

MASARYKOVA UNIVERZITA  
FAKULTA INFORMATIKY



# **Monitorování zátěže a využití výpočetních zdrojů v heterogenním výpočetním prostředí**

DIPLOMOVÁ PRÁCA

**Juraj Leždík**

Brno, jar 2016

## **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.

**Vedúci práce:** Mgr. Miroslav Ruda

## **Pod'akovanie**

## **Zhrnutie**

## **Klíčové slová**

## Contents

Obsah . . . . .	viii
1 Úvod . . . . .	1
2 Zber a uchovávanie časových rád . . . . .	5
2.1 Časové rady . . . . .	5
2.1.1 Analýza časových rád . . . . .	5
2.2 OpenTSDB . . . . .	5
2.3 InfluxDB . . . . .	6
2.3.1 Politiky udržiavania . . . . .	6
2.3.2 Kontinuálne dotazovanie . . . . .	7
2.4 RRDTOol . . . . .	7
3 Aktuálne monitorovacie riešenia . . . . .	8
3.1 Open-source . . . . .	8
3.1.1 Nagios . . . . .	8
PluginAPI . . . . .	8
Timeouty . . . . .	9
3.1.2 Zabbix . . . . .	9
Timeouty . . . . .	9
Moduly . . . . .	10
3.1.3 Icinga . . . . .	10
Externé pluginy . . . . .	10
Notifikácie a príkazy udalostí . . . . .	11
Integrácie s aplikáciami . . . . .	11
3.1.4 Cacti . . . . .	11
3.1.5 collectd . . . . .	11
Obmedzenia . . . . .	11
Zapisovací plugin Write TSDB . . . . .	11
Prahy a notifikácie . . . . .	12
3.1.6 Ostatne . . . . .	12
3.2 Ganglia . . . . .	12
3.3 Komerčné riešenia . . . . .	13
3.4 Torque . . . . .	13
3.5 libvirt/KVM . . . . .	13

3.6	<i>Docker</i>	13
3.6.1	Scout	13
3.6.2	New Relic	13
3.6.3	AppDynamics	13
3.7	<i>Hadoop</i>	13
3.7.1	New Relic	13
3.7.2	AppDynamics	14
4	<b>Cloudové technológie</b>	15
4.1	<i>Dávkové úlohy</i>	15
4.1.1	Torque	15
4.2	<i>Aplikačné kontajnery</i>	15
4.2.1	Linux cgroups	16
4.2.2	Docker	17
4.3	<i>Virtuálne stroje</i>	17
4.3.1	Hypervízor	18
4.3.2	libvirt/KVM	18
4.4	<i>MapReduce aplikačné prostredia</i>	19
4.4.1	Hadoop	19
5	<b>Metriky</b>	20
5.1	<i>Torque</i>	20
5.1.1	Zbierané metriky	20
5.2	<i>Docker</i>	22
5.2.1	Sieť	22
	Sieť typu most	22
	Prekladaná sieť	23
	Metriky siete	23
5.2.2	Pamäť	23
5.2.3	Procesor	23
5.3	<i>libvirt/KVM</i>	24
5.3.1	Metriky siete	24
5.3.2	Metriky pamäte	24
5.3.3	Metriky zápisu dát	25
5.3.4	Metriky CPU	25
5.4	<i>Hadoop</i>	25
5.4.1	Cluster Metrics API	25
5.4.2	Cluster Application API	26
6	<b>Analýza a návrh</b>	28
6.1	<i>MetaCentrum</i>	28
6.2	<i>Požiadavky na aplikáciu</i>	28
6.2.1	Vysoký monitorovací výkon	28

6.2.2	Nízka nadbytočná záťaž . . . . .	29
6.2.3	Škálovateľnosť . . . . .	29
6.3	Reakcia na dlhú odozvu modulu . . . . .	30
6.4	OpenTSDB . . . . .	30
7	<b>Implementácia</b> . . . . .	31
7.1	Techniky zbierania metrík . . . . .	31
7.1.1	Torque . . . . .	31
7.1.2	Docker . . . . .	31
7.1.3	libvirt/KVM . . . . .	32
7.1.4	Hadoop . . . . .	32
8	<b>Záver</b> . . . . .	34
	Literatúra . . . . .	34
A	<b>Kapitola príloha</b> . . . . .	35



## Chapter 1

### Úvod

Cloudová infraštruktúra MetaCentra poskytuje výpočetné prostriedky pre mnohé vedecké a výskumné organizácie. Spracovávajú sa v nej veľké objemy dát. Je tvorená mnohými uzlami rozmiestnenými naprieč celou Českou republikou. Podstatou fungovania cloudového modelu je zdieľanie veľkého výpočetného výkonu viacerými subjektami, pre ktoré by zabezpečenie vlastnej infraštruktúry predstavovalo neúmernú ekonomickú, personálnu a prevádzkovú záťaž. Princípom zdieľania je, že prostriedky by mali byť ideálne využívané všetkými rovnako. Nemalo by dochádzať k tomu, že jeden klient vyťaží cloud natoľko, že výpočetné úlohy ostatných dostanú nepomerne malý priestor. To je možné docieľiť monitorovaním využitia výpočtového výkonu a periférií. Ak máme informácie o tom, kto koľko využíval zdroje, je možné tot využívanie účtovať a zároveň do budúcnosti primerane obmedzovať.

Aby bolo možné správne vyvodzovať závery o zaťažení cloudu, je potrebné zbierať údaje o tom periodicky a kontinuálne v čase. Je potrebné sledovať parametre v pravidelných intervaloch. Každému intervalu prináleží hodnota, ktorá vypovedá o využití prostriedkov v danom momente. Takéto dáta sa nazývajú časové rady. Cloud zahŕňa množstvo uzlov, kde na každom môže byť spustených množstvo úloh. O každej je potrebné zbierať viacero parametrov - vyťaženie procesora, pamäte, periférií. To predstavuje veľké množstvo metrických údajov, ktoré je potrebné ukladať a nejakým spôsobom vyhodnocovať. Slúžia na to databázy časových rád. Účtovanie sa tiež deje v určitých pravidelných intervaloch. Je preto potrebné mať možnosť spätne dohľadať údaje o využívaní zdrojov. Uchovávať podrobné metrické dáta má výnraz na určité obdobie. Nie je potrebné vedieť ako bol využívaný cloud úlohou pred piatimi rokmi každých päť sekúnd. Okrem toho by to predstavovalo veľké požiadavky na úložné kapacity, ktoré by rástli lineárne vzhľadom na počet úloh a čas. Je preto žiaduce po nejakej dobe dáta agregovať do väčších celkov a tiež mazať už nazbierané podrobné časové rady.

