VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZPRACOVÁNÍ VELKÝCH DAT V OBLASTI PRŮMYSLU 4.0

BIG DATA PROCESSING IN INDUSTRY 4.0

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB TRUBKA**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Trubka Jakub**
Program: Informační technologie
Název: **Zpracování velkých dat v oblasti Průmyslu 4.0**
Big Data Processing in Industry 4.0
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s metodami zpracování velkých dat, jejich filtrováním, vyhodnocováním, zpracováním a vizualizací.
2. S využitím existujících implementací navrhněte a realizujte systém, který bude analyzovat informace z různých čidel, dostupných v moderních průmyslových provozech, a na základě výsledků navrhovat optimalizace specifických procesů.
3. Vyhodnoťte chování systému na dodaných historických datech a předpoklady pro integraci výsledků v reálném provozu, diskutujte robustnost a škálovatelnost realizovaného systému.
4. Vytvořte stručný plakát prezentující práci, její cíle a výsledky

Literatura:

- dle domluvy s vedoucím

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Cílem této bakalářské práce je vytvořit systém pro sběr, zpracování a uchovávání velkých dat získaných sledováním strojů v průmyslu. Výsledný systém klade důraz na snadnou škálovatelnost a rozšiřitelnost. V rámci teoretické části jsou popsána existující řešení a rozebrána problematika sběru a zpracování velkých dat. Velká pozornost je také věnována technologiím úložišť pro velká data. Praktická část se věnuje návrhu a implementaci celého systému a jeho jednotlivých částí, stejně jako následnému testování a konečnému vyhodnocení realizovaného řešení.

Abstract

This thesis deals with collecting, processing, and storing big data obtained from monitoring industry machines. The designed and implemented system focuses on extensibility and scalability attributes of the realised solution. The survey part of the text briefly describes existing solutions and discusses collecting and processing big industrial data. A special attention is also paid to the big data storage technology. The crucial part of the thesis then refers to the design and realisation of the system and its individual components, as well as its testing and final evaluation.

Klíčová slova

Průmysl 4.0, velká data, časové databáze, Raspberry Pi, Linux, SPI, IoT, bash

Keywords

Industry 4.0, big data, time-series databases, Raspberry Pi, Linux, SPI, IoT

Citace

TRUBKA, Jakub. *Zpracování velkých dat v oblasti Průmyslu 4.0.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Zpracování velkých dat v oblasti Průmyslu 4.0

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Další informace mi poskytla partnerská firma 4dot mechatronic systems. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Trubka
11. května 2020

Poděkování

Děkuji vedoucímu bakalářské práce doc. RNDr. Pavlovi Smržovi, Ph.D. za rady, trpělivost a ochotu při konzultacích. Poděkování patří také firmě 4dot Mechatronics Systems za poskytnutí prototypu jednotky pro měření dat.

Obsah

Kapitola 1

Úvod

Žijeme v době čtvrté průmyslové revoluce. Tato revoluce se projevuje rozšířenou automatizací a digitalizací továren. Továrny potenciálně generují obrovské množství různých dat, které stačí začít sbírat a zpracovávat. Tato data lze využít k zefektivnění procesů výroby, a tím zvýšení zisků, což je pro podnikatelský sektor klíčové. Jedná se tedy o jeden z důvodů vzestupu Průmyslu 4.0 v posledních letech. Jednou z rozvíjejících se oblastí tohoto odvětví je také sledování strojů a jejich prediktivní údržba. Poruchy na klíčových zařízeních mohou způsobit několikadenní odstávky celé výrobní linky, a tím pádem znatelné finanční škody. Prediktivní údržba a monitorování strojů v reálném čase může těmto situacím zabránit. Technologie umožňující prediktivní údržbu je založena na datech získaných ze senzorů umístěných na průmyslových strojích. Tyto senzory měří různé fyzikální veličiny, nejčastěji vibrace či teplotu. Pomocí matematických analýz lze následně vysledovat stav stroje, odhalit jeho nestandardní chování a faktory vedoucí k poruše, a tím poruchu odhalit či jí předcházet. Tato metoda vyžaduje zpracování obrovského množství dat.

Systém navržený a implementovaný v této bakalářské práci se opírá o existující hardwarové řešení partnerské firmy zabývající se monitorováním, prediktivní analýzou a údržbou průmyslových strojů. Technologie firmy, včetně prototypu zařízení poskytnutého pro účely vypracování této bakalářské práce, jsou popsány dále v textu. Na vývoji prototypu zařízení jsem se aktivně podílel. Hardware zařízení je navržen tak, aby cenově zpřístupnil možnost monitorování strojů a predikci poruch nejen pro velké, ale i pro menší firmy. Cílem této práce je navrhnout a implementovat software pro tuto jednotku a serverový software schopný s ní spolupracovat. Celý sytém umožňuje rozsáhlé nasazení monitorovacích zařízení v krátkém časovém úseku. Klíčovým aspektem je proto snadná softwarová příprava měřících jednotek, jejich rychlé zavedení do systému a spuštění. Další důležitou součástí je možnost škálovatelnosti serverové části, zpracovávání velkého množství naměřených hodnot a integrace celého systému u firem.

V kapitole 2 je diskutována nová generace průmyslu, důležitost využití velkých dat v této oblasti a existující řešení pro monitorování průmyslových strojů. Informace týkající se sběru, zpracování a zobrazení velkých dat, které byla využita při návrhu systému, jsou shrnuty v kapitole 3. Možnost zápisu, uchování a přístupu k obrovskému množství dat je jednou z nejdůležitějších a nejkritičtějších částí práce s velkými daty. Této tematice je proto věnována samostatná kapitola 4, popisující specifické požadavky na úložiště, základní principy často využívaných technologií a také příklady některých úložišť velkých dat. Základní požadavky a návrh celého systému obsahuje kapitola 5. Implementace systému a zajímavé úseky softwarové části práce popisuje kapitola 6. Testování jednotlivých částí systému je obsaženo v kapitole 7.

Kapitola 2

Velká data v Průmyslu 4.0

Vize Průmyslu 4.0 spočívá v dokončení digitalizace a automatizace průmyslu a směřování k autonomním továrnám. Dochází k propojení počítačových systémů zavedených v předchozí fázi třetí průmyslové revoluce a také všech součástí výrobního procesu včetně strojů. Právě průmyslové stroje jsou potenciálním zdrojem obrovského množství užitečných informací. U strojů lze sledovat například kvalitu výstupních produktů, vibrace či teplotu jednotlivých částí. Tyto informace umožňují rychlou detekci a předpověď poruch v reálném čase nebo zaznamenávání vytížení stroje v konkrétních fázích výroby. Agregovaná data mohou posloužit k vyhodnocení výrobních procesů, nalezení slabých míst a následnému zefektivnění jednotlivých procesů.

V poslední době jsou nasazovány takzvané chytré stroje, osazené různými senzory a výpočetními jednotkami pro sběr dat a komunikaci se zbytkem systému. Nejen nové stroje nabízí možnosti propojení s celým systémem, mnoho firem se zaměřuje na osazování starých strojů senzory a výpočetními jednotkami, aby i tyto stroje mohly být monitorovány a propojeny se systémem. Příklady těchto firem a jejich produktů jsou popsány dále v této kapitole.

2.1 Využití velkých dat v průmyslu

Faktické informace k využití velkých dat byly čerpány z [5]. Velká data mají v dnešním průmyslu široké využití ve všech odvětvích. V chemickém odvětví se využívají k analýze vnějších vlivů, jako teploty či tlaku, na výsledný produkt. Umožňuje tak úpravu těchto vlivů, což přináší až pětinové snížení produkce odpadních látek či šestinovou úsporu energie. Stejným způsobem dochází ke zkoumání vlivu různých faktorů na produkci i v průmyslu farmaceutickém, kde dochází až k padesátiprocentnímu navýšení produkce díky analýze velkých dat. Další odvětví, kterému analýza velkých dat pomáhá, je automobilový průmysl. Data získaná a zpracovaná z prototypů či již produkčních vozidel umožňují rychlé zjištění slabin či chyb ať už návrhové či produkční fázi vozidel. Následně je možné v produkci tyto chyby rychleji odstranit a tím docílit menších nákladů při dodatečných opravách. Další využití nachází energetický průmysl. Například u větrných elektráren se již dnes využívá analýza velkého množství dat ze senzorů pro ovládání větrných turbín a maximalizaci efektivity při výrobě energie. Historická data a data získávaná v reálném čase následně slouží k vytváření prediktivních modelů pro optimalizaci provozu nejen elektráren a pro ulehčení rozhodování vrcholového managementu a to ve všech průmyslových odvětvích.

2.2 Existující systémy

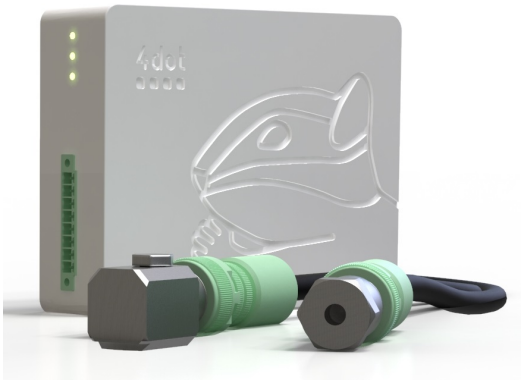
Jak již bylo zmíněno, Průmysl 4.0 je velmi rychle se rozvíjející obor. Stejně rychle vznikají systémy pro sběr a zpracování velkých dat pro průmysl. Jednou z vlastností průmyslu je jeho rozmanitost. Tím je myšleno obrovské množství specializovaných strojů a postupů, které se používají v jednotlivých odvětvích. Tato variabilita způsobuje to, že je velice těžké, a nákladné vytvořit univerzální systém. Právě z toho důvodu je velká část jak hardwarových, tak i softwarových řešení, specializovaných. Zároveň existuje nepřeberné množství parametrů, které lze sledovat – od teploty vzduchu v provozu až po výkonnost zaměstnanců. Tato sekce se zabývá existujícími hardwarovými a softwarovými řešeními pro sledování stavu průmyslových strojů (teploty, vibrací, otáček a podobně).

2.2.1 Hardwarové řešení

Zařízení pro sběr velkých dat se dají rozdělit na jednoúčelová a víceúčelová. Jednou ze zahraničních firem věnujících se vývoji jednotek pro sběr a zpracování dat je National Instruments. Firma se zabývá vývojem automatizovaného testovacího vybavení a nabízí celý ekosystém výrobků, mimo jiné i pro sběr dat ze senzorů a zpracování signálu. Pro sběr a zpracování dat nabízí jednotku cRIO zobrazenou na obrázku 2.1. Toto zařízení je víceúčelové. Velkou výhodou je, že zařízení obsahuje kromě procesoru i programovatelné hradlové pole neboli FPGA, umožňující optimalizovat některé výpočty přímo na jednotce. Další výhodou je možnost připojení velkého množství rozšiřujících modulů pro různé vstupy, od bezpečnostních modulů přes zvukové moduly až po drivery motorů. Velkou nevýhodou je cena. Plně vybavené zařízení s potřebnými rozšiřujícími moduly a softwarem pro programování může vyjít na statisíce korun za kus [17].

Příkladem jednoúčelových zařízení může být jednotka partnerské firmy zvaná Chipmunk. Jednotka je zobrazena na obrázku 2.2. Zařízení obsahuje pouze procesor bez hradlového pole a není rozšiřitelné o moduly. Zařízení má pevně dané čtyři vstupy pro senzory vibrací se snímáním o vzorkovací frekvenci až 128 kHz [1]. Kromě této firmy existují nejen ve světě, ale i v České republice, desítky firem vyvíjející podobné jednotky, určené ke sledování nepřeberného množství různých parametrů strojů. Jednoúčelové jednotky sice nabízí oproti víceúčelovým mnohem užší spektrum využití, zato jsou dostupné za řádově nižší cenu a jsou vhodnější pro masové nasazení.

Obrázek 2.1: Jednotka cRIO s rozšiřujícími moduly [17].

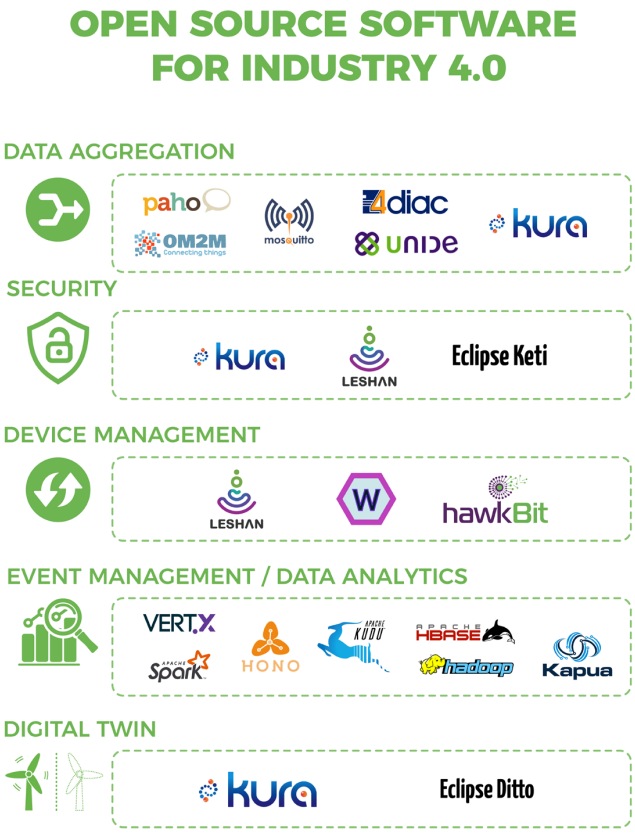


Obrázek 2.2: Jednotka Chipmunk se senzory vibrací [1].

2.2.2 Softwarové řešení

Softwarové řešení pro sběr a zpracování dat má dvě základní části. První částí je softwarové vybavení, kterým jsou vybaveny jednotky. Ty data sbírají, a v omezené míře i zpracovávají. Druhou částí řešení jsou centralizované serverové aplikace sbírající a zpracovávající data od jednotek. Tyto aplikace se starají také o přístup k výsledným informacím a jejich vyhodnocení. Existují i lokální řešení, která jsou omezena pouze na jednotku poskytující informace přímo u monitorovaného stroje (například na monitor). Tyto jednotky dále nekomunikují s jinými zařízeními. Takovým systémům se však tato podsekce nevěnuje, protože nesplňují cíl Průmyslu 4.0 – úplné propojení všech částí do jednoho celku.

Software obsažený v jednotkách je často vytvořen přímo firmou, která vyvinula dané zařízení. Se systémem komunikuje pomocí řady standardizovaných protokolů určených pro průmysl. Příkladem může být jednotka pro monitorování vibrací společnosti ifm[1] s možností výběru komunikačního rozhraní jako je PROFINET IO, Modbus, EtherCAT. Jinou cestou se vydala již zmíněná americká společnost National Instruments, která prodává jednotky bez softwarového vybavení. Programy pro tyto jednotky se vytváří v programu zvaném LabView[2] a opět lze volit různá komunikační rozhraní. Více informací ke komunikačním protokolům je obsaženo v sekci 3.5.



Obrázek 2.3: Kategorie komponent systému pro Průmysl 4.0 a jejich zástupci [10].

[1] https://www.ifm.com/gb/en/category/070/070_010/070_010_015#!/S/BD/DM/1/D/0/F/0/T/24
[2] https://www.ni.com/cs-cz/shop/labview.html

Serverová část systému slouží k příjmu dat od jednotek, jejich zpracování a uložení. Tyto systémy mohou mít mnoho funkcí a řadu různých komponent. Komponenty se mohou dělit podle funkce do několika kategorií: příjem dat a agregace, zabezpečení, správa zařízení, správa událostí, datové analýzy, datová úložiště a další. Existuje velké množství implementací jednotlivých komponent a to nejen placené, ale i volně dostupné pod open-source licencemi. Příklady zástupců jednotlivých kategorií jsou zobrazeni na obrázku 2.3.

Implementaci některých komponent nabízí společnost Eclipse. Příkladem může Eclipse Paho a Eclipse Mosquitto implementující protokol MQTT (Message Queuing Telemetry Transport). Velká sada softwarových komponent vhodných pro Průmysl 4.0 spadá pod Apache Software Foundation, která obsahuje mimo jiné i nástroje pro distribuované zpracování dat a výpočty či distribuovaná úložiště [10].

Serverové řešení jako takové tedy není omezeno na jednu určitou implementaci jako u jednotek, ale je možné vybírat z různých volně dostupných či placených komponent a postavit systém na míru podle potřeby a využití.

Kapitola 3

Zpracování velkých dat

Získaná data je potřeba zpracovat a vytěžit z nich co největší množství důležitých informací. Při menším objemu dat se nejedná o větší problém, ale pokud se bavíme o velkých datech, u kterých je potřeba zpracovávat i několik milionů hodnot za vteřinu, nastávají potíže. Programy pro zpracovávání velkých dat jsou obvykle součástí celých systémů pro práci s velkými daty včetně úložišť, vizualizačních programů a často jsou právě s úložišti pevně spjaté. Takovým příkladem je Kapacitor, který je součástí platformy Tick, jejíž schéma je vidět na obrázku 4.5. Způsob zpracování také závisí na povaze dat. Nejzákladnějším rozdělením zpracování velkých dat je proudově a blokově orientované.

3.1 Proudově orientované zpracování

Informace o proudově orientovaném zpracování dat pochází z [7]. Jedná se o zpracování toku dat v reálném čase, data jsou tudíž zpracovávána okamžitě při příchodu a ukládají se až po zpracování. V některých aplikacích data po zpracování ukládána ani nejsou, a to z důvodu jejich obrovské velikosti. Tento přístup se využívá v případech, kdy je vyžadována nízká odezva mezi příchodem dat a jejich efektem. Efektem může být výstup v podobě upozornění pro uživatele. Tato podmínka je kritická pro všechny typy monitorování. Ať už jde o kontrolu síťového provozu, protipožární ochranu či detekci poruch na průmyslových strojích. Nízká odezva je extrémně důležitá pro rychlé reakce v případech nenadálých událostí. Tuto podmínku nemůže splnit zpracování po blocích.

Proudově orientované zpracování musí zajišťovat vysokou dostupnost, aby veškeré události byly ihned zpracovány. Toho se dosahuje pomocí:

- distribuce zpracování na několik uzlů a možnosti další škálovatelnosti,
- replikace zpracování, při poruše jednoho uzlu lze zachytit událost na jiném uzlu a nedochází ke ztrátě dat,
- zpracování ve vnitřní paměti pro dosažení co nejmenší odezvy po příchodu dat a jejich zpracováním,
- paralelního zpracování proudu.

Jedním z příkladů frameworku implementujícího zpracování v reálném čase je Kapacitor či Apache Spark. Apache Spark poskytuje rozhraní pro psaní programů a jejich distribuci na výpočetních uzlech. Spark zajišťuje toleranci chyb a škálovatelnost stejně jako model

MapReduce a může zapisovat do všech databází využívající tento model. Knihovna Spark Streaming poskytuje rozšíření o možnost zpracovávání dat v reálném čase s vysokou propustností. Spark mimo jiné také poskytuje možnost integrace s nástroji z Apache hadoop, které jsou původně zaměřené na blokové zpracování dat.

