**Static web app:** [**https://thankful-hill-0a9297303.5.azurestaticapps.net**](https://thankful-hill-0a9297303.5.azurestaticapps.net)

**password: cloudservices**

**Α. Σενάρια στο Cloudsim**

**1.**

Από τον κώδικα φαίνεται πως θέλουμε να δημιουργήσουμε δύο datacenter.

*@SuppressWarnings("unused")*

*Datacenter datacenter0 = createDatacenter("Datacenter\_0");*

*@SuppressWarnings("unused")*

*Datacenter datacenter1 = createDatacenter("Datacenter\_1");*

Κατά την εκτέλεση, δημιουργούνται δύο datacenters .

*Datacenter\_0 is starting...*

*Datacenter\_1 is starting...*

Το cloudsim αναθέτει το ID 2 στο datacenter 0 και το ID 3 στο datacenter 1.

Στο τέλος απενεργοποιούνται τα 0 και 1,

*Datacenter\_0 is shutting down...*

*Datacenter\_1 is shutting down...*

Ακολουθεί απόσπασμα από debug που φαίνεται η λειτουργία των δύο datacenter,  
*========== OUTPUT ==========*

*Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time*

*35 SUCCESS 3 11 3 0,2 3,2*

*0 SUCCESS 2 0 4 0,2 4,2*

Σε κάθε center δημιουργούνται 2 hosts, ο host 0 και ο host 1. Ακολουθεί απόσπασμα αποτελέσματος,

0.1: Broker: VM #4 has been created in Datacenter #2, Host #0

0.1: Broker: VM #5 has been created in Datacenter #2, Host #1

0.2: Broker: VM #10 has been created in Datacenter #3, Host #0

0.2: Broker: VM #11 has been created in Datacenter #3, Host #1

**2.**

Βρίσκουμε την δημιουργία των Host στον κώδικα. Έχουμε 2 δύο λίστες για τους PE. Η μία λίστα έχει 4 PE(quad-core machine) και η δεύτερη έχει PE (dual-core machine). Κάθε επεξεργαστής έχει Mips =1000 όπως ορίζεται στον κώδικα ως σταθερά. Έπειτα, ακολουθούν τα υπόλοιπα χαρακτηριστικά ως σταθερές. Τέλος, δημιουργούνται οι hosts. Στον Host #0 αναθέτουμε τα quad-core machines και στον Host #1, τα dual-core machines.

Ο Host #0 έχει 4 PE, με mips=1000 ο καθένας, RAM = 2048, long storage = 1000000 και bandwidth = 10000.

Ο Host #1 έχει 2 PE, με mips=1000 ο καθένας. RAM = 2048, long storage = 1000000 και bandwidth = 10000.

**3.**

Χαρακτηριστικά datacenter: αρχιτεκτονική συστήματος, λειτουργικό σύστημα, τεχνολογία virtualization, time zone, κόστη bandwidth, επεξεργασίας, μνήμης και αποθήκευσης.

Η κλάση είναι η DatacenterCharacteristics,

*DatacenterCharacteristics characteristics = new DatacenterCharacteristics(arch, os, vmm, hostList, time\_zone, cost, costPerMem, costPerStorage, costPerBw);*

**4.**

Με τον κώδικα θέλουμε να δημιουργήσουμε είκοσι VMs,

*//Fourth step: Create VMs and Cloudlets and send them to broker*

*vmlist = createVM(brokerId,20); //creating 20 vms.*

Κάθε VM έχει τα εξής χαρακτηριστικά, όπως φαίνεται από τον κώδικα,

*//VM Parameters*

*long size = 10000; //image size (MB)*

*int ram = 512; //vm memory (MB)*

*int mips = 1000;*

*long bw = 1000;*

*int pesNumber = 1; //number of cpus*

**5.**  
Οι παράμετροι που καθορίζουν πόσες VM μπορούν να ανατεθούν είναι οι εξής: PE, ram, bandwidth, storage, mips που χρειάζεται η κάθε VM και πόσοι, συνολικά, πόροι είναι διαθέσιμοι για αυτές τις παραμέτρους, σε κάθε host.

Η κλάση VmAllocationPolicySimple διαλέγει, για host μίας VM, αυτόν που έχει τους λιγότερους PE σε λειτουργία. Για αυτό είναι μία Worst Fit πολιτική (αλγόριθμος), κάνοντας ανάθεση κάθε VM στον Host με τις περισσότερους διαθέσιμους PE.

