ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

DYNAMICKÉ VYROVNÁVANIE ZÁŤAŽE V CLOUDOVÝCH SYSTÉMOCH S KONCEPTOM ZELENÉHO POČÍTANIA

Dizertačná práca

28360020203013

Študijný program: aplikovaná informatika
Študijný odbor: informatika
Pracovisko: Katedra informatiky
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline
Školiteľ: doc. Ing. Jarmila Škrinárová, PhD.

Žilina, 2020 **Mgr. Eduard Vesel**

Poďakovanie

Chcel by som sa poďakovať svojej školiteľke doc. Ing. Jarmile Škrinárovej, PhD. za trpezlivosť, konzultácie, cenné rady a pripomienky, ktoré mi pomohli pri vypracovaní práce. Rovnako sa chcem poďakovať aj RNDr. Alžbete Michalíkovej, PhD. za pomoc a konzultácie pri riešení problematiky fuzzy logiky. Ďakujem Mariánovi a Talimu za riešenie mojich problémov s Javou. V neposlednom rade patrí veľká vďaka mojej manželke a rodine, ktorí mi mentálne pomohli dokončiť štúdium a podporovali ma počas celého štúdia.

ABSTRAKT

VESEL, Eduard: *Dynamické vyrovnávanie záťaže v cloudových systémoch s konceptom zeleného počítania*. [Dizertačná práca]. – Žilinská univerzita; Fakulta riadenia a informatiky; Katedra informatiky. – Školiteľ: doc. Ing. Jarmila Škrinárová, PhD. – Žilina: FRI UNIZA, 2020, 89 s.

Cieľom dizertačnej práce je analyzovať a prezentovať riešenie techník pre dynamické vyrovnávanie záťaže v cloudových systémoch s konceptom zeleného počítania. Hlavné ciele práce sú zostaviť návrh riešenia pre zníženie energetickej náročnosti v rámci simulačného prostredia a reálneho cloudového prostredia. Pre overenie riešenia sme v každom prostredí pripravili zdroje systému, a vybrali testovacie úlohy spĺňajúce kritéria metodiky práce. Pri zostavovaní návrhu bolo nutné zohľadniť súčasné trendy, vlastnosti overovacieho prostredia a dohodu o úrovni poskytovaných služieb. Výsledkom práce je navrhnutý funkčný algoritmus vyrovnávania záťaže s implementovanými technikami zeleného počítania.

Kľúčové slová: cloudové počítanie, vyrovnávanie záťaže, zelené počítanie

ABSTRACT

VESEL, Eduard: *Dynamic load balancing in cloud systems with concept of green computing*. [Dissertation thesis]. – University of Žilina; Faculty of Management Science and Informatics; Department of Informatics. – Supervisor: doc. Ing. Jarmila Škrinárová, PhD. – Žilina: FRI UNIZA, 2020, 89 p.

The goal of the dissertation thesis is to analyze and present a solution of techniques for dynamic load balancing in cloud systems with the concept of green computing. The main objectives of the work are to compose solution draft to reduce energy intensity within simulation environment and the real cloud environment. To verify the solutions, we have prepared system sources in each environment, and we have chosen testing tasks that fulfilled criteria of the work methodology. When composing draft, it was necessary to consider current trends, attributes of verification environment and a service-level agreement to compose a draft. The result of the thesis is proposed functional load balancing algorithm with implemented techniques of green computing.

Key words: cloud computing, load balancing, green computing

Obsah

Zoznam obrázkov

Zoznam grafov

Zoznam algoritmov

Zoznam tabuliek

Zoznam vzorcov

Zoznam skratiek a pojmov

AEEDP	Algoritmus energetickej efektívnosti – dynamické plánovanie
AEEFL	Algoritmus energetickej efektívnosti – fuzzy logika
AWS	Amazon Web Services
BSLD	Bounded SLowDown / metrika ohraničeného spomalenia
COG	Center Of Gravity / ťažisková metóda
CLI	Command Line Interface / príkazový riadok
CPU	Central Processing Unit / centrálna procesorová jednotka
DVFS	Dynamic Voltage and Frequency Scaling / technika dynamického škálovanie napätia a frekvencie
EC2	Elastic Compute Cloud
FIS	Fuzzy inferenčný systém
GB	Gigabyte
GHz	Gigahertz
HPC	High Performance Computing / vysokovýkonné počítanie
Hypervizor	Technika virtualizácie hardvéru počítača
IaaS	Infrastructure as a Service / infraštruktúra ako služba
IQR	Interquartile Range / medzikvartilové rozpätie
Linpack	Súbor knižníc používaných na testovanie výpočtového výkonu počítačov
LR	Lokálna Regresia
LTS	Long-Term Support / dlhodobá podpora
MAD	Median Absolute Deviations / mediánová absolútna odchýlka
Mbit/s	Megabit per second / megabity za sekundu
MIPS	Million Instructions Per Second / milión inštrukcií za sekundu
MPI	Message Passing Interface / rozhranie na výmenu správ
NP	Nedeterministický Polynomiálny čas
PaaS	Platform as a Service / platforma ako služba
PB	Petabyte
PFlop/s	Peta Floating-point Operations per second / počet peta operácií v pohyblivej radovej čiarke za sekundu
RAM	Random Access Memory / pamäť s priamym prístupom
SaaS	Software as a Service / softvér ako služba

SLA	Service Level Agreement / dohoda o úrovni poskytovaných služieb
Snark	Netriviálny bezmostový kubický graf, ktorého hrany nemožno zafarbiť tromi farbami
SOA	Service-Oriented Architecture / servisne orientovaná architektúra
SP	Statický Prah
vCPU	virtual Central Processing Unit / virtuálna centrálna procesorová jednotka
VM	Virtual Machine / virtuálny stroj

Úvod

V dnešnej modernej dobe zohrávajú čoraz väčšiu dôležitosť pri vykonávaní náročných úloh aj počítačové technológie ako cloudové systémy, či už vo forme súkromného alebo verejného modelu nasadenia. Využitie cloudového počítania je stále prevažne v komerčnom sektore, ale v poslednej čase sa používa aj na výskum a vedecké výpočty. Zároveň sa dnes pri distribuovaných systémoch stretávame nielen s posúdením vhodnosti použitia virtuálnych alebo fyzických počítačových zdrojov, ale významným faktorom a javom je šetrnosť k životnému prostrediu.

Vo svete informačných technológií a v oblasti vysokovýkonného počítania je etablovaná myšlienka tzv. zeleného počítania. Algoritmy pre efektívne využitie dostupných výpočtových zdrojov sa používajú na dosiahnutie cieľa v rámci vysokovýkonného počítania, ktorý je skrytý pod pojmom zeleného počítania. V spojení s poskytovateľmi služieb v oblasti cloudového počítania je ďalším riešením, aj riešenie zabezpečenia energie dátového centra z obnoviteľných zdrojov.

Používanie virtuálneho prostredia a počítačových zdrojov, ktoré sú k dispozícii na diaľku, môže spôsobiť spomalenie riešených úloh. Na druhej strane, rýchla dostupnosť a cena použitia sú výhodou na overenie a nameranie výsledkov v relatívne primeranom čase.

Preto sme sa rozhodli skúmať práve problematiku a vplyv zeleného počítania v distribuovaných systémoch na vykonávané úlohy. Za týmto účelom sme si zvolili pre otestovanie a určenie vhodných základných spôsobov vyrovnávania pracovnej záťaže simulačné prostredie CloudSim a reálne prostredie cloudu v službe Amazon Web Services.

Začiatok dizertačnej práce uvádza stanovené ciele práce. Práca je rozdelená do 7 kapitol. Kapitola 1 je zameraná na cloudové a výpočtové systémy. Obsahuje definície prostriedkov systémov. Prináša informácie o počítačovom klastri a v súčasnosti najvýkonnejších superpočítačov sveta. V závere kapitoly sa zaoberáme možnosťou vedeckých výpočtov v cloudovom prostredí.

Vyrovnávanie záťaže, rozvrhovanie, migrácia a konsolidácia virtuálnych strojov je v kapitole 2. Tiež tu objasňujeme metódy nastavenia výkonu cloudu.

Tretia kapitola obsahuje metódy používané na znižovanie energetickej náročnosti cloudov – zelené počítanie.

Návrh nových algoritmov na znižovanie energetickej náročnosti je predstavený a opísaný v kapitole 4.

Skúmanie cieľov je obsiahnuté v kapitolách 5 a 6. V týchto kapitolách najprv objasňujeme zvolené metodiky a metódy riešenia pre jednotlivé cloudové prostredia. Ďalej definujeme testovacie úlohy, prípravu zdrojov a nameranie výsledkov.

Vyhodnotenie dosiahnutých výsledkov a budúci rozvoj v danej problematike je zahrnuté v kapitole 7.

Ciele práce

Na základe analýzy teoretických východísk, v oblasti manažovania cloudov, a aktuálneho stavu tejto problematiky doma a v zahraničí vyplýva, že problém zeleného počítania je v súčasnosti jednou z hlavných tém. Táto problematika zahŕňa, v rámci cloudového počítania, optimalizáciu času dokončenia úloh, efektívne a konsolidované virtuálne zdroje a efektívne, metódy znižovania spotreby elektrickej energie cloudových systémov.

V súlade s týmito poznatkami sme stanovili hypotézy:

- H1 - Cloudové systémy, na báze virtuálnych strojov, spomaľujú činnosť fyzických výpočtových prostriedkov, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme.
- H2 - Zelené počítanie v cloudových systémoch spomaľuje činnosť počítačového cloudu, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme.
- H3 - Dekompozícia cloudovej aplikácie negatívne ovplyvňuje náklady na prevádzku výpočtového systému.

Hlavné ciele pozostávajú z nasledovných krokov:

1. Návrh metód zeleného počítania a ich implementácia v jednotlivých cloudových prostrediach.
2. Výber testovacích úloh.
3. Vykonanie experimentu a overenie funkčnosti metodík vo zvolených prostrediach.
4. Vyhodnotenie nameraných výsledkov.

K dosiahnutiu stanovených cieľov používame simulačné aj reálne prostriedky služieb cloudového počítania.

1 Cloudové a výpočtové systémy

V súčasnosti sa zvyčajne používajú viacjadrové a viacprocesorové počítače. Za posledných niekoľko rokov sa výkon najvýkonnejších počítačov sveta niekoľkonásobne zvýšil. Preto je prirodzeným vývojom tvorba a implementácia paralelných a distribuovaných výpočtov, ale aj manažovanie systémov s prihliadnutím na čo najnižšiu spotrebu elektrickej energie týchto systémov. Hlavným dôvodom k masívnemu používaniu paralelných výpočtov je narastajúca časová zložitosť algoritmov a riešených problémov. Práve viacjadrové procesory umožnili veľký nástup paralelizácie. Úlohy na nich vykonávané, sa spracúvajú na viacerých jadrách, a nie len na niekoľkých jednotlivých procesoroch. Výhodou je, že komunikácia medzi jadrami býva rýchlejšia ako komunikácia medzi procesormi. Je praktickejšie a jednoduchšie použiť výkon takýchto procesorov, aj v počte tisícov, pre dosiahnutie žiadaného cieľového výkonu. [1, 44]

K zbytočnému časovému zdržaniu, ku ktorému prichádza pri náročných sekvenčných programoch, sa dá vyhnúť použitím paralelného programovania a ich výpočtami. Zrýchlenie algoritmu je dosiahnuté vykonávaním čiastkových úloh daného programu súbežne, pri zachovaní správnosti a funkčnosti programu. [2] Paralelný výpočet obsahuje procesy, ktoré môžu byť navzájom od seba nezávislé alebo závislé. Úlohy rozdelené na procesy, ktoré bežia na viacerých procesoroch simultánne, spolu komunikujú pomocou siete alebo zbernice. [45]

1.1 Zdroje a stroje počítačového systému

Zdroj je definovaný ako prvok počítačového systému s limitovanou kapacitou. Limitovaná kapacita vyjadruje napr. počet procesorov, veľkosť pamäte a iné. Zdroje slúžia na vykonávanie úloh a pracujú s určitou rýchlosťou. Rýchlosť zdroja určuje čas potrebný na vykonanie úlohy na tomto zdroji. Záťaž zdroja, v priebehu činnosti sa môže meniť, reprezentuje jeho využitie v danom časovom intervale. Rozlišujeme fyzické a virtuálne zdroje. Podľa charakteru vykonávanej úlohy môžeme zdroje rozdeliť pre spracovanie úlohy a vstupno-výstupné operácie. [44]

Naopak, výpočtový stroj, je množina kumulatívnych zdrojov (procesor, pamäť, úložisko). Opis výpočtového stroja zahŕňa jeho názov, množinu zdrojov a ich architektúru, operačný systém, periférie, atď. Taktiež je to kapacita, záťaž, rýchlosť a umiestnenie.

Virtuálny stroj je softvérová implementácia výpočtového stroja nad virtualizačnou vrstvou (hypervízorom), ktorý vykonáva operácia tak, ako fyzický stroj.

1.2 Počítačový klaster

Počítačový klaster je jedna logická jednotka pozostávajúca z dvoch alebo viacerých počítačov, ktoré sú navzájom prepojené pomocou lokálnej počítačovej siete, prípadne technológiou infiniband. [13] Sieťovo prepojené počítače sa v podstate prejavujú ako jeden, oveľa výkonnejší stroj.

Počítačový klaster ponúka omnoho viac výpočtového výkonu, úložnej kapacity, lepšej integrity dát, vyššiu spoľahlivosť a širšiu dostupnosť prostriedkov.

Počítačové klastre sú obvykle nasadzované pri potrebe zvýšenia výpočtovej kapacity alebo spoľahlivosti. [34, 35] Toto riešenie poskytuje väčšiu efektivitu ako by mohol poskytnúť samostatný počítač. Nanešťastie sú náklady počítačového klastra, na údržbu a implementáciu, väčšie v porovnaní s jediným počítačom.

Počítačové klastre sú budované za konkrétnym účelom, kde z pohľadu zamerania našej práce patria:

- *klaster pre vysokovýkonné počítanie*, označovaný ako High-Performance Computing (HPC). Jeho využitie pomáha k zvýšeniu výpočtového výkonu pomocou viacerých počítačov, ktoré na výpočte spoločne spolupracujú. Týmto spôsobom vznikne pomerne vysokovýkonný celok, ktorý je niekoľkonásobne lacnejší, než jeden vysokovýkonný počítač.
- *klaster s vyrovnávaním záťaže,* tiež často označovaný pod pojmom load balancer. Tento typ klastra určeného pre vyrovnávanie záťaže znižuje možnú mieru záťaže tým, že službu poskytuje niekoľko počítačov (serverov), ktoré majú rovnaký obsah (služba je teda poskytovaná paralelne). Rovnaký obsah je zabezpečení replikáciou obsahu medzi všetky prepojené počítače, alebo existenciou špecializovaného centrálneho úložiska.

Najnovší zoznam superpočítačov (z novembra 2019) uvádza, že najvýkonnejším superpočítačom sveta je americký superpočítač s označeným Summit, ktorý je umiestnený v DOE/SC/Oak Ridge National Laboratory v meste Oak Ridge. Celkový výkon nameraný v benchmarku Linpack je cez 148 PFlop/s a systém pozostáva z viac ako 2 410 000 jadier, ktoré sú osadené na procesore IBM POWER9. Veľkosť pamäte RAM predstavuje vyše 2,8 PB.

Druhý najvýkonnejší superpočítač na svete, americký Sierra, má nameraný celkový výkon v Linpack cez 94 PFlop/s. [5]

Tabuľka 1.1. Počet jadier a energetická náročnosť superpočítačov [6, 7]

Názov superpočítača	Počet jadier [v tis]	Spotreba [kW]
Summit	2 414	10 096
Sierra	1 572	7 438

1.3 Gridové a cloudové počítanie

Počítačové klastre môžu byť spolu prepojené do gridov. Počítačový grid je infraštruktúra, ktorá zahŕňa integrované a spolupracujúce použitie počítačov, počítačových sietí, databáz a vedeckých nástrojov vlastnených a manažovaných viacerými organizáciami. [8]

Podľa Tanenbauma [4] je distribuovaný systém „súbor nezávislých počítačov, ktorý sa javí svojim používateľom ako jeden ucelený počítač.“ Ide o „...jedno integrované výpočtové zariadenie, aj vtedy, ak je implementované viacerými počítačmi na rôznych miestach.“

Distribuovaný systém pozostáva zo súboru samostatných nezávislých počítačov, navzájom prepojených prostredníctvom počítačovej siete, ktoré sú vybavené distribuovaným operačným systémom. Pomocou tohto softvéru je umožnené počítačom, aby manažovali a zdieľali zdroje systému, hardvéru a dát.

Jedným z významných prostriedkov v distribuovanom počítaní, ktorý pomáha vedcom pri riešení projektov, je aj medzi verejnosťou populárna platforma Folding@home. [61] V Apríli 2020 dosiahla platforma v súvislosti s bojom zameranom proti chorobe COVID-19 (hľadanie molekúl, ktoré môžu pomôcť s vytvorením lieku) výkon cez 2,4 EFlop/s, čo predstavovalo vyšší výkon ako všetkých TOP500 superpočítačov sveta dohromady. [5, 62]

Rozširovať distribuovaný systém je možné z pohľadu viacerých komponentov: počet používateľov a procesov, maximálna vzdialenosť medzi uzlami a počet domén.

Medzi výhody distribuovaných systémov patrí prepojenie geografickej vzdialenosti, zlepšenie výkonu a dostupnosti pri zachovaní autonómie a znižovania nákladov.

Gridové počítanie je podľa Fostera [9] definované ako „*hardvérová a softvérová infraštruktúra, ktorá poskytuje spoľahlivý, konzistentný a lacný prístup*

k vysokovýkonným výpočtovým kapacitám“ a „*využívanie priameho prístupu k počítačom, softvéru, dátam a ostatným zdrojom podľa potreby. Takéto využívanie je nevyhnutne vysoko riadené poskytovateľom zdrojov a používatelia majú jasne definované čo je využívané, kto je oprávnený využívať zdroje a podmienky, za ktorých prichádza k využívaniu.“*

Cloudové počítanie sa vzťahuje na poskytovanie prístupu k výpočtovým prostriedkom cez širokopásmovú sieť.

Definícia cloudu podľa Buyya a kolektív [10, 11]: *"Cloud je paralelný a distribuovaný výpočtový systém, ktorý sa skladá z množiny vzájomne prepojených, virtualizovaných a dynamicky spravovaných počítačov. Prostriedky cloudu je možné dynamicky prenajímať ako jeden alebo viac unifikovaných výpočtových zdrojov, na rôznej úrovni služieb SLA a podľa dohovorov medzi poskytovateľmi služieb a klientami."*

Gridové počítanie má stále dôležité a významné postavenie pri riešení vedeckých úloh (napr. CERN Worldwide LHC Computing Grid), ale aktuálnym cieľom sa stáva preniesť riešenie významných vedeckých úloh aj do kombinácie služieb cloudu.

Tabuľka 1.2. Porovnanie techník cloudového a gridového počítania [58]

Parameter	**Gridové počítanie**	**Cloudové počítanie**
Cieľ	Spolupráca pri zdieľaní zdrojov	Použitie služby
Účel	Výpočtovo intenzívne operácie	Štandardné, verejné, ale aj vedecké účely
Riadenie pracovného toku	V jednom fyzickom uzle	U poskytovateľa služieb
Úroveň abstrakcie	Nízka	Vysoká
Stupeň škálovateľnosti	Normálny	Vysoký
Prostredie	Heterogénne	Homogénne
Čas spustenia	Nie v reálnom čase	V reálnom čase
Operačný systém	Hocijaký štandardný OS	Hypervizor (VM), na ktorom sa spúšťajú viaceré OS
Zdroje	Obmedzené (HW je obmedzený)	Neobmedzené
Užívateľsky prístupný	Málo	Veľmi
Typ služby	CPU, sieť, šírka pásma, úložisko, ...	IaaS, PaaS, SaaS, všetko ako služba

Príklad z reálneho sveta	SETI, BOINC, Folding@home, GIMPS	Amazon Web Service (AWS), Google app
Doba odozvy	Nedá sa obslúžiť naraz v čase a musí byť naplánovaná	V reálnom čase
Konfigurácia	Zložitá, pretože používatelia nemajú administrátorské práva	Jednoduchá konfigurácia
Budúcnosť	Cloudové počítanie	Ďalšia generácia internetu

Dohodu o úrovni poskytovaných služieb (angl. Service Level Agreement, SLA) poskytuje prevádzkovateľ infraštruktúry. SLA vyjadruje záväzok poskytovať určitú kvalitu služieb. SLA zvyčajne zahŕňa záruky dostupnosti, výkonu alebo špecifikuje minimálny čas, ktorý bude systém k dispozícii používateľovi počas určitého obdobia. [11]

Počítačové cloudy sú prostriedky poskytované verejnosti na požiadanie, označované ako služba, alebo ako model nasadenia privátneho, príp. hybridného cloudu. [33] Ak potrebujeme mať k dispozícii rozsiahle počítačové prostriedky, môžeme ich získať okamžite bez toho, aby sme boli nútení investovať do novej infraštruktúry. Prístup k prostriedkom je nezávislý od miesta kde sa nachádzame, pretože k požadovaným prostriedkom pristupujeme cez počítačovú sieť internet.

Základom cloudového počítania sú štyri nosné prvky:

- Hardvér a nástroje pre virtualizáciu hardvéru.
- Pokročilé internetové technológie a služby, SOA a cloud computing môžu existovať aj samostatne, ale v kombinácii spolu sú veľmi pomocné. Použitie SOA v rámci cloud computingu predstavuje vzor pre vývoj distribuovaných systémov, ktoré premieňajú zdroje do softvérových služieb.
- Distribuované počítanie, ktoré pomáha riešiť rozsiahle problémy s ich rozdelením medzi skupiny iných počítačov, ktoré pracujú na riešení v rovnakom čase podobne ako grid.
- Autonómne manažovanie systémov.

Virtualizácia vo svete počítačov znamená technológiu, pomocou ktorej je možné dosiahnuť vytvorenie virtuálnej verzie reálnych zariadení ako napr. servery, úložné miesta alebo siete. Rovnako je možné vytvoriť virtualizované prostredie (virtualizovanie platforiem), ktorým je celý operačný systém. Týmto spôsobom sa dajú virtualizovať operačné systémy ako Microsoft Windows, Android, Linux, ktoré sú

nezávislé od hostiteľského operačného systému a môžu využívať dostupné periférne zariadenia.

Virtualizácia medzi operačným systémom a hardvérom predstavuje softvérovú vrstvu, nazývanú hypervizor, ktorá rozdeľuje zdroje do jedného alebo viacerých nových nezávislých prostredí.

S neustále sa zväčšujúcou komplexnosťou počítačových systémov, sa stalo prioritou výskumu autonómne manažovanie systémov. Cieľom je zlepšiť systém znížením angažovanosti ľudského faktora, a dosiahnuť, aby sa systémy manažovali sami.