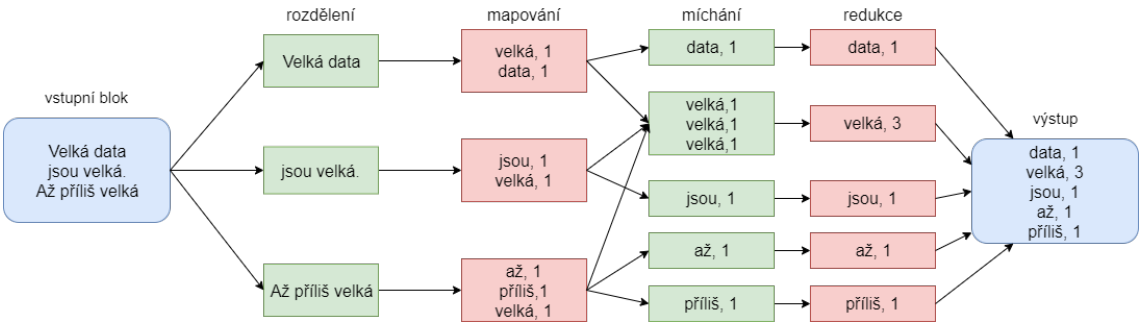
3.2 Blokově orientované zpracování

Tato metoda zpracovává data po blocích, které již byly shromážděny v paměti. Nevýhodou tedy je zpoždění získání výsledků analýz z přijatých dat. Využívá se v oblastech, kde není potřeba rychlé zpracování hodnot. Například při periodických výpočtech maxim, minim či středních hodnot z dat uložených v časových databázích. Jedním z programů, který poskytuje zpracování po blocích dat čtených z časové databáze InfluxDB, je Kapacitor. Kapacitor tedy nabízí obě možnosti zpracování dat, je proto vhodný jak pro sledování událostí v reálném čase, tak pro generování zpráv za delší časové období [22].

MapReduce

Zdrojem této podsekce je [12]. Mimo později vyvinutý Apache Spark zmíněný výše je Apache Hadoop primárně zaměřený na zpracovávání dat po blocích. K tomu využívá framework MapReduce. Jedná se o programovací model pro zpracování velkého množství dat po blocích pomocí paralelizace a je implementován v mnoha programovacích jazycích. Celý proces probíhá ve 4 fázích – rozdělení, mapování, míchání a redukování. Celý postup zpracování jednoho požadavku na výpočet počtu jednotlivých slov ve větě je vidět na obrázku 3.1. Vstupem jednotlivých fází je dvojice klíč-hodnota. Vstup se zpracovává uživatelem definovanou mapovací a redukční funkcí. Jednotlivé fáze provádí:

- v rozdělovací fázi dochází k rozdělení vstupních dat na formát, který zpracovává mapovací funkce,
- mapovací fáze vytvoří výsledek pomocí mapovací funkce, v našem případě se jedná o dvojici obsahující slovo a počet výskytů toho slova,
- následně při míchání dochází k setřídění výsledků,
- a v poslední fázi redukce dojde k agregaci a odeslání výsledku uživateli.



Obrázek 3.1: Příklad MapReduce modelu pro počítání výskytu jednotlivých slov [12].

V praxi je framework MapReduce distribuovaný, tudíž obsahuje více uzlů. Jeden ze serverů přijme požadavek a rozešle úkoly na provedení Map a Reduce funkcí na ostatní

uzly. Kvůli zamezení výpadku může provádět i duplicitní výpočet na více uzlech. Po získání výsledků provede dotázaný server zahození duplicitních mezivýsledků a odešle odpověď klientovi.

3.3 Zpracování dat ze senzorů vibrací

Tato sekce čerpá z [13]. Pro data z vibrací je důležitá funkce kvadratického průměru. Kvadratický průměr neboli RMS (Root Mean Square) se vztahuje k energii vibrací a je důležitým faktorem pro monitorování stavu strojů. Vzorec kvadratického průměru pro blok hodnot je:

$$x_{rms} = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}} \tag{3.1}$$

kde n je počet vzorků v bloku, a x_n je hodnota n-tého vzorku. Dále je pro data z vibrací důležitá maximální amplituda signálu a Rychlá Furierova Transformace. Rychlá Furierova Transformace se používá pro frekvenční analýzu vibrací která dokáže odhalit abnormální vibrace stroje způsobené poškozením určité součástky.

3.4 Komprese dat

Velká data mohou nabývat velikosti petabajtů nebo i více a je neefektivní je udržovat v nekomprimované podobě. Ať už jde o omezení síťové komunikace, či udržování dat na úložišti, je vhodné do určité míry omezit jejich velikost. Zároveň je nutné myslet na budoucí využití komprimovaných dat. Pokud mají být data rychle k dispozici pro provádění analýz nebo vizualizaci, musí být dekomprimace dat rychlá. Tato sekce se zaměřuje na popis metod využitelných pro body obsahující kombinaci celočíselné časové značky a naměřené desetinné hodnoty. Komprese tohoto formátu je potřebná při čtení a zpracování dat ze senzorů při monitorování strojů, na což je tato práce zaměřená.

3.4.1 Delta encoding

Komprese pomocí metody delta encoding je vhodná pro pravidelně či málo se měnící hodnoty celých čísel, jako je unixová časová značka. Metoda je založená na výpočtu rozdílu sousedních hodnot, vzorec pro výpočet je následující [23]:

$$\Delta_t = t_n - t_{n-1} \tag{3.2}$$

Touto metodou může dojít ke snížení rozsahu hodnoty nutné k uložení čísla. V případě unixové 32bitové časové značky, která je zaznamenávána při pravidelném měření, může dojít k citelné kompresi dat. Příklad je možné vidět v tabulce 3.1. První sloupec tabulky obsahuje nekomprimované časové značky o velikosti 32 bitů. V druhém sloupci se nachází Δt těchto hodnot. První bod se pouze přesune z důvodu zachování počátečního bodu a nedojde tedy k jeho kompresi. K uložení rozdílů, které v tomto případě nabývají pouze kladných hodnot, postačí 6 bitů poskytujících rozsah 0–127. Čím výraznější změna mezi hodnotami nastává, tím horších výsledků je dosaženo. Tato metoda tedy nemá smysl pro kompresi například řetězce ASCII znaků. Jak lze vidět na sloupci $\Delta ASCII$, výsledky nabývají i záporných hodnot. K zakódování jednoho znaku bychom potřebovali 8 bitů. Jelikož je ASCII znak reprezentován 8bitovou hodnotou, nebylo by dosaženo žádného zlepšení [23].

Časová databáze Gorilla (viz 4.4.2) využívá pro kompresi časových značek pozměněnou verzi této metody založenou na podmínce, že měření se opakuje v pravidelných intervalech. Vzorec 3.3 tedy odečítá deltu předchozích hodnot od delty aktuálních hodnot. Hodnoty jsou ukládané po dvouhodinových blocích. Hlavička obsahuje časovou značku zarovnanou na začátek daného dvouhodinového bloku a první delta využívá původní vzorec 3.2, kde t_{n-1} je časová značka z hlavičky. Hodnoty jsou následně ukládány s proměnlivou délkou[19].

$$D = (t_n - t_{n-1}) - (t_{n-1} - t_{n-2}) \tag{3.3}$$

Možnost využití této úpravy je viditelná i na vzorových datech v tabulce 3.1, kde měření probíhá zhruba jednou za minutu. Jak je vidět ve sloupci $\Delta\Delta t$ (počáteční hodnota zarovnaná na začátek dvouhodinového intervalu činí 1577664000), oproti předchozí verzi metody došlo ke kompresi první hodnoty z 32 na 5 bitů a zbylé hodnoty je možné uložit na 4 bitech.

časová značka	Δt	$\Delta\Delta t$	znak	ASCII hodnota	$\Delta ASCII$
1577664060	1577664060	60	B	66	66
1577664120	60	0	i	105	39
1577664178	58	-2	g	103	-2
1577664239	61	3	D	68	-35
1577664302	63	2	a	97	29
1577664361	59	-4	t	116	19
1577664425	64	5	a	97	-19

Tabulka 3.1: Příklad výpočtu komprimovaných hodnot Delta encoding pro pravidelné časové značky a řetězec znaků.

3.4.2 XOR encoding

Teoretické informace této podsekce pochází z [19]. Metodu využívá časová databáze Gorilla (viz 4.4.2) pro kompresi desetinných čísel s dvojitou přesností. XOR encoding funguje na podobném principu delta komprese - ukládá rozdíl sousedních hodnot. Základem metody je operace $\oplus$ (XOR) mezi dvěma po sobě jdoucími čísly. Hodnoty jsou ukládány podle následujících pravidel:

1. první hodnota je uložena bez komprese,
2. pokud je výsledek operace $\oplus$ 0 (hodnoty jsou totožné), uloží se pouze jeden nulový bit,
3. pokud je výsledek operace $\oplus$ nenulový (hodnoty se liší) a zároveň počet nulových bitů na začátku a konci výsledku **je** stejný nebo vyšší než u předchozího výsledku, jako první se uloží kontrolní bity 10 následované nenulovou částí výsledku operace,
4. pokud je výsledek operace $\oplus$ nenulový (hodnoty se liší) a zároveň počet nulových bitů na začátku a konci výsledku **není** stejný nebo vyšší než u předchozího výsledku, jako první se uloží kontrolní bity 11, následuje 5 bitů obsahující počet počátečních nul, 6 bitů obsahující délku nenulových bitů a samotné nenulové bity výsledku.

Příklad reprezentace všech tří možností uložení v paměti je možné vidět v tabulce 3.2. První hodnota v paměti je opsána podle pravidla číslo jedna. U druhé hodnoty nedošlo ke změně, a proto je uložen pouze jeden bit prefixu (prefixy jsou označeny červeně) obsahující nulu podle druhého pravidla. U třetí hodnoty došlo ke změně, a jelikož je počet nul před a po nenulových bitech nižší (předchozí hodnota má všechny bity nulové), je použito čtvrté pravidlo. První dva bity určují prefix, následujících 5 bitů vyznačených modře označuje počet nulových bitů před hodnotou, 6 bitů označených zeleně obsahuje délku výsledku a poslední je samotný výsledek operace. Čtvrté číslo je uloženo v paměti pomocí třetího pravidla, protože počet nul před výsledkem je totožný jako u předchozího a počet nul za výsledkem je o jednu vyšší. První dva bity opět označují prefix a zbytek obsahuje výsledek.

dec	hex	⊕ s předchozí	uloženo v paměti	velikost [b]
19.0625	0x4033100000000000	-	0x4033100000000000	64
19.0625	0x4033100000000000	0x0	0*b*0	1
41.25	0x4044A00000000000	0x0077B00000000000	0b110100100101011101111011	24
25.5	0x4039800000000000	0x007D200000000000	0b1011111010010	13

Tabulka 3.2: Příklad výpočtu komprimovaných hodnot pro XOR encoding.

Hlavní výhodou metody je nutnost pouze jednoho bitu k uložení nezměněné hodnoty a při menších změnách dosahuje několikanásobné komprese. Metoda je tedy vhodná například pro data pořízená z měření teploty, u kterých nedochází k velkým výkyvům.

3.5 Síťová komunikace a zabezpečení

Mnoho zařízení pro monitorování vzniklých před nebo v počátcích čtvrté průmyslové revoluce disponuje proprietárními komunikačními protokoly. Tyto protokoly jsou neveřejné a výrazně omezují větší změny v systému, jako je nákup zařízení od jiných výrobců či napojení na novější systémy. Postupem času začaly vznikat otevřené standardy jako je MODBUS TCP či EtherCAT pro lokální komunikaci. Upuštění od proprietárních komunikačních protokolů tedy značně usnadňuje vytvoření a propojení systémů IIoT (Industrial internet of things) od různých výrobců do jednoho celku.

MODBUS je bitově orientovaný asynchronní komunikační protokol využívající TCP, určený pro výměnu informací mezi vestavěnými systémy a průmyslovými aplikacemi. Protokol využívá takzvaný *pooling* neboli aktivní dotazování metodou dotaz-odpověď. Pomocí dotazu si lze vyžádat data či provést příkaz. Odpověď obsahuje potvrzení provedení vyžadovaného příkazu či vyžádaná data. Další protokol, který se využívá, je MQTT. Jedná se opět o TCP protokol, je oproti protokolu MODBUS synchronní a využívá model *publisher–subscriber* (vydavatel-odběratel). O výměnu zpráv se stará jeden centrální bod zvaný MQTT broker. Ten zprávy přijímá od vydavatelů, třídí do takzvaných témat a rozesílá všem odběratelům, kteří jsou k danému tématu přihlášeni [14].

Standardizované protokoly nejsou vhodné pro každé využití. Jsou určeny spíše pro informování o změně stavu v lokální síti a ne pro zasílání velkého množství dat pro zpracování přes internet. Protokoly vyžadují výměnu mnoha zpráv, takzvaných *handshake* zpráv, před zasláním samotné zprávy s informacemi, a nejsou zabezpečeny. Pro zasílání většího množství zpráv přes internet je vhodnější využít vlastních zjednodušených a zabezpečených UDP či TCP protokolů. Komunikaci přes TCP lze zabezpečit pomocí SSL (Secure Sockets Layer)

certifikátu, který vytváří šifrované spojení. UDP pakety lze zabezpečit například pomocí DTLS (Datagram Transport Layer Security) či jednoduchou bitovou operací XOR.

3.6 Vizualizace a vyhodnocení velkých dat ze senzorů

Vizualizace dat ze senzorů při využití existujících řešení často úzce souvisí se zvoleným úložištěm. Téměř každé úložiště velkých dat je možné propojit s nějakou vizualizační aplikací. Jednou z nejrozšířenějších vizualizačních aplikací je Grafana a slouží i k analýze dat. Grafana je volně dostupná a jedná se o aplikaci s webovým rozhraním, schopnou pracovat s daty z několika různých databází. Grafana podporuje mimo jiné i časové databáze, jako je InfluxDB či Graphite [11]. Vzhled aplikace je na obrázku 3.2.

Časová databáze InfluxDB, která je součástí ekosystému TICK, nabízí i vlastní grafické rozhraní Chronograf včetně analytického nástroje Kapacitor. Chronograf zobrazený na obrázku 7.1 nabízí podobné uživatelské rozhraní jako Grafana. V těchto rozhraních je možné využívat veškeré databázové dotazy, včetně importování či exportování hodnot. Lze vytvářet nástěnky s uloženými dotazy, sledovat hodnoty přicházející do databáze v reálném čase či sledovat vytížení samotné databáze. Analyzační nástroj Kapacitor oproti tomu slouží k detekci nastavených událostí, jako je například překročení limitu [25]. Podobné nástroje jsou dostupné pro téměř každé úložiště velkých dat v rámci jejich ekosystémů.

Obrázek 3.2: Webové rozhraní aplikace Grafana [11].

Kapitola 4

Úložiště velkých dat

S přibývajícím množstvím moderních technologií a odvětví (například IoT – Internet of Things) vznikají nové problémy s množstvím dat, které generují. Ať už jde o záznamy o uživatelích sociálních sítí, souřadnice polohy osob či vozidel sbíraných chytrými telefony, historii nákupů na internetovém obchodě nebo měření ze senzorů, je potřeba tyto informace uchovávat a zároveň musí být rychle a snadno dostupné pro zpracování. Jelikož se může jednat o obrovské množství dat, nároky na tyto systémy jsou velmi vysoké. Pro představu o jaké množství dat může jít si vezměme například službu Timeline od společnosti Google. Při povolení této služby chytrý mobilní telefon snímá polohu přístroje a vlastník připojeného účtu má zpětně přehled o svém pohybu a činnostech. Během dne může být takto vygenerováno od pár desítek po několik tisíc záznamů, které je potřeba uložit a v případě potřeby zpřístupnit. Samo o sobě se nejedná o velké množství dat, ale když vezmeme v úvahu množství lidí využívající chytrý telefon, mluvíme zde o sta milionech záznamů denně.

K ukládání takového objemu dat se využívají mimo jiné distribuované souborové systémy, cloudové nástroje či specializované databázové systémy. Tato kapitola je věnována různým typům úložišť pro tato data a je zaměřena na řešení určené pro ukládání hodnot získaných během měření různých senzorů. U tohoto typu dat je nejdůležitějším údajem každého záznamu kombinace času měření a naměřené hodnoty. Kapitola má tedy za cíl seznámit čtenáře s technologiemi, které se často využívají pro ukládání velkých dat a s vysvětlením jejich principů.

4.1 NoSQL

Jedná se o nerelační datový model podporující distribuovanou architekturu a umožňující zpracování velkého množství dat - až petabajty dat za den. Oproti relačnímu modelu obětuje přílišnou komplexnost způsobující striktní zajištění pravidel ACID (Atomicity, Consistency, Isolation, Durability) tzn. atomičnost transakcí, konzistenci dat, neovlivnění souběžných transakcí a ochrana proti ztrátě transakcí. Tato ztráta komplexnosti sice umožňuje inkonzistenci a nepřesnost, to ale kompenzuje vysokou propustností a škálovatelností. Tento model je proto vhodný pro data, u kterých nejsou drobné nepřesnosti problematické a je potřeba jich zpracovávat obrovské množství, jako například u sledování dat ze senzorů či monitorování sítě a podobně. Tento model je proto naprosto nevhodný k uchování dat vyžadující konzistenci (například bankovní transakce), kde i ztráta jednoho záznamu může znamenat katastrofu. Ztráta záznamu ale není velkým problémem třeba u monitorování

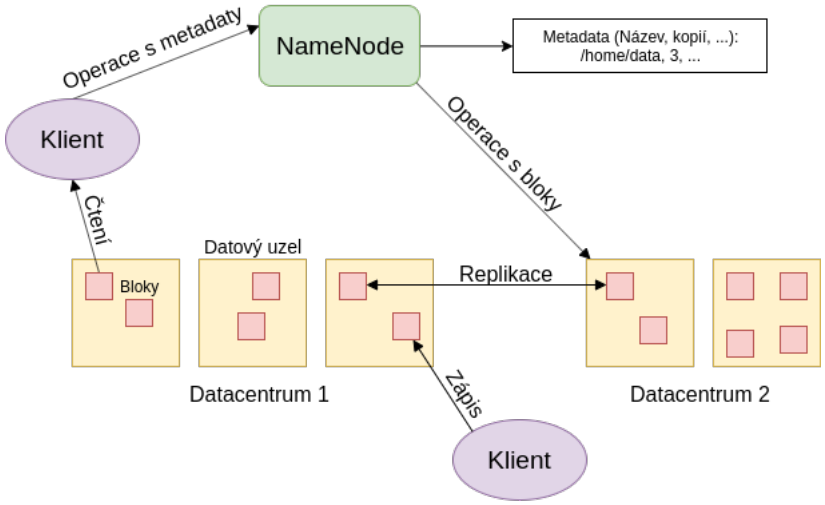
síťového vytížení, kde se stejně agreguje obrovské množství hodnot a nedojde proto k velkému ovlivnění výsledku. Možnost nezajistit konzistenci proto dovoluje právě povaha dat, pro která jsou úložiště využívající tento model určena [24].

4.2 Hadoop Distributed File System

Hadoop Distributed File System (dále jen HDFS) je součástí volně dostupného projektu Apache Hadoop. Jedná se o distribuovaný souborový systém poskytující vysokou propustnost dat a toleranci poruch. V této sekci a jejích podsekcích byly informace čerpány z dokumentace projektu HDFS[6].

4.2.1 Architektura HDFS

HDFS využívá architekturu master/slave a je distribuovaný, tudíž využívá více datových uzlů. Samotné blokové schéma systému je zobrazena na obrázku 4.1. Datové uzly se starají o samotné úložiště. Většinou je umístěn právě jeden datový uzel na fyzickém serveru. Umístění více instancí uzlu na jeden fyzický server nepřidává žádné výhody. HDFS umožňuje uživateli ukládat data do souborů, které jsou následně rozděleny na bloky a ty jsou distribuovány mezi několik datových uzlů. Datové uzly se také starají o veškeré operace s bloky - vytváření, mazání, replikace, čtení a zápis.



Obrázek 4.1: Schéma architektury HDFS [6].

Všechny bloky souboru mají stejnou velikost - vyjma posledního. Jednotlivé bloky jsou replikovány napříč uzly a tím se dosahuje tolerance poruch uzlů. Velikost bloků a počty jejich kopií (replikační faktor) jsou nastavitelné pro každý soubor zvlášť a tyto hodnoty jsou udržované v metadatech. Soubor v HDFS lze zapsat pouze jednou a jedním uživatelem.

