

MASARYKOVA UNIVERZITA  
FAKULTA INFORMATIKY



**Monitorovanie záťaže a  
využitia výpočetných zdrojov v  
heterogénnom výpočetnom  
prostredí**

DIPLOMOVÁ PRÁCA

**Juraj Leždík**

Brno, jeseň 2016



MASARYKOVA UNIVERZITA  
FAKULTA INFORMATIKY



# **Monitorovanie záťaže a využitia výpočetných zdrojov v heterogénnom výpočetnom prostredí**

DIPLOMOVÁ PRÁCA

**Juraj Leždík**

Brno, jeseň 2016



*Namiesto tejto stránky vložte kópiu oficiálneho podpísaného zadania práce a prehlásenie autora školského diela.*



## **Prehlásenie**

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Juraj Leždík

**Vedúci práce:** Mgr. Miroslav Ruda





## **Podakovanie**

Touto cestou by som sa rád poďakoval vedúcemu diplomovej práce Mgr. Miroslavovi Rudovi za umožnenie vypracovania tohto zadania ako aj za jeho čas a odborné vedenie práce. Špeciálna vďaka patrí Mgr. Borisovi Parákovi za jeho trpezlivosť, názory a cenné rady pri vypracovávaní. Osobitné poďakovanie patrí aj mojej rodine, priateľom a Nicol za podporu.

## Zhrnutie

Práca sa zaoberá zberom metrík o vyťaženie zdrojov v heterogénnej infraštruktúre MetaCentra. Popisuje koncepty poskytovania výpočetných zdrojov v rámci heterogénnej infraštruktúry. Venuje sa všeobecným problémom monitorovania, akými sú proces zberu metrík, ukladanie a správa uložených hodnôt. Identifikuje metriky a popisné dáta relevantné pre vyťaženie zdrojov. Cieľom práce je návrh monitorovacieho riešenia pre zadanú infraštruktúru a popis jeho implementácie a testovania.

## **Klíčové slová**

cloud, monitorovanie, metriky, MetaCentrum, heterogénna infraštruktúra,  
Hadoop, PBS Professional, Docker, libvirt



# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
<b>2</b>	<b>Heterogénna infraštruktúra</b>	<b>3</b>
2.1	<i>MetaCentrum</i>	3
2.2	<i>Kontrola zdrojov infraštruktúry</i>	4
2.3	<i>Cloud</i>	4
2.4	<i>Realizácia konceptov cloudu</i>	5
2.4.1	<i>Virtuálne stroje</i>	6
2.4.2	<i>Aplikačné kontajnery</i>	6
2.5	<i>MapReduce aplikačné prostredia</i>	7
2.6	<i>Gridové počítanie</i>	8
2.7	<i>Technológie využívané v MetaCentre</i>	8
2.7.1	<i>libvirt/KVM</i>	8
2.7.2	<i>Docker</i>	9
2.7.3	<i>Hadoop</i>	9
2.7.4	<i>PBS Professional</i>	11
<b>3</b>	<b>Zber a uchovávanie monitorovacích dát</b>	<b>13</b>
3.1	<i>Všeobecné problémy monitorovania</i>	14
3.1.1	<i>Identifikácia relevantných metrík</i>	14
3.1.2	<i>Analýza v reálnom čase a varovania</i>	14
3.1.3	<i>Meracie intervaly</i>	14
3.1.4	<i>Množstvo zdrojov a ich identifikácia</i>	15
3.1.5	<i>Ukladanie dát</i>	15
3.1.6	<i>Vizualizácia a analýza dát</i>	15
3.1.7	<i>Monitorovanie v heterogénnom výpočtovom prostredí</i>	16
3.2	<i>Časové rady</i>	17
3.2.1	<i>Analýza časových rád</i>	17
3.2.2	<i>Úprava historických dát vzhľadom na kapacitu</i>	18
<b>4</b>	<b>Aktuálne monitorovacie riešenia</b>	<b>19</b>
4.1	<i>Monitorovacie riešenia</i>	19
4.1.1	<i>Nagios</i>	19
4.1.2	<i>Icinga</i>	20
4.1.3	<i>Zabbix</i>	21
4.1.4	<i>Ganglia</i>	22
4.1.5	<i>AppDynamics</i>	23
4.2	<i>Ďalšie nástroje</i>	24

4.2.1	Linux cgroups . . . . .	24
4.2.2	collectd . . . . .	25
4.3	<i>Databázy časových rád</i> . . . . .	26
4.3.1	OpenTSDB . . . . .	26
4.3.2	InfluxDB . . . . .	27
4.3.3	RRDTool . . . . .	28
4.4	<i>Vhodnosť dostupného softvéru</i> . . . . .	29
<b>5</b>	<b>Metriky</b>	<b>31</b>
5.0.1	Význam popisných údajov metrik . . . . .	31
5.0.2	Formát hodnoty metriky . . . . .	32
5.0.3	Výpočet percentuálnej záťaže na procesor . . . . .	32
5.1	<i>libvirt/KVM</i> . . . . .	32
5.1.1	Procesor . . . . .	32
5.1.2	Pamäť . . . . .	33
5.1.3	Zápis a čítanie z disku . . . . .	34
5.1.4	Sieť . . . . .	34
5.1.5	Popisné údaje . . . . .	34
5.2	<i>Docker</i> . . . . .	35
5.2.1	Procesor . . . . .	35
5.2.2	Pamäť . . . . .	35
5.2.3	Sieť . . . . .	36
5.2.4	Zápis a čítanie z disku . . . . .	36
5.2.5	Popisné údaje . . . . .	37
5.3	<i>Hadoop</i> . . . . .	37
5.3.1	Metriky clusteru . . . . .	37
5.3.2	Metriky aplikácií . . . . .	38
5.3.3	Metriky uzlov . . . . .	39
5.4	<i>PBS Professional</i> . . . . .	40
5.4.1	Metriky úloh . . . . .	40
<b>6</b>	<b>Analýza a návrh</b>	<b>41</b>
6.1	<i>Požiadavky na aplikáciu</i> . . . . .	41
6.1.1	Vysoký monitorovací výkon . . . . .	41
6.1.2	Nízka nadbytočná záťaž . . . . .	41
6.1.3	Škálovateľnosť . . . . .	41
6.2	<i>Miesto nasadenia zbernej aplikácie</i> . . . . .	42
6.3	<i>Návrh monitorovacieho riešenia</i> . . . . .	42
6.3.1	Paralelizácia v rámci pluginov . . . . .	44
6.3.2	Databáza časových rád . . . . .	44

6.4	<i>Zmena granularity dát</i>	45
6.4.1	Kontinuálne a periodické zmeny granularity	45
6.4.2	Zmena granularity a počet databáz	46
6.4.3	Návrh nástroja na zmenu granularity dát	46
<b>7</b>	<b>Implementácia</b>	<b>49</b>
7.1	<i>Aplikácia na zber metrických dát</i>	49
7.1.1	Zapisovací plugin WriteTSDB	49
7.2	<i>Techniky zbierania metrík</i>	49
7.2.1	Docker	49
7.2.2	libvirt/KVM	50
7.2.3	Hadoop	50
7.2.4	PBS Professional	50
7.3	<i>Knižnice využité pri zbere metrík</i>	51
7.3.1	libcurl	51
7.3.2	libpthread	51
7.3.3	cJSON	51
7.4	<i>Agregácia historických dát v OpenTSDB</i>	52
7.5	<i>Zostavenie a nasadenie riešenia</i>	52
<b>8</b>	<b>Testovanie</b>	<b>53</b>
8.1	<i>Zber metrík</i>	53
8.1.1	libvirt	53
8.1.2	Docker	53
8.1.3	Hadoop	53
8.1.4	PBS Professional	54
8.2	<i>Databáza časových rád</i>	54
8.2.1	Úprava historických dát	54
<b>9</b>	<b>Záver</b>	<b>57</b>
	<b>Bibliografia</b>	<b>59</b>
<b>A</b>	<b>Prevodné tabuľky názvov metrík</b>	<b>63</b>
A.1	<i>libvirt</i>	63
A.1.1	Metriky	63
A.1.2	Popisné údaje	63
A.2	<i>Docker</i>	64
A.2.1	Metriky	64
A.2.2	Popisné údaje	64
A.3	<i>Hadoop Cluster</i>	65

A.3.1	Metriky . . . . .	65
A.3.2	Popisné údaje . . . . .	65
A.4	<i>Hadoop Applications</i> . . . . .	66
A.4.1	Metriky . . . . .	66
A.4.2	Popisné údaje . . . . .	66
A.5	<i>Hadoop Node</i> . . . . .	66
A.5.1	Metriky . . . . .	66
A.5.2	Popisné údaje . . . . .	67
A.6	<i>PBS Professional</i> . . . . .	67
A.6.1	Metriky . . . . .	67
A.6.2	Popisné údaje . . . . .	67



## **Zoznam tabuliek**



## Zoznam obrázkov

2.1	Infraštruktúra MetaCentra v ČR	4
2.2	Porovnanie architektúry kontajnerov a virtuálnych strojov s natívnym hypervízorom	7
2.3	Architektúra Hadoop YARN	10
2.4	Architektúra PBS Professional	11
4.1	Architektúra OpenTSDB	27
4.2	Vizualizácia časových rád OpenTSDB nástrojom Grafana	28
6.1	Architektúra monitorovacieho riešenia	43
8.1	Pôvodné a agregované dáta - vizualizácia pomocou Grafany	55



## 1 Úvod

Heterogénna infraštruktúra MetaCentra poskytuje výpočtové prostriedky pre mnohé vedecké a výskumné organizácie. Ukladajú a spracovávajú sa v nej veľké objemy dát. V súčasnosti infraštruktúru tvorí približne 700 uzlov [1], ktoré sú rozmiestnené naprieč celou Českou republikou. Podstatou fungovania heterogénnej infraštruktúry je zdieľanie veľkého výpočtového výkonu viacerými subjektami, pre ktoré by zadováženie a prevádzkovanie vlastnej infraštruktúry predstavovalo neúmernú ekonomickú, personálnu a prevádzkovú záťaž. Princípom zdieľania je, že prostriedky by mali byť využívané férovo a miera využívania by mala byť kontrolovateľná. Nemalo by dochádzať k tomu, že jeden užívateľ vyťaží zdroje natoľko, že výpočtové úlohy ostatných dostanú nepomerne malý priestor. Aby sa tomu dalo predísť, je potrebné monitorovať využitie výpočtového výkonu. Ak máme informácie o tom, kto koľko využíval zdroje, je možné toto využitie účtovať a zároveň do budúcnosti primerane regulovať. Monitorovanie má využitie aj pri plánovaní odstávok, vylepšovania infraštruktúry, kedy sa na základe historických dát dá odhadnúť miera vyťaženia v budúcnosti.

Prostredie MetaCentra predstavuje heterogénnu infraštruktúru. Na riešenie výpočtových úloh sa používajú rôzne technológie, ktoré inak pristupujú k výpočtovým zdrojom. Uzly, na ktorých sa vykonávajú výpočty, disponujú rôznymi typmi zdrojov. Každá z technológií prevádzkovaných MetaCentrom poskytuje všetky, alebo aspoň časť údajov o vyťažení zdrojov ako sú procesor, pamäť, disk a sieťové zariadenia. Mojou úlohou je identifikácia metrík podstatných pre monitorovanie zdrojov a vybratie takých metrík, ktoré je možné porovnať s metrikami ostatných technológií. Takto môžeme dostať celkový obraz o tom, ako rôzne technológie využívajú jednu skupinu zdrojov a len takto je možné vzájomne tieto technológie porovnávať.

Aby bolo možné správne vyvodzovať závery o zaťažení infraštruktúry, je potrebné o tom zberať údaje v pravidelných intervaloch. Každému intervalu prináleží hodnota, ktorá vypovedá o využití prostriedkov za uplynutý čas. Takéto dáta sa nazývajú časové rady.

Vzhľadom na veľkosť infraštruktúry zber týchto metrík vytvára nezanedbateľné požiadavky na úložný priestor. V súčasnosti existujú špecializované databázy na ukladanie časových rád. Tie sa snažia rôznymi technikami znižovať kapacitu potrebnú na ukladanie časových dát. Jedným spôsobom je agregácia viacerých historických hodnôt do jednej. Tým dochádza k spomaleniu rastu databázy v čase. Zároveň dochádza k zrýchleniu vyhodnocovania požiadaviek, ktoré zahŕňajú staré dáta. Tým, že sú hodnoty agregované,

## 1. ÚVOD

---

je odpoveď vygenerovaná rýchlejšie.

Cieľom práce je poskytnúť mechanizmus na dlhodobý zber metrík, ktorý je škálovateľný a predstavuje čo najmenšiu dodatočnú záťaž pre infraštruktúru. Metrické údaje sú základom pre nasledovné účtovanie využitia zdrojov, ktoré pre potreby MetaCentra spracoval Michal Kimle vo svojej diplomovej práci Flexibilné účtovanie výpočtových zdrojov pre heterogénne výpočtové prostredie[2].

Zároveň sa v práci zaoberám problematikou agregácie historických dát v databáze časových rád, a jedným z cieľov je navrhnúť a implementovať nástroj na vykonávanie týchto agregácií. Pri riešení daných problémov zohľadňujem už používané nástroje v MetaCentre, kde napr. v prípade agregácie historických dát sa jedná o rozšírenie už nasadenej databázy o túto funkcionality.

Práca je rozdelená do deviatich kapitol vrátane úvodu a záveru. V úvode je stručne popísaná problematika monitorovania a ukladania monitorovacích dát a nadväznosť práce na už existujúce riešenia. V druhej kapitole sa venujem popisu infraštruktúry MetaCentra a jednotlivým výpočtovým technológiám, ktoré prevádzkuje. V tretej kapitole popisujem všeobecné princípy zberu monitorovacích dát a ich ukladania. Štvrtá kapitola predstavuje prehľad aktuálne dostupných monitorovacích riešení. V piatej kapitole sa venujem metrikám, ktoré budem zberať. Šiesta kapitola predstavuje analýzu požiadaviek a návrh celkového monitorovacieho riešenia. V siedmej kapitole popisujem nástroje a knižnice použité pri implementácii navrhnutého riešenia. Osmá kapitola sa venuje testovaniu naimplementovaného riešenia z hľadiska zberu metrík a agregácie historických dát. Poslednou kapitolou je záver, v ktorom hodnotím dosiahnuté výsledky a načrtávam smer ďalšieho vývoja.

## 2 Heterogénna infraštruktúra

Heterogénna infraštruktúra obsahuje rôzne typy uzlov, od jednotlivých pracovných staníc, ktoré poskytujú len časť svojho výkonu, až po dedikované clustre, zložené z viacerých úzko prepojených uzlov. Ďalšou črtou je viacero spôsobov, akým sa sprístupňuje výpočtový výkon. Prvým spôsobom sú *cloudové služby*. Jedná sa o virtualizované prostredie, kde na jednom fyzickom stroji môže byť súbežne spustených viacero nezávislých užívateľských strojov.

Druhým spôsobom je *grid*. Grid spája mnoho uzlov aj s rôznym geografickým umiestnením do jedného výpočtového uzla, ktorý rieši náročné výpočtové úlohy. Uzly môžu poskytovať celý svoj výpočtový výkon alebo len jeho časť.

### 2.1 MetaCentrum

Projekt MetaCentrum je príkladom heterogénnej infraštruktúry. Vznikol v roku 1996 a od roku 1999 je jeho činnosť zastrešovaná organizáciou CESNET. Zaoberá sa budovaním národnej gridovej infraštruktúry a prepojením s podobnými zahraničnými projektami. Projekt je oficiálnou súčasťou Európskej gridovej iniciatívy (EGI). Úlohou MetaCentra je predovšetkým koordinácia a rozširovanie infraštruktúry či už o vlastné zdroje alebo prostredníctvom partnerov, ktorý poskytujú výpočtový výkon svojich clusterov. Jedná sa hlavne o akademickú spoluprácu. MetaCentrum spravuje výpočtové prostriedky a dátové úložiská AV, JČU, MU, MZLU, UK, VUT, ZČU. V súčasnosti infraštruktúra disponuje viac než 13400 procesorovými jadrami [1]. Obrázok 2.1 ilustruje infraštruktúru MetaCentra: MetaCentrum primárne poskytuje svoj výpočtový výkon a úložnú kapacitu. Taktiež sprístupňuje svoje programové vybavenie a vývojové prostredie a hlavne množstvo aplikácií využívaných na výskumné účely, ako napr. Ansys, Gaussian, Matlab, Mathematica. Taktiež sa venuje vývoju v oblasti gridového a cloudového počítania, napr. v oblasti plánovania, gridového middleware, optimalizácie a paralelizácie výpočtov a virtualizácie infraštruktúry. Dôležitou funkciou je účasť na medzinárodných projektoch, využívanie medzinárodnej výpočtovej infraštruktúry a využívanie skúseností na rozvoj v domácom prostredí.[3]



Obr. 2.1: Infraštruktúra MetaCentra v ČR

### 2.2 Kontrola zdrojov infraštruktúry

Bez ohľadu na to, aký prístup si užívateľ zvolí k využívaniu zdieľaných výpočtových prostriedkov, jeho výpočty sa v konečnom dôsledku budú musieť realizovať na fyzickom hardvéri poskytovateľa. Aj keď sa daná úloha vypočítava na viacerých uzloch a rozličnými postupmi, vlastník by mal mať možnosť nejakým jednotným spôsobom určiť, ako je reálne celá infraštruktúra vyťažovaná.

### 2.3 Cloud

Cloud predstavuje virtualizovanú výpočtovú infraštruktúru, ktorej fyzické výpočtové prostriedky sú zdieľané viacerými užívateľmi. Pre cloud sú typické dva koncepty [4]:

**Abstrakcia:** Podrobnosti o implementácii systému sú abstrahované užívateľom a vývojárom. Aplikácie bežia na fyzických systémoch, ktoré nie sú špecifikované, dáta sú uložené na neznámych miestach, administrácia systémov je odovzdaná iným a užívatelia majú všestranný prístup.

