GDAŃSK UNIVERSITY OF TECHNOLOGY

Imię i nazwisko autora rozprawy: Piotr Orzechowski
Dyscyplina naukowa: Informatyka Techniczna i Telekomunikacja

ROZPRAWA DOKTORSKA

Tytuł rozprawy w języku polskim: Wykorzystanie klasyfikacji funkcjonalnej usług do efektywnego zarządzania zasobami chmurowymi

Tytuł rozprawy w języku angielskim: Application of workloads' functional classification for effective management of cloud resources

Promotor *podpis*
prof. dr hab. inż. Henryk Krawczyk
Promotor pomocniczy *podpis*
dr hab. inż. Jerzy Proficz, prof. PG

Gdańsk, 2023

Streszczenie

Wykazano jak istotnym problemem jest zarządzanie chmurą obliczeniową, w tym alokacja zasobów do wykonania usług (workloadów) zgłoszonych przez użytkownika. Przeanalizowano problem podziału usług wdrażanych w środowiskach chmurowych na klasy określające ich funkcjonalność. Zaproponowano oryginalną metodę alokacji workloadów wykorzystującą wprowadzoną klasyfikację funkcjonalną oraz identyfikację tych klas na podstawie wielkości generowanego obciążenia zasobów chmurowych Na tej podstawie zaproponowano nowy mechanizm zarządzania: algorytm Complementarity Scheduling Algorithm (CSA), który pozwala na alokacje workloadów na zasoby chmurowe zwiększając efektywność przetwarzania całego systemu. Przedstawiono porównanie algorytmu CSA z typowym algorytmem zrównoważonego obciążenia stosowanym np. na platformie OpenStack.

Abstract

The significance of the computing cloud management has been shown in the present dissertation, including a vital role of the allocation of resources for user-requested workloads. The problem of classification of workloads deployed in cloud environments based on their functionality was analyzed. An original method of workload allocation was proposed, in which both the introduced functional classification and the classes' identification on the basis of generated load of cloud resources are used. Owing to the proposed method, a new management mechanism was developed: the CSW algorithm, which allows for the allocation of workloads to cloud resources, increasing the processing efficiency of the entire system. Furthermore, a comparison between the CSW algorithm and a typical load balancing algorithm used e.g. on the OpenStack platform was presented.

Spis treści

Wykaz pojęć

alokacja proces przydziału zasobów wirtualnych na węźle chmury obliczeniowej do maszyn wirtualnych (utworzonych na rzecz wykonania workloadu)

aplikacja użytkowa program lub programy wykonywalne wraz z powiązanymi bibliotekami realizujące konkretną funkcjonalność

dostawca chmury obliczeniowej organizacja udostępniająca usługę chmury obliczeniowej

dyspozytor komponent odpowiedzialny w chmurze obliczeniowej za alokację workloadów

klasy funkcjonalne workloadów grupy workloadów realizujące podobną funkcjonalność

klasyfikacja workloadu proces przydzielający workload do klasy funkcjonalnej workloadów na podstawie danych o wykorzystanych przez niego zasobach

klient chmury obliczeniowej osoba lub organizacja korzystająca z usług wdrożonych na chmurze obliczeniowej

komplementarność klas workloadów cecha określająca jaki wzajemny wpływ na efektywność wykonania mają na siebie workloady uruchomione na jednym węźle chmury

konfiguracja workloadu zestaw parametrów użytkownika zdefiniowany przy uruchamianiu workloadu

model chmury obliczeniowej sposób świadczenia usługi chmurowej opisujący rodzaj usług udostępnianych użytkownikom chmurowym

obraz maszyny wirtualnej plik z oprogramowaniem dostarczający system operacyjny i preinstalowane oprogramowanie przygotowany do uruchomienia na maszynie wirtualnej w chmurze obliczeniowej

parametry użytkownika zestaw parametrów opisujący oczekiwane obciążenie klasy workloadu, specyficzny dla każdej klasy

symulator workloadu program imitujący działanie rzeczywistego workloadu z predefiniowanym zestawem danych wejściowych i zestawem zadań do wykonania

TASKcloud usługa chmurowa w modelu IaaS dostarczana przez Centrum Informatyczne TASK

umowa SLA umowa pomiędzy dostawcą usługi a jej klientem stanowiąca o zobowiązaniach dostawcy wobec klienta w zakresie świadczonej usługi m.in. o jej dostępności, jakości działania oraz możliwych konsekwencjach niespełnienia określonych warunków umowy

usługa rozwiązanie informatyczne realizujące pewną funkcjonalność w oparciu o aplikację użytkową

użytkownik chmury obliczeniowej osoba lub organizacja korzystająca z usług chmurowych

workload praca wraz z zasobami i danymi niezbędnymi do jej wykonania; może to być usługa, aplikacja czy też zadanie, które wykorzystują pulę zasobów chmurowych wraz z danymi wejściowymi i wyjściowym

węzeł chmury (inaczej węzeł obliczeniowy, węzeł fizyczny) serwer w chmurze obliczeniowej, na którym uruchamiane są workloady

zarządzanie zasobami zespół procesów polegająca na odpowiedniej dbałości o zasoby, utrzymanie wysokiego poziomu ich dostępności oraz bezpieczeństwa, a także obejmujący przydzielanie ich użytkownikom

zasób pojemność komponentu sprzętowego (fizycznego lub wirtualnego, np. procesora, pamięci RAM, karty sieciowej), która może zostać wykorzystana w jednostce czasu przez workload uruchomiony na węźle obliczeniowym wyposażonym w ten komponent

Wykaz skrótów

API	ang. Application Programming Interface
CE	ang. Complementarity Effecitveness
CNW	ang. Complementarity Node Weigher
CONS	ang. consolidation
CSA	ang. Complementarity Scheduling Algorithm
ECS	ang. Environment Complementary Scorer
FT	ang. fault tolerance
GB	gigabajt
Gb	gigabit
GFLOPS	gigaflops
HPC	ang. High Performance Computing
HTTP	ang. Hypertext Transfer Protocol
IaaC	ang. Infrastructure as a Code
IaaS	ang. Infrastructure as a Service
IdP	ang. Identity Provider
LB	ang. load balancing

MIPS ang. Million instructions per second

OLTP ang. Online Transaction Processing

PaaS ang. Platform as a Service

PCA ang. Principal Component Analysis

QoS ang. Quality of Service

SaaS ang. Software as a Service

SDK ang. Software Development Kit

SLA ang. Service Level Agreement

SQL ang. Structured Query Language

VMs ang. virutal machines

Wykaz oznaczeń

C	zbiór klas funkcjonalnych workloadów
$C(w_i)$	funkcja określająca klasę workloadu w_i
$E(c_i)$	zbiór miar efektywności wykonania workloadów klasy c_i
F	zbiór filtrów węzłów obliczeniowych
N	zbiór węzłów chmurowych
R	zbiór zasobów wirtualnych
S	zbiór funkcji oceniających
W	zbiór workloadów
$\overline{s_i}$	znormalizowana do przedziału $[0;1]$ wartość funkcji oceniającej s_i
$a(w)$	funkcja alokacji workloadu w przydzielająca węzeł n, na który ma być zaalokowany workload w
$a^{-1}(n)$	funkcja odwrotna do funkcji alokacji, określająca zbiór workloadów zaalokowanych na węźle n
c_i	klasa funkcjonalna workloadu, $c_i \in C$
e	miara efektywności wykonania workloadu
f_i	funkcja węzła obliczeniowego i, $f_i \in F$
n	węzeł chmury obliczeniowej $n \in N$
$ns(n)$	punktowa wartość oceny węzła n
$rn(n)$	odwzorowanie określające zasoby wirtualne dostępne na węźle ob-

	liczeniowym n
$ru(n)$	odzworowanie określające zasoby wirtualne przydzielone (zarezerwowane) dla workloadów zaalokowanych na węźle n
$rw(w)$	odzworowanie określające zasoby wirtualne wymagane przez workload w
s_i	funkcja oceniająca i, $s_i \in S$
$scoreWeight(s_i)$	waga oceny funkcji s_i
w	workload $w \in W$
|N|	liczba węzłów chmurowych
|W|	liczba workloadów

Rozdział 1

Wprowadzenie

Postęp technologiczny i rozwój usług IT spowodował znaczny wzrost w świadczeniu usług chmurowych wśród różnego rodzaju organizacji. Tradycyjne rozwiązanie związane z posiadaniem własnej infrastruktury informatycznej, są zastępowane przez wynajem usług od instytucji specjalizujących się w dostarczaniu i utrzymaniu infrastruktur chmurowych. Pozwala to w znaczący sposób obniżyć organizacji koszty informatyzacji.

Istnieją różne modele świadczenie usług chmurowych, z czego trzy najbardziej popularne polegają na dostarczeniu wirtualnej infrastruktury (IaaS), platform (PaaS) oraz usług (SaaS). Niniejsza rozprawa doktorska skupia się na najniższej warstwie tj. modelu infrastruktury w formie usługi (IaaS) jednak w różnych aspektach uwzględnia również pozostałe warstwy. W modelu IaaS dostawca usług chmurowych chcąc optymalizować koszty szuka rozwiązań pozwalających na efektywne zarządzanie zasobami. Użytkownik z kolei jest w stanie określić jakie usługi będzie chciał wdrożyć na oferowanych mu zasobach chmurowych. Jednak często nie jest w stanie oszacować rozmiaru tych zasobów co prowadzi do zajmowania zwiększonego rozmiaru zasobów i ich częściowego niewykorzystywania tzw. *overprovisioning*. Z drugiej strony dostawca usług chmurowych chcąc optymalizować koszty i mając wiedzę, że zasoby użytkownika są przeszacowywane, zwiększa rozmiary wirtualnych zasobów powyżej faktycznie istniejących zasobów fizycznych tzw. *overcommitment*. Pozwala to na sprzedaż większej liczby zasobów bez rozbudowy infrastruktury chmurowej, jednak

może to prowadzić do niedotrzymywania umów z użytkownikami zwłaszcza w obszarze wydajności obliczeń chmurowych.

Różnorodność usług uruchamianych w chmurach obliczeniowych powoduje zróżnicowanie w obciążeniu oferowanych zasobów. Część zasobów, które nie są możliwe do wyizolowania przez dostawcę mogą być obiektem, o który wywoływane usługi będą rywalizować, co w konsekwencji wpłynie negatywnie na efektywność ich wykonania. Istnieje potrzeba zaradzania takim niekorzystnym sytuacjom. Pojawiające się w literaturze propozycje klasyfikacji usług chmurowych na podstawie wykorzystywanych przez nie zasobów infrastruktury ukazują potencjalne możliwości wykorzystania takiej wiedzy w procesie alokacji usług. Użytkownicy zazwyczaj posiadają wiedzę jakiego typu usługa będzie wdrożona w chmurze, natomiast nie mają precyzyjnej informacji o wielkości niezbędnych zasobów do wykonania tego typu usług.

W związku z powyższym w rozprawie doktorskiej rozważa się wprowadzenie klasyfikacji funkcjonalnej usług uruchamianych w chmurze oraz weryfikację możliwości wykrywania klasy tych usług na podstawie wykorzystywanych przez nie zasobów chmury. Jednocześnie proponuje się metodę weryfikacji możliwości wykorzystania przyjętej klasyfikacji funkcjonalnej w algorytmach alokacji w celu efektywniejszego (wg różnych kryteriów) wykorzystania zasobów chmurowych.

Główna teza rozprawy jest następująca: **Proponowana klasyfikacja funkcjonalna usług chmurowych (i odpowiadających im tzw. workloadów) pozwala na zwiększenie efektywności wykorzystania zasobów chmury obliczeniowej**. Tezę tę potwierdzono na podstawie wyników wielu eksperymentów zrealizowanych na chmurze TASKcloud, której głównym jednym z głównych projektantów i wykonawców był autor tej rozprawy doktorskiej. Chmura ta została zrealizowana przy wykorzystaniu otwartego oprogramowania (OpenStack) rozwijanego przez wielkich potentantów IT (Intel, Red Hat, Rackspace), co sprawia, że otrzymane wyniki są uniwersalne i dotyczyć mogą innych rozwiązań chmurowych.

Rozprawa zorganizowana jest w następujący sposób. W rozdziale 2 przedstawiono przegląd istniejących rozwiązań chmurowych oraz lokalne rozwiązanie chmurowe TASKcloud, na której wykonano większość eksperymentów badawczych. Rozdział 3

ukazuje problematyki efektywnego zarządzania chmurą obliczeniową. Proponowaną klasyfikację funkcjonalną oraz sposób jej wyznaczenia opisuje rozdział 4. Rozdział 5 przedstawia proponowany algorytm alokacji z wykorzystaniem klasyfikacji funkcjonalnej oraz wyniki jego działania. W rozdziale 6 podsumowano wyniki rozprawy doktorskiej oraz zasygnalizowano kierunki dalszych prac badawczych.

Rozprawa doktorska została częściowo zrealizowana w ramach prac w projektach: Komponent Rekomendacji dla Inteligentnych Chmur Obliczeniowych nr INNOTECH K3/IN3/20/227103/NCBR/14 oraz Centrum Kompetencji Smart Transdisciplinary knOwledge Services nr RPPM.01.02.00-22-0001/17.

Rozdział 2

Przegląd podstawowych rozwiązań chmurowych

Przedstawiono klasyfikację chmur obliczeniowych oraz modele świadczenia usług oferowanych w tego rodzaju infrastrukturach informatycznych. Zaprezentowano popularne rozwiązania chmurowe prezentowane przez liderów w obszarach IT. Na tym tle zaprezentowano autorską chmurę TASKcloud zbudowaną w oparciu o otwarte oprogramowanie oraz jej rozwój z wykorzystaniem technologii DevOps i elementów zwinnych metodyk wytwarzania oprogramowania. Chmura TASKcloud jest podstawą przedstawionych w rozprawie badań związanych z zarządzaniem zasobami chmurowymi.

2.1 Tendencje rozwoju chmury obliczeniowej

Rozwój technologii informatycznych wraz z rosnącą w ostatnich dekadach tendencją globalizacji przyczyniły się do zmian sposobu wykorzystania infrastruktury informatycznej. Tradycyjne rozwiązania koncentrują się na rozwiązaniach lokalnych (ang. *on-premise*), na które składają się stacje robocze i serwery połączone lokalną siecią LAN umożliwiającą uprawnionym użytkownikom na korzystanie z jej zasobów. Schemat ten umożliwia pracownikom korzystanie z różnego rodzaju usług: dyski współdzielone, serwery www, serwery usługowe i aplikacyjne. Dostępy dla

kontrahentów lub pracowników z poza fizycznej lokalizacji organizacji udostępniany jest w sieci Internet w postaci usług publicznych lub częściej, w celu zapewnienia większego bezpieczeństwa, za pomocą wirtualnych sieci prywatnych (VPN). W zależności od uprawnień przydzielany jest odpowiedni dostęp do wybranych usług i zasobów. Opisywany model wymaga jednak od organizacji podjęcia szeregu działań oraz poniesienia kosztów związanych z utrzymaniem posiadanej infrastruktury informatycznej. Głównym tego elementem jest własna serwerownia wraz z infrastrukturą pomocniczą. Konieczne jest dostarczenie stabilnego źródła zasilania z awaryjnym podtrzymaniem w przypadku zaników napięcia, wydajna klimatyzacja i odpowiednio przystosowane pomieszczenia, w których umieszczana jest ta infrastruktura, co wymaga dużych nakładów finansowych. Dodatkowo niezbędne jest utrzymanie zespołu specjalistów, którzy są w stanie zadbać o taką infrastrukturę i właściwie ją eksploatować. Kolejnym elementem, o który organizacja musi zadbać są specjaliści z zakresu zarządzania i utrzymania infrastruktury informatycznej tj. warstwy sprzętowej, zarówno sieciowej jak i serwerowej, oraz warstwy oprogramowania tj. systemów informatycznych. Utrzymanie takiej infrastruktury sieciowej i sprzętowej wymaga ciągłego monitorowania i aktualizacji co pochłania czas i wymaga sporej aktywności ze strony zespołu administratorów. Przedstawiony model jest modelem kosztownym, szczególnie dla organizacji niespecjalizujących się w rozwiązaniach IT i posiadających relatywnie nieduże zasoby do swojej działalności w postaci kilku, kilkunastu serwerów. Jeżeli organizacja nie może pozwolić sobie na przerwy w ciągłości działania, to pomimo małych rozmiarów infrastruktur informatycznych, wskazane elementy muszą być utrzymywane.

Alternatywą dla posiadania własnej infrastruktury jest kolokacja. W tym modelu organizacja (zwana kolokantem) instaluje własne zasoby sprzętowe w specjalistycznej serwerowni (zazwyczaj własnymi zasobami ludzkimi) pozbywając się problemu utrzymania infrastruktury pomocniczej. Niestety ten model w dalszym ciągu wymaga utrzymania zespołu specjalistów w warstwie infrastruktury sieciowej i serwerowej oraz generuje dodatkowe koszty, np. konieczność dojazdu do miejsca instalacji (serwerowni) w przypadku awarii sprzętowych. Część z operacji serwisowych

może być wykonana za pomocą usługi tzw. „zdalnych rąk" eliminując jednocześnie koszt dojazdu. Usługi „zdalnych rąk" są dostępne wraz z kolokacją i polegają na wykonywaniu operacji fizycznych przy infrastrukturze sprzętowej przez operatora serwerowni na polecenie kolokanta. Większość prostszych problemów czy też prac administracyjnych może być tak wykonana. W przypadku poważniejszych awarii lub w przypadku, gdy polityka bezpieczeństwa kolokanta nie pozwala na dostęp fizyczny do infrastruktury sprzętowej osób postronnych (w tym pracowników operatora serwerowni) to konieczna jest wizyta administratora organizacji osobiście w serwerowni.

Konkurencją dla przedstawionych modeli staje się model chmury obliczeniowej. W ostatnich latach coraz więcej organizacji decyduje się na wdrażanie rozwiązań informatycznych w tym modelu. Chmura obliczeniowa umożliwia organizacjom dzierżawę wirtualnych zasobów a także dostęp do różnego typu serwerów danych, usług IT jak i gotowych aplikacji. Rozwiązanie to jest interesujące dla wielu organizacji, gdyż w ogólnym ujęciu jest ono tańsze niż utrzymanie własnej infrastruktury informatycznej i technicznej czy też kolokacja. Jednocześnie zapewnia większą elastyczność i skalowalność oraz pozwala na skoncentrowanie się na istotnych aspektach własnej działalności często niezwiązanej bezpośrednio z IT. Z tego powodu popularność chmur obliczeniowych stale wzrasta, a zakres oferowanych usług poszerza się. Obecnie na rynku istnieje silna konkurencja, co przyczynia się do spadku cen usług chmurowych oraz poszerzenia portfolio dostępnych rozwiązań, a jednocześnie wpływa na zwiększenie jakości dostarczanych rozwiązań. W zależności od sposobu wykorzystania chmury obliczeniowej przez organizację możemy wyróżnić następujące typy rozwiązań chmurowych [37, 52, 74].

- chmury publiczne – dostępne w sieci Internet dla zainteresowanych organizacji, usługi wdrażane są w wydzielonej logicznie przestrzeni [104],

- chmury dedykowane – w których organizacja ma dostęp do wydzielonych zasobów dla swojej organizacji (droższe rozwiązanie, gdyż zasoby fizyczne nie są współdzielone pomiędzy organizacje) [52],

- chmury prywatne – organizacja wdraża rozwiązanie chmurowe na swojej infrastrukturze fizycznej [104].

Ostatnie z rozwiązań (chmura prywatna) jest bliższa tradycyjnemu modelowi, w którym utrzymywanie infrastruktury jest po stronie organizacji, w związku z czym koszty utrzymania są większe, jednak wdrożenie chmury pozwala na korzystanie z innych jej zalet, np. elastyczność rozwiązań, przez co czyni je atrakcyjnym w konkretnych warunkach.

W zależności od źródeł, z których pozyskujemy informacje, możemy spotkać różnorodne definicje chmury obliczeniowej. Niejednokrotnie podejmowano próbę sprecyzowania tego pojęcia [32, 31, 114]. O ile samo określenie chmura obliczeniowa jest trudne do zdefiniowania to zdecydowanie łatwiej określić pewne cechy, którymi chmura powinna się charakteryzować [web29]:

1. **Samoobsługa na żądanie** (ang. *on-demand self-service*) – użytkownik może zarządzać dostępnymi zasobami w sposób zautomatyzowany, bez konieczności kontaktu z dostawcą usługi.

2. **Swobodny dostęp sieciowy** (ang. *broad network access.*) – oferowane usługi są dostępne przez sieć i można ich używać bez względu na platformę z której korzysta klient (cienki/gruby klient, tablet, laptop, stacja robocza itp.).

3. **Pule zasobów** (ang. *resource pooling*) – fizyczne zasoby dostawcy usług są połączone i tworzą pule zasobów. Użytkownicy, którzy korzystają z usług, nie znają dokładnego położenia zasobów, których używają, ale są w stanie wskazać lokalizację na wyższym poziomie abstrakcji, np. wiedzą, że ich dane są przechowywane w Polsce, ale nie znają dokładnie na którym serwerze się one znajdują.

4. **Natychmiastowa elastyczność** (ang. *rapid elasticity*) – użytkownik ma pełną elastyczność zarządzania usługami z których korzysta i może łatwo i natychmiastowo modyfikować stopień wykorzystania dostępnych zasobów.

5. **Mierzalność świadczonych usług** (ang. *measured service*) – zasoby wykorzystywane przez użytkowników są automatycznie monitorowane. Zarówno użytkownicy jak i dostawca powinien mieć łatwy dostęp do informacji na temat aktualnie zużywanych przez niego zasobów.

Metody zarządzania klientami, zasobami, dostępem oraz integracja z innymi systemami i serwisami leży po stronie dostawcy chmury. Spełnienie wszystkich oczekiwań użytkowników stanowi duże wyzwanie dla dostawców oraz projektantów chmury obliczeniowej. Zapewnienie optymalnego zarządzania ma na celu pozyskanie jak największych korzyści i sprostanie konkurencji rynkowej. Zdobycie większej liczby klientów oznacza bowiem niższe koszty eksploatacji i utrzymania chmury.

2.2 Modele świadczenia usług w chmurze obliczeniowej

Przedstawiona powyżej charakterystyka dotyczy modelu ogólnego, który w zależności od potrzeb może zostać wdrożony w różnoraki sposób. Możemy wyróżnić trzy najczęściej występujące warianty świadczenia usług w modelu chmurowym [12]:

- oprogramowanie jako usługa (**SaaS** ang. *Software as a Service*),
- platforma jako usługa (**PaaS** ang. *Platform as a Service*),
- infrastruktura jako usługa (**IaaS** ang. *Infrastructure as a Service*).

Na Rysunku 2.1 przedstawiono wymienione modele oraz zaznaczono podstawową funkcjonalność, którą dostarczają.

W relacji z wyszczególnionymi modelami świadczenia usług chmurowych możemy wyróżnić trzech głównych aktorów związanych z jej funkcjonowaniem. Pierwsi to dostawcy chmury obliczeniowej (ang. *cloud providers*), którzy zarządzają centrami danych i oferują wirtualne zasoby chmury (IaaS). Drugą grupę stanowią użytkownicy chmury obliczeniowej (ang. *cloud users*), którzy dzierżawią od dostawców

Rysunek 2.1: Modele świadczenia usług chmurowych (źródło: własne)

chmury wirtualne zasoby obliczeniowe, takie jak np. procesory, pamięć RAM, zasoby dyskowe, a następnie tworzą z nich maszyny wirtualne (VMs, ang. *virtual machines*) i wdrażają na nich własne usługi, które następnie udostępniają w modelach PaaS lub SaaS. Użytkownicy chmury mogą również korzystać bezpośrednio z platform dostarczanych przez dostawców chmur obliczeniowej w modelu PaaS. Dzięki temu rozwiązaniu użytkownicy mogą wykorzystywać technologie konteneryzacji z pominięciem modelu IaaS. Tak przygotowane usługi dostarczane są użytkownikom końcowym (ang. *end users*). W dalszej części rozprawy doktorskiej użytkownicy końcowi nazywani będą klientami (klientami usług lub klientami aplikacji chmurowych). Klienci chmury obliczeniowej poprzez swoje żądania generują obciążenie chmury. Zarówno poprzez instancje usług w modelu SaaS utworzone wyłącznie na ich potrzeby jak i usługi współdzielone, z których korzystają [97]. Klienci chmury nie odgrywają istotnej roli w zarządzaniu chmurą przez dostawcę, ale ich oczekiwania i zachowania wpływają na decyzje podejmowane przy zarządzaniu chmurą. Wymagają oni bowiem m.in. atrakcyjnej oferty usług oraz wysokiej jakości dostarczanych rozwiązań. Z kolei dostawcy chmury koncentrują się przede wszystkim na zarządzaniu podstawowymi zasobami fizycznymi dostępnymi dla różnego typu użytkowników w formie

zasobów wirtualnych.

Każdy z zaprezentowanych na Rysunku 2.1 modeli oferuje inne funkcjonalności, które mogą być wykorzystywane przez użytkowników i klientów. W zależności od modelu zmienia się zakres funkcjonalny jak i złożoność usług.

Istotą modelu SaaS jest dostarczenie użytkownikowi usługi, wdrożonej w chmurze obliczeniowej. Aplikacja na bazie której świadczona jest usługa zazwyczaj udostępniana jest w postaci serwisu dostępnego za pomocą przeglądarki internetowej lub dedykowanej aplikacji instalowanej na urządzeniu klienta wykorzystującej interfejs (API, ang. *Application Programming Interface*) udostępniony przez aplikację osadzoną w chmurze. Z założenia klient nie ma informacji dotyczących sposobu wdrożenia i konfiguracji usługi, a tym bardziej systemu operacyjnego czy samej platformy sprzętowej. Ma on jedynie możliwość dostosowania usługi w ramach swoich indywidualnych ustawień w konfiguracji usługi. Usługi SaaS poprzez API mogą współdziałać z innymi usługami wykorzystywanymi w różnych środowiskach i wykonywanych na różnych platformach. Dzięki temu możliwe jest tworzenie złożonych rozwiązań aplikacyjnych budowanych z mniejszych usług.

Dzięki braku konieczności instalacji oprogramowania lokalnie przez klienta zmniejszane są koszty wdrożenia. Wielu klientów zainteresowane jest otrzymaniem gotowej usługi i jej praktycznym wykorzystaniem w związku z czym wykorzystanie usługi w tym modelu umożliwia im skoncentrowanie się na problemach organizacji, a nie problemach IT związanych z wdrażaniem czy też tworzeniem takich rozwiązań. W modelu SaaS oferowane jest zarówno oprogramowanie biznesowe, biurowe jak i użytkowe. Przykładami usług udostępnianych w tym modelu są systemy CRM (ang. *Customer Relationship Management*), ERP (ang. *Enterprise Resource Planning*), pakiety oprogramowania takie jak Google Docs czy Microsoft 365. Należy zauważyć, że aplikacji SaaS są projektowane z założeniem wykorzystania ich współbieżnie przez wielu klientów. Pojedyncze wdrożenie usługi pozwala na wykorzystanie go przez wielu klientów w przeciwieństwie do tradycyjnych aplikacji wdrażanych w organizacjach, gdzie dostęp jest ograniczony do pracowników organizacji lub w przypadku instalacji na pojedynczej stacji roboczej wyłącznie dla użytkownika tej stacji.

Model SaaS umożliwia więc nowe podejście, gdzie pojedyncze instancje oprogramowania są w stanie obsłużyć wiele grup klientów tzw. dzierżawców (ang. *tenants*). Architektura SaaS spełnia również wymagania dojrzałości uwzględniające łatwość konfiguracji, efektywność działania oraz zapewniania skalowalności. W szczególności istotne jest coraz częstsze udostępnianie usług SaaS dla urządzeń mobilnych co jeszcze bardziej poszerza grono potencjalnych odbiorców usług [79]. Z punktu widzenia modelu rozliczeń usługi SaaS udostępniane są w modelu dzierżawy poprzez sieć Internet. Wykorzystuje się rozwiązanie ASP (ang. *application service provider*), które coraz częściej przybierają nazwę modelu oprogramowania na żądanie (ang. *on-demand software*).

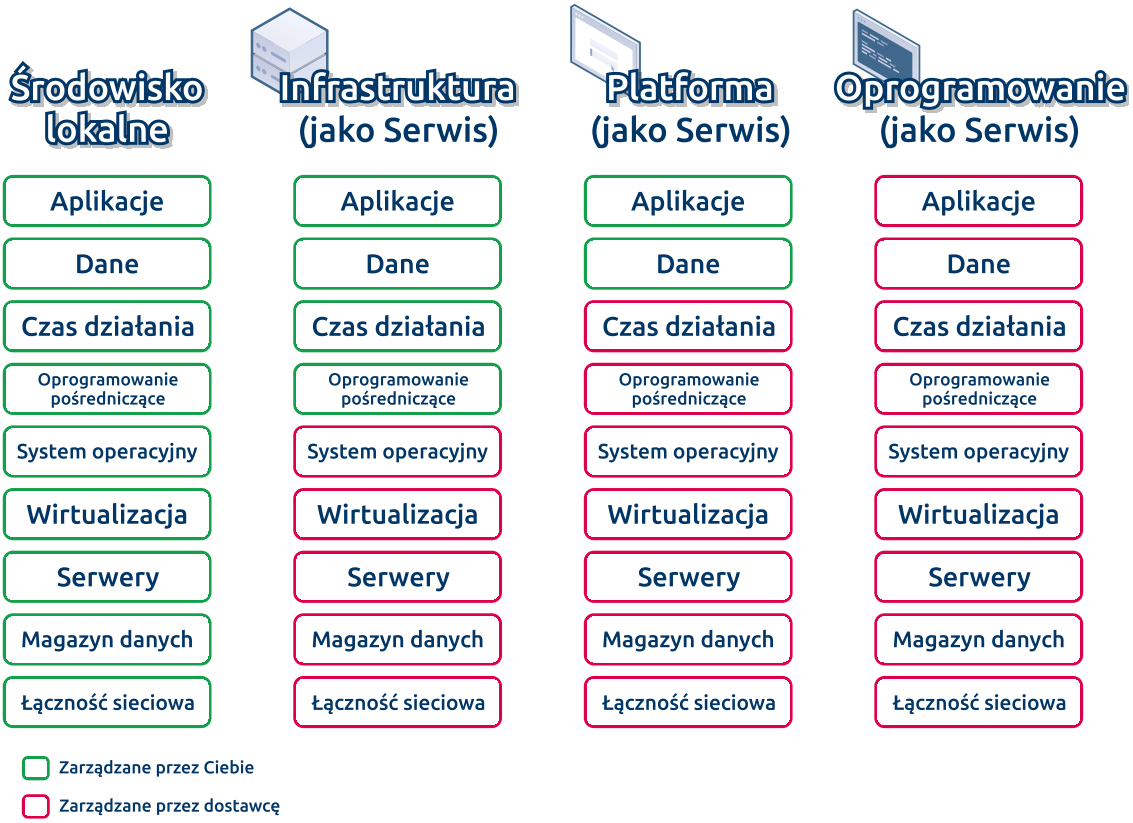
Model PaaS polega na wdrażaniu przez użytkowników chmury aplikacji wytworzonych za pomocą platformy programowej, bibliotek, narzędzi i języków programowania na dedykowanych platformach dostarczanych przez dostawców a następnie udostępnienie ich (aplikacji) użytkownikom w modelu SaaS. Koncepcja ta zakłada, że użytkownik chmury w dalszym ciągu (podobnie jak w modelu SaaS) nie ma wiedzy dotyczącej infrastruktury sprzętowej, sieciowej a także z zakresu systemów operacyjnych, gdzie wdrażana jest aplikacja. Użytkownik jest świadomy wyłącznie platformy, na którą wdrażane jest jego rozwiązanie. Warstwa PaaS wspiera cały cykl projektowania i dostarczania aplikacji oraz usług całkowicie dostępnych poprzez sieć Internet. Każda usługa charakteryzowana jest przez kod źródłowy oraz przez środowisko wykonania, w którym będzie wykonywana. Zaletą warstwy PaaS jest możliwość wykorzystania zaawansowanych mechanizmów udostępnianych w takich platformach. Przykładem może być możliwość automatyzacji procesu skalowania usługi w zależności od obciążenia (ang. *auto scaling*) czy też mechanizm automatycznego przywracania w przypadku awarii (ang. *auto healing*). Oczywiście wykorzystanie dostępnych metod nie jest w pełni automatyczne i użytkownik musi te mechanizmy odpowiednio skonfigurować, jednak jest to zdecydowanie prostsze niż opracowanie własnej implementacji wspomnianych mechanizmów. Wykorzystanie dodatkowych funkcjonalności dostępnych w modelu PaaS pozwala na utrzymywanie odpowiedniej liczby instancji aplikacji, dzięki czemu możliwa jest optymalizacja kosztów utrzy-

mania usługi oraz zwiększenie jej dostępności. Dodatkowo w modelu PaaS dostawcy udostępniają API pozwalające na automatyzację procesu wdrożenia i aktualizacji usługi. Mechanizm ten jest często wykorzystywany w modelu CI/CD (ang. *Continuous Integration and Continuous Delivery*), który umożliwia zachowanie ciągłości wprowadzania zmian w usłudze na środowiskach produkcyjnych. Usługi w modelu PaaS zazwyczaj komunikują się w oparciu o tzw. REST API, które w połączeniu z możliwościami automatyzacji i wdrażania dostępnymi w opisywanym modelu, pozwalają na łatwą kompozycję usług w nową usługę złożoną. Istnieje także możliwość ponownego wykorzystania usług (ang. *reuse*) lub jej elementów w celu wytworzenia innej usługi. PaaS dostarcza również rozbudowane usługi monitorowania zachowania aplikacji a także pomiaru jej wydajności, jak również pozwala określić liczbę równoległych dostępów do danej usługi oraz prezentuje wysokość poniesionych kosztów poprzez dedykowane systemy rozliczeń [95].

Modelem najbardziej zbliżonym do fizycznej infrastruktury jest model IaaS. W tym modelu użytkownik ma możliwość tworzenia wirtualnych środowisk sieciowych i serwerów. Dostawca usługi chmurowej ma za zadanie dostarczyć następujące zasoby: obliczeniowe, sieciowe oraz pamięć masową w celu umożliwienia użytkownikowi chmury wdrożenia dowolnego oprogramowania zaczynając od wyboru i instalacji systemu operacyjnego. W tym modelu użytkownicy nie posiadają wiedzy dotyczącej fizycznego sprzętu, mają natomiast możliwość pełnej konfiguracji swojego środowiska włączając w to instalację odpowiednich systemów operacyjnych, konfigurację sieci, pamięci masowej a w konsekwencji mają pełną kontrolę nad wdrażanym na wirtualnej infrastrukturze oprogramowaniem. Warstwa IaaS dostarcza funkcjonalności związanej z zarządzaniem wirtualną infrastrukturą m.in. tworzenie środowisk wykonawczych na maszynach wirtualnych [83]. Usługi te umożliwiają dobór odpowiednich zasobów wirtualnych dla wdrażanego oprogramowania o konkretnych wymaganiach obliczeniowych. Dobór niezbędnych zasobów wirtualnych dokonywany jest przez użytkownika z oferowanej listy zawierającej wymagane cechy oraz koszty realizacji i utrzymania zasobu. Realizacja usługi (dostarczenie zasobu) odbywa się na żądanie użytkownika. Zarządzanie dostarczonym środowiskiem wirtualnym obej-

muje zapewnienie niezawodnego dostęp do infrastruktury z sieci Internet oraz monitorowanie uruchomionej infrastruktury i podejmowanie działań naprawczych w sytuacji wystąpienia anomalii [70].

