Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej

Implementacja i symulacyjna ocena efektywności i skalowalności energooszczędnego systemu komputerowego dla centrum przetwarzania danych

Piotr Arabas, Michał Karpowicz, Ewa Niewiadomska-Szynkiewicz

Badanie są wspierane przez Narodowe Centrum Nauki.
Projekt badawczy nr 2015/17/B/ST6/01885.

20 września 2018

Warszawa 2017

Spis treści

1 Wstęp

Istotnym elementem wpływającym na jakość działania wieloprocesorowych systemów obliczeniowych takich jak klastry jest mechanizm alokacji zasobów. W klasycznych podejściach dąży się aby mechanizm ten zawierał algorytmy równoważące obciążenie poszczególnych elementów obliczeniowych co sprzyja utrzymaniu jakości obsługi (QoS) zleconych zadań na możliwie wysokim poziomie. W ostatnich czasach nastąpił istotny rozwój technologii umożliwiający budowę centrów obliczeniowych o niespotykanych wcześniej mocach przetwarzania. Niestety ciągły wzrost możliwości obliczeniowych wymaga też coraz większej ilości energii, którą konsumują maszyny obliczeniowe i łączące je urządzenia sieciowe. Ocenia się przy tym, że mimo wzrastającej wydajności energetycznej pobór mocy przez centra obliczeniowe będzie rósł, gdyż zapotrzebowanie na moc obliczeniową rośnie ciągle szybciej.

Konieczność oszczędzania energii wynika przy tym nie tylko z oczywistych względów ekonomicznych i ekologicznych, lecz również technicznych. Doprowadzenie zasilania o coraz większej mocy a także odprowadzenie coraz większej ilości powstającego ciepła jest nie tylko kosztowne, ale również kłopotliwe, wymaga coraz bardziej skomplikowanych rozwiązań.

W tej sytuacji rozwinięto osobną gałąź systemów alokacji zasobów wyposażonych w energooszczędne algorytmy. Celem niniejszego raportu jest sprawdzenie, na drodze symulacji w symulatorze klastra obliczeniowego CloudSim, skuteczności wybranych algorytmów. W następnym rozdziale zostaną przedstawione podstawowe algorytmy wykorzystywane do energooszczędnej alokacji zasobów. Następnie, w rozdziale 3, zostanie wprowadzony symulator CloudSim i opisany scenariusz testowy. Rozdział 4 zwiera opis przeprowadzonych eksperymentów włącznie z wynikami, zaś rozdział 5 podsumowanie i wnioski.

2 Dynamiczna alokacja maszyn wirtualnych

Energooszczędne algorytmy dynamicznej alokacji maszyn wirtualnych sprowadzają się do iteracyjnego powtarzania kolejnych trzech kroków:

- **wybór hosta źródłowego:** wybranie maszyn fizycznych, z których należy przenieść maszyny wirtualne.
- **wybór maszyny wirtualnej do migracji:** wybór z wskazanych poprzednio węzłów jednej lub większej liczby maszyn wirtualnych w celu dokonania ich migracji na inną maszynę fizyczną.
- **wybór hosta docelowego:** wybór hosta docelowego, na którym zostaną umieszczone wskazane wcześniej maszyny wirtualne.

Liczne warianty rozwiązania wymienionych wyżej problemów znane są z literatury. W szczególności Bieloglazov i współpracownicy proponują w [1] rozwiązanie zadania umiejscowienie maszyny wirtualnej na fizycznym węźle docelowym i wybór maszyny wirtualnej do migracji, zaś wybór hosta źródłowego opisują w [2]. Inne algorytmy proponuje m.in. Raycroft [8]. Bardziej złożone struktury sterowania chmura obliczeniową proponuje Diaconescu [4] oraz Kumar i Raghunathan [7], którzy proponują uwzględnienie sterowania stanami energetycznymi procesora. Próby znalezienia globalnego rozwiązania problemu alokacji prowadzą zazwyczaj do sformułowania złożonych zadań programowania matematycznego będących np. wariantami problemu plecakowego. Ze względu na swą złożoność bywają one często rozwiązywane metodami ewolucyjnymi [10], [6].

2.1 Identyfikacja obciążonych maszyn

2.1.1 Static threshold

Algorytm Static Threshold sprowadza się do porównywania bieżącego obciążenia hosta z predefiniowanym, stałym progiem, po którego przekroczeniu host jest uznawany za przeciążony a maszyny wirtualnych są przenoszone w celu zmniejszenia jego obciążenia.

2.1.2 Median Absolute Deviation

Poprzednio opisany algorytm nie sprawdza się przy zmiennym i nieprzewidywalnym obciążeniu, gdy różne typy aplikacji dzielą ten sam fizyczny zasób. W związku z tym Beloglazov w artykule [1] zaproponował algorytm Median Absolute Deviation opierający się na analizie historii obciążenia:

$$MAD = median(X_i - median_j(X_j) \tag{1}$$

gdzie X_i i X_j są przeszłymi pomiarami stopnia obciążenia hosta. Inaczej mówiąc MAD jest równe medianie zbioru różnic pomiędzy i-tym pomiarem wykorzystania hosta a jego medianą, przy czym próg górny definiowany jest jako:

$$T_u = 1 - s * MAD \tag{2}$$

gdzie T_u jest wartością progu górnego a s jest parametrem wpływającym na tendencję algorytmu do konsolidacji maszyn wirtualnych. Im większy parametr tym niższy próg, a co za tym idzie większa liczba migracji.

2.1.3 Interquartile Range

Algorytm Interquartile Range (IQR) korzysta z miary rozproszenia statystycznego równej różnicy pomiędzy pierwszym a trzecim kwartylem (kwartyle liczone są na podstawie danych historycznych)

$$IQR = Q_1 - Q_3 \tag{3}$$

gdzie IQR jest to parametr Interquartile Range a Q_1 i Q_3 są odpowiednio pierwszym i trzecim kwartylem. Na tej samej zasadzie co we wcześniejszym algorytmie wyznaczany jest próg górny:

$$T_u = 1 - s * IQR \tag{4}$$

gdzie s jest parametrem pełniącym tę samą funkcję co w przypadku algorytmu MAD.

2.2 Metody wyboru VM do migracji

Po wykryciu przeciążenia hosta konieczne jest wybranie konkretnej VM w celu jej przeniesienia. Algorytmy te działają iteracyjnie, to znaczy wybierają one jedną maszynę wirtualną do migracji a następnie sprawdzają czy host jest w dalszym ciągu nadmiernie obciązony. Jeżeli tak, to polityka jest po raz kolejny uruchamiana.

Algorytmy można podzielić na dwie gupy: pierwszą z nich jest klastrowany wybór maszyn wirtualnych. Algorytmy należące do tej grupy wychodzą z założenia, że złożone aplikacje posiadają wiele warstw, do których są przypisane oddzielne maszyny wirtualne. Przykładem mogą być instancje bazy danych uruchomione na szeregu maszyn, wspierane przez load balancer alokowany na innej, i wykorzystywane przez serwer aplikacyjny działający na kolejnej maszynie. W związku z tym przeniesienie tylko części maszyn wirtualnych na których znajduje się taka złożona aplikacja mogło by wpłynąć na znaczną degeneracje jej wydajności. Aby tego uniknąć algorytmy te starają się przenosić całe grupy maszyn wirtualnych zwanej klastrowanymi maszynami wirtualnymi.

Inną grupą algorytmów są algorytmy migracji pojedynczej maszyny wirtualnej. Ich działanie wynika z założenia, że każde zadanie przetwarzane przez centrum danych znajduje się na jednej maszynie wirtualnej i nie jest ono zależne od innych. Pozwala to traktować maszyny wirtualne jako niezależne od pozostałych. Poniższy raport skupia się na testowaniu takich właśnie polityk.

2.2.1 MMT - Minimum Migration Time

Polityka ta wybiera do przeniesienia maszynę wirtualną v która wymaga minimalnego czasu do ukończenia migracji w stosunku do innych maszyn wirtualnych znajdujących się na tym samym węźle. Czas jest przybliżany jako ilość pamięci RAM zużywanej przez maszynę wirtualną dzielony przez przepustowość sieci dostępną dla hosta j. Niech V_j będzie zbiorem maszyn wirtualnych na

węźle j. Polityka MMT znajduje VM która spełnia warunki przedstawione poniżej.

$$v \in V_j \mid \forall a \in V_j, \frac{RAM_u(v))}{NET_j} \leqslant \frac{RAM_u(a))}{NET_j} \tag{5}$$

gdzie $RAM_u(a)$ jest ilością pamięci operacyjnej obecnie zużywaną przez maszynę wirtualną a, zaś NET_j jest wolnym pasmem sieciowym dostępnym na węźle j. Poniżej przedstawiono kod algorytmu MMT.

```
1  public Vm getVmToMigrate(PowerHost host)
2  {
3      List<PowerVm> migratableVms = getMigratableVms(host);
4      if (migratableVms.isEmpty()) {
5          return null;
6      }
7      Vm vmToMigrate = null;
8      double minMetric = Double.MAX_VALUE;
9      for (Vm vm : migratableVms) {
10         if (vm.isInMigration()) {
11             continue;
12         }
13         double metric = vm.getRam();
14         if (metric < minMetric) {
15             minMetric = metric;
16             vmToMigrate = vm;
17         }
18     }
19     return vmToMigrate;
20 }
```

Listing 1: Szkielet algorytmu zaimplementowany w języku *Java*

Na listingu 1 widoczne jest, że pierwszym krokiem algorytmu jest filtracja maszyn wirtualnych nadających się do migracji. W kolejnych liniach następuje iteracja po wszystkich maszynach wirtualnych. Podczas każdego obrotu pętli sprawdzane jest dodatkowo czy maszyna wirtualna nie jest już w trakcie migracji. W przypadku gdy jest, następuje przejście do następnej maszyny, a w przypadku gdy nie następuje pobranie ilości pamięci RAM zużywanej przez maszynę wirtualną. Gdy okaże się, że ta ilość jest mniejsza od dotychczas minimalnej, następuje podmiana wartości minimalnej i odłożenie maszyny wirtualnej z obecnie minimalną wartością. Po sprawdzeniu wszystkich maszyn wirtualnych zwracana jest ta z minimalnym czasem migracji, co jest tożsame z ilością pamięci RAM.

2.2.2 RC - Random Choice Policy

W polityka Random Choice migrowana maszyna wirtualna wybierana jest losowo, z jednakowym prawdopodobieństwem dla wszystkich potencjalnych kandydatów.

```
1  public Vm getVmToMigrate(PowerHost host) {
2      List<PowerVm> migratableVms = getMigratableVms(host);
3      if (migratableVms.isEmpty()) {
4          return null;
5      }
6      int index = (new Random()).nextInt(migratableVms.size());
7      return migratableVms.get(index);
8  }
```

Listing 2: Szkielet algorytmu RC zaimplementowany w języku *Java*

Jak widać w listingu 2 na samym początku następuje selekcja maszyn wirtualnych nadających się do migracji a następnie losowana jest liczba całkowita odpowiadająca numerowi maszyny.