**Virtualizácia:** Cloud virtualizuje systémy zdieľaním zdrojov. Systémy a úložný priestor môže byť poskytovaný podľa potrieb z centralizovanej infra-



štruktúry, ceny sú stanovené na základe meraní, je možný prenájom viacerým subjektom a zdroje sú škálovateľné podľa aktivity.

Prevádzkovateľ cloudu môže zvoliť rôzne prístupy k tomu, akým spôsobom bude poskytovať svoje zdroje[4]:

***Infrastructure as a Service (IaaS):*** Užívateľ má možnosť využívať zdroje cloudu podľa svojich hardvérových požiadaviek. Môže si presne určiť koľko pamäte, procesorov alebo diskovej kapacity požaduje. Poskytovateľ služby spravuje hardvérovú infraštruktúru, kým užívateľ je zodpovedný za ostatné aspekty nasadenia. To zahŕňa operačný systém, aplikácie a užívateľskú interakciu so systémom.

***Platform as a Service (PaaS):*** Prevádzkovateľ služby poskytuje hardvér, operačný systém, aplikácie, služby, vývojové prostredia a kontrolné štruktúry. Užívateľ môže nasadiť vlastné aplikácie na cloudovú infraštruktúru alebo využívať aplikácie, ktoré boli naprogramované jazykmi a nástrojmi podporovanými prevádzkovateľom.

***Software as a Service (SaaS):*** Tento model sprístupňuje užívateľovi konkrétnu aplikáciu prostredníctvom tenkého klientskeho rozhrania (zväčša cez webový prehliadač) a zodpovednosť zákazníka spočíva len v spracovávaní svojich dát a v interakciách s užívateľským systémom.

Cloud umožňuje jednoducho spravovať a meniť výpočtový výkon, ktorý má užívateľ k dispozícii. Má to význam, ak sa rozrastú požiadavky užívateľa, alebo naopak z dôvodu obmedzenej prevádzky či nedostatku výpočtových úloh sa nároky môžu zmenšiť. Užívateľ môže dané prostriedky využívať hneď, bez veľkých počiatočných investícií, ktoré by si buď nemohol dovoliť, alebo ak sú výpočtové úlohy krátkodobejšieho rázu, nákup stroja s požadovaným výkonom by bol nevhodnou investíciou. Užívateľ sa nemusí starať o nákup hardvéru, jeho zostavovanie do funkčných serverov a ich následné rozširovanie a správu. Tieto služby zabezpečuje prevádzkovateľ cloudovej infraštruktúry. Pre neho je zase dôležité mať prehľad o tom, ako sú jeho inštalované kapacity využívané. Či už z pohľadu skvalitňovania vlastných poskytovaných služieb alebo vo vzťahu k užívateľovi a k tomu, v akej miere spotrebovávajú poskytovaný výkon.

### 2.4 Realizácia konceptov cloudu

Prostredie cloudu má za cieľ poskytovať užívateľom prostriedky variabilne vzhľadom na ich aktuálne požiadavky. Druhým dôležitým aspektom je izolovanosť jednotlivých užívateľov v rámci celej infraštruktúry, aby prípadné

problémy so systémom jedného užívateľa neohrozili prevádzku systémov ostatných užívateľov.

### 2.4.1 Virtuálne stroje

Virtuálne stroje poskytujú úplnú virtualizáciu fyzickej hardvérovej štruktúry. Na jednom hosťujúcom počítači môže byť spustených viacero virtuálnych strojov. Každý má svoj vlastný virtuálny procesor, pamäť, grafický procesor, pevný disk a periférie. Operačný systém spustený vo virtuálnom stroji je izolovaný od hosťovského operačného systému (ak ho hosťovský počítač má).

#### Hypervízor

Hypervízor je softvér, ktorý poskytuje virtualizačné nástroje potrebné na ich izolovaný beh v rámci jedného fyzického stroja. Rozlišujeme 2 typy [5]:

**natívny** - na hosťujúcom počítači nie je nainštalovaný žiadny operačný systém. Hypervízor spravuje hardvér hosťujúceho počítača a kontroluje beh operačných systémov, ktoré sa javia ako procesy. Príkladom je VMware ESX/ESXi<sup>1</sup>, Oracle VM Server for x86<sup>2</sup>, Citrix XenServer<sup>3</sup> alebo KVM<sup>4</sup>.

**hosťovaný** - hypervízor je spustený ako program v operačnom systéme hosťujúceho počítača. Príkladom je QEMU, VMware Workstation alebo VirtualBox.

V prostredí MetaCentra sa používa na virtualizáciu hypervízor KVM. Bližšie ho popisujem v sekcii 2.7.1.

### 2.4.2 Aplikačné kontajnery

Kontajnery predstavujú odlišný prístup k virtualizácii ako virtuálne stroje. Tiež ide o snahu spúšťať softvér v prostredí oddelenom od skutočného hardvéru a operačného systému. Na rozdiel od úplných virtuálnych strojov nie je virtualizovaný celý hardvér, ale len softvérové vybavenie nevyhnutné na spustenie programu. Rozdiel v architektúre ilustruje obrázok 2.2. Kontajnery zdieľajú jadro operačného systému hosťovského počítača a jeho zdroje.

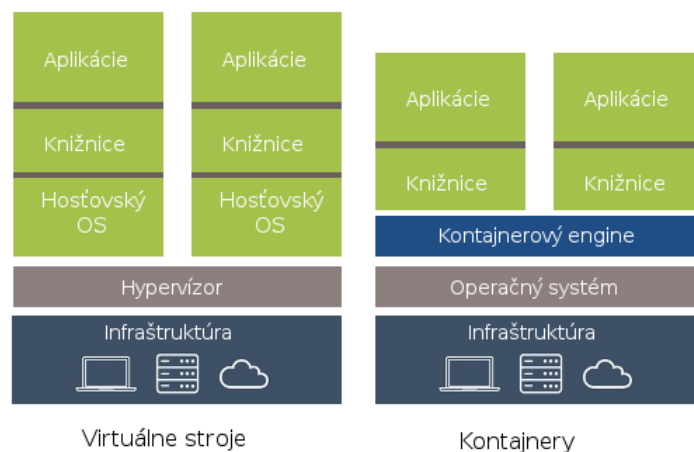
---

1. <http://www.vmware.com/>

2. <https://www.oracle.com/virtualization/vm-server-for-x86/index.html>

3. <https://www.citrix.com/products/xenserver/>

4. Kernel-based Virtual Machine



Obr. 2.2: Porovnanie architektúry kontajnerov a virtuálnych strojov s natívnym hypervízorom

Jednotlivé kontajnery dostávajú kontrolovaný prístup k procesoru, pamäti, úložnej kapacite, sieti. Takýto spôsob virtualizácie predstavuje zníženú záťaž na hostujúci počítač, pretože nie je virtualizovaný celý operačný systém a ani hardvér.

Kontajnery môžu byť spustené aj v rámci virtuálnych strojov. Cieľom tejto práce ale nie je monitorovanie takto vnorených kontajnerov.

V prostredí MetaCentra je ako kontajnerový engine testovaný softvér Docker[**Docker**].

## 2.5 MapReduce aplikačné prostredia

MapReduce predstavuje prístup k riešeniu problémov, ktoré operujú nad veľkým objemom dát v rádoch stoviek a tisícok gigabajtov[6]. Je to spôsob distribuovaného a paralelizovaného riešenia problémov, kedy dáta, nad ktorými sa uskutočňuje výpočet, sú tiež uložené distribuovane. MapReduce rozdeľuje problém na dve fázy: fáza mapovania a fáza redukovania. Každá fáza má svoje vstupné a výstupné dáta. Vo fáze mapovania dochádza k príprave dát. Vstup je spracovaný do takej štruktúry, nad ktorou prebehne výpočet. Vo fáze redukovania je spracovaný výstup fázy mapovania a vytvorené čiastkové výsledky. Následne je z vytvorených celkový výsledok. Rozdelenie problémov do týchto dvoch fáz potom umožňuje jednotlivé čiastkové

problémy riešiť na viacerých uzloch paralelne nad podmnožinou vstupných dát.

### 2.6 Gridové počítanie

Grid je výpočtový model, ktorý sprístupňuje zdroje, kontrolované viacerými nezávislými subjektami. Subjekty sa často nachádzajú v rôznych mestách alebo štátoch. Existujú aj gridy, ktorých uzly sú rozložené po celom svete (napr. BOINC<sup>5</sup>). Jednotlivé subjekty na komunikáciu využívajú bežné komunikačné protokoly, napr. Ethernet. Uzly jednotlivých subjektov kooperovane poskytujú svoj výpočtový výkon. Grid spája svoje zdroje na vyriešenie jedného veľkého výpočtového problému[7]. Využíva na to vrstvu middleware, ktorá rozdeľuje uzlom vykonávanie jednotlivých častí programu. V MapReduce aplikačnom prostredí sú uzly bližšie prepojené a dáta sú distribuovane uložené. V gridových systémoch dochádza k pomerne veľkej výmene dát medzi uzlami, kedy výpočet často čaká, kým obdrží dáta zo siete[6].

### 2.7 Technológie využívané v MetaCentre

V prostredí MetaCentra sú nasadené do produkčnej prevádzky softvéry, ktoré umožňujú využívať výpočtový výkon spomínanými technológiami. Jedná sa o heterogénnu infraštruktúru, ktorú tvoria výkonné clustre ale aj jednotlivé uzly rozmiestnené po celej republike. Virtuálne stroje sú poskytované prostredníctvom linuxového modulu jadra KVM<sup>6</sup>[8]. Na prácu s nimi je využívaná knižnica *libvirt*[9]. Na virtualizáciu v podobe kontajnerov je nasadený softvér *Docker*[10]. Technológiu distribuovaného počítania v podobe MapReduce aplikácií zabezpečuje *Apache Hadoop*[6]. Koordináciu gridových výpočtov zabezpečuje plánovač *PBS Professional*<sup>7</sup>. V nasledujúcich sekciách sa podrobnejšie venujem popisu jednotlivých softvérov.

#### 2.7.1 libvirt/KVM

KVM je plné virtualizačné riešenie pre Linux na x86 hardvér, obsahujúce virtualizačné rozšírenia (Intel VT alebo AMD-V). Pozostáva z nahrateľného modulu jadra, *kvm.ko*, ktorý poskytuje základ virtualizačnej infraštruktúry, a špecifické moduly, *kvm-intel.ko* alebo *kvm-amd.ko*, ktoré zabezpečujú virtualizáciu procesora od daného výrobcu. Je možné virtualizovať obrazy s

---

5. Berkeley Open Infrastructure for Network Computing - <https://boinc.berkeley.edu/>

6. Kernel Virtualization Module

7. <http://www.pbsworks.com/>

operačnými systémami Linux, Windows, OS X alebo inými. Každý virtuálny stroj má vlastný virtualizovaný hardvér: procesor, pamäť, sieťovú kartu, disk, grafický adaptér atď.

KVM je open-source softvér. Virtualizačný modul jadra sa nachádza v Linuxe od verzie 2.6.20.[8]

libvirt je sada nástrojov na prácu s virtualizačnými schopnosťami Linuxu (a ostatných OS). Je to voľný softvér dostupný pod licenciou GNU LGPL. Obsahuje API v jazyku C a väzby pre bežné programovacie jazyky, ako napr. Python, C#, Java, Ruby alebo Go.[9]

Knižnica libvirt predstavuje vrstvu abstrakcie, ktorá zjednocuje prácu s hypervízormi od viacerých výrobcov. Umožňuje vytvoriť a zrušiť virtuálny stroj, spúšťať ho, reštartovať a vypínať, meniť jeho konfiguráciu a pridelené zdroje a tiež zisťovať údaje o využití zdrojov.

### 2.7.2 Docker

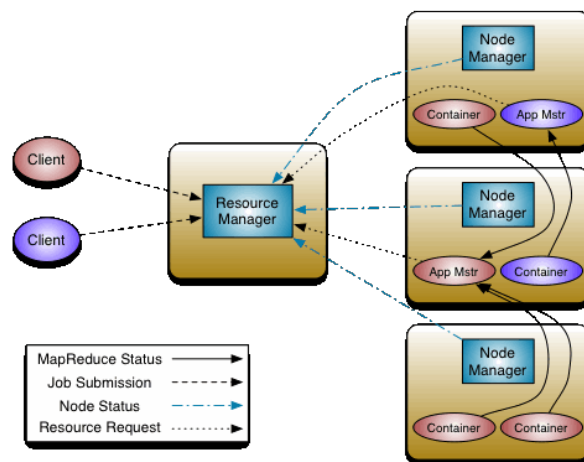
Docker predstavuje kontajnerový engine a sadu nástrojov, ktorá umožňuje zabaliť aplikáciu so všetkými jej závislosťami do kompaktnej jednotky (tzv. kontajneru) určenej na softvérový vývoj. Kontajnery Dockeru obaľujú softvér kompletným súborovým systémom, ktorý zahŕňa všetko, čo daný softvér potrebuje na spustenie: kód, nástroje potrebné na beh, systémové nástroje a knižnice. Toto zaručuje, že program bude pracovať rovnako bez ohľadu na prostredie, v ktorom je spustený.[10] Pri behu jednotlivé kontajnery zdieľajú jadro operačného systému.

Docker rozlišuje kontajnery a obrazy. Obraz predstavuje softvérový balík so všetkými závislosťami, kontajner reprezentuje jeho beh. Preto môže naraz existovať viacero kontajnerov, v ktorých je spustený ten istý softvér.

### 2.7.3 Hadoop

Projekt Apache Hadoop vyvíja open-source softvér na spoľahlivé, škálovateľné, distribuované výpočty. Umožňuje distribuovane spracovávať veľké objemy dát naprieč clustermi, používajúcimi jednoduché programovacie modely. Je navrhnutý tak, aby bol škálovateľný od jednotlivých serverov po tisíce strojov, kde každý poskytuje lokálny výpočtový výkon a úložný priestor. Nespolieha sa na vysokú dostupnosť hardvérových prostriedkov, ale je navrhnutý, aby detekoval a zvládal chyby na aplikačnej vrstve, takže poskytuje vysoko dostupnú službu nad clusterom počítačov, z ktorých každý je náchylný na chyby. [6]

Hadoop využíva MapReduce aplikačný model. Vývojárom poskytuje prog-



Obr. 2.3: Architektúra Hadoop YARN  
[11]

ramovacie prostredie, ktoré uľahčuje výmenu dát medzi jednotlivými fázami výpočtovej úlohy.

Projekt pozostáva z týchto modulov:

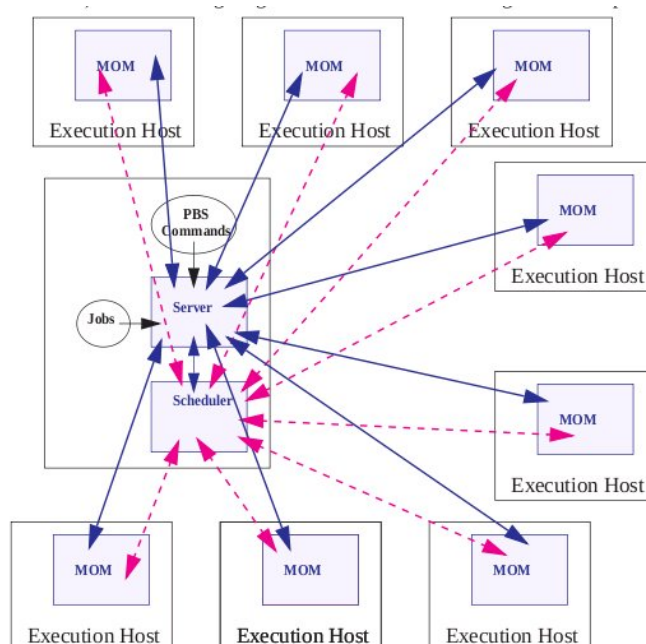
**Hadoop Common:** spoločné nástroje, ktoré podporujú ostatné Hadoop moduly

**Hadoop Distributed File System (HDFS<sup>TM</sup>):** distribuovaný súborový systém, ktorý poskytuje vysokú priepustnosť

**Hadoop YARN:** prostredie pre plánovanie úloh a správu zdrojov clusteru

**Hadoop MapReduce:** systém založený na YARN pre paralelné spracovávanie veľkých množstiev dát, prostredie pre plánovanie úloh a správu zdrojov clusteru

Nasledujúci obrázok ukazuje výpočetnú architektúru YARN-u. Komponent Resource Manager má prehľad o všetkých zdrojoch clusteru a prideľuje ich aplikáciám. Na jednotlivých uzloch Node Manager alokuje kontajnery, v ktorých sa vykonáva aplikácia. Jej beh riadi komponent ApplicationMaster, ktorý tiež vyjednáva o zdrojoch s Resource Managerom.



Obr. 2.4: Architektúra PBS Professional  
[11]

#### 2.7.4 PBS Professional

Po konzultácii so školiteľom bola jedna z platforiem v zadaní Torque nahradená platformou PBS Professional<sup>8</sup>[12]. PBS Professional je odvodený od systému Torque a takisto sa jedná o gridový plánovací softvér. PBS si udržuje prehľad o podliehajúcej infraštruktúre. Má k dispozícii zoznam dostupného hardvéru, softvérových licencií a informácie o čakajúcich a spustených úlohách. Každá úloha má definované požiadavky na zdroje, softvér a dobu, do kedy by mala byť úloha vypočítaná. PBS pridelí úlohu na stroj, ktorý vyhovuje týmto požiadavkám. Plánovač čaká, kým je požadovaný stroj dostupný a pri vykonávaní úlohy jej poskytne vstup a uloží výstup na požadovanom mieste.

