Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Produkční nasazení vybrané databáze pro ukládání senzorických dat v reálném čase

Bc. Jakub Hora

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jakub Hora

Informatika

Název práce

Produkční nasazení vybrané databáze pro ukládání senzorických dat v reálném čase

Název anglicky

Production deployment of a selected database for real-time storage of sensor data

---

Cíle práce

Hlavním cílem diplomové práce je tvorba doporučení pro produkční nasazení vybrané databáze pro ukládání senzorických dat proudící v reálném čase. Dílčími cíli je vytvoření přehledu dostupných řešení pro ukládání senzorických dat v reálném čase, jejich porovnání a zhodnocení, výběr vhodné databáze a vytvoření experimentálního pracoviště pro experimenty s různou konfigurací databáze.

Metodika

Metodika diplomové práce bude založena na studiu a analýze dostupných literárních zdrojů a na komparaci dostupných databází pro časové řady, podle které bude následně vybrána vhodná databáze pro následující experimenty. Experimenty se budou týkat různých konfigurací databáze v kontextu různých databázových požadavků a budou se provádět na experimentálním pracovišti, které bude zahrnovat server, programy Node-RED a vybranou vhodnou databázi. Pro dokumentační účely budou použity modelovací jazyky UML a CASE nástroje. Závěry a doporučení diplomové práce budou formulovat výsledky teoretických východisek a vlastní práce.

Doporučený rozsah práce

50-60 stran

Klíčová slova

databáze, databázové technologie, senzorická data, časové řady, IoT, InfluxDB, TimescaleDB, QuestDB, Grafana, Node-RED

Doporučené zdroje informací

GROFF, James a Paul WEINBERG. SQL: kompletní průvodce. Brno: CP Books, 2005. ISBN 80-251-0369-2.

HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. Big Data a NoSQL databáze. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.

MEIER, Andreas a Michael KAUFMANN. SQL & NoSQL databases: models, languages, consistency options and architectures for Big data management. Wiesbaden: Springer, 2019. ISBN 978-3-658-24548-1.

MOSTAFA, Jalal, Sara WEHBI, Suren CHILINGARYAN a Andreas KOPMANN. SciTS: A Benchmark for Time-Series Database in Scientific Experiments and Industrial Internet of Things [online]. 2022, 11 [cit. 2022-05-18]. Dostupné z: https://arxiv.org/pdf/2204.09795.pdf

PALMA, Wilfredo. Time series analysis. Hoboken: John Wiley & Sons, 2016. ISBN 978-1-118-63432-5.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

doc. Ing. Jiří Vaněk, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 31. 5. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 25. 12. 2022

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Produkční nasazení vybrané databáze pro ukládání senzorických dat v reálném čase" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2023 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Jiřímu Vaňkovi, Ph.D. za cenné připomínky a rychlou odezvu během vypracovávání diplomové práce. Dále bych chtěl poděkovat Ing. Vojtěchovi Novákovi, Ph.D. za odbornou pomoc s vypracováváním diplomové práce a nakonec Ing. Zdeňkovi Kolářovi za trpělivost a odborné konzultace nad databází časových řad InfluxDB.

Produkční nasazení vybrané databáze pro ukládání senzorických dat v reálném čase

Abstrakt

Diplomová práce se zabývá problematikou ukládání senzorických dat v reálném čase pomocí specializovaných databází časových řad.

V teoretické části jsou nejprve charakterizovány časové řady a druhy databázových technologií. V další části je vytvořen přehled vybraných nejpoužívanějších databází časových řad a shrnutí open-source databází časových řad podle 7 kritérií. V poslední části jsou charakterizovány další využité programy pro praktickou část práce a metody vícekriteriálního rozhodování.

V praktické části jsou nejprve porovnávány vybrané open-source databáze časových řad pomocí vícekriteriálního rozhodování. Jako kompromisní varianta vyšla databáze časových řad InfluxDB. Dále je vytvořeno 11 experimentálních pracovišť pro experimenty nad databází InfluxDB verze 2.x. K experimentům se zvolila senzorická data pražské hromadné dopravy z REST API Golemio. V poslední části se práce věnuje samotným čtyřem experimentům nad databází časových řad InfluxDB.

Výsledkem práce jsou doporučení pro produkční nasazení databáze časových řad InfluxDB, která vychází z teoretické a praktické části práce.

Klíčová slova: databáze, databázová technologie, senzorická data, časové řady, IoT, InfluxDB, TimescaleDB, QuestDB, Grafana, Node-RED

Production deployment of selected database for real-time storage of sensor data

Abstract

The thesis deals with the problem of real-time storage of sensor data using specialized time series databases.

The theoretical part first characterizes time series and types of database technologies. In the next part, an overview of selected most used time series databases and a summary of open-source time series databases according to 7 criteria are made. In the last section, other programs used for the practical part and methods for multi-criteria decision making are characterized.

In the practical part, selected open-source time series databases are first compared using multi-criteria decision making. The time series database InfluxDB emerged as a compromise option. Next, 11 experimental workstations are created for experiments over the InfluxDB database version 2.x. The sensor data of Prague public transport from the REST API Golemio was chosen for the experiments. The last section focuses on the four experiments over the time series database InfluxDB.

The result of the thesis are recommendations for production deployment of time series database InfluxDB based on the theoretical and practical part of the thesis.

Keywords: database, database technology, sensor data, time series, IoT, InfluxDB, TimescaleDB, QuestDB, Grafana, Node-RED

Obsah

1 Úvod

V dnešní době, kdy se snížila pořizovací cena hardwaru a popularita internetu věcí (IoT) se neustále zvětšuje, ať je to z komerčního, průmyslového nebo IT odvětví, se kladou větší nároky i na ukládání velkého objemu získaných senzorických dat a jejich vyhodnocení. Databáze pro ukládání senzorických dat musí umět takové škálování, které zvládne zpracovávat stále rostoucí objem senzorických dat v reálném čase.

Jelikož jsou senzorická data neměnná, objemově velká a uspořádaná podle času, využívají se pro jejich ukládání specializované databáze časových řad. Databáze časových řad kladou důraz na rychlost zpracování, způsob uložení dat a jejich vyhodnocení. Kvůli stále se zvětšujícímu objemu senzorických dat se právě databáze časových řad řadí k nejrychleji rostoucímu segmentu v databázovém průmyslu.

Databází časových řad existuje nespočet různých druhů, ať jsou to relační databáze, NoSQL databáze, databáze jako služba nebo volně dostupné databáze (open source). Jednotlivé databáze časových řad se od sebe velmi liší ve funkcích a dosahují odlišných výkonností.

Senzorická data nestačí pouze ukládat, ale je potřeba je určitým způsobem vizualizovat a následně je vyhodnotit nebo vyčíst určité hodnoty z těchto dat. Pro tento účel se využívá např. vizualizační program Grafana, ale některé databáze časových řad již obsahují své vlastní nástroje pro vizualizaci uložených senzorických dat.

Každá databáze časových řad má nějaké svoje odlišnosti od ostatních, které je třeba znát pro správné využívání dané databáze, a proto vznikla tato diplomová práce, jejíž cílem je tvorba přehledu dostupných řešení pro ukládání časových řad, jejich porovnání a následný výběr vhodné databáze pro experimenty a tvorba doporučení pro produkční nasazení vybrané databáze pro ukládání senzorických dat v reálném čase.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce je tvorba doporučení pro produkční nasazení vybrané databáze pro ukládání senzorických dat proudící v reálném čase. Dílčími cíli je vytvoření přehledu dostupných řešení pro ukládání senzorických dat v reálném čase, jejich porovnání a zhodnocení, výběr vhodné databáze a vytvoření experimentálního pracoviště pro experimenty s různou konfigurací databáze.

2.2 Metodika

Metodika diplomové práce bude založena na studiu a analýze dostupných literárních zdrojů a na komparaci dostupných databází pro časové řady, podle které bude následně vybrána vhodná databáze pro následující experimenty. Experimenty se budou týkat různých konfigurací databáze v kontextu různých databázových požadavků a budou se provádět na experimentálním pracovišti, které bude zahrnovat server, programy Node-RED a vybranou vhodnou databázi. Pro dokumentační účely budou použity modelovací jazyky UML a CASE nástroje. Závěry a doporučení diplomové práce budou formulovat výsledky teoretických východisek a vlastní práce.

3 Teoretická východiska

V následujících podkapitolách jsou charakterizovány časové řady, jaké jsou druhy databázových technologií pro ukládání časových řad, přehled databází pro ukládání časových řad, další využité technologie během experimentů, modely vícekriteriálního rozhodování pro porovnání, a nakonec shrnutí teoretických východisek. Pokud je český ekvivalent všeobecně použitelný, v práci byly použity české ekvivalenty. V případě, že není, byl použit anglický termín.

3.1 Časové řady

V této podkapitole jsou charakterizovány časové řady a jejich klasifikace, dekompozice, úpravy a predikce.

Senzorická data z oblasti internetu věcí (IoT) jako sledování teploty, spotřeby energie, hladin toků a mnoho dalších mají charakter časových řad, protože se vždy jedná o sekvenci naměřených údajů v pravidelných časových intervalech. Časové intervaly se mohou pohybovat v řádech sekund, ale i hodin či dnů. Senzorická data obsahují vždy naměřenou hodnotu a čas provedení měření. Tato senzorická data se pak většinou dále analyzují ve formě časových řad.

„Časová řada je soubor pozorování prováděných postupně v průběhu času“ [1]

3.1.1 Klasifikace časových řad

Časové řady lze klasifikovat podle několika kritérií. Prvním kritériem je typ ukazatele, který se sleduje. Podle tohoto kritéria se časové řady dělí na intervalové a okamžikové.

- Intervalové časové řady zahrnují ukazatele s hodnotami, které závisí na délce časového intervalu sledování, jako například objem výroby nebo spotřeba surovin.
- Okamžikové časové řady obsahují ukazatele, jejichž hodnoty se vztahují k jistým časovým okamžikům. Příkladem může být počet zaznamenaných pozitivních případů COVID-19 k určitému datu. [2]

Druhým kritériem, podle kterého lze klasifikovat, je délka intervalu sledování hodnot na dlouhodobé, krátkodobé a vysokofrekvenční časové řady.

- Dlouhodobé časové řady mají periodicitu delší než 1 rok.
- Krátkodobé časové řady se sledují v úsecích kratších než 1 rok.
- Vysokofrekvenční časové řady se sledují v úsecích kratších než 1 týden. [2]

Dalším kritériem klasifikace časových řad jsou druhy ukazatelů, podle kterých se dělí na časové řady absolutních ukazatelů a odvozených ukazatelů.

- Časové řady absolutních ukazatelů obsahují původní hodnoty, jak byly naměřeny.
- Časové řady odvozených ukazatelů obsahují hodnoty, které byly transformovány (součtové, poměrové). [3]

3.1.2 Dekompozice časových řad

Časové řady lze rozložit na několik složek, přičemž celou časovou řadu lze vyjádřit dvěma způsoby. Prvním způsobem je aditivní (součtová) dekompozice pomocí následující rovnice:

$$y_t = T_t + C_t + S_t + \varepsilon_t \tag{1}$$

a druhým způsobem je multiplikativní (součinová) dekompozice podle rovnice:

$$y_t = T_t * C_t * S_t * \varepsilon_t, \tag{2}$$

kde jednotlivé sčítance vyjadřují:

- T_i – hodnotu trendové složky
- S_i – hodnotu sezónní složky
- C_i – hodnotu cyklické složky
- ε_i = hodnotu náhodné složky [4]

Trend (T)

Trend v časových řadách znázorňuje dlouhodobé změny v průměrném chování za dlouhé období. Trend může být rostoucí, klesající, strmý, mírný a v průběhu času se může měnit. [2]

Sezónní složka (S)

Sezónnost je systematické periodické kolísání v časové řadě. Toto kolísání se se vždy opakuje každý rok ve stejné nebo podobné podobě. Tyto změny jsou způsobeny především střídáním ročních období a lidskými zvyky. Sezónnost je spojena s krátkodobými a vysokofrekvenčními časovými řadami. [2]

Cyklická složka (C)

Cyklickou složku lze také nazvat periodickou složkou nebo fluktuací okolo trendu, kde se střídá fáze růstu a fáze poklesu. Tyto změny mohou být způsobeny vnějšími vlivy, ale její příčiny mohou být i mimo ekonomickou oblast. Její eliminace je obtížná, protože je velmi těžké nalézt její příčiny vzniku, tak i z výpočetních důvodů, jelikož se charakter této složky může v čase měnit. [4]

Reziduální složka (ε)

Reziduální složka patří mezi nesystematické složky časových řad, protože je tvořena náhodnými fluktuacemi v průběhu časové řady a pokrývá chyby v měření údajů časové řady. [4]

Výslednou dekompozicí lze zjistit chování vývoje analyzovaného jevu a dále lze analyzovat časové řady jako např. odstranit sezónní složky nebo trend a výběr dalších vhodných metod pro další analýzu.

3.1.3 Úpravy časových řad

Před další analýzou časových řad je nutné výchozí data transformovat a upravit. Mezi nejčastější úpravy lze zahrnout:

- Doplnění chybějících hodnot – pokud v časové řadě chybí nějaké pozorování, tyto chybějící hodnoty se mohou doplnit nulami, aritmetickým průměrem, mediánem, lineární interpolací, trendem či odhadem založeným na modelu chování procesu. Doplněné údaje nejsou plnohodnotné a jejich přítomnost snižuje věrohodnost analýzy.
- Časový posun – představuje posunutí časové řady dopředu nebo dozadu v čase oproti původní časové řadě.
- Sezónní diference – umožňuje odstranění lineárního trendu a sezónních vlivů.

- Kumulativní součet – představuje součet hodnot části časové řady
- Vyhlazování časových řad – odstraňuje odchylky šumu vzniklé při měření v krátkých časových intervalech. Mezi metody vyhlazování časových řad patří středové klouzavé průměry, klouzavé průměry z předchozích hodnot a klouzavé mediány. [5]

3.1.4 Predikce časových řad

Předpovědi dalších hodnot je jedním z nejčastějších cílů analýzy časových řad. Predikce ale není přesná, jedná se pouze o pravděpodobný odhad, kterého lze docílit dvěma způsoby, bodovým a intervalovým odhadem.

- Bodový odhad lze vypočítat podle daného vzorce a jeho výstupem je hodnota, která je s největší pravděpodobností nejblíže odhadované hodnotě. Odhad ale není nikdy stoprocentní.
- Intervalový odhad je přesnější než bodový. Výsledkem intervalového odhadu je interval spolehlivosti, který určí, že v daném intervalu se nachází s vysokou pravděpodobností skutečná neznámá hodnota. [3]

Mezi metody pro predikci časových řad lze zahrnout metodu ARIMA (Autoregresive Integrated Moving Average) nebo pro predikci sezónních vlivů metodu SARIMA (Seasonal ARIMA). [5]

3.2 Druhy databázových technologií pro ukládání časových řad

Existuje více databázových technologií pro ukládání časových řad a v této podkapitole jsou jednotlivě vymezeny relační databáze, NoSQL databáze, cloudové databáze a databáze časových řad.

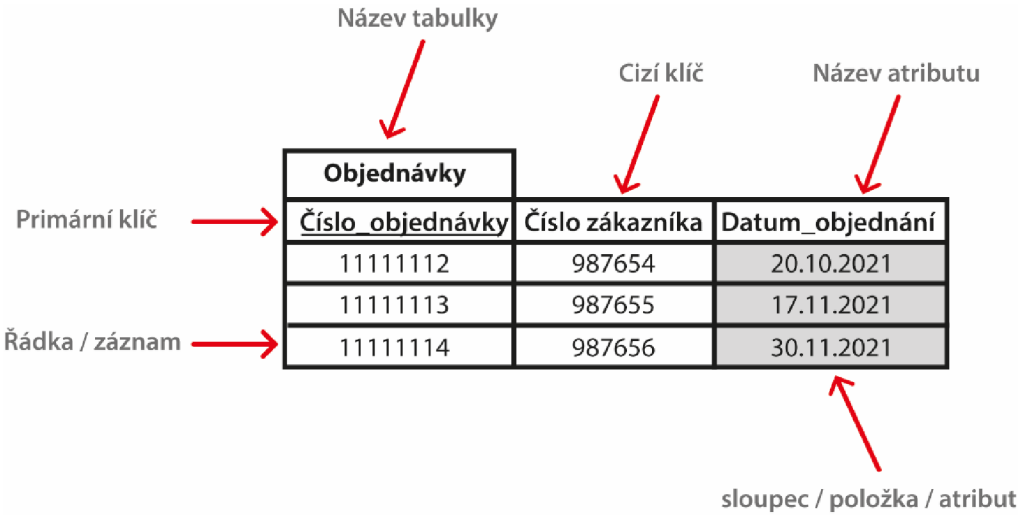
3.2.1 Relační databáze (RDBS)

Hlavním prvkem relační databáze je SŘBD (Systém řízení báze dat), anglicky Database Management System (DBMS). Ten představuje software, který je dodávaný výrobcem databáze pro popis, ukládání dat a dotazování se nad nimi. Mezi nejznámější SŘBD patří Microsoft SQL Server, Oracle Database, MySQL a PostgreSQL. [6]

Relační datový model databáze definují 12 Coddových pravidel od Dr. Edgara Franka Codda z roku 1985. Základním organizačním prvkem je tabulka a uspořádání hodnot

v řádcích a sloupcích, přičemž každá tabulka má jedinečný název, který identifikuje její obsah. Každý sloupec musí mít unikátní název, který se nesmí opakovat. [7] Struktura tabulky řádek-sloupec je znázorněna na obrázku 1.

Relační databáze vychází z předpokladu, že je známá struktura ukládání dat, a proto jsou datové struktury rozděleny na co nejmenší kompaktní celky, které jsou uloženy v samostatných tabulkách, které se spojují. [8]



Obrázek 1 - vzorová tabulka relační databáze, zdroj: vlastní zpracování

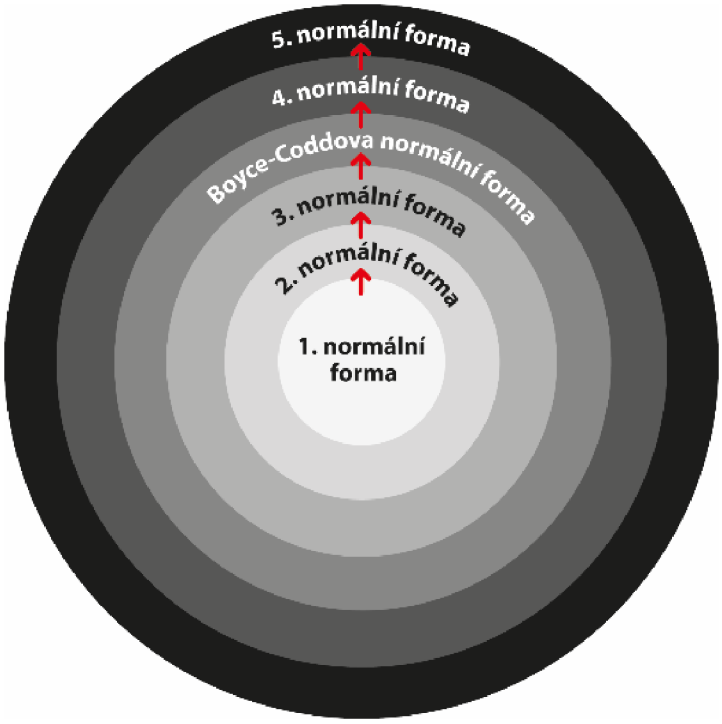
Každá tabulka má tzv. primární klíč, který značí určitý sloupec nebo kombinaci sloupců, jejichž hodnoty jednoznačně identifikují každý řádek v tabulce. Tabulka má pouze jeden primární klíč. Primární klíč vždy obsahuje pro každý řádek v tabulce unikátní hodnotu, a tak žádné dva řádky s primárním klíčem nemohou být duplikátem jiného řádku. Cizí klíč je určitý sloupec, jehož hodnota odpovídá v jiné tabulce primárnímu klíči. Primární klíč a cizí klíč tvoří relaci rodič-potomek mezi tabulkami, které je obsahují. Jedna tabulka může obsahovat více cizích klíčů. [7]

Každá relační databáze musí dodržovat datovou normalizaci, která zajišťuje správnou strukturu a konzistenci datového modelu. Datová normalizace obsahuje 6 pravidel, tzv. normální formy:

- 1. normální forma (1.NF) – relace nesmí obsahovat násobná data
- 2. normální forma (2. NF) – všechna neklíčová data musí záviset na celém primárním klíči
- 3. normální forma (3. NF) – všechna neklíčová data musí záviset pouze na klíčových hodnotách a nikoli mezi sebou

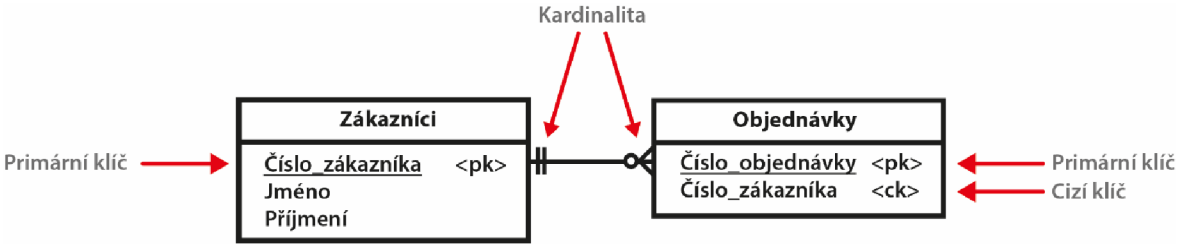
- Boyce-Coddova normální forma (BCNF) – všechna data musí záviset pouze na klíčových hodnotách a nikoli mezi sebou
- 4. normální norma (4. NF) – složený primární klíč nesmí být tvořen z nezávislých dat
- 5. normální forma (5. NF) – složený primární klíč nesmí obsahovat párové cyklické závislosti [9]

Pro normální formy platí, že jsou vnořené, tzn., že pokud relace podléhá jedné z vyšších forem, podléhá i všem nižším normálním formám, jak je uvedeno na obrázku 2.



Obrázek 2 - schéma normálních forem, zdroj: vlastní zpracování

Relační databáze obsahuje tzv. relační schéma, které definuje všechny součásti n-tice, která tvoří relaci. Vztahy mezi entitami stanovuje kardinalita, která je trojího typu. Kardinalita typu 1:1 určuje vztah jedné entity množiny A s jednou entitou množiny B, kardinalita typu 1:N určuje, že jedna entita množiny A je ve vztahu s více entitami množiny B, kardinalita typu M:N určuje, že více entit množiny A má vztah s více entit množin B. Parcialita vztahu určuje, jestli jedna entita může existovat bez instance druhé entity. Parcialita rozšiřuje kardinalitu o možnosti 0..1 a 0..M. [10] V obrázku 3 na straně 18 byla pro znázornění kardinality v relační databázi použita v ER diagramu notace vraních stop. Alternativou lze využít např. Chenovu notaci.



Obrázek 3 – vzorový ER diagram relační databáze, zdroj: vlastní zpracování

Dotazovací a manipulační operace nad databází se dělí do čtyř vlastností CRUD – Create (vytváření), Read (čtení), Update (změna) a Delete (mazání). Integrita databáze se zajišťuje pomocí vlastností transakcí, které se provádějí podle vlastností ACID – Atomicity (atomicita), Consistency (konzistence), Isolation (izolace) a Durability (trvanlivost). [11]

Jazyk SQL

Jazyk SQL (Structured Query Language) je deklarativní, dotazovací jazyk navržený firmou IBM v 70. letech 20. století. Jazyk SQL podléhá standardům ANSI/ISO. Součástí jazyka SQL jsou jazyky DDL (jazyk pro definici dat), který realizuje změny struktury databáze, DML (jazyk pro manipulaci s daty), který realizuje přidání, mazání nebo úpravy dat a DCL (jazyk pro řízení dat), který řídí přístup k datům. [7]

Jazyk SQL podporuje různé datové typy, které definují standardy ANSI/ISO např. CHAR, VARCHAR, INT, FLOAT, DATE, TIME, TIMESTAMP, INTERVAL, přičemž pro časové řady se obvykle využívají datové typy DATE, TIMESTAMP a další. [7]

Dotazy nad daty se vytvářejí za pomocí následujícího vzoru:

SELECT název sloupce FROM tabulka WHERE podmínka. [10]

Využití relační databáze pro časové řady

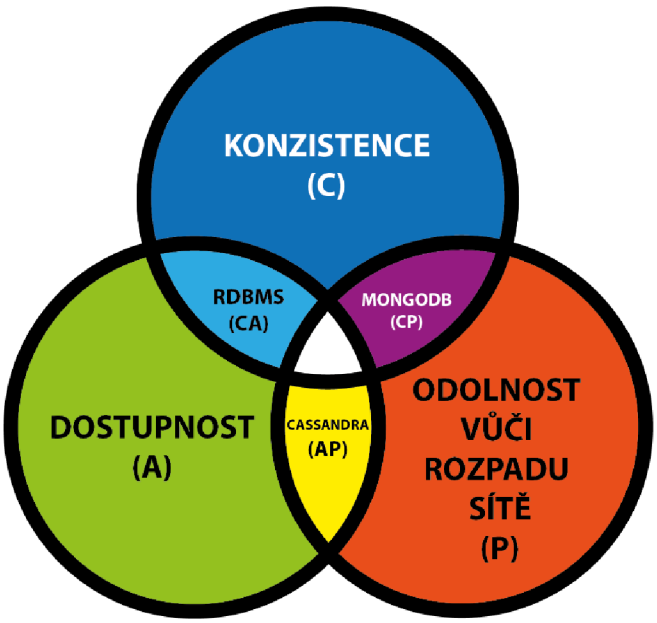
Data časových řad se v relačních databázích mohou indexovat podle času pomocí datové struktury B-stromů. Častěji se ale data časových řad rozdělují do dvou dimenzí: časového intervalu a primárního klíče. Tento způsob rozdělí data do tzv. bloků (chunků) a všechny indexy jsou vytvářeny nad nimi. Každý blok může mít pevný časový interval (např. 1 hodina) nebo pevnou velikost (např. 1 GB). Takto rozdělené tabulky se nazývají hypertabulky (hypertables). [12]

3.2.2 NoSQL databáze

Tento typ databází v posledních letech zažívají velký rozmach, protože využívají jiný způsob ukládání dat než relační databáze, který přináší mnoho výhod. Tento typ databáze je navíc vhodný pro ukládání velkého objemu dat tzv. Big Data.

Oproti relačním databázím, které využívají hlavně vertikální škálování, tedy zvyšování výpočetního výkonu daného serveru prostřednictvím přidáním výkonnějšího hardware(u), NoSQL databáze využívají především horizontální škálování, tedy místo vylepšení jednoho serveru se přidá další a data se rozdistribuují mezi ně. [8]

NoSQL je distribuovaný systém, který obsahuje vlastnosti tzv. CAP teorému – Consistency (konzistence – existuje jediná aktuální verze dat), Availability (dostupnost – všechny požadavky pro zápis dat jsou vždy systémem obslouženy) a Partition tolerance (odolnost vůči rozpadu sítě – distribuovaný systém funguje, i když se rozpadne na několik izolovaných částí). Jak lze vidět na obrázku 4, v distribuovaném systému nelze zajistit všechny tři vlastnosti na 100 % nýbrž vždy pouze dvě. Databáze typu CA zajišťuje konzistenci a dostupnost napříč všemi uzly, ale nemůže zajistit odolnost proti chybám. Z tohoto důvodu tento typ distribuované databáze nemůže existovat. Příkladem tohoto případu jsou relační databáze. Za to distribuované databáze typu CP a AP již existovat mohou. Databáze typu CP zajišťuje konzistenci a toleranci oddílů na úkor dostupnosti. Databáze typu AP poskytuje dostupnost a toleranci oddílů na úkor konzistence. [13]



Obrázek 4 – CAP teorém, zdroj: vlastní zpracování

Relační databáze využívají pro sestavování odpovědí spojování tabulek a transakční zpracování vlastnosti ACID, které umožňují efektivně realizovat jednotlivé operace zápisu dat, ale sestavování odpovědi z různých tabulek může být velmi náročné. Naopak NoSQL databáze vynaloží více úsilí při návrhu struktury dat a jejich ukládání proto, aby vyhodnocení dotazů bylo rychlejší a systém jich zvládal vyhodnotit více najednou. Alternativou ACID vlastností je model BASE – Basically Available (převážná dostupnost – systém je převážně dostupný po celou dobu, mohou se vyskytnout částečné výpadky, ale nikdy nedojde k výpadku celého systému), Soft state (volný stav – systém je dynamický a nedeterministický, neustále dochází ke změnám) a Eventual Consistency (občasná konzistence – občas je systém uveden do konzistentního stavu), který za cenu nižší míry konzistence poskytne lepší škálovatelnost. [8]

Dotazovací jazyk SQL je v rozporu s NoSQL databázemi, a tak v úplných počátcích NoSQL databáze nenabízely žádné možnosti dotazování. NoSQL databáze nemá určený jednotný dotazovací jazyk, každá NoSQL databáze využívá trochu jiný princip dotazování, který je většinou inspirovaný jazykem SQL nebo jeho rozšířením. [8]

U relačních databází je pevně dané schéma, které určuje strukturu dat, ale u NoSQL databází může být nejednotné a proměnlivé. NoSQL databází je více druhů, které se v některých vlastnostech od sebe velmi liší jako např. ve flexibilitě schématu, kterou má každý druh na jiné úrovni. Některé druhy nemají schéma uložení dat vůbec a některé mají naopak určena přesná pravidla pro rozšiřování schématu. [8]

Mezi hlavní výhody NoSQL databází patří vysoká škálovatelnost, distribuce, nižší cena a flexibilní schéma. Naopak mezi nevýhody patří absence standardizace a omezené dotazování. [10]

Klasifikace NoSQL databází

NoSQL databáze se od sebe velmi liší na mnoha úrovních, a proto se vymezily základní typy NoSQL systémů, které se odlišují v datovém modelu jednotlivých systémů.

Databáze typu klíč-hodnota

NoSQL databáze typu klíč-hodnota mohou ukládat prakticky jakékoliv objekty na základě jejich unikátních klíčů. Operace u tohoto druhu NoSQL databáze jsou

jednoduché, ale manipulovat nebo vyhledávat data lze pouze pomocí určeného klíče a ne podle uloženého obsahu. Příkladem tohoto typu databáze jsou databáze Redis a Riak. [8]

Dokumentové databáze

NoSQL dokumentové databáze ukládají a spravují různé druhy strukturovaných dokumentů, u kterých se předpokládá, že jednotlivé fragmenty dat s sebou nesou informaci o svém významu. Na rozdíl od databáze typu klíč-hodnota umožňují dokumentové databáze vyhledávání podle obsahu. Mezi dokumentové databáze se řadí např. MongoDB a CouchDB. [8]

Sloupcové databáze

NoSQL sloupcové databáze se charakterizují tím, že jejich datový model si lze představit jako tabulku, do které je možné do každého řádku volně přidávat sloupce bez nutnosti přidat je i do ostatních řádků. Sloupcové databáze bývají masivně distribuované. Do této kategorie patří např. Apache Cassandra nebo HBase. [8]

Grafové databáze

NoSQL grafové databáze se výrazně liší od předchozích typů, protože je určen pro data, která je vhodné modelovat a dotazovat jako grafy, tedy objekty a vztahy mezi nimi. Data jsou vnitřně strukturována tak, aby bylo možné efektivně vyhodnocovat různé grafové úlohy. Příkladem grafové databáze je Neo4j a OrientDB. [8]

Cloudové databáze

Cloudová databáze je databázová služba, která je přístupná prostřednictvím cloudové platformy. Tyto databázové služby podporují jak relační databáze, tak i NoSQL databáze. Tento typ databázové služby může být dvojího typu, jako virtuální stroj, který se spravuje jako tradiční databáze (např. MySQL nebo CouchDB), anebo databáze jako služba (DBaaS) (např. Microsoft Azure SQL nebo Google Cloud Bigtable) [14].

3.2.3 Databáze časových řad (TSDB)

Databáze časových řad je specializovaný databázový systém optimalizovaný pro objemná data časových řad a jejich efektivní sběr, ukládání a dotazování. TSDB nemají jasně stanovenou technologii (architekturu), z toho důvodu různé technologie TSDB

poskytují odlišný výkon. Přesto by všechny databáze časových řad měly splňovat požadavky dat časových řad, jako jsou požadavky na náročný zápis, protože se data neustále zvětšují a indexovací algoritmy, které zajišťují nízkou latenci dotazů. [15]

Databáze časových řad se liší od ostatních druhů databázových technologií v architektonickém návrhu, který umožňuje ukládání a kompresi dat s časovým razítkem, správu životního cyklu dat, sumarizaci dat, zpracování velkých skenů mnoha záznamů závislých na časových řadách a dotazy týkající se časových řad. [16] Dále TSDB podporují převzorkování dat pomocí dotazu pro určitý časový úsek (např. hodnoty jsou v časové řadě po sekundovém intervalu a pomocí dotazu lze zjistit průměrnou hodnotu za minutu), porovnání s předchozím záznamem a spojení časových řad pomocí odpovídajících časových razítek. [17]

U databází časových řad se klade velký důraz na rychlost výkonu v reálném čase, a proto často uchovávají data v paměti RAM (Random Access Memory), která je rychlejší oproti pevným diskům a SSD (Solid-State Drive). I přesto je potřeba ve většině aplikací používat pro trvalou formu úložiště pevné disky nebo SSD. [18]

3.3 Přehled databází pro ukládání časových řad

Databází pro ukládání časových řad je mnoho, zde je uveden přehled nejznámějších databází časových řad. V přehledu TSDB jsou vybráni dva zástupci relačních databází (QuestDB a TimescaleDB), dva zástupci distribuovaných databází (InfluxDB a OpenTSDB) a jeden zástupce cloudové databáze jako služby (Amazon Timestream).

