შავი ზღვის საერთაშორისო უნივერსიტეტი

კომპიუტერული ტექნოლოგიებისა და საინჟინრო საქმის ფაკულტეტი

ღრუბლიანი გამოთვლების უსაფრთხოების მოდელირების და ანალიზის ახალი მეთოდიკის დამუშავება

მედჰათ მოუსა

სადოქტორო დისერტაციის ავტორეფერატი

ინფორმატიკაში ინჟინერიის დოქტორის აკადემიური ხარისხის მოსაპოვებლად

თბილისი,2014

სამეცნიერო ხემძღვანელი :
პროფ. დოქტ. ირაკლი როდონაია .....

ექსპერტი:

1. დოქტ. გიორგი ღლონტი ----------
2. ასოც. პროფ. გიორგი მანდარია ----------

ოპონენტები:

1. ასოც. პროფ. ნიკოლოზ აბზიანიძე ----------
2. პროფ. მედეა თევდორაძე ----------

შესავალი

ავტონომიური ღრუბელი იბყრობს დიდ ყურადღებას თანამედროვე სამყაროში.ელასტიური ბუნების ღრუბელის გამოყენება, ხდის აუცილებლობას თითქმის ნებისმიერ ორგანიზაციაში .მთავარი გამოწვევაა ღრუბელის წევრების უზრუნველყოფა უსაფრთხოების და პრობლემების მიმართულებით მომსახურება. სისტემა რეაგირებს თანმიმდევრულად და სწორად განსაზღვრავს სიტუაციებში დაწყებულ კეთილთვისებიან მაგრამ უჩვეულო მოვლენების აშკარა შეტევას , ეს არის გასაღები ბიზნესსის თავდაცვის , თვითმმართველობის სამკურნალო და თვითმმართველობის ოპტიმიზაციის მისაღწევად.

უსაფრთხოების თვალსაზრისით , ავტონომიური ღრუბელზე არსებობს მრავალი სერიოზული მუქარა. მავნე პროგრამების თავდასხმებმა შეიძლება მიაყენოს უარყოფითი გავლენას, (QoS) მუშაობის ხარისხს, დროთა დრო შესაბამისად გაუარესდება . განსაკუთრებით იმ შემთხვევაში, თავდასხმის მექანიზმის საფუძველზე განაწილდება უარი (D) DoS მუშაობაში. აქედან გამომდინარე უახლესი მიღწევა ამ მოვლენებზე ბრძოლის არის რეალიზაციის ავტონომიური კომპიუტერული პარადიგმა.ავტონომიური სისტემების განსხვავებული უნარი მგრძნობელობა გვაძლევს შესაძლებლობას თვითმმართველობის კონტროლით, თვითაღდგენის და თვითმმართველობის ოპტიმიზაციის მუდმივ შეგრძნებას, რაც შექმნის მათი მუშაობას ხარისხს .მნიშვნელობა ვეგეტატიური კომპონენტების და ავტონომიური კომპონენტების ანსამბლეის განხილვა მოხდება ამ ნამუშევარში . ჩვენ ასევე გთავაზობთ ენის საკოორდინაციო ანსამბლეის კომპონენტს, რომელიც გამოიყენებით წარმოვადგინოთ კონკრეტული უსაბრთხოების საკითხების ვეგეტატიური გამოთვლა გარემოსი . აღმოაჩინოს ანომალური ქცევა ვეგეტატიური–კომპონენტის ანსამბლეის ინფორმაციის თეორიული მეტრული მიდგომა. გამოყენება აღწერილია სტატიაში .რომ ,განახორციელოს ეს მიდგომა, შეიქმნა ახალი კლასები და მეთოდები შემუშავებული იქნა სწორი გზა კომპონენტების ანსამბლეების ურთიერთქმედების, გარდა ამისა ეს კვლევა წარმოადგენს სრულყოფილ შესწავლას და ახალ მიდგომას ოპტიმიზაციის ტექნიკის რესურსების , რომელიც წარმოდგენილია სხვადასხა სცენარებით და თემატური კვლევებით.

დირესტაციის შინაარსი.

შემოთავაზებული სტრუქტურა საშუალებას გვაძლევს გამოვხატოთ ფორმალურად სასურველი ქცევა(ის რაც უნდა იყოს განხორციელებული, ოპოზიციის როგორც) cloud computing სისტემა ეფუძვნება მკაცრ მათემატიკურ აღწერას.გამოყენების ფორმალური

მეთოდები გვაძლევს საშუალებას, გარანტიას წერტილი დაუსვას შესაძლო დარღვევას .ეს არის შინაარსი დისერტაციის.

პირველ თავში (ლიტერატურის მიმოხილვა) ჩვენ დავიწყეთ მიმოხილვა Cloud Computing ძირითადი მცნებების გარშემო, სადაც არის ახსნილი კონცეპტუალური მინიშნება ამ მოდელის. გარდა ამისა ჩვენ წარმოვადგინეთ მნიშვნელობა ფორმალური ენების და მკაცრი ახსნა დინამიური ქსელების.

მეორე თავი, ჩვენ მივმართედ ფორმალურ ენებს მათი გამოყენების მეშვეობით ღრუბლის ვეგეტატიურ სისტენაში. ნაჩვენებია კონცეფცია SCEL (Software პროგრამული უზრუნველყოფა კონპონენტის ასამბლეის ენა – ეს ენა გამოიყენება ვეგეტატიური კომპიუტერული სისტემებისთვის) მისი განხორციელება და შესრულება jResp–ში, ჩვენ გთავაზობთ აგრეთვე სხვადასხვა სცენარებს, რომ ავხსნათ და გაგიმარტოდ მისი მთავარი წყვლილი ამ თემაზე , რომელიც ეფუძვნება Kolmogorov Complexity metrics სირთულეს და დამატებით გამოყენებას Kullback-Leibler metrics თუ როგორ შეიძლება განხორციელდეს რეალურ გარემოს SCEL ძალადობრივ SLA მიმართ , ასევე აღწერილია ამ თავში .

მესამე თავი, რომელიც არსებითად ეფუძვნება მიღებულ შედეგს მეორე თავში ,(Kripke გამოყენებით და SPIN როგორც ჩვენი შემოთავაზებული მოდელის) ფორმალური ინსტრუმენტის შემოწმება cloud computing გარემოსი უკვე ზუსტად დასტურდება .

ჩვენი კვლევის ბოლო ნაწილში ,ჩვენ შემოგთავაზეთ საბოლოო დასკვნას და სამომავლო გეგმებს. ამ ჩატარებული კვლევით , შესაძლებელია დაინტერესდნენ სხვა მკვლევარებიც , ვინც დაინტერესებულია იმავე მიმართულებით (მოდელირება ავტონომიური cloud computing , ოფიციალური შემოწმება და გადამოწმება კონფლიქტის პოლიტიკის ავტონომიური განაწილების სისტემებში).

მეთოდოლოგია

მიზნების მისაღწევად იქნა გამოყენებული სხვადასხვა ტექნიკა და მეთოდები:

- ფორმალურუ მეთოდები და მიდგომა ფართოდ გამოიყენება დისერტაციის დროს , როგორც ძირითადი მეთოდოლოგია. ცენტრალური პრობლემის ფორმალური მეთოდები შეძლებს უზრუნველყოს, კომპიუტერული სისტემის ქცევის გარანტიას და მკაცრ მიდგომას.ფორმალური მეთოდების საფუძველზე ჩვენ შეგვიძლია მოვძებნოთ

კონცეფცია და დავადასტუროთ, რომელიც ადასტურებს მოდელს, სისტემას და აღწერს მის სასურველ ქცევას – არის რაღაც , რაც უნდა განხორციელდეს ოპოზიციის მიერ.

- ღრუბელის სისტემები უკვე იყო აღწერილი SCEL ფორმალური ენა (Software კომპონენტი ანსამბლი ენების) მეშვეობით ,რომლებიც საშუალებას აძლევს მომხმარებლებს მოდელირების და სერვერების კომპონენტების ასევე ანსამბლეის ურთიერთქმედების ქცევის აღწერას მათ გრძნობელობის და ადაპტირების შეგრძნებას გარემოსთან .
- ძირითადი ცნებები და ელემენტები SCEL – აბსტრაქციის კომპლექტური პროგრამირება , რომლებიც , გვაძლევეს საშუალებას წარმოვიდგინოთ ქცევა, ცოდნა, აგრეგატები ,შესაბამისად კონკრეტულ კომპლექტურ კურსი და პროგრამირების მხარდაჭერის კონტექსტში ცნობიერების, თვითშეგნების და ადაპტაციის გამოიყენება დისერტაციაში.
- ვირტუალიზაციის იდეა ეფუძვნება ცალკეულ კომპონენტების ანსამბლების საფუძველზე. ტერმინი ვირტუალიზაციია ფართოდ აღწერს გამოყოფილ რესურს ან მოთხოვნას მომსახურების ძირითად ფიზიკურ მიწოდებას ამ თუ იმ სერვისით.მაგალითად ვირტუალური, კომპიუტერული პროგრამა არის ხელმისაწვდომი ასევეგვაძლევს მეტი მეხსიერების მოგების შესაძლებლობას ვიდრე ფიზიკურად დაყენებული გამოყენებით background მონაცემთა გადაცემის in დისკზე შენახვა .
- რეზიუმე დიზაინი SCEL პროგრამის ფარგლებში საფუძველზე jResp - Java განხორციელების ძირითადი SCEL ცნებები.
- უსაფრთხოების საკითხებზე ავტონომიური კომპონენტების ანსამბლები ხორციელდება ინფორმაციის თეორიული მეთოდოლოგია Kolmogorov metrics სირთულის და Kullback - Leibler metrics .ვეგეტატიური კომპონენტების დებულებას მოითხოვს, ანსამბლები (SLA) საშუალებით გადამოწმების მეთოდოლოგიას მოდელების საფუძველზე ხაზოვანი დროებით ლოგიკასა და პროგრამული უზრუნველყოფის spin ინსტრუმენტი .