**6.**

Το datacenter 0, με ID 2, προσπαθεί να δημιουργήσει είκοσι VM ( 0 έως 19 ) αλλά γίνεται επιτυχής ανάθεση μόνο των 0 έως 5 VM(σύνολο έξι VM) . Ο host #0 έχει τέσσερα VM και ο host #1 έχει δύο VM.

Συνεχίζει ο datacenter 1, με ID 3, να δημιουργεί τα υπόλοιπα VM ( 6 έως 19) αλλά και εδώ γίνεται επιτυχής ανάθεση μόνο των 6 έως 11 VM(σύνολο 6 VM). Ο host #0 έχει τέσσερα VM και ο host #1 έχει δύο VM.

Άρα, συνολικά δώδεκα VMs θα τρέξουν.

**Datacenter 0 ( ID #2 ), Host 0,**

0.1: Broker: VM #0 has been created in Datacenter #2, Host #0

0.1: Broker: VM #1 has been created in Datacenter #2, Host #0

0.1: Broker: VM #2 has been created in Datacenter #2, Host #0

0.1: Broker: VM #4 has been created in Datacenter #2, Host #0

**Datacenter 0 ( ID #2 ), Host 1**

0.1: Broker: VM #3 has been created in Datacenter #2, Host #1

0.1: Broker: VM #5 has been created in Datacenter #2, Host #1

**Datacenter 1 ( ID #3 ), Host 0**

0.2: Broker: VM #6 has been created in Datacenter #3, Host #0

0.2: Broker: VM #7 has been created in Datacenter #3, Host #0

0.2: Broker: VM #8 has been created in Datacenter #3, Host #0

0.2: Broker: VM #10 has been created in Datacenter #3, Host #0

**Datacenter 1 ( ID #3 ), Host 1**

0.2: Broker: VM #9 has been created in Datacenter #3, Host #1

0.2: Broker: VM #11 has been created in Datacenter #3, Host #1

Από το debug βλέπουμε πως δεν έγινε ανάθεση VM λόγω ram στον host #0 και λόγω mips στον host #1.

Απόσπασμα κώδικα,

*[VmScheduler.vmCreate] Allocation of VM #17 to Host #0 failed by RAM*

*[VmScheduler.vmCreate] Allocation of VM #17 to Host #1 failed by MIPS.*

Κάθε datacenter έχει δύο hosts. Στον host #0 δίνουμε τέσσερα PE και στον host #1, δύο.

Host #0 RAM / VM RAM = 2048/ 512 = μέχρι 4 VM.

Host #1 RAM / VM RAM = 2048/ 512 = μέχρι 4 VM.

Host #0 mips / VM mips = 4000/1000 = μέχρι 4 VM.

Host #1 mips / VM mips = 2000/1000 = μέχρι 2 VM.

Host storage / VM storage = 1000000/10000 = μέχρι 10 VM ο κάθε host.

Host bandwidth / VM bandwidth = 10000/ 1000 = μέχρι 10 VM ο κάθε host.

Στον host #0 φτάνουμε το όριο της RAM και αποτυγχάνει πρώτη. Στον host #1 έχουμε ακόμα RAM αλλά επειδή έχουμε φτάσει το όριο των mips , δεν έχουμε υπολογιστική ισχύ οπότε έχουμε αποτυχία MIPS.

Αν η κάθε VM είχε δύο PE(cpu), 250 MIPS kai 256 MB RAM, τι θα άλλαζε;

Host #0 RAM / VM RAM = 2048/ 256 = μέχρι 8 VM.

Host #1 RAM / VM RAM = 2048/ 256 = μέχρι 8 VM.

Host #0 mips / VM mips = 4000/ (2 \*250 ) = 4000/ 500 = μέχρι 8 VM.

Host #1 mips / VM mips = 2000/ (2 \*250 ) = 2000/ 500 = μέχρι 4 VM.

Απάντηση: Θα διπλασιάζαμε τις VM σε κάθε host. Ελέγχοντας το output μετά από εκτέλεση, παρατηρούμε πως, στο datacenter #2, δημιουργούνται 8 VM στον host #0 και 4 VM στον host #1, συνολικά 12 VM. Στο datacenter #3 δημιουργούνται 5 VM στον host #0 και 3 VM στον host #1, συνολικά 8 VM. Oπότε, δημιουργούνται και οι 20 VM.

**7.**

Δημιουργούνται σαράντα cloudlets.

