Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém na správu periodických časových radov

Bakalárska práca

2017
Filip Janitor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Systém na správu periodických časových radov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Robert Lukoťka, PhD.

Bratislava, 2017
Filip Janitor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Filip Janitor
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Systém na správu periodických časových radov
System for management of periodic time series

Cieľ: Cieľom bakalárskej práce je porovnať a vybrať technológie a navrhnúť kľúčové algoritmy pre systém na prácu s periodickými časovými radmi. Periodický časový rad je funkcia, ktorá časovej hodnote, ktorá je násobkom zvolenej periody, priradí hodnotu. Z hľadiska technológií a algoritmov sú kľúčové požiadavky na systém nasledovné:
- Systém ukladá a aktualizuje periodické časové rady a poskytuje ich klientom.
- Systém je schopný efektívne agregovať veľké množstovo časových radov (súčet, priemer, medián, 95%-ny percentil, a. i.) a poskytovať ich agregáty klientom v reálnom čase.
- Systém má zvládať jednoduché operácie na časových radoch (napr. súčet, posun) a skladať ich. Výsledky má poskytovať klientom v reálnom čase.
- Systém umožňuje vidieť časové rady v stave k danému historickému dátumu.
- Systém má byť škálovateľný.

Vedúci: RNDr. Robert Lukoťka, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 31.10.2016

Dátum schválenia: 03.05.2017 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Touto cestou by som sa rád poďakoval môjmu školiteľovi RNDr. Robertovi Lukoťkovi, PhD. za venovaný čas, odborné rady a ústretovosť. Taktiež sa chcem poďakovať mojej rodine za trpezlivosť a podporu.

Abstrakt

V práci skúmame vlastnosti systémov na správu časových radov. Poskytujeme všeobecný prehľad existujúcich systémov a ich charakteristík a popisujeme nami vykonaný experiment, zameraný na zistenie výkonu vybraných systémov pri vykonávaní agregácií. Na základe získaných výsledkov náčrtávame kľúčové dátové štruktúry a princípy, ktoré sú použiteľné pri návrhu systému na správu časových radov, vhodného do moderného prostredia.

Kľúčové slová: časové rady, databáza časových radov, noSQL

Abstract

This thesis is dedicated to examining properties of time series management systems. We provide a general overview of existing systems and their characteristics and describe conducted experiment that was focused on comparing aggregation performance of selected systems. Based on obtained results we outline key data structures and principles that can be used during design of time series management system.

Keywords: time series, time series database, noSQL

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Súčasnosť sa vyznačuje zhromažďovaním enormného množstva dát. Množstvo týchto dát má charakter časových radov, keďže pochádza zo sledovania určitých javov v čase. Tieto dáta môžu pochádzať zo senzorov používaných vo vede a výskume, monitorovacích systémov sledujúcich javy vo finančnom alebo reálnom svete okolo nás, alebo môžu vznikať v spotrebičoch, ktoré budú onedlho súčasťou internetu vecí. Dáta, ktoré majú charakter časových radov, sú pre nás užitočné a majú veľký význam, pretože nám umožňujú nahliadnuť do minulosti, nachádzať súvislosti medzi javmi, analyzovať trendy a vytvárať predpovede. Keďže sa takýchto dát vytvára mnoho, je dnes viac ako kedykoľvek predtým potrebné disponovať systémami, ktoré sú tieto dáta časových radov schopné efektívne ukladať, perzistentne uchovávať, spracúvať a kategorizovať. Samotné ukladanie však nie je jediná dôležitá vlastnosť týchto potrebných systémov. Zo zhromaždených dát potrebujeme získať relevantné štatistiky, a teda ich potrebujeme agregovať. Kvôli potenciálne veľkému množstvu spracúvaných dát sa nám javí vhodnejšie nepresúvať ich v nespracovanej podobe, ale úlohu vykonávania agregácií a analýz delegovať priamo na systém, ktorý dáta uchováva. Preto je dôležité, aby systémy na správu časových radov poskytovali sadu silných agregácií a štatistických nástrojov.

Ako sme spomenuli, systémy na správu časových radov majú uplatnenie aj vo vede a výskume, kde prebieha systematické pozorovanie a zhromažďovanie údajov o určitých javoch. Najmä v tejto oblasti sú dáta získané z meraní mimoriadne cenné, a preto je potrebné ich všetky dôkladne uchovávať. Pri takomto použití je však dôležité, aby sa systém vedel vysporiadať s chybami merania a korekciami, no aby sa žiadne, ani chybné dáta nestratili. Uchovávanie chybných dát je dôležité napríklad kvôli opakovanému vykonávaniu určitých analýz a ich auditovateľnosti. Niektoré moderné databázové systémy na tieto požiadavky reagujú tým, že umožňujú uchovávať historické verzie dát. Túto funkcionalitu považujeme za dôležitú aj v špecializovaných systémoch na správu časových radov.

V tejto práci sa venujeme trom hlavným oblastiam, týkajúcim sa systémov na správu časových radov. V prvom rade robíme všeobecný prehľad systémov, ktoré sú dnes k dispozícii spolu s ich charakteristikami, hlavnými princípmi fungovania a technikami, ktoré používajú na zefektívnenie svojej práce. V práci ďalej popisujeme vykonaný experiment, ktorý porovnáva vybrané existujúce systémy za účelom preskúmania ich

schopnosti poskytovať agregácie. Treťou oblasťou ktorej sa venujeme, je náčrt kľúčových princípov, dátových štruktúr a algoritmov, ktoré môže systém na správu časových radov použiť. Pri príprave tohto náčrtu sme brali do úvahy poznatky nadobudnuté pri práci v predchádzajúcich oblastiach.

Kapitola 1 obsahuje prehľad pojmov, ktoré sa v ďalších častiach práce vyskytujú. Kapitoly 2 a 3 sú venované všeobecnému prehľadu základných princípov návrhu a fungovania systémov na správu časových radov. Poznatky obsiahnuté v týchto kapitolách sú využité v kapitole 4, ktorá popisuje vybrané existujúce systémy na správu časových radov. Uvádza tiež v skratke príklady systémov, ktoré nevynikajú špeciálnymi vlastnosťami, a tak sme ich do užšieho výberu nedali. V kapitole 5 popisujeme vhodné predpoklady a požiadavky na moderný systém na správu časových radov. Niektoré z vybraných systémov, popísaných v kapitole 4, poskytujú možnosť vykonávať zložité agregácie, ktoré nás pri náčrte systému zaujímajú. Zaradili sme ich teda do experimentálneho porovnania, ktorého popis a výsledky sa nachádzajú v kapitole 6. Na základe poznatkov získaných zo skúmaných systémov sme identifikovali a načrtli kľúčové algoritmy a dátové štruktúry pre systém na správu časových radov, spĺňajúci naše požiadavky z kapitoly 5. Tieto poznatky sú spísané v kapitole 7.

Kapitola 1

Časové rady a operácie s nimi

V tejto kapitole sú uvedené vysvetlenia základných pojmov, ktoré sú použité v práci, vzhľadom k skutočnosti, že rôzne publikácie používajú niektoré z týchto pojmov v iných významoch.

Existuje viacero definícií časového radu, niektoré z nich v sebe zahŕňajú aj rozličné rozširujúce predpoklady. V práci sú pojem časový rad a ďalšie s ním súvisiace pojmy používané vo význame, uvedenom v nasledujúcich definíciách.

Na prácu s časovými radmi používame reprezentáciu časových údajov v počítači, ktorú nazývame *časová pečiatka*. Množinu časových pečiatok označujeme $\mathcal{T}$. Platí, že nie každý čas dokážeme reprezentovať časovou pečiatkou. Existuje teda čiastočné zobrazenie $h\colon \mathcal{C} \to \mathcal{T}$, kde $\mathcal{C}$ je množina všetkých časových údajov. Nad množinou $\mathcal{C}$ existuje úplné usporiadanie a taktiež aj nad $\mathcal{T}$ existuje úplné usporiadanie. Platí pritom, že ak $a, b \in D(h)$ a zároveň $a < b$, tak aj $h(a) < h(b)$. Definujme nasledujúce symboly:

$$\begin{aligned} (-\infty, b)_h &= \{h(x) | x < b\} \\ \langle a, b)_h &= \{h(x) | x \in \langle a, b)\} \\ \langle a, \infty)_h &= \{h(x) | x \geq a)\} \end{aligned} \tag{1.1}$$

Tieto symboly budeme nazývať *intervaly časových pečiatok*.

Reprezentáciu hodnôt v počítači, ktoré používame pri práci s časovými radmi, nazývame *údajmi*. Množinu prípustných údajov označujeme $\mathcal{U}$. V niektorých prípadoch je vhodné, aby kvôli vzájomnej logickej súvislosti údaj obsahoval informácie o viacerých veličinách. Údaje časového radu zaznamenávajúceho úpravy zdieľaného dokumentu môžu napríklad obsahovať identifikátor upravovateľa, rozsah úprav, čas strávený upravovaním, adresu počítača, z ktorého bol dokument upravovaný a podobné informácie. Údaj je v takomto prípade mapou a jednotlivé dvojice <kľúč,hodnota> tejto mapy nazývame *polia*. Ak systém umožňuje, aby údaj bol mapou, hovoríme, že takýto systém na správu časových radov *podporuje polia*.

Definícia 1.0.1. *Časovým radom nazývame čiastočnú funkciu* $f\colon \mathcal{T} \to \mathcal{U}$.

Definícia 1.0.2. *Záznam časového radu f je dvojica $(c, f(c))$.*

V rámci systému je potrebné vedieť jednotlivé časové rady dobre identifikovať a kategorizovať. Okrem názvu môže byť preto vhodné priradiť každému časovému radu množinu dvojíc <kľúč,hodnota>. Jednotlivé dvojice z tejto množiny v takomto prípade nazývame *tagy*. Výhody použitia tagov ilustruje nasledujúci príklad.

Systém uchováva časové rady obsahujúce údaje o stave procesorov v clusterovej federácií. Konkrétny časový rad musí byť priraditeľný ku konkrétnemu procesoru, a teda je potrebné uchovať informáciu o jeho polohe v rámci federácie clusterov, identifikáciu serveru v clusteri a informáciu určujúcu o ktorý konkrétny procesor v rámci serveru ide. Taktiež musí byť zrejmé, akú veličinu časový rad zaznamenáva. Môže to byť teplota, záťaž procesora a pod. Ak na identifikáciu časového radu, obsahujúceho údaje o teplote procesoru 0 v serveri 6 v clusteri 1 použijeme iba názov, ako vhodné sa javí použitie formátovacej konvencie „cluster1.node6.cpu0.teplota “. Hoci je tento tvar pomerne zrozumiteľný, nie je vhodný v prípade, že je potrebné vytvárať dopyty na rôzne množiny časových radov. Napríklad dopyt na množinu všetkých časových radov obsahujúcich údaje o teplote zo serverov 1 a 5 v clusteri číslo 4 nie je jednoduché vytvoriť len pomocou reťazcových operácií. Naproti tomu, v systéme podporujúcom tagy môžeme pomenovať tento rad ako „teplota“ a priradiť mu množinu tagov {cluster=1 , node=6 , cpu=0}. Uvedený dopyt je potom možné jednoducho vyjadriť v pseudodopytovacom jazyku podobnom SQL ako „SELECT teplota WHERE (node=1 OR node=5) AND cluster=6“

V prípade, že systém umožňuje priradiť časovému radu množinu tagov hovoríme, že systém *podporuje tagy*. Vzhľadom na identifikačný účel tagov v takomto systéme na správu časových radov ďalej predpokladáme, že množina tagov, priradená konkrétnemu časovému radu, je mu priradená v momente jeho vytvorenia a ďalej sa nemení. Predpokladáme, že množina tagov spolu s názvom vytvárajú unikátny identifikátor časového radu v rámci systému.

Definícia 1.0.3. *Aktualizácia časového radu je akákoľvek operácia, ktorá zmení hodnotu údaju resp. údajov časového radu, alebo pridá nový záznam.*

Definícia 1.0.4. *Rozlíšenie záznamov v systéme je najmenší podporovaný rozdiel časov, zodpovedajúci dvom za sebou idúcim časovým pečiatkam.*

Pri práci s časovými radmi v systéme často používame agregácie, aby sme z uložených údajov získali pre nás podstatné informácie. Poznáme hodnotové a radové agregácie.

Definícia 1.0.5. *Hodnotová agregácia je funkcia $g\colon (c_1, c_2, \ldots c_n) \to u$, kde $(c_1, c_2, \ldots c_n)$ je n-tica časových radov a u je údaj.*

Príkladom hodnotovej agregácie môže byť napríklad zistenie priemernej teploty z nameraných teplôt za posledný týždeň.

Definícia 1.0.6. *Radová agregácia je funkcia* $g\colon (c_1, c_2, \dots c_n) \to r$, *kde* $(c_1, c_2, \dots c_n)$ *je n-tica časových radov a r je časový rad.*

Príkladom radovej agregácie je napríklad derivácia alebo vývoj priemernej spotreby vody v mestskej časti, kde každý údaj výsledného radu je priemerom údajov v tom čase zo všetkých meračov v danej mestskej časti.

Agregácie delíme aj podľa počtu členov vstupnej n-tice časových radov.

Definícia 1.0.7. *Agregácia cez jeden časový rad je agregácia, kde* $n = 1$.

Definícia 1.0.8. *Agregácia cez viacero časových radov je agregácia, kde* $n > 1$.

V praxi sa používa ešte jeden špeciálny prípad radovej agregácie cez jeden časový rad. Takúto agregáciu nazývame *agregácia po skupinách (bucket alebo group aggregation)*. V prípade tejto operácie rozložíme množinu $\mathcal{T}$ na intervaly časových pečiatok. Každému z intervalov rozkladu priradíme časovú pečiatku (reprezentanta) tak, že reprezentant $y_i \in \langle x_i, x_{i+1})$. Reprezentantom jednotlivých intervalov priradíme údaje, ktoré vzniknú agregovaním zúženia vstupného časového radu na nimi reprezentované intervaly rozkladu hodnotovou agregáciou. Časový rad, ktorý týmto získame, je výsledkom agregácie po skupinách.

Definícia 1.0.9. *Spracúvaním či správou časových radov označujeme zber údajov a vytváranie záznamov, resp. zber záznamov, ich ukladanie, organizovanie a sprístupňovanie.*

Medzi funkcie systému na správu časových radov môže patriť aj schopnosť systému upozorniť na určitý jav, týkajúci sa údajov časových radov. Javom môže byť napríklad prudký pokles alebo nárast hodnôt údajov v prichádzajúcich záznamoch sledovaného časového radu. Ak systém podporuje túto funkcionalitu, hovoríme, že systém *podporuje upozorňovanie (alerting)*.

Ak systém umožňuje automaticky mazať záznamy časového radu s časovou pečiatkou, ktorej vzdialenosť od prítomnosti je väčšia ako vopred zvolená hodnota, hovoríme, že systém *podporuje expiráciu údajov*.

Kapitola 2

Vlastnosti systémov na správu časových radov

Systémy na správu časových radov majú rôzne použitie a nasadzujú sa v rôznych prostrediach. V závislosti od typu spracúvaných dát a oblasti použitia sa rôznia požiadavky a potreby kladené na konkrétny systém. Na základe existujúcich požiadaviek je možné zúžiť zameranie systému a tým, vďaka vhodným optimalizáciám, zvýšiť jeho výkon pre konkrétne použitie. Cieľom tejto kapitoly je poskytnúť všeobecný prehľad rôznych predpokladov a požiadaviek na systém na správu časových radov. Poznatky, ktoré sú obsiahnuté v tejto kapitole sme nadobudli štúdiom systémov, ktoré popisujeme v kapitole 4.

2.1 Agregácie

Systém na správu časových radov môže byť použitý v oblastiach s potrebou vykonávať na zozbieraných dátach množstvo rôznych analýz. Preto môže byť výhodné delegovať úlohy vykonávania určitých výpočtov, vedúcich k potrebným výsledkom, priamo na samotný systém, prípadne jeho serverovú časť, aby nedochádzalo k veľkému prenosu nespracovaných dát. Z tohoto dôvodu má zmysel požadovať, aby systém vedel vykonávať nad dátami, ktoré obsahuje, rôzne agregácie. Problematika kategorizácie agregácií je detailnejšie popísaná v kapitole 1.

2.2 Priestor

V závislosti od očakávaného rozsahu zhromažďovaných dát je potrebné zaoberať sa požiadavkami na priestorovú efektivitu systému. Pri menších aplikáciách môže byť výhodné využiť nedistribuovaný systém. Kvôli potenciálnej nemožnosti takýto systém horizontálne škálovať môže byť potrebné využiť rôzne optimalizácie, ktoré za cenu zní-

ženia rýchlosti alebo straty presnosti uložených dát, môžu priniesť výraznú úsporu potrebného priestoru. Za predpokladu, že množstvo spracúvaných dát má tendenciu rýchlo narastať a prekračuje možnosti systému s jedným serverom, je potrebné použiť systém podporujúci automatické horizontálne škálovanie pre pohodlné rozširovanie serverového clusteru.

2.3 Zápis a čítanie dát

Od systému, ktorý je určený na správu veľkého množstva časových radov očakávame, že je schopný naraz zapisovať veľké množstvo údajov. Podobne môžeme pre potreby analýz vyžadovať schopnosť systému poskytovať nám v čo najkratšom čase výsledky rôznych agregácií alebo priamo veľký objem nespracovaných (raw) dát. Teda v závislosti od konkrétnej aplikácie môžeme požadovať od systému zameranie sa viac na čítanie alebo viac na zápis. Systém musí byť schopný vyrovnať sa tiež so situáciami, plynúcimi z technických obmedzení spojenia s odosielateľmi dát. Môže sa jednať napríklad o oneskorovanie príchodu dát alebo predbiehanie sa správ s dátami, kedy správa obsahujúca záznam so skoršou časovou pečiatkou príde neskôr ako časovo nasledujúci záznam, ktorý už systém pravdepodobne spracoval. Správy sa môžu po ceste aj stratiť alebo nemusia byť v dôsledku poruchy odosielateľa vôbec poslané. Rozhodnutia vyplývajúce z technických obmedzení distribuovaných systémov budú popísané neskôr.

2.4 Historické verzie dát

V niektorých aplikáciách môže byť potrebné uchovávať všetky historické verzie údajov, ktoré sa v časových radoch menili a aktualizovali. V takom prípade musí byť použitý systém schopný meniť dáta z minulosti, teda musí vedieť dáta do časového radu vkladať nie len na jeho koniec (v závislosti od plynutia času) ale aj upravovať už uložené záznamy z minulosti. Pri požiadavke na uchovávanie historických verzií dát musí byť systém tiež schopný rekonštruovať a poskytovať všetky predchádzajúce verzie časového radu, to znamená stavy v čase pred aktualizáciami minulosti.

2.5 Požiadavky týkajúce sa distribuovaných riešení

Ak systém umožňuje horizontálne škálovanie, je nutné rozhodnúť sa v otázkach, ktoré sa týkajú každého distribuovaného databázového riešenia, nie len systému na správu časových radov.

Jedno z týchto rozhodnutí je zakomponované v CAP teoréme [87, 77]. Táto teoréma hovorí, že žiadny distribuovaný systém nemôže naraz poskytovať všetky tri garancie z

nasledujúceho zoznamu.

- **Konzistencia (consistency)** Systém je konzistentný práve vtedy, keď odpoveď na každý dopyt obsahuje správne (najaktuálnejšie) dáta alebo hlásenie o chybe.
- **Dostupnosť (availability)** Systém garantuje dostupnosť práve vtedy, keď každý dopyt dostane nejakú odpoveď (nie hlásenie o chybe).
- **Odolnosť voči rozdeleniu (partition tolerance)** Systém je odolný voči rozdeleniu práve vtedy, keď zostáva pracovať aj napriek strate ľubovoľného množstva správ v rámci spojení medzi jeho uzlami.

Iný pohľad na distribuované systémy poskytujú vlastnosti výťažok (harvest) a výnos (yield). [85]

Definícia 2.5.1. *Výnos označuje pravdepodobnosť, že systém odpovie na dopyt.*

Definícia 2.5.2. *Výťažok vyjadruje úplnosť, aktuálnosť a korektnosť dát v odpovediach na dopyt.*

Výnos je teda percento dopytov, ktoré boli vybavené a bolo na ne odpovedané, zatiaľčo výťažok je percento dát v odpovediach, ktoré boli požadované a korektne odoslané.

Tieto dve vlastnosti spolu úzko súvisia. Napríklad pri výskyte vnútorných chýb je potrebné rozhodovať sa, či systém uprednostní na nejaký dopyt neodpovedať (znižuje sa výnos ale výťažok zostáva rovnaký) alebo odpovedať neúplnou resp. nie presnou či neaktuálnou odpoveďou (výnos sa nemení ale znižuje sa výťažok). Pri navrhovaní distribuovaných systémov je dôležité zameriavať sa na hodnoty týchto veličín. Ďalšie informácie k týmto vlastnostiam sa nachádzajú v článku od autora Coda Hale [89].

2.6 Periodicita

Očakávané časové rady v prostredí nasadenia systému môžu mať rôzne vlastnosti týkajúce sa periodicity. V prvom rade je možné vysloviť predpoklad, že jeden časový rad je periodický v zmysle pravidelnosti jeho aktualizácií. Táto vlastnosť znamená, že ak abstrahujeme od porúch odosielateľa a strát správ na linkách spájajúcich zdroj údajov časového radu a systém, rozdiel medzi dvoma časovými pečiatkami ľubovoľných dvoch za sebou idúcich záznamov časového radu je vždy rovnaký. Takýto predpoklad umožňuje ušetriť úložný priestor, pretože nie je potrebné ukladať ku každému údaju časovú pečiatku. Na základe časovej pečiatky začiatku radu a offsetu konkrétneho údaju je možné k ľubovoľnému údaju jeho časovú pečiatku jednoducho dopočítať.

