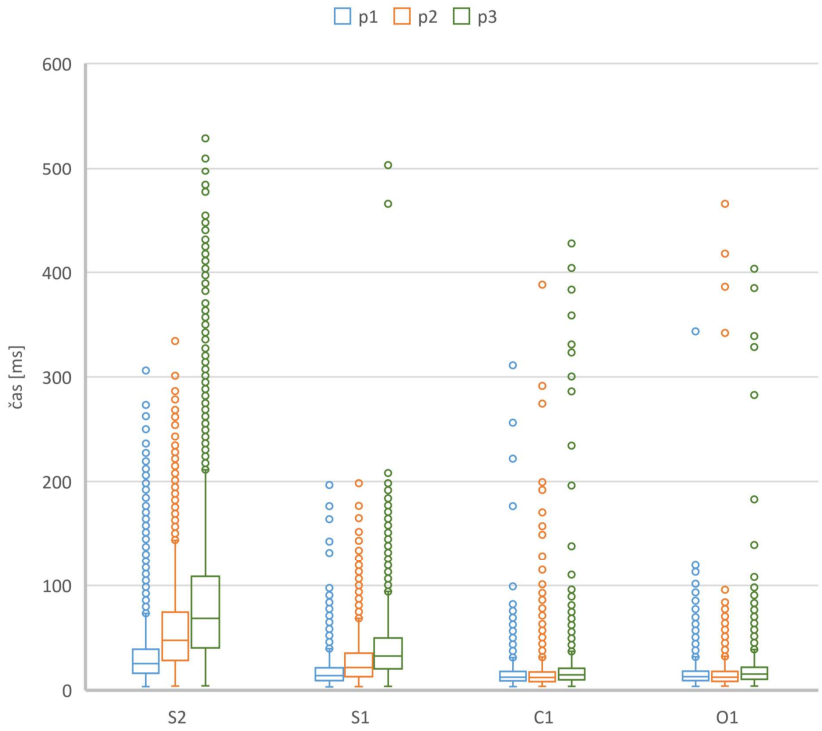
- vplyv úrovne výkonnosti CPU na rýchlosť vkladania na inštanciách S1 a S2, teda čím vyšší výkon CPU má, tak tým menší čas dosahuje
- vplyv paralelného prístupu do úložiska, ktorý zvládajú lepšie viacjadrové inštancie
- Vyššie časy na všetkých inštanciách zaznamenalo vkladanie do `ImageMetadataUtc` než do `ImageMetadataFi`.

inštancia	**p1 [ms]**	**p2 [ms]**	**p3 [ms]**
C1	14,3652	14,0502	16,9259
O1	14,6937	14,5167	17,7789
S1	16,8331	26,9967	38,5057
S2	30,8565	56,3414	81,6327

Tabuľka 7.2: Priemerné časy vkladania do tabuľky `ImageMetadataFi`

inštancia	**p1 [ms]**	**p2 [ms]**	**p3 [ms]**
C1	7,4345	25,8629	26,9018
O1	12,8533	23,8280	25,8628
S1	20,0585	28,4715	33,9419
S2	42,8791	46,9355	61,9511

Tabuľka 7.3: Priemerné časy vkladania do tabuľky `ImageMetadataUtc`



Obr. 7.1: Graf časov vkladania do tabuľky `ImageMetadataFi`

7.4 Vyhľadávanie

V prípade dopytovania formou vyhľadávania postačuje jedno databázové spojenie, aby sa dal otestovať výkon úložiska metadát. Pre objektívnosť merania na viacerých inštanciách bol čas dopytu prepočítaný na jeden snímok.