8. Portable Batch System





### 3 Zber a uchovávanie monitorovacích dát

Monitorovanie akejkoľvek (nielen počítačovej) prevádzky je dôležitá oblasť, ktorá má význam pre jej správne fungovanie, správu, zdokonaľovanie a servis. Monitorovanie nám poskytuje dáta, ktoré majú viacero spôsobov využitia:

**Prehľad o aktuálnom vyťažení zdrojov** Pri zbere dát toto zodpovedá meraniu konkrétnej hodnoty sledovaného parametru. Kvantifikáciou sledovaného zdroja dostaneme predstavu o tom, ako je používaný "teraz", čiže v prítomnosti. Môžeme tak detegovať prípadné preťaženie zdroja, jeho dostupnosť alebo zlyhanie.

**Prehľad o vyťažení zdrojov za určitú dobu** Vyťaženie zdrojov sa v priebehu času mení. Nielen z krátkodobého hľadiska, kedy napríklad prebieha výpočet náročných úloh len v určitých hodinách, ale aj zo strednodobého (v období letných prázdnin je infraštruktúra vyťažovaná menej) a dlhodobého hľadiska (požiadavky na výkon sa zväčšujú). Uchovávanie monitorovacích dát predstavuje kľúčovú požiadavku, aby sme mohli sledovať, aké trendy vo využívaní mali jednotlivé sledované zdroje v priebehu nejakej doby. Z takto nahromadených dát môžeme vypočítavať rôzne štatistiky a ďalej ich analyzovať.

**Prehľad o vyťažení zdrojov podľa parametrov** Vyťaženie zdrojov predstavuje určitú skalárnu hodnotu či už pre jednotlivé uzly alebo naprieč celou infraštruktúrou. Niekedy je však potrebné zistiť, ako boli využívané zdroje len v určitej oblasti cloudu, alebo napr. ako boli využívané disky s kapacitou nad 1 TB. Okrem jednoduchého zberu dát o vyťažení by monitorovacie riešenie malo poskytnúť aj ďalšie informácie, na základe ktorých je potom možné selektívne určovať vyťaženie zdrojov. To je možné dosiahnuť zberom doplnujúcich dát pripísaných jednotlivým metrikám.

**Prehľad o stave zdrojov z hľadiska správy a plánovania** Z monitorovacích dát vieme indentifikovať nefunkčný zdroj a nahradiť ho novým. V prípade vysokého vyťaženia zas môžeme vyvodiť záver, že zdroje už nie sú dostačujúce a je potrebné ich nejakým spôsobom rozšíriť, prípadne zlepšiť efektivitu ich využívania. Takisto vieme do určitej miery predpovedať, ako budú zdroje v budúcnosti využívané, čo je podstatné pri samotnom plánovaní odstávok, servisnej činnosti alebo zväčšovaní infraštruktúry.

**Účtovanie vyťaženia zdrojov** Dáta o používaní zdrojov sú jediným možným spôsobom ako rozumne stanoviť cenu za používané zdroje. Tak tiež na ich základe môžeme sledovať činnosť jednotlivých užívateľov v systémoch a stanoviť im akceptovateľné podmienky na využívanie služieb.

Problematicke výpočtu štatistík a účtovaniu vyťaženia zdrojov v prostredí MetaCentra sa venuje diplomová práca Michala Kimleho[2]. Pre výpočet štatistík používa metrické údaje spolu s ďalšími identifikačnými údajmi. Okrem monitorovania infraštruktúry poslúžia mnou zberané metriky na výpočet týchto štatistík.

#### 3.1 Všeobecné problémy monitorovania

Problematicku monitorovania je možné rozdeliť na viacero podproblémov, nájsť pre ne vhodné riešenia a spojiť ich tak do funkčného celku.

##### 3.1.1 Identifikácia relevantných metrík

V prvom rade je potrebné identifikovať, čo vlastne potrebujeme pre konkrétny systém monitorovať. Jedná sa o určenie zdrojov kľúčových pre dané prostredie. V tomto prípade sú to údaje o vyťažení procesorov, pamätí, sieťových rozhraní a diskov. Niektoré údaje, ktoré potrebujeme vedieť, je možné odvodiť z iných už nameraných hodnôt, napr. percentuálna miera vyťaženia procesora.

##### 3.1.2 Analýza v reálnom čase a varovania

Namerané hodnoty metrík je okrem neskoršej štatistickej analýzy potrebné sledovať a analyzovať v reálnom čase. Metrika môže mať svoje kritické hodnoty. Sú to hodnoty, ktoré naznačujú, že daný zdroj sa vymyká bežným očakávaniam o využití. Na toto je možné reagovať. Či už sa jedná o zapísanie hlásenia do logu, zobrazenie varovania alebo prípadné zaslanie emailu či SMS správy s popisom udalosti.

##### 3.1.3 Meracie intervaly

Ďalším čiastkovým problémom je granularita metrík. Zdroje sú kontinuálne využívané v čase. Tieto dáta je potrebné nejakým spôsobom digitalizovať. V pravidelných intervaloch sa pozeráme na zdroje a určujeme ich vyťaženie v priebehu daného intervalu. Rôzne zdroje sa menia v čase rôznou intenzitou, je preto dôležité určiť aj periodicitu zberu jednotlivých metrík.

#### 3.1.4 Množstvo zdrojov a ich identifikácia

Množstvo zdrojov predstavuje ďalšiu oblasť, ktorú je potrebné zvážiť. Môžeme disponovať rôznym počtom zdrojov viacerých druhov, rozmiestneným na viacerých miestach pod správou viacerých organizácií. Metrické dáta vo svojej podstate len kvantifikujú využitie zdroja vo všeobecnosti. Zaoberáme sa s ním v tom zmysle, ako by bol len jeden. Nehovoríme nič bližšie o tom, o aký zdroj sa jedná, kde je ho možné nájsť. V rámci širšieho systému je preto dôležitá jednoznačná identifikácia daného zdroja. O danom zdroji preto chceme zistiť napr. názov uzla, kde sa nachádza, prípadne výrobný model. Takéto údaje sa nazývajú metadáta.

#### 3.1.5 Ukladanie dát

Uchovávanie metrických dát je kľúčovou súčasťou monitorovacieho systému. Doba, po ktorú vlastníci zdrojov uchovávajú metrické dáta, sa líši systém od systému. Množstvo dát, ktoré je treba uložiť, závisí od viacerých parametrov. Sú nimi počet druhov zdrojov, počet jednotlivých inštancií zdrojov, počet metrík, ktoré sa o zdrojoch uchovávajú, periodicita zbieraných metrík a do určitej miery aj spôsob identifikácie daného zdroja.

Ukladanie metrických dát predstavuje zároveň úzke hrdlo monitorovacieho riešenia. Zistenie hodnôt metrík pre vyťaženie zdroja konkrétnou technológiou predstavuje v serializovanú činnosť. V určitom čase je zameraná záťaž danej technológie na zdroje a hodnoty sa po jendom ukladajú. Na časť systému, ktorá je zodpovedná za ukladanie metrík, sú preto v pravidelných intervaloch kladené pomerne veľké nároky. Je ich už ale možné spracovávať paralelne. Centralizácia úložiska by znamenala prílišnú záťaž na monitorovacie uzly, preto je vhodné, aby takéto úložisko bolo distribuované a prispôbené na paralelné spracovávanie veľkých objemov dát.

#### 3.1.6 Vizualizácia a analýza dát

Zozbierané metrické dáta je ďalej potrebné nejakým spôsobom zobrazíť. Na to sú vhodné čiarové grafy s dvoma osami, kde horizontálna os predstavuje čas a vertikálna os predstavuje hodnotu nameranej metriky. Analýza zozbieraných dát slúži na odhalenie trendov vo využívaní zdrojov a môže viesť k zdokonaľovaniu systému a k lepšiemu plánovaniu využívania zdrojov.

#### 3.1.7 Monitorovanie v heterogénnom výpočtovom prostredí

Sledovanými zdrojmi v heterogénnom výpočtovom prostredí je procesor, výpočtová pamäť, trvalý ukladací priestor a sieťová infraštruktúra. Toto je spoločné pre všetky technológie a okrem týchto zdrojov je vhodné sledovať aj parametre špecifické pre jednotlivé oblasti. Tým sa budem konkrétne venovať v kapitole Metriky.

Rozdielny je aj prístup jednotlivých technológií v tom, ako poskytujú výpočtové zdroje. Aplikačné kontajnery a virtuálne stroje sa snažia rozdeliť celú infraštruktúru na menšie viac-menej uzavreté celky, ktoré sú potom sprostredkované užívateľom. Každý tento celok má pridelené zdroje, ktoré potom využíva. Množstvo týchto zdrojov je možné meniť, ale súvisí to s reštartovaním virtuálneho stroja alebo kontajnera.

Technológia gridu predstavuje odlišný prístup, kde z menších celkov (uzlov infraštruktúry) sú vytvárané väčšie virtuálne celky, ktoré vykonávajú výpočty.

#### Problematika intervalov

U vysoko výkonných výpočtových systémov sa jednotlivé operácie vykonávajú vo veľmi krátkych časových intervaloch. Rádovo sú to nanosekundy. Jednotlivé procesy a výpočtové úlohy dostávajú na krátky čas k dispozícii všetky zdroje, čím sa z vyššieho pohľadu zabezpečuje paralelizácia. To spôsobuje vyťaženie množstva inštancií zdrojov mnohými účastníkmi. Celkový stav infraštruktúry z pohľadu vykonaných úloh a prenesených dát sa preto mení veľmi dynamicky a je vhodné zbierať dáta o zdrojoch v intervale 5 až 10 sekúnd.

Nezanedbateľnú rolu hrá aj čas potrebný na získanie jedného takéhoto snímku využitia zdrojov. Ak zberáme metriky v intervale 5 sekúnd, očakávame, že odpoveď príde v kratšom intervale. V najlepšom prípade by táto doba odpovede mala byť len malý zlomok periódy danej metriky. Táto doba je v priamej závislosti od toho, koľko subjektov infraštruktúru využíva a koľko úloh spúšťa. Ak je však doba odpovede dlhšia ako samotný interval zberu, je to potrebné nejako riešiť. Nie je vhodné vygenerovať ďalšiu požiadavku na "snímku". Žiaducejším spôsobom riadenia je také, ktoré počká, kým sa daná požiadavka vykoná, a potom v nasledovnom intervale je meranie vykonané znova. Týmto sa predíde nadmernej záťaži celého systému.

Pri jednotlivých dostupných softvérových riešeniach som zohľadňoval, ako sa systém vysporiadava z takýmto neočakávaným správaním.

#### Detekcia nových zdrojov a užívateľov

Heterogénna infraštruktúra v čase mení aj množstvo a kapacitu svojich poskytovaných zdrojov. Nie je možné staticky definovať zoznam procesorov, diskov, alebo virtuálnych strojov ktoré treba monitorovať. Podobne je to aj s užívateľmi. Proces vytvárania nových užívateľov je zautomatizovaný, takže užívatelia môžu automatizovať vykonávanie svojich výpočtových úloh. Monitorovacie riešenie sa preto musí vedieť vysporiadať s týmito dynamickými zmenami, musí ich vedieť automaticky detegovať a zberať o nich metricky.

## 3.2 Časové rady

Monitorovacie dáta vo svojej podstate predstavujú časové rady. Časová rada je sekvencia dát, kde danému časovému okamihu zodpovedá jedna hodnota. Príkladom je zaznamenávanie teplôt v priebehu roka, výšky oceánskeho prílivu alebo množstvo áut, ktoré za určitú dobu prejde jedným bodom diaľnice. Efektívnou metódou vizualizácie dát časových rád sú čiarové grafy. Horizontálna os reprezentuje plynutie času a na vertikálnej osi sú znázornené hodnoty v danom čase.

### 3.2.1 Analýza časových rád

Analýza časových rád sa primárne zaoberá získavaním štatistík o zozbieraných dátach, napr. priemerná teplota počas celého roka. Medzi ďalšie úlohy patrí:

**Exploračná analýza dát** - cieľom je detekcia vzorov v dátach. Kľúčová je grafická reprezentácia dát, ktorá môže pomôcť odhaliť súvislosti, ktoré by neboli jasné z algebraickej analýzy.[13]

**Aproximácia na funkciu** - cieľom je nájsť krivku alebo matematickú funkciu, ktorá pre daný bod v čase generuje hodnotu rovnakú alebo v určitej rozumnej odchýlke od skutočnej nameranej hodnoty.

**Predpovedanie** - v štatistike patrí do oblasti štatistického odvodzovania. Ide o snahu vytvoriť štatistický model, ktorý vie s určitou pravdepodobnosťou predpovedať vývoj časových rád v budúcnosti.

**Klasifikácia** - ide o identifikáciu určitých javov v dátach a ich priradenie k nejakej udalosti - napr. výpadok, spúšťanie virtuálneho stroja atď.

#### 3.2.2 Úprava historických dát vzhľadom na kapacitu

Pri zbere časových dát je jedným z dôležitých parametrov kapacita, ktorá je potrebná na ukladanie dát. Uvažujme nasledovný príklad:

- infraštruktúra má 5 000 uzlov
- o každom zberáme 15 metrík
- metriky zberáme v intervale 5 sekúnd
- metriky sú označené metadátami o dĺžke 100 bajtov
- časová známka má 4 bajty
- hodnota metriky má 4 bajty

Po roku budú mať zberané dáta v nekomprimovanej podobe približne 51,1 TB. Vzhľadom na MetaCentrum by takýto objem predstavoval nezanedbateľnú kapacitu.

Špecializované databázy na ukladanie časových rád výrazne znižujú požiadavky na kapacitu. Ďalej je túto kapacitu možné zmenšiť rôznymi kompresnými metódami a taktiež agregáciou historických dát. Tým sa myslí združovanie nameraných dát do väčších intervalov, ukladanie príslušných hodnôt za tieto dlhšie intervaly a mazanie nepotrebných hodnôt. Jedným z cieľom práce je navrhnúť riešenie na znižovanie kapacity, ktorú zaberajú časové rady, práve spôsobom agregácie historických dát.

## 4 Aktuálne monitorovacie riešenia

Problematike monitorovania softvéru a infraštruktúry sa venuje viacero komerčných alebo open-source aplikácií, prípadne aplikácií zadarmo. Rôznymi technológiami riešia zber dát, ich uchovávanie, vizualizáciu a operácie nad nimi. Najprv sa venujem popisu aplikácií, ktoré súvisia so samotným monitorovaním, neskôr rozoberám dostupné riešenia v oblasti uchovávania časových rád.

### 4.1 Monitorovacie riešenia

Monitorovací softvér vo všeobecnosti poskytuje ucelené monitorovacie riešenie od zberu dát, cez uchovávanie a vizualizáciu. Pri konkrétnych riešeniach sa venujem aj tomu, akým spôsobom je ich možné rozšíriť o zber ďalších dát a ako sa správajú v prípade, že metriky nie sú dostupné v požadovaných intervaloch.

#### 4.1.1 Nagios

Nagios je aplikácia, ktorá poskytuje komplexné riešenie na monitorovanie systémov. Poskytuje informácie o kľúčových komponentoch infraštruktúry vrátane aplikácií, služieb, operačného systému, sieťových protokolov, systémových metrík a sieťovej infraštruktúry. [14] Aplikácia pozostáva z jadra, ktoré riadi zber údajov, a z množstva pluginov tretích strán. Tie sa zaoberajú monitorovaním jednotlivých oblastí systému. Ďalej aplikácia poskytuje grafické užívateľské rozhranie v podobe webového rozhrania. K dispozícii sú rôzne vizualizácie nameraných dát, grafy a histogramy. Nagios disponuje aj systémom užívateľských účtov. Tie sa delia na dva typy: administrátorov a bežných užívateľov. Bežní užívatelia majú prístup k nameraným hodnotám a zobrazovaniu grafov. Administrátori môžu konfigurovať aplikáciu, pridávať služby, ktoré je potrebné monitorovať, upravovať parametre monitorovania a spravovať užívateľské účty. K dispozícii je aj uchovávanie konfigurácie aplikácie a spravovanie konfigurácií. Súčasťou je systém na upozorňovanie na kritické hodnoty, či už formou emailu alebo SMS správou. Aplikácia je vyvíjaná pre platformu CentOS a Red Hat Enterprise Linux. Namerané metriky sa uchovávajú v logovacích súboroch. Pomocou pluginov je možné ich odosielať do MySQL databázy prípadne PostgreSQL.

Nagios predstavuje centralizované riešenie pre monitorovanie, kde výkonné jadro riadi zber metrík naprieč celým systémom. Jadro a pluginy sú do-

stupné zdarma, komplexné riešenie je potrebné zakúpiť. Cena sa odvíja od množstva zariadení, ktoré je potrebné monitorovať. Zariadením sa rozumie niečo, čo má IP adresu, prípadne doménové meno - či už sa jedná o firewall, switch, router, pracovnú stanicu alebo server.

V súčasnosti Nagios disponuje pluginmi pre Docker, Hadoop aj libvirt/KVM, no ani jeden z pluginov nemonitoruje požadované metriky o vyťažení zdrojov.

#### Rozšírenia

Nagios pluginy existujú vo forme skriptov alebo spustiteľných programov. Aby mohli fungovať ako pluginy ich činnosť musí spĺňať dve kritériá. Prvým je výpis aspoň jedného riadku na štandardný výstup. Ten sa týka nameraných metrík. Od verzie 3 je podporovaných aj viacerov riadkov na výstupe. Druhým kritériom je návratová hodnota. API rozoznáva štyri návratové hodnoty. Tie zodpovedajú stavom monitorovanej služby - či je služba v poriadku, či vygenerovala nejaké varovanie, či je jej stav kritický alebo neznámy. Prednastavená maximálna dĺžka výstupu je 4 kB, je ju však možné pomocou konfigurácie zmeniť.

#### Riadenie intervalov zberu metrík

Pre jednotlivé monitorovacie sondy je možné nadefinovať interval v počte sekúnd, v ktorom je potrebné aby sonda ukončila svoju činnosť. Ak sa tak nestane, proces sondy je ukončený a jej návratová hodnota zaznamenaná ako kritická. Taktiež je to zaznamenané do logov Nagiosu. Tento mechanizmus predstavuje akúsi poslednú záchranu pred zahltením systému pluginmi, ktoré sa nesprávajú podľa očakávaní.