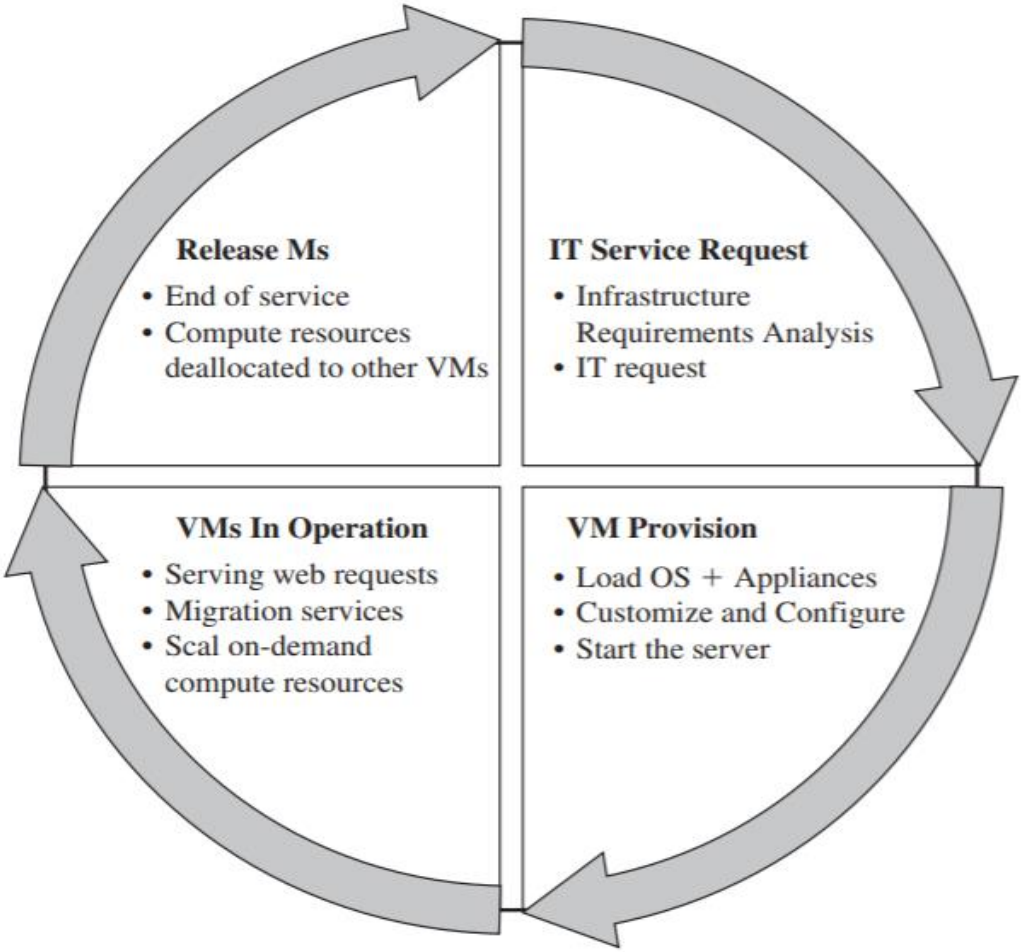
Na druhú stranu, hlavnou nevýhodou používania cloudového počítania je strata kontroly nad hardvérovými prostriedkami, posielanými dátami alebo problémami so zaťaženým komunikačnej siete. [12]

Škálovateľnosť počítačových cloudov je daná ich elasticitou. To predstavuje vyrovnávanie zaťaženia aplikačných inštancií, ktoré bežia oddelene na rôznych operačných systémoch. Procesory a šírka pásma siete je pridelená a odobraná na požiadanie. Systémové kapacity sa dajú prispôsobovať v závislosti od počtu užívateľov, inštancií a množstvo prenesených dát v danom čase. Pri cloudoch preto hovoríme o teoreticky neobmedzenej škálovateľnosti.

1.4 Virtuálne stroje

Virtualizácia hardvéru slúži na spúšťanie viacerých operačných systémov a softvéru na jednom fyzickom stroji. Virtualizáciou umožníme vytvorenie viacerých virtuálnych strojov. Každý virtuálny stroj musí obsahovať operačný systém, programy a aplikačný softvér. [10]

Životný cyklus virtuálneho stroja pozostáva z niekoľkých fáz (Obrázok 1.1). Cyklus začína doručením požiadavky poskytovateľovi infraštruktúry (*IT Service Request)*. Ten zabezpečí fyzické počítačové zdroje, ktoré budú spĺňať požiadavky tak, aby bolo možné začať poskytovať požadovaný virtuálny stroj. Následne prevádzkovateľ vytvorí virtuálny stroj (*VM Provision)*. Akonáhle je virtuálny stroj spustený, je pripravený poskytnúť požadovanú službu podľa SLA (*VMs In Operation*). Na konci životného cyklu je virtuálny stroj uvoľnený a výpočtové zdroje sú dostupné pre ostatné virtuálne stroje (*Release VMs)*. [11]



Obrázok 1.1. Životný cyklus virtuálneho stroja [1]

1.5 Vedecké výpočty v cloudovom prostredí

K rozhodnutiu počítať paralelné úlohy v cloudovom prostredí treba pristupovať individuálne a zohľadniť aj zameranie úloh. Úlohy obsahujúce intenzívne výpočty majú zvyčajne veľkú réžiu na medziprocesorovú komunikáciu a často nie sú dobre škálovateľné. [47, 48]

Skúmaniu prenosu aplikácie určenej pre HPC do cloudového prostredia sa venovali vo svojej práci Balis a kolektív. [27]

Prínosom ich štúdie je metóda prenosu pamäťovo náročnej aplikácie, ktorá umožňuje paralelizáciu pri zachovaní pomeru času komunikácie ku času výpočtu. Zároveň dosahuje uspokojivý výkon na základe výsledkov, nameraných pri nasadení v rámci cloudovej infraštruktúry. Autori dosiahli dobrý výsledok vďaka významnému zníženiu nákladov na komunikáciu (pozri kapitolu 1). Cieľovým prostredím pre experiment je použitie infraštruktúry ako služba (IaaS), ktorá umožňuje dynamické

vytváranie a prideľovanie výpočtových uzlov na požiadanie. Pre redukciu pomeru časov komunikácie k výpočtom sú použité vhodné metódy rozdeľovania, komunikácie, aglomerácie a mapovania.

Hlavnou myšlienkou je nahradiť jeden náročný výpočet veľkým počtom jednoduchších, ktoré je možné vykonávať súčasne. To má za následok jemnozrnný paralelizmus. Redukovať komunikáciu a minimalizovať jej vplyv na spracovanie v distribuovanej infraštruktúre vyžaduje aglomeráciu spomínaných jednoduchších úloh. Je použitý prístup aglomerácie, ktorý vyvažuje využitie pamäti. Hlavne preto, že sa tu vyskytuje veľa jednoduchých úloh, pre ktoré sú požiadavky na pamäť dominantným faktorom. Heuristická aglomerácia odhaduje potrebné množstvo pamäte pre spracovanie podstromu daného stromu, a tým vytvára súhrn podobných úloh.

Následné mapovanie prebieha na základe vstupného zoznamu úloh *tasks* (s hodnotami id úlohy, veľkosť úlohy, stav agregácie), kde veľkosť úlohy je vyjadrená v bajtoch a stav agregácie označuje, či je úloha označená ako aglomerovaná. Funkcia alokovania úloh *allocateTasks* vykonáva redukciu úloh zo vstupného zoznamu operáciou *allocTask* a inicializovanou premennou *emptyAllocation*. Hlavným výsledkom algoritmu sú dva zoznamy:

1. zoznam priradených úloh *taskAlloc* (s hodnotami id úlohy, vybraný stroj)
2. zoznam pridelenej pamäti *memAlloc* (s hodnotami id stroja, priradená veľkosť pamäti).

Redukcia je prechádzanie zoznamu a alokovanie každej úlohy spôsobom, že sa vyberie prvý stroj, *selMachine*, s minimálne alokovanou pamäťou. (Algoritmus 1.1)

$$
\begin{aligned}
&\text{Input: } \textbf{\textit{tasks}} = \{((0, 1000000B, Aggregate), (1, 10000B, NonAggregate), \ldots\}\\
&\text{function } \textbf{\textit{allocateTasks}}(machineCount, tasks)\\
&\qquad = foldl(allocTask, emptyAllocation, tasks)\\
&\text{where } \quad \textbf{\textit{emptyAllocation}} = (\{\}, emptyMemAllocation)\\
&\qquad \textbf{\textit{emptyMemAllocation}} = 1..machineCount \times \{0\}\\
&\text{function } \textbf{\textit{allocTask}} \quad ((taskAlloc, memAlloc), (id, memReq, isAggregate))\\
&\qquad\qquad = (taskAlloc', memAlloc')
\end{aligned}
$$

$$
\begin{aligned}
&\text{where } \quad \textbf{\textit{taskAlloc}}' = \{(id, selectedMachine)\} \cup taskAlloc\\
&\qquad \textbf{\textit{memAlloc}}' = \left\{(mId, newMem(mid, mem)) \mid (mId, mem)\right.\\
&\qquad\qquad \left. \in memAlloc\right\}\\
&\qquad \textbf{\textit{newMem}}(mId, mem) = \begin{cases} mem + memReq & mid = selMachine \\ mem & otherwise \end{cases}\\
&\qquad \textbf{\textit{selMachine}} = \begin{cases} fst(minBySnd(memAlloc)) & isAggregate \\ 1 & otherwise \end{cases}
\end{aligned}
$$

Algoritmus 1.1. Algoritmus mapovania [27]

Výpočtové prostredie je vytvorené v rámci služby AWS. Čas spracovania je meraný na rôznych konfiguráciách v počtoch od 8 až do 64 inštancií. Výskum dokazuje, že zatiaľ čo cloudové počítanie nie je také efektívne z hľadiska „hrubého“ výkonu ako HPC, môže ale poskytnúť lepší čas na poskytnutie požadovaných inštancií. Časy dostupnosti požadovaných strojov v prostredí HPC závisia od aktuálneho počtu používateľov a ich veľkostí úloh (Tabuľka 1.3).

Tabuľka 1.3. Stredná doba poskytnutia nových inštancií [27]

Prostredie	Počet inštancií	Čas (s)
HPC (čas výpočtu 2 minúty)	64	1000
HPC (čas výpočtu 5 hodín)	256	> 10000
NERSC výpočtové centrum	-	1200
Amazon EC2	64	< 250

Okrem toho cloudové počítanie poskytuje príležitosť dekomponovať problém na veľký počet inštancií, ktoré umožňujú riešiť rozsiahle problémy, vyžadujúce veľké množstvo pamäti, ktorá by nemohla byť k dispozícii na iných infraštruktúrach.

Z pohľadu celkových finančných nákladov nie je vhodné prenajať si v cloude identickú infraštruktúru, aká sa používa pri fyzických serveroch pre vysokovýkonné počítače. [21, 22] Celkovú výhodnosť cloudovej infraštruktúry ovplyvňuje aj model rezervovania požadovaných inštancií. Oproti modelu pay-as-you-go (priebežný systém platenia) ide o inštancie na požiadanie, rezervované alebo predplatené dopredu.

2 Vyrovnávanie záťaže systému

Záťaž počítača predstavuje množinu úloh, ktorá sa na tomto počítači vykonáva. Záťažou sú predovšetkým úlohy a pracovné toky úloh. Keďže virtuálny stroj je softvérovou implementáciou fyzického stroja, tiež predstavuje záťaž.

Úloha (angl. job, task) je základná jednotka, ktorá sa vykonáva na zdroji. Úloha môže mať špecifické požiadavky na množstvo a rôzne charakteristiky zdrojov, ku ktorým má byť pridelená súvisle alebo na určité časové intervaly, v ktorých ich bude používať. [10]

Úlohu J_j môžeme charakterizovať pomocou týchto pojmov a označení:

- p_j – čas spracovania úlohy (angl. processing time). Čas spracovania úlohy *j* na stroji *i*, označuje sa ako p_{ij}. Ak čas spracovania úlohy nie je známy, treba ho odhadnúť. Odhady sa robia na základe údajov z predchádzajúcich behov danej úlohy, nad množinou vstupných údajov.
- r_j – čas dostupnosti úlohy (angl. release date). Je to čas kedy je úloha pripravená na spracovanie (vrátane vstupno/výstupných operácií). Čas dostupnosti môže byť presne stanovený (napr. pri statickom rozvrhovaní alebo alokovaním na určitý čas) alebo môže byť neskôr stanovený systémom.
- d_j – termín ukončenia úlohy (angl. due date) – čas, kedy bude úloha ukončená a to aj v prípade, že nie je ešte spracovaná.
- w_j – váha úlohy (angl. weigth) – označuje dôležitosť úlohy. Váha môže vyjadrovať prioritu lokálnej úlohy pred globálnou alebo príslušnosť jej majiteľa k určitej skupine používateľov, ktorí majú prioritu pred ostatnými.
- s_j – čas zavedenia úlohy (angl. set up time) – čas potrebný na zavedenie úlohy do systému, získanie vstupných dát alebo čas potrebný na spojenie s požadovanou knižnicou. Tento čas môže závisieť od postupnosti predchádzajúcich úloh a v takom prípade s_{ij} označuje čas potrebný pre zavedenie úlohy *j* po dokončení úlohy *i*.
- S_j– čas začiatku spracovania úlohy (angl. start time) – čas, kedy sa úloha skutočne začne spracovávať **(**$S_j \geq r_j$**).**
- C_j – čas dokončenia úlohy (angl. completion time) – čas, kedy sa skončí spracovanie úlohy *j* na stroji *i*.

Stroje môžu pracovať rôznou rýchlosťou, preto čas spracovania úlohy *j* je potrebné vzťahovať na stroj *i*. Potom čas spracovania úlohy *j* na stroji *i* vyjadruje vzťah $p_{ij} = C_{ij} - S_{jj}$. [10]

Vyrovnávanie záťaže sa používa na distribúciu a rozloženie väčšieho pracovného zaťaženia uzla na menšie zaťaženie, a tým dosiahnuté zvýšenie celkového výkonu systému. V prostredí cloudového počítania je potrebné dynamicky vyrovnávať miestnu záťaž rovnomerne medzi všetky dostupné uzly. [3]

Medzi základné spôsoby vyrovnávania pracovnej záťaže v cloudových systémoch patrí rozvrhovanie, migrácia úloh a virtuálnych strojov, replikácie a elasticita.

Migrácia úlohy je definovaná ako prenos úlohy zo súčasného tzv. zdrojového uzla na cieľový miesto a pokračovanie jej vykonávania tam. Môže ísť aj o prenos už rozpracovaných úloh.

Existujú dva aspekty spojené s migráciou úloh:

- hľadanie vhodných výpočtových zdrojov s dostatočnou kapacitou na presun úlohy,
- definovanie samotného algoritmu migrácie.

Replikácia sa používa skôr v menších distribuovaných systémoch. Predstavuje pripravené kópie úlohy a dát na niektorých zvolených výpočtových uzloch vopred bez presnejšieho poznania, kde ich bude treba. Dobre navrhnutá replikácia vo všeobecnosti vedie k:

- zníženiu sieťovej komunikácie,
- vyššej odolnosti systému.

Herbst a kolektív [49] definuje, že elasticita „je rozsah, v ktorom je systém schopný prispôsobovať zmeny zaťaženia, pridávaným a odoberaným zdrojov autonómnym spôsobom tak, aby v určitých časových intervaloch dostupné zdroje zodpovedali čo najpresnejšie záťaži počítačového systému.“

Elastický klaster predstavujú služby cloudu (komerčné účely), gridu a superpočítača (vedecké účely).

Kansal a Chana [57] definujú, že “zelené počítanie je praxou implementovania politík a postupov, ktoré zlepšujú efektívnosť výpočtových zdrojov takým spôsobom, aby bolo možné znížiť spotrebu energie a vplyv na životné prostredie ich využívaním.”

Z pohľadu manažovania systému na báze zeleného počítania v tejto práci uvažujeme také základné spôsoby vyrovnávania záťaže, ktoré pri dodržaní SLA majú čo najnižšiu spotrebu elektrickej energie.

Zelené počítanie je možné efektívne implementovať do niekoľkých základných spôsobov vyrovnávania záťaže. Ide o:

- Rozvrhovanie - efektívne prideľovanie nových úloh na zdroje.
- Migráciu virtuálnych strojov.
- Konsolidáciu virtuálnych strojov.
- Metódy nastavenia výkonu cloudu.

2.1 Rozvrhovanie

Rozvrhovanie predstavuje najdôležitejší model ako vyrovnávať záťaž.

Rozvrh je priradenie spracovania úloh zdrojom na určitý časový interval tak, že žiadne dve úlohy sa nevykonávajú súčasne na tom istom zdroji a kapacita zdroja nie je prekročená. Rozvrh určuje pre každý časový okamih množinu úloh, ktoré sa v tom okamihu majú vykonávať a množinu zdrojov, na ktorých sa majú vykonávať. Najdôležitejšie kritériá optimálnosti pre rozvrh φ , ktoré treba minimalizovať sú:

- Čas dokončenia poslednej úlohy C_{max} , označovaný tiež ako maximálny čas konca úloh (angl. makespan), $C_{max} = \max_j\{C_1, \dots, C_n\}$ pre rozvrh φ je $C_{max}(\varphi) = \max_{j \in J}\{C_j\}$. Toto kritérium si zasluhuje veľkú pozornosť, prestavuje dobu spracovania celej vstupnej množiny úloh a tým aj dĺžku rozvrhu.
- Celkový čas dokončenia všetkých úloh vypočítame ako súčet časov úloh $\sum_{j=1}^{n} C_j$ alebo, ak úlohy majú rôzne váhy počítame vážený súčet časov úloh $\sum_{j=1}^{n} w_j C_j$.

Optimalizovaním týchto kritérií dokážeme výrazne šetriť elektrickú energiu, pretože čím kratšia je doba výpočtu, tým sa spotrebuje menej energie. Optimalizovaním týchto kritérií s kombinácii so systematickou a automatizovanou konsolidáciou a migráciou virtuálnych strojov dosiahneme významný prínos z hľadiska zeleného počítania.

Treba však upozorniť na fakt, že nájdenie optimálneho rozvrhu je NP – ťažký problém [47, 48, 50, 51], t.j. problém, ktorý sa nedá vyriešiť v rozumnom čase, ak

neriešime triviálnu úlohu. Cieľom je nájsť suboptimálne riešenie v polynomiálnom čase. Pri riešení často využívame obmedzenia prirodzene vyplývajúce z typu úlohy. Prehľadávame okolia už nájdených prípustných riešení, pretože tie často obsahujú lepšie riešenie a tiež striedame vstupné body pri riešení problému (napr. prvé prípustné riešenie, z ktorého sa vychádza pri hľadaní ďalších riešení), a tým preskúmavame oblasti potenciálne dobrých riešení. [44, 52]

2.2 Migrácia virtuálnych strojov

Pri virtuálnych strojoch sa preferuje, aby neboli závislé na jednom fyzickom hostiteľovi, a v prípade potreby bolo možné s nimi hýbať medzi rôznymi hostiteľmi. Táto prispôsobivosť prináša nové možnosti v zlepšení využívania cloudových aplikácií tým, že sa v systéme nájde vhodné umiestnenie virtuálnych strojov. [16]

Migrácia virtuálneho stroja umožňuje prenos virtuálneho stroja medzi fyzickými uzlami bez prerušenia a s krátkym výpadkom. Migrácia má však negatívny vplyv na výkon aplikácií bežiacich vo virtuálnych strojoch počas migrácie. [19] Migrácia virtuálneho počítača je otázkou milisekúnd, čo šetrí čas, námahu a umožňuje dosahovať plnenie SLA.

Čas migrácie a zhoršenie výkonu zaznamenané na virtuálnom stroji j (2.1) vyjadrujeme, [16]

$$T_{mj} = \frac{M_j}{B_j} \qquad U_{dj} = 0{,}1 \times \int_{t_0}^{t_0 + T_{mj}} u_j(t)dt \tag{2.1.}$$

kde U_{dj} je celkové zhoršenie výkonu virtuálneho stroja j, t_0 je čas kedy začala migrácia, T_{mj} je uplynutý čas vykonania migrácie, $u_j(t)$ je využitie procesora virtuálnym strojom j, M_j je množstvo pamäte použitej virtuálnym strojom j, a B_j je dostupná šírka siete.

Ako ukazuje Obrázok 2.1, správne vykonaná migrácia virtuálnych strojov prispieva k znižovaniu energetickej náročnosti systému a nákladov, a to konsolidáciou z málo využívaných uzlov na iné uzle. Takto uvoľnené a nepoužívané systémy môžu byť následne vypnuté. Ako spomíname pri horizontálnom škálovaní (pozri kapitolu 2.4.3), môžu sa vyskytnúť problémy s celkovými nákladmi migrácie.

Beloglazov [19] definuje celkové náklady tak, že zahŕňajú náklady spôsobené porušením SLA a náklady na dodatočnú spotrebu energie. Dodatočná spotreba energie je energia spotrebovaná cieľovým uzlom, na ktorý sa migruje virtuálny stroj, a energia spotrebovaná zdrojovým uzlom po začiatku porušenia SLA. Vzorec (2.2) vyzerá nasledovne,

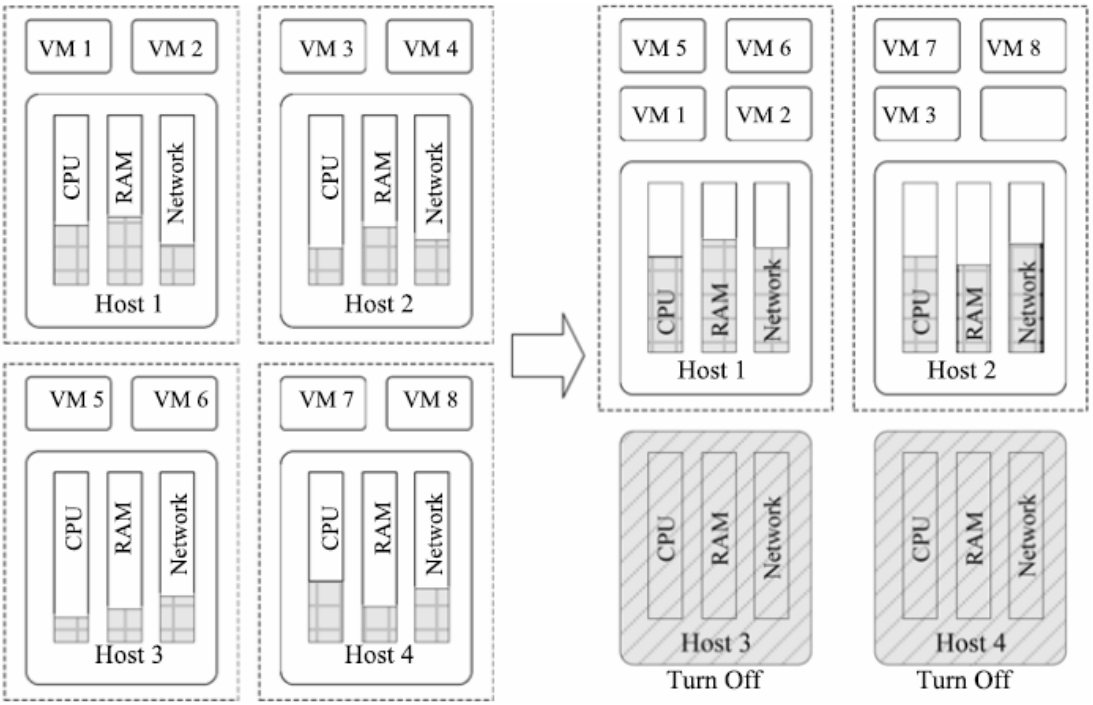
$$CO(v, m) = \begin{cases} (v - m)CO_p & \text{if } m < v,\ v - m \geq T, \\ (v - m)CO_p + 2(m - v + TO)CO_p + (m - v + TO)CO_v & \text{if } m \leq v,\ v - m < T, \\ r_tCO_p + (r - m + v)CO_p + r_tCOv & \text{if } m > v. \end{cases} \tag{2.2.}$$

kde *CO* funkcia nákladov, *v* porušenie SLA za jednotku času, *m* počet virtuálnych strojov priradených hostiteľovi, CO_p náklady na energiu za jednotku času, *TO* čas migrácie virtuálneho stroja, rCO_p náklady na energiu spotrebované hlavným hostiteľom od začiatku porušenia SLA, r_t zostávajúci čas od začiatku porušenia SLA a CO_v sú náklady na porušenie SLA za jednotku času.

Z rovnice funkcie nákladov sú definované tri prípady, ktoré pokrývajú všetky možné vzťahy medzi porušením SLA za jednotku času a počtom virtuálnych strojov priradených hostiteľovi:

- CO_1 opisuje prípad, keď k migrácii príde pred výskytom porušenia SLA, ale migrácia sa začne najneskôr pred začiatkom porušenia SLA.
- CO_2 popisuje prípad, keď k migrácii príde pred výskytom porušenia SLA, ale migrácia sa začne neskôr ako je začiatok porušenia SLA.
- CO_3 popisuje prípad, keď sa migrácia začne po začiatku porušenia SLA.