3.3.1 QuestDB

Prvním vybraným zástupcem relačních TSDB je open source QuestDB, který je napsaný v jazycích Java a C++. QuestDB používá SQL s rozšířeními pro časové řady. Open source verze je pod licencí Apache 2.0. QuestDB lze nainstalovat jako binární soubory (Windows a Linux) nebo Homebrew (MacOS). Lze ho získat také pro Docker. Přístup k webové konzoli je primárně nastaven přes TCP port 9000. Nejnovější verze systému je 6.4.3. [19]

QuestDB existuje ve více verzích, jako open source, QuestDB Cloud a QuestDB Enterprise. QuestDB Enterprise je řešení pro organizace a lze provozovat v cloudu

na Amazon Web Services (AWS), Microsoft Azure a Google Cloud Platform (GCP) anebo lze provozovat jako on-premise řešení. [20] QuestDB Cloud je dostupné na AWS. [21]

Koncepce databáze

Jelikož se jedná o relační databázi, QuestDB využívá schéma. QuestDB využívá také model úložiště založený na sloupcích, kde jsou data uložena v tabulkách a každý sloupec je uložen ve vlastním souboru a vlastním nativním formátu. Nová data se připojují na konec každého sloupce, aby bylo možné data načíst ve stejném pořadí, jako byla zpracována. Tento model využívá soubory mapované v paměti a atomické aktualizace transakcí mezi procesy. Po odevzdání dat jedním procesem tak může jiný proces data ihned číst. Data lze číst náhodně (prostřednictvím dotazů) nebo přírůstkově (jako frontu dat). [22]

Indexy v QuestDB je možné vytvořit pouze pro sloupce datového typu symbol. Tento datový typ slouží k ukládání opakujících se řetězců. Podpora indexů pro další typy momentálně není možná. Indexování v QuestDB probíhá na principu vytvoření tabulky umístění řádků pro každou odlišnou hodnotu symbolu. Vyhledávání indexovaných hodnot se provádí přímo v indexové tabulce, která poskytne umístění položek viz obrázek 5. Tím se zamezí zbytečnému prohledávání tabulky. [23]

```
Table                                    Index
|Row ID | Symbol    | Value |           | Symbol     | Row IDs       |
| 1     | A         | 1     |           | A          | 1, 2, 4       |
| 2     | A         | 0     |           | B          | 3, 6          |
| 3     | B         | 1     |           | C          | 5             |
| 4     | A         | 1     |
| 5     | C         | 0     |
| 6     | B         | 1     |
```

Obrázek 5 - příklad tabulky a indexovové tabulky [23]

QuestDB přináší rozšíření SQL, kde se stále snaží implementovat ANSI SQL a zároveň kompatibilitu s PostgreSQL. Rozšíření SQL podporuje model ukládání dat QuestDB a zjednodušuje sémantiku analýzy časových řad. Mezi rozšíření SQL patří příkazy LATEST ON pro nalezení nejnovějšího záznamu podle časového razítka pro daný klíč, SAMPLE BY pro časově založené agregace s efektivní syntaxí nebo hledání časového

razítka pomocí běžných operátorů. SQL od QuestDB se také liší od klasického SQL např. ve volitelném použití klauzulí SELECT * FROM a GROUP BY. [24]

Komprese

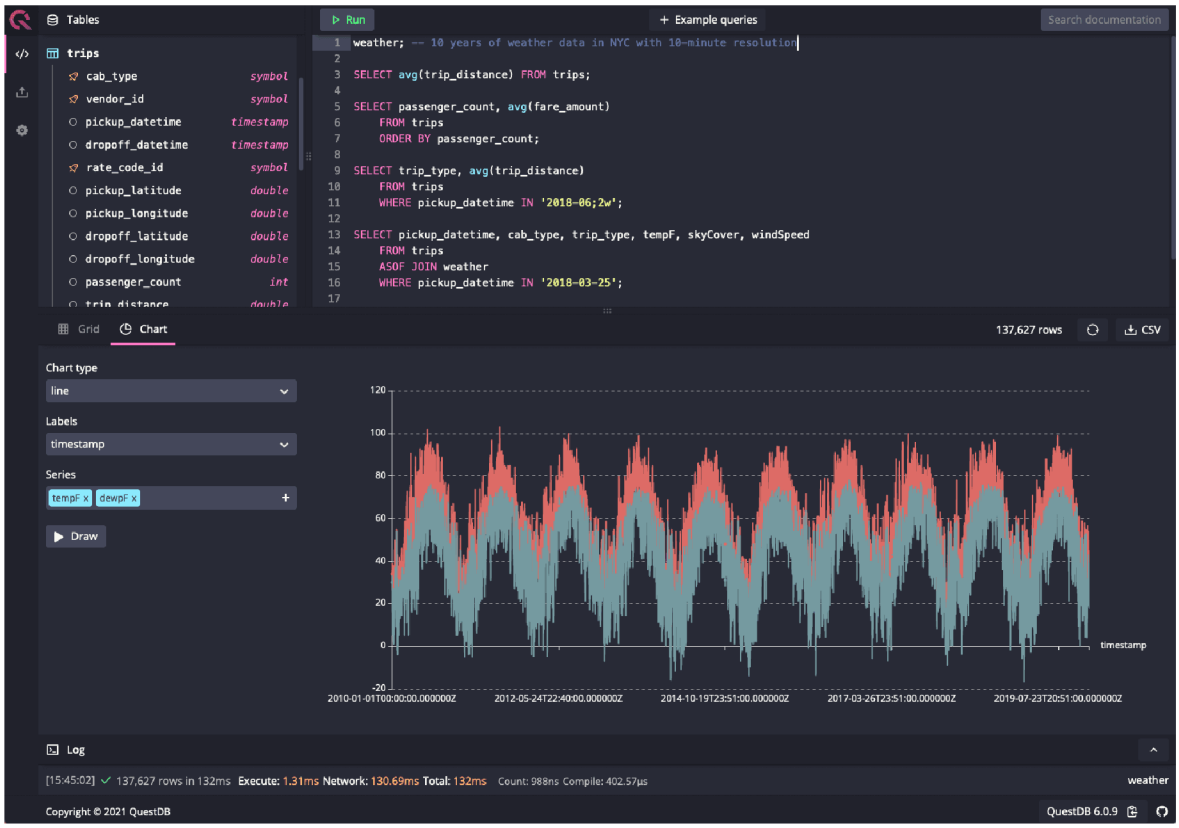
QuestDB v současné době nevyužívá žádnou kompresi dat. Pro snížení prostředků pro zápis a čtení disků se využívají tzv. příčky neboli rozdělení tabulek podle časových intervalů, kde data pro každý interval jsou uložena v samostatných sadách souboru. [25]

Zápis dat

Doporučenou metodou pro vkládání dat do QuestDB je InfluxDB Line Protocol, který dosahuje vysokého výkonu a obsahuje klientské knihovny v různých programovacích jazycích. Druhou možností pro vložení dat je vestavěná webová konzole pro nahrání CSV za pomocí SQL příkazů. Další možností je drátový protokol PostgreSQL, pomocí kterého se lze připojit a dotazovat na různé klientské knihovny a nástroje třetích stran (např. Python, Node.js, .NET, Java, C a C++ a další). Poslední možností je HTTP REST API pro kompatibilitu s širokou škálou knihoven a nástrojů. Při zápisu dat se používají příkazy SQL a také se musí nastavit schéma. [26]

Dotazování

QuestDB využívá pro dotazování dat SQL příkazy. Pro dotazování lze využít webovou konzoli, která je znázorněna na obrázku 6, dále HTTP REST API nebo drátový protokol PostgreSQL. [27]



Obrázek 6 - obrazovka webové konzole QuestDB [28]

Vizualizace dat

Vizualizace dat v QuestDB je možná ve webové konzoli, která je znázorněna na obrázku 6, která podporuje různé typy grafů. Další možností pro vizualizace dat mimo originální nástroj je např. Grafana (více v kapitole 3.4.1). [28]

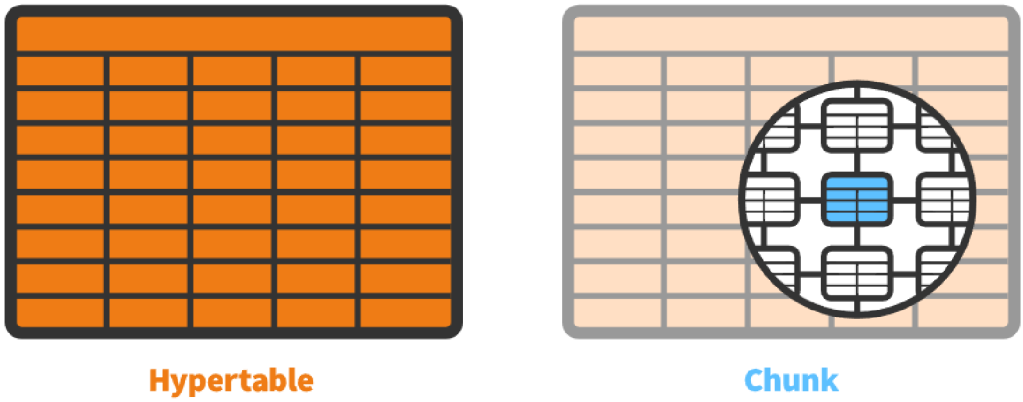
3.3.2 TimescaleDB

Druhým vybraným zástupcem relačních TSDB je open source TimescaleDB od společnosti Timescale, která je implementována jako rozšíření relační databáze PostgreSQL. Celý systém běží v rámci celkové instance PostgreSQL a díky tomu lze využívat mnoho funkcí PostgreSQL. Databáze TimescaleDB je napsána v programovacím jazyce C. TimescaleDB je k dispozici ve 3 verzích. První verze je open source s licencí Apache 2.0, která neobsahuje všechny funkce, druhou verzí je Community pod licencí TSL (Timescale License), která již obsahuje všechny funkce a třetí verzí je placená Enterprise pro komerční využití taktéž s licencí TSL. Nejnovější verze systému je v současnosti 2.7. [29]

TimescaleDB lze využívat jako hostovanou službu Timescale Cloud nebo provozovat na vlastním serveru nebo počítači se systémem Windows, různých Linuxových distribucí nebo MacOS. Poslední možností provozu TimescaleDB je pomocí spravované cloudové služby Managed Service např. přes Amazon Web Services (AWS), Microsoft Azure nebo Google Cloud Platform (GCP). [30]

Koncepce databáze

Koncepce TimescaleDB je založena na hypertabulkách, které poskytují tabulkovou abstrakci, kterou lze dotazovat pomocí standardního SQL. Hypertabulky jsou abstrakce (virtuální pohled) na mnoho jednotlivých tabulek, tzv. bloků (chunks), které jsou implementovány pomocí standardní databázové tabulky. Hypertabulky se rozdělují podle různých sloupců hodnot náležejících ke sloupci času, který může být v časovém razítku, datu nebo různých celočíselných tvarech. Podle časového intervalu rozdělení se jednotlivé záznamy umístí podle časového razítka do příslušných bloků. [31] Hypertabulky a bloky jsou znázorněny na obrázku 7.



Obrázek 7 - vzor hypertabulky a bloku [32]

TimescaleDB vytváří bloky automaticky při vkládání záznamů do databáze. Když má nový záznam časové razítko, které již je v databázi, nový záznam se vloží do odpovídajícího bloku, ale když dané časové razítko ještě v databázi není, vytvoří se nový blok podle daného času časového razítka. Interval rozdělení hypertabulky lze měnit v průběhu času, např. podle pracovní zátěže. [31]

Hypertabulka se dá rozdělit podle více sloupců pomocí hašování. Hypertabulky, které jsou rozděleny jak časem, tak i další dimenzí jsou označovány jako oddíly „čas a prostor“ a používají se pro tzv. distribuované hypertabulky, které jsou rozloženy ve více uzlech. [31]

Při škálování lze TimescaleDB nasadit jako jednu instanci (uzel), primární a záložní replikaci nebo s více uzly s horizontální škálovatelností. Jedna instance PostgreSQL s TimescaleDB využívá princip indexace bloků nejnovější tabulky může zcela uložit do paměti a vyhnout se problému s výměnou dat mezi pamětí a diskem. Primární a záložní replikace představuje primární a replikované databázové servery při streamované replikaci, která může být synchronní a asynchronní. Synchronní replika znamená, že když se na primárním serveru provede nějaká operace se schématem, než systém vrátí výsledek, automaticky tyto změny replikuje na záložní servery. Asynchronní replika funguje na principu, že systém provede zadanou operaci a po výsledku zařadí změny do fronty pro přenos. Primární server TimescaleDB může mít synchronní i asynchronní repliky. Celá tato replikace vychází z PostgreSQL. Ve víceuzlovém nasazení je distribuovaná hypertabulka rozložena mezi více uzlů tak, že každý uzel ukládá pouze část distribuované tabulky. [33]

Komprese

TimescaleDB využívá nativní kompresi, která pomocí vestavěného rámce plánovače úloh převede asynchronní konverzí jednotlivé bloky napříč hypertabulkou z nekomprimované řádkové formy do komprimované sloupcové formy. Při dotazování se data z komprimovaných bloků nejprve dekomprimují do standardní řádkové formy. Nevýhodou je, že k již komprimovaným blokům nelze přidávat nebo aktualizovat data. Komprese se provede na určitém bloku po nějaké době a kompresi lze povolovat, ale i zakazovat na určitých hypertabulkách. [34]

Zápis dat

Při vkládání dat se nejprve musí definovat schéma, které definuje, jak jsou v databázi uspořádány tabulky a indexy. Pro vkládání nových dat je možné použít standardní SQL příkaz INSERT. Pro hromadné nahrávání a migraci lze využít import dat z .csv (Comma-Separated Values), migraci z existující PostgreSQL databáze nebo migraci z jiné databáze časových řad. Pro vkládání dat v reálném čase se dají využít klientské ovladače (Python nebo Node.js) nebo fronty zpráv. [35] Pomocí příkazu UPDATE lze takto vložená data aktualizovat. Mazat data je možné pomocí SQL syntaxe DELETE po řádcích podle určité podmínky nebo lze mazat celé bloky. Je možné nastavení automatického mazání starých dat po nějaké době. [36]

Dotazování

Pro dotazování na data v TimescaleDB se používají standardní příkazy SQL a tzv. hyperfunkce, které umožní rychle provádět kritické dotazy na časové řady, analyzovat data a extrahovat smysluplné informace. Hyperfunkce jsou součástí TimescaleDB, ale pro některé je nutno nainstalovat rozšíření Timescale Toolkit PostgreSQL. [37]

Vizualizace dat

TimescaleDB nemá vlastní nástroj pro vizualizaci dat, ale vizualizace je možná např. v programu Grafana pomocí příslušného pluginu nebo v dalších vizualizačních nástrojích jako např. Tableau, PowerBI, Looker, Periscope, Mode nebo Chartio. [38]

3.3.3 InfluxDB

Prvním vybraným zástupcem distribuovaných TSDB je open source InfluxDB od společnosti InfluxData, která je napsána v programovacím jazyce Go. Open source verze je pod licencí MIT (Massachusetts Institut of Technology). InfluxDB lze nainstalovat na zařízeních s operačními systémy Windows, různé distribuce Linuxu, MacOS nebo v Docker. InfluxDB využívá ve výchozím nastavení pro komunikaci klient-server přes InfluxDB HTTP API port TCP 8086. Nejnovější verze systému je nyní 2.3. [39]

InfluxDB je možné využívat jako DBaaS InfluxDB Cloud přes AWS, Azure nebo GCP. Dále lze využívat jako InfluxDB Open source řešení se základní sadou nástrojů na jednotlivých zařízeních nebo jako předplatné InfluxDB Enterprise pro provozování na vlastní infrastruktuře (např. on-premise řešení, privátní cloud nebo veřejný cloud). [40]

InfluxDB do verze 1.8 byl součástí platformy TICK Stack, která poskytuje ukládání, zaznamenávání, vizualizování a monitorování dat časových řad. Součástí této platformy byly mimo InfluxDB také Telegraf, Chronograf a Kapacitor. [41] Od verze 2.0 obsahuje InfluxDB tyto služby pod svou vlastní platformou.

Koncepce databáze

InfluxDB využívá návrh bez schématu pro lepší správu nespojitých dat. InfluxDB využívá tzv. úložný modul, který zajišťuje, že jsou data bezpečně zapsána na disk a dotazovaná data jsou vrácena úplná a správná. Úložný modul se skládá z protokolu WAL (Write Ahead Log), mezipaměti, datového formátu TSM (Time-Structured Merge Tree)

a indexu časové řady (TSI). Protokol WAL zajišťuje trvanlivost dat v případě nečekaného selhání. V mezipaměti jsou uloženy kopie datových bodů WAL. Datový formát TSM ukládá komprimovaná data ve sloupcovém formátu s tím, že ukládá pouze rozdíly mezi hodnotami v řadě. Princip je takový, že po bezpečném uložení polí v souborech TSM se WAL zkrátí a mezipaměť se vymaže. Indexy časové řady (TSI) ukládají klíče řady seskupené podle měření, značky (tag) a pole (field). [42]

InfluxDB organizuje data časových řad při ukládání dat na disk do fragmentů (tzv. shard) nebo skupin fragmentů (shard group). Fragment obsahuje zakódovaná a komprimovaná data pro konkrétní sadu řad a je složen z jednoho nebo více souborů TSM na disku. Každý fragment obsahuje body s časovými razítky v určitém časovém rozsahu. [43]

Segment (bucket) je pojmenované místo, kde jsou uložena data časové řady. Všechny segmenty mají dobu uchování, která definuje časové období, po kterém se jednotlivá data vymažou. [44] Segmenty mají své schéma, které se dá upravovat. Schéma definuje názvy sloupců, značky, pole a datové typy. [45]

Datový model InfluxDB se skládá z měření (measurement). Značka (tag) je volitelná a reprezentuje dvojici klíč-hodnota v datové struktuře a zaznamenává běžně dostupná metadata. Všechny značky se indexují. Pole (field) reprezentuje také dvojici klíč-hodnota v datové struktuře, ale zaznamenává jak metadata, tak i skutečnou hodnotu dat. Pole obsahuje klíč uložený ve _field sloupci a hodnotu pole ve sloupci _value. Pole nejsou indexována. Dotazy na pole musí nejprve prohledat všechny hodnoty polí, a proto jsou dotazy na značky výkonější než dotazy na pole. InfluxDB podporuje následující datové typy: string, boolean, float, integer a time. [46]

Komprese

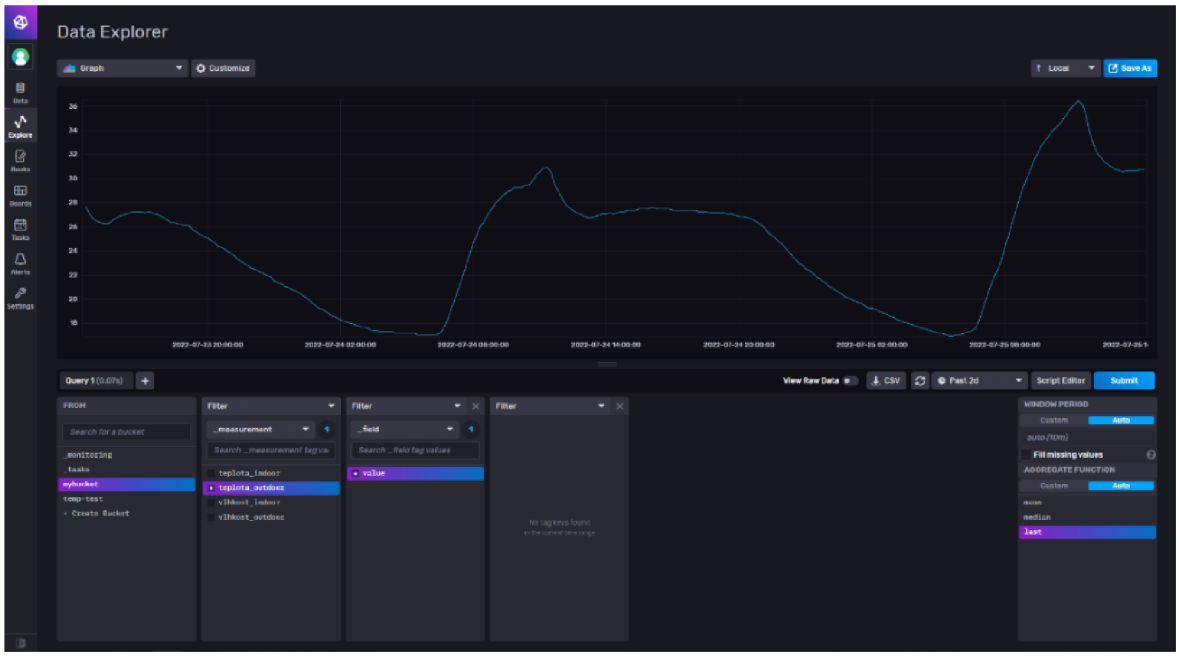
V InfluxDB se komprimují fragmenty v pravidelných intervalech. Pokud jsou povoleny komprimace, InfluxDB každou vteřinu kontroluje, zda jsou komprimace potřeba. Když nedojde k žádnému novému zápisu během pevně stanovené doby, InfluxDB zkomprimuje všechny soubory TSM. V opačném případě se seskupí soubory TSM do tzv. úrovní komprimace, které jsou dány počtem a kolikrát byl soubor komprimován, a následně se pokouší soubory kombinovat a komprimovat efektivněji. Kompresní metoda je vybírána podle datového typu sloupce, přesnosti času a intervalu měření. [43]

Zápis dat

Zápis dat do InfluxDB lze pomocí UI (User Interface – uživatelské rozhraní), Influx CLI (Command Line Interface – příkazový řádek), InfluxDB API (Application Programming Interface) nebo různých pluginů. V uživatelském rozhraní lze načíst CSV data i data pomocí různých klientských knihoven (např. Arduino, C#, Java, Node.js, PHP, Python a mnoho dalších) nebo přes pluginy Telegraf. Data lze také importovat z jiných databází. InfluxDB využívá pro zápis datových bodů textový formát Line Protokol. [47]

Dotazování

Do verze 1.8 se pro dotazování využíval vlastní dotazovací jazyk InfluxQL (Influx Query Language), který byl velmi podobný klasickému SQL. Od verze 2.0 využívá InfluxDB funkční datový skriptovací jazyk Flux určený pro dotazování, analýzu a práci s daty, který podporuje více typů zdrojů dat, např. databáze časových řad (InfluxDB), relační SQL databáze (MySQL nebo PostgreSQL) nebo CSV. [48] Flux sjednocuje kód pro dotazování, zpracování a zápis do jediné syntaxe. Syntaxe obsahuje zdroj, ze kterého se získávají data, filtr, který určuje podmínky výběru (např. časový rozsah), tvar, který definuje funkce pro úpravu struktury a proces pro zpracování dat. InfluxQL lze využít i nyní, ale jelikož využívá starý datový model, k provozu je potřeba systém upravit. Dotazovat lze pomocí UI nebo Influx CLI. [49]



Obrázek 8 - snímek obrazovky UI InfluxDB, zdroj: vlastní zpracování

Vizualizace dat

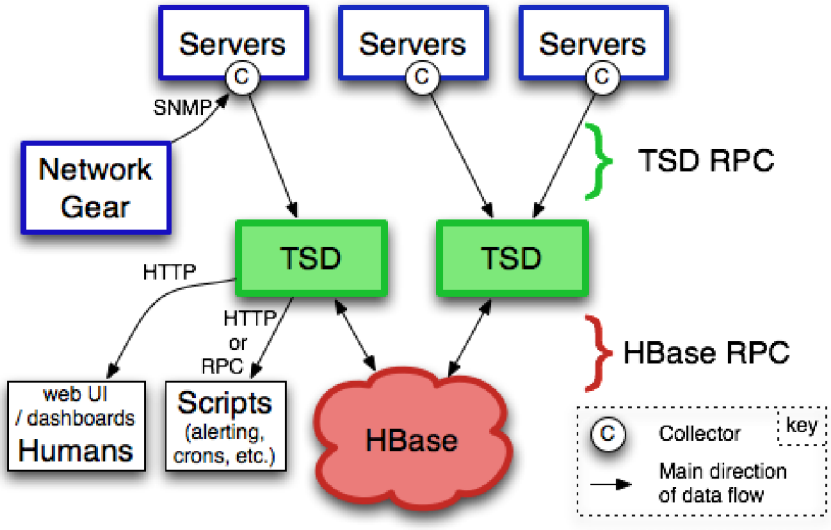
Data časových řad je možné vizualizovat pomocí uživatelského rozhraní InfluxDB, které je zobrazeno na obrázku 8. Toto uživatelské rozhraní podporuje vytváření vlastních panelů pro vizualizaci, např. grafy, měřidla, histogramy a další. Mimo originální nástroj pro vizualizaci InfluxDB podporuje další nástroje třetích stran jako např. Grafana. [50]

3.3.4 OpenTSDB

Druhým vybraným zástupcem distribuovaných TSDB je open source OpenTSDB, která je zaměřená na ukládání a je napsána v programovacím jazyku Java. Databáze je pod licencí LGPLv2.1+ a GPLv3+. OpenTSDB je k dispozici na operační systémy Linux a Windows. OpenTSDB využívá jako hlavní úložiště Apache HBase, ale lze využít také cloud Google Bigtable a Apache Cassandra. Pro spuštění OpenTSDB je potřeba nainstalovat mimo HBase také Javu, Zookeeper a GnuPlot. Aktuální verze systému je 2.4.1, ale probíhá aktivní vývoj verze 3.0.

Koncepce databáze

Koncepce OpenTSDB se skládá z Time Series Daemon (TSD) a sady nástrojů příkazového řádku (CLI). TSD zprostředkovávají zápis a čtení dat a jsou na sobě nezávislí, nesdílejí svůj stav a lze jich spustit několik podle zátěže. Každý TSD využívá k ukládání a načítání dat úložiště Apache HBase, Google Bigtable nebo Apache Cassandra. [51] Celá koncepce je znázorněna na obrázku 9.



Obrázek 9 - koncepce OpenTSDB [51]

„Datové schéma je vysoce optimalizováno pro rychlé agregace podobných časových řad, aby se minimalizoval úložný prostor.“ [51]

OpenTSDB obsahuje velké datové tabulky s názvem tsdb pro uložení všech datových bodů, menší tabulky UID s názvem tsdb-uid pro uložení mapování UID (metrika, název značky nebo hodnota značky) a případně metadat. Dále obsahuje meta tabulky, které jsou indexem různých časových řad a obsahují metadata pro každou řadu, stromové tabulky, které se chovají jako index a organizují časové řady do hierarchické struktury a souhrnné tabulky pro uložení předem agregovaných skupin pro zvýšení rychlosti dotazování. [52]

OpenTSDB využívá stejně jako InfluxDB značky (tag), které znázorňují dvojici klíč-hodnota. Každá časová řada v OpenTSDB musí mít alespoň jednu značku. V OpenTSDB jsou data tvořeny časovým razítkem, hodnotou a značkami. [53]

Komprese

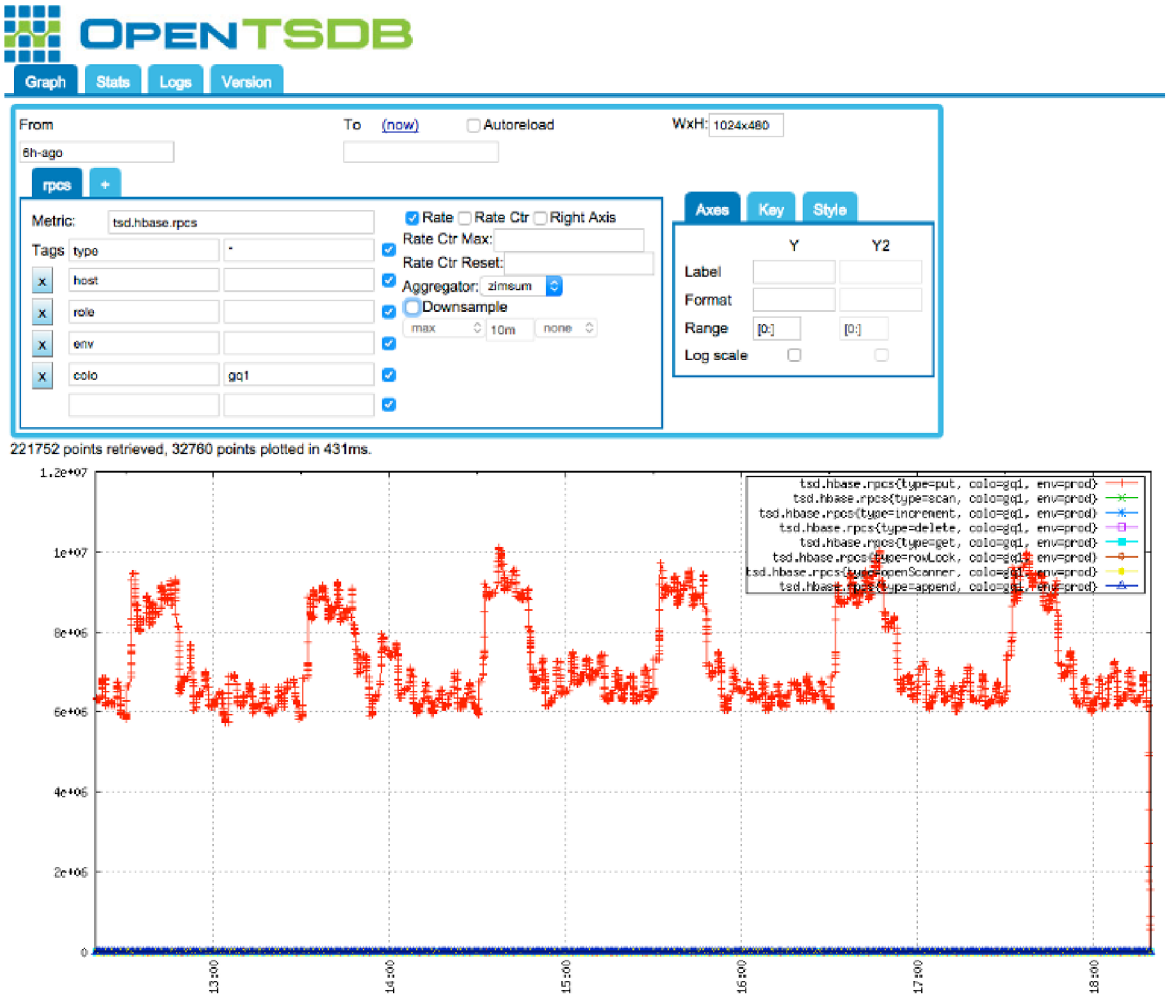
Komprese je dvojího typu, na úrovni OpenTSDB a HBase. V OpenTSDB je komprese primárně vypnutá, ale pro produkční prostředí se doporučuje. OpenTSDB podporuje kompresi typu LZO (Lempel-Ziv-Oberhumer) a Snappy. [54]

Zápis dat

Při vkládání dat není zapotřebí vytvářet schéma. Data lze vložit do OpenTSDB třemi hlavními způsoby, pomocí Telnet API, HTTP API nebo dávkového importu ze souboru přes CLI. Dalším způsobem vložením dat jsou knihovny třetích stran. [53]

Dotazování

Dotazovat na data lze v OpenTSDB pomocí nástrojů CLI a HTTP API. Původní syntaxe umožňovala jednoduché filtrování, agregaci a převzorkování. V dalších verzích se přidala podpora funkcí a výrazů. [55]



Obrázek 10 - GUI OpenTSDB [56]

Vizualizace dat

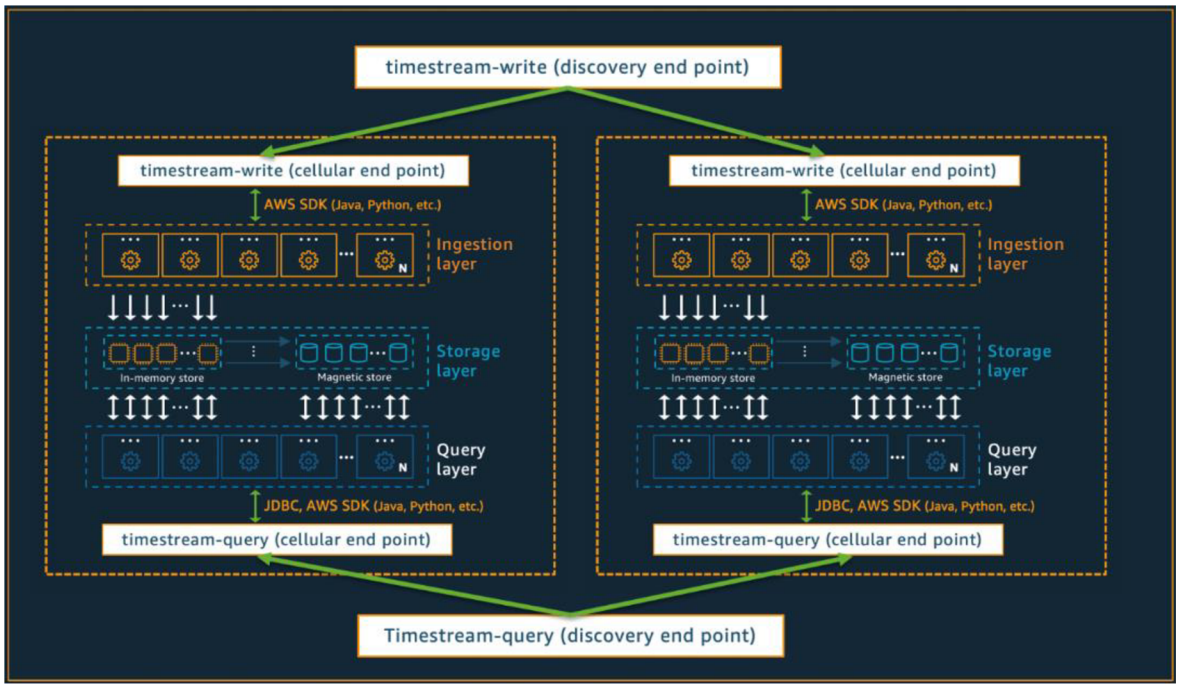
OpenTSDB nabízí jednoduché vestavěné GUI (Graphical User Interface) pro vizualizaci dat prostřednictvím webového rozhraní na portu 4242, které je znázorněno na obrázku 10. V tomto GUI lze zobrazit grafy, statistiky a protokoly. Na grafy se odkazuje dotazováním. Dále OpenTSDB podporuje program pro vizualizací třetích stran jako je Grafana. [56]

3.3.5 Amazon Timestream

Amazon Timestream je vybraným zástupcem komerčních DBaaS pro časové řady, která je spravována v cloudové službě Amazon Web Services (AWS). Jedná se o tzv. bezserverové řešení. U Amazonu Timestream neexistuje žádná předem určená cena za používání, ale ceník se odvíjí od používaných funkcí a množství dat.

