UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

Zelené počítanie

Diplomová práca

82054f6a-1a43-47be-9778-2a10a064a9f7

Študijný program: Aplikovaná informatika

Študijný odbor: 9.2.9 Aplikovaná informatika

Katedra: KIN FPV – Katedra informatiky

Vedúci diplomovej práce: doc. Ing. Škrinárová Jarmila, PhD.

Banská Bystrica, 2017 **Bc. Matúš Jančiar**

Čestné prehlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím uvedených zdrojov.

V Banskej Bystrici, dňa 22.4 2017 .....

Poďakovanie

Touto cestou chcem poďakovať doc. Ing. Jarmile Škrinárovej, PhD. vedúcej mojej diplomovej práce a Alžbete Michalíkovej, RNDr., PhD. za pomoc, čas a inšpiráciu pri písaní práce.

Abstrakt

Jančiar, Matúš: Zelené počítanie.
[Diplomová práca] Bc. Matúš Jančiar. - Univerzita Mateja Bela v Banskej Bystrici. Fakulta prírodných vied; Katedra informatiky. - Školiteľ: doc. Ing. Jarmila Škrinárová, PhD. Banská Bystrica FPV UMB, 2017, 52 s.

V tejto práci si bližšie priblížime problematiku energeticky efektívnej konsolidácie v cloudovom prostredí. Vysvetľujeme pojem virtuálizácia v cloudovom prostredí. Opisujeme hypervízor a jeho úlohu v cloude. V práci vysvetľujeme princíp fungovania algoritmov na sledovanie záťaže uzla, výber vhodného virtuálneho stroja na migrovanie a následné umiestnenie virtuálneho stroja na iný uzol. V závere použijeme model cloudu a simulačné prostredie Cloudsim na simuláciu výkonu a spotreby datacentra pri použití rôznych detekčných algoritmov a porovnávame výsledky jednotlivých algoritmov s nami navrhnutým Fuzzy algoritmom.

Kľúčové slová: cloud, zelené počítanie, migrácia, Fuzzy, DVFS.

Abstract

Jančiar, Matúš: Green Computing.

[Master Degree thesis] Bc. Matúš Jančiar. - Matej Bel University Banská Bystrica. Faculty of Natural Sciences; Department of computer sciences. - Supervisor: doc. Ing. Jarmila Škrinárová, PhD. Banská Bystrica FPV UMB, 2017, 52 s.

In this thesis, we analyze problem of Energy efficient load balancing in cloud environment. We analyze virtualization in cloud environments, hypervisor and its role in cloud. The thesis explains principles of technologies for power saving and algorithms for treshold control, virtual machine selection and virtual machine placing in terms of migration. The conclusions will use cloud model Cloudsim to simulate workload and energy consumption with using DVFS and different detection algorithms include our Fuzzy algorithm. In the end we compare simulation results.

Keywords: Cloud computing, Green Computing, Migration, Virtualization, Fuzzy, DVFS.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Cloudové počítanie spôsobilo revolúciu v informačnom a komunikačnom priemysle. Umožnilo poskytovať elastický výpočtový výkon na požiadanie na princípe pay-as-you-go. Rozmach cloudových služieb spôsobil vznik veľkých dátových centier po celom svete poskytujúc výpočtový výkon tisícok uzlov.

Na druhej strane sa ukázalo aké veľké náklady stojí také dátové centrum, či už z pohľadu spotreby elektrickej energie alebo prostriedkov na chladenie systémov. V roku 2010 predstavovala spotreba dátových centier približne 1,5% celosvetovej spotreby elektrickej energie. So stálym trendom rastu dopytu po cloudových službách bolo nutné zaviesť pokročilé nástroje spravovania výpočtových zdrojov, s ohľadom na čo najefektívnejšie a energeticky efektívne využívanie výpočtových zdrojov.

Vysokú spotrebu energie v dátových centrách je možné riešiť viacerými spôsobmi. Napríklad zavádzaním efektívnejších prostriedkov chladenia dátových centier alebo zavádzaním energeticky efektívnejších technológii, ktoré pri danom výpočtovom výkone budú mať podstatne nižšiu spotrebu ako súčasné technológie. Ďalším spôsobom je využívanie elasticity dátových centier a aplikácia hypervízorov s podporou zeleného počítania, ktoré budú dynamicky meniť výkon a následne aj spotrebu dátového centra podľa aktuálneho dopytu po výpočtovom výkone s neustálym dodržiavaním kvality poskytovania služieb (QoS).

V tejto práci analyzujeme možnosti ako dynamicky, elasticky a efektívne meniť výkon a spotrebu datacentra. V závere otestujeme niektoré dostupné algoritmy sledovania zaťaženia uzlu s našim algoritmom v modeli cloudu Cloudsim.

1 Cloudové počítanie

Cloudové počítanie nám umožňuje sprístupňovať softvér, potrebný pre našu prácu ako službu alebo prenajímať výpočtový výkon bez toho, aby sme museli fyzicky vlastniť hardvér. Definícia cloudu podľa Rajkumara Buyyu:
„Cloud definujeme ako druh paralelného a distribuovaného systému, ktorý sa skladá z prepojených virtuálnych počítačov. Tieto počítače sú dynamicky zabezpečované a poskytované ako zdroje na základe dohôd medzi poskytovateľom služby a používateľmi. Cloud je chápaný ako zbierka virtuálnych počítačov, ktoré sú prideľované každému používateľovi“.[1]

Cloudové počítanie zahrňuje množstvo technických prostriedkov, ako sú distribuované výpočty, siete, pokročilé gridové systémy, virtuálizácia, softvér, webové služby, atď. Pri jeho fungovaní sa kladie dôraz na odolnosť voči chybám, dostupnosť služieb, flexibilitu a škálovateľnosť. Záťažou v systéme chápeme zaťaženie procesora, obmedzenú kapacitu pamäte a oneskorenie siete.
Vyrovnávanie záťaže v cloudovom systéme je proces, pri ktorom sa prerozdeľujú úlohy medzi aktívne uzly distribuovaného systému za cieľom efektívneho využívania dostupných zdrojov a skrátenia doby výpočtu úloh. [8]

1.1 Hlavné charakteristiky

Hlavé charakteristiky cloudového počítania sú:

- Distribuované systémy: cloud využíva infraštruktúru gridových systémov. Jednotlivé uzly sú pospájané do klastrov a klastre do gridovej infraštruktúry.
- Internetové technológie: cloud využíva pre poskytovanie svojich služieb zákazníkom Web od verzie 2.0.
- Automatická správa systému: cloud pomocou automatickej správy reaguje sám na rôzne zmeny bez nutnosti zásahu správcu. Systém pomocou monitorov kontroluje aktuálny stav a adaptuje sa. To mu umožňuje optimálne alokovať zdroje podľa aktuálnych potrieb, čo zabezpečuje efektívne využívanie systému.

- Virtuálizácia: je jedna z kľúčových technológii pre cloudové počítanie. Pomocou nej sme schopní vytvárať rôzne prostredia na jednom fyzickom systéme. Dané prostredia sa nazývajú virtuálne, pretože simulujú rozhranie podľa potrieb. Systém vytvorí virtuálny stroj s daným operačným systémom a softvérom podľa požiadaviek zákazníka.

- Škálovateľnosť: elasticita cloudového systému umožňuje dynamické prispôsobovanie sa systému aktuálnemu zaťaženiu cloudu. Cloud podľa potreby aktivuje alebo uvedie do hibernácie dodatočné uzly, aby efektívne škáloval dostupný výpočtový výkon s jeho dopytom.

- Merateľnosť: Rovnako ako pri iných komerčných službách je nutné merať, aké výpočtové prostriedky daný používateľ na svoju prácu spotreboval.

- Princíp pay-as-you-go: na základe merateľnosti spotrebovaných služieb zákazník platí iba za spotrebované množstvo prostriedkov. Napríklad výpočtový výkon, disková kapacita alebo množstvo prenesených dát. [8]

Modely poskytovania služieb

Aj keď s príchodom kontajnerov do cloudov sa začína hovoriť o modeli Xaas, čo znamená čokoľvek ako služba. Modely poskytovania služieb sa tradične delia na tri hlavné služby a to:

- SaaS - Softvér ako služba (angl. Software a s a service)

- PaaS - Platforma ako služba (angl. Platfrom as a service)

- IaaS - Infraštruktúra ako služba (angl. Infrastructure as a service)

Softvér ako služba:

Služba je založená na poskytovaní softvéru cez internet. daný softvér nieje potrebné lokálne inštalovať, pretože beží na infraštruktúre poskytovateľa. Pre používatela takto odpadajú náklady s kúpou a správou daného softvéru.

Platforma ako služba

Daná služba poskytuje používateľovi run-time prostredie na požiadanie. používateľ následne môže vyvíjať aplikácie, ktoré bežia na infraštruktúre cloudu.

Infraštruktúra ako služba

Cloud umožňuje používateľovi vytvoriť kompletnú infraštruktúru na požiadanie pričom ponúka vysokú škálovatelnosť vytvorenej infraštruktúry.

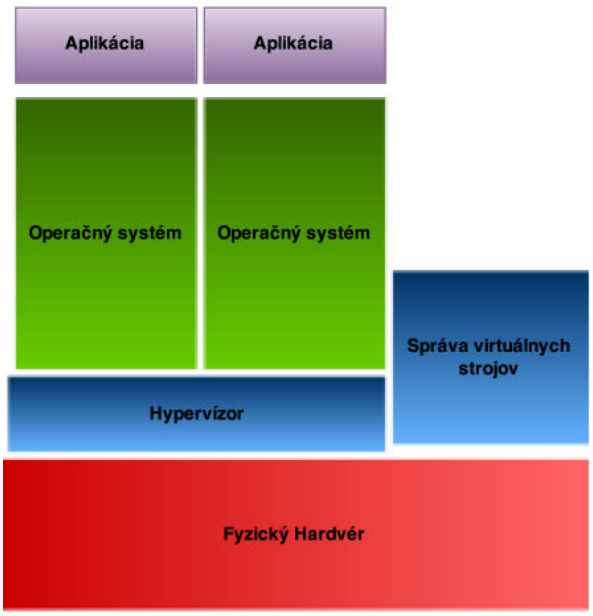
Každá služba ma iné uplatnenie. Softvér ako služba je určená hlavne na poskytovanie služieb koncovým používateľom. Na druhej strane platforma a infraštruktúra ako služba je orientovaná pre firmy a vývojárov. [8]

2 Virtualizácia

Virtualizácia je proces, v ktorom je fyzický hardvér nahradený softvérovou vrstvou. Daný proces je transparentne definovaný, aj keď fyzicky neexistuje. Pomocou virtuálizácie je možné nezávislé a súčasné fungovanie viacerých operačných systémov na jednom hostiteľskom systéme. V cloudových systémoch sa využíva hardvérová virtuálizácia, v ktorej softvérová vrstva nahrádza hardvér pre operačný systém, s ktorým klient pracuje. V súčastnosti sa využívajú dva modely hardvérovej virtuálizácie a to plná virtuálizácia a paravirtuálizácia.[8]