Avšak, kvôli migráciám virtuálnych strojov alebo ich neoptimalizovanému prideleniu, môžu spolukomunikujúce virtuálne stroje skončiť na rôznych logických jednotkách vzdialených fyzických uzloch. Tie následne medzi sebou predstavujú nákladné dátové prenosy. Ak sú komunikačné virtuálne počítače pridelené na uzly v rôznych stojanoch, sieťová komunikácia môže zahŕňať pridané sieťové prepínače a prepojenia, ktoré spotrebúvajú ďalšie značné množstvo energie. [17]



Obrázok 2.1. Optimalizácia z pohľadu využitia zdrojov [18]

K metódam a technikám konsolidácie virtuálnych strojov sa venujeme v nasledujúcej kapitole 2.3.

2.3 Konsolidácia virtuálnych strojov

Z dôvodu dynamického vytvárania a ukončovania virtuálnych strojov v cloudových datacentrách, vzniká na fyzických serveroch fragmentácia zdrojov. Táto fragmentácia vedie k zhoršeniu využívania prostriedkov daného servera. Riešením spomenutých problémov je konsolidácia virtuálnych strojov. Pomocou konsolidácie sa aktívne virtuálne stroje balia na minimálny počet fyzických serverov. Do úvahy sa berie splnenie cieľov, ktorými sú zníženie nákladov, úspora energie a maximalizácia využívania dostupných prostriedkov fyzických serverov.

Ferdaus a kol. [24] sa vo svojej práci zamerali na súčasné techniky konsolidácie virtuálnych strojov. Týmito technikami sú First Fit Decreasing, Best Fit a Best Fit Decreasing.

First Fit Decreasing je klasickým zásobníkovým „baliacim“ algoritmom, kde sú položky usporiadané v klesajúcom poradí, a v tomto poradí je každá ďalšia položka vždy umiestnená do prvého zásobníka, kam sa zmestí. [53] Dva spôsoby triedenia virtuálnych strojov, podľa dostupných jadier alebo pamäte, môžu byť vzostupnom

alebo klesajúcom poradí. Roztriedenie kvôli navýšeniu množstva voľných jadier alebo pamäte, a výberu prvého počítača, na ktorý je efektívne vhodné migrovať virtuálny stroj, implementujú prístupy Best-Fit, resp. Best Fit Decreasing. [54]

2.4 Metódy nastavenia výkonu cloudu

V prípade výskytu, alebo ešte pred výskytom výkonnostného problému systému, je potrebné správne spustiť regulačný proces aktívnych zdrojov, tak aby boli splnené kritériá pracovnej záťaže a aplikácií.

Ako uvádza Moghaddam [46], pre zabezpečenie a kontrolu výkonnostných požiadaviek existujú rôzne riešenia zahrňujúce zmeny v nastavení zdroja, pridanie nových výpočtových zdrojov (virtuálnych strojov) alebo náhradu existujúcich výpočtových zdrojov za výkonnejšie. Poskytovatelia cloudových služieb prispievajú ku všetkým typom rozhodnutí zohľadňujúcich aplikáciu SLA a dosiahnutia cieľov, akými sú rozpočtové a bezpečnostné obmedzenia počas rozhodovacieho proces.

2.4.1 Metódy identifikácie problematických aplikácií v cloude

Vlastnosti funkcií nachádzajúcich sa v tejto kategórii nie sú priamo špecifické len pre cloudové systémy, pretože sú aplikované priamo do aplikácií a ich prostredia. Na degradáciu výkonu operačného systému a aplikácií v dôsledku interných problémov, ktorými môžu byť poškodenie údajov, vyčerpanie zdrojov operačného systému alebo akumulácia číselných chýb, sú zamerané metódy aplikačnej úrovne. [36]

Pokus o identifikáciu problematickej aplikácie alebo systémového komponentu, vyčistenie interného stavu alebo reštartovanie komponentu je softvérovou možnosťou pri týchto typoch problémov. Na tejto úrovni môže byť tiež spustená aktivácia dynamických komponentov, kedy deaktivácia voliteľných komponentov napomáha systému riadiť vyššie záťaže alebo šetriť energiu, podľa stanovených požiadaviek SLA. [37]

2.4.2 Nadmerné rezervy

Celkový výkon systému a kvalita služieb ponúkaných aplikácií, ovplyvňuje riadny odhad požadovaných zdrojov a efektívne využitie ponúkaných cloudových zdrojov. Špeciálne pri systémoch s dynamickou záťažou a časovo závislými výkyvmi je odhad

zdrojov komplexným problémom. Na vyriešenie možnej nestability systému a zvládnutie zdrojových požiadaviek aplikácií, je poskytnutie statického dostatku zdrojov, ktoré majú zvládať predpokladané maximá záťaže systému. Nevýhodou tohto riešenia z pohľadu poskytovateľov cloudových služieb, celkových nákladov a energetickej náročnosti je, že vrcholy pracovnej záťaže sú zvyčajne len dočasnými udalosťami, ktoré sa vyskytujú zriedka a netrvajú dlho. To znamená, že väčšinu času sú alokované zdroje nevyužité.

V určitých špecifických vlastnostiach dynamických aplikáciách napr. webové záťaže, nie je vždy jednoduché sa vyhnúť nadrozmerným rezervám zdrojov v systéme odolnom voči poruchám. Napriek tomu sa výskumníci snažia nájsť vhodný kompromis medzi toleranciou úrovne odolnosti voči poruchám a nadmernými rezervami zdrojov tak, aby bola dosiahnutá väčšia kontrola nad funkčnosťou systému. [38]

2.4.3 Metódy automatického škálovania

Škálovateľnosť je schopnosť systému, aplikácie alebo virtuálneho stroja riadiť zmenené požiadavky na zdroje. Toto je jedna z najcennejších a prevažujúcich funkcií cloudového počítania. Vďaka škálovateľnosti môžeme v rámci virtuálneho stroja zväčšiť diskovú kapacitu pre ukladanie údajov, veľkosť operačnej pamäte RAM alebo počet dostupných procesorov, aby vyhovovali dynamicky sa meniacim požiadavkám a SLA kritériám. Z pohľadu systému môžeme meniť celkový počet virtuálnych strojov. Rozlišujeme niekoľko prístupov ku škálovateľnosti:

- *Vertikálne škálovanie* – je schopnosť elasticity virtuálneho stroja pridať zdroje za chodu na prispôsobenie sa zvyšujúcemu objemu pracovnej záťaže. [1, 2] Zvyšovanie požadovaných zdrojov je dosiahnuté škálovaním virtuálneho stroja alebo presunom aplikácie na väčšie (výkonnejšie) virtuálne stroje nasadené v počítačovom cloude. Ak je počet zdrojov, ktoré majú byť pridané väčší ako dostupné zdroje na fyzickom hostiteľovi, môže byť zvolený nový hostiteľ s dostupným množstvom zdrojov a poskytnúť požadované zdroje. V takom prípade hovoríme o migrácii virtuálneho stroja (pozri kap. 2.2).
- *Horizontálne škálovanie* – je pridanie novej jednotky (virtuálneho stroja) k existujúcej infraštruktúre, aby sa prispôsobili zvýšené objemy pracovnej záťaže. Pridaný virtuálny stroj môže mať prispôsobenú konfiguráciu svojich

zdrojov alebo byť pridaný ako predkonfigurovaná inštancia od poskytovateľa cloudových služieb. Spoločnosť Amazon so službou Amazon Web Services, (jedno z použitých reálnych prostredí pre naše overovanie), ponúka niekoľko kategórií a typov inštancií. Medzi hlavné kategórie patria inštancie všeobecného účelu a optimalizované pre (vysokovýkonné) počítanie, ukladanie dát alebo výkonu operačnej pamäte. [3] Pri aktívnom spúšťaní virtuálneho stroja je dôležitý údaj jeho štartovací času. Ak by bol tento čas spustenia dlhší než bol predpokladaný bod možného prerušenia SLA, účinok škálovacieho procesu bude rovnaký ako pri neaktívnej stratégii. Na rozdiel od vertikálneho škálovania, môže byť horizontálnym škálovaním dosiahnutý zvýšený výkon spolu so zvýšenou úložnou kapacitou.

- *Diagonálne škálovanie* - pomáha kombinovať škálovanie smerom „hore a dole“. Ako naznačuje tento termín, znižovanie je odoberanie nevyužitých výpočtových zdrojov pri znižovaní požiadaviek. Diagonálne škálovanie poskytuje flexibilitu pri pracovných zaťaženiach, ktoré vyžadujú ďalšie zdroje pre konkrétne časové prípady.

3 Zelené počítanie – metódy používané na znižovanie energetickej náročnosti cloudov

V tejto kapitole sa venujeme všeobecne známym a najpoužívanejším algoritmom prospievajúcich k dosiahnutiu myšlienky zeleného počítania.

Existuje niekoľko techník a metód, založených na fyzickom alebo softvérovom princípe, ktoré prispievajú k znižovaniu energetickej náročnosti systémov tým, že:

- Definujú modely spotreby elektrickej energie v cloudoch.
- Detegujú preťažené uzly.
- Predpovedajú zaťaženie uzlov.

3.1 Pravidlo statického prahu

Medzi najjednoduchší detekčný algoritmus preťaženia uzla sa považuje pravidlo statického prahu. [32] Uzol sa považuje za preťažený, ak procesorové využitie aktuálneho uzla prekročí špecifikovaný prah využitia.

3.2 Dynamické škálovanie napätia a frekvencie

Spotreba energie výpočtových uzlov v dátových centrách je v zásade vyjadrená spotrebou CPU, diskovým úložiskom a sieťovým rozhraním. V porovnaní s ostatnými systémovými zdrojmi, značnú časť z tejto energie spotrebováva CPU. Okrem toho, využitie CPU je často úmerné celkovému systémovému zaťaženiu. Riadenie spotreby energie a efektívne využitie je možné ovplyvniť technikou dynamického škálovanie napätia a frekvencie (angl. dynamic voltage and frequency scaling, DVFS).

Beloglazov a kol. [18] poukázali, že aplikovanie techniky DVFS na CPU končí v takmer lineárnom vzťahu spotreba-frekvencia pre server (so zvýšenou frekvenciou súvisí zvýšená spotreba, a naopak). Je to výsledkom toho, že DVFS je aplikovaný len na CPU a nie ostatné systémové komponenty.

Štúdie tiež preukázali, že server priemerne v stave nečinnosti spotrebúva o 70% elektrickej energie menej ako keď pracuje na plnom výkone CPU. Autori preto navrhli vzorec modelu spotreby (3.1) (*P*),

$$P(u) = k \times P_{max} + (1 - k) \times P_{max} \times u \qquad (3.1.)$$

kde P_{max} je maximálna spotreba energie pri plnom zaťažený servera, k je zlomok spotreby energie v stave nečinnosti (napr. 70%) a u je zaťaženie CPU.

Zaťaženie CPU sa môže priebehom času meniť z dôvodu premenlivosti pracovného zaťaženia ($u(t)$). Autori preto tiež navrhli vzorec (3.2) pre výpočet celkovej spotreby energie fyzického uzlu (E) definovanú ako integrál spotreby energie počas časového obdobia:

$$E = \int_{t_0}^{t_1} P(u(t))dt \tag{3.2.}$$

Výskum od Etinski a kol. [20] predstavuje novú metódu alebo algoritmus plánovania paralelných úloh, nazývanú MaxJobPerf. Táto politika je založená na celočíselnom lineárnom programovaní a používajúca techniku DVFS. Optimalizačným problémom je rozhodnúť, ktoré úlohy by mali byť na akej frekvencii spustené.

Politika plánovania paralelných úloh rozhoduje v akom poradí budú úlohy, potvrdené do superpočítačových centier, vykonané. Skúmané je on-line plánovanie pre asynchrónny príchod úloh. To nepredstavuje žiaden problém, ak je voľných dostatok dostupných zdrojov pre simultánne spustenie všetkých prichádzajúcich úloh. Komplikácia nastáva vtedy, ak je požadovaných viac zdrojov ako je k dispozícii. Vzhľadom k týmto obmedzeniam, ako je energetická náročnosť a počet procesorov, je potreba riešiť optimalizáciu problému. Navrhovaná politika MaxJobPerf môže plánovať vykonanie úlohy dvoma spôsobmi. V prípade, že je dostatok požadovaných zdrojov v čase príchodu úlohy, sa úloha spustí okamžite na najvyššej možnej frekvencii procesora. Ak nie je dostatok zdrojov v čase príchodu úlohy, úloha je poslaná do radu čakajúcich úloh, v ktorom sú úlohy zoradené podľa časov potvrdenia ich pridania do zoznamu.

Pri skončení niektorej z prebiehajúcich úloh a uvoľnení zdrojov, plánovač rieši spomínaný optimalizačný problém (Algoritmus 3.1). Výsledkom optimalizovania problému je, že plánovač vyberie úlohu zo zoznamu čakajúcich úloh J_i a rozhodne o frekvencii priradeného procesora. Premenná F_i nadobúda celočíselnú hodnotu pre úlohu J_i od 0 až po dostupnú frekvenciu procesora k. Naviac určuje, či je úloha J_i optimalizačným riešením vybraná pre spustenie.

$$F_i = \begin{cases} 0, & \text{úloha } J_i \text{ nie je vybraná pre spustenie} \\ k \in \{1,2,\dots,n\} & \text{úloha } J_i \text{ je vybraná pre spustenie na frekvencii } f_k \end{cases}$$

Algoritmus 3.1. Možné hodnoty pre premennú Fi [20]

Problém celočíselného lineárneho programovania, na ktorom je politika založená, je NP ťažký problém a čas riešenia rastie exponenciálne so zvyšujúcou sa veľkosťou radu čakajúcich úloh.

Porovnanie výkonu troch rôznych plánovačov (Baseline, PB-guided, MaxJobPerf), s rovnako nastaveným obmedzením výkonu, prebieha pomocou metriky (3.3) ohraničeného spomalenia (Bounded slowdown - BSLD). Táto široko používaná metrika výkonu úlohy definuje jej pomer medzi časom stráveným v systéme ($\check{C}as\check{C}akania$), čas behu ($\check{C}asBehu$) a hranici Th:

$$\text{BSLD} = \max\left(\frac{\check{C}as\check{C}akania + \check{C}asBehu}{\max(Th, \check{C}asBehu)}, 1\right) \tag{3.3.}$$

Hranica Th slúži k vyhnutiu sa vplyvu úloh s veľmi krátkou priemernou hodnotou času. V experimente bola nastavená hodnota Th na 10 minút, nakoľko úlohy s časom behu kratším ako 10 minút v rámci HPC sú klasifikované ako veľmi krátke úlohy. Pôvodný vzorec (3.3) je upravený (3.4) o redukovanú frekvenciu (f) pre úlohu (J) nasledovne:

$$\text{BSLD} = \max\left(\frac{\check{C}as\check{C}akania + Nov\acute{y}\check{C}asBehu(J,f)}{\max(Th, \check{C}asBehu)}, 1\right) \tag{3.4.}$$

Na používanie rôznych techník aplikovania zeleného počítania a ich dopadov na výkonnosť cloudových dátových centier sa venovali aj Bala et al. [31]. Tí v prostredí GreenCloud Simulator ukázali, že aj v dnešných tradičných datacentrách je možné zredukovať spotrebu energií, a tak znížiť uhlíkovú stopu. Kľúčom k dosiahnutiu tohto výsledku bolo aplikovanie režimu správy napájania založeného na technikách DVFS a Dynamic Network Shutdown.

3.3 Metóda medzikvartilového rozpätia

Štatistickou metódou na zistenie preťaženie hostiteľa je medzikvartilové rozpätie (IQR) stanovením hornej hranice využitia. [39] IQR je tiež nazývané stredný rozptyl alebo polovica. Autori vypočítali IQR z rozdielu medzi prvým (Q_1) a tretím (Q_3) kvartilom, kde Q_1 je prvých 25% a Q_3 je tretích 25% z vybranej množiny.

$$T_u = 1 - s \times IQR \tag{3.5.}$$

Horný prah zaťaženia (T_u) je vypočítaný podľa rovnice (3.5), kde *s je* bezpečnostný parameter, ktorý riadi, ako agresívne systém konsoliduje virtuálne stroje. Nižšie hodnoty premennej *s* vytvárajú väčší prah, čo spôsobuje nižšiu spotrebu energie ale výsledkom konsolidačného procesu sú vyššie narušenia SLA.

3.4 Metóda lokálnej regresie

Jednou z metód na detekciu preťaženého uzla (hostiteľa) je metóda lokálnej regresie. Výsledkom kombinácie lokálnej regresie a medzikvartilového rozpätia je metóda lokálnej robustnej regresie. [39]

Hlavnou myšlienkou metódy lokálnej regresie je vsadenie jednoduchých modelov do lokalizovaných podmnožín údajov pre vytvorenie krivky, ktorá sa blíži pôvodným údajom.

Tieto metódy sú použité na predpovedanie ďalšieho využitia CPU na základe štatistickej analýzy historických údajov využitia vybraného hostiteľa v cloudovom dátovom centre. Hostiteľ je označený ako preťažený, ak je predpokladané využitie CPU vyššie ako 100% alebo stanovený bezpečnostný parameter. [26]

3.5 Model mediánovej absolútnej odchýlky

Mediánová absolútna odchýlka je štatistický model používaný na detekciu preťaženého hostiteľa, určením vrchného a spodného prahu pre všetky virtuálne stroje spustených na tomto hostiteľovi.

Model počíta predpoveď zaťaženia pre hostiteľa. Hostiteľ sa považuje za preťaženého, pokiaľ predpokladaná hodnota zaťaženia presahuje určený vrchný prah.

V takom prípade by vybrané virtuálne stroje mali byť migrované na iného hostiteľa, za účelom zredukovania zaťaženia, aby sa predišlo prípadným porušeniam SLA.

MAD vypočítame (3.6) ako strednú hodnotu absolútnych odchýlok od strednej hodnoty jednorozmernej množiny údajov X_1, X_2, ..., X_n :

$$MAD = median_i(|X_i - median_j(X_j)|) \quad (3.6.)$$

Ak je predpokladané nižšie zaťaženie hostiteľa ako je spodný prah zaťaženia, všetky virtuálne stroje na hostiteľovi by mali byť dočasne uspané za účelom redukcie počtu aktívnych hostiteľov, čo vedie k nižšej spotrebe energie.

V prípade, že berieme do úvahy bezpečnostný parameter *s*, ktorý riadi ako agresívne systém konsoliduje virtuálne stroje, predstavuje rovnica (3.7) [39],

$$T_u = 1 - s \times MAD \quad (3.7.)$$

4 Návrh nových algoritmov na znižovanie energetickej náročnosti cloudov

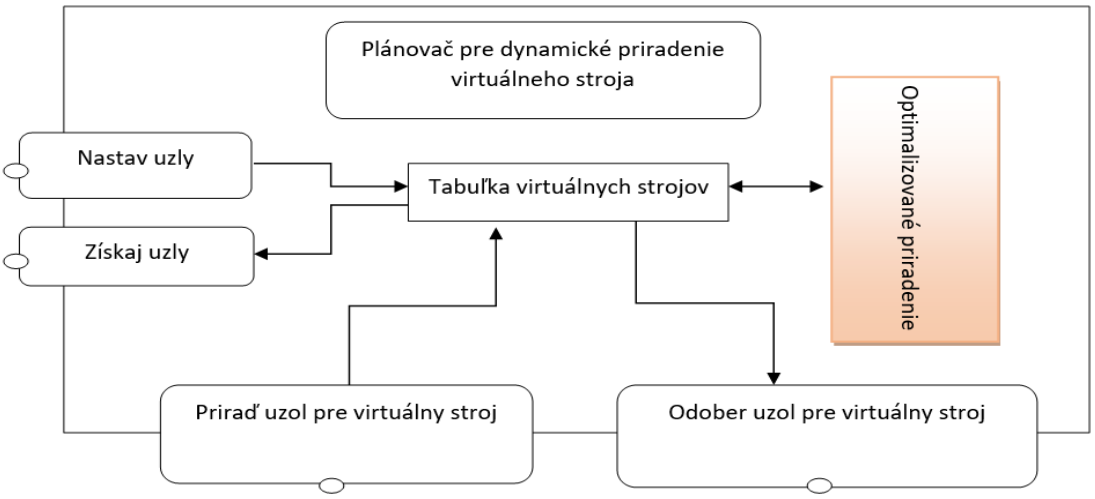
Cieľom tejto kapitoly je opísať novo navrhnuté a vytvorené algoritmy, zamerané na zelené počítanie a zníženie energetickej náročnosti cloudov. Týmito algoritmami sú:

- algoritmus energetickej efektívnosti - dynamické plánovanie (AEEDP),
- algoritmus energetickej efektívnosti – fuzzy logika (AEEFL).

4.1 Algoritmus energetickej efektívnosti - dynamické plánovanie

Prvým z predstavených vlastných navrhnutých algoritmov plánovania pre riešenie znižovania energetickej náročnosti v rámci cloudových systémov je algoritmus energetickej efektívnosti - dynamické plánovanie (AEEDP) (Obrázok 4.1).

Algoritmus je navrhnutý tak, aby sa dynamicky vykonávala migrácia virtuálnych strojov na uzle s dostupnými prostriedkami s cieľom ich neustáleho zlepšovania umiestnenia.



Obrázok 4.1. Navrhnutý plánovač pre dynamické priradenie virtuálneho stroja

Do algoritmu sme implementovali funkciu migrácie Best Fit (pozri kapitolu 2.3) pre získaný zoznam virtuálnych strojov a zaťaženia jednotlivých uzlov. Ten má za cieľ informovať, ktoré virtuálne stroje migrovať. Vrátený zoznam predstavuje postupnosť migrácií, ktoré sa majú vykonať. Každá položka v zozname obsahuje identifikátor virtuálneho stroja a identifikátor cieľového hostiteľa.

Z tohto dôvodu výsledná simulácia dátového centra spotrebováva menej energie ako pri simulácii bez aplikácii žiadnej metódy zeleného počítania.

4.2 Algoritmus energetickej efektívnosti – fuzzy logika

Druhým, hlavným navrhnutým algoritmom, je algoritmus energetickej efektívnosti – fuzzy logika (AEEFL).

Vzhľadom na to, že náš algoritmus používa fuzzy logiku, v nasledujúcej podkapitole sa venujeme opisu základných pojmov a vzťahov. V poslednej podkapitole prinášame detailný opis navrhnutého algoritmu pre detekciu preťaženia uzla.

4.2.1 Opis pojmov a vzťahov fuzzy logiky

Fuzzy inferenčné systémy (FIS) sú určené na vytváranie výstupov a vychádzajú z konceptov fuzzy logiky. Jeden z takých bol navrhnutý L. A. Zadehom, v jeho práci s fuzzy množinami. [63]

Fuzzy logika je forma logiky, v ktorej sa hodnoty pre pravdu pohybujú od nuly po jednu. Fuzzy logika je založená na ľudských vágnych rozhodnutiach a pomocou matematiky vyjadrujúca neurčitosť. Je určená na prácu s nepresnými informáciami pomocou teórie fuzzy množín, transformácií, fungovania a vyvodenia záverov. Vo všeobecnosti možno spracovanie fuzzy logiky opísať niekoľkými fázami:

- Fuzzifikácia – je proces transformácie „fuzzifikačných“ vstupov reálnych údajov do triedy funkcií príslušností.
- Inferencia – je proces spojenia vstupov fuzzy systému s jeho pravidlami na výpočet hodnôt fuzzy výstupných premenných (výstupných funkcií).
- Defuzzifikácia – je proces transformácie fuzzy výstupných funkcií na ostré hodnoty.

Pre transformáciu a spracovanie reálnych údajov do tvaru fuzzy množín so stupňom príslušnosti, zahŕňa fuzzy logika výraz jazykové premenné. Ten je určený na reprezentovanie skutočných vstupných údajov v zmysle jazykových premenných, ktoré sa vyznačujú jedným alebo viacerými jazykovými označeniami. Každé jazykové označenie má svoju vlastnú funkciu príslušnosti a základný priestor všetkých prvkov (tzv. univerzum) jazykovej premennej. Jazykové premenné a hodnoty sa vzťahujú na skutočné ľudské skúsenosti, používajúc prirodzený jazyk na vyjadrenie jednej alebo viacerých udalostí, pričom prvá je iba definíciou akéhokoľvek skutočného procesu a druhá je podobná matematickému nahradeniu.