*cloudletList = createCloudlet(brokerId,40); // creating 40 cloudlets*

Χαρακτηριστικά cloudlets,

*long length = 1000;* //Το αρχείο μεταφέρει την υπολογιστική δύναμη που χρειάζεται για να εκτελεσθεί το cloudlet στην VM

*long fileSize = 300;* //Το αρχείο μεταφέρει δεδομένα, προγράμματα κ.λ.π. που θα εκτελεσθούν στην VM

*long outputSize = 300;* //Το αρχείο αυτό περιλαμβάνει τα αποτελέσματα της εκτέλεσης του cloudlet

*int pesNumber = 1;* //Πόσα PE πρέπει να χρησιμοποιήσει το cloudlet για να εκτελεί παράλληλες διεργασίες

**8.**

Η κλάση DatacenterBroker είναι υπεύθυνη για την ανάθεση των cloudlets σε VMs.

Ο κώδικας που εκτελεί την ανάθεση των cloudlet σε VM είναι ο παρακάτω,

broker.submitVmList(*vmlist*); //ενημέρωση του broker για τις διαθέσιμες VMs (διαθέσιμοι πόροι για τα cloudlets)

*broker.submitCloudletList(cloudletList);* //ενημέρωση του broker για τα cloudlets ώστε να γίνει η ανάθεση τους

Η ανάθεση είναι απλή και γίνεται με την σειρά εμφάνισης στις λίστες vmlist και cloudletList, όπως αυτό φαίνεται στη μέθοδο **protected** **void** submitCloudlets(), της κλάσης DatacenterBroker.java. Ο broker βρίσκει την πρώτη διαθέσιμη VM από την vmlist, ψάχνει αν έχει γίνει ανάθεση στο cloudlet της cloudletList και αν δεν έχει γίνει, το αναθέτει και προχωράει στις επόμενες θέσεις των λιστών.

Απόσπασμα κώδικα

|  |  |  |  |
| --- | --- | --- | --- |
| 0.2: Broker: Sending cloudlet 0 to VM #0  0.2: Broker: Sending cloudlet 1 to VM #1  0.2: Broker: Sending cloudlet 2 to VM #2  0.2: Broker: Sending cloudlet 3 to VM #3  0.2: Broker: Sending cloudlet 4 to VM #4  0.2: Broker: Sending cloudlet 5 to VM #5  0.2: Broker: Sending cloudlet 6 to VM #6  0.2: Broker: Sending cloudlet 7 to VM #7  0.2: Broker: Sending cloudlet 8 to VM #8  0.2: Broker: Sending cloudlet 9 to VM #9  0.2: Broker: Sending cloudlet 10 to VM #10  0.2: Broker: Sending cloudlet 11 to VM #11 | 0.2: Broker: Sending cloudlet 12 to VM #0  0.2: Broker: Sending cloudlet 13 to VM #1  0.2: Broker: Sending cloudlet 14 to VM #2  0.2: Broker: Sending cloudlet 15 to VM #3  0.2: Broker: Sending cloudlet 16 to VM #4  0.2: Broker: Sending cloudlet 17 to VM #5  0.2: Broker: Sending cloudlet 18 to VM #6  0.2: Broker: Sending cloudlet 19 to VM #7  0.2: Broker: Sending cloudlet 20 to VM #8  0.2: Broker: Sending cloudlet 21 to VM #9  0.2: Broker: Sending cloudlet 22 to VM #10  0.2: Broker: Sending cloudlet 23 to VM #11 | 0.2: Broker: Sending cloudlet 24 to VM #0  0.2: Broker: Sending cloudlet 25 to VM #1  0.2: Broker: Sending cloudlet 26 to VM #2  0.2: Broker: Sending cloudlet 27 to VM #3  0.2: Broker: Sending cloudlet 28 to VM #4  0.2: Broker: Sending cloudlet 29 to VM #5  0.2: Broker: Sending cloudlet 30 to VM #6  0.2: Broker: Sending cloudlet 31 to VM #7  0.2: Broker: Sending cloudlet 32 to VM #8  0.2: Broker: Sending cloudlet 33 to VM #9  0.2: Broker: Sending cloudlet 34 to VM #10  0.2: Broker: Sending cloudlet 35 to VM #11 | 0.2: Broker: Sending cloudlet 36 to VM #0  0.2: Broker: Sending cloudlet 37 to VM #1  0.2: Broker: Sending cloudlet 38 to VM #2  0.2: Broker: Sending cloudlet 39 to VM #3 |

Θα εκτελεστούν συνολικά και τα 40 cloudlets.