Koncepce databáze

Amazon Timestream využívá tzv. buněčnou architekturu. Systém Timestream se segmentuje do několika menších kopií, tzv. buněk. Problém systému v jedné buňce nijak neovlivní činnost v ostatních buňkách. Každá buňka se skládá ze tří vrstev – vkládací vrstvy, vrstvy úložiště a dotazovací vrstvy, jak lze vidět na obrázku 11. Tyto vrstvy se škálují nezávisle. [57]



Obrázek 11 - schéma rozdělení buněk Timestream [57]

Při vkládání dat je nutné schéma rozdělení, které rozdělí data podle času i prostoru. Při zápisu dat Timestream před uložením dat do úložiště automaticky data indexuje a rozděluje. Data časových řad se rozdělí do oddílů (dlaždic) s podobnou velikostí, které představují rozdělení dvourozměrného prostoru. [57]

Amazon Timestream nabízí dvě datová úložiště – úložiště v paměti a nákladově efektivní magnetické úložiště a podporuje konfiguraci zásad pro automatické přenosy dat mezi úložišti. Data s vysokou propustností (tzv. nedávná data) jsou uložena v paměťovém úložišti pro přímý přístup pro dotazy. Historická data (data s časovou značkou starší, než je aktuální čas) se zapisují do magnetického úložiště. Data v paměťovém úložišti se po uplynutí doby uchování v paměťovém úložišti automaticky přesunou do magnetického úložiště. Data v magnetickém úložišti jsou reorganizována do vhodného formátu pro čtení

velkých objemů dat. Timestream umožňuje nastavit podle potřeby zásady uchování dat v paměťovém a magnetickém úložišti. [57]

Amazon Timestream využívá pro dotazování nad daty rozšířené SQL pro časové řady s podporou datových typů a specifických funkcí pro časové řady. Dotazy jsou zpracovávány adaptivním distribuovaným dotazovacím systémem, který využívá metadata a indexování dlaždic k přístupu k datům napříč datovým úložištím. [57]

Komprese

V dokumentaci Amazonu Timestream je pouze zmínka, že data jsou komprimována, ale více informací nenabízí viz:

„Amazon Timestream nezveřejňuje žádné informace o tom, jak jsou data uložena v konzole, ani jinak netušíme, jak dobře jsou tato data komprimována…“ [58]

Zápis dat

Data lze vkládat do Amazon Timestream pomocí sad AWS SDK (které obsahují klientské knihovny Java, Go, Python, Node.js nebo .Net), AWS CLI nebo přes AWS Lambda, AWS IoT Core, Amazon Kinesis Data Analytics for Apache Flink, Amazon Kinesis, Amazon MSK nebo Telegraf. Před vkládáním dat do Amazonu Timestream je nutné vytvořit databázi a tabulku pomocí AWS Management Console, AWS SDK nebo Timestream API. [59]

Dotazování

Pro dotazování se využívá upravený SQL. Dotazování na data lze přes AWS Management Console, AWS CLI, Timestream API nebo AWS SDK. Pomocí SQL lze provádět analytické funkce časových řad pro interpolaci, regresi nebo vyhlazování. [60] Timestream dále podporuje naplánované dotazy pro analytické dotazy v reálném čase, které počítají např. agregace, souhrny a další operace s daty. [61]

Vizualizace dat

Data časových řad z Amazonu Timestream lze vizualizovat v programech Amazon QuickSight nebo Grafana. [62]

3.3.6 Shrnutí přehledu databází pro ukládání časových řad

Pro porovnání databází pro ukládání časových řad byly vybrány open source databáze QuestDB, TimescaleDB, InfluxDB a OpenTSDB s kritérii přesnost, podporované instalace, podporované programovací jazyky, správa uživatelů, podpora XML, možnost uchovávání v paměti a vlastní vizualizace. Shrnutí podle těchto kritérií je znázorněno v tabulce 1. Samotné srovnání je v kapitole 4.1.

	QuestDB	**TimescaleDB**	**InfluxDB**	**OpenTSDB**
Přesnost	µs	ms	ns	ms
Podporované instalace	Linux, Windows, MacOS, Docker, Kubernetes, Any (no JVM), Maven, Gradle	Linux, Windows, MacOS, Docker, Kubernetes	Linux, Windows, MacOS, Docker, Kubernetes	Linux, Windows, Docker, Kubernetes
Podporované programovací jazyky	C, C++, Go, Java, Node.js, Python, Rust	.Net, C, C++, Delphi, Java, JavaScript, Perl, PHP, Python, R, Ruby, Scheme, Tcl	.Net, Clojure, Erlang, Go, Haskell, Java, JavaScript, Node.js, Lisp, Perl, PHP, Python, R, Ruby, Rust, Scala	Erlang, Go, Java, Python, R, Ruby
Správa uživatelů	NE	ANO	ANO	NE
Podpora XML	NE	ANO	NE	NE
Možnost uchovávání v paměti	ANO	NE	ANO	NE
Vlastní vizualizace	ANO	NE	ANO	ANO

Tabulka 1 – porovnání open source databází, zdroj: vlastní zpracování

3.4 Další využité programy

V experimentech této práce bude dále využit vizualizační nástroj Grafana a programovací nástroj Node-RED, které jsou charakterizovány v této podkapitole.

3.4.1 Grafana

Grafana je open source program pro dotazování, vizualizaci a monitorování dat. Prostřednictvím Grafany je možné si nastavit zasílání upozornění v reálném čase prostřednictvím např. SMS, e-mailu nebo Slack. Pomocí Grafany lze vizualizovat grafy, histogramy, tepelné mapy, měřidla a mnoho dalších variací. Do vytvořených grafů je možné také přidávat anotace. [63]

Grafana je k dispozici jako Grafana Cloud s verzemi zdarma se základními funkcemi a až pro 3 uživatele, dále verze Pro s pokročilejšími funkcemi a pouze pro 1 uživatele a poslední verze Pokročilé pro týmy nebo jako Enterprise Stack pro podniky. [64] Grafana lze provozovat na operačních systémech Linux, MacOS, Windows nebo pomocí Dockeru. Nejnovější verze systému je 9.0.5. [65]

Grafana podporuje různé zdroje dat jako např. Prometheus, InfluxDB, PostgreSQL a mnoho dalších. Od verze 3.0 je možné instalovat zdroje dat jako pluginy. Podle zvoleného zdroje dat Grafana využívá specifické editory dotazů přizpůsobené funkcím a možnostem daného zdroje. [66]

Základním stavebním kamenem vizualizace v Grafaně je panel, který má editor dotazů specifický pro vybraný zdroj dat, který umožňuje vizualizaci se širokou škálou formátování. [67] Celé GUI Grafana je zobrazeno na obrázku 12 na straně 38.

Obrázek 12 - GUI Grafana [68]

3.4.2 Node-RED

Node-RED je open source programovací nástroj pro propojení hardwarových zařízení, API a online služeb založený na Flow (v češtině tok, v práci je dále používán anglický termín flow, jelikož český ekvivalent se v komunitě nepoužívá), který vznikl v roce 2013 ve společnosti IBM. V současnosti je součástí OpenJS Foundation. V současné době je k dispozici verze programu 3.0.1. [69]

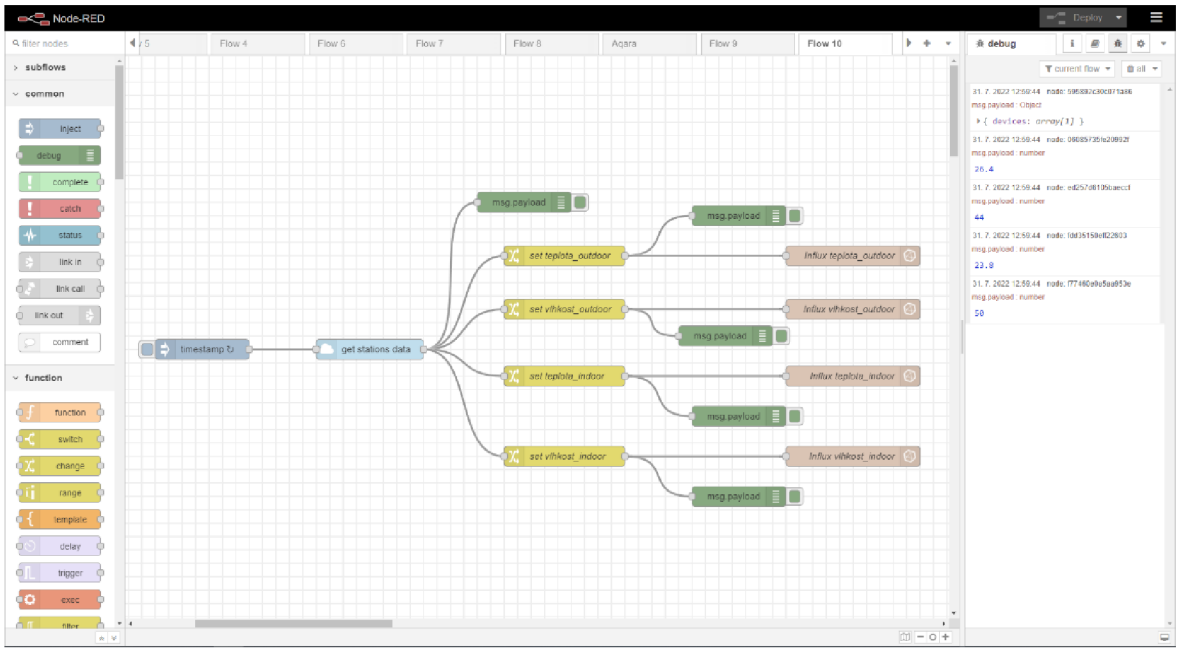
Node-RED lze provozovat lokálně na Debianu (Ubuntu nebo Diet-Pi), RPM (RedHat, Fedora nebo CentOS) a Windows. Node-RED lze spustit také pomocí Dockeru nebo na Android pomocí aplikace Termux a cloudových služeb IBM Cloud, AWS nebo Microsoft Azure. [70]

Programování založené na flow obsahuje síť uzlů (Nodes), které mají definované účely. Každý uzel dostane nějaká data, něco s těmi daty udělá a poté je předá dál. Tento model se využívá k vizuálnímu znázornění a zpřístupňuje jej širšímu okruhu uživatelů. [69]

Nástroj Node-RED poskytuje editor založený na prohlížeči. Editor obsahuje v paletě širokou škálu uzlů, která lze spravovat a instalovat nové uzly. V editoru lze pomocí editoru

formátovaného textu vytvářet funkce JavaScriptu. Node-RED využívá událostmi řízený, neblokující model Node.js. Vytvořené flow lze uložit do souboru JSON. [71]

Mezi hlavní koncepty Node-RED patří uzel (node), který je stavebním kamenem flow a je spuštěn přijetím zprávy nebo čekáním na nějakou externí událost. Po obdržení zprávy nebo události je zpracují a následně odesílají zprávu dalším uzlům ve flow. Uzel má jen jeden vstupní port a několik výstupních portů. Konfigurační (config) uzel je speciální typ uzlu, který obsahuje opakovaně použitelnou konfiguraci, kterou mohou sdílet běžné uzly ve flow. Flow je hlavní způsob organizace uzlů nebo může být reprezentován jako jedna sada připojených uzlů. Zpráva je objekt JavaScriptu, který prochází mezi uzly ve flow a v editoru jsou často označovány jako msg. Podle konvence mají zprávy vlastnost payload, která obsahuje nejužitečnější informace. Uzly se spojují dráty, které představují, jak zprávy procházejí flow. [72] Na obrázku 13 je znázorněn editor Node-RED.



Obrázek 13 - editor Node-RED, zdroj: vlastní zpracování

3.5 Modely vícekriteriálního rozhodování

V této práci byla využita pro komparaci databází pro uložení časových řad metoda vícekriteriálního rozhodování, která je daná konečnou (diskrétní) množinou m variant, které jsou hodnoceny podle n kritérií. [73]

Kritéria lze rozdělit podle několika hledisek. Rozdělení kritérií podle povahy na maximalizační kritéria (nejlepší varianty mají nejvyšší hodnoty) a minimalizační kritéria (nejlepší varianty mají nejnižší hodnoty). Další možností rozdělení kritérií je podle kvantifikovatelnosti na kvantitativní kritéria, které obsahují hodnoty variant, které tvoří objektivně měřitelné údaje a kvalitativní kritéria, jejichž hodnoty variant nelze objektivně změřit. [73]

Preference kritéria vyjadřuje důležitost daného kritéria oproti ostatním. Preference kritéria lze vyjádřit různým způsobem. Aspirační úrovně kritérií znázorňují hodnoty, kterých má být alespoň dosaženo. Pořadí kritérií vyjadřuje posloupnost kritérií od nejdůležitějšího po nejméně důležité. Váhy kritérií jsou v intervalu <0;1> a vyjadřují důležitost jednotlivých kritérií. Způsob kompenzace kriteriálních hodnot je stanoven mírou substituce mezi kriteriálními hodnotami. Kritéria ale nemusí mít také preference žádné. [73]

Úlohy vícekriteriální analýzy je možné dělit podle typu informace, která je známa o preferencích mezi kritérii a variantami. Informace o preferenci mohou být nominální (rozděluje varianty na akceptovatelné a neakceptovatelné), ordinální (vyjadřuje uspořádání podle důležitosti), kardinální (vyjadřuje o kolik či jak moc je jedno hodnocení lepší než druhé) nebo žádné. [73]

Pro stanovení vah kritérií se používají pro úlohy s žádnou informací o preferenci entropická metoda, pro úlohy s nominální informací o preferenci metoda aspiračních úrovní, pro úlohy s ordinální informací o preferenci metoda pořadí a Fullerova metoda a nakonec pro úlohy s kardinální informací bodovací metoda a Saatyho metoda. [73]

Pro výběr kompromisní varianty u žádné informace o preferenci kritérií se používají metody bodovací a pořadí, u aspirační úrovně kritérií se využívají konjunktivní a disjunktivní metody a metoda bazické varianty, u ordinální informace se využívá lexikografická metoda a u kardinální informace se využívají funkce užitku, metoda váženého součtu, metoda AHP (Analytic Hierarchy Process) a konstrukce hierarchické struktury problému. [73]

Ve vlastní práci v kapitole 4.1 byla použita Saatyho metoda pro stanovení vah z kardinální informace o preferencích kritérií. Tato metoda je vhodná, pokud váhy určuje pouze jeden expert. Jedná se o metodu kvantitativního párového porovnávání kritérií, při které se používá devítibodová stupnice.

Hlavními stupni jsou:

1 – rovnocenná kritéria *i* a *j*

3 – slabě preferované kritérium *i* před *j*

5 – silně preferované kritérium *i* přej *j*

7 – velmi silně preferované kritérium *i* před *j*

9 – absolutně preferované kritérium *i* přej *j* [73]

V případě nutnosti lze použít i mezistupně (hodnoty 2, 4, 6, 8). Porovnává se vždy každá dvojice kritérií a velikost preference *i*-tého kritéria vzhledem k *j*-tému kritériu se zapisuje do Saatyho matice S = (s_{ij}), která je čtvercová a platí pro ni, že $s_{ij} = 1/s_{ji}$. Na diagonále jsou vždy hodnoty jedna. Je-li preferováno *j*-té kritérium před *i*-tým, zapíše se do Saatyho matice převrácená hodnota (např. s_{ij} = 1/3 při slabé preferenci). Saatyho matice musí být dostatečně konzistentní, přičemž konzistence se měří indexem konzistence, který má vzorec:

$$I_S = \frac{l_{max}-n}{n-1}, \tag{3}$$

kde l_{max} je největší vlastní číslo Saatyho matice a *n* je počet kritérií. Saatyho matice se považuje za dostatečně konzistentní, pokud je index konzistence menší než 0,1

Existuje více způsobů, pomocí kterých lze odhadnout váhy v_j, ale nejčastěji se využívá postup výpočtu vah jako normalizovaného geometrického průměru řádků Saatyho matice neboli metoda logaritmických nejmenších čtverců. K tomuto postupu se nejprve vypočte geometrický průměr řádků Saatyho matice b_i pomocí vzorce:

$$b_i = \sqrt[n]{\prod_{j=1}^{n} s_{ij}} \tag{4}$$

a následně se váhy v_i vypočtou normalizací hodnot b_i pomocí vzorce:

$$v_i = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{5}$$

Ve vlastní práci v kapitole 4.1 je také použita pro výběr kompromisní varianty metoda váženého součtu. Tato metoda vyžaduje kardinální informace, kriteriální matici *Y* a vektor

vah kritérií *v*. Vychází z principu maximalizace užitku a konstruuje celkové hodnocení pro každou variantu. Celkový užitek varianty $u(a_i)$ je vyjádřen váženým součtem hodnot dílčích funkcí užitku:

$$u(a_i) = \sum_{i=1}^{m} v_j \,.\, u_j(y_{ij}), \tag{6}$$

kde v_j jsou váhy kritérií a u_j jsou dílčí funkce užitku jednotlivých kritérií.

Výpočet této metody je ve třech hlavních krocích. V prvním kroku se určí ideální varianta *H* s ohodnocením $(h_1, \ldots, h_n)$, a bazální varianta *D* s ohodnocením $(d_1, \ldots, d_n)$. Ve druhém kroku se vytvoří standardizovaná kriteriální matice R, jejíž prvky se získají pomocí vzorce:

$$r_{ij} = \frac{y_{ij} - d_j}{h_j - d_j}, \tag{7}$$

kde y_{ij} je hodnota varianty a_i podle kritéria *j*, d_j je hodnota bazální varianty D a h_j je varianta ideální varianty H. V této kriteriální matici již odpovídá bazální variantě hodnota nula a ideální variantě hodnota jedna. Ve třetím kroku se pro jednotlivé varianty vypočte agregovaná funkce užitku. Varianta s nejvyšší hodnotou agregovaného užitku se považuje za kompromisní variantu. Vzorec agregovaného užitku $u(a_i)$:

$$u(a_i) = \sum_{j=1}^{n} v_j r_{ij}, \tag{8}$$

kde v_j jsou váhy kritérií a r_{ij} jsou hodnoty standardizované kriteriální matice R.

3.6 Shrnutí

V teoretické části práce byly nejdříve charakterizovány časové řady a jejich klasifikace, dekompozice, úpravy a predikce. V následující části byly charakterizovány druhy databázových technologií pro ukládání časových řad, konkrétně relační databáze, NoSQL databáze a databáze časových řad. Dále byl vytvořen přehled nejpoužívanějších databází časových řad se dvěma zástupci relačních databází (QuestDB a TimescaleDB), dvěma zástupci NoSQL databází (InfluxDB a OpenTSDB) a jedním zástupcem cloudových

databází (Amazon Timestream). U těchto databází byla charakterizována jejich koncepce, komprese, zápis dat, dotazování a vizualizace dat. Z tohoto přehledu byly vybrány open-source databáze QuestDB, TimescaleDB, InfluxDB a OpenTSDB a 7 kritérií pro porovnání v praktické části práce. Následně byly vyznačeny programy Grafana a Node-RED, které byly využity při experimentech. Nakonec byly charakterizovány modely vícekriteriálního rozhodování.

4 Vlastní práce

V následujících podkapitolách jsou porovnávány open source databáze pro ukládání časových řad pomocí vícekriteriálního rozhodování, ze kterých byla následně vybrána nejvhodnější databáze, na které byly prováděny experimenty. Dále jsou ve vlastní práci charakterizovány použitá experimentální data, použitá databáze časových řad InfluxDB a další pomocné aplikace, které byly použity při experimentech a experimentální pracoviště, na kterých byly provedeny experimenty. V poslední podkapitole jsou charakterizovány jednotlivé experimenty nad databází časových řad InfluxDB.

4.1 Porovnání vybraných databází pro ukládání časových řad

V této kapitole jsou porovnávány databáze pro ukládání časových řad pomocí vícekriteriálního rozhodování viz tabulka 1 v kapitole 3.3.6 – Shrnutí. Všechny porovnávané databáze jsou open source, konkrétně se jedná o QuestDB, TimescaleDB, InfluxDB a OpenTSDB. Tyto databáze jsou porovnávány podle sedmi kritérií. Tato kritéria jsou pro přehlednost v následujících tabulkách přejmenována podle legendy v tabulce 2.

Celý název	**Zkrácený název**
Přesnost	K1
Podporované instalace	K2
Podporované programovací jazyky	K3
Správa uživatelů	K4
Podpora XML	K5
Možnost uchovávání v paměti	K6
Vlastní vizualizace	K7

Tabulka 2 – legenda názvu kritérií, zdroj: vlastní zpracování

Kritéria se rozdělují na dva základní typy, na kvantitativní a kvalitativní. Mezi kvantitativní kritéria se řadí přesnost, podporované instalace, podporované programovací jazyky a mezi kvalitativní kritéria se řadí správa uživatelů, podpora XML, možnost uchovávání v paměti a vlastní vizualizace.

Kritérium přesnost bylo pro lepší představitelnost ohodnocení jednotlivých variant klasifikováno počtem bodů podle práce s desetinnými řády a kritéria podporované instalace a podporované programovací jazyky byly ohodnoceny počtem bodů podle počtu kusů. Kvalitativní kritéria byla ohodnocena tak, že ANO bylo ohodnoceno jako jedna, NE jako nula. Všechna kritéria jsou maximalizační. Výsledné ohodnocení je znázorněno v tabulce 3.

	K1	K2	K3	K4	K5	K6	K7
QuestDB	6	8	7	0	0	1	1
TimescaleDB	3	5	13	1	1	0	0
InfluxDB	9	5	16	1	0	1	1
OpenTSDB	3	4	6	0	0	0	1
Charakter kritéria	max	max	max	max	max	max	max

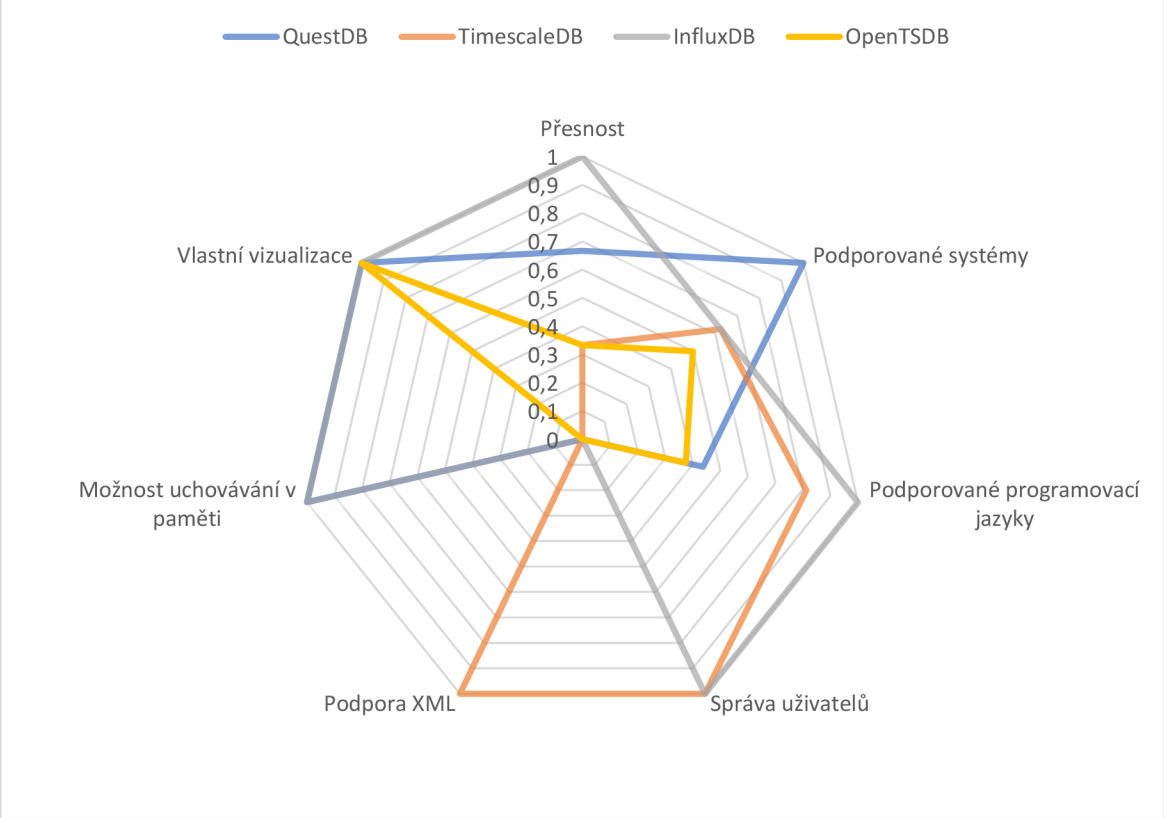
Tabulka 3 – kriteriální matice po kvantifikaci a určení charakteru kritérií, zdroj: vlastní zpracování

Pro lepší představu o posouzení dominancí některých variant byla všechna ohodnocení převedena na stejnou škálu v rozmezí od nuly do jedné, které je znázorněno v tabulce 4.

	K1	K2	K3	K4	K5	K6	K7
QuestDB	0,66	1	0,44	0	0	1	1
TimescaleDB	0,33	0,63	0,81	1	1	0	0
InfluxDB	1	0,63	1	1	0	1	1
OpenTSDB	0,33	0,5	0,38	0	0	0	1
Charakter kritéria	max	max	max	max	max	max	max

Tabulka 4 – kriteriální matice po převodu ohodnocení na stejnou škálu, zdroj: vlastní zpracování

V grafu 1 na straně 46 je znázorněn paprskový graf variant podle tabulky 4. Z tabulky 4 a grafu 1 je patrné, že varianty QuestDB a InfluxDB dominují variantu OpenTSDB. S touto variantou se tedy již dále nebude počítat.



Graf 1 – paprskový graf variant, zdroj: vlastní zpracování

Po odebrání varianty OpenTSDB vyšla kritéria možnost uchovávání v paměti a vlastní vizualizace pro všechny varianty stejně, ale byla ponechána obě kritéria, protože ve skutečnosti se jedná o dvě rozdílná kritéria, která nejsou stejná a mají rozlišovací schopnost.

4.1.1 Saatyho metoda pro stanovení vah

Pro stanovení vah byla využita Saatyho metoda, kde jsou v tabulce 5 na straně 46 všechna kritéria vzájemně porovnána a zapsána jejich preference. Index konzistence této Saatyho matice vyšel podle vzorce 3 0,0948. Jelikož je index konzistence menší než 0,1, Saatyho matice se považuje za dostatečně konzistentní. Následně byly vypočteny geometrické průměry řádků podle vzorce 4 a jednotlivé váhy podle vzorce 5, které jsou uvedeny v tabulce 5 na straně 47.

	K1	K2	K3	K4	K5	K6	K7	Geom. průměr	Váha (preference)
K1	1	3	3	5	3	3	5	2,967197	0,32
K2	1/3	1	1	5	1	1/3	5	1,157139	0,13
K3	1/3	1	1	5	1	1/3	5	1,157139	0,13
K4	1/5	1/5	1/5	1	1/5	1/5	3	0,370592	0,04
K5	1/3	1	1	5	1	1/3	5	1,157139	0,13
K6	1/3	3	3	5	3	1	5	2,167834	0,23
K7	1/5	1/5	1/5	1/3	1/5	1/5	1	0,270754	0,03

Tabulka 5 – Saatyho matice, zdroj: vlastní zpracování

4.1.2 Metoda váženého součtu pro výběr kompromisní varianty

Pro výběr kompromisní varianty byla využita metoda váženého součtu, při které byly do kriteriální matice připsány zjištěné váhy jednotlivých kritérií a byla zjištěna ideální a bazální varianta. Viz tabulka 6.

	K1	K2	K3	K4	K5	K6	K7
QuestDB	6	8	7	0	0	1	1
TimescaleDB	3	5	13	1	1	0	0
InfluxDB	9	5	16	1	0	1	1
Charakter kritéria	max	max	max	max	max	max	max
Váhy	0,32	0,13	0,13	0,04	0,13	0,23	0,03
Ideální varianta – H	9	8	16	1	1	1	1
Bazální varianta – D	3	5	7	0	0	0	0

Tabulka 6 – kriteriální matice s váhami a ideální a bazální variantou, zdroj: vlastní zpracování

Následně byla za pomocí vzorce 7 vytvořena standardizovaná matice, která je znázorněna v tabulce 7. Nakonec byl vypočten podle vzorce 8 agregovaný užitek pro každou variantu, který vyšel nejvýhodněji pro variantu databáze InfluxDB.

	K1	K2	K3	K4	K5	K6	K7	Agregovaný užitek	Pořadí
QuestDB	0,50	1	0	0	0	1	1	0,549247	2.
TimescaleDB	0	0	0,67	1	1	0	0	0,248617	3.
InfluxDB	1	0	1	1	0	1	1	0,749748	1.

Tabulka 7 – standardizovaná matice s agregovaným užitkem a pořadí, zdroj: vlastní zpracování

Výsledkem tohoto vícekriteriálního rozhodování bylo zjištěno, že jako kompromisní neboli nejvhodnější databáze časových řad podle zvolených kritérií je InfluxDB. S touto databází byly prováděny experimenty v dalších kapitolách.

4.2 Experimentální data

Pro experimenty s vybranou databází InfluxDB byla vybrána veřejně dostupná senzorická data pražské hromadné dopravy v reálném čase z REST API pražské datové platformy Golemio [74], jejichž vydavatel je ROPID (Regionální organizátor pražské integrované dopravy). Data z této platformy obsahují aktuální informace o polohách a zpožděních všech vozidel v rámci PID (Pražské integrované dopravy) pomocí jejich GPS. Každé vozidlo odesílá data s aktuálními souřadnicemi, zpoždění, minulou a následující stanicí, číslem linky a vozu a další.

Data z této REST API jsou aktualizována každých 10 sekund a obsahují časovou značku s přesností na sekundy. Počet zpráv se v průběhu dne a týdne mění podle vytíženosti dopravy. Ve všední den se počet zpráv pohybuje kolem 1 400–1 600 a o víkendu kolem 700 při každé aktualizaci. Pro získání dat z REST API byl nutný klíč. Pro jeho získání bylo potřeba se registrovat na webových stránkách REST API Golemio [75], kde byl následně vygenerován klíč.

Pro potřeby experimentů byla ze všech dat vybrána pouze ta, která jsou uvedena v tabulce 8 na straně 49.

Název	Popis	Datový typ
time	Datum a čas pořízení dat	dateTime
delay	Aktuální zpoždění vozu	double
delay_last	Zpoždění vozu z předchozí zastávky	double
position	Aktuální pozice vozu, jestli je v zastávce nebo na cestě	string
last_stop	Kód předchozí zastávky	string
last_stop_order	Pořadí předchozí zastávky	double
next_stop	Kód následující zastávky	string
next_stop_order	Pořadí následující zastávky	double
line	Název linky	string
vehicle	Název vozu	string
type	Typ dopravy, např. tramvaj nebo autobus	string
final_stop	Název konečné stanice vozu	string

Tabulka 8 - datový slovník, zdroj: vlastní zpracování

4.3 InfluxDB a pomocné aplikace

Při experimentech byla využita databáze časových řad InfluxDB OSS verze 2.6.1, Node-RED verze 3.0.2 pro přípravu a odesílání dat, Telegraf verze 1.25.1 pro získávání metrik a Grafana verze 9.3.6 pro vizualizaci dat.

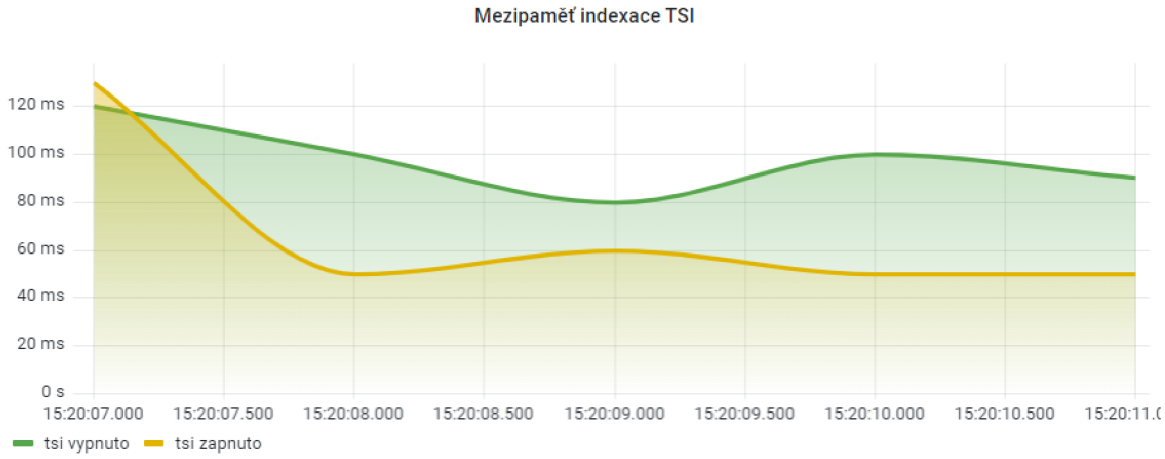
4.3.1 InfluxDB

Pro produkční využití databáze InfluxDB vývojáři společnosti InfluxData vytvořili speciální placenou verzi databáze InfluxDB Enterprise, která je založená na clusteru na vlastní infrastruktuře. Ovšem lze využít také bezplatnou open source verzi InfluxDB OSS (open source system), která byla využita pro experimenty.

Při instalaci samotné databáze InfluxDB bylo v konfiguračním souboru config.toml, nacházející se ve složce /etc/influxdb, zakázáno odesílání telemetrických dat do InfluxData a prodloužena délka relace na 4 hodiny. Bohužel vypnutí autentizace, která byla možná ve starší verzi 1.x již ve verzi 2.x není. Celý obsah konfiguračního souboru config.toml je zobrazen na straně 50.

```
bolt-path = "/var/lib/influxdb/influxdb.bolt"
engine-path = "/var/lib/influxdb/engine"
reporting-disabled = true
session-lenght = 240
```

V InfluxDB verze 1.x byly dvě možnosti indexování; do paměti, která byla zapnutá v základní verzi nebo tzv. TSI indexování. Ve verzi 2.x je již pouze TSI indexování. V oficiální dokumentaci o konfiguraci InfluxDB [76] je uvedeno, že v základní konfiguraci je nastavena velikost vyrovnávací mezipaměti indexu TSI pro ukládání dříve vypočtených výsledků řad na hodnotu 100 a nastavením této hodnoty na 0 se mezipaměť deaktivuje. Po nainstalování InfluxDB bylo zjištěno, že v základní konfiguraci programu je tato hodnota nastavena na 0. To znamená, že po nainstalování InfluxDB je tato funkce indexování TSI do mezipaměti vypnuta a informace v oficiální dokumentaci je chybná. Pro účely experimentů by byla tato funkce nicméně vypnuta, aby dotazy v experimentech nebyly zkresleny vypsáním dotazů z paměti RAM. Rozdíly mezi zapnutou a vypnutou indexací do mezipaměti jsou znázorněny v grafu 2.