Wymienione modele wraz z podziałem na elementy, na które ma wpływ użytkownik chmury obliczeniowej w różnych modelach oraz elementy, które dostarcza dostawca prezentuje Rysunek 2.2. Kolorem zielonym oznaczono elementy, którymi zarządza użytkownik chmury (infrastruktury informatycznej w przypadku modelu *on-premise*). Kolorem niebieskim oznaczono elementy, za które odpowiada i którymi zarządza dostawca usługi chmurowej. W zależności od posiadanych kompetencji jak i potrzeb użytkownik może wybrać najbardziej odpowiedni dla siebie model, z którego chce skorzystać. Z drugiej strony w zależności od modelu świadczenia usług chmurowych dostawca chmury musi odpowiednio dopasować strategię zarządzania infrastrukturą chmury.



Rysunek 2.2: Porównanie modelu dostarczania infrastruktury informatycznej *on-premise* z popularnymi modelami świadczenia usług chmurowych (źródło: własne)

2.3 Globalne chmury obliczeniowe

W chwili obecnej na rynku obecnych jest wielu globalnych dostawców, liderów rozwiązań chmurowych. Wykorzystują oni swoje dedykowane rozwiązania, które udostępniają klientom jak i bazują na otwartym oprogramowaniu. W przypadku budowy chmury obliczeniowej w oparciu o otwarte oprogramowanie jest on dostosowywane przez modyfikację szaty graficznej lub poprzez dodanie warstwy pośredniej przygotowanej przez dostawcę, np. w postaci dedykowanego portalu.

W Tabeli 2.1 przedstawiono w kolejności malejącej charakterystykę najbardziej popularnych chmur obliczeniowych, których wykorzystanie na rynku waha się od 32% do 10%. Sumarycznie wymienieni dostawcy na pierwszy kwartał 2023 roku posiadają 65% rynku usług PaaS, IaaS zarówno publicznych jak i dedykowanych [web34]. Spośród wielu istniejących usług chmurowych dostępnych w ofercie dostawców zasygnalizowano jedynie część z nich w celu podkreślenia, że ich liczba i zakres oferowanych możliwości jest bardzo duży. Jak wynika z Tabeli 2.1 w ofercie pojawiają się również wyspecjalizowane narzędzia obejmujące zagadnienia zarządzania infrastrukturą chmurową, bazujące na sztucznej inteligencji, analityce Big Data czy też systemach IoT. Prezentowane usługi to gotowe do użycia usługi wdrożone w poszczególnych środowiskach dostawców. Niemniej opracowanie tych rozwiązań praktycznych poprzedzone było zaawansowanymi badaniami teoretycznymi. W szczególności z punktu widzenia rozprawy jest problematyka związana z zarządzaniem chmurą, w której nadal wiele problemów jest sprawą otwartą [45]. Wynika to ze specyfiki zarówno funkcjonowania chmury obliczeniowej jak i obsługi jej klientów i użytkowników.

Tabela 2.1: Charakterystyka usług dostarczanych przez trzech największych dostawców usług chmurowych [20]

Nazwa chmury	Usługi obliczeniowe	Usługi magazynu danych	Usługi sieciowe	Narzędzia wytwarzania	Narzędzia zarządzania	Analityka Big Data	Sztuczna Inteligencja	IoT
AWS (Amazon)	Amazon EC2	Amazon Elastic Block Storage	Amazon VPC	AWS Tools & SDKs	AWS Cloud Formation	AWS Data Pipeline	Amazon Kendra	AWS IoT Platform
Azure (Microsoft)	Azure Virtual Machine	Azure Page Blobs	Azure VNet	Azure SDK	Azure Resource Manager	Azure Data Factory	Azure Cognitive Search	Azure IoT Platform
Google Cloud	Compute Engine	Persistent Disk	Virtual Private Cloud	Cloud SDK	Cloud Deployment Manager	Cloud Data Fusion	Vertex Matching Engine	Google Cloud IoT

W Tabeli 2.1 zostały zebrane informacje o reprezentatywnych usługach udostępnianych przez różnych dostawców. Skupiono się na trzech kategoriach usług tj. obliczeniowych, sieciowych oraz magazynu danych oraz dodano informację o dodatkowych narzędziach wytwarzania, zarządzania zasobami oraz z zakresu sztucznej inteligencji, analizy danych i internetu wszechrzeczy. Możemy zauważyć, że w każdym z tych obszarów dostawcy posiadają dedykowane rozwiązanie, które umożliwia realizację usług przez użytkowników. Z punktu widzenia modelu IaaS najbardziej istotnym są narzędzia zarządzania oraz narzędzia ułatwiające automatyzację procesu wdrożenia i utrzymania środowiska. Użytkownicy mogą ręcznie wskazywać zasoby wykorzystując istniejące proste narzędzia lub interfejs graficzny. Jednocześnie mają możliwość skorzystania z zaawansowanych narzędzi i interfejsów wspierających automatyzację tego typu procesów, które oferują dostawcy. Przykładami takich usług są: AWS CloudFormation, Microsoft Azure Resource Manager, czy też Google Cloud Deployment Manager. Wspomniane rozwiązania zostały opisane poniżej z uwzględnieniem praktycznych aspektów przydziału zasobów. W niniejszej pracy ze względu na złożoność usług nie było możliwe ich opisanie w szczegółach . Pełne informacje dotyczące powyższych rozwiązań można znaleźć w dokumentacji dostarczanej przez dostawców.

AWS CloudFormation to usługa pozwalająca użytkownikom na zamodelowanie i uruchomienie zasobów w chmurze AWS. Dzięki niej użytkownicy mogą poświęcić mniej czasu na zarządzanie zasobami i przeznaczyć go na wytwarzanie i wdrażanie swoich aplikacji uruchamianych na przygotowanych zasobach. Usługa pozwala na utworzenie szablonów, które opisują dowolne zasoby AWS (takie jak instancje Amazon EC2 czy też Amazon RDS DB), a następnie za pomocą usługi CloudFormation zadbać o utworzenie i konfigurację opisanych zasobów. Użytkownicy nie muszę dzięki temu tworzyć i konfigurować pojedynczych zasobów oraz poznawać szczegółów zależności pomiędzy nimi. Całością zarządza usługa CloudFormation. Użytkownicy mają również możliwość utworzenia pojedynczych zasobów oddzielnie, a następnie mają możliwością przekonfigurowania ich w taki sposób by działały razem. Wszystkie te operacje tworzenia wirtualnej infrastruktury są bardzo złożone i czasochłonne. W

szczególności nowi i niedoświadczeni użytkownicy mogą stracić dużo czasu zanim uda im się wdrożyć własną usługę na satysfakcjonującym poziomie jakościowym. Ten złożony proces użytkownik może zastąpić poprzez utworzenie wspomnianego szablon CloudFormation lub zmodyfikować jeden z już istniejących. Na podstawie szablonu, który opisuje wszystkie żądane przez użytkownika zasoby i ich właściwości możliwe jest szybkie i co najważniejsze powtarzalne utworzenie żądanej infrastruktury. Szablony pozwalają na definiowanie mechanizmów auto skalowania, równoważenia i dystrybucji ruchu jak i wykorzystania gotowych usług takich jak np. bazy danych. Mechanizm szablonów udostępnia łatwy sposób zarządzania zestawem zasobów podobnie jak pojedynczym zasobem. Za pomocą pojedynczych poleceń można np. usunąć cały zestaw zasobów opisany w szablonie lub zmodyfikować szablon a następnie zaktualizować istniejącą wirtualną infrastrukturę.

Kolejnym narzędziem jest Azure Resource Manager pozwalający na wdrażanie i zarządzanie zasobami w chmurze Microsoft Azure. Narzędzie dostarcza warstwę zarządzania pozwalającą użytkownikom na tworzenie, aktualizowanie i usuwanie zasobów z wykorzystaniem dedykowanych kont użytkownika. Użytkownicy mogą wykorzystywać dodatkowe funkcje zarządzania takie jak kontrola dostępu, blokady i metadane, aby zabezpieczać i organizować wymagane zasoby przy wdrażaniu własnego oprogramowania. Skorzystanie z narzędzia możliwe jest za pomocą różnych interfejsów API platformy Azure jak i dedykowanego oprogramowania lub bibliotek SDK. Żądanie użytkownika jest odbierane, uwierzytelniane a po przejściu procesu autoryzacji przekazywane do odpowiedniej usługi platformy Azure. Dzięki zastosowaniu spójnego interfejsu API użytkownicy otrzymują w odpowiedzi na żądania zunifikowane rezultaty działania poleceń, zaś skorzystanie z szerokiej gamy narzędzi i interfejsów jest zdecydowanie łatwiejsze. Wszystkie funkcje dostępne z poziomu portalu dostępne są również za pośrednictwem programu PowerShell, interfejsu wiersza poleceń platformy Azure. Usługa Azure Resource Manager została zaprojektowana z myślą o bezpieczeństwie i wysokiej dostępności.

Google Cloud Deployment Manager to usługa wdrażania infrastruktury, która automatyzuje tworzenie i zarządzanie zasobami Google Cloud. Narzędzie umożli-

wia wykorzystanie elastycznych szablonów i plików konfiguracyjnych oraz używanie ich do tworzenia i wdrażania zasobów Google Cloud Platform, takimi jak obliczenia, bazy danych, pamięć masowa oraz sieć z wykorzystaniem różnych usług Google Cloud, takimi jak Cloud Storage, Compute Engine czy Cloud SQL. Zarządzanie tymi środowiskami chmurowymi odbywa się przy użyciu podejścia „infrastruktura jako kod" (IaaC, ang. *Infrastructure as a Code*). Korzystanie z Google Cloud Deployment Manager jest skutecznym sposobem zarządzania i automatyzacji środowiska chmurowego użytkownika. Tworząc zestaw deklaratywnych szablonów, Deployment Manager umożliwia użytkownikom spójne wdrażanie, aktualizowanie i usuwanie zasobów.

Wymienieni dostawcy oferują komercyjne zamknięte rozwiązania z szeroką gammą usług i narzędzi zarządzania infrastrukturą chmurową. Opisane narzędzia oferują bardzo podobną funkcjonalność ograniczoną do współpracy z konkretnym rozwiązaniem danego dostawcy. Rynek oprogramowania chmurowego nie jest jednak zamknięty i ograniczony tylko do rozwiązań komercyjnych. Istnieją rozwiązania darmowe, tzw. otwarte oprogramowanie wdrażane przez dostawców komercyjnych i udostępniane publicznie jak i wdrażane w postaci chmur prywatnych w organizacjach. Najbardziej znanymi tego przykładem jest oprogramowanie OpenStack [web32].

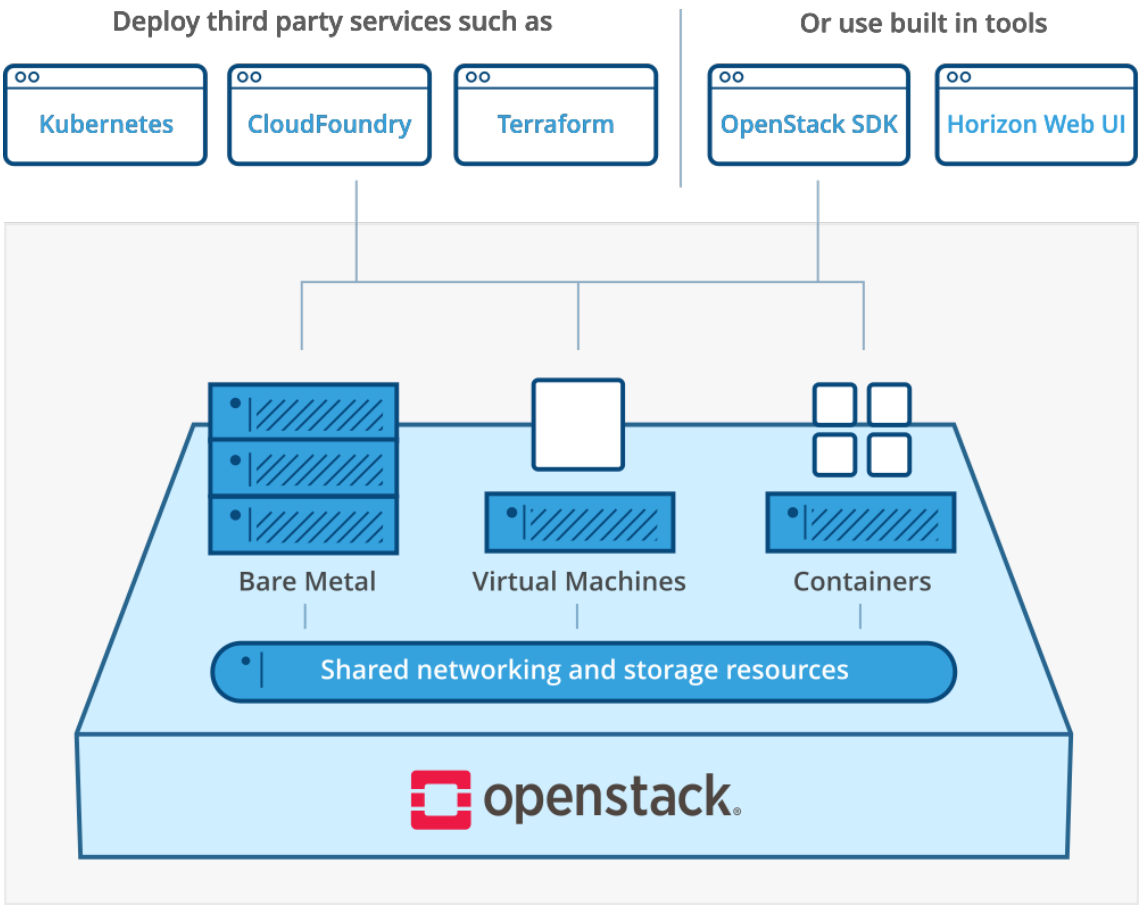
OpenStack jest otwartym oprogramowaniem rozwijanym przez społeczność światową z dużym wsparciem komercyjnych przedsiębiorstw takich jak chociażby Intel, Red Hat (obecnie IBM), Dell, Rackspace. Ten model wytwarzania zapewnia z jednej strony komercyjne finansowanie rozwoju oprogramowania a z drugiej kontrolę społeczności nad rozwojem kodu i funkcjonalności.

Podstawową funkcjonalnością, z myślą o której został stworzony OpenStack jest udostępnianie wirtualnej infrastruktury w modelu IaaS. OpenStack umożliwia tworzenie wirtualnej infrastruktury obliczeniowej (maszyny wirtualne), sieciowej (sieci wirtualne, routery, reguły dostępu sieciowego, kontrola pasma) jak i magazynu danych (magazyn obiektowy i blokowy). Wraz z kolejnymi wersjami oprogramowania oraz rozwojem technologii tj. wzrostem znaczenia technologii konteneryzacji oraz wymaganiom na przetwarzanie w czasie rzeczywistym, OpenStack został rozbudo-

wany o możliwości uruchamiania nie tylko maszyn wirtualnych, ale również zarządzania serwerami fizycznymi jak i platformami konteneryzacji. Zarządzanie platformą OpenStack możliwe jest za pomocą aplikacji dostarczonej w postaci portalu internetowego tj. komponentu OpenStack Horizon. Funkcjonalność komponentu nie pokrywa jednak w pełni wszystkich funkcjonalności dostarczanych przez OpenStack - w szczególności nie wspiera nowych lub niestandardowych komponentów. Podstawowym i zalecanym narzędziem do zarządzania infrastrukturą w chmurze OpenStack jest wykorzystanie interfejsów REST API. Jest to możliwe z wykorzystaniem bezpośrednio biblioteki OpenStack SDK lub za pomocą wykorzystania dostarczanych narzędzi CLI czy też gotowych bibliotek dla różnych języków programowania, np. języka Python. Dodatkową zaletą OpenStack jest to, że podobnie jak wspomniane chmury Google, Amazon czy Microsoft znajduje się on na liście wspieranych rozwiązań zarządzania infrastruktura chmurową takich jak Ansible [web9] czy Terraform [web36] czyli dwa bardzo popularne narzędzia do zarządzania i konfigurowania infrastruktury informatycznej (w tym wirtualnej). Ten fakt potwierdza, że oprogramowanie wytwarzane w modelu otwartego oprogramowania stawiane jest w jednym szeregu z rozwiązaniami światowych gigantów IT.

Wysokopoziomowy diagram oprogramowania OpenStack wraz z zaznaczeniem metod dostępu do usługi chmurowej prezentuje Rysunek 2.3. Na diagramie zaznaczono trzy podstawowe sposoby wykorzystania zasobów obliczeniowych tj. serwery fizyczne (ang. *bare metal*), maszyny wirtualne (ang. *virtual machines*) oraz kontenery (ang. *containers*). Wyszczególniono również warstwę sieciową i magazynu danych. Zaznaczono wspomniane metody dostępu do usług chmurowych, czyli OpenStack SDK oraz usługę webową Horizon Web UI. Dodatkowo zaprezentowano przykładowe usługi, które mogą zostać wykorzystane do wdrożenia i integracji z oprogramowaniem OpenStack tj. platformę konteneryzacji Kubernetes, narzędzie wspierające wytwarzania oprogramowania w środowiskach chmurowych CloudFoundry czy narzędzie do automatyzacji i zarządzania wirtualną infrastrukturą Terraform [web36].

W praktyce organizacje decydują się zazwyczaj na skorzystanie z usług jednego dostawcy. Dzieje się tak ponieważ w przypadku wykorzystania różnych rozwiązań



Rysunek 2.3: Ogólny schemat architektury oprogramowania OpenStack (źródło: [web32])

chmurowych, musi zostać przeprowadzony proces dostosowania narzędzi do wdrażania usług w chmurach zapewniających różne interfejsy dostępowe. Jak zaprezentowano w Tabeli 2.1 każdy z dostawców udostępnia swoje dedykowane rozwiązania do zarządzania infrastrukturą. W chwili obecnej nie ma rozwiązań, które w pełni pozwalają z centralnej konsoli zarządzać zasobami u wielu dostawców. Istnieją natomiast narzędzia, które potrafią integrować się z najpopularniejszymi rozwiązaniami chmurowymi i udostępnić dla nich spójny interfejs, za pomocą którego można budować i zarządzać swoją wirtualną infrastrukturą. Takie oprogramowanie wspomaga zarówno problem tworzenia multicloud [40] jak i adresuje niemniej ważny problem migracji infrastruktury wirtualnej pomiędzy różnymi chmurami. Narzędzia te zapewniają odpowiedni poziom abstrakcji w udostępnianym interfejsie i pozwalają, np. wykorzystać skrypty przygotowane do postawienia wirtualnej infrastruktury, na

Google Cloud do utworzenia bliźniaczej infrastruktury na Microsoft Azure. Przykładami takich narzędzi są właśnie wspomniane Terraform czy Ansible.

Reasumując wykorzystywane aplikacje w chmurze obliczeniowej generując obciążenia jej zasobów, takie jak moc obliczeniowa, sieć i pamięć masowa. Jednak dużą zaletą dostawców chmury publicznej są też usługi pomocnicze wyższego poziomu, takie jak funkcje bezserwerowe (ang. *serverless*), uczenie maszynowe i możliwości big data. Niemniej te usługi mogą wiązać się z pewnymi zależnościami, które mogą nie być oczywiste i w konsekwencji może to prowadzić do nieoczekiwanego wysokiego zużycia zasobów i zaskakujących kosztów. Wykorzystanie rozwiązań chmurowych zarówno dla prostych jak i bardziej zaawansowanych usług wiąże się z problemem zapewnienia niezawodności i bezpieczeństwa, który jest jednym z najistotniejszych problemów zarówno po stronie dostawców usług chmurowych jak i użytkowników [49]. To wszystko sprawia, że zarządzanie zasobami nie jest wydzielonym elementem, a mocno wkomponowanym w mechanizmy funkcjonowania chmury obliczeniowej.

2.4 Chmura TASKcloud

Rosnące zapotrzebowanie na usługi takie jak maszyny wirtualne wymusiły decyzję o dostarczeniu usług chmurowych również przez Centrum Informatyczne Trójmiejskiej Akademickiej Sieci Komputerowej (dalej CI TASK) [web18]. Potrzeba jej budowy wynikała zarówno z potrzeb użytkowników naukowych korzystających z zasobów CI TASK, zapotrzebowania wewnętrznego na zasoby obliczeniowe w ramach realizowanych projektów jak i zapotrzebowania ze strony przedsiębiorców współpracujących z CI TASK. Przeprowadzona analiza zakończyła się decyzją o dostarczeniu rozwiązania w modelu *Infrastructure as a Service* zakładając, że w przyszłości da ono największe możliwości dalszego rozwoju. Przy budowaniu usługi chmury obliczeniowej zespół CI TASK bazował na doświadczeniu zdobytym w ramach realizacji projektów CD NIWA [web17] oraz KRICO [web25].

Dostarczenie usług chmurowych przez organizację jest złożonym procesem składającym się z wielu etapów i wymagającym dużego wysiłku. W szczególności, gdy ce-

lem jest świadczenie usług dopasowanych do wymagań klientów oraz posiadających wysoką jakość i niezawodność. Możemy wyróżnić co najmniej trzy ścieżki, którymi można było dojść do celu, którym było świadczenie usługi IaaS przez CI TASK:

1. świadczenie usług jako odsprzedający (ang. *reseller*) usług komercyjnego dostawcy,

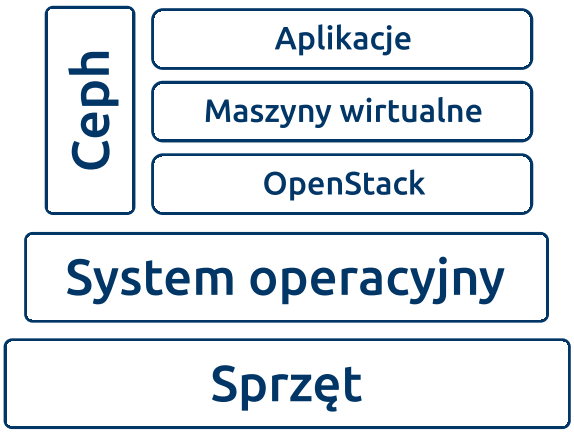
2. zakupienie na rynku w pełni przygotowanego rozwiązania wraz z instalacją i wdrożeniem na własnej infrastrukturze,

3. wdrożenie rozwiązania w oparciu o otwarte oprogramowanie z wykorzystaniem własnych zasobów osobowych.

Z uwagi na naukowy charakter jednostki została podjęta decyzja o przygotowaniu i wdrożeniu usługi samodzielnie w oparciu o dostępne otwarte oprogramowanie.

Analiza wymagań klientów, niezbędnych do udostępnienia funkcjonalności oraz zapewnienia dojrzałości wybranego rozwiązania sprawiła że zdecydowano się na wykorzystanie oprogramowania OpenStack [web32]. Oprogramowanie to miało zapewnić udostępnienie usług chmurowych w modelu IaaS. Dodatkowo niezbędne było wybranie odpowiedniego oprogramowania do zarządzania magazynem danych. W tym wypadku wybór padł również na otwarte oprogramowanie a konkretnie rozwiązanie Ceph [web20]. Oprogramowanie Ceph zostało wykorzystane jako dostawca urządzeń blokowych (wolumenów) dla maszyn wirtualnych w OpenStack. Od chwili podejmowania decyzji aż do dnia dzisiejszego są to najbardziej popularne otwarte oprogramowania dostępne na rynku [web39, web30]. Poglądowy szkic architektury rozwiązania chmury TASKcloud przedstawiono na Rysunku 2.4.

Podstawą dla przygotowywanego rozwiązania jest infrastruktura sprzętowa (warstwa *hardware*), na której posadowiono warstwę systemów operacyjnych. Część z serwerów została wydzielona na potrzeby przygotowania magazynu danych w oparciu o oprogramowanie Ceph. Druga (większa) część została przeznaczona do wdrożenia oprogramowania OpenStack. Kolejną warstwą nad oprogramowaniem chmurowym są maszyny wirtualne korzystające z magazynu danych Ceph, a ostatnią warstwą są



Rysunek 2.4: Przyjęta architektura rozwiązania TASKcloud (źródło: własne)

aplikacje (również wdrażane w postaci kontenerów), które także korzystają z magazynu danych.

W kolejnych podrozdziałach 2.4.1, 2.4.2, 2.4.3 zaprezentowano wdrożone rozwiązanie TASKcloud, którego środowisko testowe było miejscem prowadzenia eksperymentów badawczych opisanych w dalszej części rozprawy doktorskiej. Jest to chmura lokalna, która od chmur globalnych różni się przede wszystkim wielkością zasobów dostarczanych dla użytkowników. W związku z tym problemy do rozwiązania są podobne przy czym ich rozmiar jest znacznie mniejszy. Autor rozprawy doktorskiej jest jednym z głównych architektów i wykonawców rozwiązania TASKcloud oraz współautorem narzędzi służących do zarządzania tą chmurą obliczeniową. Szczegóły rozwiązania opisano w kolejnych podrozdziałach.

2.4.1 Magazyn danych Ceph

Oprogramowanie Ceph to rozwiązanie zapewniające wysoką wydajność, niezawodność i skalowalność rozproszonej pamięci masowej. W warstwie fizycznej Ceph zamiast drogich macierzy używa zwykłego sprzętu (tzw. *commodity hardware*) a konkretnie serwerów dyskowych, dzięki czemu jest zdecydowanie tańszy w zakupie i utrzymaniu. Wewnętrzny zarządca oprogramowania Ceph bazując na wprowadzonej konfiguracji tworzy tzw. mapę awarii (ang. *crush map*), która opisuje strukturę

sprzętową, na której wdrożony jest Ceph. Struktura zawiera w sobie relację pomiędzy pojedynczymi dyskami, serwerami, szafami rack a nawet strefami dostępności. Informacje zawarte w mapie awarii są wykorzystywane przez mechanizmy zapisu danych. Możemy wyróżnić dwa mechanizmy zapisu danych: replikacja oraz tzw. erasure coding. Właściwy mechanizm wybierany jest w zależności od przypadków użycia. W przypadku replikacji system utrzymuje kilka (zazwyczaj 3) kopie tego samego obiektu. Mechanizm erasure coding pozwala na zmniejszenie narzutu na przechowywanie kopii poprzez przechowywanie obiektów kontrolnych, które w przypadku uszkodzenia części obiektu pozwolą na jego odtworzenie. Przy obydwu mechanizmach proces zapisu korzysta z mapy awarii w celu określenia, na których dyskach obiekty mają zostać zapisane tak, by w przypadku awarii pojedynczego dysku lub serwera dane nie zostały utracone.

Ideą Ceph jest składowanie danych w postaci obiektów, przy czym pliki nie mapują się na obiekty w stosunku 1:1. Zazwyczaj pojedyncze pliki składają się z wielu obiektów. W chmurze TASKcloud rozwiązanie Ceph zostało wdrożona jako magazyn danych dla różnego rodzaju środowisk. Jest wykorzystywane zarówno jako magazyn dla hostów (serwerów fizycznych), maszyn wirtualnych oraz bezpośrednio przez usługi wdrożone na maszynach. W zależności od zapotrzebowania wykorzystywana jest jedna z trzech metod dostępu do danych w Ceph. Podstawowym mechanizmem jest interfejs obiektowy. Koncepcja obiektowej pamięci masowej to propozycja, która rozwiązuje problemy z tradycyjnymi systemami plików, takie jak słaba skalowalność i ograniczenia pojemności. Obiektowa pamięć masowa nie ma ograniczeń co do wielkości i można ją łatwo skalować, np. wdrażając dodatkowe serwery do Cepha. Ceph zapewnia implementację dwóch interfejsów obiektowych: S3 (standard Amazon) oraz Swift (standard OpenStack). Wymienione interfejsy udostępniane są jako usługi REST API za pomocą protokołu HTTP/HTTPS. Istnieje wiele gotowych bibliotek w różnych językach programowania jak również gotowe aplikacje wykorzystujące wspomniane interfejsy za pomocą których można gromadzić dane w magazynie obiektowym. Przykładowe scenariusze użycia obiektowej pamięci masowej mogą obejmować przechowywanie danych do analizy dużych zbiorów danych

lub archiwizowanie/tworzenie kopii zapasowych danych przy użyciu dedykowanych narzędzi wspierających magazyn obiektowy. Zazwyczaj magazyn obiektowy wykorzystuje mechanizm *erasure coding* do zabezpieczenia danych.

Drugim typem udostępnianej pamięci masowej jest pamięć blokowa. Mechanizm ten jest bardziej podobny do tradycyjnych wolumenów (dysków twardych) jak i wirtualnych urządzeń udostępnianych przez oprogramowanie, np. takiej jak iSCSI. Wolumeny a właściwie urządzenia blokowe (ang. *block devices*) mogą być montowane bezpośrednio do serwerów fizycznych lub maszyn wirtualnych zapewniając im bezpieczny magazyn danych (najczęściej wykorzystujący pod spodem mechanizm replikacji).

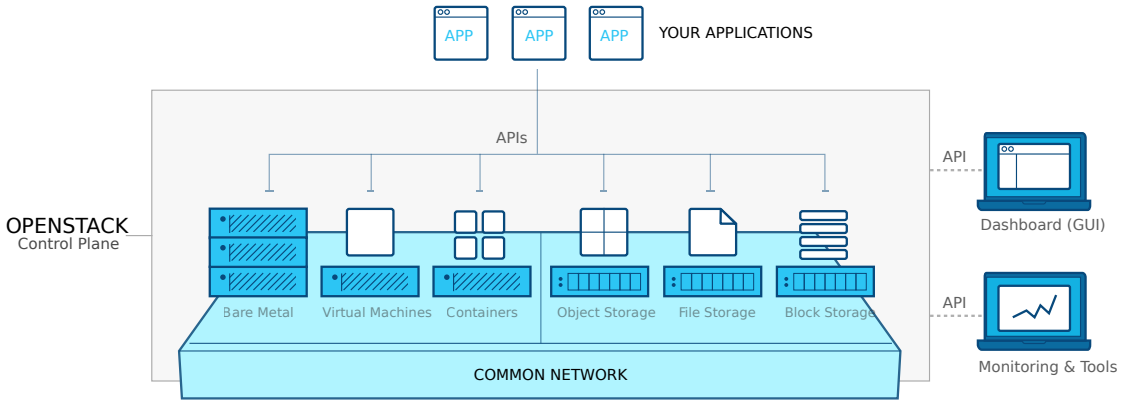
Ostatnim, ale nie mniej ważnym rodzajem pamięci masowej zapewnianej przez Ceph jest system plików CephFS. Jest to rozproszony system plików, który może być udostępniany jako współdzielony wolumen dla różnych maszyn wirtualnych lub serwerów. Pozwala współbieżnie zapisywać i odczytywać dane przez wiele procesów.

Aktualnie wykorzystywana infrastruktura magazynu danych CEPH w chmurze TASKcloud to 17 serwerów danych o sumarycznej pojemności prawie 4 PB. Magazyn danych wykorzystywany jest w TASKcloud jako źródło wolumenów dla maszyn wirtualnych. Domyślną pulą na wolumeny jest pula oparta o dyski HDD, natomiast przygotowywana dla użytkowników jest również wysokowydajna pula oparta o dyski SSD o sumarycznej pojemność 30 TB. W przypadku obydwu pól wykorzystywana jest 3-krotna replikacja ze względu na mniejszy narzut wydajnościowy. Dodatkowo TASKcloud daje użytkownikom dostęp do obiektowego magazynu danych udostępnianego za pomocą interfejsu S3 zgodnego z Amazon S3. Usługa przewidziana jest głównie pod kopie zapasowe danych użytkowników.

2.4.2 Oprogramowanie OpenStack

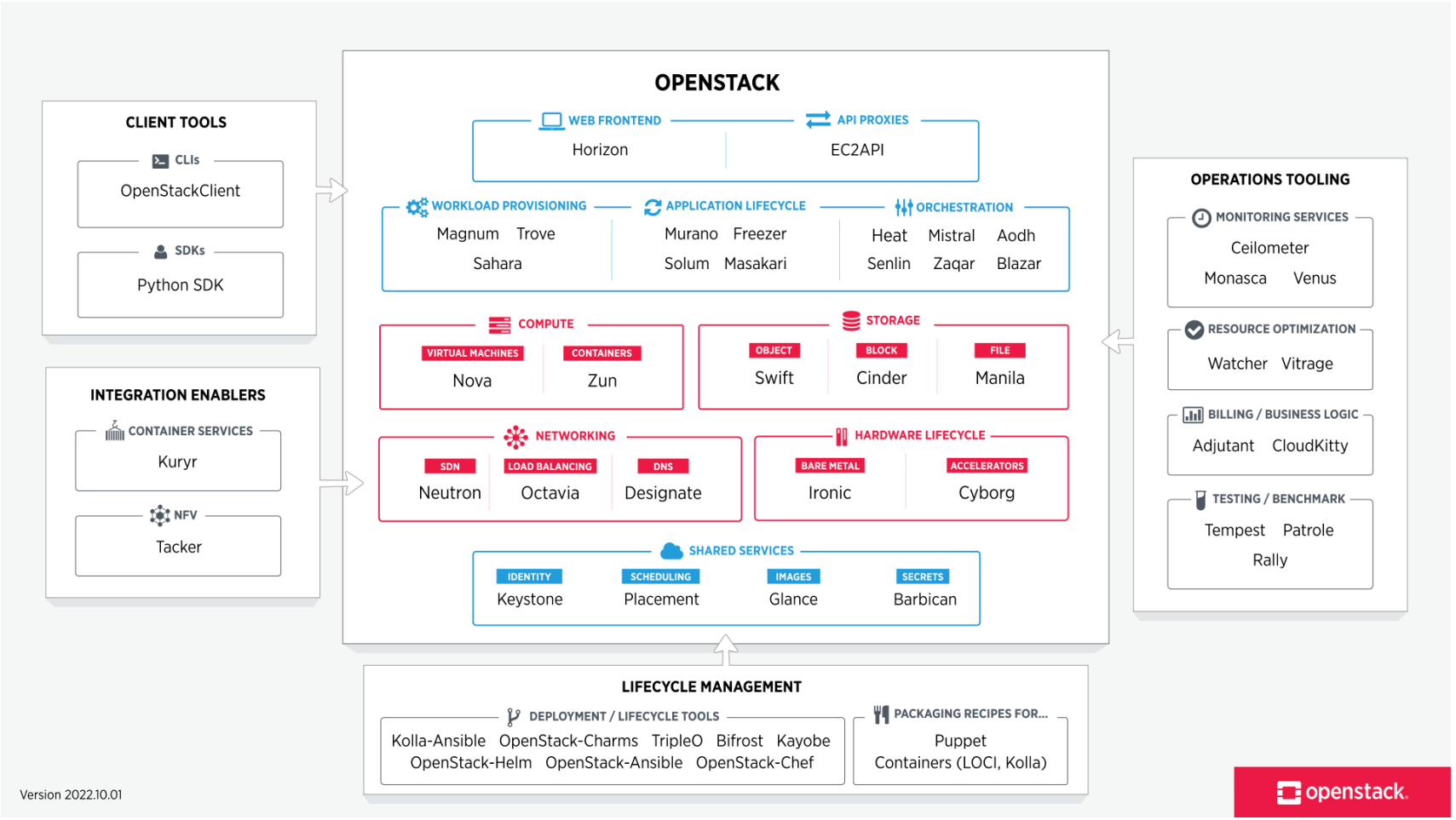
Chmura TASKcloud została zbudowana przy użyciu oprogramowania OpenStack, które implementuje, jak już wspomniano wcześniej, model dostarczania zasobów nazwany *Infrastructure as a Service*. Rozwiązanie obsługuje pełną izolację zasobów,

takich jak sieć, pamięć masowa czy zasoby obliczeniowe. Użytkownicy w chmurze mogą w pełni zarządzać w ramach limitów wszystkimi przydzielonymi zasobami. Mogą tworzyć wirtualną infrastrukturę sieciową, konfigurować maszyny z uprawnieniami administratora i dołączać do nich tzw. wolumeny czyli wirtualne dyski. Użytkownicy mogą również korzystać z obiektowej pamięci masowej i organizować obiekty za pomocą koszyków (ang. *buckets*). Ogólna koncepcja udostępnianych przez OpenStack zasobów została przedstawiona na Rysunku 2.5.