2.2.3 MC - Maximum Correlation Policy

Maximum Correlation (MC) jest metodą bazuje zaproponowaną w [11]. Ideą tego algorytmu jest fakt, że im większa korelacja zużycia zasobów przez aplikacje działające na nadmiernie obciążonym hoście tym większe prawdopodobieństwo przeciążenia serwera. Zgodnie z tą ideą do migracji wybierane są maszyny wirtualne, które odznaczają się najbardziej skorelowanym zużyciem CPU z innymi VM znajdującymi się na tym samym serwerze fizycznym.

```
1  public Vm getVmToMigrate(PowerHost host) {
2      List<PowerVm> migratableVms = getMigratableVms(host);
3      if (migratableVms.isEmpty()) {
4          return null;
5      }
6      List<Double> metrics = null;
7      try {
8          metrics = getCorrelationCoefficients(getUtilizationMatrix(
               migratableVms));
9      } catch (IllegalArgumentException e) {
10         return getFallbackPolicy().getVmToMigrate(host);
11     }
12     double maxMetric = Double.MIN_VALUE;
13     int maxIndex = 0;
14     for (int i = 0; i < metrics.size(); i++) {
15        double metric = metrics.get(i);
16        if (metric > maxMetric) {
17            maxMetric = metric;
18            maxIndex = i;
19        }
20    }
21    return migratableVms.get(maxIndex);
22 }
```

Listing 3: Szkielet algorytmu MC zaimplementowany w języku *Java*

W pokazany powyżej kod funkcji pierwszy krok działania algorytmu polega na przygotowaniu danych historycznych, tak aby przekształcić je do postaci wektora o rozmiarze j, gdzie j jest ilością maszyn wirtualnych znajdującą się na przetwarzanym węźle. Każdy element wektora zawiera współczynnik korelacji j-tej maszyny wirtualnej z innymi. W ostatnim kroku wybierany jest element o minimalnej wartości, a następnie algorytm zwraca VM odpowiadającą temu elementowi. Dodatkowo Jeżeli podczas wyznaczania współczynników korelacji wystąpi sytuacja wyjątkowa, algorytm wykorzysta politykę zapasową, którą zazwyczaj jest MMT.

2.2.4 MINU - MinimalUtilization

Polityka Minimum Utilization wybiera do migracji maszyny wirtualne, które w najmniej obciążają procesor. W związku z tym spodziewać się należy, że stosowanie tej polityki będzie powodowało wzrost liczby migracji. Kryterium wyboru jest przedstawione w postaci równania poniżej.

$$v \in V_j \mid \forall a \in V_j, \frac{MIPSc_j(v))}{MIPS_T} \leqslant \frac{MIPSc_j(a))}{MIPS_T} \tag{6}$$

gdzie $MIPSc_j(v)$ jest obciążeniem procesora przez maszynę wirtualną v, zaś $MIPS_T$ jest całkowitą mocą obliczeniową procesora dostępną na węźle.

```
1  public Vm getVmToMigrate(PowerHost host) {
2      List<PowerVm> migratableVms = getMigratableVms(host);
3          if (migratableVms.isEmpty()) {
4              return null;
5          }
6      Vm vmToMigrate = null;
7      double minMetric = Double.MAX_VALUE;
8      for (Vm vm : migratableVms) {
9          if (vm.isInMigration()) {
10             continue;
11         }
12         double metric = vm.getTotalUtilizationOfCpuMips(CloudSim.
               clock()) / vm.getMips();
13         if (metric < minMetric) {
14             minMetric = metric;
15             vmToMigrate = vm;
16         }
17     }
18     return vmToMigrate;
19 }
```

Listing 4: Szkielet algorytmu MINU zaimplementowany w języku *Java*

Listing 4 przedstawia kod algorytmu Minimal Utilization służącego wybieraniu maszyny wirtualnej do przeniesienia na inny węzeł centrum obliczeniowego. W liniach 2-5 następuje filtracja maszyn wirtualnych, a następnie sprawdzenie czy na hoście znajdują się jakieś maszyny wirtualne nadające się do przeniesienia. Następnie w liniach 8-17 następuje iteracja po wszystkich dostępnych maszynach wirtualnych. Każda maszyna jest sprawdzana na okoliczność znajdowania się w trakcie migracji. W razie braku spełnienia tego kryterium następuje dalsze przetwarzanie, które polega na pobraniu obecnego obciążenia CPU generowanego przez VM a następnie sprawdzeniu czy otrzymana wartość jest minimalna. Jeżeli tak odkładane są odpowiednie wartości i przetwarzanie przenosi się na kolejną VM. W linii 18 zwracana jest maszyna wirtualna z minimalną wartością utylizacji CPU.

2.2.5 CTH - Closest to Threshold

Polityka Closest to threshold (CTH) została zaproponowana przez Zhanga i innych w artykule [12] i jest modyfikacją polityki Largest VM First. Należy spodziewać się, że wybieranie bardziej obciążonych maszyn wirtualnych zamiast tych największych spowoduje zmniejszenie się liczby migracji.

```
1  public Vm getVmToMigrate(PowerHost host) {
2      List<PowerVm> migratableVms = getMigratableVms(host);
3      if (migratableVms.isEmpty()) {
4          return null;
5      }
6          //counting difference between threshold mips  and total
               allocated mips
7
8      double thresholdMips = threshold*host.getTotalMips();
9      double totalAllocatedMips = host.getTotalAllocatedMips();
10     double difference = 0.0;
11     if(thresholdMips >= totalAllocatedMips){
12         difference = thresholdMips - totalAllocatedMips;
13     }else{
14        difference = totalAllocatedMips - thresholdMips;
```

```
15      }
16
17      Vm vmToMigrate = null;
18      double minMetric = Double.MAX_VALUE;
19      for (Vm vm : migratableVms) {
20          if (vm.isInMigration()) {
21              continue;
22          }
23          double vmMips = vm.getTotalUtilizationOfCpuMips(CloudSim.
                clock());
24          double metric = vmMips - difference;
25          if(metric > 0) {
26              if (metric < minMetric) {
27                  minMetric = metric;
28                  vmToMigrate = vm;
29              }
30          }
31      }
32
33      if(vmToMigrate == null){
34          for (Vm vm : migratableVms) {
35              if (vm.isInMigration()) {
36                  continue;
37              }
38
39              double vmMips = vm.getTotalUtilizationOfCpuMips(
                    CloudSim.clock());
40              double metric = vmMips - difference;
41              if(metric < 0) {
42                  if (Math.abs(metric) < minMetric) {
43                      minMetric = Math.abs(metric);
44                      vmToMigrate = vm;
45                  }
46              }
47          }
48      }
49          return vmToMigrate;
50  }
```

Listing 5: Szkielet algorytmu CTH zaimplementowany w języku *Java*

Listing 5 przedstawia zarys algorytmu (CTH). Tak jak w innych przypadkach najpierw następuje oddzielanie maszyn, które mogą być przeniesione od pozostałych. W liniach 8-16 następuje obliczenie różnicy między progową ilością CPU i ilością zaalokowaną na całym hoscie, a następnie iteracja po wszystkich VM w celu znalezienia minimalnej wartości metryki. Metryka ta jest liczona jako różnica ilości CPU absorbowanej przez obecnie przetwarzaną VM i różnicy obliczonej w liniach 8-16. Wynikiem tej operacji jest odległość utylizacji hosta od progu nadmiernego obciążenia po przeniesieniu maszyny wirtualnej. Jeżeli ta metryka jest większa od zera to znaczy, że przeniesienie maszyny spowoduje spadek obciążenia hosta poniżej progu. Alternatywnie, w liniach 34-49 następuje szukanie VM, której dealokacja spowoduje jak największy spadek utylizacji węzła. W tym wypadku dealokacja VM nie spowoduje spadku utylizacji serwera poniżej progu co wymusi wybór kolejnej maszyny do migracji.

2.2.6 CUV - CPU Utilization Variance

W proponowanym przez Selim'a i innych[9] algorytmie wybierana jest maszyna wirtualna o najmniejszej wartości metryki opisanej zamieszczonym poniżej wzorem (7). Algorytm przebiega według następującego schematu:

1. Wykrywane są maszyny wirtualne na węźle źródłowym, które mogą być zmigrowane.
2. Przed alokacją pierwszego dostępnego hosta docelowego obliczana. jest utylizacja tego hosta tak jakby pierwsza z dostępnych maszyn była na niego przeniesiona.
3. Obliczany jest współczynnik CUV pomiędzy zaalokowanym hostem a innymi hostami. Zgodnie z równaniem poniżej CUV jest zdefiniowany jako suma kwadratów odległości średniej utylizacji hostów od utylizacji j-tego hosta, podzielona przez ilość hostów.

$$CUV = \frac{\sum_{j=1}^{M}(U_j - \mu)^2}{M} \tag{7}$$

   gdzie μ jest średnią utylizacją hosta, Uj jest utylizacją j-tego hosta a M jest ilością hostów.
4. Należy powtarzać kroki 4 i 5 dla tej samej maszyny ale tak jakby była migrowana na inny dostępny węzeł docelowy. Ten krok jest powtarzany do momentu aż współczynnik dla CUV dla tych przypadków zostanie policzony.
5. Następnie należy powtórzyć krok 6 dla wszystkich N zdolnych do migracji maszyn wirtualnych na tym hoście. Następnie jest konstruowana macierz VU o rozmiarze NxM, gdzie element V_{ij} jest współczynnikiem CUV liczonym w sytuacji migracji maszyny wirtualnej i na węzeł j.

$$VU_{i,j} = \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,j} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ v_{i,1} & v_{i,2} & \cdots & v_{i,j} \end{pmatrix} \tag{8}$$

6. Ostatnim krokiem algorytmu jest wykrycie dla której maszyny wirtualnej (w którym wierszy macierzy) występuje minimalny współczynnik. Po wykryciu tej VM jest ona zwracana jako wynik algorytmu

2.2.7 LVF - Largest VM first

Pomysł algorytmu LVF przedstawiono w [12], sprowadza sie on do stwierdzenia, że najbardziej naturalnym wyborem podczas migracji jest przenoszenie maszyn wirtualnych, które są największe. Podstawą metody jest kryterium:

$$v \in V_j \mid \forall a \in V_j, CPU_j(v) \leqslant CPU_j(a) \tag{9}$$

gdzie $CPU_j(v)$ jest zadeklarowaną mocą procesora na maszynie wirtualnej v, a $CPU_j(a)$ jest zadeklarowaną mocą procesora na wszystkich pozostałych VM.

```
1    public Vm getVmToMigrate(PowerHost host) {
2        List<PowerVm> migratableVms = getMigratableVms(host);
3        if (migratableVms.isEmpty()) {
4            return null;
5        }
6        Vm vmToMigrate = null;
7        double maxMetric = Double.MIN_VALUE;
8        for (Vm vm : migratableVms) {
9            if (vm.isInMigration()) {
```

```
10                      continue;
11                  }
12                  double metric = vm.getMips();
13                  if (metric > maxMetric) {
14                      maxMetric = metric;
15                      vmToMigrate = vm;
16                  }
17              }
18              return vmToMigrate;
19  }
```

Listing 6: Szkielet algorytmu LVF zaimplementowany w języku *Java*

Listing 6 przedstawia zarys algorytmu LVF, algorytm ten jako kryterium traktuje zadeklarowaną pojemność danej maszyny wirtualnej. Podczas każdej iteracji pobierana jest zadeklarowana wielkość CPU dla danej VM, a następnie sprawdzane jest czy pobrana wartość jest jak do tej pory najmniejsza, jeżeli tak to odpowiednie wartości są odkładane, a jeżeli nie to następuje przejście do przetwarzania następnej VM. Na koniec zwracana jest VM o najmniejszej zadeklarowanej mocy procesora.