Aby boli monitorovacie údaje spoľahlivé, je potrebné zabezpečiť pe-

riodický zber metrických údajov. Monitorovanie sa uskutočňuje pravidelným pýtáním sa na vyťaženie daného zdroja. Aplikácia, ktorá zdroj využíva, vytvorí odpoveď a pošle ju späť monitorovacej aplikácii. Problém nastáva, ak odpoveď nie je vytvorená v dostatočne krátkom intervale. Povedzme, že chcem zberať údaje každých päť sekúnd. Aplikácia, ktorá práve rieši úlohy ale môže byť plne zaneprázdnená a nebude odpovedať na výzvy na údaje o vyťaženi. Tento problém je potrebné riešiť. Ak k tomu dôjde, nemali by byť vytvárané ďalšie výzvy na túto zaneprázdnenú aplikáciu v takej miere, ako keď plynulo odpovedala. Monitorovacia aplikácia by si mala zapamätať poslednú nameranú hodnotu a tú odoslať ako aktuálnu. Zároveň by sa mala zaneprázdnenej aplikácii pýtať v menej častých intervaloch. Ak aplikácia začne odpovedať, všetko je v poriadku. Ak nie, je možné reagovať reštartovaním monitorovacej aplikácie. Ak ani táto reakcia problém nevyrieši, komplikácia nastala zrejme v dotazovanej aplikácii.

Cloud predstavuje heterogénnu infraštruktúru. Na riešenie výpočetných úloh sa používajú rôzne technológie a aplikácie. Rôzne aplikácie pristupujú k výpočetným problémom odlišnými postupmi. Všetky úlohy sú ale počítané na akoby jednom veľkom a veľmi výkonnom počítači. Tieto aplikácie teda používajú spoločné zdroje. Metrické dáta, ktoré budem zberať, by takisto mali vypovedať o vyťaženi tých istých druhov zdrojov. Je preto potrebné identifikovať, ktoré metrické dáta z jednej aplikácie je možné porovnať s údajmi z inej aplikácie. Takto môžeme dostať celkový obraz o tom, ako rôzne technológie využívajú jednu skupinu zdrojov a len takto je možné vzájomne tieto technológie zrovnávať a účtovať ich využitie.

a

a

a

a

a Toto už je ten pôvodný úvod

V súčasnosti sú informačné technológie rozšírené vo všetkých oblastiach nášho života. Používame jednocelové integrované obvody v elektrospotrebičoch a predmetoch dennej, ako napr. v automatickej práčke, v mikrovlnnej rúre, alebo v automobile. Spoločnosť ako celok denne využíva aj veľké výpočtové výkony na komplexné úlohy. Riadenie dopravných systémov, uskutočňovanie finančných transakcií, výpočty súvisiace s predpoveďami počasia, sociálne siete. Stretávame sa s počítačmi v štátnej správe, vo vzdelávacích inštitúciách a vo veľkej miere v priemyselnej výrobe. Zdrojom veľkého množstva dát je vedecký výskum a experimenty, ktoré skúmajú svet okolo nás. Spoločným rysom všetkých týchto využití je jedno - spracovávanie informácií.

Podstatným ukazovateľom pri návrhu akéhokoľvek systému je objem spracovávaných dát. V jednoúčelových zariadeniach je potrebné spracovať menej vstupov. Jedná sa prevažne o interakciu s užívateľom prípadne s okolitým prostredím. Je kladený dôraz na rýchlu odozvu. Pri výpočtoch, ktorých výsledky využíva celá spoločnosť, je objem dát oveľa väčší. Ak by mala každá organizácia, ktorá potrebuje spracovať veľké množstvo informácií, budovať vlastnú infraštruktúru, nebolo by to vo všetkých prípadoch možné a zdroje by často mohli zostať neefektívne využité. Preto prišla snaha o budovanie veľkých systémov s množstvom výpočetného výkonu, ktoré by poskytovali zdieľaný výkon pre viacero subjektov. Nazývajú sa gridy.

Jednotlivé výpočetné problémy sa v nich spracovávajú v podobe dávkových úloh. Obsahujú dáta, nad ktorými sa budú vykonávať výpočty, programy, ktoré definujú, čo sa má s dátami urobiť, a takisto požiadavky na zdroje, ktoré bude úloha na svoje spracovanie potrebovať. Potom je úloha predaná plánovaču. Ten určí, kedy je úlohu možné spustiť, alokuje vyžadované prostriedky a určí uzly, na ktorých sa bude táto úloha spracovávať. Počas behu zároveň priebežne monitoruje záťaž výpočetných uzlov a to ako sú spotrebúvané prostriedky systému.

Infraštruktúra gridu spravidla predstavuje homogénne prostredie s jedným operačným systémom, kde je možné riešiť s veľkou efektivitou podobné problémy. S postupom času však pribúdali stále komplexnejšie a špecifickejšie úlohy, ktoré bolo treba riešiť. Existujúce prostredie neposkytovalo potrebnú flexibilitu, aby sa mohlo prispôbiť rôznorodým požiadavkam používateľov.

Reakciou na dané problémy bolo vytváranie heterogénneho výpočetného prostredia, ktoré poskytuje viacero odlišných prístupov k riešeniu výpočetných úloh. Nazýva sa cloud. Spája v sebe virtualizáciu na hardvérovej alebo softvérovej úrovni, aplikačné kontajnery a tzv. MapReduce prístup k riešeniu problémov. Rôznorodosť ponúkaných výpočetných modelov dáva užívateľom a organizáciám väčší priestor a zároveň aj motiváciu využívať tento druh služieb. Už nie sú obmedzovaní jedným operačným systémom, jednou architektúrou výpočetných clusterov. Avšak rozmanité prostredie zo sebou prináša aj niekoľko problémov. Aby bolo možné efektívne nakladať so zdrojmi cloudu, je dôležité mať možnosť zmerať, akou mierou sú využívané. Tieto merania sa uskutočňujú kontinuálne v čase v pravidelných intervaloch. Je tak generované množstvo dátových dvojíc - čas a hodnota sledovaného ukazovateľa v tomto čase. Takéto dáta sa nazývajú časové rady.