Datové uzly spravuje *NameNode*. Jedná se o hlavní server, který se stará o hierarchii souborového systému, metadata a upravuje přístup klientům. Jakoukoliv změnu umístění či vlastností (například replikační faktor, název, id) zaznamenává do logu a mění v metadatech. *NameNode* se stará o veškerá rozhodnutí týkající se replikace bloků. Od každého datového uzlu periodicky dostává upozornění indikující správnou funkčnost (takzvaný *Heartbeat*) a seznam jednotlivých uložených bloků.

4.2.2 HBase

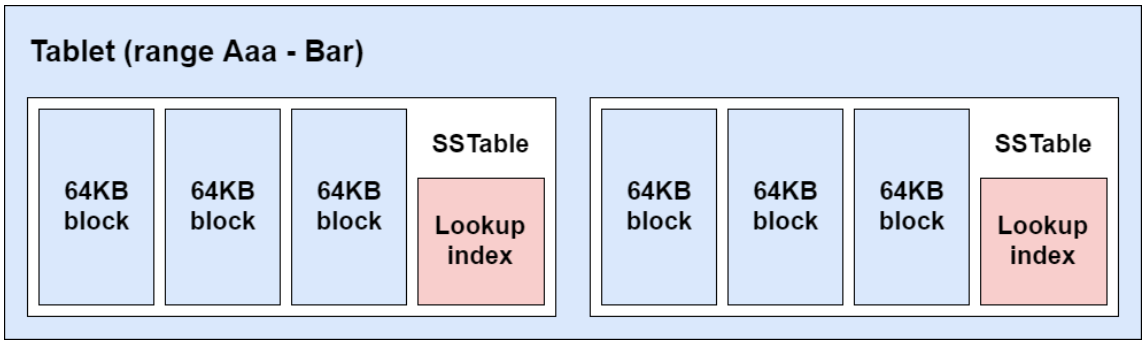
HBase je taktéž součást Apache Hadoop a jedná se o sloupcově orientovaný nerelační databázový systém operující nad HDFS (viz 4.2). HBase je nástroj vhodný pro zpracovávání dat v reálném čase, náhodný přístup k velkému množství dat a je lineárně škálovatelný. Nepodporuje strukturované dotazy jako SQL. Systém lze lineárně škálovat. Každá tabulka musí obsahovat primární klíč a k uloženým datům lze přistupovat pouze přes tento primární klíč [4].

4.3 Google BigTable

Jedná se o distribuovaný systém využívaný pro ukládání dat a je navržen tak, aby byl schopný pracovat s ohromným množstvím dat na velkém počtu serverů. V této sekci čerpám informace z [8].

BigTable je multidimenzionální tříděná mapa. Základní jednotkou je netypovaný řetězec znaků, který je interně uložený v blocích ve tříděných tabulkách řetězců znaků zvaných SSTable. Tyto bloky jsou dostatečně malé na to, aby je bylo možné nahrát do operační paměti (typicky 64 kB). Na konci každé tabulky je uložen index, který je nahrán do paměti při otevření tabulky. BigTable indexuje podle řetězce znaků, přesněji podle řádku, sloupce a verze (většinou se jedná o časovou značku). Záznamy mají lexikografické uspořádání. Tabulky jsou ukládány po blocích zvaných *tablet* podle rozsahu a pokud tento blok překročí určitou velikost, je rozdělen na dva tablety (bloky jsou typicky 100-200 MB velké). Schéma tabletu je možné vidět na obrázku číslo 4.2. Každý záznam obsahuje verzi, která je záznamům přidělována automaticky jako časová značka aktuálního času při zápisu, tím se předchází možným kolizím hodnot. Verzi lze přidělovat i z klientské aplikace, která BigTable využívá, ale potom je na klientské aplikaci, aby zajistila unikátnost záznamů. Verze záznamu je využívána pro mazání starých a neaktuálních hodnot. Klient může specifikovat kolik záznamů nebo jaký časový úsek do minulosti má být uchováván.

Samotný systém obsahuje dva typy serverů - master a tablet server. Master server je puštěn pouze v jedné instanci a stará se o rozdělení tabletů tablet serverům, detekci expirace tablet serverů, vyvažování zatížení, zálohy a změny schémat systému. Tablet servery se starají o tablety, zápis, čtení a rozdělení tabletů při překonání určité velikosti. Klientské aplikace komunikují přímo s tablet servery. Ke správě záloh se využívá služba zvaná *Chubby*. Pro zjištění polohy uložených dat slouží tříúrovňový B-strom, obsahující metadata o tabletech.



Obrázek 4.2: Schéma tabletu [2].

4.4 Time-series databáze

Časová databáze (lépe známé pod anglickým názvem *time-series database*) je speciálním typem databáze určené a optimalizované pro zaznamenávání měření obsahující časovou značku. Tyto databáze se využívají ve finančním sektoru (historie stavu akcií pro predikce jejich budoucího vývoje), telekomunikacích (informace o stavu a využití sítí), nebo při monitoringu datových center (monitorování teploty, využití procesorů, poruch) či průmyslových strojů.

Jak už tedy název napovídá, pracují s časovou značkou, která se vyskytuje u každého záznamu, a je vždy primárním klíčem nebo alespoň jeho součástí. Časová databáze typicky obsahuje datové body, které představují měření v daném čase. Oproti relačním databázím se téměř nepočítá s využitím funkcí pro aktualizaci (UPDATE) či mazání (DELETE) hodnot a dotazy (SELECT) jsou velmi specifické. Složité složené dotazy plné podmínek nejsou časté pro data tohoto typu a nejčastějším typem dotazů je získání určitého časového okna měřených dat, a to co nejefektivněji. Právě na tyto dotazy jsou časové databáze optimalizovány[16].

Časové databáze jsou obvykle postaveny nad distribuovanými souborovými systémy, jako jsou HDFS či Bigtable popsané dříve v této kapitole, a s jejich pomocí dosahují vysoké rychlosti čtení a zápisu hodnot a škálovatelnosti systému.

Kromě času obsahují časové databáze jako OpenTSDB nebo InfluxDB dva další typy hodnot - značky (*tag*) a datová políčka (*field*). Databáze následně zajišťuje unikátnost záznamu pomocí primárního klíče složeného z kombinace času, značek a měřené hodnoty. Záznamy jsou následně ukládány v balících podle značek a v těchto balících jsou seřazeny podle času. Tento přístup zrychluje možnost agregace uložených hodnot [9].

4.4.1 Řádkově/sloupcově orientované ukládání

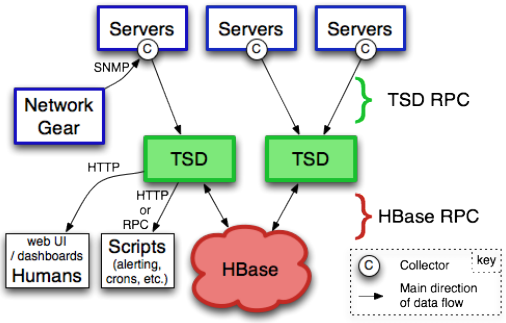
Databáze využívají k ukládání dat dva způsoby - jsou sloupcově nebo řádkově orientované. Řádkově orientovaná metoda je používaná relačními databázemi. Ukládá a získává data po řádcích. Tento způsob uložení není efektivní pro agregaci a analýzu většího množství hodnot a dosahuje horší úrovně komprese dat než ukládání po sloupcích. Čtení a zápis je ale snazší. Sloupcově orientované databáze oproti tomu ukládají a získávají data po sloupcích. Příkladem je HBase, která je základem některých časových databází. Touto metodou lze dosáhnout rychlé agregace velkého množstvím hodnot a umožňuje vysokou úroveň komprese díky podobnosti hodnot, které jsou za sebou uložené. Přestože sloupcově orientované databáze dominují úložištím velkých dat, lze nalézt i databáze které kombinují oba způsoby, jako je TimescaleDB [15].

4.4.2 Zástupci

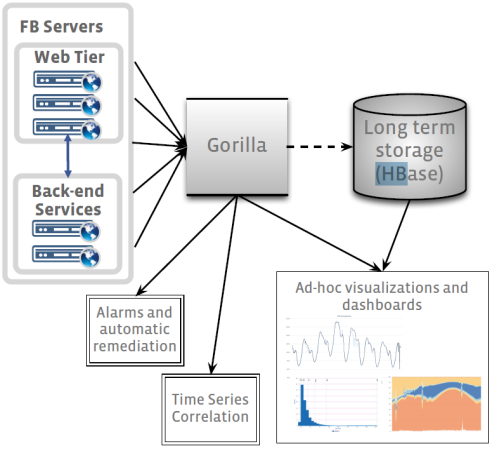
S rostoucí potřebou zpracování a uložení dat z měření se v posledních letech rozšířil počet různých časových databází. Nejedná se již jen o obecné implementace jak tomu bylo dříve, ale i o databáze určené ke specifickým účelům, na míru vyvinutým potřebám firem, jako je Facebook či Google. Nejznámější a nejpoužívanější time-series databáze jsou následující:

OpenTSDB

Jedná se o škálovatelnou volně dostupnou časovou databázi pracující nad HBase (popsanou v 4.2.2) či Google Bigtable (popsaný v sekci 4.3). OpenTSDB nabízí rozhraní pro zápis a



Obrázek 4.3: Architektura OpenTSDB [27].



Obrázek 4.4: Schéma využití databáze Gorilla [19].

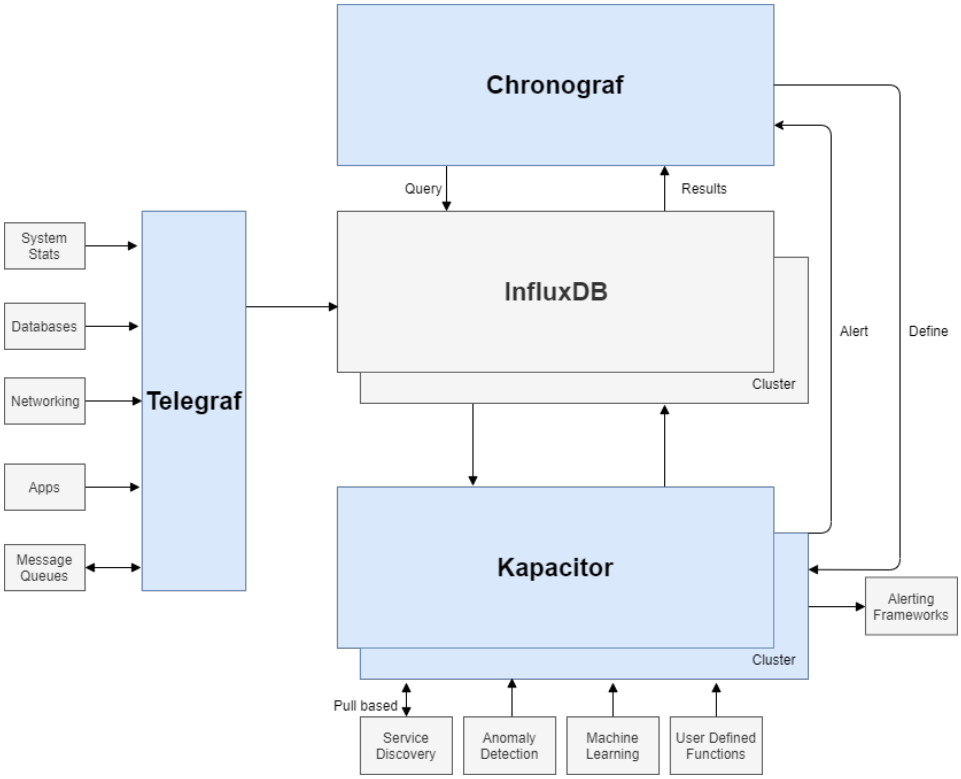
čtení a stará se o veškeré operace související se správou HBase. Data lze získat v grafické formě či ve formě souboru ve formátu json.

Jak je vidět na obrázku 4.3, OpenTSDB je vlastně několik samostatných TSD instancí, které zastřešují přístup k úložišti a starají se o veškerou komunikaci. Uživatel tedy přistupuje k datům pouze s pomocí poskytnutého rozhraní. Databáze je optimalizovaná pro rychlou agregaci hodnot. Čas je uložen ve formě unixové časové značky a hodnoty jsou 64bitová celá či desetinná čísla. Přístup k datům je možný pomocí vestavěného grafického rozhraní či přes HTTP rozhraní [27].

InfluxDB

Informace o InfluxDB byly čerpány z [25]. Databáze InfluxDB byla vytvořena již od základu jako časová a je k dispozici volně dostupná i v komerční verzi. Jedná se o NoSQL databázi implementovanou v jazyce Go. Komerční verze nabízí dodatečnou funkcionalitu nutnou pro nasazení ve větším měřítku - například možnost využití instancí na více serverech či pokročilé monitorování. Jedná se o nejpopulárnější časovou databázi. InfluxDB se skládá z databází, které obsahují jednotlivé hodnoty uspořádané podle měření (*Measurement*). Měření se dá přirovnat k tabulkám. Záznam obsahuje název měření, set tagů podle kterých se indexuje, set hodnot podle kterých se neindexuje a časovou značku v přesnosti na nanosekundy. Počet tagů není omezený na rozdíl od OpenTSDB. InfluxDB poskytuje jazyk pro dotazy podobný SQL dotazům. Pro každé měření lze nastavit jak dlouho a v kolika kopiích mají být hodnoty uchovány. InfluxDB využívá svůj vlastní úložný engine zvaný TSMT (Time Structured Merge Tree). Databáze ukládá data po blocích zvaných *shard*, které jsou organizovány podle nastavení životnosti dat do časových úseků. Základní časový úsek pro životnost dat kratší než dva dny je jedna hodina, a pro životnost delší než 6 měsíců je časový úsek jednoho bloku 7 dní.

K datům lze přistupovat přes HTTP a UDP komunikační rozhraní, přes konzoli podobně jako u MySQL či přes webovou aplikaci Chronograf. InfluxDB a Chronograf jsou součástí platformy TICK. Součástí této platformy je i modul Telegraf, který slouží k monitorování serverů, na kterých je spuštěn InfluxDB, a engine pro zpracovávání dat v reálném čase



Obrázek 4.5: Schéma platformy Tick [25].

Kapacitor. Na obrázku číslo 4.5 je znázorněno schéma platformy TICK s databází InfluxDB ve svém jádru.

Gorilla

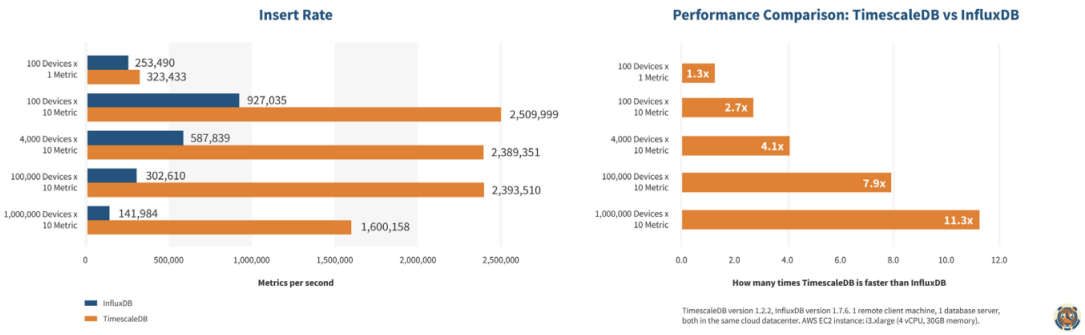
Technické informace této části pochází z [19]. Gorilla je *in-memory* databáze (data jsou uložena v operační paměti) vytvořená firmou Facebook, která slouží pro monitorování dat zapisovaných do HBase databáze. Vizualizované využití této databáze je možné vidět na obrázku 4.4. Bod uložený v databázi obsahuje klíč ve formě posloupnosti znaků, 64bitové časové značky a desetinného čísla obsahujícího hodnotu. Jelikož se jedná o *in-memory* databázi, která funguje jako mezipaměť pro dlouhodobější uchování dat na discích, jsou hodnoty udržovány po omezenou dobu. Množství bodů je ale natolik velké, že požadavky na paměť by byly v řádech desítek terabajtů, proto databáze využívá kompresních algoritmů popsaných v 3.4.1 a 3.4.2. Tento algoritmus umožňuje průměrnou kompresi 16 bajtů na 1,37 bajtu. Gorilla se také od ostatních časových databází odlišuje svou úzkou specializací na monitorování systémů v reálném čase. Databáze má za cíl co nejrychleji odhalovat chyby v interních systémech. Databáze musí být schopná bez výpadku přijímat a zapisovat velké množství bodů i za cenu dočasné nedostupnosti pro čtení nebo ztráty starších bodů. Proto není kladen důraz na jednotlivé body, ale na agregované analýzy. Gorilla je distribuovaná mezi několik regionů bez zajištění konzistence a dotazy jsou směrovány na nejbližší datová centra. To umožňuje tolerovat výpadek celého regionu.

Hodnoty jsou uloženy ve formě *TSMap*. Ta je složena z vektorů sdílených ukazatelů na časové série a mapy obsahující dvojice název série a ukazatel na její umístění v paměti. Při smazání série nejsou data smazána, ale paměť je pouze označena pro znovupoužití. Výlučný

přístup k datům zajišťuje zámek na *TSmap* a 1-bajtový zámek s aktivním čekáním na každé časové sérii.

TimescaleDB

Zdrojem informací o architektuře TimescaleDB je [28]. Oproti výše zmíněným databázím je TimescaleDB založená na relačním databázovém modelu využívající PostgreSQL. TimescaleDB je zaměřená na lepší paralelní zpracování a škálování nejen na více uzlů, ale i na jednom samotném uzlu. Je ale nutno zmínit, že v době psaní této práce bylo škálování na více uzlů stále ve fázi vývoje. Oproti databázím, jako je InfluxDB, nerozděluje data do bloků o jedné dimenzi (v případě zmíněné databáze InfluxDB podle času), které lze následně ukládat na různých uzlech, ale do bloků o více dimenzích, ke kterým uživatel přistupuje přes struktury zvané *hypertable*. Data jsou rozdělena primárně podle času a dále podle klíče, kterým může být například identifikátor. Právě toto rozdělení umožňuje škálovat databázi i na jednom uzlu. *Hypertable* je abstrakce tabulky, jak je známá například z MySQL. Dotazy či jiné interakce s databází se provádí nad těmito tabulkami. Jedná se tedy virtuálně o standardní schéma obsahující sloupce. Minimálně jeden sloupec musí obsahovat časovou značku. Další sloupce obsahující klíče, podle kterých se provádí rozdělení dat, jsou volitelné. Jednotlivé bloky jsou implementovány pomocí PostgreSQL tabulek a obsahují určitý časový a prostorový interval, který závisí na schématu. Data se nepřekrývají a to snižuje počet přístupů k jednotlivým fyzickým tabulkám. Data jsou v databázi uložena jak v řádkovém, tak ve sloupcovém formátu. Jakmile jsou data dostatečně stará, jsou komprimovaná z řádkového formátu do sloupcového. Tato architektura umožňuje TimescaleDB mnohem vyšší propustnost bodů pro zápis na jednom uzlu, než konkurenční InfluxDB. Porovnání mezi InfluxDB a TimescaleDB v počtu zapsaných bodů na jednom uzlu za vteřinu je možné vidět na obrázku 4.6.