2.2.8 MU - Maximal Utilization

Metoda ta działa na przeciwnej zasadzie niż polityka MINU, a mianowicie do migracji wybiera maszyny które w danej chwili utylizują największą ilość procesora.

```
1  public Vm getVmToMigrate(PowerHost host) {
2      List<PowerVm> migratableVms = getMigratableVms(host);
3      if (migratableVms.isEmpty()) {
4          return null;
5      }
6      Vm vmToMigrate = null;
7      double minMetric = Double.MIN_VALUE;
8      for (Vm vm : migratableVms) {
9          if (vm.isInMigration()) {
10             continue;
11         }
12         double metric = vm.getTotalUtilizationOfCpuMips(CloudSim.
               clock()) / vm.getMips();
13         if (metric > minMetric) {
14             minMetric = metric;
15             vmToMigrate = vm;
16         }
17     }
18     return vmToMigrate;
19 }
```

Listing 7: Szkielet algorytmu MAXU zaimplementowany w języku *Java*

Struktura kodu jest podobna do poprzednich. W czasie iteracja po wszystkich VM, wybierana jest maszyna wirtualna o najwyższym poziomie utylizacji procesora. W ostatnim kroku wybrana maszyna wirtualna jest zwracana jako wynik działania algorytmu.

2.3 Metody wyboru maszyny docelowej

Algorytmy konsolidacji maszyn wirtualnych można podzielić na trzy główne kategorie biorąc pod uwagę kryterium wyboru maszyny docelowej. Do Pierwszej z nich należą algorytmy losowo wybierające docelowego hosta. Sama metoda losowego wyboru nie wymaga wyjaśnienia, po za faktem, że

wybór jest dokonywany z hostów dostępnych, czyli takich które nie zostały wykluczone ze względu na ich stan, status lub obecną utylizację.

Kolejną grupą są chciwe heurystyki które są najbardziej rozpowszechnione w literaturze. Ich cechą wspólną jest to, że najpierw próbują one dokonać wyboru węzła przeznaczenia z pośród węzłów już włączonych. Poza rozwiązania przez nie sugerowane nie są rozwiązaniami optymalnymi a suboptymalnymi. Wedle definicji istnieje nawet możliwość, że w pewnych przypadkach takie algorytmy dostarczają rozwiązania nieprawidłowe. Używa się ich wtedy kiedy zastosowanie algorytmu wyznaczającego rozwiązanie optymalne jest zbyt kosztowne. Z racji na ich prostotę algorytmy te często mogą grzęznąć w minimach lokalnych.

Ostatnią grupą są algorytmy meta-heurystyczne. Różnią się one tym od chciwych heurystyk, że próbują uniknąć ugrzęźnięcia w minimach lokalnych. Przykładami takich heurystyk które mogą być zastosowane w celu wyboru węzła docelowego są algorytmy ewolucyjne [10], albo optymalizacja kolonią mrówek [5].

2.3.1 WPC - Watts Per Core

Algorytm Watts Per Core opiera się on na szukaniu węzła docelowego tak aby jego migracja spowodowała jak najmniejszy przyrost zużycia mocy na węźle docelowym. Poniżej przedstawiono kod algorytmu zaimplementowany w języku Java.

```
1  public PowerHost findHostForVmFromPassed(Vm vm, List<? extends Host
       > possibleDestHosts, Set<? extends Host> excludedHosts) {
2      double minPower = Double.MAX_VALUE;
3      PowerHost allocatedHost = null;
4
5      for (Host h : possibleDestHosts) {
6          PowerHost host = (PowerHost) h;
7          if (excludedHosts.contains(host)) {
8              continue;
9          }
10         if (host.isSuitableForVm(vm)) {
11             if (getUtilizationOfCpuMips(host) != 0 &&
                   isHostOverUtilizedAfterAllocation(host, vm)) {
12                 continue;
13             }
14
15             try {
16                 double powerAfterAllocation =
                       getPowerAfterAllocation(host, vm);
17                 if (powerAfterAllocation != -1) {
18                     double powerDiff = powerAfterAllocation -
                           host.getPower();
19                     if (powerDiff < minPower) {
20                         minPower = powerDiff;
21                         allocatedHost = host;
22                     }
23                 }
24             } catch (Exception e) {
25             }
26         }
27     }
28     return allocatedHost;
29 }
```

Listing 8: Szkielet algorytmu WPC zaimplementowany w języku *Java*

Na listingu 8 przedstawiono kod algorytmu Watts Per Core. Jako jego wejście jest przekazywana VM, którą należy przenieść na innego hosta, drugim argumentem wejściowym jest kolejnym jest lista węzłów z pośród których będzie wybierany host docelowy. Ostatnim jest lista węzłów wyłączona z wyboru. Podczas działania procedury następuje iteracja po wszystkich hostach przekazanych jako drugi argument wejściowy. W iteracjach sprawdzane jest czy w razie przeniesienia VM na danego hosta nie stanie się on nadmiernie obciążony. W kluczowej sekcji pobierany obliczana jest różnica zużycia mocy przed i po alokacji danej maszyny wirtualnej. Następnie sprawdzane jest, czy ta różnica jest jak do tej pory minimalna. W przypadku, gdy tak jest dane hosta są odkładane. Po sprawdzeniu wszystkich dostępnych hostów zwracany jest ten na którego migracja powoduje najmniejszy przyrost mocy.

2.3.2 MU - Maximal Utilization

Polityka Maximal Utilization polega na wybieraniu węzłów, których poziom utylizacji jest jak najwyższy. Jej założeniem jest, że host docelowy wybierany jest z pośród aktywnych. Jest to najbardziej oczywista polityka, wybrana jao punkt odniesienia dla pozostałych.

```
1  public PowerHost findHostForVmFromPassed(Vm vm, List<? extends Host
       > possibleDestinationHosts, Set<? extends Host> excludedHosts) {
2      double maxUtilization = -100d;
3      PowerHost allocatedHost = null;
4      PowerHost sourceHost = (PowerHost) vm.getHost();
5      if(sourceHost == null){
6          Log.printConcatLine("SourceHost is null. Initial VM
               allocation.");
7      }
8      for (Host h : possibleDestinationHosts) {
9          PowerHost host = (PowerHost) h;
10         if (excludedHosts.contains(host)) {
11             continue;
12         }
13         if (host.isSuitableForVm(vm)) {
14             if (getUtilizationOfCpuMips(host) != 0 &&
                   isHostOverUtilizedAfterAllocation(host, vm)) {
15             continue;
16             }
17
18             try {
19                 double destinationUtilization = host.
                       getUtilizationOfCpu();
20
21                 if (destinationUtilization > maxUtilization) {
22                     maxUtilization = destinationUtilization;
23                     allocatedHost = host;
24                 }
25
26             } catch (Exception e) {
27                 Log.printConcatLine("Invalid destination");
28             }
29         }
30     }
31     return allocatedHost;
32 }
```

Listing 9: Szkielet algorytmu MU zaimplementowany w języku *Java*

W listingu 9 przedstawiono szkielet algorytmu MU. Sprowadz się on do poszukiwania hosta charakteryzującego się maksymalną utylizacją spośród wszystkich aktywnych.

2.3.3 PCK - Packing

Polityka wyboru hosta docelowego Packing [8] jest przeciwieństwem polityki Stripping. Z dostępnych węzłów wybiera ona ten, który w danym momencie obsługuje największą ilość maszyn wirtualnych a następnie migruje na niego daną maszynę wirtualną.

```
1  private PowerHost findHostForVmFromPassed(Vm vm, List<? extends
       Host> destHosts, Set<? extends Host> excludedHosts) {
2      int maximalVmPerHost = Integer.MIN_VALUE;
3      PowerHost allocatedHost = null;
4      for(Host h : destHosts){
5          PowerHost host = (PowerHost) h;
6              if(excludedHosts.contains(host)){
7                  continue;
8              }
9              if(host.isSuitableForVm(vm)){
10                 if (getUtilizationOfCpuMips(host) != 0 &&
                       isHostOverUtilizedAfterAllocation(host, vm)) {
11                     continue;
12                 }
13
14                 try {
15                     int numberOfVmsInCurrentHost = host.getVmList().
                           size();
16                     if (numberOfVmsInCurrentHost >= -1) {
17                         if(numberOfVmsInCurrentHost >
                               maximalVmPerHost){
18                             maximalVmPerHost =
                                   numberOfVmsInCurrentHost;
19                             allocatedHost = host;
20                         }
21                     }
22                 } catch (Exception e) {
23                 }
24             }
25         }
26         return allocatedHost;
27 }
```

Listing 10: Szkielet algorytmu PCK zaimplementowany w języku *Java*

W listingu 10 został przedstawiony szkielet algorytmu który odzwierciedla jego działanie. Tak jak w pozostałych algorytmach ten algorytm posiada wiedzę na temat wszystkich dostępnych hostów i sprawdza po kolei wszystkie z nich. Każdy z hostów jest sprawdzany pod względem użyteczności hosta jako hosta docelowego migracji. W razie pozytywnego przejścia weryfikacji z węzła jest pobierana ilość znajdujących się na nim VM. Następnie spośród wrzystkich pobranych danych jest wybierana ta o maksymalnej wartości, a host jej odpowiadający jest zwracany jako wynik działania algorytmu.