Iný, silnejší predpoklad týkajúci sa periodicity spája väčšie množstvo časových radov. V takom prípade tieto rady okrem toho, že jednotlivo spĺňajú vlastnosť z predchádzajúceho odseku, majú aj navzájom rovnakú frekvenciu aktualizácie. Príkladom oblasti, v ktorej má tento predpoklad opodstatnenie je systém zbierajúci dáta z množiny senzorov, z ktorých každý má rovnakú frekvenciu odosielania meraní. Každý senzor má priradený svoj časový rad a vzniknutá množina časových radov spĺňa popisovanú vlastnosť. V takejto situácií je možné predpoklad ešte zosilniť a rozšíriť. Systém môže vyžadovať, aby neboli časové rady voči sebe navzájom fázovo posunuté. V predchádzajúcej ilustrácií to môže znamenať, že všetky senzory z množiny budú spustené naraz, a tak budú merania všetkých prebiehať v rovnakých časoch (vzniknuté záznamy pri spoločnom meraní budú mať rovnaké časové pečiatky).

2.7 Nemenná minulosť

V určitých prípadoch je možné predpokladať, že historické dáta časových radov sa nikdy nebudú meniť. Slabšia forma tohoto predpokladu umožňuje dopĺňať chýbajúce záznamy z minulosti (táto potreba môže vzniknúť chybou prenosu, oneskorením doručenia dát alebo výpadkom odosielateľa). Silnejšia forma vyžaduje, aby akékoľvek prichádzajúce záznamy ľubovoľného časového radu mali časovú pečiatku s neskoršou hodnotou ako má pečiatka posledného spracovaného záznamu v danom rade. Chybám prenosu dát sa prakticky nedá zabrániť, tento predpoklad však umožňuje systému ignorovať prichádzajúci záznam, ktorý bol v dôsledku chyby prenosu predbehnutý.

2.8 Typy údajov

Typy údajov, ktoré je potrebné v systéme ukladať, je niekedy možné vopred obmedziť. Častokrát môže byť potrebné ukladať len číselné hodnoty v určitom rozsahu a s určitou presnosťou. Toto obmedzenie môže byť napríklad vhodné pri systéme, ktorý je určený na zaznamenávanie vývoju cien akcií. Vopred známe typové obmedzenia môžu niektoré časti návrhu a optimalizácie systému uľahčiť. V iných prípadoch nie je možné dátové typy údajov takýmto spôsobom vopred určiť, pretože systém podporuje polia, alebo umožňuje ako údaj ukladať ľubovoľný objekt.

2.9 Životnosť údajov

V niektorých prípadoch je možné vopred určiť mieru dôležitosti dát v závislosti od ich veku. Na základe toho je možné historické dáta uchovávať v agregovanej podobe a tým šetriť priestor. Príkladom môže byť situácia, keď používame senzor, ktorý každú minútu

posiela do systému číselnú informáciu o svojom stave. Pre potreby presných analýz je nutné uchovávať všetky namerané údaje z posledných desiatich dní. Pre záznamy staršie ako desať dní nám stačí uchovávať iba jednu hodnotu pre každú hodinu. Vhodná hodnota môže byť napríklad priemerom údajov za daný časový interval, v tomto prípade hodinu. Staré záznamy môžeme zo systému vymazať a nahradiť ich agregovanými, ktorých je menej. V niektorých prípadoch dokonca nie je potrebné uchovávať ani agregované staré údaje a staré záznamy je možné priamo mazať.

Kapitola 3

Spôsoby uchovávania časových radov

Účelom tejto kapitoly je poskytnúť ucelený prehľad rôznych spôsobov uchovávania dát časových radov. Popisujeme aj rôzne možnosti optimalizácií práce systému na správu časových radov, využívajúceho noSQL DBMS. Rozoberané metódy sú popísané abstraktne, nezávisle od konkrétnych dostupných riešení. Poznatky uvedené v tejto kapitole sme získali štúdiom systémov z kapitoly 4. V nej sa na princípy, popísané v tejto kapitole odvolávame.

3.1 Riešenia nevyužívajúce DBMS

V tejto časti popíšeme techniky uchovávania časových radov, ktoré pracujú so súbormi v súborovom systéme bez použitia DBMS. Pre riešenie krátkodobých problémov, napríklad potreby sledovania čiastkových výstupov testovaného programu stačí často využiť jednoduché dátové štruktúry a ukladanie do súborov bez špeciálnej vnútornej hierarchie. Takéto riešenia sú však nevyhovujúce na použitie v prípadoch, ak potrebujeme pracovať s veľkým množstvom dát, garantovať ich uchovanie a aj ich vhodne organizovať. Problémy, ktoré vznikajú pri takýchto riešeniach v reálnom použití, sú rozoberané v publikácií o systéme OpenTSDB [84].

Rýchly vývoj na poli súborových systémov v posledných rokoch spôsobil, že dostupné súborové systémy majú vlastnosti, ktoré pôvodne zabezpečovali DBMS. Súborové systémy a úložiská dnes vedia zabezpečiť prácu v distribuovanom režime, replikáciu, copy-on-write funkcionalitu, ochranu voči poruchám hardvéru či možnosť vytvárať záznamy stavu (snapshots). V prípade vytvorenia vhodného dátového modelu spolu s použitím takýchto súborových systémov je možné vytvoriť výkonný systém na správu časových radov. Takéto systémy sa líšia od riešení popísaných v predchádzajúcom odseku vo využívaní špeciálnych formátov súborov a plného využívania nástrojov poskytnutých súborovým systémom.

Príkladmi pokročilých súborových systémov a úložísk, ktoré sú reálne využívané v

systémoch na správu časových radov, popísaných v kapitole 4 sú napríklad ZFS [69], Ceph [68] či HDFS [29].

Špeciálny druh riešení, ktorý často nevyužíva DBMS sú riešenia využívajúce princíp round-robin t.j. dáta, v tomto prípade záznamy časových radov, ukladajú do kruhového bufferu a najstaršie dáta nahrádzajú novými. Táto vlastnosť ich predurčuje na riešenie záležitostí s krátkym trvaním a na spracúvanie časových radov, ktorých historické hodnoty netreba vôbec uchovávať. Hoci v niektorých oblastiach sú veľmi vhodné, vo všeobecnosti sa práve kvôli tejto vlastnosti nedajú využiť.

3.2 Riešenia využívajúce relačné DBMS

Oproti predchádzajúcim spomenutým sú tieto riešenia založené na skutočných databázových systémoch, a teda vo všeobecnosti poskytujú oveľa silnejšie garancie perzistentnosti a efektívnosti. Podľa organizácie dát v tabuľkách môžeme deliť relačné DBMS na dva typy.

- **Riadkové** DBMS využívajúce tento model sú vo všeobecnosti rozšírenejšie. Dáta v tabuľkách sú fyzicky ukladané po riadkoch.
- **Stĺpcové (column-store)** DBMS využívajúce tento model majú údaje v tabuľkách uložené po stĺpcoch - t.j. fyzicky sú pri sebe zhromaždené údaje z jedného stĺpca a nie riadku. [97]

Systém spracúvajúci časové rady založený na relačnom DBMS môže používať bežne využívaný hviezdicový (star schema) design. Databáza v takom prípade obsahuje dimenzionálnu tabuľku obsahujúcu identifikátory a detaily o spracúvaných časových radoch a inú, tzv. tabuľku faktov, obsahujúcu samotné záznamy spolu s identifikátormi časových radov, ku ktorým patria.

Ukazuje sa však, že databázové systémy používajúce relačný model a SQL rozhranie sú príliš všeobecné a mnoho z funkcií, ktoré poskytujú, v skutočnosti systém pre prácu s časovými radmi nevyužije. Taktiež majú vo všeobecnosti problém s horizontálnym škálovaním. Toto obmedzenie sa snažia vyriešiť moderné newSQL systémy, no stále zostávajú v platnosti ich obmedzenia vyplývajúce zo silných garancií (napríklad vyžadovanie schémy dát) a používaného SQL rozhrania, ktoré sú však nevyužiteľné v prípade, že používame niektoré z optimalizácií - napríklad zníženie počtu stĺpcov s komprimáciou. Takisto, niektoré operácie špecifické pre časové rady sú pomocou SQL rozhrania veľmi ťažko vykonateľné. Napríklad derivácia časového radu, ktorá odčítava za sebou idúce záznamy, downsampling, pri ktorom je potrebné v GROUP BY klauzule dopytu vyjadriť časové intervaly alebo agregácia cez niekoľko fázovo posunutých radov, kde je potrebné interpolovať hodnoty niektorých z nich. Hoci pre projekty menšieho

rozsahu je možné dosiahnuť postačujúci výkon aj s využitím RDBMS, z dlhodobého hľadiska takéto rozhodnutie neprináša výhody [84].

3.3 Riešenia využívajúce noSQL DBMS

S príchodom a popularizáciou noSQL databázových systémov sa ukázalo, že ich vlastnosti ich robia vhodnými kandidátmi na uchovávanie a spracúvanie dát, ktoré majú charakter časových radov. Ako bude uvedené v kapitole 4, mnohé z relevantných existujúcich špecializovaných riešení používajú práve noSQL DBMS.

Označenie noSQL [88] sa v dnešnej dobe používa najmä na pomenovanie databázových systémov, ktorých dátový model je iný ako relačný. Tieto systémy častokrát na rozdiel od relačných systémov používajúcich SQL rozhranie negarantujú ACID vlastnosti transakcií. Namiesto prísnych garancií sa zameriavajú na značné uľahčenie horizontálnej škálovateľnosti, flexibilnosť dátových schém a jednoduchosť dizajnu. Pojem noSQL je veľmi široký a označuje systémy rôznych druhov, z ktorých viaceré sú vhodné aj na prácu s časovými radmi a popisujeme ich v nasledujúcich častiach.

Key-value databázy

Key-value databázy [39, 79] sú najjednoduchšou formou noSQL databázových systémov. Databázou je v ich prípade jednoduché asociatívne pole, teda páry <kľúč,hodnota>, kde hodnota môže byť ľubovoľný binárny objekt (binary large object - BLOB). Tieto databázové systémy poskytujú možnosť vkladať nové páry a tiež mazať a vyhľadávať dáta, ak je známy ich kľúč. Jednoduchosť základného dátového modelu umožňuje týmto systémom dosiahnuť vo vhodných aplikáciách vysoký výkon.

Dokumentové databázy

Dokumentové databázy [96, 79] sú prirodzeným rozšírením key-value databáz. Hlavným rozdielom medzi nimi je, že zatiaľ čo pri key-value databázach samotný databázový systém nedokáže pracovať s dátami uloženými vo value, dokumentové systémy vyžadujú, aby bol tieto dáta uložené vo vopred určenom formáte, napríklad JSON, XML alebo BSON. Tento formát umožňuje systému rozumieť, čo sa nachádza v dátach a tak je možné dopyty robiť nielen na dokumenty pomocou ich kľúču (tak ako je to v key-value databázach), ale aj na dáta v rámci dokumentu. Okrem formátu však systém nekladie na schému dokumentov žiadne požiadavky.

Databázových systémov týchto dvoch typov existuje pomerne mnoho a majú rôzne vlastnosti. Vo všeobecnosti nepodporujú ACID princípy a transakčnosť, no často garantujú aspoň atomickosť zápisu v rámci jedného záznamu, teda dokumentu resp. key-value páru. Niektoré systémy za určitých podmienok umožňujú pracovať v čiastočne

transakčnom režime, napríklad tým, že vedia zabezpečiť izoláciu, ale nie atomickosť prebiehajúcich transakcií [46]. Iné systémy, napríklad BerkleyDB [52],CouchDB [17], HyperDex [66], Aerospike [1], BergDB [8], LevelDB [42], OrientDB [54], RavenDB [64] a MarkLogic [105] poskytujú podporu ACID transakčnosti, niektoré však s rôznymi obmedzeniami.

Napriek absencii podpory klasických izolovaných transakcií je možné v niektorých prípadoch iba prostredníctvom interakcie s databázou zabezpečiť detekciu konfliktov krátkych transakcií (tento princíp používa systém Arctic [3]). Iné systémy poskytujú klientovi možnosť riešiť prípadné konflikty spôsobené tým, že viacerí klienti naraz zapíšu do toho istého záznamu [17], alebo umožňujú zamykať rôzne veľké logické celky obsahu [70]. Spomínané systémy sa rôznia aj z hľadiska poskytovania možnosti aktualizácie uložených záznamov. Niektoré používajú zámky na kontrolu prístupu a umožňujú meniť obsah uložených záznamov. Iné pracujú bez zámkov a pri každej úprave vytvoria na disku nový záznam.

Key-value systémy často poskytujú možnosť usporiadania kľúčov [79] a sú optimalizované na dopyty, ktoré prechádzajú cez určitý rozsah za sebou idúcich kľúčov v usporiadaní. Dokumentové databázy zas častokrát podporujú vytváranie indexov na určité polia dokumentov za účelom zrýchlenia dopytov, používajúcich tieto polia [79, 47].

Niektoré systémy podporujú databázové generovanie unikátnych kľúčov alebo poskytujú sadu funkcií, ktoré upravujú hodnotu poľa v dokumente tak, aby systém porozumel tejto operácií a vedel v jej prípade poskytnúť určité garancie. Sú to napríklad funkcie inkrement, dekrement, append do poľa a podobne [48].

Spomínané NoSQL systémy sa zameriavajú na využívanie denormalizovaných dát, aby sa minimalizovala potreba prechádzať cez záznamy v databáze pomocou uložených referencií v jednotlivých záznamoch. Načítavanie záznamov z databázy a spájanie množín záznamov (analógia join funkcionality v RDBMS) sú totiž pomalé operácie. Oba druhy popisovaných systémov majú vstavaný limit na veľkosť jednotlivých záznamov, pričom toto obmedzenie býva rádovo v stovkách kB až desiatkach MB.

Stĺpcovo orientované (column-family) databázy

Hoci sa pojem stĺpcové databázy často vyskytuje v súvislosti s noSQL DBMS, v skutočnosti je tento názov nejednoznačný a označujú sa ním dve nesúvislé a úplne rozdielne množiny databázových systémov. Na ich odlíšenie budeme používať označenie navrhované v článku, ktorý publikoval Daniel Abadi [72]. V tejto časti neuvažujeme o takzvaných stĺpcových (column-store) systémoch, ktoré sú založené na relačnom dátovom modeli. Popisujeme iba takzvané stĺpcovo orientované (column-family store) systémy, ktoré operujú nad viacdimenzionálnou mapou dát, ktorá je opäť založená na princípe

asociatívnych polí. Konkrétna hodnota z databázy sa získava pomocou jej umiestnenia, t.j. obyčajne je identifikovaná riadkom a stĺpcom v ktorom sa nachádza. Na rozdiel od relačných databáz tieto systémy nekladú žiadne požiadavky na schému dát, a tak je možné mať v rôznych riadkoch rôznu schému, a teda aj rôzne stĺpce. Taktiež je možné dynamicky pridávať stĺpce do jednotlivých riadkov. V závislosti od konkrétneho systému bývajú možnosti organizácie dát rozšírené aj o možnosti vytvárať tzv. triedy stĺpcov (column-family), super stĺpce (super columns), super triedy stĺpcov (super column families) a skupiny stĺpcov (column groups). Významy týchto pojmov závisia od konkrétnych systémov. Stĺpcovo orientované databázy, podobne ako key-value a dokumentové databázy, obsahujú vstavané limity na veľkosť jednotlivých záznamov.

Databázy polí (array databázy)

Databázy, narábajúce s dátami ako multidimenzionálnymi poľami n-tíc [35], sú pomerne novým a nie veľmi rozšíreným konceptom. Napriek tomu ich niektoré z ich vlastností stavajú do pozície výhodných kandidátov pre určité aplikácie, okrem iných aj v oblasti systémov na správu časových radov. Dáta časových radov, keďže sú jednodimenzionálne, môžu byť prirodzene uchovávané v podobe polí. Očakáva sa, že vďaka svojmu dátovému modelu umožnia tieto systémy pracovať s komplexnými dopytmi a tým sa stanú vhodnými pre oblasti, kde je potrebné vykonávať zložité analýzy.

3.4 Optimalizácie pre systémy využívajúce noSQL

Táto časť obsahuje popis vybraných spôsobov optimalizácie uchovávania dát časových radov, ktoré sú použiteľné v prípade, že systém používa noSQL DBMS.

3.4.1 Znižovanie počtu záznamov v databáze

Pre každý záznam časového radu je v princípe potrebné evidovať jeho časovú pečiatku a údaj. V prípade, že systém pracuje naraz s viacerými časovými radmi, môže byť vhodné ukladať okrem týchto dát aj iné metadáta, ktoré systému uľahčujú identifikáciu časového radu, ku ktorému konkrétny záznam patrí. Takúto predstavu možno veľmi jednoduchým spôsobom implementovať pri používaní niektorých typov noSQL databázových systémov. Konkrétne pre key-value a dokumentové databázy je možné ukladať si záznamy, ktoré používajú pár <časová pečiatka, ID časového radu> ako kľúč a samotný údaj ako value resp. obsah dokumentu. Tento prístup však nie je optimálny vzhľadom k tomu, že pre bežné požiadavky (napríklad pri potrebe vizualizovať údaje časového radu v určitom intervale) je potrebné načítanie veľkého množstva záznamov patriacich časovému radu, čo spôsobuje veľké množstvo dopytov do databázy. Pre lepšie pochopenie niektorých z nasledujúcich optimalizácií je možné predstaviť si tento

model relačným spôsobom. Jednotlivé záznamy sú pripodobnené k riadkom v dvojstĺpcovej tabuľke. Prvý stĺpec obsahuje kľúč záznamu a druhý stĺpec samotné dáta. Pre zjednodušenie uvažujeme o systéme spracúvajúcom iba jeden časový rad, a teda hodnotou stĺpca s kľúčom je časová pečiatka. Takáto predstava sa pre zjednodušenie používa vo vysvetleniach optimalizácií z publikácie o systéme OpenTSDB [84], kde sa autori zameriavajú na využitie optimalizácií pre konkrétny stĺpcovo orientovaný databázový systém Apache HBase. V stĺpcovo orientovaných databázach je možné túto relačnú predstavu priamo implementovať, keďže pojmy v nej použité majú význam v stĺpcovo orientovanej databáze. Tieto úvahy sú však rovnako dobre aplikovateľné aj pre iné typy databázových systémov využívajúce iné dátové modely.

Zníženie počtu riadkov

Koncept zníženia počtu riadkov (v relačnej predstave) má za cieľ znížiť počet dopytov potrebných na načítanie dát časového radu. Toto sa dá dosiahnuť tým, že jeden záznam v databáze (riadok v relačnej predstave) sa nevyužije iba na uloženie jedného záznamu časového radu, ale na uloženie záznamov z intervalu vhodnej dĺžky (časového okna). Vzniknutý riadok v relačnej predstave má mnoho stĺpcov, kde každý stĺpec zodpovedá offsetu voči začiatku časového okna v danom riadku (viď. obrázok 3.1). V stĺpci je možné uložiť aj priamo pár <časová pečiatka,údaj> v prípade, že hodnoty časového radu nezískavame pravidelne a využitie virtuálneho indexu stĺpca na určenie presnej časovej pečiatky nie je možné. V stĺpci je vtedy nutné buď ukladať celú časovú pečiatku, alebo aspoň hodnotu, pomocou ktorej ju vieme dopočítať. Takýto prístup neumožňuje šetriť ukladací priestor úplnou elimináciu dát týkajúcich sa časových pečiatok, stále však umožňuje znížiť počet dopytov do databázy.V prípade vhodných predpokladov nie je nutné pamätať si časové pečiatky každého stĺpca, pretože tieto je možné vypočítať vďaka indexu stĺpca. Predpoklady, ktoré to umožňujú, sú periodicita uchovávaného časového radu alebo garancia, že časové pečiatky získavaných dát časového radu majú pravidelnú charakteristiku. To znamená, že napríklad záznamy prichádzajú výlučne v časoch 00:00:00 a 12:00:00, no nie pre každý takýto čas nutne nejaký záznam musí prísť. V prvom prípade je možné nastaviť offsety tak, aby mal každý stĺpec v sebe údaj. V druhom prípade to možné nemusí byť a môžu vznikať tzv. riedke riadky - riadky, v ktorých sú stĺpce neobsahujúce údaj. Pre správne fungovanie takejto optimalizácie je potrebné vhodne zvoliť veľkosť časového okna v závislosti od veľkosti jednotlivých údajov, rýchlosti prichádzania nových záznamov, či konkrétneho použitého databázového riešenia. V publikácii o systéme OpenTSDB [84] autori uvádzajú, že pre ich riešenie sa osvedčilo nastaviť časové okno tak, aby v každom riadku uchovávali 100 až 1000 záznamov časového radu.

a)

ID	Časová pečiatka	Údaj
1	10:00:00	47.0
1	10:00:01	45.5
1	10:00:02	40.1
...		
1	10:00:59	35.5
1	10:01:00	38.7

b)

ID	Časová pečiatka	+0	+1	+2	...	+59
1	10:00:00	47.0	45.5	40.1	...	35.5
1	10:01:00	38.7				

Obr. 3.1: Zníženie počtu riadkov. a) Stav pred znížením počtu riadkov. b) Stav po znížení počtu riadkov. Stĺpec s časovými pečiatkami obsahuje časové pečiatky začiatkov okien. Časové pečiatky údajov v jednotlivých stĺpcoch sú vypočítateľné pomocou indexu a časovej pečiatky začiatku okna.