2.1 Plná virtualizácia

Pri tomto modeli virtualizácie obr.1 operačné systémy pracujú pod dohľadom hypervízora. Hypervízor sleduje tok inštrukcii z operačného systému na hardvér a aplikuje binárny preklad. Binárny preklad je proces prekladu inštrukcii. Najskôr sa inštrukcie rozdelia na kritické a nekritické. Kritické inštrukcie sú tie, ktoré ovládajú hardvér, ohrozujú bezpečnosť systému alebo ich nieje možné vykonať v dôsledku virtuálizácie systému. Hypervýzor následne napodobňuje (emuluje) vykonávanie kritických inštrukcii. Nekritické inštrukcie sú preposielané priamo na hardvér. Pri plnej virtuálizácii sa kombinuje binárny preklad s priamym prístupom inštrukcii, pričom operačný systém je oddelený od hardvéru. Plná virtuálizácia umožňuje použitie ktoréhokoľvek operačného systému bez nutnosti jeho úpravy, no z pohľadu výkonu trpí oneskorením a vyššou výpočtovou náročnosťou kvôli binárnemu prekladu.[2]

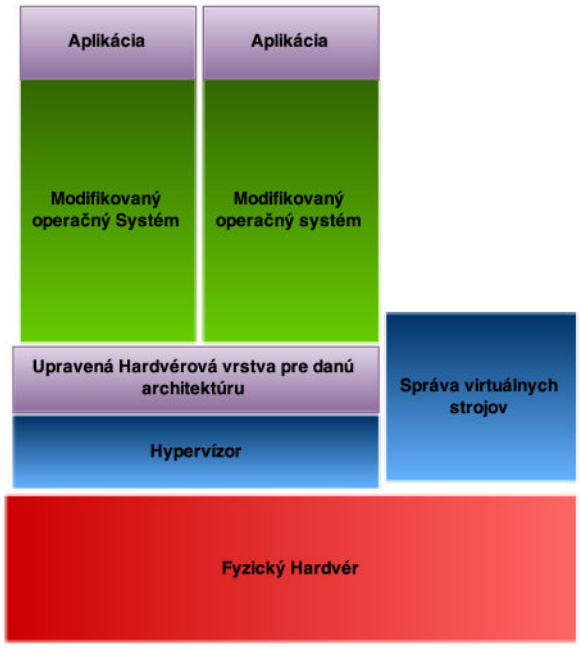


Obr. 1: Model plnej virtuálizácie

2.2 Paravirtualizácia

Model paravirtualizácie obr.2 používa upravené operačné systémy. Úprava spočíva v implementácii inteligentného kompilátora do jadra operačného systému. Kompilátor nahrádza inštrukcie, ktoré sa v dôsledku virtuálizácie nedajú vykonať hypervolaniami. Hypervolanie je žiadosť aplikácie o službu, ktorú poskytuje hypervízor. Pri paravirtuálizácii sa výkon virtuálneho stroja blíži výkonu systému fyzicky nainštalovanom na stroji. Danou úpravou jadra

je možné virtualizovať architektúry, ktoré nepodporujú plnú virtuálizáciu. Nevýhoda paravirtuálizácie je nutnosť používania upravených operačných systémov.[3]



Obr. 2: Model paravirtuálizácie

3 Fuzzy logika

V nasledujúcej kapitole si priblížime fuzzy logiku a jej využitie, pretože v ďalších kapitolách budeme rozoberať náš algoritmus na detekciu preťaženého uzlu postavenom na báze fuzzy logiky.

Fuzzy logika je matematická disciplína odvodená z teórie fuzzy množín, v ktorej sa logické výroky ohodnocujú stupňom príslušnosti (vágnosti) alebo stupňom zapálenia daného pravidla, ktorého hodnoty sú v intervale od 0 do 1. V klasickej výrokovej a predikátovej logike, sa výroky ohodnocujú buď ako pravdivé, alebo nepravdivé. V binárnom tvare ako 0 alebo 1. Fuzzy logika je vhodnejšia pre množstvo reálnych rozhodovacích úloh. Funkcia príslušnosti vo fuzzy logike umožňuje priradiť príslušnosť k množinám v rozmedzí od 0 do 1, vrátane oboch hraničných hodnôt. Fuzzy logika tak umožňuje matematicky vyjadriť pojmy ako „trochu“, „dosť“ alebo „veľa“. Presnejšie, umožňuje vyjadriť čiastočnú príslušnosť k množine.

Stupeň príslušnosti je často zamieňaný s pravdepodobnosťou. Tieto pojmy sú ale rozdielne.

Fuzzy hodnota je priradená funkcii príslušnosti k vágne definovaným množinám a nepredstavuje pravdepodobnosť nejakého javu. Príkladom môže byť napríklad 30 ml vody v 100 mililitrom pohári spolu s dvoma fuzzy množinami: Plná a Prázdna. Čiastočne naplnený pohár potom pripadá z 0.7 k Prázdnej a z 0.3 k Plnej. [12]

Fuzzy Premenné

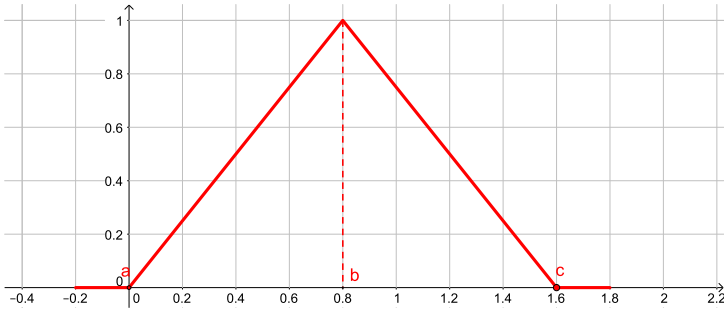
Na rozdiel od klasickej matematiky, kde sa väčšinou pri výpočtoch používajú numerické hodnoty, fuzzy logika využíva lingvistické premenné na opis pravidiel alebo deju.

Typy funkcii príslušnosti:

Funkcie príslušnosti fuzzy veličín môžu mať akýkoľvek tvar. My si priblížime najčastejšie používané.

Používané funkcie príslušnosti:

- Trojuholníková funkcia príslušnosti. Daná funkcia je opísaná pomocou troch parametrov a,b,c.

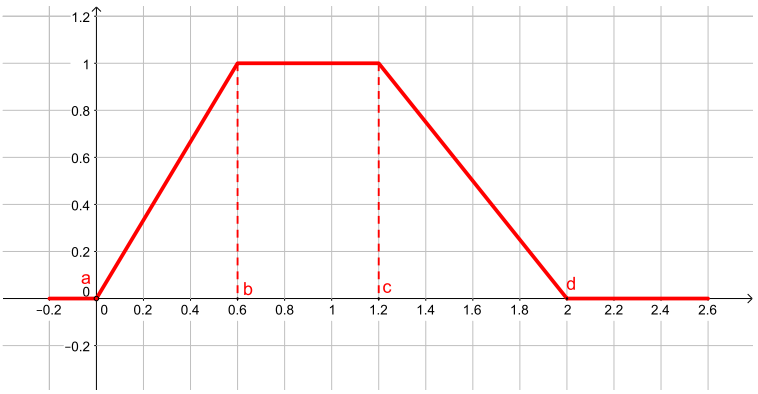


Obr. 3: Príklad trojuholníkovej funkcie

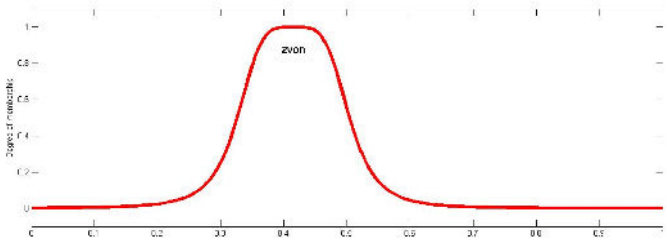
- Lichobežníková funkcia príslušnosti. Je zadaná pomocou štyroch parametrov a,b,c,d.

  Trojuholníkové a lichobežníkové funkcie su miestami až príliš hrubým zjednodušením a nemusia úplne zodpovedať skutočnému významu hodnôt jazykových premenných.

- Zvonové funkcie príslušností. Sú vytvorené pomocou kvadratických funkcii. Sú určené pomocou parametrov a vždy existuje interval, na ktorom nadobúdajú nulovú hodnotu.



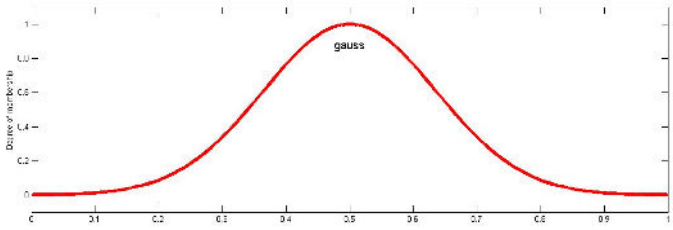
Obr. 4: Príklad lichobežníkovej funkcie



Obr. 5: Príklad zvonovej funkcie

- Gaussova funkcia príslušnosti.

  Gaussova funkcia príslušnosti vychádza zo štatistických metód a opisuje dáta pomocou Gaussovej krivky s dvomi parametrami.



Obr. 6: Príklad Gaussovej funkcie

Základné pojmy Fuzzy množín

- Universum, je základný priestor všetkých prvkov, s ktorými pracujeme.

  Platí: $\forall x \in X : \mu_X(x) = 1$

- Prázdna množina, je fuzzy podmnožina univerza X kde platí, že všetky prvky univerza X majú stupeň príslušnosti do danej fuzzy podmnožiny rovný nule. Predpis prázdnej

podmnožiny.
Platí: $\forall x \in X : \mu_{\varnothing}(X) = 0$

- Rovnosť fuzzy množín, A,B. Nastáva, keď sa rovnajú ich funkcie príslušnosti vo všetkých bodoch univerza X.
  Predpis: $A = B \Leftrightarrow \forall x \in X : \mu_A(x) = \mu_B(x)$

- Množina A je podmnožinou fuzzy množiny B. Ak hodnoty funkcie príslušnosti množiny A sú menšie, nanajvýš rovné hodnotám množiny B.
  Predpis: $A \subseteq B \Leftrightarrow \forall x \in X : \mu_A(x) \leq \mu_B(x)$

- Nosič, je množina všetkých prvkov univerza X, kde stupeň príslušnosti do danej fuzzy množiny je väčší ako nula.
  Predpis: $Supp(A) = \{x \in X; \mu_A(x) \geq 0\}$

- Výška označuje najmenšie z horných ohraničení fuzzy množiny. Táto funkcia je odvodená z matematickej funkcie suprémum.
  Predpis: $hgt(A) = sup\mu_A(x); x \in X$

- Jadro, fuzzy množiny A je množina všetkých bodov univerza X, kde stupeň príslušnosti danej fuzzy množiny je rovný jednej.
  Predpis: $Ker(A = \{x \in X; \mu_A(x) = 1\}$

3.1 Základné štandardné operácie s fuzzy množinami

Ako základné operácie s fuzzy množinami označujeme doplnok, zjednotenie a prienik. Aj keď existuje viacero ďalších operácii, v tejto práci sa im venovať nebudeme. Nasledujúce operácie sa tiež zvyknú nazývať Zadehove po ich tvorcovi L.A Zadehovi.