კვლევის მიზანი

როგორც წესი , შემოთავაზებული კვლევები შეიცავს , აღმოვაჩინოთ ძირითადი საფრთხეები,რომელი ფენებიცაა IaaS და PaaS , შესაბამისად, საჭიროებენ შესწავლას , რა არის Cloud Computing მისი მთავარი კომპონენტები და მისი შესაძლებლობები . შემდეგი ხაზი

კვლევის განვითარების, არის შეისწავლონ საიმედო და სრულყოფილი მეთოდები მოსალოდნელი საბრთხეების დასაცავად ამ კონტექსტში, მთავარი ამოცანები სასწავლო მეთოდიკის განზაზღვრულია ქვემოთ:

- საფუძვლიანად გამოვიკვლიოთ ტერიტორიები და მასთან დაკავშირებული სამუშაობები.
- იმისთვის , რომ გავიგოთ და განმარტოთ, კონკრეტული მიზნები და დინამიური პროგრამული სისტემის თვალსაზრისით დარღვევის გარკვეული უსაფრთხოების პოლიტიკა, იმოქმედებს საერთო ხარისხის SLA მომსახურების შესაბამისად.
- გამოიძიოს და აღწეროს მიმდინარე თანამედროვე გამოწვევები პოტენციურ საფრთხეებზე, რომელიც შეიძლება არსებობდეს ტრადიციული სისტემების მიმართ და მის გადასვლას ფიზიკური გარემოში, ზოგადად, IaaS და PaaS.
- განსაზღვრონ და შეისწავლონ პოტენციური მექანიზმები თავდასხმის შიგნით და სხვადასხვა hypervisors შორის hypervisors .
- უზრუნველყოს გარკვეული რეკომენდაციები შემცირების რისკი და გზების მშენებლობას მაღალი უზრუნველყოფილი ინფრასტრუქტურით.
- ანალიზი დაკავშირებული პრობლემების თვალსაზრისით თვითმმართველობის მართვის კომპონენტები შიგნით განაწილებული სისტემები და მისცეს რამდენიმე ოპტიმიზებული გადაწყვეტილებები გასაუმჯობესებლად მთელი ინფრასტრუქტურისა.
- შემუშავება და განხორციელება , ფორმალური მეთოდები და ტექნიკა მოდელირება ასევე ანალიზი უსაფრთხოების cloud computing ავტონომიური სისტემებში.
- შეიქმნა ალგორითმები, რომლებსაც შეუძლია გაუმკლავდეს გამოთვლითი დიდი პროცესების გავრცელება და დინამიური კომპიუტერული ქსელების , რომლებიც
- ავტონომიური გამოთვლითი სისტემები (AS) საფუძველზე და ურთიერთქმედების კომპოზიტური ანსამბლების მათი ურთიერთობა რესურსების გავრცელება ალგორითმი (DRS).
- რომ შემუშავდეს გადაწყვეტილება (AC)ინფრასტრუქტურაში ალტერნატიული მიმდინარე განაცხადების გაშვებული კონკტერული კომპონენტების ანსამბლი,გარკვეულ შემთხვევებში გადავიდეს სხვა ჯანსაღ მონაცემების ცენტრის ფარგლებში.

- ფორმალური შემოწმების მეთოდების განვითარება , მოდელის შემოწმება,რომელიც შეიძლება გამოიყენებულ იქნას აღმოაჩინოს კომფლიქტების ავტონომიური კომპიუტერული პოლიტიკა.

ძირითადი წვლილის და სამეცნიერო სიახლე :

ახალი მიდგომა ანალიზის და შეფასების სწრაფად სფეროში IT - Cloud Computing ავტონომიური (ACC) - არის შემოთავაზებული თეზისი. ACC არის პერსპექტიული მიდგომა , რათა მივაღწიოთ მაღალი დონის ადაპტაციას და გამომსახველობას ასევე შესაძლებლობა თვითმმართველობის კონტროლის კომპონენტების IT სისტემებიში. მკაცრი მათემატიკური მეთოდები, ეწოდება ფორმალური მეთოდები გასაუმჯობესებლად პროგნოზირებადი და საიმედოობის AC შემოთავაზებული ვარიანტი, გამოიყენებუალია დისერტაციაში. ფორმალური მეთოდები გთავაზობთ მოსახერხებელი აბსტრაქცია გაუმკლავდეთ მაღალი განზომილება ავტონომიური ღრუბელის, და მათი საჭიროება ადაპტირება ცვლილებების სამუშაო გარემოს და ცვალებად მოთხოვნებს.

• კომპლექტი პროგრამული აბსტრაქცია , რომელიც საშუალებას გაძლევთ წარმოადგენს ქცევა, ცოდნა და საბჭოები შესაბამისად განსაზღვრული პოლიტიკისა და პროგრამირების მხარდაჭერის კონტექსტში ცნობიერების, თვითშეგნების და ადაპტაციის წარმოდგენილი დისერტაცია. საფუძველზე ამ აბსტრაქცია , პროგრამული კომპონენტი ანსამბლი ენა (SCEL) , ისევე როგორც ენის kernel ფორმალური მიზეზის შესახებ ავტონომიური ქცევის.

ფორმალური მიზეზის შესახებ ქცევის ავტონომიური სისტემები გამოიყენება ახალ კონტექსტში - უსაფრთხოების გამოვლენის მუქარის პროგრამები, როგორიცაა DDoS.

ფორმალური დიზაინის SCEL განახლება და ამას ემატება ახალი განვითარებული ობიექტი, რომელიც საშუალებას აძლევს მკვლევარებს,რათა შეიქმნას და პორმალურად გამოიძიოს სხვადასხვა სცენარის მავნე პროგრამები მკაცრი მათემატიკური ფორმით.

ახალი მიდგომა (და მასთან დაკავშირებული სტრუქტურების)ურთიერთქმედების AC და მათი ქცევა შემოთავაზებულია თეზისში.

ეს საშუალებას აძლევს ვეგეტატიური კომპონენტების რაოდენობრივად დონეზე საფრთხეების რეგულარულად და შესაბამისი ზომები მიიღონ. კერძოდ, ვებ მომსახურება გაშვებული AC-VMs საპრთხის შემთხვევაში ,შეუძლია ოპერატიულად გადავიდეს სხვა მიმდინარე

კომპონენტზე,რომ შეინარჩუნოს უსაბრთხოება.გარდა ამისა , ძალიან მნიშვნელოვანია რომ შევინარჩუნოთ მომსახურების ხარისხი (SLA).

• ინფორმაციის თეორიული მეტრულ - Kolmogorov სირთულის (KS) - დონეზე საფრთხეების ახალ კონტექსტში, ორიგინალურად იზომება რიცხვითი შეფასებების გამოყენებით KS . KS გამოყენებული ინტერფეისების ცოდნა კომპონენტი და ის კომპიუტერული ავტონომიური მენეჯერები AMS კომპონენტის. განსხვავებით ცნობილი მეთოდების გამოყენებით KS in IT სისტემები, ავტონომიური კომპიუტერული და პარალელური დამოუკიდებელი KS ყველა არსებული კომპონენტების საფუძველზე ე.წ. რიცხობრივი შეფასების პაკეტის ნაკადები.

• მიუხედავად იმისა , KS არ არის computable , KS დაახლოების გამოყენებით შეკუმშვის მექანიზმი შემოთავაზებული ონლაინში. ახალი პროგრამული უზრუნველყოფა ობიექტების jResp (Java განხორციელების SCEL) შემუშავებულია და დასძინა, რომ განხორციელების ამ მიდგომის KS . კერძოდ, ახალი კლასების და მეთოდების " complexitySensors " განვითარებულია.

• ახალი ინტერპრეტაცია KS . ეს საშუალებას აძლევს მომხმარებლებს ინტერპრეტაცია KS წელს ალბათური მეთოდი (ცნობილია, რომ ყველა KS არის ინფორმაცია თეორიული კონცეფცია , არ სავარაუდო) და გამოიყენოს იგი ფორმალური გადამოწმების ავტონომიური კომპონენტი ანსამბლებში.

• ახალი სტრუქტურული მოდელი ფორმალური შემოწმების ტექნიკა მოუწოდებს შემოთავაზებული მოდელის გამოვლენის კონფლიქტების პოლიტიკის AC და SLA დარღვევებს. ეს მეთოდი ძირითადად იყენებს შემოთავაზებული ინტერპრეტაციას KS . კერძოდ, ალბათური ინტერპრეტაცია KS გამოიყენება ნაშრომის რაოდენობრივად პასუხი დროის განაწილება მათ შორის (downtime) და ვირტუალური მანქანები , ფაქტობრივად განმარტავს მიგრაციის ვირტუალური მანქანები , რომელიც ძალიან მნიშვნელოვანია, რომ აღმოაჩინოს დარღვევები SLA .

პრაქტიკული მნიშვნელობა და მნიშვნელობა

მთავარი მიზანი სასწავლო არის დაამკვიდრონ შესაბამისი სტრუქტურა, რომელიც შეიძლება მიიღო დღევანდელ IT ინფრასტრუქტურაში (იხ. 2.16). წვლილი კვლევის შექმნილია ძირითადი ნაკადი შესწავლა ავტონომიური Cloud Computing. ნაშრომში შეიძლება კიდევ უფრო განვითარებული უსაფრთხოების პოლიტიკის კონტროლისა და კონფლიქტების ავტონომიური გამოთვლითი გარემოში

სტრუქტურა და შესასრულებელი სამუშაო:

კვლევითი ნაშრომის 188 გვერდები , სამი ნაწილისგან შედგება , ბიბლიოგრაფია და სიაში მოღვაწეები და სიაში ტაბულები.

განსაზღვრის პრობლემა:

კონცეფციები ავტონომიური კომპონენტების (ACS) და ვეგეტატიური კომპონენტის ანსამბლები (ACE) წარმოდგენილია , როგორც საშუალებების მართვის სისტემის კარგად ნაცნობი , დამოუკიდებელი და გავრცელებული ერთეული, რომელიდ გზაც ურთიერთქმედებაში არის გარკვეული. ACs პირები არიან გამორჩეული ბლოკები ცოდნისა და რესურსების ბლოკები, შეგიძლიათ ითანამშრომლონ დროის სხვადასხვა როლებით. ACs ცნობიერებაში უკვე შესაძლებელია ინფორმაციის ცოდნის შენახვა. ამ ობიექტების ასევე საშუალებას ACs შესანახად და მივიღოთ ინფორმაცია მათი სამუშაო გარემოსი , გამოიყენოს მისი გადამისამართება და ადაპტირება მათი საქციელის. თითოეული აღჭურვილია AC ინტერფეისი შედგება კომპლექტით ატრიბუტები, როგორიცაა გათვალისწინებული ფუნქციონალური, სივრცითი კოორდინატები, ჯგუფის წევრობა, ნდობის , რეაგირების დრო და ა.შ.