Zníženie počtu stĺpcov a kompresia

Zníženie počtu stĺpcov je optimalizácia, ktorú autori využívajú v OpenTSDB [84] ako nadstavbu predchádzajúcej optimalizácie. Využívajú možnosť zlúčiť stĺpce získané predchádzajúcou optimalizáciu do vlastnej dátovej štruktúry, obsahujúcej potrebné metadáta a túto štruktúru uložiť do jedného stĺpca. Môže to znamenať napríklad uložiť údaje zo stĺpcov do poľa a jeho vhodnú reprezentáciu uložiť do jedného stĺpca v danom riadku. Autori túto úpravu popisujú v súvislosti so stĺpcovo orientovanou databázou Apache HBase, kde má vďaka existencií veličín ako riadok či stĺpec praktický význam. Pri použití dokumentovej alebo key-value databázy sa táto úprava vykonáva súčasne s predchádzajúcou optimalizáciou. Pre ilustráciu použime príklad načrtnutý v úvode tejto sekcie. Úprava zníženia počtu riadkov spôsobí, že miesto toho, aby sme si pre jeden záznam časového radu ukladali jeden záznam v databáze (key-value pár alebo dokument), zlúčime viacero záznamov časového radu do jednej štruktúry reprezentujúcej časové okno, ktorú uložíme ako ako jeden key-value pár, respektíve vhodne štruktúrovaný dokument. Je zjavné, že takáto úprava už v sebe zahrnula operáciu, ktorú sme označili ako zníženie počtu stĺpcov v stĺpcovo orientovanej databáze, keďže iba v nej má zmysel uvažovať o stĺpcoch na úrovni databázy.

Druhá časť tejto optimalizácie má však význam aj mimo stĺpcovo orientovaných databázových systémov. Ide o spôsob šetrenia miestom, keď získanú dátovú štruktúru reprezentujúcu časové okno skomprimujeme vhodným kompresným algoritmom a do databázy uložíme výsledok ako BLOB. Takýto prístup je vhodný aj v key-value aj stĺpcovo orientovanej databáze, keďže takéto databázové systémy uloženú hodnotu v databázovom zázname spracúvajú ako BLOB bez ohľadu na jej obsah. Pri dokumentových systémoch stratíme možnosť pracovať s dátovou štruktúrou priamo v databázovom systéme, keďže vzniknutý BLOB nebude v potrebnom formáte a je nutné ho v dokumente uložiť iba ako hodnotu.

Komprimovanie a predchádzajúce optimalizácie vytvárajú potrebu ošetrenia ďalších potenciálnych problémov v úrovni nad databázou. Kompresia vyžaduje nájsť a použiť vhodný kompresný algoritmus, ktorý nespôsobí prílišné spomalenie operácií vkladania záznamov a čítania z databázy. Zhromažďovanie záznamov do časových okien v pamäti a až ich následné vkladanie do databázy po skupinách vytvára potrebu ošetrenia prípadného zlyhania hardvéru a databázového systému tak, aby nedošlo k strate naakumulovaných dát práve spracúvaného a vytváraného časového okna. Ošetrenie takýchto situácií je možné napríklad pomocou vytvárania logu či iných mechanizmov obnovy po zlyhaní. Pri použití týchto optimalizácií je tiež nutné zabezpečiť, aby proces, ktorý poskytuje klientovi rozhranie pre čítanie dát, vedel pristupovať k samotnej databáze (nad ňou napríklad v reálnom čase dekomprimovať dáta) a zároveň k pamäti, ktorá obsahuje spracúvané údaje aktuálneho časového okna, ktoré ešte neboli uložené do databázy. Je potrebné aby bol tento proces schopný poskytovať odpovede na dotazy, v ktorých odpovedi je časť dát už v databáze a časť ešte nie. Príkladom takéhoto dopytu môže byť požiadavka na zaslanie údajov potrebných pre vizualizáciu časového radu v intervale presahujúcom rozsah časového okna, končiacom v prítomnosti.

Existuje ešte jeden alternatívny postup zavedenia týchto optimalizácií, ktorý je popísaný v publikácii o OpenTSDB [84]. Operáciu kompresie resp. kompresie a zníženia počtu stĺpcov v prípade stĺpcovo orientovanej databázy je možné presunúť na pozadie. Záznamy sú vtedy do databázy vkladané postupne, vtedy keď prichádzajú a sú vkladané do štruktúry časového okna s veľkým počtom stĺpcov. Nezávislý proces tieto dáta po nejakom čase z databázy vyberie a upraví do skomprimovanej podoby so zníženým počtom stĺpcov. Dáta v neskomprimovanej podobe sú potom nahradené skomprimovanými.

Sémantická kompresia

Ak majú hodnoty zaznamenávaného časového radu predvídateľný charakter, t.j. nachádzajú sa v nich určité vzory, je možné ušetriť úložný priestor pomocou stratovej, sémantickej kompresie. Príkladom takejto kompresie je optimalizácia Date-Delta-Compaction (DDC). Pre potreby tejto optimalizácie je potrebné určiť vopred tolerovateľnú odchýlku uložených hodnôt v systéme od hodnôt, ktoré boli prijaté od odosielateľa. Systém si udržiava informáciu o predpokladanej hodnote údaju nasledujúceho záznamu. Po obdržaní skutočnej hodnoty tieto dve porovná a ak je rozdiel menší ako tolerovaná odchýlka, žiadnu hodnotu neuloží, pretože je možné ju zrekonštruovať. Nadobudnutá odchýlka sa môže kumulovať, ak je takto spracovaných viacero záznamov za sebou, preto si systém udržiava informáciu aj o tejto skutočnosti. Ak kumulatívna odchýlka tiež presiahne tolerovateľnú hodnotu, systém uloží informáciu, ktorá je určená na korekciu a návrat k skutočnej hodnote. Tento postup je spolu s ilustráciou popísaný v dokumentácii

systému Chronix [13].

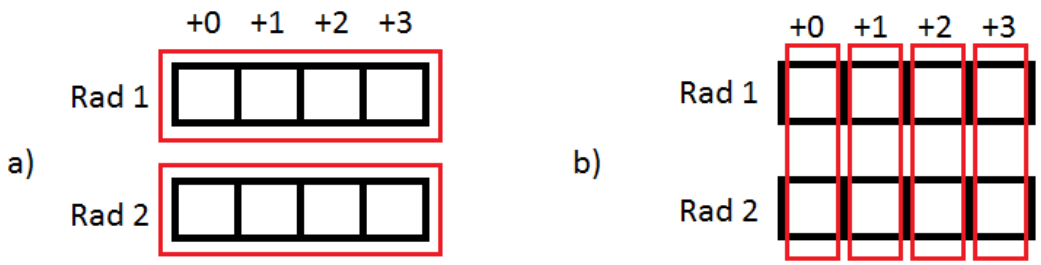
3.4.2 Vnútorná agregácia a preagregácia

V závislosti od našich požiadaviek na presnosť dát môžeme v systéme za účelom ušetrenia miesta, či zníženia počtu zápisov do databázy dáta po alebo aj pred uložením agregovať a ukladať iba vzniknuté agregáty. Takýto postup nazývame vnútorná agregácia. Agregovať dáta pred vkladaním do databázy môže byť vhodné napríklad vtedy, ak je vopred určené, že nepotrebujeme uchovávať záznamy s menším časovým rozdielom ako napríklad jedna minúta. Ak systém v takejto situácií prijme väčšie množstvo záznamov, ktorých všetky časové pečiatky spadajú do časového okna jednej minúty, je vhodné agregovať tieto záznamy a do databázy vložiť až ich agregát, reprezentujúci celú minútu. Agregácie prebiehajúce po uložení dát do databázy môžu byť naopak vhodné, ak má systém určenú potrebnú životnosť údajov. Historické údaje, ktoré nie je potrebné uchovávať vo veľmi presnej podobe je možné agregovať a v záujme ušetrenia priestoru v systéme ukladať už len ich agregáty.

Preagregácia je optimalizácia zameraná na urýchlenie poskytovania relevantných výsledkov klientovi v prípade, že klient pravidelne dopytuje dáta z časti nejakého časového radu, do ktorej pribúdajú nové záznamy, aby s nimi ďalej pracoval, napríklad aby ich agregoval a vizualizoval. Vtedy môže byť vhodné nechať bežať na pozadí proces, ktorý žiadané agregácie vypočíta a ich výsledky uloží do databázy ako nový časový rad ešte pred samotným dopytom klienta. Toto je možné iba za predpokladu, že systém je schopný tieto agregácie vykonať nezávisle od klienta. Pri dopyte tak klient nemusí čakať na vypočítanie agregácie prebiehajúcej u neho alebo na strane servera, ale čaká iba čas potrebný na čítanie z databázy. Tento spôsob je obzvlášť vhodný ak potrebné operácie proces klienta nedokáže vykonať iba s použitím dát, ktoré sa od jeho posledného dopytu pridali, ale vždy z databázy dopytuje aj staršie dáta, napr. z nejakého fixného časového intervalu. Vtedy dochádza k zbytočnému opakovanému čítaniu tých istých historických dát klientom. Preagregácia tomu môže zamedziť tým, že klientovi bude systém poskytovať iba samotné výsledky získané agregáciou najnovších dát, ktoré už klient nemusí ďalej agregovať.

3.4.3 Usporiadanie po meraniach

V predchádzajúcich častiach sme implicitne uvažovali, že dáta sú v databáze logicky usporiadané po jednotlivých časových radoch. Takýto pohľad však nie je jediný vhodný. Na obrázku 3.2 ilustrujeme takéto prirodzené usporiadanie dát a budeme ho označovať horizontálne členenie. Alternatívny prístup k členeniu dát ilustrujeme na obrázku 3.2 ako členenie vertikálne - t.j. logicky súvisiacou jednotkou dát nie je samotný časový rad,



Obr. 3.2: Horizontálne a vertikálne členenie. Pre zjednodušenie reprezentujeme časové rady ako polia údajov, ktorých index nám slúži na dopočítanie časovej pečiatky. Každý štvorček reprezentuje jeden prvok poľa. Údaje ohraničené červeným rámčekom sú v databáze uložené spolu. a) Horizontálne členenie. b) Vertikálne členenie po meraniach.

ale meranie v konkrétnom čase prechádzajúce cez viacero časových radov. Usporiadanie po meraniach sa rozoberá v publikácií o systéme tsdb [83].

V nasledujúcich úvahách abstrahujeme od konkrétnych dátových štruktúr, použitých na reprezentáciu uložených údajov v databáze a zameriame sa iba na skutočnosť, že dáta, ktoré spolu logicky súvisia, sú v databáze pravdepodobne organizované fyzicky pri sebe. Vzhľadom k tejto skutočnosti je zjavná hlavná nevýhoda vertikálneho členenia, ktorou je rýchlosť načítavania dát z databázy pri čítaní dát z dlhého časového intervalu. Ak totiž klient potrebuje získať informácie iba o jednom časovom rade, napríklad všetky jeho údaje z rozsiahleho časového intervalu, vzniká potreba prechádzať a načítavať všetky záznamy v databáze počas tohoto intervalu, ktoré však obsahujú informácie aj o mnohých iných časových radoch. Tieto informácie sú pre klienta nepotrebné a ak systém spracúva časových radov veľa, neúmerne zvyšujú množstvo dát, ktoré je potrebné preniesť na získanie správnej informácie o jednom časovom rade. Podobný problém vzniká, ak používame agregácie cez jeden časový rad.

Túto nevýhodu môžu vykompenzovať výhody, z ktorých niektoré získame pridaním určitých vhodných predpokladov. Hlavnou výhodou takejto logickej organizácie je rýchlosť vkladania nových záznamov na koniec časového radu. V prípade aktualizácie viacerých časových radov naraz nie je potrebné aktualizovať každý zvlášť a s ním súvisiace záznamy v databáze. Namiesto toho stačí do databázy vložiť jeden nový záznam, ktorý vďaka vertikálnemu členeniu obsahuje všetky nové dáta k aktualizovaným časovým radom. Ak naviac predpokladáme aj to, že spracúvané časové rady majú spoločnú periódu a nie sú voči sebe fázovo posunuté, ešte výraznejšie to zrýchli aktualizácie a vnútorne zhustí vkladané záznamy. Spoločná perióda bez fázového posunu totiž garantuje, že vo vkladanom zázname nebudú prázdne miesta patriace časovým radom, ktoré v danom čase neočakávajú aktualizáciu.

Výrazná výhoda takéhoto riešenia sa ukazuje aj pri použití distribuovaného systému v prípade, že je potrebné vypočítať radovú agregáciu cez množstvo časových radov. Vertikálne usporiadanie nám vtedy môže pomôcť zabezpečiť kolokáciu záznamov týkajúcich sa jedného času a práve vďaka tejto vlastnosti je možné výpočet agregácie efektívne paralelizovať. Každý uzol distribuovaného systému môže počítať agregáciu všetkých časových radov na intervale, ktorý pokrýva, úplne nezávisle od ostatných uzlov. Časový rad, ktorý je výsledkom takejto agregácie, je potom iba zreťazením čiastkových výsledkov z jednotlivých uzlov.

Vertikálne logické členenie je tiež kompatibilné s niektorými princípmi optimalizácie, ktoré boli popísané v predchádzajúcich častiach. Princíp zníženia počtu riadkov je možné aplikovať napríklad tak, že jeden záznam v databáze bude obsahovať viacero vertikálnych celkov - t.j. záznamy k viacero časom. Taktiež je možné podobným spôsobom za podmienky použitia vhodného databázového systému znižovať počty stĺpcov a výsledné objekty komprimovať dostupnými kompresnými algoritmami.

Kapitola 4

Prehľad dostupných systémov

Táto kapitola obsahuje popis vybraných dostupných systémov použiteľných na správu časových radov a tiež techník, ktoré používajú. Vzhľadom na množstvo podobných systémov na trhu sme sa rozhodli detailne popísať iba niektoré z nich, a to také, ktoré sú kvôli použitému prístupu zaujímavé alebo korešpondujú s požiadavkami na systém, ktorého návrhu sa venuje kapitola 7. Takéto systémy napríklad poskytujú agregácie cez viacero časových radov alebo umožňujú uchovávať historické verzie.

4.1 Vybrané systémy nepoužívajúce osobitné DBMS

Táto časť sa venuje systémom, ktoré využívajú na uchovávanie dát časových radov súborový systém bez toho, aby používali osobitný DBMS.

Akumuli

Akumuli [2] je open-source nedistribuovaný systém, zameraný na vysokú priepustnosť, kompresiu, odolnosť voči výpadku a jednoduchosť. Je stále vo vývoji a v pláne je rozšírenie jeho funkcionality. Dáta časových radov uchováva vo vlastných dátových štruktúrach a metadáta v SQLite relačnej databáze. Dáta sú uchovávané v 4GB prealokovaných súboroch s fixnou veľkosťou, tzv. volumes, ktoré sú logicky usporiadané do kruhového bufferu. Znamená to, že Akumuli umožňuje uchovávať len určitý interval dát a staršie záznamy zahadzuje. Podporuje iba aktualizácie, ktoré vkladajú nové dáta na koniec časových radov, niekoľko jednoduchých hodnotových agregácií a tiež agregácie po skupinách. Údaje časových radov uchováva ako dátový typ float.

Akumuli organizuje dáta do tzv. chunks, kde chunk je množina záznamov z nejakého časového okna a platí, že jednotlivé chunks sa neprekrývajú. Množina všetkých chunks má charakter riadkovej tabuľky, aby sa zachovala vysoká rýchlosť písania a pridávania nových chunks. Vnútorne je však každý chunk stĺpcový, pretože na kompresiu časových pečiatok sa používa iný algoritmus ako na kompresiu údajov. Kompresia je bezstratová.

Na vyhľadávanie a rýchlu orientáciu v dátach sú použité špeciálne B+ stromy [50], ktoré umožňujú rýchle dopyty na intervaly a sú optimalizované na pripájanie nových dát na koniec. Akumuli sa pri organizácií dát zameriava na zníženie počtu diskových operácií a na to, aby bola väčšina z nich sekvenčná. Systém Akumuli nepodporuje tagy a nemá vlastný vizualizátor uložených časových radov.

Graphite

Graphite [27] je systém na kolekciu, vizualizáciu a správu časových radov, využívajúci vlastnú Whisper databázu a vlastný nástroj na vizualizáciu. Whisper databáza používa časové pečiatky so sekundovým rozlíšením a uchováva ich ku každému údaju. Pre optimálne využitie priestoru očakáva periodicitu uchovávaných časových radov, v prípade chýbajúcej hodnoty totiž ukladá špeciálnu hodnotu NULL. Dokáže však pracovať aj v režime, ktorý nevyžaduje periodicitu. Whisper používa kruhový buffer fixnej veľkosti optimalizovaný tak, aby čítanie ako aj písanie do časového radu prebehlo ako jedna súvislá disková operácia.

Údaje sú v databáze uchovávané ako dátový typ double. Systém umožňuje doplňať chýbajúce historické záznamy a poskytuje niekoľko druhov agregácií, vrátane agregácií cez viacero časových radov. Ako alternatívu k Whisper databáze systém dokáže pracovať aj s Ceres úložiskom, ktoré uľahčuje prácu v distribuovanom režime, lepšie zvláda ukladanie časových radov, ktorých záznamy prichádzajú nepravidelne, no stále pracuje so súbormi v súborovom systéme. Veľa doplnkovej funkcionality je poskytovanej pomocou externých pluginov.

TrailDB

TrailDB [63, 103] je open-source knižnica v jazyku C na prácu s časovými radmi, využívajúca vlastný formát súborov, ktorý nazýva TrailDB. Každý takýto súbor obsahuje záznamy niekoľkých časových radov. Záznamy časových radov podporujú polia. Na časové rady, ktoré uchováva, nekladie žiadne požiadavky týkajúce sa periodicity - na ich záznamy sa pozerá ako na sériu nezávislých udalostí. TrailDB súbory obsahujú dáta časových radov skomprimované po jednotlivých časových radoch, aby sa urýchlila ich dekompresia pri následnej práci s nimi. Vďaka tomu sa dekomprimujú len tie časové rady, s ktorých dátami sa aktuálne pracuje. Zaujímavou vlastnosťou tohoto dátového modelu je, že vzniknuté súbory sú nemenné (immutable), t.j. môžu sa použiť iba na čítanie. Takýto formát je podľa autorov vhodnejší pre kompresiu a pre škálovanie, keďže nemennosť uľahčuje prácu v paralelnom prostredí. Súbor sa postupne tvorí v pamäti a až po jeho úplnom vytvorení sa skomprimovaný zapíše na disk. Časové pečiatky sú 64 bitové celé čísla.

BTrDB

BTrDB (Berkeley Tree Database) [74, 11, 81] je open-source vysokovýkonná výskumná databáza, používajúca stromy na organizovanie dát časových radov. Stromy obsahujú v listoch záznamy časových radov a v ostatných vrcholoch obsahujú preagregované štatistiky pre svoje podstromy, umožňujúce vykonávať niektoré agregácie zrýchlene. BTrDB podporuje časové pečiatky s nanosekundovým rozlíšením, uchovávanie historických verzií, kompresiu uložených dát a horizontálne škálovanie. Umožňuje vizualizovať časové rady a poskytuje štatistické agregácie. Metadáta ukladá v MongoDB dokumentovej databáze a dáta ukladá buď v lokálnom súborovom systéme alebo v Ceph úložisku v prípade, že pracuje v distribuovanom režime. Vývoj tohoto systému stále prebieha.

DalmatinerDB

DalmatinerDB [20, 73] je open-source databázový systém založený na frameworku pre distribuované výpočty Riak-core. Pre správne fungovanie potrebuje ZFS súborový systém, na ktorý deleguje funkcie ako je ochrana konzistentnosti (journaling), obnovenie po zlyhaní (crash recovery) a ochrana voči poruchám úložiska (bitrot avoidance). Časové rady ukladá s časovými pečiatkami s milisekundovým rozlíšením a údaje ukladá ako celé čísla. Údaje, ktoré nie sú celočíselné je preto potrebné pred vkladaním vhodne upraviť, napríklad pri meraniach to môže znamenať zmeniť jednotky a 1,5m zapísať ako 1500mm. DalmatinerDB podporuje hodnotové aj radové agregácie, vrátane agregácií cez viacero radov (nazýva ich funkcie kombinácií). Predpokladá nemennú minulosť a pracuje spôsobom best effort t.j. v prípade priveľkého množstva zápisov uprednostní uchovanie dostatočnej väčšiny prichádzajúcich záznamov pred snahou zapísať každý jeden záznam. Chýbajúce záznamy potom dokáže doplniť interpoláciou. V minulosti systém používal na prijímanie záznamov UDP protokol, no v aktuálnych verziách je už použitý protokol TCP, čo znižuje množstvo záznamov, ktoré sa stratia a je ich potrebné dopĺňať interpoláciou.

InfluxDB

InfluxDB [32] je open-source databáza, použitá v systéme spoločnosti Influx. Na ukladanie dát používa súbory v súborovom systéme. Napriek tomu, že samotná databáza je open-source, funkcionalita, umožňujúca systém distribuovať a horizontálne škálovať, je podporovaná len v platenej verzií. InfluxDB umožňuje vizualizovať dáta pomocou externého nástroja Grafana. Systém podporuje rôzne analytické a predpovedacie funkcie, hodnotové aj radové agregácie, matematické operácie radu s konštantou a pomocou určitých konštrukcií dopytovacieho jazyka, podobného SQL, umožňuje vykonať aj niektoré agregácie cez viacero radov. Predpokladá nemennú minulosť, umožňuje však odstraňovať dáta po skupinách (bulk delete). Taktiež poskytuje možnosť organizovať

časové rady pomocou tagov, nastavovať im expiráciu a podporuje polia. V rámci hodnôt polí umožňuje uchovávať dátové typy int,float,bool a string. Rozlíšenie časových pečiatok v systéme je jedna nanosekunda.