#### 4.1.2 Icinga

Icinga predstavuje open-source monitorovací nástroj, ktorý je nástupcom Nagiosu. Je založená na princípe paralelnej činnosti viacerých vlákien, čo poskytuje možnosť vykonávať množstvo meraní za krátky časový interval. Zabezpečuje monitorovanie záťaže zdrojov ako disk, procesor, pamäť a sieť a to v operačných systémoch na základe Linuxu a aj v prostredí Windows. Ďalej podporuje zberanie dát o mnohých ďalších službách, ako je monitorovanie databáz, odozvy serverov, webových služieb, výkonu JVM<sup>1</sup>, sledovanie logov. Podporuje monitorovanie virtualizačnej platformy VMWare. Poskytuje

---

1. Java Virtual Machine



aj bežné mechanizmy v oblasti monitorovania ako generovanie notifikácií o neštandardnom správaní, na ktoré je možné reagovať spustením nejakého systémového príkazu. Na ukladanie dát používa svoju databázu, no podporuje aj odosielanie dát do databázy časových rád InfluxDB.

V súčasnosti Icinga nie je schopná monitorovať ani jednu z technológií využívaných v prostredí MetaCentra.

#### Rozšíriteľnosť aplikácie

Icinga je rozšíriteľná prostredníctvom tzv. check príkazov. Jedná sa o konfiguračné objekty, kde užívateľ definuje príkaz, ktorý sa má spustiť, spolu s hodnotami jeho argumentov. Icinga určí stav tejto kontroly podľa návratovej hodnoty príkazu.[15] Hodnoty a názvy metrík sú vracané vo forme textového výstupu.

#### 4.1.3 Zabbix

Zabbix je open-source aplikácia na monitorovanie systémov od malých systémov s malým počtom uzlov až po veľké firemné prostredia s tisíckami strojov. Architektúra aplikácie pozostáva zo servera a agentov. Úlohou agentov je zber monitorovacích dát a ich odosielanie serveru. Komunikácia týchto dvoch častí môže prebiehať dvoma spôsobmi. V prvom prípade si agent vyžiada zoznam metrík, ktoré má sledovať. Následne serveru odosiela všetky tieto metriky po jednom. Druhou alternatívou je postup, kedy sa server agenta pýta na jednotlivé hodnoty metrík a ten mu ich odosiela. Server spravuje konfiguráciu jednotlivých agentov, čo uľahčuje ovládanie ich správania naprieč celou infraštruktúrou.

Aplikácia zberá údaje o dostupných zdrojoch uzlov, o počte procesorov, dostupnej pamäti a úložnej kapacite. Taktiež zberá údaje o aktuálnom vyťažení týchto zdrojov. Okrem toho sleduje dostupnosť a parametre služieb ako FTP, DNS, HTTP, SMTP, SSH a rôznych ďalších. Taktiež poskytuje údaje o procesoch bežiacich v systéme a o užívateľoch. Zaujímavým prvkom je monitorovanie logov, ich analýza a vytváranie varovaní v prípade, že je to nutné.

Zabbix vie zberané dáta vizualizovať pomocou grafov. Súčasťou je aj sledovanie zberaných hodnôt, ich kontrola na požadovaný rozsah a generovanie notifikácií. Poskytuje aj manažment užívateľov samotnej aplikácie a ich práv na zaobchádzanie s ňou. Zabbix podporuje autodetekciu nových prvkov v infraštruktúre. Kontroluje zadaný sieťový rozsah na nové uzly, služby na nich bežiace alebo automaticky registruje nových spustených agentov.

Na zvládanie záťaže vo veľkých systémoch je zavedený systém proxy serveru.

rov. Tie zhľukujú dáta z niekoľkých agentov a až potom sú odosielané centrálnemu serveru.

Zabbix má vlastnú databázu na uchovávanie zozbieraných dát. Rozdeľuje ich na históriu a trendy. História sú dáta tak, ako boli namerané, trendy predstavujú agregované dáta z pohľadu dlhších období, napr. v rádoch rokov. Tiež je možné využiť na ukladanie dát SQL databázy - MySQL, PostgreSQL, SQLite.

V súčasnosti Zabbix prostredníctvom stiahnuteľných modulov podporuje monitorovanie Docker kontajnerov a virtuálnych strojov.

#### Rozšírenia

Aplikáciu je možné rozšíriť o vlastné moduly zberajúce dáta tromi spôsobmi.

**užívateľské parametre** - v tomto prípade užívateľ definuje v agentovi názov metriky a príkaz, ktorý sa má vykonať na zber jej hodnoty

**externé kontroly** - jedná sa o kontrolu na strane servera, kedy je tiež definovaný skript zberajúci dáta a server ho spúšťa

**system.run** - jedná sa o kontrolu na strane agenta, ktorá podporuje väčší výstup spusteného príkazu ako len hodnotu metriky

[16] Okrem toho poskytuje aj programové API a vývoj vlastných modulov ako zdieľaných knižníc, ktoré musia implementovať požadované funkcie.

#### Riadenie intervalov zberu metrík

Kontrola zberu jednotlivých metrík z hľadiska maximálnej dĺžky trvania zberu sa v Zabbixe odohráva na úrovni agenta a servera. Na úrovni agenta ide o čas, ktorý je možné stráviť spracovávaním a získavaním jednotlivých metrík. Pre server je potom definovaná doba, ktorú je ochotný čakať na odpoveď agenta. Tieto doby je možné konfigurovať v rozsahu od 1 do 30 sekúnd.[17] Zabbix nespracuje jednoduchú kontrolu na hodnotu metriky, ktorá trvá dlhšie ako túto dobu.[18] Dokumentácia však neuvádza, ako aplikácia riadi beh externých procesov, ktoré zbierajú dáta.

#### 4.1.4 Ganglia

Ganglia je škálovateľný distribuovaný systém na monitorovanie výkonných systémov ako sú clustre a gridy. Je založená na hierarchickom návrhu som zameraním na federácie clusterov. Využíva bežné štandardy, ako napr. XML

na reprezentáciu dát, XDR na prenos dát a RRDTool na ukladanie a vizualizáciu dát.[19] Cieľom aplikácie je zabezpečiť čo najmenšiu nabytočnú záťaž na uzloch a vysokú mieru súbežnosti. Ganglia vznikla na University of California v rámci Berkeley Millennium Project. Je to open-source projekt pod licenciou BSD. V súčasnosti je nasadená na mnohých operačných systémoch a podporuje viacero procesorových architektúr.

Ganglia má architektúru monitorovacieho démona, ktorý je spustený na každom monitorovanom uzle, a démona, ktorý uchováva zozbierané údaje do round-robin archívu. K dispozícii sú aj nástroje na získanie aktuálnych údajov konkrétnej sondy a tiež webové rozhranie na zobrazovanie zaznamenaných údajov.

Hadoop je možné nakonfigurovať tak, aby odosielať metriky zbernému démonovi Ganglie. [20] Ostatné integrácie v súčasnosti neobsahuje.

#### Rozšíriteľnosť aplikácie

K dispozícii je mnoho zásuvných modulov v podobe skriptov operačného systému alebo programov napísaných v skriptovacích jazykoch ako Ruby, Perl či Python. Plugíny poskytujú monitorovanie prihlásených užívateľov v systéme, rôznych aplikácií a služieb, ako napr. DNS, HTTP, ďalej podporujú zber senzorických údajov (teplota procesora, otáčky ventilátora), monitorovanie rýchlosti siete či napr. ZFS súborového systému.

#### 4.1.5 AppDynamics

AppDynamics predstavuje zaujímavé riešenie v oblasti monitorovania. Snaží sa do istej miery zjednotiť a prepojiť monitorovanie aplikácií a infaštruktúry a následne tak ľahšie identifikovať problémy s výkonom. Obsahuje bohaté vizualizácie nie len v podobe grafov o vyťažení, ale aj mapy a plošné grafy pozostávajúce s uzlov prepojených čiarami, ktoré reprezentujú vyťaženie spojenia jednotlivých služieb - napr. webový server, databáza a servery aplikačnej logiky. Monitoruje vyťaženie zdrojov ako sú procesor, pamäť, vstupno-výstupné operácie a sieť u jednotlivých uzlov infraštruktúry a dáva ich do súvisu s tým, aké role tieto uzly vykonávajú v danej business aplikácii.

Okrem toho disponuje aj integráciami s IaaS a PaaS cloudovými službami od Amazonu, RedHatu či Microsoftu a integráciami s mnohými aplikáciami vrátane zberu metrick o zdrojoch v Hadoope a Dockeri.[21]

AppDynamics predstavuje komerčné riešenie. Verzia s obmedzeným monitorovacím záberom a s obmedzenou dĺžkou ukladania údajov je k dispozícii aj zdarma, plná verzia je platená.

### Rozšíriteľnosť aplikácie

Nakoľko je AppDynamics komerčný softvér, rozšírenia sú k dispozícii od výrobcu. Neexistuje programové rozhranie ani iný spôsob, napr. spúšťanie rozšírení ako samostatných programov a analýza ich výstupu.

## 4.2 Ďalšie nástroje

V súvislosti s monitorovaním existujú aj ďalšie programy a nástroje, ktoré sa dajú využiť na zber dát. Niektoré poskytujú riešenia v oblasti riadenia zberu a zápisu dát (napr. collectd), iné sa zaoberajú riadením prístupu k zdrojom a môžu slúžiť ako zdroj dát.

### 4.2.1 Linux cgroups

Linux cgroups je technológia linuxového jadra, ktorá umožňuje limitovať, sledovať a izolovať spotrebu prostriedkov systému jednotlivými procesmi. Zavádza stromovo organizované kontrolné skupiny. Kontrolná skupina obsahuje obmedzenia pre jeden systémový prostriedok, tzv. subsystém. Príklad subsystémov, ktoré poskytuje Red Hat Enterprise Linux:

**blkio** - tento subsystém nastavuje limity na zápis a čítanie z blokových zariadení, ako napríklad HDD, SSD alebo USB disk.

**cpu** - tento subsystém využíva systémový plánovač na poskytovanie prístupu k procesoru pre jednotlivé úlohy.

**cpuacct** - tento subsystém generuje automatické správy o tom, koľko procesorových zdrojov využila úloha v danej skupine.

**cpuset** - tento subsystém priradzuje jednotlivé procesory (na viacprocesorovom systéme) a pamäťové uzly úlohám v skupine.

**devices** - tento subsystém povoľuje alebo zakazuje prístup k zariadeniam podľa úloh v skupine.

**freezer** - tento subsystém zastavuje alebo obnovuje vykonávanie úloh v skupine

**memory** - tento subsystém nastavuje limity na využívanie pamäte úlohami v skupine a generuje automatické správy na pamäťové zdroje použité úlohami.

*net\_cls* - tento subsystém označuje sieťové pakety identifikátorom triedy, čo umožňuje linuxovému ovládaču prevádzky identifikovať pakety, ktoré pochádzajú od konkrétnej úlohy.

*net\_prio* - tento subsystém poskytuje spôsob, ako dynamicky nastavovať prioritu sieťovej prevádzky pre sieťové rozhrania.

*ns* - toto je menný subsystém.

[22]

Pre každý subsystém existuje jeden strom kontrolných skupín. Ďalej skupina obsahuje zoznam procesov, ktoré podliehajú definovaným obmedzeniam. V strome kontrolných skupín sa proces vyskytuje len raz. V heterogénnom prostredí však jednotlivé technológie využívajú a spúšťajú mnoho procesov, ktoré by sa obtiažne priradzovali jednotlivým užívateľom v súvislosti so spustenými aplikáciami, preto tento spôsob monitorovania nie je úplne vhodný. Takisto zberané metriky o subsystémoch sú agregované pre kontrolnú skupinu, tj. predstavujú súčet pre skupinu procesov, takže nie je možné jednoznačne určiť koľko zdrojov spotreboval ten ktorý proces.

#### 4.2.2 collectd

Collectd je open-source monitorovacia aplikácia vyvinutá v jazyku C. Má architektúru jadra a pluginov, ktoré sa starajú o mnoho činností súvisiacich s monitorovaním od zbierania dát, cez zapisovanie metrík a sledovanie prahových hodnôt a upozorňovanie. Jadro predovšetkým ovláda a kontroluje vykonávanie týchto aktivít v požadovaných intervaloch. V súčasnosti je k dispozícii viac než 90 pluginov. Poskytujú monitorovanie základných výpočtových zdrojov ako sú procesor, pamäť, disky a sieťové rozhrania, ďalej monitorovanie mnohých aplikácií (napr. databázy, e-mail, webové servery) a protokolov. Zvolený programovací jazyk dáva dobré predpoklady na výkon, čo umožňuje beh na systémoch bez podpory skriptovacích jazykov alebo cron démona, ako napríklad zabudované systémy. Zároveň ale disponuje optimalizáciami a prvkami, aby zvládal stovky tisíc metrík.[23]

Collectd nepodporuje vizualizáciu dát ani historickú analýzu. Nepredstavuje preto úplné monitorovacie riešenie. Samotný zápis dát je uskutočňovaný pomocou pluginov a teda je možné zvoliť z viacerých spôsobov uchovávanía dát. V súčasnosti podporuje zápis do viacerých databáz, napr. MongoDB, OpenTSDB, alebo metódou HTTP POST či do súborov RRD.

V súčasnosti obsahuje plugin pre monitorovanie virtuálnych strojov prostredníctvom knižnice libvirt.

##### Rozšíriteľnosť aplikácie

Collectd poskytuje viacero spôsobov, akým je ho možné rozšíriť. Prvým aspektom je zber metrík. Umožňuje zber dodatočných metrík pomocou spúšťania systémových príkazov a pomocou rozšírení v rôznych programovacích jazykoch (Perl, Python, Java). Ďalej je možné využiť natívne API v jazyku C. Cez toto API je možné aplikáciu rozšíriť aj v ohľade notifikácií, zápisu metrických dát a spôsobu logovania udalostí.

##### Riadenie intervalov zberu metrík

V prípade, že hodnota metriky nedorazila v požadovanom intervale, je generované hlásenie. Neprichádza k tomu ale hneď po prvej chybe. Collectd zavádza mechanizmus časových limitov. To znamená, že meranie niekoľkokrát zopakuje v požadovanom intervale a až potom vygeneruje hlásenie s varovaním. Počet opakovaní je možné definovať.

### 4.3 Databázy časových rád

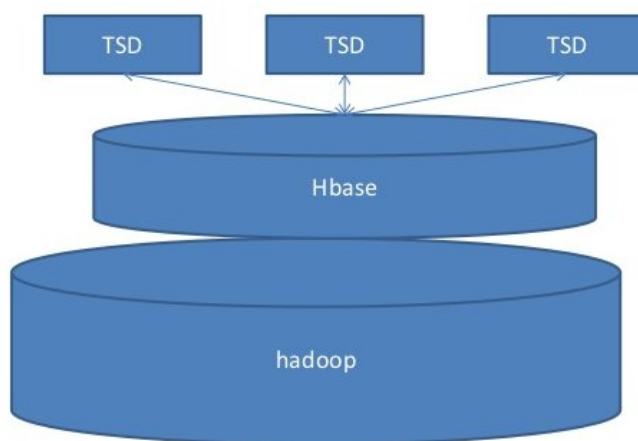
Na uchovávanie zozbierných dát sú najvhodnejšie databázy časových rád. Sú optimalizované na uchovávanie veľkého objemu dát a spracovávanie tisícok zápisov za sekundu. Vizualizácia nie je primárnou funkciou, databázy majú buď vlastné zabudované riešenia, alebo sú poskytované ako samostatný softvér.

#### 4.3.1 OpenTSDB

OpenTSDB je open-source databáza na uchovávanie a sprístupňovanie veľkých objemov časových rád. Pozostáva z Time Series Daemon (TSD) a z utilít pre príkazový riadok. Interakcia s OpenTSDB je primárne realizovaná cez jedného alebo viacerých TSD. Každý TSD je nezávislý. Neexistuje žiadny riadiaci proces, žiadny zdieľaný stav, takže je možné spustiť toľko TSD, koľko je potrebné na zvládnutie požadovanej záťaže. Každý TSD používa open-source databázu HBase na ukladanie a vyberanie dát časových rád. HBase schéma je vysoko optimalizovaná na rýchlu agregáciu podobných časových rád, aby minimalizovala požiadavky na úložný priestor. Používatelia TSD nemusia pristupovať do HBase priamo. S TSD je možné komunikovať cez jednoduchý protokol podobný Telnetu, cez HTTP API alebo cez jednoduché GUI.[24]

Na obrázku je znázornená architektúra OpenTSDB. Viacero démonov môže

zapisovať dáta do jednej databázy, ktorá je potom distribuovaná v rámci Hadoop architektúry.



Obr. 4.1: Architektúra OpenTSDB  
[25]

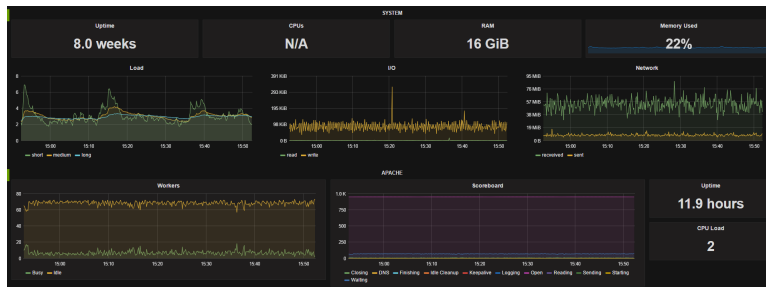
OpenTSDB uchováva čas zozbierania metriky, jej hodnotu a súbor popisných dát, ktoré sa nazývajú tagy. Na ich základe je potom možné v name-raných dáta vyhľadávať, zhľukovať ich a analyzovať. Jednou z odlišností od iných databáz je, že má schopnosť uchovávať "surové" nezhlukované dáta po veľmi dlhú dobu (čiže od začiatku merania až do prítomnosti).