40 cloudlets /12 VMs = 3 γύροι ανάθεσης στα VM από 0-11. Απομένουν 4 cloudlet. Ξεκινάει 4ος γύρος ανάθεσης στα VM 0-3.

|  |  |
| --- | --- |
| VM | Cloudlets |
| 0 | 4 |
| 1 | 4 |
| 2 | 4 |
| 3 | 4 |
| 4 | 3 |
| 5 | 3 |
| 6 | 3 |
| 7 | 3 |
| 8 | 3 |
| 9 | 3 |
| 10 | 3 |
| 11 | 3 |

**9.**

Η πολιτική που ακολουθείται είναι η time sharing και στα cloudlet και στα PE των VM.

1. **Cloudlet πολιτική**

//create VMs

Vm[] vm = new Vm[vms];

for(int i=0;i<vms;i++){

vm[i] = new Vm(i, userId, mips, pesNumber, ram, bw, size, vmm, new **CloudletSchedulerTimeShared()**);

list.add(vm[i]);

}

1. **VM πολιτική**

Host #0

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList1,

new **VmSchedulerTimeShared**(peList1)

)

);

Host #1

hostList.add(

new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList2,

new **VmSchedulerTimeShared**(peList2)

)

);

**10.**

**Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time**

* Το cloudlet ID είναι το αναγνωριστικό του cloudlet που εκτελείται.
* Status SUCCESS σημαίνει πως πέτυχε η εκτέλεση του cloudlet.
* DATACENTER ID είναι σε ποιο Datacenter ανήκει το VM και κατ’ επέκταση το cloudlet που εκτελέστηκε.
* VM ID είναι το VM το οποίο τρέχει όταν εκτελείται ένα cloudlet.
* Time είναι ο χρόνος εκτέλεσης ενός cloudlet.
* Start time, πότε ξεκίνησε το cloudlet.
* Finish time πότε τελείωσε το cloudlet.

Όλα τα cloudlet ξεκίνησαν ταυτόχρονα.

VMs 0-3

4 cloudlets per VM \* 4 Vms = 16 cloudlets, κάθε cloudlet 4 sec.

VMs 4-11

3 cloudlets per VM \* 8 Vms = 24 cloudlets, κάθε cloudlet 3 sec.

Για αρχή, υποθέτουμε πως όλες οι VMs έχουν 3 cloudlets με χρόνο εκτέλεσης 3 sec. Άρα όλα τα cloudlets εκτελέστηκαν μέσα σε 3 sec σε κάθε VM. Πραγματικός χρόνος εκτέλεσης ενός cloudlet, 3cloudlet per VM/ 3 sec = 1 sec. Επειδή υπάρχει χρονικός διαμοιρασμός, όλα τελειώσανε στα 3 sec καθώς δεν εκτελέστηκαν με την σειρά, στο οποίο θα είχαμε ίδιο χρόνο εκτέλεσης (1 sec), αλλά διαφορετικό χρόνο εκκίνησης και τερματισμού κάθε cloudlet .Οπότε, 3 cloudlet/ 3 sec = 1 sec κάθε cloudlet.

Εκτέλεση των cloudlets σε σειρά

|  |  |  |
| --- | --- | --- |
| **Time** | **Start time** | **Finish time** |
| 1 | 0.2 | 1.2 |
| 1 | 1.2 | 2.2 |
| 1 | 2.2 | 3.2 |

Προσθέτοντας τον χρόνο αν ήταν σε σειρά, μας δίνει 3 sec. Άρα, στην περίπτωση μας, διαμοιράστηκαν αυτόν τον συνολικό χρόνο των 3 sec, άργησε η ολοκλήρωση της εκτέλεση τους κατά 2 sec, ώστε να λειτουργήσουν ταυτόχρονα.

Εκτέλεση των cloudlets παράλληλα (χρονικός διαμοιρασμός)

|  |  |  |
| --- | --- | --- |
| **Time** | **Start time** | **Finish time** |
| 3 | 0.2 | 3.2 |
| 3 | 0.2 | 3.2 |
| 3 | 0.2 | 3.2 |

Αφού βρήκαμε πως κάθε cloudlet διαρκεί 3 sec, με πραγματικό χρόνος εκτέλεσης το 1 sec, επιστρέφουμε στο παράδειγμα μας. Τα παραπάνω ισχύουν για τις VMs 4-11. Οι VMs 0-3 έχουν από 4 cloudlet, 1 cloudlet διαμοιρασμένο παραπάνω. Ακολουθώντας την προηγούμενη λογική, εάν αφαιρέσουμε τα 3 sec, θα έχουμε 4 sec – 3sec = 1 sec για τα εναπομείναντα cloudlet των VMs 0-3, το οποίο συνάδει.