Cieľom tejto práce je preskúmať jednotlivé technológie, ktoré sa v het-

erogénnom výpočetnom prostredí používajú, zistiť, aké metrické dáta je možné zberať v súvislosti s jednotlivými výpočtnými modelmi, identifikovať, ktoré metriky je v týchto modeloch možné považovať za ekvivalentné. Zároveň sa budem venovať prieskumu aktuálnych monitorovacích riešení, ktoré zberajú metrické dáta. Výstupom práce je návrh a implementácia monitorovacej aplikácie, slúžiacej na spoľahlivý zber metrík a ich odosielanie do centralizovanej databázy časových rád. Aplikácia bude integrovaná do existujúcej infraštruktúry Národnej gridovej infraštruktúry – MetaCentra.

Práca je rozdelená do ôsmich kapitol vrátane úvodu a záveru. V úvode je načrtnutá súčasná situácia a s ňou súvisiaci problém, ktorým sa moja práca zaoberá. V druhej časti sa venujem tematike časových rád a prieskumu databáz, ktoré tieto dáta uchovávajú a spravujú. Tretia časť sa zaoberá cloudovými technológiami. Štvrtú kapitolu tvorí prieskum softvéru, ktorý slúži na monitorovanie systémov a zber metrických dát. Piata časť sa venuje metrikám, ktoré je možné o daných technológiách zberať. V šiestej kapitole analyzujem požiadavky na integráciu do cloudovej infraštruktúry a problémy spojené so zberom metrických dát. Súčasťou je návrh monitorovacej aplikácie. Siedma kapitola popisuje implementáciu a techniky použité na zber metrických dát a ich odosielanie do databázy. Súčasťou sú výsledky z testovania aplikácie nasadenej v MetaCentre. Osmou kapitolou je záver, kde zhŕňam zistené poznatky a výsledky, ktoré dosiahla moja implementácia.

## Chapter 2

### Zber a uchovávanie časových rád

#### 2.1 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.

##### 2.1.1 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*

*Aproximácia na funkciu*

*Predpovedanie*

*Klasifikácia*

Mám sa viac o nich rozpísať?

#### 2.2 OpenTSDB

OpenTSDB je databáza na uchovávanie a sprístupňovanie veľkých objemov časových dát. 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. Všetka komunikácia sa deje na tom istom porte (TSD odhadne protokol klienta pohľadom na prvých niekoľko bajtov, ktoré obdrží). [?]

Na vizualizáciu dát existuje nástroj Metrilyx. Je to open-source webový engine, ktorý vytvára grafy zo zhromaždených dát. Je možné meniť časové rozpätie, za ktoré sa majú grafy metrík zobrazíť.

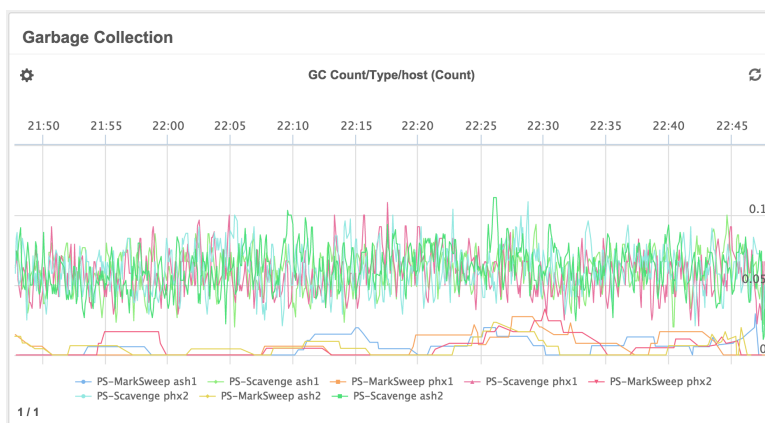


Figure 2.1: Vizualizácia časových rád OpenTSDB pomocou Metrilyx [?]

### 2.3 InfluxDB

InfluxDB je platforma na zbieranie, uchovávanie, vizualizáciu a správu časových dát. Užívateľ môže vytvoriť viacero databáz. Dáta sa zapisujú a čítajú pomocou rozhrania príkazového riadka, rôznych klientskych knižníc, alebo pomocou HTTP API. Na vizualizáciu dát používa modul *chronograf*.

#### 2.3.1 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

### 2.3.2 Kontinuálne dotazovanie

Databáza je schopná periodicky vykonávať požiadavky na dáta. Cieľom je zmenšovanie 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.

## 2.4 RRDTool

RRDTool je nástroj na uchovávanie, spravovanie a vizualizáciu časových dát. Využíva round-robin databázu. Je to databáza s dopredu danou maximálnou veľkosťou. V prípade, že príde požiadavka na zápis hodnoty a databáza je už plná, dôjde k prepisu nasjtaršej 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 ukladané do round-robin archívu.

## Chapter 3

# Aktuálne monitorovacie riešenia

### 3.1 Open-source

#### 3.1.1 Nagios

##### PluginAPI

Scripts and executables must do two things (at a minimum) in order to function as Nagios plugins:

Exit with one of several possible return values Return at least one line of text output to STDOUT The inner workings of your plugin are unimportant to Nagios. Your plugin could check the status of a TCP port, run a database query, check disk free space, or do whatever else it needs to check something. The details will depend on what needs to be checked - that's up to you.

##### Return Code

Nagios determines the status of a host or service by evaluating the return code from plugins. The following tables shows a list of valid return codes, along with their corresponding service or host states.

##### Plugin Return Code Service State Host State

0 OK UP

1 WARNING UP or DOWN/UNREACHABLE\*

2 CRITICAL DOWN/UNREACHABLE

3 UNKNOWN DOWN/UNREACHABLE

Note Note: If the use\_aggressive\_host\_checking option is enabled, return codes of 1 will result in a host state of DOWN or UNREACHABLE. Otherwise return codes of 1 will result in a host state of UP. The process by which Nagios determines whether or not a host is DOWN or UNREACHABLE is discussed here.