Pre fuzzifikačnú fázu sú často navrhnuté lichobežníkové alebo trojuholníkové funkcie príslušnosti. Napríklad lichobežníková funkcia môže byť vyjadrená piatimi hranami, z ktorých každá leží vo vopred definovaných intervaloch. Pre fuzzy množinu F, ľubovoľná lichobežníková funkcia príslušnosti $\mu_F(w)$ so základným priestorom všetkých prvkov univerza $W = \{w\}$ je vyjadrená nasledujúcim systémom,

$$\forall w \in W: \mu_F(\mathrm{w}|\mathrm{min}, \mathrm{málo}, \mathrm{veľa}, \mathrm{max}) = \begin{cases} 0, \mathrm{w} < min \\ \dfrac{\mathrm{w} - \mathrm{min}}{\mathrm{málo} - \mathrm{min}}, \mathrm{min} \leq \mathrm{w} < málo \\ 1, \mathrm{málo} \leq \mathrm{w} \leq \mathrm{veľa} \\ \dfrac{\mathrm{max} - \mathrm{w}}{\mathrm{max} - \mathrm{veľa}}, \quad \mathrm{veľa} < w \leq \mathrm{max} \\ 0, \mathrm{w} > max \end{cases} \tag{4.1.}$$

kde min, max sú rozsahy, v rámci ktorých sa nachádzajú uvažované jazykové hodnoty (jazykové označenia) a málo, veľa sú stredné body.

Systémy fuzzy logiky zahrňujúce fázy fuzzifikácie, spracovania pravidiel a defuzzifikácie sa nazývajú fuzzy inferenčný systém alebo fuzzy regulátor. Jedným z najbežnejších je fuzzy model typu Mamdani, ktorý navrhol E. H. Mamdani [65] a je rozšírený medzi širokou vedeckou komunitou.

Fuzzy model typu Mamdani je založený na princípoch fuzzy logiky, inšpirovaných prácami Zadeha [63, 64] a predstavený v publikácii [65]. Mamdani sa zamýšľal pracovať s reálnymi vstupmi, často s nepresnými informáciami vychádzajúcich zo skutočného sveta. Fuzzy model sa skladá z troch hlavných fáz: fuzzifikácie, inferencie a defuzzifikácie.

Fuzzifikácia je rovná fuzzy logike. Reálne informácie sú konvertované do fuzzy množín stupňov príslušnosti. Vyskytujú sa tu rôzne funkcie príslušnosti, ktoré udržiavajú fuzzifikáciu. Najčastejšími sú to trojuholníkové a lichobežníkové (Vzorec 4.1), a menej používané ako sigmoidové, zvonové alebo Gausovské funkcie príslušnosti. Ako sme uviedli v predchádzajúcej časti, kroky fuzzifikácie uplatňujú jazykové premenné a ich hodnoty, ako fuzzy množiny a podmnožiny. Pre každú jazykovú premennú sú vytvorené pravidlá, resp. báza pravidiel.

Pre vytvorenie výstupnej premennej fuzzy funkcie, pozostáva inferenčná fáza z tzv. krokov agregácie, aktivácie, akumulácie a zaoberaním sa bázou pravidiel.

Agregácia znamená vykonanie všetkých aktivovaných výrokov zo súboru pravidiel (A, ALEBO, NIE; resp. AND, OR NOT atď.) a aplikovanie ich operátorov. Výsledkom

je fuzzy množina, ktorá môže byť výsledkom prieniku (A resp. AND), zjednotenia (ALEBO resp. OR) alebo doplnku (NIE resp. NOT).

Na základe výrokovej hodnoty, sa vo fáze aktivácie, vytvorí záver pre každé pravidlo z bázy pravidiel.

Akumulačná fáza zahŕňa spojenie „aktivovaných“ pravidiel tak, aby sa vybral najvhodnejší výsledok a vytvorila sa výstupná funkcia miery príslušnosti.

Nakoniec môže byť výstupná funkcia miery príslušnosti jazykovej premennej prevedená na tvar ostrých údajov. Toto sa vykonáva počas defuzzifikačnej fázy. [66] Existuje mnoho rôznych metód defuzzifikácie, pričom niektoré z nich sú:

- Metóda ťažiska (MT).
- Metóda stupňov
- Metóda najvýznamnejšieho maxima (výsledok môže byť najviac vľavo, v strede, najviac vpravo).

V našom algoritme je pre defuzzifikáciu použitá metóda ťažiska. To znamená výpočet ťažiska pre výstupnú funkciu ako (4.2),

$$\forall x \in X: \mathrm{MT} = \frac{\int x\mu_O(x)dx}{\int \mu_O(x)dx} \tag{4.2.}$$

kde O je výstupná fuzzy množina a $X = \{x\}$ sú všetky prvky univerza.

4.2.2 Detailný opis algoritmu pre detekciu preťaženia uzla

V tejto podkapitole objasňujeme detailný opis algoritmu, vychádzajúc zo zdrojového kódu a ďalších dodatočných informácií vyprodukovaných pridanými Java knižnicami.

Ako prezrádza stanovený názov, technikou pre dosiahnutie zníženia spotreby elektrickej energie dátového centra je použitie kombinácie princípov fuzzy logiky pre detegovanie, či je uzol preťažený. Zníženie spotreby je dosiahnuté sledovaním nízkeho zaťaženia, resp. preťaženia uzlov. Je potrebné poznamenať, že pre fuzzy logiku používame voľne dostupnú jFuzzyLogic Java knižnicu [59], ktorá zahŕňa fuzzy riadiace jazykové štandardy a notácie.

Existuje niekoľko techník detekcie preťaženia uzla, kým sa vykonajú úlohy konsolidácie virtuálnych strojov. Takýmito z nich sú mediánová absolútna odchýlka (MAD), medzikvartilové rozpätie (IQR), regresné stratégie a iné. [39]

Všetky vyššie uvedené techniky aplikujú ako vstup históriu využitia uzla a vytvárajú prahy (tzv. thresholds) pre MAD a IQR, a predpokladané využitie uzla pre regresné metódy. V princípe môžu byť výstupy techník MAD, IQR a regresia zlúčené, aby sme tak získali všetky výhody z každej metódy. Za týmto účelom sme použili fuzzy logiku, ktorá sa často používa v prípade, keď je ťažké (alebo dokonca nemožné) vytvoriť priamy matematický alebo fyzikálny model kvôli neistote v dátach. Fuzzy logika zahŕňa FIS. FIS obsahuje vstupné premenné, výstupné premenné, hodnoty príslušnosti a pravidlá. Výsledkom sú vytvorené výstupné hodnoty výstupných premenných na základe vstupných hodnôt a navrhnutých pravidiel.

V našom navrhnutom algoritme sme nastavili MAD, IQR a lokálnu regresiu (LR) ako vstupné hodnoty, a ako výstupnú hodnotu považujeme premennú preťaženie (t. j. preťaženie uzla), nachádzajúcu sa v rozsahu {0,1}, ktorá zodpovedá podmienkam pravdepodobnosti preťaženia uzla.

Na vytvorenie pravdepodobnosti preťaženia uzla sme použili lichobežníkové funkcie príslušnosti a stanovených 27 pravidiel. FIS sa pre zistenie preťaženia uzla zakaždým generuje dynamicky, pretože prahy pre MAD, IQR a predpokladané využitie uzla sú v priebehu času rozdielne.

Blokový diagram na detekciu preťaženia uzla je znázornený na obrázku 4.2. Na začiatku je potrebné vypočítať a získať hodnoty pre aktuálne využitie uzlov, prahy MAD a IQR, a predpokladané využitie pomocou LR. V prípade, že sú všetky tri metódy vypočítané a k dispozícii, prechádzame do stavu vytvárania FIS, vyhodnotenia výstupu FIS, aby sme sa rozhodli, či je hostiteľ nadmerne preťažený alebo nie (na základe porovnania výstupu FIS s premennou *prahPreťaženia*). V opačnom prípade, keď nemáme všetky 3 metódy vypočítané, prechádzame k rozhodovaciemu stavu *Aký počet metód bol vypočítaný?* . Môžu existovať prípady (vyskytujúce sa veľmi zriedka), keď nastane chyba (napríklad nezaradená história využitia), a až žiadna z metód nemusí byť vypočítaná.

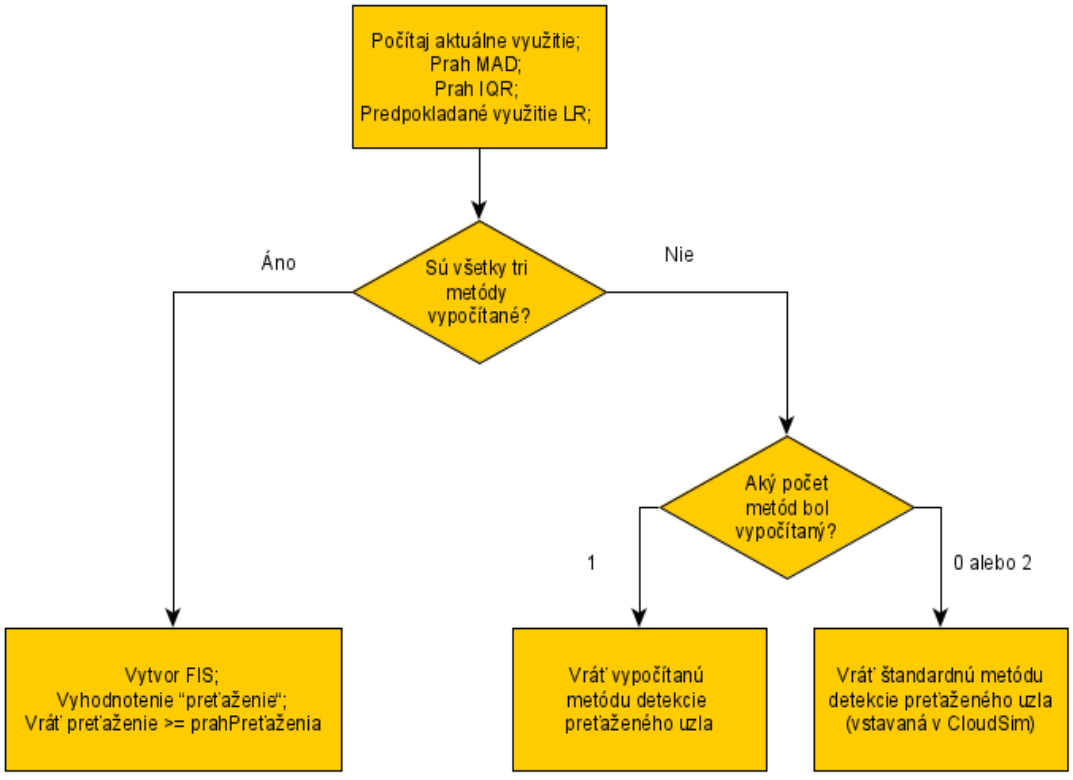
Podľa počtu metód, ktoré máme k dispozícii v tomto bode, nastávajú nasledovné varianty:

- 1 metóda - ak vypočítame iba jeden metódu (MAD, LR alebo IQR), použijeme túto metódu na vyhodnotenie preťaženia uzla.

- 0 alebo 2 metódy - ak máme nula alebo práve dve metódy (MAD a LR, MAD a IQR alebo IQR a LR), použijeme preddefinovanú metódu (politiku) „núdzového“ prideľovania virtuálnych strojov (balík org.cloudbus.cloudsim.examples.power.RunnerAbstrakt v CloudSim), ktorá môže byť:

PowerVmAllocationPolicyMigrationAbstract fallbackVmSelectionPolicy = new PowerVmAllocationPolicyMigrationStaticThreshold(hostList, vmSelectionPolicy, 0.7).

Tu je štandardne preddefinovaná politika prideľovania virtuálnych strojov *PowerVmAllocationPolicyMigrationStaticThreshold*, stratégia detekcie nadmerného využitia uzla pomocou prekročenia stanovenej hodnoty statického prahu (hodnota v príklade 0.7). Pretože je spomenutá metóda distribuovaná v rámci prostredia CloudSim, nazývame ju vstavaná.



Obrázok 4.2. Blokový diagram

Ako sme uviedli vyššie, vypočítavame samotné aktuálne využitie uzla (Algoritmus 4.1). Aktuálne využitie dostaneme pomerom požadovaných MIPS k celkovým

dostupným MIPS uzla. Ďalej vypočítavame prahy pre MAD (Algoritmus 4.2) a IQR (Algoritmus 4.3), a predpokladané využitie uzla pre LR (Algoritmus 4.4). Predpokladané využitie je aktuálne využitie uzla vypočítané pomocou lokálnej regresie.

```
// aktuálne využitie
double celkovéPožadovanéMips = 0;
for (Vm vm : uzol.získajZoznamVm()) {
    celkovéPožadovanéMips += vm.získajAktuálnePožadovanéCelkovéMips();
}
double využitie = celkovéPožadovanéMips / uzol.získajCelkovéMips();
```

Algoritmus 4.1. Výpočet aktuálneho využitia uzla

```
// prvý vstup – MAD
double hornýPrahMAD = 0;
try {
    hornýPrahMAD = 1 – získajBezpečnostnýParameter () *
    získajVyužitieUzlaMad(_uzol);
    numVstupy++;
} catch (NeplatnýArgument) {
    hornýPrahMAD = -1;
}
```

Algoritmus 4.2. Výpočet prahu pre MAD

```
// druhý vstup – IQR
double hornýPrahIQR = 0;
try {
    hornýPrahIQR = 1 – získajBezpečnostnýParameter () *
    získajVyužitieUzlaIQR(_uzol);
    numVstupy++;
} catch (NeplatnýArgument) {
    hornýPrahIQR = -1;
}
```

Algoritmus 4.3. Výpočet prahu pre IQR

```
// tretí vstup – LR
// pre lokálnu regresiu zadávame vstupný rozsah medzi {0,1} a rozdelenie na 3 intervaly (LOW, MEDIUM a HIGHT)
double[] históriaVyužitia = _uzol.získajHistóriuVyužitia();
// používame veľkosť 10, aby regresia dostatočne reagovala na najnovšie hodnoty
int dĺžka = 10;
double predpokladanéVyužitie = 0;
if (históriaVyužitia.dĺžka >= dĺžka) {
    double[] históriaVyužitiaInverzná = new double[length];
    for (int i = 0; i < dĺžka; i++) {
        históriaVyužitiaInverzná[i] = históriaVyužitia[dĺžka – i - 1];
    }
    double[] odhady = null;
    try {
        odhady = získajParameterOdhadu(históriaVyužitiaInverzná);
        double intervalyMigráie = Math.ceil(získajMaximálnyČasMigrácieVM(_uzol) / (získajIntervalPlánovania());
        predpokladanéVyužitie = odhady[0] + odhady[1] * (dĺžka + intervalyMigráie);
        //pre lokálnu regresiu
        zadajBezpečnostnýParameter(bezpečnostnýParameterLR);
        predpokladanéVyužitie *= získajBezpečnostnýParameter();
        numVstupy++;
    } catch (NeplatnýArgument) {
        predpokladanéVyužitie = -1;
    }
}
else {
    predpokladanéVyužitie = -1;
}
```

Algoritmus 4.4. Výpočet lokálnej regresie predpokladaného využitia uzla

Tri hodnoty vypočítané vyššie pre *hornýPrahMAD*, *hornýPrahIQR* a *predpokladanéVyužitie* sú použité, a dosadené, pre vytvorenie FIS (Algoritmus 4.5).

```
FIS súbor = vytvorFISstavUzla(hornýPrahMAD, hornýPrahIQR, predpokladanéVyužitie);
```

Algoritmus 4.5. Vytváranie FIS

Ďalej ako môžeme vidieť (Algoritmus 4.6), FIS pozostáva z troch vstupných premenných: *MAD*, *IQR* a *LR*, a jednej výstupnej premennej *preťaženie*.

```
Premenná MAD = new Premenná(“MAD”);
Premenná IQR = new Premenná(“IQR”);
Premenná LR = new Premenná(“LR”);
blokFunkcie.nastavPremennú(MAD.získajNázov(), MAD);
blokFunkcie.nastavPremennú(IQRzískajNázov(), IQR);
blokFunkcie.nastavPremennú(LR.získajNázov(), LR);

Premenná preťaženie = new Premenná(“preťaženie”);
blokFunkcie.nastavPremennú(preťaženie.získajNázov(), preťaženie);
```

Algoritmus 4.6. Definovanie premenných pre FIS

K fuzzifikácii vstupov používame lichobežníkovú funkciu príslušnosti. Fuzzifikačná fáza zahŕňa výpočty stupňa príslušnosti pre konkrétne jazykové premenné. Každá jazyková premenná má svoje vlastné funkcie príslušnosti. V našom prípade sú to vždy lichobežníkové funkcie príslušnosti.

Všetky premenné, vstupy aj výstupy, majú svoje jazykové premenné. *MAD*, *IQR* a *LR* majú jazykové premenné NÍZKE (LOW), STREDNÉ (MEDIUM) a VYSOKÉ (HIGHT). Rovnaké pojmy NÍZKE, STREDNÉ, VYSOKÉ a nový pojem VEĽMI_VYSOKÉ (VERY_HIGHT), sú definované pre premennú *preťaženie*.

Ako sme už spomenuli, pre každú jazykovú premennú sú definované funkcie príslušnosti a všetky majú lichobežníkový tvar. Na Algoritmoch 4.7 a 4.8 je uvedený príklad pre vstupnú premennú MAD v jazyku Java a následne zápis vo FIS.

```
FunkciaPríslušnosti NÍZKE_MAD = new FunkciaPríslušnostiLichobežníková (
    new Hodnota(0),
    new Hodnota(nizkyPrahVyužitia),
    new Hodnota(nizkyPrahVyužitia + 0.1 * nizkyPrahVyužitia));

FunkciaPríslušnosti STREDNÉ_MAD = new FunkciaPríslušnostiLichobežníková
(
    new Hodnota(nizkyPrahVyužitia - 0.1 * nizkyPrahVyužitia),
    new Hodnota(nizkyPrahVyužitia + 0.2 * nizkyPrahVyužitia),
    new Hodnota(hornýPrahMAD - 0.2 * hornýPrahMAD),
    new Hodnota(hornýPrahMAD + 0.1 * hornýPrahMAD));

FunkciaPríslušnosti VYSOKÉ_MAD = new FunkciaPríslušnostiLichobežníková (
    new Hodnota(hornýPrahMAD - 0.1 * hornýPrahMAD)),
    new Hodnota(hornýPrahMAD),
    new Hodnota(5),
    new Hodnota(5));

JazykováPremenná jpNÍZKE_MAD = JazykováPremenná(“NÍZKE”,
NÍZKE_MAD);
JazykováPremenná jpSTREDNÉ_MAD = JazykováPremenná(“STREDNÉ”,
STREDNÉ_MAD);
JazykováPremenná jpVYSOKÉ_MAD = JazykováPremenná(“VYSOKÉ”,
VYSOKĚ_MAD);

MAD.pridaj(jpNÍZKE_MAD);
MAD.pridaj(jpSTREDNÉ_MAD);
MAD.pridaj(jpVYSOKÉ_MAD);
```

Algoritmus 4.8. Definovanie hodnôt pre jazykovú premennú MAD

```
FUZZIFIKÁCIA MAD
    PREMENNÁ VYSOKÉ := TRAPE 0.8629868233082707
    0.9588742481203008 5 5;
    PREMENNÁ NÍZKE := TRAPE 0.0 0.0 0.1 0.11000000000000001;
    PREMENNÁ STREDNÉ := TRAPE 0.09 0.12000000000000001
    0.7670993984962406 1.0547616729323308;
END_FUZZIFIKÁCIA
```

Algoritmus 4.7. Definovanie hodnôt pre jazykovú premennú MAD v zápise FIS

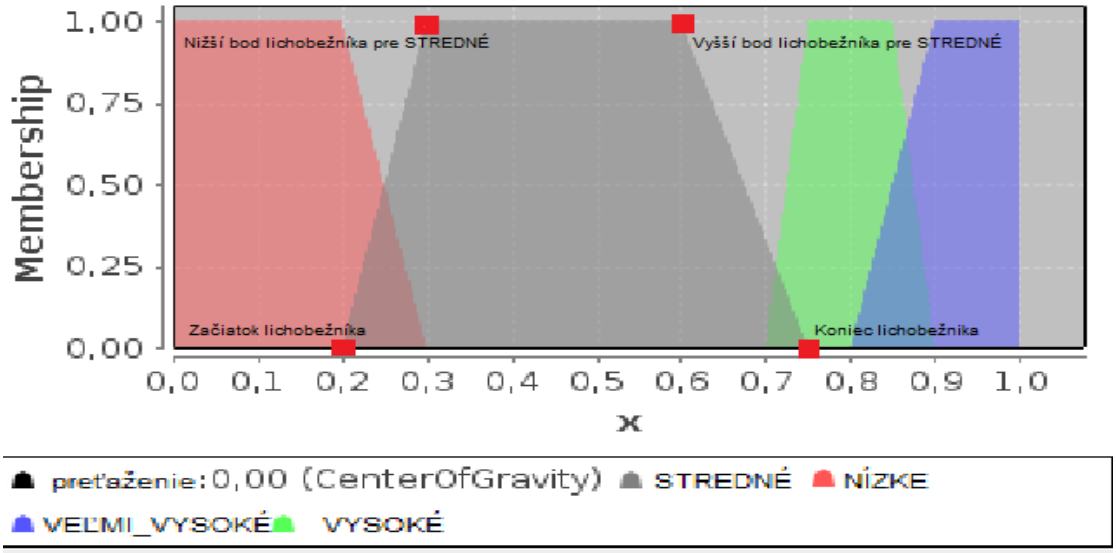
Z Algoritmov 4.7 a 4.8 vyplýva, že funkcie príslušnosti MAD boli vytvorené dynamicky. Vychádzajú z premenných *nizkyPrahVyužitia* a *hornýPrahMAD.*

nizkyPrahVyužuitia je definovaná na začiatku triedy a je súčasne rovná 0.1. Táto hodnota nám definuje minimálne možné využitie. V našom prípade (počas simulácií) s najväčšou pravdepodobnosťou nenastane prípad s hodnotou nižšou alebo rovnou 0.1

kvôli detekcii nevyužitia uzla v CloudSim. V algoritme sa to nachádza len ako poistenie sa.

hornýPrahMAD je jedna z troch vstupných premenných (Algoritmus 4.5). Dôsledkom toho v Algoritme 4.8 uvádzame konkrétny prípad FIS, ako sme ho vykonali v zápise vo FIS.

TRAPE znamená lichobežníkovú funkciu príslušnosti. Jej stavy ilustruje Obrázok 4.3.



Obrázok 4.3. Lichobežníková funkcia príslušnosti

K vytvoreniu výstupu, používa FIS defuzzifikáciu pomocou logiky pravidiel Mamdani (Algoritmus 4.9 a Algoritmus 4.10), techniku na preklad stupňa výstupu funkcie príslušnosti v jazykových premenných do ostrých hodnôt (pozri kap. 4.2.1).

Stupeň výstupu funkcie príslušnosti znamená, že môžeme mať napríklad takéto inferenčné výsledky:

- MAD : VYSOKÉ s 0.8 stupňom, NÍZKE s 0 stupňom, STREDNÉ s 0,3 stupňom
- IQR : VYSOKÉ s 0.85 stupňom, NÍZKE s 0 stupňom, STREDNÉ s 0,4 stupňom
- LR : VYSOKÉ s 0.70 stupňom, NÍZKE s 0 stupňom, STREDNÉ s 0,2 stupňom.