Εκτέλεση των cloudlets σε σειρά

|  |  |  |
| --- | --- | --- |
| **Time** | **Start time** | **Finish time** |
| 1 | 0.2 | 1.2 |
| 1 | 1.2 | 2.2 |
| 1 | 2.2 | 3.2 |
| 1 | 3.2 | 4.2 |

Προσθέτοντας τον χρόνο, αν ήταν σε σειρά, μας δίνει 4 sec. Άρα, στην περίπτωση μας, διαμοιράστηκαν αυτόν τον συνολικό χρόνο των 4 sec, άργησε η ολοκλήρωση της εκτέλεση τους κατά 3 sec, ώστε να λειτουργήσουν ταυτόχρονα.

Εκτέλεση των cloudlets παράλληλα (χρονικός διαμοιρασμός)

|  |  |  |
| --- | --- | --- |
| **Time** | **Start time** | **Finish time** |
| 4 | 0.2 | 4.2 |
| 4 | 0.2 | 4.2 |
| 4 | 0.2 | 4.2 |
| 4 | 0.2 | 4.2 |

Γίνεται αντιληπτό, πως υπάρχει χρονικός διαμοιρασμός round-robin στις VMs. Τα 4-11 VMs τελειώσανε πιο γρήγορα, λόγω λιγότερων cloudlets και συνέχισε ο διαμοιρασμός στα 0-3 VMS.

**Συμπέρασμα:** Όλα τα cloudlets έχουν πραγματικό χρόνο εκτέλεσης το 1 sec και διαμοιρασμένο χρόνο εκτέλεσης όσο το άθροισμα του χρόνου εκτέλεσης όλων των cloudlets σε κάθε VM. Χρόνος αποπεράτωσης του συστήματος είναι τα 4 sec που τελειώνουν τα τελευταία cloudlets. Επιβεβαιώνεται η time shared policy αφού τα cloudlet χρησιμοποιούν τους πόρους με βάση ένα χρονικό διάστημα που τους γίνεται διαθέσιμο.

**Α.**

**1.Αλλαγή των cloudlets σε space sharing και των PE (των VM) σε space sharing**

Ο χρόνος εκτέλεσης του κάθε cloudlet είναι 1 sec. Οι χρόνοι εκκίνησης και τερματισμού αλλάζουν.

Εκτέλεση σε space sharing (σε σειρά)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **VMs** | **Cloudlets** | **Time** | **Start time** | **Finish time** | **Κάθε VM έχει δικούς της, αποκλειστικούς πόρους** |
| 0-11 | 0-11 | 1 | 0.2 | 1.2 | 12 VMs |
| 0-11 | 12-23 | 1 | 1.2 | 2.2 | 12 VMs |
| 0-11 | 24-35 | 1 | 2.2 | 3.2 | 12 VMs |
| 0-3 | 36-39 | 1 | 3.2 | 4.2 | 4 VMs |

Η εκτέλεση των cloudlets γίνεται με σειριακή εκτέλεση ομάδων cloudlets. Βέβαια, κάθε ομάδα εκτελείται παράλληλα χωρίς όμως να υπάρχει διαμοιρασμός αφού τα VMs έχουν αποκλειστικούς πόρους και για αυτό όλα τα cloudlets εκτελούνται μέσα σε 1 sec.

Ο συνολικός χρόνος εκτέλεσης του συστήματος παραμένει ο ίδιος και στις δύο πολιτικές, time και space sharing, δηλαδή 4 sec.

**2.Αλλαγή των cloudlets σε space sharing**

Η εκτέλεση έχει το ίδιο αποτέλεσμα με την περίπτωση 1 (cloudlets σε space sharing και των PE (των VM) σε space sharing).

**3.Αλλαγή των PE (των VM) σε space sharing**

Η εκτέλεση έχει ίδιο αποτέλεσμα με το αρχικό παράδειγμα, δηλαδή τα cloudlets να είναι σε time sharing.