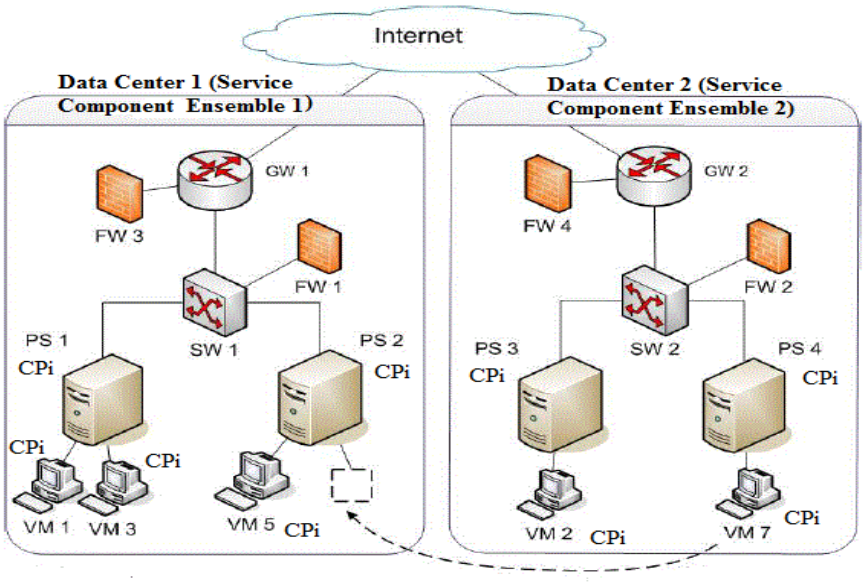
ატრიბუტები გამოიყენება ACs დინამიურად ორგანიზებაში და თავად შევიდა ACE . ინდივიდუალური ACs შეიძლება გამოირჩევა არა მხოლოდ პარტნიორებს გამოყენებას მათი პიროვნების , მაგრამ ისინი ასევე შეგიძლიათ პარტნიორები გამოყენებით ატრიბუტები ინტერფეისები ინდივიდუალური ჰაერის კონდიციონერები . Predicates მეტი ასეთი ატრიბუტები გამოყენებული, რათა დადგინდეს საკომუნიკაციო ობიექტური ქმედებები, რითაც ერთგვარი ატრიბუტი დაფუძნებულია ურთიერთობაზე. ამდენად, ფორმირების წესიენდოგენური ACE ჩანაწერები ACs :

ჯგუფის წევრები ასოცირებული ურთიერთდამოკიდებულება ურთიერთობები განისაზღვრება predicates . ACE არ არის ხისტი ფიქსირებული ქსელი , არამედძალიან დინამიური სტრუქტურაა, სადაც საკომუნიკაციო ACS " დინამიურად არის შექმნილი . შემოთავაზებული აბსტრაქცია არის საფუძველი SCEL (Software კომპონენტი ანსამბლი ენაზე) პროგრამირების ენა kernel ვეგეტატიური გამოთვლითი სისტემებით.

შემდეგი სცენარი განიხილება. განაცხადის Singleton ,ამჟამად მუშაობს ერთი ვირტუალური მანქანა მონაცემთა ცენტრში 2 (VM7 სამსახურში კომპონენტი ანსამბლი 2) .. ეს app მუშაობს მხოლოდ თავის ადგილზე, და მას განაცხადის შედგება ერთი წერტილი , არანაირი დამატებითი შემთხვევები არ გვხვდება . უკანასკნელი ექსპერიმენტი sessionthe განაცხადების მუდმივად მაღალი CPU გამოყენებას. ეს ზრდა შეიძლება იყოს გამოწვეული ან კანონიერი საგზაო შეშუპება ან კოორდინირებული შეტევა (DDoS) , PaaS გაშვებული წინააღმდეგ.

მიმწოდებელებსს შეიძლება შეცდომით მიაჩნიათ, მათი რესურსები შეიძლება გაფართოვდეს და შეიძლება ვერ გაუმკლავდნენ ფინანსურად,პროგრამის გაშვება გაზრდის ხარისხს და შეამცირებს ღირებულებას. (თუ არა და მიმწოდებელი იქნება დაკავებული მატებითი რესურსებით) , ასევე უაზროდ დახარჯავს ენერგიებს.

Figure 1: Cloud Platform and CPi Migration



Source: Pandey, Voorsluys, Niu, Khandoker, & Buyya, 2012, Page-48

შესაბამისად, აუცილებელია განვასხვავოთ ეს ორი საქმე, დასაწყისში ეს განსხვავება , უფრო მეტი ხარისხის დაცვით. . გთავაზობთ ამ დაცვას, ასევე უსაფრთხოების ზომების.Traffic მიედინება კვანძის (CPi) უნდა იყოს გაანალიზებული გამოყენებით Kolmogorov სირთულის metrics (იხ. ტექსტი). შეხვედრისას, მუდმივი მონიტორინგი metrics (სპეციალური probe ხორციელდება ცალკე მოდულთნ) CPU ერთად დატვირთვის პროცესორი ამჟამად შესრულებაშია. იმ შემთხვევაში, თუ ერთდროულად ზრდა ამ ორი ინდიკატორების რეგისტრირებულია მინიმუმ 3 ზედიზედ დროის ერთეული, იმ დასკვნამდე, რომ რეალური DDoS შეტევები უნდა იყოს შედგენილი. შედეგად , განაცხადი უნდა გადავიდესCPI, სადაც ის დაიწყებს მეორე CPI (რომელიც შეიძლება მიეკუთვნებიან band ან სხვა ანსამბლებს). ახალი CPI უნდა იყოს ნაპოვნი ზოგიერთი მოთხოვნებით: დონის სირთულის და CPU უნდა იყოს საკმაოდ დაბალი , ინტეგრირებული ტექნიკა ინდექსი (რომელიც მოიცავს მაჩვენებლებს , როგორიცაა პროცესორის სიჩქარე, ხელმისაწვდომი მეხსიერება , დისკზე, რაოდენობა ბირთვით და ა.შ.) უნდა აკმაყოფილებდეს განაცხადის რესურსების (მათ ქვეყნდება CPI ინტერფეისი, სადაც განაცხადის გაშვებული). იმ შემთხვევაში, თუ საჭირო CPI ი, პროგრამის მიგრაცია არსებობს , რაც შეიძლება მალე , და შეწყვიტოს მისი პროგრესი "მველი" CPI. პროცესი შეიძლება ფორმალურად აღწერილი SCEL განცხადებებს. ჩვენ ვივარაუდოთ, რომ , გარდა ID, ინტერფეისები, და გთავაზობთ ატრიბუტებს " ComplexityLevel ", " CPULoad " და " მეხსიერების" ისტორიის და ინფორმაციის განახლება.

,ძირითადი ინფრასტრუქტურა (როგორც წესი საწყისი ეკრანები, გეითვეი ან სპეციალური კვლევები) არიან `ელემენტს.CPI, სადაც პროგრამა გაშვებულია არის SCEL კომპონენტი: $\mathcal{I}[\mathcal{K}, \Pi, AM[ME]]$

P$_{\text{ComplexityMonitor}}$ ≜ **qry**(“*ComplexityLevel*”, “high”) @ self.
get(“*ComplexityHigh*”, false) @self.
put((“*ComplexityHigh*”, true) @self. **qru**(“*ComplexityLevel*”,”low”)@self.
get(“*ComplexityHigh*”,true)@self.
put(“*ComplexityHigh*”,false)@self.P$_{\text{ComplexityMonitor}}$
P$_{\text{CPULoad}}$ ≜ **qry**(“*CPUloadLevel*”,“low”)@ self. **get**(“*CPULow*”, false) @self.
put((“*CPULow*”, true) @self. **qru**(“*CPUloadLevel*”,”high”)@self.
get(“*CPULow*”,true)@self.
put(“*CPULow*”,false)@self. P$_{\text{CPULoad}}$
P$_{\text{MigrateCP i}}$ ≜ **qry**(“*Cloud service*”, ?X)@ self
get(“*Cloud service_args*”, ?*sessionId,* ?*memoryValue,* ?*CPUValue*) @self.

/* retrieving from the knowledge repository the process implementing a required functionality id and bounding it to a process variable X */
/* searching an item *c* among components belonging to the ensemble identified by predicate Ω */
qry(“*CPiId*”, ?c) @Ω .
/* storing actual parameters of the process to be executed in the found component c : moving from VM7 to /*MV5 on fig.2 */
put(“*Cloud service*”, ?*sessionId*, ?*memoryValue*, ?*CPUValue*)@c
get(“*Cloud service*”, “*sessionId*”, “*terminated*”) @self.
/* removing the process from the knowledge repository of ‘old’ CPi */
get(“*Cloud service*”, “*sessionId*”, X) @self.nil
/* eliminating the process in ‘old’ CPi */

Here the predicate Ω is determined as follows:

Ω (*I*) =(*I* .ComplexityLevel=”low”)∧ (*I* CPULoad <75)∧(*I* .Memory>=500 and is used for group-oriented communication in the action qry(“CPiId”, ?c) @.