Rysunek 2.5: Zasoby udostępniane przez oprogramowanie OpenStack (źródło: [web32])

Z pełnej gamy funkcjonalności dostarczanych przez OpenStack, TASKcloud dostarcza: warstwę sieciową, maszyny wirtualne, magazyn obiektowy oraz blokowy. W kolejnych fazach rozwoju usługi planowane było dodanie możliwości udostępniania węzłów fizycznych (ang. *bare metal*), bezpośredniego uruchamiania w infrastrukturze kontenerów (ang. *cointainers*) oraz montowania współdzielonych zasobów dyskowych (ang. *file storage*)



Rysunek 2.6: Schemat architektury oprogramowania OpenStack (źródło: [web32])

Oprogramowanie OpenStack to zbiór komponentów, które wspólnie realizują koncepcję chmury obliczeniowej. W zależności od doboru wdrożonych komponentów chmura dostarcza użytkownikom różne funkcjonalności. Rysunek 2.6 przedstawia najważniejsze, tzw. oficjalne (ang. *official*) i podstawowe (ang. *core*), komponenty przyjęte przez społeczność (ang. *community*) oraz zgodne z ideą rozwoju OpenStack. Komponenty możemy podzielić na kilka grup funkcjonalnych. Do najważniejszych grup należą komponenty:

- **obliczeniowe** (ang. *compute*) - odpowiadające za uruchamianie maszyn wirtualnych i kontenerów,
- **sieciowe** (ang. *networking*) - odpowiadające za zarządzanie sieciami w tym wirtualnymi,
- **magazynu danych** (ang. *storage*) - odpowiadające za magazyn obiektowy, urządzenia blokowe i współdzielony magazyn danych,
- **zarządzanie cyklem życia infrastruktury** (ang. *hardware lifecycle*) - odpowiadające za zarządzanie cyklem życia serwerów fizycznych dla użytkowników, również z akceleratorami, np. kartami GPGPU.

Poza głównym nurtem społeczność OpenStack rozwija dodatkowe komponenty rozszerzające możliwości funkcjonalne chmury OpenStack dla użytkowników [web31], np. uruchamianie scenariuszy (Sahara, Trove), zarządzanie cyklem życia maszyn (Murano, Freezer), czy orkiestracja (Heat, Mistral). Jednocześnie prowadzone są prace nad komponentami wspierającymi wdrożenie i zarządzanie chmurą obliczeniową. Należy tu wyróżnić narzędzia instalacyjne, za pomocą których można wdrożyć OpenStack na maszynach fizycznych bezpośrednio na poziomie systemu operacyjnego (OpenStack-Ansible), w kontenerach (Kolla-Ansible) czy też wdrożyć go na Kubernetesie (OpenStack-Charms, OpenStack-Helms). Od strony zarządzania możemy wyróżnić komponenty odpowiedzialne za monitoring usług (Ceilometer), optymalizację zasobów (Watcher), czy naliczanie zużycia i kosztów dla użytkowników (CloudKitty). Społeczność dostarcza również dwa rodzaje interfejsów dostępowych zarówno dla użytkowników jak i dostawców. Są to interfejs graficzny (Horizon i

rozwijany od niedawna Skyline) oraz programistyczny w postaci CLI (ang. *command line interface*) w ramach pakietu OpenStackClient oraz jako bibliotekę Python SDK.

Wszystkie te komponenty tworzą w całości rozbudowany system świadczenia usług chmurowych zaczynając od modelu IaaS, od którego zaczynała społeczność OpenStack, poprzez PaaS i SaaS, które są coraz intensywniej rozwijane.

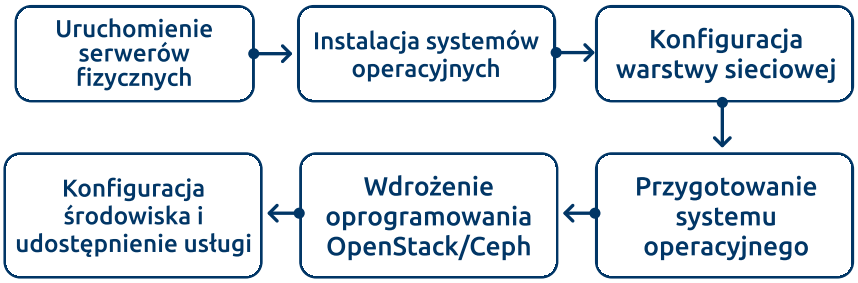
2.4.3 Wdrożenie TASKcloud w Centrum Informatycznym TASK

Przygotowanie produkcyjnej wersji chmury TASKcloud było procesem bardzo złożonym pomimo szerokiej dostępności gotowych narzędzi możliwych do wykorzystania. Wymagało to weryfikacji kilku koncepcji oraz zapoznania się z istniejącymi narzędziami. Społeczność skupiona wokół oprogramowania Ceph, które zostało wykorzystane jako magazyn danych oraz OpenStack wykorzystanego do świadczenia usług chmurowych, przygotowała dedykowane narzędzia do instalacji ww. oprogramowania. Ze względu na prostotę zarządzania i aktualizacji, jako podstawowe narzędzie do wdrożenia, zdecydowano się na wykorzystanie metody wdrożenia usług w oparciu o kontenery. Wykorzystanie kontenerów pozwoliło na zmniejszenie zależności pomiędzy warstwą systemu operacyjnego a wdrażanymi komponentami. Wszystkie zależności poszczególnych komponentów oprogramowania OpenStack są zainstalowane w odpowiednich wersjach w kontenerze. Aktualizacja pojedynczego komponentu wymaga jedynie wyłączenia kontenera ze starą wersją i uruchomienie drugiego z nową wersją. W przypadku konieczności cofnięcia wersji, np. po nieudanej aktualizacji, należy wykonać proces w odwrotnej kolejności. Cała procedura nie wymaga zatem ingerencji na poziomie systemu operacyjnego dzięki czemu minimalizowane są potencjalne problemy z konfliktami w paczkach systemowych.

Dla magazynu danych Ceph zostało wykorzystane narzędzie Ceph-Ansible [web19] a do oprogramowania OpenStack Kolla-Ansible [web24]. Obydwa narzędzia bazują na skryptach automatyzujących (ang. *playbooks*), które konfigurują i wdrażają oprogramowanie na określonych w definicjach środowisk (ang. *inventory*) serwerach fi-

zycznych. Całość infrastruktury została podzielona na części. Utworzono środowiska testowe (ang. *staging*) oraz produkcyjne (ang. *production*) dla każdej usługi. Na środowiska testowe składały się te same typy serwerów fizycznych co w środowiskach produkcyjnych. Jedyną różnicą była liczba serwerów w każdym ze środowisk. Przydzielenie każdego typu z serwerów do środowisk miała na celu weryfikację poprawności działania usługi i narzędzi służących do przygotowania tych środowisk.

Obszarem wdrażanym za pomocą gotowych narzędzi przygotowywanych przez społeczność są warstwy programowe OpenStack oraz Ceph widoczne na rys. 2.4. Wymaganiem początkowym dla każdego z narzędzi jest posiadanie uruchomionych serwerów fizycznych z zainstalowanym systemem operacyjnym oraz prawidłowo skonfigurowaną warstwą sieciową i dostępową a także przygotowanymi plikami konfiguracyjnymi dla OpenStack i Ceph. Stąd też koniecznym było zaadresowanie przygotowania tej warstwy. Aby zapewnić powtarzalność działania oraz zminimalizować ryzyko błędów, jednym z podstawowych założeń było wykorzystanie wszędzie, gdzie to możliwe mechanizmów automatyzacji. Podstawowym narzędziem do automatyzacji poszczególnych etapów zostało narzędzie Ansible. Z jednej strony cieszy się dużą popularnością i posiada rozbudowaną funkcjonalność, z drugiej jest wykorzystywane przez wybrane instalatory OpenStack i Ceph, dzięki czemu zapewnia wykorzystanie tej samej technologii na wszystkich etapach wdrożenia. Rys. 2.7 prezentuje poszczególne etapy wdrożenia, które przeobrażają w sposób zautomatyzowany pustą infrastrukturę na gotową usługę.



Rysunek 2.7: Kolejne etapy niezbędne do produkcyjnego uruchomienia oprogramowania Ceph i OpenStack (źródło: własne)

Wdrożenie TASKcloud obejmowało następujące etapy:

1. uruchomienie i konfiguracja serwerów fizycznych,
2. instalacja systemów operacyjnych,
3. konfiguracja warstwy sieciowej w środowiskach ceph i cloud,
4. konfiguracja i instalacja wymaganego oprogramowania na systemach operacyjnych,
5. wdrożenie oprogramowania Ceph i OpenStack,
6. podstawowa konfiguracja oprogramowania Ceph i OpenStack niezbędna do udostępnienia usługi.

Pierwszym z etapów była konfiguracja sprzętowa i instalacja systemów operacyjnych na serwerach fizycznych. Jako podstawowy system operacyjny zgodnie z zaleceniami Kolla Ansible oraz Ceph Ansible został wybrany CentOS. Instalacja ręczna systemów na prawie 200 serwerach nie była akceptowalna stąd podjęto próbę wykorzystania gotowego narzędzia. Niestety na rok rozpoczęcia prac nie było gotowych narzędzi, które spełniały oczekiwania zespołu, albo nie dawały możliwości instalacji systemu CentOS jak Ubuntu Maas [web26], albo były nieaktualne i nie wspierały wszystkich funkcjonalności jak Cobbler [web21]. Po analizie rozwiązań podjęto decyzję o przygotowaniu własnego narzędzia nazwanego *bare-metal-manager* (**BMM**) opracowanego m.in. przez autora rozprawy doktorskiej. Narzędzie to pozwala na instalację wybranego systemu operacyjnego, dodanie konfiguracji użytkowników i sieci po zakończeniu instalacji, a także zmianę domyślnego źródła startu systemu na dysk twardy. Podstawowym systemem obsługiwanym przez to narzędzie był CentOS 7. **BMM** zostało przygotowany w sposób pozwalający na wydzielenie środowisk i przypisanie roli serwerom oraz dedykowanych konfiguracji. W narzędziu zostało również przygotowane REST API, które za pomocą pojedynczego wołania pozwalało na przeinstalowanie serwera, którego nazwa była podawana w zadanych parametrach.

Kolejnym etapem występującym przed wykorzystaniem narzędzi wdrożenia oprogramowania Ceph i Openstack była konfiguracja systemów operacyjnych. W tym celu utworzono dedykowane narzędzia nazwane *ceph-deployment* i *openstack-manager*,

które pozwalają na dostosowanie świeżo zainstalowanego systemu operacyjnego do uruchomienia na nich właściwego narzędzia wdrażającego Ceph i OpenStack. Narzędzie wykorzystywało autorskie implementacje skryptów Ansible tzw. role, pozwalające na konfigurację następujących elementów systemu: użytkownicy i grupy systemowe [web6], sieć [web4], iptables [web3], lldp [web7], ssh [web5], system plików [web8], docker [web2]. Zastosowanie ról Ansible pozwoliło na wykorzystanie tych samych implementacji do tworzenia wszystkich środowisk Ceph i OpenStack co znacznie przyspieszyło proces przygotowania systemów operacyjnych. Ostatnim elementem koniecznym do instalacji docelowego oprogramowania było przygotowanie konfiguracji początkowych. Konfiguracje obejmują wszystkie parametry niezbędne do uruchomienia oprogramowania Ceph i OpenStack w tym m.in. definicje serwerów w środowiskach (ich adresacje i role) oraz hasła do poszczególnych komponentów, które ustawiane są przez instalatory. Całość konfiguracji tworzona przez przygotowane skrypty bazowała na wydzielonych repozytoriach. W trakcie tworzenia plików konfiguracyjnych szablony uzupełniane są automatycznie na podstawie danych z definicji środowisk. Dodatkowo, aby uniknąć przechowywania haseł w repozytoriach, by zminimalizować ryzyko wycieku krytycznych informacji, przyjęto że wszystkie hasła są przechowywane w wewnętrznym systemie Vault [web42] i wstrzykiwane do konfiguracji na etapie jej przygotowania. Na tak skonfigurowanym systemie z ustawionymi wszystkimi sieciami, użytkownikami, dostępami i gotową konfiguracją możliwe było uruchomienie dedykowanych narzędzi do wdrożenia Ceph i OpenStack.

Wykorzystanie narzędzi Ceph-Ansible i Kolla-Ansible na wcześniej przygotowanej infrastrukturze sprowadza się do uruchomienia pojedynczego skryptu. W trakcie wykonania skryptu narzędzie montuje odpowiednie wolumeny do kontenerów oraz przygotowuje usługi systemowe (ang. *services*) do automatycznego włączania usług Ceph i OpenStack. Z kolei kontenery bazują na obrazach przygotowanych przez zespół CI TASK we własnym repozytorium kontenerów. Społeczność użytkowników chmury dostarcza gotowe kontenery jednak ze względu na wprowadzane niewielkie modyfikacje koniecznym było rozszerzenie istniejących kontenerów i budowanie ich samodzielnie.

Po zakończeniu instalacji oprogramowania Ceph i OpenStack są uruchomione i należy przystąpić do wprowadzenie konfiguracji początkowej. W przypadku systemu Ceph niezbędne jest również ustawienie mapy awarii definiującej zależności pomiędzy dyskami i mającej kluczowy wpływ na sposób zapisywania replik danych na dyskach oraz definicja puli danych udostępnianych do OpenStack. Natomiast po stronie OpenStack konieczna jest definicja stref dostępności węzłów obliczeniowych (ang. *compute nodes*), agregatów hostów (ang. *host aggregates*) oraz sieci zewnętrznych (ang. *external networks*).

Tak przygotowane usługi chmurowe po uprzednich testach mogą być udostępnione użytkownikom końcowym. Etap utrzymania działających usług wymaga odpowiedniego zarządzania i aktualizacji. Aby zapewnić wiarygodność i wysoką jakość świadczonych usług wymagane jest zastosowanie odpowiedniej metody wytwarzania, utrzymywania i wdrażania zmian w usłudze.

W dziedzinie inżynierii oprogramowania zdefiniowano wiele różnorodnych metod wytwarzania oprogramowania [25]. W zależności od częstotliwości zmian w specyfikacji, dynamiki zmian technologii i standardów oraz umiejętności i możliwości zespołów wytwórczych można zastosować różne metody. Różne rozwiązania są odpowiednie dla różnego rozmiaru rozwijanego oprogramowania od tworzenia prostej aplikacji, złożonego oprogramowania, do wdrażania i zarządzania wieloma zintegrowanymi programami w jednym złożonym rozwiązaniu. Właśnie przykładem takiego złożonego rozwiązania jest wdrożenie usługi chmurowej TASKcloud.

Cieszącymi się dużą popularnością metodami są metody zwinne i DevOps [111]. Ze względu na mały (kilkuosobowy) zespół wybrany proces dodatkowo musiał zapewniać łatwość utrzymania i aktualizacji oraz przejrzystość zmian zachodzących w oprogramowaniu. Z tego powodu do przygotowania i utrzymania środowiska chmurowego wykorzystano połączenie hybrydowe tych metod. Pomimo wyboru otwartego oprogramowania, niezbędnym było wytworzenie dużej liczby komponentów wspierających proces instalacji i konfiguracji chmury opisanych powyżej. Stąd też skoncentrowano się na wykorzystaniu automatyzacji i innych praktyk DevOps z metodą iteracyjnego definiowania wymagań i wprowadzania zmian.

Proces wytwarzania i rozwoju usługi TASKcloud realizowany jest w cyklach, głównie powiązanych z wydaniem nowej wersji oprogramowania głównego tj. Ceph lub OpenStack, jednak możliwa jest realizacja cyklu zawierającego dowolny inny zakres. Możemy wyróżnić cykle dodania nowej funkcjonalności, naprawy błędów czy też aktualizacji systemu operacyjnego lub ról Ansible od których zależne jest oprogramowania główne.

Pojedynczy cykl przeprowadzany jest w następujący sposób. Określany jest zakres zmian, które mają zostać wprowadzone. Sprawdzane są zależności oraz potencjalne aktualizacje, jakie mogą zostać wdrożone w cyklu wydawniczym. Następnie definiowane są zadania implementacyjne na poziomie skryptów konfiguracyjnych lub instalacyjnych. W kolejnym kroku przeprowadzana jest implementacja zdefiniowanych zmian. Wszystkie repozytoria przechowywane są w postaci kodów źródłowych w systemie kontroli wersji Git. Pozwala to na zachowanie wszystkich wprowadzanych zmian a także zastosowanie dobrych praktyk wytwarzania oprogramowania związanych z przeglądem kodu. Po przejściu etapu przeglądu kodu, którego dokonuje członek zespołu nieprzygotowujący nowej implementacji, zmiany wprowadzane są do gałęzi głównej repozytorium i są wdrażane na środowisko testowe. Wtedy następuje etap testowania. W tym celu zostało wykorzystane i zautomatyzowane uruchomienie komponentu OpenStack Tempest [web33], który przeprowadza ponad 1500 testów weryfikując poprawność działania poszczególnych funkcjonalności systemu. Po weryfikacji poprawności działania usługi po zmianach są one wdrażane na systemie produkcyjnym z zachowaniem ciągłości działania usługi.

W chwili obecnej infrastruktura chmurowa TASKcloud jest wdrożona na ponad 200 serwerach HP ProLiant, Actina S2600 i Huwei RH1288 połączonych redundantnie siecią 10 Gigabit Ethernet. Sumarycznie zapewnia ponad 9000 wirtualnych procesorów, 24 TB pamięci RAM i prawie 4 PB efemerycznej pamięci dyskowej (SSD i HDD) za pomocą systemu Ceph. Dla każdego użytkownika dostępne są obrazy maszyn wirtualnych zawierające instalatory systemów operacyjnych Ubuntu i CentOS w różnych wersjach, które są aktualizowane (o bieżące poprawki bezpieczeństwa) automatycznie w cyklu tygodniowym.

Należy podkreślić, że konfiguracja środowiska chmurowego nawet takiego jakim jest TASKcloud jest bardzo złożonym procesem wymagającym ogromnej wiedzy oraz doświadczenia w zakresie IT. Sprawne i niezawodne funkcjonowanie tego środowiska wymaga dodatkowych analiz oraz często wykorzystania metod prób i błędów - w szczególności przy korzystaniu z otwartego oprogramowania, które niestety nierzadko mają bardzo ubogą dokumentację. Przyjęte rozwiązania przy uruchomieniu i rozwoju chmury TASKcloud sprawdziło się i jest ona wykorzystywana na bieżąco przez badaczy jak i użytkowników komercyjnych. Potwierdzeniem dojrzałości rozwiązania jest jej wykorzystanie do świadczenia usług dla Ośrodka Przetwarzania Informacji Państwowego Instytutu Badań, który realizuje na tej infrastrukturze usługi powiązane z systemem antyplagiatowym.

Chmura TASKcloud została również wykorzystana do badań nad optymalnym zarządzaniem zasobami przedstawionymi w niniejszej rozprawie doktorskiej. Istotną rolę odegrała tutaj współpraca z firmą Intel Technology Poland w ramach projektu KRICO [web25]. Poza tym sporo rozstrzygnięć projektowych wymagało również rozpatrzenia pewnych problemów teoretycznych, po rozwiązaniu których następował etap weryfikacji praktycznej. Podobne podejście zastosowano w pracach nad rozwiązaniami efektywnego zarządzania zasobami chmury obliczeniowej, które są tematyką niniejszej rozprawy doktorskiej.

Rozdział 3

Problematyka efektywnego zarządzania zasobami chmury obliczeniowej

Przeanalizowano funkcjonowanie chmury obliczeniowej z punktu widzenia klienta, użytkownika oraz dostawcy. Przedstawiono sposoby rozliczeń wykonywanych usług oraz metody szacowania niezbędnych zasobów do ich realizacji. Na tej podstawie sformułowano uniwersalny schemat zarządzania zasobami, stosowane kryteria optymalizacji oraz wyjaśniono główny cel rozprawy doktorskiej.

3.1 Uogólniony schemat funkcjonowania chmury

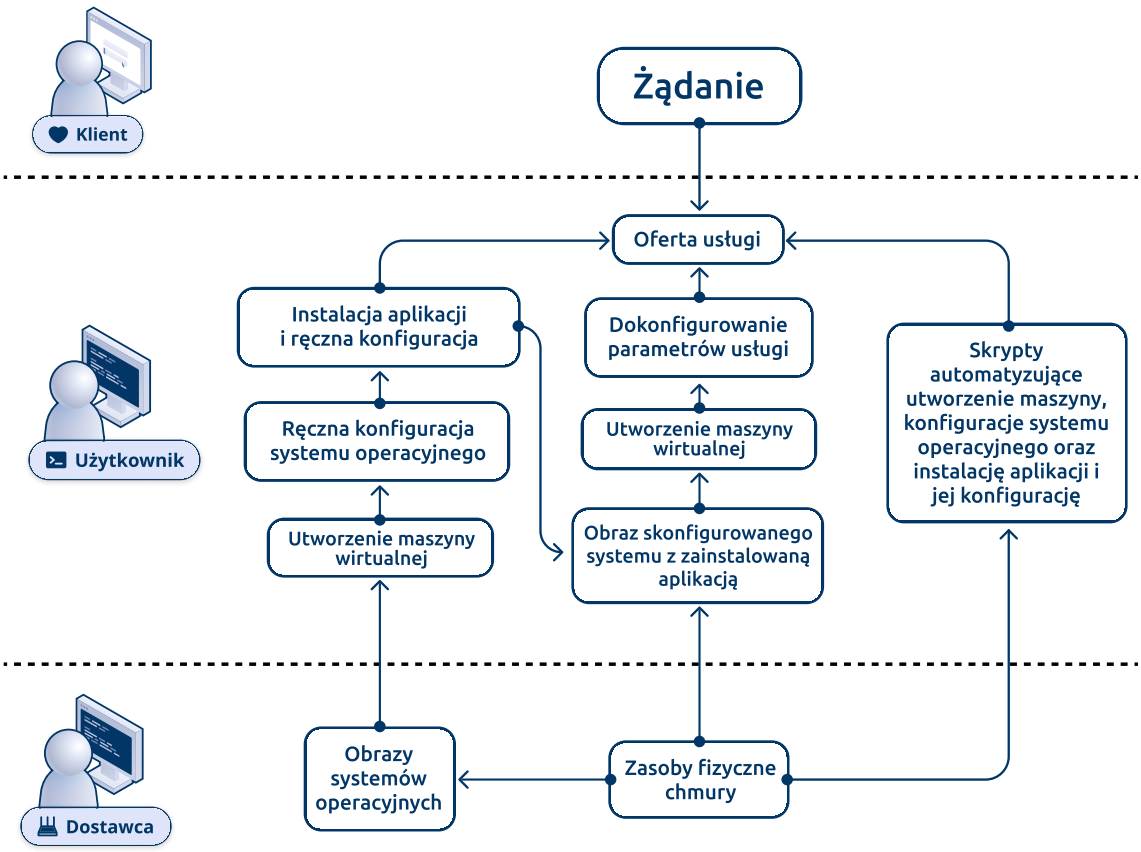
Z uwagi na specyfikę chmury obliczeniowej oraz różne wymagania użytkowników problem zarządzania zasobami nie jest sprawą prostą i nadal otwartą zarówno w modelu IaaS jak i w pozostałych modelach świadczenia usług chmurowych. Opracowane algorytmy zarządzania powinny uwzględniać wiele aspektów dotyczących wielkości oferowanych zasobów, wykorzystania aplikacji w środowisku wirtualnym (również skonteneryzowanym) jak i strategie optymalizacyjne wynikające z zawartych umów SLA (ang. *Service Level Agreement*) [80]. Zaprezentowano ogólny schemat zarządzania, koszty rozliczenia wykonywanych usług oraz kryteria optymalizacji zarządzania

zasobami chmurowymi wykorzystywane przy opracowaniu algorytmów alokacji.

Ze względu na obszar prowadzonych badań i tematykę rozprawy rozważania przedstawione w dalszej części pracy będą skupiały się głównie na modelu IaaS. Rozpatrzono problemy zarządzania z punktu widzenia klienta, użytkownika i dostawcy chmury. Usystematyzowano podstawowe pojęcia oraz zaprezentowano podstawowe przykłady postępowania prowadzące do wyznaczenia wielkości zasobów chmury wymaganych do realizacji zgłaszanych zadań klientów. Podkreślono, że istotą zarządzania jest zrozumienie zadań klientów i ich odwzorowanie na niezbędne zasoby wirtualne, które następnie są przypisywane do zasobów fizycznych oferowanych przez poszczególne węzły chmury. Zdefiniowano również główne problemy do rozwiązania w rozprawie.

Głównym celem działania chmury obliczeniowej jest wykonywanie różnego rodzaju żądań lub inaczej zgłoszeń, które oprócz odpowiedniej infrastruktury obliczeniowej wymagają efektywnych procesów zarządzania. Procedury te związane są z działalnością głównych aktorów, do których należą: klienci, użytkownicy i dostawcy chmury obliczeniowej. Klienci są najliczniejszą z grup, ich żądania sprowadzają się do dwóch głównych kategorii: dostępu do wskazanej usługi (np. skorzystania z serwera pocztowego, czy WWW) bądź wykonania konkretnej usługi (np. poprawienia jakości nagranego przez klienta filmu). Obsługą tego typu żądań zajmuje się użytkownik chmury, który wykorzystując pewne zasoby wirtualne dostarczone przez dostawcę, przygotowuje i oferuje usługi przydatne klientom i wynikające z ich potrzeb. Przygotowanie odpowiedniego środowiska przez użytkownika wymaga dużego doświadczenia i wiedzy. Pewne operacje składające się na tworzenie i zarządzanie takim środowiskiem mogą być wykonywane ręcznie, przy wykorzystaniu różnego typu narzędzi własnych (skryptów), bądź usług oferowanych przez dostawców chmury (patrz podrozdział 2.3). Przy skalowalnych rozwiązaniach produkcyjnych takie środowisko może być skonfigurowane w pełni automatyczne, tzn. w taki sposób, że kolejne żądania klientów uruchamiają odpowiednio przygotowane skrypty i procesy automatyzujące, dzięki czemu użytkownik chmury dostarczający usługę nie musi brać udziału w obsłudze żądań (patrz automatyzacja wdrożenia opisana w podroz-

dziale 2.4).



Rysunek 3.1: Ścieżki przygotowania środowiska chmurowego przez użytkownika (źródło: własne)

Istnieje wiele różnych schematów postępowania przy tworzeniu i zarządzaniu środowiskiem usług wdrożonym w modelu IaaS. Różnią się one w zależności od kategorii rozpatrywanych usług jak też wykorzystywanej architektury chmury oraz stopnia zaawansowania wprowadzonej automatyzacji. Na Rysunku 3.1 przedstawiono przykładowe trzy możliwości tworzenia środowiska chmurowego w zależności od stopnia automatyzacji procesu. Pierwsza z nich (ścieżka w pełni manualna) wymaga od użytkownika pełnej kontroli nad procesem przygotowania usługi oraz wyzwolenia konkretnych działań. Na początku użytkownik wykorzystując gotowy obraz systemu operacyjnego przygotowany przez dostawcę uruchamia maszynę wirtualną. Następnie instaluje niezbędne do działania usługi pakiety systemowe i konfiguruje je, tzn. wprowadza wymagane ustawienia poziomu bezpieczeństwa, tworzy użytkow-

ników systemowych oraz i grupy systemowe. W kolejnym kroku instaluje aplikację lub zbiór aplikacji na bazie, których planuje świadczyć usługę. Aplikacje również wymagają odpowiedniej konfiguracji (np. ustawienie dostępnego rozmiaru pamięci, ścieżek w systemie plików do składowania danych) oraz zdefiniowania pewnych zachowań systemowych dotyczących aplikacji (np. ustawienie procesów wykonujących kopie zapasowe czy usługi systemowej uruchamiającej aplikacje po restarcie systemu operacyjnego). Tak przygotowaną usługę użytkownik udostępnia w postaci oferty dla klientów, którzy mogą jej używać na określonych zasadach i zgodnie z określonym cennikiem wywołując konkretne żądania w zależności od udostępnionej funkcjonalności.

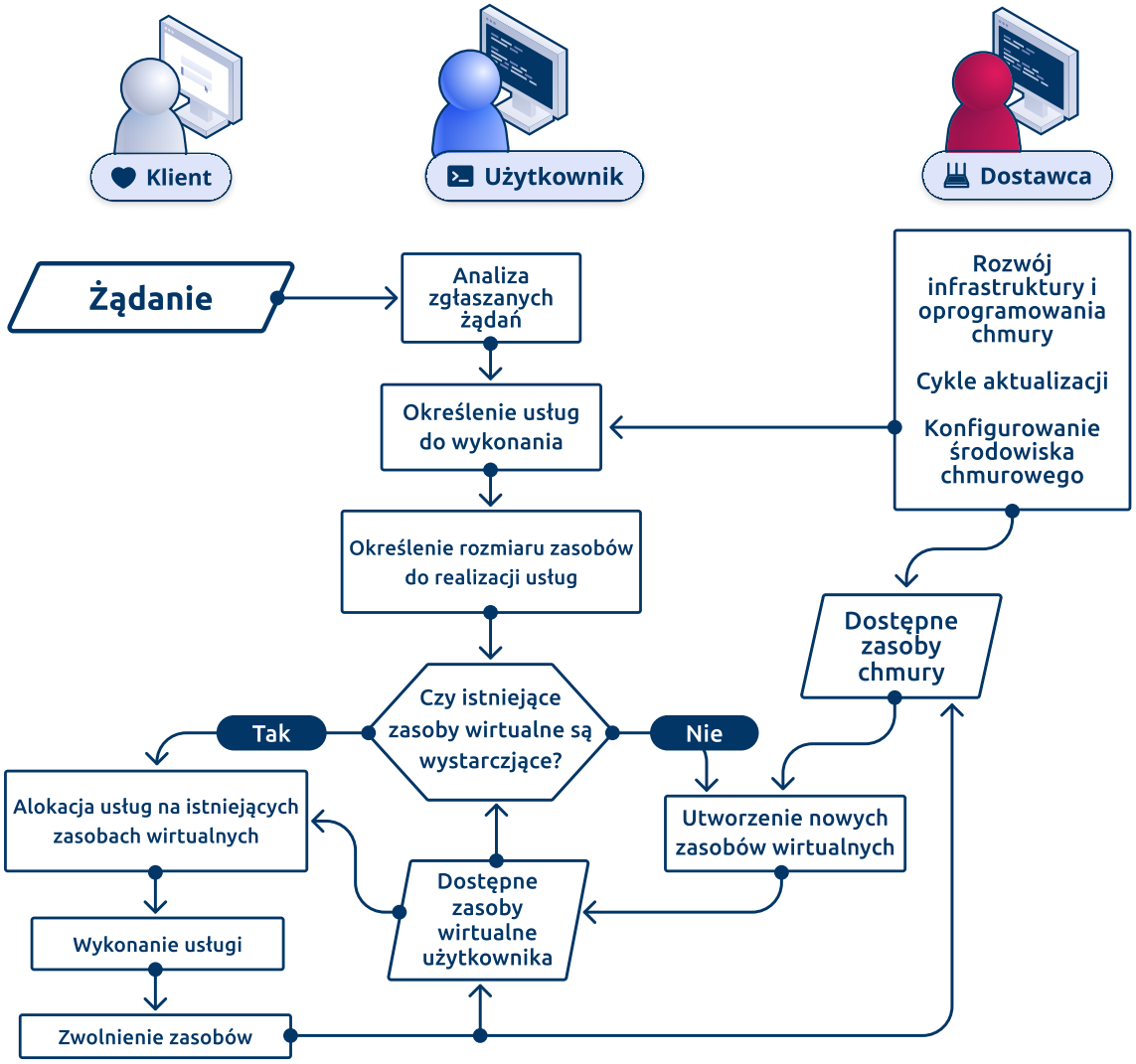
Drugą ścieżką jest wykorzystanie przez użytkownika już przygotowanej maszyny wirtualnej do świadczenie usługi dla klientów. Po konfiguracji pierwszej maszyny użytkownik może przygotować obraz maszyny, który pozwoli mu na uruchomienie kolejnych "bliźniaczych"maszyn wirtualnych. Maszyny te po uruchomieniu będą posiadały zainstalowane wcześniej pakiety i wprowadzone konfiguracje. Użytkownik wykorzystując tę ścieżkę zmuszony jest jednak do zadania ustawień w nowej maszynie wirtualnej związanych z takimi parametrami, które mogą się zmieniać przy uruchomieniu. Przykładem takich parametrów mogących mieć wpływ na działanie skonfigurowanych usług są adres IP maszyny wirtualnej czy jej nazwa. Użytkownik wykorzystując gotowy obraz musi zadbać o odpowiednie zmiany, przetestować poprawność działania usługi i dopiero może rozpocząć jej świadczenie dla klientów. Ścieżka ta jest zdecydowanie szybsza jednak nadal wymaga pewnej pracy ręcznej po stronie użytkownika co uniemożliwia natychmiastową reakcję na żądanie klienta.

Ostatnią zaprezentowaną ścieżką jest pełna automatyzacja procesu wdrożenia usługi. Wykorzystując wspomniane narzędzia takie jak Terraform czy Ansible można opisać w skryptach cały proces konfiguracji od uruchomienia maszyny wirtualnej do udostępnienia usługi klientowi. Automatyzacja obejmuje wtedy wszystkie elementy procesu tj. uruchomienie maszyny wirtualnej, konfigurację systemu operacyjnego wraz z wprowadzeniem koniecznych ustawień i zmian. Następnie dotyczy instalacji aplikacji i jej konfigurację wraz z odczytaniem ustawień takich jak przydzielone

dynamicznie (przez oprogramowanie chmurowe) adresy sieciowe czy nazwy oraz ich wprowadzenie do konfiguracji aplikacji. Dzięki pełnej automatyzacji usługa może powoływać nowe instancje (np. skalować się horyzontalnie) bez ręcznej ingerencji użytkownika.

Podstawą dla poprawnego działania usług użytkownika są działania dotyczące zarządzania chmurą obliczeniową przez dostawcę. Jego głównym zadaniem jest utrzymanie całej infrastruktury chmurowej składającej się z węzłów chmurowych (serwerów fizycznych) reprezentujących określone możliwości obliczeniowe. Możliwości te wynikają z wykorzystywanej technologii i określonej architektury węzła, w tym szybkości obliczeniowej procesora (MIPS), wielkości pamięci operacyjnej (GB), szybkości transmisji sieciowej (Gb/s), czy pojemności pamięci zewnętrznej (GB). Na infrastrukturę sprzętową nakładane są odpowiednie warstwy oprogramowania, które kreują przestrzenie wirtualne co zaprezentowano na Rysunku 2.4. Następnie dopiero utworzone przestrzenie wirtualnych zasobów służą do realizacji zadań wskazanych przez użytkownika chmury. Przyjmując powyższej opisaną strukturę chmury obliczeniowej na Rysunku 3.2 przedstawiono ogólny schemat działania chmury z uwzględnieniem roli klienta, użytkownika i dostawcy. Z niego wynika potrzeba odpowiedniego zarządzania usługami klientów oraz zasobami wirtualnymi i fizycznymi chmury.