- Štandardný prienik, nech X je základný priestor a $A, B \in F(X)$. Štandardný prienik fuzzy množín A, B je množina $A \cap B \in F(X)$ s funkciou príslušnosti:
  $\mu_{A \cap B}(x) = min(\mu_A(x, \mu_B(x)$.

- Štandardné zjednotenie, nech X je základný priestor a $A, B \in F(X)$. Potom štandardné zjednotenie fuzzy množín A, B je množina $A \cup B \in F(X)$ s funkciou príslušnosti:
  $\mu_{A \cap B} = max(\mu_A(x), \mu_B(x)$.

- Štandardný doplnok, Nech X je základný priestor a $A \in F(X)$. Štandardný doplnok fuzzy množiny A je množina $\bar{A} \in F(X)$ s funkciou príslušnosti:
  $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$.

Trojuholníkové normy (t-normy)

Trojuholníkové normy sú funkcie, používané na modelovanie konjuncie vo fuzzy logike. Rovnako sú dôležité aj pri zovšeobecnení prieniku fuzzy množín.
T-norma musí spĺňať nasledujúce vlastnosti:

- komutatívnosť: $T(x, y) = T(y, x)$.
- asociatívnosť: $T(x, T(y, z)) = T(T(x, y), z)$.
- neklesajúcosť: $y \leq z \Rightarrow T(x, y) \leq T(x, z)$.
- hraničná podmienka: $T(x, 1) = x$.

Základné t-normy

Nech $T : (x, y) \in \langle 0, 1\rangle^2$. nasledujúce t-normy označujeme ako základné t-normy:

- minimová: $T_M(x, y) = min(x, y)$.
- súčinová: $T_P(x, y) = xy$.
- Lukasiewiczova: $T_L(x, y) = max(0, x + y - 1)$.
- drastický súčin: $\begin{cases} \min(x,y), ak\ max(x, y) = 1 \\ 0, inde \end{cases}$

Základné vlastnosti t-noriem

Pre každú t-normu $T(x, y) \in \langle 0, 1\rangle$ platí nasledovne:

1. $T(1, x) = x$.
2. $T(0, x) = T(x, 0) = 0$.
3. $T(x, y) \leq$ x a zároveň $T(x, y) \leq y$.

4. $T_D(x, y) \leq T(x, y) \leq T_M(x, y)$.

5. $T_D(x, y) \leq T_L(x, y) \leq T_P(x, y) \leq T_M(x, y)$.

Trojuholníkové konormy (t-konormy)

Nech $S : (x, y) \in \langle 0, 1\rangle^2$. nasledujúce t-konormy označujeme ako základné t-konormy:

- komutatívnosť: $S(x, y) = S(y, x)$.
- asociatívnosť: $S(x, S(y, z) = S(S(x, y), z)$.
- neklesajúcosť: $S(x, y) \leq S(x, z)$ ak $y \leq z$.
- hraničná podmienka: $S(x, 0) = x$.

Základné t-konormy

1. maximová t-konorma: $S_M(x, y) = max(x, y)$.
2. súčinová t-konorma: $S_P(x, y) = x + y - xy$.
3. Lukasiewiczova t-konorma: $S_L(x, y) = min(1, x + y)$.
4. drastický súčet: $\begin{cases} \max(x,y), ak\ min(x, y) = 0 \\ 1, inde \end{cases}$

Uvedené T-normy a t-konormy sú si vzájomne duálne. Týmto spôsobom je možné odvodzovať t-konormy z t-noriem, no musia spĺňať základné vlastnosti (komutatívnosť, asociatívnosť, neklesajúcosť a hraničnú podmienku).

3.2 Fuzzy odvodzovanie

Fuzzy odvodzovanie je proces, pri ktorom získavame informácie zadané vágnymi pojmami. Príklad: „ Ak je zima, oblečiem sa teplo“. Pri fuzzy odvodzovaní sa používajú pravidlá typu AK- Potom (IF-THEN) v tvare: AK[predpoklad] - POTOM[záver]. V prípade nášho príkladu: „ **AK** Teplota je nízka, **POTOM** Oblečenie je teplé“. Slová „Teplota“ a „Oblečenie“

voláme jazykové premenné a ich hodnoty „nízka“ a „teplé“ voláme hodnoty jazykových premenných. Jazykové premenné, ktoré sa nachádzajú v predpoklade, nazývame vstupné jazykové premenné. Jazykové premenné v dôsledku zas nazývame výstupné jazykové premenné. Pomocou operátorov konjunkcie, disjunkcie a negácie následne dokážeme vyskladať aj zložitejšie pravidlá.

Defuzifikácia

Opačným procesom k fuyzzyfikácii je defuzifikácia. Ide o metódy, pomocou ktorých určujeme konkrétnu hodnotu výstupu z danej fuzzy množiny. My si priblížime najčastejčie používané metody.

- LOM (largest of maximum), metóda je založená na určení intervalu, kde fuzzy množina dosahuje maximum a vyberie y-ovú súradnicu bodu, ktorý ako posledný nadobúda danú hodnotu.
- MOM (mean of maximum), metóda je založená na určení intervalu, kde fuzzy množina dosahuje maximum. následne vyberie stredný bod daného intervalu a vráti jeho y-ovú hodnotu.
- SOM (smallest of maximum), metóda určí interval, kde daná fuzzy množina dosahuje maximum a vráti z-ovú hodnotu prvého bodu z daného intervalu.
- Bisector, metóda pracuje s plochou, ktorú ohraničuje fuzzy množina a vráti bod, ktorý rozdelí danú plochu na polovicu.
- Centroid, metóda vráti y-ovú súradnicu ťažiska plochy ohraničenej fuzzy množinou.

Pri výbere defuzzifikačnej metódy je potrebné brať ohľad na to, že každá spomenutá metóda môže vrátiť iný výsledok pre rovnakú fuzzy množinu. Preto výber vhodnej metódy závisí od problému, ktorý riešime.

Základné spôsoby modelovania fuzzy relácii

V súčastnosti sa uplatňujú dva spôsoby modelovania a to:

- pomocou karteziánskeho súčinu, založeného na danej t-norme,
- pomocou logickej implikácie.

Z pohľadu modelovania fuzzy relácii existuje viacero metód ako napríklad Mamdariho metóda, Larsenova metóda alebo Takagi-Sugenova metóda. Pre potreby tejto práce si priblížime Takagi-Sugenovu metódu alebo Takagi-Sugeno fuzzy inferenčný systém.

Takagi-Sugenova metóda

Autormi tejto myšlienky sú T.Takagi, M. Sugeno a G. Kang. Navrhnutá bola v roku 1985. Metóda je vhodná na popis javov, o ktorých vieme, že nie su lineárne, ale je ich možné čiastočne linearizovať. Metóda sa vzužíva v rôznych vedných oblastiach. Na vstupe pracujeme s bázou pravidiel $B = P_1, P_2, ...P_k$, pričom každé pravidlo je v znení:

$$P_j : \mathbf{AK}\ X_1 jeA_{ij} \mathbf{A} X_2 jeA_{2j} \mathbf{A}\ ....X_n jeA_{nj},$$
$$\mathbf{POTOM}\ Y \text{ je } a_{1j}x_1 + a_{2j}x_2 + ...a_{nj}x_n + b_j.$$

Kde $j \in \{1, 2, ..., k\}$ a $x_1, x_2, ..., x_n$ sú ostré pozorovania premenných $X_1, X_2, ..., X_n$.

Vstupno-výstupná funkcia

Vstupno-výstupná funkcia φ aproximuje vzťah medzi veličinami na vstupe a na výstupe. Vstupno-výstupná funkcia nemusí byť nutne len lineárna, preto rozlišujeme rôzne rády regulátorov.

- Regulátor nultého rádu, na výstupe daného regulátora je konštanta. V tomto prípade má pravidlo tvar:

$$P_j :\ \mathbf{AK}\ X_1 jeA_{1j} \mathbf{A} ... X_n jeA_{nj}$$
$$\mathbf{POTOM}\ Y \text{ je } b_j.$$

- Regulátor prvého rádu, daný regulátor má na výstupe lineárnu funkciu. Potom pravidlo má tvar:

$$P_j: \ \textbf{AK}\ X_1\ je\ A_{1j}\textbf{A}...X_n\ je\ A_{nj}$$
$$\textbf{POTOM}\ Y\ \text{je}\ \ a_{1j}x_1 + a_{2j}x_2 + ...a_{nj}x_n + b_j.$$

  Kde $a_{1j}, ..., a_{nj}, b_j$ sú reálne čísla.

- Regulátor druhého rádu, regulátor ma na výstupe polynomickú funkciu najmenej druhého stupňa.

Riadenie a regulácia

Fuzzy sa v regulačných systémoch využíva dvoma spôsobmi:

- Priame fuzzy riadenie, fuzzy algoritmus priamo prijíma hodnoty z riadeného systému a reaguje na ne tak, že posiela priamo riadiace zásahy do systému.
- Nepriame fuzzy riadenie, fuzzy algoritmus síce prijíma hodnoty zo systému, ale spracovanie vstupov a riadiace zásahy robí klasický lineárny regulátor. Fuzzy regulátor zisťuje v akom pracovnom bode sa systém nachádza a prepína medzi viacerými lineárnymi regulátormi. [12]

4 Hypervízor

Základným prvkom hardvérovej virtuálizácie je hypervízor (angl. virtual machine manager VMM). Hypervízor vytvára a spravuje prostredie, v ktorom sa inštalujú klientske operačne systémy vo forme virtuálnych strojov. Virtuálne stroje pracujú súčasne a nezávisle od seba. V súčasnosti sa používajú dva typy hypervízorov:

- Prirodzený hypervízor: Tiež nazývaný ako hypervízor prvého typu. Hypervízor pracuje priamo nad fyzickým hardvérom a nahrádza operačný systém. Jeho výhodou je priama komunikácia s hardvérom. Fyzické zdroje sú následne rozdeľované medzi virtuálne stroje.