Na vizualizáciu dát existuje nástroj Grafana[26]. Je to open-source webový engine, ktorý vytvára grafy zo zhromaždených dát. Je možné meniť časové rozpätie, za ktoré sa majú grafy metrík zobrazíť. Na obrázku je znázornených viacero metrík, ktorých názov sa nachádza v spodnej časti. Ich hodnoty sa v čase menia, čo je zachytené čiarovým grafom, kde horizontálna os predstavuje čas a vertikálna os hodnotu metriky v tom čase.

#### 4.3.2 InfluxDB

InfluxDB je platforma v jazyku Go na zbieranie, uchovávanie, vizualizáciu a správu časových dát. InfluxDB je určená na použitie ako úložisko dát v takých prípadoch, ktoré zahŕňajú veľké množstvo dát označených časovou známkomou a na analýzu dát v reálnom čase.[28] Užívateľ môže vytvoriť via-

#### 4. AKTUÁLNE MONITOROVACIE RIEŠENIA



Obr. 4.2: Vizualizácia časových rád OpenTSDB nástrojom Grafana [27]

cero nezávislých databáz. Dáta sa zapisujú a čítajú pomocou rozhrania príkazového riadka, rôznych klientskych knižníc, alebo pomocou HTTP API. Na vizualizáciu dát používa modul *chronograf*.

#### Politiky udržiavania

Každá databáza obsahuje pravidlá, ktoré definujú, po akú dobu majú byť ukladané dáta časových rád, a koľko kópií dát má byť vytvorených. Jedna databáza môže mať niekoľko takýchto politík. Pri zápise do nej je možné špecifikovať, ktorá politika sa má pre zápis použiť. Pri vytvorení databázy je automaticky vytvorená jedna politika.

#### Kontinuálne dotazovanie a agregácia

Databáza je schopná periodicky vykonávať požiadavky na dáta. Cieľom je znižovanie objemu dát. Ide o zhlukovanie dát s vysokou frekvenciou zberu, čím vzniknú dáta s menšou hustotou zberu. Táto hodnota je potom zvyčajne uložená do inej databázy.

#### 4.3.3 RRDTool

RRDTool je nástroj na uchovávanie, spravovanie a vizualizáciu časových dát. Využíva round-robin databázu. Je to databáza s dopredu daným maximálnou časovým rozpätím, za ktoré uchováva metrické dáta. V prípade, že príde požiadavka na zápis hodnoty a databáza je už plná, dôjde k prepisu najstaršej hodnoty. Tento nástroj takisto obsahuje funkcie na konsolidáciu dát. Konsolidovaná hodnota je typicky priemer, minimum alebo maximum



z viacerých hodnôt zozbieraných za dlhší časový úsek. Tieto hodnoty sú tak tiež ukladané do round-robin archívu.

#### 4.4 Vhodnosť dostupného softvéru

Nevýhodou niektorých z dostupných programov je spôsob rozšíriteľnosti. Nevýhodou Icingy a Nagiosu je možnosť rozšíriť ich len pomocou spustiteľných súborov, čo znižuje efektivitu zberu metrík. Medzi jednotlivými kontrolami metrík nie je možné zdieľať prostriedky, ako sú otvorené spojenia, ale je potrebné ich pri každom volaní vytvoriť a po skončení zberu ukončiť. Nevýhodou ďalších riešení je technika ukladania dát. Zabbix používa vlastnú databázu, no je možné ho prepojiť s SQL databázami. Tie však nie sú vhodné na zber veľkého objemu časových dát. Taktiež Nagios využíva na ukladanie dát len SQL databázu. Icinga podporuje okrem vlastnej databázy aj databázu časových rád InfluxDB, čo je jej výhodou. Nevýhodou Ganglie je ukladanie dát v round-robin archíve. Táto technika nie je vhodná pre dlhodobejšie ukladanie dát, čo je jednou z požiadaviek v prostredí MetaCentra.

Využitie nástroja RRDTool nie je takisto vhodné z hľadiska dlhodobého uchovávanía dát, pretože hodnoty sú po určitom čase nahradené novými. V prostredí MetaCentra totiž monitorovacie dáta nie sú zberané len pre účely monitorovania, ale aj pre potreby účtovania, takže je potrebné uchovávať dáta dlhodobo. Výhodou InfluxDB oproti databáze OpenTSDB je technika agregácie historických dát do väčších celkov. Výhodou OpenTSDB však je využívanie distribuovanej databázy HBase, ktorá je momentálne v MetaCentre produkčne využívaná. OpenTSDB je možné rozšíriť o zhlukovanie historických dát, čo je jedným z cieľov tejto práce. Ukladanie dát do OpenTSDB podporuje len collectd.

Nevýhodou používania komerčného riešenia by bola nedostatočná rozšíriteľnosť a možnosť úpravy. V prostredí MetaCentra sa môžu meniť nasadené technológie a tomu bude potrebné prispôsobiť zber metrických údajov. Komerčné riešenie by mohlo predstavovať problém, pretože rozšírenie pre danú technológiu by mohlo prísť za dlhú dobu, alebo by nemuselo prísť vôbec.



## 5 Metriky

Je potrebné určiť, ktoré zdroje monitorovať a ktoré metriky o nich zberať, aby mal zber dát potrebnú výpovednú hodnotu. V prípade MetaCentra podstatné zdroje predstavujú *procesor, pamäť, pevné disky a sieťové rozhrania*. Je vhodné mať také metriky pre využívané technológie, ktoré je možné medzi sebou porovnať. V prípade procesora sa jedná o jeho aktuálne vyťaženie, ale zároveň je zaujímavým údajom aj prepočítaný procesorový čas. Táto metrika môže byť efektívnym nástrojom pre následné účtovanie jednotlivým užívateľom. Keďže majú ale tieto technológie principiálny rozdiel vo svojom určení, nie je možné pre každú z nich zberať rovnaké dáta o každom zdroji. V prípade distribuovaných výpočtov napríklad nie je možné efektívne zistiť aktuálne vyťaženie procesora. Takéto úlohy sa počítajú na viacerých uzloch infraštruktúry. O poradí a rozmiestnení požiadaviek na zdroje rozhoduje riadiaca aplikácia, takže sa nejedná o jeden procesor. Zároveň sa dynamicky môže meniť počet procesorov, na ktorých sa daná úloha počíta.

Názvy metrických údajov uvádzam tak, ako sú pomenované vo výstupoch získavaných od jednotlivých technológií. Metriky o rovnakých zdrojoch ukladám pod rovnakými názvami aj pre rôzne technológie (líšia sa len ich prefixy). V prílohe ?? sa nachádzajú tabuľky, ktoré popisujú pod akým názvom sú v databáze uložené metriky, ktoré zmieňujem v texte.

### 5.0.1 Význam popisných údajov metrických

Metrické dáta vypovedajú o využití zdrojov. Vieme určiť aký dlhý čas a v akej miere bol využívaný výpočtový výkon. Z týchto údajov ale nevieme presne určiť, kto zdroje využíval. Pre to, aby mali tieto dáta zmysel pre neskoršie účtovanie, je potrebné mať možnosť ich priradiť k užívateľom, či už sa jedná o vlastníka virtuálneho stroja alebo užívateľa, ktorý spustil konkrétnu úlohu. Na základe týchto ďalších popisných údajov je potom možné zisťovať, kto zdroje vyťažoval za časové obdobie najviac a podľa toho stanoviť prípadnú cenu za používanie zdrojov. Okrem identity je vhodné metriky popisovať aj ďalšími údajmi, ako je miesto, kde sa zdroj nachádza, názov stroja, ktorý poskytuje svoje kapacity, a ďalšie špecifické údaje, ktoré sa týkajú jednotlivých technológií, ktoré budem popisovať konkrétnejšie v ďalších častiach.

### 5.0.2 Formát hodnoty metriky

Metriky v podobe záznamov obsahujú čas, kedy bola hodnota nameraná, samotnú hodnotu nejakého sledovaného javu a popisné dáta. Na metrické hodnoty sa môžeme pozerieť ako na prírastky alebo ako na absolútne hodnoty. Prírastky hovoria o rozdieli aktuálne nameranej hodnoty a poslednej hodnoty. Absolútne hodnoty predstavujú aktuálnu nameranú hodnotu využitia zdroja.

### 5.0.3 Výpočet percentuálnej záťaže na procesor

Žiadna z monitorovaných technológií neposkytuje údaje o percentuálnej záťaži, ktorú vyvíja na procesor. libvirt a Docker poskytujú len údaje o čase, ktorý kontajner alebo virtuálny stroj strávil počítaním na procesore.

Linuxové jadro cez systém súborov */proc* sprístupňuje svoje dátové štruktúry. V súbore */proc/stat* sa nachádzajú rôzne štatistiky o systéme, medzi ktoré patrí aj využitie procesora. Obsahuje viacero čísiel, ktoré identifikujú množstvo času, ktorý procesor strávil vykonávaním rôznych úloh. Jednotky týchto čísiel sú v *USER\_HZ* (typicky stotiny sekundy).[29] Presné množstvo času, ktoré predstavuje táto jednotka, je určené jednou z konštánt jadra.

Metriky sa zberajú v pravidelných intervaloch. V každom intervale zistím čas prepočítaný danou technológiou a celkový prepočítaný čas prostredníctvom súboru */proc/stat*. Zistím rozdiely oproti posledným nameraným hodnotám a dám ich do pomeru. Táto hodnota predstavuje percentuálnu záťaž, ktorú vyvinul kontajner alebo virtuálny stroj na hosťujúci procesor.

## 5.1 libvirt/KVM

Knižnica libvirt poskytuje údaje o všetkých sledovaných zdrojoch.

### 5.1.1 Procesor

Libvirt poskytuje údaje o tom, koľko času prepočítal daný virtuálny stroj na svojich virtuálnych procesoroch. Ďalej je možné zistiť, koľko času virtuálny stroj strávil na procesore hosťujúceho počítača. Z hľadiska využitia zdrojov je podstatný druhý údaj. Virtuálne procesory môžu, ale nemusia byť priamo namapované na fyzické procesory. Takisto záťaž na fyzický procesor nepredstavuje len čas prepočítaný na virtuálnom procesore, ale aj režijné náklady spojené s virtualizáciou.

**cpu\_time** - suma času v nanosekundách, ktorý strávil virtuálny stroj na procesore hosťujúceho počítača [30]

**cpu\_load** - hodnota v percentách, ktorá predstavuje záťaž virtuálneho stroja na procesor hosťujúceho počítača

### 5.1.2 Pamäť

O pamäti zberám nasledovné metriky:

**memory** - množstvo pamäte v kB, ktoré použité virtuálnym strojom (zo štruktúry virDomainInfo [31]).

**VIR\_DOMAIN\_MEMORY\_STAT\_SWAP\_IN** - množstvo pamäte v kB, ktoré bolo načítané zo swapovacieho priestoru.

**VIR\_DOMAIN\_MEMORY\_STAT\_SWAP\_OUT** - množstvo pamäte v kB, ktoré bolo odložené do swapovacieho priestoru.

**VIR\_DOMAIN\_MEMORY\_STAT\_MAJOR\_FAULT** - počet prípadov, kedy bolo potrebné načítať chýbajúcu stránku virtuálnej pamäte z disku.

**VIR\_DOMAIN\_MEMORY\_STAT\_MINOR\_FAULT** - počet prípadov, kedy chýbajúca stránka virtuálnej pamäte je načítaná vo fyzickej pamäti, ale nie je o tom záznam v riadacej jednotke pamäte v procesore.

**VIR\_DOMAIN\_MEMORY\_STAT\_UNUSED** - množstvo pamäte v kB, ktoré je úplne nevyužité systémom.

**VIR\_DOMAIN\_MEMORY\_STAT\_AVAILABLE** - množstvo pamäte v kB, o ktorej virtuálny stroj vie, že ju môže využiť. Táto hodnota môže byť menšia, ako veľkosť pamäte pripísaná virtuálnemu stroju, ak sa používa technika pamäťového balónu.

**VIR\_DOMAIN\_MEMORY\_STAT\_ACTUAL\_BALLOON** - aktuálna veľkosť pamäťového balónu v kB.

**VIR\_DOMAIN\_MEMORY\_STAT\_RSS** - množstvo pamäte v kB obsadené procesom, v ktorom beží virtuálny stroj. [32]

Množstvo poskytovaných štatistík závisí od použitého hypervízora. V prípade MetaCentra sa mi podarilo zberať len niektoré z týchto štatistík. Zároveň ich jednotky neboli kilobajty, ale bajty.

### Pamäťový balón

Technika pamäťového balónu umožňuje virtuálnym strojom na jednom hostiteľskom počítači zdieľať pamäť podľa aktuálnych požiadaviek. Ak existujú virtuálne stroje, ktoré pridelenú pamäť využívajú málo, hosťujúci stroj môže túto pamäť prideliť virtuálnym strojom, ktoré jej požadujú viac.

#### 5.1.3 Zápis a čítanie z disku

**rd\_req** - množstvo požiadaviek na čítanie

**rd\_bytes** - počet prečítaných bajtov

**wr\_req** - množstvo požiadaviek na zápis

**wr\_bytes** - počet zapísaných bajtov

#### 5.1.4 Sieť

Pre jednotlivé sieťové rozhrania je možné zbierať tieto metriky:

**rx\_bytes** - počet prijatých bajtov

**rx\_dropped** - počet prichádzajúcich zahodených bajtov

**rx\_error** - počet chybných bajtov

**rx\_packets** - počet prijatých paketov

**tx\_bytes** - počet odoslaných bajtov

**tx\_dropped** - počet zahodených bajtov pri pokuse o odoslanie

**tx\_errors** - počet odoslaných chybných bajtov

**tx\_packets** - počet odoslaných paketov

#### 5.1.5 Popisné údaje

Okrem metrických údajov zberám aj nasledovné popisné údaje:

**host** - názov hosťujúceho počítača

**guest\_name** - názov virtuálneho stroja

## 5.2 Docker

O kontajneroch je možné zberať metriky o všetkých požadovaných zdrojoch, tj. procesor, pamäť, sieť a vstupno-výstupné diskové operácie.

### 5.2.1 Procesor

Procesor predstavuje jeden z najdôležitejších údajov o vyťažení zdrojov. Docker poskytuje viaceré metriky o procesore, ktorých hodnoty predstavujú výpočtový čas strávený na procesore v jednotkách nanosekúnd:

**total\_usage** - predstavuje sumárnu hodnotu času v nanosekundách, ktorý strávil kontajner na všetkých jadrách spolu

**cpu\_load** - hodnota v percentách, ktorá predstavuje záťaž kontajnera na procesor hostujúceho počítača

### 5.2.2 Pamäť

Cez API Dockeru je možné získať mnoho podrobných údajov o pamäti. Nie všetky však má význam zberať, preto uvádzam len tie metriky, ktoré zberám.

**usage** - spotreba pamäte v bajtoch

**failcnt** - počet chýb v pamäti

**rss** - množstvo pamäte v bajtoch, ktoré kontajner využíva priamo v RAM, tj. okrem stránok uložených na disku alebo okrem častí kódu kontajneru, ktorý ani nebol nahratý do pamäte

**pgmajfault** - počet prípadov, kedy bolo potrebné načítať chýbajúcu stránku virtuálnej pamäti z disku.

**pgfault** - počet prípadov, kedy chýbajúca stránka virtuálnej pamäte je načítaná vo fyzickej pamäti, ale nie je o tom záznam v riadiacej jednotke pamäti v procesore.

**limit** - obmedzenie na množstvo použiteľnej pamäti v bajtoch. Buď je táto hodnota nastavená pri spustení kontajnera, alebo predstavuje množstvo pamäte inštalovanej na hostujúcom počítači.

### 5.2.3 Sieť

Aby mohli medzi sebou jednotlivé kontajnery komunikovať, Docker im poskytuje sieťové rozhrania. Každé rozhranie má nakonfigurovanú sieť, do ktorej patrí. Na to, aby kontajnery spolu mohli komunikovať, musia byť členmi rovnakej siete. Komunikácia naprieč sieťami nie je možná. Užívatelia si môžu definovať vlastné siete. Docker na vytvorenie týchto sietí poskytuje dva ovládače.

*sieť typu most* - jednoduchý typ určený pre malé siete.

*prekladaná sieť* - Docker umožňuje vytvoriť aj sieť, v ktorej sa nachádza viacero hostujúcich počítačov zároveň. To umožňuje komunikovať medzi sebou aj kontajnerom, ktoré sú spustené v rozličných sieťach, prípadne na inom hostujúcom počítači.

Metriky siete

Pre jednotlivé sieťové rozhrania je možné zbierať tieto metriky:

**rx\_bytes** - počet prijatých bajtov

**rx\_dropped** - počet prichádzajúcich zahodených bajtov

**rx\_error** - počet chybných bajtov

**rx\_packets** - počet prijatých paketov

**tx\_bytes** - počet odoslaných bajtov

**tx\_dropped** - počet zahodených bajtov pri pokuse o odoslanie

**tx\_errors** - počet odoslaných chybných bajtov

**tx\_packets** - počet odoslaných paketov

### 5.2.4 Zápis a čítanie z disku

Docker poskytuje metriky o vstupno-výstupných operáciách aj o počte bajtov, ktoré tieto operácie preniesli.

**io\_service\_recursive-Read** - predstavuje počet operácií čítania

**io\_serviced\_recursive-Write** - predstavuje počet operácií zápisu

**io\_service\_bytes\_recursive-Read** - predstavuje počet prečítaných bajtov

**io\_service\_bytes\_recursive-Write** - predstavuje počet zapísaných bajtov



### 5.2.5 Popisné údaje

Okrem metrických údajov zberám aj nasledovné popisné údaje:

**container\_id** - je to identifikačný reťazec kontajnera, ktorý ho jednoznačne identifikuje na hosťujúcom stroji

**host** - názov hosťujúceho počítača

**image\_name** - názov Docker obrazu, ktorému zodpovedá spustený kontajner

## 5.3 Hadoop

### 5.3.1 Metriky clusteru

Hadoop poskytuje o clusteri cez svoje Cluster Metrics API YARN-u rôzne údaje. Niektoré reprezentujú využitie zdrojov ako procesor a pamäť, iné poskytujú prehľad o aplikáciách a o uzloch v rámci celého clusteru. Využívam všetky informácie poskytnuté týmto API.