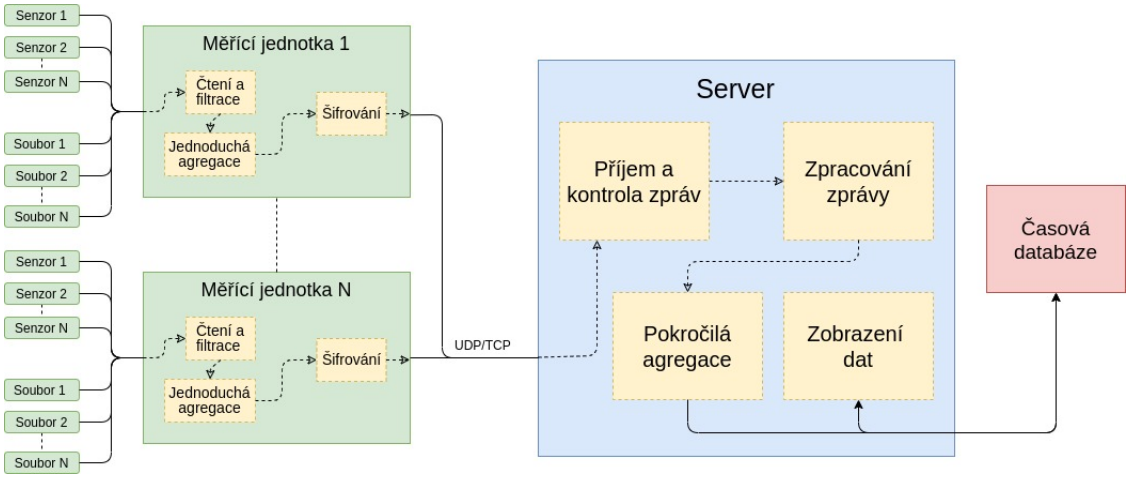
Obrázek 4.6: Porovnání InfluxDB a TimescaleDB v počtu vložených bodů za vteřinu na jednom uzlu [29].

Kapitola 5

Návrh systému

V této kapitole je popsán návrh systému pro sběr, zpracování, vyhodnocení a zobrazení dat z různých senzorů umístěných na průmyslových strojích.

Hlavními požadavky na systém je jeho snadná škálovatelnost, rychlost a modularita. Požadavky vychází ze zkušeností partnerské firmy s jejich vlastním systémem. Škálovatelností je myšlena možnost rychlého nasazení nových jednotek se senzory a možnost jednoduchého rozšíření výpočetní síly serverové části, aby byla schopná efektivně zpracovávat data od různého množství jednotek. Modularitou systému je míněna možnost snadné výměny jednotlivých částí bez hlubších zásahů do programové části. Například změna úložiště dat, možnosti nasazení různých implementací jednotek, změna komunikačních protokolů a podobně. Systém by také měl být schopný zpracovávat data v reálném čase a dostupnost uložených dat by měla být okamžitá.



Obrázek 5.1: Zjednodušené schéma systému pro sběr, zpracování, vyhodnocení a zobrazení dat.

Na obrázku 5.1 je zobrazeno jednoduché schéma navrženého systému. Systém obsahuje měřící jednotky, které čtou data o pevně nastavené vzorkovací frekvenci z připojených senzorů, nebo v případě testování ze souborů. Měřící jednotka kromě čtení provádí filtrování vstupních dat (například snížení vzorkovací frekvence). Vyfiltrovaná data jsou následně agregována jednoduchými, rychlými algoritmy. Po dokončení agregace jsou data zašifrována a pomocí protokolu UDP odesílána na server. Měřící jednotka je též schopná posílat na server neupravené bloky hodnot obsahující důležité úseky měření pro pokročilejší zpracování

dat na serveru. Tato data jsou odesílána přes protokol TCP. Data agregovaná na jednotce slouží pro detekci fatálních poruch a sledování vytížení strojů. Tyto hodnoty mohou být využity pro spouštění odesílání nezpracovaných dat. Neagregovaná data zpracovávána serverovou částí neslouží k detekci poruch, ale jejich predikci. Partnerská firma k vypracování této práce poskytla software pro zpracování těchto hodnot a prototyp zařízení pro sběr dat ze senzorů vibrací a teploty. Návrh programové části pro měřící jednotku je ovlivněn vlastnostmi tohoto prototypu.

Zprávy přijímá serverová část. O příjem se starají dva programy – jeden pro zpracování UDP zpráv obsahující filtrovaná, agregovaná data a druhý pro příjem TCP zpráv s neupravenými hodnotami. Programy zprávy dešifrují a provedou jejich kontrolu. Bloky neupravených dat jsou ukládány ve formátu TDMS[1] na úložiště pro pokročilé zpracování. Soubory slouží i jako záloha. Více informací k formátu TDMS a účelu těchto dat se nalézá v sekci 5.3.2. Hodnoty z UDP zpráv, které jsou již předzpracované procházejí pokročilou agregací (pokud je to nutné). Následně jsou zapsány do časové databáze. Pokud tyto hodnoty překročí určitou hodnotu je uživatel či správce notifikován.

5.1 Měřící jednotka

Měřící jednotka partnerské firmy na které byl vyvíjen software pro sběr a zpracování dat se skládá z mini počítače Raspberry Pi verze 3b (dále jen RPi) a rozšiřující desky. Partnerskou firmou vyvinutá deska je k RPi připojena přes GPIO (General-Purpose Input/Output) piny. Deska obsahuje ve starší verzi čtyři vstupy pro senzory vibrací a v novější verzi i tři konektory pro snímání teploty. Návrh softwaru pro jednotku, jehož schéma je možné vidět na obrázku 5.2, je určen právě pro počítač RPi a je ovlivněn jeho parametry. Návrh také počítá s připojením starší verze desky s možností připojení čtyř senzorů vibrací, protože novější verze desky nebyla k dispozici pro testování. Měřící jednotka běží na operačním systému Raspbian, který je linuxovou distribucí speciálně upravenou pro RPi.

RPi bylo zvoleno z důvodu snadné dostupnosti, nízké ceny, výkonu a hlavně kvůli jeho rozšířenosti a velké komunitě. Pro RPi a linuxovou distribuci Raspbian existuje mnoho knihoven, které usnadňují implementaci komunikace s připojenými periferiemi přes výstupní piny.

5.1.1 Čtení a filtrování dat

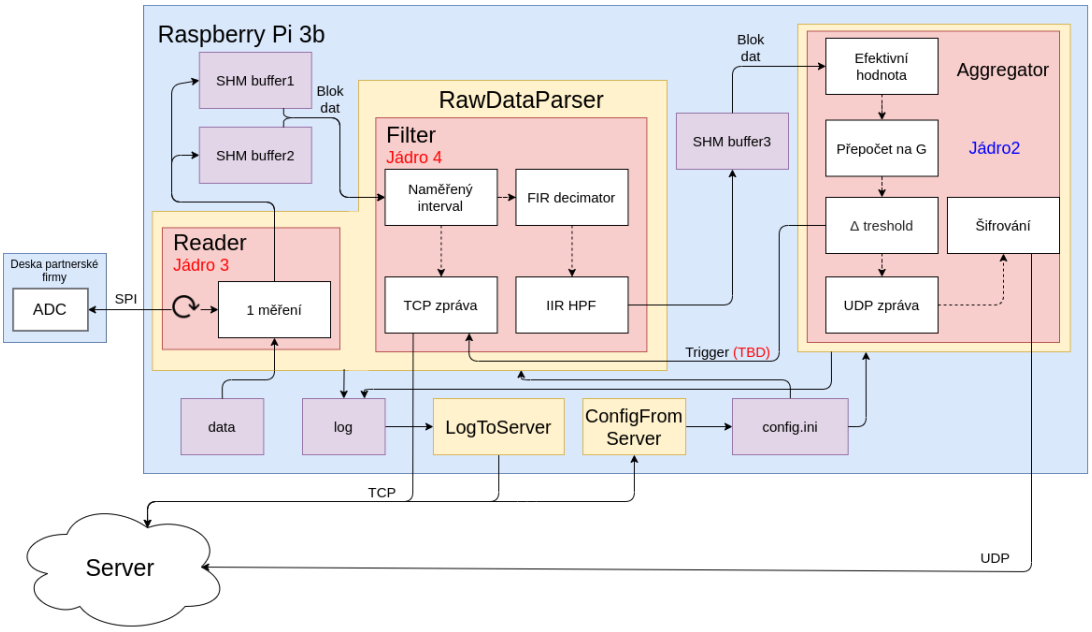
Hlavní částí návrhu jednotky je program *RawDataParser*, který se skládá ze dvou částí – procesu pro čtení dat (*Reader*) a procesu pro filtrování dat (*Filter*). Hodnoty ze senzorů vibrací poskytuje analogově digitální převodník ADS131A04 od výrobce Texas Instruments. AD převodník je umístěný na rozšiřující desce a s RPi komunikuje přes sériové periferní rozhraní (dále jen SPI). Proto program pro čtení dat podporuje komunikaci přes SPI, a z důvodu testování má i podporu čtení dat ze souboru. AD převodník na desce vzorkuje data ze senzorů o frekvenci až 128 kHz a proto software *Reader* teoreticky zvládá číst až 512 000 hodnot za vteřinu (128 000 hodnot za vteřinu ze čtyř senzorů). Velikost jedné hodnoty je 24 b. Rozšiřující deska notifikuje RPi o připravenost dat pro čtení pomocí sestupné hrany signálu pojmenovaném *data ready* na nastaveném pinu. Čtení ze souboru nabízí stejnou možnost získávání dat o nastavené vzorkovací frekvenci jako přes SPI.

Přečtená data jsou po časových úsecích zapisována do bloků sdílené paměti, ze které je čte filtrovací proces. Bloky jsou dva a vždy dochází k zápisu do jednoho z nich a čtení

[1] tdms – Technical Data Management Streaming File

z druhého. Takto navržené sdílení dat umožní, aby čtecí proces nečekal na uvolnění bloku sdílené paměti a tím nedocházelo ke ztrátě vzorků. Jakmile je časový úsek přečten filtrovacím procesem dojde ke snížení vzorkovací frekvence pomocí filtrů a takto zpracovaná data jsou umístěna do třetího bloku sdílené paměti pro agregační program. Neupravený časový úsek o původní vzorkovací frekvenci je v případě potřeby odesílána pomocí protokolu TCP na server pro účely výpočetně náročnějšího zpracování.

Jednotlivé procesy programu *RawDataParser* jsou umístěny na izolovaných jádrech procesoru. Toto rozhodnutí bylo učiněno z důvodu zvolené platformy a operačního systému. Nejedná se totiž o *real-time* operační systém. Z toho důvodu není možné zajistit běh programu v reálném čase, pokud jsou programy umístěny na jádrech s více puštěnými procesy. Více k této problematice se nachází v 6.2.2.



Obrázek 5.2: Schéma softwaru měřící jednotky.

5.1.2 Agregace dat

Jak již bylo zmíněno na začátku kapitoly, data agregovaná jednotkou slouží pro zobrazení vytížení stroje a případně i jako indikátor pro odesílání nezpracovaných dat na server. Návrh proto počítá s triggerem, který upozorňuje filtrovací proces při překročení určité hodnoty. Tato hranice může být nastavena například na zapnutí stroje. Filtrovací proces po upozornění začne odesílat nezpracované bloky dat na server pomocí komunikačního protokolu přes TCP. Funkcionalita triggeru není kritická, ale slouží ke snížení síťové komunikace a potřebné výpočetní síly na serveru. V případě kdy je sledovaný stroj vypnutý, nebude docházet ke zbytečnému odesílání dat k pokročilému zpracovávání. Tato funkce ale může být překážkou pro stabilní zpracovávání dat není proto součástí výsledného systému. Nezpracovaná data jsou tedy vždy odesílána na server v pravidelných intervalech nehledě na stav stroje.

Cílem agregačního programu je redukce dat se sníženou vzorkovací frekvencí předávaných přes sdílenou paměť programem pro čtení a filtrování na zhruba 1-100 vzorků za vteřinu. Toto rozpětí bylo vybráno proto, aby bylo možné zachovat nejdůležitější informace při velkých změnách v měření. Díky tomu by při běžné činnosti nemělo docházet k pří-

lišnému zatěžování serverové strany obrovským množstvím hodnot. Za zvážení také stojí omezení zasílání agregovaných dat pokud je stroj nečinný. Neodesílání dat ve vypnutém stavu ale může zamezit sledování stroje v reálném čase, protože aktualizace hodnot probíhá po delším časovém intervalu.

V prvním kroku jsou data redukována na 100 vzorků za vteřinu. Tato redukce probíhá pomocí výpočtu kvadratického průměru popsaného v sekci 3.3. V dalším kroku dochází k přepočtu hodnoty na zrychlení g podle vzorce:

$$g = \frac{x_{out} - x_{rest}}{sensitivity} \tag{5.1}$$

kde x_{out} je výstupní hodnota v mV, x_{rest} je klidová hodnota akcelerometru v mV a $sensitivity$ je citlivost senzoru v mV/g. Citlivost je dohledatelná v technické dokumentaci senzorů [26]. Po přepočtu na g dochází v dalším kroku k zahození hodnot pomocí prahovací funkce hodnoty a času. Ponechány jsou pouze hodnoty, které se liší o nastavený rozdíl či došlo k překonání určeného časového intervalu od poslední uložené hodnoty. Po dokončení agregace je z hodnot poskládána UDP zpráva. Zpráva je zašifrována a odeslána na server. Tato implementace zpracovává data ze senzorů vibrací. Konfigurační soubor ovšem obsahuje položku typ senzoru a v případě připojení jiných typů senzorů je možné snadno doimplementovat jiný typ přepočtu na fyzikální veličiny či agregace.

5.1.3 Další programy a funkce

Nejdůležitějším programem jednotky je *RawDataParser* věnující se zpracováním dat. Kromě tohoto programu ovšem jednotka obsahuje i jiné podpůrné programy, skripty a soubory.

Instalační skripty

V rámci zjednodušení nasazení velkého počtu jednotek za co nejkratší možnou dobu je součástí návrhu sada instalačních skriptů, která automaticky při prvním zapojení inicializuje jednotku. Cílem sady skriptů je, aby pomocí spuštění jednoho skriptu na počítači s připojenou SD kartou určenou pro RPi došlo k vytvoření obrazu systému na kartě. Na kartu jsou nakopírovány všechny zbylé instalační skripty a potřebné soubory. Po vložení takto nachystané karty dojde ke kompletní instalaci a inicializaci všech částí softwaru jednotky, a to včetně určení unikátního ID, stáhnutí konfiguračního souboru ze serveru a spuštění samotného měření. Tato inicializace jednotky probíhá automatizovaně bez jakéhokoliv zásahu zvenčí.

Konfigurační soubor

Konfigurační soubor poskytuje možnost jednoduchého nastavení jednotky. Obsahuje veškeré důležité informace pro běh programů a to je:

- identifikátor jednotky obsahující unikátní sériové číslo procesoru RPi,
- zapnutí zápisu logů, cestu k souborům s logy, nastavení odesílání logů na server
- zdroj čtení, rychlost čtení, nastavení pinů, cesta k zálohám a podobně,
- parametry pro agregaci hodnot,
- nastavení senzorů, jejich piny, zapnutí, citlivost a typ,

- IP adresy a porty k serverům pro odesílání dat.

Konfigurační soubor je automaticky stahovaný ze serveru. Informace určené k časté změně jsou na serveru uložené v relační databázi jejíž návrh je popsán v podsekci 5.3.3. Upravovat pomocí databáze je možné nastavení agregace či parametry senzorů. Nastavení jako cesty k zálohám či IP adresy serverů je možné změnit jen v serverové aplikaci pro generování konfiguračního souboru. Jednotka periodicky zjišťuje, zda došlo ke změně konfiguračního souboru. Při změně dochází ke stažení nové konfigurace a restartu jednotky. Aktivní dotazování bylo zvoleno z toho důvodu, že může nastat situace, kdy jednotku není možné notifikovat z vnější sítě o změně parametrů. Soubor má formát .ini. Formát .ini se člení na sekce záznamů a každý záznam obsahuje dvojici klíč, hodnota. Příklad obsahu souboru vypadá takto:

```
;aggreagation parameters
[aggregation]
delta = 10.2
average = 80

#server parameters
[server]
udp_ip: 255.255.255.255
udp_port: 42
```

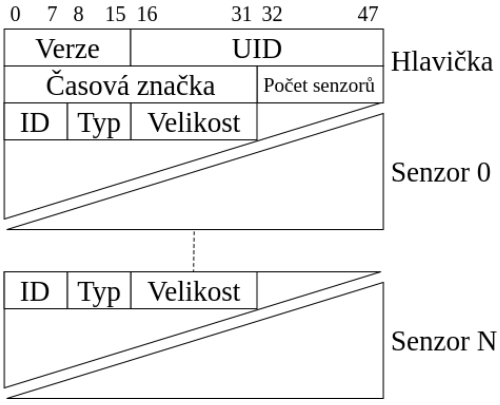
Logování

Chybové výstupy všech programů jsou ukládány do logovacích souborů. Logovací soubory jsou odesílány na server. Záznamy jsou ukládány v následujícím formátu: *[<časová značka>] [<název programu>] [<stav>] <zpráva>* Příkladem takového záznamu je: *[2020-01-29 14:58:21.227] [Reader] [error] Unable to parse config file: missing frequency value*

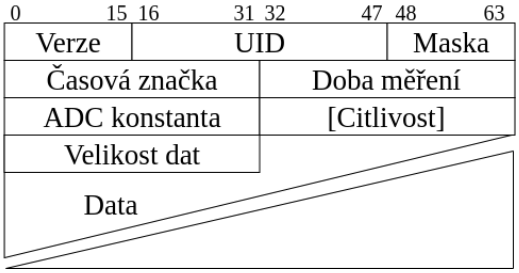
5.2 Komunikační protokoly

UDP protokol je určený ke sledování dat s co nejmenším možným zpožděním. To znamená časté zasílání menšího množství již agregovaných dat připravených k přímému zápisu do databáze. Ztráta paketu nemá vliv na systém. Komunikační protokol je zobrazen na obrázku číslo 5.3a. Protokol obsahuje verzi, unikátní identifikátor jednotky, časovou značku začátku měření a počet senzorů v datové části. Hlavička senzoru je dlouhá 32 bitů. ID senzoru číslované od 0 označuje index připojeného senzoru. Typ dat označuje jakým způsobem byla data zpracována a určuje do jaké databáze jsou vložena. Velikost označuje počet bajtů za hlavičkou senzoru obsahující hodnoty. Verze protokolu vyjadřuje v jakém formátu jsou hodnoty odesílány. Maximální délka paketu byla zvolena 512 bajtů, což je minimální délka **datového bloku** paketu, kterou musí umět každé síťové zařízení zpracovat [20].

Komunikační protokol založený na TCP slouží k odeslání časového úseku nezpracovaných dat na pokročilé zpracování. Paket obsahuje oproti UDP zprávě navíc dobu měření ve vteřinách, ADC konstantu pro přepočet hodnot a pole citlivostí jednotlivých senzorů obsažených ve zprávě.



(a) Schéma UDP paketu.



(b) Schéma TCP paketu.

Obrázek 5.3: Komunikační protokoly.

5.3 Návrh serveru

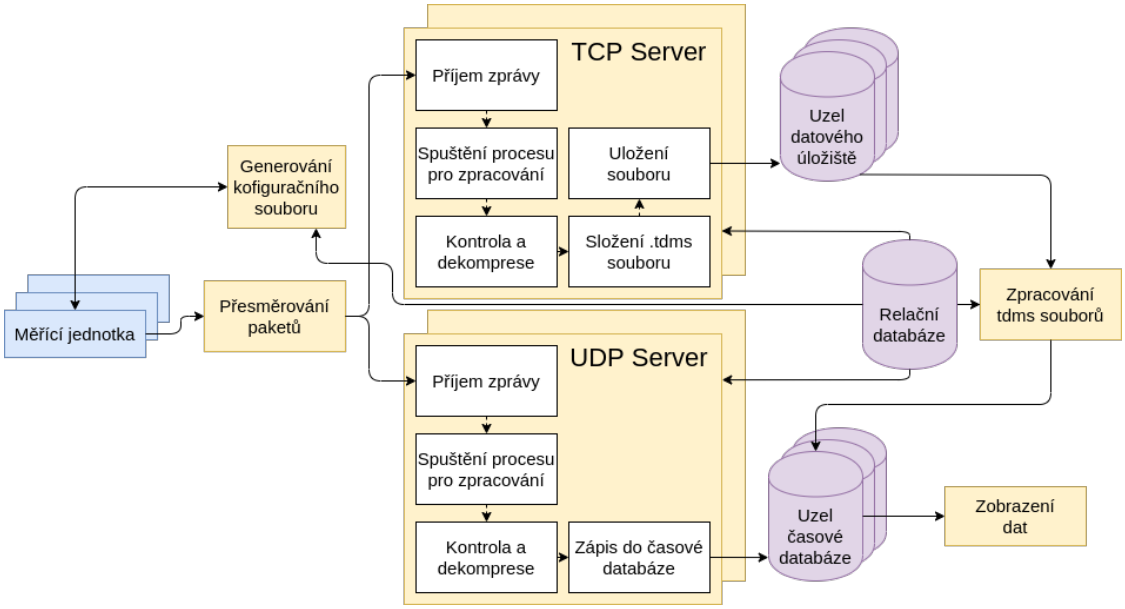
Serverová část systému je navržena tak, aby mohla fungovat pouze z jednoho fyzického zařízení. V případě potřeby je ovšem možné jednotlivé programy distribuovat na více serverů. Blokové schéma se nachází na obrázku 5.4. Návrh obsahuje:

- aplikace UDP a TCP serveru,
- relační databázi pro správu informací o jednotkách,
- časovou databázi pro ukládání zpracovaných naměřených hodnot,
- datové úložiště pro zálohování nezpracovaných úseků měření ve formě TDMS souborů
- algoritmy partnerské firmy pro pokročilé zpracování TDMS souborů
- webovou aplikaci pro zobrazení grafů z časové databáze
- program pro generování konfiguračních souborů z databáze
- program pro přesměrování datového toku mezi více serverů.