2.3.4 RR - Round Robin

Ideą polityki Round Robin [8] jest aby wszystkie hosty zorganizować w bufor cykliczny. Następnie podczas każdej iteracji algorytmu jako host docelowy jest wybierany host następny w buforze.

```
1  public PowerHost findHostForVmFromAll(Vm vm,  Set<? extends Host>
       excludedHosts) {
2
3      double minPower = Double.MAX_VALUE;
4      PowerHost allocatedHost = null;
5      List<PowerHostStateAware> hostList = getHostList();
6
7      PowerHostStateAware host;
8      int count = 0;
9      while ((host = getNextHost()) != null) {
10       if(count>=hostList.size()){
11           break;
12       }
13       count++;
14       if (excludedHosts.contains(host)) {
15           continue;
16       }
17       if (host.isSuitableForVm(vm)) {
18           if (getUtilizationOfCpuMips(host) != 0 &&
                 isHostOverUtilizedAfterAllocation(host, vm)) {
19               continue;
20           }
21           allocatedHost = host;
22           break;
23       }
24   }
25   return allocatedHost;
26 }
```

Listing 11: Szkielet algorytmu RR zaimplementowany w języku *Java*

Listing 11 przedstawia schemat algorytmu Round Robin. Podczas pierwszego obiegu pętli wskaźnik znajduje się na węźle, który był wybrany jako węzeł docelowy podczas poprzedniego uruchomienia algorytmu lub na pierwszym węźle, jeżeli jest to pierwsze uruchomienie algorytmu. Następnie hosty są sprawdzane po kolei pod względem możliwości migracji. Pierwszy host, który przejdzie weryfikację jest wybierany jako host docelowy. Wskaźniki bufora są ustawione na niego a następnie jest on zwracany jako wynik działania algorytmu.

2.3.5 STP - Stripping

Polityka Stripping jest przeciwieństwem Packing. Jej działanie polega na wyborze hosta docelowego w taki sposób, aby maszyny wirtualne były migrowane na węzły na których znajduje się jak najmniejsza liczba maszyn wirtualnych.

```
1  private PowerHost findHostForVmFromPassed(Vm vm, List<? extends
       Host> possibleDestHosts, Set<? extends Host> excludedHosts) {
2      int minimalVmPerHost = Integer.MAX_VALUE;
3      PowerHost allocatedHost = null;
4      for(Host h : possibleDestHosts){
5          PowerHost host = (PowerHost) h;
6          PowerHostStateAware hostStateAware = (PowerHostStateAware)
               host;
7          if(excludedHosts.contains(host)){
8              continue;
9          }
10         if (host.isSuitableForVm(vm)){
```

```
11              if (getUtilizationOfCpuMips(host) != 0 &&
                    isHostOverUtilizedAfterAllocation(host, vm)) {
12                  continue;
13              }
14
15              try {
16                  int numberOfVmsInCurrentHost = host.getVmList().size
                        ();
17                  if (numberOfVmsInCurrentHost >= 0) {
18                      if(numberOfVmsInCurrentHost < minimalVmPerHost){
19                          minimalVmPerHost = numberOfVmsInCurrentHost;
20                          allocatedHost = host;
21                      }
22                  }
23              } catch (Exception e) {
24              }
25          }else{
26          }
27      }
28      return allocatedHost;
29  }
```

Listing 12: Szkielet algorytmu STP zaimplementowany w języku *Java*

Algorytm posiada strukturę podobną do poprzednich. Podczas jego wykonywania następuje iteracja po wszystkich węzłach. Każdy z nich jest sprawdzany czy pomieści rozpatrywaną maszynę wirtualną. Jeżeli przejdzie on tę weryfikację następuje wyznaczenie liczby maszyn wirtualnych, które się na nim znajdują. Następnie wybierany jest host, na którym jest najmniejsza liczba VM i ten host jest zwracany jako wynik działania algorytmu.

3 Eksperymenty symulacyjne

3.1 Symulator CloudSim

CloudSim [3] jest środowiskiem, które poprzez wykorzystanie dostarczonych elementów i programowanie własnych może być wykorzystane w badaniach symulacyjnych klastrów i chmur obliczeniowych. W jego architekturze mozna wyróżnić trzy główne warstwy:

- warstwa kodu użytkownika
- warstwa symulatora
- silnik CloudSim

Silnik CloudSim odpowiada za podstawow funkcjonalności takie jak kolejkowanie, przetwarzanie zdarzeń, tworzenie encji wchodzących w skład symulowanego centrum obliczeniowego, komunikacja między tymi komponentami i zarządzanie czasem symulacji.

Warstwa symulatora wspiera modelowanie i symulację wirtualizowanych centrów obliczeniowych, jak również udostępnią interfejs do zarządzania maszynami wirualnymi. Komponenty tej warstwy realizują podstawowe funkcjonalności, takie jak dostarczanie serwerów dla maszyn wirtualnych, zarządzanie wykonywaniem zadań i monitorowanie zmieniającego się stanu centrum obliczeniowego. Jest to warstwa pozwalająca użytkownikowi m. in. na symulowanie różnych strategi, przydzielanie zasobów dla zadań bądź grupowanie maszyn wirtualnych.

Najwyższą warstwę tworzy kod użytkownika, który definiuje serwery fizyczne, maszyny wirtualnyne i zadania przez nie wykonywane. Zmieniając tę warstwę użytkownik może generować różne

Serwer	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	93.7	97	101	105	110	116	121	125	129	133	135

Tabela 1: Zużycie mocy przez wybrane serwery pod różnymi obciążeniami podane w Watach

obciążenia symulowanej chmury, modelować rozmiar i dostępność chmury, jak również implementować różne techniki przydzielenia zadań w danej chmurze.

Do budowy modelu chmury obliczeniowej wykorzystuje się rozszerzenia komponentu *Datacenter* i mechanizmów w nim zawartych. Komponent ten zarządza liczbą hostów jak również liczbą i przypisaniem maszyn wirtualnych. Host jest odpowiednikiem fizycznego serwera w chmurze obliczeniowej. Charakteryzuje go moc przetwarzania wyrażana w milionach instrukcji na sekundę (MIPS - Million Instructions per Second), pojemność RAM, pojemność dysku twardego, jak również przepustowość sieci. Warstwę wirtualizacji w symulatorze CloudSim implementują komponenty *Host* i *Vm* wraz z politykami odpowiadającymi za współdzielone przez maszyny wirtualne zasobów obliczeniowych fizycznego serwera. Podobnie sytuacja przedstawia się w przypadku szeregowania wielu zadań na jednej maszynie wirtualnej. W tym przypadku również stosuje się różne polityki współdzielenia zasobów chmury obliczeniowej. Symulacji zmiennego obciążenia służy komponent *UtilizationModel*, jak również rozszerzenie klasy *CloudletScheduler*, które zostało przygotowane do pobierania obciążenia na podstawie danej chwili czasowej. Symulator CloudSim udostępnia również zbiór komponentów, które pozwalają na symulowanie centrów obliczeniowych w sposób, który bierze pod uwagę zużycie energii. Głównym z tych komponentów jest *PowerModel*, który odpowiada za wyznaczanie zużycia energii przez pojedynczego hosta w zależności od jego obciążenia. Komponent *PowerDatacenter* zlicza moc zużywaną przez wszystkie węzły należące do danego centrum obliczeniowego.

3.2 Scenariusze symulacji

Aby uczynić eksperymenty symulacyjne bardziej wiarygodnymi uzyto danych opisujących obciążenia pochodzących z realnego systemu. Pochodzą one z projektu CoMon, który jest infrastrukturą monitorującą dla PlanetLab. Dane gromadzone w tym projekcie dotyczyły utylizacji CPU i zostały zebrane z ponad tysiąca maszyn wirtualnych zlokalizowanych w ponad 500 miejscach na całym świecie. Interwał pomiędzy kolejnymi pomiarami wynosi 300 sekund. W symulatorze są dostępne zbiory z 10 wybranych dni, a w symulacjach zostały użyte próbki pochodzące z jednego z tych dni.

Używając opisanych wcześniej danych wejściowych zostały przesymulowane wszystkie kombinacje algorytmów selekcji maszyn wirtualnych (VM) i algorytmów wyboru hostów docelowych (HD). Jako algorytm wyznaczania hostów nadmiernie obciążonych wykorzystano jeden z algorytmów dostępnych w symulatorze CloudSim, a mianowicie algorytm Static Threshold. W ten sposób uzyskano 40 kombinacji algorytmów, z których każda stanowi osobny scenariuszem testowym.

Głównym celem przeprowadzenia testów było wykrycie wpływu różnych algorytmów selekcji maszyn wirtualnych i wyboru hosta docelowego na zużycie energii. Kolejnym z celów jest przeprowadzenie ich w taki sposób aby wszystkie zlecone zadania zostały wykonane w zadanym czasie, bez opóźnień. Aby dokonać analizy porównawczej podczas przeprowadzania każdej symulacji zostały zebrane odpowiednie dane, które posłużyły do wyliczenia metryk uzytych w trakcie analizy.

W symulacjach algorytmów zostały użyte dwie konfiguracje sprzętowe z dwu-rdzeniowymi procesorami: HP ProLiant ML110 G4 (Intel Xeon 3040, 2 rdzenie 1860 MHz, 4 GB) i HP ProLiant ML110 G5 (Intel Xeon 3075, 2 rdzenie 2660 MHz, 4 GB). Konfiguracja i zużycie mocy serwerów jest pokazane w tabeli 1.

Celem działania systemu jest przetwarzanie zadań nadesłanych przez użytkowników. Stara się on realizować to w taki sposób, aby uzyskać jak największą oszczędność w zużyciu energii. Ten cel realizowany jest głównie poprzez dynamiczną realokację maszyn wirtualnych na których będą przetwarzane zadania. Na początku maszyny wirtualne wraz z zadaniami są alokowane na

oddzielnych maszynach fizycznych. Następnie przez okres całej symulacji system stara się przenosić maszyny wirtualne zgodnie z zapotrzebowaniem, tak aby upakować je na jak najmniejszej ilości hostów. Najważniejszym etapem działania symulowanego systemu jest cykliczne badanie stanu centrum obliczeniowego i próba wyznaczenia mapy migracji.