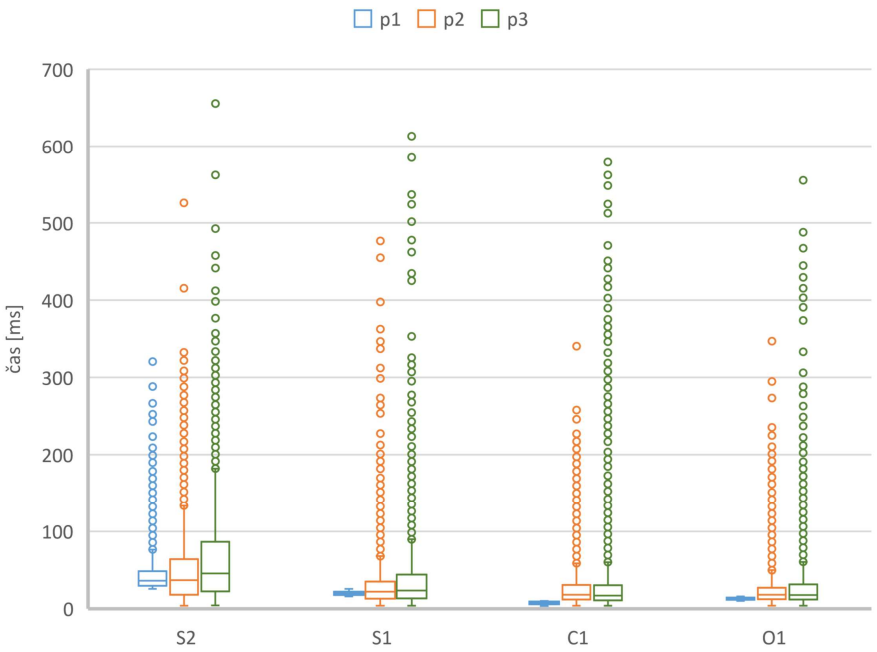
7.4.1 Náhodný prístup

Na grafe 7.3 a v tabuľke 7.4 pre `ImageMetadataFi` a na grafe 7.4 a v tabuľke 7.4 pre `ImageMetadataUtc` je vidieť:

- pozitívny vplyv výkonu CPU na dobu dopytov
- vyhľadávanie podľa sekvencie snímkov a vyhľadávanie na základe časového rozpätia podáva na inštanciách O1 a C1 prakticky rovnaké výsledky

7.4.2 Vyhľadávanie podľa predlohy

Na grafe 7.5 a v tabuľke 7.6 je vidieť:



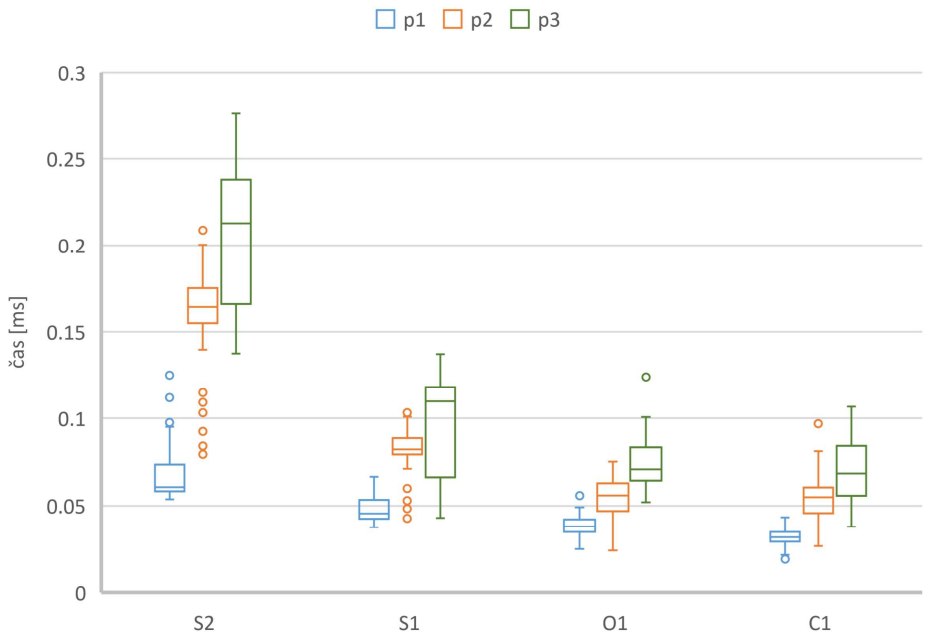
Obr. 7.2: Graf časov vkladania do tabuľky `ImageMetadataUtc`

inštancia	**p1 [ms]**	**p2 [ms]**	**p3 [ms]**
C1	0,031707	0,053544	0,070190
O1	0,038478	0,054972	0,074048
S1	0,047857	0,080412	0,094745
S2	0,068337	0,158423	0,203460

Tabuľka 7.4: Priemerné časy vyhľadávania podľa rozsahu sekvecií v tabuľke `ImageMetadataFi`

- pozitívny vplyv výkonu CPU na čas dopytov (S1, S2 a O1)
- pozitívny vplyv vyššieho počtu jadier na čas paralelných dopytov (S1 a S2 verzus O1 a C1)
- pozitívny vplyv na čas dopytov pri rozdelení dát na viacero inštancií (porovnanie O1 a C1)

Na testovacích dátach sme dosiahli priaznivé výsledky zodpovedajúce horizontálnemu aj vertikálnemu škálovaniu.