ეს ძირითადი განსაზღვრავს ანსამბლი კომპონენტებს, რომლებიც აქვეყნებენ მათი ინტერფეისების ატრიბუტებს “ComplexityLevel”,CPULoad” and” Memory” ერთად შესაბამის ღირებულებებს.. ჩვენ ვივარაუდოთ, რომ ეს ატრიბუტიკა უზრუნველყოფს ინტერფეისის თითოეული კომპონენტის განახლება მიიღოს დინამიურად ღირებულებების შესაბამისი კვლევა (სენსორები) შედეგად მუდმივი მონიტორინგი (ზონდირების) გამოთვლითი გარემოში. ჩვენ ვივარაუდოთ, რომ ატრიბუტი "ComplexityLevel" იძლევა მითითებით დიაპაზონი [0:1] სირთულის დონე (იხილეთ განმარტება ქვემოთ ტექსტში) მონაცემები შემოვა მეშვეობით ანსამბლი, ატრიბუტი "CPULoad" - იმ დიაპაზონში [0 : 100], ატრიბუტი "მეხსიერება" - იმ დიაპაზონში [0:1000]. ამ კონტექსტში მნიშვნელობა ძირითადი ასეთია: იპოვოს კომპონენტი CPI (ან კომპონენტების), სადაც "ComplexityLevel" არის დაბალი (ანუ ნაკლები 0.15), "CPULoad" არის არანაკლებ 75 და ხელმისაწვდომი მეხსიერების ინდექსი "მეხსიერება" უფრო მეტი 500.

დამოუკიდებლად მომსახურების კომპონენტის, რომელიც ღრუბელის მომსახურება ხორციელდება ("ძველი" CPI ან ახლად ნაპოვნი "მიმღები მიგრაცია სერვისი" CPI) SCEL განცხადებები, რომელიც აღწერს პროცესს Ps ახორციელებს მოახერხა ელემენტს ME ასეთია:

P$_s$ ≜ **get** (“*Cloud service*”, ?*sessionId*, ?*memoryValue*, ?*CPUValue*)@self.
get(¨*CPUload*”, ?L) @self.
/* L is a current CPUload of the component
get(“*memory*”, ?M) @self.
/* M is a current allocated memory
put(“*CPUload*”, (L+ *CPUValue*))@self.
put(“*memory*”, (M- *memoryValue*))@self.

P$_s$ [X(*sessionId, memoryValue, CPUValue*)]
/* the new process (additionally to the already running process Ps), having actual parameters sessionId, memoryValue, CPUValue, starts */

In the thesis the run-time Java implementation of the SCEL formal code (expressed in jResp environment) has been developed. The main classes of jResp, corresponding to the above SCEL code, are as follows: **ServiceComponent, CloudService, ServiceCaller, RequestHandler, OfferAgent**. The scenario, described above, is realized by means of jResp classes **Scenario** and **Main** (the structure of datacenter):

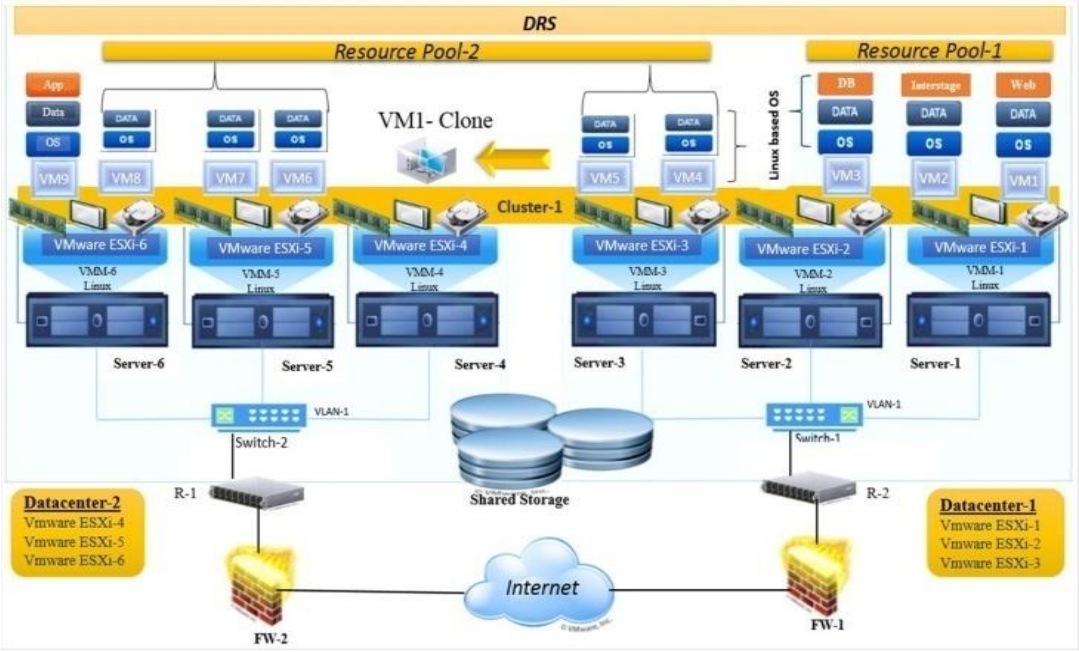


Fig.2 Structure of the datacenter.

```
public class Main {
private Scenario scenario;
private Random r = new Random();
private Target id;
 public Main(int size){
        this( new Scenario(size, new Random() , 900, 1000, 99, 100) );
}
/* above the minimum and maximum values of components' memory (900 and 1000,
/* respectively) and minimum and maximum values of components' CPU rates (99 and
/* 100, respectively) are defined

public Main(Scenario scenario) {
        this.scenario = scenario;
        instantiateNet();
 }

private void instantiateNet() {
        Random r = new Random();
        SimulationScheduler sim = new SimulationScheduler();
        SimulationEnvironment env = new SimulationEnvironment(sim, new
         RandomSelector(r), new DeterministicDelayFactory(1.0) );
```

```
                sim.schedulePeriodicAction(new SimulationAction() {

            VirtualPort vp = new VirtualPort(10);
            Hashtable<String, Node> nodes = new Hashtable<String, Node>();
            for(int i=0; i<scenario.getSize(); i++){
                    Node n = new Node("ServiceComponent"+i, new TupleSpace());
                    n.addPort(vp);
                    n.addSensor(scenario.getComplexitySensor(i));
                    n.addSensor(scenario.getCpuSensor(i));
                    n.addSensor(scenario.getMemorySensor(i));
                    n.addActuator(scenario.getServiceInvocationActuator(i, n));
                    n.addAttributeCollector(scenario. getComplexityAttributeCollector ());
                    n.addAttributeCollector(scenario.getCpuLoadAttributeCollector());
                    n.addAttributeCollector(scenario.getCpuRateAttributeCollector(i));
                    n.addAttributeCollector(scenario.getMemoryAttributeCollector())
                    n.put(new Tuple("REQUEST", 1, new CloudService("1", 10, 2.0),
                    n.getLocalAddress() ));
                    n.put( new Tuple( "LOCATION" , n.getLocalAddress() ) );
                    Agent a= new RequestHandler();
                    n.addAgent(a);
                    a=new OfferAgent();
                    n.addAgent(a);
                    nodes.put(n.getName(), n);
          }
           for (Node n: nodes.values()) {
                n.start();
               /* for simulation option:
               /* env.simulate(10000);

            public static void main(String[] args) {
            int size=8; /* Here 8 components (virtual machines) start
             /* high complexity level (=1, no security threats) is assigned to  all virtual   machines
              ServiceComponent c0 = new ServiceComponent ( 0 ,1, 0 , 1 );
              ServiceComponent c1 = new ServiceComponent ( 1 , 1, 100 , 1 );
              ServiceComponent c2 = new ServiceComponent ( 2 , 1, 100 , 1 );
             ……
             ServiceComponent  c7= new ServiceComponent (7, 1, 100,1);

              new Main( new Scenario( c0 , c1 , c2, c3, c4, c5, c6, c7 ));
        }
}
```

In order to include into autonomic components ensembles the process of regular monitoring and updating of Kolmogorov complexity level, we need to make significant changes in jResp sensors implementation. Regularly recalculated and obtained Complexity metrics attributes have to be incorporated in knowledge space of components. For this reason, the necessary changes are shown and described. First, all attributes sensors are extensions of base class **AbstractSensor** (which is one of the basic classes of jResp):

```
public abstract class AbstractSensor extends Observable {
                protected String name;
                protected Tuple value;
                public AbstractSensor(String name) {
                this.name = name;
    }
                public String getName() {
                return name;
    }
```

```
        public final Tuple getValue() {
        return value;
    }
        public final void setValue( Tuple t ) {
        this.value = t;
        this.setChanged();
        this.notifyObservers(t);
    }
}
```

In order to regularly update complexity level attribute, components' autonomic managers have to connect to the netflow enabled router (see later) as sensor client using socket technology. On the other hands, the router has to get this connection (through socket technology).

```
public class ComplexitySensorClient  extends AbstractSensor {
    private String serverAddress;
    private int serverPort;
    private Gson gson = RESPFactory.getGSon();
    private long refreshTime

  public ComplexitySensorClient t(String name , String serverAddress , int serverPort , long refreshTime )
    throws IOException {
        super( name );
        this.serverAddress = "148.169.2.45"; /* this is a network address of the router
        this.serverPort = 1024;
        this.refreshTime = refreshTime;
        new Thread( new SensorThread() ).start();
    }

  public class SensorThread implements Runnable {
        @Override
        public void run() {
            while (true) {
                Socket s;
                try {
                    System.out.println(getName()+" requests a complexity
                     value...");
                    s = new Socket(serverAddress,serverPort);
                    BufferedReader reader = new BufferedReader(new
                   InputStreamReader(s.getInputStream()));
                   Tuple t = gson.fromJson(reader, Tuple.class);
                  /* Json for serialization of tuples and messages
                    reader.close();
                    s.close();
                    System.out.println(getName()+" delivers a complexity
                     value...");
                    setValue(t);
                  } catch (UnknownHostException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                setValue(null);
                try {
                    Thread.sleep(refreshTime);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                    return ;
```