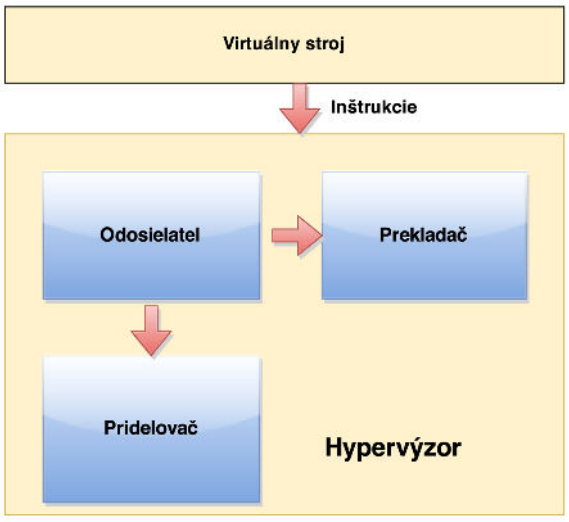
- Hosťovaný hypervízor: Tiež nazývaný ako hypervízor druhého typu. Tento hypervízor k poskytovaniu virtualizačných služieb potrebuje podporu operačného systému. Programy su spracovávané operačným systémom, ktorý komunikuje s virtuálnym hardvérom klientského operačného systému.

Hypervízor sa vnútorne skladá z troch modulov:

- odosielateľ (angl. dispatcher),
- pridelovač (angl. allocator),
- prekladač (angl. interpreter),

Odosielateľ predstavuje vstup systému. Inštrukcie vydané virtuálnymi strojmi príjme a presmeruje do pridelovača a prekladača. Pridelovač dozerá na to, aké prostriedky budú použité, keď sa niektorý virtuálny stroj pokúsi spustiť inštrukcie, ktoré by mohli alebo potrebovali zmeniť aktuálne zdroje virtuálneho stroja.

Hypervízor spolu s hardvérom hostiteľských strojov plne realizuje hardvérovú virtualizáciu. Klientské operačné systémy sú transparentne a nezávisle spúšťané nad hypervízorom bez vedomia, že by boli virtualizované. Hypervízor musí pri svojej práci spĺňať nasledujúce vlastnosti:



Obr. 7: Vnútorná štruktúra hypervízora

1. Rovnocennosť: klientovi pracujúcom pod hypervízorovým dohľadom sa má pracovať rovnako ako by pracoval priamo na fyzickom stroji.
2. Kontrola zdrojov: hypervízor ma plnú kontrolu nad virtuálnymi strojmi.
3. Efektívnosť: väčšina inštrukcii virtuálnych strojov je vykonaná bez zásahu hypervízora.[8]

V nasledujúcej kapitole si priblížime dva voľne dostupné hypervízory KVM a XEN.

Hypervízor XEN

XEN je voľne dostupný hypervízor, využívaný v linuxových a unixových distribúciach. Využíva sa v cloudoch ako napríklad Amazon EC2 alebo IBM Soft Layer. Ide o prirodzený hypervízor, Xen umožňuje používateľom dynamicky vytvárať inštancie operačných systémov vo forme virtuálnych strojov a ľubovolne na nich pracovať. Pri vývoji Xenu sa vývojári rozhodli oddeliť bezpečnostné politiky od ostatných mechanizmov. To znamená, že pokiaľ to nieje krajne nutné, tak kontrolné a autorizačné mechanizmy nezasahujú do vnútorných procesov systému ako napríklad jednotlivé procesory sú zdielané medzi používateľmi alebo ktoré pakety je možne odoslať resp. prijať. Hypervízor poskytuje len základné kontrolné operácie a na bezpečnosť používa iné systémy. Týmto sa dosiahol takmer identický výkon Xenu so základnými linuxovými distribúciami.[4]

Hypervízor KVM

KVM alebo Kernel based Virtual Machine patri medzi hosťované hypervízory. Na svoje fungovanie potrebuje procesory s podporou virtuálizácie. V dobe keď vyšli na trh procesory s podporou virtuálizácie, už nebol potrebný binárny preklad. Práve vďaka tomu je KVM výkonovo schopný konkurovať prirodzeným hypervízorom. V dnešných linuxových distribúciach je KVM súčasťou jadra ako sada nástrojov pre virtuálizáciu.[13]

5 Dynamická správa výkonu

Dynamická správa výkonu (angl. Dynamic Power management, DPM) je súbor technológii, ktoré umožňujú upravovať spotrebu uzla dynamickou zmenou napätia na jadre procesora a zmenou taktovacej frekvencie, alebo jeho uvedením do hibernácie. Správu výkonu sa delí na dve úrovne, kde sa uplatňujú rôzne technológie. Na softvérovej úrovni využíva technológiu dynamického taktovania (angl. Dynamic Voltage and Frequency Scaling, DVFS) a na hardvérovej úrovni zas technológiu dynamického deaktivovania komponentov (angl. Dynamic Component Deactivation, DCD).

- DVFS (angl. Dynamic Voltage and Frequency scaling) je technológia umožňujúca meniť napätie a frekvenciu procesora a tým dynamicky meniť jeho výkon a spotrebu podľa aktuálnych požiadaviek. Treba dbať na to, že použitie tejto technológie na systém vyžaduje komplexnejší pohľad na prácu celého systému. Zmena napätia na jadre procesora môže priniesť rôzny efekt vzhľadom na rôznorodosť architektúr a technológii výroby. Na druhej strane môže zmena frekvencie ovplyvniť rozvrhovanie úloh, pretože mnohé heuristiky berú pri plánovaní do úvahy aktuálny výkon procesora. Rovnako môže zmena taktu na procesore zmeniť dobu výpočtu práve bežiacej úlohy a tým ovplyvniť celý rozvrh.

- DCD (angl. Dynamic Component Deactivation) je technológia umožňujúca dynamické deaktivovanie komponentov. Systém pri nízkej vyťaženosti uvedie nepotrebné periférie do hibernácie a znížiť tak spotrebu a v prípade potreby ich dokáže opätovne aktivovať vo veľmi krátkom čase. Metodiky sledovania komponentov môžeme rozdeliť na prediktívne a stochastické. Prediktívne metodiky sú založené na sledovaní aktuálneho zaťaženia a odhadovaní nasledujúceho vývoja. Efektivita tejto metodiky je veľmi závislá od presného odhadu medzi príchodzími a nasledujúcimi udalosťami. Zlá predikcia vedie k strate výkonu a aj k vyššej spotrebe. [7]

5.1 Energeticky efektívna distribúcia virtuálnych strojov

Problém ako čo najefektívnejšie prenajímať výpočtový výkon sa dá riešiť pomocou efektívnej distribúcie virtuálnych strojov. To znamená alokácie a migrácie virtuálnych strojov na aktívne uzly a uvedenie neaktívnych uzlov do hybernácie. Množstvo aktívnych uzlov sa bude dynamicky upravovať podľa aktuálnych potrieb. Pri danej správe virtuálnych strojov je potrebné riešiť nasledujúce problémy:

1. Identifikovať neefektívne využívaný uzol. Uviesť ho do hibernácie a odmigrovať virtuálne stroje na iné aktívne datacentrum.
2. Rozhodnúť či je uzol preťažený a odmigrovať vybrané virtuálne stroje na iné uzly za dodržania Poskytovania kvality služieb (QoS).
3. Rozhodnúť či je uzol neefektívne využívaný a odmigrovať virtuálne stroje na iné uzly za dodržania poskytovania kvality služieb, a zvýšenia efektivity dátového centra.
4. V prípade migrácie virtuálneho stroja z uzlu. Rozhodnúť, ktorý virtuálny stroj bude odmigrovaný na iný uzol.

5.2 Problém migrácie virtuálneho stroja

Majme jeden fyzický server, alebo uzol a M virtuálnych strojov alokovaných na uzle. Čas budeme považovať za lineárnu diskrétnu veličinu, kde jeden dielik je jedna sekunda. Výpočtové prostriedky uzlu využívané virtuálnymi strojmi sú charakterizované len výkonom procesorov. Využívanie procesora virtuálnym strojom sa dynamicky mení a môže nastať stav, kedy požiadavky virtuálnych strojov sú väčšie ako dostupný výkon procesora. V danom momente považujeme uzol za preťažený. Preťaženie spôsobuje porušenie dohody o úrovni poskytovaných služieb (angl. service-level agreement, SLA) medzi poskytovateľom cloudovej služby a zákazníkom. Porušenie SLA vedie k penalizácii poskytovateľa cloudovej služby a musí ju znášať na svoj úkor. Vhodnou migráciou virtuálneho stroja na iný uzol je možné vyhnúť sa preťaženiu uzla. Následným uvoľnením prostriedkov uzla sa vyhneme preťaženiu. Problém spočíva v otázke, kedy je pre poskytovateľa služby vhodnejšie využiť migráciu a kedy znášať preťažený uzol, pretože migrácia rovnako vyžaduje výpočtové prostriedky a trvá čas T_{mig}.

Migrácia je vhodnejšia keď jej náklady sú nižšie ako penalizácia spôsobená preťažením a teda platí vzťah (1).

$$C_n + C_{mig} * T_{mig} < C_n + C_p * T_p \tag{1}$$

Kde C_n je cena za poskytnutý výpočtový výkon, C_{mig} je cena za prostriedky vynaložené na migráciu, C_p penalizácia spôsobená preťažením a T_p je doba, kedy bol uzol preťažený. [7]

5.3 Meranie porušovania SLA

Dodržiavanie kvality služieb (QoS) v cloudovom prostredí je nutnosť. Keďže definícia kvality služieb vychádza z dohody o úrovni poskytovaných služieb (SLA) je nutné definovať, čo rozumieme pod porušením SLA a ako bude merané. Dohoda o úrovni poskytovaných služieb je splnená ak 100% výpočtového výkonu, vyžiadaného aplikáciami vo virtuálnom stroji, bude virtuálnemu stroju prístupná v ktoromkoľvek okamihu a ohraničenie výkonu bude maximálny výpočtový výkon alokovaný pre daný virtuálny stroj. Porušenie SLA sa bude merať ako podiel času kedy neboli dostupné potrebné výpočtové prostriedky vo virtuálnom stroji z celkovej doby výpočtu úlohy.

5.4 Algoritmy konsolidácie uzlov

Problém migrácie virtuálneho stroja je zovšeobecnením problematiky konsolidácie virtuálnych strojov. V reálnej situácii máme veľké množstvo uzlov a virtuálnych strojov, ktorých požiadavky sa neustále menia. Musíme sledovať aj iné parametre ako zaťaženie procesorov ako napríklad využívanie pamäte, sieťová komunikácia a mnohé iné. Cieľom je poukázať čo musí algoritmus konsolidácie sledovať a ako sa má rozhodovať, aby bola dosiahnutá čo najvyššia efektivita dátového centa.
My si problematiku konsolidácie rozdelíme na nasledujúce podproblémy:

1. Kedy je uzol využívaný nedostatočne?
2. Kedy je uzol preťažený?
3. Ktorý virtuálny stroj bude odmigrovaný?
4. Kam bude odmigrovaný vybraný virtuálny stroj?