**Συμπέρασμα:** Μόνο η πολιτική των cloudlets επηρεάζει το αποτέλεσμα, ανεξαρτήτως αν τα VMs έχουν time sharing ή space sharing. Στο time sharing δεσμεύεται η επεξεργαστική ισχύς ανά χρονικά διαστήματα, τα οποία χρησιμοποιεί κάθε VM από λίγο, δημιουργώντας έναν “ενιαίο” πόρο. Στο space sharing, ο “ενιαίος” πόρος είναι τόσος, όσο οι ταυτόχρονες VMs που βρίσκονται σε λειτουργία. Στο παράδειγμα μας, και στις δύο περιπτώσεις, απαιτούνται οι ίδιοι πόροι από τις VMs, οπότε δεν έχει σημασία πως δεσμεύονται οι πόροι.

**Παράδειγμα VM sharing**

Κάθε cloudlet χρησιμοποιεί 1000 mips για να τρέξει μέσα σε 1 sec. Τα 40 cloudlets θα χρειαστούν 40.000 mips. Πρέπει να δεσμεύσουμε 40.000 mips από τo σύστημα (VMs).

Κάθε VM έχει host #0 με 4000 mips(4 PE) Και host #1 με 2000 mips(2 PE).

Time sharing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **VMs** | **Cloudlets** | **MIPS** | **Time** | **Total execution time** |
| 0-3 | 4 ανά VM | Χρονικός διαμοιρασμός 4.000 ανά VM \* 4 VM | 4 sec |  |
| 4-11 | 3 ανά VM | Χρονικός διαμοιρασμός 3.000 ανά VM \* 8 VM | 3 sec |  |
| **Σύνολο** 40.000 mips με χρονικό διαμοιρασμό των VMs | | | | 4 sec |

Space sharing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **VMs** | **Cloudlets** | **MIPS** | **Total time** | **Total execution time** |
| 0-11 | 0-11 | 12.000 mips με 1000 ανά VM | 1 sec |  |
| 0-11 | 12-23 | 12.000 mips με 1000 ανά VM | 1 sec |  |
| 0-11 | 24-35 | 12.000 mips με 1000 ανά VM | 1 sec |  |
| 0-3 | 36-39 | 4.000 mips με 1000 ανά VM | 1 sec |  |
| **Σύνολο** 40.000 mips με ακολουθιακό διαμοιρασμό των Vms | | | | 4 sec |

Επιβεβαιώνεται το συμπέρασμα, η πολιτική των PE των VM, δεν επηρεάζει το αποτέλεσμα.

Cloudlets

1. Στο time shared, τα cloudlets θα δεσμεύσουν **χρονικά**, τους πόρους, για παράλληλη εκτέλεση.
2. Στο space shared, για να μην υπάρξει συμφόρηση, πρέπει να τελειώσει κάθε cloudlet πριν αναλάβει το επόμενο στην σειρά. Τα cloudlets θα δεσμεύσουν, **ομαδικά και ακολουθιακά**, ένα την φορά σε κάθε VM. Για αυτό ομαδοποιούνται ανά τον αριθμό των VM.

Η απόδοση παραμένει ίδια και στις δύο περιπτώσεις.

**Αλλαγή των VM σε time shared oversubscription**

Ανατίθενται οχτώ VM στο Datacenter #2, τέσσερα VM σε κάθε host.

Ανατίθενται οχτώ VM στο Datacenter #3, τέσσερα VM σε κάθε host.

Παρατηρούμε πως η πολιτική αυτή, επέτρεψε να δημιουργηθούν 2 παραπάνω VM στον host #1 του κάθε Datacenter. Στο time shared και space shared είχαμε μόνο δύο VM λόγω περιορισμού των PE. Στο oversubscription μας επιτρέπεται να γίνεται κατανομή VM ακόμα και αν ξεπερνάει την CPU χωρητικότητα.

Output

Όλα τα cloudlet ξεκινάνε ταυτόχρονα. Διαφέρει ο χρόνος εκτέλεσης του κάθε cloudlet και ο συνολικός χρόνος εκτέλεσης. Το Datacenter #3 χρησιμοποιεί κάθε φορά 4 VM με 2 cloudlets, ενώ το Datacenter #2, χρησιμοποιεί κάθε φορά 4 VM με 3 cloudlets.

Χρόνοι εκτέλεσης

Datacenter #3 = 2 μονάδες χρόνου (8 από τα cloudlets και 4 VMs)

Datacenter #2 = 3 μονάδες χρόνου (12 από τα cloudlets και 4 VMs)

Datacenter #3 = 4 μονάδες χρόνου (8 από τα cloudlets και 4 VMs)

Datacenter #2 = 6 μονάδες χρόνου (12 από τα cloudlets και 4 VMs)

Παρατηρούμε πως υπάρχει σταδιακή επιδείνωση της απόδοσης. Στο time shared και space shared είχαμε χρόνο 4 μονάδων του χρόνου για να εκτελεστούν όλα. Στο time shared oversubscription έχουμε 6, δηλαδή 2 μονάδες του χρόνου πιο αργά τελείωσαν όλα τα cloudlets.