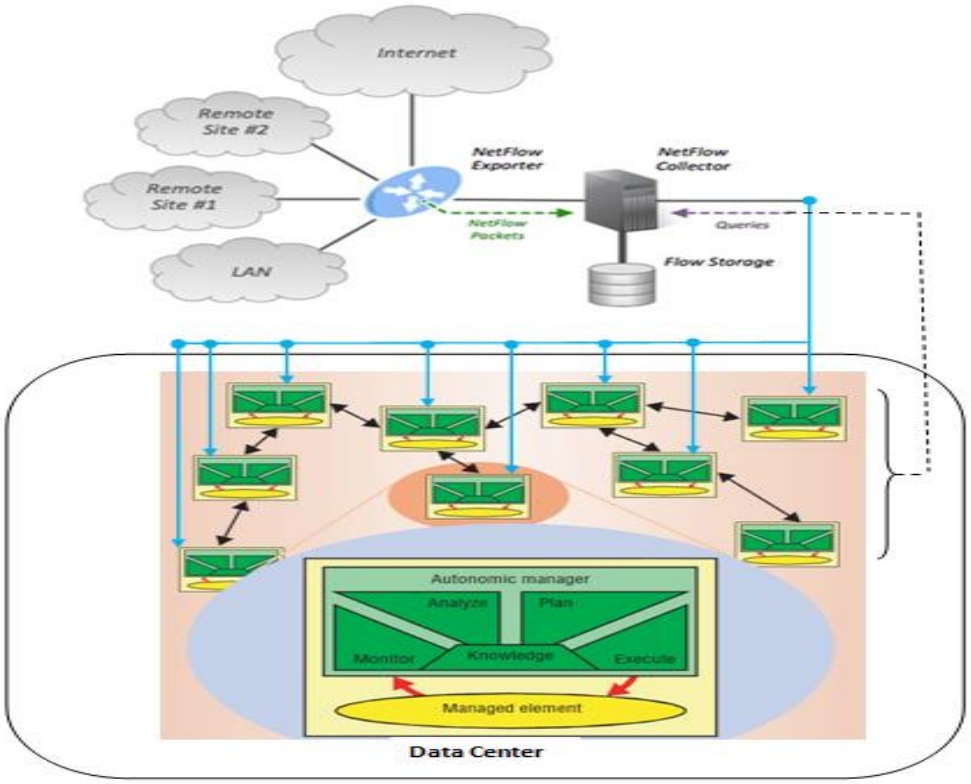
```
        }
    }
```

კლასი ComplexitySensorServer საშუალებას იძლევა დისტანციური კომპონენტებით ხელმისაწვდომობის სენსორის მეშვეობით კავშირის და ახლიდან სირთულის დონეზე გაშიფრვა. განხორციელების კლასის არისიდენტური ComplexitySensorClient (socket ტექნოლოგიის კიდე ერთხელ მეშვეობით). შედეგად, ხდება განახლება სირთულის დონე ატრიბუტი რომელიც გვხვდება მეთოდი updateSensorsValue ServiceComponent კლასის ხორციელდება რეგულარულად (კონკრეტულ ვადაში)

მიდგომა ავტონომიური კომპიუტერული უსაფრთხოებისა და ანომალიის გამოვლენის, განვითარებული თეზისი, მცნებები netflows , მათი საინფორმაციო თეორიული metrics და კომპონენტები " ავტონომიური მენეჯერი არსებითად leveraged . ქსელი ნაკადი შეიძლება განისაზღვროს მრავალი გზით. საერთო გაგებით , ნაკადი არის მთელი რიგი პაკეტი რამდენიმე ატრიბუტი (s). თითოეული პაკეტი , რომელიც მიემართება ფარგლებში როუტერი ან შეცვლა გამოკვლევა კომპლექტი IP პაკეტად ატრიბუტებში. ეს ატრიბუტიკა არის IP პაკეტი მისამართის ან ანაბეჭდი პაკეტი და განსაზღვრავს, თუ პაკეტი არის უნიკალური ან მსგავსია სხვა პაკეტი. ყველა პაკეტი იგივე წყარო / დანიშნულების IP მისამართი წყარო / დანიშნულების პორტი, ოქმი ინტერფეისი და დონის მომსახურების დაჯგუფებულია ნაკადით და შემდეგ პაკეტი და ბაიტი შეფასებული. ეს მეთოდოლოგია fingerprinting ან განსაზღვრის ნაკადის scalable , რადგან დიდი რაოდენობით ქსელის ინფორმაციის შემოკლება შევიდა მონაცემთა ბაზაში netflow.მაშტაბურად დიდი რაოდენობის ქსელის ინფორმაციის შემოკლება შევიდა მონაცემთა ბაზაში netflow მოუწოდა netflow cache.

Netflow-ჩართულია მოწყობილობა (netflow ექსპორტიორი: როუტერი ან შეცვლა) უგზავნის netflow კოლექციონერი (fig.3) ერთი ნაკადი, როგორც კი ნათესავ კავშირს დაუკავშრდება იწურება. პაკეტი მიტაცებული netflow კოლექციონერი ინახება ნაკადის ბაზაშ პორტი და მისამართი IP დისტრიბუციის მაღალ კორელაციაშია, ქსელის ტრაფიკის. ამ მიზეზით, ჩვენ მხოლოდგანვიხილავთ წყაროს და დანიშნულების IP.

Figure.3 ურთიერთქმედების netflow მოწყობილობა და ავტონომიური კომპონენტები



Source: Haag, User Documentation nfdump & NfSen , 2014

ნაკადების დაგროვილ ნაკადის შენახვა, ხდება subdivided და შედის კომპონენტის ნაკადებშ,. დამიედინება, რომელსაც აქვს კომპონენტი IP მისამართი როგორც დანიშნულების მისამართი, დაჯგუფება იგზავნება შესაბამის კომპონენტთან (უფრო ზუსტად, რომ ავტონომიური მენეჯერი კომპონენტი - ეს სიდიდეები აღინიშნება ლურჯი ისრებით fig.3). მიღების შემდეგ მათი განკუთვნილი ნაკადების კომპონენტი ავტონომიური მენეჯერის შეგიძლიათ დაიწყოთ დამუშავება, რათა გამოავლინოს პათოლოგიური ქცევის ნაკადების შესაბამისად, ტექნიკით, რომელიც აღწერილია და იქნება შემდგომში. ამ ნაშრომის სხვადასხვა ფაილების კერძოდ titles (შესაბამისი კონკრეტული SCPI ს IP მისამართები) შესანახად კომპონენტის ნაკადები გამოიყენება. რაც შეიძლება მარტივად რომ ვივარაუდოთ , ინფორმაცია , რომელიც მოიცავს დაკვირვების ქმედებების ერთ მიზეზს , თუ არა ისინი, ხარვეზებით ან თავდასხმელები , მაღალ კორელაციაშია . მაღალ კორელაციაშია მონაცემები აქვს მაღალი შეკუმშვის . Kolmogorov სირთულე , K (x), სიმებიანი მონაცემები ზომები არის ზომით პატარა პროგრამა , რომელსაც შეუძლია წარმოადგინოს მოცემული ნაჭერი

მონაცემები . ეს ზომავს ხარისხი randomness მოცემული მონაცემების ხანგრძლივობას უმოკლესს პროგრამის გენერირებით სიმებიანი ზომით ტოლია. ყველა სხვა შემთხვევაში, ეს მცირე ზომა სიმებიანი პროგრამის ზომის ხდება პატარა, როგორც უფრო სისტემატური , ნიმუში არის შესამჩნევი საწყის სტრიქონში. გვერდითი ეფექტი ეს ღონისძიება არის მისი უნარი წარმოაჩინოს კორელაცია განსხვავებული ცალი მონაცემების.ეფექტი ეს მხარე გამოყენებულია , რათა შეიმუშავონ ეფექტური მეთოდი გამოვლენის DDoS შეტევების. DDoS შეტევის გამოვლენის ალგორითმი იყენებს ფუნდამენტური თეორემა Kolmogorov სირთულე , რომელიც აცხადებს, ნებისმიერი ორი შემთხვევითი strings X და Y,

$$K (XY) \leq K (X) + K (Y) + c,$$

სადაც K (X) და K (Y) არიან სირთულეების შესაბამისი strings, c არის მუდმივი და K (XY) არის ერთობლივი სირთულის გაერთიანების სიმები . მარტივად რომ ვთქვათ, ერთობლივი Kolmogorov სირთულის ორი სიმები რომელიც ნაკლებია ან ტოლი თანხა სირთულეების ინდივიდუალური სიმები. ექვივალენტი ფლობს , როდესაც ორი strings X და Y სრულიად შემთხვევითი , ანუ ისინი სრულიად არ არიან დაკავშრებული ერთმანეთს. სხვა ეფექტით ესერთობლივი ურთიერთობა სირთულის strings ამცირებს როგორც კორელაცია strings იზრდება. ინტუიციურად, თუ ორი სიმები დაკავშირებულია, მათ აქვთ საერთო მახასიათებლები და საერთო შაბლონები. თვალსაზრისით გამოვლენის DDoS შეტევები, საკუთრების მიერ მოცემული უტოლობა (1) გამოყენებული გამოირჩეოდნენ შორის შეთანხმებული უარის თქმის -of -service შეტევა და საქმეების მოძრაობის გადატვირთვისაგან. ვარაუდი არის ის, რომ თავდამსხმელი ასრულებს თავდასხმის გამოყენებით დიდი რაოდენობით მსგავსი პაკეტებს (თვალსაზრისით მათი ტიპის , დანიშნულების მისამართი, აღსრულების ნიმუში და ა.შ.) sourced საწყისი სხვადასხვა ადგილებში , მაგრამ განკუთვნილია იგივე დანიშნულებისათვის. ამდენად, არსებობს უამრავი მსგავსება მოძრაობის ნიმუში. Kolmogorov სირთულის დაფუძნებული გამოვლენის ალგორითმი სწრაფად ახდენს იდენტიფიცირებას ასეთი ნიმუშის. მეორეს მხრივ , საქმე ლეგიტიმური მოძრაობის გადატვირთვისაგან ქსელში ტენდენცია აქვს სხვადასხვა საგზაო ტიპს. მოძრაობის ნაკადები არ არიან მაღალ კორელაციაშია და , როგორც ჩანს, ალგორითმი ნიმუშები

ყველა განსხვავებული ნაკადი პაკეტი (გამოირჩევა წყარო და დანიშნულების მისამართი), რათა დადგინდეს , თუ არსებობს დიდი რაოდენობით კორელაცია პაკეტებსნაკადია. თუ ეს