Obr. 7.3: Graf časov náhodného prístupu podľa sekvencie

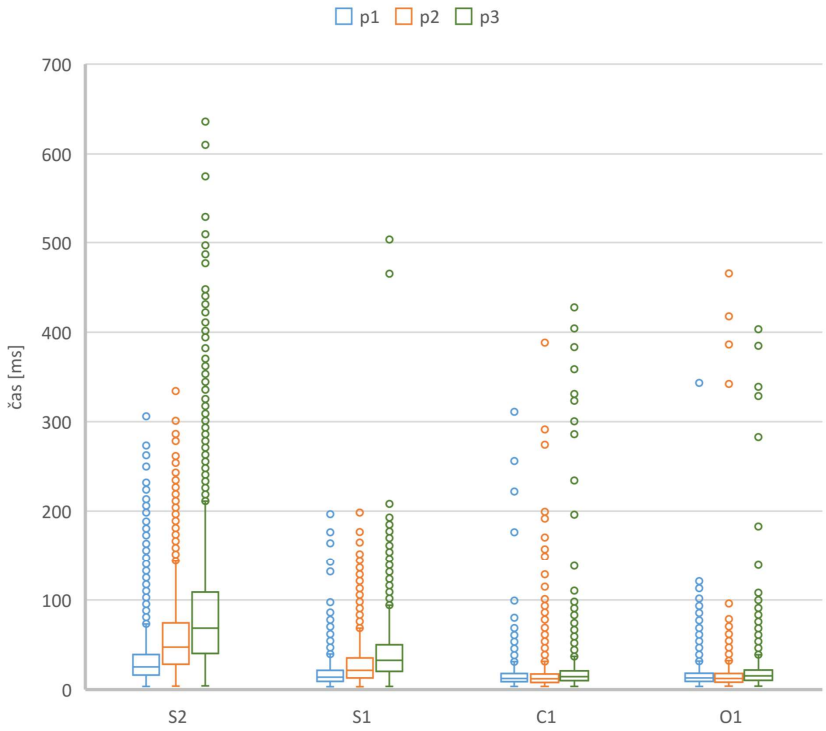
inštancia	p1 [ms]	p2 [ms]	p3 [ms]
C1	7,4345	25,8629	26,9018
O1	12,8533	23,8280	25,8628
S1	20,0585	28,4715	33,9419
S2	42,8791	46,9355	61,9511

Tabuľka 7.5: Priemerné časy vyhľadávania podľa časového rozsahu z tabuľky `ImageMetadataUtc`

7.5 Nároky na priestor na disku

Nároky na priestor na disku sú závislé od:

1. počtu uložených snímkov
2. situácie, ktorá je zachytená na snímku, tj. od počtu detekovaných objektov
3. úrovne detailu metadát, ktoré generuje analytický proces
4. réžie PostgreSQL databázy, ktorá je závislá na konfigurácii

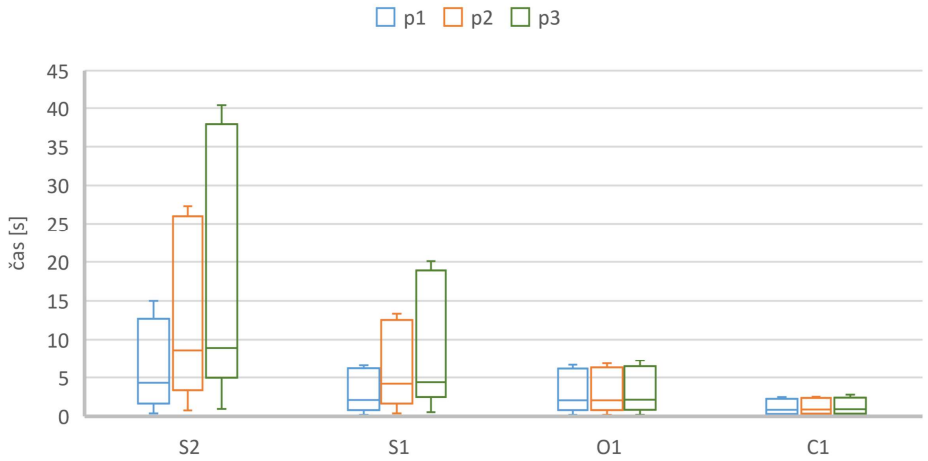


Obr. 7.4: Graf časov vyhľadávania podľa časového rozsahu z tabuľky `ImageMetadataUtc`

inštancia	**p1** [s]	**p2** [s]	**p3** [s]
C1	1,2470	1,2186	1,1634
O1	3,2766	3,2330	3,1200
S1	8,5016	6,3455	3,1410
S2	17,0296	13,0984	6,4341

Tabuľka 7.6: Priemerné časy vyhľadávania podľa predlohy

Tabuľka 7.7 zachytáva skutočné využitie diskového priestoru. Je nutné pripomenúť, že testovacie metadáta sú naimportované vo dvoch tabuľkách. Hodnota réžie PostgreSQL je spočítaná po niekoľkých iteráciách testovania bez naimportovaných metadát.
Nároky na diskový priestor čisto len tabuliek v úložisku sú dvakrát vyššie než oproti súborom vo formáte MessagePack (v tabuľke 7.1). Zmierniť nároky na úložný priestor bez výraznej straty výkonnosti riešenia by mala zabezpečiť kompresia metadát pomocou algoritmu L4Z.