##### Plugin Output Spec

At a minimum, plugins should return at least one of text output. Beginning with Nagios 3, plugins can optionally return multiple lines of output. Plugins may also return optional performance data that can be processed by



external applications. The basic format for plugin output is shown below:

```
TEXT OUTPUT | OPTIONAL PERFDATA
LONG TEXT LINE 1
LONG TEXT LINE 2
...
LONG TEXT LINE N | PERFDATA LINE 2
PERFDATA LINE 3
...
PERFDATA LINE N
```

The performance data (shown in orange) is optional. If a plugin returns performance data in its output, it must separate the performance data from the other text output using a pipe (|) symbol. Additional lines of long text output (shown in blue) are also optional. [?]

#### Timeouty

Format: service\_check\_timeout=<seconds> Example: service\_check\_timeout=60

This is the maximum number of seconds that Nagios will allow service checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each service check normally finishes executing within this time limit. If a service check runs longer than this limit, Nagios will kill it off thinking it is a runaway processes.

[?]

Nie je mechanizmus pre reštart alebo individuálne timeouty. Má zasťaralý Hadoop plugin, používa staré REST API. Last Release Date 2010-11-05 [?]

#### 3.1.2 Zabbix

##### Timeouty

Timeout processing Zabbix will not process a simple check longer than Timeout seconds defined in Zabbix server configuration file. [?]

Zabbix Agent Timeout no 1-30 3 Spend no more than Timeout seconds on processing [?]

Zabbix Server Timeout no 1-30 3 Specifies how long we wait for agent, SNMP device or external check (in seconds). [?]

Neuvádza sa nič o zastavovaní procesov.

#### Moduly

Loadable modules offer a performance-minded option for extending Zabbix functionality.

There already are ways of extending Zabbix functionality by way of: user parameters (agent metrics) external checks (agent-less monitoring) system.run[] Zabbix agent item. They work very well, but have one major drawback, namely fork(). Zabbix has to fork a new process every time it handles a user metric, which is not good for performance. It is not a big deal normally, however it could be a serious issue when monitoring embedded systems, having a large number of monitored parameters or heavy scripts with complex logic or long startup time.

Zabbix 2.2 comes with support of loadable modules for extending Zabbix agent, server and proxy without sacrificing performance.

A loadable module is basically a shared library used by Zabbix daemon and loaded on startup. The library should contain certain functions, so that a Zabbix process may detect that the file is indeed a module it can load and work with.

Loadable modules have a number of benefits. Great performance and ability to implement any logic are very important, but perhaps the most important advantage is the ability to develop, use and share Zabbix modules. It contributes to trouble-free maintenance and helps to deliver new functionality easier and independently of the Zabbix core code base. [?]

#### 3.1.3 Icinga

By default the Icinga 2 daemon is running as icinga user and group using the init script. Using Debian packages the user and group are set to nagios for historical reasons. [?]

#### Externé pluginy

Icinga determines the status of a host or service by evaluating the return code from plugins. The following tables shows a list of valid return codes, along with their corresponding service or host states [?]

#### Notifikácie a príkazy udalostí

Unlike notifications, event commands for hosts/services are called on every check execution if one of these conditions match: The host/service is in a soft state The host/service state changes into a hard state The host/service state recovers from a soft or hard state to OK/Up

Ide len o spustenie nejakého systémového príkazu. [?]

#### Integrácie s aplikáciami

these tiny pure shell+awk plugins for monitoring your hadoop cluster are a enhanced and uptodate version of exchange.nagios.org check\_hadoop-dfs.sh [?]

#### 3.1.4 Cacti

Cacti is a complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. Cacti provides a fast poller, advanced graph templating, multiple data acquisition methods, and user management features out of the box. [?]

#### 3.1.5 collectd

There are some key differences we think set collectd apart. For one, it's written in C for performance and portability, allowing it to run on systems without scripting language or cron daemon, such as embedded systems. At the same time it includes optimizations and features to handle hundreds of thousands of data sets. It comes with over 90 plugins. It provides powerful networking features and is extensible in numerous ways.

#### Obmedzenia

It does not generate graphs. It can write to RRD files, but it cannot generate graphs from these files. Monitoring functionality has been added in version 4.3, but is so far limited to simple threshold checking. [?]

#### Zapisovací plugin Write TSDB

The Write TSDB plugin writes metrics to OpenTSDB, an open-source distributed time-series database based on Apache HBase. [?]

**Host Address** Hostname or address to connect to. Defaults to localhost.

**Port Service** Service name or port number to connect to. Defaults to 4242.

**HostTags String** When set, HostTags is added to the end of the metric. It is intended to be used for name=value pairs that the TSD will tag the metric with. Dots and whitespace are not escaped in this string.

**StoreRates false / true** If set to true, convert counter values to rates. If set to false (the default) counter values are stored as is, as an increasing integer number.

**AlwaysAppendDS false / true** If set the true, append the name of the Data Source (DS) to the "metric" identifier. If set to false (the default), this is only done when there is more than one DS.

#### Prahy a notifikácie

The only action the Threshold plugin takes itself is to generate and dispatch a notification. Every time a value is out of range, notification is dispatched. Also, all values that match a threshold are considered to be relevant or "interesting". As a consequence collectd will issue a notification if they are not received for Timeout iterations. for example, Timeout is set to "2" (the default) and some hosts sends it's CPU statistics to the server every 60 seconds, a notification will be dispatched after about 120 seconds. It may take a little longer because the timeout is checked only once each Interval on the server.

When a value comes within range again or is received after it was missing, an "OKAY-notification" is dispatched. [?]

#### 3.1.6 Ostatne

**Zenoss**

**Munin**

### 3.2 Ganglia

Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node

overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes. Ganglia is a BSD-licensed open-source project that grew out of the University of California, Berkeley Millennium Project [?] Pre Gangliu je dostupný monitorovací plugin pre Hadoop. [?]

### **3.3 Komerčné riešenia**

### **3.4 Torque**

Nenašiel som žiadne komerčný softvér na monitorovanie Torque.

### **3.5 libvirt/KVM**

### **3.6 Docker**

#### **3.6.1 Scout**

Scout runs within Docker containers without any special configuration. [?]

#### **3.6.2 New Relic**

<http://newrelic.com/docker>

#### **3.6.3 AppDynamics**

<https://www.appdynamics.com/community/exchange/extension/docker-monitoring-extension/>

### **3.7 Hadoop**

#### **3.7.1 New Relic**

<http://newrelic.com/plugins>

### 3.7.2 AppDynamics

The Hadoop monitoring extension captures metrics from Hadoop Resource Manager and/or Apache Ambari and displays them in Appdynamics Metric Browser.