Jednotlivé části návrhu jsou detailněji popsány v následujících podsekcích.

5.3.1 UDP server

Aplikace slouží pro příjem UDP zpráv obsahující zpracovaná data určená k zápisu do časové databáze. Při příchodu zprávy dochází k načtení verze protokolu a UID jednotky. Po načtení



Obrázek 5.4: Schéma návrhu serverových aplikací a úložišť.

informací dochází k dešifrování obsahu zprávy a k zápisu hodnot do příslušných sérií v časové databázi. Aplikace musí být schopna zpracovávat velké množství zpráv v krátkém časovém úseku, po přijetí každé zprávy proto dochází ke spuštění nového procesu či vlákna, jehož účelem je zpracování zprávy.

5.3.2 TCP server

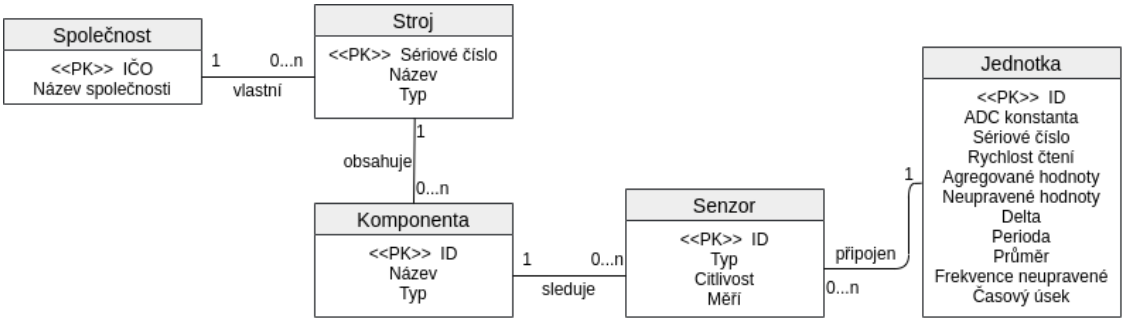
Zasílání neupravených dat se děje z důvodu, že RPi nemá dostatečný výkon pro aplikování pokročilých analýz nad daty o vysoké vzorkovací frekvenci. Data agregovaná na jednotce jsou schopna detekovat poruchu při velké změně hodnot, ale právě agregací se ztrácí informace, které mohou indikovat blížící se poruchu. TCP server slouží k přijímání bloků těchto dat a jejich zapsání do souborů ve formátu TDMS a uložení na datové úložiště. Formát byl zvolen z důvodu využití aplikace partnerské firmy pro pokročilé zpracování dat. Tato aplikace je vytvořena v programu LabView a vstupem jsou právě soubory ve formátu TDMS.

5.3.3 Relační databáze

Jednoduchá relační databáze slouží pro správu a nastavení jednotek, senzorů a přehled monitorovaných strojů. Schéma databáze je vidět na obrázku 5.5. Databáze obsahuje tabulky se společnostmi, jimi vlastněnými stroji a komponenty jednotlivých strojů. Senzory jsou připojené ke komponentům a tabulka senzorů obsahuje informaci o citlivosti a zda je senzor zapojen. Senzory přísluší k měřící jednotce, tabulka jednotky obsahuje zbylé informace potřebné pro vygenerování konfiguračního souboru.

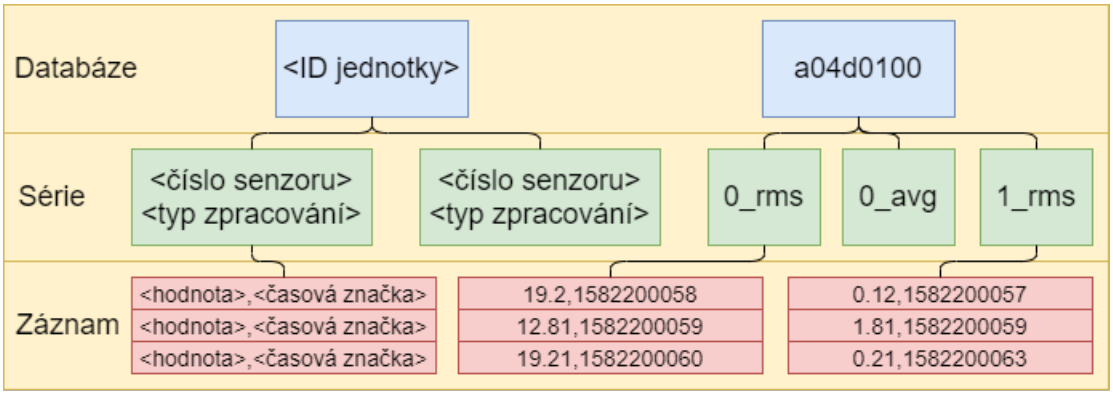
5.3.4 Časová databáze

Tento typ úložiště slouží k ukládání naměřených zpracovaných dat. Schéma na obrázku 5.6 obsahuje jednu databázi pro každou jednotku, jejíž název je unikátní ID jednotky.



Obrázek 5.5: Schéma návrhu relační databáze.

Jednotlivé databáze obsahují samotné časové série. Jméno časové série je odvozeno podle čísla senzoru, kterým byly hodnoty naměřeny, a podle způsobu, kterým byly zpracovány. Záznam je kombinací hodnoty a času. Pro zobrazování hodnot je využito grafické rozhraní zvolené databáze.



Obrázek 5.6: Schéma rozložení dat v časové databázi.

5.3.5 Ostatní programy a funkcionalita

Součástí návrhu je program pro přesměrování paketů. Tento program umožňuje připojení více instancí TCP a UDP serverů. Pro tuto funkcionalitu je zvoleno již existující volně dostupné řešení a jeho využití v systému je řešeno v rámci testování.

Další součástí je generování konfiguračního souboru. Program při zadání identifikátoru jednotky provede vygenerování konfigurace do souboru v místě spuštění. Konfigurace jsou následně jednotkami staženy přes HTTP dotaz.

Kapitola 6

Implementace

Tato kapitola popisuje implementaci jednotlivých programů, popsaných v návrhu. Kromě popisu implementace kódové části a ukázky nejzajímavějších částí kódu, se kapitola věnuje i postupu instalace a přípravy jednotlivých částí a návodům. Návody jsou umístěny v přílohách.

6.1 Server

Jako operační systém pro server byla zvolena linuxová distribuci Ubuntu, respektive Ubuntu server. Zvolena byla z důvodu její rozšířenosti, podpory a dostupnosti použitých knihoven. Některé příkazy použité při instalaci a přípravu serveru jsou v umístěny v příloze A.

6.1.1 Příprava serveru

Po instalaci operačního systému je vhodné povolit vzdálený přístup přes SSH (Secure Shel) pro možnost vzdálené správy. Zároveň je vhodné změnit port služby, aby bylo možné přistupovat k více zařízení v jedné síti přes SSH z internetu a z bezpečnostních důvodů deaktivovat účet *root*. Port je potřeba povolit a pro přístup z vnější sítě je nutné nastavit na routeru přesměrování portu na daný server. Jednotlivé příkazy jsou vypsány v A.1. Veškeré použité knihovny a příkazy k jejich instalaci jsou v příloze A.2.

6.1.2 Časová databáze

Při instalaci a nastavení časové databáze IfluxDB a webového rozhraní Chronograf bylo postupováno podle oficiální dokumentace[1]. Po instalaci je vhodné upravit nastavení v konfiguračním souboru */etc/influxdb/influxdb.conf*. V souboru je nutné nastavit IP adresu, port a zapnout HTTP API s autorizací. V konfiguračním souboru lze nastavit i zálohování, dobu uchovávání dat a podobně. InfluxDB ukládá a čte data z velkého množství souborů a proto je nezbytné zvýšit limit otevřených souborů pro proces. Pokud se neprovede zvýšení limitu, tak se databáze po vložení určitého množství dat stane nespustitelnou. Nespustitelnou zůstane až do navýšení limitu, protože nedokáže otevřít veškeré požadované soubory. Pro zvýšení limitu je potřeba v souboru */etc/systemd/system/influxd.service* upravit nebo přidat řádek *LimitNOFILE=infinity*. Dále je nezbytné v souboru */etc/security/limits.conf* upravit limity pro uživatele a do terminálu zadat příkaz *sudo sysctl -w fs.file-max=65000* a následně restartovat počítač. Zvýšení limitu lze zkontrolovat příkazem

[1] https://docs.influxdata.com/

grep files /proc/<influxdb pid>/limits. Pokud se zvýšení provedlo, je hodnota limitu vyšší než jeden milion.

V dalším kroku je potřeba vytvořit účty a tím zamezit přístupu k databázi bez hesla. Do časové databáze se v terminálu připojuje příkazem *influx -username <username> -password <password>*, pokud není vytvořen žádný účet není nutné použít parametry účtu a hesla. Administrátorský účet se vytváří pomocí příkazu *CREATE USER <username> WITH PASSWORD '<password>' WITH ALL PRIVILEGES*. Jednotlivé aplikace přistupující k databázi mají své vlastní účty s nezbytnými právy.

Pro zobrazování dat z databáze je využita webová aplikace Chronograf dostupná s databází Influx. Aplikace je určena pouze pro lokální zobrazování dat a přístup do Chronografu nelze omezit pomocí přihlašovacích údajů. Přihlašovací údaje k databázi má uloženy globálně pro všechna připojení. Z důvodu testování je vytvořen pro aplikaci účet s právy pro čtení, aby bylo umožněno k zobrazení dat přistupovat i mimo lokální síť. Při každém vytvoření nové databáze je potřeba jednotlivým uživatelským účtům (mimo administrátorké účty) přiřadit práva k nové databázi. Z toho důvodu byl vytvořen skript *privileges.sh*, který automaticky jednotlivým účtům přidá určená práva k datovým databázím s názvem ve formátu *dat_<UID>*, a tím usnadní přidávání nových databází. Skript je umístěn na přiloženém médiu. Chronograf je po spuštění dostupný pomocí lokálních webových stránek na portu 8888. Pro zobrazení dat je nutné se připojit pomocí účtu vytvořeném v InfluxDB. Pro přístup k Chronografu mimo lokální síť, je nezbytné na routeru nastavit přesměrování portu 8888 na zařízení s databází.

6.1.3 Relační databáze

Pro relační databázi byla zvolena MariaDB s webovým rozhraním Adminer. Pro snadnější instalaci je na přiloženém médiu skript *mariadb.sh*, který provede kompletní instalaci obou aplikací. Skript nainstaluje mariaDB, PHP, curl a webový server nginx. Dále vytvoří složku */var/www/html/sql* pro webové rozhraní Adminer. Do složky je stažena aplikace Adminer, dojde k přidání informací o webovém serveru do konfiguračního souboru */etc/nginx/sites-enabled/default* a */etc/hosts* a k dalším nutným úpravám konfiguračních souborů nginx a PHP pro zprovoznění webové aplikace Adminer. Po instalaci mariaDB je přidán administrátorský účet (přihlašovací údaje lze nalézt ve skriptu) a služby jsou spuštěny. Webové rozhraní pro správu databáze je dostupné na portu 9999. Databáze vytvořená podle návrhu 5.5 je v exportované podobě bez hodnot uložená na přiloženém médiu v soubor *bp_mariadb.sql*.

6.1.4 TCPServer

Program se spouští s argumenty *-p <port>* pro nastavení portu na kterém má TCP server naslouchat a *-v* pro výpis logů do konzole. Program nevyužívá žádný konfigurační soubor. Pro logování je využita knihovna spdlog[2]. Knihovna se skládá pouze z hlavičkových souborů a není proto nutná její instalace. Ukládání logů rozdělených na informační a chybové se provádí do několika rotujících souborů o maximální velikosti 5 MB. Po spuštění provede program inicializaci soketu a čeká na připojení. Při navázání spojení od klienta dojde k vytvoření nového procesu. Původní proces čeká na připojení dalšího klienta a nově vzniklý proces provede příjem a zpracování zprávy. Při příjmu dojde nejdříve k uložení hlavičky paketu do struktury 6.1. Po načtení hlavičky se provede alokace ukazatele dat *data* na velikost *dataSize* a k příjmu naměřených hodnot.

[2] https://github.com/gabime/spdlog

```
#define NUMBER_OF_CHANNELS

#pragma pack(push, 1)
struct TCPMessage{
    /* Header */
    uint16_t version;
    uint32_t uid;
    uint16_t channelMask;
    uint32_t timestamp;
    uint32_t duration;
    float adcConstant;
    float sensitivity[NUMBER_OF_CHANNELS];
    /* Data */
    uint32_t dataSize;
    uint8_t *data;
};
#pragma pack(pop)
```

Výpis 6.1: Struktura pro příjem TCP zprávy.

Data TCP zprávy ve formě struktury jsou zpracována třídou *Message*. Třída z políčka *channelMask* vyčte, které kanály jsou obsaženy ve zprávě a zkontroluje velikost dat a informace v hlavičce. Pokud je zpráva poškozená, dojde k zaznamenání chyby a zahození zprávy. Po zpracování je instance třídy *Message* zastřešující přijatá data předána třídě *TDMSFile*, která vytváří strukturu souboru ve formátu *tdms*. Formát souboru se skládá ze tří částí:

- hlavička zvaná *Lead In* obsahující informace o verzi, velikosti dat a podobně,
- sekce *Meta Data* s informacemi o jednotlivých kanálech a
- část *Raw Data* s daty z kanálů uložených po blocích [18].

V době psaní této práce nebyla k dispozici žádná knihovna pro operační systém Linux a programovací jazyk C++, která by tvorbu těchto souborů umožňovala. Vytvořená třída *TDMSFile*, starající se o tvorbu souboru, neobsahuje plnou funkcionalitu pro práci se soubory, je totiž určena pouze k vytvoření souboru s pevně danou strukturou. Tato vlastnost umožnila třídu velice zjednodušit. Příkladem zjednodušení je tvorba hlavičky, která má pevnou délku a jediné části, u kterých dochází ke změně je velikost a offset dat. Funkce pro zápis hlavičky do souboru vypadá následovně:

```
#define TOC_MASK 0x0000000e
#define TDMS_VERSION 0x00001269

void TDMSFile::writeLeadIn() {
    uint32_t tocMask = TOC_MASK;
    uint32_t versionNumber = TDMS_VERSION;
    //first 4 bytes is TDSm tag
    fwrite("TDSm", sizeof(uint8_t), 4, tdmsFile);
    //4 bytes ToC mask
    fwrite(&tocMask, sizeof(uint32_t), 1, tdmsFile);
```

```
11      //4 bytes version number
12      fwrite(&versionNumber, sizeof(uint32_t), 1, tdmsFile);
13      //8 bytes content length without lead in length
14      fwrite(&dataLen, sizeof(uint64_t), 1, tdmsFile);
15      //8 bytes raw data offset without lead in length
16      fwrite(&rawDataOffset, sizeof(uint64_t), 1, tdmsFile);
17  }
```

Výpis 6.2: Funkce pro zápis hlavičky tdms souboru.

Meta Data obsahují informace o kanálech nezbytné ke zpracování dat jako citlivost, konstanta AD převodníku a podobně. Některé údaje této části jsou opět neměnné. Naměřené hodnoty jsou do souboru zapsané po blocích ve formě 32b celočíselných hodnot. Název souboru obsahuje identifikátor jednotky a časovou značku počátku měření hodnot. Po vytvoření souboru dojde k ukončení procesu pro zpracování. Součástí programu je i knihovna *libzip* pro komprimaci souborů.

Zdrojový kód programu psaného v C++ je umístěn na přiloženém médiu ve složce *ServerSW/RawServer/*. Příkazy použité k přeložení programu jsou v příloze v sekci A.3.

6.1.5 UDPServer

Program se opět spouští s argumenty *-p <port>* a *-v* a je implementovaný v jazyce C++. Součástí je i konfigurační soubor obsahující přihlašovací údaje do časové databáze. Program po spuštění zpracuje konfigurační soubor a vytvoří soket na daném portu. Při příjmu zprávy od jednotky uloží přijatá data do pole o velikosti 512 bajtů, vytvoří instanci třídy *Message*, která má na starosti zpracování zprávy a spustí nové vlákno pro zápis dat do databáze. Server je omezen v počtu vláken které může spustit v jeden okamžik, tato hodnota se nastavuje změnou makro konstanty *NUM_OF_THREADS*. Hodnota by měla být nastavena podle výkonu zařízení na kterém je aplikace spuštěna. Nově spuštěné vlákno provede zpracování zprávy. Při zpracování zprávy je postupně plněna struktura 6.3 navržena podle komunikačního protokolu.

```
1   struct SensorData{
2       uint32_t offset = 0;
3       float value = 0;
4   };
5
6   struct Sensor{
7       uint8_t sensorId = 0;
8       uint8_t dataType = 0;
9       uint16_t dataSize = 0;
10      SensorData *data = nullptr;
11  };
12
13  struct UDPMessage{
14      uint16_t version = 0;
15      uint32_t uid = 0;
16      uint32_t timestamp = 0;
17      uint16_t numberOfSensors = 0;
18      Sensor *sensor = nullptr;
```

```
19 };
```

Výpis 6.3: Struktura dat zpracovaných z UDP zprávy.

Pro zápis dat do časové databáze Influx je využito HTTP komunikační rozhraní a knihovna curlpp. Vkládání hodnot probíhá přes HTTP požadavek obsahující přihlašovací údaje, upřesňující parametry a databázový příkaz. Metodou GET jsou předávány příkazy (*write/read*) a parametry. Nezbytné parametry jsou: název cílové databáze *db*, uživatelský účet *u* a heslo *p*. Příkaz může obsahovat i pokročilejší parametry jako *precision*, který udává přesnost časové značky. Výchozí přesnost je v nanosekundách. Databázový dotaz je předáván metodou *POST*, je možné zapisovat několik hodnot najednou. Hodnoty jsou odděleny ukončením řádku a obsahují název časové série, hodnoty tagů a polí, naměřená data a časovou značku. Metoda pro zápis dat iteruje podle počtu senzorů a hodnot obsažených ve struktuře zprávy. Veškeré body zapíše do řetězce znaků a všechny body obsažené v jedné UDP zprávě zapíše v jednom dotazu. Příklad přidání jednoho bodu do SQL dotazu může vypadat následovně:

```
1 writeQuery += "sensor_" + sensorIndex + "_" + "rms_delta" + " value=" +
     data.value) + " " + data.timestamp"\n";
```

Výpis 6.4: Přidání jednoho bodu do databázového dotazu

Pokud jde o nultý senzor s hodnotou 10.0 a časovou značkou 555555 nově přidaný řádek obsahuje *sensor_0_rms_delta value=10.0 555555*.