Graf 2 – doba vypsání dotazu při zapnuté a vypnuté mezipaměti indexace TSI, zdroj: vlastní zpracování

Pro experimenty bylo využito rozhraní InfluxDB UI a InfluxDB API. InfluxDB poskytuje také rozhraní InfluxDB CLI, ale to využito nebylo. Rozhraní InfluxDB UI nabízí několik funkcí, které nejsou k dispozici v InfluxDB CLI, jako např. vizualizace dat nebo upozornění, ale některé funkce jsou naopak pouze v InfluxDB CLI, jako např. přidání uživatele nebo mazání specifických dat. Pro dotazování a práci s daty byl využit jazyk Flux a ne InfluxQL, který je primárně určen pro InfluxDB verze 1.x a její architekturu s databázemi a zásady uchovávání. Ovšem dotazovací jazyk InfluxQL lze využít také

na InfluxDB verze 2.x, ale bylo by potřeba segment namapovat na starou architekturu, tedy na databází a zásady uchovávání.

V InfluxDB lze nastavit šifrování komunikace mezi klienty a serverem pomocí certifikátu TLS (Transport Layer Security). InfluxDB podporuje konkrétně tři typy certifikátů TLS. Prvním druhem jsou certifikáty jedné domény podepsaný certifikační autoritou, druhým jsou zástupné certifikáty podepsané certifikační autoritou a posledním typem jsou certifikáty s vlastním podpisem. Společnost InfluxData ve své dokumentaci [77] důrazně doporučuje povolit HTTPS, zejména pokud jsou požadavky do InfluxDB odesílány přes veřejnou síť. Nicméně některé společnosti při produkčním využití nepoužívají žádné šifrovací certifikáty, protože nepoužívají InfluxDB ve veřejné síti, ale pouze v místní síti LAN (Local Network Area). Pro experimenty nebyly certifikáty použity.

Při prvním spuštění InfluxDB UI byly nastaveny informace o uživateli, jako přihlašovací jméno a heslo, název organizace a název úvodního segmentu. Uživatelské rozhraní mimo přidávání a prohlížení dat nabízí notebooky pro vytváření a komentování procesů a datových toků, grafické panely (Dashboards) pro vizualizaci dat, úkoly (Tasks) pro zpracování a upravení dat pomocí scriptu v jazyce Flux a také upozornění (Alerts) pro monitorování dat.

Obrázek 14 - InfluxDB UI Dashboard, zdroj: vlastní zpracování

Funkce notebooků a upozornění nebyly při experimentech použity, použity byly grafické panely a úkoly. Grafické panely byly použity pro zobrazení metrik z jednotlivých experimentálních pracovišť, které jsou zobrazeny na obrázku 14 na straně 51 a úkoly pro automatický výpočet průměrné hodnoty využití paměti RAM z jednotlivých experimentálních pracovišť za 24 hodin. Výsledná hodnota se automaticky uložila do nového segmentu. Příklad skriptu v jazyce Flux je uveden níže. V InfluxDB verze 1.x byla funkce úkoly pojmenována jako průběžné dotazy (Continuous queries) a je vhodná pro úpravu granularity dat.

```
option task = {name: "mem_dp_4GB", every: 24h0m0s}

from(bucket: "dp_4GB")
      |> range(start: -task.every)
      |> filter(fn: (r) => r._measurement == "procstat" and r._field ==
"memory_rss")
      |> mean()
      |> to(bucket: "mem_dp_4GB", timeColumn: "_stop")
```

4.3.2 Telegraf

Pro monitorování samotné databáze InfluxDB a serverových metrik byl vybrán open source serverový agent Telegraf od společnosti InfluxData. Tento agent byl vybrán, protože je od stejné společnosti jako InfluxDB a obsahuje v InfluxDB přímou podporu. Telegraf obsahuje více než 300 zásuvných modulů k různým platformám.

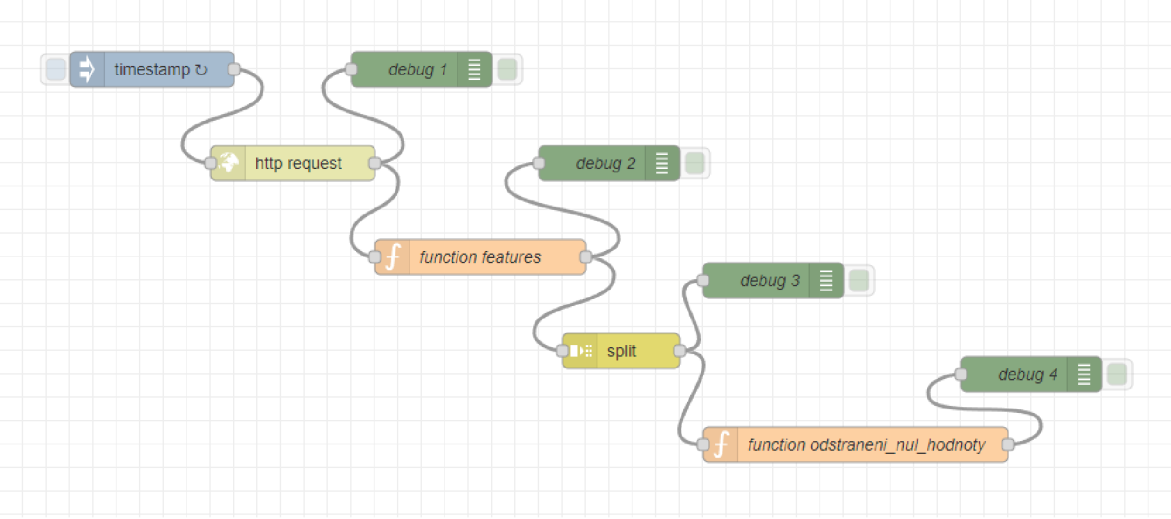
Při instalaci Telegrafu byly přidány do konfiguračního souboru telegraf.conf, který se nachází v /etc/telegraf, vstupní a výstupní zásuvné moduly (input a output plugins). Jako výstupní modul byl zvolen modul pro InfluxDB verze 2.x, do kterého byla přidána URL adresa InfluxDB, vygenerovaný klíč (token) z InfluxDB, organizace a segment do kterého byly zapisovány metriky. Jako vstupní moduly byly zvoleny cpu, disk, diskio, kernel, mem, processes, procstat pro metriky služby InfluxDB, ve kterém byl přidán název uživatele služby InfluxDB, swap, system a prometheus pro interní metriky z InfluxDB, ve kterém byla přidána URL adresa InfluxDB s metrikami, která je ve tvaru adresa:port/metrics. Celý obsah konfiguračního souboru telegraf.conf je zobrazen na straně 53.

```
[global_tags]
[agent]
 interval = "10s"
 round_interval = true
 metric_batch_size = 1000
 metric_buffer_limit = 10000
 collection_jitter = "0s"
 flush_interval = "10s"
 flush_jitter = "0s"
 precision = "0s"
[inputs.cpu]
 percpu = true
 totalcpu = true
 collect_cpu_time = false
 report_active = false
 core_tags = false
[inputs.disk]
 ignore_fs = ["tmpfs", "devtmpfs", "devfs", "iso9660", "overaly", "aufs",
"squashfs"]
[inputs.diskio]
[inputs.kernel]
[inputs.mem]
[inputs.processes]
[inputs.procstat]
 user = "influxdb"
[inputs.swap]
[inputs.system]
[inputs.prometheus]
 urls = ["http://51.178.87.201:8086/metrics"]
[outputs.influxdb_v2]
 urls = ["http://51.91.78.71:8086"]
 token = "TOKEN"
 organization = "CZU"
 bucket = "telegraf"
```

4.3.4 Node-RED

Pro zpracování experimentálních dat z REST API Golemio a následného odesílání do databáze InfluxDB byl použit nástroj Node-RED. Pro odesílání dat do databáze InfluxDB byl do Node-RED nainstalován z knihovny balíček InfluxDB [78] s uzly InfluxDB In, InfluxDB Out a Influx Batch. Následně byl v těchto uzlech nastaven server InfluxDB, do kterého se odesílají data. V nastavení serveru byly zadány informace o názvu, verzi InfluxDB, URL a klíči.

Jako první byl použit uzel Timestamp, kde byla nastavena URL adresa REST API Golemio se všemi vozidly, do hlavičky byl vložen vygenerovaný klíč a doba opakování na interval 10 sekund. Za tento uzel byl vložen uzel HTTP Request, kde byla zvolena metoda GET pro získání dat z REST API a odpověď jako parsovaný JSON objekt. Následně byl vložen uzel Function, kde bylo upraveno pole z předchozího uzlu a uzel Split pro rozdělení pole na jednotlivé objekty. Posledním uzlem tohoto flow byl použit další uzel Function pro odstranění záznamů s vynechanou hodnotou u položky typ. Celé základní flow je znázorněno na obrázku 15. Celé rozšířené flow je k dispozici v příloze A.



Obrázek 15 – základní flow v editoru Node-RED, zdroj: vlastní zpracování

Za základní flow následovaly části určené přímo ke konkrétním experimentům, jejichž počet byl určen podle experimentů. V každé části byl vložen uzel Function, ve kterém byly ze všech informací v jednotlivých zprávách z REST API Golemio vybrány pouze konkrétní informace, tedy čas pořízení, aktuální zpoždění, zpoždění z přechozí zastávky, pozice vozu, číslo linky, číslo vozu, typ dopravy, konečná stanice, přechozí zastávka, pořadí předchozí zastávky, následující zastávka a pořadí následující zastávky. Dále byly v uzlu vybrané

informace rozděleny podle jednotlivých experimentů jako pole nebo značky. Informace o času pořízení byla převedena na unixový formát času. Nakonec byl přidán uzel InfluxDB Out, ve kterém byl vybrán server, do kterého se odesílají data a nastaveny informace o organizaci, segmentu, měření a časová přesnost, která byla zvolena na sekundy, jelikož data z REST API Golemio obsahují čas s přesností pouze na sekundy. Uzel InfluxDB Out musel být přidán zvlášť pro každý server, segment nebo měření. V databázi InfluxDB musel být vytvořen daný segment, jelikož uzel InfluxDB Out vytvořit segment nedokáže.

4.3.5 Grafana

Pro vizualizaci výsledků experimentů byl využit program Grafana. Mohly být použity také grafické panely v uživatelském rozhraní InfluxDB, ale vizualizace v programu Grafana stále nabízí více funkcí a možností grafické úpravy než grafické panely InfluxDB. Ve vizualizačním programu Grafana byl zvolen zdroj databáze InfluxDB, kde byl nastaven dotazovací jazyk Flux, URL adresa, organizace, token a základní segment.

4.4 Experimentální pracoviště

Vývojáři společnosti InfluxData v oficiální dokumentaci pro databázi InfluxDB open source verze 2.x neuvádějí žádné minimální hardwarové požadavky. Pouze v dokumentaci pro InfluxDB verze 1.8 [79] vývojáři doporučují využití SSD disků nebo optimalizované paměti cloudových instancích. Naopak pevné disky nejsou otestovány, a tudíž se nedoporučují. Pro produkční využití se také doporučuje uložit adresáře wal a data na oddělená úložná zařízení, kvůli optimalizaci sporů o disk během velkého zatížení při zápisu. V následující tabulce 9 vývojáři společnosti InfluxData uvádějí limity počtu operací pro jednotlivé konfigurace serveru.

vCPU nebo CPU	RAM	IOPS	Zápis za sekundu	Dotazy* za sekundu	Unikátní série
2-4 jádra	2-4 GB	500	< 5 000	< 5	< 100 000
4-6 jader	8-32 GB	500-1000	< 250 000	< 25	< 1 000 000
8+ jader	32+ GB	1000+	> 250 000	> 25	> 1 000 000

Tabulka 9 – limitní operace InfluxDB pro jednotlivé konfigurace [79]

Pro experimenty s databází časových řad InfluxDB byly zvoleny tři různé experimentální pracoviště. Většina experimentálních pracovišť byla vybrána se slabší konfigurací, konkrétně s menší pamětí RAM, protože u těchto pracovišť se rychleji zobrazí nedostatky databáze, než u velkého produkčního serveru s lepší konfigurací a větší pamětí RAM.

4.4.1 Kubernetes cluster v IBM Cloud

Prvním experimentálním pracovištěm byl zvolen 30denní jednouzlový Kubernetes cluster se dvěma virtuálními jádry procesoru, 4 GB RAM, 100 GB úložištěm a operačním systémem Ubuntu 18 v IBM Cloudu pomocí studentského zvýhodnění IBM Academic. Na webových stránkách společnosti InfluxData je článek [80], ve kterém je uvedeno, že InfluxDB Cloud je součástí katalogu IBM a že lze InfluxDB nasazovat z prostředí IBM Cloudu, ale v katalogu IBM Cloudu se databáze InfluxDB vůbec nenachází. Zřejmě se jedná o starší informace, které jsou v současné době již zastaralé. Nicméně InfluxDB byl po konzultaci s pracovníky IBM nakonec nainstalován na Kubernetes cluster v IBM Cloudu pomocí Helm Charts. Bohužel 30denní cluster neobsahuje perzistentní úložiště a nelze ho ani k němu přikoupit. Pro perzistentní úložiště by se musel zakoupit klasický Kubernetes cluster. Celková kapacita clusteru v Kubernetes je 4 GB RAM, ale uzel pro aplikace obsahuje pouze 2,64 GB RAM, a proto bylo toto pracoviště vyhodnoceno jako nevhodné pro InfluxDB.

Nakonec bylo toto experimentální pracoviště použito jako server s Node-RED. Bohužel druhý den po nasazení se objevila chyba v Node-RED a musel být reinstalován, ale tím se projevila absence perzistentního úložiště a o všechna data v Node-RED se přišlo. Po této zkušenosti byl Node-RED přesunut na jiné experimentální pracoviště.

4.4.2 Raspberry Pi

Dalším experimentálním pracovištěm byl zvolen jednodeskový počítač Raspberry Pi 4 Model B se čtyř jádrovým procesorem, 4 GB RAM a 250 GB SSD s nainstalovaným operačním systémem Raspbian GNU/Linux 11 (Bullseye). Veškerá práce s minipočítačem byla prováděna pomocí programu PuTTY a protokolu SSH (Secure Shell). Na minipočítači byl nainstalován Docker Compose, ve kterém byly spuštěny programy Node-RED a Grafana.

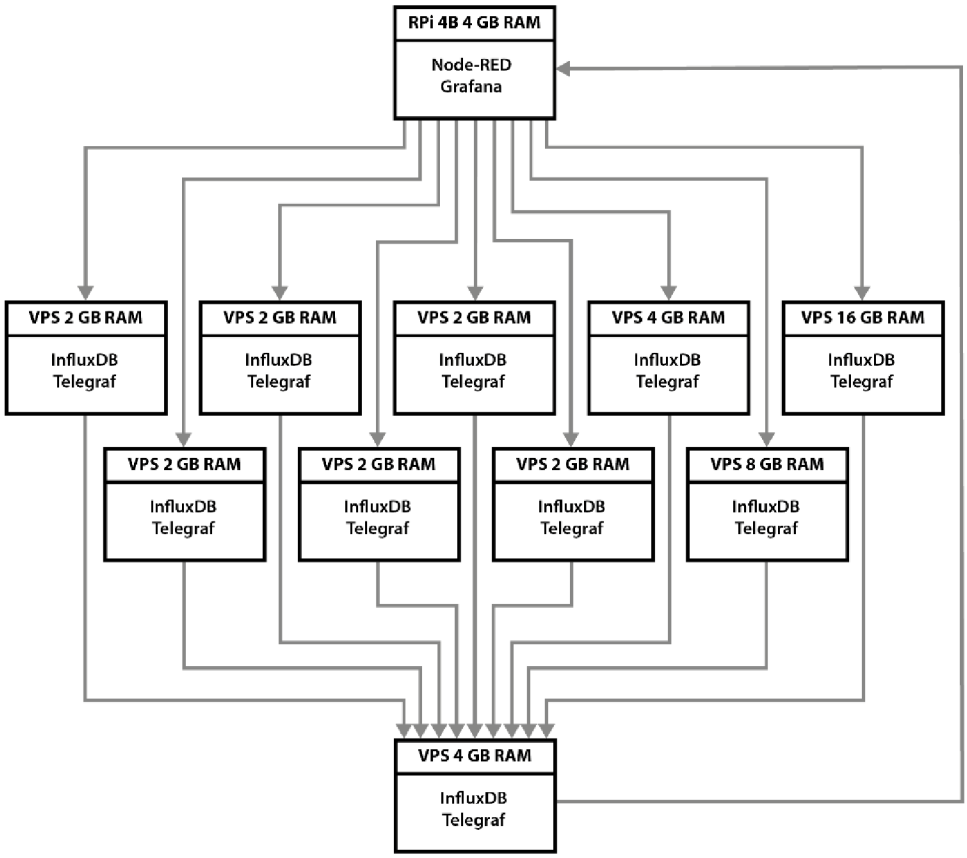
4.4.3 Virtuální stroj od společnosti OVH Cloud

Jako zbylé experimentální pracoviště byly vybrány virtuální servery (VPS – Virtual Private Server) od společnosti OVH Cloud, která disponuje datovými centry po celém světě. Konkrétně bylo zakoupeno pět virtuálních serverů s jedno jádrovým procesorem, 2 GB RAM a 20 GB SSD, jeden s jedno jádrovým procesorem, 2 GB RAM a 40 GB SSD, dva s dvou jádrovým procesorem, 4 GB RAM a 80 GB SSD, dále jeden se čtyř jádrovým procesorem, 8 GB RAM a 160 GB SSD, a nakonec jeden s osmi jádrovým procesorem, 16 GB RAM a 160 GB SSD. U všech serverů bylo zvoleno datové centrum ve Štrasburku ve Francii. Na všech virtuálních serverech byl nainstalován operační systém Ubuntu verze 22.04 bez grafického rozhraní a pouze s příkazovou řádkou. K serverům se přistupovalo pomocí programu PuTTY a protokolu SSH. Všechna experimentální pracoviště jsou uvedena v tabulce 10.

Typ	Operační systém	Procesor	RAM	Disk
Raspberry Pi 4B	Raspbian 11	4 CPU	4 GB	250 GB SSD
VPS OVH Cloud	Ubuntu 22.04	1 vCPU	2 GB	20 GB SSD
VPS OVH Cloud	Ubuntu 22.04	1 vCPU	2 GB	20 GB SSD
VPS OVH Cloud	Ubuntu 22.04	1 vCPU	2 GB	20 GB SSD
VPS OVH Cloud	Ubuntu 22.04	1 vCPU	2 GB	20 GB SSD
VPS OVH Cloud	Ubuntu 22.04	1 vCPU	2 GB	20 GB SSD
VPS OVH Cloud	Ubuntu 22.04	1 vCPU	2 GB	40 GB SSD
VPS OVH Cloud	Ubuntu 22.04	2 vCPU	4 GB	80 GB SSD
VPS OVH Cloud	Ubuntu 22.04	4 vCPU	4 GB	80 GB SSD
VPS OVH Cloud	Ubuntu 22.04	4 vCPU	8 GB	160 GB SSD
VPS OVH Cloud	Ubuntu 22.04	8 vCPU	16 GB	160 GB SSD

Tabulka 10 – seznam všech experimentálních pracovišť, zdroj: vlastní zpracování

Na jednotlivých serverech byly nainstalovány programy InfluxDB a Telegraf a následně spuštěny jako služba. Devět virtuálních serverů bylo použito pro experimenty s různou konfigurací nainstalované databáze InfluxDB. Mimo InfluxDB byl nainstalován také Telegraf pro monitorování databáze a serveru. Jeden virtuální server se 4 GB RAM byl použit jako hlavní databáze, do které byly odesílány metriky ze všech serverů. Celé schéma všech použitých experimentálních pracovišť je zobrazeno na obrázku 16 na straně 58.



Obrázek 16 – schéma experimentálních pracovišť, zdroj: vlastní zpracování

4.5 Experimenty

Dohromady byly vybrány 4 experimenty nad databází časových řad InfluxDB, které byly konzultovány s odborníkem z oblasti databáze InfluxDB. Při experimentech byl sledován současný zápis a čtení. Na všechna uložená data od začátku ukládání byly pokládány 4 dotazy. Mezi jednotlivými dotazy byla na každém experimentálním pracovišti vždy provedena deseti minutová pauza pro vyčištění paměti RAM, aby výsledek měření nebyl zkreslený využitou kapacitou paměti RAM. Všechny dotazy byly pokládány na všechna dosud uložená data. Prvním dotazem bylo vypsání zpoždění všech vozů linky číslo 22. Výsledek tohoto dotazu je obsažen v příloze B. Dotaz v jazyce Flux je následující.

```
from(bucket: "tag-4")
 |> range(start: -21d, stop: now())
 |> filter(fn: (r) => r._measurement == "tag-4")
 |> filter(fn: (r) => r._field == "f_delay")
 |> filter(fn: (r) => r.t_line == "22")
 |> group(columns: ["t_vehicle"])
 |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
```

Druhým dotazem bylo vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty, výsledek je uveden v příloze C.

```
from(bucket: "tag-4")
 |> range(start: -21d, stop: now())
 |> filter(fn: (r) => r._measurement == "tag-4")
 |> filter(fn: (r) => r._field == "f_delay")
 |> filter(fn: (r) => r.t_type == "tramvaj")
 |> group(columns: ["t_line"])
 |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
```

Třetím dotazem bylo vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty. Příklad výsledku je k dispozici v příloze D.

```
from(bucket: "tag-4")
 |> range(start: -21d, stop: now())
 |> filter(fn: (r) => r._measurement == "tag-4")
 |> filter(fn: (r) => r._field == "f_delay")
 |> filter(fn: (r) => r.t_type == "tramvaj")
 |> filter(fn: (r) => r.t_line == "11" or r.t_line == "17" or r.t_line == "22")
 |> group(columns: ["t_line"])
 |> aggregateWindow(every: 1m, fn: mean)
```

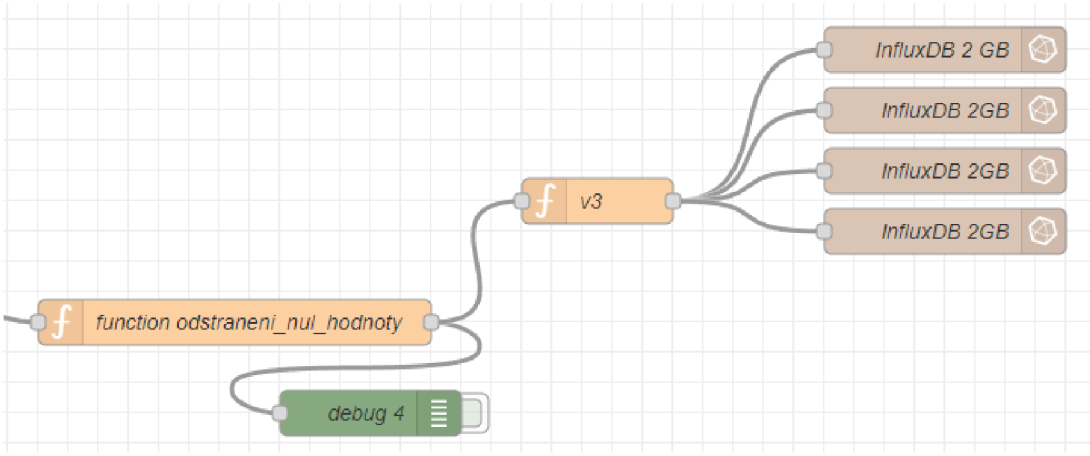
Čtvrtým dotazem bylo vypsání průměrného zpoždění autobusové linky číslo 107 za jednotlivé hodiny. Výsledek je obsažen v příloze E.

```
from(bucket: "tag-4")
 |> range(start: -21d, stop: now())
 |> filter(fn: (r) => r._measurement == "tag-4")
 |> filter(fn: (r) => r._field == "f_delay")
 |> filter(fn: (r) => r.t_line == "107")
 |> group(columns: ["_start"])
 |> aggregateWindow(every: 1h, fn: mean)
 |> sort(columns: ["_time"])
```

4.5.1 Experiment 1 – Odesílání dat jednotlivě vs. dávkově

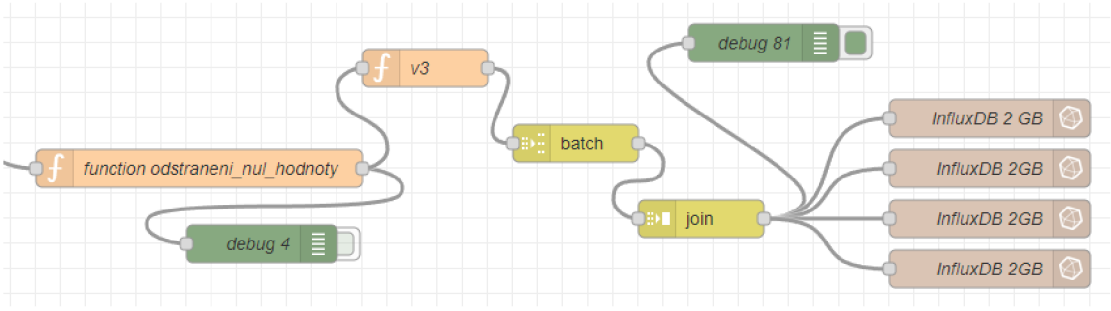
Prvním experimentem bylo zjištění, jaký má vliv, když se do databáze InfluxDB odesílají data jednotlivě (tedy každá zpráva zvlášť) nebo dávkově, kdy jsou všechny zprávy

za časový úsek odeslány v jednom poli (array). Odesílání jednotlivých zpráv do InfluxDB v nástroji Node-RED je znázorněno na obrázku 17. Za uzel Function s rozdělením informací do značek a polí, který je nazván na obrázku 17 v3 byly přidány rovnou uzly InfluxDB Out.



Obrázek 17 – flow odesílání jednotlivých zpráv do InfluxDB v nástroji Node-RED, zdroj: vlastní zpracování

Dávkové odesílání dat do databáze InfluxDB v nástroji Node-RED je znázorněno na obrázku 18. Za uzel Function (v3) byly ještě přidány uzly Batch, ve kterém bylo nastaveno seskupení podle časového intervalu 10 sekund, který pozdrží všechny zprávy po dobu 10 sekund a následně je pošle a Join, který všechny zprávy seskupí do jednoho pole. Následně byl již vložen uzel InfluxDB Out pro odeslání dat do databáze. Alternativou by byl možný uzel InfluxDB Batch, který je určený pro dávkový zápis dat a vyžaduje odlišné požadavky formátu dat oproti uzlu InfluxDB Out. Nicméně obě varianty bylo možné využít, ale nakonec byla použita varianta s uzlem InfluxDB Out, který byl použit i při jednotlivým odesíláním zpráv.



Obrázek 18 – flow dávkového odesílání dat do InfluxDB v nástroji Node-RED, zdroj: vlastní zpracování

Experiment byl proveden na experimentálním pracovišti VPS se 2 GB RAM. Postupně bylo v Node-RED přidáváno více uzlů InfluxDB Out, s tím že do jednoho uzlu bylo odesíláno každých 10 sekund 1 100 zpráv. Při experimentu bylo sledováno využití paměti RAM a chybové zápisy do databáze InfluxDB.

4.5.2 Experiment 2 – Následky vysoké kardinality

Druhým experimentem bylo zjištění, jaký vliv má kardinalita v databázi InfluxDB. Kardinalita v InfluxDB je počet jedinečných dat uložených v databázi jako značky, které jsou indexovány. Kardinalita je započítávána za jednotlivé segmenty. V experimentu byly ukládány experimentální data jako značky a pole třemi způsoby, kdy každý způsob byl ukládán do databáze InfluxDB na samostatném experimentálním pracovišti (VPS 2 GB RAM) a do samostatného segmentu. Do všech tří způsobů byly uloženy stejná data, jen v jiné struktuře. V prvním způsobu ukládání bylo do segmentu uloženo celkem 8 značek a 3 pole, ve druhém způsobu 6 značek a 5 polí a ve třetím způsobu 4 značky a 7 polí. Pro lepší orientaci bylo před každý název přidáno písmeno t pro značku a f pro pole. Konkrétní uložení dat je zobrazeno v tabulce 11.

	1. Způsob (tag-8)	**2. Způsob (tag-6)**	**3. Způsob (tag-4)**
značky	t_line	t_line	t_line
	t_vehicle	t_vehicle	t_vehicle
	t_type	t_type	t_type
	t_final_stop	t_final_stop	t_final_stop
	t_last stop	t_last_stop	
	t_last_stop_order	t_next_stop	
	t_next_stop		
	t_next_stop_order		
pole	time	time	time
	f_delay	f_delay	f_delay
	f_position	f_position	f_position
		f_last_stop_order	f_last_stop
		f_next_stop_order	f_last_stop_order
			f_next_stop
			f_next_stop_order

Tabulka 11 – struktura dat v experimentu 2, zdroj: vlastní zpracování

Tato data byla ukládána do databáze InfluxDB dávkově každých 10 vteřin po dobu 21 dní. Během současného ukládání byly na uložená data pokládány každý den všechny 4 dotazy, které jsou uvedeny v kapitole 4.5 Experimenty. Dotazy byly zadávány na všechna uložená data od začátku ukládání. Při experimentu byla na každém experimentálním pracovišti zaznamenávána doba vypsání jednotlivých dotazů, průměrné využití paměti RAM databází InfluxDB za 24 hodin a míra kardinality jednotlivých segmentů.

4.5.3 Experiment 3 – Výpočet předem vs. výpočet v dotazu

Ve třetím experimentu byly porovnávány rozdíly, když se matematické operace provádějí před uložením dat nebo až při následném dotazu na uložená data. V tomto experimentu byla použita jednoduchá matematická operace rozdíl aktuálního zpoždění a zpoždění z předchozí zastávky.

Pro tento experiment byly použity 2 experimentální pracoviště (VPS 2 GB RAM). Do databází InfluxDB na každém experimentálním pracovišti byla data ukládána dávkově dvěma rozdílnými způsoby každých 10 vteřin po dobu 21 dní. V prvním způsobu ukládání byly do segmentu uloženy celkem 4 značky a 8 polí, s tím že pole rozdíl zpoždění (v databázi nazvána f_delay_diff) obsahovalo hodnotu, která byla v nástroji Node-RED vypočtena rozdílem hodnot aktuálního zpoždění a zpoždění z předchozí zastávky. Na obrázku 19 je znázorněna struktura uložení dat v nástroji Node-RED.

```
msg.payload = [{
    // @ts-ignore
    time: new Date(Time) / 1000,
    f_delay: Delay,
    f_delay_last: Delay_last,
    f_position: Position,
    f_last_stop: Last_stop,
    f_last_stop_order: Last_stop_order,
    f_next_stop: Next_stop,
    f_next_stop_order: Next_stop_order
},
{
    t_line: Line,
    t_vehicle: Vehicle,
    t_type: Type,
    t_final_stop: Final_stop
}];
return msg;
```

Obrázek 19 – struktura zprávy s výpočtem předem v nástroji Node-RED, zdroj: vlastní zpracování

Ve druhém způsobu ukládání byly do segmentu uloženy taktéž 4 značky a 8 polí, ale pole vypočteného rozdílu zpoždění bylo nahrazeno zpožděním z přechozí zastávky (v databázi nazváno f_delay_last). Na obrázku 20 je znázorněno rozložení dat v nástroji Node-RED.

```
msg.payload = [{
    // @ts-ignore
    time: new Date(Time) / 1000,
    f_delay: Delay,
    f_delay_diff: Delay - Delay_last,
    f_position: Position,
    f_last_stop: Last_stop,
    f_last_stop_order: Last_stop_order,
    f_next_stop: Next_stop,
    f_next_stop_order: Next_stop_order
},
{
    t_line: Line,
    t_vehicle: Vehicle,
    t_type: Type,
    t_final_stop: Final_stop
}];
return msg;
```

Obrázek 20 – struktura zprávy s výpočtem v dotazu v nástroji Node-RED, zdroj: vlastní zpracování

Pro dotazování na data byly použity 3 dotazy, které byly opakovány každý den. Konkrétně vypsání všech vozů linky číslo 22, vypsání průměrného zpoždění všech linek tramvají a vypsání průměrného zpoždění 3 linek tramvaje, konkrétně linek číslo 11, 17 a 22. Dotazy byly pokládány na všechna data od začátku ukládání. Na data, která byla uložená prvním způsobem, tedy s výpočtem v programu Node-RED byly dotazy pokládány stejným způsobem jako v předchozím experimentu 2 (ve stejném stylu, jako jsou zobrazeny v kapitole 4.5 Experimenty). Příkladem prvního dotazu na data uložená prvním způsobem v jazyce Flux je následující.

```
from(bucket: „vypocet_predem“)
 |> range(start: -21d, stop: now())
 |> filter(fn: (r) => r._measurement == „vypocet_predem“)
 |> filter(fn: (r) => r._field == „f_delay_diff“)
 |> filter(fn: (r) => r.t_line == „22“)
 |> group(columns: [„t_vehicle“])
 |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
```

Na data, která byla uložena druhým způsobem, tedy s výpočtem v dotazu byly pokládány dotazy, které obsahovaly matematickou funkci, která odečetla od sebe pole aktuálního zpoždění (f_delay) a zpoždění z předchozí zastávky (f_delay_last). Příkladem prvního dotazu na data uložená tímto způsobem v jazyce Flux je následující.

```
from(bucket: "vypocet_v_dotazu")
 |> range(start: -21d, stop: now())
 |> filter(fn: (r) => r._measurement == "vypocet_v_dotazu")
 |> filter(fn: (r) => r._field == "f_delay" or r._field == "f_delay_last")
 |> filter(fn: (r) => r.t_line == "22")
 |> group(columns: ["t_vehicle"])
 |> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn: "_value")
 |> map(fn: (r) => ({r with _value: r.f_delay - r.f_delay_last}))
 |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
```

Mezi důležité ukazatele experimentu patřila doba vypsání jednotlivých dotazů na obou experimentálních pracovištích a průměrné využití paměti RAM databází InfluxDB za 24 hodin. V tabulce 12 je uvedena struktura dat v experimentu 3.