This extension works only with the standalone machine agent.

Metrics include:

Hadoop Resource Manager App status and progress: submitted, pending, running, completed, killed, and failed app count Memory size, memory usage Allocated containers, container count in different states Node status, count of nodes in different states Scheduler capacity, app and container count Ambari Individual host metrics including CPU, disk, memory, JVM, load, network, process, and RPC metrics Service component metrics including CPU, disk, memory, JVM, load, network, process, RPC, and component-specific metrics [?]

## Chapter 4

# Cloudové technológie

### 4.1 Dávkové úlohy

#### 4.1.1 Torque

TORQUE Resource Manager poskytuje kontrolu nad dávkovými úlohami a distribuovanými výpočtovými zdrojmi. Je to pokročilý open-source product, založený na pôvodnom PBS projekte. Zahŕňa významné pokroky v oblastiach škálovania, spoľahlivosti a funkcionality a je v súčasnosti používaný desiatkami tisícov vládnych, akademických a komerčných webových stránok po celom svete. Torque môže byť voľne používaný, modifikovaný a distribuovaný je v rámci obmedzení svojou licenciou. [?]

Monitorovanie tejto aplikácie bude prebiehať prostredníctvom volania jej ovládacích príkazov:

***momctl*** - sledovanie záťaže riadiaceho procesu a zisťovanie, ktoré úlohy sa práve spracovávajú

***printjob*** - sledovanie zdrojov, ktoré spotrebovávajú konkrétna úloha

### 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 nasledovný obrázok: V kontajneri môže byť spustený nezávislý operačný systém spolu s požadovanými aplikáciami. Host'ovský počítač, na ktorom sú kontajnery spustené, má jeden operačný systém a jednu množinu prostriedkov, ktoré tieto kontajnery zdieľajú. Jednotlivé kontajnery dostávajú kontrolovaný prístup k výpočtovému výkonu, pamäti, úložnej kapacite, sieti a prípadne ďalším prostriedkom.

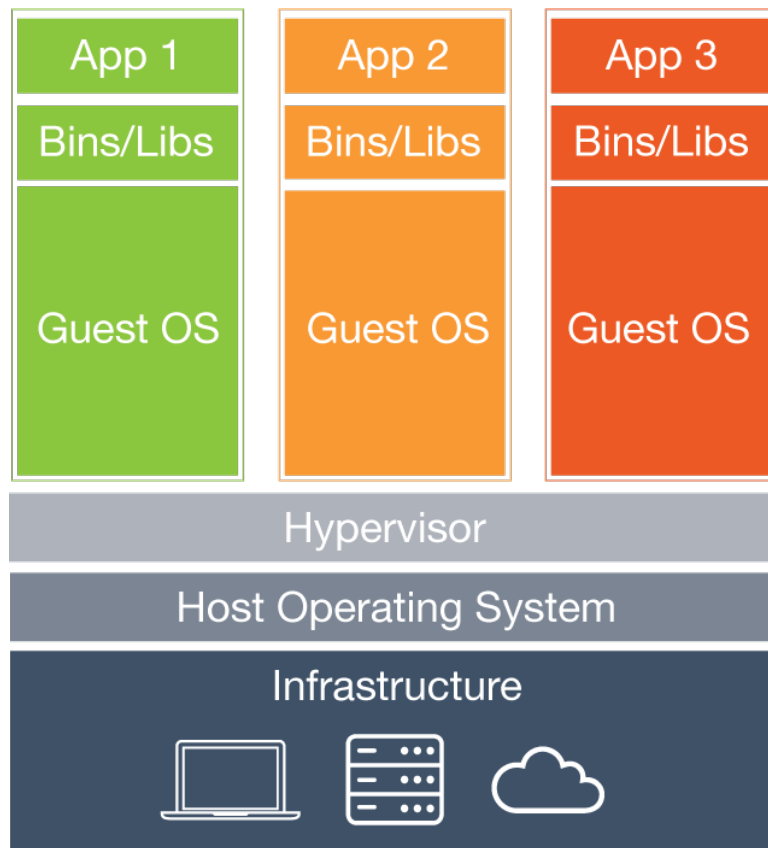


Figure 4.1: Porovnanie architektúry Docker a virtuálnych strojov  
[?]

#### 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** - this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.).

**cpu** - this subsystem uses the scheduler to provide cgroup tasks access to the CPU.



**cpuacct** - this subsystem generates automatic reports on CPU resources used by tasks in a cgroup.

**cpuset** - this subsystem assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup.

**devices** - this subsystem allows or denies access to devices by tasks in a cgroup.

**freezer** - this subsystem suspends or resumes tasks in a cgroup.

**memory** - this subsystem sets limits on memory use by tasks in a cgroup, and generates automatic reports on memory resources used by those tasks.

**net\_cls** - this subsystem tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task.

**net\_prio** - this subsystem provides a way to dynamically set the priority of network traffic per network interface.

**ns** - the namespace subsystem.

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.

#### 4.2.2 Docker

Docker umožňuje zabaliť aplikáciu so všetkými jej závislosťami do štandardizovanej jednotky (tzv. kontajner) určenej na softvérový vývoj. Kontajner Dockeru obalujú 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. [?]

#### 4.3 Virtuálne stroje

Virtuálne stroje poskytujú úplnú virtualizáciu. Na jednom hostujúcom 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

host'ovského operačného systému (ak ho host'ovský počítač má). Takéto riešenie má jednu bezpečnostnú výhodu oproti aplikačným kontajnerom. Nežiadúce fungovanie jedného virtuálneho stroja neovplyvňuje beh ostatných. V súčasnosti existuje mnoho úrovní virtuálnych strojov. Emulácia inštrukčnej sady, prekladanie programov za behu a ich optimalizácia, vysokoúrovňové virtuálne stroje (napr. Java) a systémové virtuálne stroje používané ako jednotlivcami tak na serveroch.

#### 4.3.1 Hypervízor

Hypervízor je softvér, ktorý vytvára a zabezpečuje beh virtuálnych strojov. Rozlišujeme 2 typy:

**natívny** - na host'ujúcom počítači nie je nainštalovaný žiadny operačný systém. Hypervízor spravuje hardvér host'ujúceho počítača a kontroluje beh operačných systémov, ktoré sa javia ako procesy. Príkladom je VMware ESX/ESXi, Oracle VM Server for x86 alebo Citrix XenServer.