3.3 Metryki QoS

3.3.1 Koszty migracji

Migracja dynamiczna pozwala przenieść maszynę wirtualną między hostami bez jej zawieszania i z krótkim okresem wyłączenia. Dynamiczna migracja może jednak obniżać efektywność zadań wykonywanych na maszynie wirtualnej. W pracy [2] wykazano, że spadek wydajności na takiej maszynie może osiągać 10% utylizacji CPU. Co więcej w modelu kosztu migracji założono, że 10% pojemności CPU używanej przez VM jest alokowane w węźle docelowym na czas migracji. To oznacza, że każda migracja może spowodować naruszenie SLA. Z tego powodu należy starać się ograniczyć liczbę migracji. Długość czasu migracji zależy od ilości pamięci RAM zużywanej przez migrowaną maszynę wirtualną i od dostępnej przepustowości łącza sieciowego. Czas migracji i degradacja wydajności spowodowaną migracją obliczano za pomocą poniższych formuł:

$$T_{m_j} = \frac{M_j}{B_j} \tag{10}$$

$$U_{d_j} = 0.1 * \int_{t_0}^{t_0+T_{m_j}} u_j(t)dt \tag{11}$$

gdzie U_{d_j} jest całkowitą degradacją wydajności maszyny wirtualnej j, t_0 jest czasem rozpoczęcia migracji, T_{m_j} jest czasem do zakończenia migracji, $u_j(t)$ jest utylizacją j-tej maszyny wirtualnej w chwili t, M_j jest ilością pamięci RAM używaną przez j-tą VM, a B_j jest dostępną przepustowością sieci.

3.3.2 Metryki naruszenia SLA

Wymagania jakościowe są zwykle formalizowane jako SLA, które mogą być zdefiniowane jako minimalna przepustowość, albo maksymalny czas odpowiedzi dla danego systemu. Z racji tego, że wyżej wymienione parametry zależą od zadania jakie dostawca otrzymuje do wykonania, należało zaproponować metryki, które są niezależne od rodzaju zadania. W opracowanym systemie założono, że SLA jest wypełnione, gdy 100% obciążenia zarządanego przez zadania jest dostarczone przez dostawcę. W badaniach wykorzystano metryki SLATAH i PDM wykorzystane w pracy [2].

Obie z nich są metrykami, które służą do pomiaru jakości usług świadczonych przez centrum danych. SLATAH, czyli SLA Violation Time per Active Host jest to metryka, pokazująca stosunek czasu w którym host był nadmiernie obciążony (100% obciążenia), do czasu w jakim host był aktywny (utylizacja CPU większa niż 100%):

$$SLATAH = \frac{1}{N}\sum_{i=1}^{N}\frac{T_{S_i}}{T_{A_i}} \tag{12}$$

gdzie N jest liczbą hostów, T_{S_i} jest całkowitym czasem, podczas którego host i był obciążony w 100% co doprowadziło do naruszeń SLA a T_{A_i} jest całkowitym czasem w którym host i był aktywny.

PDM, czyli Performance Degradation due to Migration, jest to metryka, która pokazuje całkowity spadek efektywności maszyn wirtualnych w związku z migracjami:

$$PDM = \frac{1}{M}\sum_{j=1}^{M}\frac{C_{d_j}}{C_{r_j}} \tag{13}$$

gdzie M jest liczbą maszyn wirtualnych, C_{d_j} całkowitym przybliżonym spadkiem efektywności przetwarzania na maszynie wirtualnej j a C_{r_j} jest całkowitą zażądaną mocą obliczeniową CPU dla j-tej maszyny wirtualnej podczas całego jej działania.

4 Wyniki eksperymentów

4.1 Porównanie polityk wyboru hosta docelowego

4.1.1 Algorytm selekcji Closest To Threshold

Wyniki testów, w których selekcji maszyn wirtualnych do migracji dokonano za pomocą algorytmu Closest To Threshold, a węzeł docelowy migracji był wybierany z pomocą wszystkich rozpatrywanych polityk przedstawiono w tabelach 2 i 3. W tabeli 2 przedstawiono sumaryczne wyniki. Wskazują one, że algorytmy PCK i MU charakteryzują się największą średnią utylizacją hosta a co za tym idzie najmniejszą ilością aktywnych hostów. Również te algorytmy okazały się najefektywniejsze pod kątem zużycia energii. Dodatkowo pod względem łącznej liczby migracji wygrywa polityka PCK a zaraz za nią jest WPC. Dopiero na trzecim miejscu znajduje się algorytm MU. Najgorsze wyniki, jeżeli chodzi o ilość migracji mają symulacje CTH-STP i CTH-RR. Pod względem średniego czasu migracji najkorzystniejszy okazał się algorytm WPC, a zaraz po nim uplasował się algorytm RR. Pod względem dwóch metryk opisujących jakość obsługi zadań prawie wszystkie polityki są do siebie zbliżone. Metryka SLATAH jest na poziomie 6% a metryka PDM na poziomie 0.1%.

nazwa testu	średnia utylizacja HD	ilość migracji	zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	SLA ze względu na utylizację (SLATAH)	SLA ze względu na migrację(PDM)
1-AF-thr-cth-mu	0.71	25485	141.1417	47.85	31.84	0.06495	0.00190
1-AF-thr-cth-pck	0.71	22650	139.3931	48.06	31.58	0.06777	0.00182
1-AF-thr-cth-rr	0.61	32110	159.9687	58.11	20.88	0.06685	0.00153
1-AF-thr-cth-stp	0.61	47580	171.7643	64.23	22.59	0.08733	0.00207
1-AF-thr-cth-wpc	0.59	23659	153.8972	54.70	20.44	0.06244	0.00120
1-AF-thr-cth-wpca	0.59	23659	153.8972	54.70	20.44	0.06244	0.00120

Tabela 2: Tabela przedstawiająca dane dotyczące algorytmu selekcji CTH.

W tabeli 3 przedstawiono pomiary czasów jakich dokonano podczas symulacji. W każdym z przypadków pomiary czasów selekcji VM są bardzo zbliżone. Natomiast najkrótszym średnim czasem wyznaczania hosta docelowego do migracji charakteryzuje się polityka RR. Za nimi znajdują się PCK, WPC, MU. Najgorszy wynik dał algorytm STP. Odchylenia standardowe dla wszystkich eksperymentów oprócz CTH-RR są porównywalne.

4.1.2 Algorytm selekcji maszyn wirtualnych CUV

Tabele 4 i 5 przedstawiają zestawienie wyników dla testów, w których selekcji maszyn dokonano za pomocą polityki CPU Utilization Variance, zaś jako politykę wyboru hosta docelowego zastosowano

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-cth-mu	0.00062	0.00150	0.02618	0.04326
1-AF-thr-cth-pck	0.00082	0.00360	0.02205	0.02541
1-AF-thr-cth-rr	0.00071	0.00057	0.00677	0.00532
1-AF-thr-cth-stp	0.00049	0.00050	0.04783	0.04621
1-AF-thr-cth-wpc	0.00044	0.00051	0.02433	0.04343
1-AF-thr-cth-wpca	0.00043	0.00050	0.02469	0.04817

Tabela 3: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM Closest To Threshold i dla wszystkich algorytmów wyboru hosta docelowego

wszystkie rozważane algorytmy. Najlepszą średnią utylizacją hosta cieszą się eksperymenty PCK i MU (68%). Wszystkie pozostałe mają taką samą średnią utylizację na poziomie 63%. Za to ze wszystkich pozostałych to algorytm WPC posiada najmniejszą ilość aktywnych hostów, która jednak bardzo niewiele różni się od ilości aktywnych hostów dla testów CUV-PCK i CUV-MU. Najmniejszą ilością migracji może poszczycić się eksperyment CUV-PCK, a zaraz po nim są CUV-MU i CUV-WPC. Za to pod względem kryterium średniego czasu migracji prowadzi eksperyment WPC. Pod względem metryk PDM i SLATAH algorytmy te różnią bardzo nieznacznie.

nazwa testu	średnia utylizacja HD	łączna ilość migracji	łączne zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	slatah	pdm
1-AF-thr-cuv-mu	0.68	15716	142.1317	48.03	27.20	0.06608	0.00200
1-AF-thr-cuv-pck	0.68	13053	140.7186	48.35	27.56	0.06086	0.00186
1-AF-thr-cuv-rr	0.63	22360	156.9872	56.72	22.26	0.06918	0.00190
1-AF-thr-cuv-stp	0.63	32884	162.4976	60.03	22.55	0.07730	0.00245
1-AF-thr-cuv-wpc	0.63	16125	145.6639	50.94	21.32	0.06038	0.00152

Tabela 4: Dane dotyczące kombinacji algorytmu selekcji CUV ze wszystkimi algorytmami selekcji HD.

Tabela 5 przedstawia pomiary czasów selekcji VM dla algorytmu CUV i pomiary czasów dla wszystkich algorytmów wyboru hosta docelowego. Dla tej polityki selekcji zauważalny jest wzrost czasów wykonywania obliczeń , w porównania do poprzedniej polityki selekcji. Wzrost nastąpił o cztery rzędy wielkości. Co wydaje się dyskwalifikować politykę selekcji CUV ze względu na jej słabą wydajność. Czasy selekcji HD są przybliżone do odpowiadających czasów w poprzedniej grupie testów. I tak jak w poprzedniej grupie testów minimalny czas uzyskuje eksperyment CUV-RR a maksymalny CUV-STP.

4.1.3 Algorytm selekcji maszyn wirtualnych LVF

Dane przedstawione w tabeli 6 potwierdzają wcześniejsze obserwacje. Algorytmy MU i PCK posiadają największą średnią utylizację hosta. Dodatkowo na jej podstawie można stwierdzić, że najlepszy średni czas migracji uzyskał eksperyment WPC. Na kolejnych miejscach pod względem tej metryki są MU i PCK. Algorytm WPC jest również najbardziej korzystny ze względu na poziom

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-cuv-mu	1.69447	1.56408	0.01467	0.02894
1-AF-thr-cuv-pck	1.75617	1.61284	0.01258	0.02983
1-AF-thr-cuv-rr	2.20282	1.87893	0.00423	0.00152
1-AF-thr-cuv-stp	1.15744	0.97304	0.02813	0.03022
1-AF-thr-cuv-wpc	2.62470	22.45722	0.01414	0.03670

Tabela 5: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM CPU Utilization Variance i dla wszystkich algorytmów wyboru hosta docelowego

metryk opisujących pogwałcenia SLA chociaż różnice między nimi są bardzo niewielkie. Najgorzej zarówno pod względem liczby migracji jak również pogwałceń SLA przedstawia się algorytm STP.

nazwa testu	średnia utylizacja HD	łączna ilość migracji	łączne zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	slatah	pdm
1-AF-thr-lvf-mu	0.70	20911	142.0881	48.30	28.09	0.05737	0.00178
1-AF-thr-lvf-pck	0.70	19141	140.4795	48.60	28.06	0.05571	0.00173
1-AF-thr-lvf-rr	0.62	26774	158.9312	57.57	19.88	0.06281	0.00143
1-AF-thr-lvf-stp	0.63	35663	163.1648	60.32	21.34	0.07315	0.00201
1-AF-thr-lvf-wpc	0.59	21804	153.9159	54.70	18.05	0.05440	0.00114
1-AF-thr-lvf-wpca	0.59	21804	153.9159	54.70	18.05	0.05440	0.00114

Tabela 6: Dane zbiorcze dotyczące algorytmu selekcji LVF.