Po zgłoszeniu żądania przez klienta następuje jego analiza w środowisku usługi dostarczanym przez użytkownika. Na podstawie specyfiki żądania wybierana jest usługa lub konkretna funkcjonalność usługi na bazie, której żądanie zostanie obsłużone. Następnie w ramach mechanizmów zawartych w usłudze określany jest rozmiar zasobów potrzebnych do realizacji żądania. W zależności od dostępności zasobów wirtualnych utworzonych na potrzeby usługi podejmowana jest decyzja o rozszerzeniu dostępnych zasobów lub wykorzystaniu już istniejących. W przypadku braku zasobów następuje utworzenie nowych zasobów wirtualnych w przestrzeni użytkownika. W kolejnym kroku następuje przekierowanie żądania klienta do nowych lub już istniejących zasobów w celu jego obsługi. Po wykonaniu usługi w przypadku, gdy zasoby są niewykorzystywane mechanizm zarządzania usługą (lub użytkownik)

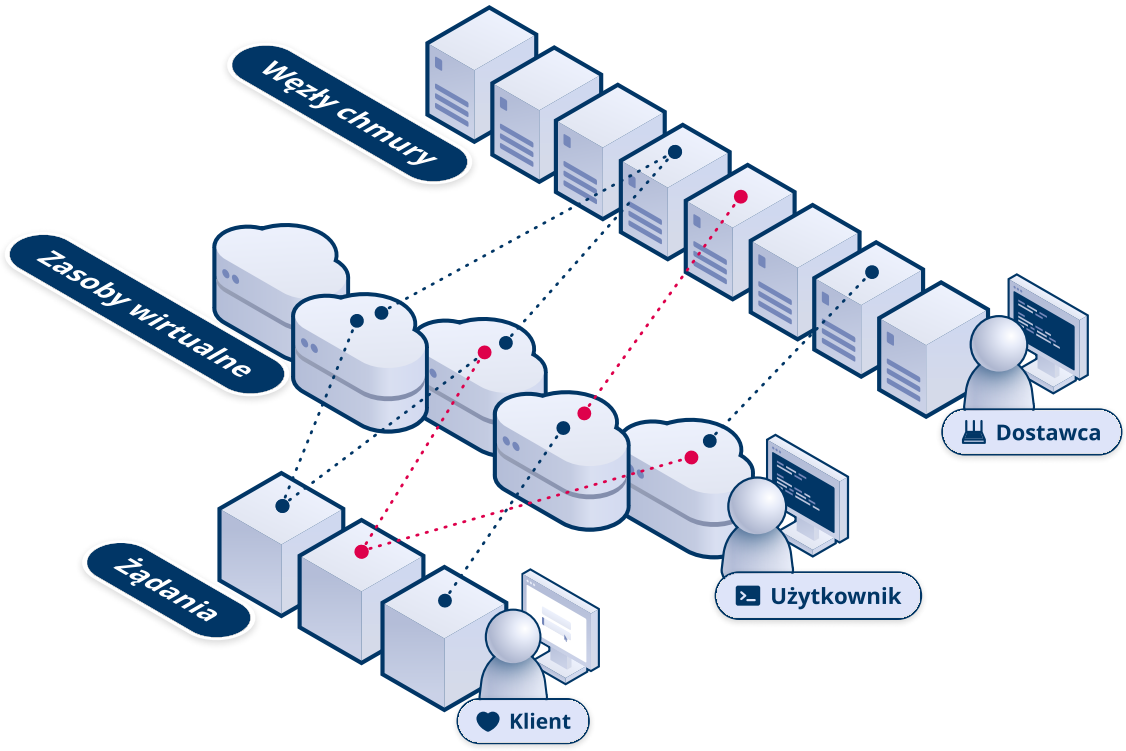


Rysunek 3.2: Ogólny schemat działania chmury z uwzględnieniem roli klienta, użytkownika i dostawcy (źródło: własne)

może podjąć decyzję o ich zwolnieniu. W tle do powyższego procesu działają mechanizmy zarządzania po stronie dostawcy. Musi on zapewnić zarówno usługi jak i zasoby niezbędne do realizacji żądań klientów przez użytkowników. Jednocześnie dostawca stale dba o aktualizację oprogramowania chmurowego w celu zapewnienia wysokiego poziomu bezpieczeństwa.

Sprawne funkcjonowanie chmury wymaga, więc odpowiednich relacji pomiędzy klientami a użytkownikami chmury oraz między użytkownikami a dostawcami chmury. Te relacje są określane poprzez zawieranie odpowiednich umów SLA. Klientów interesuje przede wszystkim dostęp bądź wykorzystanie odpowiednich usług oraz wy-

sokość opłat za te usługi. Z kolei użytkowników chmury interesuje łatwość tworzenia i wykorzystania usług chmurowych dostarczanych w środowisku utrzymywanym przez dostawcę chmury. Zyski użytkowników wynikają bezpośrednio z odpowiednich kontraktów z klientami oraz dostawcami usług chmurowych. Tego typu problemy mieszczą się również w zagadnieniach właściwego zarządzania chmurą obliczeniową. W dalszych rozważaniach proponujemy trójwarstwowy model zarządzania chmurą przedstawiony na Rysunku 3.3.



Rysunek 3.3: Trójwarstwowy model odwzorowania żądań klientów na zasoby wirtualne i fizyczne chmury obliczeniowej (źródło: własne)

Model ten podkreśla istotę dwóch odwzorowań:

- żądań użytkowników na zasoby wirtualne,
- alokacji zasobów wirtualnych na zasoby fizyczne chmury obliczeniowej w celu efektywnego wykorzystania tych zasobów.

W powyższym modelu należy uwzględnić zmienność uwarunkowań. Zarówno żądania jak również wykorzystanie zasobów zmienia się w czasie w zależności od ilo-

ści i rodzaju żądań klientów i w konsekwencji wykonywanych usług. W związku z tym dużym wyzwaniem jest zapewnienie elastyczności środowiska chmurowego minimalizując koszty eksploatacji i utrzymania chmury obliczeniowej [23]. Cecha ta powinna nie tylko dotyczyć pojedynczych klientów, ale również bądź przede wszystkim wielu różnych użytkowników realizujących jednocześnie swoje usługi chmurowe. Należy przy tym uwzględnić specyfikę architektury chmury, która obecnie wykorzystuje wirtualizację na poziomie maszyn wirtualnych bądź kontenerów [75, 54]. Dodatkowo poza typowymi rozwiązaniami chmurowymi wyróżnić można chmurę brzegową związaną z wykorzystaniem różnego typu urządzeń cyfrowych (Internetu Rzeczy - Edge computing) [16], oraz współpracę wielu chmur obliczeniowych (ang. *multicloud*) [40].

Zapewnienie elastyczności zarządzania zasobami chmur obliczeniowych polega więc na dynamicznym dostosowaniu oferowanych zasobów chmury do bieżących potrzeb użytkowników oraz zmiennych uwarunkowań otoczenia. Jest to dużym wyzwaniem z uwagi na zmienność żądań klientów i trudność przewidzenia wielkości zasobów niezbędnych do realizacji takich żądań. Na ogół użytkownicy estymują te wielkości na podstawie obserwacji żądań klientów i wykorzystywanych przez nich zasobów w określonym przedziale czasu [78]. Aspekt ten to pierwsze z wyróżnionych na Rysunku 3.3 odwzorowań. Jednocześnie konieczne jest właściwe przypisanie zasobów wirtualnych do zasobów fizycznych czyli tzw. alokacja. Zasoby wirtualne zaalokowane na tych samych zasobach fizycznych konkurują ze sobą. W związku z tym należy odpowiednio zarządzać alokacją by minimalizować ryzyka związane z brakiem dostępności do zasobów fizycznych oraz spełniać zadeklarowane wymagania jakościowe QoS (ang. *Quality of Service*). Problem ten nawiązuje do drugiego odwzorowania przedstawionego na Rysunku 3.3.

Strategia zarządzania chmurą obliczeniową powinna uwzględniać również dynamikę zmian żądań użytkowników. W takiej strategii dynamicznego zarządzania istotny jest związek pomiędzy zasobami wymaganymi do wykonania danej usługi, a zasobami możliwymi do wykorzystania w danym momencie, oferowanymi przez chmurę. Na ogół wykorzystuje się odpowiednie narzędzia, z jednej strony wspomaga-

jące predykcję zasobów niezbędnych do wykonania konkretnych usług, z drugiej zaś zdolnych do określenia listy wolnych zasobów możliwych do zaangażowania w celu wykonania tych żądań, przy uwzględnieniu konkretnej infrastruktury obliczeniowej chmury. W drugim przypadku istotną rolę odgrywają odpowiednie systemy monitorowania zasobów. W praktyce chodzi o zapewnienie jak największej elastyczności funkcjonowania chmury przy minimalnych kosztach jej eksploatacji oraz osiągnięciu jak największej wydajności. Tak zdefiniowane procesy optymalizacyjne dotyczące chmur obliczeniowych powinny być realizowane przez odpowiednie pakiety zarządzania [43]. Procesy te powinny spełniać zadane kryteria optymalizacji wykorzystania zasobów, w tym na przykład prowadzić do równoważenia obciążeń serwerów chmury. Jednocześnie procesy zarządzania chmurą powinny regularnie monitorować stan serwerów, by uniknąć kierowania ruchu do uśpionych (np. uszkodzonych) serwerów. W konsekwencji wymagane są więc dodatkowe funkcje zarządzające, np. oprócz monitorowania stanu węzła, również migracji zasobów wirtualnych na inne węzły, co ostatecznie komplikuje implementację mechanizmów zarządzania chmurą.

3.2 Metody rozliczania usług

Istotnym elementem zarządzania usługami jest metoda rozliczania kosztów związanych z ich wykonaniem. W zależności od zadeklarowanego sposobu rozliczeń wykorzystanych zasobów możemy wyróżnić dwa modele rozliczeń [29, 23] za zużycie zasobów oraz rezerwację zasobów. Model płatności za zużycie polega na monitorowaniu przez dostawcę wielkości zasobów wykorzystywanych przez użytkowników a następnie wyliczeniu kosztu za dany okres rozliczeniowy. Drugi model, model rezerwacji, opiera się na zadeklarowaniu przez użytkownika wielkości potrzebnych zasobów, które będzie potrzebował i rozliczeniu opłaty za te zasoby bez względu na to czy użytkownik faktycznie je wykorzystywał czy też nie. Zasadniczą różnicą pomiędzy tymi metodami jest cena. Pierwszy z modeli jest droższy ze względu na mniejszą przewidywalność przychodów przez dostawców chmurowych oraz większą dynamikę zmian wykorzystywanych zasobów. Dodatkowym aspektem z punktu widzenia do-

stawcy jest zachowanie większej puli wolnych zasobów w chmurze tak by zapewnić skalowalność dostarczanej usługi.

Dostawca chmury monitoruje wielkości faktycznie zarezerwowanych zasobów fizycznych oraz usług dodatkowych na poziomie IaaS przez użytkowników chmury. Ci zaś czerpią profity od klientów na podstawie wykorzystania ich usług na poziomie SaaS. Sposób realizowania rozliczeń i ich cena opisywana jest w umowach i dotyczyć może miesięcznej opłaty za wykorzystanie odpowiednich zasobów wirtualnych, uwzględniając również typ wykorzystywanych usługi, czy przyjęty poziomu dostępności i jakości usług. Zawarta umowa może też uwzględniać pewne odszkodowania związane z naruszeniem uzgodnionych wymagań (np. dłuższego czasu niedostępności) zawartych w porozumieniu SLA.

Bez względu na kategorię rozliczeń użytkownikowi zależy na otrzymaniu usług o jak najwyższej jakości, natomiast dostawca usług chce zmaksymalizować swoje dochody. Aby zaspokoić oczekiwania i potrzeby obydwu stron niezbędne jest znalezienie optymalnego modelu naliczania kosztów za świadczone usługi. W zależności od modelu, w którym udostępniane są obliczenia w chmurze i zmiennych branych pod uwagę przy takiej analizie, możemy wyróżnić różne modele naliczania kosztów [96, 5, 118, 19, 119, 69]. W przypadku modelu SaaS dość częstym rozwiązaniem, jest wprowadzenie opłat miesięcznych za dostęp dla użytkownika końcowego (licencje per user). Model PaaS oferuje naliczanie za czas uruchomienia kontenera i wykorzystywanych prze niego zasobów. W przypadku naliczania kosztów za wykorzystanie zasobów w modelu IaaS najczęściej stosuje się wspomnianą metodę *pay-as-you-go* lub inaczej *pay-per-use* [12, 47].

W dynamicznym i szybko zmieniającym się obszarze technologii chmurowych niezbędne jest dostarczenie właściwej usługi naliczania kosztów oraz dobrze zdefiniowanego cyklu naliczania. Aspekt ten jest niezwykle istotny zarówno dla dostawców jak i użytkowników chmurowy obliczeniowej. Dostawcy muszą dopasować swoją ofertę biorąc pod uwagę wiele aspektów takich jak infrastruktura, którą dysponują, rynek docelowy, koszty własne, politykę zniżek oraz sposób naliczania za wykorzystane zasoby obliczeniowe. Z drugiej strony użytkownicy mierzą się z rosnącą liczbą

dostępnych ofert i muszą analizować, które z nich zapewnią im oczekiwane rezultaty przy jak najniższym koszcie tak by zaplanować odpowiednio własny budżet.

W literaturze możemy znaleźć wiele przykładów systemów, które pozwalają monitorować i wyliczać koszty za wykorzystywane zasoby wspierające użytkowników chmury obliczeniowej. Przykładami mogą być rozproszony system monitorowania wykorzystania zasobów w modelu przedpłaconym [122], czy systemy oparte o wbudowane rozwiązania Internetu Rzeczy [127]. Na szczególną uwagę zasługuje centryczny model dla użytkownika usług chmurowych zbierający dane poprzez usługi sieciowe udostępniane przez dostawców [76], który pozwala na porównanie potencjalnych kosztów infrastruktury w różnych chmurach obliczeniowych. Kolejnym przykładem jest system zapewniający weryfikowalność kosztów ponoszonych przez klienta [98]. Możemy również wskazać kompleksowe modele pozwalające na symulację modelu finansowego i wyliczenie kosztów wdrożenia rozwiązań chmurowych biorąc pod uwagę różne aspekty [100].

Możemy również wyróżnić wiele rozwiązań wspierających dostawców usług chmurowych w monitorowaniu zużycia i naliczaniu kosztów na użytkowników. W artykule [62] autorzy definiują wymagania dla systemu monitorującego wykorzystanie zasobów rozproszonych geograficznie. W opisie zawarto wymagania jakie powinien spełniać taki system a następnie zaproponowano odpowiedni system monitorujący wraz z przeprowadzeniem jego weryfikacji. Artykuł [29] przedstawia architekturę systemu do monitorowania i naliczania kosztów dla modelu federacyjnego usług chmurowych. Kompleksowe rozwiązanie dla dostawców proponują autorzy w pracy [35]. Zaproponowano w niej model specyfikowania informacji o kosztach i ich naliczania, metodę automatycznego generowania rachunków w oparciu o zestawy reguł zdefiniowanych w modelu oraz przedstawiają analizę automatycznych systemów rozliczeń oraz narzędzie prototypowe implementujące zaproponowane rozwiązanie.

Niezależnie od wybranego modelu rozliczeń oraz przyjętych wymagań jakościowych niezbędne jest odpowiednie szacowanie wielkości zasobów wykorzystywanych w chmurze obliczeniowej. Jest to istotny aspekt zarówno po stronie użytkownika jak i dostawcy, gdyż w obydwu przypadkach przekłada się na koszty. W związku z tym

konieczne jest wykorzystanie odpowiednich procesów zarządzania zasobami by te wielkości oszacować. W rozprawie doktorskiej ograniczamy się do odpowiedniego alokowania zasobów z wykorzystaniem klasyfikacji funkcjonalnej wykonywanych usług. Jest to oryginalne podejście, gdyż na ogół badacze koncentrują się jedynie na konkretnie wykonywanych usługach.

3.3 Szacowanie wymaganych zasobów chmurowych usług

W systemach chmurowych zarządzanie chmurą może być rozpatrywane na kilku poziomach. Pierwszy z nich to zarządzanie użytkownikami. Systemy chmurowe wykorzystują wbudowane mechanizmy lub są podłączone do zewnętrznych systemów uwierzytelniających i autoryzujących użytkowników. Przykładem może być wbudowany komponent Keystone w systemie OpenStack lub zewnętrzny dostawca tożsamości (IdP ang. *Identity Provider*), np. oprogramowanie Keycloak wykorzystywane w chmurze TASKcloud. Ten poziom zarządzania polega na udzielaniu dostępów do usług chmury obliczeniowej.

Kolejnym poziomem zarządzania jest zarządzanie przydziałem zasobów. Dla każdego użytkownika jak i grupy użytkowników niezbędna jest możliwość przydziału zasobów często udostępniana w postaci projektów lub przestrzeni, w których użytkownik tworzy swoje wirtualne zasoby (patrz podrozdział 3.1). Taka forma grupowania zasobów umożliwia w łatwy sposób przydzielanie limitów na zasoby dla każdego projektu oddzielnie oraz daje dużą elastyczność w przydzielaniu dostępów dla użytkowników. Zazwyczaj funkcjonalność ta realizowana jest na poziomie wbudowanych komponentów chmury obliczeniowej, np. dla chmury TASKcloud opartej o OpenStack jest to wspomniany komponent Keystone (patrz podrozdział 2.4).

Najniższym poziomem jest zarządzanie zasobami i ich wykorzystaniem przez dostawców chmury obliczeniowej. Jest to najbardziej złożony z wyszczególnionych powyżej poziomów zarządzania skupiający się na odpowiednim przydzieleniu zasobów i ich rozłożeniu pomiędzy węzłami chmury. Autorzy rozwiązań oprogramowania

chmurowego w trakcie opracowywania systemów zarządzania chmurą obliczeniową bardzo mocno skupiają się na rozwiązaniach optymalnego wykorzystania zasobów sprzętowych udostępnianych w chmurze. Szczególną wagę do sposobu realizacji tego aspektu przykładają dostawcy usług chmurowych. Niedokładne oszacowanie rzeczywistych zasobów wykorzystywanych przez użytkowników prowadzą albo do rezerwacji zbyt dużego wolumenu zasobów, które ostatecznie są niewykorzystywane (ang. *overcommitment*), bądź do niedoboru zasobów, co wymusza przeniesienie obliczeń na inny węzeł, na którym są wolne zasoby. W konsekwencji opisane zjawisko prowadzi do rozrostu infrastruktury po stronie dostawców a tym samym zwiększenia kosztów zakupu, utrzymania i serwisowania. Są więc pożądane rozwiązania, które minimalizują zjawisko przeszacowywania rezerwowanych zasobów, gdyż pozwalają zwiększyć liczbę maszyn wirtualnych, które mogą być uruchomione na określonej liczbie serwerów fizycznych, co przekłada się bezpośrednio na większy zysk dostawcy. Efektywniejsze wykorzystanie zasobów sprzętowych ma również bezpośredni wpływ na redukcję zużycia energii elektrycznej i ilość wytwarzanej energii cieplnej. Aspekt ten jest bardzo istotny z punktu widzenia infrastruktur informatycznych o dużej mocy obliczeniowej. Tym samym odpowiednie zarządzanie zasobami przyczynia się do ochrony środowiska naturalnego zgodnie z koncepcją zielonych obliczeń (ang. *green computing*). Najczęściej wykorzystywanymi metodami doboru węzła chmurowego dla danych zasobów wirtualnych są algorytmy bazujące na danych informujących o bieżącej zajętości węzłów chmurowych. Możemy wyróżnić trzy podstawowe metody alokacji usług na zasoby:

- rozlokowanie zasobów wirtualnych na losowym węźle,
- dążenie do rozlokowania zasobów wirtualnych tak aby wszystkie węzły były obciążone możliwie równomiernie (ang. *load balancing*). Z reguły oznacza to, że nowe zasoby wirtualne są umieszczane na najmniej obciążonym węźle chmury obliczeniowej,
- rozlokowanie zasobów wirtualnych maksymalizujące wykorzystanie węzłów (ang. *consolidation*). Najczęściej nowe zasoby są umieszczane na najbardziej obcią-

żonym węźle, który spełnia jeszcze wymagania wydajnościowe, aby te zasoby tam uruchomić.

Tematyka alokowania zasobów oraz modelowania obciążenia komponentów sprzętowych jest często poruszana w literaturze. W wielu publikacjach przedstawione są zagadnienia związane z szacowaniem stopnia wykorzystania zasobów, np. w kontekście szeregowania zadań [64, 17], w tym w systemach czasu rzeczywistego [68, 88]. Naukowcy zajmują się również problemami zależności pomiędzy komponentami sprzętowymi w istniejących architekturach. Przykładem mogą być ograniczenia wykorzystania procesora w przypadku oczekiwania na dostęp do pamięci operacyjnej. W ramach eksperymentów opracowywane są modele, które pozwalają na estymację maksymalnej wydajności poszczególnych komponentów, w oparciu o symulacje [103], analizę ograniczeń wydajnościowych [116] czy algorytmy sztucznej inteligencji [112]. Jednocześnie oprócz optymalizacji wydajności sprzętowej opracowywane są modele optymalizujące inne istotne aspektu z punktu widzenia środowisk obliczeniowych w tym m.in. środowisk rozproszonych. Często tematyką poddawaną w rozważaniach związaną z zarządzaniem obciążeniem zasobów obliczeniowych jest modelowanie zużycia energii elektrycznej [117], określanie granic wydajności przyszłych generacji sprzętu [30] a także projektowanie bardziej wydajnych algorytmów równoległych [113]. W większości prace te skupiają się jednak na przetwarzaniu w systemach HPC (ang. *high performance computing*) i nie zawsze uwzględniają specyfikę chmury obliczeniowej.

Z punktu widzenia chmury obliczeniowej najistotniejszym elementem są obciążenia generowane przez zasoby wirtualne alokowane na potrzeby użytkownika. W modelu IaaS zasoby te reprezentowane są w postaci maszyn wirtualnych, które w różny sposób obciążają zasoby fizyczne. Obciążenie wynika z konkretnej aplikacji lub zbioru aplikacji uruchomionych na potrzeby realizacji usługi świadczonej przez użytkownika klientom. Sama definicja obciążenia przekłada się na zużycie lub wykorzystanie zasobów. W literaturze anglojęzycznej związanej z tematyką chmur obliczeniowych występuje definicja workloadu, która tłumaczone wprost na język polski mogłaby być mylnie przetłumaczona również jako obciążenie. Pojęcie workload nie

jest jednak tożsame z obciążeniem zdefiniowanym powyżej tzn. wykorzystaniem lub utylizacją fizycznych zasobów obliczeniowych węzła. Workload jest pojęciem szerszym mówiącym o pracy, która musi zostać wykonana na potrzeby przetwarzania wskazanego zadania. Może to być usługa, aplikacja czy też praca, która wykorzystuje pulę zasobów chmurowych wraz z danymi wejściowymi i wyjściowymi [web43]. Możemy powiedzieć, że zarówno baza danych, kontener, mikrousługa czy maszyna wirtualna - wszystkie te byty są workloadem i generują faktyczne obciążenie na węzłach chmurowych. W związku z brakiem rzetelnego tłumaczenia w literaturze dotyczącej zagadnień chmurowych w dalszej części rozprawy zostanie użyte angielskie określenie workload. Pojęcie będzie używane w kontekście opisywanego modelu IaaS, tzn. będzie reprezentować maszynę wirtualną wraz z osadzoną na niej usługą przetwarzającą zadany zestaw danych wejściowych i generującą fizyczne obciążenie na węźle chmury obliczeniowej. Analizie worklodów poświęcono następny rozdział rozprawy doktorskiej.

W literaturze opisywane są zróżnicowane techniki modelowania infrastruktury chmurowej jak i obciążenia generowanego przez workloady w środowisku chmury obliczeniowej. W zależności od przyjętych ograniczeń i zastosowań proponowane są różne podejścia do zagadnienia efektywnego przetwarzania w chmurze i modelowania zasobów chmurowych. Do proponowanych metod wykorzystywane są różne metody modelowania często łączone ze sobą w tzw. modele hybrydowe. Należy w tym miejscu wspomnieć o kilku ciekawych podejściach do modelowania infrastruktury chmury obliczeniowej. W pracy [4] autorzy proponuję budowę modelu chmurowego w oparciu o model kolejkowy G/G/N dla dynamicznego przydzielania zasobów (maszyn wirtualnych). Innym przykładem są badania zaprezentowane w artykule [55], gdzie przedstawiono szeregowanie zadań w środowisku wieloprocesorowym w oparciu o model oparty o łańcuchy Markowa. W artykule [43] proponowane jest tworzenie empirycznych modeli usług pozwalających na przewidywanie zmian zapotrzebowania na zasoby na podstawie monitorowania aktualnego obciążenia generowanego przez workloady. Kolejny model wspomagający alokację workloadów dla aplikacji warstwowych w chmurze obliczeniowej przedstawiony w pracy [73] wykorzystuje meta-model,

połączenie modelu empirycznego i modeli teoretycznych. Możemy również znaleźć przykłady algorytmów szacujących wykorzystanie zasobów w chmurze w oparciu o: (a) transformatę Fouriera i model skończonego łańcucha Markowa [36], (b) metody statystyczne [33], (c) sztuczne sieci neuronowe [53] i (d) techniki wzmocnionego uczenia (ang. *reinforcement learning*) połączone z modelem kolejkowym [110].

Jednocześnie w literaturze możemy odnaleźć opisy prac związanych z predykcją zapotrzebowania na zasoby sprzętowe w zależności o rodzaju workloadu. Przykładem takiej analizy są artykuły [109, 42], w których autorzy rozpatrują zapotrzebowanie na zasoby dla usług udostępniających i przetwarzających dane multimedialne. Podobne badania dla serwerów wymiany komunikatów przedstawiają autorzy w pracy [102]. W przypadku chmury obliczeniowej ogólnego przeznaczenia tj. niespecjalizowanej dla konkretnego typu workloadów pomocnym może okazać się zdefiniowanie podziału workloadów na kategorie o podobnych charakterystykach generowanego obciążenia. W oparciu o taki podział można przeprowadzić badania związane z optymalizacją efektywności przetwarzania i alokacji workloadów w chmurze w zależności od kategorii workloadu. Zagadnienie podziału obciążeń chmury obliczeniowej w zależności od kategorii aplikacji było już przedmiotem rozważań w literaturze [61, 66, 93]. W większości przypadków autorzy skupiali się na właściwej alokacji workloadów lub zmianach w wykorzystaniu zasobów w trakcie tzw. cyklu życia obciążenia w zadanym okresie czasu, a nie dokonania alokacji na początku życia workloadu.

W artykule [108] autorzy proponują algorytm nazwany *co-plots*. Działanie algorytmu polega na redukcji wymiarów z przestrzeni wielowymiarowej do dwuwymiarowej na przykładzie danych z logów przechowujących informacje z wykonania workloadów na superkomputerach. Autorzy tej pracy wykazują możliwość redukcji rejestrowanych danych do dwóch wymiarów a następnie modelują obciążenia generowane przez workloady. Na podstawie przeprowadzonych eksperymentów pokazują, że nie jest możliwe przygotowanie jednego modelu dla różnych typów workloadów. W związku z tym proponują parametryczny model obciążenia odnoszący się do zidentyfikowanych wcześniej dwóch cech. Wnioski płynące z przeprowadzonej analizy

mówią, że istniejące modele nie są w stanie dobrze odwzorować zmieniających się w czasie workloadów oraz że stosowana metodologia zarządzania workloadami jest nieefektywna i problematyczna.

W artykule [77] zaproponowano metodę klasyfikacji workloadów i określania ich charakterystyki na podstawie danych udostępnionych z chmury Google Cloud. Analizę przeprowadzono na podstawie następujących parametrów: wykorzystanie procesora, pamięci oraz czas trwania workloadu. Następnie w oparciu o te parametry wykorzystano algorytm k-średnich do wyodrębnienia 8 reprezentatywnych klas. Oprócz przedstawionej metodologii autorzy opisali rezultaty analizy charakterystyk workloadów, gdzie zauważają, że większość workloadów trwa bardzo krótko i nie zużywają one dużej ilości zasobów. Natomiast większość zasobów jest alokowana przez małą liczbę zadań trwających bardzo długo.

W [89] autor poddaje analizie powszechnie stosowany do separacji klastrów algorytm k-średnich. Wykazuje, że nie jest on idealnym rozwiązaniem i podaje przykład, w którym w systemie kolejkowym algorytm nie sprawdza się. We wnioskach autor wykazuje, że zastosowanie algorytmu k-średnich jest możliwe do dokonanie automatycznej kategoryzacji workloadów na podstawie charakterystyk obciążenia jednak wymagane jest dokładne przeanalizowanie rezultatów działania algorytmu.

Praca [125] przedstawia klasyfikator oparty na algorytmie PCA (ang. *Principal Component Analysis*) oraz algorytm k-najbliższych sąsiadów, który posłużył do wsparcia procesu alokacji workloadów. Autorzy zasymulowali działanie różnego typu workloadów za pomocą 14 symulatorów. Na podstawie zebranych metryk (CPU System/User, Network Bytes In/Out, Disk IO IN/OUT, Swap IN/OUT) wyselekcjonowano cztery typy workloadów: intensywnie korzystające z procesora, intensywnie korzystające z wejścia/wyjścia, intensywnie korzystające z pamięci i bezczynne. W oparciu o zdefiniowane kategorie i zebrane dane wytrenowano odpowiedni klasyfikator. Następnie zmodyfikowano algorytm alokacji workloadów tak, by brał pod uwagę informację o kategorii workloadu i nie alokował workloadów tej samej kategorii na tym samym węźle. Zaprezentowane eksperymenty były podzielone na dwie fazy. W pierwszej z nich symulatory zostały uruchomione bez wiedzy o kategoriach wor-

kloadów, natomiast drugiej została wykorzystana wiedza o workloadzie. W trakcie testów zebrano informacje o wydajności każdego z symulatorów - metryki zwracanej przez symulator. Porównanie wyników pokazało ponad 20% wzrost wydajności działania symulatorów, gdy wykorzystywana była informacja o kategorii workloadu i workloady tej samej kategorii nie były uruchamiane na tym samym węźle.

W artykule [128] autorzy przedstawiają model FBWC (ang. *Feedback-Based Workload Classification*) służący profilowaniu systemu operacyjnego pod działanie konkretnej kategorii workloadu. Autorzy zakładają pięć kategorii workloadów: intensywnie korzystające z procesora (CPU-intensive), intensywnie korzystające z pamięci (memory-intensive), intensywnie korzystające z operacji we/wy (I/O-intensive), mieszane (ang. *compound*). Zaproponowany model składa się z agenta monitorującego 22 wybrane metryki, procesora danych, klasyfikatora TSRSVM (ang. *Training Set Refresh Support Vector Machine*), narzędzia zarządzającego oraz profilera systemu operacyjnego. Do testów wykorzystano 16 syntetycznych testów o różnych charakterystykach. Przeprowadzone eksperymenty wykazały, że zaproponowany klasyfikator ma większą skuteczność niż klasyfikator maszyny wektorów nośnych czy k-najbliższych sąsiadów. W przeprowadzonych testach dokładność klasyfikacji osiągała 95%. Wyniki klasyfikacji były bezpośrednio przekładane na modyfikacje w systemie operacyjnym workloadu. Zmiany na tym poziomie skutkowały ponad 20% zwiększeniem wydajności testowanych symulatorów.

W pracy [92] została opisana rozproszona platforma iBalloon, która jest w stanie prognozować wykorzystanie zasobów generowanych przez dwuwarstwową aplikację internetową. Zaproponowana biblioteka wykorzystuje algorytm bazujący na metodzie uczenia przez wzmacnianie i wykorzystujący cztery metryki jako parametry wejściowe: wykorzystanie procesora i pamięci, liczba operacji we/wy i wykorzystana przestrzeń wymiany. Autorzy wykazują, że biblioteka iBalloon zapewnia wyniki zbliżone do optymalnych (minimalizacja czasu odpowiedzi serwera) i wykazują wyższość ich rozwiązania nad innymi metodami, takimi np. autoregresyjną średnią ruchomą (ARMA, ang. *autoregressive moving average*) [11].

Inną biblioteką przeznaczoną do automatycznej konfiguracji parametrów ma-

szyn wirtualnych jest VCONF [93]. Autorzy przedstawiają wyniki eksperymentów dla trzech rodzajów obciążeń: e-commerce (TPC-E), przetwarzanie transakcji online (TPC-C) i serwer aplikacji (SPECweb). Testy zostały wykonane dla różnych konfiguracji sprzętowych i dla różnych parametrów obciążeń, m.in. różnej liczby użytkowników. Działanie biblioteki opiera się na pomiarach trzech parametrów: czasu wykorzystania procesora, obciążenia procesorów wirtualnych oraz ilości wykorzystanej pamięci RAM. Algorytm oparty jest na sztucznej sieci neuronowej i łańcuchach Markowa. Jego wykorzystanie do dynamicznej rekonfiguracji maszyn wirtualnych poprzez modyfikację ustawień systemowych doprowadziło również do wzrostu wydajności chmury.

W artykule [65] zaprezentowano natomiast nową metodę migracji maszyn w chmurze obliczeniowej. Na potrzeby analizy autorzy zaproponowali następującą klasyfikację obciążeń: bezczynna, aplikacje transakcyjne, serwery plików, obliczenia naukowe, serwery aplikacyjne. W pracy została przedstawiona analiza wymagań dotyczących migracji maszyny z różnymi workloadami dla każdej z zaproponowanych klas.

Kolejną propozycją podziału workloadów ze względu na prezentowaną funkcjonalność zaprezentowano w [60]. Zaproponowano następujący ich podział: serwery aplikacyjne, bazy danych, serwery plików. W eksperymentach gromadzono następujące metryki: wykorzystanie CPU i pamięci oraz opóźnienia na operacjach wejścia-wyjścia. W artykule zaproponowano dwie metody modelowania workloadów generowanych przez różne klasy oparte o sieci neuronowe i metodę wektorów nośnych. Dla obydwu metod zaprezentowano wyniki działania modeli i zweryfikowano ich dokładność oraz wykazano, że są one dokładniejsze od modeli regresyjnych z którymi dokonano porównania.

W przedstawionych publikacjach możemy wyróżnić dwa trendy. Autorzy skupiają się na podziale workloadów ze względu na intensywność wykorzystania zasobów sprzętowych lub starają się scharakteryzować konkretną grupę funkcjonalną workloadów. W przeprowadzanych badaniach istnieje luka w postaci próby utworzenia ogólnych kategorii funkcjonalnych workloadów i weryfikacji czy dla tak zdefinio-

wanych kategorii można utworzyć klasyfikator, który działałby z wysoką skutecznością podobną do przedstawionych powyżej (około 95%). Dodatkowo dla zaproponowanych kategorii można rozważyć zagadnienie wzajemnego wpływu workloadów różnych kategorii na siebie w czasie kiedy zostaną uruchomione na jednym węźle chmurowym. To są zagadnienia, które zostały przeanalizowane i rozwiązane w rozprawie doktorskiej.

3.4 Problemy optymalizacji zarządzania zasobami

Koszt wykorzystania zasobów chmurowych wynika bezpośrednio z czasu ich wykorzystania, ceny jednostkowej oraz wielkości wykorzystanych zasobów. Dostawca chmury monitoruje za pomocą dedykowanych narzędzi wielkości fizycznie zarezerwowanych zasobów przez użytkowników chmury obliczeniowej na poziomie IaaS, zazwyczaj są to zasoby podstawowe typu zajętość procesora, pamięci RAM, wykorzystana przepustowość sieci czy zajętość dysku, jednak często monitorowanie są również inne parametry, np. wykorzystanie buforów sieciowych czy przełączenia kontekstu procesora. Oprócz tego użytkownicy chmury mogą czerpać korzyści od użytkowników końcowych na podstawie wykorzystania ich usług i aplikacji np. na poziomie PaaS lub SaaS. Model rozliczeń opisywany jest już w umowach (patrz podrozdział 3.2) i może dotyczyć miesięcznej opłaty za wykorzystanie zasobów wirtualnych, uwzględniając również typ wykorzystywanych usług i aplikacji, czy też przyjęty poziom dostępności i jakości usług. Zawarta umowa może też uwzględniać pewne odszkodowania związane z naruszeniem uzgodnionych wymagań (np. dłuższego czasu niedostępności chmury).