	Výpočet předem	**Výpočet v dotazu**
značky	t_line	t_line
	t_vehicle	t_vehicle
	t_type	t_type
	t_final_stop	t_final_stop
pole	time	time
	f_delay	f_delay
	f_delay_diff	f_delay_last
	f_position	f_position
	f_last_stop	f_last_stop
	f_last_stop_order	f_last_stop_order
	f_next_stop	f_next_stop
	f_next_stop_order	f_next_stop_order

Tabulka 12 – struktura dat v experimentu 3, zdroj: vlastní zpracování

4.5.4 Experiment 4 – Výpočet dotazu v závislosti na paměti RAM

V posledním experimentu byl sledován vliv množství paměti RAM, které má databáze InfluxDB k dispozici, na dotazy na uložená data. Bylo zkoumáno, jaké rozdíly budou v dotazování na data v případě, že databáze InfluxDB bude mít k dispozici pouze 2 GB paměti RAM nebo 4 GB, 8 GB nebo 16 GB. V experimentu byly použity celkem 4 experimentální pracoviště, konkrétně VPS 2 GB RAM, VPS 4 GB RAM, VPS 8 GB RAM a VPS 16 GB RAM. Na všechny experimentální pracoviště byla ukládána data dávkově po 10 sekundách po dobu 21 dní stejnými způsoby jako v experimentech 2 a 3 do samostatných segmentů a pokládány každý den stejné dotazy jako v předchozích experimentech. Na každém experimentálním pracovišti byla data podle struktury ukládána do 5 segmentů, které jsou znázorněny v tabulce 13. Konkrétně 3 segmenty podle experimentu 2 (tag-8, tag-6 a tag-4) a 2 segmenty podle experimentu 3 (vypocet_predem a vypocet_v_dotazu).

VPS 2 GB RAM	VPS 4 GB RAM	VPS 8 GB RAM	VPS 16 GB RAM
tag-8	tag-8	tag-8	tag-8
tag-6	tag-6	tag-6	tag-6
tag-4	tag-4	tag-4	tag-4
vypocet_predem	vypocet_predem	vypocet_predem	vypocet_predem
vypocet_v_dotazu	vypocet_v_dotazu	vypocet_v_dotazu	vypocet_v_dotazu

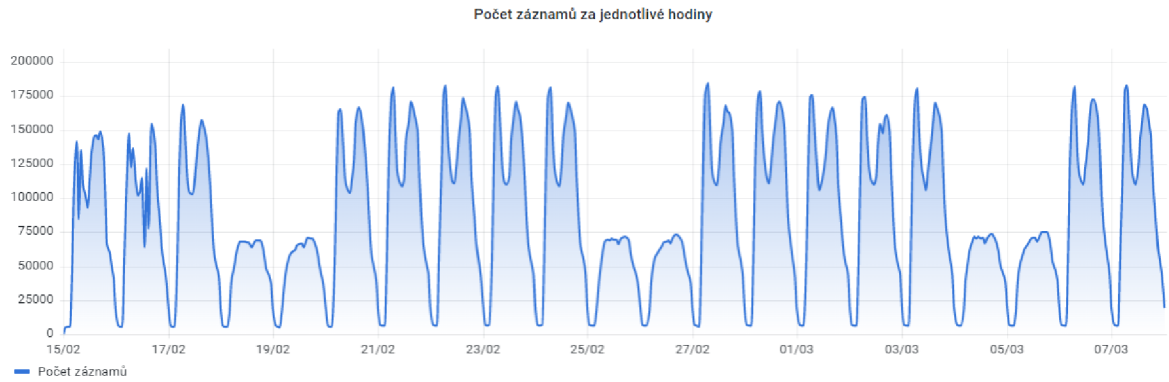
Tabulka 13 – segmenty s daty v experimentální pracovištích v experimentu 4, zdroj: vlastní zpracování

Jelikož bylo do každého experimentálního pracoviště ukládáno 5 rozdílných struktur dat, tak na rozdíl od předchozích experimentů bylo v tomto případě ukládáno do databáze InfluxDB pět krát více senzorických dat v reálném čase než při experimentech 2 a 3.

Při experimentu byla na všech experimentálních pracovištích sledována doba vypsání jednotlivých dotazů na uložená data a průměrné využití paměti RAM databází InfluxDB za 24 hodin.

5 Výsledky a diskuse

V kapitole 4.1 Porovnání vybraných databází pro ukládání časových řad byla vybrána pomocí vícekriteriálního rozhodování jako nejvhodnější databáze časových řad pro experimenty InfluxDB. Jako experimentální data pro experimenty byla vybrána veřejně dostupná senzorická data z pražské hromadné dopravy z API Golemio. Pomocí tohoto API bylo během experimentů ve všední den v dopravní špičce, tedy okolo 6-7 hodiny ranní a 2-5 hodiny odpolední, ukládáno až 1 900 zpráv každých 10 sekund. Některé tyto zprávy byly duplicitní a objevovali se tak až ve 4 po sobě jdoucích periodách aktualizace. To mělo za následek, že v databázi InfluxDB tyto hodnoty byly několikrát přepsány stejnými hodnotami a celkový počet zapsaných bodů se nerovnal počtu příchozích zpráv. V grafu 3 jsou uvedeny průměrné množství uložených bodů za 1 hodinu za celé období experimentů. Z grafu vyplývá, že průměrné množství zapsaných bodů v době dopravní špičky bylo kolem 511 bodů každých 10 sekund. Stejně tomu tak bylo ve dnech víkendu, kde bylo odesíláno z API v době dopravní špičky kolem 700 zpráv, ale počet uložených bodů je v tuto dobu kolem 200.



Graf 3 – průměrné množství uložených bodů v databázi InfluxDB, zdroj: vlastní zpracování

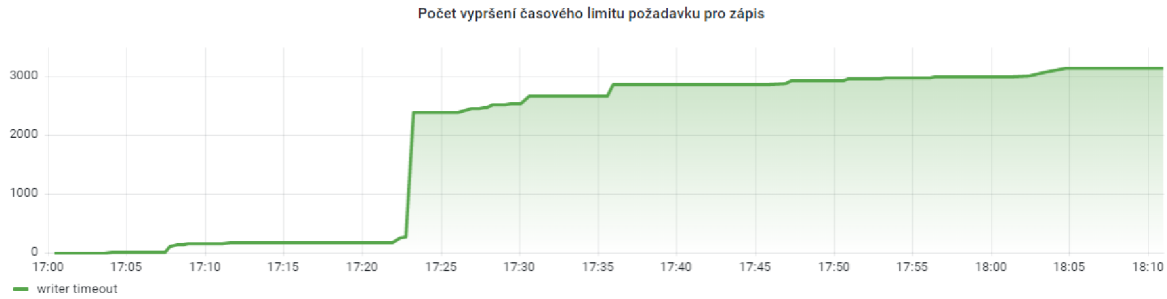
V grafu 3 lze zaznamenat výkyvy uložených bodů 1. a 2. den experimentů, které byly způsobeny problémy s programem Node-RED v experimentálním prostředí Kubernetes v IBM Cloudu, jak bylo zmíněno v kapitole 4.4.1 Kubernetes cluster v IBM Cloudu. Graf 3 se vztahuje k experimentům 2 a 3, kde byla data ukládána do jednoho segmentu. V experimentu 4 byla data ukládána do 5 segmentů a tudíž bylo ukládáno pět krát více dat, než tomu bylo v tomto případě.

5.1 Výsledky experimentu 1

V prvním experimentu byl zkoumán vliv odesílání zpráv do databáze InfluxDB jednotlivě nebo dávkově každých 10 sekund. Během experimentu byly v nástroji Node-RED přidávány uzly InfluxDB out s dalšími měřeními a tím byl násoben počet zpráv k uložení.

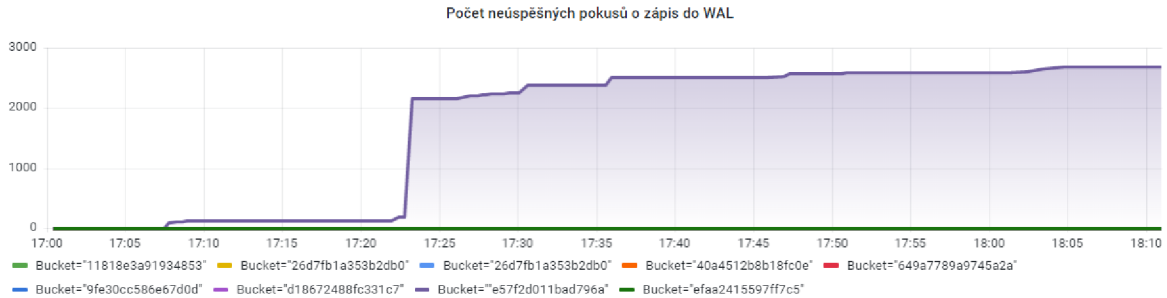
5.1.1 Odesílání dat jednotlivě

Při ukládání dat jednotlivě do databáze InfluxDB nastaly problémy, když byl přidán v nástroji Node-RED šestý uzel InfluxDB Out a do databáze InfluxDB bylo odesíláno jednotlivě 6 600 zpráv každých 10 sekund. Při tomto množství odesílaných zpráv nastaly problémy s ukládáním dat do databáze InfluxDB.



Graf 4 – počet vypršení časového limitu požadavku pro zápis při odesílání dat jednotlivě, zdroj: vlastní zpracování

Při tomto množství odesílaných zpráv databáze InfluxDB přestala být schopna zapisovat všechna data do fragmentů a do protokolu WAL. Tyto problémy byly projeveny v počtu vypršení časového limitu požadavku pro zápis dat do fragmentu, které jsou zobrazeny v grafu 4, v počtu neúspěšných pokusů o zápis do protokolu WAL, které jsou znázorněny v grafu 5 a mimo jiné také v nárůstu využití paměti RAM.



Graf 5 – počet neúspěšných pokusů o zápis do WAL při odesílání dat jednotlivě, zdroj: vlastní zpracování

V grafu 5 na straně 68 je znázorněno, jak vzrostl počet neúspěšných pokusů o zápis do WAL segmentu e57f2d011bad796a, do kterého byly zprávy ukládány, oproti ostatním segmentům. Oba grafy 4 a 5 vypadají obdobně, ale vypršení časového limitu požadavku pro zápis do fragmentu se objevil celkem 3 162× a neúspěšné pokusy o zápis do WAL celkem 2 697×. V důsledků těchto problémů v databázi InfluxDB nebyly některé zprávy nakonec zapsány vůbec nebo s velkým zpožděním.

5.1.2 Odesílání dat dávkově každých 10 sekund

V druhé části experimentu byly zprávy odesílány dávkově každých 10 sekund. Postupně byly opětovně přidávány po 5 minutách v nástroji Node-RED další uzly InfluxDB Out. Mírně obdobné problémy s ukládáním dat jako v případě odesílání jednotlivých zpráv začaly být až při přidání třicátého třetího uzlu. Tedy až v momentě, kdy bylo do databáze InfluxDB odesíláno každých 10 sekund 33 dávek dat po 1 100 zprávách. Vypršení časového limitu požadavku pro zápis do fragmentu se objevil pouze 5× a pouze jeden neúspěšný pokus o zápis do WAL.

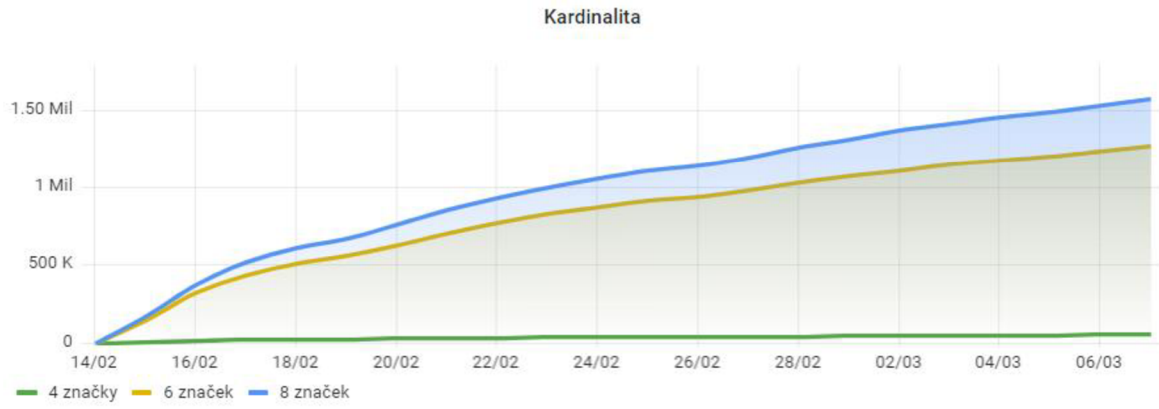
Databáze InfluxDB byla schopna při dávkovém odesílání dat přijmout 5,5× více zpráv než při odesíláním zpráv jednotlivě. Výsledkem tohoto experimentu je doporučení pro odesílání většího množství zpráv dávkově do databáze InfluxDB, a ne po jednotlivých zprávách.

5.2 Výsledky experimentu 2

V druhém experimentu byl zkoumán vliv množství značek a s nimi spojené kardinality na dotazování a chod databáze InfluxDB. Data byla ukládána do databáze InfluxDB třemi způsoby, kde v prvním způsobu byly uloženy jako značky číslo linky, číslo vozu, typ dopravy a konečná stanice. V druhém způsobu bylo uloženo celkem 6 značek, které byly oproti prvnímu způsobu doplněny o kód předchozí a následující zastávky. V posledním způsobu bylo přidáno do značek pořadí předchozí a následující zastávky oproti předchozímu způsobu uložení.

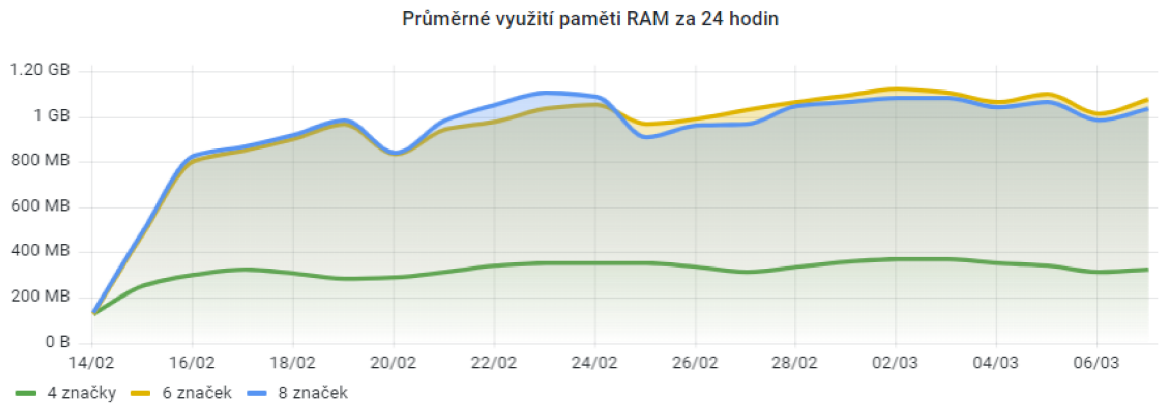
V grafu 6 na straně 69 je znázorněna kardinalita podle jednotlivých segmentů, do kterých byla data se značkami uložena. V segmentu, který obsahoval pouze 4 značky,

bylo dosaženo maximální kardinality 52 241. Ve druhém segmentu se 6 značkami bylo dosaženo maximální kardinality 1 273 237 a v posledním segmentu s 8 značkami maximální kardinality 1 579 147.



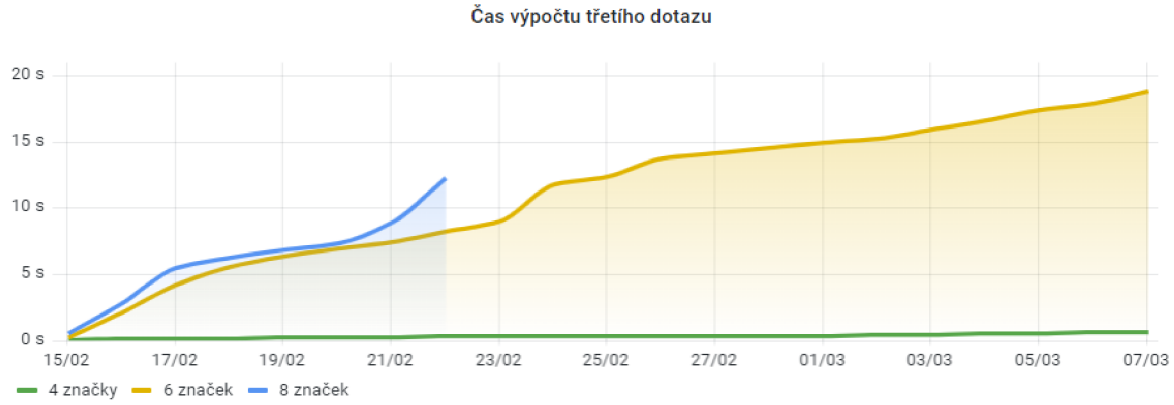
Graf 6 – kardinalita v experimentu 2, zdroj: vlastní zpracování

Jak je znázorněno v grafu 7, vysoká kardinalita měla za následek větší využití paměti RAM, kde segmenty s 6 a 8 značkami měly mnohem větší využití než segment se 4 značkami, a tudíž s menší kardinalitou. V polovině doby trvání experimentu se využití paměti RAM u segmentu s 8 značkami snížilo oproti segmentu s 6 značkami. To bylo ovlivněno tím, že u segmentu s 8 značkami byly vynechány 2 dotazy, protože je databáze InfluxDB již nebyla schopna zpracovat kvůli nedostatku paměti RAM. Poklesy využití paměti RAM u všech segmentů byly způsobeny tím, že databáze InfluxDB jednou za týden pročistí mezipaměť RAM. Dalším důvodem poklesů paměti RAM jsou dny víkendů, kdy množství přijímaných dat bylo menší.



Graf 7 – průměrné využití paměti RAM za 24 hodin v experimentu 2, zdroj: vlastní zpracování

V grafu 8 je zobrazen čas výpočtu třetího dotazu na uložená data, tedy dotazu na vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty. V grafu 8 databáze InfluxDB se segmentem s 8 značkami a vysokou kardinalitou vykazovala vysoké vytížení paměti RAM a od 9. dne dotazování již nebyla schopna dotazy zpracovat kvůli nedostatku paměti. Segment se 6 značkami a poměrně také vysokou kardinalitou dosahoval vysokých hodnot času výpočtu dotazu oproti segmentu se 4 značkami. Všechny výsledky včetně grafů s časy výpočtů dalších dotazů byly přidány do přílohy F.

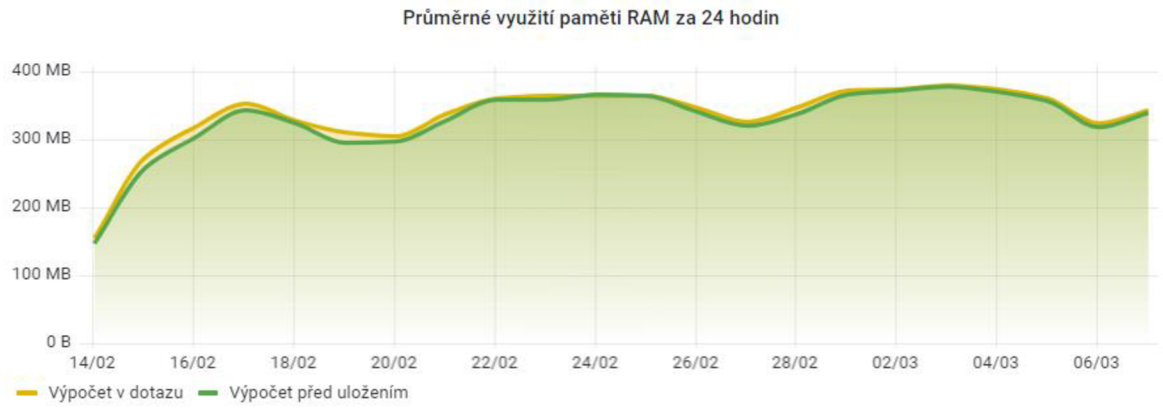


Graf 8 – čas výpočtu třetího dotazu v experimentu 2, zdroj: vlastní zpracování

Při druhém dotazu měly obdobné problémy s nedostatkem paměti RAM segmenty se 6 a s 8 značkami. V ostatních dotazech bylo prokazatelné, že čím byla vyšší kardinalita, tím byl větší čas výpočtu jednotlivých dotazů. Na základě výsledku druhého experimentu lze doporučit ukládání dat do databáze InfluxDB s nejmenším možným počtem dat jako značky, jelikož větší počet značek zvyšuje kardinalitu, která má za následek větší využití paměti RAM a delší čas výpočtu dotazů.

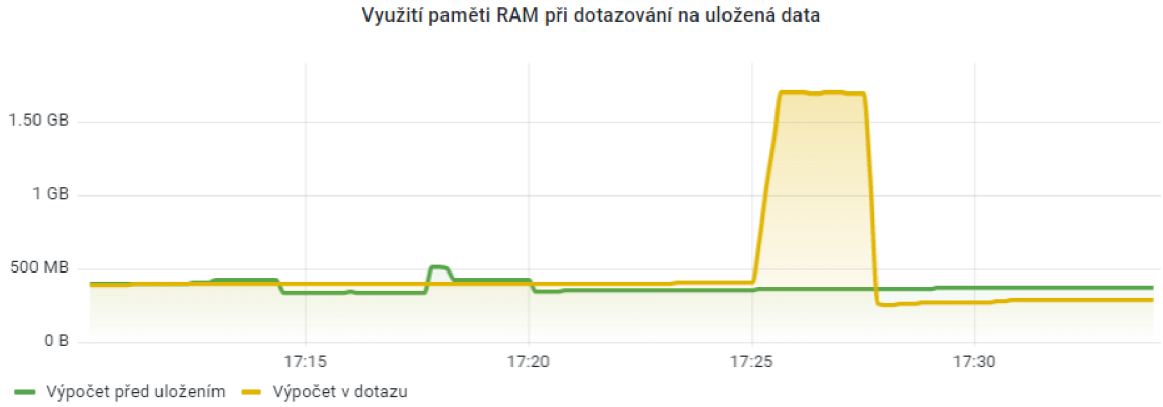
5.3 Výsledky experimentu 3

Ve třetím experimentu byly porovnávány rozdíly, když se matematická operace rozdíl uskuteční před uložením dat a když až při dotazování. Oba způsoby byly provozovány na dvou experimentální pracovištích. V grafu 9 na straně 71 je zobrazené průměrné využití paměti RAM pracoviště, na kterém byl výpočet proveden v dotazu a pracoviště, na kterém byl výpočet proveden již před uložením dat v nástroji Node-RED. Z grafu 9 je patrné, že výpočet předem nebo v dotazu neměl žádný významný vliv na průměrné využití paměti RAM za 24 hodin.



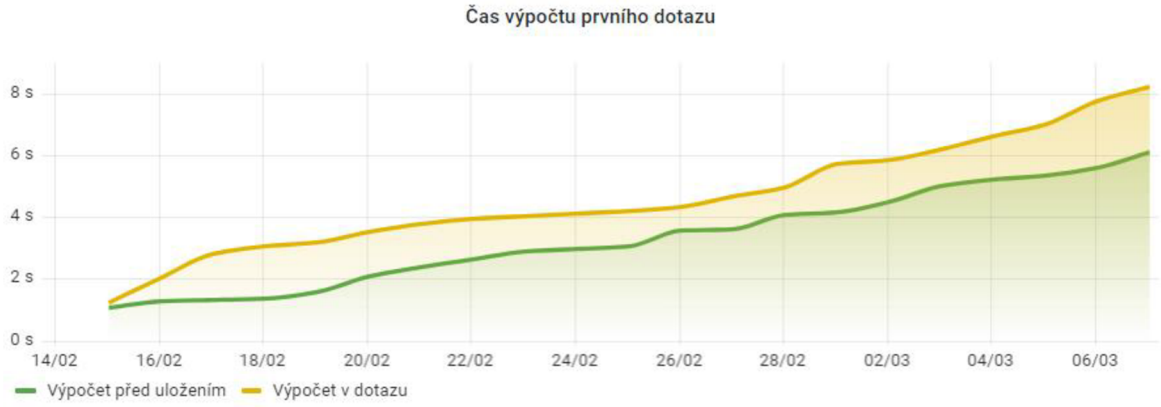
Graf 9 – průměrné využití paměti RAM za 24 hodin v experimentu 3, zdroj: vlastní zpracování

Jak je uvedeno v grafu 10, během experimentu bylo prokázáno, že dotaz, který obsahuje matematický výpočet využívá větší množství paměti RAM než dotaz, který pouze vyčítá hodnoty. V grafu 10 je znázorněn dotaz na vypsání již uložených vypočtených dat v čase 17:17, při kterém se zvýšilo využití paměti RAM pouze na hodnotu 514 MB a také dotaz s výpočtem v čase 17:25, při kterém stouplo využití paměti RAM na hodnotu 1,7 GB. Vzhledem k tomu, že větší využití paměti při dotazování je krátkodobé, tento fakt nijak neovlivnil průměrné využití paměti RAM za 24 hodin, které je zobrazeno v grafu 9 a rozdíl ve využití paměti tak vidět není.



Graf 10 – využití paměti RAM při dotazování na uložená data v experimentu 3, zdroj: vlastní zpracování

V grafu 11 na straně 72 je znázorněn čas výpočtu prvního dotazu na uložená data, tedy dotazu na vypsání rozdílu zpoždění vůči aktuálnímu zpoždění a zpoždění z předchozí zastávky u všech vozů linky číslo 22. Z grafu 11 je zřejmé, že dotaz s výpočtem trvá déle zpracovat, než když dotaz pouze vyčítá data. Grafy s časy výpočtů dalších dotazů byly přidány do přílohy G.



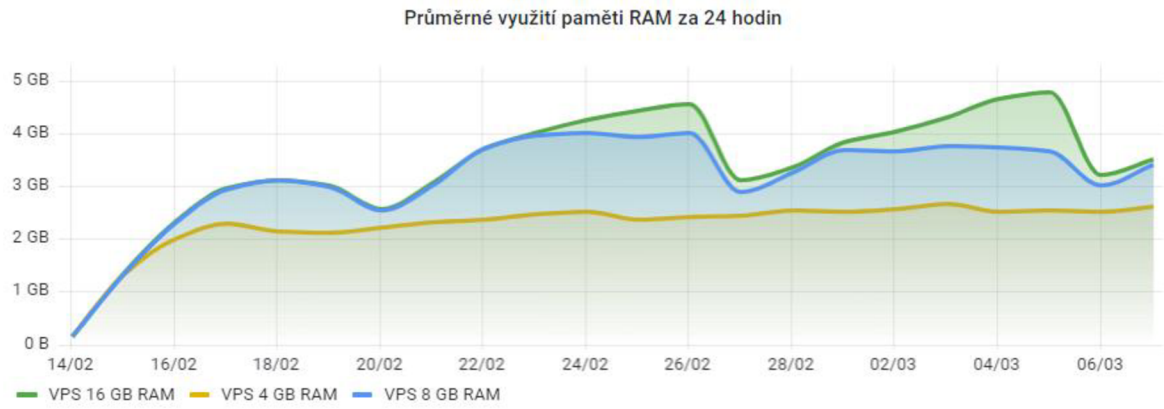
Graf 11 – čas výpočtu prvního dotazu v experimentu 3, zdroj: vlastní zpracování

Při pokládání druhého a třetího dotazu byl rozdíl mezi druhy výpočtu větší než při prvním dotazu. Konkrétně tedy výpočet v dotazu trval mnohem déle, než když byl výpočet proveden před uložením dat. Při vkládání druhého dotazu 18. den již nebyla databáze InfluxDB schopna zpracovat výpočet v dotazu v důsledku nedostatku paměti RAM. Na základě výsledku třetího experimentu lze konstatovat, že výpočet v dotazu má vyšší využití paměti RAM a delší dobu zpracování než výpočet před uložením.

5.4 Výsledky experimentu 4

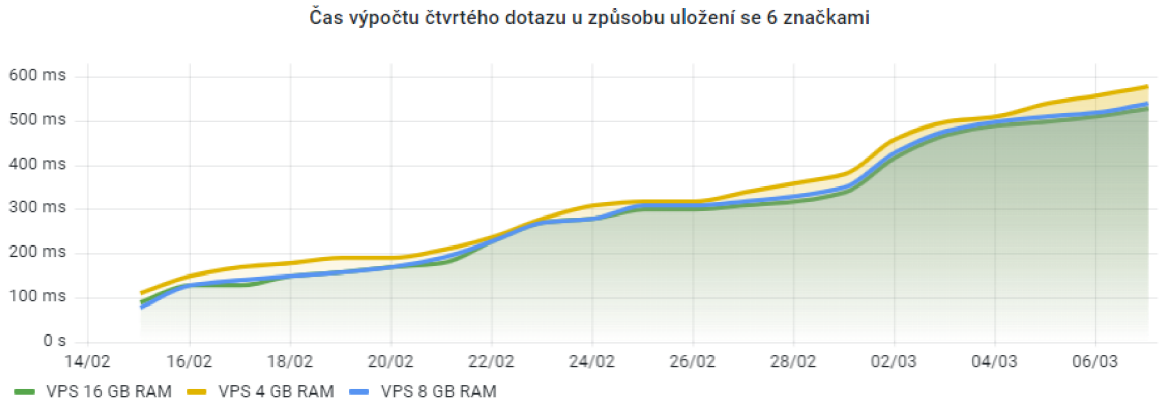
V posledním experimentu byl sledován vliv celkové paměti RAM, kterou má k dispozici databáze InfluxDB na rychlost dotazů na uložená data. V tomto experimentu bylo ukládáno 5× více dat než v předchozích experimentech. Již druhý den ukládání a dotazování na data nebyla schopna databáze InfluxDB na experimentálním pracovišti VPS 2 GB RAM schopna ukládat data a zpracovávat dotazy kvůli nedostatku paměti RAM, z tohoto důvodu byla z experimentu vyloučena.

V grafu 12 na straně 73 je znázorněno průměrné využití paměti RAM jednotlivých experimentálních pracovišť, které bylo ovlivněno celkovým množstvím paměti RAM experimentálních pracovišť. Ze začátku období experimentu bylo využití paměti u databází InfluxDB na experimentálních pracovištích VPS 16 GB RAM a VPS 8 GB RAM téměř stejné, ale postupně bylo u VPS 16 GB RAM využití paměti větší, jelikož byla větší celková kapacita paměti.



Graf 12 – průměrné využití paměti RAM za 24 hodin v experimentu 4, zdroj: vlastní zpracování

V grafu 13 je zobrazen čas výpočtu čtvrtého dotazu na uložená data se 6 značkami, tedy dotazu na vypsání průměrného zpoždění autobusové linky číslo 107 za jednotlivé hodiny. Čas výpočtu byl největší u databáze InfluxDB na experimentálním pracovišti VPS 4 GB RAM, zatímco u zbylých databází na experimentálních pracovištích VPS 8 GB RAM a VPS 16 GB RAM byl čas výpočtu téměř totožný a v některých momentech byl kratší čas zpracování dotazu u databáze 8 GB RAM a někdy u databáze s 16 GB RAM. Při dalších dotazech byl rozdíl trvání výpočtu mezi databází se 4 GB RAM a ostatními někdy větší, než je tomu tak v grafu 13. Všechny grafy s časy výpočtů jednotlivých dotazů jsou k dispozici v příloze H.



Graf 13 – čas výpočtu čtvrtého dotazu u způsobu uložení s 6 značkami v experimentu 4, zdroj: vlastní zpracování

Celkově bylo při pokládání dotazů patrné, že databáze InfluxDB na experimentálním pracovišti VPS 4 GB RAM má menší kapacitu paměti, a tudíž ke zpracování dotazů bylo možné využít menší množství paměti, které mělo za následek delší zpracování dotazů než v ostatních případech. Databáze InfluxDB na experimentálních pracovištích VPS 8 GB RAM a VPS 16 GB RAM měly téměř stejnou dobu zpracování jednotlivých dotazů.

Z výsledku tohoto experimentu lze konstatovat, že databáze InfluxDB lze provozovat i na pracovištích s menší kapacitou paměti RAM, ale doba zpracování dotazu bude delší. Pracoviště ale nesmí obsahovat příliš malou kapacitu paměti RAM, protože když je kapacita paměti příliš malá, databáze InfluxDB nezvládne ukládat data a už nestačí zpracovávat dotazy. Na druhou stranu zde neplatí, že čím více paměti RAM bude mít databáze InfluxDB k dispozici, tím rychlejší bude zpracování dotazů.

5.5 Diskuse

Ze samotného porovnání databází časových řad a následných experimentů s databází InfluxDB lze konstatovat, že databáze InfluxDB by mohla být uplatněná v rámci senzorických řešení České zemědělské univerzity.

Samotné experimenty uvedené v této práci nad databází časových řad InfluxDB by bylo možné převést i na jinou NoSQL databázi využívající princip značek a polí. Následné výsledky by měly být obdobné ne-li stejné, ale bylo by potřeba výsledky na zvolené databázi ověřit.

6 Závěr

V teoretické části byl vytvořen přehled nejpoužívanějších databází časových řad. V přehledu byly charakterizováni 2 zástupci relačních databází QuestDB a TimescaleDB, 2 zástupci NoSQL databází InfluxDB a OpenTSDB a nakonec jeden zástupce cloudových databází Amazon Timestream. Z tohoto přehledu byly vybrány pro porovnání open-source databáze QuestDB, TimescaleDB, InfluxDB a OpenTSDB se 7 kritérii. Konkrétně se jednalo o kritéria přesnost, podporované instalace, podporované programovací jazyky, správa uživatelů, podpora XML, možnost uchovávání v paměti a vlastní vizualizace.

V praktické části byly vybrané databáze časových řad porovnány podle 7 kritérií pomocí metod vícekriteriálního rozhodování. Konkrétně pomocí Saatyho metody pro stanovení vah a metody váženého součtu pro výběr kompromisní varianty. Výsledkem vícekriteriálního rozhodování byla vyhodnocena jako kompromisní varianta databáze časových řad InfluxDB.

Pro účely experimentů nad databází InfluxDB verze 2.x s různou konfigurací bylo vytvořeno 11 experimentálních pracovišť. Jako experimentální pracoviště byly použity Raspberry Pi 4B a virtuální servery od společnosti OVH Cloud. Pro experimenty byly použity mimo InfluxDB OSS také programy Telegraf pro monitorování databáze, Node-RED pro zpracování experimentálních dat a Grafana pro vizualizaci výsledků experimentu. Jako experimentální data byla vybrána volně dostupná senzorická data pražské hromadné dopravy z REST API Golemio, která jsou aktualizována každých 10 sekund. Nakonec byly vybrány 4 experimenty nad databází InfluxDB. Konkrétně experiment 1 – odesílání dat jednotlivě vs. dávkově, experiment 2 – následky vysoké kardinality, experiment 3 – výpočet předem vs. výpočet v dotazu a experiment 4 – výpočet dotazu v závislosti na paměti RAM.