Tabela 7 pokazuje, że najkrótszy średni czas selekcji maszyny wirtualnej wykazuje algorytm STP. Najgorzej(najdłuższe czasy) przedstawiają się pod tym względem algorytmy MU i PCK. Za to najkrótszy średni czas wyboru HD posiada eksperyment RR. Zaraz za nim ustawiają się testy MU i PCK. Odchylenia standardowe we wszystkich przypadkach są prawie jednakowej wielkości (pomijając oczywiście eksperyment RR w którym są one o rząd wielkości mniejsze).

4.1.4 Algorytm selekcji maszyn wirtualnych MAXU

W tabelach 8 i 9 przedstawiono zbiorcze dane dotyczące testów, w których jako politykę selekcji maszyny wirtualnej stosowano politykę Maximal Utilization of VM, a jako politykę wyboru hosta docelowego migracji stosowano wszystkie brane pod uwagę algorytmy. Z danych zawartych w tabeli 8 widoczne jest, że pod względem średniej utylizacji hosta wygrywa algorytm PCK ale niewiele ustępuje mu MU. Polityka WPC wraz z polityką RR przedstawia najgorsze wyniki jeżeli chodzi o średnią utylizację hosta. Pod względem zużycia energii również dominują polityki MU i PCK. Nieco gorsza od nich jest polityka WPC. Pod względem liczby migracji dominuje polityka PCK. Natomiast pod względem średniej liczby aktywnych hostów prowadzi polityka MU. We wszystkich przypadkach pogwałcenia SLA znajdują się na zbliżonym poziomie, chociaż na uwagę zasługuje najlepszy wynik algorytmów PCK i WPC. Zdecydowanie najmniejszą liczbą pogwałceń SLA ze względu na migrację charakteryzuje się WPC.

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-lvf-mu	0.00061	0.00431	0.01895	0.02840
1-AF-thr-lvf-pck	0.00080	0.00603	0.01701	0.02417
1-AF-thr-lvf-rr	0.00043	0.00050	0.00516	0.00256
1-AF-thr-lvf-stp	0.00023	0.00042	0.03213	0.02632
1-AF-thr-lvf-wpc	0.00028	0.00045	0.01971	0.03316
1-AF-thr-lvf-wpca	0.00031	0.00092	0.02194	0.04442

Tabela 7: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM Largest VM First i dla wszystkich rozważanych algorytymów wyboru hosta docelowego

nazwa testu	średnia utylizacja HD	łączna ilość migracji	łączne zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	slatah	pdm
1-AF-thr-maxu-mu	0.68	14833	140.4433	47.14	27.22	0.06042	0.00193
1-AF-thr-maxu-pck	0.69	12795	139.9885	47.99	27.08	0.05685	0.00188
1-AF-thr-maxu-rr	0.62	22546	157.4006	56.86	21.96	0.06783	0.00181
1-AF-thr-maxu-stp	0.63	32350	162.9641	60.29	22.39	0.07675	0.00243
1-AF-thr-maxu-wpc	0.62	17001	147.1446	51.62	20.97	0.05996	0.00151

Tabela 8: Tabela przedstawiająca dane dotyczące algorytmu selekcji MAXU.

W tabeli 9 widać że najkorzystniejszym ze względu na średni czas selekcji VM jest algorytm RR, a na kolejnych miejscach znajdują algorytmy WPC i STP. Polityki PCK i MU znajdują się dwóch ostatnich miejscach. Za to jeżeli chodzi o średni czas wyznaczenia hosta docelowego do migracji, owe algorytmy prezentują się najbardziej korzystnie.

4.1.5 Algorytm selekcji maszyn wirtualnych MC

W tabelach 10 i 11 przedstawiono zbiorcze wyniki dla testów, które jako politykę wyboru maszyny wirtualnej do selekcji stosują Maximal Corellation i jako politykę wyboru hosta docelowego stosują wszystkie rozpatrywane algorytmy z tej kategorii. Pod względem głównego kryterium, czyli zużycia a energii tak jak w poprzednich opisywanych grupach testów dominują polityki MU i PCK, Przedstawiają one również najlepsze wyniki jeżeli chodzi o średnią utylizacje hosta i średnią ilość aktywnych hostów. Obie te polityki generują nieco większy poziom metryk obrazujących pogwałcenia SLA, od najkorzystniejszej pod tym względem polityki WPC. Tabela 11 ukazuje, że najkorzystniejszą ze względu na średni czas wyboru hosta docelowego (poza RR) są polityki MU i PCK. Niewiele ustępuje im polityk WPC. Pod względem średniego czasu selekcji VM wygrywa algorytm STP, który niestety jest najgorszy, jeżeli chodzi o czas selekcji HD. Reszta algorytmów pod tym względem nie wykazuje żadnych znaczących różnic.

4.1.6 Algorytm selekcji maszyn wirtualnych MINU

W tabelach 10 i 11 przedstawiono zbiorcze wyniki dla testów, które jako politykę wyboru maszyny wirtualnej do selekcji stosują Maximal Corellation a jako politykę wyboru hosta docelowego sto-

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-maxu-mu	0.00045	0.00182	0.01301	0.03166
1-AF-thr-maxu-pck	0.00056	0.00338	0.01240	0.03050
1-AF-thr-maxu-rr	0.00033	0.00047	0.00427	0.00183
1-AF-thr-maxu-stp	0.00035	0.00048	0.03026	0.02950
1-AF-thr-maxu-wpc	0.00029	0.00046	0.01639	0.03924

Tabela 9: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM Maximal Utilization of Vm i dla wszystkich algorytymów wyboru hosta docelowego

nazwa testu	średnia utylizacja HD	łączna ilość migracji	łączne zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	slatah	pdm
1-AF-thr-mc-mu	0.71	25406	140.9081	47.81	31.82	0.06435	0.00190
1-AF-thr-mc-pck	0.71	23140	138.6883	47.60	31.54	0.06668	0.00184
1-AF-thr-mc-rr	0.61	30904	160.1363	58.21	20.80	0.06716	0.00152
1-AF-thr-mc-stp	0.61	44991	170.2394	63.50	22.62	0.08581	0.00214
1-AF-thr-mc-wpc	0.58	23548	157.2010	56.20	20.10	0.06163	0.00122

Tabela 10: Tabela przedstawiająca dane dotyczące algorytmu selekcji MC.

sują wszystkie rozpatrywane algorytmy. Z danych zawartych w tabeli 12 wynika, że najlepszymi algorytmami jeżeli chodzi o energooszczędność są algorytmy MU i PCK. Wykazują się one również najwyższą średnią utylizacją hostów i najmniejszą średnią liczbą aktywnych hostów. W tej grupie najgorszym algorytmem jest algorytm WPC, który jest najgorszy pod względem energooszczędności, za to dominuje on jeżeli chodzi o liczbę migracji. Posiada on również jeden z najniższych poziomów pogwałceń SLA ze względu na migrację.

W tabeli 13 można zauważyć, że jednymi z najkorzystniejszych jeżeli chodzi o średni czas selekcji VM są algorytmy WPC i STP. Ze względu na średni czas wyznaczania hosta docelowego najkorzystniejszy jest algorytm WPC a zaraz za nim są MU i PCK. Należy jednak pamiętać że pomimo dobrych czasów algorytm WPC w tej konfiguracji sprawuje się wyjątkowo niekorzystnie pod względem energooszczędności.

4.1.7 Algorytm selekcji maszyn wirtualnych MMT

W tabelach 14 i 15 przedstawiono zbiorcze dane dotyczące testów, w których algorytmem selekcji maszyn wirtualnych do migracji jest był Minimal Migration Time, natomiast jako politykę wyboru węzła docelowego użyto wszystkich rozważanych algorytmów. W tabeli 14, tak jak we wszystkich poprzednich przypadkach, widać,że pod względem energooszczędności, średniej ilości aktywnych hostów, i średniej ich utylizacji algorytmy MU i PCK przedstawiają najlepsze wyniki. Najgorszym względem energooszczędności jest algorytm STP, który znacznie odbiega od pozostałych zarówno pod tym względem jak również pod względem ilości aktywnych hostów. Jego użycie powoduje również znaczny wzrost liczby migracji w stosunku do pozostałych polityk. Ze względu na poziom metryk dotyczących SLA dominuje polityka WPC. Zaraz po niej znajdują się polityki PCK i MU.

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-mc-mu	0.00988	0.00910	0.02214	0.02863
1-AF-thr-mc-pck	0.01228	0.00955	0.02226	0.02859
1-AF-thr-mc-rr	0.00961	0.00504	0.00545	0.00289
1-AF-thr-mc-stp	0.00647	0.00339	0.04264	0.03654
1-AF-thr-mc-wpc	0.00882	0.00510	0.02303	0.03588

Tabela 11: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM Maximal Corelation i dla wszystkich algorytymów wyboru hosta docelowego

nazwa testu	średnia utylizacja HD	łączna ilość migracji	łączne zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	slatah	pdm
1-AF-thr-minu-mu	0.72	45154	139.1113	47.03	36.35	0.06670	0.00182
1-AF-thr-minu-pck	0.72	41831	137.9137	47.54	36.99	0.06363	0.00176
1-AF-thr-minu-rr	0.60	44578	165.9403	60.98	20.35	0.07027	0.00124
1-AF-thr-minu-stp	0.61	60043	177.3917	67.09	21.53	0.08625	0.00163
1-AF-thr-minu-wpc	0.44	31038	189.2608	70.62	19.61	0.06628	0.00082

Tabela 12: Tabela przedstawiająca dane dotyczące algorytmu selekcji MINU.

Na podstawie tabeli 15 zaobserwowano fakt, że wszystkie stosowane polityki wyboru hosta docelowego mają przybliżone czasy wyboru HD (poza algorytmem RR). Ponadto po raz kolejny okazało się że stosowanie algorytmu WPC i RR wpływa na skrócenie średniego czasu migracji. Jednak różnice między nimi są tak niewielkie, że wydają się nie mieć znaczenia.

4.1.8 Algorytm selekcji maszyn wirtualnych RR

W tabelach 16 i 17 zbiorcze wyniki testów, w których polityką stosowaną do wyboru maszyny wirtualnej do migracji była polityka Random Selection a jako politykę wyboru hosta docelowego użyto każdą z rozpatrywanych. Pod względem energooszczędności najlepsze wyniki uzyskały algorytmy MU i PCK, a na trzecim miejscu znalazł się algorytm WPC. Pod względem utylizacji hosta jest on najgorszy, ale jego stosowanie skutkuje najmniejszą liczbą migracji. Ten algorytm jest również najbardziej korzystny jeżeli chodzi o średni czas migracji maszyn wirtualnych. Wszystkie polityki migracji wykazują się podobną wysokością metryk badających poziom pogwałceń SLA.