**host'ovaný** - hypervízor je spustený ako bežný program v operačnom systéme host'ujúceho počítača. Príkladom je QEMU, VMware Workstation alebo VirtualBox.

[?]

#### 4.3.2 libvirt/KVM

KVM<sup>1</sup> je plné virtualizačné riešenie pre Linux pre x86 hardvér, obsahujúce virtualizačné rozšírenia (Intel VT or AMD-V). Pozostáva z nahrateľného modulu jadra, kvm.ko, ktorý poskytuje základ virtualizačnej infraštruktúry a špecifický modul, kvm-intel.ko alebo kvm-amd.ko. Je možné virtualizovať obrazy s operačnými systémami Linux aj Windows. Každý virtuálny stroj má vlastný virtualizovaný hardvér: sieťovú kartu, disk, grafický adaptér, atď. KVM je open-source softvér. Virtualizačný modul jadra sa nachádza v Linuxe od verzie 2.6.20. [?]

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. [?]

1. Kernel-based Virtual Machine

## 4.4 MapReduce aplikačné prostredia

### 4.4.1 Hadoop

Projekt Apache Hadoop vyvíja open-source softvér na spoľahlivé, škálovateľné, distribuované výpočty. Apache Hadoop je prostredie, ktoré umožňuje distribuované spracovávanie veľkých množstiev dát naprieč clustermi, používajúcimi jednoduché programovacie modely. Je navrhnutý tak, aby bol škálovateľný od jednotlivých serverov po tisíce strojov, kde každý poskytuje lokálny výpočetný 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.

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

## Chapter 5

### Metriky

#### 5.1 Torque

##### 5.1.1 Zbierané metriky

Príkaz `qstat -j` poskytuje viacero režimov výstupu.

**Cluster Queue Format**      the cluster queue name  
an average of the normalized load average of all queue hosts.  
the number of currently used slots.  
the number of slots reserved in advance.  
the number of currently available slots.  
the total number of slots.  
the number of slots which is in at least one of the states 'aoACDS'  
and in none of the states 'cdsuE'  
the number of slots which are in one of these states or in any  
combination of them: 'cdsuE'

**Reduced Format**      the job ID.  
the priority of the job determining its position in the pending  
jobs list.  
the name of the job,  
the user name of the job owner.  
the status of the job - one of d(letion), E(rror), h(old), r(unning),  
R(estarted), s(uspended), S(uspended), t(ransferring), T(hreshold)  
or w(aiting).  
the submission or start time and date of the job.  
the queue the job is assigned to (for running or suspended jobs  
only).

the number of job slots or the function of parallel job tasks if -g t is specified.

the array job task id. Will be empty for non-array jobs. See the -t option to qsub(1) and the -g above for additional information.

If the -t option is supplied, each status line always contains parallel job task information as if -g t were specified and each line contains the following parallel job sub-task information:

the parallel task ID (do not confuse parallel tasks with array job tasks),

the status of the parallel task - one of r(unning), R(estarted), s(uspended), S(uspended), T(hreshold), w(aiting), h(old), or x(exited).

the cpu, memory, and I/O usage,

the exit status of the parallel task,

and the failure code and message for the parallel task.

**Full Format**      the queue name

the queue type - one of B(atch), I(nteractive), C(heckpointing), P(arallel), T(ransfer) or combinations thereof or N(one),

the number of used and available job slots

the load average of the queue host,

the architecture of the queue host and

the state of the queue - one of u(nknown) if the corresponding sge\_execd(8) cannot be contacted, a(larm), A(larm), C(alendar suspended), s(uspended), S(ubordinate), d(isabled), D(isabled), E(rror) or combinations thereof.

resource availability information is printed following the queue status line.

the job ID,

the priority of the job determining its position in the pending jobs list.

the job name,

the job owner name,

the status of the job - one of t(ransferring), r(unning), R(estarted), s(uspended), S(uspended) or T(hreshold) (see the Reduced Format section for detailed information),

the submission or start time and date of the job.

the number of job slots or the function of parallel job tasks if -g t is specified.

If the -t option is supplied, each job status line also contains

the task ID,

the status of the task - one of r(unning), R(estarted), s(uspended), S(uspended), T(hreshold), w(aiting), h(old), or x(exited) (see the Reduced Format section for detailed information),

the cpu, memory, and I/O usage,

the exit status of the task,

and the failure code and message for the task.

Potrebné metriky poskytuje prepínač -t v redukovanom formáte, preto som použil túto metódu výstupu.

## 5.2 Docker

### 5.2.1 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

Je to jednoduchý typ siete určený pre malé siete. Je ju možné vytvoriť príkazom

```
$ docker network create --driver bridge NÁZOV_SIETE
```

Po vytvorení siete je možné spustiť kontajnery v tejto sieti príkazom

```
$ docker run --net=NÁZOV_SIETE --name=NÁZOV_KONTAJNERA
```

### Prekladaná sieť

Docker umožňuje vytvoriť aj sieť, v ktorej sa nachádza viacero hostov zároveň. To umožňuje komunikovať medzi sebou aj kontajnerom, ktoré sú spustené v rozličných k a ani na jednom hoste. An overlay network

### 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

### 5.2.2 Pamäť

**usage** - spotreba pamäte

**failcnt** - počet chýb

### 5.2.3 Procesor

**percpu\_usage** - využitie jednotlivých procesorov/jadier

**usage\_in\_usermode** -

**total\_usage** -

**usage\_in\_kernelmode** -

**system\_cpu\_usage** -

### 5.3 libvirt/KVM

#### 5.3.1 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

#### 5.3.2 Metriky pamäte

**VIR\_DOMAIN\_MEMORY\_STAT\_SWAP\_IN** - The total amount of memory written out to swap space (in kB).

**VIR\_DOMAIN\_MEMORY\_STAT\_SWAP\_OUT** - Page faults occur when a process makes a valid access to virtual memory that is not available. When servicing the page fault, if disk IO is required, it is considered a major fault. If not, it is a minor fault. These are expressed as the number of faults that have occurred.

**VIR\_DOMAIN\_MEMORY\_STAT\_MAJOR\_FAULT** - počet chybných bajtov

**VIR\_DOMAIN\_MEMORY\_STAT\_MINOR\_FAULT** - počet prijatých paketov

**VIR\_DOMAIN\_MEMORY\_STAT\_UNUSED** - The amount of memory left completely unused by the system. Memory that is available but used for reclaimable caches should NOT be reported as free. This value is expressed in kB.