**B. Εκτέλεση όλων των VM χωρίς oversubscription**

**α) Χωρίς αύξηση των Host και των PE, μόνο χαρακτηριστικά**

* Κάθε host έχει 1000 mips, 2048 ram, storage 1000000, bw 10000.
* Κάθε Vm έχει 1000 mips, 512 ram, storage 10000, bw 1000.

Δε χρειάζεται να αλλάξουμε το storage και το bandwidth. Εξάλλου, το Debug έδειξε πως η ανάθεση έχει πρόβλημα είτε στη ram είτε στις mips.

Με 4 PE host #0 και 2 PE host #1, έχουμε 6000 mips διαθέσιμα. Αυτό το set up μας δίνει 6 VM στο κάθε Datacenter. Αν διπλασιάσουμε τα χαρακτηριστικά τους, θα έχουμε τις διπλάσιες VM.

Βάζουμε 2000 mips per PE και συνολικά 4096 RAM στο Datacenter. Έτσι, κατά την εκτέλεση, ανατίθενται 12 VM στο Datacenter #2 και 8 VM στο Datacenter #3. Συνολικά 20 VMs.

Όλα τα cloudlets εκτελούνται ταυτόχρονα, με χρόνο εκτέλεσης κάθε cloudlet, 2 μονάδες του χρόνου. Συνολικός χρόνος εκτέλεσης όλων των cloudlets, είναι 2 μονάδες του χρόνου. Ο λόγος της διαφοράς είναι εμφανής, αν και τα VM είναι περισσότερα, έχουμε περισσότερα mips, τα οποία βοηθάνε στην γρήγορη εκτέλεση των διεργασιών.

**β)** **Αύξηση των PE, χωρίς αύξηση των host**

Αφού 6 PE δίνουν 6 VM, θα κάνουμε και τον host #1, τετραπύρηνο. Με 8 PE, θα έχουμε θεωρητικά 8 VM. Άρα δε μας φτάνουν δύο τετραπύρηνοι hosts, θέλουμε παραπάνω PE. Θέλουμε ακόμα δύο PE, δηλαδή συνολικά 10 VM για να τρέξουν σε κάθε Datacenter, συνολικά 20 VMs.

Όμως εδώ, δημιουργείται ένα άλλο πρόβλημα, αυτό της RAM. H RAM δεν αλλάζει, παραμένει σταθερή, οπότε θα τελειώσει ο πόρος της μνήμης, ασχέτως πόσα PE θα προσθέσουμε. Άρα άτοπο.

Δε γίνεται να δημιουργήσουμε 20 VM, προσθέτοντας μόνο PE.

γ) **Αύξηση των hosts**

Έχουμε δύο Hosts. Αν προσθέσουμε έναν τετραπύρηνο host, θα αποκτήσουμε ακόμα 4 VM.

1. Host #0 4 PE = 4000 mips, 2048 RAM => 4 VM
2. Host #1 4 PE = 4000 mips, 2048 RAM => 4 VM
3. Host #2 2 PE = 2000 mips, 2048 RAM => 2 VM

10 VMs ανά Datacenter. Δύο Datacenter \* 10 VM = 20 VM συνολικά.

Όλα τα cloudlets εκτελούνται ταυτόχρονα, με χρόνο εκτέλεσης κάθε cloudlet, 2 μονάδες του χρόνου. Συνολικός χρόνος εκτέλεσης όλων των cloudlets, είναι 2 μονάδες του χρόνου. Σαφής η διαφορά από το αρχικό παράδειγμα, με διαφορά 2 μονάδες του χρόνου.

Ένα Datacenter έχει,

Συνολικοί πόροι με 3 hosts: 10 PE \* 1000mips =10.000 mips και 6.144 RAM.

10 VMS => 10.000 mips και 5012 RAM.

Συνολικοί πόροι με 2 hosts: 6 PE \* 1000mips =6.000 mips και 4.096 RAM.

6 VMS => 6.000 mips και 3072 RAM.

Οπότε τα cloudlets, με τις 4 VM που προσθέσαμε, διαμοιράζονται 4.000 mips παραπάνω, για αυτό και η εκτέλεση είναι πιο γρήγορη.