Prometheus

Prometheus [55, 76] je open-source systém, ktorý nekladie predpoklady na periodicitu časových radov, organizuje ich pomocou tagov (nazýva ich labels) a umožňuje vykonávať hodnotové aj radové agregácie cez jeden aj viacero časových radov. Podporuje upozorňovanie, preagregáciu a aj expiráciu dát. Údaje uložených časových radov majú dátový typ double, pričom dáta sú rozdelené do tzv. chunks a následne komprimované. Dáta ukladá do append-only súborov, pričom každý časový rad má vlastný súbor. Hoci systém nie je založený na distribuovanom úložisku, umožňuje horizontálne škálovanie tým, že jednotlivé servery, ktoré sú úplne autonómne môžu pracovať v master a worker režime. Každý worker v takom prípade zbiera dáta z disjunktnej množiny zdrojov, a teda pracuje v akoby nedistribuovanom režime. Master sa stará o organizáciu práce worker zariadení a tiež riešenie dopytov, ktorých výsledky obsahujú dáta z viacerých worker zariadení. Na vizualizáciu uložených dát je možné použiť nástroj Grafana ale aj vstavaný prostriedok.

Gorilla(Beringei)

Gorilla [95, 82, 9] je škálovateľná in-memory databáza vyvíjaná v spoločnosti Facebook, zameraná na ukladanie veľkého množstva rôznych časových radov s údajmi typu float. Počas jej pôvodného použitia slúžila ako write-through cache pri zápise do stĺpcovo orientovaného databázového systému Apache Hbase, pričom záznamy uchovávala po dobu 26 hodín s 15 sekundovým rozlíšením. Distribuovanosť zaručovalo použitie súborového systému GlusterFS, pričom databáza mala na starosti kolokáciu dát patriacich jednotlivým časovým radom a vytváranie nekolokovaných replík. Gorilla podporuje upozorňovanie, pri ukladaní znižuje počet riadkov, pričom v rámci okna ukladá pre zefektívnenie využitia priestoru komprimované údaje a časové pečiatky. Detailný popis princípov kompresie je dostupný v oficiálnej technickej publikácií [95]. Ako vizualizačný prostriedok je možné používať externý nástroj Grafana. Táto databáza bola v roku 2016 sprístupnená ako open-source pod názvom Beringei.

Druid

Druid [102, 22] je horizontálne škálovateľná open-source in-memory stĺpcovo orientovaná databáza, určená na prácu s časovými radmi, zameraná na rýchle agregácie a prechádzanie cez uložené dáta. Podporuje polia, preagregáciu a hodnotové agregácie.

Taktiež umožňuje používateľovi vytvoriť vlastné agregačné funkcie. Predpokladá nemennú minulosť a pridávanie nových záznamov výlučne na koniec časových radov.

SiriDB

SiriDB [60] je open-source škálovateľná databáza špecializovaná na uchovávanie časových radov, zameraná na rýchlosť vkladania a stabilitu. Pôvodne bola vyvíjaná v jazyku Python, no pre zvýšenie výkonu bola prepísaná do jazyku C. Obsahuje vstavaný nástroj na vizualizáciu uložených radov. Poskytuje vlastný dopytový jazyk podobný SQL a podporuje hodnotové a radové agregácie cez jeden a viacero časových radov. Na periodicitu časových radov nekladie požiadavky a pri práci s viacerými časovými radmi ich vhodne downsampluje a chýbajúce hodnoty interpoluje.

Atlas

Atlas [5, 90] je open-source in-memory škálovateľná databáza na spracúvanie dát časových radov, pôvodne vyvíjaná pre službu Netflix. Atlas predpokladá periodicitu dát a normalizuje ich v prípade, že prichádzajú v nesprávnych časoch. Ukladá údaje typu float s časovými pečiatkami s milisekundovým rozlíšením. V prípade chýbajúcich dát ukladá špeciálnu hodnotu NaN. Podporuje tagy, expiráciu dát a ponúka vlastný nástroj na vizualizáciu. Systém umožňuje používať hodnotové aj radové agregácie cez jeden a viacero radov, downsampling a podporuje upozorňovanie. Poskytuje tiež možnosť ukladať na disk staršie dáta do perzistentných dátových štruktúr za cenu zníženia rýchlosti prístupu k nim.

4.2 Vybrané systémy používajúce relačné DMBS

V tejto časti sa venujeme vybraným systémom, ktoré dáta časových radov ukladajú do relačných DBMS.

Tgres

Tgres [101, 100] je open-source rozšírenie databázového systému PostgresSQL, určené na uchovávanie časových radov, ktoré je momentálne vo vývoji. Vizualizácia uložených dát je zabezpečená pomocou externých nástrojov, napríklad pomocou prostriedku Grafana. V budúcnosti je očakávaná podpora tagov. Dáta sú uložené v kruhovom bufferi a systém využíva optimalizáciu zníženia počtu riadkov.

4.3 Vybrané systémy používajúce noSQL DBMS

V tejto časti popisujeme systémy, ktoré využívajú na uchovávanie dát časových radov noSQL DBMS.

OpenTSDB

Jedným z najstarších open-source špecializovaných systémov na správu časových radov je OpenTSDB [51, 84]. Ako úložisko používa stĺpcovo orientovanú databázu Apache HBase. Je horizontálne škálovateľný, pričom napriek tomu, že architektúra obsahuje master, principiálne neobsahuje SPoF (single point of failure - jediný bod zlyhania). V distribuovanom režime existencia SPoF závisí od použitého distribuovaného súborového systému (predvolený HDFS má SPoF). Ovládanie tohoto systému je kvôli jeho veku pomerne zložité, ale z hľadiska funkcionality poskytuje veľa možností. Podporuje množstvo radových aj hodnotových agregácií, vrátane agregácií cez viacero radov, preagregáciu, downsampling a interpoláciu chýbajúcich hodnôt počas agregácií. Ako optimalizáciu používa znižovanie počtu riadkov, stĺpcov aj kompresiu, no operáciu zníženia počtu stĺpcov a kompresiu vykonáva až po skompletizovaní časového okna, čo znižuje výkon. Dôvodom je, že jednotlivé záznamy sa do databázy najprv vkladajú postupne po jednom, potom sa vyberú a skomprimované vložia späť. OpenTSDB podporuje tagy, ukladá numerické hodnoty int a float a pracuje s časovými pečiatkami s milisekundovým rozlíšením. Na vizualizáciu používa externé open-source vizualizátory alebo svoj vstavaný, jednoduchý vizualizátor. Systém Argus [4, 104] je nadstavbou OpenTSDB a umožňuje používať pokročilé doplňujúce nástroje ako je upozorňovanie či vylepšený vizualizátor.

TSDB

TSDB [83] je už nepodporovaná nadstavba nad key-value databázou BerkleyDB. Je zameraná na kompresiu, umožňuje horizontálnu škálovateľnosť a prácu v režime s kruhovým bufferom. Predpokladá periodicitu cez všetky rady bez fázového posunu a vďaka tomu úspešne využíva usporiadanie po meraniach. Voči úpravám dát z minulosti nemá žiadne obmedzenia a umožňuje ich upravovať rovnakým spôsobom ako sa pridávajú nové. Neuchováva však historické verzie.

Chronix

Chronix [12, 92] je open-source škálovateľný systém na správu časových radov, využívajúci vyhľadávací server Apache Solr, ktorý používa dokumentový dátový model. Chronix vnútorne rozdeľuje časové rady do tzv. chunks, ktoré následne komprimuje.

Okrem toho umožňuje aj stratovú kompresiu Date-Delta-Compaction popísanú v kapitole 3. Uložené dáta je možné pomocou vstavaných prostriedkov vizualizovať, agregovať cez jeden aj viacero časových radov hodnotovo a radovo, a taktiež v nich sémanticky vyhľadávať (similarity search). Chronix podporuje tagy a umožňuje ukladať numerické hodnoty.

Axibase TimeSeries database

Axibase [7, 6] je komerčná databáza špecializovaná na uchovávanie časových radov. Ako úložisko používa Apache HBase. Umožňuje vizualizáciu uložených dát, uchovávanie historických verzií, horizontálne škálovanie, downsampling, kompresiu, organizáciu pomocou tagov, interpoláciu chýbajúcich hodnôt a hodnotové aj radové agregácie cez jeden aj viacero časových radov. Údaje ukladá ako numerické hodnoty s časovými pečiatkami záznamov s milisekundovým rozlíšením. K dispozícií je aj voľne dostupná community verzia, ktorá však nepodporuje prácu v distribuovanom režime, uchovávanie historických verzií, predpovede, kompresiu a iné funkcie.

Arctic

Arctic [3, 75] je open-source škálovateľné rozšírenie dokumentovej databázy MongoDB, napísané v jazyku Python, podporujúce kompresiu, uchovávanie historických verzií a viaceré spôsoby ukladania záznamov pre lepší výkon v prípade špecifického použitia. Napríklad, ak nie je potrebné uchovávať historické verzie je možné použiť iné úložisko ako v prípade, že ich je potrebné uchovávať. Podporuje dátové typy údajov z PANDAS Python data analysis knižnice a jazyku Python.

NewTS

NewTS [49] je open-source škálovateľné rozšírenie stĺpcovo orientovanej databázy Apache Cassandra, uchovávajúce časové rady s numerickými údajmi s milisekundovým rozlíšením časových pečiatok, podporujúce tagy a hodnotové aj radové agregácie cez jeden a viacero časových radov. Pri agregácií časových radov, ktoré sú fázovo posunuté alebo sú neperiodické, používa hodnoty získané interpoláciou.

Respawn

Respawn [78] je výskumná distribuovaná databáza, zameraná na zhromažďovanie a uchovávanie dát pochádzajúcich z IoT senzorov naraz v rôznych rozlíšeniach. Pri svojej práci využíva dva druhy zariadení, ktoré nazýva Cloud cluster a hranové uzly. Cloud cluster je cluster serverov, zatiaľ čo hranové uzly sú jednoduché a lacné embedded zariadenia. Dáta zo senzorov najskôr prichádzajú do hranových uzlov, kde sú automaticky

downsamplované tak, aby sa dosiahli žiadané rozlíšenia (uchovávajú sa však aj pôvodné hodnoty) a potom sú postupne posielané v rôznych rozlíšeniach do Cloud clusteru. Dopyty sú vykonávané pomocou Cloud clusteru. Architektúra tohoto systému je založená na predpoklade, že noSQL dopyty na embedded zariadeniach majú podobnú rýchlosť ako SQL dopyty na relačnú databázu bežiacu na výkonných serveroch. Systém teda napriek použitiu málo výkonného hardvéru môže rýchlosťou konkurovať systémom používajúcim SQL relačnú databázu. Respawn podporuje ukladanie dát typu float, menej ako sekundové rozlíšenie časových pečiatok a kompresiu.

4.4 Iné existujúce systémy na správu časových radov

Veľmi populárnym a mimoriadne jednoduchým open-source systémom, používajúcim kruhový buffer je RRDtool [94]. Vzhľadom k tomu, že v svojej základnej verzií neobsahuje skoro žiadnu funkcionalitu zameranú na agregácie a analýzy, takmer výlučne sa používa spolu s nadstavbami.

Niektoré komerčné systémy na správu časových radov sú založené na klasických RDBMS a hybridných relačných databázach (relačných databázach, podporujúcich niektoré z noSQL dátových modelov). Príkladmi takýchto systémov sú IBM Infomix TimeSeries [33], stĺpcová databáza Kdb+ [38, 41] a CitusDB [14].

Mimoriadne populárnym úložiskom, ktoré používajú mnohé systémy na správu časových radov je noSQL stĺpcovo orientovaná databáza Apache Cassandra. Systémy, ktoré ju používajú sú napríklad Metrictank [45], KairosDB [37], Hawkular [28], Blueflood [10, 91], Orestes [53, 86], Rhombus [56], DataBus [21], Cyanite [19], RHQ [57] a Heroic [30]. Väčšina z týchto systémov podporuje tagy a sú jednoducho horizontálne škálovateľné. Tieto systémy však nepodporujú uchovávanie historických verzií a poskytujú iba niekoľko základných agregácií cez jeden rad. Vývoj mnohých z nich už zastal a nie sú ďalej podporované.

Iné z dostupných systémov sú vybudované na noSQL DBMS používajúcich dokumentový alebo key-value model. Príkladmi takýchto systémov sú RiakTS [58], Seriesly [98], KsanaDB [40] a Cube [18]. Mnohé z nich už sú nepodporované a všetky poskytujú len základnú sadu agregácií cez jeden rad.

Ďalšie systémy, ktoré poskytujú flexibilitu pri výbere úložísk, nad ktorými pracujú sú Gnocchi [26] a SiteWhere [61].

4.5 Nešpecializované systémy

Táto časť obsahuje prehľad vybraných systémov, ktoré poskytujú funkcionalitu, umožňujúcu využiť ich na prácu s časovými radmi, hoci na prácu s nimi nie sú primárne

určené.

Elasticsearch

Elasticsearch [23, 93] je open-source škálovateľný vyhľadávací nástroj, používajúci dokumentový model, ktorý poskytuje viacero rozširujúcich doplnkov. Jedným z nich je napríklad rozhranie na vykresľovanie grafov Kibana. Hoci nie je primárne určený na prácu s časovými radmi, umožňuje vykonávať niektoré hodnotové agregácie a poskytuje možnosť naprogramovať si vlastné funkcie. Dáta v ňom je možné uchovávať ako string, int, float a bool. Existuje aj platená verzia, ktorá pracuje v cloude.

Warp 10

Warp 10 [65] je open-source databáza rozširujúca funkcionalitu systémov na uchovávanie časových radov tým, že zaznamenáva tzv. Geo Time Series, čo sú časové rady s uloženou geografickou polohou danou zemepisnou šírkou, dĺžkou a nadmorskou výškou. Obsahuje vstavaný nástroj na vizualizáciu uložených dát. Podporuje tagy, downsampling a agregácie po skupinách cez jeden alebo viac radov. Údaje môžu byť typu string, float a int. Systém dokáže pracovať v samostatnom režime, vtedy používa LevelDB key-value databázu, alebo v distribuovanom režime, kedy používa stĺpcovo orientovanú databázu Apache HBase.

FiloDB

FiloDB [24, 80] je open-source stĺpcovo orientovaná škálovateľná databáza, založená na Apache Cassandra a Apache Spark, podporujúca princípy funkcionálneho programovania. V distribuovanom režime pracuje bez SPoF a podporuje uchovávanie historických verzií, kompresiu a dopyty v jazyku podobnom SQL. Pri jej vývoji autori kládli dôraz na jej vysoký výkon a všestrannosť. Jedným z jej možných použití je aj ukladanie časových radov.

4.5.1 Monitorovacie systémy

Okrem uvedených riešení existuje viacero komerčných monitorovacích systémov, ktoré v rôznom rozsahu podporujú prácu s časovými radmi. Mnohé z nich sú poskytované ako služba na cloude, a teda neposkytujú možnosť lokálne ukladať nazbierané dáta. Väčšina z nich je zameraná na poskytnutie prehľadných nástrojov potrebných na monitorovanie IoT a iných systémov. Medzi tieto nástroje patrí prehľadné UI, nástroje na podporu upozorňovania a analytické funkcie. Príkladom sú systémy ako InfiniFlux [31], Signal fx [59], Circonus [36], Datadog [25], WaveFront [67], Librato [43], InstrumentalApp [34], TempoIQ [62] a Yanza [71].

Kapitola 5

Predpoklady a požiadavky na systém

V tejto kapitole popisujeme vhodné predpoklady a požiadavky na moderný systém na správu časových radov.

Vzhľadom na užitočnosť schopnosti systému vykonávať rôzne druhy agregácií požadujeme, aby systém tieto funkcie podporoval a najmä aby poskytoval možnosť vykonávať agregácie cez viacero časových radov. Medzi základné hodnotové agregácie patrí napríklad súčet, priemer, maximum, minimum, počet záznamov v intervale, počet záznamov s odlišnými údajmi v intervale a medián. Podobne, príkladom základných radových agregácií cez viacero časových radov sú agregácie ako súčet cez viacero časových radov, priemer, maximum či minimum cez viacero radov. Medzi dôležité radové agregácie patria aj derivácia, downsampling, rozdiel hodnôt údajov nasledujúcich záznamov či počítanie predpovedí.

Od systému požadujeme, aby sa vedel vysporiadať s chybami merania a korekciami, ktoré sa v bežnej prevádzke dejú, a zároveň aby žiadne namerané dáta nemazal. Toto je uskutočniteľné tým, že systém bude uchovávať historické verzie dát. Na rozdiel od väčšiny súčasných systémov popísaných v kapitole 4, ktoré nielen, že neumožňujú uchovávať historické verzie, ale častokrát neumožňujú ani meniť dáta, ktoré už raz boli uložené a pre zjednodušenie používajú predpoklad nemennej minulosti, od systému vyžadujeme schopnosť uchovávať a sprístupňovať historické verzie uložených dát. Predpokladáme však, že potreba aktualizácie uložených dát v zmysle pridania nového záznamu na koniec časového radu nastáva výrazne častejšie ako aktualizácia v zmysle vytvorenia novej verzie historických dát. Taktiež predpokladáme, že potreba napĺňať systém starými dátami, t.j. záznamami, ktorých časové pečiatky reprezentujú skorší čas ako pečiatka najnovšieho záznamu napĺňaného časového radu vzniká veľmi výnimočne a vo veľkej väčšine prípadov systém spracúva požiadavky na vytvorenie novej verzie existujúcich historických dát alebo pridanie nového záznamu na koniec časového radu. Akceptujeme tiež, ak vytvorenie novej verzie trvá dlhšie ako pridanie nového záznamu na koniec časového radu.

Ako bolo spomínané v úvode práce, množstvo dát, ktoré systém musí byť schopný spracovať je veľké a je ešte umocňované potrebou uchovávať všetky historické verzie. Systém preto musí byť horizontálne škálovateľný, aby sa vedel s takouto záťažou vysporiadať. Vzhľadom na obmedzený rozsah práce sa však horizontálnou škálovateľnosťou nebudeme dopodrobna zaoberať.

Čo sa týka dopytov, prioritu majú aktuálne verzie dát, a preto je potrebné, aby aktuálne dáta a štatistiky z nich systém vedel sprístupniť čo najrýchlejšie. Pri dopytoch na historické verzie sme ochotní tolerovať určité oneskorenie. Taktiež agregácie, pracujúce s historickými verziami môžu byť oproti agregáciám, pracujúcim s aktuálnymi verziami relatívne pomalé.

Požadujeme tiež, aby podobne ako veľká väčšina systémov popísaných v kapitole 4 systém poskytoval možnosť získavať informácie o uložených dátach v užívateľsky zrozumiteľnom formáte v reálnom čase pomocou vstavaných vizualizátorov alebo tým, že poskytuje rozhranie pre špecializované vizualizátory dostupné na trhu. Opäť kvôli obmedzenému rozsahu práce sa problematike vizualizátorov nebudeme venovať, hoci si uvedomujeme dôležitosť tejto funkcionality.

Predpokladáme, že časové rady, ktoré systém spracúva, majú periodický charakter so spoločnou periódou bez fázového posunu, resp. tento predpoklad spĺňajú skupiny logicky súvisiacich časových radov. V oboch prípadoch platí, že každý spracúvaný časový rad má periodický charakter.

Údaje, ktoré systém ukladá, sú obmedzené na numerické hodnoty (float, int a pod.). Údaje časových radov vyžadujeme uchovávať presne, teda nechceme, aby systém používal na ich kompresiu stratové algoritmy, alebo aby ich nejak vnútorne agregoval.

Kapitola 6

Experimentálne porovnanie

Cieľom tejto kapitoly je zhrnúť priebeh a výsledky experimentu, pri ktorom sme zisťovali schopnosti systémov, ktoré sú momentálne dostupné. Vzhľadom na veľké množstvo rôznych systémov na správu časových radov, ktoré sú na trhu, existujú mnohé experimentálne porovnania rôznych množín z nich. Často sú tieto experimenty iniciované samotnými vývojármi konkrétnych systémov, aby poukázali na vyšší výkon svojho systému a jeho výhody oproti konkurencii. Takéto testy sú však zamerané hlavne na rýchlosť aktualizácií časových radov v systéme. Experimenty merajú priepustnosť systému a tiež počet záznamov, ktoré dokáže spracovať počas určitého časového intervalu v distribuovanom alebo nedistribuovanom režime. V praxi však táto vlastnosť nie je jediná dôležitá. V experimente sme porovnávali schopnosť systémov rýchlo vykonávať agregácie a tým získavať relevantné štatistiky. Konkrétne sme sa zamerali na radové agregácie cez viacero časových radov, keďže tieto agregácie sú zložitejšie a nie je veľa systémov, ktoré ich podporujú.

Nakoľko nás zaujímajú systémy, ktoré umožňujú uchovávať dáta perzistentne, do experimentu sme nezahrnuli systémy pracujúce v in-memory režime alebo systémy používajúce kruhový buffer, hoci niektoré z nich nami požadované agregácie podporujú.

6.1 Nutné podmienky pre zaradenie systému do experimentu

Na to, aby systém mohol byť zaradený do experimentu, musel poskytovať možnosť ukladať spracúvané dáta časových radov na lokálny disk. Okrem toho bolo nutné, aby umožňoval vytvárať radový agregát cez viacero radov so zachovaním rozlíšenia záznamov, použitím operácie aritmetický priemer.

Počas prípravy na experiment sa ukázalo, že je nevyhnutné, aby systém umožňoval vkladanie veľkého množstva dát zrýchleným spôsobom t.j. nie vkladať záznamy jednotlivo alebo po malých skupinách, ale napríklad načítaním zo súboru.