Obr. 7.5: Graf časov vyhľadávania podľa predlohy

inštancia	**tabuľky [MiB]**	**réžia [MiB]**	**spolu [MiB]**
S1	959,98	2582	3541,98
O1	959,98	2547	3506,98

Tabuľka 7.7: Nároky na úložný priestor testovacích dát

7.6 Zhrnutie výsledkov testovania

Testovania preukázalo naplnenie funkčných požiadaviek na výkon V1, V2 a V3.

Dvojnásobne väčšie nároky na úložný priestor na chápeme ako daň za rýchlejšie vyhľadávanie. Fakt, že navrhované vyhľadávanie je skutočne rýchlejšie než prehľadávanie celého priestoru, bude nutné ešte potvrdiť meraniami. Počas vývoja riešenia prebehli potvrdzujúce merania iba na neformálnej úrovni.

Záver

Ciele tejto záverečnej práce, s výnimkou sémantického vyhľadávania, na základe splnených funkčných požiadaviek sa podarilo naplniť. Výsledok tejto diplomovej práce je návrh a implementácia základov univerzálneho úložiska na výsledky z analýzy obrazu a sním spojeným API a návrhom manažovacej databázy.

V kontexte umelej inteligencie a počítačového videnia navrhované riešenie môže hrať úlohu dátových skladov pre metadáta z analýzy obrazu. S rastúcim rozšírením spracovávania obrazu budú úložiska takéhoto typu čím ďalej, tým častejšie vyžadované.

Okrem cieľov, ktoré sme si stanovili výsledky tejto práce navyše pomohli k zjednoteniu formátov ako výstupov z analýzy obrazu v spoločnosti Quantasoft a umožnili ich uchovávať v špecializovanom úložisku. Výstupy tejto práce sú nasadené v prostredí spoločnosti.

Práca do budúcnosti

Od doby prvotného zadania až po implementáciu sa v rámci spoločnosti Quantasoft zmenili priority, čo sa dotklo sémantického vyhľadávania. Túto funkcionalitu sme posunuli na realizáciu v blízkej budúcnosti.

Počas implementácie sme nestihli podrobne pretestovať presnosť vyhľadávania podľa predlohy. Ďalej by bolo vhodné sa pozrieť na možnosti indexácie identifikátorov objektov pomocou GiST. Testovanie by bolo vhodné spustiť na omnoho väčšej dátovej množine a na širšej palete inštancií ako napríklad klustre s viacerými uzlami alebo na NVIDIA Jetson Nano, ako predstaviteľovi edge zariadení

Implementáciu budeme rozširovať o podporu vizuálnych objektov ako sú vozidlá, ľudské postavy, batožina a iné.

Literatúra

[1] P., G.: Evolution of Object Detection and Localization Algorithms [online]. February 2018, [cit. 2019-06-20]. Dostupné z: https://towardsdatascience.com/evolution-of-object-detection-and-localization-algorithms-e241021d8bad

[2] Corporation, N.: Analyze Data From Cameras, Sensors and IoT Gateways in Real-Time [online]. June 2019, [cit. 2019-06-20]. Dostupné z: https://developer.nvidia.com/deepstream-sdk

[3] Corporation, M.: *rowversion (Transact-SQL)*. July 2017, [cit. 2019-06-01]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/data-types/rowversion-transact-sql?view=sql-server-2017

[4] Furuhashi, S.: MessagePack: It's like JSON. but fast and small [online]. March 2019, [cit. 2019-06-03]. Dostupné z: https://web.archive.org/web/20190603222434/http://msgpack.org/

[5] Kulkarni, A.: MessagePack for C# (.NET, .NET Core, Unity, Xamarin) [online]. February 2017, [cit. 2019-06-01]. Dostupné z: https://github.com/neuecc/MessagePack-CSharp

[6] Hampton, L.: Eye or the Tiger: Benchmarking Cassandra vs. TimescaleDB for time-series data [online]. May 2018, [cit. 2019-06-15]. Dostupné z: https://blog.timescale.com/time-series-data-cassandra-vs-timescaledb-postgresql-7c2cc50a89ce/