**appsSubmitted** - počet nahratých aplikácií.

**appsCompleted** - počet dokončených aplikácií.

**appsPending** - počet aplikácií, ktoré čakajú na svoje spustenie.

**appsRunning** - počet aplikácií práve počítaných v clusteri.

**appsFailed** - počet aplikácií, ktorých výpočet zlyhal.

**appsKilled** - počet aplikácií, ktorých beh bol ukončený pred dokončením výpočtu.

**reservedMB** - množstvo pamäte v MB rezervovanej pre beh aplikácií.

**availableMB** - množstvo pamäte v MB, ktorá je dostupná pre beh aplikácií.

**allocatedMB** - množstvo pamäte v MB, ktorá je práve alokovaná bežiacimi aplikáciami.

**totalMB** - celkové množstvo pamäte clusteru.

**reservedVirtualCores** - počet virtuálnych jadier rezervovaných pre beh aplikácií.

**availableVirtualCores** - počet dostupných virtuálnych jadier.

**allocatedVirtualCores** - počet alokovaných virtuálnych jadier.

**totalVirtualCores** - celkový počet virtuálnych jadier v clusteri.

**containersAllocated** - počet alokovaných kontajnerov.

**containersReserved** - počet rezervovaných kontajnerov.

**containersPending** - počet kontajnerov čakajúcich na umiestnenie.

**totalNodes** - počet fyzických uzlov clusteru.

**activeNodes** - počet aktívnych uzlov clusteru.

**lostNodes** - počet nedostupných uzlov.

**unhealthyNodes** - počet "nezdravých" uzlov - tj. uzly, ktoré majú napríklad málo voľného diskového priestoru, alebo sú inak obmedzené.

**decommissionedNodes** - počet uzlov vyradených z prevádzky.

**rebootedNodes** - počet reštartovaných uzlov.

Ako popisné dáta k metrikám pripájam tento údaj:

**cluster\_id** - identifikátor clusteru

### 5.3.2 Metriky aplikácií

Cluster Application API YARN-u poskytuje informácie o jednotlivých aplikáciách, ktoré sú spustené na clusteri. Niektoré údaje využívam ako zdroj metrických dát, iné ako metadáta. Niektoré nemá význam zberať. Zbieram nasledovné metrické dáta o spustených aplikáciách:

**allocatedMB** - suma pamäte v megabajtoch, ktorá je alokovaná kontajnermi, v ktorých beží aplikácia.

**allocatedVCores** - súčet virtuálnych jadier, ktoré sú alokované kontajnermi, v ktorých beží aplikácia.

**elapsedTime** - čas v milisekundách, ktorý uplynul od spustenia aplikácie.

**runningContainers** - počet kontajnerov, v ktorých je spustená aplikácia.

**memorySeconds** - množstvo pamäte v megabajtoch, ktoré aplikácia alokovala, vynásobené časom behu aplikácie v milisekundách.

**vcoreSeconds** - počet procesorových jadier, ktoré aplikácia alokovala, vynásobený časom behu aplikácie v milisekundách.

Ako popisné dáta k metrikám pripájam tieto údaje:

**id** - identifikátor aplikácie v rámci clusteru.

**user** - užívateľ, ktorý spustil aplikáciu.

**name** - názov aplikácie.

**clusterId** - identifikátor clusteru, v ktorom je spustená aplikácia.

### 5.3.3 Metriky uzlov

Nodes API systému YARN poskytuje metriky o využití jednotlivých uzlov clusteru. Podobne ako aplikačné API, poskytuje aj nadbytočné informácie, ktoré nemajú pre monitorovanie využitia zdrojov význam. Zbieram nasledovné metrické dáta o využití zdrojov uzlov:

**usedMemoryMB** - celkové množstvo pamäte v megabajtoch aktuálne využívané uzlom na beh aplikácií.

**availMemoryMB** - celkové množstvo pamäte v megabajtoch aktuálne dostupné uzlu na beh aplikácií.

**usedVirtualCores** - počet virtuálnych jadier, ktoré uzol využíva na beh aplikácií.

**availableVirtualCores** - počet virtuálnych jadier, ktoré uzol využíva na beh aplikácií.

**numContainers** - celkový počet kontajnerov aktuálne spustených na uzle.

Ako popisné dáta k metrikám pripájam tieto údaje:

**rack** - umiestnenie uzla v racku.

**id** - identifikátor uzla.

**nodeHostName** - názov uzla.

**clusterId** - identifikátor clusteru.

### 5.4 PBS Professional

#### 5.4.1 Metriky úloh

PBS Profesional poskytuje údaje o tom, ako jednotlivé úlohy spotrebovávali zdroje. O jednotlivých úlohách zberám nasledovné metriky:

**resources\_used.cpupercent** - hodnota v percentách, ktorá predstavuje záťaž úlohy na procesory

**resources\_used.cput** - čas, ktorý úloha strávila počítaním na procesoroch

**resources\_used.mem** - veľkosť pamäte alokovanej pre úlohu

**resources\_used.ncpus** - počet procesorov využívaných úlohou

**resources\_used.vmem** - veľkosť alokovanej virtuálnej pamäte pre úlohu

**resources\_used.walltime** - celkový čas, po ktorý je úloha spustená

Ako popisné dáta k metrikám pripájam tieto údaje:

**Job Id** - identifikátor úlohy

**Job\_Owner** - užívateľ, ktorý spustil úlohu

## 6 Analýza a návrh

Prostredie MetaCentra predstavuje heterogénnu infraštruktúru s rôznymi technológiami sprostredkovávajúcimi výpočetný výkon. V nasledujúcich častiach rozoberám požiadavky na monitorovacie riešenie a popisujem jeho návrh.

### 6.1 Požiadavky na aplikáciu

#### 6.1.1 Vysoký monitorovací výkon

Infraštruktúra MetaCentra pozostáva z približne 700 výpočtových uzlov. Na nich je spustených viacero výpočetných úloh, virtuálnych strojov alebo kontajnerov. Všetky tieto položky je potrebné monitorovať, čo dáva počet v rádoch tisícok. O každej z týchto položiek je zberaných viacero metrík. Ak vezmeme do úvahy príklad z 3.2.2, do databázy bude ukladaných 75 000 zázamov v priebehu 5 sekúnd, čo predstavuje 15 000 záznamov za sekundu.

#### 6.1.2 Nízka nadbytočná záťaž

Primárnou úlohou infraštruktúry je poskytovanie výpočtového výkonu a prostriedkov. Je žiaduce, aby monitorovacia aplikácia predstavovala čo najmenšiu záťaž pre systémy, na ktorých je spustená. Ak by monitorovanie samotné spotrebovávalo príliš veľa zdrojov, tak by samotná výpočtová kapacita infraštruktúry bola obmedzená už len behom monitorovacej aplikácie. Nízke nároky na výkon je možné doceliť výberom efektívneho programovacieho jazyka, napr. C, C++. V prípade použitia interpretovaného skriptovacieho jazyka, ako napríklad Python či Ruby, pribúda nadbytočná záťaž. Podobne nie je vhodným riešením používať jazyky, ktoré na svoj beh potrebujú ďalšiu vrstvu v podobe virtuálneho stroja, ako napr. Java.

#### 6.1.3 Škálovateľnosť

Monitorovacia aplikácia by mala poskytovať zrovnateľné výsledky v oblasti rýchlosti odozvy v prípade, že bude spustená na jednom uzle, ale aj v prípade, že jej úlohou bude monitorovať stovky výpočtových uzlov s množstvom spustených inštancií, ktoré treba sledovať. To je možné doceliť paralelizovaním dotazov na metriky a z hľadiska ukladania využitím distribovanej databázy.

### 6.2 Miesto nasadenia zbernej aplikácie

Zber metrík je možné realizovať v prípade virtuálnych strojov alebo aplikáčnych kontajnerov aj z ich vnútra. Bolo by možné v každej takejto virtualizovanej inštancii spustiť jednoduchú monitorovaciu aplikáciu, ktorá by interagovala priamo s virtualizovaným operačným systémom a potom by odosielať dáta. To ale nie je vhodné riešenie, nakoľko sa nedá predpokladať virtualizovaný operačný systém a zároveň to predstavuje zásah do užívateľského priestoru. Je lepšie realizovať zber metrík na základe komunikácie s aplikáciami, ktoré zabezpečujú virtualizáciu alebo spúšťajú úlohy.

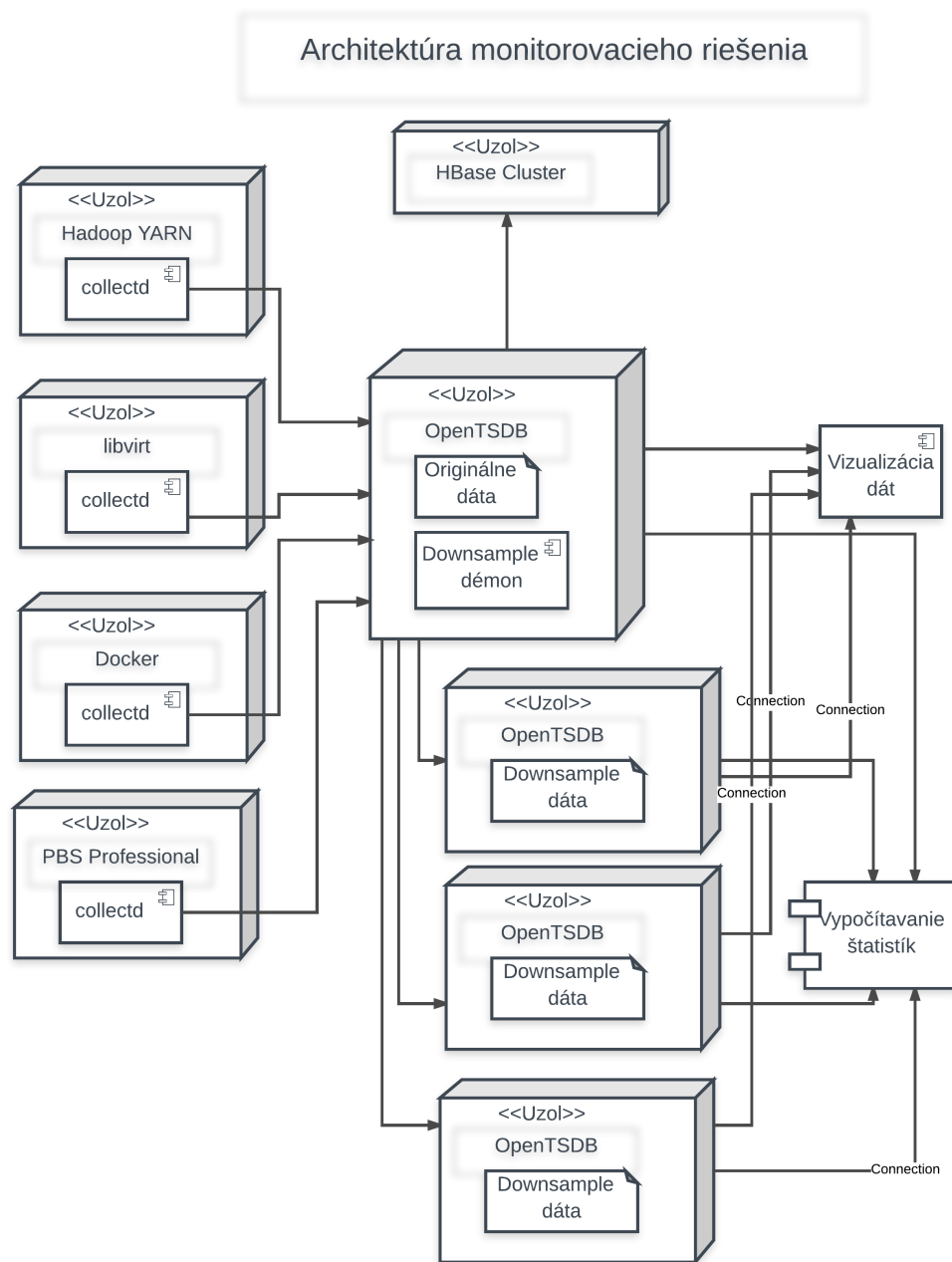
### 6.3 Návrh monitorovacieho riešenia

Monitorovacie riešenie bude využívať viacero súčastí, ktoré zabezpečujú jednotlivé činnosti súvisiace s monitorovaním tak, ako som ich rozoberal v čase 2.1 Všeobecné problémy monitorovania. Aplikácia bude zberať metriky, ktoré sú popísané v predošlej kapitole. Na nasledujúcom obrázku je architektúra celého riešenia spolu s informáciami, ako a kde sú nasadené jednotlivé komponenty. Samotný proces zberu metrických dát bude zabezpečený aplikáciou *collectd*. Oproti ostatným dostupným aplikáciám disponuje veľkou mierou rozširiteľnosti, kde je možné rozširovať nie len škálu dát, ktoré sa majú zberať, ale aj spôsob, ako a kam sa budú dáta zapisovať a tiež spôsoby, akými je možné upozorniť na hodnoty mimo definovaných medzí. Ďalšou z výhod zvolenej aplikácie je, že na uskutočňovanie zberu a zápisu používa viacero paralelne bežiacich vlákien, ktoré si medzi sebou rovnomerne delia záťaž. Ich počet je možné definovať pri konfigurácii.

Jednou z ďalších výhod *collectd* oproti iným softvérom je spôsob, akým pristupuje k spúšťaniu kontrol. Podľa konfigurácie pri štarte zistí, ktoré moduly sa budú používať. Potom sú nakonfigurované jednotlivé moduly a následne inicializované. V tomto kroku sa vytvoria spojenia, ktoré sú udržiavané počas celého behu aplikácie a zdieľané medzi jednotlivými kontrolami metrík. Nie je efektívne, aby boli tieto prostriedky inicializované pri každej požiadavke na metriku, pretože by to spomaľovalo proces samotného zberu dát. Takýto spôsob zberu dát bol jedinou možnosťou v niektorých iných dostupných aplikáciách.

Potom nasleduje fáza behu. *Collectd* beží v režime démona na pozadí a periodicky spúšťa jednotlivé moduly, ktoré zisťujú metrické dáta. V rámci modulov sa periodicky zisťuje, ktoré virtuálne stroje alebo kontajnery sú spustené a z ktorých sa tým pádom budú zberať metriky.

Po nameraní hodnoty *collectd* inicializuje zápis do databázy časových rád.



Obr. 6.1: Architektúra monitorovacieho riešenia

Modul má optimalizovanú činnosť v podobe cachovania hodnôt a metriky sa neodosielajú po jednom, ale vo väčších dávkach.

Démon takisto riadi ukončovanie činnosti modulov. Až v tejto fáze moduly uvoľnia všetky prostriedky, ktoré mali naalokované.

Samotný collectd bez pluginov predstavuje riadiacu aplikáciu v podobe démona, ktorá na svoju inštaláciu vyžaduje minimum závislostí. To je ďalšou z výhod, keďže démon na zber metrick bude nainštalovaný na každom monitorovanom uzle.

Na zápis monitorovacích dát bude využitá databáza časových rád OpenTSDB. Budem sa jej venovať v nasledujúcej časti.

Takýto návrh zabezpečuje decentralizované a škálovateľné riešenie, kedy je distribuovaný nielen zber metrických údajov, ale aj ich zápis a uchovávanie. Collectd rozšírim o jednotlivé pluginy, ktoré budú zberať metrické údaje o využití zdrojov jednotlivými technológiami MetaCentra. Pluginy budú mať schopnosť automatickej detekcie novo spustených inštancií kontajnerov a virtuálnych strojov.

### 6.3.1 Paralelizácia v rámci pluginov

Collectd poskytuje mechanizmus pracovných vlákien, v ktorých vykonávajú pluginy zber dát. To umožňuje paralelne spúšťať jednotlivé pluginy. Môže ale nastať situácia, kedy plugin zberá dáta o stovkách kontajnerov či virtuálnych strojov. Pluginy som preto navrhol tak, že využívajú paralelizáciu pri zisťovaní monitorovacích dát o jednotlivých inštanciách.

### 6.3.2 Databáza časových rád

Ako databázu na uchovávanie časových dát som si zvolil OpenTSDB. Dôvodom je používanie databázy HBase. Je to distribuovaná databáza určená pre veľké objemy dát v rádoch stoviek miliónov a milárd záznamov. Je typom NoSQL databázy. Oproti SQL databázam je lineárne škálovateľná. Ak dôjde k zdvojnásobeniu výpočtových zdrojov, dôjde aj k zdvojnásobeniu výkonu databázy. To je dôležité pri zbere časových dát z mnohých uzlov, ktoré sa v infraštruktúre MetaCentra nachádzajú. Ďalším dôvodom je, že v MetaCentre je aktuálne databáza HBase využívaná, čo predstavuje zjednodušenie nasadenia.

Zmena granularity dát na úrovni výstupu z databázy

Monitorovacie dáta môžu byť uchovávané po dobu rokov až desiatky rokov. Súčasťou môjho riešenia bude aj sada nástrojov, ktoré budú umožňovať



zmenu granularity už zapísaných metrických dát. Tým je možné prispieť k úspore úložnej kapacity a zároveň kontinuálne reagovať na zastarávanie údajov.

Pre potreby monitorovania záťaže a reakcie na danú hodnotu je vo svojej podstate významná len aktuálna záťaž prostriedkov, prípadne záťaž v okne niekoľkých hodín dozadu. Potreba uchovávaní dát je tu z dôvodu účtovania a pre možnosť sledovať vývoj využívania zdrojov za dlhšie obdobie. Podľa toho, ako dlho zberáme metriku, sa mení aj požiadavka na interval, v akom je potrebné hodnoty uchovávať. Čím staršie hodnoty sú, tým menšia granularita nám postačuje.