Z tabeli 17 wynika, że najlepszym czasem wyboru hosta docelowego (poza RR) charakteryzuje się algorytm MU, zaś najdłuższym algorytm STP. Czasy selekcji maszyn wirtualnych są bardzo zbliżone ale najgorszy charakteryzuje algorytm PCK.

4.2 Porównanie polityk wyboru maszyny wirtualnej do migracji.

Poniżej zamieszczone są tabele zestawiające wyniki zgrupowane pod względem algorytmu wyboru hosta docelowego.

Tabele 18 i 19 przedstawia zbiorcze wyniki dla symulacji trzech najefektywniejszych pod względem energooszczędności algorytmów selekcji VM.

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-minu-mu	0.00157	0.00609	0.04371	0.02877
1-AF-thr-minu-pck	0.00287	0.01020	0.04342	0.03067
1-AF-thr-minu-rr	0.00109	0.00074	0.00710	0.00489
1-AF-thr-minu-stp	0.00059	0.00051	0.06110	0.03937
1-AF-thr-minu-wpc	0.00041	0.00051	0.03859	0.03533

Tabela 13: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM Minimal Utilization of Vm i dla wszystkich algorytymów wyboru hosta docelowego

nazwa testu	średnia utylizacja HD	łączna ilość migracji	łączne zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	slatah	pdm
1-AF-thr-mmt-mu	0.71	30060	140.4176	47.50	28.85	0.05392	0.00162
1-AF-thr-mmt-pck	0.71	27805	138.3403	47.55	29.35	0.05283	0.00153
1-AF-thr-mmt-rr	0.60	36772	163.4058	59.81	16.29	0.05687	0.00121
1-AF-thr-mmt-stp	0.59	54932	178.8559	67.39	20.11	0.08138	0.00201
1-AF-thr-mmt-wpc	0.52	27260	168.2598	61.20	14.07	0.04646	0.00083

Tabela 14: Tabela przedstawiająca dane dotyczące algorytmu selekcji Minimal Migration Time.

4.2.1 Algorytm wyboru hosta docelowego Maximal Utilization

Z pierwszej części tabeli 18 wynika, że pod względem energooszczędności z algorytmem MU najlepiej współpracuje polityka MINU, a na kolejnych miejscach są MMT i MAXU. Widać tutaj, że poziomy zużycia energii dla wszystkich polityk są zbliżone. Średnia utylizacja jest najwyższa dla polityki MINU tak samo jak średnia ilość aktywnych hostów. Na kolejnych miejscach pod tym względem znajdują się MMT i MMC. Minimalną liczbę migracji powoduje użycie polityki MAXU albo LVF, a poziom pogwałceń SLA dla wszystkich algorytmów znajduje się na tym samym poziomie - między 5.5% a 6.6%. Najkrótszym czasem selekcji VM charakteryzują się RS, LVF i MAXU, a najkrótszy czas wyboru HD ma CUV, LVF, MAXU.

4.2.2 Algorytm wyboru hosta docelowego Packing

W drugiej części tabeli 18 mozna zauważyć, że polityka selekcji hosta docelowego PCK najbardziej energooszczędnie działa w połączeniu z politykami selekcji maszyn wirtualnych MMT, MC i CTH. Pod względem średniej utylizacji i średniej ilości aktywnych węzłów najlepiej wypadają w tym przypadku MC i MMT. Najmniejszą liczbą migracji skutkuje użycie polityki MAXU lub CUV. Co ciekawe pod względem średniego czasu migracji najlepiej prezentuje się polityka MAXU. Kryteria odpowiadające za jakość usług są na jednakowym poziomie dla wszystkich algorytmów selekcji VM. Najkrótszym czasem selekcji VM odznaczają się CTH jak również LVF i MAXU. Najkrótszy czas wyboru hosta docelowego migracji uzyskuje dle tego przypadku polityka CUV lub MAXU.

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-mmt-mu	0.00089	0.00555	0.02672	0.02790
1-AF-thr-mmt-pck	0.00119	0.00828	0.02697	0.02877
1-AF-thr-mmt-rr	0.00069	0.00053	0.00614	0.00394
1-AF-thr-mmt-stp	0.00036	0.00049	0.05269	0.03285
1-AF-thr-mmt-wpc	0.00036	0.00048	0.03032	0.03790

Tabela 15: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM Minimal Migration Time i dla wszystkich algorytymów wyboru hosta docelowego

nazwa testu	średnia utylizacja HD	łączna ilość migracji	łączne zużycie mocy	średnia ilość aktywnych hostów	średni czas migracji	slatah	pdm
1-AF-thr-rs-mu	0.71	26057	141.1243	47.86	31.90	0.06578	0.00191
1-AF-thr-rs-pck	0.71	23814	139.3251	47.96	32.64	0.06381	0.00191
1-AF-thr-rs-rr	0.61	31377	162.3699	59.49	20.71	0.06808	0.00153
1-AF-thr-rs-stp	0.62	43363	167.9107	62.46	22.43	0.07105	0.0020192
1-AF-thr-rs-wpc	0.56	23196	159.2264	57.13	20.14	0.06180	0.00123

Tabela 16: Tabela przedstawiająca dane dotyczące algorytmu selekcji RS.

4.2.3 Algorytm wyboru hosta docelowego Watts Per Core

W ostatniej części tabel 18 i 19 przedstawiono dane zbiorcze dotyczące polityki selekcji hosta docelowego Watts Per Core wraz ze wszystkimi rozpatrywanymi politykami wyboru maszyny wirtualnej. Najbardziej energooszczędne w tym przypadku okazują się polityki CTH, LVF i MAXU. Pod względem utylizacji hosta jak również ilości aktywnych hostów najlepsze są CUV i MAXU. Najmniejszą liczbą migracji skutkuje użycie polityk CUV i MAXU. Średni czas migracji jest najkrótszy przy użyciu polityki LVF. Pod względem czasów wykonywania algorytmu selekcji HD czasy wyboru różnią się nieznacznie. Najgorsze są w przypadku MINU i MMT. Wynika to prawdopodobnie z faktu, iż są w nich migrowane dosyć duże ilości maszyn wirtualnych. Czas selekcji maszyn wirtualnych jest również zbliżony, nie licząc algorytmu CUV, którego średni czas wyboru znacznie odbiega od innych pomimo stosunkowo niewielkiej liczby migracji. Minimalnym zużyciem energii spośród polityk dostępnych wcześniej w bibliotece CloudSim charakteryzuje się kombinacja algorytmów MC-WPC.

5 Wnioski

Analizując przedstawione w poprzednim rozdziale wyniki mozna łatwo zauważyć, że dużo większy wpływ na energooszczędność ma polityka wyboru hosta docelowego do migracji, niż polityka selekcji maszyny wirtualnej.

Potwierdzenie znajduje również intuicyjne twierdzenie, że najważniejszym aspektem jeżeli chodzi o energooszczędność jest maksymalizowanie średniej utylizacji hostów w celu umożliwienia wyłączenia jak największej liczby nieużywanych. Potwierdzają to wyniki testów, w których liczba

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-rs-mu	0.00072	0.00484	0.02292	0.02609
1-AF-thr-rs-pck	0.00102	0.00779	0.02492	0.02905
1-AF-thr-rs-rr	0.00048	0.00058	0.00542	0.00365
1-AF-thr-rs-stp	0.00038	0.00049	0.04059	0.03174
1-AF-thr-rs-wpc	0.00032	0.00047	0.02444	0.04678

Tabela 17: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla algorytmu selekcji VM Random Selection i dla wszystkich algorytmów wyboru hosta docelowego

aktywnych hostów w znacznej mierze odpowiada późniejszemu poziomowi zużycia energii.

Podczas przeprowadzania symulacji zaobserwowano również fakt, że zbyt agresywne dobranie polityki wyboru hosta niedociążonego powoduje oscylacje algorytmów, które starają się alokować w pełni pojedyncze hosty, dopiero później przechodząc do następnych.

Jeżeli chodzi o polityki wyboru maszyny wirtualnej, to te wykorzystane w powyższych eksperymentach należy indywidualnie dobierać do danego algorytmu selekcji HD, ponieważ nie dla każdego algorytmu selekcji hosta docelowego ta sama polityka selekcji VM okazuje się tą najlepszą. Spowodowane może to być rozdzieleniem tych dwóch faz realokacji, co sprawia, że w momencie wybierania maszyny wirtualnej nie wiadomo, na którą maszynę fizyczną zostanie ona przeniesiona.

Zaobserwowano również, że największy wpływ na minimalizację zużycia energii ma liczba aktywnych hostów. A w związku z tym największą oszczędność dają te algorytmy które maksymalizują utylizacje hostów.

Najgorsze wyniki pod względem energooszczędności daje algorytm selekcji hosta docelowego STP (niezależnie od wybranej polityki selekcji VM do migracji). Spowodowane jest to sposobem działania algorytmu, który stara się przenieść wyznaczone wcześniej maszyny wirtualne na jak największą ilość hostów (aktywnych bądź nie), co zwiększa liczbę hostów aktywnych. Jednocześnie ze względu na niskie obciążenie części hostów podejmowanajest kolejna próba realokacji.

Najlepszymi algorytmami ze względu na energooszczędność są algorytmy PCK i MU. Nieznacznie odbiega od nich dostępny w środowisku CloudSim algorytm WPC. Niestety pomimo lepszej średniej utylizacji hosta charakteryzują się one większą częstością zmian liczby aktywnych hostów niż algorytm WPC, nie wpływa to jednak niekorzystnie na zużycie energii. Algorytmy MU i PCK są również najkorzystniejsze ze względu na liczbę migracji. Polityki te są również jednymi z lepszych jeżeli chodzi o czas migracji.

Wadą tych algorytmów jest gwałtowność zmian liczby aktywnych hostów, co wynika z sposobu w jaki algorytm wypełnia hosty maszynami wirtualnymi. Mianowicie przy realokacji oba z tych algorytmów starają się najpierw zapełnić jednego hosta. Potem dopiero zapełniają następnego, itd. W połączeniu ze sposobem wyznaczania hostów niewystarczająco obciążonych, który polega na tym, że jako niewystarczająco obciążone uznawane są hosty, na które nie są migrowane żadne VM i których utylizacja jest większa od zera, powoduje to, że w wielu cyklach jako niewystarczająco obciążone wybierane są hosty o dużym poziomie utylizacji, następuje próba ich wyłączenia i zmigrowania znajdujących się na nich maszyn wirtualnych. Algorytmy WPC i RR działają inaczej. Starają się one równomiernie wypełniać hosty maszynami wirtualnymi, a co za tym idzie mniej z hostów jest uznawanych za niedociążone i nie są one tak często wyłączane. Dlatego można w nich zaobserwować mniejszą częstość zmian ilości aktywnych hostów.