Z toho vyplýva, že ich môžeme previesť na jazykové výrazy pre premennú *preťaženie* uplatňujúc pravidlá.

```
FunkciaPríslušnosti NÍZKE_PR = new FunkciaPríslušnostiLichobežníková (
    new Hodnota(0),
    new Hodnota(0),
    new Hodnota(0.2),
    new Hodnota(0.3));

FunkciaPríslušnosti STREDNÉ_PR = new FunkciaPríslušnostiLichobežníková (
    new Hodnota(0.2),
    new Hodnota(0.3),
    new Hodnota(0.6),
    new Hodnota(0.75));

FunkciaPríslušnosti VYSOKÉ_PR = new FunkciaPríslušnostiLichobežníková (
    new Hodnota(0.7),
    new Hodnota(0.75),
    new Hodnota(0.85),
    new Hodnota(0.9));


FunkciaPríslušnosti VEĽMI_VYSOKÉ_PR = new
FunkciaPríslušnostiLichobežníková (
    new Hodnota(0.8),
    new Hodnota(0.9),
    new Hodnota(1),
    new Hodnota(1));

JazykováPremenná jpNÍZKE_PR = JazykováPremenná(“NÍZKE”, NÍZKE_PR);
JazykováPremenná jpSTREDNÉ_PR = JazykováPremenná(“STREDNÉ”,
STREDNÉ_PR);
JazykováPremenná jpVYSOKÉ_PR = JazykováPremenná(“VYSOKÉ”,
VYSOKĚ_PR);
JazykováPremenná jpVEĽMI_VYSOKÉ_PR =
JazykováPremenná(“VEĽMI_VYSOKÉ”, VEĽMI_VYSOKĚ_PR);

preťaženie.pridaj(jpNÍZKE_PR);
preťaženie.pridaj(jpSTREDNÉ_PR);
preťaženie.pridaj(jpVYSOKÉ_PR);
preťaženie.pridaj(jpVEĽMI_VYSOKÉ_PR);
preťaźenie.nastavDefuzzifikáciu(new DefuzifikáciaMetódaŤažiska (preťaženie));
```

Algoritmus 4.9. Definovanie jazykových premenných výstupu preťaženie a defuzzifikácia

```
FUZZIFIKÁCIA MAD
   PREMENNÁ VYSOKÉ := TRAPE 0.8629868233082707
   0.9588742481203008 5 5;
   PREMENNÁ NÍZKE := TRAPE 0.0 0.0 0.1 0.11000000000000001;
   PREMENNÁ STREDNÉ := TRAPE 0.09 0.12000000000000001
   0.7670993984962406 1.0547616729323308;
END_FUZZIFIKÁCIA
```

Algoritmus 4.10. Definovanie jazykových premenných výstupu preťaženie v zápise fuzzy riadenia a defuzzifikácia

Používame lichobežníkové funkcie príslušnosti pre jazykové premenné *preťaženie*. Zároveň, METÓDA := COG alebo DefuzifikáciaMetódaŤažiska, znamená použitie metódy ťažiska k získaniu výsledku defuzzifikácie premennej *preťaženie*. DEFAULT := NC znamená, že ak by nastalo nezadefinované pravidlo, výstupná hodnota bude nulová. V našom prípade sme pokryli všetky možné situácie, v algoritme sa to nachádza len pre informáciu. ROZSAH (voliteľné) znamená výstupný rozsah klasických hodnôt.

V tomto bode máme zadefinované všetky jazykové premenné pre *MAD*, *IQR* a *LR*, a premennú *preťaženie*. Ako sme uviedli, na vytvorenie výstupu, FIS uplatňuje pravidlá vytvorených pomocou hodnôt vstupných jazykových premenných. Tieto pravidlá sú veľmi podobné s bežnými logickými príkazmi (napr. Ak...Potom, atď.). Napríklad môže byť rozhodnutie, ktorého výstup vyzerá „*ak MAD je VYSOKÉ, IQR NÍZKE a LR je VYSOKÉ, potom PREŤAŽENIE je VYSOKÉ“*.

Takže máme pravidlá (Tabuľka 4.1) pre všetky prípady a uplatňujeme ich na vytvorenie výstupu. V Algoritme 4.11 uvádzame príklad definície bloku pravidla zapísaného v Java. Každý takto definovaný *BlokPravidla* obsahuje vytvorené pravidlá. Zápis vybraných vytvorených pravidiel sú znázorňujú Algoritmus 4.12 a Algoritmus 4.13

```
BlokPravidla blokPravidla = new BlokPravidla(blokFunkcie);
blokPravidla.nastavNázov (“No1“);
blokPravidla.nastavRuleAccumulationMethod(new
RuleAccumulationMethodMax());
blokPravidla.nastvaRuleActivationMethod(new RuleActivationMethodMin());
```

Algoritmus 4.11. Definovanie bloku pravidla

blokPravidla.nastavRuleAccumulationMethod(new RuleAccumulationMethodMax() znamená pravidlo metódy, ktoré uplatňuje stratégiu

maximalizovania počas fázy fuzzy akumulácie. Ako príklad, vezmime do úvahy dve súčasne vyskytnuté pravidlá zo všetkých, ktorými sú tieto:

- Pravidlo17: (MAD je VYSOKÉ A IQR je NÍZKE) A LR je VYSOKÉ potom PREŤAŽENIE je VYSOKÉ
- Pravidlo18: (MAD je NÍZKE A IQR je VYSOKÉ) A LR je VYSOKÉ potom PREŤAŽENIE je VYSOKÉ

Pre logickú operáciu A (AND) aplikujeme algoritmus minimalizácie na splnenie De Morganovho zákona. Predpokladajme MAD : VYSOKÉ s 0.8 stupňom, NÍZKE s 0 stupňom; IQR : NÍZKE s 0.5 stupňom, VYSOKÉ s 0 stupňom a LR : VYSOKÉ s 0.6 stupňom, teda AND operátory vytvárajú minimum z trojice (0.8;0.5;0.6) ako 0.5 pre Pravidlo17 and minimum z trojice (0;0;0.6) ako 0. Potom, v súlade s minimálnou (*min*) aktivačnou stratégiou, sme stanovili výsledky pre jednu jazykovú premennú *VYSOKÉ* výstupnej premennej *preťaženie,* pretože sa rovná výsledkom všetkých A logických operácií pre pravidlá Pravidlo17 a Pravidlo18. Takže máme súčasne 0 a 0.5. V súlade s maximovou (*max*) agregačnou stratégiou, vyberáme ako stupeň príslušnosti 0.5 pre jazykovú premennú VYSOKÉ.

blokPravidla.nastvaRuleActivationMethod(new RuleActivationMethodMin()) znamená použitie stratégie minimalizovania počas fázy aktivácie pravidla.

```
// PRAVIDLO 3: AK ((MAD je VYSOKÉ) A (IQR je STREDNÉ) A (LR je VYSOKÉ) POTOM preťaženie je VYSOKÉ
Pravidlo pravidlo3 = new Pravidlo("Pravidlo3", blokPravidla);
PremennáPravidla premenná1_3 = new PremennáPravidla(MAD, "VYSOKÉ", false);
PremennáPravidla premenná2_3 = new PremennáPravidla(IQR, "STREDNÉ", false);
PremennáPravidla premenná3_3 = new PremennáPravidla(LR, "VYSOKÉ", false);
VýrazPravidla predchodcaA_3_0 = new VýrazPravidla(premenná1_3,
                                premenná2_3, RuleConnectionMethodAndMin.get());
VýrazPravidla predchodcaA_3_1 = new VýrazPravidla(predchodcaA_3_0,
                                premenná3_3, RuleConnectionMethodAndMin.get());
pravidlo3.nastavPredchodcov(predchodcaA_3_1);

rule3.pridajZáver(preťaženie, "VYSOKÉ", false);
blokPravidla.pridaj(pravidlo3);
```

Algoritmus 4.12. Príklad zápisu pravidla3

Z uvedeného algoritmu vyššie vidíme prípad, kedy *((MAD* je *VYSOKÉ)* A *(IQR* je *STREDNÉ)* A *(LR* je *VYSOKÉ)*, ktorého záverom bude *preťaženie* je *VYSOKÉ*.

```
// PRAVIDLO 6: AK ((MAD je VYSOKÉ) A (IQR je STREDNÉ) A (LR je STREDNÉ) POTOM preťaženie je STREDNÉ
Pravidlo pravidlo6 = new Pravidlo("Pravidlo6", blokPravidla);
PremennáPravidla premenná1_6 = new PremennáPravidla(MAD, "VYSOKÉ", false);
PremennáPravidla premenná2_6 = new PremennáPravidla(IQR, "STREDNÉ", false);
PremennáPravidla premenná3_6 = new PremennáPravidla(LR, "STREDNÉ", false);
VýrazPravidla predchodcaA_6_0 = new VýrazPravidla(premenná1_6,
                                premenná2_6, RuleConnectionMethodAndMin.get());
VýrazPravidla predchodcaA_6_1 = new VýrazPravidla(predchodcaA_6_0,
                                premenná3_6, RuleConnectionMethodAndMin.get());
pravidlo3.nastavPredchodcov(predchodcaA_6_1);
rule3.pridajZáver(preťaženie, "STREDNÉ", false);
blokPravidla.pridaj(pravidlo6);
```

Algoritmus 4.13. Príklad zápisu pravidla6

Znova, z uvedeného algoritmu vyššie vidíme prípad *((MAD* je *VYSOKÉ)* A *(IQR* je *STREDNÉ)* A *(LR* je *STREDNÉ)*, ktorého záverom bude *preťaženie* je *STREDNÉ*.

RuleConnectionMethodAndMin.get() znamená splnenie De Morganovho zákona, algoritmus minimalizácie použitý pre operátor A (and). Tabuľka 1.1 obsahuje zoznam všetkých navrhnutých pravidiel a ich vstupov/výstupov, kde ACT : MIN (pravidlo aktivačnej stratégie); ACCU : MAX (stratégia pre aktiváciu); AND : MIN (algoritmus minimalizácie) znamenajú vyššie vysvetlené stratégie a algoritmy pre operátor A (and). [60]

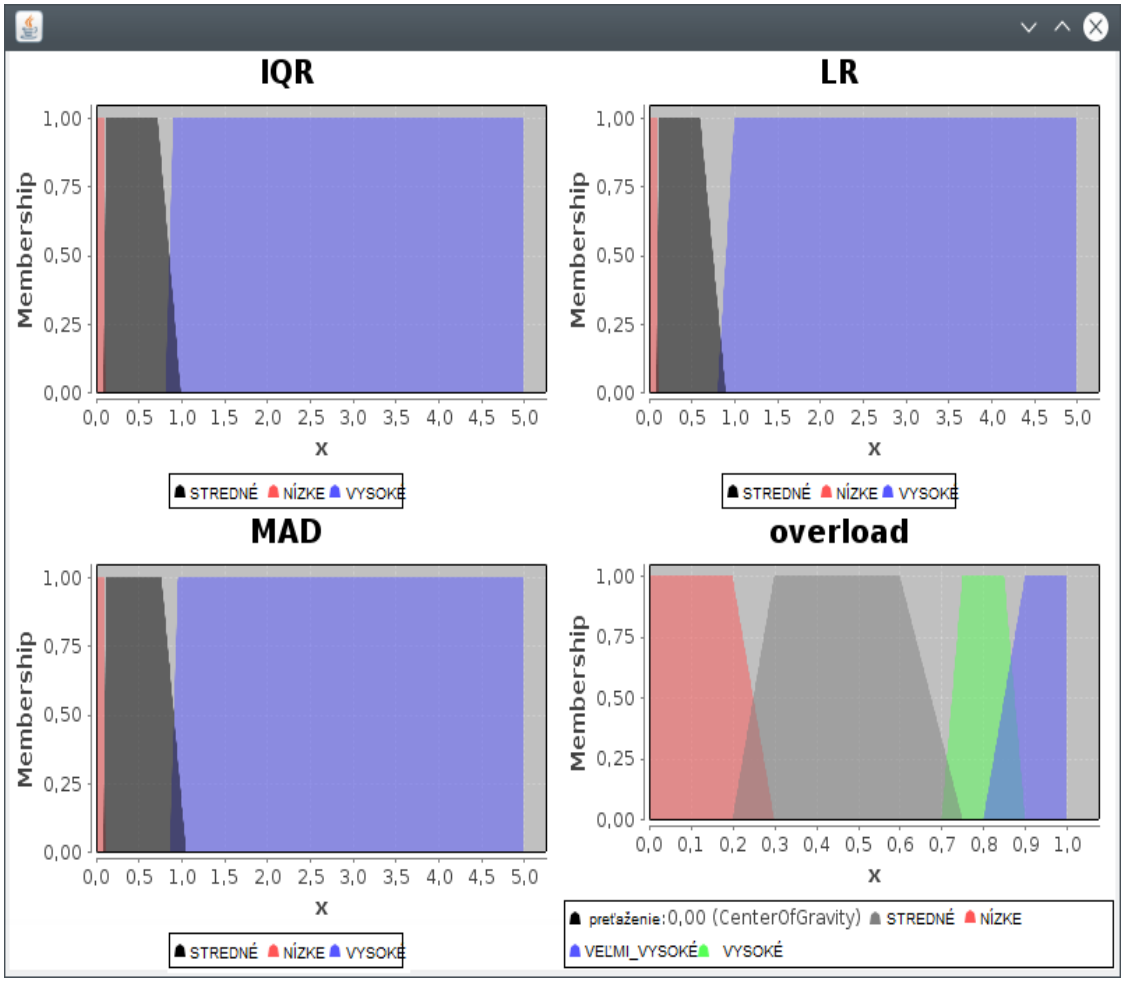
Tabuľka 4.1. Zoznam navrhnutých pravidiel

Názov	Premenná	Vstup	Výstup
ACT	MIN		
ACCU	MAX		
AND	MIN		
Pravidlo	Pravidlo1	(MAD je VYSOKÉ A IQR je VYSOKÉ) A LR je VYSOKÉ	preťaženie je VEĽMI_VYSOKÉ
Pravidlo	Pravidlo2	(MAD je VYSOKÉ A IQR je VYSOKÉ) A LR je STREDNÉ	preťaženie je VYSOKÉ

Pravidlo	Pravidlo3	(MAD je VYSOKÉ A IQR je STREDNÉ) A LR je VYSOKÉ	preťaženie je VYSOKÉ
Pravidlo	Pravidlo4	(MAD je STREDNÉ A IQR je VYSOKÉ) A LR je VYSOKÉ	preťaženie je VYSOKÉ
Pravidlo	Pravidlo5	(MAD je STREDNÉ A IQR je STREDNÉ) A LR je VYSOKÉ	preťaženie je STREDNÉ
Pravidlo	Pravidlo6	(MAD je VYSOKÉ A IQR je STREDNÉ) A LR je STREDNÉ	preťaženie je STREDNÉ
Pravidlo	Pravidlo7	(MAD je STREDNÉ A IQR je VYSOKÉ) A LR je STREDNÉ	preťaženie je STREDNÉ
Pravidlo	Pravidlo8	(MAD je STREDNÉ A IQR je STREDNÉ) A LR je STREDNÉ	preťaženie je STREDNÉ
Pravidlo	Pravidlo9	(MAD je NÍZKE A IQR je STREDNÉ) A LR je STREDNÉ	preťaženie je STREDNÉ
Pravidlo	Pravidlo10	(MAD je STREDNÉ A IQR je NÍZKE) A LR je STREDNÉ	preťaženie je STREDNÉ
Pravidlo	Pravidlo11	(MAD je STREDNÉ A IQR je STREDNÉ) A LR je NÍZKE	preťaženie je STREDNÉ
Pravidlo	Pravidlo12	(MAD je NÍZKE A IQR je STREDNÉ) A LR je NÍZKE	preťaženie je NÍZKE
Pravidlo	Pravidlo13	(MAD je NÍZKE A IQR je NÍZKE) A LR je STREDNÉ	preťaženie je NÍZKE
Pravidlo	Pravidlo14	(MAD je STREDNÉ A IQR je NÍZKE) A LR je NÍZKE	preťaženie je NÍZKE
Pravidlo	Pravidlo15	(MAD je NÍZKE A IQR je NÍZKE) A LR je NÍZKE	preťaženie je NÍZKE
Pravidlo	Pravidlo16	(MAD je VYSOKÉ A IQR je VYSOKÉ) A LR je NÍZKE	preťaženie je VYSOKÉ
Pravidlo	Pravidlo17	(MAD je VYSOKÉ A IQR je NÍZKE) A LR je VYSOKÉ	preťaženie je VYSOKÉ
Pravidlo	Pravidlo18	(MAD je NÍZKE A IQR je VYSOKÉ) A LR je VYSOKÉ	Preťaženie je VYSOKÉ
Pravidlo	Pravidlo19	(MAD je NÍZKE A IQR je NÍZKE) A LR je VYSOKÉ	preťaženie je STREDNÉ
Pravidlo	Pravidlo20	(MAD je NÍZKE A IQR je VYSOKÉ) A LR je NÍZKE	preťaženie je STREDNÉ
Pravidlo	Pravidlo21	(MAD je VYSOKÉ A IQR je NÍZKE) A LR je NÍZKE	preťaženie je STREDNÉ

Pravidlo	Pravidlo22	(MAD je NÍZKE A IQR je VYSOKÉ) A LR je STREDNÉ	preťaženie je VYSOKÉ
Pravidlo	Pravidlo23	(MAD je NÍZKE A IQR je STREDNÉ) A LR je VYSOKÉ	preťaženie je VYSOKÉ
Pravidlo	Pravidlo24	(MAD je VYSOKÉ A IQR je STREDNÉ) A LR je NÍZKE	preťaženie je VYSOKÉ
Pravidlo	Pravidlo25	(MAD je VYSOKÉ A IQR je NÍZKE) A LR je STREDNÉ	preťaženie je VYSOKÉ
Pravidlo	Pravidlo26	(MAD je STREDNÉ A IQR je NÍZKE) A LR je VYSOKÉ	preťaženie je VYSOKÉ
Pravidlo	Pravidlo27	(MAD je STREDNÉ A IQR je VYSOKÉ) A LR je NÍZKE	preťaženie je VYSOKÉ

Vytvorený FIS hodnôt príslušnosti je zobrazený obrázku nižšie.



Obrázok 4.4. Vytvorený FIS hodnôt príslušnosti (vygenerovaný)

Obrázok 4.4 znázorňuje vykreslené funkcie príslušnosti pre premenné FIS *IQR*, *LR*, *MAD* a *preťaženie*. Zároveň sú tu graficky zobrazené všetky hodnoty pre jazykové premenné „NÍZKE“, „STREDNÉ“, „VYSOKÉ“ a „VEĽMI_VYSOKÉ“ pre vstupné a výstupné premenné. Z grafického zobrazenia vidíme tvary jednotlivých funkcií. Os X znamená konkrétne hodnoty, os Y (príslušnosť) znamená stupeň príslušnosti jazykových premenných pre klasické vstupné alebo výstupné premenné.

Ďalej zabezpečujeme do existujúceho FIS pre MAD, IQR prahy aktuálneho využitia a pre LR – predpokladané LR využitie. Potom voláme funkciu *vyhodnotenie()* k vytvorenie FIS výstupu z našich predchádzajúcich troch vstupov: MAD, IQR a LR (Algoritmus 4.14).

Inak povedaná, *vyhodnotenie()* počíta stupeň príslušnosti, následne aplikujeme inferenčnú fázu a nakoniec vytvárame konkrétne hodnoty premenných preťaženia uzla.

```
fis.získajPremennú(“MAD“).nastavHodnotu(využitie);
fis.získajPremennú(“IQR“).nastavHodnotu(využitie);
fis.získajPremennú(“LR“).nastavHodnotu(predpokladanéVyužitie);
fis.vyhodnotenie();
pridajZáznamHistórie(uzol, fis.získajPremennú(“preťaženie“).získajHodnotu();
return(fis.získajPremennú((“preťaženie“).získajHodnotu()>=prahPreťaženie);
```

Algoritmus 4.14. Zadanie vstupných premenných do existujúceho FIS a získanie FIS klasického výstupu

FIS vytvára premennú *preťaženie* v rozsahu {0,1}. Nakoniec prichádza k rozhodnutiu, či je tento hostiteľ preťažený alebo nie, na základe porovnania výstupu FIS s premennou *prahPreťaženie*.

5 Metodiky a experimentálne overovanie navrhnutých algoritmov

Táto kapitola obsahuje modely riešenia, postupnosť metód, nami navrhnutú metodiku práce a experimentálne overenie nami navrhnutých algoritmov pre dosiahnutie stanovených cieľov práce v simulačnom prostredí.

5.1 Metodika overovania riešenia v simulačnom prostredí CloudSim

Cieľom tejto podkapitoly je opísať zvolené použité prostriedky a algoritmy v simulačnom prostredí CloudSim.

Základom pre nameranie výsledkov simulácie je v prvom rade vytvorenie simulovaného dátového centra, na ktorého uzloch budú umiestňované virtuálne stroje. V simulácii budeme pracovať s jedným datacentrom, v ktorom bude celkovo k dispozícii 800 uzlov (heterogénneho typu, rozdelených na dve rovnako veľké skupiny) a k nim v priebehu času priradených 1052 virtuálnych strojov s vopred definovanou rozdielnou pracovnou záťažou. Medzi vlastnosti uzlov, ktoré sú opísané v Tabuľke 5.1, patrí:

- Použitý typ procesora.
- Počet prvkov spracovania.
- Výkon.
- Veľkosť pamäte RAM.

Počas simulácie je celkovým sledovaným obdobím 1 deň, resp. 86 400 sekúnd. Toto časové obdobie je stanovená počtom zaznamenaných informácií o využití procesora (pozri kapitolu 5.2.1) jednotlivých virtuálnych strojov, ktoré máme k dispozícii.

Zároveň v simulácii používame 4 rozdielne konfigurácie pre virtuálne stroje. Medzi vlastnosti virtuálnych uzlov, ktoré sú opísané v Tabuľke 5.2, patrí:

- Počet prvkov spracovania.
- Výkon.
- Veľkosť pamäte RAM.

Tabuľka 5.1. Vlastnosti použitého uzla

Názov	HP ProLiant ML110 G4	HP ProLiant ML110 G5
Procesor	Intel Xeon 3040	Intel Xeon 3075
Počet prvkov spracovania	2	2
Výkon (MIPS)	1860	2660
Pamäť RAM (GB)	4096	4096

Pre porovnanie a vyhodnotenie výsledkov aplikovaného konceptu zeleného počítania používame 6 techník (pozri kapitolu 3 a 4):

- statický prah (SP) [19, 39]
- dynamické škálovanie napätia a frekvencie (DVFS) [19, 39]
- lokálna regresia (LR) [19, 39]
- mediánová absolútna odchýlka (MAD) [19, 39]
- algoritmus energetickej efektívnosti – dynamické plánovanie (AEEDP)
- algoritmus energetickej efektívnosti – fuzzy logika (AEEFL)

Zvolené techniky zeleného počítača a navrhnuté algoritmy spolu s vybranou testovacou pracovnou záťažou implementujeme do vytvoreného simulovaného dátového centra. Počas overovania účinnosti zvolených techník sme museli zvoliť aj najvhodnejší prístup pre výber virtuálneho stroja, ktorý má byť migrovaný. Sériu meraní sme porovnali s prístupmi aplikujúcich maximálnu koreláciu, minimálny čas migrácie a náhodný výber. Prístup maximálnej korelácie vyšiel s najlepšími výsledkami spomedzi ostatných, a preto len ten uvádzame ďalej v práci.

Aby sme získali východiskovú (referenčnú) spotrebu elektrickej energie simulovaného dátového centra za stanovený čas, ktorú môžeme neskôr porovnávať s výsledkami po aplikovaní techník zeleného počítania, vykonáme najskôr simuláciu bez aplikovania akejkoľvek techniky. Ďalej v práci označené ako bezoptimalizačná technika.