OpenTSDB podporuje zmenu granularity vrátených metrických dát pri vyhodnocovaní požiadaviek. Ak by v odpovedi z databázy mali byť státisíce dátových bodov, je možné matematickou funkciou upraviť viacero hodnôt za určité obdobie do jednej hodnoty. Táto agregáčna funkcia môže predstavovať súčet, priemer alebo inú operáciu. V súčasnosti databáza nepodporuje zmenu na úrovni premazávania záznamov.

## 6.4 Zmena granularity dát

Zmena granularity už zapísaných dát je daná dvojicami parametrov, ktoré pre účely tejto práce nazývam *historický interval* a *požadovaná granularita*. *Historický interval* definuje, s akými dátami pracovať a *požadovaná granularita* určuje interval, v ktorom sa za danú dobu majú uchovávať dáta. Napr. dáta staršie ako rok je potrebné uchovávať v intervale jedného dňa a dáta staršie ako mesiac v intervale jednej hodiny. Dvojicu týchto intervalov nazývam *pravidlá uchovávania*. Súbor týchto pravidiel potom definuje celú *prolitiku uchovávania*. Jednou súčasťou monitorovacieho riešenia bude nástroj, ktorý zmení granularitu dát v databáze OpenTSDB do požadovanej podoby.

### 6.4.1 Kontinuálne a periodické zmeny granularity

Zmena granularity dát môže prebiehať kontinuálnym dohľadom nad dátami alebo periodicky po uplynutí určitého intervalu.

Pri kontinuálnej zmene je rozhodujúci najkratší historický interval a k nemu prislúchajúci interval požadovanej granularity. Po uplynutí najkratšieho historického intervalu od začiatku zberu sa pre najstaršie namerané dáta začne uplatňovať príslušné pravidlo uchovávania. Následne po uplynutí príslušného intervalu požadovanej granularity je možné vykonať agregáciu. Ak uvážime politiku uchovávania z 6.4, tak dáta namerané po mesiaci od začiatku zberu by mali byť uchované v intervale jednej hodiny. Po uplynutí

mesiacu a jednej hodiny od začiatku zberu, sa vykoná agregácia dát za prvú hodinu zberu a hodnota sa uloží do databázy.

Po uplynutí druhého najkratšieho historického intervalu a príslušného intervalu požadovanej granularity dôjde k zmene dát podobným spôsobom. Čiže v prípade politiky uchovávaní z 6.4 po uplynutí roka a jedného dňa budú agregované dáta za prvý deň zberu.

Pri periodickej zmene sa agregácia vykoná raz za určitú dobu, ale nie skôr ako po uplynutí najkratšieho intervalu požadovanej granularity (inak by bol vstupný interval kratší ako interval požadovanej granularity). Vstupom sú hodnoty namerané po poslednej úspešnej zmene granularity a výstupom sú agregované hodnoty. Pre účely tejto práce uvažujem pod periodickou zmenou takú zmenu, ktorá sa vykonáva v intervale výrazne dlhšom (napr. stonásobok) ako je najkratší interval požadovanej granularity. V prípade uvažovaného príkladu z 6.4, kde je tento interval 1 hodina, by periodická zmena bola uskutočňovaná napr. raz za mesiac.

### 6.4.2 Zmena granularity a počet databáz

Po vykonaní agregácie či už v kontinuálnom alebo periodickom režime je potrebné uložiť získané agregované dáta. Je potrebné vymazať dáta s nadbytočnou hustotou a namiesto nich uložiť nové dáta. Druhou možnosťou je uchovávanie metrík s rôznou granularitou v rôznych časových databázach. V tomto prípade sa jedná o získavanie dát z jednej databázy, ich úpravu, uloženie do ďalšej databázy a vymazanie z pôvodnej databázy.

### 6.4.3 Návrh nástroja na zmenu granularity dát

Pri využívaní len jednej databázy pre všetky pravidlá uchovávaní nastáva problém so skresľovaním výsledkov. V období medzi prvým a druhým najkratším historickým intervalom sú uchované zaokrúhľované dáta, z ktorých by boli následne vyrátavané agregované metriky po uplynutí druhého najkratšieho historického intervalu. Tým sa znižuje presnosť uložených dát. Ďalšou nevýhodou je potreba uchovávať všetky agregované dáta v pamäti a zapísať ich až po vymazaní pôvodných dát. V prípade, že vymazanie neprebehne úspešne, je možné ho opakovať, no po viacerých zlyhaniach môže dôjsť k nekonzistenciám uchovávaných dát pri pokuse uložiť nové agregované dáta. Kontinuálne aj periodické zmeny granularity majú spoločný princíp. Dochádza k agregácii dát, ktoré boli namerané od poslednej úspešnej agregácie a zároveň sa premiestnili (vzhľadom na časovú známku) z jedného historického intervalu do druhého. Nevýhodou periodickej zmeny je však vysoký

objem dát, ktorý by bol spracovávaný. Ak by napríklad bola spúšťaná periodická zmena každý mesiac, tak pre príklad veľkosti infraštruktúry uvažovanej v 3.2.2, by bolo potrebné spracovať približne 39 miliárd záznamov. OpenTSDB je optimalizovaná pre prácu s veľkým množstvom záznamov, no takáto záťaž by mohla spôsobovať veľké oneskorenia pri výpočte agregovaných metrík a ich prenášaní.

Ako vhodné riešenie sa preto ponúka nástroj, ktorý využije kontinuálnu zmenu metrík a samostatnú databázu pre každé pravidlo z politiky uchovávaní. Pri kontinuálnej zmene podľa uvažovaného príkladu infraštruktúry a pravidiel uchovávaní, sa každú hodinu vypočítajú agregované metriky z 54 miliónov záznamov z pôvodnej databázy a do databázy pre prvé pravidlo sa uloží 75 000 nových záznamov. Zároveň si nástroj bude pre každé pravidlo pamätať čas poslednej úspešnej zmeny granularity. Táto hodnota bude ukladaná na disk. Ak zlyhá zmena granularity a bude potrebné nástroj reštartovať, tak bude mať informáciu o tom, ktoré dáta treba zahrnúť v ďalšom výpočte agregovaných metrík. Nepriamym dôsledkom takéhoto návrhu je aj to, že nástroj bude tým pádom možné využiť aj pre periodickú zmenu granularity.

Z databázy s pôvodnými dátami sa budú mazať dáta až vtedy, ak vek prvých nameraných metrík bude väčší ako najdlhší historický interval a príslušný interval požadovanej granularity (v uvažovanom príklade teda jeden rok a jeden deň). Je to z toho dôvodu, aby bola zachovaná čo najväčšia presnosť pri výpočte agregovaných štatistík aj pre najdlhšie historické intervaly. Mazanie z ostatných databáz sa bude uskutočňovať kaskádovito, tj. pre uvažovaný príklad sa z databázy pre mesačný historický interval budú mazať dáta staršie ako rok.

Nástroj bude predstavovať konfigurovateľné rozšírenie databázy OpenTSDB. Ako konfiguračné dáta poslúži politika uchovávaní spolu s adresami, na ktorých sú spustené TSD príslušných databáz. Okrem toho je potrebné nástroj konfigurovať metrikami, ktoré majú byť prevzorkované, spolu s metódou agregácie viacerých hodnôt do jednej. Nástroj bude bežať ako démon na mieste, kde je nasadená primárna OpenTSDB, v ktorej sú dáta s najvyššou podrobnosťou. S plynutím času bude uskutočňovať agregáciu a mazanie hodnôt po uplynutí príslušných intervalov tak, ako som popisoval vyššie.



## 7 Implementácia

### 7.1 Aplikácia na zber metrických dát

Využijem existujúcu aplikáciu na zbieranie metrických dát *collectd*. Tá poskytuje mechanizmy na periodické spúšťanie merania metrík, analýzu zozbieraných hodnôt a generovanie hlásení. Na zbieranie jednotlivých metrík som vytvoril moduly pre tento program. Tieto údaje následne bude odosielať do databázy OpenTSDB pomocou modulu WriteTSDB.

#### 7.1.1 Zapisovací plugin WriteTSDB

Plugin pre *collectd* s názvom WriteTSDB zapisuje metriky do OpenTSDB databázy.[33] Je napísaný v jazyku C. Bolo potrebné ho upraviť, aby požadovaným spôsobom generoval názvy metrík z údajov, ktoré mu predávajú pluginy zbierajúce metriky. Plugin využíva mechanizmy dočasného zhlukovania správ a ich odosielania vo väčších dávkach. Je možné nakonfigurovať adresu a port, na ktorom je spustený démon OpenTSDB. Taktiež je možné nakonfigurovať tagy špecifické pre uzol.

### 7.2 Techniky zbierania metrík

Rôzne technológie MetaCentra poskytujú svoje metrické údaje cez rôzne programové rozhrania alebo prostredníctvom textového výstupu zvyčajne dostupného prostredníctvom HTTP servera.

#### 7.2.1 Docker

Na komunikáciu s Dockerom je možné využiť *príkazy aplikácie v príkazovom riadku* alebo *Remote API*. Používanie príkazov aplikácie môže byť o čosi rýchlejšie, ale následne by bolo potrebné analyzovať textový výstup programu. Rozhodol som sa použiť Remote API. Toto API funguje pre účely monitorovania na princípe REST. Zberná aplikácia sa bude pripájať na demónu Dockeru, ktorý využíva lokálny Unixový socket, čo minimalizuje časovú odozvu. Odpovede sú generované vo formáte JSON, čo predstavuje zjednotenie spracovania výstupu. Každému kontajneru zodpovedá URL adresa, ktorá zahŕňa identifikačný reťazec kontajnera.

### 7.2.2 libvirt/KVM

Na získavanie metrických údajov o virtuálnych strojoch som použil knižnicu libvirt. Je to knižnica napísaná v jazyku C. Poskytuje množstvo rozhraní, ktoré umožňujú vytvárať virtuálne stroje, zapínať ich a vypínať, meniť ich konfiguráciu. Poskytuje tiež údaje o tom, ako virtuálny stroj využíva virtualizované zdroje.

Najprv je potrebné vytvoriť spojenie s hostujúcim počítačom. Toto je udržiavané počas celého zberu metrík. Cez vytvorené spojenie je možné zisťovať, ktoré virtuálne stroje sú spustené. V periodických intervaloch aktualizujem zoznam bežiacich strojov a odosiadam metriky len o nich.

Collectd už obsahoval plugin pre libvirt, no bolo potrebné doplniť niektoré popisné údaje metrík, ako názov hostujúceho počítača, názov virtuálneho stroja a iné. Tento plugin zbieral informácie o tom, ako sú vyťažené virtuálne procesory v podobe prepočítaného procesorového času. Tento údaj pre monitorovania zdrojov z pohľadu vlastníka infraštruktúry nie je potrebný. Nahradil som ho preto údajom o tom, ako je vyťažený procesor hostujúceho stroja prevádzkou virtuálneho stroja. Poskytujem údaje v podobe využitého procesorového času aj v podobe percentuálnej záťaže.

### 7.2.3 Hadoop

Na monitorovanie Hadoopu som si zvolil REST API YARN-u. YARN zabezpečuje správu zdrojov a plánovanie/monitorovanie vykonávania úloh do dvoch oddelených démonov.[34] ResourceManager je autoritou, ktorá rozhoduje o využívaní zdrojov aplikáciami v celom systéme. NodeManager je agent, nainštalovaný na jednotlivých uzloch, ktorý sa stará o beh kontajnerov a monitorovanie zdrojov, ktoré spotrebovávajú. Toto nahlasuje ResourceManagerovi. Preto som zvolil toto API.

REST API poskytuje odpovede vo formáte JSON a XML. Vo svojej implementácii využívam odpovede vo formáte JSON.

### 7.2.4 PBS Professional

PBS Professional poskytuje rozhranie v jazyku C prostredníctvom knižnice libpbs a hlavičkového súboru pbs\_ifl.h. Pri inicializácii pluginu dôjde k vytvoreniu spojenia na PBS server, ktorý poskytuje metriky o ním spravovaných úlohách. Toto spojenie je využívané počas celej doby trvania zberu metrík.

### 7.3 Knižnice využité pri zbere metrík

Pluginy pre collectd na zber metrických dát využívajú nasledovné knižnice.

#### 7.3.1 libcurl

libcurl je voľne šíriteľná klientska knižnica v jazyku C určená na manipuláciu so zdrojmi identifikovanými pomocou URL. Podporuje množstvo protokolov ako FTP, FILE, HTTPS, IMAPS, LDAP, POP3, SMTP, SCP atď. Podporuje taktiež SSL certifikáty, HTTP POST a PUT metódy a rôzne formy autentifikácie. V mojej implementácii túto knižnicu využívam na získavanie metrík zo systému Hadoop. Využívané REST API sú chránené autentifikačným mechanizmom Kerberos. libcurl vie využiť autorizačné tokeny, ktoré sú vygenerované pri autentifikácii pomocou Kerberosu. Posiela ich spolu s HTTP GET žiadosťou a týmto spôsobom sa autentifikuje voči Hadoopu.

libcurl je ľahko prenositeľná medzi rôznymi platformami, ako sú Solaris, BSD platformy, Windows, Linux, OS X a iné. Je vláknovo bezpečná, rýchla, dobre zdokumentovaná a kompatibilná s protokolom IPv6. [35]

#### 7.3.2 libpthread

libpthread je knižnica, ktorá umožňuje programovanie aplikácií s paralelne vykonávanými procedúrami. Tie sa nazývajú vlákna. Paralelizáciu v systéme na vyššej úrovni prináša koncept procesov. Pri vytváraní procesu ale dochádza k režijnej záťaži v operačnom systéme. Procesy si v systéme udržujú viacero parametrov, ako identifikačné číslo, identifikáciu užívateľa, premenné prostredia, zásobník, haldu, registre, zdieľané knižnice a reagujú na signály. Vlákna využívajú tieto atribúty a zavádzajú menšiu množinu parametrov a kontrolných signálov, ktoré umožňujú paralelné vykonávanie aj v rámci jedného procesu.

Túto knižnicu využívam v plugine pre Docker. V pravidelných intervaloch dochádza k zisťovaniu, ktoré kontajnery sú spustené, a na REST API zodpovedajúcich kontajnerov sú potom paralelne vytvorené a zaslané požiadavky na metrické údaje.

#### 7.3.3 cJSON

cJSON je miniatúrna knižnica v jazyku C, ktorej úlohou je prevod reťazca vo formáte JSON do hierarchickej štruktúry objektov. Zjednodušuje to analýzu obdržaných informácií vo formáte JSON. Keďže jazyk C nepozná objekty, namiesto toho sú používané štruktúry jazyka C. Knižnica neobsahuje žiadne

závislosti, na jej činnosť sú potrebné iba dva súbory - jeden so zdrojovými kódmi a jeden hlavičkový súbor.

### 7.4 Agregácia historických dát v OpenTSDB

Implementácia nástroja na agregáciu časových rád je v jazyku Java. Databáza OpenTSDB poskytuje rozhranie v podobe REST API, ktoré generuje odpovede vo formáte JSON, a taktiež nástroje príkazového riadka. Na získavanie dát z databázy s najpodrobnejšími záznamami som ale použil funkcie samotnej OpenTSDB. To odstraňuje problémy s prácou s veľkými textovými výstupmi, ako napr. uloženie celého výstupu do pamäte, jeho spracovanie a vytvorenie hierarchie objektov. Obdržané dáta potom odosielam do príslušných databáz podľa politiky udržiavania prostredníctvom REST API.

### 7.5 Zostavenie a nasadenie riešenia

Zdrojové kódy a inštrukcie na zostavenie a nasadenie mnou implementovaného riešenia sa nachádzajú v repozitári na adrese

"<https://github.com/delavei/diplomka>".

Na otestovanie funkčnosti je potrebný aspoň jeden uzol, ktorý je zdrojom metrík, a aspoň jedna inštancia OpenTSDB. Počet inštancií OpenTSDB sa líši v závislosti od definovanej politiky udržiavania. Vizualizovať agregované metriky je možné prostredníctvom nástroja Grafana.



## 8 Testovanie

### 8.1 Zber metrík

Test zberu metrík sa odohrával v intervale 5 sekúnd, v prípade technológie PBS Professional bol interval nastavený na 20 sekúnd. Dôvodom je, že aktualizácia dát na riadiacom serveri prebieha v dlhších intervaloch. Miesto, kde bola nasadená zberná aplikácia počas testovania, je popísané pri každej technológii. Zber metrík prebiehal po dobu jedného dňa.

#### 8.1.1 libvirt

Zber metrík o virtuálnych strojoch bol testovaný v infraštruktúre MetaCentra na jednom z hosťujúcich strojov, kde boli spustené virtuálne stroje. Zber metrík prebiehal podľa očakávaní a úspešne boli ukladané do testovacej inštancie OpenTSDB. Podrobnejšie o tejto databáze píšem v nasledujúcej sekcii 8.2.

#### 8.1.2 Docker

V súčasnosti Docker nie je nasadený v produkčnej prevádzke MetaCentra. Zber a odosielanie metrík som preto testoval na vlastnom počítači, s desiatimi spustenými kontajnermi. Zber a odosielanie metrík bolo úspešne otestované.

#### 8.1.3 Hadoop

MetaCentrum prevádzkuje dedikovaný Hadoop cluster s 24 uzlami na adrese *hador.ics.muni.cz*. Prostredníctvom Hadoop YARN REST API som úspešne zberal metriky o tomto clusteri a jeho uzloch. Počas testovania neboli spustené žiadne aplikácie, tak som testoval zber metrík pre už ukončené aplikácie. Tých bolo v čase testovania 9427. Zber metrík prebiehal úspešne. Odosielanie metrík a zápis do databázy neprebiehal pri takomto počte aplikácií správne, nakoľko limitom bola rýchlosť môjho internetového pripojenia. Pri znížení počtu aplikácií na 1000 prebiehalo odosielanie a zápis úspešne. Keďže o každej aplikácii je zberaných 6 metrík, za jeden interval bolo po sieti do databázy odosielaných 6000 metrík.