**VIR\_DOMAIN\_MEMORY\_STAT\_AVAILABLE** - The total amount of usable memory as seen by the domain. This value may be less than the amount of memory assigned to the domain if a balloon driver is in use or if the guest OS does not initialize all assigned pages. This value is expressed in kB.

**VIR\_DOMAIN\_MEMORY\_STAT\_ACTUAL\_BALLOON** - Current balloon value (in KB).

**VIR\_DOMAIN\_MEMORY\_STAT\_RSS** - Resident Set Size of the process running the domain. This value is in kB

**VIR\_DOMAIN\_MEMORY\_STAT\_NR** - The number of statistics supported by this version of the interface. To add new statistics, add them to the enum and increase this value.

### 5.3.3 Metriky zápisu dát

**rd\_req** - number of read requests

**rd\_bytes** - number of read bytes

**wr\_req** - number of write requests

**wr\_bytes** - number of written bytes

**errs** - In Xen this returns the mysterious 'oo\_req'.

### 5.3.4 Metriky CPU

V dokumentácii sa presne neuvádza, budem musieť zistiť z implementácie.

## 5.4 Hadoop

### 5.4.1 Cluster Metrics API

Toto API poskytuje metriky o celom clusteri.

**appsSubmitted** - The number of applications submitted

**appsCompleted** - The number of applications completed

**appsPending** - The number of applications pending

**appsRunning** - The number of applications running

**appsFailed** - The number of applications failed  
**appsKilled** - The number of applications killed  
**reservedMB** - The amount of memory reserved in MB  
**availableMB** - The amount of memory available in MB  
**allocatedMB** - The amount of memory allocated in MB  
**totalMB** - The amount of total memory in MB  
**reservedVirtualCores** - The number of reserved virtual cores  
**availableVirtualCores** - The number of available virtual cores  
**allocatedVirtualCores** - The number of allocated virtual cores  
**totalVirtualCores** - The total number of virtual cores  
**containersAllocated** - The number of containers allocated  
**containersReserved** - The number of containers reserved  
**containersPending** - The number of containers pending  
**totalNodes** - The total number of nodes  
**activeNodes** - The number of active nodes  
**lostNodes** - The number of lost nodes  
**unhealthyNodes** - The number of unhealthy nodes  
**decommissionedNodes** - The number of nodes decommissioned  
**rebootedNodes** - The number of nodes rebooted

#### 5.4.2 Cluster Application API

**id** - The application id  
**user** - The user who started the application  
**name** - The application name  
**Application Type** - The application type

- queue** - The queue the application was submitted to
- state** - The application state according to the ResourceManager - valid values are members of the YarnApplicationState enum: NEW, NEW\_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED
- finalStatus** - The final status of the application if finished - reported by the application itself - valid values are: UNDEFINED, SUCCEEDED, FAILED, KILLED
- progress** - The progress of the application as a percent
- trackingUI** - Where the tracking url is currently pointing - History (for history server) or ApplicationMaster
- trackingUrl** - The web URL that can be used to track the application
- diagnostics** - Detailed diagnostics information
- clusterId** - The cluster id
- startedTime** - The time in which application started (in ms since epoch)
- finishedTime** - The time in which the application finished (in ms since epoch)
- elapsedTime** - The elapsed time since the application started (in ms)
- amContainerLogs** - The URL of the application master container logs
- amHostHttpAddress** - The nodes http address of the application master
- allocatedMB** - The sum of memory in MB allocated to the application's running containers
- allocatedVCores** - The sum of virtual cores allocated to the application's running containers
- runningContainers** - The number of containers currently running for the application
- memorySeconds** - The amount of memory the application has allocated (megabyte-seconds)
- vcCoreSeconds** - The amount of CPU resources the application has allocated (virtual core-seconds)

## Chapter 6

### Analýza a návrh

#### 6.1 MetaCentrum

Projekt MetaCentrum vznikol v roku 1996 a od roku 1999 je jeho činnosť zastrešovaná organizáciou CESNET. Zaoberá sa budovaním národnej gridovej infraštruktúry a prepojením s podobnými projektami za hranicami Českej republiky. 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 výpočetný výkon svojich clusterov. Jedná sa hlavne o akademickú spoluprácu. MetaCentrum spravuje výpočetné prostriedky a dátové úložiská AV, JČU, MU, MZLU, UK, VUT, ZČU. V súčasnosti disponuje (stav k 30.7. 2010) 1500 jadrami CPU, 100 TB využiteľnej diskovej kapacity v podobe poľa a 400 TB kapacity v podobe pások. Služby využíva 385 registrovaných aktívnych užívateľov, ktorí spolu na 750 tisíc úlohách využili 7 miliónov hodín procesorového času.

MetaCentrum primárne poskytuje svoj výpočetný 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 výpočetnej infraštruktúry a využívanie skúseností na rozvoj v domácom prostredí.

#### 6.2 Požiadavky na aplikáciu

##### 6.2.1 Vysoký monitorovací výkon

Cloudová infraštruktúra MetaCentra pozostáva z mnohých výpočetných uzlov. Sú prepojené sieťami s veľkou prenosovou kapacitou a vysokou

prieupustnosťou. Je potrebné zbierať dáta o využití množstva zapojených clusterov, na ktorých je tiež spustených mnoho výpočetných úloh či virtuálnych strojov. Metriky sú zbierané periodicky v určitých intervaloch z jednotlivých uzlov. Veľká záťaž je kladená na databázu, do ktorej sú tieto metrické dáta pravidelne odosielané.

### 6.2.2 Nízka nadbytočná záťaž

Primárnou úlohou cloudu je poskytovanie svojho výpočetného výkonu a prostriedkov. Je žiadúce, aby monitorovacia aplikácia predstavovala čo najmenšiu záťaž pre systémy, na ktorých je spustená. Ak by monitorovanie samotné spotrebovávalo príliš veľa zdrojov, takto získané metriky by nemali požadovanú presnosť. Je to možné doceliť výberom efektívneho programovacieho jazyka, napr. C, C++, prípadne rýchle skriptovacie jazyky ako Python, Ruby či Go. Nie je príliš vhodné používať jazyky, ktoré na svoj beh potrebujú ďalšiu vrstvu v podobe virtuálneho stroja, ako napr. Java.