Współpraca użytkownik - dostawca chmury wymaga wielu uzgodnień natury formalnej, prawnej, jak i technicznej. Odbywa się to poprzez zawarcie odpowiedniego kontraktu. Większość dostawców oferuje swoim klientom standardowe SLA. Dotyczą one następujących kwestii:

- obowiązków w zakresie przetwarzania danych i poziomu jakości usług (dostępności wydajności odpowiedzialności za utratę danych), a także poziom

wsparcia dostawcy,

- poziomu ryzyka podjętych zobowiązań oraz czynności zmniejszających skutki jego wystąpienia, w tym parametry dopuszczalne długości przerw zaplanowanych i nieplanowanych, przestojów pracy chmury, obowiązków ujawniania pewnych informacji osobom trzecim np. dla organów ścigania, czy podawania tożsamości podmiotów świadczących usługi,

- czasu obowiązującego porozumienia, możliwości jego wypowiedzenia, cennika opłat w zależności od zakresu usług i poziomu ich wykorzystania, a także formy pomocy oferowanych przez dostawcę w okresie przejściowym, ochronę praw własności oraz wskazanie obowiązujących przepisów, które będą rozstrzygały różne sporne kwestie włącznie do odwoływania się do sądów.

Parametr dostępności usług określa się poprzez procentowy okres czasu, w którym może być ona wykorzystana. Dla przemysłu dostępność 99,99% oznacza, że przerwy w dostawie usługi mogą wynosić 1.01 minuty na tydzień, 4 minuty i 38 sekund na miesiąc, czy 52 minut i 56 sekund na rok. Zapewnienie dostępności może być gwarantowane w różny sposób np. poprzez nadmiarowość zasobów (np. kilka rozproszonych centrów danych) w tym redundancję zestawów danych. Warunki SLA mogą nie uwzględniać krótkich przerw (np. od 5 do 30 minut w Google Chrome Engine). Pomiar czasów dostępności można określić przez zewnętrzne narzędzia próbkujące np. Pingdom, czy przez logowanie odpowiedniej informacji z serwera przy wykorzystaniu wewnętrznych systemów monitorujących (MRTG, TRTG). SLA może uwzględniać też zwrot kosztów za przestoje, w różnej formie, dokonanie przelewu na konto użytkownika, w formie udzielenia kredytów na dalsze usługi. Wysokość tego typu kosztu może się wahać od 25% do 30% wielkości opłat za zrealizowane usługi. W praktyce określa się szczegółowo pewne wyjątkowe przypadki, np. czas wymiany uszkodzonego serwera (do 24 godzin). W takim przypadku dostępność jest mniejsza od 70% w skali miesiąca. Dostępność większa od 99,9% dotyczy na ogół jedynie dostępu do sieci Internet bądź zapewnienia gwarantowanego zasilania. Firmy oferujące usługi podają swój cennik usług, gdzie wysokość kosztów zależy od parametrów tych usług. Przy czym wybrane usługi dla pewnych parametrów mogą być bezpłatne.

Na ogół koszt wykorzystania chmury w warstwie IaaS jest proporcjonalny do następujących parametrów:

- liczba wirtualnych procesorów (od 1 core),
- wielkość pamięci RAM (od 512 MB),
- wielkość pamięci dyskowej, gdzie pierwsze, np. 40 GB tej pamięci jest w cenie innych zasobów, zaś dalsze 100 GB jest już płatne),
- transferu danych rozumianego jako liczbę przesyłanych pakietów, przy czym np. do 50 GB/miesiąc bezpłatnie,
- wielkości przekazywanego backupu, płatny np. od 1 GB,
- liczba publicznych adresów IP,
- dostępu do VLAN poprzez liczbę wykorzystywanych licencji,
- liczba stworzonych wirtualnych sieci,
- liczba routerów (wirtualne urządzenia),
- liczba operacji odczytu i zapisu w przestrzeni wymiany danych (IOPS).

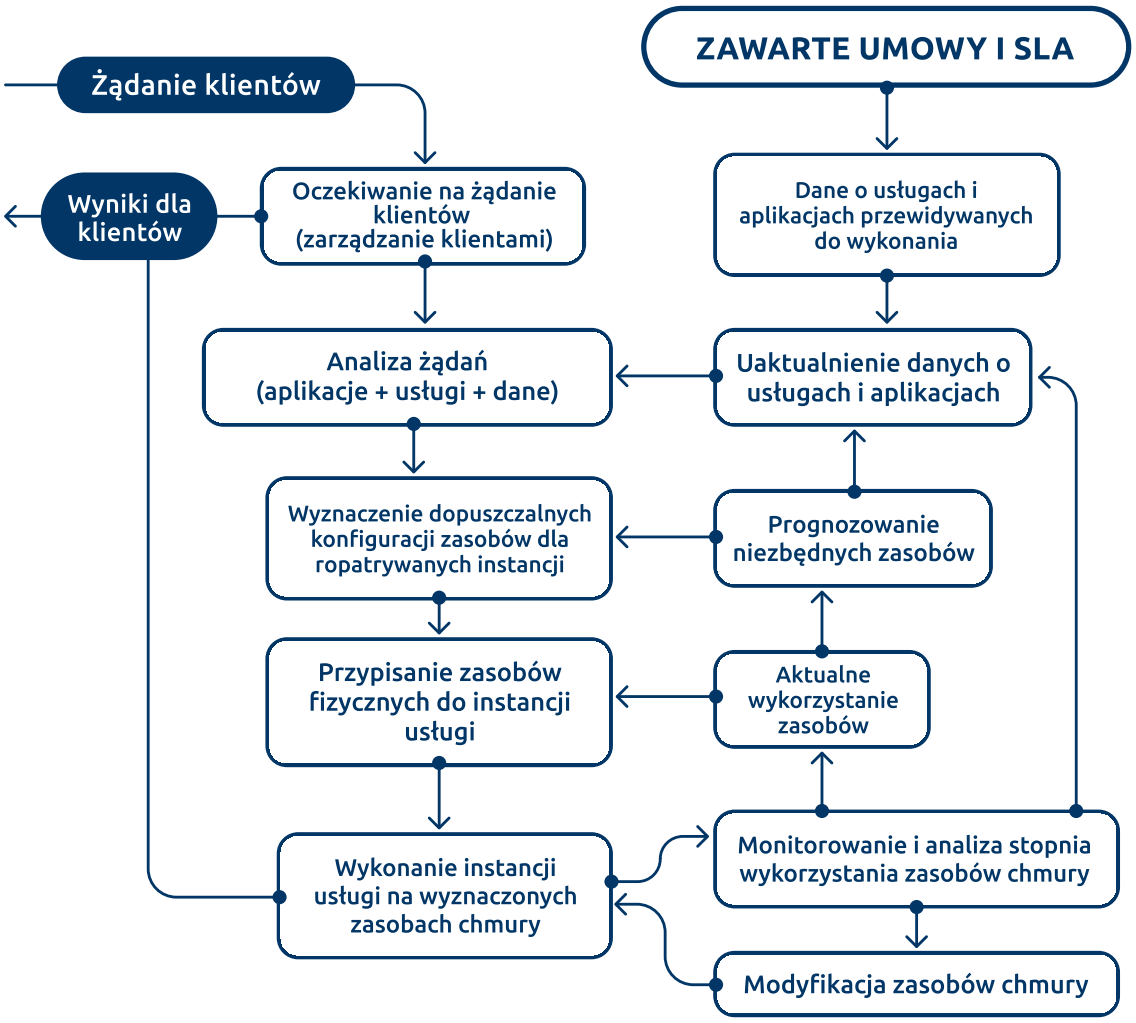
Możliwe jest również zapewnienie dostępu do innych, bardziej specjalistycznych usług takich jak:

- liczba reguł sieciowych w usłudze transmisją (ang. *load balancing*),
- liczba i rodzaj wykorzystanych licencji na system operacyjny, np. Windows Server, Red Hat Linux,
- optymalny dobór zasobów do wykonywanej aplikacji liczba rdzeni, wielkość pamięci RAM, częstotliwość i wielkość HDD,
- monitorowanie obliczeń w czasie rzeczywistym (np. ocena wykorzystania zasobów)

- dostęp do konsoli w przeglądarce do zarządzania serwerem,
- możliwość wykorzystania protokołu REST API do włączania, wyłączania serwerów,
- wybór systemu operacyjnego o zadanej konfiguracji,
- wykorzystanie powiadomień SMS przez helpdesk,
- wykorzystanie programów zrównoważenia obciążenia wskazanych serwerów,
- kontrola bieżących wydatków związanych z wykorzystaniem odpowiednich zasobów,
- wsparcie techniczne w przypadku różnego typu problemów przy wykorzystaniu chmury obliczeniowej.

Wielkość opłat jak i zakres dodatkowych możliwości różnią się w zależności od rozwiązań chmurowych, dąży się jednak do ustandaryzowania świadczeń usług w chmurze, co jednak ze względu na różnorodność rozwiązań nie jest sprawą prostą. Przy płatności za usługi wykorzystuje się systemy, które umożliwiają szybkie np. przez jedno kliknięcie oraz bezpieczne regulowanie opłat między klientem posiadającym kartę kredytową, a dostawcą usługi. Tego typu rozwiązania (PayU, PayPal) odgrywają rolę pośredników przy tego typu finansowych transakcjach internetowych.

Rysunek 3.4 prezentuje uproszczony schemat zarządzania chmurą obliczeniową z punktu widzenia dostarczania zasobów i alokacji workloadów. Pierwszym elementem jest etap oczekiwania na żądania przychodzące od klientów. Po odebraniu żądania następuje etap analizy, w którym weryfikowane jest co ma zostać uruchomione oraz czy podane są wszystkie niezbędne do wykonania żądania parametry. Kolejnym etapem jest weryfikacja czy w chwili żądania dostępne są zasoby spełniające żądane parametry. Wymagane mogą być dostępy do specyficznych urządzeń jak np. akceleratory graficzne czy też licencji, do których dostęp w niektórych przypadkach może być niemożliwy, np. ze względu na już uruchomione inne usługi i wyczerpany limit dostępnych licencji. W kolejnym kroku z dostępnych zasobów wybierane są, te które



Rysunek 3.4: Schemat zarządzania workloadami chmury obliczeniowej (źródło: własne)

zgodnie z aktualnie wybranym w chmurze sposobem alokacji są najbardziej preferowane do uruchomienia zadania. W ostatnim kroku następuje uruchomienie usług na wybranych zasobach.

Równolegle w chmurze w sposób ciągły działają procesy monitorujące pracę chmury oraz wpływające na proces alokacji maszyn wirtualnych. Wraz z uruchomieniem workloadu następuje rozpoczęcie procesu monitorowania wykorzystanych przez niego zasobów oraz weryfikowane są wolne zasoby. Potencjalnie taki proces może wysyłać odpowiednie informacje do dostawcy i informować o kończących się zasobach. Chmura obliczeniowa bazując na zebranych informacjach może również wykorzystywać mechanizmy prognozowania dostarczając dostawcy oszacowania na

potrzebne zasoby w oparciu o dane historyczne. Wszystkie gromadzone informacje o workloadach są jednocześnie weryfikowane pod kątem spełnienia przez dostawcę warunków SLA zawartych w umowach.

Zgodnie z ideologią chmury obliczeniowej a konkretnie aksjomatem traktującym o nieskończonych zasobach nie powinna nastąpić sytuacja, w której użytkownik nie będzie mógł zrealizować żądania uruchomienia nowego workloadu. Zasoby w chmurach udostępniane przez dostawców są bardzo duże i sukcesywnie rozbudowywane, ale jednak są zawsze ograniczone z góry. W skrajnym przypadku, w szczególności przy lokalnych rozwiązaniach, możliwe jest że zasoby zostaną wykorzystane i uruchomienie nowego workloadu nie będzie możliwe. Większość systemów chmurowych nie ma możliwości kolejkowania żądań uruchomienia workloadów stąd w przypadku braku zasobów nastąpi odrzucenie żądania z odpowiednim kodem błędu. Użytkownik może w takim przypadku zmniejszyć wymagania dotyczące uruchamianego workloadu, np. zmniejszając wymaganą liczbę wirtualnych procesorów lub zmniejszając wymaganą wydajność magazynu danych. Może to pozwolić na uruchomienie maszyny, o ile wymagania zostaną na tyle zmniejszone, że uda się przydzielić wolne zasoby.

W zależności od modelu chmury obliczeniowej, w którym świadczone są usługi, zarządzanie zasobami rozumiane jest w różny sposób. Dla modelu IaaS zarządzanie chmurą polega na przydzielaniu dla maszyny wirtualnej zasobów danego węzła i tworzeniu maszyn wirtualnych. W przypadku modelu PaaS zarządzanie skupia się na przydzielaniu kontenerów (ang. *pods*) do węzłów obliczeniowych platformy (ang. *worker*). Natomiast dla modelu SaaS zarządzanie polega na przydzielaniu klientów do usługi a następnie jej odpowiednim zarządzaniem, tak by działała w sposób wydajny.

Zarządzanie chmurą obliczeniową a konkretnie alokacja zasobów może być wykonywana pod kątem różnych kryteriów optymalizacji. W artykule [72] autorzy wyróżniają sześć kategorii algorytmów alokacji zasobów dzieląc je pod kątem przyjętego kryterium optymalizacyjnego na przykładzie modelu IaaS.

- Optymalizacja kosztu - w której brane pod uwagę mogą być zarówno koszty do-

stawcy, korzyści dostawcy, koszty użytkownika oraz koszt samych zasobów [129, 50, 9, 90, 87].

- Optymalizacja efektywności wykonania - dla tej kategorii brane pod uwagę mogą być takie kryteria jak czas odpowiedzi usługi, czas wykonania, przepustowość sieci, koszt wykonania, czy też priorytetyzacja zadań [94, 8, 71, 1].

- Optymalizacja wykorzystania energii elektrycznej - w tej grupie kluczowym elementem jest minimalizacja kosztu energii [44, 121, 18, 41, 26, 24, 27].

- Optymalizacja zrównoważenia obciążenia - polegająca na równomiernym rozłożeniu obciążenia pomiędzy węzłami chmury obliczeniowej [85, 51, 3].

- Optymalizacja pod kątem QoS - algorytmy alokacji skupiają się na optymalizacji dostępności, niezawodności, zachowania odporności na błędy i krótkiego czasu przywrócenia po awarii jak również zachowania parametrów SLA dla działających usług [101, 63, 59, 39, 7].

- Optymalizacja wykorzystania zasobów - algorytmy minimalizują wykorzystanie zasobów [15, 99, 46].

W zależności od przyjętego przez dostawcę modelu oraz celów, które chce osiągnąć mogą zostać przyjęte różne algorytmy alokacji żądań maszyn wirtualnych. Jednocześnie mnogość rozwiązań pokazuje jak ważnym aspektem zarządzania chmurą są algorytmy alokacji zadań i pomimo, że część z nich jest przedstawiana wyłącznie teoretycznie i nigdzie nie wykorzystywana to staje się ona podstawą do dalszych poszukiwać optymalnych rozwiązań.

W rozprawie doktorskiej, wykorzystując doświadczenia projektu KRICO [56], w którym autor rozprawy był jednym z głównych wykonawców i projektantów pakietu rekomendacji dla środowisk chmurowych, zaproponowano oryginalną metodę estymacji rozmiarów zasobów oraz klasyfikatora dla workloadów uruchamianych na żądania użytkowników chmury. Wspomniana klasyfikacja jest powiązana z różnymi klasami funkcjonalnymi workloadów, które są uruchamiane przez użytkownika i mają

zostać wykonane zgodnie z warunkami SLA. Takie podejście jest możliwe po zdefiniowaniu takich klas oraz przeprowadzeniu odpowiednich eksperymentów w celu zweryfikowania na ile jest to w praktyce możliwe. Znając wnoszone obciążenie generowane przez workload oraz aktualne obciążenie zasobów chmury możemy dokonać optymalizacji alokacji workloadu znając jego klasę funkcjonalną. W rozdziale 4 zaproponowano podział workloadów na klasy funkcjonalne oraz przedstawiono klasyfikator pozwalający na wykrycie klasy workloadu na podstawie charakterystyki wykorzystywanych zasobów obliczeniowych. Następnie w rozdziale 5 zaproponowano praktyczne wykorzystanie klas funkcjonalnych do optymalizacji procesu alokacji workloadów w chmurze obliczeniowej.

Rozdział 4

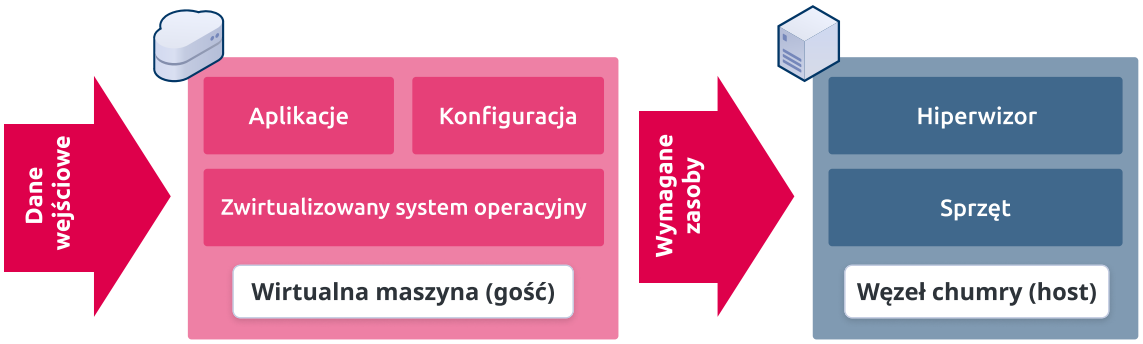
Klasy funkcjonalne workloadów chmurowych

Wyjaśniono pojęcie workloadu oraz zaprezentowano jego podstawowe elementy składowe. Zaproponowano podział funkcjonalny workloadów i zaprezentowano środowisko do przeprowadzenia eksperymentów pozwalających na wyróżnienie kategorii workloadów z wykorzystaniem algorytmu klasteryzacji. Zweryfikowano spójność otrzymanych kategorii z klasami funkcjonalnymi. Opracowano klasyfikator pozwalający na identyfikację klas workloadów na podstawie wykorzystywanych zasobów chmury obliczeniowej. Ukazano możliwości wykorzystania opisanej klasyfikacji funkcjonalnej i klasyfikatora do estymacji i alokacji zasobów.

4.1 Charakterystyka workloadów

Rozprawa doktorska, jak wspomniano wcześniej, skupia się na rozwiązaniu przedstawionych problemów świadczenia usług chmurowych w modelu IaaS. Z punktu widzenia dostawcy najważniejszymi elementami w modelu IaaS są maszyny fizyczne tworzące infrastrukturę informatyczną chmury oraz maszyny wirtualne tworzone przez użytkowników. Obydwoma te byty muszą być w odpowiedni sposób zarządzane przez dostawcę. Na Rysunku 4.1 przedstawiono element łączący maszyny fizyczne i wirtualne nazywany workloadem. Zgodnie z przyjętą nomenklaturą maszyny wirtu-

alne są powszechnie nazywane gościem (ang. *guest*), a maszyny fizyczne nazywane są hostami (ang. *hosts*) [10].



Rysunek 4.1: Odwzorowanie składowych workloadu na zasoby chmury obliczeniowej (źródło: [56])

Pojęcie workloadu używane w kontekście modelu IaaS reprezentuje maszynę wirtualną uruchomioną w środowisku hipernadzorcy na maszynie fizycznej z wydzielonymi zasobami, które mogą zostać przez nią wykorzystane. Definicja workloadu jest jednak szersza niż sama maszyna wirtualna, czyli z punktu widzenia systemu operacyjnego jest to proces uruchomiony na węźle chmurowym. Przez worklad rozumiemy wszystkie czynności, które są wykonywane w ramach realizacji usługi. Stąd też w pojęciu workloadu mieści się również system operacyjny gościa zainstalowany w maszynie wirtualnej oraz aplikacja zainstalowana i skonfigurowana do realizacji usługi. Dodatkowo przy rozważaniu i opisywaniu workloadu należy uwzględnić dane wejściowe, które determinują ostateczną charakterystykę zasobów, które są niezbędne do realizacji usługi. W zależności od rodzaju workloadu danym wejściowym towarzyszy pewna specyfika. W przypadku workloadów nieinteraktywnych takich jak symulacje naukowe zazwyczaj jest to określony i podany przy uruchomieniu workloadu zestaw danych. Natomiast dla workloadów interaktywnych najczęściej danymi wejściowymi jest pewien wzorzec zachowania użytkownika [126].

W usłudze chmurowej IaaS każda maszyna wirtualna uruchomiona jest na pojedynczym fizycznym węźle chmurowym. Węzeł dobierany jest przez mechanizmy zarządzania chmurą, szczegółowo opisane w rozdziale 5. Na jednym węźle fizycznym może działać jednocześnie jedna lub więcej maszyn wirtualnych. Możliwość bez-

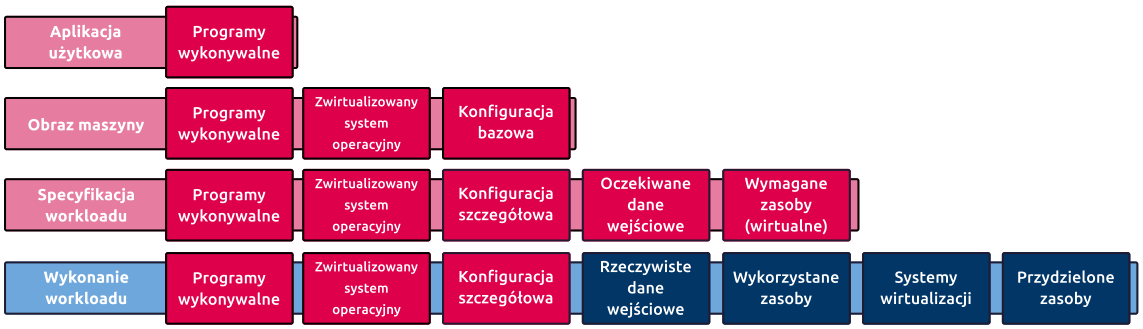
piecznego (z zachowaniem izolacji zasobów) i wydajnego uruchamiania wielu maszyn wirtualnych na jednym węźle jest jedną z kluczowych zalet przetwarzania w chmurze.

W związku z rosnącą popularnością wykorzystania chmur obliczeniowych, przeprowadzono wiele badań związanych z analizą workloadów, które są w nich uruchamiane. W badaniach skupiano się głównie na optymalizacji wykorzystania zasobów chmurowych i kosztów z tym związanych [6]. Jednym z obszarów badań były prace związane z odpowiednią alokacją konkretnych typów workloadów oraz predykcją dynamiki zmian obciążenia generowanego przez workload. Odpowiednie mechanizmy predykcji pozwalały na przeprowadzenie migracji przed przewidywanym wzrostem wykorzystania zasobów a w konsekwencji pozwalały na równoważenie obciążenia pomiędzy węzłami [28, 106, 34] oraz uniknięcia problemu hałaśliwych sąsiadów (ang. *noisy-neighbor*) [67].

Migracja maszyn wirtualnych jest jednym z podstawowych mechanizmów wykorzystywanych przez dostawców do utrzymania ciągłości działania maszyn wirtualnych użytkowników. Mechanizm pozwala na bezprzerwowe (z punktu widzenia użytkownika) przeniesienie maszyny gościa na inny węzeł, dzięki czemu dostawca może poddać pracom konserwacyjnym węzeł fizyczny przy jednoczesnym zachowaniu poziomu SLA. Chociaż migracja maszyn jest jednym z kluczowych aspektów efektywnego zarządzania zasobami chmury to niestety wiąże się on z kosztami [91]. Stąd też prowadzone są badania mające na celu przewidywanie zapotrzebowania na zasoby dla różnych typów workloadów przed ich uruchomieniem [14]. Celem tych badań jest dostarczenie oprogramowaniu do zarządzania chmurą dodatkowej wiedzy o obciążeniach, tak aby zoptymalizować pierwszą alokację workloadu.

W celu usystematyzowania nazewnictwa na Rysunku 4.2 przedstawiono definicje podstawowych pojęć dotyczących wykonywania workloadów w modelu IaaS:

- **Aplikacja użytkowa** – program lub programy wykonywalne wraz z powiązanymi bibliotekami realizujący zadaną funkcjonalność. Przykładową aplikacją może być baza danych SQL, np. MariaDB czy narzędzie współdzielenia plików NextCloud.



Rysunek 4.2: Podstawowe elementy definiujące workload IaaS (na czerwono zaznaczono elementy niezależne od środowiska, w którym wykonywany jest workload, natomiast na niebiesko elementy, które zmieniają się w zależności od środowiska w którym workload został uruchomiony) (źródło: [56])

- **Obraz maszyny** – przygotowany i dostępny w chmurze obraz maszyny wirtualnej, posiadający zainstalowaną jedną lub więcej aplikacji użytkowych, oraz przygotowanym systemem operacyjnym, np. Ubuntu Server, OpenSuse, Microsoft Windows Server. Zazwyczaj dostawy udostępniają użytkownikom zbiór obrazów, które mogą zostać wykorzystane.
- **Specyfikacja workloadu** (ang. workload specification) – obraz maszyny wirtualnej z konkretnym systemem operacyjnym, rozbudowany w stosunku do gotowego obrazu o oczekiwane dane wejściowe oraz przewidywane zasoby wirtualne do realizacji usługi. Przykładowa specyfikacja workloadu to przygotowany obraz Ubuntu Server z bazą danych MariaDB realizujący średnio 1000 zapytań na minutę. Wymagania sprzętowe to 2 wirtualne procesory, 4 GB pamięci RAM oraz 10 GB pamięci masowej.
- **Wykonanie workloadu** – uruchomienie workloadu zgodnie ze specyfikacją w środowisku chmurowym z wykorzystaniem konkretnego hipernadzory. Wymagania na zasoby zastępowane są już konkretnie przydzielonymi zasobami, a w miejsce oczekiwanych danych wejściowych pojawiają się rzeczywiste dane i/lub strumienie zapytań. W przypadku wykonania workloadu możliwe jest monitorowanie wykorzystania (zbieranie metryk) przydzielonych zasobów.

Na przykładzie TASKcloud wykonanie workloadu jest równoważne z uruchomie-

niem maszyny wirtualnej bazującej na przygotowanym przez dostawcę lub użytkownika obrazie systemu operacyjnego. Na etapie uruchomienia użytkownik definiuje jego specyfikację wybierając odpowiednią odmianę maszyny (ang. *flavor*). Odmiana maszyny jest predefiniowanym opisem zasobów wirtualnych, które mają być przydzielone maszynie przy uruchomieniu. Przykładowa odmiana dostępna w TASKcloud to odmiana o nazwie 2CPU-5RAM-20HDD, która definiuje przydzielenie maszynie 2 wirtualnych rdzeni procesora, 5 GB pamięci operacyjnej oraz 20 GB przestrzeni przechowywania danych na dyskach HDD. Po uruchomieniu maszyny użytkownik loguje się do uruchomionego systemu operacyjnego zazwyczaj za pomocą protokołu SSH i może rozpocząć wdrożenie aplikacji użytkowej zgodnie z określonymi wymaganiami.

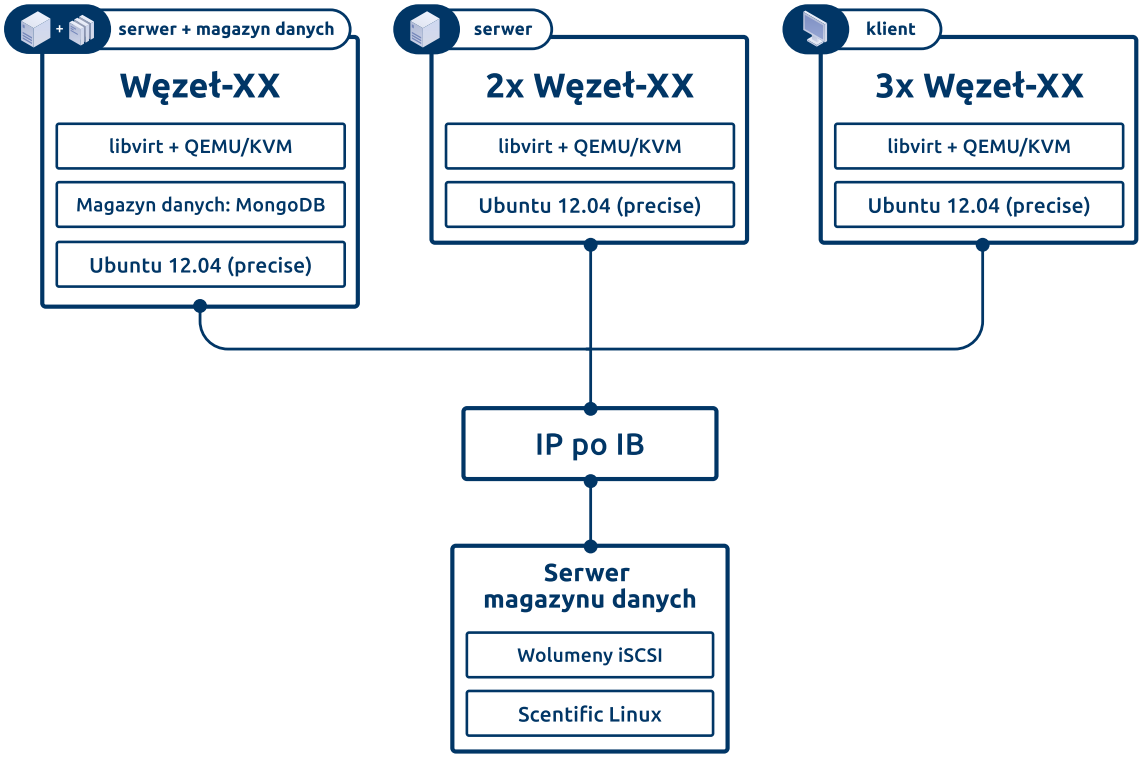
W rozdziale 3 przedstawiono metody estymacji obciążenia chmury obliczeniowej przez zgłoszony do wykonania workload oraz podano reprezentatywne kryteria optymalizacji jego wykonania przy znajomości danych o wykorzystywanych przez workload zasobach. Z reguły rozważane w literaturze problemy teoretyczne nie są adekwatne do potrzeb zarządzania chmurą. Dlatego w rozprawie (podobnie jak w wielu innych publikacjach [93, 92, 128, 125]) zdecydowano się na wykorzystanie metod eksperymentalnych opartych o symulatory, które uwzględniają zarówno charakterystykę badanej chmury obliczeniowej jak też charakterystykę metod obliczeniowych wykorzystywanych w analizowanych symulatorach.

Jednocześnie należy zaznaczyć, że wykorzystanie konkretnej instancji chmury nie wyklucza zastosowania opracowanej metody dla innych rozwiązań. Eksperymenty wykonywane dla konkretnej chmury mogą zostać przeanalizowane i na tej podstawie może być wybrany podzbiór eksperymentów przeprowadzony dla innych rodzajów chmur w celu oceny poprawności działania zaproponowanych rozwiązań. W związku z wykorzystaniem do testów środowiska TASKcloud zbudowanego na otwartym oprogramowaniu zakłada się, że opracowane rozwiązania będą miały zastosowanie również dla różnych chmur zbudowanych w oparciu o to samo rozwiązanie.

4.2 Środowiska laboratoryjne do przeprowadzenia eksperymentów

W ramach przeprowadzonych prac w projekcie KRICO zostały przygotowane trzy środowiska laboratoryjne, w których przeprowadzane były kolejne dwie fazy eksperymentów. Obserwowano wykonanie różnych workloadów na różnych węzłach chmurowych w celu identyfikacji zachodzących relacji pomiędzy klasą workloadu a wykorzystywaną wielkością zasobów obliczeniowych do ich wykonania.

Środowiskiem laboratoryjnym, w którym przeprowadzano pierwsze testy w ramach fazy I było środowisko zbudowane z wykorzystaniem węzłów obliczeniowych wydzielonych z superkomputera Galera+ wdrożonego w Centrum Informatycznym Trójmiejskiej Akademickiej Sieci Komputerowej Politechniki Gdańskiej. Uproszona architektura środowiska została przedstawiona na Rysunku 4.3).



Rysunek 4.3: Środowisko laboratoryjne dla eksperymentów w fazie I, węzły superkomputera Galera+ CI TASK (źródło: własne)

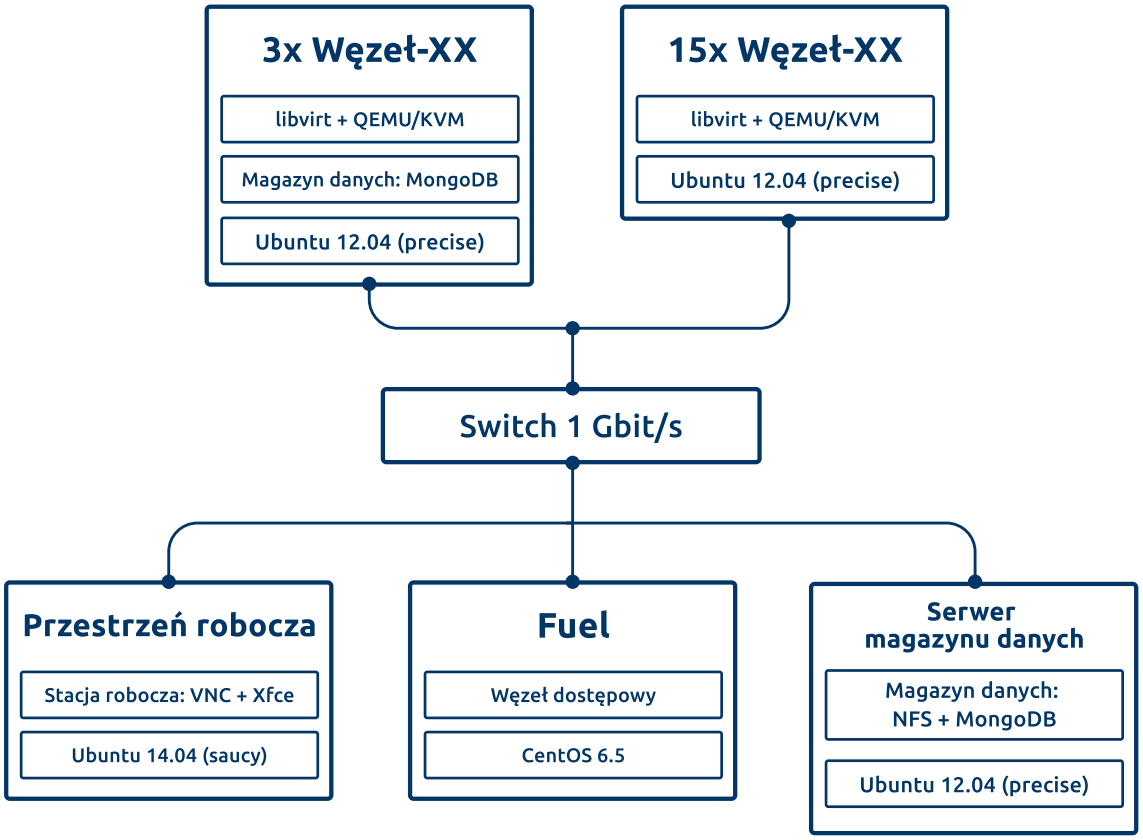
Środowisko zostało zbudowane z sześciu węzłów obliczeniowych. Każdy z nich był

wyposażonych w dwa procesor Intel(R) Xeon(R) L5640 i 16 GB pamięci operacyjnej. Węzły zostały połączone siecią Ethernet o przepustowości 1 Gbit/s oraz miały zamontowane dedykowane zasoby magazynu danych po sieci Infiniband w technologi IP over IB. Na węzłach został zainstalowany system operacyjny Ubuntu Server 12.04. Do wirtualizacji wykorzystano oprogramowanie QEMU/KVM wykorzystywane w środowisku docelowym tj. w oprogramowaniu OpenStack. Na maszynach gości wykorzystano system operacyjny Ubuntu Server 14.04.