V ďalšej časti si predstavíme niekoľko algoritmov pre dynamickú konsolidáciu virtuálnych strojov založenú na analýze údajov využívania zdrojov virtuálnymi strojmi.

6 Algoritmy na detekciu preťaženia uzlu

Každý aktívny uzol prepočítava pomocou algoritmu detekcie preťaženia svoje vyťaženie. Následne je uzol schopný sledovať, kedy hrozí preťaženie a je nutná migrácia v dôsledku konsolidácie.

Statický prah využívania procesora

Statický prah využívania procesora (angl. Static Theshold) jedná sa o jeden z najjednoduchších detekčných algoritmov. Algoritmus sleduje zaťaženie procesora a porovnáva ho so staticky danou úrovňou, po prekročení ktorej uzol označí za preťažený.[7]

Algoritmus výpočtu mediánu

Algoritmus výpočtu mediánu (angl. Median Absolute Deviation) ide o algoritmus s dynamickým prepočítavaním prahu preťaženia uzla. Algoritmus prepočítava prah preťaženia na základe údajov o využívaní procesorov počas celej doby života virtuálneho stroja. Porovnáva dobu, kedy bol procesor v špičke, s celkovou dobou. Čím častejšie sa na procesore objavujú výkonové špičky, tým je väčšia pravdepodobnosť preťaženia uzla. Táto metodika robí algoritmus odolnejší proti krátkodobým náhodným špičkám.[7]

Algoritmus určovania medzikvartilového rozpätia

Algoritmus určovania medzikvartilového rozpätia (angl. Interquartile Range) jedná sa o algoritmus s dynamickým prepočítavaním prahu preťaženia uzla. Algoritmus je založený na štatistickej metóde merania rozptylu hodnôt medzi prvým a tretím kvartilom rozsahu nameraných hodnôt. Na rozhodovanie využíva údaje o využívaní procesora za daný časový interval.[11]

Algoritmus lokálnej regresie

(angl. Local Regresion) Algoritmus zaznamenáva zaťaženie procesora v danom okamihu a údaje zaznamenáva. Následne sa namerané údaje pokúša aproximovať pomocou polynomickej funkcie prvého rádu a pomocou vypočítanej trendovej krivky sleduje a predpovedá možné preťaženie uzla.[11]

Algoritmus lokálnej robustnej regresie

(angl. Robust local regression) Algoritmus je kombináciou lokálnej regresie a medzikvartálneho rozpätia. Algoritmus funguje rovnako ako algoritmus lokálnej regresie, no pracuje z hodnotami medzi prvým a tretím kvartálom nameraných hodnôt. Táto metodika ho robí odolnejším voči krátkodobým špičkám. [11]

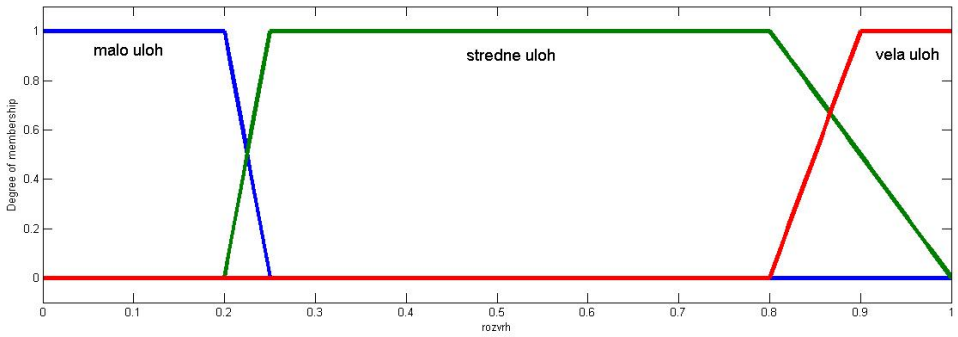
Fuzzy algoritmus

Pre danú problematiku som sa rozhodol navrhnúť vlastný algoritmus na sledovanie zaťaženia uzla využitím fuzzy logiky a Takagi-Sugeno Fuzzy inferenčného systému.
Vstupné veličiny algoritmu sú:

- Premenná **rozvrh**. Ide o Fuzzy veličinu opísanú lichobežníkovými funkciami, ktorá vyjadruje aktuálnu úroveň zaťaženia všetkých procesorov uzla v percentách. obsahuje premenné s parametrami:

Tabuľka 1: Vstupná premenná rozvrh

Názov	typ	a	b	c	d
MALO_uloh	Lichobežníková	-1	0	0,20	0,25
STREDNE_uloh	Lichobežníková	0,2	0,25	0,8	1
VELA_uloh	Lichobežníková	0,8	0,9	1	1,1

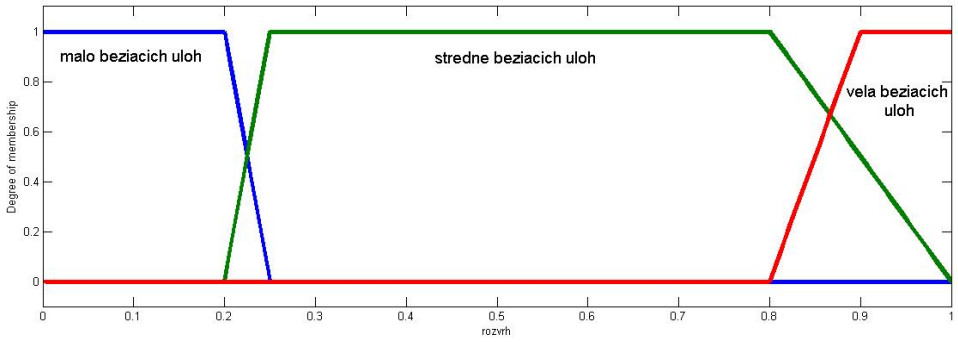


Obr. 8: Vstupná premenná rozvrh

- Premenná **bežiace úlohy** . Ide o fuzzy premennú opísanú lichobežníkovými funkciami, ktorá vyjadruje percentuálnu úroveň využívania priradených prostriedkov všetkých virtuálnych strojov alokovaných na daný uzol.

Tabuľka 2: Vstupná premenná bežiace úlohy

Názov	typ	a	b	c	d
MALO_beziacich_uloh	Lichobežníková	-1	0	0,20	0,25
STREDNE_beziacich_uloh	Lichobežníková	0,2	0,25	0,8	1
VELA_beziacich_uloh	Lichobežníková	0,8	0,9	1	1,1



Obr. 9: Vstupná premenná bežiace úlohy

Na výstupe má algoritmus fuzzy premennú: **vystup**. Ide o o výstupnú premennú fuzzy inferenčného systému nultého rádu, výstupom ktorej je aktuálne zaťaženie uzla.

Inferenčný systém je opísaný pomocou nasledujúcich pravidiel:

Tabuľka 3: Báza pravidiel FIS

Premenná	Hodnota	Premenná	Hodnota	Výstup
Rozvrh	MALO_uloh	Beziace_ulohy	MALO_beziacich_uloh	LOW
Rozvrh	STREDNE_uloh	Beziace_ulohy	MALO_beziacich_uloh	GOOD
Rozvrh	VELA_uloh	Beziace_ulohy	MALO_beziacich_uloh	GOOD
Rozvrh	MALO_uloh	Beziace_ulohy	STREDNE_beziacich_uloh	GOOD
Rozvrh	STREDNE_uloh	Beziace_ulohy	STREDNE_beziacich_uloh	GOOD
Rozvrh	VELA_uloh	Beziace_ulohy	STREDNE_beziacich_uloh	GOOD
Rozvrh	MALO_uloh	Beziace_ulohy	VELA_beziacich_uloh	OVERUSED
Rozvrh	STREDNE_uloh	Beziace_ulohy	VELA_beziacich_uloh	OVERUSED
Rozvrh	VELA_uloh	Beziace_ulohy	VELA_beziacich_uloh	OVERUSED

7 Algoritmy výberu virtuálneho stroja

V tejto kapitole si priblížime základné algoritmy na výber vhodného virtuálneho stroja na odmigrovanie.

Algoritmus minimálnej doby migrácie

Algoritmus je založený na výbere virtuálneho stroja s najkratšou dobou migrácie. Dobu migrácie prepočítava ako podiel využívanej operačnej pamäte a šírky prenosového pásma medzi uzlami.

Algoritmus náhodného výberu

Algoritmus náhodne vyberie virtuálny stroj zo zoznamu alokovaných virtuálnych strojov.

Algoritmus maximálnej korelácie

Algoritmus vyberá zo zoznamu alokovaných virtuálnych strojov ten, ktorého odmigorvanie bude mať najväčší vplyv na výkon uzla. Algoritmus zakladá na predpoklade, že ponechaním virtuálnych strojov s veľkými výpočtovými požiadavkami dôjde v krátkej dobe opätovne k preťaženiu uzla. Preto vyberie virtuálny stroj s najväčšími požiadavkami na výpočtový výkon a ktorého odmigrovanie bude mať najväčší vplyv na výkon uzla. [7]

8 Algoritmy umiestnenia migrujúceho virtuálneho stroja

Pri umiestňovaní migrujúceho virtuálneho stroja musíme brať do úvahy aktuálne voľné prostriedky aktívnych uzlov, možnosť aktivácie hibernujúceho uzla, dobu migrácie virtuálneho stroja, či opätovné preťaženie ďalšieho uzla. Preto môžeme tento problém považovať za NP-zložitý. My si predstavíme algoritmus použitý v simulácii.

Algoritmus najvhodnejšieho umiestnenia

(angl. The Power Aware Best Fit Decreasing). Algoritmus si drží zoznam aktívnych uzlov, ich dostupné výpočtové prostriedky a ich aktuálnu spotrebu. Daný zoznam usporiada podľa aktuálneho zaťaženia a virtuálny stroj alokuje na uzol, kde bude mať migrovaný virtuálny stroj najmenší vplyv na spotrebu daného uzla. Algoritmus je vhodný pre heterogénne dátové centrá a uprednostňuje uzly s najlepším pomerom výkonu a spotreby. [7]

9 Model cloudu a simulačné prostredie Cloudsim

Vývoj, testovanie a implementácia nových algoritmov do cloudovej infraštruktúry je zložitý proces. Preto je lepšie na vývoj použiť vhodné simulačné prostredie skôr, než dôjde k samotnej implementácii do reálneho systému. V súčastnosti existuje niekoľko simulačných prostredí. Nektoré v aktuálnej verzii majú už zahrnutú aj podporu pre vývoj algoritmov a heuristík pre zelené počítanie ako napríklad GreenCloud alebo Cloudsim. V danej práci som sa rozhodol použiť simulačné prostredie Cloudsim vo verzii 4.0. Simulačné prostredie Cloudsim je projekt, ktorý vedie University of Melbourne v Austrálii. Simulátor je napísaný v programovacom jazyku Java. Práca zo simulátorom spočíva v editácii a implementácii zdrojových kódov, ktoré sú voľne prístupné. [9]

10 Simulácia

Pre otestovanie nášho Fuzzy algoritmu a porovnanie jeho výkonu s uvedenými algoritmami využijeme model cloudu Cloudsim s nasledujúcimi nastaveniami.