Následne vykonáme rovnakú simuláciu pre každú vyššie spomenutú techniku zeleného počítania a sledujeme hodnotiace výstupné parametre:

- čas dokončenia simulácie,
- energetická náročnosť dátového centra,
- počet migrácií virtuálnych strojov,
- dosiahnutú úroveň SLA,

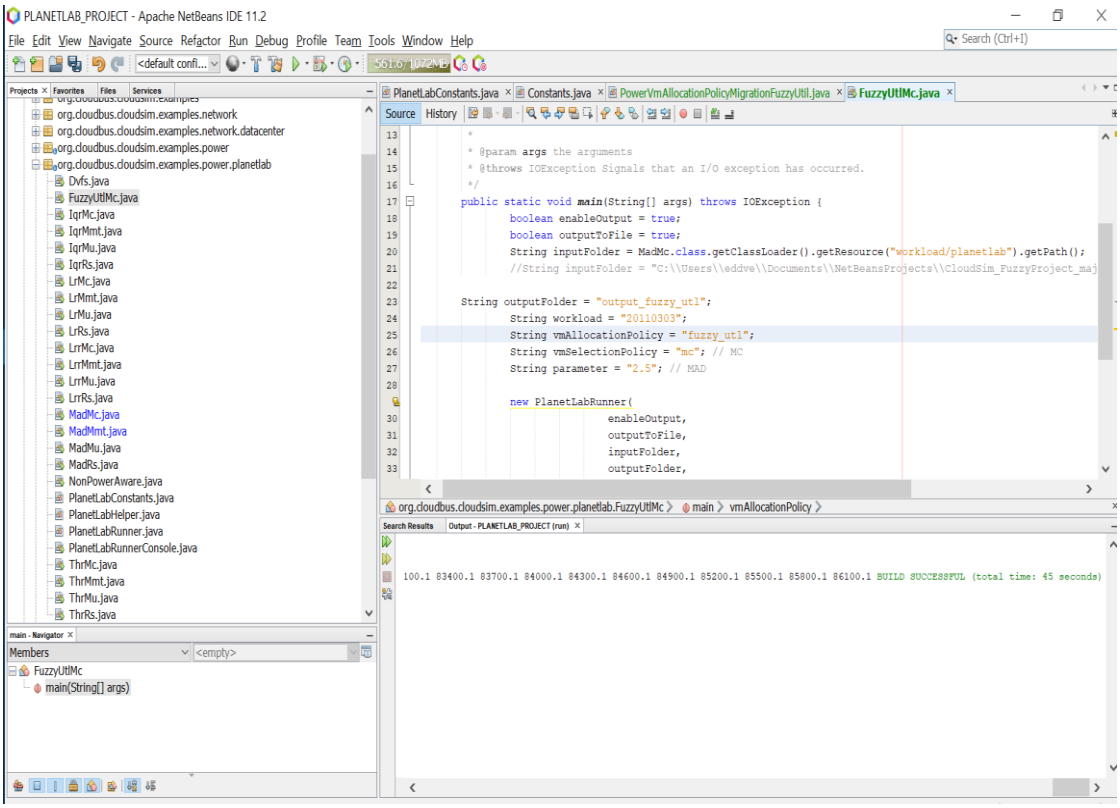
- počet hibernácií uzlov.

Tabuľka 5.2. Typy virtuálnych strojov použitých pre simuláciu

Názov	VM1	VM2	VM3	VM4
Počet prvkov spracovania	1	1	1	1
Výkon (MIPS)	2500	2000	1000	500
Pamäť RAM (GB)	870	1740	1740	613

5.2 Simulačné prostredie CloudSim

CloudSim je simulačné prostredie (Obrázok 5.1) podporujúce množstvo operačných systémov. Umožňuje simuláciu diskrétnych udalostí pre výpočty v rámci simulovaných cloudových prostredí, vyvinutý laboratóriom CLOUDS pri Melbournskej Univerzite. [25]



Obrázok 5.1. Prostredie CloudSim

S rastom používania cloudového počítania bol spustený projekt zameraný na vytvorenie všeobecného prostredia, rozšíriteľný o simuláciu, ktorá podporuje

modelovanie a integrované experimenty pre rozsiahle cloudové služby. Z dôvodu úzkeho vzťahu medzi cloudovým a gridovým počítaním, bol CloudSim vyvíjaný nad simulátorom GridSim. Využíva jeho základnú implementáciu pre vkladanie nových abstrakcií prispôsobenú pre cloudové systémy (simulácia infraštruktúry, virtualizácia alebo modelovanie dátových centier, politiky plánovania, prideľovanie zdrojov a úloh). CloudSim, a zdrojové balíky pre vytváranie jednotlivých komponentov cloudu, sú napísané v programovacom jazyku Java.

CloudSim podporuje poskytovanie zdrojov na dvoch úrovniach: na úrovni uzla a na úrovni virtuálnych strojov. Na úrovni uzla môžeme určiť, koľko z celkového výkonu bude spracované a pridelené každému virtuálnemu stroju. Na úrovni virtuálneho stroja dostávajú a používajú aplikácie časť výpočtového výkonu priradená virtuálnemu stroju. Medzi podporovanými distribučnými modelmi cloudového počítania v CloudSim sú zahrnuté IaaS, PaaS a SaaS.

5.2.1 Testovacie úlohy pre simulačné prostredie CloudSim

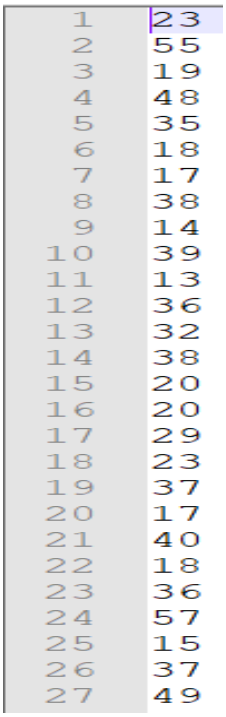
Úlohu, pracovnú záťaž, reprezentujú v programe CloudSim tzv. cloudlety. [40] Cloudlet opisuje vlastnosti vzťahujúce sa k pracovnému zaťaženiu, ktoré chceme aby bolo prenesené na virtuálny stroj. Cloudlet je tiež modelová trieda, ktorá definuje špecifikácie simulačnému nástroju, zodpovedajúca reálnej aplikácii. Veľkosť cloudletu je odhadovaná veľkosť množiny pokynov, ktoré sa majú vykonať v priebehu simulačného testu a udáva sa v bajtoch. Spolu s tým sú určené ďalšie dva atribúty, pre veľkosť vstupného súboru a veľkosť výstupného súboru. Zmienené atribúty sú veľmi dôležité, pretože sa používajú na výpočty súvisiace so šírkou prenosového pásma.

Pre naše testovanie sme vybrali množinu údajov, ktoré sú používané ako cloudlety pre simuláciu. Množina údajov pozostáva z hodnôt záťaže procesora z reálneho počítačového systému (uzla). Podľa počtu cloudletov sa v priebehu simulácie vytvára rovnaký počet virtuálnych strojov. Každý cloudlet obsahuje informácie o záťaži, ktorú bude predstavovať virtuálny stroj na priradenom uzle.

Štruktúra zdrojového súboru určeného ako cloudlet je zobrazená na Obrázok 5.2. Ľavý stĺpec zodpovedá poradovému číslu záznamu a pravý stĺpec je hodnota MIPS použitá ako záťaž pre virtuálny stroj. Celkovo máme k dispozícii informácie o 1052 reálnych počítačových systémoch. Z každého jedného reálneho počítačového systému

je dostupných 288 informácií záťaže procesora, ktoré boli zaznamenané každých 5 minút počas jedného dňa (z 3. marca 2011, označenie 20110303).

Tieto voľne dostupné údaje pochádzajú z organizácie PlanetLab, ktorá má v súčasnosti viac ako 1350 výpočtových uzlov (nodov) geograficky rozmiestnených na viac ako 700 miestach. [41]



Obrázok 5.2. Ukážka obsahu cloudletu

5.2.2 Príprava zdrojov

Pretože je CloudSim simulátor, nie je na ňom možné spúšťať bežné aplikácie. Je určený na vykonávanie simulácií, experimentovanie s rôznymi algoritmami (plánovania, zeleného počítania), prideľovania a migrovania virtuálnych strojov.

Počet a vlastnosti vytváraných jednotlivých virtuálnych strojov, sú definované pomocou premenných *public final static int,* ktoré určujú*:*

- *VM_TYPES* – počet virtuálnych strojov.
- VM_MIPS – výkon virtuálneho stroja.
- VM_PES – počet prvkov spracovania.
- VM_RAM – veľkosť operačnej pamäte.
- VM_BW – rýchlosť siete.
- VM_SIZE – veľkosť virtuálneho stroja.

Počet a vlastnosti vytváraných jednotlivých uzlov sú definované rovnakým spôsobom ako v predchádzajúcom prípade virtuálne stroje. Rozdielom je len prefix namiesto VM -> HOST a SIZE -> STORAGE.

Ako sme spomenuli v kapitole pre testovaciu úlohu (pozri kapitolu 5.2.1), z uvedených informácií využitia procesora jednotlivých uzlov, máme pre simuláciu dostupné informácie pre 1052 virtuálnych strojov.

5.3 Experimentálne overovanie navrhnutých algoritmov

Tieto experimenty navrhujeme s cieľom potvrdiť alebo vyvrátiť hypotézu *H2 - Zelené počítanie v cloudových systémoch spomaľuje činnosť počítačového cloudu, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme.*

Z dôvodu, že aplikácia každej z vybraných techník zeleného aj nezeleného počítania, má vplyv na simuláciu z pohľadu jej časového dokončenia a získaniu výsledkov, vykonali sme ich spustenie celkovo desať krát. Rozdiely v nameraných časoch dokončenia simulácie sú ovplyvnené ich náročnosťou na počítačový systém, na ktorom sú spúšťané. Náročnosť je tvorená počtom migrácií a spôsobom detekcie preťaženia uzla jednotlivými technikami. Aj tento údaj vykresľuje rozdielny prístup jednotlivých riešení. Presné označenie testovacej úlohy je workload.planetlab.20110303.

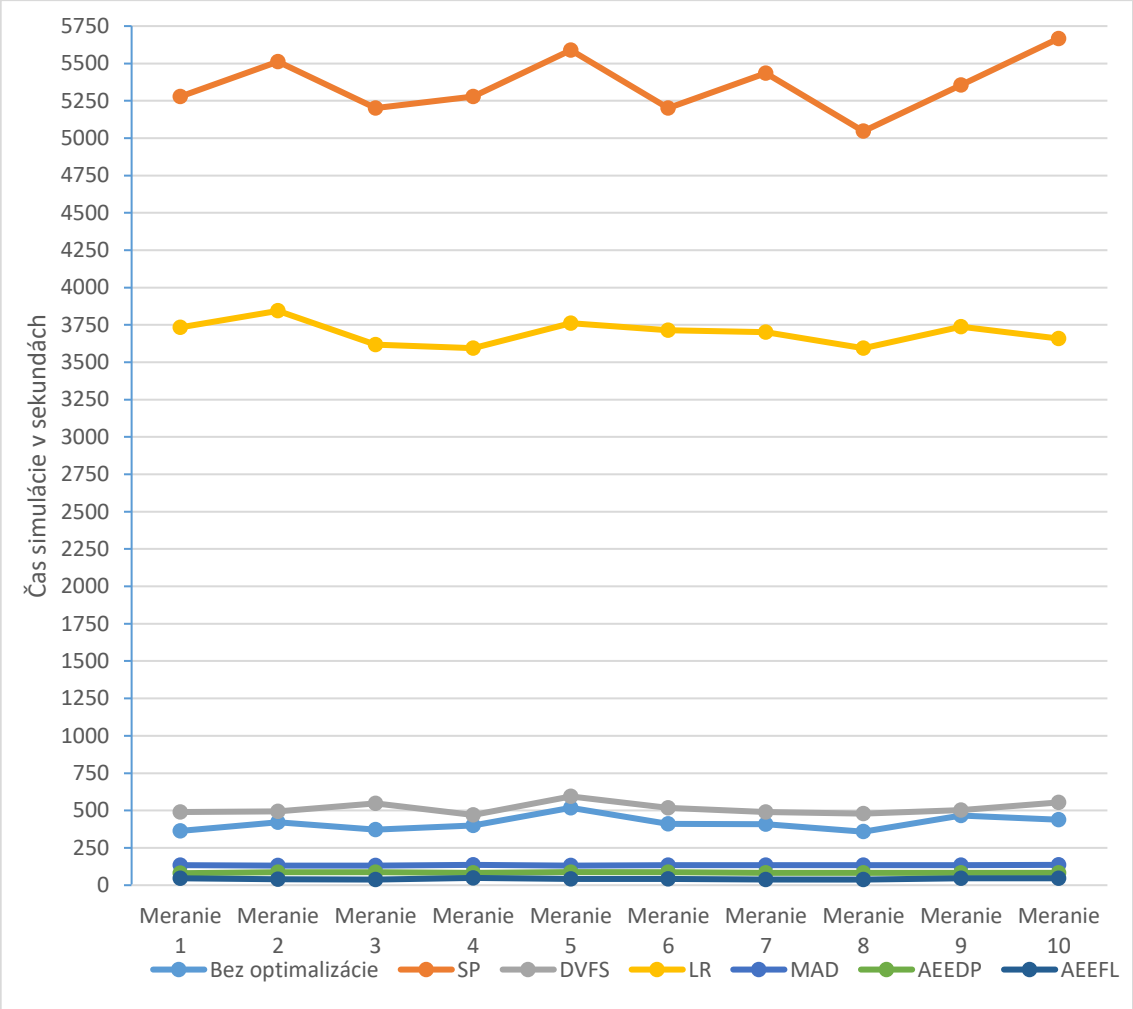
Po vykonaní série desiatich simulácií sú časy zaznamenané v Tabuľke 5.3. Z nameraných časových údajov vidíme minimá a maximá trvania doby simulácie v sekundách pre konkrétnu techniku. Priemerné rozpätie medzi najkratším a najdlhším nameraným časom všetkých techník je 18,85%. Zo všetkých techník bol najkratší nameraný čas pre AEEFL. Pre lepšiu prehľadnosť je najkratší dosiahnutý čas každej techniky zvýraznený tučným.

Tabuľka 5.3. Namerané výsledky času trvania simulácie v sekundách (najkratší čas zvýraznený tučným)

Číslo merania	Bez optimalizácie	SP	DVFS	LR	MAD	AEEDP	AEEFL
1	363	5279	489	3734	134	**80,24**	46,65
2	421	5511	493	3844	**131**	87,11	39,34
3	372	5201	547	3617	**131**	85,87	38,54
4	400	5278	**470**	3595	136	83,44	48,77
5	518	5589	594	3762	**131**	87,82	42,78

6	411	5201	518	3714	134	87,38	42,12
7	409	5434	489	3702	134	82,58	**38**
8	**358**	**5046**	479	**3594**	133	81,78	38,17
9	466	5356	503	3737	134	83,68	46,23
10	439	5667	554	3658	136	82,94	46,25

Namerané výsledky času trvania simulácie v sekundách sú graficky vykreslené na Grafe 5.1.



Graf 5.1. Celková spotreba elektrickej energie k počtu vykonaných migrácií

Tabuľka 5.4 uvádza údaj mediánu pre výsledky času trvania simulácie v sekundách za všetky vykonané merania (Tabuľka 5.3).

Tabuľka 5.4. Medián pre výsledky času trvania simulácie v sekundách (najkratší čas zvýraznený tučným)

Údaj	Bez optimalizácie	SP	DVFS	LR	MAD	AEEDP	AEEFL
Medián meraní	410	5317,5	498	3708	134	83,56	**42,45**

V Tabuľke 5.5 je uvedená celková energetická náročnosť všetkých simulovaných virtuálnych strojov za časový úsek 1 deň. Simulované dátové centrum sa skladá z 800 uzlov a 1052 virtuálnych strojov.

Možné zníženie energetickej náročnosti, oproti simulácii bez optimalizácie, je dosiahnuté konsolidáciou virtuálnych strojov, uvoľnením uzla a tým jeho neaktivitu. V tomto stave je možné uzol hibernovať a tým výrazne znížiť celkovú spotrebu energie. Aj v tomto prípade vyšla technika AEEFL zo všetkých testovaných najlepšie, keď sme zaznamenali zníženie energetickej náročnosti oproti bez optimalizácie až o 2559% (25 násobne menšiu). V porovnaní s inými technikami zeleného počítania, bol najmenší rozdiel medzi SP a AEEFL, a to konkrétne o 69,64% v prospech AEEFL.

Tabuľka 5.5. Energetická náročnosť dátového centra v kWh za sledované obdobie (najnižšia spotreba zvýraznená tučným)

Typ úlohy	Bez optimalizácie	SP	DVFS	LR	MAD	AEEDP	AEEFL
workload.20110303	2410,80	153,76	803,91	150,33	176,13	289,14	**90,64**

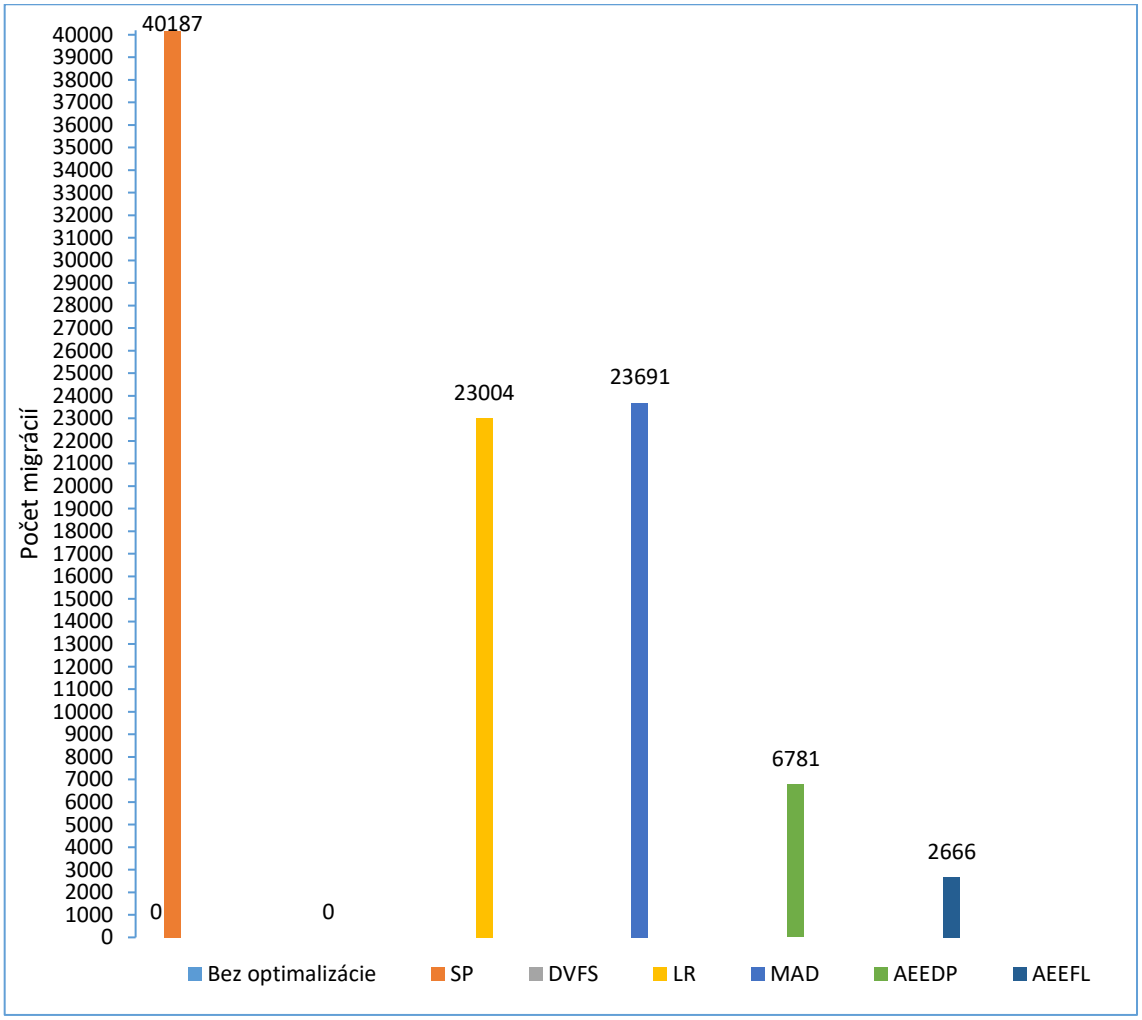
Počet vykonaných migrácií virtuálnych strojov počas simulácie vytvoreného cloudového prostredia pre navrhnuté algoritmy a jednotlivé techniky zeleného počítania s kritériom výberu virtuálneho stroja algoritmu maximálnej korelácie, sú uvedené v Tabuľke 5.6. Z výsledkov v tabuľke vidíme, že náš algoritmus používajúci fuzzy logiku sa javí najlepšie. Počet migrácií je pri tejto technike až o 88,4% nižší ako pri najlepšom výsledku zo známych techník LR a až o 93.3% nižší v porovnaní s technikou SP.

Menší počet migrácií virtuálnych strojov predstavuje efektívnu konsolidáciu, dodržanie celkového kritéria SLA a v konečnom dôsledku aj menšie množstvo sieťového zaťaženia.

Tabuľka 5.6. Počet vykonaných migrácií virtuálnych strojov za sledované obdobie

Typ úlohy	Bez optimalizácie	SP	DVFS	LR	MAD	AEEDP	AEEFL
workload.20110303	0	40187	0	23004	23691	6781	2666

Počet vykonaných migrácií virtuálnych strojov za sledované obdobie je graficky vykreslený na Grafe 5.2.



Graf 5.2. Počet vykonaných migrácií virtuálnych strojov za sledované obdobie

Počet vykonaných migrácií má priamy vplyv na zníženie energetickej náročnosti, ale aj na dodržiavanie, resp. porušovanie SLA.

V Tabuľke 5.7 je zaznamenané miera porušenia SLA v percentách. Tieto porušenia SLA sú v CloudSime automaticky počítané podľa metriky pre výpočet porušenia SLA [19] počas simulácie a vyjadrené vzťahom 5.1:

$$SLA = OTF * PDM \quad (5.1.)$$

OTF (vzťah 5.2) vyjadruje dobu preťaženia uzla a *PDM* (vzťah 5.3) zníženie výkonu uzla v dôsledku migrácie.

$$OTF = \frac{1}{N}\sum_{i=1}^{N}\frac{Ts_i}{Ta_i} \quad (5.2.)$$

$$PDM = \frac{1}{M}\sum_{j=i}^{M}\frac{Cd_j}{Cr_j} \quad (5.3.)$$

N je počet uzlov; Ts_i je celkový čas, počas ktorého bol uzol *i* preťažený (využitý nad stanovenú hranicu), vedúci k porušeniu SLA; Ta_i je celkový čas, kedy bol uzol *i* v aktívnom stave; *M* je počet virtuálnych strojov; Cd_j je odhad zhoršenia výkonu virtuálneho stroja *j* spôsobený migráciou a Cr_j je celková požadovaná CPU kapacita virtuálnym strojom *j*, počas jeho životnosti. [19]

Pri celkovom hodnotení neberieme do úvahy techniky SP a DVFS pretože sa pri nich nevykonávajú migrácie. Z ostatných techník dosiahla najlepší výsledok naša technika AEEFL, využívajúca fuzzy logiku pre výber preťaženého uzla. Výsledok porušenie SLA bol v tomto prípade o 13.97% nižší ako dosiahla druhá najlepšia technika LR.