Zdrojový kód programu psaného v C++ je umístěn na přiloženém médiu ve složce *ServerSW/AggregationServer/*. Příkazy použité k přeložení programu jsou v příloze v sekci A.3. Ke správnému fungování je potřeba dodat konfigurační soubor s přihlašovacími údaji do časové databáze.

6.1.6 Generování konfiguračního souboru

Pro generování konfiguračního souboru ve formátu .ini slouží program *ConfigGenerator*, umístěný na na přiloženém médiu ve složce *ServerSW/ConfigGenerator/*. Program je vytvořený v jazyce C++ a vyžaduje konfigurační soubor s přihlašovacími údaji do relační databáze. Logování chyb opět probíhá pomocí knihovny spdlog. Program vyžaduje při spuštění argument obsahující identifikační číslo jednotky. Po spuštění se vygeneruje konfigurační soubor do aktuální složky. Ke komunikaci s relační databází je využita knihovna *mysqlcpp-conn*. Program obsahuje třídu *ConfigGenerator*, která se stará o generování konfiguračního souboru. Třída obsahuje tři veřejné metody:

- *void generateConfig();* získá všechny informace o jednotce z databáze, přesněji z tabulek *Unit* a *Sensor*,
- *std::string getConfigString();* vrací řetězec znaků obsahující konfigurační soubor,
- *void saveConfig(std::string path);* uloží konfigurační soubor, parametr *path* udává cestu k souboru.

Tato třída a její veřejné metody umožňují připojení k například TCP serveru, který by na dotaz žádost jednotky odeslal konfigurační soubor. V této práci je získání konfiguračního souboru řešeno přes HTTP dotaz a proto tento program pouze ukládá konfigurační soubor.

6.2 Jednotka

Jak již bylo zmíněno, jednotka disponuje operačním systémem Raspbian. Jednotku není potřebné nijak připravovat, jediné co je třeba udělat pro její zprovoznění je vytvořit obraz systému na SD kartu pomocí instalačního skriptu. Jednotka obsahuje program pro čtení a filtrování, agregaci, stahování konfiguračního souboru a nahrávání logů na server.

6.2.1 Instalační skripty

Jedná se o sadu skriptů a souborů sloužících k téměř bezzásahovému zprovoznění jednotky. skripty jsou uloženy na přiloženém médiu ve složce *ServerSW/SDCardInit/*. Jediný skript, se kterým pracuje uživatel je *DiskCreator.sh*. Tento skript slouží k překopírování všech potřebných souborů a ostatních souborů na SD kartu. Ve skriptu je potřeba upravit řádek *mount /dev/sdd2 /mnt/sdcard* podle zařízení, na kterém je SD karta připojena. Skript kromě kopírování provede na kartě změnu inicializačního skriptu */etc/rc.local*. Tento skript se vykoná při každém spuštění. Do souboru přidá řádek spouštějící první instalační skript. Po prvním zapnutí jednotky s nachystanou SD kartou je vykonán skript *RTPatch.sh*, který ze souboru */proc/cpuinfo* zjistí verzi procesoru, podle ní zjistí verzi RPi a podle zjištěné verze aplikuje jeden z předkompilovaných real-time patchů dostupných pro verze 3b, 3b+ a 4b. Po aplikaci patche je nutný restart, po kterém se spustí skript *init.sh*. Další skript provede postupně:

- spuštění SSH,
- instalaci knihoven,
- inicializaci watchdogu,
- stažení a překlad programů z git repozitáře,
- nastavení programů jaky systémové služby,
- přidání skriptů do služby crontab pro jejich periodické spouštění,
- smazání nepotřebných zdrojových souborů a
- a finální restart jednotky.

Po finálním restartování je stažen konfigurační soubor a jednotka začíná měřit a odesílat data.

Skript pro kopírování souborů se spouští příkazem *sudo bash ./DiskCreator.sh*. Na SD kartě musí již být vytvořen obraz operačního systému. Obraz operačního systému Raspbian Buster Lite[3] lze vytvořit například pomocí aplikace balenaEtcher[4]. Následně stačí jednotku připojit k internetu, vložit SD kartu a jednotka do pár minut začne měřit a odesílat data na server.

[3] https://www.raspberrypi.org/downloads/raspbian/
[4] https://www.balena.io/etcher/

6.2.2 Čtení a filtrování dat

Čtení a filtrování dat provádí jeden program spouštějící dva procesy. Každý proces má alokované jedno fyzické jádro procesoru, které není k dispozici operačnímu systému. Problémem zvolené platformy je, že systém neběží v reálném čase a uživatel nemá plnou kontrolu nad vykonáváním programů. To by mohlo způsobit vynechávání velkého množství vzorků při čtení dat o vysoké vzorkovací frekvenci. Alokace jader a instalace preemptivního real-time patche[5] pro kernel z velké části řeší problém zvolené platformy. Alokace jader se provádí při inicializaci jednotky a to přidáním parametru *rootwait/isolcpus=2,3 rootwait* souboru */boot/cmdline.txt*. Při pouštění programu je potřeba říci systému, na jakých jádrech a s jakou prioritou má program spustit. Příklad spuštění vypadá následovně:

```
$ sudo taskset -c 2,3 chrt -f 99 nice -n 20 <program> <program arguments>
```

Příkaz *taskset -c 2,3* zbůsobí spuštění programu na jádrech 2 a 3, příkaz *chrt -f 99* nastaví plánování procesu FIFO (First In First Out) a na nejvyšší prioritu (hodnota 99). Příkaz *nice -20* opět nastavuje nejvyšší možnou prioritu procesu.

Po spuštění programu dojde k načtení konfiguračního souboru ve formátu .ini do struktury *Config*. K zjednodušení parsování konfiguračního souboru je využita knihovna *INIReader.h*[6]. Po načtení konfiguračního souboru je inicializována statická třída pro logování využívající stejnou knihovnu spdlog jako serverové aplikace a provede se vytvoření nového procesu.

Původní proces se stará o čtení dat z nastaveného rozhraní. Proces si vytvoří instanci třídy pro čtení. Třídy pro čtení jsou vytvořené dvě, jedna pro čtení ze souboru a druhá pro čtení z SPI. Obě třídy implementují stejné rozhraní a je tedy možné měnit jejich instance. K naslouchání signálu *data ready* a komunikaci přes piny pomocí SPI ve stejnojmenné třídě je využita knihovna bcm2835 psaná v jazyce C a je využitelná na všechny verzi RPi. Knihovna poskytuje přístup k pinům a dalším vstupně-výstupním zařízením a službám jako je SPI nebo I^2C. Třída obsahuje dvě veřejné metody, jednu pro inicializaci spojení a druhou pro nekonečné čtení z rozhraní. Tyto dvě metody jsou zděděny z rozhraní *InterfaceReader*. Při inicializaci dochází k nastavení SPI a převodníku ADS131A04 podle dokumentace[7] a vytvoření pojmenované sdílené paměti. Sdílená paměť a semafory zajišťující výlučný přístup ke sdílené paměti jsou realizovány knihovnou boost. Vytvořená sdílená paměť má velikost bloku dat určitého časového intervalu upřesněného v konfiguračním souboru.

Vyčítání a odesílání dat provádí funkce *void bcm2835_spi_transfernb(char *tbuf, char *rbuf, uint32_t len)*. Hexadecimální hodnoty registrů použité pro nastavení převodníku jsou umístěny ve zdrojovém souboru *Spi.h*. Hodnoty jsou připravené k přenosu při klesající hraně signálu *data ready*, a proto je využita z knihovny funkce *bcm2835_gpio_afen(int pin_number)*, která při padající hraně signálu na daném pinu nastaví událost (nastaví bit v registru). Získání dat probíhá aktivním čekáním na událost pomocí funkce *bool bcm2835_gpio_eds(int pin)*. Po nastání události jsou data přenesena a zapsána do jedné ze dvou sdílených pamětí. Jakmile je sdílená paměť naplněna, provede se odemčení semaforu dané paměti a tím se povolí čtení filtrovacím procesem. Následně dojde k uzamčení semaforu druhé paměti pro čtení a začne zápis do druhé sdílené paměti. Díky této implementaci je vždy dostupný jeden blok sdílené paměti jak pro zápis tak i čtení.

V případě čtení ze souboru je dosaženo vzorkování na požadované frekvenci vyvedením hodinového signálu na volný pin. Tento pin je potřeba fyzicky propojit s pinem nastave-

[5] https://wiki.linuxfoundation.org/realtime/start
[6] https://github.com/benhoyt/inih
[7] http://www.ti.com/lit/ds/symlink/ads131a04.pdf

ným jako *data ready*. Vyvedení hodinového signálu je provedeno za použití knihovny *pigpio* funkcí *void gpioHardwareClock(int pin_number, int frequency)*. Naslouchání signálu a přenos hodnot probíhá stejně jako v případě třídy SPI.

Filtrovací proces po uvolnění semaforu zkopíruje obsah paměti do bufferu a uloží čas měření. Data z jednotlivých senzorů jsou zpracována decimačním filtrem, a tím je snížena vzorkovací frekvence na 8000 Hz. Data se sníženou vzorkovací frekvencí jsou následně zapsána do jednoho ze dvou bloků sdílené paměti, který slouží pro výměnu dat mezi filtrovacím procesem a agregačním programem. Následně je zkontrolováno, zda přijatý blok měření má být odeslán ve formě TCP zprávy a pokud ano, jsou původní nefiltrovaná data odeslána přes TCP spojení.

6.2.3 Agregace dat

Agregaci hodnot provádí program *Reader* psaný v C++. Program je umístěn na přiloženém médiu ve složce */UnitSW/Aggregation* a spouští se s parametrem *-p <path>* s cestou ke konfiguračnímu souboru. Program pro logování využívá knihovnu spdlog.

Program po spuštění načte z konfiguračního souboru do struktury veškeré informace. Po načtení konfigurace dojde k vytvoření instance třídy *Aggregation*, inicializuje ji a spustí agregaci. Při inicializaci je otevřena sdílená paměť pro čtení, semafory a je alokována paměť pro zpracovávaná data. Sdílená paměť musí být vytvořena již před spuštěním tohoto programu. Program data zpracovává po časových úsecích, jejichž velikost je daná v konfiguračním souboru. Po získání bloku dat pro zpracování je zavolána metoda *void calculateRMS();*, která přijatá data převede na zrychlení a z bloku hodnot o snížené vzorkovací frekvenci spočítá kvadratický průměr, jehož vzorec je v 3.1. Metoda je zobrazená na 6.5. Metoda v každém kroku hlavní smyčky spočítá jednu hodnotu kvadratického průměru pro každý senzor. Počet vzorků, které jsou průměrovány do jedné hodnoty, je nastaveno v konfiguračním souboru. Metoda vypočítané hodnoty ukládá do dvourozměrného pole bodů rozděleného podle senzorů. Počet bodů pro průměrování je ve výchozím stavu nastaven tak, aby po přepočtení na kvadratický průměr zbývalo 100 bodů měření z každého senzoru. Po výpočtu kvadratického průměru je aplikován algoritmus $\Delta treshold$. První bod každého bloku je vždy uložen pro odeslání. Ostatní body ponechány pro odeslání v případě, že rozdíl jejich hodnoty a hodnoty posledního uloženého bodu překračuje v konfiguraci nastavený práh, nebo pokud bylo od posledního uloženého bodu zahozeno příliš bodů (časový práh opět nastavený v konfiguračním souboru). Body pro odeslání jsou uloženy do jednorozměrného pole struktur *DataPoint*. Struktura vypadá takto:

```
1 struct DataPoint{
2     float value;
3     uint32_t time;
4     uint32_t timeOffset;
5     uint8_t sensorNumber;
6 };
```

a obsahuje hodnotu bodu, unixovou časovou značku času, kdy byl bod naměřen ve vteřinách, časový ofset v nanosekundách a index senzoru. Po zpracování celého bloku hodnot je pole ponechaných bodů předáno třídě *UDPClient*, která z bodů poskládá zprávu odpovídající protokolu 5.3a a zprávu odešle na server. Po odeslání zprávy program čeká na další blok dat.

```
1  void Aggregation::calculateRMS() {
2      int rmsIndex = 0;
3      for(int dataIndex = 0; dataIndex < valuesPerChannel; dataIndex +=
           config.average){
4          float valueSum[NUMBER_OF_SENSORS] = {0};
5          for(int blockOffset = 0; blockOffset < config.average;
               blockOffset++){
6              for(int channelIndex = 0; channelIndex < NUMBER_OF_SENSORS;
                   channelIndex++){
7                  float valueInG = buffer[dataIndex + blockOffset +
                       channelIndex*valuesPerChannel];
8                  valueInG *= config.adcConstant;
9                  valueInG *= config.sensorSensitivity[channelIndex];
10                 valueSum[channelIndex] += valueInG*valueInG;
11             }
12         }
13         for(int sensorIndex = 0; sensorIndex < NUMBER_OF_SENSORS;
               sensorIndex++){
14             rms[sensorIndex][rmsIndex] =
                   sqrtf(valueSum[sensorIndex]/config.average);
15         }
16         rmsIndex++;
17     }
18 }
```

Výpis 6.5: Výpočet kvadratického průměru z bloku dat několika senzorů.

6.2.4 Ostatní programy a funkcionalita

K získání konfiguračního souboru ze serveru slouží skript *getconfig.sh* umístěný na přiloženém médiu ve složce *UnitSW/*. Konfigurační soubory jsou ze serveru získávány pomocí HTTP dotazu na portu 9999. Skript si udržuje hlavičku HTTP dotazu získanou při předchozím stažení souboru obsahující informaci o poslední změně souboru. Skript periodicky získává ze serveru nové informace o souboru a pokud dojde na ke změně souboru na serveru (indikující změna políčka *Last-Modified*), stáhne se nový soubor a restartuje se měření s novými parametry. Hlavička HTTP odpovědi, ze které skript získává informace vypadá následovně:

```
HTTP/1.1 200 OK
Server: nginx/1.16.1 (Ubuntu)
Date: Thu, 19 Mar 2020 09:44:44 GMT
Content-Type: application/octet-stream
Content-Length: 832
Last-Modified: Thu, 19 Mar 2020 09:31:21 GMT
Connection: keep-alive
ETag: "5e733be9-340"
Accept-Ranges: bytes
```

Kapitola 7

Testování

Tato kapitola se věnuje testování nejen systému jako celku, ale i testování jednotlivých komponent. Kromě testování funkčnosti systému, je kapitola též zaměřena na zátěžové testy serverových aplikací. Do zátěžových testů patří například zjištění, do kolika jednotek jsou jednotlivé aplikace schopné zpracovávat data bez zahlcení. Další částí jsou testy programu pro balancování zatížení serverových aplikací. Ten umožňuje spuštění několika instancí TCP a UDP serveru na více zařízeních, a tím navýšit limit zpracovávaných zpráv. Kapitola je také věnovaná testování zabezpečení síťové komunikace mezi jednotkou a serverovými aplikacemi a komprimace odesílaných dat.

7.1 Test instalačních skriptů a kompletní funkce systému

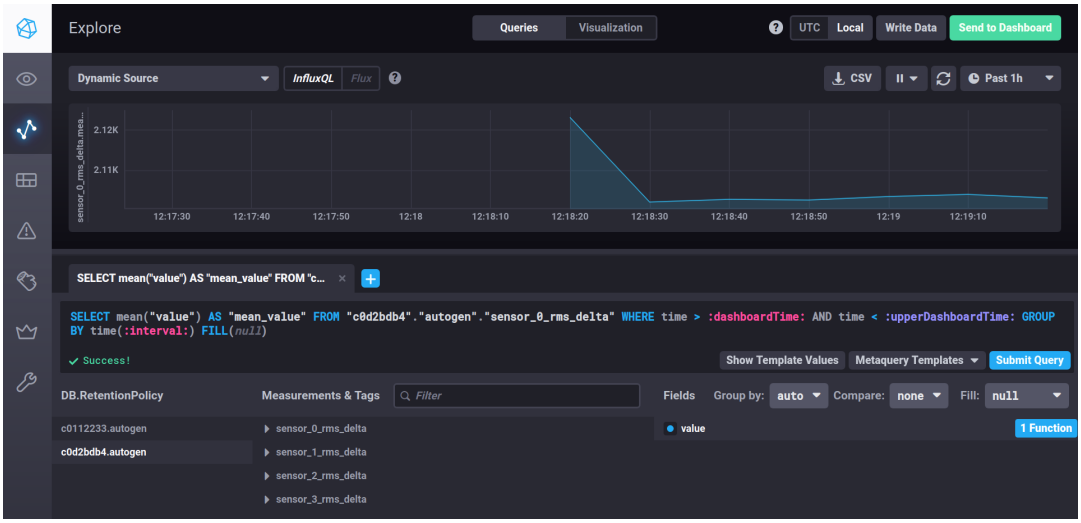
Cílem tohoto testu je vyzkoušet správnou funkčnost instalačních skriptů. Při správném fungování by po přípravě microSD karty a databáze měla jednotka během pár minut bez jakéhokoliv zásahu začít měřit a naměřená data odesílat na server.

Tento test byl prováděn s rozšiřující deskou ve verzi 1.0. Desky ve verzi 1.0 a 1.1 nedisponují unikátním identifikátorem a z toho důvodu bylo zvoleno jako ID kombinace MAC adresy RPi a typu jednotky. Před inicializací jednotky je nutné jednotku v databázi vytvořit a k tomu je potřeba znát ID. Pro zjištění identifikátoru byla do RPi vložena karta s neupravenou instalací operačního systému Raspian a spuštěn příkaz:

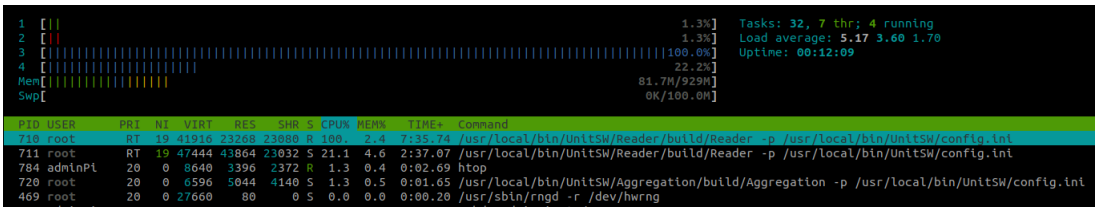
```
$ echo "c0$(cat /sys/class/net/eth0/address | tr -d ':' | tail -c 7)"
```

Tento příkaz na standardní výstup vypíše dvě hodnoty pro námi zvolený typ jednotky ($c0$) a posledních šest hodnot MAC adresy jednotky. V případě testované jednotky to je $d2bdb4$. Do databáze je tedy přidána jednotka s tímto ID. Následně jsou v databázi vytvořeny čtyři senzory s potřebnými parametry a pomocí programu *ConfigGenerator* je vygenerován konfigurační soubor *config*, jehož obsah je možné vidět na přiloženém médiu v kořenovém adresáři. Před přidáním nové jednotky do systému musí být vytvořena i příslušná databáze v časové databázi a uživatelským účtům musí být přidělena práva pomocí skriptu *privileges.sh*. Po provedení těchto kroků je již serverová část připravena na provoz nové jednotky. Dalším krokem pro zprovoznění nové jednotky je vytvoření obrazu operačního systému Raspbian na kartě a spuštění skriptu *SDCardInit.sh*. Jednotka s připravenou kartou musí být před spuštěním připojena k internetu pomocí ethernetového kabelu, jinak nedojde ke stažení knihoven a softwaru. Tímto je dokončena příprava na instalaci.

Jednotka s připravenou kartou byla spuštěna v 11:48:21 a instalace byla dokončena v 12:15:15. Jednotka má nastavené dvouminutové zpoždění pro spuštění čtecích programů



Obrázek 7.1: První přijatá data po dokončení instalace.



Obrázek 7.2: Využití procesoru při čtení a rozložení procesů na jednotlivých jádrech.