6.2 Poznámky o vykonávaní experimentu

Pre spustenie experimentu bolo potrebné najskôr vybrané systémy naplniť testovacími dátami. Testovacie dáta boli vytvárané generátorom pseudonáhodných čísel s fixným seedom, aby sme ich pre experiment jednoduchým spôsobom vedeli získať dostatočné množstvo. Súbor testovacích dát obsahoval záznamy 2000 periodických časových radov so spoločnou periódou bez fázového posunutia. Rozlíšenie záznamov v testovacích dátach bolo jedna sekunda a každý časový rad obsahoval 86400 záznamov. Pri experimente každý zo systémov bežal na virtuálnom počítači s operačným systémom Debian 8.6 Jessie s kernelom 3.16.0-4-amd64. Virtuálne počítače mali k dispozícií 2 jadrá procesoru Intel Core i5-2430M a 3072MB pamäte. Samotný experiment spočíval v spustení dopytu na získanie radového agregátu cez všetky rady z testovacích dát tak, že každý záznam výsledného agregátu obsahoval údaj, ktorý bol priemerom údajov všetkých čas. radov v tom čase.

Experiment mal teda pre každý systém nasledovný priebeh. Najskôr sme systém spolu so všetkými potrebnými nástrojmi nainštalovali a spustili. Pomocou dostupných prostriedkov sme doň importovali testovacie dáta a vykonali sme nad nimi dopyt. Dĺžku výpočtu sme odmerali pomocou dostupných systémových prostriedkov.

V nasledujúcich odsekoch popíšeme detaily priebehu experimentu jednotlivých systémov a taktiež dôvody, prečo sme niektoré z nich z experimentu vylúčili. Detaily týkajúce sa použitých skriptov v jazyku Python či formátu súborov a dopytov sa nachádzajú v prílohe A.

6.2.1 NewTS

Systém Newts je nadstavbou stĺpcovo orientovanej databázy Apache Cassandra, inštalovať teda treba najskôr samotnú databázu. Keď sme postupovali podľa návodu na stránke NewTS, systém sa nám nedarilo spustiť. Pokúsili sme sa teda nainštalovať Cassandru spôsobom, ktorý je na oficiálnych stránkach Apache a až potom nainštalovať systém NewTS. Tento postup bol úspešný.

Dokumentácia uvádza možnosť importovať dáta vo formátoch CSV a JSON. Žiaľ, k preferovanému formátu CSV neposkytuje dostatok informácií na to, aby sme dokázali vytvoriť súbory správnej štruktúry. K formátu JSON dokumentácia obsahuje niekoľko malých príkladov, na základe ktorých sme boli schopní vytvárať súbory požadovanej štruktúry. Vzhľadom k tomu, že pri importe jeden súbor obsahuje serializované pole objektov, už relatívne malé veľkosti súborov (rádovo v desiatkach MB) spôsobovali kolaps importeru a systému. Testovacie dáta sme teda museli uložiť do JSON súborov veľkosti rádovo jednotiek MB.

Krátka dokumentácia žiaľ neobsahuje dostatok informácií ohľadom spôsobu tvore-

nia dopytov a ich odosielania. Počas experimentu sme očakávali, že tak ako pri niektorých iných systémoch sa nám podarí správny dopyt postupne vytvoriť s pomocou GUI a skúšania rôznych príkazov. Zistili sme však, že minimalistické GUI nefungovalo tak, ako sme očakávali a ako pravdepodobne malo, a nepodarilo sa nám pomocou neho odosielať dopyty. Preto sme nemohli analýzou správania zistiť, ako skonštruovať správny dopyt. Vzhľadom k tomuto stavu sme museli systém NewTS z experimentu vylúčiť.

6.2.2 OpenTSDB

OpenTSDB je systém, ktorý sa často objavuje v porovnaniach a benchmarkoch vzhľadom na jeho vek a tiež veľké množstvo funkcií. Hoci bol tento systém jeden z prvých špecializovaných systémov na správu časových radov, jeho vývoj stále pokračuje a je aj naďalej podporovaný. Inštalácia OpenTSDB sa skladá z dvoch častí. Najskôr je potrebné nainštalovať a spustiť stĺpcovo orientovanú databázu Apache HBase. V prípade, že databázu využívame iba v nedistribuovanom móde, sú jej inštalácia a spustenie pomerne priamočiare, problémy údajne robí spustenie distribuovaného režimu. Druhá časť je samotná inštalácia OpenTSDB. Postup na oficiálnej stránke je síce stručný a prehľadný, no počas build fázy dochádzalo k chybe a zdrojové súbory neboli skompilovateľné. Technická podpora nám v tejto veci nevedela okamžite pomôcť, no otvorila tému s týmto problémom na oficiálnom GitHub repozitári. Riešenie, ktoré umožňovalo pokračovať v inštalácií poskytol o mesiac iný používateľ, ktorý narazil na podobný problém. O ďalší mesiac bolo jeho riešenie zabudované do zdrojových súborov a teda ďalej by sa táto chyba pri inštalácií nemala objavovať.

OpenTSDB umožňuje import dát z textových súborov, počas experimentu sme však zistili, že aj po rozdelení dát do súborov v rádovo desiatkach MB dochádzalo k rôznym chybám. Po zmenšení súborov na cca. 10MB sa import podaril.

V dátovom modeli OpenTSDB má zmysel vykonávať agregácie cez viacero čas radov len v prípade, že agregované čas. rady majú rovnaký názov a líšia sa len množinami tagov. Túto charakteristiku dátového modelu sme pochopili až pri interakcií so systémom. Z tohoto dôvodu sme museli počas experimentu prerobiť formát našich testovacích dát tak, aby sa nami požadovaný dopyt dal skonštruovať. Detaily týkajúce sa importu dát a dopytovania sa nachádzajú v prílohe A

6.2.3 Axibase TimeSeries database

Axibase TimeSeries database systém bol pravdepodobne najjednoduchšie nainštalovateľný a spustiteľný spomedzi porovnávaných systémov. Dokumentácia tohoto systému je tiež rozsiahla a prehľadná. Systém poskytuje po spustení užívateľsky príjemné webové rozhranie, v ktorom je možné systém konfigurovať. Jednou z konfigurovateľných

súčastí je aj CSV parser. Práve táto vlastnosť je na Axibase systéme výnimočná a mimoriadne nápomocná. Systém totiž poskytuje možnosť importu dát vo formáte CSV, no nevyžaduje žiadne konkrétne usporiadanie dát v rámci súborov. O všetko sa stará konfigurovateľný parser, vďaka čomu je možné jednoducho importovať dáta získané exportom do CSV z iných systémov, ktoré dáta v CSV formáte ukladajú vo vopred stanovenej štruktúre, a to bez úpravy týchto CSV súborov. Počas experimentu sme mali problém s nahrávaním dát do systému, keďže vzhľadom k veľkosti súborov ich obsah nebolo možné vkladať do webového rozhrania ako text, a tak sme ich nahrávali pomocou nástroja wget, no tento prístup nefungoval. Po konzultácií s technickou podporou sme odhalili, že systém obsahuje nezdokumentovaný limit 1GB na nahrávanie súborov. Technická podpora dokumentáciu promptne doplnila a taktiež nám pomohla pri ďalšej otázke, týkajúcej sa formulovania dopytov a významu pojmov entita a metrika, ktoré sa v systéme používajú na organizáciu dát. Pri importovaní sme množinu testovacích dát rozdelili na súbory veľkosti cca. 160MB čo zabezpečilo bezproblémové nahrávanie. Podobne ako pri OpenTSDB, dátový model predpokladá, že agregácia cez viacero časových radov má význam iba v prípade, keď sa jedná o rovnaký typ časového radu (metriky), ale má rôzne entity. Detaily o importe dát a dopytoch sa nachádzajú v prílohe A.

Napriek našej snahe, aj po zväčšení Java heap pre samotný systém, databázu HBase a framework Hadoop a zvýšení timeout limitov v scanneroch v HBase, mal systém problémy odpovedať na testovacie dopyty. Systém po krátkom čase aktivity prestal pracovať a odpovedať na ďalšie požiadavky. V záznamoch o aktivite sme neskôr našli záznamy o vyhodených výnimkách.

6.2.4 InfluxDB

InfluxDB systém je rozsiahlo a podrobne zdokumentovaný, preto je jeho inštalácia a spustenie pomerne jednoduchá a priamočiara. Dokumentácia upozorňuje, že pri nahrávaní súborov je vhodné zmenšiť ich tak, aby obsahovali okolo 5000 záznamov. Naše súbory obsahovali po 4500 záznamoch, čo spôsobilo, že počet súborov veľmi narástol, no podarilo sa nám ich úspešne nahrať. Detaily, týkajúce sa importu a dopytov sú uvedené v prílohe A.

6.2.5 Chronix

Systém Chronix obsahuje napriek strohej dokumentácií pomerne jednoduchý a jasný návod s postupom na inštaláciu produktu chronix.server. Samotná inštalácia bola bezproblémová. Nástroje ako vizualizátor spolu s CSV importerom však nie sú súčasťou základnej inštalácie a treba ich inštalovať osobitne ako produkt chronix.examples. Táto inštalácia bola tiež jednoduchá, no dokumentácia už bola priveľmi strohá. Obsahovala

nie celkom zrozumiteľné vysvetlenie formátu importovaných súborov. Táto nezrozumiteľnosť bola spôsobená najmä používaním viacerých pojmov (fields, attributes) v neintuitívnych kontextoch. Pre správny import bolo podľa dokumentácie potrebné tiež vykonať zásah do konfiguračného súboru samotného chronix.server tak, aby prijal tagy (dokumentácia ich označuje ako atribúty) importovaných časových radov. Počas experimentu sa však ukázalo, že systém nemal problém s akceptovaním atribútov, ktoré neboli definované v konfiguračnom súbore. Je možné, že jednalo o neaktuálnu verziu dokumentácie a systém medzitým začal podporovať takéto automatické spracúvanie tagov. Taktiež sme si všimli, že vzhľadom k faktu, že hodnotové časti atribútov importovaných časových radov sú určené v názve importovaného CSV súboru, oddeľované znakom podtržník, importer interpretoval aj koncovku vstupného súboru ako časť posledného z atribútov. Odstránenie koncovky súborov neprichádzalo do úvahy, pretože importer pri hľadaní súborov na import v priečinkoch filtruje súbory podľa koncovky (importuje súbory s koncovkou .csv a .gz). Keďže naše testovacie dáta boli v systéme použité len raz, táto anomália pri importe neprekážala. V prípade, že by sme potrebovali importovať dáta do systému v reálnom použití a narazili by sme na tento problém, situácia by bola oveľa komplikovanejšia. Počas pokusov o import sme tiež narazili na problém, keď importer bez akýchkoľvek objasňujúcich chybových hlásení nebol schopný importovať naše dáta, ktoré boli formátované presne podľa vzoru z dokumentácie. Kontaktovali sme s týmto problémom aj technickú podporu, no nedostali sme odpoveď. Analýzou zdrojového kódu importeru a priložených príkladov jeho používania sa nám podarilo identifikovať zdroj problémov s importom. Napriek tomu, že dokumentácia popisovala možnosť uvádzať časovú pečiatku v tvare počet milisekúnd od 1.1.1970 (pri vhodných úpravách konfiguračného súboru pre importer), zapnutie tejto funkcionality spôsobovalo zlyhanie importu. V skripte, ktorý generoval testovacie dáta sme teda museli zmeniť formát času a vhodným spôsobom upraviť konfiguračný súbor pre importer.

Dokumentácia nepôsobí veľmi uceleným dojmom a je aj málo obsažná. Po importe našich dát sme však očakávali, že s pomocou dát, ktoré sú dostupné priamo od vývojárov ako sada na testovanie a pomocou GUI nástroja budeme schopný aj napriek malému rozsahu dokumentácie analýzou zistiť, akú formu majú dopyty mať. Žiaľ, aj keď sme nasledovali videonávod na stránke a snažili sme zamerať sa na zaručene správne vložené dáta, poskytnuté vývojármi, nepodarilo sa nám nástroj využiť v náš prospech. Vyskúšali sme tiež použiť externý GUI nástroj Grafana, no tiež bezvýsledne. Keďže sme počas experimentu neboli schopní vytvoriť správny dopyt, aj systém Chronix sme museli kvôli nedostatočnej dokumentácií z experimentu vylúčiť.

6.2.6 BTrDB

BTrDB systém má webstránku, ktorá rozoberá hlavné princípy fungovania tohoto systému a obsahuje odkaz na GitHub repozitár. Ani na stránke ani v repozitári sa však nenachádza žiadna ucelená dokumentácia. Počas vykonávania experimentu sme síce našli určité informácie o systéme na inej stránke, tieto však boli na prvý pohľad neaktuálne a týkajúce sa už nepodporovaných verzií. Napriek tomu sa nám systém podarilo nainštalovať a čiastočne spustiť, pričom pomocou informácií nájdených v konfiguračných súboroch sme sa snažili dozvedieť sa o ňom viac. Žiaľ, táto snaha nepriniesla žiadne výsledky. Dňa 14. marca 2017 bol z GitHub repozitára vymazaný jediný dostupný návod na inštaláciu a bol nahradený odkazom na stránku projektu, ktorý v sebe zahŕňa BTrDB. Tento projekt je primárne zameraný na spracúvanie tzv. smart grid dát, čo sú dáta pochádzajúce zo smart elektrickej infraštruktúry. Tento projekt je však stále vo vývoji, a tak jeho dokumentácia nie je dostatočná. V čase písania práce obsahovala iba dočasný návod, popisujúci ako spustiť projekt pomocou jeho komponentov. Ani táto dokumentácia však neobsahovala ďalšie informácie týkajúce sa možnosti importu dát či detailnejšieho popisu API. Pokúsili sme sa kontaktovať technickú podporu na e-mailovej adrese uvedenej na stránke a taktiež aj aktívneho vývojára, ktorého e-mailovú adresu sme získali z jeho GitHub profilu, no na naše otázky sme nedostali žiadnu odpoveď. Systém BTrDB sme teda napriek jeho veľkému potenciálu z testu vylúčili.

6.2.7 Prometheus

Prometheus je systém, ktorý je konceptuálne úplne odlišný od ostatných, ktoré sme chceli testovať. Zatiaľ čo ostatné systémy sú pasívne, teda prijímajú prichádzajúce dáta od ich zdrojov, Prometheus je aktívny, teda zbiera dáta z určených zdrojov vo vopred stanovenej frekvencii. Tento proces nazýva scraping. Keďže Prometheus dokáže pracovať len v takomto režime, nepodporuje import dát pomocou súborov a do experimentu sme ho pre veľkosť testovacích dát nezahrnuli.

6.2.8 SiriDB

Inštaláciu SiriDB zo začiatku sprevádzali problémy s dostupnosťou potrebných knižníc. Nakoniec sa po manuálnom vyhľadaní týchto knižníc podarilo nainštalovať server spolu s ostatnými súčasťami. Zistili sme, že názov databázy, ktorá sa vytvára podľa návodu na inštaláciu, nekorešponduje s predvoleným názvom databázy, ku ktorej sa pripája http server. Táto skutočnosť vyšla najavo až potom, čo sme venovali čas hľadaniu príčiny problémov s http serverom, ktorý sa nevedel správne pripojiť. Analýzou zdrojových súborov sme však našli spôsob, ako náš problém vyriešiť. Stručná dokumentácia síce

obsahuje zmienku o importe CSV súborov, neobsahuje však popis ich formátu. Na základe priloženého skriptu na vytváranie CSV súborov z finančných údajov dostupných na Google finance sa nám však podarilo zistiť správnu štruktúru vstupných súborov. Zistili sme tiež, že pri importe je vhodné neprekračovať v rámci jedného súboru limit 2 000 000 záznamov. Po vyriešení spomenutých problémov sme ocenili intuitívnosť a prehľadnosť CLI SiriDB, čo nám v systéme uľahčilo skúšanie rôznych dopytov. Dokumentácia písania dopytov je síce strohá, ale výstižná. S drobnou pomocou GUI, CLI a testovacích dát sa nám podarilo postupne zistiť, ako presne má dopyt vyzerať. Pri interakcií so systémom pomocou skúšobných dopytov sme zistili, že systém obsahuje nezdokumentovaný vstavaný limit, umožňujúci spracovať iba 1 000 000 záznamov v rámci jedného dopytu. Naše testovacie dopyty však pracovali s oveľa väčšou vzorkou dát. Kontaktovali sme technickú podporu, ktorá promptne zapracovala náš návrh a vydala novú verziu systému, umožňujúcu tento limit meniť. CLI však stále obsahovalo prednastavený čas, po ktorého uplynutí prestalo čakať na výsledok dopytu. Táto skutočnosť spôsobila, že v prípade väčších dopytov server odpoveď vypočítal a aj odoslal, no nevedeli sme sa k nej a ani k informácii o čase strávenom výpočtom dostať, keďže CLI medzičasom prestalo čakať na odpoveď a po jej prijatí vyhodilo chybové hlásenie. Formát súborov na import, potrebné príkazy a dopyt sú uvedené v prílohe A.

6.2.9 DalmatinerDB

DalmatinerDB sa nám podarilo nainštalovať a spustiť, hoci pre svoju prácu potrebuje ZFS súborový systém. Aktívny vývoj spôsobil, že niektoré z príkazov v návode na inštaláciu boli už neaktuálne, čo zapríčinilo, že inštalácia podľa návodu nebola možná. Po krátkej komunikácií s technickou podporou sa problém podarilo vyriešiť a nám sa podarilo systém spustiť. Technická podpora nám tiež objasnila, že importer dát momentálne neexistuje. Z toho dôvodu sme tento systém do porovnania nezahrnuli.

6.3 Výsledky a zhodnotenie experimentu

Výsledky experimentu sú uvedené v tabuľke 6.1

Najväčším problémom experimentu bola výrazná nejednotnosť v pojmoch a modeloch používaných v dokumentáciách jednotlivých systémov a nedostatok informácií v týchto dokumentáciách. Najjednoduchšie sa nám pracovalo s produktom Axibase a po vyriešení počiatočných nezrovnalostí aj so SiriDB. Systém OpenTSDB bol tiež prekvapivo dobre použiteľný a užívateľsky príjemný, hoci to bolo pravdepodobne spôsobené tým, že sme ho nemuseli používať v distribuovanom režime. Produkt InfluxDB mal z nášho subjektívneho hľadiska najlepšie spracovanú dokumentáciu, no jeho dátový model na nás zo začiatku pôsobil veľmi komplikovane, čo pri experimente spôsobilo určité

ťažkosti.

Náš experiment bol zameraný na testovanie výkonu systémov pracujúcich v nedistribuovanom režime. Nebral do úvahy možné situácie, keď vhodné dáta nie sú kolokované resp. dochádza k poruchám spojenia v rámci clusteru, ktoré vznikajú pri distribuovaných výpočtoch. Napriek tomu sme presvedčení, že vysoký výkon v single-node režime je dôležitým predpokladom k dosiahnutiu vysokému výkonu v distribuovanom režime.

Z experimentu vyplynulo, že napriek množstvu dostupných systémov je stále nedostatok tých, ktoré podporujú agregácie cez viacero časových radov a sú v praxi použiteľné. Vyšlo tiež najavo, že dátové modely niektorých systémov neumožňujú agregovať ľubovoľné časové rady, ale možnosť vykonávať agregácie cez viacero časových radov je častokrát obmedzená na časové rady, ktoré spolu určitým spôsobom súvisia. Možnosť plnohodnotne vykonávať agregáciu cez viacero ľubovoľných nezávislých časových radov zo systémov, pri ktorých sa nám podarilo formulovať dopyty, poskytoval len systém SiriDB.

Problém je podľa nás hlavne v tom, že veľa systémov je vyvíjaných malou skupinou ľudí, čo spôsobuje, že nie sú dostatočne zdokumentované a taktiež často po krátkom čase prestávajú byť podporované. Systém OpenTSDB je v tomto ohľade dlhoročnou výnimkou.

Zo systémov, ktoré sa nám podarilo spustiť a v krátkej dobe naplniť testovacími dátami mali všetky okrem InfluxDB problém s dopytmi, pri ktorých bolo potrebné spracúvať veľké množstvo dát. Systém OpenTSDB víťazil v prípadoch, keď sa mu podarilo dopyt vybaviť. Hoci v našom experimente bežali systémy na pomerne slabom hardvéri a v reálnom nasadení by mali k dispozícií oveľa viac pamäte a výpočtovej sily, myslíme si, že nami použitá vzorka dát, na ktorej prebiehal experiment bola tiež relatívne malá v porovnaní s dátami, ktoré je potrebné spracúvať a agregovať v reálnom nasadení.

Je teda zjavné, že je vhodné zaoberať sa problematikou návrhu a vývoja systémov na správu časových radov zameraných na agregácie, keďže trh nie je nasýtený systémami dostatočnej kvality a dostupnosti.

Tabuľka 6.1: Výsledky experimentu. V experimente sme merali čas vykonávania dopytu s agregáciou cez všetky časové rady testovacej vzorky dát na intervale uvedenej dĺžky. Uvedené údaje sú výstupom príkazu time, v prípade SiriDB sú výstupom interného časovača systému. Výstup príkazu time obsahuje v položke real informáciu o celkovom čase uplynutom od zadania požiadavky po skončenie jej vykonávania, v položke user obsahuje informáciu o procesorovom čase, ktorý proces strávil v user móde a v položke sys informáciu o procesorovom čase, ktorý proces strávil v kernel móde.