Kolejnym środowiskiem, w którym były przygotowywane skrypty automatyzujące wykonanie i zarządzanie eksperymentami w fazie II było środowisko zbudowane na zasobach superkomputera Tryton uruchomionego w CI TASK. Środowisko wdrożono na 16 serwerach HP ProLiant DL380 Gen9. Każdy z nich był wyposażony w dwa procesory Intel(R) Xeon(R) CPU E5-2670 v3, 128 GB pamięci operacyjnej, lokalne dyski HDD o łącznej pojemności 3.8 TB oraz karty sieciowe 10 Gbit/s. Na serwerach zainstalowano system operacyjny Ubuntu 12.04. Następnie w środowisku wdrożono oprogramowanie OpenStack w wersji Kilo.

Środowiskiem, w którym zostały przeprowadzone wszystkie symulacje fazy II było środowisko przygotowane przez partnera projektu KRICO firmę Intel Technology Poland. Uproszczoną architekturę środowiska prezentuje Rysunek 4.4.

W środowisku wykorzystano dedykowane 21 serwerów, na których zainstalowano system operacyjny Ubuntu 12.04 oraz oprogramowanie OpenStack w wersji Juno. Wszystkie serwery były zbudowane w oparciu o platformę Intel Grizzly Pass z wykorzystaniem dwóch typów procesorów Intel(R) Xeon(R) E5 2680 oraz 2697, osadzonymi 64 GB pamięci operacyjnej oraz lokalnym dyskiem HDD o pojemności 1 TB. Wszystkie serwery były połączone siecią Ethernet o przepustowości 1 Gbit/s. Środowisko chmurowe zostało wdrożone za pomocą narzędzia Mirantis Fuel umożliwiające prosty, zautomatyzowany proces instalacji. W środowisku został wydzielony jeden węzeł dedykowany do gromadzenia danych z narzędzia monitorującego wdrożonego na wszystkich węzłach obliczeniowych środowiska.



Rysunek 4.4: Środowisko laboratoryjne dla eksperymentów fazy II, laboratorium Intel Technology Poland (źródło: własne)

4.3 Selekcja parametrów wykorzystania zasobów chmury obliczeniowej

Prowadzone w literaturze rozważania dotyczące klas workloadów uruchamianych w chmurze oraz metryki brane pod uwagę przy próbach dokonania podziału na klasy obciążeń bazują na danych gromadzonych na klastrach obliczeniowych HPC. W wielu przypadkach dane gromadzone są z uruchomienia symulatorów lub przyjętych modeli obciążeń [93, 92, 128, 125]. W sieci Internet możemy znaleźć zapisy metryk z rzeczywistych obciążeń z chmury obliczeniowej jednak są to surowe dane bez dodatkowych informacji o rodzaju obciążenia, które opisują [web15]. Należy zauważyć, że węzły obliczeniowe systemów HPC i chmur obliczeniowych oparte są o podobne architektury sprzętowe. Serwery mogą różnić się wydajnością, systemami chłodzenia czy też wielkością dostępnej pamięci RAM na korzyść klastrów obliczeniowych ze

względu na wykorzystywanie w chmurach tańszych i bardziej dostępnych rozwiązań (ang. *commodity hardware*). Z kolei w węzłach chmurowych częściej możemy spotkać lokalne dyski w przeciwieństwie do węzłów obliczeniowych HPC, gdzie systemy w większości ładowane są z dysków sieciowych, a dane do obliczeń pobierane z magazynów danych wysokiej wydajności HPS (*High Performance Storage*). W związku z wykorzystywaniem tych samych architektur sprzętowych przy charakterystyce węzła obliczeniowego możemy posłużyć się odniesieniami do węzła obliczeniowego HPC.

W kontekście chmury obliczeniowej najważniejszą rolę z punktu widzenia alokacji i wykonywania obliczeń pełnią węzły obliczeniowe (ang. *compute nodes*) stąd też w dalszej części pracy jeżeli mowa będzie o węźle chmurowym to należy przez to rozumieć węzeł obliczeniowy. Wraz z rozwojem technologii dostępne na rynku architektury sprzętowe przechodziły kolejne zmiany, gdzie główną siłą napędową były kolejne generacje procesorów. Na przestrzeni lat zmiany były mniej lub bardziej widoczne od platform jednoprocesorowych, przez wieloprocesorowe, modyfikujące sposoby dostępu do pamięci, na wspierające różnego rodzaju akceleratory obliczeniowe [38, 105]. Poniżej przedstawiono analizę metryk i narzędzi pozwalających na monitorowanie wykorzystanie zasobów węzła chmury obliczeniowej, które następnie po przeprowadzonej selekcji były brane pod uwagę w dalszych rozważaniach. Analiza została początkowo przeprowadzona w ramach realizacji projektu KRICO (Komponent Rekomendacji dla Inteligentnych Chmur Obliczeniowych) [web25] i była zadaniem realizowanym przez autora rozprawy. Celem wyboru odpowiedniego zestawu metryk było nie tylko scharakteryzowanie zachowania węzła obliczeniowego, ale również możliwości określenia na podstawie tego zbioru klasy workloadu, który został uruchomiony w chmurze. Z tego powodu pod uwagę brane były różne metryki w tym uwzględniające specyficzne parametry elementów sprzętowych jak np. przełączenia kontekstu procesora, obsługa przerwań czy też wykorzystanie pamięci cache procesora na różnych poziomach, wydajność operacji wejścia/wyjścia.

Wybór grup metryk bazował na opracowaniu [21], gdzie opisano zestawy metryk odzwierciedlające wykorzystanie węzła obliczeniowego przez działające na nim usługi. Na bazie przeprowadzonych analiz i przeglądu literatury wyselekcjonowano

zestaw parametrów możliwych do gromadzenia z poziomu systemu operacyjnego Linux. Najbardziej istotnymi zestawami parametrów były parametry procesora, pamięci oraz operacji wejścia-wyjścia.

W przypadku metryk procesora zbierano informacje pochodzące od ogólnych parametrów monitorowanego workloadu jak `/proc/loadavg`, gdzie zawarte są również dane dotyczące zarówno uruchomionych procesów, jak i tych czekających na odczyty z urządzeń wejścia-wyjścia. W analizie uwzględniono aspekty związane z potencjalnie nieliniową wydajnością względem liczby procesów wynikającą z faktu przełączania kontekstu przy dużej liczbie procesów [web40]. Dodatkowo przy małej liczbie zadań możliwe są przełączenia bez czyszczenia w pełni pamięci podręcznych procesora, jednak duża liczba zadań może powodować konieczność przeniesienia danych procesu z/do pamięci podręcznej. Z tego względu do zbioru metryk dołączono metryki związane z pamięcią podręczną L1 i L2, w tym m.in.

- liczba odwołań przy zapisie,
- liczba odwołań przy odczycie,
- liczba nietrafień przy zapisie,
- liczba nietrafień przy odczycie.

Pozwoliło to na uzyskanie pełniejszego obrazu obciążenia systemu, gdyż 100% obciążenie może wskazywać różny stan systemu, np. jedno zadanie wysycające możliwości procesora lub też wiele zadań, które wzajemnie się wywłaszczają. Liczba zmian kontekstów została uwzględniona bazując na danych zawartych w pliku [web40] `/proc/$$/status`. Do metryk dołączono również informacje o liczbie wykonanych operacji *fork* od uruchomienia systemu oraz liczbie procesów uruchomionych i oczekujących na uruchomienie. Wraz z obciążeniem zostały również uwzględnione metryki zawierające informacje o cyklach przeznaczonych przez procesora dla:

- przestrzeni użytkownika,
- przestrzeni użytkownika z niskim priorytetem,

- przestrzeni jądra,
- stanu bezczynności,
- obsługi przerwań,
- obsługi przerwań programowych.

Ze względu na architektury wieloprocesorowe należało również uwzględnić kopiowanie pamięci podręcznych pomiędzy procesorami. Wykorzystano w tym celu informację o identyfikatorze procesora, na którym proces był ostatnio uruchomiony zawartą w pliku `/proc/$$/stat`.

W grupie metryk związanych z operacjami wejścia-wyjścia najważniejszymi czynnościami są operacje `iowait` [web16]. Statystyka mówi o wielkości czasu spędzonego na oczekiwaniu na zakończenie operacji wejścia-wyjścia. W systemie tego typu dane dostępne są na dwóch poziomach - w formie zagregowanej w pliku `/proc/stat` bądź za pomocą narzędzia `top`. Istotnym jest zebranie danych z podziałem na urządzenia blokowe. Takie metryki zostały dodane do zbioru monitorowanych metryk bazując na informacjach uzyskanych z narzędzia `iostat`. Narzędzie zostało wykorzystano również ze względu na wnioski w [web38], gdzie stwierdzono, że `iowait` nie powinien być jedyną metryką monitorowania wejścia/wyjścia, i niezbędne jest gromadzenie informacji wydajnościowych właśnie narzędziem `iostat`.

Kolejnymi parametrami gromadzonymi na potrzeby analizy działania obciążenia węzła obliczeniowego były liczba stron zapisanych i odczytanych `page` oraz liczba zapisów i odczytów bloków `swap`. Im większe są te wskazane wartości tym może mieć do bardziej niekorzystny wpływ na działanie zadań. Wartości tych wskazanych metryk były odczytywane za pomocą narzędzia `dstat` [web23].

Dodatkowymi metrykami związanymi z odczytem i zapisem stron pamięci zgodnie z [web22], były nietrafienia stron *major page faults* (konieczność pobrania danych z dysku) oraz *minor page faults* (kiedy dane są już w pamięci). W analizie uwzględniono również parametry związane z operacjami buforowania danych z dysku w pamięci operacyjnej. Wzięto pod uwagę:

- `MemTotal` – cała dostępna pamięć RAM.
- `MemFree` – wolna pamięć RAM (patrz bufory poniżej).
- `Buffers` – pamięć zaalokowana dla operacji zapisu na dysk.
- `Cached` – dane odczytane z dysku i umieszczone w pamięci RAM.

W celu analizy typów workloadów dodano do zbioru dynamiczne metryki związane z metrykami podanymi powyżej tj. liczby stron odczytanych i zapisanych w jednostce czasu pgpgin/s pgpgout/s oraz liczby zdarzeń typu fault fault/s majflt/s. Ze względu na kontekst wirtualizacji w chmurze do zbioru metryk charakteryzujących węzeł obliczeniowy dodano metryki okresu czasu dotyczący aktywności pojedynczego systemu operacyjnego w środowisku zwirtualizowanym oraz czasu działania/uruchomienia wirtualnego procesora dla systemu gościa.

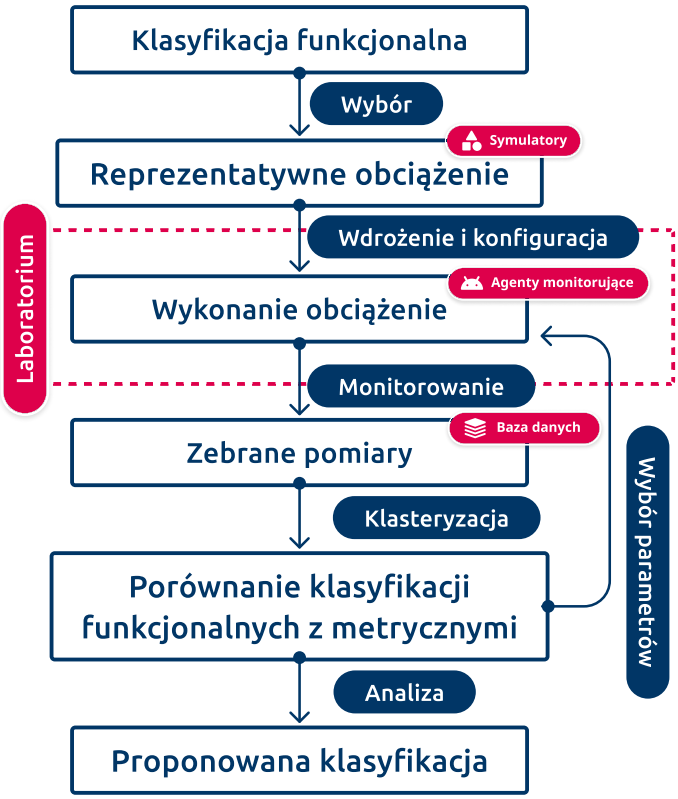
Na bazie przeprowadzonych analiz [21] wyznaczono zbiór 115 metryk systemowych, które były rozważane w kontekście zestawu na podstawie, którego możliwa byłaby klasyfikacja workloadów chmurowych. Pełen opis zestawu metryk wybranych w fazie I analizy prezentuje Tabela A.1 w Dodatku A.

W oparciu o wybrane metryki węzła obliczeniowego należało wyselekcjonować te z nich, które niosą ze sobą najwięcej informacji w kontekście rozróżniania kategorii obciążenia. Docelowy zbiór metryk powinien właściwie charakteryzować działanie węzła obliczeniowego oraz umożliwić łatwy sposób gromadzenia ich wartości, najlepiej za pomocą narzędzi dostępnych standardowo w systemie. Niezwykle istotnym jest, by gromadzone metryki nie były nadmiarowe: zarówno ze względu na koszt ich monitorowania jak i przechowywania. Dodatkowo ze względu na liczbę metryk zachodziła obawa negatywnego wpływu tzw. przekleństwa wymiarowości [115] na rezultaty uzyskiwane z użyciem metod wykorzystujących ocenę odległości. W pierwotnie opracowanym zbiorze występowała również duża korelacja pomiędzy przyjętymi metrykami, w związku z czym przeprowadzono ponowną analizę z użyciem metod analitycznych jak i oceny eksperckiej [22, 21, 107] i ograniczono zbiór do 53 metryk. Redukcję wymiarowości przeprowadzono z użyciem metod skalowania wielowymiarowego [58] oraz podejścia opartego na analizie składowych głównych [48]. W celu

ewaluacji eksperckiej oceny metryk przeprowadzonej w [21] wykonano analizę korelacji między poszczególnymi metrykami w zbiorach inicjalnych i zredukowanych. Wskazuje ona na znaczne zmniejszenie się skorelowanych cech przy jednoczesnym zachowaniu wariancji co pozwoli zachować jakość klasyfikacji danych. Listę metryk powstałą w fazie II prezentuje Tabela B.1 w Dodatku B.

4.4 Wybór klas workloadów chmurowych

Prace badawcze przedstawione w rozdziale 3.3 dotyczyły funkcjonalnych i automatycznie wyznaczanych klasyfikacji workloadów. W rozprawie doktorskiej proponowane jest rozszerzenie takiego podziału funkcjonalnego na podstawie przeglądu literatury oraz znajomości usług udostępnianych przez dostawców chmurowych. Wyniki przeprowadzonej takiej analizy zostały zawarte w pracy [84].



Rysunek 4.5: Scenariusz przeprowadzanych eksperymentów dla wyznaczenia klas workloadów (źródło: własne)

Zaproponowano procedurę eksperymentu, którego wynikiem miało być określenie zbioru klas funkcjonalnych workloadów chmurowych prezentuje schemat na Rysunku 4.5.

W pierwszym etapie wybrano klasy funkcjonalne workloadów. Wybrano następujące klasy:

1. workloady przetwarzania dużych zbiorów danych (ang. *Big Data*),
2. workloady buforowania danych (ang. *Data Caching)*,
3. workloady przetwarzania transakcyjnego (ang. *on-line transactions processing, OLTP*),
4. workloady strumieniowania danych multimedialnych (ang. *Streaming*),
5. workloady naukowe (ang. *Science*),
6. workloady prezentacji danych (ang. *Web Serving*).

Dla każdej z klasy dobrano reprezentantów (zwane dalej symulatorami klas), które imitowały działanie usługi reprezentującej daną klasę w środowisku laboratoryjnym. Symulatory zostały dobrane w oparciu o opisywane w literaturze narzędzia do symulowania workloadów chmurowych oraz o narzędzia przeznaczone do generowania obciążeń imitujących rzeczywiste workloady. W większości zastosowane symulatory są jednocześnie narzędziami oceniającymi wydajność testowanych workloadów. Listę wykorzystanych symulatorów prezentuje Tabela 4.1.

W zależności od klasy workloadu zaproponowano parametry sterujące symulatorem. Przyjętym założeniem był fakt, że parametry sterujące symulatorem miały być łatwe do zrozumienia dla użytkownika chmury. Dla części klas łatwiej jest użytkownikowi zdefiniować, np. spodziewaną liczbę zapytań do bazy danych lub liczbę równoległych klientów serwera aplikacyjnego niż potrzebny do tego celu rozmiar pamięci RAM. W związku z powyższym parametry te w dalszej części pracy będą nazywane parametrami użytkownika. Parametry użytkownika wybrane dla każdej z klas funkcjonalnych workloadów zaprezentowano w Tabeli 4.2.

Tabela 4.1: Symulatory (reprezentanci) workloadów wykorzystane w trakcie przeprowadzania eksperymentów

Lp.	**Kategoria workloadu**	**Nazwa symulatora**
1.	Big Data	PUMA Map Reduce [web12]
2.	Data Caching	Cloudsuite – caching [web10] Memtier [memtier] [web27]
3.	OLTP	OLTPBench [web11] CloudSuite – data serving [web10]
4.	Streaming	Cloudsuite – streaming [web10]
5.	Science	HPCG [hpcg] [web13] HPCC [hpcc] [web14]
6.	Web serving	Cloudsuite – web serving [web10] WP Mark Twenty Twelve [web41]

Tabela 4.2: Klasy workloadów wraz z parametrami użytkownika oraz liczbą uruchomionych testów

Lp.	**Kategoria obciążenia**	**Parametry użytkownika**	**Liczba konfiguracji**
1.	Big Data	• |processors|:/<int>/liczba procesorów do wykorzystania • |memory|:/<float>/rozmiar pamięci do wykorzystania[GiB] • |data|:/<float>/przewidywany rozmiar danych [GiB]	10
2.	Data Caching	• |memory|:/<float>/rozmiar bufora [GiB] • |ratio|:/<float>/przeywidywany stosunek odczytów do zapisów [0.0 - 1.0] • |clients|:/<int>/liczba równoczesnych klientów/połączeń	22
3.	OLTP	• |data|:/<float>/przewidywany rozmiar danych[GiB] • |clients|:/<int>/liczba równoczesnych klientów/połączeń	23
4.	Streaming	• |bitrate|:/<float>/średnia przepustowość strumienia[Mbps] • |clients|:/<int>/liczba równoległych klientów/połączeń	25
5.	Science	• |processors|:/<int>/liczba procesorów do wykorzystania • |memory|:/<float>/rozmiar pamięci do wykorzystania[GiB]	25
6.	Web Serving	• |clients|:/<int>/liczba równoległych klientów/połączeń	9

Następnie rozpoczęto etap kalibracji każdego z przyjętych symulatorów. Dla każdego z nich przebiegał on w następujący sposób. W pierwszym kroku uruchamiano dwie maszyny wirtualne na dwóch węzłach w środowisku laboratoryjnym. Maszyny miała przydzielone wszystkie 12 rdzeni danego węzła, 12 GB pamięci RAM oraz magazyn danych o pojemności 100 GB. Następnie przygotowywano na jednej maszynie testowany workload (np. bazę danych) a na drugiej maszynę kliencką z symulatorem (np. pakietem OLTPbench). W przypadku symulatorów niewymagających maszyny klienckiej symulator przygotowywano na jednej maszynie (np. w przypadku klasy HPC). Na każdym z węzłów na poziomie systemu operacyjnego hosta został zainstalowany agent monitorujący, który odczytywał wybranych 115 metryk i wysyłał je do specjalnie przygotowanej bazy danych MongoDB. Na tak przygotowanym środowisku można było rozpocząć właściwy etap kalibracji. Na podstawie empirycznej weryfikacji dobierano wskazane parametry użytkownika (patrz Tabela 4.2) dla kalibrowanego symulatora w taki sposób, aby przydzielone do maszyny wirtualnej zasoby były maksymalnie wykorzystane:

- Dobierano losowo wartość parametru.
- Uruchamiano symulator.
- Po zakończeniu symulacji na podstawie danych dostarczanych przez agenta monitorującego weryfikowano czy osiągnięto wysycenie zasobów.
- Jeśli nie, zwiększano wartości parametrów i wracano do kroku 2.
- Jeśli tak, parametry były zapisywane jako maksymalne wartości parametrów użytkownika dla danego symulatora w testowanym środowisku.

W trakcie procesu kalibracji symulatorów niezbędne było stałe śledzenie logów symulatora w celu weryfikacji czy działa on prawidłowo i nie są generowane, żadne informacje świadczące o błędach, które mogłyby negatywnie wpłynąć na dobór wartości parametrów.

Po zakończeniu procesu kalibracji dla każdego z symulatorów zostały określone maksymalne wartości parametrów użytkownika. Zbiór parametrów dla danej klasy

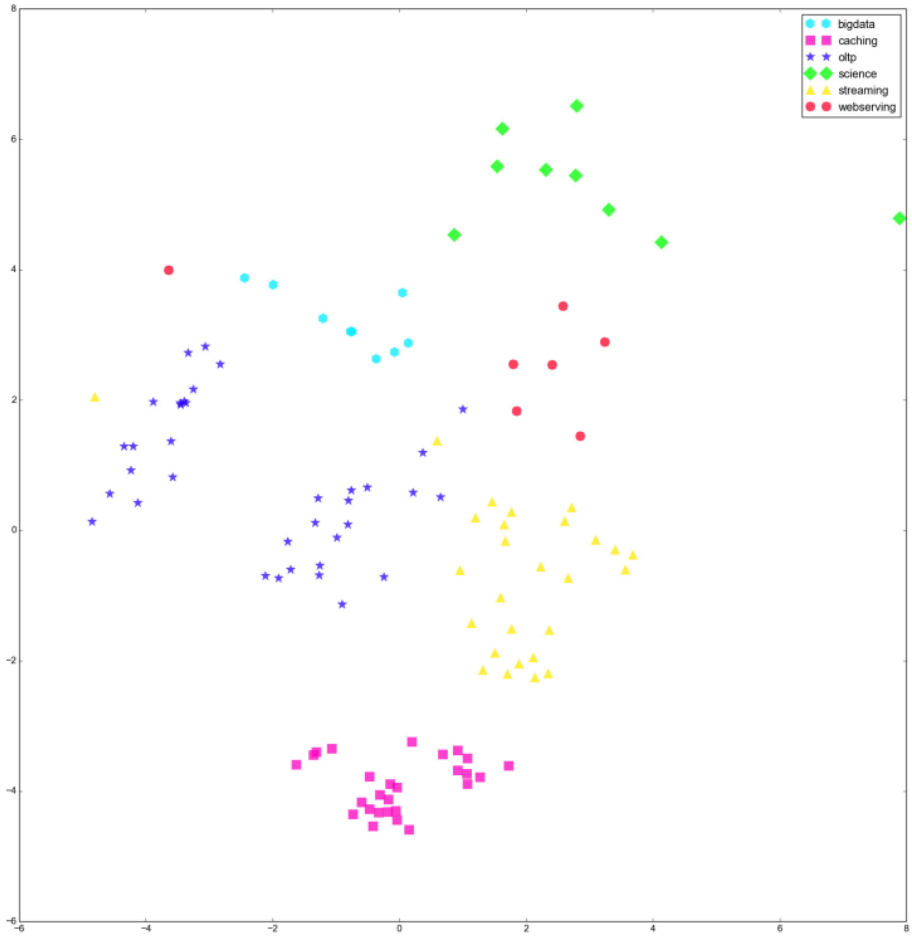
wraz z wartościami każdego z nich został nazwany konfiguracją workloadu. W celu zwiększenia liczby konfiguracji utworzono dodatkowe konfiguracje z wartościami parametrów o połowę mniejszymi od maksymalnych. Liczbę konfiguracji przygotowanych dla każdej z klas workloadów prezentuje Tabela 4.2.

Na tak przygotowanym zestawie konfiguracji rozpoczęto uruchamianie testów tzn. kolejnych symulatorów przygotowanych zgodnie z ustaloną konfiguracją. Dla zapewnienia powtarzalności wykonania eksperymentów proces uruchomienia i wykonania każdego z testów został zautomatyzowany. W trakcie działania symulatorów workloadów w środowisku laboratoryjnym agent monitorujący odczytywał i przesyłał do bazy danych wartości 53 metryk wybranych w podrozdziale 4.3. Agent monitorujący był zaimplementowany z wykorzystaniem niskopoziomowej biblioteki PAPI [web1]. Metryki były odczytywane co 5 sekund i zapisywane w nierelacyjnej bazie danych MongoDB. Przeprowadzono 114 testów na 10 symulatorach reprezentujących 6 klas funkcjonalnych. W trakcie fazy I eksperymentów za pomocą agenta monitorującego zgromadzono ponad 136 tysięcy próbek danych pomiarowych.

Kolejnym etapem było przygotowanie danych do analizy. W pierwszej kolejności odfiltrowano wartości odbiegające od normy. Dla wszystkich wartości każdej metryki wyznaczono wartość 3 i 97 percentyla a następnie odfiltrowani próbki, które były poniżej i powyżej wyliczonych progów percentyli. W kolejnym kroku dla każdej metryki wyliczono średnią wartość z przebiegu całego testu. Na tak przygotowanych próbkach w ostatnim kroku przeprowadzono ich normalizację (zachowano charakterystykę próbek zmieniając jego długość do wartości 1). Przygotowanie danych do procesu klasteryzacji zostało przeprowadzone zgodnie z dobrymi praktykami opisanymi w literaturze [124]. W rezultacie dla każdego uruchomionego testu z zadaną konfiguracją otrzymano pojedynczy wektor reprezentujący charakterystykę wykorzystanych zasobów w trakcie działania workloadu. Każda próbka posiadała 53 wymiary ukazując wartości uwzględnionych parametrów (Załącznik nr 1).

W celu zobrazowania danych na płaszczyźnie wykorzystano algorytm MDS (ang. *Multidimensional scaling*) [58]. W uproszczeniu algorytm ten przetwarza dane zmniejszając liczbę wymiarów, w których są reprezentowane z zachowaniem proporcji w od-

ległościach pomiędzy punktami. Punkt w przestrzeni reprezentuje pojedynczy test. Rysunek 4.6 prezentuje dane przygotowane do procesu klasteryzacji po przetworzeniu uzyskanych wyników za pomocą algorytmu MDS z przestrzeni 53-wymiarowej do 2-wymiarowej.



Rysunek 4.6: Wizualizacja danych z fazy I eksperymentów po przeskalowaniu do przestrzeni 2-wymiarowej (źródło: własne)

Każda z klas funkcjonalnych (patrz Tabela 4.1) została na wykresie przedstawiona za pomocą punktów reprezentujących poszczególne testy workloadów zostały oznaczone z klas funkcjonalnych Na wykresie każda z zaproponowanych kategorii obciążeń została przedstawiona za pomocą innej grafiki i koloru. Na uwagę zasługują skupienia punktów o tym samym kolorze i kształcie (tzn. tej samej klasie funkcjonalnej). Może to świadczyć o podobnej charakterystyce zasobów wykorzystywanych przez workloady tej samej klasy.

Po przygotowaniu tego typu danych zostały one wykorzystane do przeprowadzenia procesu nienadzorowanej klasteryzacji workloadów. Klasteryzację przeprowadzono za pomocą algorytmu k-średnich [2]. W związku z tym, że liczba klas w problemie była znana a priori przeprowadzono klasteryzację z użyciem parametru k=6 określającego liczbę oczekiwanych typów workloadów.

Wyniki uzyskane z algorytmu k-średnich poddano porównaniu z założoną klasyfikacją funkcjonalną. Klasteryzacja przeprowadzona na podstawie charakterystyk zasobów zużywanych przez workloady w 94% pokrywała się z zakładanymi klasami funkcjonalnymi. Rezultat działania algorytmy zweryfikowano za pomocą 10-krotnej walidacji krzyżowej. Przedstawione wyniki dowodzą, że dla zgromadzonych danych zaproponowane klasy funkcjonalne workloadów mogą być rozróżnione na podstawie charakterystyki wykorzystania zasobów węzła obliczeniowego. W przypadku zróżnicowanego zbioru danych można założyć, że istnieje prawdopodobieństwo wystąpienia klastrów o innej charakterystyce nie pasującej do założonych powyżej klas workloadów.

4.5 Klasyfikacja workloadów chmurowych

4.5.1 Opracowanie komponentu klasyfikacji

Identyfikacja klasy funkcjonalnej, do której należy workload ma istotne znaczenie w kontekście efektywnego zarządzania chmurą obliczeniową. Zróżnicowane wymagania na zasoby przez różne klasy funkcjonalne może zostać wykorzystana w procesie alokacji przy uruchamianiu jak i migracji workloadów.

W celu zwiększenia i zróżnicowania rozmiaru zbioru uczącego w celu osiągnięcia większej skuteczności otrzymanego klasyfikatora przeprowadzono dodatkowy etap eksperymentów. Zwiększono liczbę konfiguracji dla każdego z symulatorów poprzez dodanie wartości pośrednich parametrów użytkownika (wcześniej parametry były tak wyskalowane, aby wysycić zasoby, w tych eksperymentach rozszerzono konfiguracje o pośrednie wartości parametrów) oraz uwzględniono również przypadki z mi-

nimalnymi wartościami parametrów tak by zasymulować workloady o bardzo niskim zużyciu zasobów. Następnie dla zdefiniowanych konfiguracji uruchomiono ich symulatory workloadów w taki sam sposób, jak przy pierwszym etapie eksperymentów. Uzyskaną liczbę konfiguracji dla poszczególnych klas funkcjonalnych workloadów przedstawiono w Tabeli 4.3.

Tabela 4.3: Liczba symulacji przeprowadzona na potrzeby zgromadzenia danych wejściowych dla klasyfikatora

Lp.	**Klasa workloadu**	**Liczba konfiguracji**
1.	Big Data	80
2.	Data Caching	120
3.	OLTP	165
4.	Streaming	334
5.	Science	160
6.	Web Serving	211

Wszystkie testy dla każdej z konfiguracji zostały przeprowadzone w rzeczywistym środowisku chmurowym opartym o oprogramowanie OpenStack. W pierwszej kolejności skrypty uruchamiające symulacje i monitorujące działanie symulatorów zostały przygotowane i przetestowane w środowisku testowym TASKcloud. Następnie pełny zbiór symulacji został przeprowadzony w środowisku chmurowym przygotowanym przez partnera projektu KRICO firmę Intel Technology Polska. Celem było niezależne zweryfikowanie uzyskanych wyników.

Na zebranych danych uruchomiono klasyfikator zbudowany na bazie sieci neuronowej. Dane zgromadzone w fazie II symulacji podzielono na 3 zbiory: uczący (30%), walidujący (35%) i testowy (35%). W procesie uczenia wykorzystano jednowarstwową sieć neuronową z 18 neuronami. W pierwszej warstwie jako funkcja aktywacji została użyta funkcji sigmoidalna natomiast w warstwie wyjściowej funkcja softmax. Jako algorytm uczący został wykorzystany algorytm SGD (ang. *Stochastic Gradient Descent*) [13].

Do procesu uczenia wykorzystano ograniczony zbiór gromadzonych w trakcie wcześniejszych eksperymentów metryk. W tym wypadku nie przeprowadzano dodatkowej analizy, a selekcja polegała na wyborze metryk, które byłe możliwe do monitorowania z poziomu hosta indywidualnie dla każdej maszyny gościa. Na etapie eksperymentów przyjęte było założenie, że maszyna gościa jest jedyną maszyną na węźle obliczeniowym. W związku z czym wśród 53 gromadzonych wartości metryk były również takie, których nie można było z powodu ograniczeń zarówno technologicznych jak i dostępnego oprogramowania odczytać dla każdej maszyny z osobna. W momencie kiedy przeprowadzane były eksperymenty, nie istniały bowiem odpowiednie narzędzia rejestrujące pełen zbiór wybranych parametrów per maszyna wirtualna. Przy założeniu, że klasyfikator miałby służyć do określania kategorii workloadu, który działa w rzeczywistym środowisku, gdy na tym samym węźle działa więcej maszyn wirtualnych, konieczne było wprowadzenie opisanego ograniczenia zbioru metryk. Ostateczną listę zaakceptowanych metryk przedstawiono w Tabeli 4.4.

W trakcie prac nad klasyfikatorem przeprowadzone testy wykazały błędną klasyfikację obciążeń, w przypadku gdy nie generowały żadnych zadań (były w stanie *idle*) lub wykorzystywały niewielką ilość zasobów. W związku z tym została dodana dodatkowa klasa Idle. Wprowadzona modyfikacja pozwoliła na zwiększenie jakości klasyfikacji finalnej wersji klasyfikatora. Wyniki działania klasyfikatora zostały przedstawione i przeanalizowane w rozdziale 4.5.3.

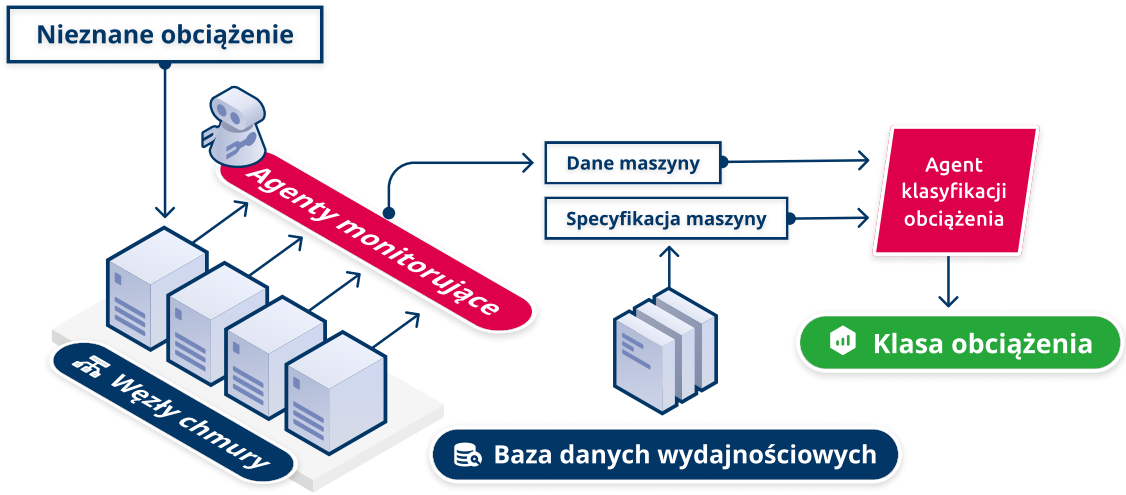
4.5.2 Implementacja klasyfikatora

Podstawowym wymaganiem dla klasyfikatora było rozpoznawanie kategorii workloadu chmurowego na podstawie wartości metryk (opisujących wykorzystanie zasobów) gromadzonych w trakcie jego działania. Jak wspomniano poznanie klasy workloadu jest ważne przede wszystkim z punktu widzenia operatora. Wiedza ta dostarcza operatorowi informacje, która może wspomóc wybór workloadu, który ma zostać przeniesiony z przeciążonego węzła obliczeniowego lub na którym węźle zaalokować nowy workload.