(zamezení restartovací smyčky způsobené možnou nestabilitou čtecího programu) a první agregované hodnoty byly tedy do databáze zapsány o zhruba dvě až tři minuty později, viz obrázek 7.1. Na obrázku je možné vidět, že byly přijaty agregované hodnoty ze čtyř senzorů. K jednotce nebyl připojen senzor, jedná se tedy pouze o šum na analogovém vstupu. Konfigurační soubor byl před spuštěním úspěšně stažen ze serveru. Na obrázku 7.2 je vidět rozložení procesů na jednotlivých jádrech. Z obrázku je patrné, že program *Reader* využívá dvě procesorová jádra, která pro něj byla alokována. Proces využívající třetí jádro provádí čtení dat z nastaveného rozhraní. Stoprocentní využití jádra způsobuje aktivní čekání na signál *data ready*. Proces spuštěný na čtvrtém jádře aplikuje filtry na bloky dat získaných od čtecího procesu a využívá jádro průměrně na 50%. Oba procesy mají RT prioritu. Agregační program je spuštěn na zbývajících jádrech společně se systémem. Soubory s logy byly vytvořeny správně na nastavené cestě a obsahovaly informaci o neúspěšném pokusu o navázání TCP spojení. Při kontrole konfiguračního souboru byla objevena chyba v IP adrese pro TCP server. IP adresa byla opravena na serveru a opravený soubor byl jednotkou během následujících pěti minut stažen a došlo k restartování čtení a byla navázána i TCP komunikace viz obrázek 7.3. Výstup instalačního skriptu je směřován do souboru *install_log.txt*. Tato

Obrázek 7.3: Informace o příchozí zprávě na TCP server.

vlastnost je důležitá pro kontrolu instalace. Pokud dojde při instalaci k jakékoliv chybě, je možné záznam o ní vyhledat právě v tomto souboru. Soubor z testovací instalace je přiložen v kořenové složce na přiloženém médiu. Výstup skriptu byl zkontrolován a neobsahuje žádná chybová hlášení.

Tento test ověřil funkčnost bezzásahové inicializace jednotky a dobu nutnou k jejímu zprovoznění. Čas potřebný pro zaměstnance pro přidání jednotky do databáze a přípravu instalační karty je zhruba deset minut a je zde prostor pro další urychlení. Například vytvořením jednoduchého uživatelského rozhraní, přes které by bylo možné na jednom místě jednoduše přidat jednotku do databáze a vytvořit její instalační kartu. Bezzásahová instalace jednotky trvala zhruba 27 minut a měření bylo zahájeno po 30 minutách. Tímto testem byla vyzkoušena i kompletní funkcionalita celého systému.

7.2 Komprimace zpráv

Cílem testu bylo porovnat kompresní algoritmy pro využití při odesílání neagregovaných dat a zvolit nejvhodnější algoritmus pro implementaci v jednotce. TCP zprávy obsahující nezpracovaná data mohou nabývat velikosti až několik MB. Z toho důvodu je vhodné využít rychlý kompresní algoritmus pro snížení velikosti, který může jednotka využívat v reálném čase. Pro test byly vybrány běžně využívané kompresní algoritmy *deflate*, *LZO* (Lempel–Ziv–Oberhumer), *LZMA* (Lempel-Ziv-Markov-Chain), *BZip2* a metoda *Delta compression* popsaná v 3.4.1. Algoritmy byly testovány na nezpracovaných historických datech dodaných partnerskou firmou v binární podobě. Algoritmy byly testovány na souborech o velikosti zhruba 500 kB, 3 MB a 10 MB. Testované soubory jsou umístěny na přiloženém médiu ve složce *Testing/*. K testování algoritmů *deflate* a *Delta compression* byl vytvořen program *DataCompress*, umístěný ve stejné složce jako testované soubory. Ostatní algoritmy byly testovány pomocí skriptu *compress.sh*, vytvořeného pro účely testu. Výsledky obsahuje tabulka 7.1.

alg.	500kB		3MB		10MB	
	velikost[kB]	doba[ms]	velikost[kB]	doba[ms]	velikost[MB]	doba[ms]
LZO	222,0	2	1275,0	7	4,1	21
Delta	212,1	24	1234,5	138	4,0	429
BZip2	80,2	89	455,9	219	1,5	522
LZMA	92,6	138	494,6	787	1,7	3201
Deflate	108,8	438	612,5	2048	2,0	6669

Tabulka 7.1: Výsledky testů kompresních algoritmů.

Výsledky testu jsou seřazeny podle doby trvání komprese. Test ukázal, že algoritmus *Deflate*, běžně využívaný knihovnou *libzip*, je pro řešení naprosto nevhodný. Oproti ostatním algoritmům je doba potřebná pro kompresi až čtyřikrát vyšší a to s horším kompresním poměrem. Nejlepšího kompresního poměru dosáhly algoritmy *BZip2* a *LZMA*. *BZip2* je až šestkrát rychlejší než *LZMA*, a proto pokud záleží na kompresním poměru, je nejlepší volbou. U řešeného systému je ale velmi důležitá rychlost komprese, protože jednotka potřebuje provádět kompresi v reálném časem, aby nedocházelo k výpadkům. Algoritmus *Delta* dosahuje zhruba 75% úspory a lepší rychlosti než předchozí algoritmy. *LZO* ovšem dosahuje

stejného kompresního poměru při $10 - 20$ násobně kratší době. Tento algoritmus je tedy pro účely komprese implementován v jednotce.

7.3 Propustnost serverů

Cílem těchto testů je určit množství zpráv, které jsou UDP a TCP servery schopné přijmout za určitý čas. Určením limitu příchozích zpráv na server je možné vyvodit, kolik jednotek je možné obsluhovat jednou instancí. Množství zpráv, které jsou schopné aplikace zpracovat, se odvíjí i od výkonu zařízení na kterém jsou aplikace spuštěny a rychlosti připojení. Veškeré testy UDP a TCP serverů byly prováděny na počítači vybaveném čtyřjádrovým procesorem intel core i7 6700k, 32 GB RAM paměti. Síťová komunikace probíhala přes gigabitové spojení.

Pro účely testu byl vytvořen program *MessageGenerator*, umístěný na přiloženém médiu ve složce *Testing/*. Program po spuštění generuje podle nastavení buď UDP nebo TCP zprávy. Program vytváří několik vláken. Každé vlákno představuje jednu virtuální jednotku, která v nastavených intervalech odesílá zprávy. Program se spouští s následujícími parametry:

- *-m <UDP/TCP>* volba režimu,
- *-u <počet jednotek>* počet jednotek,
- *-f <milisekundy>* čas v milisekundách mezi zprávami od jednotky,
- *-p <port>* port cílového zařízení,
- *-a <IPv4>* IPv4 adresa cílového zařízení.

Generované zprávy mají formát popsaný v 5.2. UDP zprávy jsou plně zaplněny do velikosti 512 B. TCP zprávy obsahují měření o délce deseti vteřin ze čtyř senzorů o vzorkovací frekvenci 128000 Hz. Každé vlákno, neboli virtuální jednotka, má přiřazen speciální identifikátor, například *f0010012*. *f* označuje test, následující tři hodnoty *001* určují číslo testu a poslední čtyři hodnoty udávají index virtuální jednotky.

7.3.1 UDP server

První test UDP serveru ověřoval funkčnost programu pro generování zpráv. Pro účely testu byly v časové databázi vytvořeny jednotky *f0010000* a *f0010001* a pomocí skriptu *privileges.sh* byla přidána práva jednotlivým uživatelským účtům. Cílem testu bylo do databází těchto dvou jednotek vložit sto dvacet hodnot ke každému senzoru z deseti zpráv. Po spuštění programu byl skutečně do databáze pod každý senzor vložen požadovaný počet hodnot. Každá zpráva je pro snadnější vyhodnocení v databázi reprezentována jako jedna vteřina. Pokud se tedy pošle deset zpráv po 50 ms rozestupech, v databázi jsou hodnoty roztaženy do deseti vteřin. Podobným způsobem byla otestována funkčnost na TCP serveru, ze tří zařízení bylo odesláno deset zpráv. Na serverové straně bylo vygenerováno třicet souborů.

U každého provedeného testu bylo potřeba pro všechna zařízení vyhodnotit počet přijatých zpráv a spočítat množství chybějících zpráv. Jelikož by u desítek zařízení bylo toto vyhodnocení velmi zdlouhavé, byl vytvořen vyhodnocovací skript *evaluateLoadTest.sh*. Vstupním parametrem skriptu je *-u <jednotek>* pro počet zařízení a *-d <čas>* určující dobu mezi zprávami v milisekundách. Výstupem skriptu je soubor s tabulkou obsahující informace

o počtu chybějících zpráv v jednotlivých minutách u všech zařízení. Příklad výstupu vyhodnocovacího skriptu pro deset jednotek s jednou zprávou za vteřinu po dobu deseti minut je možné vidět na obrázku 7.4. Každý řádek představuje údaje týkající se jedné jednotky. První sloupec označuje ID jednotky, druhý sloupec celkový počet zpracovaných zpráv a celkový počet odeslaných zpráv. Každý další sloupec představuje jednu minutu testu. Pokud je ve sloupci hodnota 60/60 znamená to, že bylo přijato 60 zpráv a odesláno také 60 zpráv. Skript je umístěn na přiloženém médiu ve složce *Testing/*.

Unit	received/send	0-1min	1-2min	2-3min	3-4min	4-5min	5-6min	6-7min	7-8min	8-9min	9-10min
f0020000	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020001	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020002	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020003	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020004	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020005	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020006	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020007	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020008	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60
f0020009	600/600	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60	60/60

Obrázek 7.4: Ukázka výstupu vyhodnocovacího skriptu.

Pro průběh každého testu bylo nutné vytvořit jednotlivé databáze, což je u desítek zařízení velmi zdlouhavé. K přidávání jednotek do databáze pojmenovaných podle pravidla zmíněného dříve byl vytvořen skript *createTestDatabases.sh* umístěný ve stejné složce jako skript vyhodnocovací. Vstupními argumenty jsou počet jednotek *-u <jednotek>* a číslo testu *-t <číslo testu>*. Skript se pokusí před vytvořením databází smazat staré databáze se stejným názvem.

Výsledky testů bez zabezpečení je možné vidět v tabulce 7.2. Sloupec *jednotek* udává počet virtuálních jednotek, *zpoždění* stanovuje dobu mezi zprávami od jedné jednotky, *délka t.* určuje dobu trvání testu, *přijato* určuje v procentech kolik odeslaných zpráv bylo přijato a *zpráv/s* udává kolik zpráv dohromady UDP server obdržel každou vteřinu. Výstupy všech testů vygenerovaných vyhodnocovacím skriptem jsou na přiloženém médiu ve složce *Testing/output/*.

Prvních devět testů bylo určeno pro hrubý odhad maximálního limitu zpracovaných zpráv. Z testů vyplývá, že UDP server zvládá bez problému zpracovat a zapsat do databáze sto zpráv za vteřinu. Při tisíci zprávách za vteřinu docházelo již k 1-4% ztrátám. Při deseti tisících zprávách za vteřinu byla ztrátovost přes 90 %. Tyto hodnoty naznačují, že maximální množství zpracovaných zpráv na testovaného zařízení je zhruba $950-980$ za vteřinu ze všech jednotek.

Testy číslo 10, 11 a 12 sloužily k přesnějšímu určení bezpečného limitu zpracovaných zpráv. Bezpečné množství, při které nedošlo k žádné ztrátě, je zhruba 600 zpráv za vteřinu. Počet zpráv lze zvýšit zlepšením parametrů zařízení, na kterém je UDP server spouštěn. Je potřeba také brát v potaz, že na stejném zařízení byla spuštěna i časová databáze. Maximální počet agregovaných hodnot, které je jednotka schopná vygenerovat ve verzi 1.0, je 400 hodnot za vteřinu ze všech senzorů. Jedna UDP zpráva protokolu používaného v této práci pojme až 56 hodnot. Jednotka je tedy schopná vygenerovat maximálně 8 UDP zpráv za vteřinu. Testovaný UDP server bez zabezpečení zvládne zpracovávat data bez ztráty od 75 jednotek.

Testy 12 a 13 z tabulky 7.2 byly provedeny se zašifrovanými zprávami. Zprávy byly šifrovány pomocí operace XOR[1]. Šifrování funguje na principu bitového klíče, který je stejně dlouhý jako UPD zpráva. Klíč zná pouze serverová a klientská aplikace a na jednotku je dodán při její instalaci. Každý bit odesílané zprávy je změněn operací XOR s odpovídajícím

[1] https://teambi0s.gitlab.io/bi0s-wiki/crypto/xor/

bitem klíče. Zpráva je následně odeslána a na serveru stejným způsobem dekódována. Jak je možné vidět v tabulce, šifrování mělo vliv na množství zpracovaných zpráv. Už u 600 zpráv docházelo k drobným výpadkům kolem 0,07 %, snížení počtu zpráv na 500 za vteřinu tento výpadek vyřešilo.

Pokud by bylo potřebné zpracovávat data od více jednotek, je nutné využít více instancí serveru a program pro vyvažování zátěže testovaný v 7.4.

č. testu	jednotek	zpoždění[ms]	délka t.[min]	přijato[%]	zpráv/s
1	1	1000	10	100	1
2	10	1000	10	100	10
3	100	1000	10	100	100
4	1	100	10	100	10
5	10	100	10	100	100
6	100	100	10	97, 75	1000
7	1	10	10	99, 99	100
8	10	10	10	96, 78	1000
9	100	10	10	8, 91	10000
10	90	100	10	91, 98	900
11	80	100	10	95, 85	800
12	60	100	10	100	600
13	60	100	10	99, 93	600
14	50	100	10	100	500

Tabulka 7.2: Výsledky zátěžových testů UDP serveru.

7.3.2 TCP server

Pro testování TCP serveru bylo opět využito testovacího programu *MessageGenerator*. Cílem testu bylo zjištění, od kolika jednotek je server schopný přijímat data. Výsledky testů jsou v tabulce 7.3. Oproti předchozímu testu přibyl sloupec *interval*, určující jak dlouhý časový úsek měření zpráva obsahuje. Během každého testu byly určitou dobu odesílány zprávy s fixním zpožděním. K zahlcení došlo, pokud server po uplynutí určené doby nezpracoval požadovaný počet zpráv.

Počet maximálního množství jednotek je určen několika faktory. Závisí na délce zprávy, kterou jednotka odesílá, a také jak často tyto zprávy odesílá. TCP protokol a implementace jednotky umožňují jako nejnáročnější možnou kombinaci nastavení zprávy obsahující jednu vteřinu měření odesílané každou vteřinu, tzn. téměř nepřetržitý datový tok od každé jednotky. Toto nastavení by ovšem vedlo k rychlému zahlcení serverové aplikace a mohlo by způsobit výpadky v měření. Proto je jako nejhorší možný scénář zvoleno nastavení měření o délce deseti vteřin a odesílání všech dat. Testy 1, 2, 3 a 4 se zabývají právě touto situací. Z výsledku lze odvodit, že TCP server je schopný zpracovávat data od 50 jednotek, které posílají veškerá naměřená data po deseti vteřinových úsecích (zhruba pět zpráv za vteřinu).

Systém ovšem není koncipován na odesílání veškerých naměřených nezpracovaných dat, ale na odesílání intervalů nezpracovaných dat po určitém čase. Testy 5 a 6 vychází ze situace, kdy každá jednotka odesílá desetivteřinové úseky měření jednou za minutu. Podle

dosavadních výsledků by server měl v této situaci zvládat zpracovávat data až od 300 jednotek a test číslo 5 to potvrzuje.

č. testu	jednotek	zpoždění[s]	délka t.[min]	interval[s]	přijato[%]	zpráv/s
1	10	10	5	10	100	1
2	100	10	5	10	59,1	10
3	50	10	5	10	100	5
4	70	10	5	10	83,62	7
5	300	60	10	10	100	5
6	300	60	10	10	100	5

Tabulka 7.3: Výsledky zátěžových testů TCP serveru.

Další test byl proveden se zabezpečenou komunikací. Spojení bylo zabezpečeno pomocí knihovny open-ssl a k implementaci byl využit příklad z [3]. Klíč vygenerovaný k tomuto testu je umístěn na přiloženém médiu v kořenovém adresáři. Z testu vyplývá, že zabezpečení spojení nemělo vliv na maximální počet zpracovaných zpráv.

Při testování TCP serveru bylo zjištěno, že je schopný bez zahlcení zpracovávat 5 zpráv za vteřinu obsahující desetivteřinová měření. Počet jednotek, od kterých je schopný data zpracovávat, záleží na jejich nastavení. Při odesílání dat každou minutu je schopný zpracovávat data až od 300 jednotek, což je mnohem více, než je schopný zpracovávat UDP server.

7.4 Škálovatelnost

Jak již bylo prokázáno v předchozích testech, celý systém může být spuštěn na jednom zařízení s limitem zhruba 75 jednotek bez zabezpečené komunikace a 65 jednotek se zabezpečenou komunikací. Při nutnosti obsluhy více jednotek je nutné systém škálovat. Škálovatelnost serverové části se dělí na dvě oblasti – zpracování dat a úložiště.

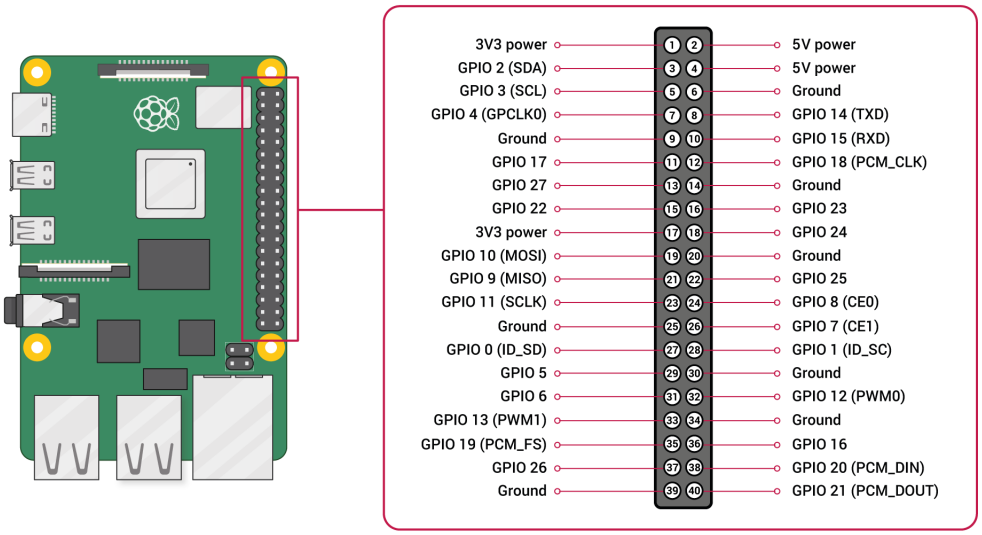
Škálování úložiště se odvíjí od zvoleného řešení. V případě testované časové databáze InfluxDB lze využít placenou verzi, schopnou distribuovat data mezi více uzlů[2].

Množství dat zpracovaných UDP a TCP serverem lze zvýšit spuštěním více instancí těchto programů na více uzlech a jejich rovnoměrným vytížením. Rozložit vytížení (anglicky *Load balancing*) lze pomocí hardwarového zařízení nebo softwaru. Pro tento test byl zvolen program *pen*[3]. Program byl spuštěn s parametry *pen -r -U 9944 127.0.0.1:9943 192.168.1.28:9943*. UDP zprávy přijímané na portu 9944 byly tedy distribuovány mezi dvě zařízení na port 9943. Výsledky testu je možné vidět v tabulce 7.4. Testy systému s více instancemi UDP serveru na více zařízeních prokázaly, že došlo ke zvýšení limitu počtu zpracovávaných zpráv. V testu číslo 1,2 a 3 byla využita tři zařízení. První zařízení generovalo zprávy, druhé zařízení provádělo zpracování poloviny zpráv a přeposílalo druhou polovinu třetímu zařízení, které provádělo zpracování druhé poloviny a byla na něm spuštěna databáze. V těchto testech došlo k téměř zdvojnásobení počtu zpracovaných zpráv. Ideální stav je, když každá část systému má své dedikované zařízení. To znamená, že je v systému zařízení sloužící pouze k rozložení zatížení, několik zařízení pouze pro příjem zpráv a další pouze s databázovými uzly.