	InfluxDB	OpenTSDB	Axibase TimeSeries database	SiriDB
1 deň	real 14m42.376s user 0m0.812s sys 0m0.492s	pád systému	pád systému	pád systému
3 hodiny	real 1m47.559s user 0m0.123s sys 0m0.048s	real 0m13.585s user 0m0.024s sys 0m0.016s	pád systému	request timed out
1 hodina	real 0m35.831s user 0m0.020s sys 0m0.036s	real 0m8.203s user 0m0.004s sys 0m0.024s	pád systému	43.212s
20 minút	real 0m12.300s user 0m0.016s sys 0m0.020s	real 0m4.484s user 0m0.020s sys 0m0.016s	Pád systému alebo správa o chybe po: real 23m15.829s user 0m0.084s sys 0m0.092s	21.200s

Kapitola 7

Návrh systému

Táto časť práce obsahuje niekoľko náčrtov kľúčových princípov fungovania systému zadaného v kapitole 5. Taktiež obsahuje popis použitých dátových modelov či dátových štruktúr a porovnanie uvedených návrhov.

Rozhodli sme sa zamerať na použitie noSQL systémov, pretože ponúkajú jednoduché rozhranie a riešia veľa problémov, týkajúcich sa škálovania, perzistencie dát, obnovy po zlyhaní, organizácie dát a podobne. Ich nevýhodou je, že čiastočne odstraňujú možnosť podieľať sa na fyzickom usporiadaní dát, či už sa jedná o kolokáciu v rámci distribuovaného systému alebo priamo o fyzickú organizáciu na disku. Riešenia používajúce súborový systém môžu dáta organizovať a usporiadať na disku tak, aby boli zápisy alebo čítania sekvenčné. V prípade použitia noSQL systému je nutné túto úlohu prenechať použitému databázovému systému.

Počas prípravy náčrtu a štúdia zdrojových kódov niektorých z open-source systémov na správu časových radov sme dospeli k záveru, že predpoklad nemennej minulosti návrh takýchto systémov výrazne uľahčuje a práve úloha uchovávať historické verzie sa ukázala ako najzložitejšia.

7.1 Dokumentový náčrt

V rámci náčrtu sa zameriame na key-value databázy a ich rozšírenie, dokumentové databázy. Ich dátový model sa nám zdá vhodný pre použitie na prácu s dátami, ktoré majú charakteristiku časových radov v prípade, že chceme využívať zníženie počtu riadkov, zníženie počtu stĺpcov a kompresiu a zároveň umožniť uchovávanie historických verzií.

Uvedieme dva hlavné, konceptuálne odlišné náčrty. **Náčrt používajúci stromy** vyžaduje od použitej databázy silnejšie garancie ACID transakčnosti a má zložitejšiu štruktúru. Jeho nevýhodou je aj fakt, že pri ňom vytvárame stromovú štruktúru pomocou databázy, ktorá častokrát vnútorne tiež používa na organizáciu dát stromovú

štruktúru. Jeho výhody sa preto pravdepodobne naplno prejavia v prípade, že je plne integrovaný s použitou databázou, resp. pracuje priamo so súborovým systémom. Použitie tohoto návrhu však umožňuje výrazne zrýchliť hodnotové agregácie cez jeden časový rad. **Náčrt používajúci kľúče** je jednoduchší a v závislosti od použitého spôsobu uchovávania verzií je možné ho použiť aj s databázou, ktorá poskytuje minimálne garancie týkajúce sa transakcií. Jeho výhodou je aj versatilita, pričom pri použití predpokladu, že časové rady v systéme majú spoločnú periódu bez fázového posunutia, je možné ho jednoducho použiť na uchovávanie časových radov usporiadaných podobne, ako je to v usporiadaní po meraniach. Absencia zložitejšej štruktúry mu však neumožňuje niektoré z agregácií výrazne zrýchliť.

Predtým, než prejdeme k samotným návrhom, je nutné ujasniť si niektoré pojmy, ktoré budeme používať. Pre zjednodušenie budeme v nasledujúcich častiach označovať záznamy v databáze, t.j. dokumenty v dokumentových databázach a value časti v key-value databázach spoločným názvom *dokumenty*. Doposiaľ sme v práci používali pojem časová pečiatka v súvislosti so záznamami časových radov. Aby bolo možné správne uchovávať a sprístupňovať historické verzie dát, je potrebné každej veličine, v našom prípade záznamu časového radu, priradiť časovú pečiatku verzie. Časová pečiatka verzie označuje čas, v ktorom začal byť platný záznam, ku ktorému patrí.

7.1.1 Dátová štruktúra na uchovávanie časového radu

Dátová štruktúra, určená na uchovávanie časového radu, sa skladá z dvoch častí. Prvou časťou je **pole aktuálnych záznamov**. Toto pole obsahuje aktuálne záznamy časového radu spolu s časovými pečiatkami verzií týchto záznamov. Druhou časťou tejto štruktúry sú **polia historických verzií**. Každý záznam časového radu z poľa aktuálnych záznamov má priradené pole jeho historických verzií.

V rámci poľa aktuálnych záznamov nie je vďaka predpokladu o periodicite uchovávaných časových radov potrebné uchovávať časové pečiatky záznamov. Tieto časové pečiatky vieme jednoducho dopočítať pomocou indexu v poli a časovej pečiatky začiatku časového radu. Polia historických verzií môžeme tiež vhodne organizovať tak, aby sme vedeli jednoznačne priradiť pole historických verzií k záznamu v poli aktuálnych záznamov. V takom prípade opäť nie je potrebné ukladať v poli historických verzií časové pečiatky záznamov, pretože pre jedno pole sú všetky rovnaké. Stačí teda ukladať údaj a časovú pečiatka verzie záznamu, do ktorého daný údaj patrí.

Takáto štruktúra je vhodná, pretože umožňuje uchovávanie historických verzií spôsobom podobným princípu fat-node, teda v rámci samotnej štruktúry, nie pomocou referencií, ako je to v prípade princípu path-copying. Táto vlastnosť je pri použití v nami uvažovaných databázových systémov výhodná, pretože, ako bolo spomínané, sú zamerané na denormalizáciu a je v nich snaha minimalizovať používanie dokumento-

vých referencií. Avšak aby bola takáto dátová štruktúra v reálnych databázach skutočne použiteľná, je najskôr potrebné vyriešiť niekoľko problémov.

Hlavným zdrojom problémov je obmedzená veľkosť dokumentov v existujúcich systémoch. V prvom rade táto skutočnosť spôsobuje, že nie je možné celý časový rad uložiť do spomenutej štruktúry. Tento problém riešime tým, že definičný obor každého časového radu rozložíme na intervaly. Do spomenutej štruktúry budeme vždy ukladať zúženie časového radu na konkrétny interval. Takýto úsek časového radu nazývame časovým oknom. Vzhľadom na periodickosť časových radov je vhodné, aby boli pre rady s rovnakou periódou časové okná rovnakej dĺžky. Docielime tým rovnaký počet záznamov v rámci jedného okna a taktiež nám táto vlastnosť pomôže s usporadúvaním dát, ako to bude popísané v ďalších častiach.

Predchádzajúci odsek sa primárne zaoberal aktuálnymi verziami dát. Problém použitej dátovej štruktúry spočíva aj v tom, že ak aj použijeme časové okná vhodnej veľkosti, po pridaní množstva historických verzií, týkajúcich sa záznamov v danom okne sa dokument, obsahujúci dátovú štruktúru preplní. Je teda zjavné, že samotné rozdelenie časového radu na časové okná za účelom pohodlného mapovania medzi dokumentami a dátovými štruktúrami nestačí. Je nutné rozdeliť dátovú štruktúru obsahujúcu časové okno do viacerých dokumentov. Tieto dokumenty delíme na **primárne** a **sekundárne**. Primárne dokumenty obsahujú pole aktuálnych záznamov a časti polí historických verzií. Sekundárne dokumenty obsahujú iba polia historických verzií. Znamená to teda, že každá dátová štruktúra je umiestnená v jednom primárnom a niekoľkých sekundárnych záznamoch. Javí sa nám vhodné organizovať polia historických verzií tak, že najnovšie historické verzie sú najbližšie k poľu aktuálnych verzií a teda k primárnemu dokumentu. Pri organizácií týchto polí máme tiež dve možnosti usporiadania. Prvou možnosťou sú **polia fixnej dĺžky**. Vtedy má každý záznam časového radu v primárnom aj ľubovoľnom sekundárnom zázname priradené pole historických verzií fixnej dĺžky. Ak počet verzií nejakého záznamu prekročí počet prvkov tohoto poľa, je potrebné na zapisovanie historických verzií tohoto záznamu použiť ďalší sekundárny dokument. Druhou možnosťou sú **polia dynamickej dĺžky**. V takomto prípade je vopred určený iba počet historických verzií, ktoré môžeme uchovávať v jednom dokumente. Nový sekundárny dokument sa vytvára až po tom, čo sa predchádzajúci dokument úplne naplní. Využitím tohoto prístupu zefektívnime využívanie miesta a znížime počet dopytov do databázy.

Keďže historické verzie záznamov postupne pribúdajú, musíme vedieť riešiť situáciu, kedy dôjde k preplneniu primárneho alebo sekundárneho dokumentu. Opäť máme niekoľko možností, ako postupovať. Je zjavné, že do ďalšieho sekundárneho dokumentu nestačí presunúť iba jeden preplňujúci údaj spolu s časovou pečiatkou verzie jeho záznamu, pretože by k preplňovaniu dochádzalo neustále. Je však možné presunúť do ďalšieho dokumentu vhodný počet, napríklad polovicu historických verzií záznamu, ktorého aktualizácia spôsobila preplnenie. Pri poliach dynamickej dĺžky to však môže

spôsobiť, že sa presunie polovica záznamov z krátkeho poľa, zatiaľčo verzie z dlhého poľa iného záznamu časového radu, ktoré napĺňajú takmer celý dokument, zostanú nedotknuté. Preto je môže byť vhodné presunúť polovicu zo všetkých historických verzií v danom dokumente. Posledné dva spomenuté princípy spôsobia, že sa po ich použití dlhšiu dobu upravovaný dokument nepreplní.

Tieto operácie sú pomerne zložité, preto je nutné, aby boli časové okná zvolené tak, aby k takýmto preplneniam nedochádzalo s vysokou frekvenciou. Zároveň je dôležité všimnúť si, že najaktuálnejšia verzia dát sa vždy nachádza v primárnom dokumente a reťaz sekundárnych dokumentov sa tiež nemení, iba sa na koniec pridávajú nové a presúva sa ich obsah. Preto môže byť vhodné v jednotlivých dokumentoch uchovávať aj referenciu na nasledujúci sekundárny dokument.

V predchádzajúcich odsekoch sme sa venovali použitiu uvedenej dátovej štruktúry v rámci dokumentov v databáze. V prípade použitia polí dynamickej veľkosti na uchovávanie historických verzií je však potrebné vyriešiť aj to, akým spôsobom s nimi systém po prečítaní dokumentu z databázy pracuje v pamäti. Prístupy sa rôznia od použitých jazykov, v ktorých je systém naprogramovaný. Niektoré jazyky od programátora nevyžadujú, aby sa zaoberal nízkoúrovňovými záležitosťami akými je alokácia pamäte. To môže prácu s konceptom dynamických polí sprehľadniť a uľahčiť. Bez ohľadu na použitý jazyk považujeme za zaujímavý prístup použitie spájaných zoznamov, ktoré uľahčujú presúvanie jednotlivých záznamov medzi poľami, ktoré patria do rôznych dokumentov pri preplnení a taktiež nám umožňujú pohodlne pridávať záznamy na začiatok. Pri čítaní dokumentu by teda mohli byť priamo vytvárané spájané zoznamy a po vykonaní úprav by boli zas serializované a uložené do dokumentov.

7.1.2 Náčrt používajúci stromy

Pre jednoduchosť popíšeme náčrt s použitím len jedného časového radu. Princíp spočíva v tom, že nad dátovými štruktúrami, obsahujúcimi záznamy časových radov vybudujeme k-árny strom, ktorý nám slúži na orientáciu a zrýchlenie hodnotových agregácií. Každý časový rad má v systéme svoj vlastný strom. Listy stromu obsahujú referencie na primárne dokumenty, obsahujúce štruktúru popísanú v predchádzajúcej časti. Vrcholy stromu neobsahujú záznamy časových radov, obsahujú však štatistické údaje, týkajúce sa ich podstromu a údaje pomáhajúce pri orientácií v rámci časových okien. Príkladom takýchto údajov môžu byť napríklad priemerná hodnota aktuálnych údajov v podstrome, minimum, maximum, minimálna časová pečiatka, maximálna časová pečiatka, hĺbka podstromov a iné. Väčšinu týchto údajov vieme vypočítať len s pomocou údajov zo synov, preto je jednoduché pri aktualizáciách časového radu tieto hodnoty na ceste z listu ku koreňu aktualizovať. Javí sa nám vhodné obmedziť a vopred zhora ohraničiť počet referencií na synov, ktoré vrchol stromu môže obsahovať. Snahou je zís-

kať plytké stromy, znížiť potrebný počet rebalansovaní a zabezpečiť efektívne plnenie stromu. V tomto nám môže pomôcť predpoklad periodickosti a fakt, že stromy rastú väčšinou do jednej strany - pridávaním nových záznamov časového radu na koniec. Vďaka tomu môžeme zabezpečiť, že sa stromy budú plniť efektívne a všetky okrem krajných vrcholov na jednotlivých úrovniach budú naplnené. Docieliť to môžeme tým, že až keď sa aktuálny strom naplní, vytvoríme nový koreň a aktuálny strom pod neho podvesíme ako najkrajnejšieho syna.

Jednotlivé vrcholy stromu môžu byť v prípade potreby priamo mapované na dokumenty v databáze, v niektorých prípadoch to ale nie je potrebné a môžeme umiestniť viac vrcholov do jedného dokumentu.

Čo sa týka funkcie vrcholov v strome, umožňujú nám jednoducho sa pohybovať v rámci jedného časového radu a čo je najhlavnejšie, zrýchľujú hodnotové agregácie, týkajúce sa aktuálnych verzií údajov časových radov pomocou štatistických údajov, ktoré obsahujú. Nevýhodou tohoto usporiadania je, že pre orientáciu v rámci primárnych a sekundárnych dokumentov môžeme využiť len nimi tvorený spájaný zoznam. Ak by sme si chceli v listoch pamätať miesto referencie na primárny dokument pole referencií na primárny a sekundárne dokumenty, zvýšilo by to počet referencií, ktoré treba udržiavať v konzistentnom stave a taktiež by bolo potrebné riešiť potenciálny stav, keď sa dokument obsahujúci vrchol stromu naplní a treba ho rozdeliť do viacero dokumentov. Mohli by sme použiť podobnú dátovú štruktúru ako pre uchovávanie verzií záznamov časových radov, no toto riešenie považujeme za technicky náročné a nepraktické v prostredí, kde je možnosť transakcií obmedzená. Mohli by totiž vzniknúť situácie, že by sa preplnil nejaký dokument obsahujúci list, ktorý by sa následne musel rozdeliť a pridanie novej referencie by spôsobilo preplnenie dokumentu obsahujúceho jeho otca, a tak ďalej až do koreňu.

Z predchádzajúcich odsekov je zjavné, že návrh používajúci stromy pracuje s množstvom referencií a pri každej aktualizácií upravuje rôzne hodnoty vo vrcholoch stromu, a teda v rôznych dokumentoch. Je preto nutné tieto úpravy robiť v transakciách s dostatočnými garanciami, aby sa zabránilo strate konzistencie referencií a pomocných údajov v strome a taktiež aby sa zabránilo ukladaniu dát do dátových dokumentov, ktoré nie sú správne referencované listami v strome.

7.1.3 Náčrt využívajúci usporiadanie kľúčov

Tento náčrt je konceptuálne jednoduchší a nevyžaduje tak silné transakčné garancie, ako stromový náčrt. Využíva možnosť lexikografického usporiadania kľúčov v niektorých databázových systémoch. Každému časovému radu priradíme unikátny identifikátor. Ak chceme získať horizontálne usporiadanie, použijeme formát kľúčov <UID_RADU ZAČIATOK_OKNA VERZIA>, kde jednotlivé časti kľúča oddelíme vhodným symbo-

lom, ktorý sa v rámci týchto častí nenachádza. Ak chceme použiť usporiadanie podobné vertikálnemu usporiadaniu po meraniach, použijeme formát <ZAČIATOK_OKNA UID_RADU VERZIA>. Vďaka takémuto formátu kľúčov je zaručené, že súvisiace dokumenty sú pri sebe a dokážeme sa k nim rýchlo dostať pomocou dopytu na interval kľúčov. V prípade použitia dokumentovej databázy môžeme indikátor verzie z kľúča odstrániť a umiestniť ho do iného indexovaného poľa. To nám uľahčí a sprehľadní vyhľadávanie aktuálnych verzií.

Máme dve možnosti, ako využiť verzie. Prvou je použitie dátovej štruktúry, popísanej v predchádzajúcich častiach. V takomto prípade verzia v kľúči označuje iba to, či je dokument primárny alebo sekundárny a v druhom prípade aj jeho polohu v rámci postupnosti sekundárnych dokumentov. Tento prístup má výhodu v tom, že vzhľadom na predpoklad o nie častých úpravách minulosti dokážeme stále pohodlne pracovať s kľúčmi idúcimi za sebou a zbytočne nenačítavame veľa dokumentov, keďže potrebu vytvoriť sekundárny dokument nemáme až tak často.

Iná možnosť je uchovávanie verzií celých časových okien buď pomocou skopírovania celého dokumentu s časovým oknom a úpravy aktualizovaných záznamov, alebo uložením iba záznamov, ktorým pribudla nová verzia. Prvý prístup urýchľuje čítanie, vytvára však v databáze veľké množstvo duplikátov. V druhom prístupe je nevýhodou, že pre získanie dát z celého časového okna je potrebné načítať viacero dokumentov. Oba tieto prístupy sú odlišné od prístupu používajúceho navrhnutú dátovú štruktúru, keďže okrem sprístupňovania historických verzií záznamov uložených časových radov umožňujú rekonštruovať stav všetkých dát v databáze k určitému času. Taktiež vyžadujú ešte slabšie transakčné garancie, pretože pri pridaní novej verzie je principiálne potrebné vložiť iba jeden nový dokument.

V prípade, že použijeme uchovávanie verzií v načrtnutej dátovej štruktúre a chceme ešte viac zoptimalizovať čítanie intervalov kľúčov pre získanie aktuálnych verzií, môžeme formát kľúča pre horizontálne členenie upraviť tak, že prehodíme časti kľúča, aby sme zoskupili logicky súvisiace časti, v tomto prípade dokumenty s aktuálnou verziou. Docielime to formátom <UID VERZIA ZAČIATOK_ OKNA>. Nevýhodou je, že takto sa vzdialia sekundárne dokumenty od primárnych. V prípade vertikálneho členenia je možné použiť formát <VERZIA ZAČIATOK_OKNA UID>, ale v takom prípade už sú informácie o sekundárnych dokumentoch v usporiadaní veľmi vzdialené od primárnych.

Z popísaných alternatív sa nám najpraktickejšie javia usporiadania kľúčov popísané v prvom odseku. Výber horizontálneho alebo vertikálneho usporiadania je determinovaný prioritou, ktorú pre nás má rýchlosť vykonávania radových agregácií. Použitie dátovej štruktúry na uchovávanie verzií vyžaduje, aby použitá databáza mala určité transakčné garancie. Tie zabránia strate alebo poškodeniu dát v prípade práce s viacerými dátovými dokumentami pri preplnení niektorých z nich. Druhý prístup ucho-

vávania verzií po vhodnom zvolení veľkosti časového okna považujeme za praktickejší, pretože potrebuje minimálne garancie transakčnosti a poskytuje aj funkcionalitu rekonštrukcie stavu všetkých dát.

7.1.4 Kompresia

Na základe skúmania rôznych existujúcich systémov sme dospeli k záveru, že benefity z kompresie údajov časových radov a časových pečiatok, ako sú úspora miesta a zníženie množstva dát, ktoré treba prenášať, výrazne prevažujú straty spôsobené potrebou dáta komprimovať a dekomprimovať, či nutnosťou ukladať dáta v dokumentoch ako BLOB. Tento stav je ešte umocnený tým, že vďaka predpokladu o type údajov vieme vopred obmedziť typ dát, ktoré uchovávame, a tým kompresiu ešte zoptimalizovať.

Na kompresiu dát môžeme použiť dostupné rýchle algoritmy ako napr. Lz4 [44], alebo aj špecializované algoritmy pre záznamy časových radov, použité napr. v Gorilla [95] alebo Akumuli [15], ktoré môžu byť oproti generickým algoritmom pri vhodnom použití menej náročné na systémové zdroje a efektívnejšie. Tieto algoritmy môžeme použiť na kompresiu poľa s aktuálnymi verziami, kde sú uchované iba časové pečiatky verzií a údaje a tiež v poliach historických verzií. Keďže polia historických verzií ako aj polia aktuálnych verzií obsahujú dvojice <časová pečiatka verzie, údaj>, tieto dáta majú tiež charakter záznamov časových radov a môžeme ich komprimovať vhodným algoritmom. V prípade potreby je ešte možné rozdeliť každé z týchto polí na pole časových pečiatok a pole údajov a využiť na orientáciu v nich indexy. V takomto prípade môžeme použiť rozdielny kompresný algoritmus pre údaje a pre časové pečiatky. Takýto prístup používa napr systém Akumuli [15].

7.1.5 Expirácia dát

Oba náčrty umožňujú prirodzenú expiráciu dát. V prípade náčrtu využívajúceho kľúče nám nerobí problém mazať časové okná, v prípade vertikálneho usporiadania a spoločnej doby expirácie dokonca môžeme naraz mazať skupiny dokumentov.