### 6.2.3 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 desiatky až stovky výpočetných uzlov s množstvom spustených aplikácií. To je možné doceliť vhodným paralelizovaním dotazov na metriky

Monitorovacia aplikácia bude pozostávať z dvoch častí.

**démon** - jeho úlohou je zber metrík a odosielanie do databázy

**zásúvné moduly** - ich úlohou je zisťovanie metrických dát a odosielanie démonovi

Démon je program, ktorý je spustený raz a beží v systéme na pozadí. Podľa konfigurácie pri štarte zistí, ktoré moduly sa budú používať. Takisto dôjde ku konfigurácii jednotlivých modulov, napr. nastavenie potrebných ciest k požadovaným súborom. Následne dôjde k inicializácii jednotlivých modulov. V tejto fáze moduly inicializujú prostriedky, ktoré potrebujú v priebehu zberu metrík. Napr. pripojenie na správcu kontajnerov alebo hypervízora. Nie je efektívne, aby boli tieto prostriedky inicializované pri každej požiadavke na metriku, pretože by to spomaľovalo proces samotného zberu dát. Potom nasleduje fáza behu. Démon periodicky spúšťa jednotlivé moduly, ktoré zisťujú metrické dáta. Tie následne vracajú ako

odpoveď démonovi. V prípade, že je potrebné démona ukončiť, dôjde najprv k ukončeniu jednotlivých modulov. V tejto fáze moduly uvoľnia všetky prostriedky, ktoré mali naalokované.

### 6.3 Reakcia na dlhú odozvu modulu

Ak je hodnota nejakej metriky mimo určitý rozsah, je generované hlásenie. Na to je možné reagovať. Na situáciu, keď časť zodpovedná za zbieranie dát neodpovedá, je ale možné reagovať len reštartovaním celej monitorovacej aplikácie. Nie je možné jednotlivé pluginy ovládať nezávisle.

V princípe nejde nijako odlišiť, či daný modul čaká na údaje alebo došlo k chybe a modul neodpovedá. Preto bude potrebné vytvoriť niektoré pluginy tak, aby jedna časť bola neustále dostupná a reagovala na výzvy od riadiacej aplikácie. Bude definovaný časový interval na vrátenie hodnoty. V prípade ak plugin úspešne v časovom intervale zistil dané metrické dáta, vráti ich riadiacej aplikácii. V prípade, že v danom intervale plugin neobdržal metrické dáta, vráti poslednú hodnotu. Zároveň sa nebudú vytvárať nové požiadavky na tento údaj. Tento časový interval si bude môcť užívateľ nastaviť pre každú sondu. Predmetom testovania bude zistiť, aký interval by bol vhodný pre tú ktorú sondu.

Ďalšou prahovou hodnotou bude počet opakovaní, pri ktorých plugin vracia poslednú hodnotu danej metriky. Ak dôjde k prekročeniu tejto hodnoty, bude reštartovaná celá monitorovacia aplikácia.

### 6.4 OpenTSDB

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 linárne škálovateľná. Ak dôjde k zdvojnásobeniu výpočetných zdrojov, dôjde aj k zdvojnásobeniu výkonu databázy. To je dôležité pri zbere časových dát z mnohých uzlov, ktoré sa v gridovej infraštruktúre MetaCentra nachádzajú. Ďalším dôvodom je, že v MetaCentre je aktuálne databáza HBase využívaná.

## Chapter 7

### Implementácia

Využijem existujúcu aplikáciu na zbieranie metrických dát *collectd*. Na zbieranie jednotlivých metrík som vytvoril moduly pre tento program. Tieto údaje následne bude odosielať do databázy OpenTSDB pomocou modulu WriteTSDB.

#### 7.1 Techniky zbierania metrík

##### 7.1.1 Torque

Na zbieranie metrík som použil príkaz *qstat -j*.

Prints either for all pending jobs or the jobs contained in *job\_list* various information. The *job\_list* can contain *job\_ids*, *job\_names*, or wildcard expression *sge\_types(1)*.

For jobs in E(rror) state the error reason is displayed. For jobs that could not be dispatched during in the last scheduling interval the obstacles are shown, if 'schedd\_job\_info' in *sched\_conf(5)* is configured accordingly.

For running jobs available information on resource utilization is shown about consumed cpu time in seconds, integral memory usage in Gbytes seconds, amount of data transferred in io operations, current virtual memory utilization in Mbytes, and maximum virtual memory utilization in Mbytes. This information is not available if resource utilization retrieval is not supported for the OS platform where the job is hosted.

##### 7.1.2 Docker

Budem zberať metriky o spotrebovávaných zdrojoch pre každý kontajner. Na komunikáciu s Dockerom je možné využiť:

**príkazy aplikácie v príkazovom riadku**

**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 a odpovede vracia vo formáte JSON, čo predstavuje zjednodušenie spracovania výstupu. Démon Dockeru "počúva" na lokálnom sockete, čo by nemalo spôsobovať výrazné oneskorenie odpovede.

O každom kontajneri spustenom v Dockeri budem sledovať využívanie týchto zdrojov :

*procesor*

*pevný disk*

*sieť*

*pamäť*

Dáta je možné zberať buď ako prúd, alebo po jednorazových žiadostiach. To ešte nemám veľmi preštudované, neviem aké sú tam intervaly, tak sa rozhodnem až neskôr. Predpokladám ale, že z hľadiska rýchlosti a alokácie bude cesta asi ten stream.

### 7.1.3 libvirt/KVM

Na monitorovanie virtuálnych strojov použijem existujúcu implementáciu sond v jazyku Python. Je šírená pod licenciou open-source a vznikla v rámci organizácie Cesnet.

### 7.1.4 Hadoop

The Hadoop YARN web service REST APIs are a set of URI resources that give access to the cluster, nodes, applications, and application historical information. The URI resources are grouped into APIs based on the type of information returned. Some URI resources return collections while others return singletons. HTTP Requests

To invoke a REST API, your application calls an HTTP operation on the URI associated with a resource.

Summary of HTTP operations

Currently only GET is supported. It retrieves information about the resource specified.

Security

The web service REST API's go through the same security as the web UI.



[?]

## **Chapter 8**

### **Záver**

**Appendix A**

**Kapitola priloha**