Tabuľka 4: Nastavenia simulácie

Parameter	Hodnota
Simulovaná doba	24 hod.
Hypervízor	Xen
Počet uzlov	50
Počet virtuálnych strojov	50
Plánovací interval	300
Velkosť cloudletov	náhodne od 2500 B

Tabuľka 5: Nastavenia virtuálnych strojov

typ virtuálneho stroja	Alokovaný výkon (MIPS)	Počet procesorov	Velkosť operačnej pamäte RAM (MB)	Šírka prenosového pásma (MBit/s)	Alokovaná úložná kapacita (MB)
1	2500	1	870	100	2500
2	2000	1	1740	100	2500
3	1000	1	1740	100	2500
4	500	1	613	100	2500

Tabuľka 6: Parametre uzlov

Názov	HP ProLiant ML110 G4	HP ProLiant ML110 G5
Procesor	Intel Xeon 3040	Intel Xeon 3075
Počet jadier	2	2
Takt (MHz)	1860	2660
Operačná pamäť	4096	4096
Šírka prenosového pásma (MBit/s)	1000	1000
Úložná kapacita (MB)	1000	1000

V simulácii budeme sledovať datacentrum o počte 50 uzlov, na ktorých bude pracovať 50 virtuálnych strojov po dobu 24 hodín. V simulácii alokujeme virtuálne stroje 4 rôznych parametrov podľa tabuľky 5 a ako výpočtové uzly sú použité parametre serverov podľa tabuľky 6 o počte 25 uzlov G4 a 25 uzlov G5. Úlohy budú alokováne na virtuálne stroje v podobe cloudletov, ktoré budú mať veľkosť od 2500 B. Priebeh simulácie a výsledky budú uložené do textového súboru.

Merané veličiny

V simulácii budeme merať a porovnávať nasledujúce veličiny:

- Spotreba: množstvo elektrickej energie, spotrebovanej na činnosť datacentra počas doby simulácie.
- Počet migrácii: počet migrácii virtuálnych strojov počas simulácie.
- SLA: Percentuálne vyjadrenie porušenia SLA. Údaj berieme ako podiel času, kedy došlo k porušeniu (z dôvodu preťaženia) z celkovej doby výpočtu úlohy.
- SLA z dôsledku migrácie: vyjadruje percentuálny podiel porušenia SLA z dôvodu prebiehajúcej migrácie z celkového SLA.
- Počet vypnutí uzlov: koľkokrát došlo k uvedeniu uzla do hibernácie.

Výsledky simulácie

V tejto kapitole sú uvedené namerané hodnoty simulácii pre detekčné algoritmy.
Algoritmy s predikciou:

- Výpočet mediánu (MAD). Tabuľka 8.
- Určovanie medzikvartilového rozpätia (IQR). Tabuľka 9.
- Lokálna regresia (LR). Tabuľka 10.
- Lokálna robustná regresia (LRR). Tabuľka 11.

Algoritmy bez predikcie:

- Statický prah preťaženia (THR). Tabuľka 7.
- Fuzzy algoritmus (FIS). Tabuľka 12.

Každý detekčný algoritmus je použitý s nasledujúcimi algoritmami výberu virtuálneho stroja. Algoritmus minimálnej doby migrácie (MMT), algoritmus náhodného výberu (RS) a algoritmus maximálnej korelácie (MC).

Tabuľka 7: Statický prah preťaženia. Namerané hodnoty

THR alg.	MMT	MC	RS
Spotreba (kW)	41,81	40,85	38,02
Počet migrácii	4839	4392	3981
Celková SLA (%)	3,25	3,09	4,45
SLA z dôvodu migrácie (%)	23	27	27
Počet vypnutí uzlov	1424	1389	1286

Tabuľka 8: Výpočet mediánu. Namerané hodnoty

MAD alg.	MMT	MC	RS
Spotreba (kW)	45,61	44,99	44,57
Počet migrácii	5265	4778	4868
Celková SLA (%)	1,31	1,53	1,35
SLA z dôvodu migrácie (%)	23	26	26
Počet vypnutí uzlov	1528	1468	1458

Tabuľka 9: Určovanie medzikvartálneho rozpätia. Namerané hodnoty

IQR alg.	MMT	MC	RS
Spotreba (kW)	47,85	46,86	46,97
Počet migrácii	5502	5085	5132
Celková SLA (%)	1,05	1,13	1,26
SLA z dôvodu migrácie (%)	23	26	26
Počet vypnutí uzlov	1549	1517	1528

Tabuľka 10: Lokálna regresia. Namerané hodnoty

LR alg.	MMT	MC	RS
Spotreba (kW)	35,57	34,35	34,07
Počet migrácii	2872	2203	2148
Celková SLA (%)	3,16	3,17	3,33
SLA z dôvodu migrácie (%)	13	14	14
Počet vypnutí uzlov	806	685	662

Tabuľka 11: Lokálna robustná regresia. Namerané hodnoty

LRR alg.	MMT	MC	RS
Spotreba (kW)	35,37	34,35	34,25
Počet migrácii	2808	2203	2067
Celková SLA (%)	3,16	3,17	3,33
SLA z dôvodu migrácie (%)	13	14	14
Počet vypnutí uzlov	806	685	662

Tabuľka 12: Fuzzy algoritmus. Namerané hodnoty

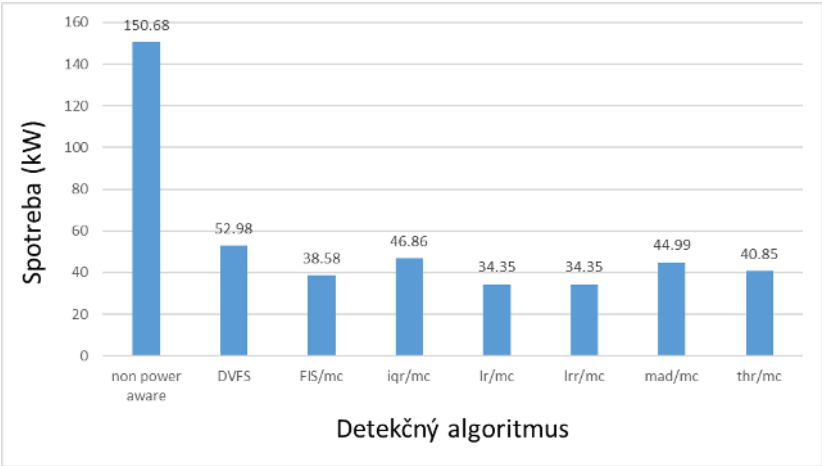
FIS alg.	MMT	MC	RS
Spotreba (kW)	39,07	38,58	38,41
Počet migrácii	4559	4125	4132
Celková SLA (%)	4,5	4,33	4,8
SLA z dôvodu migrácie (%)	24	27	28
Počet vypnutí uzlov	1339	1295	1323

Analýza výsledkov

Merania ukázali vplyv na výkon pri použití rôznych algoritmov. Z pohľadu algoritmov výberu sa ukázalo, že všetky použité algoritmy sa držali na podobnej úrovni. Zaujímavosťou je, že aj algoritmus náhodného výberu preukázal porovnateľné výsledky ako ostatné algoritmy. To nasvedčuje faktu, že v simulácii výber virtuálneho stroja mal len malý vplyv na celkový výkon, hoci v reálnej situácii by sa výsledky mohli líšiť. Z pohľadu detekčných algoritmov sa ukázal rozdiel medzi proaktívnou a reaktívnou politikou rozhodovania v tabuľke 13. Proaktívne algoritmy lokálnej regresie, určovania medzikvartilového rozpätia a výpočtu mediánu dokázali predpovedaním vývoja znížiť celkovú mieru porušovania SLA na úroveň približne jedného percenta pri spotrebe v rozmedzí 34 až 47kW v závislosti od merania.
Simulácia dokázala konkurencieschopnosť nášho Fuzzy algoritmu s ostatnými algoritmami, keď sa nameraná spotreba pohybovala v rozpätí 38,14 až 39,07 kW pri úrovni porušenia SLA 4,33%.

Tabuľka 13: Porovnanie jednotlivých detekčných algoritmov

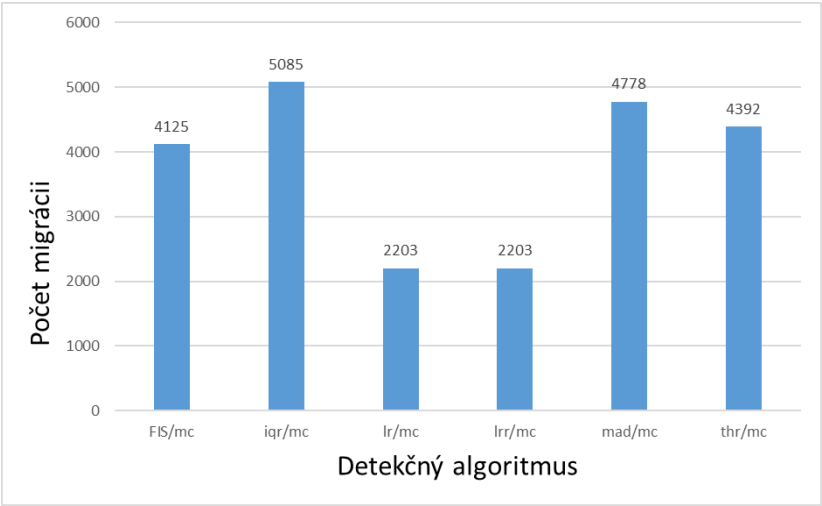
	Bez predikcie		S predikciou					
	THR	FIS	IQR	LR	LRR	MAD	DVFS	Bez úpravy (non Power aware)
Spotreba (kW)	40,85	38,58	46,86	34,35	34,35	44,99	52,98	150,68
Počet migrácii	4392	4125	5085	2203	2203	4778	0	0
Celková SLA (%)	3,09	4,33	1,13	3,17	3,17	1,53	0	0
SLA z dôvodu migrácie (%)	27	27	26	14	14	26	0	0
Počet vypnutí uzlov	1389	1295	1517	685	685	1468	29	0



Obr. 10: Graf nameranej spotreby

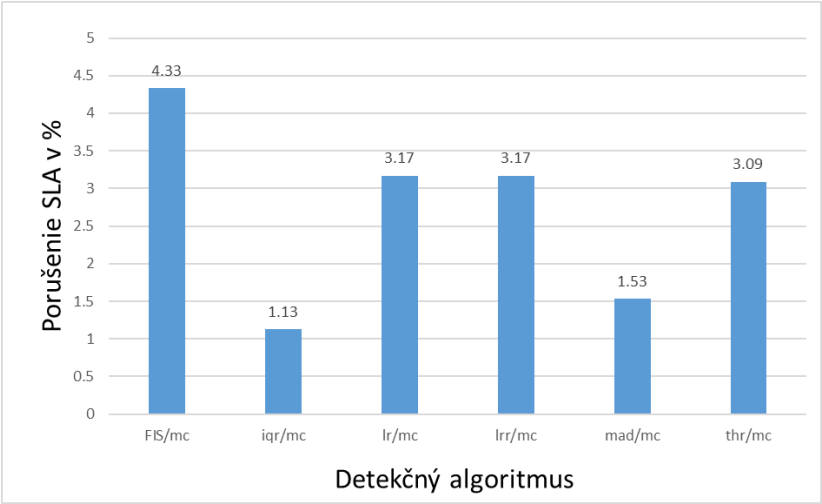
Na obrázku 10 môžeme vidieť vplyv jednotlivých heuristík na spotrebu. Datacentrum bez použitia akejkoľvek konsolidácie malo za dobu simulácie spotrebu 150,68 kW. Pri použití technológie DVFS klesla spotreba na 52,98 kW. Využitím algoritmov pre zelené počítanie sme sa dostali až na najnižšiu nameranú hodnotu 34,35 kW. Použitím nášho fuzzy algoritmu sa spotreba datacentra dostala na úroveň 38,58 kW.