**C. Υλοποίηση αλλαγής πολιτικής**

**α) Αλλαγή πολιτικής στα 5 και 8**

5 - Ανάθεση VM σε hosts

Φτιάχνουμε μία νέα κλάση κάνοντας implement το interface VmAllocationPolicy. H κλάση VmAllocationPolicySimple είναι implementation του interface VmAllocationPolicy. Ας ονομάσουμε την κλάση μας VmAllocationPolicyMyOwnPolicy.

Αλλάζουμε τον κώδικα ως εξής,

*// 6. Finally, we need to create a PowerDatacenter object.*

*Datacenter datacenter =* ***null****;*

***try*** *{*

*datacenter =* ***new*** *Datacenter(name, characteristics,* ***new*** *VmAllocationPolicyMyOwnPolicy(hostList), storageList, 0);*

*}* ***catch*** *(Exception e) {*

*e.printStackTrace();*

*}*

***Return*** *datacenter;*

8 - Ανάθεση cloudlets σε VM

Μπορούμε να κάνουμε implement την κλάση DatacenterBroker σε μία δική μας κλάση και κάνοντας override τις μεθόδους submitVmlist() και submitCloudletList().

**β) Αλλαγή πολιτικής στα 9a και 9b**

9a - αλλαγή πολιτικής δρομολόγησης cloudlet σε VM (time,space, etc)

Φτιάχνουμε μία νέα κλάση κάνοντας implement την κλάση CloudletScheduler. H κλάση CloudletSchedulerTimeShared και η κλάση CloudletSchedulerSpaceShared είναι implementations της κλάσης κλάση CloudletScheduler. Ας ονομάσουμε την κλάση μας CloudletSchedulerMyPolicyShared.

Αλλάζουμε τον κώδικα ως εξής,

*vm[i] =* ***new*** *Vm(i, userId, mips, pesNumber, ram, bw, size, vmm,* ***new*** *CloudletSchedulerMyPolicyShared ());*

9b - αλλαγή πολιτικής ανάθεσης PE σε VM (time,space, etc)

Φτιάχνουμε μία νέα κλάση κάνοντας implement την κλάση SchedulerVm. H κλάση VmSchedulerTimeShared και η κλάση VmSchedulerSpaceShared είναι implementations της κλάσης κλάση VmScheduler. Ας ονομάσουμε την κλάση μας VMSchedulerMyPolicyShared.

Αλλάζουμε τον κώδικα ως εξής,

***new***VMSchedulerMyPolicyShared *(peList1)*

***new***VMSchedulerMyPolicyShared *(peList2)*

**C) Αλλαγή πολιτικής με μία της προτίμησής μας**

Η VmAllocationPolicySimple διαλέγει για host ενός VM, τον Host με τις περισσότερες διαθέσιμες PE. Μπορούμε, με βάση αυτή την πολιτική και μερική τροποποίηση της, να την αλλάξουμε σε VmAllocationBestFit. Στην BestFit διαλέγουμε τον host με τις λιγότερες διαθέσιμες PE, που μπορεί όμως να δεχθεί την VM. Αν και στο παράδειγμα μας, είναι καλύτερος ο Simple, λόγω του ότι είναι πιο απλή πολιτική και όλες οι VM είναι ίδιες, όμως σε περίπτωση που έχουμε διαφορετικές, τότε θα πρέπει να βρούμε μία πολιτική που να εφαρμόσει καλύτερο balancing στους πόρους.

Το αρχείο simple το αφήνουμε σχεδόν ίδιο. Η κύρια αλλαγή είναι που τροποποιούμε την allocateForVmHost() μέθοδο ώστε να ελέγχει όχι τους περισσότερους διαθέσιμους PE αλλά τους λιγότερους PE που να χωράνε το VM.

Στο org.cloubus.clousim, βάζουμε το VmAllocationPolicyBestFit.java, δηλαδή εκεί που βρίσκονται τα VmAllocationPolicySimple.java και VmAllocationPolicy.java αρχεία.

Κάνουμε import το αρχείο στο CloudSimExample6,

***import*** *org.cloudbus.cloudsim.VmAllocationPolicyBestFit;*

Αλλάζουμε την πολιτική όταν δημιουργούμε το Datacenter,

*Datacenter datacenter =* ***null****;*

***try*** *{*

*datacenter =* ***new*** *Datacenter(name, characteristics,* ***new*** *VmAllocationPolicyBestFit(hostList), storageList, 0);*

*}* ***catch*** *(Exception e) {*

*e.printStackTrace();*

*}*

***return*** *datacenter;*