Náčrt používajúci stromy je tiež použiteľný, ak je potrebné expirovať dáta. Stačí postupne označovať dátové dokumenty, ktoré obsahujú iba expirované záznamy časového radu. Po označení všetkých synov najkrajnejšieho listu, obsahujúceho referencie na dokumenty s najstaršími časovými oknami, ich môžeme spolu so spomenutým listom vymazať a vhodne označiť referenciu naň v jeho otcovi. Takýmto spôsobom vymazávanie postupne dosiahne až ku koreňu. Vtedy zostane na začiatku poľa referencií v koreni voľné miesto a referencie v ňom možno posunúť, aby sa zaplnilo a vzniklo miesto na druhom konci.

7.1.6 Agregácie cez viacero časových radov

V prípade, že rady so spoločnou periódou agregujeme agregáciou cez viacero radov, v prípade oboch náčrtov môžeme zabezpečiť, aby boli dátové dokumenty vhodne zarovnané a aby časové okná spracúvaných radov k sebe navzájom pasovali. Takéto zarovnanie nám môže uľahčiť pri výpočte organizáciu dát v pamäti. Pri náčrte využívajúcom usporiadanie kľúčov a využití usporiadania podobného vertikálnemu získavame výhody, popísané v kapitole 3.

7.1.7 Zrýchlenie agregácií

Požiadavkou na načrtávaný systém sú aj rýchle agregácie. Je však dôležité uvedomiť si, že napriek dôležitosti agregácií majú prioritu operácie zápisu, ktoré nesmú byť zablokované. Z tohoto dôvodu niektoré existujúce systémy uvádzajú ako jednu zo svojich predností to, že nevykonávajú žiadne preagregácie a agregácie počítajú až vtedy, keď si ich výsledky klient skutočne vyžiada. Zabraňujú tak míňaniu prostriedkov v čase, keď nie je spracúvaný žiaden dopyt a tým zachovávajú vysokú priepustnosť. Toto však môže vyústiť do neefektivity. Napríklad ak zákazník často vyžaduje výsledok nejakej konkrétnej agregácie, môže sa stať, že niektoré výpočty systém vykoná zbytočne viac krát a ak by si systém pamätal doplňujúce informácie, výsledky by v skutočnosti mohli byť poskytnuté v kratšom čase.

Hodnotové agregácie cez jeden alebo viac časových radov je možné častokrát počítať za pomoci priebežne získavaných dát počas toho, ako sa tieto dáta zapisujú a taktiež výsledné hodnoty jednoducho uchovávať v pamäti, či zapisovať, keďže nezaberajú priveľa priestoru.

Zložitejšie sú radové agregácie cez viacero časových radov. Tiež existuje možnosť priamo ich počítať z prichádzajúcich záznamov a vznikajúci časový rad udržiavať v pamäti. To môže byť vzhľadom na distribuovanosť a rozsah získaných dát nepraktické. Druhou možnosťou je ukladať ho na disk priamo ako materializovaný časový rad. V prípade, že klient zadá dopyt na agregované dáta, netreba čítať veľa radov a počítať agregácie, stačí prečítať jeden. Nevýhodou je, že takýto v reálnom čase vytváraný časový rad vyžaduje rovnakú réžiu a prostriedky ako prichádzajúce nové záznamy iných časových radov, a preto pri neopatrnom používaní môže preťažovať systém a blokovať zápis dát pochádzajúcich z reálneho sveta. S takouto možnosťou je preto nutné narábať opatrne. Na zodpovednosť používateľa sa nemôžeme spoľahnúť, pretože musíme počítať s eventualitou, že ak by mal možnosť vopred systému zadať, ktoré agregácie chce mať k dispozicii zrýchlene, vytvorí ich priveľa. Je teda potrebné snažiť sa pozorovaním klientovej aktivity zistiť, ktoré zo zadaných zrýchlených agregácií sú naďalej potrebné a ktoré sú už neaktuálne. Táto úloha však nie je jednoduchá a vyžaduje s klientom určitú interakciu. Napríklad klient môže zadávať dopyty na veľké množstvo agregácií,

ktoré nepotrebuje zrýchľovať. Zároveň občas zadáva dopyt na jednu, ktorej výsledky potrebuje čo najrýchlejšie. Samotným pozorovaním aktivity a dopytov nedokážeme správne identifikovať túto potrebu.

Okrem počítania agregácií v reálnom čase ich výpočet môžeme presunúť na čas, keď na to má systém dostatok prostriedkov a počítať ich na pozadí. Nevýhodou je, že zatiaľ čo v reálnom čase nemusí byť potrebné čítať dáta z databázy, pretože prichádzajúce záznamy postačujú na výpočet, výpočet na pozadí potrebuje najskôr uložené záznamy prečítať a až potom ich môže spracovať.

Počítanie agregácií z prichádzajúcich záznamov sa dá spojiť s agregáciou na pozadí tak, že keď systému hrozí, že sa preťaží, môže prestať počítať v reálnom čase, aby sa zameral len na prijímanie nových záznamov a tento výpadok si poznačiť. Neskôr, keď sa situácia zlepší a prostriedky sa uvoľnia, môže chýbajúce časti agregátov dopočítať na pozadí, hoci za cenu čítania vynechaných hodnôt z databázy.

7.2 Náčrt používajúci databázu polí

Napriek svojmu relatívne malému rozšíreniu na trhu poskytujú moderné databázy polí funkcionalitu, ktorá je vhodná na uchovávanie dát časových radov. Napríklad projekt SciDB [99] poskytuje prakticky všetky potrebné nástroje na ukladanie takýchto dát okrem agregácií. Systém je však pripravený na používanie zložitých analytických funkcií, ktoré zadáva sám klient a tak má jeho dopytovací jazyk dostatočnú silu na to, aby sa tieto agregácie dali naprogramovať. Systém používajúci SciDB by bol iba vhodnou nadstavbou, ktorá by mala za úlohu vhodným spôsobom prekladať dopyty, zautomatizovať inicializáciu polí pre časové rady pri ich vytvorení a poskytovať rozhranie pre použitie vizualizátorov a komunikáciu s klientom. Systém by však mohol využívať výhody distribuovanosti, robustnosti, kompresie a uchovávania verzií poskytovaného priamo SciDB.

Vzhľadom k relatívne nízkej rozšírenosti databáz polí a ich špecifickému dátovému modelu neexistuje veľa relevantných porovnaní, z ktorých by bolo jednoznačné, nakoľko je dátový model polí v týchto databázach rýchly, a teda či sa oplatí systém stavať pomocou neho.

7.3 Zamietnuté náčrty

V tejto časti sa venujeme niektorým z náčrtov, ktorými sme sa zaoberali, no nezdali sa nám vhodné.

Na začiatku sme uvažovali o použití plne perzistentných dátových štruktúr na uchovávanie časových radov, čo by znamenalo, že ľubovoľná úprava dát vrátane pridania

nového záznamu na koniec by vytvorila novú verziu a tiež, aby sme sa vyhli duplikácií dát, by sme museli používať množstvo referencií na staré verzie. Vzhľadom k tomu, že sa nám ako najvýhodnejšie javilo použitie dokumentových a key-value systémov v náčrtoch, tento spôsob sa nám nezdal vhodný, pretože by vyžadoval veľké množstvo referencií na dokumenty a taktiež referencie v rámci dátových štruktúr by museli byť serializovateľné.

Predtým, než sme sa detailnejšie zoznámili s niektorými noSQL DBMS sme uvažovali o riešení, ktoré by využívalo fakt, že časový rad je prirodzený agregát a teda by bolo vhodné udržiavať ho v jednom dokumente. Takéto riešenie kvôli obmedzeniam súčasných systémov na množstvo dát, ktoré je možné uložiť v jednom dokumente, nie je použiteľné. Dá sa však skombinovať s predchádzajúcou úvahou o perzistentných štruktúrach. V tomto prípade by skutočne platilo, že jeden dokument obsahuje celý časový rad, v skutočnosti by však obsahoval iba časť dát a tie zvyšné by bolo potrebné zrekonštruovať pomocou referencií na predchádzajúce verzie. V krajnom prípade by teda jeden dokument obsahoval iba jeden záznam časového radu a referenciu na predchádzajúcu verziu. Je zjavné, že takéto riešenie by na databázu vytvorilo priveľkú záťaž, pretože by pri čítaní bolo potrebné pracovať s obrovským množstvom dokumentov a referencií. Pomôcť by mohlo zníženie počtu riadkov tým, že by sa do jedného dokumentu vložilo viacero záznamov časových radov spolu s poznačením ich verzií. Tento spôsob však prestáva mať charakter plne perzistentnej štruktúry a stáva sa iba nepraktickou variáciou s referenciami na to, čo sme popísali ako náčrt pomocou kľúčov.

Uvažovali sme aj o riešení, ktoré by používalo strom so štatistikami podobný tomu v popisovanom náčrte, ktorý by sa správal ako perzistentná dátová štruktúra. Umožňovalo by to úplne eliminovať polia verzií a s tým súvisiace delenie na primárne a sekundárne dátové záznamy. Pri vytvorení novej verzie nejakého záznamu časového radu by sa skopíroval obsah dátového dokumentu s časovým oknom, ktoré tento záznam obsahuje do nového dokumentu a tam vhodne upravil. V novom dokumente je ešte možné uložiť referenciu na dokument so starou verziou a časovú pečiatku verzie pre celé okno. Pri takejto úprave je samozrejme potrebné upraviť aj verzie stromových záznamov na ceste ku koreňu a tiež ich vnútorné agregačné štatistiky. Toto je možné urobiť pomocou referencií a princípu path-copying. Je tiež potrebné do mapovania medzi identifikátorom časového radu a referenciou na koreň stromu pridať pole s referenciami na všetky historické korene. Niektoré systémy používajú buffer na zachytávanie prichádzajúcich záznamov tak, aby písali naraz záznamy z celých časových okien. Práca v takomto režime zabezpečí, že dokumenty sú prístupné iba na čítanie a po zapísaní sa nemenia. Pomocou takéhoto prístupu vieme jednoducho rekonštruovať predchádzajúce verzie pomocou historických koreňov a nemennosť dokumentov má výhody aj pri paralelizácií. Toto riešenie však v databáze vytvára veľa dokumentov a pri niektorých dopytoch na staré verzie spôsobuje potrebu čítať a zapisovať veľa dokumen-

tov. Preto je, podobne ako náčrt používajúci stromy, možno vhodné skôr v prípade, že je integrované s úložiskom alebo používa súborový systém.

7.4 Distribuovanosť

Ako bolo spomenuté v kapitole 5, problémy týkajúce sa distribuovaných riešení presahujú rozsah práce. Popíšeme však aspoň základné možnosti, ktoré sa v systémoch na správu časových radov z kapitoly 4 používajú a sú teda vhodné v reálnom nasadení. Systém môže pracovať v distribuovanom režime bez SPoF alebo so SPoF. Môže však pracovať v master-worker režime tak, že jednotlivé worker uzly sú plne autonómne a zbierajú záznamy z vlastnej množiny zdrojov. Vykonávajú tiež agregácie iba nad svojimi dátami. Ak je potrebné agregovať cez viacero časových radov, ktoré sú rozdelené vo viacerých uzloch, spracovanie čiastkových výsledkov zabezpečí master. Takéto usporiadanie znemožňuje vertikálne usporiadanie po meraniach, pretože by to spôsobovalo, že by všetky nové záznamy prúdili vždy len do jedného uzlu.

Záver

Spracúvanie dát časových radov je v dnešnej dobe kľúčovou súčasťou mnohých procesov. Je preto potrebné mať k dispozícií systémy, ktoré s takýmito dátami dokážu pracovať efektívne.

Pre lepšie zoznámenie sa s problematikou sme naštudovali a spracovali základný prehľad dnes dostupných systémov na správu časových radov, ich vlastností a techník, ktoré využívajú. Vďaka tomuto prehľadu sme boli schopní lepšie pochopiť princípy ich fungovania a taktiež vybrať z nich niektoré, ktoré sme podrobili experimentálnemu porovnaniu. V ňom sme skúmali schopnosť jednotlivých systémov vykonávať radové agregácie cez veľké množstvo časových radov. Ukázalo sa, že mnohé z nich nie sú schopné vyhovieť požiadavkam, ktoré sa nám javia ako realistické. Z tohto dôvodu sme usúdili, že potreba navrhovať a vyvíjať takéto systémy je stále aktuálna a načrtli sme možné základné princípy systému, ktorý by bol použiteľný v modernom prostredí a spĺňal by požiadavky vyplývajúce z reálneho použitia.

Literatúra

[1] ACID support in Aerospike. [Citované 2017-5-3] Dostupné z https://www.aerospike.com/docs/architecture/assets/AerospikeACIDSupport.pdf.

[2] Akumuli. [Citované 2017-2-9] Dostupné z http://akumuli.org/.

[3] Arctic TimeSeries and Tick store. [Citované 2017-2-9] Dostupné z https://github.com/manahl/arctic.

[4] Argus documentation. [Citované 2017-2-9] Dostupné z https://github.com/salesforce/Argus.

[5] Atlas documentation. [Citované 2017-2-10] Dostupné z https://github.com/Netflix/atlas/wiki.

[6] Axibase time series database. [Citované 2017-2-9] Dostupné z https://axibase.com/wp-content/uploads/2015/02/ATSD-Presentation-usecase.pdf.

[7] Axibase time series database documentation. [Citované 2017-2-9] Dostupné z https://github.com/axibase/atsd.

[8] BergDB. [Citované 2017-5-3] Dostupné z http://bergdb.com/.

[9] Beringei. [Citované 2017-2-9] Dostupné z https://github.com/facebookincubator/beringei/.

[10] Blueflood project and community. [Citované 2017-2-9] Dostupné z https://github.com/rackerlabs/blueflood/wiki/Blueflood-project-and-community.

[11] BTrDB: Berkeley tree database. [Citované 2017-2-9] Dostupné z http://btrdb.io/index.html.

[12] Chronix. [Citované 2017-2-9] Dostupné z https://github.com/ChronixDB.

[13] Chronix time series. [Citované 2017-1-24] Dostupné z https://github.com/ChronixDB/chronix.timeseries/blob/master/README.md.

[14] Citus documentation. [Citované 2017-2-9] Dostupné z `https://docs.citusdata.com/en/v6.0/`.

[15] Compression algorithms in Akumuli. [Citované 2017-5-3] Dostupné z `https://docs.google.com/document/d/1yLsN1j8xxnm_b0oN6rFSgWOnCHP-OlJC5pBKZQwTAPc/pub`.

[16] CouchDB the definitive guide. eventual consistency. [Citované 2017-5-3] Dostupné z `http://guide.couchdb.org/draft/consistency.html`.

[17] CouchDB wiki. ACID Properties. [Citované 2017-5-3] Dostupné z `https://wiki.apache.org/couchdb/Technical%20Overview#ACID_Properties`.

[18] Cube time series data collection & analysis. [Citované 2017-2-9] Dostupné z `https://square.github.io/cube/`.

[19] Cyanite. [Citované 2017-2-10] Dostupné z `http://cyanite.io/`.

[20] DalmatinerDB documentation. [Citované 2017-2-9] Dostupné z `https://dalmatiner.readme.io/docs`.

[21] DataBus documentation. [Citované 2017-2-9] Dostupné z `https://github.com/deanhiller/databus/tree/master/doc/DataBus%20Documentation`.

[22] Druid vs. Key/Value stores (HBase/Cassandra/OpenTSDB). [Citované 2017-2-10] Dostupné z `http://druid.io/docs/latest/comparisons/druid-vs-key-value.html`.

[23] Elasticsearch as a time series data store. [Citované 2017-2-10] Dostupné z `https://www.elastic.co/blog/elasticsearch-as-a-time-series-data-store`.

[24] FiloDB documentation. [Citované 2017-2-10] Dostupné z `https://github.com/filodb/FiloDB#filodb-vs-cassandra-data-modelling`.

[25] Get started with Datadog. [Citované 2017-2-10] Dostupné z `http://docs.datadoghq.com/`.

[26] Gnocchi – metric as a service. [Citované 2017-2-9] Dostupné z `http://gnocchi.xyz/index.html`.

[27] Graphite documentation. [Citované 2017-2-9] Dostupné z `http://graphite.readthedocs.io/en/latest/index.html`.

[28] Hawkular overview. [Citované 2017-2-9] Dostupné z `http://www.hawkular.org/overview/index.html`.

[29] HDFS documentation. [Citované 2017-4-17] Dostupné z `https://hadoop.apache.org/docs/r1.2.1/index.html`.

[30] Heroic documentation. [Citované 2017-2-9] Dostupné z `https://spotify.github.io/heroic/#!/docs/overview`.

[31] InfiniFlux the world's fastest time series DBMS. [Citované 2017-2-10] Dostupné z `http://doc.infiniflux.com/get-started`.

[32] InfluxDB documentation. [Citované 2017-2-9] Dostupné z `https://docs.influxdata.com/influxdb/v1.2/`.

[33] Infomix TimeSeries solution. [Citované 2017-2-9] Dostupné z `http://www.ibm.com/support/knowledgecenter/en/SSGU8G_12.1.0/com.ibm.tms.doc/ids_tms_010.htm`.

[34] Instrumental documentation. [Citované 2017-2-10] Dostupné z `https://instrumentalapp.com/docs`.

[35] Introduction to array databases. [Citované 2017-2-28] Dostupné z `https://www.rd-alliance.org/group/array-database-assessment-wg/wiki/introduction-array-databases`.

[36] IronDB powered by Circonus. [Citované 2017-2-10] Dostupné z `http://login.circonus.com/resources/docs/irondb/`.

[37] KairosDB documentation. [Citované 2017-2-9] Dostupné z `https://kairosdb.github.io/docs/build/html/index.html`.

[38] kdb+ documentation. [Citované 2017-2-9] Dostupné z `http://code.kx.com/mkdocs/developer/`.

[39] Key-value stores. [Citované 2017-1-24] Dostupné z `http://db-engines.com/en/article/Key-value+Stores`.

[40] KsanaDB documentation. [Citované 2017-2-9] Dostupné z `https://github.com/zzzmanzzz/KsanaDB/wiki`.

[41] Kx offers nanosecond timestamps in updated database. [Citované 2017-2-9] Dostupné z `http://www.wallstreetandtech.com/latency/kx-offers-nanosecond-timestamps-in-updated-database/d/d-id/1262409`.

[42] LevelDB. [Citované 2017-5-3] Dostupné z `https://github.com/google/leveldb`.

[43] Librato knowledge base. [Citované 2017-2-9] Dostupné z `https://www.librato.com/docs/kb/`.

[44] Lz4 - extremely fast compression. [Citované 2017-5-3] Dostupné z `https://github.com/lz4/lz4a`.

[45] Metrictank. [Citované 2017-2-9] Dostupné z `https://github.com/raintank/metrictank`.

[46] MongoDB documentation. Isolated. [Citované 2017-5-3] Dostupné z `https://docs.mongodb.com/manual/reference/operator/update/isolated/`.

[47] MongoDB documentation. Indexes. [Citované 2017-5-3] Dostupné z `https://docs.mongodb.com/manual/indexes/`.

[48] MongoDB documentation. Update Operators. [Citované 2017-5-3] Dostupné z `https://docs.mongodb.com/manual/reference/operator/update/`.

[49] NewTS documentation. [Citované 2017-2-9] Dostupné z `https://github.com/OpenNMS/newts/wiki`.

[50] Numeric b+tree reference. [Citované 2017-4-28] Dostupné z `https://docs.google.com/document/d/1jFK8E3CZSqR5IPsMGojm2LknkNyUZA7tY51N6IgzW_g/pub`.

[51] OpenTSDB documentation. [Citované 2017-2-9] Dostupné z `http://opentsdb.net/docs/build/html/index.html`.

[52] Oracle Berkeley DB. [Citované 2017-5-3] Dostupné z `http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index-085366.html`.

[53] Orestes. [Citované 2017-2-9] Dostupné z `https://github.com/davidvgalbraith/orestes`.

[54] OrientDB manual - version 2.2.x. Transactions. [Citované 2017-5-3] Dostupné z `http://orientdb.com/docs/last/Transactions.html`.

[55] Prometheus documentation. [Citované 2017-2-9] Dostupné z `https://prometheus.io/docs/introduction/overview/`.

[56] Rhombus. [Citované 2017-2-9] Dostupné z `https://github.com/Pardot/Rhombus`.

[57] RHQ documentation. [Citované 2017-2-10] Dostupné z `https://docs.jboss.org/author/display/RHQ/Home`.

[58] RiakTS documentation. [Citované 2017-2-9] Dostupné z `http://docs.basho.com/riak/ts/1.4.0/`.

[59] Signal fx resource library. [Citované 2017-2-10] Dostupné z `https://signalfx.com/library/`.

[60] SiriDB documentation. [Citované 2017-2-10] Dostupné z `http://siridb.net/docs/`.

[61] SiteWhere 1.6.0 documentation. [Citované 2017-2-10] Dostupné z `http://documentation.sitewhere.org/index.html`.

[62] TempoIQ documentation. [Citované 2017-2-10] Dostupné z `https://app.tempoiq.com/docs/html/index.html`.

[63] TrailDB documentation. [Citované 2017-2-9] Dostupné z `http://traildb.io/docs/`.

[64] Transaction support in RavenDB. [Citované 2017-5-3] Dostupné z `https://ravendb.net/docs/article-page/3.5/Csharp/client-api/faq/transaction-support`.

[65] The Warp 10 platform. [Citované 2017-2-10] Dostupné z `http://www.warp10.io/`.

[66] Warp transactions for HyperDex. [Citované 2017-5-3] Dostupné z `http://hyperdex.org/warp/`.

[67] WaveFront resources. [Citované 2017-2-10] Dostupné z `https://www.wavefront.com/resources/`.

[68] Welcome to Ceph. [Citované 2017-4-17] Dostupné z `http://docs.ceph.com/docs/master/`.

[69] What is ZFS? [Citované 2017-4-17] Dostupné z `http://docs.oracle.com/cd/E19253-01/819-5461/zfsover-2/`.