Graf na obrázku 11 zobrazuje počet migrácii virtuálnych strojov za celú dobu simulácie.



Obr. 11: Graf počtu migrácii

Najmenej migrácii nastalo pri použití algoritmu lokálnej regresie. Simulácia tiež ukázala, že použitie lokálnej robustnej regresie malo rovnaký vplyv na systém. Pravdepodobne pre nízky počet krajných nameraných hodnôt ako sú krátkodobé výkonové špičky v meraniach. Náš fuzzy algoritmus za dobu simulácie previedol celkovo 4125 migrácii, čo sa dá považovať za priemerný výsledok z celkového nameraného rozsahu.



Obr. 12: Graf nameraného porušenia SLA

Celková percentuálna miera porušovania SLA je zobrazená na obrázku 12. Z nameraných hodnôt žiaden algoritmus neprekročil úroveň 5%. Najnižšiu úroveň porušovania SLA sme namerali pri algoritmoch medzikvartilového rozpätia (IQR) a výpočtu mediánu (MAD). Čo dokazuje výhodu použitia proaktívnej metodiky sledovania zaťaženia uzla. Náš fuzzy algoritmus dosiahol úroveň porušovania SLA 4,33%. Fuzzy algoritmus v tomto smere dopláca reaktívnou metodikou sledovania zaťaženia uzla.

11 Nasledujúci vývoj

Simulácia ukázala potenciál využitia fuzzy logiky v algoritmoch pre zelené počítanie. Náš fuzzy algoritmus preukázal dobrý výkon aj z hľadiska výslednej spotreby no rovnako aj v dodržiavaní SLA. Súčasná verzia algoritmu funguje na reaktívnej báze, no simulácie ukázali vhodnosť použitia proaktívnej logiky a teda sledovania vývoja a následného odhadovania vývoja zaťaženia.

Preto by som sa zameral na implementáciu vstupnej premennej, ktorá by bola prognózou vývoja zaťaženia ako je napríklad metóda výpočtu mediánu. Vhodné by bolo upraviť výstupnú premenou Fuzzy inferenčného systému, kde by namiesto konštánt a teda funkcie nultého rádu bol vývoj zaťaženia opísaný funkciu, čo by spresnilo výpočet aktuálneho zaťaženia pri aktuálnom stave uzla. V súčasnej verzii algoritmu sú na opis vstupných premenných použité lichobežníkové funkcie, no je známe že tieto funkcie niesu vhodné na opis niektorých javov, lebo ich môžu skresľovať. Preto by som pouvažoval nad použitím iných funkcii, ktoré by presnejšie opisovali danú vstupnú premennú.

Je vhodné pri budúcom vývoji pouvažovať nad použitím ANFIS systému, čo je Fuzzy inferenčný systém, ktorý využíva neurónovú sieť na učenie resp, úpravu funkcii vstupných premenných. Algoritmus by v počiatočnom stave pracoval rovnako ako súčastná verzia postupom času by neurónova sieť upravovala tvar vstupných premenných pre presnejšie opisovanie aktuálneho stavu uzla.

Záver

V tejto práci som sa zaoberal problematikou energeticky efektívneho využívania cloudových systémov. Analyzoval som cloud, jeho základné časti. Analyzoval som dostupné technológie na energeticky efektívnu správu dátového centra a základné algoritmy pre zelené počítanie. Následne som navrhol vlastný algoritmus na detekciu preťažených uzlov založený na Fuzzy inferenčnom systéme Takagi-Sugeno. V simulácii som použitím modelu cloudu Cloudsim simuloval prácu datacentra použitím opísaných detekčných a selekčných algoritmov a porovnával som ich s mojim Fuzzy algoritmom. Sledoval som spotrebu, počet migrácii, počet hibernácii a mieru porušovania SLA za danú dobu simulácie.

Zoznam bibliografických odkazov

[1] Rajkmuar Buyya a spol. *Mastering Cloud Computing*, 2013 ISBN-10: 0124114547

[2] Kai Hwang a spol. *Cloud computing Virtialization clases*. [online] dostupne na internete `https://technet.microsoft.com/en-us/magazine/hh802393.aspx`, 2017

[3] Gunit Gupta, *Virtualization and its type*. [online] dostupné na internete `http://cloudsim-setup.blogspot.sk/2012/12/virtualization-and-its-type.html`, 2017

[4] Lars Kurth a spol. *Xen Project Software. Overview* [online] dostupné na internete `http://wiki.xen.org/wiki/Xen_Project_Software_Overview`, 2017

[5] Rajkmuar Somani, Jyotsana Ojha *A Hybrid approach for VM Load Balancing in Cloud Using Cloudsim*, IEEE, 2014

[6] Shreenath Acharya, Demian Anthony D'Mello, *A Taxonomy of Live Virtual Machine (VM) Migration Mechanisms in cloud Computing Enviroment*, IEEE 2013

[7] Anton Beloglazov *Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing*. [online] dostupne na internete `http://beloglazov.info/thesis.pdf`, 2017

[8] Matúš Jančiar *Algoritmy vyrovnávania zát'aže v cloudovom prostredí*, 2015

[9] Anupinder Singh *Cloudsim Simulation Framework* [online] dostupné na internete `http://www.superwits.com/library/cloudsim-simulation-framework`, 2017

[10] Patricia Arroba, José L.Ayala, Rajkumar Buyya *Dynamic Voltage and Frequency Scaling-aware dynamic consolidation of virtual machines for energy efficient cloud data centers*, IEEE, 2016

[11] Anton Beloglazov, Rajkumar Buyya *Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation*

of Virtual Machines in Cloud Data Centers [online] dostupné na internene http://beloglazov.info/papers/2012-optimal-algorithms-ccpe.pdf, 2017

[12] Anna kolesárová, Monika Kováčová *FUZZY MNOŽINY a ich aplikácie*, 2004 ISBN 80-227-2036-4

[13] Casper van der Poll *Resource usage in hypervisors* [online] dostupné na internete https://ivi.fnwi.uva.nl/sne/wp/wp-content/uploads/2016/02/scriptie-Casper-van-der-Poll.pdf, 2015

[14] Christine Mayap Kamga *CPU frequency based on DVFS*, IEEE, 2012

[15] Mohammad Alaul Haque Monil, Rashedur M. Rahman *VM consolidation approach based on heuristics, fuzzy logic, and migration control* [online] dostupné na internete https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-016-0059-7, 2016

[16] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, Mihai Alexandru *Energy-aware simulation with DVFS*, ScienceDirect, 2013

[17] Umang Singh, Ayushi Sharma *CloudSim Simulator Used for Load balancing in Cloud Computing*, [online] dostupné na internete http://www.ijetae.com/files/Volume6Issue4/IJETAE_0416_47.pdf, 2016

[18] Nidhi Jain Kansal1, Inderveer Chana *Cloud Load Balancing Techniques : A Step Towards Green Computing*, [online] dostupné na internete http://www.chinacloud.cn/upload/2012-02/12021215192379.pdf, 2012

[19] Juan Rada-Vilela *Fuzzylite The FuzzyLite Libraries for Fuzzy Logic Control*, [online] dostupné na internete http://www.fuzzylite.com/java/, 2017

Zoznam príloh

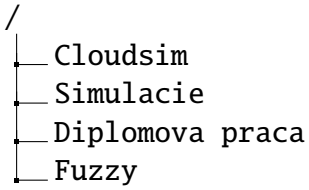
Príloha A: Sprievodné CD so zdrojovým kódom

Príloha B: Používateľská príručka

Príloha C: Systémová príručka

Príloha A

Sprievodné CD so zdrojovým kódom



UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

Zelené počítanie

Používateľská príručka

Banská Bystrica, 2017

Bc. Matúš Jančiar

Obsah

1 Funkcia programu

Cloudsim je model cloudu a simulačné prostredie určené na vývoj a testovanie algoritmov určených pre cloudové prostredie. Vďaka veľkému počtu nastaviteľných parametrov v simulácii a zeleného počítania je vhodný na simuláciu nášho algoritmu. Cloudsim neobsahuje grafické rozhranie a všetky nastavenia sa vykonávajú formou úpravy zdrojových kódov. Preto každá simulácia vykonaná v praktickej časti bude prístupná ako samostatný jar súbor, ktorý vytvorí textový súbor s obsahom celej simulácie.

2 Súpis obsahu média

Súčasťou priloženého média je:

- Zdrojový kód simulačného prostredia Cloudsim 4.0.
- Zdrojový kód fuzzy algoritmu.
- Spustiteľné jar. súbory jednotlivých simulácii vykonaných v simulátore.
- Používateľská príručka v elektronickej podobe.

3 Požiadavky na technické prostriedky

Program nemá žiadne špeciálne požiadavky na technické prostriedky. Na spustenie je potrebné mať nainštalované Java Runtime Enviroment (JRE 8.0) a novšie.

4 Opakovanie simulácie

každá simulácia je prístupná na zopakovanie ako jar. súbor. Po spustení súboru sa vytvorí v adresári priečinok s textovým súborom obsahujúci informácie z celého priebehu simulácie.

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

Zelené počítanie

Systémová príručka

Banská Bystrica, 2017 **Bc. Matúš Jančiar**

Obsah

1 Funkcia programu

Fuzzy algoritmus je navrhnutý na sledovanie zaťaženia uzla v cloudovom systéme. Na základe aktuálneho vyťaženia procesorov a miery využívania alokovaných prostriedkov prepočítava stupeň zaťaženia uzla. ak zaťaženie presiahne stanovenú mieru, systém spustí proces migrácie virtuálneho stroja z daného uzla.