### 8.1.4 PBS Professional

Pre účely testovania boli v prostredí MetaCentra vytvorené dva virtuálne stroje. Na jednom bol spustený PBS Server aj komponent MoM, ktorý riadi vykonávanie úlohy na jednom uzle. Na druhom virtuálnom stroji bol spustený len komponent MoM. Prostredníctvom servera som spúšťal interaktívne úlohy, v ktorých som vyťažoval predovšetkým procesor. Po určitom čase bola spotreba nahlásená serveru a collectd správne zmeral a odoslal metriky do databázy.

## 8.2 Databáza časových rád

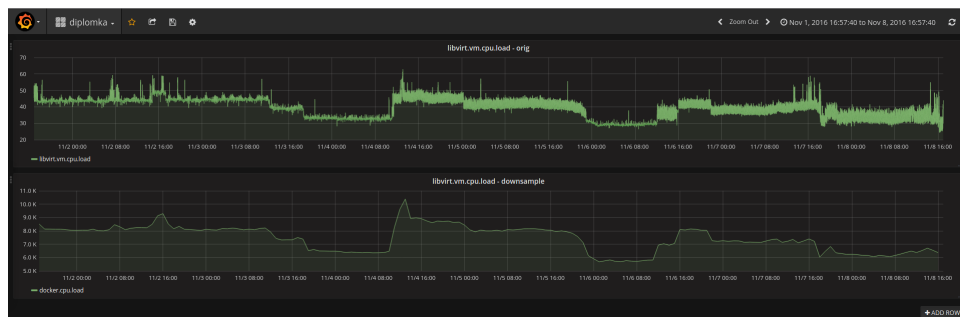
Na testovacie účely som používal databázu OpenTSDB. Jej démon bol spustený na mojom počítači. Využíval databázu HBase prevádzkovanú v Hadoop clusteri MetaCentra.

### 8.2.1 Úprava historických dát

Testovanie úpravy historických dát prebehlo na kópií aktuálnej databázy monitorovacích dát. Zníženie granularity metrík prebehlo úspešne. Nasledujúci obrázok z vizualizačného nástroja Grafana ilustruje, ako vyzerajú dáta so zníženou granularitou. Na hornej časti obrázka je granularita zberanej metriky 2 sekundy, na spodnej časti je granularita zmenená na jeden údaj za hodinu, ktorý predstavuje priemer hodnôt. Graf zobrazuje dáta v rozpätí jedného týždňa.

Pri testovaní bolo zistené, že niektoré agregované hodnoty sa líšili od hodnôt, ktoré boli očakávanými výsledkami agregáčnej funkcie. Dôvodom je, že databáza OpenTSDB ukladá dáta na hodinovom základe. Tj. dôjde k vytvoreniu identifikátora, ktorý predstavuje čas zaokrúhlený na hodiny a pod týmto identifikátorom sú ukladané hodnoty metrík spadajúcich do tohoto časového rozpätia. Po uplynutí hodiny sa vytvorí ďalší identifikátor a dáta sú ukladané tam. Ak v priebehu tejto hodiny nejaké hodnoty chýbajú, agregáčny mechanizmus OpenTSDB ich dopočíta metódou interpolácie. Preto napríklad hodnoty maxima nemusia byť vždy reálne. To predstavuje skreslenie hodnôt v databázach s agregovanými údajmi.

Tento problém by bolo možné vyriešiť agregáciou záznamov priamo na úrovni HBase.



Obr. 8.1: Pôvodné a agregované dáta - vizualizácia pomocou Grafany



## 9 Záver

Cieľom tejto práce bolo navrhnuť a implementovať monitorovacie riešenie pre zber a uchovávanie metrík v heterogénnej infraštruktúre MetaCentra. Navrhnuté riešenie malo byť rozšíriteľné, škálovateľné a s dôrazom na nízku nadbytočnú záťaž.

V úvodných častiach je popisovaná infraštruktúra MetaCentra, jednotlivé využívané technológie spolu s výpočtovými modelmi, ktoré jednotlivé technológie reprezentujú. Ďalej nasleduje rozbor jednotlivých aspektov zberu metrík a ich ukladania. V ďalšej kapitole rozoberám aktuálne monitorovacie riešenia s pohľadom na rozšíritenost a riadenie zberných intervalov. V piatej kapitole sú popisované metriky, ktoré som identifikoval ako podstatné pre prehľad o využití zdrojov jednotlivými technológiami MetaCentra. Ďalšia kapitola sa venuje analýze požiadaviek a návrhu monitorovacieho riešenia. Čitateľovi je predstavená architektúra celého riešenia spolu s tým, ako budú jednotlivé komponenty nasadené v rámci infraštruktúry MetaCentra. Je popísaná funkcionálna jednotlivých komponentov a spôsob toku dát v rámci systému. Posledné časti práce sa venujú implementácií navrhnutého riešenia a popisu, akým bolo riešenie otestované.

Ďalším krokom by mohlo byť prepracovanie agregáčného nástroja tak, že by upravoval časové rady priamo na úrovni záznamov v HBase. Výsledkom by tak bola len jedna databáza s hodnotami upravenými do požadovanej podoby podľa politiky udržiavania. Zároveň by to zjednodušilo integráciu s vizualizačnými nástrojmi a systémom pre účtovanie.



## Bibliografia

- [1] MetaCentrum. *Hardware*. 2016. URL: <http://metavo.metacentrum.cz/pbsmon2/hardware>.
- [2] Michal Kimle. ?Flexibilné účtovanie výpočtových zdrojov pre heterogénne výpočtové prostredie? Diplomová práca. Masarykova Univerzita, Fakulta informatiky, 2016.
- [3] MetaCentrum. *Informační leták k činnosti MetaCentra*. 2010. URL: [http://www.metacentrum.cz/export/sites/metacentrum/downloads/PR/letak\\_NGI.pdf](http://www.metacentrum.cz/export/sites/metacentrum/downloads/PR/letak_NGI.pdf).
- [4] Barrie A. Sosinsky. *Cloud computing bible*. Wiley, 2011.
- [5] Gerald J. Popek, Robert P. Goldberg. ?Formal requirements for virtualizable third generation architectures? In: *Communications of the ACM* 17 (1974). URL: <http://dl.acm.org/citation.cfm?doid=361011.361073> (cit. 10.12.2016).
- [6] Tom White. *Hadoop The Definitive Guide*. 3. O'Reilly, 2012. ISBN: 978-1-449-31152-0.
- [7] Ian Foster. ?What is the Grid? A Three Point Checklist? In: *Argonne National Laboratory & University of Chicago* (2002). URL: <http://dlib.cs.odu.edu/WhatIsTheGrid.pdf> (cit. 03.01.2017).
- [8] Ritzau Warnke. *qemu-kvm and libvirt*. 4. Books on Demand GmbH, 2010. ISBN: 978-3-8370-0876-0.
- [9] *libvirt: The virtualization API*. 2014. URL: <http://libvirt.org/>.
- [10] Docker Inc. *What is Docker?* 2016. URL: <http://www.docker.com/what-docker>.
- [11] *PBS Job Scheduler Important Command And Architecture - Student CPU*. 2014. URL: <http://www.pbsworks.com/pdfs/PBSProUserGuide13.1.pdf>.
- [12] Inc. Altair Engineering. *PBS Professional 13.1 User's Guide*. 2016. URL: <http://www.studentcpu.com/2012/09/pbs-job-scheduler-important-command-and.html>.
- [13] John Tukey. *Exploratory data analysis*. Reading, Mass: Addison-Wesley Pub. Co, 1977. ISBN: 9780201076165.
- [14] LLC Nagios Enterprises. *Nagios - Network, Server and Log Monitoring Software*. 2016. URL: <https://www.nagios.com/>.
- [15] Icinga Development Team. *11.1. Icinga Plugin API*. 2015. URL: <http://docs.icinga.org/latest/en/pluginapi.html>.

## BIBLIOGRAFIA

---

- [16] Zabbix SIA. *5 Loadable modules [Zabbix Documentation 2.2]*. 2016. URL: <https://www.zabbix.com/documentation/2.2/manual/config/items/loadablemodules>.
- [17] Zabbix SIA. *1 Zabbix Server [Zabbix Documentation 1.8]*. 2016. URL: [https://www.zabbix.com/documentation/1.8/manual/processes/zabbix\\_server](https://www.zabbix.com/documentation/1.8/manual/processes/zabbix_server).
- [18] Zabbix SIA. *19 Items [Zabbix Documentation 1.8]*. 2016. URL: <https://www.zabbix.com/documentation/1.8/manual/config/items>.
- [19] Ganglia. *Ganglia Monitoring System*. 2016. URL: <http://ganglia.info/>.
- [20] Ganglia. *GangliaMetrics - Hadoop Wiki*. 2016. URL: <https://wiki.apache.org/hadoop/GangliaMetrics>.
- [21] AppDynamics. *AppDynamics Exchange*. 2016. URL: <https://www.appdynamics.com/community/exchange/>.
- [22] Inc. Red Hat. *Chapter 1. Introduction to Control Groups (Cgroups)*. 2016. URL: [https://access.redhat.com/documentation/en-US/Red\\_Hat\\_Enterprise\\_Linux/6/html/Resource\\_Management\\_Guide/ch01.html](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html).
- [23] Florian Forster. *Start page - collectd - The system statistics collection daemon*. 2016. URL: <https://collectd.org/>.
- [24] The OpenTSDB Authors. *OpenTSDB - A Distributed, Scalable Monitoring System*. 2015. URL: <http://opentsdb.net/overview.html>.
- [25] Robert Chen. *Opentsdb*. 2016. URL: <http://www.slideshare.net/thecupoflife/opentsdb-in-a-real-environment>.
- [26] *Grafana.net*. 2016. URL: <https://grafana.net>.
- [27] raintank. *Apache Overview dashboard for Grafana - Grafana.net*. 2016. URL: <https://grafana.net/dashboards/331>.
- [28] Inc. InfluxData. *InfluxDB Version 1.0 Documentation*. 2016. URL: <https://docs.influxdata.com/influxdb/v1.0/>.
- [29] Michael Kerrisk Daniel Quinlan. *proc(5) - Linux manual page*. 2008. URL: <http://man7.org/linux/man-pages/man5/proc.5.html>.
- [30] *libvirt: Module libvirt-domain from libvirt*. 2016. URL: <https://libvirt.org/html/libvirt-libvirt-domain.html#virDomainGetCPUStats>.
- [31] *libvirt: Module libvirt-domain from libvirt*. 2014. URL: <https://libvirt.org/html/libvirt-libvirt-domain.html#virDomainInfo>.
- [32] *libvirt: Module libvirt-domain from libvirt*. 2016. URL: <https://libvirt.org/html/libvirt-libvirt-domain.html#virDomainMemoryStatStruct>.
- [33] *Plugin:Write TSDB - collectd Wiki*. 2015. URL: [https://collectd.org/wiki/index.php/Plugin:Write\\_TSDB](https://collectd.org/wiki/index.php/Plugin:Write_TSDB).



## BIBLIOGRAFIA

---

- [34] Apache Software Foundation. *Apache Hadoop 2.7.2 - Apache Hadoop YARN*. 2016. URL: <https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html>.
- [35] *libcurl - the multiprotocol file transfer library*. 2016. URL: <https://curl.haxx.se/libcurl/>.



## A Prevodné tabuľky názvov metrík

### A.1 libvirt

#### A.1.1 Metriky

Názov metriky v texte	Názov metriky v OpenTSDB
cpu_time	virt.cpu_time
cpu_load	virt.cpu_load
memory	virt.memory.total
VIR_DOMAIN_MEMORY_STAT_SWAP_IN	virt.memory.swap_in
VIR_DOMAIN_MEMORY_STAT_SWAP_OUT	virt.memory.swap_out
VIR_DOMAIN_MEMORY_STAT_MAJOR_FAULT	virt.memory.major_fault
VIR_DOMAIN_MEMORY_STAT_MINOR_FAULT	virt.memory.minor_fault
VIR_DOMAIN_MEMORY_STAT_UNUSED	virt.memory.unused
VIR_DOMAIN_MEMORY_STAT_AVAILABLE	virt.memory.available
VIR_DOMAIN_MEMORY_STAT_ACTUAL_BALLOON	virt.memory.actual_balloon
VIR_DOMAIN_MEMORY_STAT_RSS	virt.memory.usage
rd_req	virt.disk_ops.read
rd_bytes	virt.disk_octets.read
wr_req	virt.disk_ops.write
wr_bytes	virt.disk_octets.write
rx_bytes	virt.if_octets.rx
rx_dropped	virt.if_dropped.rx
rx_error	virt.if_errors.rx
rx_packets	virt.if_packets.rx
tx_bytes	virt.if_octets.tx
tx_dropped	virt.if_dropped.tx
tx_packets	virt.if_packets.tx
tx_error	virt.if_errors.tx

#### A.1.2 Popisné údaje

Popisný údaj v texte	Tagy v OpenTSDB
host	host
guest_name	guest_name

## A.2 Docker

### A.2.1 Metriky

Názov metriky v texte	Názov metriky v OpenTSDB
total_usage	docker.cpu_time
cpu_load	docker.cpu_load
usage	docker.memory.usage
failcnt	docker.memory.fail_count
rss	docker.memory.rss
pgfault	docker.memory.pgfault_minor
pgmajfault	docker.memory.pgfault_major
limit	docker.memory.limit
io_service_recursive-Read	docker.disk_ops.read
io_service_bytes_recursive-Read	docker.disk_octets.read
io_serviced_recursive-Write	docker.disk_ops.write
io_service_bytes_recursive-Write	docker.disk_octets.write
rx_bytes	docker.if_octets.rx
rx_dropped	docker.if_dropped.rx
rx_error	docker.if_errors.rx
rx_packets	docker.if_packets.rx
tx_bytes	docker.if_octets.tx
tx_dropped	docker.if_dropped.tx
tx_packets	docker.if_packets.tx
tx_error	docker.if_errors.tx

### A.2.2 Popisné údaje

Popisný údaj v texte	Tagy v OpenTSDB
container_id	container_id
host	host
image_name	image_name

### A.3 Hadoop Cluster

#### A.3.1 Metriky

Názov metriky v texte	Názov metriky v OpenTSDB
appsSubmitted	hadoop_cluster.cluster.apps_submitted
appsCompleted	hadoop_cluster.cluster.apps_completed
appsPending	hadoop_cluster.cluster.apps_pending
appsRunning	hadoop_cluster.cluster.apps_running
appsFailed	hadoop_cluster.cluster.apps_failed
appsKilled	hadoop_cluster.cluster.apps_killed
reservedMB	hadoop_cluster.cluster.reserved_mb
availableMB	hadoop_cluster.cluster.available_mb
allocatedMB	hadoop_cluster.cluster.allocated_mb
totalMB	hadoop_cluster.cluster.total_mb
reservedVirtualCores	hadoop_cluster.cluster.reserved_virtual_cores
availableVirtualCores	hadoop_cluster.cluster.available_virtual_cores
allocatedVirtualCores	hadoop_cluster.cluster.allocated_virtual_cores
totalVirtualCores	hadoop_cluster.cluster.total_virtual_cores
containersAllocated	hadoop_cluster.cluster.containers_allocated
containersReserved	hadoop_cluster.cluster.containers_reserved
containersPending	hadoop_cluster.cluster.containers_pending
totalNodes	hadoop_cluster.cluster.total_nodes
activeNodes	hadoop_cluster.cluster.active_nodes
lostNodes	hadoop_cluster.cluster.lost_nodes
unhealthyNodes	hadoop_cluster.cluster.unhealthy_nodes
decommissionedNodes	hadoop_cluster.cluster.decommissioned_nodes
rebootedNodes	hadoop_cluster.cluster.rebooted_nodes

#### A.3.2 Popisné údaje

Popisný údaj v texte	Tagy v OpenTSDB
clusterId	cluster_id

## A.4 Hadoop Applications

### A.4.1 Metriky

Názov metriky v texte	Názov metriky v OpenTSDB
allocatedMB	hadoop_apps.apps.allocated_mb
allocatedVCores	hadoop_apps.apps.allocated_vcores
elapsedTime	hadoop_apps.apps.elapsed_time
runningContainers	hadoop_apps.apps.running_containers
memorySeconds	hadoop_apps.apps.memory_seconds
vcoreSeconds	hadoop_apps.apps.vcore_seconds

### A.4.2 Popisné údaje

Popisný údaj v texte	Tagy v OpenTSDB
id	id
clusterId	cluster_id
name	name
user	user

## A.5 Hadoop Node

### A.5.1 Metriky

Názov metriky v texte	Názov metriky v OpenTSDB
usedMemoryMB	hadoop_node.node.used_memory_mb
availMemoryMB	hadoop_node.node.available_memory_mb
usedVirtualCores	hadoop_node.node.used_virtual_cores
availableVirtualCores	hadoop_node.node.available_virtual_cores
numContainers	hadoop_node.node.containers_running

**A.5.2 Popisné údaje**

Popisný údaj v texte	Tagy v OpenTSDB
id	node_id
clusterId	cluster_id
rack	rack
nodeHostName	host

**A.6 PBS Professional****A.6.1 Metriky**

Názov metriky v texte	Názov metriky v OpenTSDB
resources_used.cpubercent	pbs_pro.cpu_load
resources_used.cput	pbs_pro.cpu_time
resources_used.mem	pbs_pro.memory.available
resources_used.ncpus	pbs_pro.cpu_count
resources_used.vmem	pbs_pro.virtual_memory
resources_used.walltime	pbs_pro.walltime

**A.6.2 Popisné údaje**

Popisný údaj v texte	Tagy v OpenTSDB
Job Id	job_id
Job_Owner	job_owner