Z teoretické a praktické části práce lze vyhodnotit pro produkční nasazení databáze InfluxDB doporučení pro povolení šifrování dat pomocí certifikátu TLS, zapnutí vyrovnávací mezipaměti indexace TSI a využití úkolů (Tasks) pro uložení dat podle své vlastní granularity. Pro provozování databáze InfluxDB sami vývojáři ze společnosti InfluxData doporučují využití SSD disků nebo optimalizované paměti cloudových instancí. Z výsledků experimentů lze doporučit využít dávkové odesílání zpráv do databáze InfluxDB (např. jednou za 10 sekund), jelikož z experimentu vyplývá, že databáze InfluxDB

je schopna při dávkovém odesílání dat přijmout minimálně 5,5× více zpráv než při odesíláním zpráv jednotlivě. Z výsledků druhého experimentu lze konstatovat, že je výhodnější kvůli vysoké kardinalitě ukládat data různými pohledy s méně značkami, případně i duplicitně, než ukládat všechna data najednou do jednoho měření nebo segmentu. Vysoká kardinalita má za následek větší využití paměti RAM a delší čas výpočtu dotazů. Z výsledků třetího experimentu lze konstatovat, že pokud jsou známy matematické operace, které budou často používány s daty, je doporučeno uložit data již vypočtená a následně již pouze vypisovat uložená vypočtená data. Výpočet v dotazu má větší využití paměti RAM a delší čas výpočtu dotazů oproti pouhému vypisování již uložených dat. Posledním doporučením vyplývající z experimentu 4 je mít velkou kapacitu paměti RAM, jelikož při nižší kapacitě paměti RAM může docházet v databázi InfluxDB k delší době zpracování dotazů nebo dokonce může databáze InfluxDB přestat zvládat ukládat data. Naopak při vyšší kapacitě paměti RAM neplatí, že čím větší je kapacita, tím rychlejší je reakce na dotazy.

Databáze časových řad InfluxDB lze provozovat pomocí grafického rozhraní InfluxDB UI, ve kterém lze vizualizovat data pomocí jednoduchých grafů, tvořit upozornění nebo automatizace. Druhým způsobem provozování je využití příkazové řádky InfluxDB CLI a vizualizačního programu Grafana, který obsahuje více možností vizualizace než grafické rozhraní InfluxDB UI. Pro produkční nasazení se doporučuje více využití příkazové řádky InfluxDB CLI s vizualizačním programem Grafana, jelikož některé funkce databáze InfluxDB (jako např. přidání uživatele) jsou k dispozici jen pomocí rozhraní InfluxDB CLI.

Databáze InfluxDB od verze 2.x využívá jinou architekturu a dotazovací jazyk než ta, která byla použita ve verzi 1.x. Přestože InfluxDB verze 2.x vyšla již na podzim roku 2020, stále mnoho uživatelů i v produkční sféře zůstávají u verze 1.x. Důvody, proč zůstávají u verze 1.x je mnoho, mezi nimi je nový dotazovací jazyk Flux, který má úplně jinou syntaxi oproti staršímu InfluxQL, těžší migrace dat mezi verzemi a neúplná dokumentace a kompatibilita mezi databází InfluxDB 2.x a dalšími aplikacemi.

7 Seznam použitých zdrojů

[1] CHATFIELD, Chris. *Time-Series Forecasting*. 1. vyd. London: CRC Press LLC, 2000. ISBN 978-1584880639.

[2] ARLT, Josef a Markéta ARLTOVÁ. *Ekonomické časové řady: [vlastnosti, metody modelování, příklady a aplikace]*. 1. vyd. Praha: Grada Publishing, a.s, 2007. ISBN 978-80-247-1319-9.

[3] LÖSTER, Tomáš, Hana ŘEZANKOVÁ a Jitka LANGHAMROVÁ. *Statistické metody a demografie*. 1. vyd. Praha: Vysoká škola ekonomie a managementu, 2009. ISBN 978-80-86730-44-8.

[4] KROPÁČ, Jiří. *Statistika: náhodné jevy, náhodné veličiny, základy matematické statistiky, indexní analýza, regresní analýza, časové řady*. 1. vyd. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2010. ISBN 978-80-214-3866-8.

[5] PALMA, Wilfredo. *Time series analysis*. 1. vyd. Hoboken: John Wiley & Sons, 2016. ISBN 978-1-118-63432-5.

[6] OPPEL, Andy. *SQL bez předchozích znalostí: průvodce pro samouky*. 1. dopl. vyd. Brno: Computer Press, 2012. ISBN 978-80-251-1707-1.

[7] GROFF, James a Paul WEINBERG. *SQL: kompletní průvodce*. 1. vyd. Brno: CP Books, 2005. ISBN 80-251-0369-2.

[8] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. *Big Data a NoSQL databáze*. 1. vyd. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.

[9] HARRINGTON, Jan. *Relational Database Design and Implementation: Clearly Explained*. 3. vyd. Burlington: Morgan Kaufmann Publishers, 2009. ISBN 978-0-12-374730-3.

[10] MEIER, Andreas a Michael KAUFMANN. *SQL & NoSQL databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. 1. vyd. Wiesbaden: Springer, 2019. ISBN 978-3-658-24548-1.

[11] PETROV, Alex. *Database internals: a deep dive into how distributed data systems work*. 1. Sebastopol: O'Reilly Media, 2019. ISBN 978-1-492-04034-7.

[12] TELNAROVÁ, Zdeňka. Using Relational Databases for Time Series Data. In: *AIP Conference Proceedings* [online]. Ostrava: AIP Publishing, 2018 [cit. 2022-07-16]. Dostupné z: https://aip.scitation.org/doi/epdf/10.1063/1.5079066

[13] CAP Theorem. In: IBM. *IBM* [online]. 2019 [cit. 2022-07-10]. Dostupné z: https://www.ibm.com/cloud/learn/cap-theorem

[14] STEDMAN, Craig. What is a cloud database? An in-depth cloud DBMS guide. In: *TechTarget: Cloud Computing* [online]. 2022 [cit. 2022-07-17]. Dostupné z: https://www.techtarget.com/searchcloudcomputing/definition/cloud-database

[15] MOSTAFA, Jalal, Sara WEHBI, Suren CHILINGARYAN a Andreas KOPMANN. *SciTS: A Benchmark for Time-Series Database in Scientific Experiments and Industrial Internet of Things* [online]. 2022, 11 [cit. 2022-05-18]. Dostupné z: https://arxiv.org/pdf/2204.09795.pdf

[16] ANDLINGER, Paul. Time Series DBMS as a new trend?. In: *DB-Engines* [online]. Kalchreuth: IT GmbH, 2015 [cit. 2022-07-19]. Dostupné z: https://db-engines.com/en/blog_post/45

[17] DIX, Paul. Why Time Series Matters for Metrics, Real-Time Analytics and Sensor data. *Influxdata* [online]. San Francisco: Influxdata, 2021, 19 [cit. 2022-07-18]. Dostupné z: https://www.influxdata.com/resources/why-time-series-matters-for-metrics-real-time-and-sensor-data/

[18] LAM, Kam-Yiu a Tei-Wei KUO. *Real-Time Database Systems: Architecture and Techniques*. Boston: Kluwer Academic Publishers, 2002. ISBN 0-792-37218-2.

[19] Introduction. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/

[20] QuestDB Enterprise. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/enterprise

[21] QuestDB Cloud. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/cloud

[22] Storage model. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/concept/storage-model

[23] Indexes. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/concept/indexes

[24] SQL extensions. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/concept/sql-extensions/

[25] Partitions. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/concept/partitions

[26] Insert data. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/develop/insert-data/#http-rest-api

[27] Query data. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/develop/query-data#querying-data

[28] Web console. In: *QuestDB* [online]. 2022 [cit. 2022-07-24]. Dostupné z: https://questdb.io/docs/develop/insert-data/#web-console

[29] KULKARNI, Ajay. How we are building a self-sustaining open-source business in the cloud era. In: *Timescale* [online]. 2018 [cit. 2022-07-20]. Dostupné z: https://www.timescale.com/blog/how-we-are-building-an-open-source-business-a7701516a480/

[30] Install TimescaleDB. In: *TimescaleDocs* [online]. 2022 [cit. 2022-07-19]. Dostupné z: https://docs.timescale.com/install/latest/

[31] Hypertables. In: *TimescaleDocs* [online]. 2022 [cit. 2022-07-20]. Dostupné z: https://docs.timescale.com/timescaledb/latest/overview/core-concepts/hypertables-and-chunks/#partitioning-in-hypertables-with-chunks

[32] Architecture & Concepts. In: *Timescale* [online]. 2022 [cit. 2022-07-21]. Dostupné z: https://legacy-docs.timescale.com/v1.7/introduction/architecture

[33] Scalling TimescaleDB. In: *TimescaleDocs* [online]. 2022 [cit. 2022-07-21]. Dostupné z: https://docs.timescale.com/timescaledb/latest/overview/core-concepts/scaling/#single-instance-node

[34] Native compression. In: *TimescaleDocs* [online]. 2022 [cit. 2022-07-21]. Dostupné z: https://docs.timescale.com/timescaledb/latest/overview/core-concepts/compression/

[35] THINATH, Alex. Loading data into Managed Service for TimescaleDB: tutorials for five common methods: Including bulk upload, migration, and inserting data. In: *Timescale* [online]. 2022 [cit. 2022-07-21]. Dostupné z: https://kb-managed.timescale.com/en/articles/3575240-loading-data-into-managed-service-for-timescaledb-tutorials-for-five-common-methods

[36] Writing data. In: *TimescaleDocs* [online]. 2022 [cit. 2022-07-21]. Dostupné z: https://docs.timescale.com/timescaledb/latest/how-to-guides/write-data/#writing-data

[37] Querying data. In: *TimescaleDocs* [online]. 2022 [cit. 2022-07-21]. Dostupné z: https://docs.timescale.com/timescaledb/latest/how-to-guides/query-data/#querying-data

[38] Visualizing data. In: *Timescale* [online]. 2022 [cit. 2022-07-21]. Dostupné z: https://legacy-docs.timescale.com/v1.7/using-timescaledb/visualizing-data

[39] Install InfluxDB. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-22]. Dostupné z: https://docs.influxdata.com/influxdb/v2.3/install/

[40] Simplicity matters. Build once. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-22]. Dostupné z: https://www.influxdata.com/products/

[41] InfluxDB 1.x. In: *InfluxData* [online]. 2022 [cit. 2022-07-23]. Dostupné z: https://www.influxdata.com/time-series-platform/

[42] InfluxDB storage engine. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-22]. Dostupné z: https://docs.influxdata.com/influxdb/v2.3/reference/internals/storage-engine/

[43] InfluxDB shards and shard groups. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-23]. Dostupné z: https://docs.influxdata.com/influxdb/v2.3/reference/internals/shards/

[44] Glossary. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-23]. Dostupné z: https://docs.influxdata.com/influxdb/v2.3/reference/glossary/#compaction

[45] Manage bucket schemas. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-23]. Dostupné z: https://docs.influxdata.com/influxdb/cloud/organizations/buckets/bucket-schema/

[46] InfluxDB data elements. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-23]. Dostupné z: https://docs.influxdata.com/influxdb/v2.3/reference/key-concepts/data-elements/

[47] Write data to InfluxDB. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-23]. Dostupné z: https://docs.influxdata.com/influxdb/v2.3/write-data/

[48] Flux documentation. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-22]. Dostupné z: https://docs.influxdata.com/flux/v0.x/

[49] Flux query basics. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-22]. Dostupné z: https://docs.influxdata.com/flux/v0.x/get-started/query-basics/

[50] Visualize data with the InfluxDB UI. In: *InfluxData Documentation* [online]. 2022 [cit. 2022-07-23]. Dostupné z: https://docs.influxdata.com/influxdb/v2.3/visualize-data/

[51] How does OpenTSDB work?. In: *OpenTSDB* [online]. 2021 [cit. 2022-07-26]. Dostupné z: http://opentsdb.net/overview.html

[52] HBase Schema. In: *OpenTSDB 2.4 documentation* [online]. 2021 [cit. 2022-07-27]. Dostupné z: http://opentsdb.net/docs/build/html/user_guide/backends/hbase.html

[53] Writing Data. In: *OpenTSDB 2.4 documentation* [online]. 2021 [cit. 2022-07-26]. Dostupné z: http://opentsdb.net/docs/build/html/user_guide/writing/index.html

[54] FAQ. In: *OpenTSDB* [online]. 2022 [cit. 2022-07-26]. Dostupné z: http://opentsdb.net/faq.html

[55] Querying or Reading Data. In: *OpenTSDB 2.4 documentation* [online]. 2021 [cit. 2022-07-27]. Dostupné z: http://opentsdb.net/docs/build/html/user_guide/query/index.html

[56] GUI. In: *OpenTSDB 2.4 documentation* [online]. 2021 [cit. 2022-07-27]. Dostupné z: http://opentsdb.net/docs/build/html/user_guide/guis/index.html#index-0

[57] Architecture. In: *Amazon Web Services* [online]. 2021 [cit. 2022-07-25]. Dostupné z: https://docs.aws.amazon.com/timestream/latest/developerguide/architecture.html

[58] BOOZ, Ryan. TimescaleDB vs. Amazon Timestream: 6000x Higher Inserts, 5-175x Faster Queries, 150x-220x Cheaper. In: *Timescale* [online]. 2020 [cit. 2022-07-26]. Dostupné z: https://www.timescale.com/blog/timescaledb-vs-amazon-timestream-6000x-higher-inserts-175x-faster-queries-220x-cheaper/

[59] Writes. In: *Amazon Web Services* [online]. 2021 [cit. 2022-07-25]. Dostupné z: https://docs.aws.amazon.com/timestream/latest/developerguide/writes.html

[60] Query. In: *Amazon Web Services* [online]. 2021 [cit. 2022-07-25]. Dostupné z: https://docs.aws.amazon.com/timestream/latest/developerguide/queries.html

[61] Scheduled Query. In: *Amazon Web Services* [online]. 2021 [cit. 2022-07-25]. Dostupné z: https://docs.aws.amazon.com/timestream/latest/developerguide/scheduled-query.html

[62] Amazon Timestream: Developer Guide. In: *Amazon Web Services* [online]. 2021 [cit. 2022-07-25]. Dostupné z: https://docs.aws.amazon.com/timestream/latest/developerguide/timestream.pdf

[63] Introduction to Grafana. In: *Grafana Labs* [online]. 2022 [cit. 2022-08-01]. Dostupné z: https://grafana.com/docs/grafana/latest/introduction/

[64] Pricing. In: *Grafana Labs* [online]. 2022 [cit. 2022-08-01]. Dostupné z: https://grafana.com/pricing/

[65] Install Grafana. In: *Grafana Labs* [online]. 2022 [cit. 2022-08-01]. Dostupné z: https://grafana.com/docs/grafana/latest/setup-grafana/installation/

[66] Data sources. In: *Grafana Labs* [online]. 2022 [cit. 2022-08-01]. Dostupné z: https://grafana.com/docs/grafana/latest/datasources/

[67] About Grafana panels. In: *Grafana Labs* [online]. 2022 [cit. 2022-08-01]. Dostupné z: https://grafana.com/docs/grafana/latest/panels/

[68] Grafana. In: *Grafana Labs* [online]. 2022 [cit. 2022-08-01]. Dostupné z: https://grafana.com/grafana/

[69] About. In: *Node-RED* [online]. 2022 [cit. 2022-07-30]. Dostupné z: https://nodered.org/about/

[70] Getting Started. In: *Node-RED* [online]. 2022 [cit. 2022-07-30]. Dostupné z: https://nodered.org/docs/getting-started/

[71] Node-RED: Low-code programming for event-driven applications. In: *Node-RED* [online]. 2022 [cit. 2022-07-30]. Dostupné z: https://nodered.org/

[72] Node-RED Concepts. In: *Node-RED* [online]. 2022 [cit. 2022-07-30]. Dostupné z: https://nodered.org/docs/user-guide/concepts

[73] ŠUBRT, Tomáš. *Ekonomicko-matematické metody*. 2. uprav. vyd. Plzeň: Aleš Čeněk, s.r.o., 2015. ISBN 978-80-7380-563-0.

[74] Public Transport | Golemio Output Gateway API. In: *Swagger UI* [online]. 2022 [cit. 2022-12-25]. Dostupné z: https://api.golemio.cz/v2/pid/docs/openapi/#/

[75] API Keys Management. In: *API Golemio* [online]. 2023 [cit. 2023-02-25]. Dostupné z: https://api.golemio.cz/api-keys/auth/sign-in

[76] InfluxDB configuration options. In: *InfluxData Documentation* [online]. 2023 [cit. 2023-02-20]. Dostupné z: https://docs.influxdata.com/influxdb/v2.6/reference/config-options/#storage-series-id-set-cache-size

[77] Enable TLS encryption. In: *InfluxData Documentation* [online]. InfluxData, Inc., 2023 [cit. 2023-02-23]. Dostupné z: https://docs.influxdata.com/influxdb/v2.6/security/enable-tls/

[78] Node-red-contrib-influxdb. In: *Node-RED* [online]. 2023 [cit. 2023-03-03]. Dostupné z: https://flows.nodered.org/node/node-red-contrib-influxdb

[79] Hardware sizing guidelines. In: *InfluxData Documentation* [online]. 2023 [cit. 2023-02-17]. Dostupné z: https://docs.influxdata.com/influxdb/v1.8/guides/hardware_sizing/#influxdb-oss-guidelines

[80] IBM Cloud. In: *InfluxData* [online]. 2023 [cit. 2023-02-17]. Dostupné z: https://www.influxdata.com/partners/ibm-bluemix/

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam grafů

8.4 Seznam použitých zkratek

ACID - Atomicity Consistency Isolation Durability
AHP - Analytic Hierarchy Process
ANSI - American National Standards Institute
API - Application Programming Interface
ARIMA - Autoregresive Integrated Moving Average
AWS - Amazon Web Services
BASE - Basically Available Soft sate Eventual Consistency
CAP - Consistency Availability Partition
CLI - Command Line Interface
CRUD - Create Read Update Delete
CSV - Comma- Separated Values
DBaaS - DataBase as a Service
DCL - Data Control Language
DDL - Data Definition Language
DML - Data Manipulation Language
GCP - Google Cloud Platform
GUI - Graphical User Interface

HTTP - HyperText Transfer Protocol
InfluxQL - Influx Query Language
IoT - Internet of ThingS
ISO - Internal Organization of Standardization
LZO - Lempel-Ziv-Oberhumer
NoSQL - Not Only SQL
OSS - Open Source System
RAM - Random Access Memory
RDBS - Relational Database
REST - Representational State Transfer
SARIMA - Seasonal Autoregresive Integrated Moving Average
SQL - Structured Query Language
SŘBD - Systém Řízení Báze Dat
SSH - Secure Shell
TCP - Transmission Control Protocol
TICK - Telegraf InfluxDB Chronograf Kapacitor
TLS - Transport Layer Security
TSD - Time Series Daemon
TSDB - Time Series Database
TSI - Time Series Index
TSL - TimeScale License
TSM - Time-Structured Merge Tree
UI - User Interface
VPS - Virtual Private Server
WAL - Write Ahead Log

Přílohy

Příloha A Kód vytvořeného flow z programovacího nástroje Node-RED

Příloha A obsahuje kód vytvořeného flow z programovacího nástroje Node-RED, který byl použit při experimentech.

Příloha A je k dispozici ve formátu JSON v přiloženém DVD a ke stažení na adrese URL:

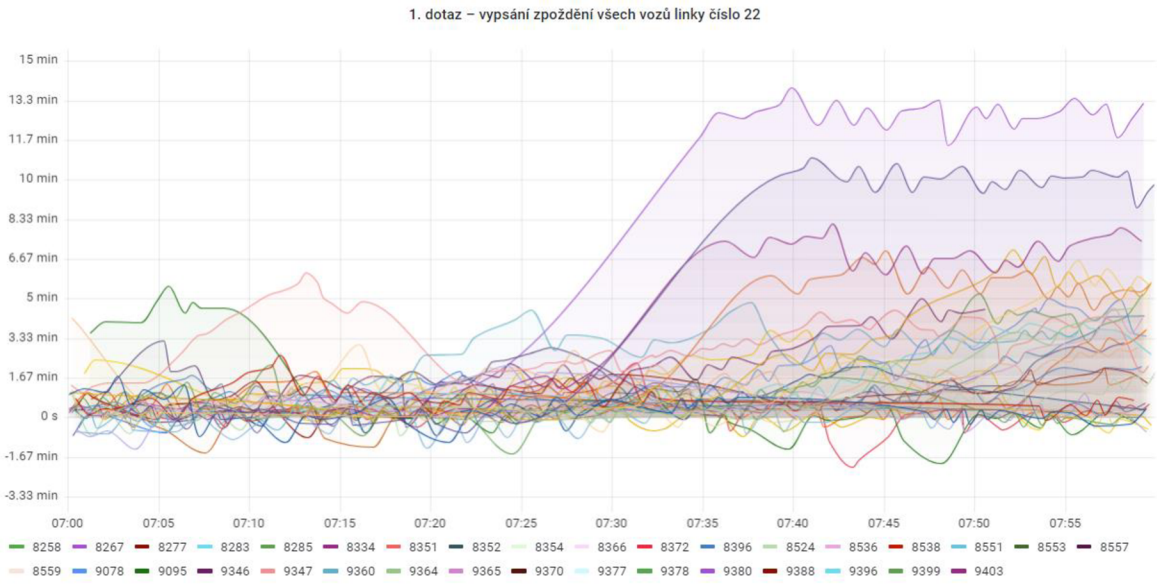
https://drive.google.com/file/d/1YMGPTU0aE31YfRVPx7OOyk2wg3lLL8xD/view?usp=share_link

Příloha B Výsledek dotazu 1

Příloha B obsahuje výsledek dotazu 1 – vypsání zpoždění všech vozů linky číslo 22.

Příloha B je přiložena níže a také je k dispozici ve formátu PNG v přiloženém DVD a ke stažení na adrese URL:

https://drive.google.com/file/d/1l9fCS_TJ-td20BL54m1Gu5V78VzVL_dq/view

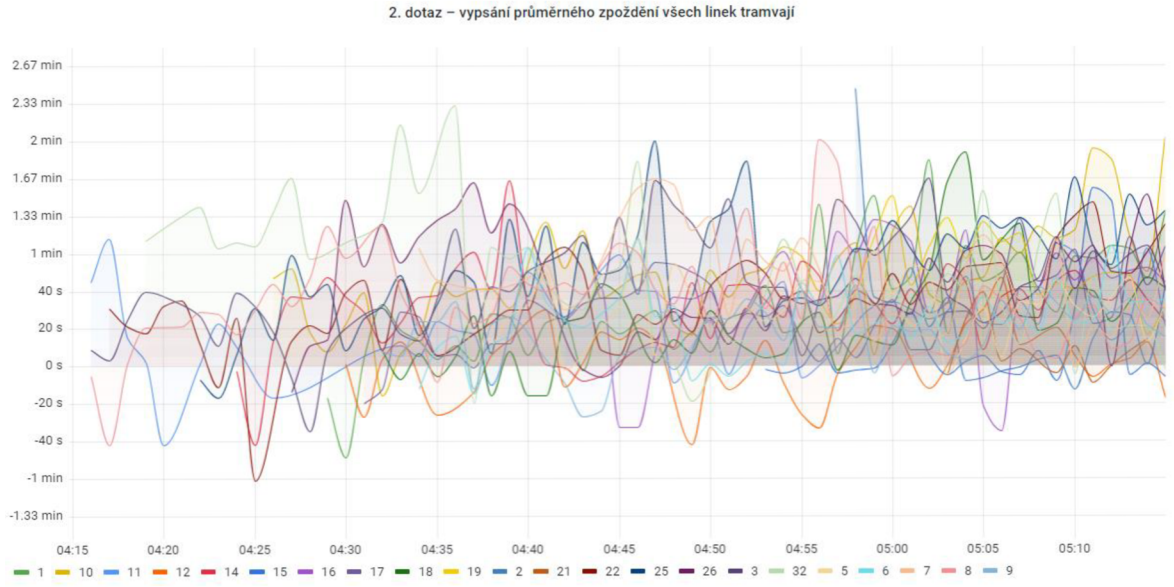


Příloha C Výsledek dotazu 2

Příloha C obsahuje výsledek dotazu 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty.

Příloha C je přiložena níže a také je k dispozici ve formátu PNG v přiloženém DVD a ke stažení na adrese URL:

https://drive.google.com/file/d/1lItOZKKh5sjfmgrk3hibKEZqv3uiKCFb/view?usp=share_link



Příloha D Výsledek dotazu 3

Příloha D obsahuje výsledek dotazu 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty.

Příloha D je přiložena níže a také je k dispozici ve formátu PNG v přiloženém DVD a ke stažení na adrese URL:

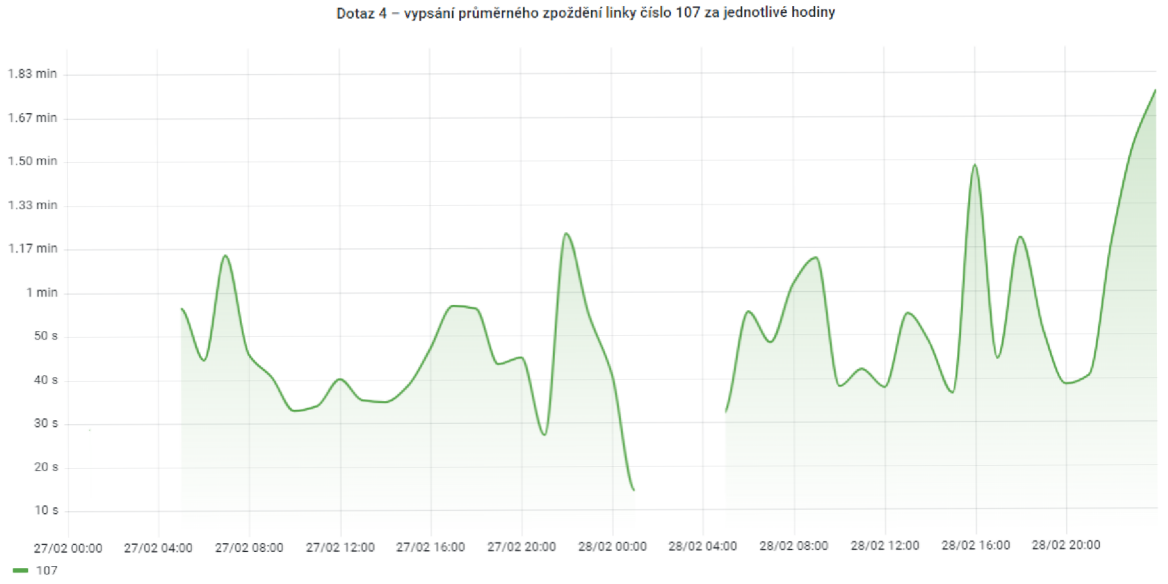
https://drive.google.com/file/d/1HDpKByK3jvA9Qprql78VU5-XCAVuzS8S/view?usp=share_link

Příloha E Výsledek dotazu 4

Příloha E obsahuje výsledek dotazu 4 – vypsání průměrného zpoždění autobusové linky číslo 107 za jednotlivé hodiny.

Příloha E je přiložena níže a také je k dispozici ve formátu PNG v přiloženém DVD a ke stažení na adrese URL:

https://drive.google.com/file/d/19nNYUGwKaHWJDKKOpZgI6mU_4p45ZpiD/view?usp=share_link



Příloha F Výsledky měření experimentu 2

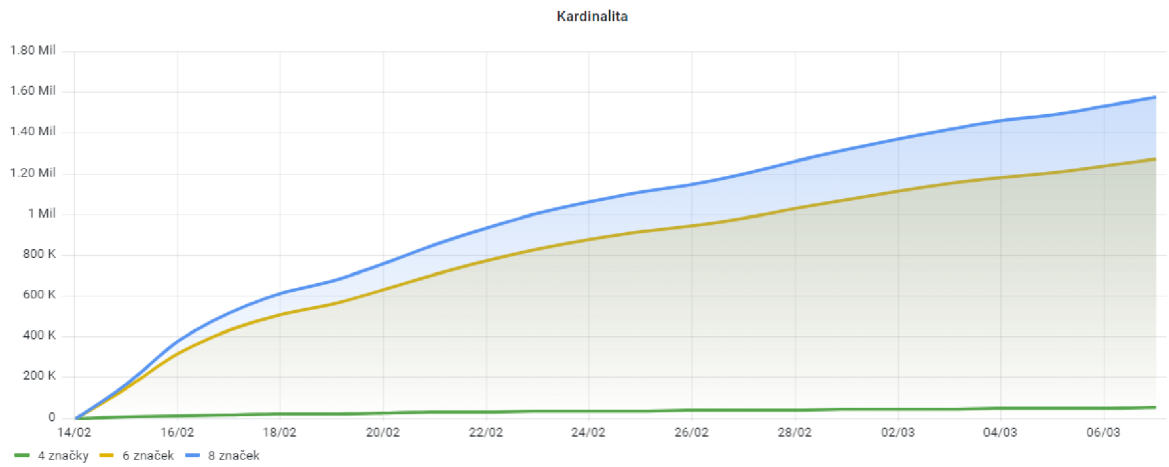
Příloha F obsahuje výsledky měření experimentu 2 – Následky vysoké kardinality.