2 Analýza riešenia

Algoritmus je napísaný v programovacom jazyku Java. Pre prácu s fuzzy logikou používa knižnicu Fuzzylite vo verzii 6.0. Súčasná verzia algoritmu je navrhnutá pre použitie v simulátore cloudu Cloudsim vo verzii 4.0. pre implementáciu do iného systému je nutná úprava algoritmu.

3 Požiadavky na technické prostriedky

Program nemá žiadne špeciálne požiadavky na technické prostriedky. Na spustenie je potrebné mať nainštalované Java Runtime Enviroment (JRE 8.0) a novšie. Všetky potrebné knižnice k práci zo simulátorom su priložené v priečinku simulátora Cloudsim.

4 Popis Algoritmu

4.1 Zdrojový kód

```
1  package org.cloudbus.cloudsim.power;
2
3  import java.util.List;
4
5  import org.cloudbus.cloudsim.Host;
6  import org.cloudbus.cloudsim.Vm;
7
8  import com.fuzzylite.*;
9  import com.fuzzylite.activation.*;
10 import com.fuzzylite.defuzzifier.*;
11 import com.fuzzylite.factory.*;
12 import com.fuzzylite.hedge.*;
13 import com.fuzzylite.imex.*;
14 import com.fuzzylite.norm.*;
15 import com.fuzzylite.norm.s.*;
16 import com.fuzzylite.norm.t.*;
17 import com.fuzzylite.rule.*;
18 import com.fuzzylite.term.*;
19 import com.fuzzylite.variable.*;
20
21 /**
22 * A VM allocation policy that uses a Fuzzy CPU utilization
      Threshold (FIS) to detect host over
23 * utilization.
24 *
25 *
26 * @author Matus Janciar
27 */
28 public class PowerVmAllocationPolicyMigrationStaticThreshold_FIS
      extends PowerVmAllocationPolicyMigrationAbstract {
29
30         /** The static host CPU utilization threshold to detect over
               utilization.
```

```
31          * It is a percentage value from 0 to 1
32          * that can be changed when creating an instance of the
                class. */
33          private double utilizationThreshold = 0.9;
34
35          /**
36          * Instantiates a new
                PowerVmAllocationPolicyMigrationStaticThreshold.
37          *
38          * @param hostList the host list
39          * @param vmSelectionPolicy the vm selection policy
40          * @param utilizationThreshold the utilization threshold
41          */
42          public PowerVmAllocationPolicyMigrationStaticThreshold_FIS(
43          List<? extends Host> hostList,
44          PowerVmSelectionPolicy vmSelectionPolicy,
45          double utilizationThreshold) {
46                  super(hostList, vmSelectionPolicy);
47                  setUtilizationThreshold(utilizationThreshold);
48          }
49
50          /**
51          * Checks if a host is over utilized, based on CPU usage.
52          *
53          * @param host the host
54          * @return true, if the host is over utilized; false otherwise
55          */
56          @Override
57          protected boolean isHostOverUtilized(PowerHost host) {
58                  addHistoryEntry(host, getUtilizationThreshold());
59                  double totalRequestedMips = 0;
60                  double maxRequestedMips = 0;
61                  for (Vm vm : host.getVmList()) {
62                          totalRequestedMips +=
                                vm.getCurrentRequestedTotalMips();
63                          maxRequestedMips +=
                                vm.getCurrentRequestedMaxMips();
64                  }
```

```
65                  double utilization = totalRequestedMips /
                        host.getTotalMips();
66                  double zatazenie =((host.getTotalMips()-
                        host.getMaxAvailableMips())/host.getTotalMips())
                        ;
67                  double vytazenieVM=
                        (totalRequestedMips/maxRequestedMips);
68                  //      return utilization >
                        getUtilizationThreshold();
69                  /*FIS engine ********************************/
70                  Engine engine = new Engine();
71                  engine.setName("FISbroker");
72                  engine.setDescription("FIS pre broker");
73
74                  InputVariable inputVariable = new InputVariable();
75                  inputVariable.setName("Rozvrh");
76                  inputVariable.setDescription("");
77                  inputVariable.setEnabled(true);
78                  inputVariable.setRange(0, 1);
79                  inputVariable.setLockValueInRange(false);
80                  inputVariable.addTerm(new Trapezoid("MALO_uloh",-1,
                        0.000,0.2 ,0.25 ));
81                  inputVariable.addTerm(new
                        Trapezoid("STREDNE_uloh",0.2 , 0.25,0.8 ,1 ));
82                  inputVariable.addTerm(new
                        Trapezoid("VELA_uloh",0.8,0.9 ,1 ,1.1));
83                  engine.addInputVariable(inputVariable);
84
85                  InputVariable inputVariable2 = new InputVariable();
86                  inputVariable2.setName("Beziace_ulohy");
87                  inputVariable2.setDescription("");
88                  inputVariable2.setEnabled(true);
89                  inputVariable2.setRange(0, 1);
90                  inputVariable2.setLockValueInRange(false);
91                  inputVariable2.addTerm(new
                        Trapezoid("MALO_beziacich_uloh",-1, 0.000,0.2
                        ,0.25 ));
92                  inputVariable2.addTerm(new
                        Trapezoid("STREDNE_beziacich_uloh",0.2 ,
```

```
                0.25,0.8 ,1 ));
93              inputVariable2.addTerm(new
                Trapezoid("VELA_beziacich_uloh",0.8,0.9 ,1
                ,1.1));
94              engine.addInputVariable(inputVariable2);
95
96              OutputVariable outputVariable = new OutputVariable();
97              outputVariable.setName("Vystup");
98              outputVariable.setDescription("");
99              outputVariable.setEnabled(true);
100             outputVariable.setRange(0, 1);
101             outputVariable.setLockValueInRange(false);
102             outputVariable.setAggregation(null);
103             outputVariable.setDefuzzifier(new
                WeightedAverage("TakagiSugeno"));
104             outputVariable.setDefaultValue(Double.NaN);
105             outputVariable.setLockPreviousValue(false);
106             outputVariable.addTerm(new Constant("LOW", 0));
107             outputVariable.addTerm(new Constant("GOOD", 0.5));
108             outputVariable.addTerm(new Constant("OVERUSED", 1));
109             engine.addOutputVariable(outputVariable);
110
111             RuleBlock ruleBlock = new RuleBlock();
112             ruleBlock.setName("");
113             ruleBlock.setDescription("");
114             ruleBlock.setEnabled(true);
115             ruleBlock.setConjunction(new Minimum());
116             ruleBlock.setDisjunction(new Maximum());
117             ruleBlock.setImplication(new Minimum());
118             ruleBlock.setActivation(new General());
119             ruleBlock.addRule(Rule.parse("if Rozvrh is MALO_uloh
                and Beziace_ulohy is MALO_beziacich_uloh then
                Vystup is LOW", engine));
120             ruleBlock.addRule(Rule.parse("if Rozvrh is
                STREDNE_uloh and Beziace_ulohy is
                MALO_beziacich_uloh then Vystup is GOOD",
                engine));
121             ruleBlock.addRule(Rule.parse("if Rozvrh is VELA_uloh
                and Beziace_ulohy is MALO_beziacich_uloh then
```

```
                    Vystup is GOOD", engine));
122                 ruleBlock.addRule(Rule.parse("if Rozvrh is MALO_uloh
                    and Beziace_ulohy is STREDNE_beziacich_uloh then
                    Vystup is GOOD", engine));
123                 ruleBlock.addRule(Rule.parse("if Rozvrh is
                    STREDNE_uloh and Beziace_ulohy is
                    STREDNE_beziacich_uloh then Vystup is GOOD",
                    engine));
124                 ruleBlock.addRule(Rule.parse("if Rozvrh is VELA_uloh
                    and Beziace_ulohy is STREDNE_beziacich_uloh then
                    Vystup is GOOD", engine));
125                 ruleBlock.addRule(Rule.parse("if Rozvrh is MALO_uloh
                    and Beziace_ulohy is VELA_beziacich_uloh then
                    Vystup is OVERUSED", engine));
126                 ruleBlock.addRule(Rule.parse("if Rozvrh is
                    STREDNE_uloh and Beziace_ulohy is
                    VELA_beziacich_uloh then Vystup is OVERUSED",
                    engine));
127                 ruleBlock.addRule(Rule.parse("if Rozvrh is VELA_uloh
                    and Beziace_ulohy is VELA_beziacich_uloh then
                    Vystup is OVERUSED", engine));
128                 engine.addRuleBlock(ruleBlock);
129
130
131                 engine.setInputValue("Rozvrh",vytazenieVM);
132                 engine.setInputValue("Beziace_ulohy",zatazenie );
133                 engine.process();
134                 return  outputVariable.getValue() >
                    getUtilizationThreshold();
135                 //              return utilization >
                    getUtilizationThreshold();
136
137         }
138         /**
139         * Sets the utilization threshold.
140         *
141         * @param utilizationThreshold the new utilization threshold
142         */
```

```
143         protected void setUtilizationThreshold(double
                utilizationThreshold) {
144                 this.utilizationThreshold = utilizationThreshold;
145         }
146
147         /**
148         * Gets the utilization threshold.
149         *
150         * @return the utilization threshold
151         */
152         protected double getUtilizationThreshold() {
153                 return utilizationThreshold;
154         }
155 }
```

4.2 Popis riešenia

Systém spúšťa algoritmus procedúrou: *isHostOverUtilized(PowerHost host)*. Kde PowerHost je daný uzol. Algoritmus do premenných **totalRequestedMips** a **maxRequestedMips** pomocou cyklu for načíta maximálne alokované prostriedky (maxRequestedMips) a aktuálne využívané prostriedky (totalRequestedMips). Následne do premennej **utilization** vloží aktuálny pomer medzi **totalRequestedMips / host.getTotalMips()**.
Do premennej **zatazenie** vypočíta aktuálnu mieru zateženia uzla podla vzorca:

$$((host.getTotalMips() - host.getMaxAvailableMips())/host.getTotalMips())$$

Do premennej (vytazenieVM) prepočíta aktuálnu mieru využívania prostriedkov všetkých virtuálnych strojov na uzle podľa vzorca:

$$\textbf{vytazenieVM} = (totalRequestedMips/maxRequestedMips);$$

Následne sa vytvori objekt **engine**. Ide o Fuzzy inferencný systém, v ktorom sa vytvoria a nastavia vstupné a výstupné veličiny. Následne je do FIS nahraná báza pravidiel **ruleBlock**. po inicialiácii FIS sa do vstupnej premennej **Rozvrh** vloží ostrá hodnota z premennej **vytazenieVM** a do vstupnej premennej **Beziace_ulohy** ostrá hodnota z premennej **zatazenie**. Potom sa pomocou procedúry **engine.process()** spustí výpočet na FIS. Následne algoritmus porovná vypočítané zaťaženie s nastavenou hranicou a vracia systému **TRUE** ak je uzol preťažený alebo **FALSE** ak nie je.