Tabela 4.4: Ostateczna lista metryk gromadzonych z poziomu węzła obliczeniowego na potrzeby przeprowadzenia klasyfikacji workloadow

Lp.	Metryka	Opis
1.	cpu:time	wykorzystany czas procesora (1.0 == 1 logiczny CPU)
2.	cpu:cache:references	liczba dostępów do pamięci L3 [1/s]
3.	cpu:cache:misses	liczba nietrafionych dostępów do pamięci L3 [1/s]
4.	ram:used	wykorzystana pamięć RAM [GB]
5.	disk:bandwidth:read	wykorzystane pasmo odczytu danych z dysku [MiB/s]
6.	disk:bandwidth:write	wykorzystane pasmo zapisu danych z dysku [MiB/s]
7.	disk:operations:read	liczba operacji odczytu z dysku [1/s]
8.	disk:operations:write	liczba operacji zapisu na dysku [1/s]
9.	network:bandwidth:send	pasmo sieciowe wykorzystane przy wysyłaniu danych [MiB/s]
10.	network:bandwidth:receive	pasmo sieciowe wykorzystane do odbioru danych [MiB/s]
11.	network:packets:send	liczba wysłanych pakietów [1/s]
12.	network:packets:receive	liczba odebranych pakietów [1/s]

Na Rysunku 4.7 przedstawiono schemat działania klasyfikatora. Wykorzystanie zasobów przez nieznany workload na węzłach chmurowych jest monitorowane przez agenty monitorujące. Gromadzone są wartości wszystkich metryk przedstawionych w Tabeli 4.4. Dane pomiarowe zapisywane są w bazie danych wydajnościowych, która jednocześnie jest bazą dostarczającą informacje o wykorzystaniu zasobów przez workloady, dla których została zdefiniowana ich klasa w trakcie uruchamiania. Z dwóch źródeł dostarczane są różne dane: specyfikacja maszyny (węzła chmury), na którym uruchomiony jest workload oraz dane o wykorzystaniu zasobów pobierane na bie-



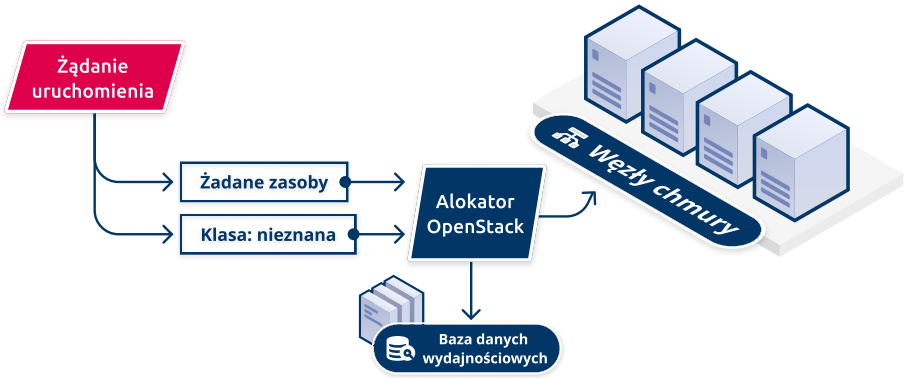
Rysunek 4.7: Schemat procesu klasyfikacji workloadu nieznanego typu (źródło: własne)

żąco od agentów. Na podstawie tych danych klasyfikator wyznacza klasę, do której powinien zostać przypisany oceniany workload.

Schemat uruchomienia workloadu bez określonej klasy przedstawiono na Rysunku 4.8. Wykorzystany klasyfikator w celu przypisania workloadu do klasy funkcjonalnej musi na wejściu otrzymać wartości metryk zarejestrowanych podczas jego działania. W przypadku nowych workloadów, w których użytkownik nie podał klasy funkcjonalnej, nie ma zebranych odpowiednich danych, aby dokonać klasyfikacji. W związku z tym żądanie użytkownika wraz z informacją o braku danych dotyczących klasy obciążenia trafia do systemu zarządzania chmurą. Każde wykonanie workloadu jest monitorowane i zebrane dane są zapisywane w bazie danych wydajnościowych w celu umożliwienia dokonania klasyfikacji np. przy procesie migracji.

Następnie workload jest uruchamiany bez uwzględnienia optymalizacji związanych z wiedzą o przynależności do klasy funkcjonalnej (przy czym proponowana optymalizacja jego uruchamiania została przedstawiona w rozdziale 5.3.1).

Implementacja klasyfikatora zakłada oddzielne instancje klasyfikatora, dla różnych konfiguracji sprzętowych (typów węzłów obliczeniowych). Istnieje jedna domyślna implementacja klasyfikatora zbudowana w oparciu o dane zgromadzone w trakcie eksperymentów opisanych w niniejszej rozprawie. Obiekt klasyfikatora dla



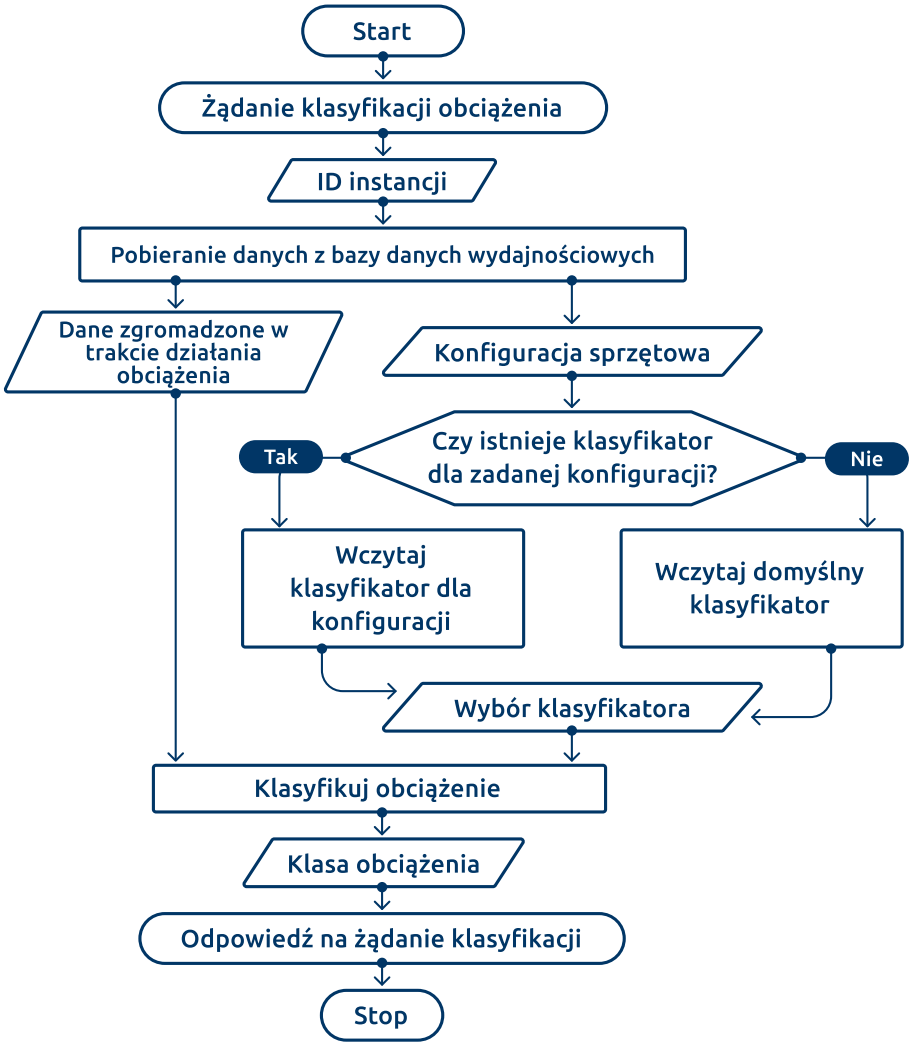
Rysunek 4.8: Schemat uruchomienia nieznanego obciążenia w chmurze (źródło: własne)

konkretnej konfiguracji sprzętowej jest tworzony tylko w momencie kiedy zebrany jest odpowiednio duży zbiór danych uczących dla każdej z klas. Blokowy diagram działania klasyfikatora przedstawiono na Rysunku 4.7.

Żądanie klasyfikacji workloadu odbywa się na podstawie identyfikatora instancji (maszyny wirtualnej) zapisanej w systemie OpenStack. Na podstawie identyfikatora pobierane są dane dotyczące obciążenia generowanego przez workload w trakcie działania, o ile workload był już wcześniej uruchomiony. Dodatkowo pobierane są informacje o konfiguracji sprzętowej, na której był wykonywany. W następnym kroku dla pobranej konfiguracji dobierany jest klasyfikator dla konkretnej konfiguracji sprzętowej. Jeżeli zadany klasyfikator nie istnieje pobierany jest wówczas domyślny klasyfikator. Kolejnym krokiem jest dokonanie klasyfikacji na podstawie danych zgromadzonych z monitoringu działania workloadu. W odpowiedzi na żądanie zwracana jest sugerowana klasa workloadu.

4.5.3 Ocena działania klasyfikatora

Po przygotowaniu prototypu klasyfikatora zostały przeprowadzone testy weryfikujące jego skuteczność. W trakcie testów wykorzystano dane zgromadzone w trakcie eksperymentów. Proces testowania obejmował wielokrotne uruchomienie procesu uczenia, weryfikacji i testów. Ocena klasyfikacji następowała na podstawie metryki

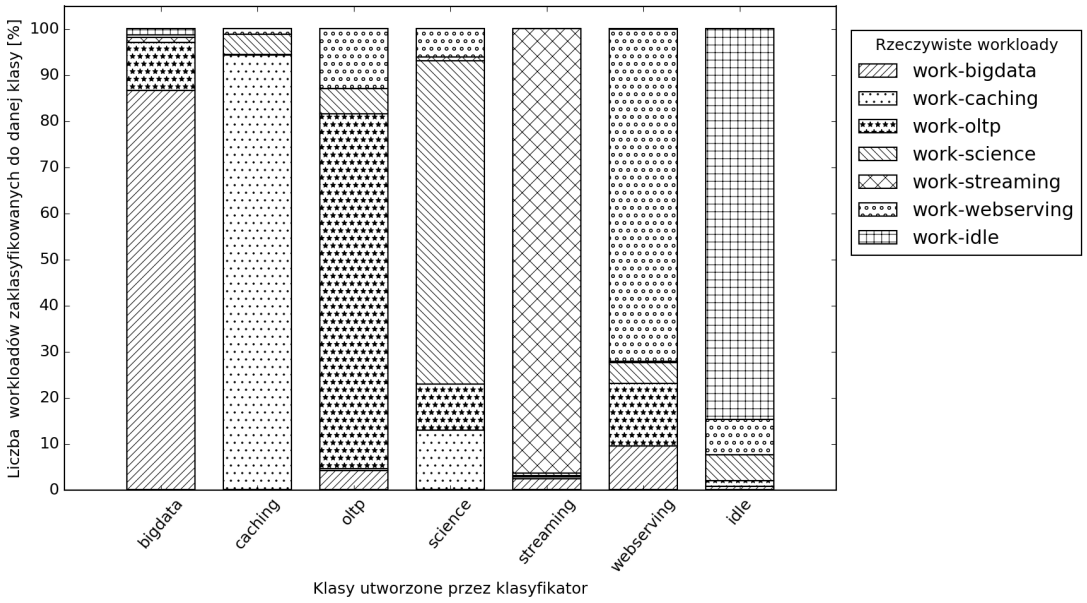


Rysunek 4.9: Schemat obsługi żądania klasyfikacji nieznanego obciążenia (źródło: własne)

f-measure.

Na Rysunku 4.10 przedstawiono diagram prezentujący do jakich kategorii funkcjonalnych klasyfikator przydzielił obciążenia ze zbioru testowego. Średnia wartość metryki *f1-score* wyniosła 83,2% a odchylenie standardowe 3,3%. Przedstawione wartości są wartościami uśrednionymi z próby 500 testów.

Zaprezentowane wyniki działania komponentu pokazują istnienie relacji pomiędzy klasami funkcjonalnymi a klastrami wyznaczonymi za pomocą metod uczenia maszynowego z wykorzystaniem informacji o zasobach chmurowych wykorzystywa-



Rysunek 4.10: Wyniki działania klasyfikatora na testowym zbiorze danych (źródło: własne)

nych przez workloady. Zaproponowany komponent nie jest rozwiązaniem idealnym, tzn. mającym 100% skuteczności. Jest to spowodowane dużym zróżnicowaniem charakterystyki zużywanych zasobów przez workloady w zależności od sposobu ich wykorzystania. Niemniej wykorzystanie klasyfikacji z ponad 83% poprawnością pozwoli określić jakiej klasy workload został uruchomiony w chmurze. Co ważne zaproponowany mechanizm nie wymaga dostępu do maszyny użytkownika chmury obliczeniowej.

Potencjalnym kierunkiem rozwoju i dalszych badań nad skutecznością klasyfikatora byłoby rozszerzenie zbioru danych uczących. Rozszerzenie powinno zostać zrealizowane dwutorowo. Z jednej strony należy zwiększyć liczbę symulatorów opartych o inne niż dotychczas wykorzystane aplikacje. Z drugiej strony należy pozyskać dane z wykonania rzeczywistych workloadów w środowisku chmury obliczeniowej przy czym istotne jest, by dane były wzbogacone o klasę workloadu oraz wartości parametrów użytkownika. Dopiero dla danych w takiej postaci będzie możliwe rozszerzenie zbioru uczącego klasyfikatora. Dodatkowo istnieje możliwość ulepsze-

nia klasyfikacji za pomocą analizy charakterystyki wartości dla zebranych metryk w czasie zamiast wykorzystania wartości średnich z całego czasu wykonania workloadu. Funkcjonowanie chmury TASKcloud umożliwia zbieranie tego typu danych na bieżąco i dokumentowanie zaproponowanej metody klasyfikacji po odpowiednim okresie jej funkcjonowania, gdy liczba różnorodnych żądań użytkowników zbliży się do liczby analizowanych workloadów podczas symulacji.

Informacja o klasie workloadu może zostać również wykorzystana do badania wzajemnego wpływu różnych klas workloadów kiedy są one uruchomione na jednym węźle obliczeniowym. Zdefiniowanie metryki opisującej wzajemny wpływ poszczególnych klas workloadów na siebie oraz uwzględnienie tej wiedzy na etapie alokacji może znacząco wpłynąć na efektywność wykonywania workloadów. Wiedzę tę można wykorzystać w mechanizmach zarządzania chmurą obliczeniową zarówno przy uruchamianiu jak i przy migracji workloadów. Proponowane praktyczne wykorzystanie klasyfikacji funkcjonalnej przedstawiono w następnym rozdziale rozprawy doktorskiej.

Rozdział 5

Algorytmy alokacji zasobów chmurowych

Efektywne zarządzanie zasobami chmurowymi wymaga odpowiednich algorytmów alokacji. Ich implementacje u różnych dostawców mogą różnić się diametralnie jednak kluczowy w nich wszystkich jest odpowiedni dobór kryteriów alokacji zasobów. Zaproponowano trzy popularne podejścia do alokacji workloadów na zasoby chmury obliczeniowej oraz implementację algorytmów alokacji w rozwiązaniach produkcyjnych. Bazując na rezultatach prac związanych z wyborem klas funkcjonalnych workloadów (rozdział 4.4) oraz możliwościami klasyfikacji workloadów (rozdział 4.5.3) zdefiniowano komplementarność workloadów oraz zaproponowano modyfikację algorytmu alokacji uwzględniającą tę własność.

5.1 Model alokacji workloadów na zasobach chmury obliczeniowej

Proces alokowania workloadów na zasoby obliczeniowe wynika z ogólnego schematu działania chmury, który przedstawiono na Rysunku 3.2. Żądania wykonania usługi wysyłane przez klienta, są obsługiwane przez użytkownika chmury a następnie mapowane na żądanie uruchomienia workloadu i przyznania mu zasobów wirtualnych.

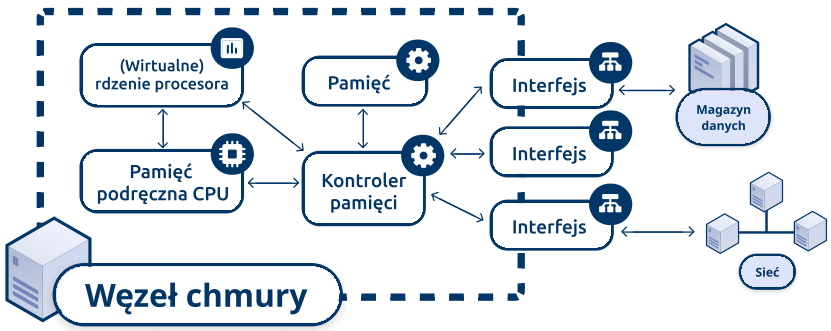
Następnie po stronie dostawcy chmury komponent dyspozytora (ang. *scheduler*) realizuje proces alokacji workloadu i przydziela mu konkretne zasoby fizyczne. Odwzorowanie żądań klientów najpierw na zasoby wirtualne, a następnie na fizyczne zaprezentowano na Rysunku 3.3. W niniejszym rozdziale obszarem zainteresowania będzie drugie odwzorowanie a konkretnie proces alokacji zasobów wirtualnych na zasoby fizyczne.

Zasoby wirtualne, będące elementem workloadu (patrz Rysunek 4.2), są opisem wymagań i specyfikacją zasobów fizycznych, w tym również rozmiaru tych zasobów, które mają zostać przydzielone do workloadu. Możemy je podzielić na dwa typy: ilościowe i jakościowe. Zasoby ilościowe opisują ile zasobów powinno zostać przydzielone do realizacji workloadu. Przykładami takich zasobów jest liczba wirtualnych rdzeni procesora, rozmiar pamięci operacyjnej, rozmiar przestrzeni dyskowej. Z drugiej strony zasoby jakościowe opisują jakimi cechami powinny charakteryzować się przydzielone zasoby. Mogą to być cechy określające konkretną wydajność, np. przepustowość sieci większa niż 100 Mbit/s, wydajność magazynu danych większa niż 200 operacji wejścia/wyjścia na sekundę, jak również dodatkowe wymagania wymagające specyficznego zasobu, np. procesora ze wsparciem szyfrowania lub dostępu do akceleratora graficznego.

Rysunek 5.1 prezentuje rozważany model węzła obliczeniowego. Zaznaczono na nim podstawowe elementy architektury mające swoje odwzorowanie w zasobach ilościowych (procesor, pamięć, magazyn danych) jak i jakościowych (interfejsy sieciowe czy interfejsy do magazynu danych, które muszą spełniać określone warunki np. dotyczące przepustowości).

W modelu tym zaznaczono również zasoby, które nie mają bezpośredniej reprezentacji w zasobach wirtualnych wyspecyfikowanych w żądaniu uruchomienia workloadu, jednak mogące mieć znaczący wpływ na efektywność jego działania. Przykładami takich zasobów jest pamięć cache procesora czy kontroler pamięci [86], których nie możemy wyizolować i zarezerwować na wyłączność dla konkretnego workloadu.

Zbiór workloadów uruchomionych w chmurze obliczeniowej reprezentuje zbiór $\mathrm{W}=\{\mathrm{w}_1, w_2, ..., w_{|W|}\}$. W przyjętym modelu oznaczmy przez $\mathrm{N}=\{\mathrm{n}_1, n_2, ..., n_{|N|}\}$ zbiór



Rysunek 5.1: Rozważany model węzła chmury obliczeniowej (źródło: własne)

dostępnych węzłów obliczeniowych w chmurze. W rozdziale 4 zaproponowano 7 klas funkcjonalnych workloadów. Zbiór tych klas oznaczamy jako $C = \{1, 2, 3, 4, 5, 6, 7\}$, natomiast klasę funkcjonalną, do której należy workload w określamy jako $c(w) \in C$.

Alokację workloadów, czyli przypisanie workloadów do węzłów obliczeniowych, możemy zdefiniować jako odwzorowanie $a(w)$ (ang. *allocation*) i zapisujemy zgodnie z równaniem 5.1.

$$n = a(w), \text{ gdzie } n \in N, w \in W \tag{5.1}$$

Możemy zdefiniować też odwzorowanie odwrotne $a^{-1}(n)$, które opisuje jakie workloady są zaalokowane na węźle obliczeniowym n zgodnie ze wzorem w równaniu 5.2.

$$w = a^{-1}(n), \text{ gdzie } w \in W, n \in N \tag{5.2}$$

Oznaczmy zasoby wirtualne dostępne w chmurze obliczeniowej przez R, zaś zasoby wirtualne dostępne na węźle n jako $rn(n)$. Wszystkie zasoby wirtualne dostępne w chmurze możemy wtedy opisać wzorem 5.3.

$$R = \sum_{n \in N} rn(n) \tag{5.3}$$

Pewien podzbiór zasobów wirtualnych R, jest zdefiniowany w żądaniu użytkownika chmury jako zasoby wirtualne, które mają zostać przydzielone dla workloadu w, co oznaczamy jako $rw(w)$. Workload w chmurze obliczeniowej w modelu IaaS

reprezentowany jest przez maszynę wirtualną. Specyfikacja workloadu posiada wymagania na zasoby wirtualne niezbędne do jego wykonania, które są przydzielane maszynie wirtualnej w trakcie uruchamiania. Klasy funkcjonalne workloadów zdefiniowane w rozdziale 4 zostały opracowane na podstawie wykorzystywanych zasobów, podczas rzeczywistego ich zużycia w trakcie działania (patrz Rysunek 4.2). Natomiast zasoby wirtualne rozważane w niniejszym rozdziale stanowią w przypadku specyfikacji workloadu wymagane zasoby (wirtualne), a w przypadku wykonania workloadu (uruchomienia maszyny) to zasoby przydzielone maszynie na węźle obliczeniowym.

Zasoby wirtualne reprezentowane jako R mogą zostać zdefiniowane, jak podano powyżej, w zależności od konkretnego środowiska chmurowego. W procesach alokacji w środowiskach chmurowych brane pod uwagę są wszystkie wyspecyfikowane zasoby wirtualne. Dla modelowanego środowiska przyjmujemy w dalszych rozważaniach rozpatrywanie zbioru R jako zbioru z pojedynczym zasobem wirtualnym tj. liczba wirtualnych rdzeni procesora. Przyjęte uproszczenie pozwala w klarowny sposób przedstawić w przykładach mechanizmy alokacji, jednak zaproponowany model jak i rozważane algorytmy w łatwy sposób można rozszerzyć o kolejne rodzaje zasób w zależności od wymagań w konkretnym środowisku chmury obliczeniowej. W tym celu należy zdefiniować zbiór R jako wektor zasobów wirtualnych oraz zdefiniować sumowanie poszczególnych zasobów (elementów wektora) jako sumę poszczególnych zasobów.

Odwzorowanie $rn(n)$, będzie oznaczało liczbę wszystkich wirtualnych rdzeni dostępnych na węźle obliczeniowym, natomiast dla workloadu w, $rw(w)$ będzie oznaczało liczbę wirtualnych rdzeni, które mają zostać przydzielone dla niego na węźle obliczeniowym. Zdefiniujmy również $ru(n)$, które będzie oznaczało liczbę wirtualnych rdzeni przydzielonych (zajętych) przez workloady na węźle n. Wówczas możemy policzyć jako sumę liczbę zasobów wirtualnych wykorzystywanych przez workloady na węźle n, zgodnie z równaniem 5.4.

$$ru(n) = \sum_{w \in a^{-1}(n)} rw(w) \tag{5.4}$$

Przedstawiony model oraz wprowadzone odwzorowania zostaną wykorzystane do zaprezentowania ulepszonego modelu alokacji w dalszej części rozdziału.

5.2 Implementacja mechanizmu alokacji workloadów w oprogramowaniu OpenStack

Jednym z najważniejszych elementów zarządzania chmurą obliczeniową jest alokacja obciążeń na węzłach chmury. Heterogeniczność zasobów, różnorodność, zmienność i nieprzewidywalność obciążeń, a także różnorodność wymagań użytkowników chmury sprawiają, że najbardziej przydatne są metody uniwersalne, proste i efektywne. Można je formułować na podstawie ogólnych modeli, które możemy stosować na różnych poziomach zarządzania chmurą [82].

Alokacja obciążeń, jak wspomniano w rozdziale 3.4, może być realizowana według różnych kryteriów i za pomocą różnych algorytmów. Coraz częściej można spotkać algorytmy bardzo złożone i wykorzystujące nowe rozwiązania w dziedzinie IT. Przykładami są rozwijane od lat algorytmy heurystyczne [120] czy wykorzystanie algorytmów mrówkowych [123, 57].

W praktyce najczęściej jednak wykorzystywane są proste rozwiązania, które są łatwe w modyfikacji i elastyczne. Najczęściej wykorzystywane są algorytmy równoważące obciążenie węzłów chmurowych (LB (ang. load balancing)) lub agregujące workloady zmniejszając liczbę wykorzystywanych węzłów chmurowych (CONS (ang. consolidation)). Dodatkowo do wspomnianych algorytmów w zależności od specyfikacji użytkownika możliwe jest dodanie mechanizmu odporności na awarie (FT (ang. fault tolerance)).

Zaprezentowane algorytmy do zarządzania alokacją workloadów wykorzystywane są w różnych oprogramowaniach do zarządzania chmurą obliczeniową. Wybór algorytmu oraz kryterium alokacji zależy od dostawcy chmury obliczeniowej. W jego gestii jest zachowanie właściwej proporcji pomiędzy kosztami (w konsekwencji przychodem pozyskiwanym z realizacji usługi) a wydajnością zasobów i efektywnością

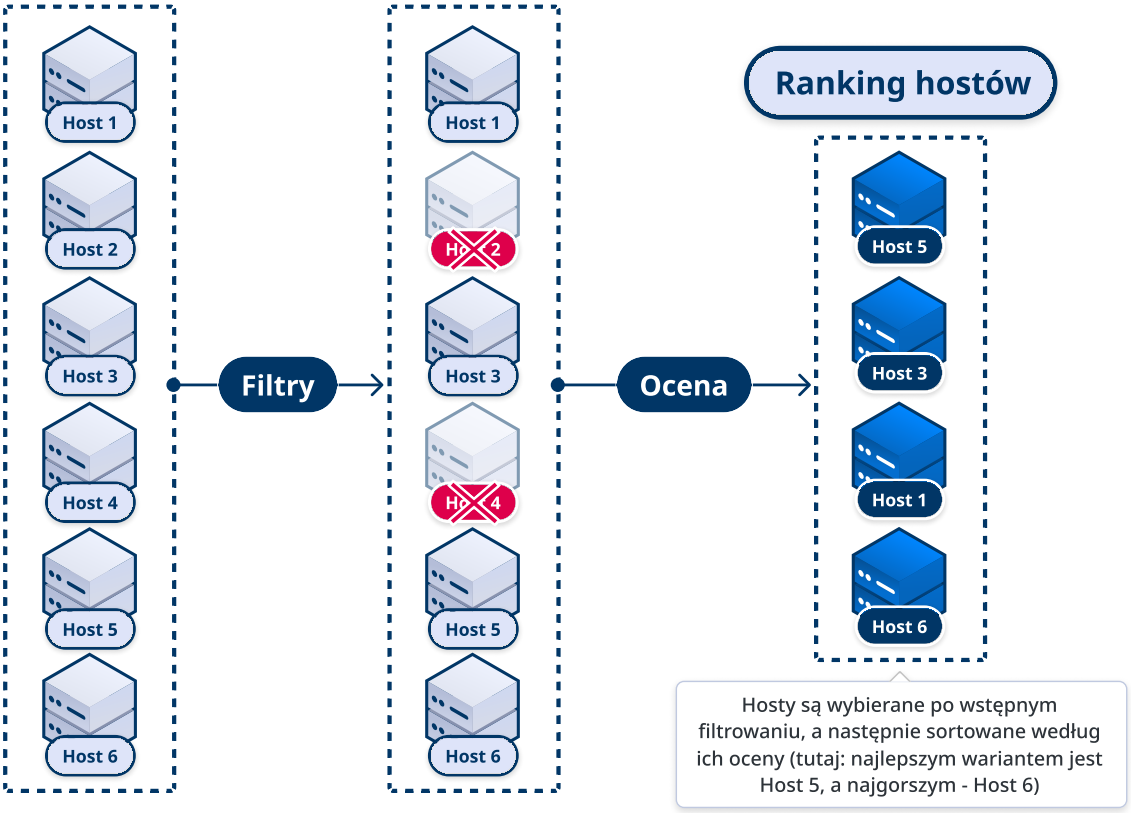
wykonania workloadów. Dostawca chmury może w trakcie pracy środowiska chmurowego przekonfigurować lub zmienić algorytm alokacji w związku z czym istnieje realna potrzeba dostarczenia prostej metody jego modyfikacji bez dokonywania przerw w funkcjonowaniu chmury.

Ta idea łatwej rekonfiguracji algorytmu alokacji została zastosowana przez twórców oprogramowania OpenStack przy projektowaniu komponentu dyspozytora - specjalnego komponentu chmury obliczeniowej. Konfiguracja dyspozytora, który jest elementem pakietu OpenStack Nova odpowiedzialnego za zarządzanie maszynami wirtualnymi, możliwa jest za pomocą tekstowych plików konfiguracyjnych. Zastosowanie zmian wymaga restartu wyłącznie modułu dyspozytora i nie ma wpływu na działanie całości chmury obliczeniowej.

Dystrybutor w oprogramowaniu OpenStack został zrealizowany w bardzo prosty i jednocześnie bardzo elastyczny pod kątem modyfikacji sposób. Mechanizm alokacji został podzielony na dwa etapy: filtrowania (ang. *filtering*) i oceny (ang. *weighting*). Ideowy schemat działania dystrybutora OpenStack przedstawia Rysunek 5.2.

W etapie filtrowania dyspozytor odbiera przesłane od użytkownika żądanie uruchomienia maszyny wirtualnej, odczytuje zdefiniowane w chmurze węzły obliczeniowe i rozpoczyna etap ich filtrowania pod kątem możliwości uruchomienia na nich maszyny wirtualnej zgodnie ze specyfikacją przesłaną w żądaniu. Filtrowanie odbywa się sekwencyjnie na zbiorze funkcji filtrujących F (patrz Tabela 5.1). Algorytm wywołuje kolejne funkcje filtrujące f_i zgodnie z kolejnością zdefiniowaną przez dostawcę w konfiguracji dyspozytora. Funkcja filtrująca to metoda z zaimplementowanym specyficznym kryterium weryfikującym możliwość uruchomienia maszyny wirtualnej na danym węźle. W Tabeli 5.1 przedstawiono przykładowe filtry dostarczane z systemem OpenStack.

Metoda działania filtrów zostanie przedstawiona na przykładzie filtra f_4 *ImagePropertiesFilter*. Zadaniem filtru jest weryfikacja czy węzeł obliczeniowy spełnia wymagania zdefiniowane w specyfikacji obrazu systemu operacyjnego, z którego ma być uruchomiona maszyna wirtualna zgodnie z żądaniem uruchomienia workloadu przez użytkownika. W momencie dodawania obrazu systemu operacyjnego do sys-



Rysunek 5.2: Ideowy schemat działania dystrybutora w oprogramowaniu OpenStack (źródło: własne)

temu OpenStack możliwe jest zdefiniowanie wymagań tego systemu wobec węzła. Przykładowymi parametrami, które opisują obraz są: architektura procesora, na której możliwe jest uruchomienie systemu operacyjnego (np. arm lub x86_64), minimalny rozmiar dysku na system operacyjny (np. 8 GB), czy wymaganie istnienie na węźle konkretnego urządzenia, (np. karty GPU lub zwirtualizowanego urządzenia pseudolosowego). Z drugiej strony każdy węzeł posiada swoje parametry, które go opisują. Działanie filtra polega na weryfikacji czy wszystkie wymagania określone w obrazie są spełniane przez kolejne węzły obliczeniowe chmury. Filtr odrzuci wszystkie te węzły obliczeniowe, które nie będą spełniały wszystkich wymagań. Pseudokod filtru realizującego filtrowanie węzłów pod kątem spełniania wymagań obrazu systemu operacyjnego, z którego ma zostać uruchomiona maszyna wirtualna prezentuje Listing 1.

Listing 1 Pseudokod funkcji filtrującej ImagePropertiesFilter weryfikującej zgodność węzła obliczeniowego z wymaganiami obrazu systemu operacyjnego, z którego ma zostać uruchomiony workload

```
1: N ← zbiór węzłów w chmurze obliczeniowej
2: vm ← specyfikacja maszyny wirtualnej reprezentującej wymagania workloadu w
3: vm.imageProperties ← parametry opisujące wymagania obrazu systemu operacyjnego, z którego ma zostać uruchomiona maszyna wirtualna vm
4: n.properties ← parametry opisujące węzeł obliczeniowy n
5: function filter(N, vm):
6:     matchingNodes = array() //tablica z listą węzłów spełniających wymagania obrazu
7:     for all n in N do
8:         for all (key, value) in vm.imageProperties  do
9:             if value not equal n.properties[key] then
10:                break //przerwanie sprawdzania dla węzła n w przypadku wystąpienia pierwszej niezgodności
11:            end if
12:            matchingNodes = matchingNodes + n
13:        end for
14:    end for
15: return  matchingNodes
```

Tabela 5.1: Wykaz przykładowych filtrów zaimplementowanych i możliwych do wykorzystania w dyspozytorze systemu OpenStack

F	Nazwa filtra	Kryterium filtrowania
f_1	AggregateMultiTenancyIsolation	odrzuca węzły niedostępne dla danego użytkownika
f_2	AvailabilityZoneFilter	filtruje węzły w wybranej strefie dostępności
f_3	ComputeFilter	filtruje aktywne węzły
f_4	ImagePropertiesFilter	weryfikuje zgodność węzła obliczeniowego ze specyfikacją hosta
f_5	IoOpsFilter	weryfikuje węzły pod kątem wydajności dysku
f_6	IsolatedHostsFilter	weryfikuje czy workload ma zostać uruchomiony na izolowanym hoście
f_7	JsonFilter	meta filtr, pozwalający dostawcy ustawić własne kryterium
f_8	PciPassthroughFilter	weryfikuje dostępność urządzeń PCI na węźle
f_9	ServerGroupAntiAffinityFilter	weryfikuje czy węzeł należy do określonej grupy serwerów

Funkcja filtrująca przyjmuje na wejściu zbiór węzłów oraz specyfikację maszyny wirtualnej do uruchomienia na rzecz realizacji workloadu. Następnie weryfikuje, na których węzłach obliczeniowych z otrzymanych w zbiorze wejściowym możliwe jest uruchomienie maszyny wirtualnej sprawdzając zdefiniowane w funkcji kryterium (linie 7-14). Funkcja tworzy nowy zbiór węzłów będący podzbiorem węzłów obliczeniowych otrzymanych na wejściu, które spełniają wszystkie wymagania obrazu systemu operacyjnego (linia 8-13). Dyspozytor po otrzymaniu zbioru węzłów z filtra przekazuje go do kolejnego filtra (zgodnie z kolejnością zdefiniowaną w konfiguracji.

Danymi wejściowymi dla kolejnego filtra jest zbiór węzłów obliczeniowych wybrany przez filtr poprzedni. W wyniku działania łańcucha filtrów powstaje zbiór węzłów obliczeniowych składający się włącznie z węzłów, które spełniają wszystkie wymagania wszystkich zdefiniowanych filtrów. W ramach dostarczonego pakietu dyspozytora zaimplementowanych i gotowych do użycia jest ponad 20 filtrów, które mogą być dodane w konfiguracji dyspozytora przez dostawcę (część z nich została przedstawiona wcześniej w Tabeli 5.1).

Następnie dyspozytor przechodzi do kolejnego etapu oceny węzłów chmurowych. Etap oceny polega na wyliczeniu wartości punktowej dla każdego węzła i na podstawie utworzonego rankingu węzłów wybiera węzeł do uruchomienia workloadu. Podobnie jak dla etapu filtrowania zdefiniowane są w konfiguracji dyspozytora filtry, tak dla etapu oceny zdefiniowany jest zbiór funkcji oceniających. Zbiór funkcji oceniających oznaczamy przez S. Przykładowe funkcje oceniające dostarczane z oprogramowaniem OpenStack przedstawiono w Tabeli 5.2

Tabela 5.2: Wykaz przykładowych funkcji oceniających zaimplementowanych i możliwych do wykorzystania w dyspozytorze systemu OpenStack

S	Nazwa funkcji oceniającej	Kryterium oceny
s_1	CPUWeigher	dostępność wirtualnych rdzeni procesora
s_2	DiskWeigher	dostępność dysku
s_3	PCIWeigher	dostępność urządzeń PCI
s_4	RAMWeigher	dostępność pamięci RAM

Każda z funkcji oceniających ma zdefiniowaną właściwość węzła, według której dokonuje oceny w postaci punktów. Algorytm przyznawania punktów nie jest jednolity dla różnych właściwości. Wspólną cechą dla każdej z funkcji jest rezultat jej działania, każda z nich zwraca do dyspozytora wartość punktową z pewnego zakresu przyznaną dla węzła chmurowego. W zależności od funkcji oceniającej możliwe są różne zakresy zwracanych ocen. Dla funkcji s_1 (z Tabeli 5.2) wynikiem działania jest liczba dostępnych wirtualnych rdzeni procesora na węźle w związku z czym ocena

będzie liczbą naturalną z ograniczeniem z góry do maksymalnej dostępnej liczby wirtualnych rdzeni. W przypadku TASKcloud maksymalną wartością będzie dla tej funkcji oceny wartość 48. Z kolei dla funkcji s_4 oceniającej dostępność pamięci RAM oceną punktową jest rozmiar pamięci dostępnej na węźle w MB. Dla TASKcloud maksymalną wartością będzie w tym wypadku 122 880 MB (120 GB). Jak widać na powyższych przykładach możliwe jest powstanie dużych dysproporcji w ocenie - ocena punktująca pamięć RAM jest 4 rzędy wielkości większa od oceny dostępnych wirtualnych rdzeni procesora. Aby zachować proporcję przy jednoczesnym zachowaniu możliwości zwiększania lub zmniejszania wagi poszczególnych ocen zastosowano mechanizm normalizacji i współczynników oceny.