[2] https://www.influxdata.com/blog/influxdb-clustering/
[3] https://github.com/UlricE/pen

č. testu	jednotek	zpoždění[ms]	délka t.[min]	přijato[%]	zpráv/s
1	100	50	10	69.63	2000
2	50	50	10	100	1000
3	70	50	10	98.65	1400

Tabulka 7.4: Výsledky zátěžových testů UDP s rozložením vytížení.

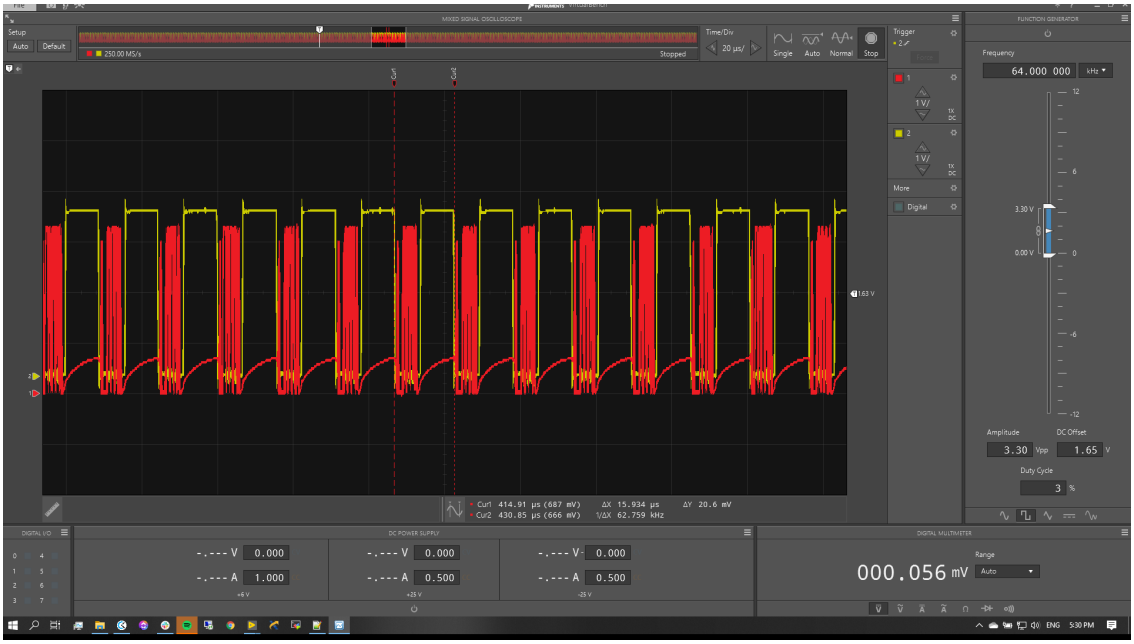


Obrázek 7.5: Rozložení pinů na Raspberry Pi [21].

7.5 Měření jednotky

Cílem tohoto testu bylo ověřit, zda RPi vyčítá data o správné vzorkovací rychlosti z rozšiřující desky nebo souboru a nedochází ke ztrátě dat. Pokud jednotka čte data ze souboru, je vyveden hodinový signál o nastavené frekvenci na jeden z GPIO pinů a tento pin je fyzicky propojen s pinem nastaveným jako *data ready*. Výstupní frekvence výstupního pinu byla změřena pomocí osciloskopu. Na obrázku 7.5 je možné vidět rozložení pinů na RPi. Sonda byla připojena k vývodu hodinového signálu na pinu číslo 7 (GPCLK0). Frekvence výstupního signálu byla nastavena na 128 kHz a souhlasila s hodnotou odečtenou na osciloskopu. Rychlost čtení z rozhraní SPI byla také ověřována pomocí osciloskopu. K RPi s připojenou rozšiřující deskou byly připojeny dvě sondy. Prví sonda byla připojena k pinu číslo 15 (GPIO 22), na který je přiváděn signál *data ready*. Druhá sonda byla připojena na pin číslo 21 (GPIO 9), který funguje jako MISO (Master In Slave Out – příchozí data z SPI od rozšiřující desky). Komunikace byla zaznamenána nejdříve při vzorkovací frekvenci 64 kHz a posléze i při frekvenci 128 kHz. Část komunikace mezi deskou a RPi z prvního testu je možné vidět na obrázku 7.6.

Na obrázku je možné vidět žlutě značený signál *data ready*. Signál má skutečně frekvenci 64 kHz, nastavení AD převodníku tudíž proběhlo správně. Červeně značený signál značí čtení dat z převodníku. Data jsou čtena při sestupné hraně signálu *data ready*. V měření nebylo jediné vynechání čtení. Z obrázku je patrné, že RPi strávilo čtením dat méně než



Obrázek 7.6: Komunikace přes SPI při 64 kHz mezi RPi a rozšiřující deskou.

polovinu doby před příchodem další sestupné hrany. Stejným způsobem bylo změřeno i čtení dat o vzorkovací frekvenci 128 kHz. Stejně jako při předchozím testu nedocházelo k vynechání dat, zařízení využívalo zhruba 90 % času mezi jednotlivými sestupnými hranami pro čtení. Vyšší vzorkovací frekvenci by testovaná implementace nestíhala číst.

7.6 Detekce překročení limitu s historickými daty

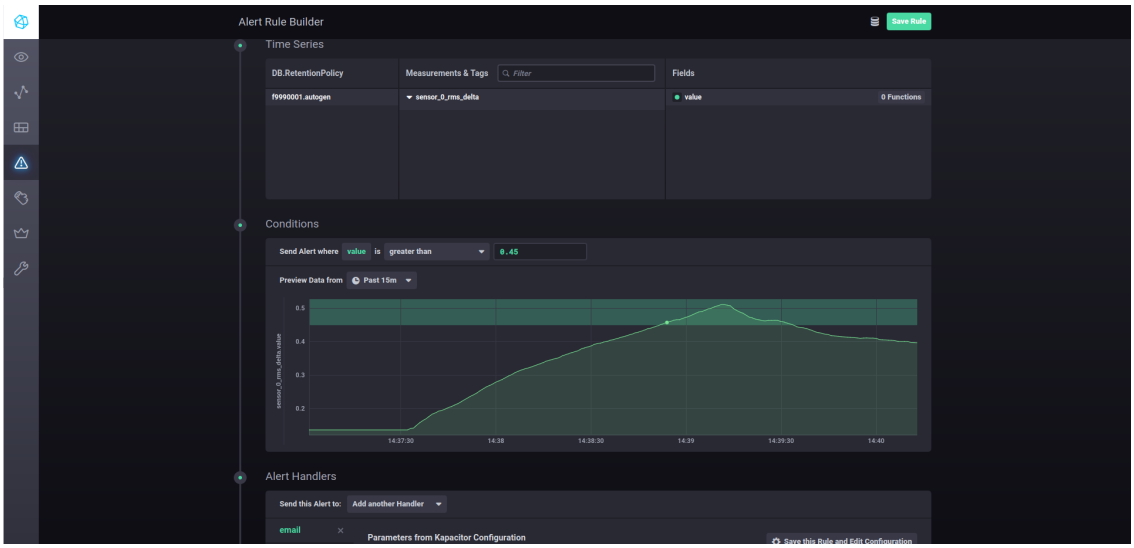
Cílem testu bylo ověřit sledování příchozích dat do databáze a upozornění na překročení limitu u senzoru. K testu byla využita historická agregovaná data partnerské firmy obsahující poruchu stroje. Data byla obdržena ve formátu csv. Pro účely testu byl vytvořen program *CSVImport*, umístěný na přiloženém médiu ve složce *Testing/*. Vstupními argumenty jsou *-f <soubor>* s názvem souboru, který má být importován, *-a <ipv4>* adresa serveru a *-p <port>* s portem cílového serveru. Program postupně čte hodnoty ze souboru a vytváří UDP zprávy, které odesílá UDP serveru. Pro tento test nemohlo být využito čtení ze souboru na jednotce, protože byly poskytnuty již agregované hodnoty a implementace čtení ze souboru na jednotce vyžaduje nezpracovaná data o nastavené vzorkovací frekvenci. Csv soubor *data.csv*, použitý k tomuto testu, je umístěn na přiloženém médiu v kořenovém adresáři. Každý řádek souboru obsahuje unixovou časovou značku v milisekundách a naměřenou hodnotu.

K detekci překonání nastaveného limitu hodnoty bylo využito nástroje Kapacitor, dostupného s databází InfluxDB. Při instalaci nástroje bylo postupováno podle dokumentace[4]. Po spuštění nástroje je možné ho ovládat přes webové rozhraní Chronograf. Limit poruchy byl u partnerské firmy nastaven na hodnotu 0.45, pro účely testu byla nastavena stejná hodnota. Jako způsob upozornění o překročení limitu bylo zvoleno odeslání emailu. Data byla odesílána po jedné vteřině a okamžitě po překročení nastaveného limitu byl odeslán email

[4]https://docs.influxdata.com/kapacitor/v1.5/introduction/installation/

s přednastavenou zprávou na zvolenou emailovou adresu. Stránku pro správu upozornění i s překročenou hodnotou je možné vidět na obrázku 7.7.

Test prokázal schopnost implementovaného systému upozornit uživatele při překročení nastaveného limitu jednotky.



Obrázek 7.7: Správa pravidla pro Kapacitor v aplikaci Chronograf.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat systém pro sběr a zpracování dat ze senzorů dostupných v průmyslových provozech. Výsledný systém obsahuje serverovou část, jednotku a komunikační protokoly pro zasílání dat. Serverový systém je navržen a implementován tak, aby mohl operovat pouze na jednom zařízení a zvládal přitom zpracovávat data nejméně padesáti jednotek. Zároveň, pokud je potřeba počet jednotek navýšit, je možné systém rozložit na více zařízení a jednotlivé komponenty škálovat. Serverové aplikace byly vytvořeny v jazyce C++ a jsou určeny pro operační systém Ubuntu server. Pro ukládání informací o systému je využita relační databáze a jako úložiště měřených dat je použita časová databáze. Jednotka byla poskytnuta partnerskou firmou a skládá se z počítače Raspberry Pi a rozšiřující desky s konektory pro čtyři senzory vibrací. Rozšiřující deska vzorkuje data ze senzorů o frekvenci 128000 Hz. Software jednotky, implementovaný v této práci, je tedy schopný zpracovávat až půl milionu hodnot za vteřinu. Data jsou odesílána na serverovou část dvěma způsoby. První způsob využívá navržený komunikační protokol 5.3a postavený na UDP. Zprávy obsahují filtrované, agregované hodnoty s frekvencí až sto hodnot za vteřinu, sloužící pro odhalení poruchy. Druhý způsob využívá protokol 5.3b postavený na TCP. Zprávy potom obsahují bloky nezpracovaných dat a umožňují provádění pokročilých analýz na serveru. V obou případech mohou být zprávy podle nastavení jednotky zašifrované. K zabezpečení UDP zpráv je využito metody XOR šifrování, k zabezpečení TCP spojení je využito SSL. Software jednotky je implementován v jazyce C++ a obsahuje i instalační skripty pro rychlou a bezzásahovou instalaci.

Základními požadavky na systém je jeho škálovatelnost, rychlost a modularita. Škálovatelnost serverové části je dosažena několika metodami popsanými v 7.4. Systém je implementován tak, aby veškeré komponenty bylo možné samostatně škálovat. Serverová část také není závislá na implementaci jednotky, data mohou přicházet i od jiného typu jednotek než té implementované v této práci. Musí ovšem využívat navržené komunikační protokoly. Systém je také navržen na více typů senzorů. Na historických datech partnerské firmy byla ověřena schopnost systému upozornit uživatele při poruše stroje detekované překročením nastaveného limitu u příchozích dat.

Přínos této práce je ve využití volně dostupných systémů, jako je ekosystém TICK, pro zpracování velkých dat s levnými jednodeskovými zařízeními pro sběr dat, jako je například Raspberry Pi. Propojením volně dostupných knihoven a programů je možné vytvořit škálovatelný systém pro monitorování průmyslových strojů, na kterém je možné stavět další pokročilejší funkcionalitu. Takové řešení je v porovnání s dostupnými komerčními produkty, které jsou většinou šity na míru, řádově levnější a univerzálnější. To potvrzuje i snaha part-

nerské firmy o pokračování vývoje systému navrženého a implementovaného v této práci a zájmu zákazníků o aplikaci podobných systémů v praxi (ZKL, Škoda Auto a podobně).

Literatura

[1] 4DOT. *OBRÁBĚCÍ STROJE* [online]. [cit. 2020-03-08]. Dostupné z: http://4dot.cz/obrabeci-stroje.php.

[2] ALBERTON, L. *NoSQL Databases: Why, what and when* [online]. 1. vyd. únor 2011 [cit. 2019-12-28]. Dostupné z: https://www.alberton.info/talks/show/id/7.

[3] AMLENDRA. *Ssl server client programming using openssl in c.* [cit. 2020-04-06]. Dostupné z: https://aticleworld.com/ssl-server-client-using-openssl-in-c/.

[4] APACHEHBASETEAM. *Apache HBase ™ Reference Guide* [online]. Apache hadoop, 24.3.2020 [cit. 2019-12-10]. Dostupné z: https://hbase.apache.org/book.html.

[5] BEKKER, A. *Big Data in Manufacturing: Use Cases + Guide on How To Start* [online]. Únor 2020 [cit. 2020-03-07]. Dostupné z: https://www.scnsoft.com/blog/big-data-in-manufacturing-use-cases.

[6] BORTHAKUR, D. *HDFS Architecture Guide* [online]. Apache hadoop, srpen 2019 [cit. 2019-12-03]. Dostupné z: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[7] BUYYA, R., CALHEIROS, R. N. a DASTJERDI, A. V. *Big Data: Principles and Paradigms*. 1. vyd. Cambridge: Morgan Kaufmann, červen 2016. ISBN 0128053941.

[8] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A. et al. Bigtable: A Distributed Storage System for Structured Data. In: *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 1. vyd. Seattle, VA: USENIX Association, Březen 2006, s. 205–218. ISBN 1931971471.

[9] DUNNING, T. *Time series databases : new ways to store and access data*. 1. vyd. Sebastopol, CA: O'Reilly Media, prosinec 2014. ISBN 1491914726.

[10] ECLIPSE.ORG. *Open Source Software for Industry 4.0* [online]. 2017 [cit. 2020-03-16]. Dostupné z: https://iot.eclipse.org/resources/white-papers/Eclipse%20IoT%20White%20Paper%20-%20Open%20Source%20Software%20for%20Industry%204.0.pdf.

[11] GRAFANALABS. *The analytics platform for all your metrics.* [cit. 2020-03-17]. Dostupné z: https://grafana.com/grafana/.

[12] GURU99. *What is MapReduce? How it Works - Hadoop MapReduce Tutorial* [online]. [cit. 2019-12-03]. Dostupné z: https://www.guru99.com/introduction-to-mapreduce.html.

[13] IMV. *Vibration technical guide* [online]. [cit. 2020-03-16]. Dostupné z: https://www.imv.co.jp/e/pr/vibration_measuring/chapter03/#ci.

[14] Jaloudi, S. Communication Protocols of an Industrial Internet of Things Environment: A Comparative Study. *Future Internet*. Březen 2019, sv. 11, č. 03. DOI: 10.3390/fi11030066.

[15] Kumar, A. *Difference between Row oriented and Column oriented data stores in DBMS* [online]. [cit. 2019-12-29]. Dostupné z: https://www.geeksforgeeks.org/difference-between-row-oriented-and-column-oriented-data-stores-in-dbms/.

[16] Namiot, D. Time Series Databases. In: Kalinichenko, L. A. a Starkov, S., ed. *Selected Papers of the XVII International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2015), Obninsk, Russia, October 13-16, 2015*. 1. vyd. Obninsk: CEUR-WS.org, Prosinec 2015, sv. 1536, s. 132–137 [cit. 2019-11-26]. CEUR Workshop Proceedings. ISSN 1613-0073.

[17] NI. *CompactRIO Systems* [online]. [cit. 2020-03-08]. Dostupné z: https://www.ni.com/cs-cz/shop/compactrio.html.

[18] NI. *TDMS File Format Internal Structure* [online]. [cit. 2020-02-27]. Dostupné z: https://www.ni.com/cs-cz/support/documentation/supplemental/07/tdms-file-format-internal-structure.html.

[19] Pelkonen, T., Franklin, S., Cavallaro, P., Huang, Q., Meza, J. et al. Gorilla: A Fast, Scalable, In-Memory Time Series Database. *PVLDB* [online]. 1. vyd. 2015, sv. 8, č. 12, s. 1816–1827. DOI: 10.14778/2824032.2824078. Dostupné z: http://www.vldb.org/pvldb/vol8/p1816-teller.pdf.

[20] Postel, J. *Internet Protocol* [online]. Internet Requests for Comments RFC 791. Září 1981 [cit. 2020-02-18]. Dostupné z: http://www.rfc-editor.org/rfc/rfc791.txt.

[21] Raspberry. *GPIO* [online]. Raspberry Pi Foundantion [cit. 2020-04-06]. Dostupné z: https://github.com/raspberrypi.

[22] Schaedel, M. *Batch Processing vs. Stream Processing: What's the Difference?* [online]. Březen 2018 [cit. 2020-01-04]. Dostupné z: https://www.influxdata.com/blog/batch-processing-vs-stream-processing/.

[23] Smith, S. W. *Digital signal processing : a practical guide for engineers and scientists*. 1. vyd. Boston: Newnes, listopad 2003. ISBN 075067444x.

[24] Strauch, C. *NoSQL Databases* [online]. 1. vyd. Stuttgart: Hochshule der Medien, únor 2011 [cit. 2019-12-23]. Dostupné z: https://christof-strauch.de/nosqldbs.pdf.

[25] Syeda Noor Zehra Naqvi, S. Y. *Time Series Databases and InfluxDB* [online]. 1. vyd. Brusel: Université libre de Bruxelles, prosinec 2017 [cit. 2019-12-28]. Dostupné z: https://cs.ulb.ac.be/public/_media/teaching/influxdb_2017.pdf.

[26] TEConnectivity. *Converting the signal output of a dc accelerometer to acceleration (G)* [online]. 2018 [cit. 2020-02-17]. Dostupné z: https://www.te.com/content/dam/te-com/documents/sensors/global/Converting_the_Signal_Output_of_A_DC_Accelerometer_to_Acceleration_Application%20Note.pdf.

[27] TheOpenTSDBAuthors. *OpenTSDB* [online]. [cit. 2019-12-23]. Dostupné z: http://opentsdb.net/overview.html.

[28] Timescale. *Architecture & Concepts* [online]. [cit. 2019-12-29]. Dostupné z: https://docs.timescale.com/latest/introduction/architecture.

[29] Timescale. *Benchmarking TimescaleDB vs. InfluxDB for Time-Series Data* [online]. 1. vyd. Timescale [cit. 2019-12-29]. Dostupné z: https://assets.timescale.com/whitepapers/20190610_Timescale_WhitePaper_Benchmarking_Influx.pdf.

Příloha A

Příkazy a soubory

Tato sekce obsahuje potřebné příkazy a ukázky souborů.

A.1 SSH

```
$ sudo apt install openssh-server #install ssh
$ sudo sed -i 's/#Port 22/Port 60022/g' /etc/ssh/sshd_config #change
    ssh port
$ sudo sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin no/
    g' /etc/ssh/sshd_config #disable root login
$ sudo ufw allow 60022/tcp #allow new ssh port
$ sudo systemctl restart ssh #restart ssh service
```

A.2 Potřebné knihovny pro server

```
$ sudo apt install libzip-dev
$ sudo apt install cmake
$ sudo apt install git
$ sudo apt install libmysqlcppconn-dev
$ sudo apt install pkg-config libcurlpp-dev libcurl4-openssl-dev
$ sudo apt install libboost-all-dev
$ sudo apt install libssl-dev
```

A.3 Přeložení projektů na serveru

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```