სადაც K (X) და K (Y) არიან სირთულეების შესაბამისი strings, c არის მუდმივი და K (XY) არის ერთობლივი სირთულის გაერთიანების სიმები . მარტივად რომ ვთქვათ, ერთობლივი Kolmogorov სირთულის ორი სიმები რომელიც ნაკლებია ან ტოლი თანხა სირთულეების ინდივიდუალური სიმებია. ექვივალენტი ფლობს , როდესაც ორი strings X და Y სრულიად შემთხვევითია , ანუ ისინი სრულიად უკავშირდებიან ერთმანეთს. სხვა ეფექტი ეს ურთიერთობა რომ ერთობლივი სირთულის strings ამცირებს როგორც კორელაცია strings იზრდება. ინტუიციურად, თუ ორი სიმები დაკავშირებული, მათ საერთო მახასიათებლები და ამით საერთო შაბლონებს. თვალსაზრისით გამოვლენის DDoS შეტევები, საკუთრების მიერ მოცემული უტოლობა (1) გამოყენებული გამოირჩეოდნენ შორის შეთანხმებული უარის თქმის -of -service შეტევა და საქმეების მოძრაობის გადატვირთვისაგან. ვარაუდი არის ის, რომ თავდამსხმელი ასრულებს თავდასხმის გამოყენებით დიდი რაოდენობით მსგავსი პაკეტი (თვალსაზრისით მათი ტიპის , დანიშნულების მისამართი, აღსრულების ნიმუში და ა.შ.) sourced საწყისი სხვადასხვა ადგილებში , მაგრამ განკუთვნილია იგივე დანიშნულების. ამდენად, არსებობს უამრავი მსგავსება მოძრაობის ნიმუში. Kolmogorov სირთულის დაფუძნებული გამოვლენის ალგორითმი სწრაფად იდენტიფიცირება ასეთი ნიმუში. მეორეს მხრივ , საქმე ლეგიტიმური მოძრაობის გადატვირთვისაგან ქსელში ტენდენცია აქვს სხვადასხვა საგზაო ტიპს. მოძრაობის ნაკადების არ არიან მაღალ კორელაციაშია და , როგორც ჩანს, შემთხვევითი. ამიტომ, ალგორითმი ნიმუშები ყველა განსხვავებული ნაკადი პაკეტი (გამოირჩევა წყარო და დანიშნულების მისამართი), რათა დადგინდეს , თუ არსებობს დიდი რაოდენობით კორელაცია პაკეტებსნაკადი. თუ ეს განსაზღვრული უნდა იყოს , მაშინ ყველა საეჭვო ნაკადების ერთი კვანძის კვლავ კორელაციაშია ერთმანეთთან , რათა დადგინდეს, რომ ეს არის მართლაც თავდასხმის და არა შემთხვევაში მოძრაობის გადატვირთვისაგან.

მიუხედავად იმისა, რომ ცნობილია, რომ , ზოგადად, Kolmogorov სირთულის არ არის computable , სხვადასხვა მეთოდები არსებობს გამოთვლაც გათვლით სირთულის. ეს

შესაძლებელია მოიპოვოს ზედა ზღვარი ის (რომელიც პრაქტიკულად არის ძალიან კარგი დაახლოების) გამოყენებით უნივერსალური შეკუმშვის ალგორითმს, როგორიცაა Lempel და Ziv. კომპონენტი ავტონომიური მენეჯერი გამოიყენება მონიტორინგის შესაბამისი (საკუთარი) ნაკადების და გაანგარიშების აფასებს სირთულის დონეს . კომპონენტი სხდომაზე კვალი შედგება, ყველა პაკეტი ნაკადების დრო ზოგიერთი განსაზღვრულ დროს (სხდომა). ეს კვალი რომლებიც concatenated და მოაქცია შევიდა ASCII string (გამოყენებით ინსტრუმენტები nfdump , tcpdump ან Win_RK). მაშინ ეს string s არის შეკუმშული გამოყენებით Lempel – Ziv-ს ალგორითმი (LZW.java), რითაც მოპოვების new მოკლე string s1 . ახლა Kolmogorov სირთულის (KC), როგორც თანაფარდობა სიგრძე და სიგრძის s1 შეიძლება იყოს გათვლილი . იმის გათვალისწინებით, რომ სიგრძეზე s1 არ შეიძლება უფრო , რომ სიგრძე s, KC ყოველთვის შორის 0 და 1.

შეფასდეს შესაძლო სპექტრს KS metrics ღირებულების მიანიშნებს ბევრი მსგავსება მოძრაობის ნიმუში (აქედან გამომდინარე, დიდი რაოდენობით მსგავსი პაკეტი (თვალსაზრისით მათი ტიპის , დანიშნულების მისამართი, აღსრულების ნიმუში, და ა.შ. , რომელიც საეჭვო თვალსაზრისით DOS ან DDoS შეტევაა) , მრავალი სიმულაციური ექსპერიმენტი განხორციელდა დისერტაციის .. ცნობილი სიმულაციური ინსტრუმენტი CloudSim - ფარგლებში მოდელირებისა და სიმულაციის of cloud computing ინფრასტრუქტურა და მომსახურება - უკვე გამოიყენება . მიზანი სიმულაციური იყო განსაზღვრა გავლენას , რომელიც სხვადასხვა პარამეტრები ავტონომიური პაკეტი მიედინება შორის კომპონენტები SCPI ანსამბლები. შედეგად სიმულაციური ექსპერიმენტი ჩვენ დაადგინეთ, რომ რიგ Kolmogorov სირთულე metrics მნიშვნელობა მიანიშნებს საკმაოდ მაღალი დონის DDoS საფრთხეების სპექტრი 0 ÷ 0.15 . ეს მნიშვნელობა მიღებულია ნაშრომის სერიოზულ საფრთხედ .

რათა შეამციროს ალბათობა ე.წ. " ცრუ დადებითი განგაშის " (რომ არის, როდესაც განგაშის DDoS შეტევა მკურნალობის ჩნდება პირობებში რეგულარულად "ჯანსაღი" საგზაო) სხვა ინფორმაციის თეორია metrics , კერძოდ, Kullback - Leibler განსხვავების მეტრულ გამოიყენება ნაშრომის . იმ შემთხვევაში , როდესაც ორივე metrics (Kolmogorov სირთულის დონეზე და Kullback - Leibler განსხვავების მეტრულს) აქვს

სათანადო დაბალი ღირებულებები , მაშინ ჩვენ ვივარაუდოთ, რომ ალბათობა DDoS შეტევა არის ძალიან მაღალი. In jResp გამოთვლაშიKullback - Leibler განსხვავების მეტრულად ხორციელდება ერთხელ კლასში ComplexitySensorServer (ანალოგიურად რადგან ის გამოთვლილია განთავსების Kolmogorov სირთულე metrics).

IT ინფრასტრუქტურის გათვალისწინებული datacenter მფლობელები / ოპერატორები უნდა აკმაყოფილებდეს სხვადასხვა SLAs შეუქმნან კლიენტებს. SLAs შეიძლება რესურსების დაკავშირებული (მაგალითად, თანხის კომპიუტერული ძალა , მეხსიერების / საცავები, ქსელის სიჩქარეს) , შესრულების დაკავშირებული (მაგალითად, მომსახურების დროს ან გამტარუნარიანობა), ან თუნდაც ხარისხის მომსახურების დაკავშირებული (მაგალითად, 24-7 ხელმისაწვდომობა, მონაცემთა უსაფრთხოების , პროცენტული დონე მოითხოვს). Datacenter მოიცავს დიდი რაოდენობით პოტენციურად ჰეტეროგენული სერვერები შერჩეული კომპლექტი ცნობილი და კარგად ხასიათდება სერვერის ტიპის. კერძოდ, სერვერები მოცემული ტიპის მოდელირებული მათი მიერ გადამუშავების მოცულობა და ძირითადი მეხსიერების ზომა , ისევე როგორც ფუნქციონირებს ხარჯი (ენერგეტიკული ღირებულება), რომელიც პროპორციულია მათი საშუალო ენერგომოხმარებლების. თითოეულ კლიენტთან აწარმოებს , ერთი ან მეტი VMs , რომელიც სრულდება გარკვეული სერვერებიdatacenter . თითოეულ კლიენტისთვის ასევე შეიქმნა SLA კონტრაქტი datacenter მფლობელი. შესრულება თითოეული კლიენტის cloud computing სისტემის მონიტორინგი უნდა და საჭირო გადაწყვეტილებები უნდა იქნას მიღებული , რათა დააკმაყოფილოს SLA მოთხოვნებს. კლიენტის სისტემა განაცხადის პროგრამული უზრუნველყოფა, რომ შეუძლია აწარმოოს რიგი მოთხოვნები ყოველი ერთეულის. მოდელზე რეაგირების დრო კლიენტებს, ჩვენ ვივარაუდოთ, რომ inter- ჩამოსვლის დროს მოითხოვს თითოეული კლიენტი დაიცვას ექსპონენციალური განაწილების ფუნქცია მსგავსია, inter- ჩამოსვლის დროს მოთხოვნები e-commerce პროგრამები. მინიმალური დაშვებული inter- ჩამოსვლის დროს მოთხოვნები მითითებულია SLA ხელშეკრულებით. ექსპონენციალური განაწილების ფუნქცია გამოიყენება მოდელზე მომსახურების დროს კლიენტებს ამ სისტემაში. საფუძველზე

ამ მოდელის რეაგირების დრო განაწილებისVM (განთავსებული სერვერზე j) არის ექსპონენციალური განაწილების საშუალო :

$$\bar{R}_{ij} = \frac{1}{C_j^p \phi_{ij}\mu_{ij} - \alpha_{ij}\lambda_i} \qquad (1)$$

სად μij ნიშნავს მომსახურების განაკვეთი თვისოეული კლიენტიjth სერვერზე როდესაც ერთეული გადამამუშავებელი სიმძლავრე გამოყოფილია VM ამ კლიენტს. VM ერთეული განისაზღვრება, როგორც ძირითადი ერთეულის ვირტუალური რესურსი, რომელიც უკავშირდება კომპლექტი ფიზიკური რესურსები, როგორიცაა CPU დრო, ძირითადი მეხსიერება, საცავები, ელექტროენერგიის და ა.შ. რეალური ღრუბელი სისტემები, ნებისმიერი ვირტუალური რესურსი მომხმარებელს შეუძლია იყოს ჯერადი VM ერთეული.