Tabuľka 5.7. Prehľad porušení SLA

Merania	Bez optimalizácie	SP	DVFS	LR	MAD	AEEDP	AEEFL
Porušenie SLA (v %)	0	0,01619	0	0,00677	0,00739	0,01518	0,00594

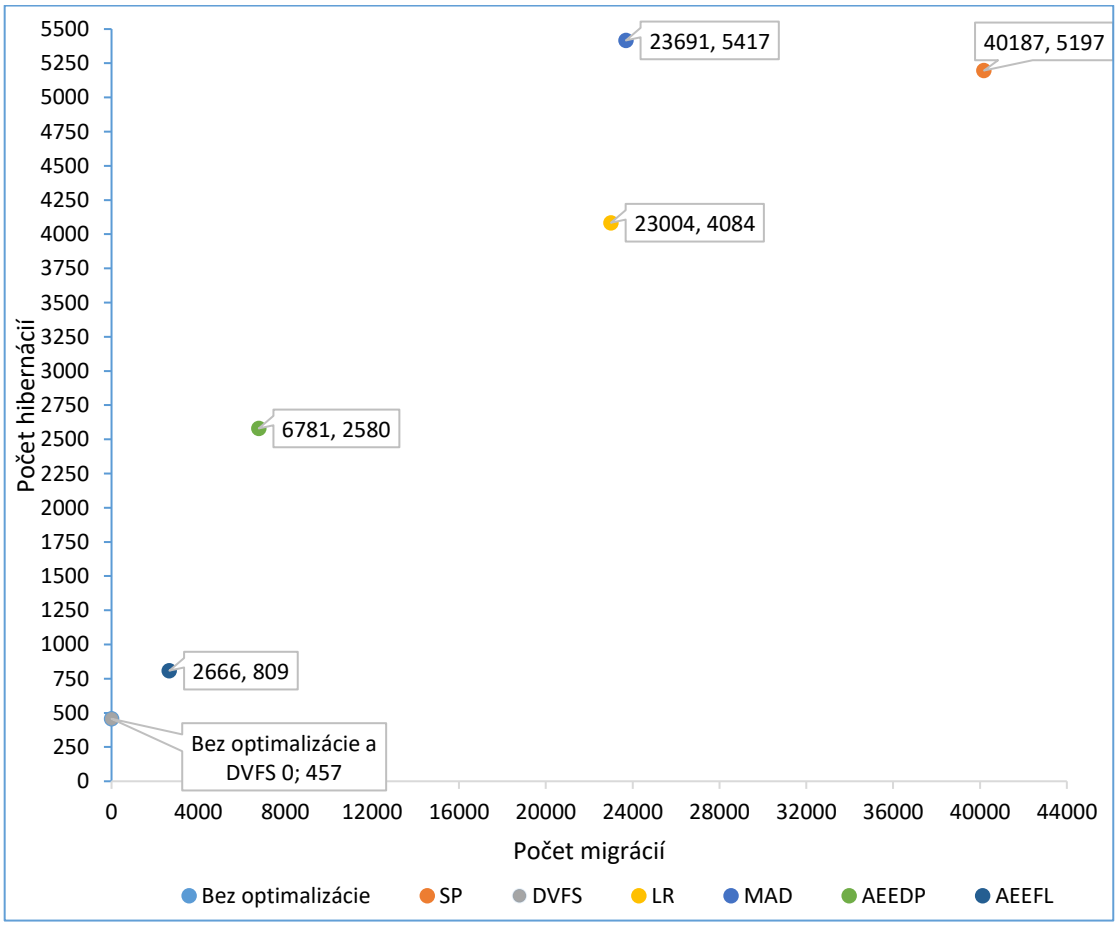
Počet hibernovaných uzlov a priemerné časy (v sekundách) medzi jednotlivými hibernáciami počas dennej simulácie dátového centra pre jednotlivé techniky uvádza Tabuľka 5.8. Ako sme spomenuli, uzol môže byť hibernovaný vtedy, keď nie je zaťažený nad stanovenú úroveň minimálneho zaťaženia. Najskôr je potrebné správne

identifikovať uzol, a v prípade, keď sa na identifikovanom uzle nachádzajú virtuálne stroje, prebehne ich migrácia na iný uzol.

Tabuľka 5.8. Počet hibernovaných uzlov a priemerné časy medzi jednotlivými hibernáciami (najväčší počet a najmenší čas zvýraznený tučným)

Meraní údaj	Bez optimalizácie	SP	DVFS	LR	MAD	AEEDP	AEEFL
Počet hibernácií	457	5197	457	4084	**5417**	2580	809
Priemerný čas (s)	3336,21	954,43	3336,21	1150,6	**1001,46**	3449,3	1081,32

Vzťah, medzi počtom hibernácií uzlov a vykonaných migrácií virtuálnych strojov za sledované obdobie, je graficky vykreslený na Grafe 5.3. Z grafu je možné jasne konštatovať, že platí vzťah, čím viac je vykonaných migrácií virtuálnych strojov, tým prichádza k väčšiemu počtu hibernácií uzlov.



Graf 5.3. Počet hibernácií uzlov a vykonaných migrácií virtuálnych strojov za sledované obdobie

Na základe analýzy získaných výsledkov vykonaných simulácií a meraní, môžeme vyhodnotiť vplyv jednotlivých metód a algoritmov na stanovené kritéria. Z pohľadu výsledného času vykonania simulácie (Tabuľka 6.1) pozorujeme vzťah, medzi časom vykonania a počtom uskutočnených migrácií virtuálnych strojov počas nej. Pri technike statického prahu je čas simulácie a aj počet migrácií najvyšší. K miernej odchýlke prichádza pri technike LR a MAD, pričom výsledok počtu migrácií je podobný ale čas vykonania simulácie je niekoľkonásobne väčší pre techniku LR.

Výsledný rozdiel v celkovej elektrickej spotrebe simulovaného dátového centra počas doby 1 deň je medzi najhoršou technikou (bez optimalizácie), a najlepšou (algoritmus energetickej efektívnosti – fuzzy logika) viac ako 25 násobný.

Už z povahy aplikovania techniky bez optimalizácie a techniky DVFS, je zrejmé, že nepríde, a ani neprišlo k vykonaniu žiadnej migrácii.

Ako vidíme na grafe 5.4, neplatí v každom prípade vzťah, že väčší počet migrácií sa automaticky rovná nižšej celkovej elektrickej spotrebe. Na algoritme AEEFL môžeme vidieť, že aj napriek vykonaniu menšieho počtu migrácií oproti technikám s vyšším počtom, sme dosiahli najlepšiu (najnižšiu) spotrebu dátového centra počas simulovanej doby.

Graf 5.4. Celková spotreba elektrickej energie k počtu vykonaných migrácií

Experimenty overenia riešenia v simulačnom prostredí CloudSim ukázali, že hypotéza H2 je vyvrátená. To znamená, že neplatí, že zelené počítanie v cloudových systémoch spomaľuje činnosť počítačového cloudu, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme.

Aplikovaním techník zeleného počítania sa dosiahla výrazná úspora elektrickej energie v rámci simulovaného dátového centra a miera porušenia SLA vzrástla len minimálne, oproti neaplikovaniu techniky bez optimalizácie (Tabuľka 5.7). V tomto smere sa meraniami najlepšie ukázal náš algoritmus používajúci fuzzy logiku pre detekciu preťaženia uzla. Hodnota porušenia SLA, počas doby sledovanej simulácie 1 deň na zvolenej testovacej úlohe, predstavovala akceptovanú hodnotu iba 0,00594%.

6 Metódy riešenia aplikácií v cloudových systémoch s cieľom dosiahnuť zelené počítanie

Táto kapitola obsahuje modely riešenia, postupnosť metód, nami navrhnutú metodiku práce a experimentálne overenie úloh v reálnom prostredí.

6.1 Metodika overovania riešenia v reálnom cloudovom systéme

V ponuke dostupného verejného cloudu, vhodného na vykonanie vysokovýkonného počítania, existuje niekoľko poskytovateľov služieb – Amazon, IBM, Microsoft Azure, Google Cloud atď. Pre výskum v reálnom cloudovom prostredí sme zvolili službu Amazon Web Services, konkrétne Amazon EC2, od spoločnosti Amazon. Modelom nasadenia je infraštruktúra ako Služba.

Dôvodom pre výber a uprednostnenie služby Amazon EC2 je všeobecne široká ponuka inštancií určených pre HPC účely. Dostupné sú inštancie zamerané na intenzívne počítanie prebiehajúce pomocou GPU, výpočtovo orientované, pamäťovo orientované intenzívne počítanie alebo aj požiadavky na nízku odozvu prepojovacej siete medzi uzlami. Napriek použitiu verejného cloudového počítania sú výhody v neexistujúcom čakaní na pridelenie požadovaných výpočtových zdrojov (zdroje sú dostupné takmer okamžite) a z pohľadu paralelizácie výpočtov možnosť vytvorenia elastického klastra do šírky. Preto sme vytvorili virtuálny klaster typu inštancie *c4.4xlarge*, ktorá je výpočtovo orientovaná, s dynamickým počtom spustených nodov v počte od 2 po maximálne 5. Vlastnosti inštancie sú uvedené v Tabuľke 6.1. Konkrétny typ inštancie bol vybraný po otestovaný spolu s menej a viac výkonnými typmi inštancií. Pri výkonnejších inštanciách sme pozorovali problém spôsobený veľkým počtom dostupných procesorov a paralelizáciou úlohy. Pri najväčšom počte grafov, 9 967 500 (pozri kapitolu 6.1.1), nebolo možné dokončiť výpočet z dôvodu nedostatku pamäte.

V prostredí reálneho cloudu sme použili 3 typy testovacích úloh (pozri kapitolu 6.1.1).

Tabuľka 6.1. Vlastnosti inštancie c4.4xlarge v službe Amazon EC2

Názov	Hodnota
vCPU[1]	16
Pamäť RAM (GB)	30
Výkon siete	Vysoký

Referenčné merania pre prvý druh úlohy sme urobili bez aplikovania optimalizačnej metódy. Jednou možnosťou je skrátenie doby výpočtu.

Pri druhom a treťom type testovanej úlohy sme urobili výpočty na reálnom klastri s virtuálnymi strojmi, pričom sme merali dobu výpočtu pre každú úlohu. Následne sme porovnali namerané výsledky z reálneho cloudového prostredia s fyzickým počítačovým klastrom.

6.1.1 Testovacie úlohy pre reálne prostredie cloudového systému

V reálnom prostredí cloudového systému sme sa rozhodli použiť tri typy testovacích úlohy. Prvá úloha, využíva programovací model rozhrania na výmenu správ (MPI) s názvom Cloverleaf. [42] Cloverleaf vychádza z množiny miniaplikácií, ktoré sú určené pre vysokovýkonné počítače a vytvorené spoluprácou niekoľkých britských inštitúcií, známych ako spoločenstvo UK Mini-App Consortium.

Druhý typ úlohy [56], predstavuje paralelný program na ofarbovanie hrán grafov, ktorý obsahuje rôzne nadväzujúce a neprerušiteľné podúlohy. Úloha používa sadu grafov z databázy House of Graphs, ktorá pozostáva z 19 935 34-vrcholových snarkov. Výpočet úlohy sme vykonali pre niekoľko veľkostí množín o veľkosti:

- 100 grafov,
- 1 000 grafov,
- 1 000 000 grafov,
- 9 967 500 grafov.

Program obsahuje dekompozície na podúlohy. Následne boli dekompozície úloh vytvorené tak, aby každý výpočet hranového ofarbenia grafu trval približne rovnako dlhý čas.

[1] Každý vCPU je vlákno na fyzickom procesore typu 2.9 GHz Intel Xeon E5-2666 v3

Úlohy tretieho typu sú tiež sady grafov z databázy House of Graphs [66]. V tomto prípade sme overovali nasledovný počet grafov pre snarky s 34 vrcholmi:

- 19 935 grafov,
- 3 833 587 grafov,
- 25 286 953 grafov.

Z dôvodu intenzívnej pamäťovej náročnosti vyššie uvedených grafov s 34 vrcholmi, sme museli rozdeliť dve najväčšie sady grafov do niekoľkých desiatok dávok.

6.2 Reálne prostredie cloudového systému

Možnosti ponúkané zvolenou službou Amazon EC2 spĺňajú naše požiadavky pre stanovené kritéria jednorazového testovania úlohy v rámci vysokovýkonného počítania. Týmito kritériami sú výber dostupného regiónu datacentra, dostupných výpočtových zdrojov a nástroja na manažovanie vytvoreného klastra.

Vhodne zvolený dostupný región, v ktorom je umiestnené datacentrum, je dôležitý z niekoľkých hľadísk:

- Prvým, je dosiahnutie relatívnej blízkosti výpočtových zdrojov k miestu lokálneho používateľa. Táto vzdialenosť ovplyvňuje primárne kvalitu internetového spojenia, jeho odozvu.
- Druhým, je paleta ponúkaných výpočtových zdrojov. Každé datacentrum umožňuje použiť len tie výpočtové zdroje, ktoré sú mu priradené. Rozdiely existujú v typoch a počtoch rodiny procesorov, grafických procesorov, ale aj špeciálnych procesorov určených pre umelú inteligenciu.

Aby sme dosiahli vyššie spomenuté kritériá vytvorili sme klaster v európskom regióne, lokalita London.

6.2.1 Príprava zdrojov

Hardvérové vybavenie virtuálneho klastra (počet procesorov, jadier a pamäte RAM) sme zvolili tak, aby bolo možné na týchto prostriedkov počítať testovacie úlohy. Vybraná testovacia úloha je opísaná v kapitole 6.1.1. Z ponuky všetkých dostupných konfigurácií virtuálnych strojov [3] je pri overovaní riešenia použitý typ inštancie optimalizovaný na intenzívne výpočty *c4.4xlarge*. Model procesora, nad ktorým sú

vytvárané vCPU v rámci inštancie, predstavuje Intel Xeon E5-2666 v3 s taktovacou frekvenciou 2.9 GHz. Ďalšie vlastnosti tohto typu inštancie sú uvedené v Tabuľke 6.2.

Tabuľka 6.2. Hardvérové prostriedky inštancie

Komponenty	**Počet / Veľkosť**
vCPU (ks)	16
RAM (GB)	30

Základným prvkom softvérového vybavenia je použitý operačný systém Ubuntu Linux 16.04 LTS. V rámci operačného systému sme museli pre riadne spustenie úlohy a jej vykonávanie ďalej doinštalovať podporné nástroje pre kompilovanie jazyka C (v ktorom je napísaná zdrojová úloha) a OpenMPI.

6.2.2 Správca klastra

Aby sme mohli jednoducho, rýchlo a automatizovane vytvárať resp. odstraňovať vysokovýkonné klastre, v rámci služby AWS, použili sme voľne dostupný softvér správcu klastra AWS ParallelCluster. [28] AWS ParallelCluster sa inštaluje v prostredí operačného systému na lokálnom počítači. Prvotná konfigurácia po inštalácii prebieha pomocou nastavení *account credentials* a *security key* patriacim k vytvorenému účtu v AWS.

Pri nastavovaní konfiguračného súboru (Obrázok 6.1) pred prvotným vytvorením klastra, je potrebné definovať základné parametre. Pomocou konfiguračného súboru máme takmer neobmedzené možnosti nastavenia vlastností a počtov vytváraných virtuálnych klastrov.

```
[global]
cluster_template = WorkGC
update_check = true
sanity_check = true
[aws]
aws_region_name = eu-west-2
[cluster cloverleafGC]
key_name = key-#######
compute_instance_type = c4.4xlarge
master_instance_type = c4.4xlarge
initial_queue_size = 2
max_queue_size = 5
maintain_initial_size = false
scheduler = torque
cluster_type = ondemand
s3_read_resource = NONE
shared_dir = cloverleafGC
master_root_volume_size = 10
compute_root_volume_size = 10
base_os = ubuntu1604
[scaling custom]
scaledown_idletime = 1
```

Obrázok 6.1. Konfiguračný súbor vytváraného klastra

Vzhľadom na náš plánovaný výskum je najdôležitejšie vhodne zvoliť a nastaviť parametre týkajúce sa:

- typu inštancie požadovanej pre master nod,
- typu inštancie požadovanej pre výpočtový nod,
- zvoleného plánovača rozvrhovania záťaže,
- počiatočnej veľkosti klastra (počet inštancií),
- maximálnej veľkosti klastra (počet inštancií).

Medzi ostatné parametre patria požadovaný región, v ktorom sú inštancie vytvorené, typ úložiska, šifrovanie, nastavenie siete. Medzi spomenuté plánovače rozvrhovania záťaže patria AWS Batch, SGE, Torque a Slurm. Zadefinovali sme tiež parameter pre automatické škálovanie pre zníženie vytvorených inštancií v prípade ich nečinnosti minimálne 1 minútu.

My sme vybrali a použili plánovač Torque. Manažovanie virtuálneho klastra sa vykonáva prostredníctvom CLI.

6.3 Experimentálne overovanie úlohy CloverLeaf

Po vytvorení zadefinovaného klastra, prihlásení sa na nod je potrebné vytvoriť súbor so spúšťacími parametrami a kompiláciou.

Namerané časy (Tabuľka 6.3), bez aplikácie optimalizačnej metódy zeleného počítania, zodpovedajú základnému stavu. Tieto hodnoty budú slúžiť ako počiatočný stav pri ďalšom porovnávaní s vybranými optimalizačnými metódami.

Tabuľka 6.3. Namerané výsledky v sekundách

Typ úlohy	Bez optimalizácie
clover_bm_short.in	0.029
clover_bm.in	1.162
clover_bm16_short.in	0.813
clover_bm16.in	19.767

Z nameraných východiskových hodnôt bez použitia optimalizácie je možnosťou ich zlepšenia použitie optimalizačných techník pri plánovači Torque. [10]

6.4 Experimentálne overovanie úlohy ofarbovania grafov

Táto časť práce slúži na potvrdenie, príp. vyvrátenie hypotézy *H1 - Cloudové systémy, na báze virtuálnych strojov, spomaľujú činnosť fyzických výpočtových prostriedkov, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme.* Rovnako tiež skúmame potvrdenie, príp. vyvrátenie hypotézy *H3 - Dekompozícia cloudovej aplikácie negatívne ovplyvňuje náklady na prevádzku výpočtového systému*.

Pri druhej zvolenej úlohe sa venujeme správaniu úlohy a jej porovnaniu fyzického počítačového klastra (Tabuľka 6.4) a využitím dostupných výpočtových systémov s virtualizáciou na báze virtuálnych strojov (pozri kap. 6.1). Hlavným cieľom skúmania druhej úlohy je vplyv virtualizácie na algoritmy obsahujúce rekurzívne funkcie.

Tabuľka 6.4. Hardvérové prostriedky pre fyzický počítačový klaster

Názov	Fyzický počítačový klaster
Počet uzlov (ks)	5
Počet procesorov na uzol (ks)	16
Operačná pamäť RAM (GB)	4

Ako sme spomenuli v kapitole 6.1.1, výpočet sme vykonali celkovo pre 4 veľkosti množiny grafov. Pri malých úlohách (100, 1 000 a 1 000 000 grafov), sa na základe celkových časov výpočtu preukazoval systém s virtualizáciou podstatne lepší ako systém s fyzickými strojmi (Tabuľka 6.5, Tabuľka 6.6 a 6,7).

Tabuľka 6.5. Namerané hodnoty pre úlohu 100 grafov

Názov	Fyzický počítačový klaster	Virtuálny klaster
Celkový čas výpočtu	< sekunda	< sekunda
CPU čas výpočtu	3 sekundy	< sekunda
Priemerný čas ofarbovania jedného grafu	0,03 sekundy	0.00052

Tabuľka 6.6. Namerané hodnoty pre úlohu 1 000 grafov

Názov	Fyzický počítačový klaster	Virtuálny klaster
Celkový čas výpočtu	< sekunda	< sekunda
CPU čas výpočtu	16 sekúnd	3,244 sekundy
Priemerný čas ofarbovania jedného grafu	0,016 sekundy	0.003244 sekundy

S narastajúcou veľkosťou ofarbovanej úlohy sa ale zvolené výpočtové systémy začali výkonnostne približovať.

Tabuľka 6.7. Namerané hodnoty pre úlohu 1 000 000 grafov

Názov	Fyzický počítačový klaster	Virtuálny klaster
Celkový čas výpočtu	44:58 minút	15:10 minút
CPU čas výpočtu	34:59 minút	2:07 minút
Priemerný čas ofarbovania jedného grafu	0,002 sekundy	0.000127

Ako vidíme na predchádzajúcich tabuľkách, použitie virtualizácie pri výpočtoch spojených s hranovým ofarbovaním grafov pomocou paralelných algoritmov obsahujúcich rekurzívne funkcie, nevytvára spomalenie alebo oneskorenie výpočtov. V každom z vykonaných meraní boli dosiahnuté časy na virtuálnom klastri niekoľkonásobne menšie.

Pre najnáročnejšiu úlohu, 9 967 500 grafov, sú uvedené namerané hodnoty v Tabuľke 6.8.

Tabuľka 6.8. Namerané hodnoty pre úlohu 9 967 500 grafov

Názov	Fyzický počítačový klaster	Virtuálny klaster
Celkový čas výpočtu (hod.)	02:41:13	02:52:35
CPU čas výpočtu (hod.)	04:53:38	03:20:54
Minimálny počet rekurzívnych volaní	31	548
Maximálny počet rekurzívnych volaní	1 283 104	866 762
Minimálny čas ofarbovania (ms)	< 1	< 1
Maximálny čas ofarbovania (ms)	1 260	29

Hypotézu H1 môžeme vyvrátiť z pohľadu experimentov, ktoré reprezentujú namerané hodnoty v tabuľkách 6.5, 6.6 a 6.7. Akonáhle sa výpočtová zložitosť (resp. náročnosť) úlohy zvyšuje, výkon medzi fyzickým počítačovým systémom a virtuálnym klastrom sa začína vyrovnávať (pozri tabuľku 6.8). V tomto hraničnom prípade už ide o ofarbovanie veľkej množiny grafov – cca 10 miliónov. Pričom sa dá predpokladať,

že pri ďalšom zväčšovaní množiny grafov sa čas výpočtu na cloude s virtuálnymi strojmi podstatne zhorší.

Na základe nameraných hodnôt konštatujeme, že pokiaľ nejde o mimoriadne veľké výpočtové úlohy, môžeme vyvrátiť hypotézu *H1 - Cloudové systémy, na báze virtuálnych strojov, spomaľujú činnosť fyzických výpočtových prostriedkov, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme.*

Pri vyhodnocovaní a skúmaní hypotézy *H3* sme použili tri rôzne veľkosti sady grafov s 34 vrcholmi. Kritériom pre vyvodenie záveru je porovnanie dosiahnutých časov výpočtu ofarbovania grafov s použitím dekomponovaných podúloh, Porovnávanie robíme znova medzi výpočtami na fyzickom výpočtovom klastri a cloudovom klastri s virtuálnymi strojmi. Znova, doktorand Mgr. Adam Dudáš poskytol programy a namerané výsledky časov ofarbovania grafov, ktoré robil na fyzickom výpočtovom klastri, bez použitia virtuálnych strojov. Pri overovaní na fyzickom počítačovom klastri (Tabuľka 6.4) bola pridelená operačná pamäť:

- Množina grafov 3 833 587 potrebovala 994,59 GB RAM.
- Množina grafov 25 286 953 potrebovala 3 219,34 GB RAM.

Najmenšou testovacou množinou bolo 19 935 grafov. Model dekompozície úlohy bol rovnaký pre oba sledované počítačové systémy.

Pri úlohách o veľkosti množiny grafov 3 833 587 a 25 286 953 sme pozorovali výrazne rozdiely medzi jednotlivými typmi klastrov, a to z pohľadu úspešnosti dokončenia úlohy. Ani pri experimentálnej testovacej zostave virtuálneho klastra s dostupnou operačnou pamäťou 768GB RAM, nebola úloha pri použití jednotného modelu dekompozície úspešne dokončená. Zvyšujúca sa pamäťová náročnosť uvedených úloh, s pôvodným modelom dekompozície, spôsobovala zlyhanie vykonania vo virtuálnom prostredí.