[70] What type of locking does MongoDB use? [Citované 2017-5-3] Dostupné z `https://docs.mongodb.com/manual/faq/concurrency/#what-type-of-locking-does-mongodb-use`.

[71] Yanza. [Citované 2017-2-10] Dostupné z `http://yanza.com/`.

[72] Daniel Abadi. Distinguishing two major types of column-stores. [Citované 2017-1-24] Dostupné z `http://dbmsmusings.blogspot.sk/2010/03/distinguishing-two-major-types-of_29.html`.

[73] Steven Acreman. Playing with Dalmatiner DB. [Citované 2017-2-9] Dostupné z `https://blog.dataloop.io/2016/03/21/playing-with-dalmatiner-db/`.

[74] Michael P Andersen and David E Culler. BTrDB: Optimizing storage system design for timeseries processing. In *FAST*, pages 39–52, 2016.

[75] James Blackburn. Arctic: High-performance IoT and financial data storage with Python and MongoDB. [Citované 2017-2-9] Dostupné z `http://www.slideshare.net/JamesBlackburn1/2015-pydata-highperformance-iot-and-financial-data-storage-with-python-and-mongodb`.

[76] Brian Brazil. Scaling and federating Prometheus. [Citované 2017-2-9] Dostupné z `https://www.robustperception.io/scaling-and-federating-prometheus/`.

[77] Julian Browne. Brewer's CAP theorem. [Citované 2017-1-24] Dostupné z `http://www.julianbrowne.com/article/viewer/brewers-cap-theorem`.

[78] Maxim Buevich, Anne Wright, Randy Sargent, and Anthony Rowe. Respawn: A distributed multi-resolution time-series datastore. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 288–297. IEEE, 2013.

[79] Rick Cattell. Scalable SQL and NoSQL data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

[80] Evan Chan. Introduction to FiloDB. [Citované 2017-2-10] Dostupné z `http://velvia.github.io/presentations/2014-filodb/#/`.

[81] Adrian Colyer. BTrDB: Optimizing storage system design for timeseries processing. [Citované 2017-2-9] Dostupné z `https://blog.acolyer.org/2016/05/04/btrdb-optimizing-storage-system-design-for-timeseries-processing/`.

[82] Adrian Colyer. Gorilla: A fast, scalable, in-memory time series database. [Citované 2017-2-9] Dostupné z `https://blog.acolyer.org/2016/05/03/gorilla-a-fast-scalable-in-memory-time-series-database/`.

[83] Luca Deri, Simone Mainardi, and Francesco Fusco. tsdb: A compressed database for time series. In *International Workshop on Traffic Monitoring and Analysis*. Springer, 2012.

[84] Ted Dunning and Ellen Friedman. Time series databases. 2015.

[85] Armando Fox and Eric A Brewer. Harvest, yield, and scalable tolerant systems. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 174–178. IEEE, 1999.

[86] Dave Galbraith. Orestes: a time series database backed by Cassandra and Elasticsearch. [Citované 2017-2-9] Dostupné z `http://davidvgalbraith.com/orestes/`.

[87] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59, 2002.

[88] Omar Hajoui, Rachid Dehbi, Mohammed Talea, and Zouhair Ibn Batouta. An advanced comparative study of the most promising noSQL and newSQL databases with a multi-criteria analysis method. *Journal of Theoretical and Applied Information Technology*, 81(3):579, 2015.

[89] Coda Hale. You can't sacrifice partition tolerance. [Citované 2017-1-24] Dostupné z `https://codahale.com/you-cant-sacrifice-partition-tolerance/#ft2l`.

[90] Brian Harrington and Roy Rapoport. Introducing Atlas: Netflix's primary telemetry platform. [Citované 2017-2-10] Dostupné z `http://techblog.netflix.com/2014/12/introducing-atlas-netflixs-primary.html`.

[91] Lakshmi Kannan. Blueflood multi-tenanted time series datastore. [Citované 2017-2-9] Dostupné z `http://www.lakshmikannan.me/slides/2014-02-19-sf-metrics-meetup/#/`.

[92] Florian Lautenschlager. Chronix - a fast and efficient time series storage based on Apache Solr, 2016. [Citované 2017-2-9] Dostupné z `https://speakerdeck.com/florian_lautenschlager/chronix-a-fast-and-efficient-time-series-storage-based-on-apache-solr`.

[93] Michael Lussier. An introduction to Elasticsearch aggregations. [Citované 2017-2-10] Dostupné z `https://qbox.io/blog/elasticsearch-aggregations`.

[94] Tobias Oetiker. RRDtool. [Citované 2017-2-9] Dostupné z `https://oss.oetiker.ch/rrdtool/`.

[95] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.

[96] Ayende Rahien. That noSQL thing - document databases. [Citované 2017-1-24] Dostupné z `https://ayende.com/blog/4459/that-no-sql-thing-document-databases`.

[97] Margaret Rouse. Columnar database. [Citované 2017-3-5] Dostupné z `http://searchdatamanagement.techtarget.com/definition/columnar-database`.

[98] Dustin Sallings. Why so, Seriesly? [Citované 2017-2-9] Dostupné z `http://dustin.sallings.org/2012/09/09/seriesly.html`.

[99] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. The architecture of SciDB. In *Scientific and Statistical Database Management*, pages 1–16. Springer, 2011.

[100] Gregory Trubetskoy. Introducing Tgres - a time series DB on top of PostgreSQL. [Citované 2017-2-10] Dostupné z `https://grisha.org/blog/2016/07/29/state-of-tgres-2016/`.

[101] Gregory Trubetskoy. Storing time series in PostgreSQL efficiently. [Citované 2017-2-10] Dostupné z `https://grisha.org/blog/2015/09/23/storing-time-series-in-postgresql-efficiently/`.

[102] Eric Tschetter. Introducing Druid: Real-time analytics at a billion rows per second. [Citované 2017-2-10] Dostupné z `http://druid.io/blog/2011/04/30/introducing-druid.html`.

[103] Ville Tuulos. Announcing TrailDB - an efficient library for storing and processing event data. [Citované 2017-2-9] Dostupné z `http://tech.adroll.com/blog/data/2016/05/24/traildb-open-sourced.html`.

[104] Tom Valine. Argus: Time-series monitoring and alerting. [Citované 2017-2-9] Dostupné z `https://medium.com/salesforce-open-source/argus-time-series-monitoring-and-alerting-d2941f67864#.ojytdqjqf`.

[105] Mike Wooldridge. How MarkLogic supports ACID transactions. [Citované 2017-5-3] Dostupné z `http://developer.marklogic.com/blog/how-marklogic-supports-acid-transactions`.

Príloha A

NewTS

Formát súborov a skript na ich vytvorenie

Každý súbor obsahuje pole objektov.

```
[
{ "resource": {"id":"%ENTITA%, "attributes:{%TAGY%}}, "
   timestamp": %ČASOVÁ_PEČIATKA% ,"name": "%NÁZOV_ČASOVÉ
   HO_RADU%", "type": "%TYP%", "value": %ÚDAJ% },
...
]
```

Časová pečiatka je udávaná v milisekundách od 1.1.1970. Typ je v našom prípade GAUGE, čo označuje časový rad, ktorý môže ľubovoľne rásť aj klesať. Tagy sú páry reťazcov v JSON podobe.

```
def generateNewTSJSON():
    random.seed(100)
    myfile = open('/home/main/newtsData0.json', 'w+')
    myfile.write("[\n")
    indicator = 0
    counter = 0
    iterator = 0
    for i in range(1356998400, 1357084800):
        for j in range(2000):
            if iterator == 10000:
                counter = counter + 1
                iterator = 0
                myfile.write("]")
                myfile.close()
                myfile = open('/home/main/newtsData'+str(
                   counter)+'.json', 'w+')
                myfile.write("[\n")
```

```
                indicator = 0
            if indicator == 0:
                myfile.write('{ "resource": {"id": "XXX"},
                   "timestamp": '+str(i*1000) + ',"name": "
                   metric'+str(j)+'", "type": "GAUGE","
                   value": '+ str(random.randint(1, 5000))
                   + " }\n")
                indicator = 1
                iterator = iterator + 1
            else:
                myfile.write(',{ "resource": {"id": "XXX
                   "},"timestamp": '+str(i*1000) + ',"name
                   ": "metric'+str(j)+'", "type": "GAUGE","
                   value": '+ str(random.randint(1, 5000))
                   + " }\n")
                iterator = iterator + 1
    myfile.write("]")
    myfile.close()
```

Príkaz na nahranie dát do systému

```
curl -D - -X POST -H "Content-Type: application/json" -d @%
   SÚBOR% http://%ADRESA_SERVERU%:8080/samples
```

OpenTSDB

Formát súborov a skript na ich vytvorenie

OpenTSDB používa vlastný formát citlivý na biele znaky. Každý záznam musí mať aspoň jeden tag, tagy nesmú obsahovať medzery a je možné ich vymenovať viacero v jednom riadku. Časová pečiatka môže byť uvedená v sekundách alebo milisekundách, my sme použili sekundy od 1.1.1970. Vzhľadom na charakteristiku dopytovacieho jazyka sme použili jednotný názov pre všetky časové rady a odlíšili sme ich pomocou tagu.

```
%NÁZOV_ČASOVÉHO_RADU% %ČASOVÁ_PEČIATKA% %ÚDAJ% %KĽÚČ_TAGU1
   %=%HODNOTA_TAGU1% ...
```

```
def generateOpentsdb():
    random.seed(100)
    counter = 0
```

```
iterator = 0
myfile = open('/home/main/opentsdbData'+str(counter)+'.
   dat', 'w+')
for i in range(1356998400,1357084800):
    for j in range(2000):
        if iterator == 200000:
            counter = counter+1
            iterator = 0
            myfile.close()
            myfile = open('/home/main/opentsdbData'+str
               (counter)+'.dat', 'w+')
        myfile.write("metrika "+str(i)+" "+str(random.
           randint(1,5000))+" symbol="+"metric"+str(j)
           +"\n")
        iterator = iterator + 1
myfile.close()
```

Príkaz na nahranie dát do systému

Pred samotným importom nových časových radov je nutné oznámiť systému, aké mená sa budú používať. Robí sa to pomocou skriptu, ktorý systém zapína.

```
%BUILD_ADRESÁR_OPENTSDB%/tsdb mkmetric %NÁZOV1% ...
```

Import prebieha taktiež pomocou skriptu, ktorým sa systém zapína.

```
%BUILD_ADRESÁR_OPENTSDB%/tsdb import %SÚBOR%
```

Dopyt

```
start=%ČASOVÁ_PEČIATKA% end=%ČASOVÁ_PEČIATKA% m=avg:%NÁZOV_
   ČASOVÉHO_RADU%
```

Dopyt sa nahráva pomocou príkazu (medzery v dopyte nahradíme znakom &):

```
curl "http://%ADRESA_SERVERU%:4242/api/query?%DOPYT%"
```

Axibase TimeSeries database

Formát súborov a skript na ich vytvorenie

V našich testovacích dátach sme použili nasledovnú štruktúru CSV súborov. Túto štruktúru však možno meniť pomocou konfigurácie CSV parsera. Každý riadok obsahoval práve jeden záznam časového radu a časové rady sa líšili názvom entity. Samotný názov mali kvôli potrebám dátového modelu spoločný.

```
"%ČASOVÁ_PEČIATKA%","%ENTITA%","%NÁZOV_ČASOVÉHO_RADU%","%
   HODNOTA_TAGU%","%ÚDAJ%"
```

Časová pečiatka bola udávaná v sekundách od 1.1.1970. Kľúč tagu bol definovaný v konfigurácií parseru a teda bol tiež pre všetky časové rady spoločný.

```
def generateAxibase():
    counter = 0
    iterator = 0;
    random.seed(100)
    myfile = open('/home/main/axibaseData'+str(counter)+'.
       csv', 'w+')
    for i in range(1356998400,1357084800):
        for j in range(2000):
            if iterator ==  3456000:
                myfile.write("\n")
                iterator = 0
                counter = counter + 1
                myfile.close()
                myfile = open('/home/main/axibaseData'+str(
                   counter)+'.csv', 'w+')
            myfile.write('"'+str(i)+'","metric'+str(j)+'","
               test","XXX","'+str(random.randint(1,5000))
               +'"\n')
            iterator = iterator + 1;
    myfile.close()
```

Príkaz na nahranie dát do systému

```
wget --verbose --password=%HESLO% --user=%MENO% --auth-no-
   challenge --header="Content-type: text/csv"  --post-file
   =%SÚBOR% "http://%ADRESA_SERVERU%:8088/csv?config=%NÁ
   ZOV_PARSERA%"
```

Dopyt

```
SELECT datetime, AVG(value)
  FROM "%NÁZOV_ČASOVÉHO_RADU%"
GROUP BY datetime
```

Dopyt sa podľa dokumentácie nahráva nahráva pomocou príkazu:

```
curl http://%ADRESA_SERVERU%:8088/api/sql --insecure --
   verbose --user %MENO%:%HESLO% --request POST --data '%
   DOPYT%'
```

Nám však tento spôsob nefungoval, preto sme po analýze niektorých skriptov od výrobcu použili nasledovný tvar:

```
curl http:///%ADRESA_SERVERU%:8088/api/sql --insecure --
   verbose --user %MENO%:%HESLO% --request POST --data-
   urlencode 'q=%DOPYT%'
```

InfluxDB

Formát súborov a skript na ich vytvorenie

Dokumentácia popisuje formát Line Protocol, použiteľný na import dát. Tento formát je citlivý na biele znaky a každý riadok v ňom obsahuje jeden záznam. Štruktúra riadkov je nasledovná:

```
%NÁZOV_ČASOVÉHO_RADU%,%KĽÚČ_TAGU1%=%HODNOTA_TAGU1% ...
   value=%ÚDAJ% %ČASOVÁ_PEČIATKA%
```

Časová pečiatka je udávaná v nanosekundách od 1.1.1970.

```
def generateInflux():
    random.seed(100)
    counter = 0
    iterator = 0
    myfile = open('/home/main/influxData'+str(counter)+'.
       dat', 'w+')
    for i in range(1356998400, 1357084800):
        for j in range(2000):
            if iterator==4500:
                myfile.close()
                iterator = 0
                counter = counter +1
                myfile = open('/home/main/influxData'+str(
                   counter)+'.dat', 'w+')
            myfile.write('metrika,testtype=metric'+str(j)+'
                value='+str(random.randint(1, 5000))+' '+
               str(i*1000000000)+'\n')
            iterator += 1
    myfile.close()
```

Príkaz na nahranie dát do systému

Pred začatím nahrávania je potrebné v systéme vytvoriť databázu. To je možné buď pomocou príkazov v CLI na serveri alebo pomocou POST dopytu odoslaného nasledujúcim príkazom:

```
curl -i -XPOST 'http://%ADRESA_SERVERA%:8086/query' --data-
   urlencode "q=CREATE DATABASE %NÁZOV_DATABÁZY%"
```

Nahrávanie jednotlivých súborov je zabezpečené príkazom:

```
curl -i -XPOST 'http://%ADRESA_SERVERA%:8086/write?db=%NÁ
   ZOV_DATABÁZY%' --data-binary @%SÚBOR%
```

Dopyt

Podobne ako pri OpenTSDB, až po interakcií so systémom sme plne porozumeli dátovému modelu systému a zistili sme z neho vyplývajúce obmedzenia pre agregácie cez viacero čas. radov. Tieto agregácie sú možné len v rámci jedného tzv. merania (measurement). Opäť je teda nutné použiť spoločný názov a odlíšiť jednotlivé časové rady pomocou tagov.

```
SELECT mean(value) FROM %NÁZOV_ČASOVÉHO_RADU% WHERE time <
   %HORNÁ_HRANICA_ČASU% GROUP BY time(1s)
```

Dopyt sa nahráva pomocou príkazu:

```
curl -G 'http://%ADRESA_SERVERA%:8086/query' --data-
   urlencode "db=%NÁZOV_DATABÁZY%" --data-urlencode "%DOPYT
   %"
```

alebo na serveri:

```
influx -database '%NÁZOV_DATABÁZY%' -execute "%DOPYT%"
```

Chronix

Formát súborov a skript na ich vytvorenie

Dokumentácia popisuje nutnosť vopred si premyslieť, aké tagy (atribúty) budú importované časové rady používať a informácie o nich doplniť do súboru schema.xml. Tento súbor sa nachádza v priečinku `%CHRONIX_PRIEČINOK%/server/solr/chronix/conf/` Dokumentácia uvádza v časti venovanej atribútom aj informáciu o nutnosti uviesť informácie o názve, začiatku a konci časového radu, popis však nie je dostatočný na to, aby sa dalo zistiť, kam tieto informácie patria. V rámci experimentu sme zistili, že systém si s importom vie poradiť aj bez explicitného špecifikovania týchto údajov.

V konfiguračnom súbore importeru je potrebné určiť názvy tagov (atribútov) v takom poradí, v akom sa budú nachádzať v názvoch importovaných súborov. Hodnoty v názve súboru treba uviesť v uvedenom poradí a oddeliť ich podtržníkom. V konfiguračnom súbore je tiež potrebné zadať aj typ oddeľovača použitého v CSV súboroch. My sme použili bodkočiarku.

Prvý riadok importovaného súboru musí obsahovať hlavičku, ktorá sa skladá z povinného prvého stĺpca a vymenovaných názvov časových radov, ktoré chceme importovať.

```
DATE;%NÁZOV_ČASOVÉHO_RADU_1%;%NÁZOV_ČASOVÉHO_RADU2% ...
```

ďalšie riadky majú formát

```
%ČASOVÁ_PEČIATKA%;%ÚDAJ_ČASOVÉHO_RADU1%;%ÚDAJ_ČASOVÉ
   HO_RADU2% ...
```

Časová pečiatka má formát špecifikovaný v konfiguračnom súbore. V experimente sme použili formát `dd.mm.YY HH:MM:SS`.

```
import time
def generateChronix():
    random.seed(100)
    counter = 0
    iterator = 0
    myfile = open('/home/main/importer-0.5-beta/data/XXX.
       csv', 'w+')
    myfile.write("DATE")
    for j in range(2000):
        myfile.write(";metric"+str(j))
    myfile.write("\n")
    for i in range(1356998400, 1357084800):
        if iterator == 10000:
            iterator = 0
            myfile.close()
            a = input("next") #aby boli konzistentné tagy,
               súbor musíme prepisovať. Je teda potrebné po
               čas behu skriptu urobiť prestávku a
               importovať vytvorený súbor
            myfile = open('/home/main/importer-0.5-beta/
               data/XXX.csv', 'w+')
            myfile.write("DATE")
            for j in range(2000):
```

```
                myfile.write(";metric"+str(j))
            myfile.write("\n")
        myfile.write('%s.%03d' % (time.strftime('%d.%m.%Y %
           H:%M:%S', time.gmtime(i)), 000))
        for j in range(2000):
            myfile.write(";"+str(random.randint(1, 5000)))
        myfile.write("\n")
        iterator += 1
    myfile.close()
```

Príkaz na nahranie dát do systému

Súbory s dátami je potrebné umiestniť do priečinku `data/` v priečinku, v ktorom sa nachádza importer. Potom stačí spustiť skript `import.sh`, ktorý spustí importer s všetkými potrebnými argumentami.

SiriDB

Formát súborov

Formát súboru nie je dokumentovaný, no vďaka analýze zdrojových skriptov sme zistili, že formát je veľmi podobný formátu systému Chronix. Jednotlivé polia v CSV súbore sú oddelené čiarkou. Prvý riadok súboru je hlavičkou, obsahuje prázdny prvý stĺpec a za ním vymenované názvy časových radov. Na nasledujúcich riadkoch už sú údaje spolu s časovou pečiatkou. Prvý riadok má formát:

```
,%NÁZOV_ČASOVÉHO_RADU_1%,%NÁZOV_ČASOVÉHO_RADU2% ...
```

Ďalšie riadky majú formát:

```
%ČASOVÁ_PEČIATKA%,%ÚDAJ_ČASOVÉHO_RADU_1%,%ÚDAJ_ČASOVÉ
   HO_RADU2% ...
```

```
def generateSiriDB():
    random.seed(100)
    counter = 0
    iterator = 0;
    myfile = open('XXX.csv', 'w+')
    for j in range(2000):
        myfile.write(",metric"+str(j))
    myfile.write("\n")
    for i in range(1356998400, 1357084800):
        myfile.write(str(i))
```

```
        for j in range(2000):
            myfile.write(","+str(random.randint(1, 5000)))
        myfile.write("\n")
        iterator += 1
        if iterator ==  1000:
            iterator = 0
            myfile.close()
            a = input("next") #použili sme prepisovanie sú
               boru podobne ako pri Chronix
            myfile = open('XXX.csv', 'w+')
            for j in range(2000):
                myfile.write(",metric"+str(j))
            myfile.write("\n")
    myfile.close()
```

Príkaz na nahranie dát do systému

Súbory sa do systému nahrávajú pomocou CLI. Príkaz je

```
import_csv %SÚBOR%
```

Dopyt

SiriDB podporuje agregácie cez viacero radov pomocou tzv. merge funkcionality. Dopyty sa píšu do CLI alebo odosielajú pomocou webového rozhrania. CLI je jednoduchšie použiteľné a lepšie zdokumentované, rozhodli sme sa teda použiť to.

```
select * from %ČASOVÉ_RADY% merge as %VÝSLEDOK% using mean
   (%ROZLÍŠENIE%)
```

Časové rady je možné v dopyte vymenovať, alebo použiť regulárny výraz. V našom prípade sme používali názvy čas. radov `metric0` až `metric1999` a tak sme zvolili regulárny výraz `/metric.*/`. Rozlíšenie bolo 1s. Na odmeranie času, potrebného na dopyt sme použili vstavaný CLI príkaz timeit, ktorý stačí pridať pred dopyt.