როგორც ზემოთ იყო აღნიშნული , იმ შემთხვევაში, მაღალი ალბათობა DDoS შეტევა დაუყოვნებლივ მიგრაცია კომპონენტიVM (სადაც კომპონენტი აწარმოებს გაკეთებული) სხვა VM (რომელიც შეირჩევა გამოყენებით ანსამბლის კომპონენტები ავტონომიური მენეჯერთა ბაზაში და გაცემის სპეციალური SCEL განაცხადი qry) აუცილებელია. დროს მიგრაციის მხედველობაში განსაზღვრის რეაგირების დრო საფუძველზე (1). მიგრაციაVM შორის

სერვერები იწვევს downtime კლიენტის პროგრამაში. ხანგრძლივობა downtime არის დაკავშირებული მიგრაციის ტექნიკაზე, გამოიყენება datacenter . Downtime ასევე არის ფუნქცია ლინკი სიჩქარე და VM მეხსიერების ზომა .

ამ ნაშრომის შემდეგი მიდგომა განახლება რეაგირების დრო განაწილებისVM (1) ხორციელდება. მიუხედავად იმისა, რომ Kolmogorov სირთულის (KS) თავისი ორიგინალური განსაზღვრება არ არის ალბათური ცნება (ეს არის ერთი საინფორმაციო თეორია) , ჩვენ ვიყენებთ KS სახით მიღებული გამოყენებით Lempel - Ziv შეკუმშვის ალგორითს (რომელიც აკონვერტებს ორიგინალური KS მაჩვენებს ,შედის ღირებულებების სპექტრის 0 ÷ 1). ახლა ჩვენ შეგვიძლია განვიხილოთ მაჩვენებელი , როგორც ზოგიერთი დაახლოებული ალბათობა არსებობა malware (DDoS)

საფრთხეები. კერძოდ, ჩვენშევაფასებთ ალბათობა იმისა, რომ jth სერვერზე გამოყოფილი ith კლიენტის VM ექვემდებარება malware საფრთხე შეტევას (და საჭიროებს მიგრაციას), როგორც ღირებულება: prob (MigrVMij) = 1 KSij . მაშინ , ფორმულით (1) უნდა განახლება დასძინა ტერმინი წარმოადგენს სავარაუდოდ downtimeVMij :

(2)

აქ ჩვენ ვივარაუდოთ, რომ VM მეხსიერების ზომა არის მუდმივი და უდრის 1,024 მბ. ეს ღირებულება downtime იქნება არსებითად გამოიყენება მიდგომა ფორმალური გადამოწმების ავტონომიური გამოთვლითი სისტემის განვითარებული თეზისი.

ავტონომიური სისტემები განისაზღვრება, როგორც სისტემები, " მართვას თავს შესაბამისად ადმინისტრატორის მიზნებს ".ეფექტურობა თვითმმართველობის მართვის დამოკიდებულია ხარისხის ავტონომიური კომპიუტერული პოლიტიკის, გამოიყენება გამოხატოს ამ მიზნების მისაღწევად. მკაფიოდ განსაზღვრული პოლიტიკით გამოიწვიოს არაეფექტური თვითმმართველობის მართვა; კონფლიქტის პოლიტიკით შეიძლება იყოს downright საზიანო ავტონომიური სისტემა.ფორმალური ტექნიკა უწოდა, მოდელის შემოწმება შეიძლება გამოყენებულ იქნას აღმოაჩინოს კონფლიქტების ავტონომიური კომპიუტერული პოლიტიკა. მოდელი შემოწმებას წარმოადგენს ფორმალური ტექნიკა დამადასტურებელი თუ არა სისტემა აკმაყოფილებს მისი დაზუსტებას. ტექნიკა მოიცავს მშენებლობის მათემატიკურად დაფუძნებული მოდელის სისტემის ქცევას, და შემოწმებას, რომ სისტემის თვისებები მითითებული ფორმალურად დროებითი ლოგიკა ახდენენ ამ მოდელს. თითოეული ქონება, ტექნიკით უკომპრომისო counterexample შედგება შესრულების გზით , რომელიც ქონებას არ ფლობს . შედეგი ეფუძნება ამომწურავი ანალიზის სახელმწიფო ფართის განხილული მოდელის - მახასიათებელს , რომელიც ადგენს მოდელი შემოწმების გარდა, დამატებითი ტექნიკას , როგორიცაა ტესტირება და სიმულაცია .

სისტემის მოდელი ყველაზე ხშირად გამოყენებული მოდელის შემოწმებას უწოდაKripke სტრუქტურა. იგი შედგება სახელმწიფო გარდამავალ გრაფაში, სადაც S წარმოადგენს სასრულ კომპლექტი ქვეყნების ტრანზიტულ გრაფას.

რომელ სისტემას შეუძლია იყოს, არის კომპლექტი საწყის მდგომარეობის , არის მიზეზი , რომელიც განსაზღვრავს ყველა შესაძლო გადასვლებს მდგომარეობებს შორის , $L : S \to 2^{AP}$ და მარკირების ფუნქცია, რომელიც აფიქსირებს თითოეული მდგომარეობით დადგენილი ატომური წინადადებები , რომლებიც მართალი მდგომარეობაა.

გამოიყენება დროებითი ლოგიკარომელიც არის linear temporal logic დროებით ლოგიკა (LTL). მიდგომა დამადასტურებელი ავტონომიური კომპიუტერული პოლიტიკის აღწერილი ნაშრომის იყენებს LTL , რომელიც არის ლოგიკა , რომელიც დასძენს დროებით ოპერატორების ცხრილი 2 და გაანგარიშების მათთვის პირველი რიგის ლოგიკას.

◯φ	φ is true in the *next* moment in time
□φ	φ is true in *all* future moments
◇φ	φ is true in *some* future moment
φ $\mathcal{U}$ ψ	φ is true *until* ψ is true

ცხრილი 2. ძირითადი წესები ხაზოვანი დროებითი Logic (LTL)

LTL ფორმულა როგორიცაა φ და ψ ამ ცხრილში არის კომბინაცია ატომური წინადადებები , ლოგიკური ოპერატორები და LTL დროებით ოპერატორები, ◯ □ ◇ და U. იძლევა Kripke სტრუქტურას $M \models [](\Diamond a)$

$M = (S, S_0, R, L)$ და LTL ფორმულა φ , ნოტაცია გამოიყენება $M \models \phi$ რომ სისტემის მოდელი M აკმაყოფილებს LTL ფორმულას φ . მაგალითად, $a \in AP$ ატომური წინადადება ამბობს $M = (S, S_0, R, L)$ ომ, ეს ორი ტიპი საბოლოო ჯამში, მართალია და მისაწვდომია ავტონომიურ სისტემაში:

მიდგომა გამოიყენება ნაშრომის დამადასტურებელი ავტონომიური კომპიუტერული პოლიტიკა მოითხოვს, რომ ორი სახის ინფორმაცია ხელმისაწვდომია ავტონომიური სისტემა:

A performance model რომელიც განსაზღვრავს სისტემის პარამეტრებსა და განსაზღვრავს სტრუქტურის მოდელს და ასევე მათ შორისაც არსებულ სხვა პარამეტრებ თუ კი არსებობს .

ნაბიჯები ჩართული მშენებლობის Kripke სტრუქტურა და LTL ფორმულები ქვემოთ :

1. კომპლექტი სისტემის მდგომარეობის S და კომპლექტი თავდაპირველი სისტემის მდგომარეობის $S_0 \subseteq S$ არიან მიღებული ავტონომიური სისტემის სტრუქტურულ მოდელისაგან.

2.მარკირების ფუნქცია $R \subseteq S \times S$ მოპოვებული შესრულება მოდელი სისტემაში.

3.მდგომარეობისგადასვლის მიზეზი $R \subseteq S \times S$ არის განსაზღვრული ოპერაციის წესები (ანუ, სამოქმედო პოლიტიკა) სისტემა .

4. და ბოლოს, სისტემის შეზღუდვების განსაზღვრული მიზანი პოლიტიკა შეესაბამება LTL ფორმულები φ , რომ მოდელი M ყოველთვის უნდა დააკმაყოფილოს : M ⊨ □ φ . ანალოგიურად, საბოლოო სახელმწიფო პირობები გამოიხატება მიზანი პოლიტიკა შეესაბამება LTL ფორმულები y რომ მოდელი M უნდა " საბოლოოდ" დააკმაყოფილოს : M ⊨ ------ ψ

M ⊨ φ და M ⊨ ψ მტკიცებით მიღებული ნაბიჯი 4 მოწმდება გამოყენებით სტანდარტული LTL მოდელი კონტროლიორი. თუ ეს მტკიცება ჭეშმარიტია, მაშინ პოლიტიკა კომპლექტი არის უფასო კონფლიქტის , ანუ მისი სამოქმედო პოლიტიკით მიიღოს სისტემის ნებისმიერი საწყისიმდგომარეობის მოქმედი საბოლოო მდგომარეობა, გადასვლას მხოლოდ შუალედური აცხადებს, რომ დააკმაყოფილოს იმ ვარიანტების განსაზღვრული შეზღუდვა მიზანი პოლიტიკისა. იმ შემთხვევაში, თუ ერთი ან მეტი მტკიცება არ არის ჭეშმარიტი , პოლიტიკა კომპლექტი შეიცავს კონფლიქტებს. Counterexample მიერ გამომუშავებული მოდელის კონტროლი ყოველი ასეთი მტკიცება შეიძლება გამოყენებულ იქნას იდენტიფიცირების მიუწვდომელობის შემთხვევაშ, რომ არ შეესაბამება სისტემის ინვარიანტების ანდა მიღ $S_0 \subseteq S$ მელი საბოლოო მდგომარეობების , შესაბამისად, წარმოადგენს ამოსავალი წერტილის მოგვარების პოლიტიკის კონფლიქტს. საილუსტრაციოდ განაცხადის მოდელის შემოწმება კონფლიქტის ამოცნობა ავტონომიური კომპიუტერული პოლიტიკაა, მიგვაჩნია, რომ საქმის შესწავლა , რომელიც წარმოადგენს VMs მიგრაციის ზემოთ აღწერილია.