Z tohto dôvodu sme museli obe zvyšné množiny grafov dekomponovať na maximálne rovnako veľké podúlohy. Výskumom sme dospeli k nasledovnému modelu dekompozície:

- Množina 3 833 587 grafov bola rozdelená na 39 dávok.
- Množina 25 286 953 grafov bola rozdelená na 253 dávok

Následne boli podúlohy v dávkach spracované za účelom namerania časov dokončenia. Namerané hodnoty sú uvedené v Tabuľke 6.9 a 6.10. Z nameraných výsledkov vidíme, že pri zmenenom dávkovaní úlohy množiny grafov 3 833 587,

dosiahol virtuálny klaster o 36,33% lepší čas dokončenia. Naopak, pri čase dokončenia množiny grafov 25 286 953, sa na virtuálnom klastri prejavilo zhoršenie o 11,74%

Tabuľka 6.9. Namerané hodnoty pre množinu 3 833 587 grafov

Názov	Fyzický počítačový klaster[2]	Virtuálny klaster
Interval dokončenia úlohy (celkový čas + CPU čas výpočtu)	66:58 minút	42:38 minút

Tabuľka 6.10. Namerané hodnoty pre množinu 25 286 953 grafov

Názov	Fyzický počítačový klaster[2]	Virtuálny klaster
Interval dokončenia úlohy (celkový čas + CPU čas výpočtu)	04:19:09 hodín	04:49:35 hodín

Uvažujme, že najväčší vplyv na znižovanie energetickej náročnosti pri prevádzke výpočtových systémov, z pohľadu aplikácií, má znižovanie času výpočtu jednotlivých úloh. Na základe skúmania úloh s 3,8 miliónmi grafov a 25 miliónmi grafov s dekompozíciami úlohy na rovnako veľké výpočtové podúlohy (pričom je zrejmé, že rôzne architektúry si vyžadujú rôzne dávky úloh) môžeme konštatovať, že zamietame hypotézu *H3 – Dekompozícia cloudovej aplikácie negatívne ovplyvňuje náklady na prevádzku výpočtového systému*.

[2] Výsledky poskytnuté doktorandom Mgr. Adamom Dudášom

7 Vyhodnotenie a pokračovanie výskumu

Záver práce, a táto kapitola, je venovaná vyhodnoteniu nameraných výsledkov pomocou navrhnutých algoritmov a zvolenej metodiky overovania.

V prvých troch kapitolách, tvoriacich hlavnú teoretickú časť práce, sme objasnili aktuálnu situáciu v oblasti počítačových a distribuovaných systémov. Ďalej, to bola aktuálna situácia v oblasti vyrovnávania záťaže systému a metód používaných pri znižovaní energetickej náročnosti cloudov.

V kapitole 4 sme predstavili návrh dvoch nových algoritmov zameraných na znižovanie energetickej náročnosti cloudov. Oba algoritmy vychádzali z rozdielneho prístupu ako dosiahnuť stanovené ciele. Prvý, algoritmus energetickej efektívnosti – dynamické plánovanie, vykonáva neustále migrovanie virtuálnych strojov na základe zoznamu uzlov a ich zaťaženia. Práve zlepšovanie umiestňovania virtuálnych strojov a znižovanie záťaže uzlov spôsobilo v konečnom dôsledku zníženie energetickej náročnosti. Druhým algoritmom je algoritmus energetickej efektívnosti - fuzzy logika. Fuzzy logika je použitá pre detekciu preťaženého uzla, čo má za následok konsolidáciu virtuálnych strojov a jeho uvoľnenie. Vyhodnotenie, či je uzol preťažený, vyplýva z výstupnej premennej fuzzy inferenčného systému a bázy dvadsiatich siedmich pravidiel. Do navrhnutého fuzzy inferenčného systému vstupujú tri hodnoty z aplikovaných metód zeleného počítania.

Dosiahnuté výsledky a namerané časy z experimentálneho overovania v jednotlivých cloudových prostrediach sme opísali v kapitolách 5 a 6. Po získaní výsledkov aplikovaním vybraných optimalizačných metód v simulačnom prostredí CloudSim, vidíme rozdiel v celkovej spotrebe elektrickej energie medzi jednotlivými technikami. Príjemným zistením je, že na vybranej testovanej úlohy získal najlepšie výsledky algoritmus, ktorý sme navrhli. Ide o algoritmus postavený na princípoch fuzzy logiky. Výsledok sme dosiahli pri akceptovateľnom a minimálnom porušení SLA, na základe dobre pripraveného vyhodnocovacieho systému pravidiel v rámci inferenčného systému.

Pri druhom zvolenom cloudovom prostredí, ktorým bol reálny cloudový systém, boli experimenty zamerané na vyhodnotenie hypotéz H1 a H3. Obe hypotézy boli vyvrátené, vychádzajúc z porovnania nameraných časov dokončenia úloh medzi fyzickým počítačovým klastrom a virtuálnym klastrom. Testovacími úlohami boli rôzne veľkosti sady grafov s ofarbovaním hrán.

Experimenty overenia riešenia v simulačnom prostredí CloudSim (pozri kapitolu 5.1) ukázali, že hypotéza H2 je vyvrátená. To znamená, že neplatí, že *zelené počítanie v cloudových systémoch spomaľuje činnosť počítačového cloudu, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme*.

Na základe vykonania experimentálneho overovania (pozri kapitolu 6.4) konštatujeme, že pokiaľ nejde o mimoriadne veľké výpočtové úlohy, môžeme vyvrátiť hypotézu *H1 - Cloudové systémy, na báze virtuálnych strojov, spomaľujú činnosť fyzických výpočtových prostriedkov, čo vedie k zhoršeniu kritérií kvality pri rozvrhovaní úloh v systéme*.

Podľa výsledku skúmania dávkovania úloh v použitých prostrediach konštatujeme záver, že zamietame hypotézu *H3 - Dekompozícia cloudovej aplikácie negatívne ovplyvňuje náklady na prevádzku výpočtového systému.*.

Do budúcnosti navrhujeme rozvoj zlepšenia danej problematiky, a to skúmaním vylepšenia predstavených vlastných algoritmov. Napriek úspešnému splneniu kritérií v simulovanom cloudovom prostredí, by mohli byť získané výsledky zlepšené ešte s menšou mierou porušenia SLA so zachovaním zníženej energetickej náročnosti. V reálnom cloudovom prostredí je priestor na vylepšenie užší, ale predstavuje väčšiu výzvu. Tú by mohol predstavovať zmenený prístup k spracovaniu testovacej úlohy a spôsobu plánovania v takomto prostredí.

Literatúra

[1] J. Yang, C. Liu, Y. Shang, Z. Mao a J. Chen, „Workload predicting-based automatic scaling in service clouds", in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, ser. CLOUD '13*, júl 2013, s. 810-815. DOI: 10.1109/CLOUD.2013.146.

[2] G. Moltó, M. Caballer, a C. de Alfonso, „Automatic memory-based vertical elasticity and oversubscription on cloud platforms", in *Future Generation Computer Systems*, marec 2016, s. 1–10. DOI: 10.1016/j.future.2015.10.002.

[3] Amazon AWS, „*Amazon EC2 Instance Types*", [Online]. Available: <https://aws.amazon.com/ec2/instance-types/>. [Cit. November 2019].

[4] S. A. Tanenbaum a V. M. Steen, *Distributed Operating Systems*, 2nd. New Jersey, USA: Pearson Education, 2007, s. 705, ISBN: 0-13-239227-5.

[5] TOP500.org., „NOVEMBER 2019", [Online]. Available: <https://www.top500.org/lists/2019/11/>. [Cit. December 2019].

[6] TOP500.org., „*Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband"* , [Online]. Available: <https://www.top500.org/system/179397>. [Cit. December 2019].

[7] TOP500.org., „*Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband*" , [Online]. Available: <https://www.top500.org/system/179398>. [Cit. December 2019].

[8] R. Buyya a S. Venugopal, „*A Gentle Introduction to Grid Computing and Technologies*", Computer society of India, [Online]. Available: <http://gridbus.cs.mu.oz.au/~raj/papers/GridIntro-CSI2005.pdf>. [Cit. Marec 2019].

[9] I. Foster, „*What is the Grid? A Three Point Checklist*", Argonne National Laboratory & University of Chicago, [Online]. Available: <http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf>. [Cit. Január 2019].

[10] J. Škrinárová, *Elastický klaster*, Banská Bystrica: Univerzita Mateja Bela, 2016, s. 15, ISBN: 978-80-557-0642-9.

[11] R. Buyya, J. Broberg a A. Goscinski, *Cloud computing, principles and paradigms*, New Yersey: John Wiley & Sons, 2016, s. 664, ISBN: 978-0-470-88799-8.

[12] E. Vesel, *Privátny cloud pre VUJE Trnava*, Banská Bystrica: FPV UMB, diplomová práca, 2014, s. 63.

[13] G. Pfister, *High Performance Mass Storage and Parallel I/O*, IEEE: Press and Wiley Press. 2001, Kapitola 42, An Introduction to the InfiniBand Architecture, s. 617–632, ISBN: 978-0-471-20809-9.

[14] M. Wehner, L. Oliker a J. Shalf, „A real cloud computer“, *IEEE Spectrum*, roč. 46, č. 10, s. 24-29, okt. 2009, ISSN: 0018-9235. DOI: 10.1109/MSPEC.2009.5267992.

[15] V. Šípková, *„Gridové počítanie*“, Enabling Grids for E-sciencE, [Online]. Available: <http://indico.cern.ch/getFile.py/access?contribId=2&resId=0&materialId=slides&confId=42534>. [Cit. Máj 2019].

[16] T. Wood, P. Shenoy, A. Venkataramani a M. Yousif, „Black-box and gray-box strategies for virtual machine migration“, in *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation*, apríl 2007, s. 1-17.

[17] J. Chabarek, J. Sommers, P. Barford, C. Estan a D.T.S. Wright, „Power awareness in network design and routing“, in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, máj 2008, s. 457–465. DOI: 10.1109/INFOCOM.2008.93.

[18] A. Beloglazov, J. Abawajy a R. Buyya, „Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing“, in *Future Generation Computer Systems*, máj 2012, s. 755-768. DOI: 10.1016/j.future.2011.04.017.

[19] A. Beloglazov, „*Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing*“, Department of Computing and Information Systems The University of Melbourne, [Online]. Available: <https://beloglazov.info/thesis.pdf>. [Cit. Apríl 2019].

[20] M. Etinski, J. Corbalan, J. Labarta a M. Valero, „Parallel job scheduling for power constrained HPC systems“, *Parallel Computing*, roč. 38, č. 12, s. 615-630, dec. 2012, ISSN: 0167-8191. DOI: 10.1016/j.parco.2012.08.001.

[21] J. Emeras, S. Varrette, V. Plugaru a P. Bouvry, „Amazon Elastic Compute Cloud (EC2) vs. In-House HPC Platform: A Cost Analysis“, *IEEE Transactions on*

Cloud Computing, roč. 7, č. 2, s. 456-468, jún 2019. DOI: 10.1109/TCC.2016.2628371.

[22] A. Prabhakaran a J. Lakhsmi, "Cost-Benefit Analysis of Public Clouds for Offloading In-House HPC Jobs", in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, júl 2018, s. 57-64. DOI: 10.1109/CLOUD.2018.00015.

[23] S. Palúch a Š. Peško, *Kvantitatívne metódy v logistike*, Žilina: Žilinská Univerzita v Žiline, 2006. s. 185, ISBN: 80-8070-636-0.

[24] M.H. Ferdaus, M. Murshed, R.N. Calheiros a R. Buyya, „Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic“, in *20th European International Conference on Parallel Processing (Europar 2014)*, aug. 2014, s. 306-317. DOI: 10.13140/2.1.3871.5526.

[25] The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, „*CloudSim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services*“, [Online]. Available: <http://www.cloudbus.org/cloudsim/>. [Cit. Február 2019].

[26] M. Daraghmeh, S. Bani Melhem, A. Agarwal, N. Goel a M. Zaman, “Linear and Logistic Regression Based Monitoring for Resource Management in Cloud Networks”, in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, aug. 2018, s. 259–266. DOI: 10.1109/FiCloud.2018.00045.

[27] B. Balis, K. Figela, K. Jopek, M. Malawski a M. Pawlik, „Porting HPC applications to the cloud: A multi-frontal solver case study“, *Journal of Computational Science*, roč. 18, s. 106-116, jan. 2016, ISSN 1877-7503. DOI: 10.1016/j.jocs.2016.09.006.

[28] Amazon AWS, „*AWS ParallelCluster*“, [Online]. Available: <https://aws.amazon.com/hpc/parallelcluster/>. [Cit. September 2019].

[29] M.A. Netto, R.N. Calheiros, E.R. Rodrigues, R.L. Cunha, a R. Buyya, “HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges”, *ACM Computing Surveys (CSUR)*, roč. 51, č. 1, s. 1-29, apr. 2018, ISSN: 0360-0300. DOI: 10.1145/3150224.

[30] S. Salaria, K. Brown, H. Jitsumoto a S. Matsuoka, "Evaluation of HPC-Big Data Applications Using Cloud Platforms", in *2017 17th IEEE/ACM International*

Symposium on Cluster, Cloud and Grid Computing (CCGRID), júl 2017, s. 1053-1061. DOI: 10.1109/CCGRID.2017.143.

[31] M. Bala a Devanand, „Performance evaluation of cloud datacenters using various green computing tactics“, in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, máj 2015, s. 956-961. url: <http://www.sciencedirect.com/science/article/pii/S0167404811001672>.

[32] A. Beloglazov a R. Buyya, „Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints“, *IEEE Transactions on Parallel and Distributed Systems*, roč. 24, č. 7, s. 1366-1379, aug. 2012, ISSN: 1045-9219. DOI: 10.1109/TPDS.2012.240.

[33] E. Vesel, „Private cloud solution for an engineering company“, *Ad Alta: journal of interdisciplinary research*, roč. 5, č. 1, s. 108-110, 2015, ISSN: 1804-7890.

[34] E. Vesel a J. Škrinárová, „Teaching effectiveness of high-performance computing using an online course“, *International education and research journal (IERJ)*: *peer reviewed international journal*, roč. 2, č. 7, s. 96-97, júl 2016, ISSN: 2454-9916.

[35] J. Škrinárová, E. Vesel a A. Dudáš, „Model of education and training strategy for the high performance computing“, in *ICETA 2016*: *14th IEEE international conference on Emerging eLearning technologies and applications*, nov. 2016, s. 315-320. DOI: 10.1109/INFORMATICS.2017.8327282.

[36] K. Vaidyanathan a K. S. Trivedi, „A comprehensive model for software rejuvenation“, *Proceedings of the IEEE Transactions on Dependable and Secure Computing*, roč. 2, č. 2, s. 124–137, apr. 2005, ISSN:1545-5971. DOI: 10.1109/TDSC.2005.15.

[37] M. Xu, A. V. Dastjerdi a R. Buyya, „Energy efficient scheduling of cloud application components with brownout“, *IEEE Transactions on Sustainable Computing*, roč. 1, č. 2, s. 40–53, jan. 2017, ISSN: 1536-1233. DOI: 10.1109/TSUSC.2017.2661339.

[38] C. Qu, R. N. Calheiros a R. A. Buyya, „Reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances, *Journal of Network and Computer Applications*, roč. 65, s. 167–180, apr. 2016, ISSN: 1084-8045. DOI: 10.1016/j.jnca.2016.03.001.

[39] A. Beloglazov a R. Buyya, „*Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers*“, Concurrency and Computation: Practice and experience, [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.224.7146&rep=rep1&type=pdf>. [Cit. August 2019].

[40] Cloudsim Tutorials, „*Cloudlet in Cloudsim Simulation*“, [Online]. Available: <https://www.cloudsimtutorials.online/cloudlet-in-cloudsim-simulation/>. [Cit. Júl 2019].

[41] PLANETLAB An open platform for developing, deploying, and accessing planetary-scales services, „*PlanetLab*“, [Online]. Available: <https://www.planet-lab.org/node/1>. [Cit. Jún 2019].

[42] UK-MAC, „*CloverLeaf*“, [Online]. Available: <https://uk-mac.github.io/CloverLeaf/>. [Cit. Jún 2019].

[43] A. Chaudhari a A. Kapadia, „*Load Balancing Algorithm for Azure Virtualization with Specialized VM*“, International Journal of Innovations in Engineering and Technology (IJIET), [Online]. Available: <https://pdfs.semanticscholar.org/758e/d0a3ea2e4d369d9882342233d75b4c32391c.pdf>. [Cit. September 2019].

[44] P. Martincová, *Rozvrhovanie úloh v gride*, Žilina: Žilinská Univerzita v Žiline, habilitačná práca, 2008.

[45] S. Palúch a Š. Peško, *Kvantitatívne metódy v logistike*, Žilina: Žilinská Univerzita v Žiline, 2006, s. 185, ISBN-80-8070-636-0.

[46] S. K. Moghaddam, „*Anomaly-aware Management of Cloud Computing Resources*“, School of Computing and Information Systemaddam, School of Computing and Information Systems The University of Melbourne, [Online]. Available: <http://www.cloudbus.org/students/SaraKardaniPhDThesis2019.pdf>. [Cit. Október 2019].

[47] B. Sivák a J. Škrinárová, „The Neural Networks in Quantum Computing“, In *Electronic Computers and Informatics ECI 2006: proceedings of the Seventh International Scientific Conference*, Košice: Faculty of Electrical Engineering

and Informatics of the Technical University of Košice, 2006. s. 48-51, ISBN: 80-8073- 598-0.

[48] J. Suchý, J. Škrinárová a P. Trhan, „Neural Network Modelling of Friction in Robot Joints“, in *Proceedings of the 8th International Symposium ISMCR* 98, Praha: ČTU, 1998, s. 157-162, ISBN: 8001018148.

[49] N.R. Herbst, S. Kounev a R. Ruessner, „*Elasticity in cloud computing: what it is, and what it is not*“, 10th International Conference on Autonomic Computing (ICAC '13), [Online]. Available: <https://www.usenix.org/system/files/conference/icac13/icac13_herbst.pdf>. [Cit. Júl 2019].

[50] A. Barak a Z. Drezner, „Gossip-Based Distributed Algorithms for Estimating the Average Load of Scalable Computing Clusters and Grids“, in *Proc. 2004 Int'l Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04)*, NV: Las Vegas, June 2004.

[51] J. Suchý, J. Škrinárová a P. Trhan, „Modelling of Non-Linear Phenomena in Robot Joints with Artificial Neural“, in *Proceedings of 7th International Workshop on Robotics in Alpe-Adria-Danube Region RAAD, 1998*, s. 285-289, ISBN: 8096796275.

[52] J. Janáček, *Matematické programování*, Žilina: Žilinská Univerzita v Žiline, 1999, ISBN: 80-7100-573-8.

[53] D. György, „*The tight bound of first fit decreasing bin-packing algorithm is FFD(I) ≤ 11/9 OPT(I) + 6/9*“, B. Chen, M. Paterson, and G. Zhang (Eds.): ESCAPE 2007, LNCS 4614 (2007), [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.158.4834&rep=rep1&type=pdf> [Cit. Júl 2019].

[54] G. K. Shyam a S. S. Manvi, „Resource allocation in cloud computing using agents“, in *2015 IEEE International Advance Computing Conference (IACC)*, jún 2015, s. 458–463. DOI: 10.1109/IADCC.2015.7154750.

[55] Spec.org, „*SPECpower_ssj2008*“, [Online]. Available: <http://www.spec.org/power_ssj2008/results/res2009q4/power_ssj2008-20091104-00213.html>. [Cit. Jún 2019].

[56] E. Vesel, J. Škrinárová a A. Dudáš, „Comparison of in-house HPC calculation with public cloud computing for parallel algorithm containing recursive functions“, in ICETA 2019: 17th International conference on emerging elearning technologies and applications : Information and communication technologies in learning, nov. 2019, s. 805-809. ISBN 978-1-7281-4967-7.

[57] N. J. Kansal a I. Chana, „Cloud load balancing techniques: A step towards green computing", *International Journal of Computer Science*, roč. 9, č. 1, s. 238-246, jan 2012, ISSN (online): 1694-0814.

[58] S. M. Hashemi a A. K. Bardsiri, „Cloud Computing Vs. Grid Computing “, *ARPN Journal of Systems and Software*, roč. 2, č. 5, s. 188-194, máj 2012, ISSN: 2222-9833.

[59] jFuzzyLogic The most complete fuzzy logic library in Java. The de-facto standard for research and industry applications. [Online]. Available: <http://jfuzzylogic.sourceforge.net/html/index.html>. [Cit. Marec 2020].

[60] IEC 1131 - PROGRAMMABLE CONTROLLERS Part 7 - Fuzzy Control Programming. [Online]. Available: <http://jfuzzylogic.sourceforge.net/html/pdf/iec_1131_7_cd1.pdf>. [Cit. Marec 2020].

[61] Folding@home. [Online]. Available: <https://foldingathome.org/>. [Cit. Apríl 2020].

[62] People Running Folding@Home Accidentally Created The World's Biggest Supercomputer. [Online]. Available: <https://www.sciencealert.com/so-many-people-are-running-folding-home-that-it-s-created-the-world-s-biggest-supercomputer>. [Cit. Apríl 2020].

[63] L.A. Zadeh, „Fuzzy sets”, in *Information and Control*, roč. 8, č. 3, s. 338-353, jún 1965, DOI: 10.1016/S0019-9958(65)90241-X

[64] L.A. Zadeh, „Outline of a New Approach to the Analysis of Complex Systems and Decision Processes”, in *IEEE Transactions on Systems, Man, and Cybernetics*, roč. SMC-3, č. 1, s. 28-44, jan 1973, DOI: 10.1109/TSMC.1973.5408575

[65] E.H.Mamdani a S.Assilian, „An experiment in linguistic synthesis with a fuzzy logic controller“, in *International Journal of Man-Machine Studies*, roč. 7, č. 1, s. 1-13, nov. 1973, DOI: 10.1016/S0020-7373(75)80002-2

[65] I. Iancu, *A Mamdani type fuzzy logic controller*, in Fuzzy Logic Controls, Concepts, Theories and Applications, Dadios, P.E., Ed.; Intech: Rijeka, Croatia, 2012; s. 325–350, ISBN: 978-953-51-0396-7.

[66] House of Graphs - Snarks. [Online]. Available: <https://hog.grinvin.org/Snarks>. [Cit. Marec 2020].

Zoznam publikovaných prác autora

[1] ***Optimization design for parallel coloring of a set of graphs in the High-Performance Computing [print]*** / Dudáš Adam (40%) - Škrinárová Jarmila (40%) - Vesel Eduard (20%). In: IEEE 15th International Scientific Conference on Informatics [print] : proceedings. - 1. vyd. - New York: Institute of Electrical and Electronics Engineers, 2019. - ISBN 978-1-7281-3178-8. - s. 93-99 [print].

[2] ***Comparison of in-house HPC calculation with public cloud computing for parallel algorithm containing recursive functions [electronic]*** / Vesel Eduard (40%) - Škrinárová Jarmila (40%) – Dudáš Adam (20%). In: ICETA 2019 [electronic] : 17th IEEE International conference on emerging elearning technologies and applications : Information and communication technologies in learning. Starý Smokovec, Slovakia. November 21-22, 2019 : proceedings. - 1. vyd. - Denver: Institute of Electrical and Electronics Engineers, 2019. - ISBN 978-1-7281-4967-7. - s. 805-809 [online].

[3] ***Model of education and training strategy for the management of HPC systems [electronic]*** / Škrinárová Jarmila (40%) - Dudáš Adam (40%) - Vesel Eduard (20%). In: Informatics 2017, roč. 14 [print, electronic] : IEEE International Scientific Conference on Informatics. - 1. vyd. - Danvers: Institute of Electrical and Electronics Engineers, 2017. - ISBN 978-1-5386-0888-3. - s. 400-405 [print].

[4] ***Model of education and training strategy for the high performance computing*** / Škrinárová Jarmila (50%) - Vesel Eduard (50%). In: ICETA 2016 [elektronický zdroj] : 14th IEEE international conference on Emerging eLearning technologies and applications : November 24-25, 2016, Starý Smokovec, The High Tatras, Slovakia. - [S.l.]: IEEE, 2016. - ISBN 978-1-5090-4699-7. - CD-ROM, s. 315-320.

[5] ***Teaching effectiveness of high-performance computing using an online course*** / Vesel Eduard (90%) - Škrinárová Jarmila (10%). In: International education and research 6ournal (IERJ) [elektronický zdroj] : peer reviewed international journal. - ISSN 2454-9916. - Vol. 2, no. 7 (2016), s. 96-97. [online].

[6] ***Private cloud solution for an engineering company*** / Vesel Eduard (100%). In: Ad Alta : journal of interdisciplinary research. - ISSN 1804-7890. - Vol. 5, iss. 1 (2015), , s. 108-110.