[7] M., F.: TimescaleDB vs. InfluxDB: Purpose built differently for time-series data [online]. June 2019, [cit. 2019-06-15]. Dostupné z: https://blog.timescale.com/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/

[8] Kulkarni, A.: Why SQL is beating NoSQL, and what this means for the future of data [online]. May 2018, [cit. 2019-06-15]. Dostupné z:

https://blog.timescale.com/why-sql-beating-nosql-what-this-means-for-future-of-data-time-series-database-348b777b847a/

[9] Group, T. P. G. D.: *PostgreSQL: Documentation: 11: F.9. cube [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/cube.html

[10] Group, T. P. G. D.: *Tablespaces [online]*. May 2019, [cit. 2019-06-21]. Dostupné z: https://www.postgresql.org/docs/11/manage-ag-tablespaces.html

[11] Group, T. P. G. D.: *Table Partitioning [online]*. May 2019, [cit. 2019-06-21]. Dostupné z: https://www.postgresql.org/docs/11/ddl-partitioning.html

[12] Group, T. P. G. D.: *Comparison of Different Solutions [online]*. May 2019, [cit. 2019-06-11]. Dostupné z: https://www.postgresql.org/docs/11/different-replication-solutions.html

[13] Group, T. P. G. D.: *Composite Types [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/rowtypes.html

[14] Cao, Q.; Shen, L.; Xie, W.; aj.: VGGFace2: A dataset for recognising faces across pose and age. In *International Conference on Automatic Face and Gesture Recognition*, 2018.

[15] Schroff, F.; Kalenichenko, D.; Philbin, J.: FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015, doi: 10.1109/cvpr.2015.7298682. Dostupné z: http://dx.doi.org/10.1109/CVPR.2015.7298682

[16] Lu, W.; Hou, J.; Yan, Y.; aj.: MSQL: efficient similarity search in metric spaces using SQL. *The VLDB Journal*, ročník 26, č. 6, Dec 2017: s. 829–854, ISSN 0949-877X, doi:10.1007/s00778-017-0481-6. Dostupné z: https://doi.org/10.1007/s00778-017-0481-6

[17] Group, T. P. G. D.: *Query Language (SQL) Functions [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/xfunc-sql.html

[18] Group, T. P. G. D.: *Structure of PL/pgSQL [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/plpgsql-structure.html

[19] Group, T. P. G. D.: *C-Language Functions [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/xfunc-c.html

[20] Group, T. P. G. D.: *Procedural Languages [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/external-pl.html

[21] Group, T. P. G. D.: *Function Overloading [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/xfunc-volatility.html

[22] Group, T. P. G. D.: *Function Overloading [online]*. May 2019, [cit. 2019-06-01]. Dostupné z: https://www.postgresql.org/docs/11/xfunc-overload.html

[23] Taveira de Oliveira, E.: pg similarity [online]. May 2019, [cit. 2019-06-01]. Dostupné z: https://github.com/eulerto/pg_similarity

[24] Jones, E.; Oliphant, T.; Peterson, P.; aj.: Distance computations (scipy.spatial.distance) [online]. 2001–, [cit. 2019-06-01]. Dostupné z: https://docs.scipy.org/doc/scipy/reference/spatial.distance.html

Dodatok **A**

Zoznam použitých skratiek

AI Artificial Inteligence, umelá inleligencia.

CAD Computer Assisted Drawing.

CV Computer Vision.

FullHD Full High Definition, rozlíšenie obrazu 1920 na 1080 bodov.

LPR License Plate Recognition.

LTS Long Term Support.

ML Machine Learning, strojové učenie.

OLTP Online Transaction Processing.

RGBA Obrazový bod s červenou, zelenou a modrou zložkou vrátane priehľadnosti.

UDF User Defined Functions.

UTC Coordinated Universal Time.

Dodatok **B**

Obsah priloženého DVD

- `data` ..... adresár s testovacími dátami
- `exe` ..... adresár so spustiteľnou formou implementácie
  - `api` ..... adresár so spustiteľnou formou implementácie API
    - `net_standard` ..... NuGet balíčky pre .NET Core
    - `python` ..... spustiteľná forma pre Python
  - `imagemetadata` ..... skompilované knižnice formátu metadát
  - `test` ..... adresár s testovacími skriptami
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `api` ..... API v C#, Python a C++
    - `db` ..... úložiska metadát
    - `imagemetadata` ..... ImageMetadata formátu
    - `mgmt` ..... manažment databázy
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text`
  - `thesis.pdf` ..... text práce vo formáte PDF