სტრუქტურული მოდელი. მონიტორინგი , კონტროლდება მონაცემთა ცენტრში შესაბამისი პარამეტრებით. ამ საქმის შესწავლით არიან:

1. *Structural model.* The monitored and controlled data-centre parameters relevant for this case study are:
2. Number of homogeneous servers within cluster = 6;
3. Capacity of a single server in cluster : $C_j^p = 3\,\text{GHz}$ (assumed as a unit of server capacity)

   $C_j^m = 8$ GB of RAM; j=1÷6
4. Number of clients' classes k=2;
5. Number of clients assigned to each server is randomly generated between 1 and 3, each client is randomly picked from one of two classes;
6. Portion of the i^{th} client's request served by the j^{th} server (host of a VM) α_{ij} is uniformly selected between 0 and 1;
7. Average request rates of the i^{th} client λ_i are chosen uniformly between 0.1 and 1 request per second (the minimum allowed inter-arrival time of the requests is specified in the SLA contract).
8. Processing capacity of the j^{th} server allocated to the i^{th} client's VM is calculated as $C_j^p \phi_{ij}$, where ϕ_{ij} is
9. the portion of processing resources of the j^{th} server that is allocated to the i^{th} client (assumed to be equal to 1/(number of clients) assigned to the server j^{th});
10. Service rate μ_{ij} of the i^{th} client on the j^{th} server (of capacity 1), that is, μ_{ij} is the service rate of the i^{th} client on the j^{th} server when a unit of processing capacity is allocated to the VM of this client; μ_{ij} are set based on the highest clock frequency for the servers.

To demonstrate the principal capabilities of model checking we use a simplified version of the above-mentioned model: just one client is assigned to each server (single VM runs on each

11. server, namely, VM7 in Service Component Ensemble 2 of datacenter 1 [2]), α_{ij}=1, ϕ_{ij}=1, λ=λ_{ij} is uniformly distributed between 0.1 and 1 request per second, $\mu = C_j^p \phi_{ij} \mu_{ij}$ is uniformly distributed between 1 and 3 request per second.

So, the response time (1) for the simplified case looks like $R_1 = 1/(\mu - \lambda)$ and the total response time will be equal to :

$$R = 1/(\mu - \lambda) + (1 - KS) * DT(LinkSpeed) \qquad (3)$$

For the structural model monitored and controlled datacenter parameters are:

- The request rate $x > 0$ (i.e $x = \lambda$)
- The service rate $y > 0$ (i.e $y = \mu$)
- The Kolmogorov complexity metric $z > 0$

We will assume that $0.1 \le x \le 1,\ 1 \le y \le 3,\ 0 \le z \le 1$

Performance model. There are "internal" parameters (i.e. parameters that are neither monitored nor controlled, but are calculated based on the parameters defined in the structural model):

Total response time

$$T = 1/(y-x) + (1-z)*DT$$

Goal policies. The average total processing time should eventually be in accordance with SLA requirements: $T \le T_{SLA}=4$sec.

Policy verification. To verify the correctness of this policy set, we need to construct the Kripke structure and to derive the LTL formulas associated with the structural and performance system models,

1) Constructing Kripke structure $M = (S, S_0, R, L)$ for the autonomic data centre. Each combination of values that can be taken by the monitored and controlled parameters of the system corresponds to a different state $s \in S$. The monitored parameters are x, y and z, so the set of states is

$$S \equiv \{ (x,y,z) \mid 0.1 \le x \le 1,\ 1 \le y \le 3,\ 0 \le z \le 1\}$$

The set of initial states $S_0 \equiv \{(x, y, z \mid 0.1 \le x \le 1,\ 1 \le y \le 2,\ 0 \le z \le 0.5\}$

Now we define the set of atomic propositions AP for the Kripke structure by including in AP atomic propositions representing the values of the configuration parameters for the system. We use the notation [X = a] as an atomic proposition stating that the value of the parameter X is a . In addition, we need to determine the truth values for the conditions representing the invariants $T \le T_{SLA}$ defined by the goal policies for the system. Therefore, we also include within AP atomic propositions of the form $[Y \le a]$, where Y is variable and a is a constant . As a result, the complete set of atomic propositions AP used to define the labelling function $L : S \rightarrow 2^{AP}$ consists of all atomic propositions:

$AP=\{[X=a] \mid X \in \{x, y, z, \}, a \ge 0\}, \cup \{[Y \le a] \mid Y \in \{ T \}, a \ge 0\}$

12. For example, given the state s =(0.1, 1.2 , 0.15), in which the server (host) of VM7 is receiving user requests at 0.1 transactions /second, service rate of the server is 1.2 services/second and Kolmogorov complexity metric (currently calculated for the server) is 0.15 , several atomic propositions that hold in state s are [x=0.1], [y=1.2] and [z=0.15].

The variation of metrics x, y and z must be encoded by transition relations

$$R_1 \equiv \{(s_j, s_k) \mid x(s_j) \neq x(s_k),\ y(s_j) \neq y(s_k),\ z(s_j) \neq z(s_k)\}$$

2) Derivir [] the LTL properties to be verified. The invariant "always T<4" maps to formula: $[T \le 4]$

The LTL properties derived so far were verified using the SPIN model checker. To improve the efficiency of the SPIN, the advantages of the SPIN's implementation of the state compression algorithm and of the partial-order reduction algorithm were used. The verification of the LTL properties was carried out for three system models characterized by different datacenter performance parameters:

a) Link Speed = 100Mbps

b) Link Speed = 1 Gbps
c) Link Speed = 10 Gbps

When the verification of the LTL properties was performed for scenario a), SPIN detected several possible constraint violations (T>4 sec). For each violation case, the corresponding counterexample traces were generated by SPIN. For scenarios b) and c) SPIN found no invalid states, thus confirming that the policy set is conflict free in these scenarios.

This case study demonstrates that a set of autonomic computing policies that is valid for one scenario can exhibit conflicts when the autonomic system operates in a different scenario. Model checking provides the unique capability to verify autonomic computing policies exhaustively and for the precise scenario in which the self-managing system operates.

დასკვნა და სამომავლო საქმიანობა

როგორც უკვე განვმარტე cloud computing არის ახალი მოდელი, რომლის კვლევაც ჩატარდა გამოვლინდა მისი დიზაინის და უსაფრთხოების მართვის სტრუქტურა, რომელიც გვაძლევს შასუალება გამოვავლინოთ კონფლიქტი AC's ინვარუმენტში KS ინტერპრეტაციის .ეს ინტერპრეტაცია გამოყენებული იქნა რომ გამოთვალოს საჭირო დროში downtime მიგრაციის შემთხვევაში AC ერთიდან სხვაზე .აქედან გამომდინარე SLA დარღვევები იქნება შესაბამისად გამოაშკარავებული. შიარდა ამისა, ეს ნაჩვენებია თეზისში , რომ KS შეიძლება გამოყენებულ იქნას string შეკუმშვის მექანიზმით და შესაბამისი მეთოდები jResp არის დამუშავებული.გარდა ამისა ჩვენ ვაგრძელებთ სტანდარტული ტოპოლოგიის დაძლევის თანამედროვე ინფრასტრუქტური მოთხოვნებს, მაგალითად, ნაკლოვანი ტოლერანტობის ტექნიკით pool რესურსის მიდგომას და ცოცხალ მიგრაციას VMs შორის წევრების კონსეტრაციას.პროტოტიპის მიმდინარეობას საშუალება ეძლევა მიმართოს რამოდენიმე სპეციპიკურ რაოდენობას , რომელიც ხელს შეუწყობს ხენი კვლევის მიზანს, ცოდნის საცავში , რომ არის არსებითი ნაწილი ნებისმიერ ავტონომიურ კომპონენტში,რომელიც ასახავს ბუნებრივიქცევისგანლაგებას VM და მის ურთიერთქმედებას დანარჩენ სხვა VMS - თან. აქედან გამომდინარე, ჩანს, რომ ავტონომიური კომპონენტი წარმოადგენს პერსპექტიულ მიდგომას, რათა მივაღწიოთ მაღალი დონის გამომსახველობა და

უსაფრთხოების დამატებითი თვითმმართველობის ცნობიერების შესაძლებლობების ახალი IT ტექნოლოგიების ინფრასტრუქტურა.

და ბოლოს, სამომავლო საქმიანობა დაკავშირებული იქნება ბიოლოგიური მიდგომის "ხელოვნური იმუნური სისტემის" და მისი ნიმუშის საფუძველზე ჩვენი მუშაობა. რომ არის, სიმულაცია, ისე, როგორ SCEL და მისი შემოგარენი შეიძლება მკაცრად შემუშავებული და რეკონსტრუქციის მოძიება საფრთხეების ადრეულ ეტაპზე, მიუხედავად იმისა, საფრთხე იქნება "შიგნით თუ გარეთ", და შესაბამისად მივიღოთ ადეკვატური რეაგირება.

პუბლიკაციები

დისერტაციის ძირითადი იდეები ასახულია შემდეგ პუბლიკაციებში:

Rodonaia, I., Rodonaia, V., & Mousa, M. (2013). Using of information theory metrics in security Modeling of Autonomic Cloud Computing. Transactions Automated Control Systems, Georgian Technical University No 1(14). 240-247.

Milnikov, A., Mousa, M., Rodonaia, I., & Rodonaia, V. (2013). A technique of formal security modeling in automatic cloud computing environment. Proceeding of the 11 International Conference on applied Electromagnetics, Wireless and Optical Communications (ELECTROSIENCE' 13)", June 25-27, Dubrovnik, Croatia. . 69-74.

Mousa, M. (2013). Modeling of cloud computing security using Kolmogorov Complexity,. AICT2013 Conference Organizing Committee, Application of Information and Communication Technologies,, (pp. 200-205). BAKU, AZARBIJAN, 24-28.

Mousa, M. (2013). Optimization of Autonomic Component Ensembles Resources. Journal of Technical Science & Technologies International Black Sea University, Volume 2, 2nd-issue., 41-44.

Mousa, M., Rodonaia, I., & Rodonaia, V. (2013). Security modeling of autonomic -component ensembles. Journal of Technical Scienc & Technologies International Black Sea University, 2(1), 15-20.

Mousa, M., Rodonaia, I., Rodonaia, V., Shonia, O., & Prangishvili, A. (2014). Formal verification in autonomic-component ensembles,. 5th International Conference on Circuits, Systems, Control, and Signals. Salerno, Italy. 57-61.