Příloha F je přiložena na následujících stranách a také je k dispozici ve formátu PDF v přiloženém DVD a ke stažení na adrese URL:

https://drive.google.com/file/d/14ETyDkh0D6XtadhbzgYYDl0XBukgJ3ge/view?usp=share_link

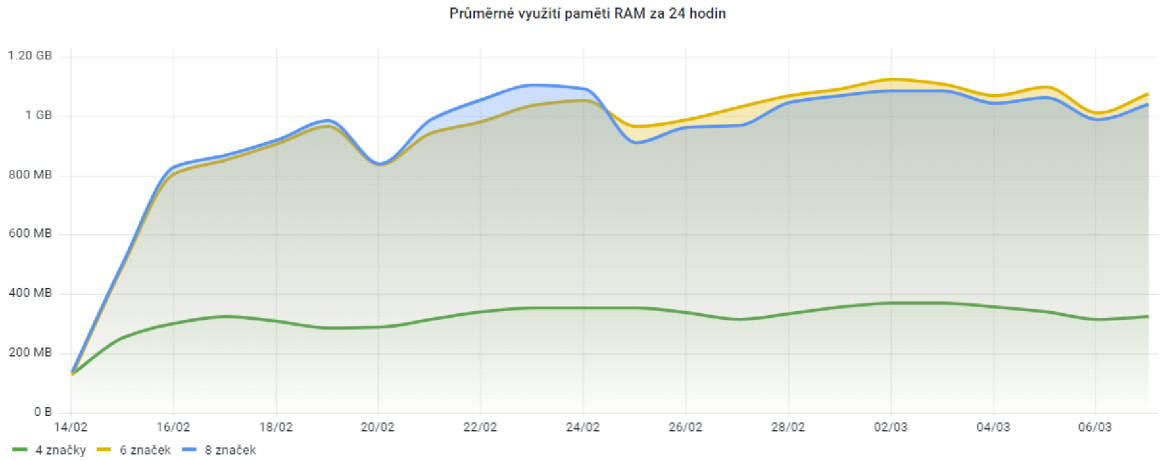
Kardinalita databáze

Den	4 značky	6 značek	8 značek
1.	6 881,68	148 838,26	170 593,13
2.	13 825,79	322 319,99	380 391,39
3.	18 259,87	435 125,82	518 561,58
4.	21 218,84	511 311,12	612 073,78
5.	23 234,57	562 744,79	674 897,78
6.	26 186,53	634 539,95	762 941,10
7.	29 251,06	711 153,97	856 319,25
8.	31 855,94	776 766,78	937 759,16
9.	34 073,89	831 887,52	1 006 336,77
10.	36 008,04	879 726,90	1 066 671,67
11.	37 525,69	917 654,40	1 114 203,03
12.	38 650,23	944 935,92	1 148 831,16
13.	40 392,05	985 877,29	1 200 940,34
14.	42 343,52	1 033 580,21	1 262 841,01
15.	44 116,79	1 076 215,50	1 319 620,82
16.	45 785,73	1 116 213,81	1 372 951,99
17.	47 386,06	1 154 359,70	1 422 539,50
18.	48 602,79	1 184 768,87	1 461 803,42
19.	49 525,64	1 207 880,87	1 491 833,26
20.	50 837,96	1 239 467,20	1 534 039,63
21.	52 241,09	1 273 237,61	1 579 147,48



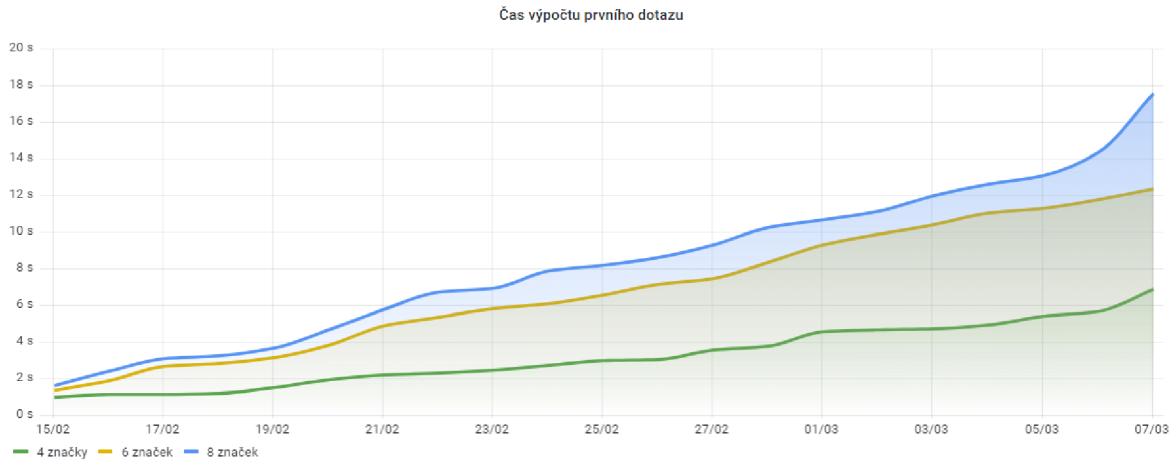
Průměrné využití paměti RAM

Den	4 značky [MB]	6 značek [MB]	8 značek [MB]
1.	255,205	496,989	506,467
2.	301,489	808,197	832,013
3.	324,274	854,785	870,747
4.	309,854	908,201	922,042
5.	286,667	967,114	987,264
6.	290,469	837,795	841,080
7.	317342	943,795	988,685
8.	341,839	983,290	1 059,138
9.	353,557	1 037,273	1 106,920
10.	353,864	1 055,321	1 092,305
11.	353,733	967,293	911,971
12.	337,442	991,207	963,474
13.	316,432	1 032,465	971,374
14.	336,098	1 070,099	1 049,947
15.	3596,15	1 094,672	1 070,699
16.	370,864	1 125,791	1 088,080
17.	371,670	1 109,830	1 086,387
18.	358,551	1 070,666	1 046,396
19.	342,096	1 100,415	1 066,023
20.	314,730	1 013,968	989,494
21.	324,687	1 077,733	1 040,549



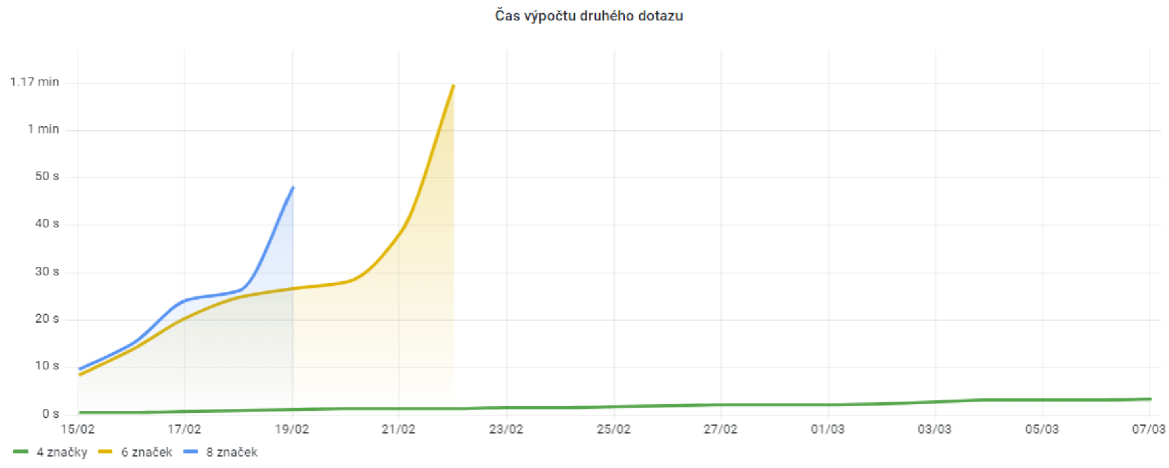
Dotaz 1 – vypsání zpoždění všech vozů linky číslo 22

Den	4 značky [s]	6 značek [s]	8 značek [s]
1.	1,16	1,36	1,62
2.	1,12	1,88	2,24
3.	1,15	2,65	3,08
4.	1,19	2,83	3,23
5.	1,51	3,12	3,64
6.	1,90	3,84	4,64
7.	2,20	4,87	5,76
8.	2,27	5,36	6,70
9.	2,45	5,80	6,93
10.	2,74	6,07	7,89
11.	2,97	6,58	8,22
12.	3,05	7,14	8,62
13.	3,54	7,48	9,29
14.	3,79	8,37	10,26
15.	4,57	9,30	10,69
16.	4,64	9,66	11,17
17.	4,72	10,39	11,97
18.	4,93	11,02	12,63
19.	5,38	11,33	13,11
20.	5,67	11,77	14,38
21.	6,86	12,36	17,57



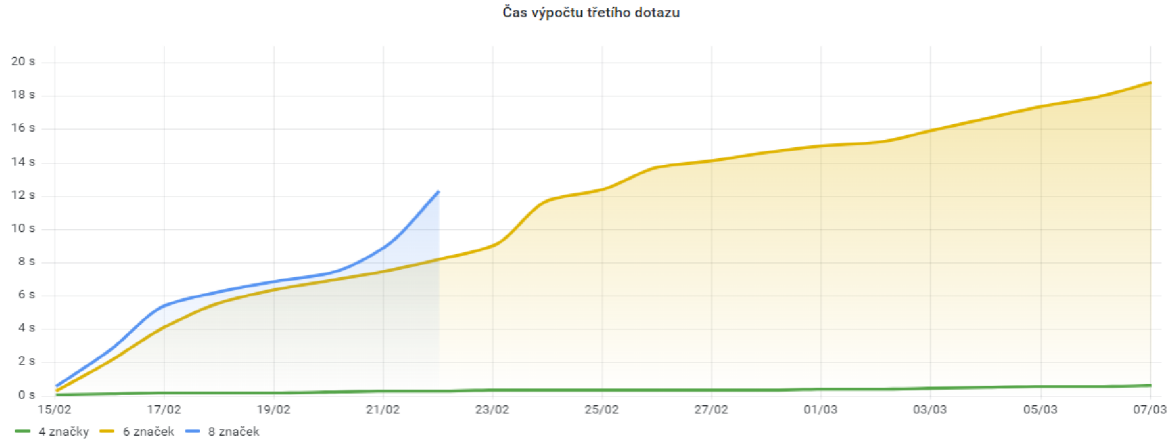
Dotaz 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty

Den	4 značky [s]	6 značek [s]	8 značek [s]
1.	0,49	8,32	9,59
2.	0,56	13,87	15,07
3.	0,70	20,63	24,30
4.	0,89	24,91	26,28
5.	1,04	26,61	48,21
6.	1,27	28,04	
7.	1,31	26,61	
8.	1,34	28,04	
9.	1,47	38,68	
10.	1,55	69,65	
11.	1,77		
12.	1,86		
13.	2,17		
14.	2,19		
15.	2,23		
16.	2,35		
17.	2,66		
18.	3,07		
19.	3,15		
20.	3,23		
21.	3,36		



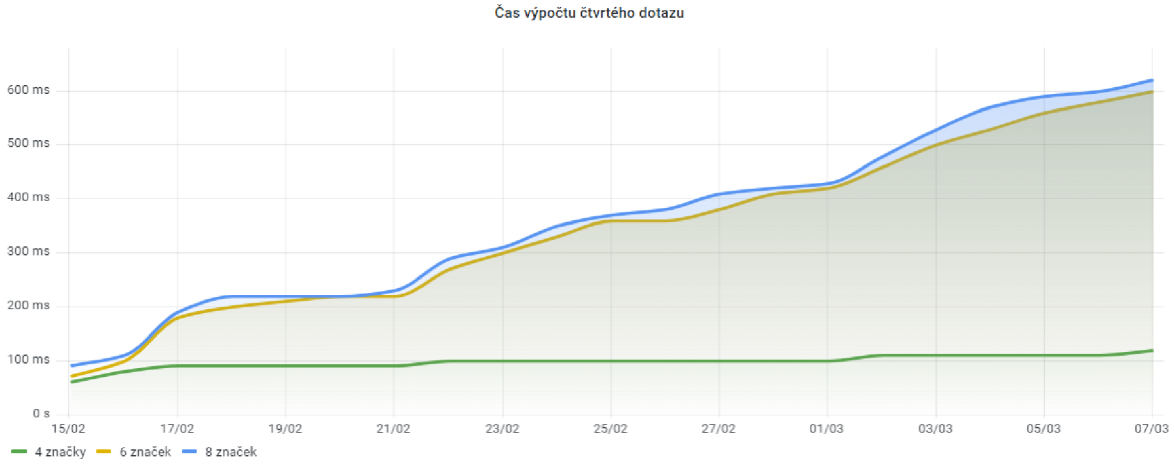
Dotaz 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty

Den	4 značky [s]	6 značek [s]	8 značek [s]
1.	0,08	0,28	0,56
2.	0,14	2,15	2,79
3.	0,17	4,19	5,45
4.	0,19	5,59	6,26
5.	0,22	6,41	6,90
6.	0,23	6,94	7,41
7.	0,28	7,51	8,95
8.	0,32	8,24	12,32
9.	0,35	9,07	
10.	0,36	11,78	
11.	0,36	12,42	
12.	0,37	13,74	
13.	0,36	14,14	
14.	0,38	14,61	
15.	0,39	15,01	
16.	0,42	15,26	
17.	0,49	15,95	
18.	0,53	16,69	
19.	0,58	17,41	
20.	0,60	17,97	
21.	0,61	18,85	



Dotaz 4 – vypsání průměrného zpoždění autobusové linky číslo 107 za jednotlivé hodiny

Den	4 značky [s]	6 značek [s]	8 značek [s]
1.	0,06	0,07	0,09
2.	0,08	0,10	0,11
3.	0,09	0,18	0,19
4.	0,09	0,20	0,22
5.	0,09	0,21	0,22
6.	0,09	0,22	0,22
7.	0,09	0,22	0,23
8.	0,10	0,27	0,29
9.	0,10	0,30	0,31
10.	0,10	0,33	0,35
11.	0,10	0,36	0,37
12.	0,10	0,36	0,38
13.	0,10	0,38	0,41
14.	0,10	0,41	0,42
15.	0,10	0,42	0,43
16.	0,11	0,46	0,48
17.	0,11	0,50	0,53
18.	0,11	0,53	0,57
19.	0,11	0,56	0,59
20.	0,11	0,58	0,60
21.	0,12	0,60	0,62



Příloha G Výsledky měření experimentu 3

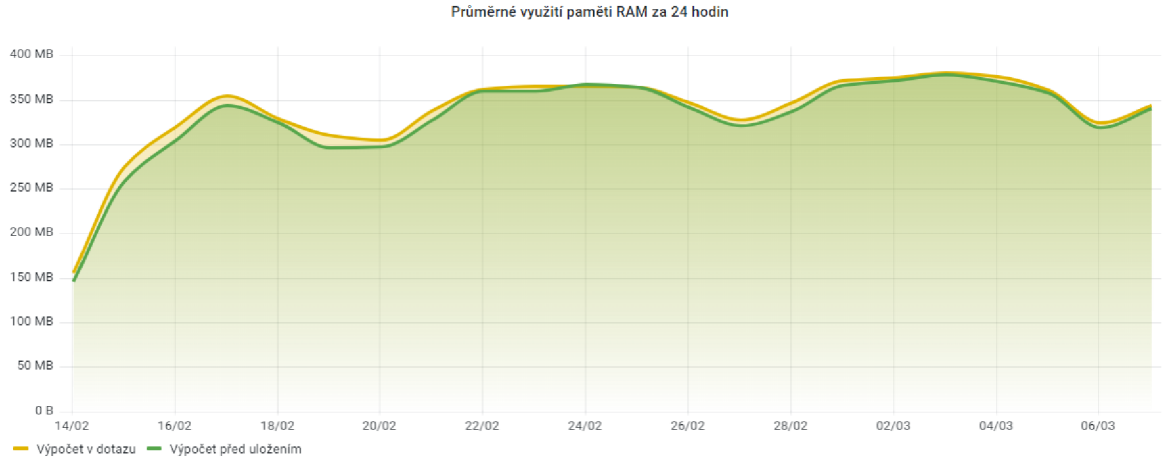
Příloha G obsahuje výsledky měření experimentu 3 – Výpočet předem vs. výpočet v dotazu.

Příloha G je přiložena na následujících stranách a také je k dispozici ve formátu PDF v přiloženém DVD a ke stažení na adrese URL:

https://drive.google.com/file/d/1DF44cE_roZAHNiH8orpwDbaRsVDd7hdZ/view?usp=share_link

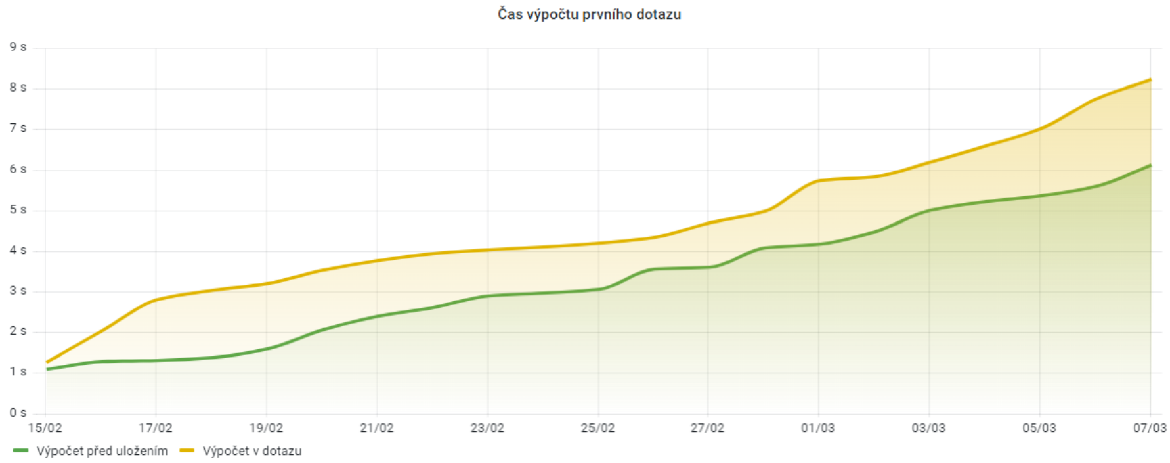
Průměrné využití paměti RAM

Den	Výpočet v dotazu [MB]	Výpočet před uložením [MB]
1.	274,642	259,138
2.	320,165	304,804
3.	354,829	343,660
4.	328,470	324,247
5.	310,819	296,786
6.	305,458	298,108
7.	338,367	328,421
8.	362,587	360,295
9.	365,528	360,707
10.	365,450	367,976
11.	365,061	364,695
12.	347,586	342,403
13.	327,715	321,050
14.	347,688	337,706
15.	372,641	366,864
16.	375,188	372,429
17.	380,508	378,509
18.	376,137	371,089
19.	361,261	357,964
20.	324,863	318,865
21.	343,704	340,417



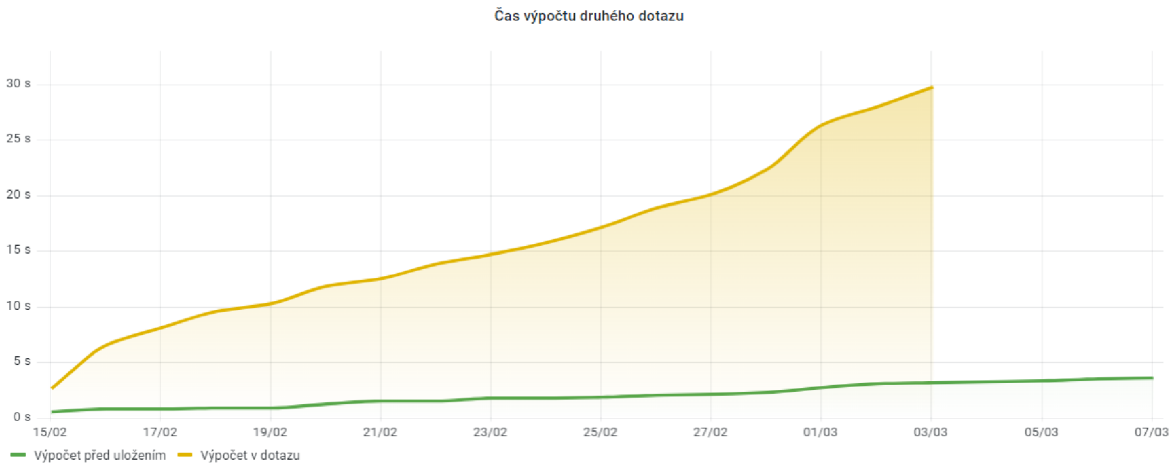
Dotaz 1 – vypsání zpoždění všech vozů linky číslo 22

Den	Výpočet v dotazu [s]	Výpočet před uložením [s]
1.	1,26	1,10
2.	2,05	1,29
3.	2,81	1,32
4.	3,05	1,38
5.	3,21	1,59
6.	3,54	2,08
7.	3,79	2,41
8.	3,96	2,63
9.	4,04	2,90
10.	4,11	2,98
11.	4,22	3,07
12.	4,35	3,56
13.	4,70	3,62
14.	4,99	4,09
15.	5,76	4,19
16.	5,85	4,50
17.	6,21	5,01
18.	6,61	5,24
19.	7,02	5,36
20.	7,76	5,61
21.	8,24	6,12



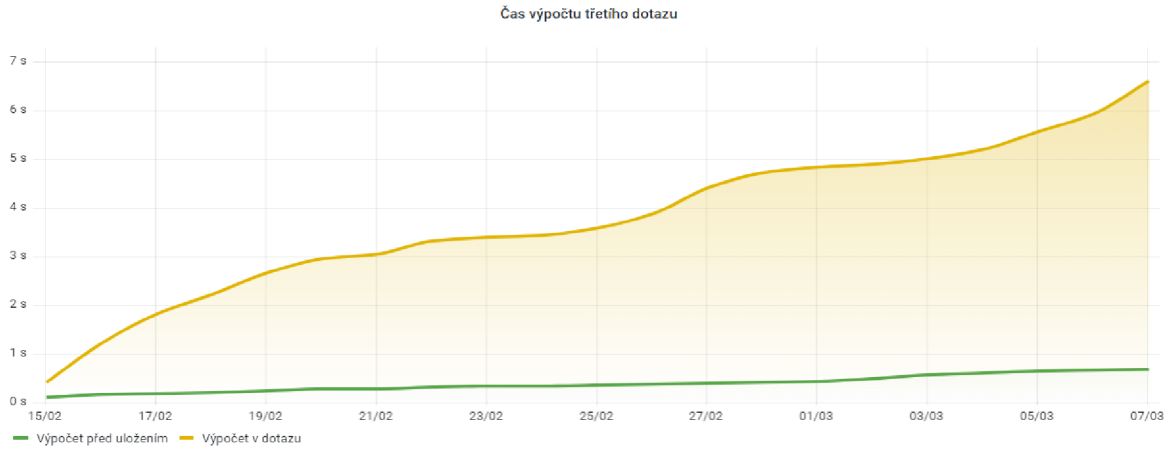
Dotaz 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty

Den	Výpočet v dotazu [s]	Výpočet před uložením [s]
1.	2,66	0,57
2.	6,58	0,79
3.	8,15	0,85
4.	9,59	0,88
5.	10,29	0,91
6.	11,89	1,25
7.	12,59	1,48
8.	13,85	1,53
9.	14,71	1,75
10.	15,75	1,80
11.	17,17	1,85
12.	18,93	2,01
13.	20,11	2,09
14.	22,40	2,34
15.	26,41	2,73
16.	28,03	3,11
17.	29,74	3,19
18.		3,29
19.		3,37
20.		3,52
21.		3,64



Dotaz 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty

Den	Výpočet v dotazu [s]	Výpočet před uložením [s]
1.	0,41	0,10
2.	1,22	0,16
3.	1,81	0,18
4.	2,22	0,21
5.	2,67	0,25
6.	2,95	0,27
7.	3,04	0,28
8.	3,31	0,31
9.	3,40	0,33
10.	3,44	0,33
11.	3,59	0,35
12.	3,87	0,37
13.	4,41	0,39
14.	4,73	0,41
15.	4,84	0,44
16.	4,90	0,49
17.	5,02	0,57
18.	5,21	0,61
19.	5,56	0,64
20.	5,94	0,66
21.	6,61	0,68



Příloha H Výsledky měření experimentu 4

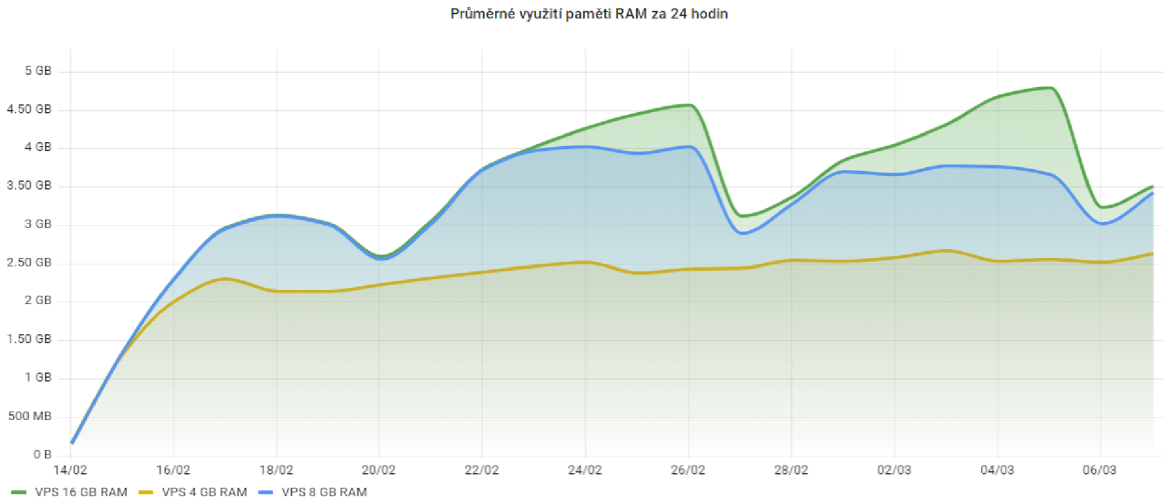
Příloha H obsahuje výsledky měření experimentu 4 – Výpočet dotazu v závislosti na paměti RAM.

Příloha H je přiložena na následujících stranách a také je k dispozici ve formátu PDF v přiloženém DVD a ke stažení na adrese URL:

https://drive.google.com/file/d/1tiXtm4ssYzGhQeeb37Te8jBkP-JnVc5e/view?usp=share_link

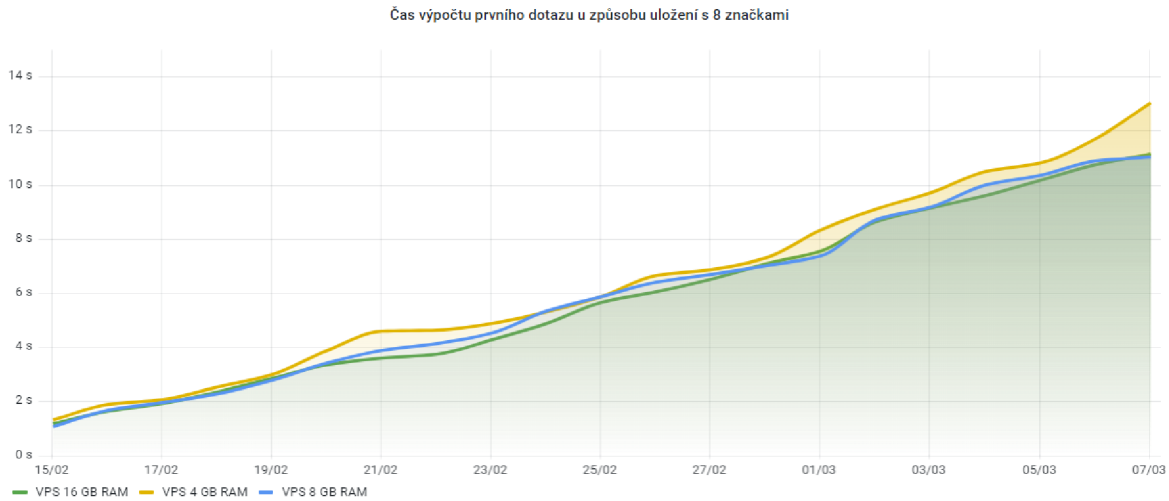
Průměrné využití paměti RAM

Den	VPS 4 GB RAM [MB]	VPS 8 GB RAM [MB]	VPS 16 GB RAM [MB]
1.	1 323,028	1 345,467	1 352,611
2.	2 018,621	2 318,649	2 320,046
3.	2 305,962	2 959,062	2 973,141
4.	2 147,891	3 123,202	3 134,812
5.	2 140,696	3 012,937	3 021,092
6.	2 232,130	2 560,858	2 593,277
7.	2 319,934	3 030,925	3 078,837
8.	2 394,195	3 727,485	3 740,234
9.	2 476,078	3 974,446	4 027,658
10.	2 520,370	4 034,289	4 271,521
11.	2 379,007	3 940,923	4 463,145
12.	2 434,107	4 028,678	4 574,441
13.	2 448,492	2 900,917	3 129,267
14.	2 548,225	3 290,828	3 377,632
15.	2 531,123	3 706,722	3 858,848
16.	2 581,002	3 667,144	4 051,536
17.	2 671,847	3 783,899	4 319,588
18.	2 533,745	3 765,621	4 688,527
19.	2 555,025	3 668,868	4 798,439
20.	2 520,285	3 030,356	3 232,712
21.	2 629,658	3 428,420	3 520,319



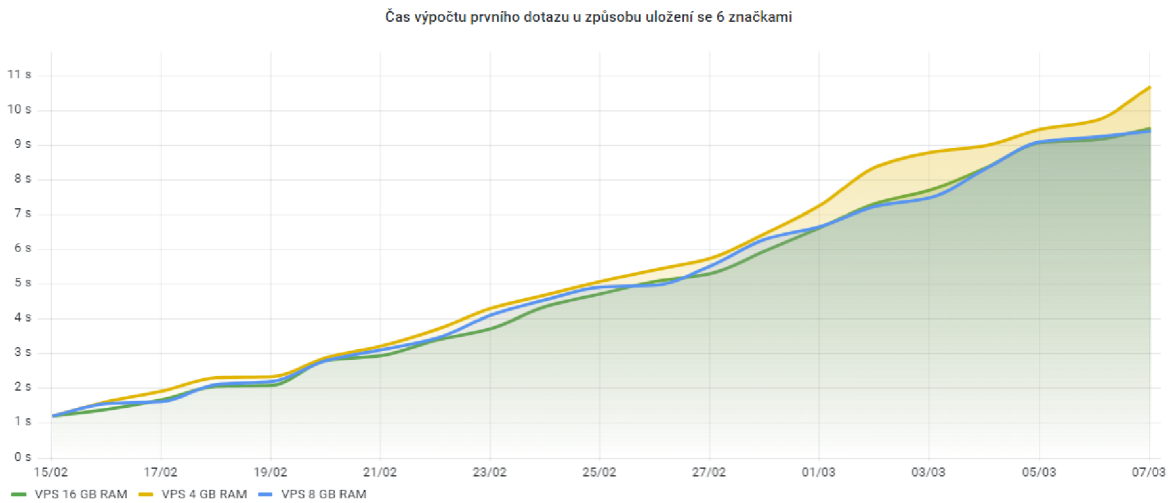
Dotaz 1 – vypsání zpoždění všech vozů linky číslo 22 u způsobu uložení s 8 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	1,32	1,08	1,19
2.	1,89	1,68	1,64
3.	2,07	1,99	1,95
4.	2,54	2,31	2,36
5.	3,01	2,78	2,86
6.	3,89	3,45	3,38
7.	4,61	3,90	3,62
8.	4,63	4,13	3,75
9.	4,91	4,55	4,29
10.	5,31	5,36	4,90
11.	5,88	5,89	5,66
12.	6,66	6,43	6,07
13.	6,88	6,70	6,53
14.	7,33	7,03	7,10
15.	8,36	7,39	7,56
16.	9,14	8,73	8,68
17.	9,74	9,21	9,16
18.	10,53	10,03	9,64
19.	10,85	10,36	10,21
20.	11,71	10,90	10,78
21.	13,04	11,04	11,16



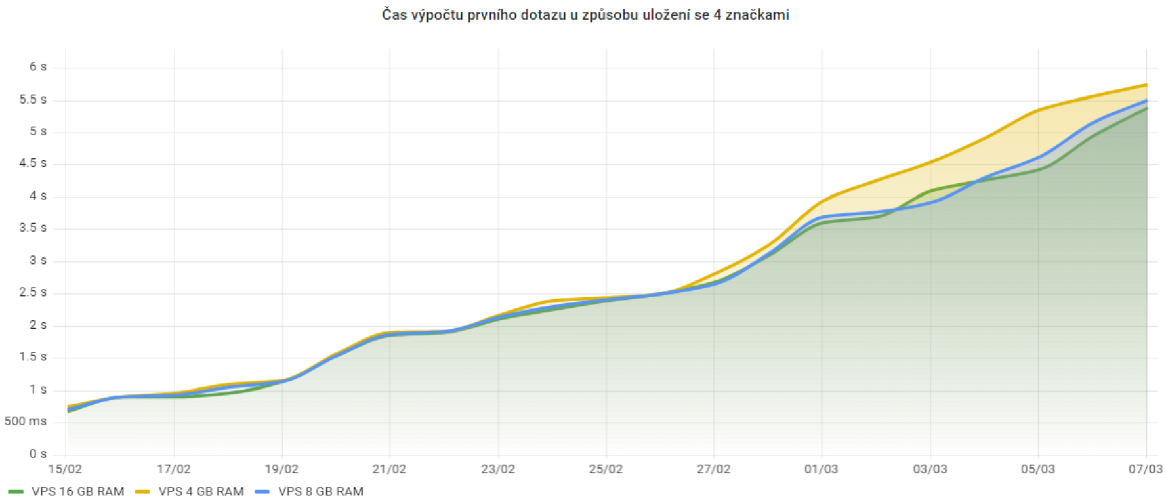
Dotaz 1 – vypsání zpoždění všech vozů linky číslo 22 u způsobu uložení se 6 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	1,20	1,20	1,20
2.	1,62	1,56	1,41
3.	1,94	1,63	1,68
4.	2,32	2,13	2,08
5.	2,34	2,21	2,10
6.	2,89	2,82	2,81
7.	3,23	3,12	2,96
8.	3,71	3,46	3,40
9.	4,33	4,11	3,73
10.	4,71	4,56	4,36
11.	5,08	4,92	4,72
12.	5,43	4,99	5,09
13.	5,76	5,53	5,31
14.	6,49	6,31	5,97
15.	7,30	6,67	6,65
16.	8,39	7,25	7,33
17.	8,82	7,51	7,72
18.	9,02	8,35	8,37
19.	9,48	9,11	9,08
20.	9,74	9,26	9,18
21.	10,70	9,43	9,51



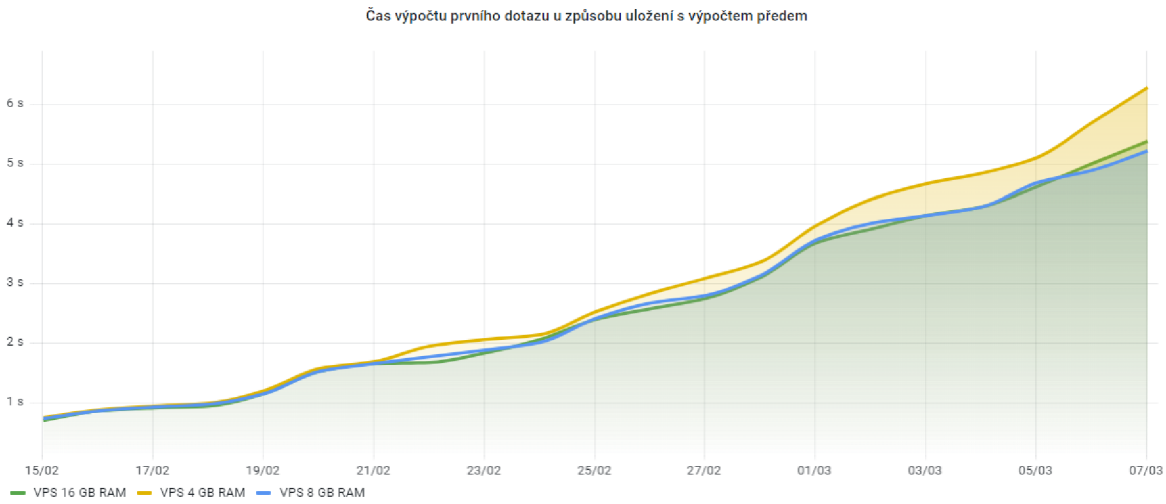
Dotaz 1 – vypsání zpoždění všech vozů linky číslo 22 u způsobu uložení se 4 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,76	0,71	0,68
2.	0,91	0,90	0,90
3.	0,96	0,93	0,91
4.	1,10	1,05	0,96
5.	1,16	1,15	1,14
6.	1,58	1,55	1,54
7.	1,90	1,88	1,86
8.	1,92	1,92	1,91
9.	2,12	2,15	2,12
10.	2,40	2,31	2,26
11.	2,44	2,42	2,40
12.	2,51	2,50	2,50
13.	2,82	2,65	2,69
14.	3,27	3,13	3,10
15.	3,96	3,70	3,61
16.	4,26	3,77	3,70
17.	4,55	3,93	4,10
18.	4,92	4,31	4,27
19.	5,36	4,62	4,43
20.	5,57	5,17	4,95
21.	5,74	5,51	5,39



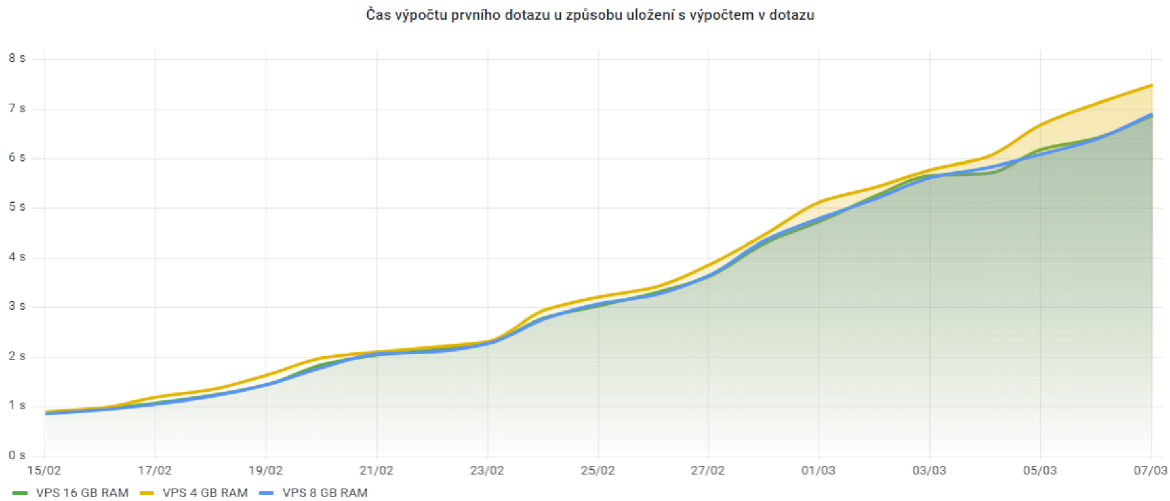
Dotaz 1 – vypsání zpoždění všech vozů linky číslo 22 u způsobu uložení s výpočtem předem