Wysokość metryki PDM jest proporcjonalna do poziomu spadków efektywności wykonywania zadania podczas migracji maszyny wirtualnej. Maksymalizacja obciążeń hostów prowadzi bezpośrednio do zwiększania ryzyka podniesienia poziomu metryki SLATAH.

Z powyższych rozważań wynika, że aby zminimalizować zużycie energii należy przede wszystkim skupić się na wyborze algorytmu selekcji hosta docelowego, który będzie maksymalizował jego utylizacje. Następnie należy indywidualnie rozważyć wybór polityki selekcji dla każdego z nich. Z przeprowadzonych badań wysnuto wnioski, że najbardziej nadającymi się do tego celu algorytmami

nazwa testu	średnia utyli-zacja HD	łączna ilość migra-cji	łączne zużycie mocy	średnia ilość aktyw-nych hostów	średni czas migra-cji	slatah	pdm
1-AF-thr-cth-mu	0.71	25485	141.1417	47.85	31.84	0.06495	0.00190
1-AF-thr-cuv-mu	0.68	15716	142.1317	48.03	27.20	0.06608	0.00200
1-AF-thr-lvf-mu	0.70	20911	142.0881	48.30	28.09	0.05737	0.00178
1-AF-thr-maxu-mu	0.68	14833	140.4433	47.14	27.22	0.06042	0.00193
1-AF-thr-mc-mu	0.71	25406	140.9081	47.81	31.82	0.06435	0.00190
1-AF-thr-minu-mu	0.72	45154	139.1113	47.03	36.35	0.06670	0.00182
1-AF-thr-mmt-mu	0.71	30060	140.4176	47.50	28.85	0.05392	0.00162
1-AF-thr-rs-mu	0.71	26057	141.1243	47.86	31.90	0.06578	0.00191
1-AF-thr-cth-pck	0.71	22650	139.3931	48.06	31.58	0.06777	0.00182
1-AF-thr-cuv-pck	0.68	13053	140.7186	48.35	27.56	0.06086	0.00186
1-AF-thr-lvf-pck	0.70	19141	140.4795	48.60	28.06	0.05571	0.00173
1-AF-thr-maxu-pck	0.69	12795	139.9885	47.99	27.08	0.05685	0.00188
1-AF-thr-mc-pck	0.71	23140	138.6883	47.60	31.54	0.06668	0.00184
1-AF-thr-minu-pck	0.72	41831	137.9137	47.54	36.99	0.06363	0.00176
1-AF-thr-mmt-pck	0.71	27805	138.3403	47.55	29.35	0.05283	0.00153
1-AF-thr-rs-pck	0.71	23814	139.3251	47.96	32.64	0.06381	0.00191
1-AF-thr-cth-wpc	0.59	23659	153.8972	54.70	20.44	0.06244	0.00120
1-AF-thr-cuv-wpc	0.63	16125	145.6639	50.94	21.32	0.06038	0.00152
1-AF-thr-lvf-wpc	0.59	21804	153.9159	54.70	18.05	0.05440	0.00114
1-AF-thr-maxu-wpc	0.62	17001	147.1446	51.62	20.97	0.05996	0.00151
1-AF-thr-mc-wpc	0.58	23548	157.2010	56.20	20.10	0.06163	0.00122
1-AF-thr-minu-wpc	0.44	31038	189.2608	70.62	19.61	0.06628	0.00082
1-AF-thr-mmt-wpc	0.52	27260	168.2598	61.20	14.07	0.04646	0.00083
1-AF-thr-rs-wpc	0.56	23196	159.2264	57.13	20.14	0.06180	0.00123

Tabela 18: Tabela przedstawiająca dane dotyczące wszystkich polityk selekcji VM dla trzech najlepszych algorytmów wyboru hosta docelowego (MU, PCK i WPC).

są PCK i MU.

Jeżeli chodzi o algorytmy selekcji należy pamiętać że mają one znacznie mniejsze znaczenie. Jednak porównując dane liczbowe zawarte w tabeli 18 można dojść do wniosku że najlepiej z algorytmem PCK działają polityki MMT i MC, a najlepiej z algorytmem MU działają polityki MMT, MC i MINU. Z tego można wnioskować, że dla tych dwóch algorytmów wyboru hosta docelowego najlepiej jeżeli chodzi o kryterium energooszczędności działają polityki selekcji maszyny wirtualnej MMT i MINU.

Inne kryteria takie jak poziom SLATAH i PDM są na zbliżonym poziomie pomiędzy 4.6% a 6.6% jeżeli chodzi o SLATAH i pomiędzy 0.08% a a 0.2% dla PDM. Jest to niewielka rozbieżność zarówno jeżeli chodzi o SLATAH jak i PDM. Dodatkowo zwycięska pod względem energooszczędności kombinacja, czyli MMT-PCK charakteryzuje się poziomami tych metryk w wysokości odpowiednio 5.2& i 0.15%

Podsumowując, w wyniku przeprowadzonych eksperymentów jako najlepszą wybrano kombinacje polityk MMT-PCK. Dalsza poprawa działania algorytmu jest możliwa przede wszystkim na drodze poszerzania modelu zużycia mocy. W pierwszej kolejności o nowe elementy takie jak zużycie energii przez RAM, magazyny danych itp. Kolejnym krokiem mogłoby być wzięcie pod uwagę energii zużywanej na chłodzenie serwerowni i znajdujących się w niej urządzeń, tak aby uzyskać całosciowy model instalacji. Na podkreślenie zasługuje względna szybkośc działania badanych algorytmów. Daje to nadzieję na ich zastosowanie w wysokoskalowalnych systemach klastrowych.

nazwa testu	średni czas selekcji VM	odch stand. czasu selekcji VM	średni czas selekcji HD	odch. stand. czasu selekcji HD
1-AF-thr-cth-mu	0.00062	0.00150	0.02618	0.04326
1-AF-thr-cuv-mu	1.69447	1.56408	0.01467	0.02894
1-AF-thr-lvf-mu	0.00061	0.00431	0.01895	0.02840
1-AF-thr-maxu-mu	0.00045	0.00182	0.01301	0.03166
1-AF-thr-mc-mu	0.00988	0.00910	0.02214	0.02863
1-AF-thr-minu-mu	0.00157	0.00609	0.04371	0.02877
1-AF-thr-mmt-mu	0.00089	0.00555	0.02672	0.02790
1-AF-thr-rs-mu	0.00072	0.00484	0.02292	0.02609
1-AF-thr-cth-pck	0.00082	0.00360	0.02205	0.02541
1-AF-thr-cuv-pck	1.75617	1.61284	0.01258	0.02983
1-AF-thr-lvf-pck	0.00080	0.00603	0.01701	0.02417
1-AF-thr-maxu-pck	0.00056	0.00338	0.01240	0.03050
1-AF-thr-mc-pck	0.01228	0.00955	0.02226	0.02859
1-AF-thr-minu-pck	0.00287	0.01020	0.04342	0.03067
1-AF-thr-mmt-pck	0.00119	0.00828	0.02697	0.02877
1-AF-thr-rs-pck	0.00102	0.00779	0.02492	0.02905
1-AF-thr-cth-wpc	0.00044	0.00051	0.02433	0.04343
1-AF-thr-cuv-wpc	2.62470	2.45722	0.01414	0.03670
1-AF-thr-lvf-wpc	0.00028	0.00045	0.01971	0.03316
1-AF-thr-maxu-wpc	0.00029	0.00046	0.01639	0.03924
1-AF-thr-mc-wpc	0.00882	0.00510	0.02303	0.03588
1-AF-thr-minu-wpc	0.00041	0.00051	0.03859	0.03533
1-AF-thr-mmt-wpc	0.00036	0.00048	0.03032	0.03790
1-AF-thr-rs-wpc	0.00032	0.00047	0.02444	0.04678

Tabela 19: Tabela przedstawiająca średnie czasy wykonania i odchylenia standardowe dla trzech najlepszych pod względem efektywności algorytmów selekcji hosta docelowego.

Literatura

[1] Raykumar Buuya and Anton Beloglazov. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, pages 755 – 768, 2012.

[2] Raykumar Buuya and Anton Beloglazov. Optimal online deterministic algorithms and adaptive heuristics for energy and performance. *Concurr. Comput. Pract. Exp.*, pages 1397–1420, 2012.

[3] The Cloud Computing and University of Melbourne Distributed Systems (CLOUDS) Laboratory. Cloudsim documentation.

[4] Dragos Diaconescu, Florin Pop, and Valentin Cristea. Energy-aware placement of vms in a datacenter. pages 1–6, 2012.

[5] Fahimeh Farahnakian, Adnan Ashraf, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, Ivan Porres, and Hannu Tenhunen. Using ant colony system to consolidate vms for green cloud computing. pages 1–12, 2015.

[6] N. Janani, R.D. Shiva Jegan, and P. Prakash. Optimization of virtual machine placement in cloud environment using genetic algorithm. *Research Journal of Applied Sciences*, 2015.

[7] Mohan Raj Velayudhan Kumar and Shriram Raghunathan. Heterogeneity and thermal aware adaptive heuristics for energy efficient consolidation of virtual machines in infrastructure clouds.mohanrajvelayudhankumar. *Journal of Computer and System Sciences*, pages 191–121, 2016.

[8] Patrick Raycroft, Rayan Jansen, Mateusz Jarus, and Paul R. Brenner. Performance bounded energy efficient virtual machine allocation in the global cloud. *Sustainable Computing Informatics Systems*, pages 1–9, 2014.

[9] Gamal Eldin I. Selim1, Mohamed A. El-Rashidy, and Nawal A. El-Fishawy. An efficient resource utilization technique for consolidation of virtual machines in cloud computing environments. *33rd National Radio Science Conference*, pages 1–9, 2016.

[10] Perla Ravi Theja1 and S. K. Khadar Babu. An adaptive genetic algorithm based robust qos oriented green computing scheme for vm consolidation in large scale cloud infrastructures. *Indian Journal of Science and Technology,*, pages 1–13, 2015.

[11] Akshat Verma, Gargi Dasgupta, Tapan Kumar, Nayak Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using cons olidation. *IBM India Research Lab*, pages 5–6, 2011.

[12] Xiangliang Zhang, Zon-Yin Shae, Shuai Zheng, and Hani Jamjoom. Virtual machinemigration in an over-committed cloud. *IBM T. J. Watson Research Center*, pages 3–5, 2012.