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,76	0,73	0,71
2.	0,88	0,87	0,87
3.	0,95	0,93	0,91
4.	1,00	0,98	0,95
5.	1,21	1,15	1,15
6.	1,58	1,52	1,53
7.	1,69	1,66	1,65
8.	1,94	1,77	1,68
9.	2,06	1,89	1,83
10.	2,14	2,01	2,06
11.	2,53	2,41	2,40
12.	2,84	2,67	2,58
13.	3,09	2,81	2,76
14.	3,36	3,14	3,11
15.	3,98	3,74	3,69
16.	4,41	4,01	3,92
17.	4,68	4,14	4,14
18.	4,85	4,29	4,29
19.	5,11	4,69	4,62
20.	5,71	4,90	5,01
21.	6,29	5,23	5,39



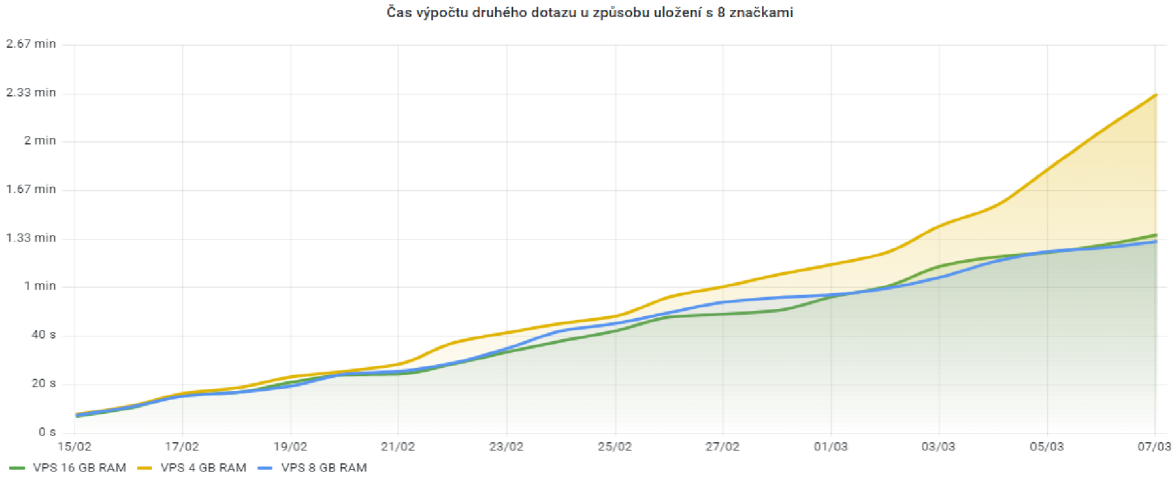
Dotaz 1 – vypsání zpoždění všech vozů linky číslo 22 u způsobu uložení s výpočtem v dotazu

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,91	0,86	0,87
2.	0,98	0,94	0,96
3.	1,20	1,06	1,08
4.	1,36	1,21	1,23
5.	1,64	1,44	1,45
6.	1,99	1,80	1,86
7.	2,12	2,08	2,06
8.	2,21	2,11	2,14
9.	2,32	2,29	2,29
10.	2,94	2,77	2,80
11.	3,21	3,09	3,04
12.	3,42	3,26	3,30
13.	3,89	3,67	3,65
14.	4,48	4,36	4,30
15.	5,15	4,81	4,75
16.	5,43	5,21	5,27
17.	5,79	5,63	5,68
18.	6,05	5,82	5,71
19.	6,70	6,10	6,20
20.	7,13	6,41	6,42
21.	7,49	6,91	6,87



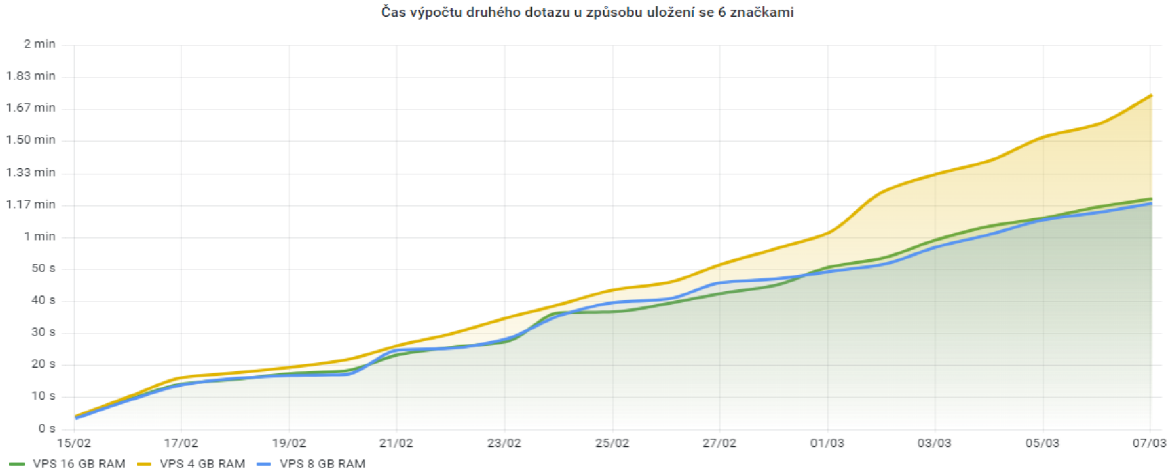
Dotaz 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty u způsobu uložení s 8 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	7,63	7,28	7,00
2.	11,11	10,85	10,33
3.	16,66	15,53	15,74
4.	18,63	16,97	17,03
5.	23,52	19,46	20,97
6.	25,58	24,44	24,01
7.	28,73	25,53	24,60
8.	37,44	28,98	28,83
9.	41,53	35,17	33,46
10.	45,46	42,16	38,21
11.	48,42	45,46	42,49
12.	56,19	50,03	48,07
13.	60,43	53,91	49,09
14.	65,42	55,91	50,65
15.	69,47	57,19	56,22
16.	74,66	59,87	60,53
17.	85,56	64,31	69,02
18.	93,56	70,95	72,57
19.	109,16	74,84	74,78
20.	125,07	76,43	77,79
21.	140,00	79,05	81,79



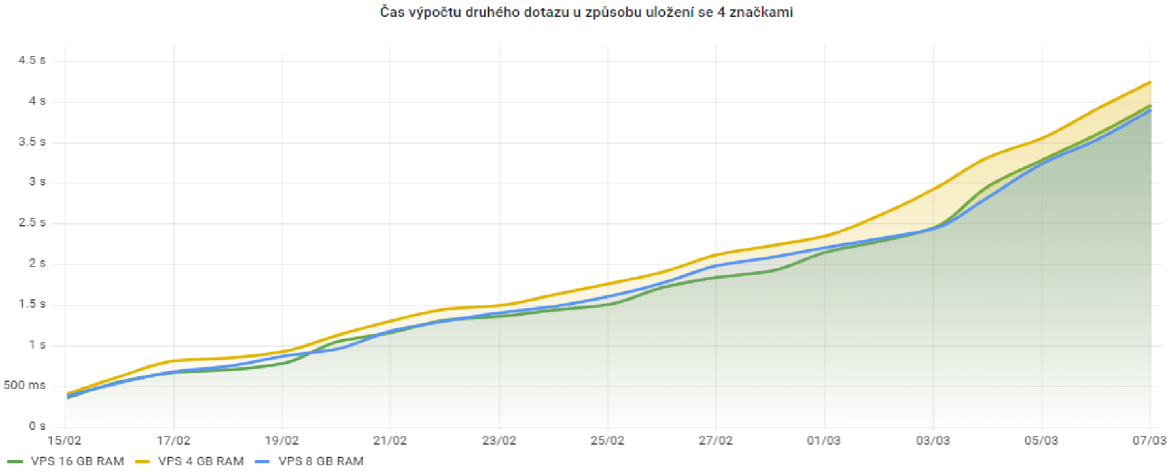
Dotaz 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty u způsobu uložení se 6 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	4,22	3,53	3,75
2.	10,37	9,12	9,51
3.	16,49	14,04	14,49
4.	17,82	15,95	15,91
5.	19,60	16,86	17,63
6.	21,91	17,23	18,25
7.	26,30	24,78	23,47
8.	29,95	25,38	25,79
9.	34,82	28,24	27,39
10.	39,09	35,65	36,46
11.	43,75	39,65	37,00
12.	45,89	40,93	39,30
13.	51,67	45,94	42,70
14.	56,41	47,18	45,00
15.	61,53	49,43	50,82
16.	74,43	51,56	53,76
17.	79,91	57,18	59,34
18.	84,08	61,09	63,57
19.	91,54	65,52	66,27
20.	95,43	67,99	69,73
21.	104,55	70,83	72,06



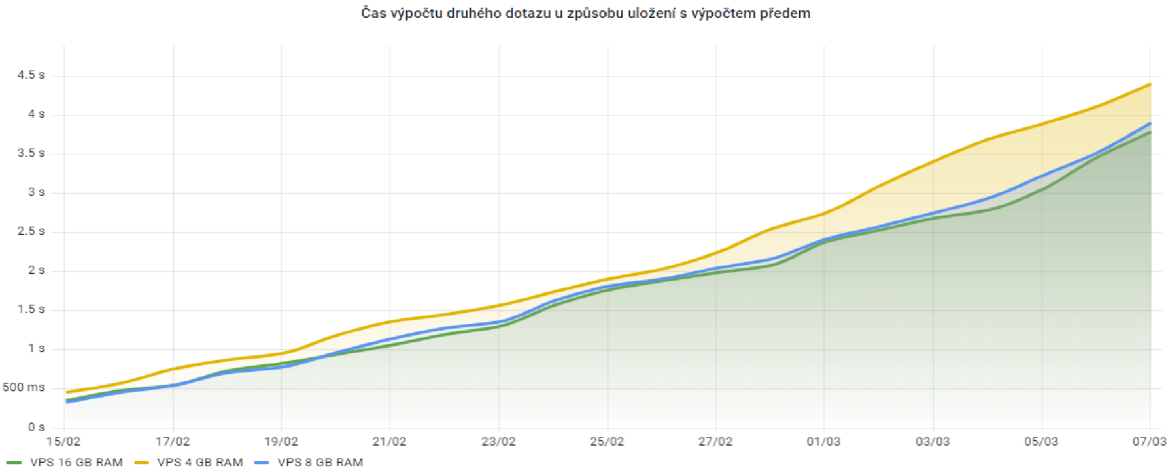
Dotaz 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty u způsobu uložení se 4 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,41	0,38	0,36
2.	0,63	0,55	0,56
3.	0,82	0,69	0,67
4.	0,85	0,75	0,71
5.	0,93	0,87	0,79
6.	1,13	0,96	1,05
7.	1,31	1,19	1,17
8.	1,46	1,31	1,32
9.	1,50	1,41	1,37
10.	1,63	1,49	1,44
11.	1,77	1,61	1,51
12.	1,91	1,78	1,72
13.	2,12	1,99	1,84
14.	2,24	2,09	1,92
15.	2,36	2,21	2,16
16.	2,61	2,32	2,29
17.	2,94	2,45	2,46
18.	3,33	2,84	2,97
19.	3,56	3,25	3,29
20.	3,92	3,54	3,61
21.	4,25	3,91	3,96



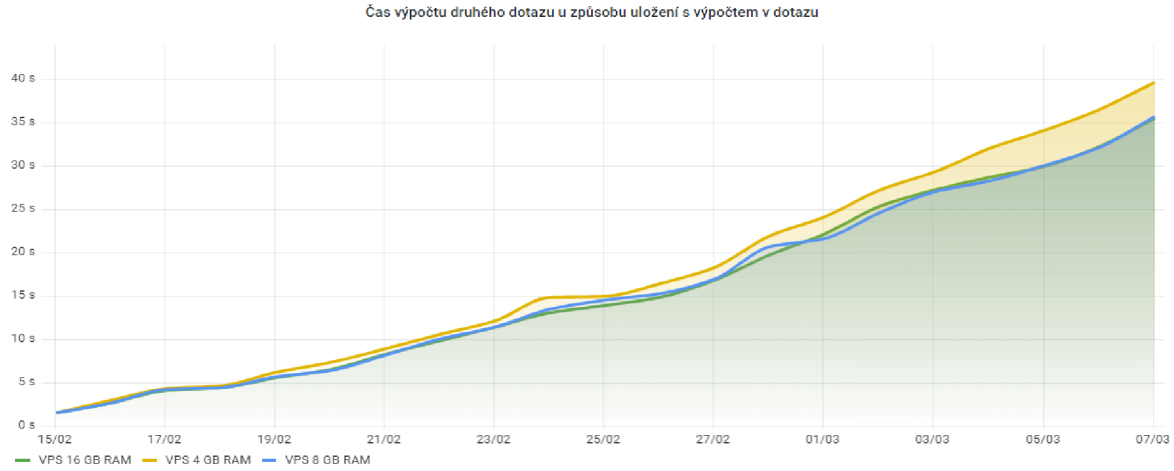
Dotaz 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty u způsobu uložení s výpočtem předem

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,46	0,33	0,36
2.	0,58	0,46	0,48
3.	0,76	0,55	0,55
4.	0,88	0,72	0,74
5.	0,96	0,79	0,83
6.	1,19	0,97	0,95
7.	1,37	1,15	1,06
8.	1,46	1,28	1,20
9.	1,58	1,37	1,31
10.	1,75	1,63	1,57
11.	1,91	1,82	1,77
12.	2,04	1,91	1,89
13.	2,25	2,05	1,99
14.	2,55	2,17	2,09
15.	2,76	2,42	2,39
16.	3,11	2,59	2,54
17.	3,42	2,76	2,69
18.	3,70	2,95	2,80
19.	3,90	3,24	3,06
20.	4,12	3,53	3,47
21.	4,41	3,91	3,80



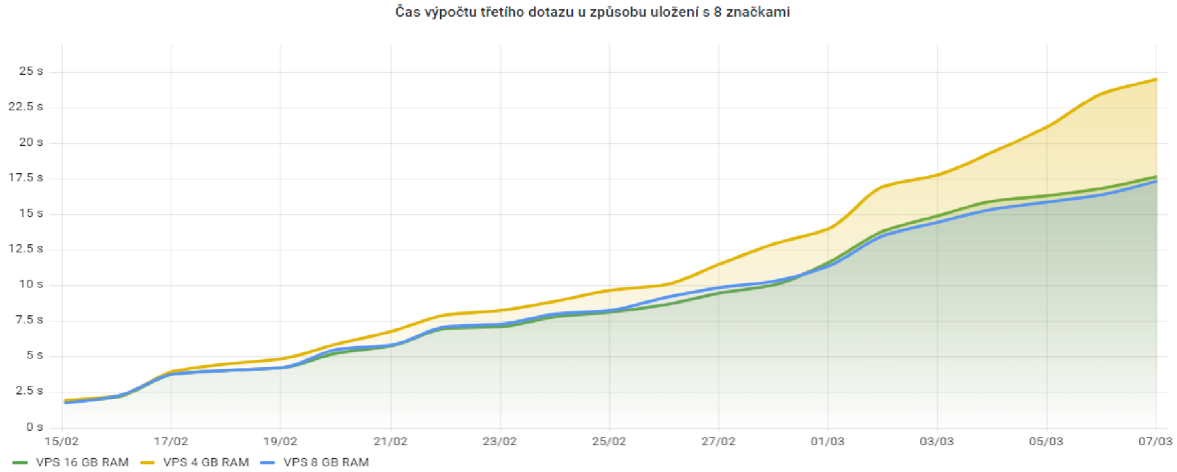
Dotaz 2 – vypsání průměrného zpoždění všech linek tramvají za jednotlivé minuty u způsobu uložení s výpočtem v dotazu

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	1,55	1,49	1,47
2.	2,99	2,63	2,71
3.	4,37	4,21	4,10
4.	4,68	4,41	4,43
5.	6,16	5,64	5,56
6.	7,32	6,39	6,54
7.	8,97	8,23	8,35
8.	10,63	10,07	9,88
9.	12,11	11,47	11,42
10.	14,84	13,51	13,05
11.	15,02	14,61	13,98
12.	16,45	15,29	14,85
13.	18,29	17,00	16,87
14.	21,93	20,71	19,78
15.	24,15	21,68	22,15
16.	27,25	24,73	25,43
17.	29,34	27,12	27,28
18.	32,06	28,36	28,80
19.	34,19	30,12	30,04
20.	36,61	32,23	32,36
21.	39,77	35,72	35,59



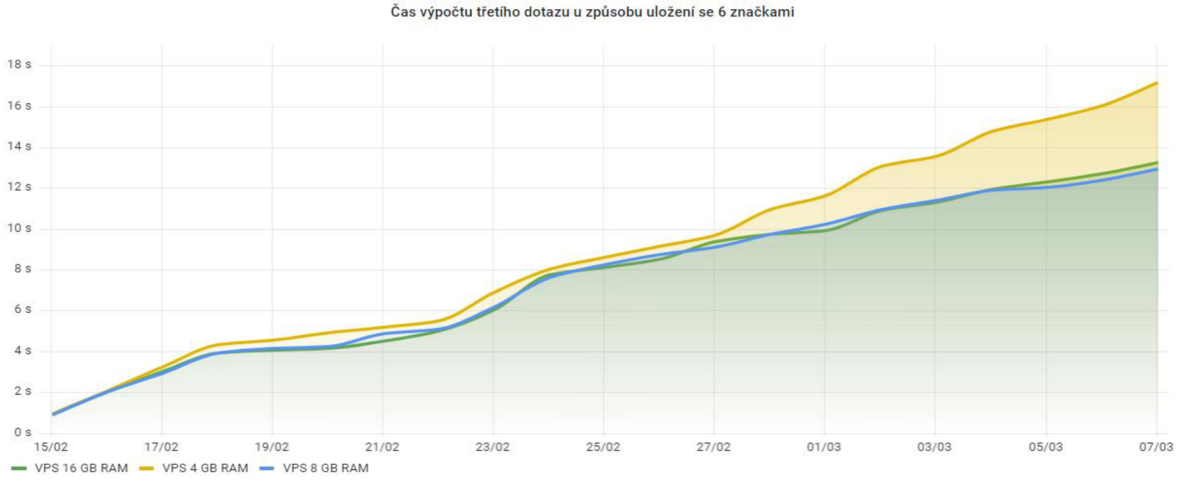
Dotaz 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty u způsobu uložení s 8 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	1,97	1,77	1,81
2.	2,27	2,26	2,24
3.	4,01	3,84	3,81
4.	4,51	4,07	4,08
5.	4,92	4,29	4,26
6.	5,90	5,53	5,28
7.	6,82	5,89	5,81
8.	7,95	7,17	7,01
9.	8,32	7,32	7,14
10.	8,92	8,06	7,86
11.	9,69	8,32	8,17
12.	10,11	9,17	8,68
13.	11,54	9,89	9,51
14.	12,96	10,34	10,11
15.	14,07	11,42	11,70
16.	17,00	13,56	13,86
17.	17,87	14,50	14,93
18.	19,42	15,41	15,97
19.	21,22	15,93	16,35
20.	23,52	16,43	16,85
21.	24,56	17,39	17,69



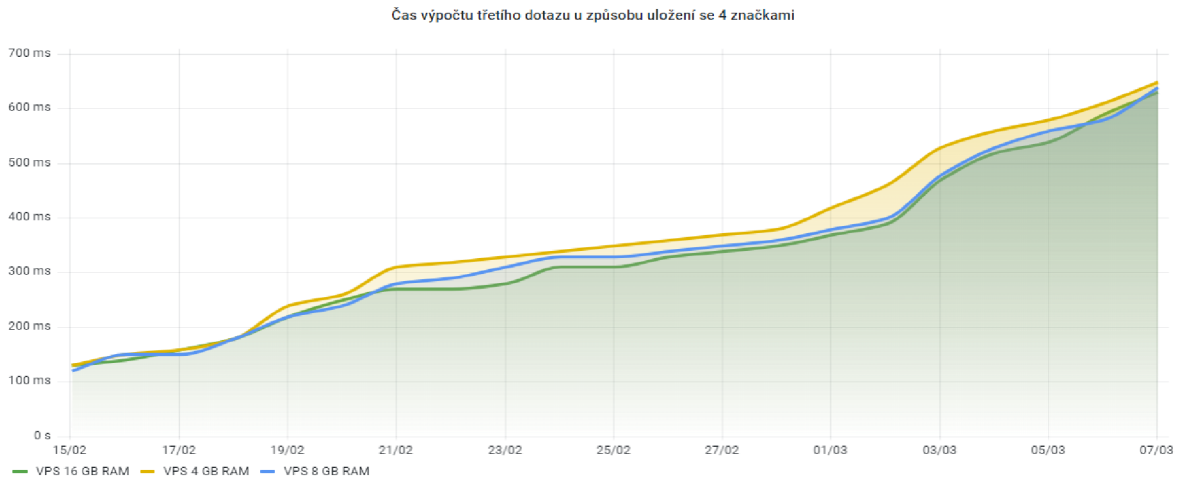
Dotaz 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty u způsobu uložení se 6 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,98	0,91	0,93
2.	2,08	2,05	2,06
3.	3,27	2,95	3,04
4.	4,33	3,94	3,96
5.	4,56	4,16	4,06
6.	4,93	4,24	4,16
7.	5,22	4,89	4,55
8.	5,51	5,13	5,04
9.	6,90	6,19	6,07
10.	8,03	7,62	7,79
11.	8,61	8,27	8,14
12.	9,17	8,78	8,56
13.	9,72	9,11	9,38
14.	10,98	9,78	9,75
15.	11,66	10,24	9,93
16.	13,11	10,99	10,91
17.	13,60	11,43	11,33
18.	14,81	11,92	11,96
19.	15,39	12,08	12,33
20.	16,07	12,42	12,75
21.	17,19	12,98	13,27



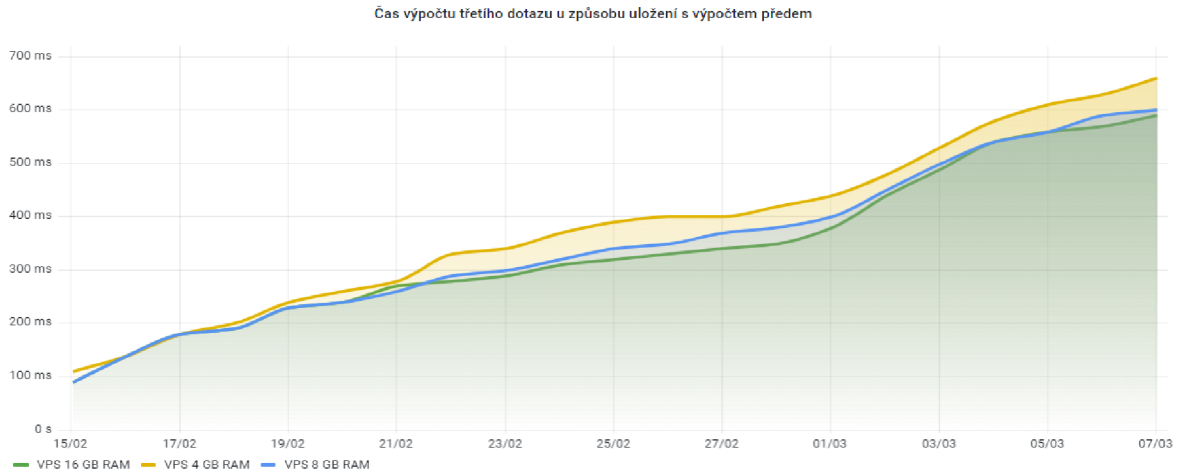
Dotaz 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty u způsobu uložení se 4 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,13	0,12	0,13
2.	0,15	0,15	0,14
3.	0,16	0,15	0,16
4.	0,18	0,18	0,18
5.	0,24	0,22	0,22
6.	0,26	0,24	0,25
7.	0,31	0,28	0,27
8.	0,32	0,29	0,27
9.	0,33	0,31	0,27
10.	0,34	0,33	0,28
11.	0,35	0,33	0,31
12.	0,36	0,34	0,31
13.	0,37	0,35	0,33
14.	0,38	0,36	0,34
15.	0,42	0,38	0,35
16.	0,46	0,40	0,37
17.	0,53	0,48	0,39
18.	0,56	0,53	0,47
19.	0,58	0,56	0,52
20.	0,61	0,58	0,54
21.	0,65	0,64	0,63



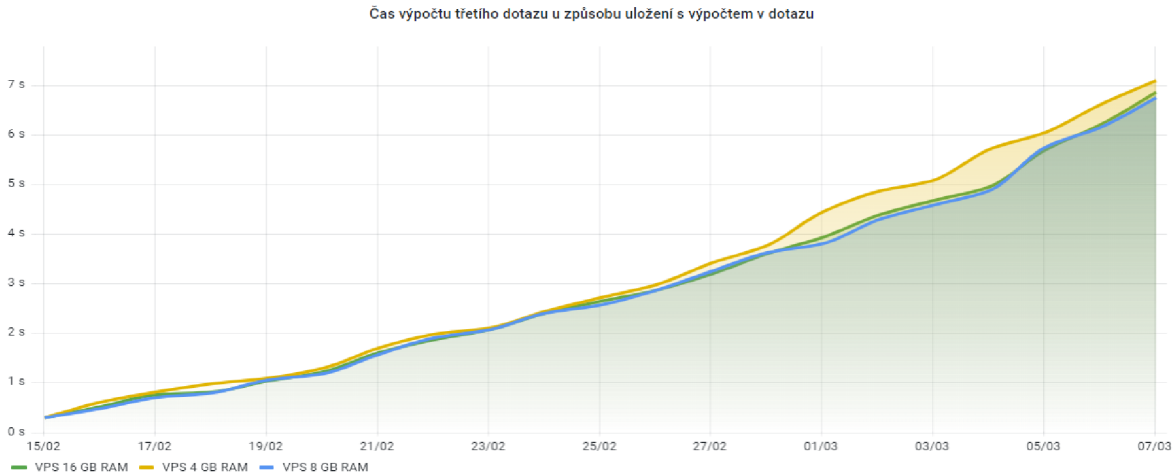
Dotaz 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty u způsobu uložení s výpočtem předem

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,11	0,09	0,09
2.	0,14	0,14	0,14
3.	0,18	0,18	0,18
4.	0,20	0,19	0,19
5.	0,24	0,23	0,23
6.	0,26	0,24	0,24
7.	0,28	0,26	0,27
8.	0,33	0,29	0,28
9.	0,34	0,30	0,29
10.	0,37	0,32	0,31
11.	0,39	0,34	0,32
12.	0,40	0,35	0,33
13.	0,40	0,37	0,34
14.	0,42	0,38	0,35
15.	0,44	0,40	0,38
16.	0,48	0,45	0,44
17.	0,53	0,50	0,49
18.	0,58	0,54	0,54
19.	0,61	0,56	0,56
20.	0,63	0,59	0,57
21.	0,66	0,60	0,59



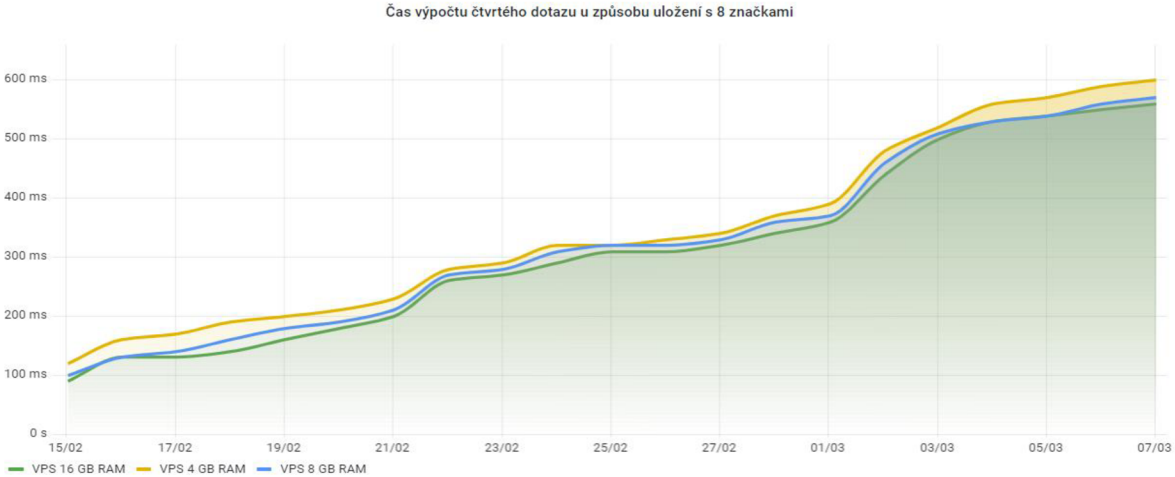
Dotaz 3 – vypsání průměrného zpoždění tří linek tramvaje, konkrétně linek číslo 11, 17 a 22, za jednotlivé minuty u způsobu uložení s výpočtem v dotazu

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,31	0,30	0,30
2.	0,62	0,49	0,53
3.	0,82	0,72	0,76
4.	0,99	0,81	0,83
5.	1,11	1,06	1,05
6.	1,31	1,20	1,23
7.	1,71	1,59	1,62
8.	1,99	1,92	1,88
9.	2,11	2,08	2,08
10.	2,46	2,41	2,41
11.	2,73	2,59	2,65
12.	2,98	2,88	2,87
13.	3,43	3,27	3,21
14.	3,79	3,63	3,61
15.	4,46	3,83	3,96
16.	4,87	4,31	4,39
17.	5,10	4,59	4,70
18.	5,72	4,89	4,97
19.	6,06	5,76	5,71
20.	6,63	6,17	6,22
21.	7,11	6,76	6,88



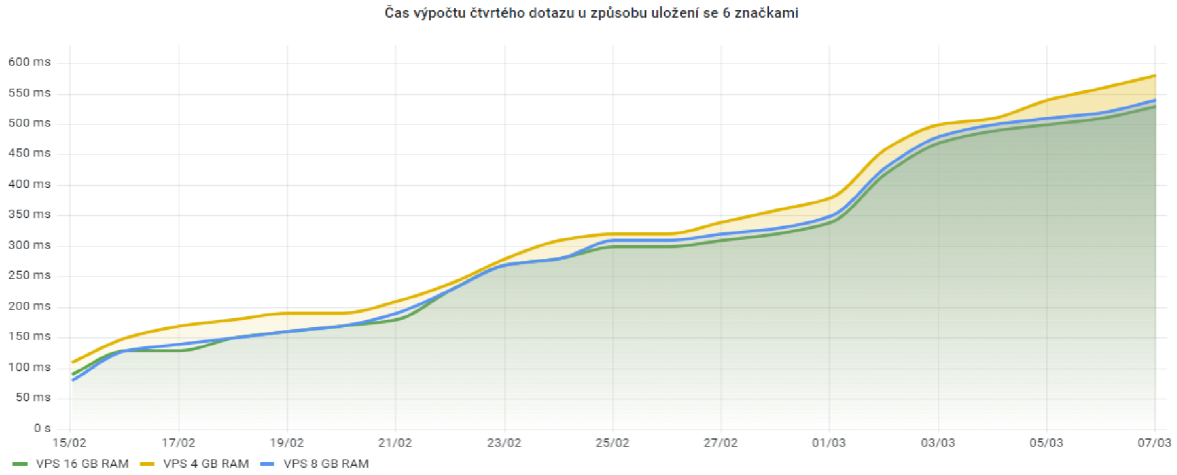
Dotaz 4 – vypsání průměrného zpoždění autobusové linky číslo 107 za jednotlivé hodiny u způsobu uložení s 8 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,12	0,10	0,09
2.	0,16	0,13	0,13
3.	0,17	0,14	0,13
4.	0,19	0,16	0,14
5.	0,20	0,18	0,16
6.	0,21	0,19	0,18
7.	0,23	0,21	0,20
8.	0,28	0,27	0,26
9.	0,29	0,28	0,27
10.	0,32	0,31	0,29
11.	0,32	0,32	0,31
12.	0,33	0,32	0,31
13.	0,34	0,33	0,32
14.	0,37	0,36	0,34
15.	0,39	0,37	0,36
16.	0,48	0,46	0,44
17.	0,52	0,51	0,50
18.	0,56	0,53	0,53
19.	0,57	0,54	0,54
20.	0,59	0,56	0,55
21.	0,60	0,57	0,56



Dotaz 4 – vypsání průměrného zpoždění autobusové linky číslo 107 za jednotlivé hodiny u způsobu uložení se 6 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,11	0,08	0,09
2.	0,15	0,13	0,13
3.	0,17	0,14	0,13
4.	0,18	0,15	0,15
5.	0,19	0,16	0,16
6.	0,19	0,17	0,17
7.	0,21	0,19	0,18
8.	0,24	0,23	0,23
9.	0,28	0,27	0,27
10.	0,31	0,28	0,28
11.	0,32	0,31	0,30
12.	0,32	0,31	0,30
13.	0,34	0,32	0,31
14.	0,36	0,33	0,32
15.	0,38	0,35	0,34
16.	0,46	0,43	0,42
17.	0,50	0,48	0,47
18.	0,51	0,50	0,49
19.	0,54	0,51	0,50
20.	0,56	0,52	0,51
21.	0,58	0,54	0,53



Dotaz 4 – vypsání průměrného zpoždění autobusové linky číslo 107 za jednotlivé hodiny u způsobu uložení se 4 značkami

Den	VPS 4 GB RAM [s]	VPS 8 GB RAM [s]	VPS 16 GB RAM [s]
1.	0,06	0,06	0,06
2.	0,07	0,07	0,06
3.	0,07	0,07	0,06
4.	0,07	0,07	0,06
5.	0,07	0,07	0,06
6.	0,08	0,07	0,07
7.	0,09	0,08	0,07
8.	0,09	0,08	0,08
9.	0,09	0,09	0,08
10.	0,09	0,09	0,08
11.	0,09	0,09	0,08
12.	0,10	0,09	0,08
13.	0,10	0,09	0,09
14.	0,10	0,09	0,09
15.	0,10	0,09	0,09
16.	0,11	0,10	0,09
17.	0,11	0,10	0,10
18.	0,11	0,10	0,10
19.	0,11	0,10	0,10
20.	0,11	0,10	0,10
21.	0,12	0,11	0,10