Mechanizm normalizacji polega na wybraniu wartości maksymalnej dla danej funkcji oceniającej ze zbioru ocen wszystkich węzłów obliczeniowych $max(s_i(N))$. Następnie ocena uzyskana przez węzeł dla danej funkcji oceniającej s_i jest dzielona przez maksymalną ocenę, którą dla tej funkcji uzyskał węzeł obliczeniowy. Dzięki takiemu zabiegowi wartości dla każdej z ocen będą z przedziału $[0; 1]$. Normalizację $\overline{s_i}$ przedstawia równanie 5.5.

$$\overline{s_i(n)} = \frac{s_i(n)}{max(s_i(N))} \in [0; 1] \tag{5.5}$$

W celu zachowania elastyczności konfiguracji dla dostawcy chmury w oprogramowaniu Openstack zastosowano mechanizm wag ($scoreWeight(s_i)$) dla poszczególnych funkcji oceniających. Mechanizm ten ma za zadanie umożliwić kontrolowanie wagi ocenianej cechy w ogólnej ocenie każdego węzła $ns(n)$ jednocześnie wpływając na pozycję węzła w ostatecznym rankingu. Możliwości modyfikacji sposobu alokacji workloadów tj. zmiany algorytmu LB na CONS za pomocą mechanizmu wag został przedstawiony w dalszej części rozdziału.

Wzór na wyliczenie ogólnej oceny ns węzła n przedstawiono we wzorze 5.6.

$$ns(n) = \sum_{s_i \in S} \overline{s_i(n)} * scoreWeight(s_i) \tag{5.6}$$

Po wyliczeniu ocen dla wszystkich węzłów tworzony jest ranking węzłów obli-

czeniowych na podstawie ogólnej oceny węzłów ns i jest on sortowany w kolejności malejącej. Dyspozytor wybiera do alokacji maszyny wirtualnej workloadu węzeł obliczeniowy z najwyższą oceną. Jeśli jest więcej niż jeden węzeł z taką samą oceną to wykorzystywana jest funkcja losowo wybierająca z tych węzłów jeden, na który ma zostać zaalokowana maszyna wirtualna. Pseudokod procesu alokacji wykonywany przez dyspozytora przedstawia Listing 2

Zastosowany mechanizm jest mechanizmem bardzo elastycznym i prostym w użyciu. Poniżej przedstawiono wykorzystanie tego mechanizmu dla opisanych przykładów algorytmów LB, CONS oraz mechanizmem FT. Dla uproszczenia rozważań pod uwagę wzięto jedno kryterium optymalizacji tj. liczba wykorzystanych na węźle obliczeniowym wirtualnych rdzeni procesora za pomocą funkcji s_1 (patrz Tabela 5.2). Wybór kryterium podyktowany był prostym algorytmem oceny, który w łatwy sposób można było przedstawić w postaci tabel.

Przyjmijmy następujące środowisko obliczeniowe: wszystkie węzły są homogeniczne, środowisko składa się z 3 węzłów obliczeniowych każdy o liczbie wirtualnych rdzeni procesora równej 48. Każdy z węzłów obliczeniowych działa prawidłowo i możliwa jest alokacja na nim nowych maszyn wirtualnych. Aktualnie w środowisku jest uruchomionych 5 workloadów w_i reprezentowanych odpowiednio przez maszyny wirtualne vm_i, $i = 1, 2, 3, 4, 5$. Wykorzystanie zasobów wirtualnych przez maszyny wirtualne (VMs) reprezentujące workloady na węzłach obliczeniowych prezentuje Tabela 5.3.

Tabela 5.3: Aktualne wykorzystanie wirtualnych rdzeni procesora na przykładowym trzy węzłowym środowisku

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5
n_1	4	8			
n_2					8
n_3			16	2	

Listing 2 Pseudokod procesu alokacji wykonywany przez dyspozytora

```
1: N ← zbiór węzłów w chmurze obliczeniowej
2: vm ← specyfikacja maszyny wirtualnej realizującej workload w
3: F ← zbiór funkcji filtrujących
4: S ← zbiór funkcji oceniających
5: scoreWeight ← funkcja zwracająca wagę funkcji oceniającej
6: filteredNodes ← lista węzłów, na których możliwa jest alokacja maszyny wirtualnej
7: filteredNodes = N
8: for all f in F do
9:    filteredNodes=f.filter(filteredNodes,vm)
10: end for
11: nodesScore = array() //tymczasowa lista z oceną węzłów
12: for all s in S do
13:    tempNodesScore = array() //tymczasowa lista wartości ocen węzłów dla funkcji oceniającej przed normalizacją
14:    for all n in filteredNodes do
15:       tempNodesScore[n]=s(n)
16:    end for
17:    for all n in filteredNodes do
18:       nodesScore[n] = nodesScore[n] + (scoreWeight(s) * tempNodesScore[n] / max(tempNodesScore))
19:    end for
20: end for
21: selectedNode = null
22: n = max(nodesScore)
23: if length(n) > 1 then
24:    selectedNode = random(n)
25: else if length(n) equals 1 then
26:    selectedNode = n[0]
27: end if
28: return selectedNode
```

Należy zaalokować dwa nowe workloady reprezentowane przez maszyny wirtualne (vm_6 i vm_7), które będą wykorzystywały po 8 rdzeni każda. Wykorzystujemy wyłącznie funkcję oceniającą zasób wirtualnych rdzeni procesora s_1. Waga funkcji oceniającej $scoreWeight(s_1)$ jest domyślnie ustawiona na wartość 1.

Rozpatrzmy przypadek równomiernego rozłożenie zasobów bez mechanizmu FT. Alokacja maszyny wirtualnej vm_6 przebiega zgodnie z Listingiem 2. W Tabeli 5.4 zaprezentowano wyliczenia dla kolejnych etapów oceny węzłów.

Tabela 5.4: Dodatkowa alokacja maszyny vm_6 algorytmem LB z wykorzystaniem funkcji oceniającej s_1 w środowisku przedstawionym w Tabeli 5.3

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5	$ru(n)$	$s_1(n)$	$ns(n)$
n_1	4	8				**12**	36	0,9
n_2					8	**8**	40	1
n_3			16	2		**18**	30	0,75

W kolumnie $ru(n)$ przedstawiono liczbę wykorzystywanych (przez działające workloady) rdzeni wirtualnych procesora na węźle obliczeniowym (patrz wzór 5.4). Kolumna $s_1(n)$ prezentuje wynik punktowy dla węzła n po zastosowaniu funkcji oceniającej s_1, wynik punktowy po znormalizowaniu i przemnożeniu przez wagę $scoreWeight(s_1)$ przedstawia kolumna $ns(n)$ (wyliczony zgodnie ze wzorem 5.6). Przy alokacji maszyny wirtualnej vm_6 (reprezentującą workload w_6) dyspozytor wybiera węzeł n_2 ponieważ miał on najwyższy wynik punktowy tj. $n_2 = a(w_6)$.

Alokacja maszyny wirtualnej vm_7 przebiega w analogiczny sposób przy czym uwzględniona jest już maszyna wirtualna vm_6 na węźle n_2. Wyliczenia dla kolejnych etapów oceny węzłów przy procesie alokacji maszyny vm_7 przedstawia Tabela 5.5.

Przy alokacji maszyny wirtualnej vm_7 (reprezentującą workload w_7) dyspozytor wybiera węzeł n_1 (tj. $n_1 = a(w_7)$), ponieważ ma on najwyższy wynik punktowy (jest pierwszy w rankingu). Porównując obciążenie węzłów przed alokacją nowych maszyn $ru(N) = [12, 8, 18]$ z obciążeniem po alokacji $ru(N) = [12, 16, 18]$ widzimy, że

Tabela 5.5: Alokacja maszyny vm_7 algorytmem LB z wykorzystaniem funkcji oceniającej s_1 w środowisku przedstawionym w Tabeli 5.3 z dodanym dodatkowym workloadem na węźle n_2

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5	vm_6	$ru(n)$	$s_1(n)$	$ns(n)$
n_1	4	8					**12**	36	1
n_2					8	8	**16**	32	0,89
n_3			16	2			**18**	30	0,83

zostało ono zrównoważone. Wynika z tego fakt, że dla domyślnych wartości funkcji oceniających $scoreWeight(s_i)$ ustawionych na wartość 1, OpenStack alokuje maszyny wirtualne z zachowaniem równomiernego rozłożenia obciążenia.

W celu zmiany sposobu alokacji maszyn wirtualnych na konsolidację w oprogramowaniu OpenStack jedyną zmianą, której musi dokonać dostawca chmury jest zmiana wartości funkcji oceniających $scoreWeight(s_i)$ na wartość -1 dla zasobów, które należy konsolidować. Poniżej przedstawiono analizę problemu alokacji maszyn vm_6 i vm_7 w rozważanym środowisku przedstawionym w Tabeli 5.3. Jedyną zmianą jest zmiana wartości wskaźnika $scoreWeight(s_1)$ na wartość -1. Proces alokacji maszyny wirtualnej vm_6 z wartościami wyliczeń dla kolejnych etapów oceny węzłów przedstawia Tabela 5.6.

Tabela 5.6: Alokacja maszyny wirtualnej vm_6 z wykorzystaniem algorytmu CONS i funkcji oceniającej s_1 dla środowiska przedstawionego w Tabeli 5.3

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5	$ru(n)$	$s_1(n)$	$ns(n)$
n_1	4	8				**12**	36	-0,9
n_2					8	**8**	40	-1
n_3			16	2		**18**	30	-0,75

Maszyna wirtualna vm_6 (reprezentująca workload w_6) zostanie przez dyspozytora przypisana do węzła n_3, $n_3 = a(w_6)$. Proces alokacji maszyny wirtualnej vm_7 z wartościami wyliczeń dla kolejnych etapów oceny węzłów obliczeniowych przedstawia Tabela 5.7.

Tabela 5.7: Alokacja maszyny wirtualnej vm_7 z wykorzystaniem algorytmu CONS i funkcji oceniającej s_1

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5	vm_6	$ru(n)$	$s_1(n)$	$ns(n)$
n_1	4	8					**12**	36	-0,9
n_2					8		**8**	40	-1
n_3			16	2		8	**26**	22	-0,55

Przy alokacji maszyny wirtualnej vm_7 (reprezentującej workload w_7) dyspozytor wybiera ponownie węzeł n_1 ($n_1 = a(w_7)$), ponieważ ma on najwyższy wynik punktowy. Porównując obciążenie węzłów przed alokacją nowych maszyn $ru(N) = [12, 8, 18]$ z obciążeniem po alokacji $ru(N) = [12, 8, 26]$ widzimy, że po zmianie wartości wskaźnika $scoreWeight(s_1)$ na wartość -1 algorytm konsoliduje maszyny wirtualne i obydwie maszyny zostały przypisane do najbardziej obciążonego węzła obliczeniowego. Przedstawione algorytmy alokacji workloadów nie są alokacjami optymalnymi pod kątem czy to równoważenia czy też konsolidacji obciążenia. Optymalizowana jest pojedyncza alokacja workloadu rozpatrywanego w danym momencie. Może istnieć bardziej optymalna alokacja, ale mogłaby być ona zastosowana w momencie alokowania wszystkich workloadów na raz, a nie sekwencyjnie w kolejności żądań przychodzących od użytkownika.

W środowisku OpenStack możliwe jest również w łatwy sposób dodanie do algorytmu alokacji mechanizmu odporności na awarie (mechanizmu FT). Jest to realizowane za pomocą funkcjonalności o nazwie *server groups* i polityce tej grupy o nazwie *anti-affinity*. Aby uzyskać efekt równoważenia obciążenia lub konsolidacji z dodatkową dbałością o odporność na awarie dyspozytor musi w konfiguracji dodać

dodatkowy filtr o nazwie *ServerGroupAntiAffinityFilter* (filtr f_9 patrz Tabela 5.1). Wymagana jest również działanie po stronie użytkownika chmury. Użytkownik musi zdefiniować byt nazwany grupą serwerów (ang. *ServerGroup*) i ustawić w nim politykę alokacji maszyn w obrębie grupy na różnych serwerach (ang. *anti affinity*). Następnie przy uruchamianiu maszyn, które mają być objęte utworzoną polityką odporności na awarię, użytkownik musi wskazać grupę serwerów, do której mają te maszyny zostać przypisane. OpenStack do specyfikacji uruchamianych maszyn dodaje odpowiednią adnotację, która jest interpretowana przez filtr f_9.

Załóżmy, że użytkownik utworzył wymaganą grupę serwerów z odpowiednią polityką i zgłasza żądanie uruchomienia dwóch workloadów reprezentowanych przez maszyny wirtualne vm_6 i vm_7 każda o wymaganiach na wirtualne rdzenie w liczbie 8, zaznaczając przy tym dodanie nowych maszyn do grupy serwerów z polityką *anti-affinity*. Aktualny stan środowiska prezentuje Tabela 5.8. Wprowadzono jedną różnicę w stosunku do Tabeli 5.3 zmieniając wymagania maszyny wirtualnej vm_5 z 8 wirtualnych rdzeni procesora na 2. Zmiana została wprowadzona w celu pokazania działania filtra odporności na awarię, które przy stanie środowiska przedstawionego w Tabeli 5.3 nie byłoby możliwe. Dodatkowo dostawca chmury skonfigurował algorytm alokacji tak by równoważył obciążenie ustawiając wartość funkcji oceniającej $scoreWeight(s_1)$ na 1.

Tabela 5.8: Wykorzystanie wirtualnych rdzeni na drugim środowisku testowym

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5
n_1	4	8			
n_2					2
n_3			16	2	

Przy alokacji maszyny wirtualnej vm_6 w pierwszej kolejności (zgodnie z Listingiem 2) następuje etap filtrowania. W filtrze *ServerGroupAntiAffinityFilter* filtr weryfikuje na podstawie metadanych w maszynach wirtualnych, które działają już na

węźle obliczeniowym czy nie jest na nim uruchomiona maszyna wirtualna z tej samej grupy serwerów. Weryfikacja następuje po identyfikatorze grupy tworzonym przez OpenStack w momencie utworzenia grupy przez użytkownika. Identyfikator grupy jest zapisany w metadanych nowej maszyny wirtualnej, która ma zostać uruchomiona jak i w metadanych maszyn wirtualnych uruchomionych na węzłach. Ponieważ jest to pierwsza maszyna wirtualna z tej grupy wszystkie węzły pozostają w puli możliwych do alokacji. Proces alokacji maszyny wirtualnej vm_6 z wartościami wyliczeń dla kolejnych etapów oceny węzłów przedstawia Tabela 5.9.

Tabela 5.9: Alokacja maszyny wirtualnej vm_6 z wykorzystaniem algorytmu równoważenia obciążenia i odpornością na awarie z funkcją oceniającą s_1

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5	$ru(n)$	$s_1(n)$	$ns(n)$
n_1	4	8				**12**	36	0,78
n_2					2	**2**	46	1
n_3			16	2		**18**	30	0,65

Ze względu na najmniejsze obciążenie węzła n_2 zostaje on wybrany jako węzeł do alokacji maszyny vm_6. Następnie dyspozytor przystępuje do alokacji maszyny vm_7. W przypadku, gdy maszyna wirtualna nie byłaby w tej samej grupie serwerów co maszyna wirtualna vm_6 to zgodnie z wartościami przedstawionymi w Tabeli 5.10 trafiłaby ona również na węzeł n_2.

Maszyna została jednak dodana do odpowiedniej grupy serwerów, która ma zapewnić odporność na awarie w związku z czym węzeł n_2 nie przechodzi etapu filtrowania i nie jest on poddawany ocenia. Ocenie podlegają natomiast pozostałe węzły a wartości wyliczeń dla kolejnych etapów oceny węzłów przedstawia Tabela 5.11.

W rezultacie algorytm równoważenia obciążenia z dodanym filtrem uwzględniającym odporność na awarie przypisał maszynę wirtualną vm_6 (reprezentującą workload w_6) do węzła n_2 ($n_2 = a(w_6)$) a maszynę wirtualną vm_7 (reprezentującą

Tabela 5.10: Alokacja maszyny wirtualnej vm_7 z wykorzystaniem algorytmu równoważenia obciążenia bez odporności na awarie z funkcją oceniającej s_1

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5	vm_6	$ru(n_k)$	$s_1(n_k)$	$ns(n_k)$
n_1	4	8					**12**	36	0,95
n_2					2	8	**10**	38	1
n_3			16	2			**18**	30	0,79

Tabela 5.11: Alokacja maszyny wirtualnej vm_6 z wykorzystaniem algorytmu równoważenia obciążenia i odpornością na awarie z funkcją oceniającej s_{cpu}

VMs / węzły	vm_1	vm_2	vm_3	vm_4	vm_5	vm_6	$ru(n_k)$	$s_1(n_k)$	$ns(n_k)$
n_1	4	8					**12**	36	1
n_3			16	2			**18**	30	0,83

workload w_7) do węzła n_1 ($n_1 = a(w_7)$). Takie zachowanie algorytmu powoduje, że awaria pojedynczego węzła nie spowoduje nigdy niedostępności obydwu maszyn. Możemy więc stwierdzić, że uszkodzenie jednej z nich nie spowoduje niedostępności usługi wdrożonej w obrębie jednej grupy serwerów z polityką anti affinity. W przypadku innych usług, które nie miały dodane odpowiedniej polityki przez użytkownika istnieje ryzyko, że w przypadku wykorzystywania przez dostawcę chmury algorytmu CONS workloady, które realizują usługę zostaną zaalokowane na jednym węźle i w przypadku jego awarii cała usługa przestanie być działać.

Przedstawione powyżej przykłady dla dwóch algorytmów alokacji z równoważeniem obciążenia i konsolidacją oraz równoważenia obciążenia z dodatkowym kryterium odporności na awarie prezentują działanie algorytmu w Listingu 2, którym posługuje się dyspozytor oprogramowania OpenStack. Jednocześnie zaprezentowano, że zmiany konieczne do wykonania w konfiguracji dyspozytora w celu zmiany jego zachowania są niewielkie. Tak elastyczna i konfigurowalna implementacja dyspozy-

tora w systemie OpenStack przekłada się na prostotę użytkowania przez dostawców chmury obliczeniowej. Umożliwia to również wprowadzenie nowych mechanizmów zarządzania i ich faktyczne przetestowanie w rzeczywistym środowisku chmurowym. To eliminuje potrzebę wykorzystania symulatorów takich jak CloudSim [web28].

Opisany powyżej mechanizm alokacji w oprogramowaniu OpenStack zostanie rozszerzony o zaproponowaną w dalszej części rozdziału cechę komplementarności workloadów służącą zwiększeniu efektywności ich wykonywania.

5.3 Komplementarność klas workloadów

W rozdziale 4.4 zaproponowano klasy funkcjonalne dla workloadów uruchamianych w chmurze obliczeniowej, a następnie w rozdziale 4.5.3 wykazano możliwość rozpoznania klasy workloadu na podstawie wykorzystywanych przez niego zasobów obliczeniowych. Bazując na tej wiedzy w niniejszym podrozdziale wyróżniono cechę komplementarności klas workloadów. W rozdziale 5.3.1 przedstawiono definicję i sposób badania tej cechy. Rozdział 5.3.2 opisuje przeprowadzone eksperymenty oraz metodę wyznaczenia macierzy komplementarności dla zdefiniowanych klas workloadów. Praktyczne wykorzystanie tej cechy do alokacji workloadów zostało zaprezentowane w rozdziale 5.3.3.

5.3.1 Model komplementarnych workloadów

W oparciu o klasy workloadu przedstawione w 4.4 zaproponowano model alokacji uwzględniający wpływ na efektywność działania workloadu jednej klasy na inny workload z tej samej lub innej klasy, zaalokowany na tym samym węźle chmurowym. Wzajemny wpływ klas workloadów nazwano cechą komplementarności ze względu na wzajemne uzupełnianie się klas w wykorzystaniu zasobów wirtualnych chmury [81].

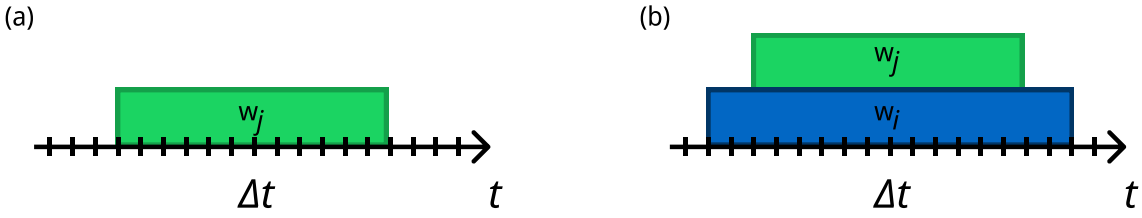
W celu badania komplementarności workloadów wprowadzono miarę efektywności wykonania klas workloadów oznaczaną jako e. Dla każdej klasy workloadów $c_i \in C$ możemy zdefiniować jedną lub więcej miar ze zbioru $E(c_i)$ charakteryzują-

cych tę klasę. Przyjęto, że takie miary określa się w zadanym przedziale czasu Δt. Przykładami pojedynczych miar efektywności wykonania workloadów są:

- średni czas odpowiedzi na określony zestaw wywołań HTTP w przedziale czasu Δt - dla klasy Web Serving,
- średnia liczba operacji na sekundę dla określonego zbioru obliczeń zrealizowanych w przedziale czasu Δt - dla klasy Science,
- średnia liczba transakcji na sekundę dla określonego zbioru zapytań SQL w przedziale czasu Δt - dla klasy OLTP.

Przyjmijmy do rozważań dwa workloady w_i oraz w_j należące do klas odpowiednio $C(w_i)$ oraz $C(w_j)$. Dla każdej klasy workloadów mamy zdefiniowaną miarę $e \in E$. Dla workloadu w_i jest to miara $e(C(w_i))$, natomiast dla workloadu w_j jest to miara $e(C(w_j))$.

Zakładamy, że najpierw pojawia się żądanie wykonania workloadu w_i, a następnie w_j. W zależności od aktualnego stanu środowiska oraz przyjętego algorytmu alokacji workloady w_i oraz w_j mogą zostać zaalokowane na jeden lub dwa węzły obliczeniowe. Obydwie możliwości zaprezentowano na Rysunku 5.3.



Rysunek 5.3: Możliwe alokacje workloadu w_j: (a) na pusty węzeł, (b) na węzeł, gdzie uruchomiony jest workload w_i (źródło: własne)

Na Rysunku 5.3 (a) przedstawiono alokację w_j na pusty węzeł obliczeniowy. Przez cały czas działania tego workloadu był on jedynym workloadem uruchomionym na tym węźle obliczeniowym.

W drugim przypadku przedstawionym na Rysunku 5.3 (b) workload w_j został uruchomiony na węźle chmurowym, na którym był już zaalokowany inny workload

w_i. W tym wypadku miara efektywności opisuje wyrażenie $e(C(w_j)|C(w_i))$, które możemy opisać jako miarę efektywności działania workloadu klasy $C(w_j)$ przy jednoczesnym działaniu na węźle workloadu klasy $C(w_i)$.

Uwzględniając fakt, że klasy workloadów mają podobną charakterystykę (co dowiedziono tworząc klasyfikator dla workloadów) możemy przyjąć, że również w podobny sposób różne workloady z tej samej klasy będą zmieniały swoją efektywność w przypadku działania z workloadem innej klasy. Na tej podstawie możemy zdefiniować miarę komplementarności dla klasy $k = C(w_j)$ z klasą workloadów $l = C(w_i)$ i oznaczyć ją jako $\alpha_{k,l}$. Sposób wyliczenia miary komplementarności przedstawia wzór 5.7.

$$\alpha_{k,l} = \frac{e(k|l)}{e(k)} = \frac{e(C(w_i)|C(w_j))}{e(C(w_i))} \tag{5.7}$$

Miara komplementarności $\alpha_{k,l}$ określa w jaki sposób zmieni się efektywność wykonania workloadu klasy k jeżeli będzie działał równolegle z workloadem klasy l na jednym węźle obliczeniowym. Wyliczając wartości miar komplementarności dla wszystkich klas workloadów zdefiniowanych w rozdziale 4.4 możemy zbudować macierz komplementarności $\alpha = [\alpha_{k,l}]$ opisującą wzajemne zależności pomiędzy wszystkimi klasami workloadów.

Należy przy tym zauważyć, że macierz ta niekoniecznie musi być macierzą symetryczną. Oznacza to, że wpływ workloadów klasy k na workloady klasy l nie musi być taki sam jak workloadów klasy l na workloady klasy k. Należy podkreślić, że zaproponowany model zakłada opracowanie macierzy komplementarności klasy workloadów dla środowiska homogenicznego. W przypadku środowiska heterogenicznego należałoby wyznaczyć macierz komplementarności dla każdego typu węzła chmurowego niezależnie.

5.3.2 Wyznaczenie macierzy komplementarności klas workloadów

Eksperymenty opisane w niniejszym rozdziale przeprowadzono w środowisku laboratoryjnym w chmurze TASKcloud. Do eksperymentów wykorzystano jeden węzeł chmurowy wdrożony na serwerze fizycznym Huawei RH1288. Węzeł posiada dwa fizyczne procesory Intel(R) Xeon(R) CPU E5-2670 v3 o częstotliwości taktowania 2.30GHz. Płyta główna ma osadzone 8 z 16 banków pamięci z wykorzystaniem modułów o rozmiarze 16 GB. Moduły osadzone są równomiernie pomiędzy procesory. Sumarycznie węzeł ma 128 GB pamięci RAM. Na węźle chmurowym zainstalowano system operacyjny CentOS Linux 7. Chmura TASKcloud w momencie przeprowadzania eksperymentów wykorzystywała oprogramowanie OpenStack w wersji Train.

Na potrzeby eksperymentów na węźle chmurowym utworzono dwie maszyny wirtualne dzieląc zasoby serwera fizycznego na dwie równe części. Z poziomu administratora (dostawcy chmury) wymuszono, aby maszyny znajdowały się na tym samym serwerze fizycznym oraz aby żadna inna maszyna nie była na nim uruchomiona. Każda z maszyn wirtualnych dysponowała 24 wirtualnymi rdzeniami (VCPU), 60 GB pamięci RAM (8 GB zostało zarezerwowane na działanie systemu operacyjnego węzła chmurowego) oraz lokalnym dyskiem SSD o pojemności 240 GB. Na maszynach wirtualnych zainstalowano system operacyjny CentOS 7.

Przeprowadzone badania miały na celu utworzenie macierzy komplementarności i empiryczną weryfikację w jakim stopniu workloady różnych klas mogą wpływać wzajemnie na siebie pod względem efektywności działania. Do eksperymentów wybrano 4 z 7 zaproponowanych w rozdziale 4.4 klas workloadów. Były to klasy workloadów: OLTP, Science, Web Serving oraz uwzględniono klasę Idle. Decyzja o wyborze tych tych 3 klas (pomijając klasę Idle) podyktowana była faktem, że są to klasy, z którymi w ramach utrzymania chmury TASKcloud autor rozprawy ma największą styczność i rezultaty eksperymentów mogą być wykorzystane w rzeczywistym rozwiązaniu. Opracowana metoda jest metodą uniwersalną, którą można zastosować dla pozostałych klas jednak przeprowadzenie eksperymentów dla każdej klasy jest bardzo czasochłonne.

Dla każdej z 3 wybranych klas dobrano symulator generujący reprezentatywne obciążenie. Dla klasy OLTP wykorzystano symulator sysbench [web35] generujący obciążenie na silniku bazy danych PostgreSQL. Symulatorem obciążenia dla klasy Science był symulator HPCG [web13]. Z kolei dla klasy Web Serving użyto symulator Siege [web37] do obciążenia aplikacji WordPress. Każdy z symulatorów po zakończeniu pracy generuje raport zawierający miary efektywności wykonania przeprowadzonych obliczeń lub zapytań. Dla każdej z klasy wybrano po jednym parametrze efektywności wykonania e, który został wykorzystany w dalszej analizie. I tak dla poszczególnych klas wybrano następujące parametry:

- OLTP – liczba wykonanych transakcji na bazie PostgreSQL,
- Science – średnia liczba operacji wykonanych na maszynie wirtualnej wyrażona w GFLOPS,
- Web Serving – liczba wykonanych zapytań zapytań do aplikacji WordPress.

Wartości parametrów określane są w różnych jednostkach, jednak proponowana metoda nie wymaga porównywania ich między sobą. W związku z czym nie istnieje problem normalizacji parametrów.

W ramach wstępnych testów zweryfikowano działanie symulatorów i powtarzalność rezultatów ich działania (wartości parametrów, które zwracają). Pomimo faktu, że symulatory były uruchamiane w kontrolowanym środowisku, na wybrany węzeł obliczeniowy trafiały wyłącznie workloady, które były poddawane testom, to wyniki symulatorów dotyczące mierzonych wartości nieznacznie się różniły. Ponieważ przypisanie zasobów obliczeniowych w każdym teście było takie samo, to najprawdopodobniej obserwowane różnice wynikały z wywłaszczania zasobów. W środowisku chmurowym a konkretnie w wirtualizatorze nie ma możliwości ich izolacji i podziału. Takimi zasobami są, np. wspomniana pamięć cache procesora lub przepustowość szyny pamięci. W związku z zaobserwowanymi różnicami pomiarów zdecydowano o przeprowadzeniu większej liczby symulacji, tak aby zweryfikować otrzymane wyniki i sprawdzić jak duże różnice występują pomiędzy kolejnymi symulacjami. Istotne jest by ostatecznie uwiarygodnić otrzymane wyniki.

W ramach eksperymentów przeprowadzono 15 zestawów symulacji na dwóch maszynach wirtualnych. Pojedynczy zestaw obejmował 9 symulacji, w których na przygotowanych maszynach wirtualnych uruchamiano równolegle symulatory dla dwóch klas workloadów oraz 3 symulacje (zwane dalej symulacjami odniesienia), w których na jednej maszynie uruchomiono symulator wybranej klasy a druga maszyna wirtualna reprezentowała workload klasy idle. Lista wszystkich przeprowadzonych symulacji w ramach eksperymentu prezentuje Tabela 5.12.

Tabela 5.12: Lista przeprowadzanych symulacji w ramach eksperymentów badawczych

Lp.	**Workload #1**	**Workload #2**	**Liczba symulacji**
1	OLTP	Idle	15
2	OLTP	OLTP	15
3	OLTP	Science	15
4	OLTP	Web Serving	15
5	Science	Idle	15
6	Science	OLTP	15
7	Science	Science	15
8	Science	Web Serving	15
9	Web Serving	Idle	15
10	Web Serving	OLTP	15
11	Web Serving	Science	15
12	Web Serving	Web Serving	15

W trakcie eksperymentów przeprowadzono zatem 180 symulacji. Eksperymenty zostały przygotowane w taki sposób, by każda symulacja trwała dokładnie 10 min a symulacje prowadzone równolegle na dwóch maszynach były uruchamiane w tym samym momencie. Po każdej symulacji zapamiętano wybraną wartość miary efektywności wykonania workloadu, którą wyliczał symulator na zakończenie wykonania workloadu. Miara ta była wyliczana przez cały czas trwania pojedynczego testu, a za

jej wyliczenie odpowiedzialny był wewnętrzny mechanizm symulatora. W przypadku symulacji, gdzie obciążone były dwie maszyny zebrano dwa razy więcej wyników. Dla przykładu w jednym zestawie symulacji przeprowadzono symulacje w parach klasa OLTP z klasą Science oraz klasa Science z klasą OLTP. W związku z tym uzyskano nie 15 a 30 wyników. Dotyczyło to przypadku pomiarów efektywności wykonania workloadu OLTP w sytuacji kiedy na równoległej maszynie działał workload typu Science. Analogicznie zgromadzono podwojoną liczbę wyników dla pozostałych par klas, za wyjątkiem klasy Idle.

Dla każdej z par klas workloadów oraz dla symulacji odniesienia dla uzyskanych wartości miar efektywności wykonania workloadu wyliczono średnią arytmetyczną oraz odchylenie standardowe. Uzyskane rezultaty prezentuje Tabela 5.13. Odchylenie standardowe dla uzyskanych wyników mieści się w przedziale $< 0,8\%; 5,5\% >$ jedynie w przypadku pary klas Science i Idle jest ono większe i wynosi $8,4\%$ co może mieć związek ze sposobem przypisania workloadu do poszczególnych rdzeni procesora i wymianą danych pomiędzy procesorami.

Tabela 5.13: Średnia arytmetyczna i odchylenie standardowe dla wyników uzyskanych z 15 zestawów symulacji.

Lp.	**Rozpatrywana para workloadów** $(C_1\&C_2)$	**Średnia arytmetyczna wartości** $e(C_1)$	**Odchylenie standardowe wartości** $e(C_1)$
1	OLTP & Idle	3046281,20	60865,90
2	OLTP & OLTP	2066845,87	82546,60
3	OLTP & Science	1821061,60	35820,42
4	OLTP & Web Serving	2086248,25	113630,90
5	Science & Idle	8,68	0,73
6	Science & OLTP	8,12	0,40
7	Science & Science	6,38	0,26
8	Science & Web Serving	8,13	0,37
9	Web Serving & Idle	1089213,20	9322,04
10	Web Serving & OLTP	885134,43	26602,86
11	Web Serving & Science	800503,32	17697,52
12	Web Serving & Web Serving	859504,43	28959,79

Na podstawie uzyskanych wyników utworzono macierz $\alpha = [\alpha_{k,l}]$, w której poszczególne wiersze i kolumny reprezentują klasy workloadów poddane eksperymentom. Następnie zgodnie z równaniem 5.7 obliczono wartości współczynników komplementarności wybranych klas workloadów. Dodatkowo w Tabeli 5.13 uwzględniono klasę Idle, dla której przyjęto, że współczynnik efektywności wynosi 1 dla każdej klasy. Wartość 1 wynika z faktu, że w momencie, gdy workload z dowolnej klasy k działa jako jedyny na węźle to nie zmienia się jego efektywność wykonania zgodnie z równaniem 5.8:

$$\alpha_{k,idle} = \frac{e(k|idle)}{e(k)} = \frac{e(k)}{e(k)} = 1 \tag{5.8}$$

Tabelę z empirycznie wyliczonymi wartościami współczynnika komplementarności dla wybranych klas workloadów przedstawia Tabela 5.14.

Tabela 5.14: Wartości współczynników komplementarności workloadów dla wybranych klas funkcjonalnych

	Science	**OLTP**	**Web Serving**	**Idle**
Science	0,74	0,94	0,94	1,0
OLTP	0,60	0,68	0,68	1,0
Web Serving	0,73	0,81	0,79	1,0
Idle	1,0	1,0	1,0	1,0

Na podstawie uzyskanych wyników utworzono macierz komplementarności zaprezentowaną w równaniu 5.9.

$$\alpha = \begin{bmatrix} 0,74 & 0,94 & 0,94 & 1,0 \\ 0,60 & 0,68 & 0,68 & 1,0 \\ 0,73 & 0,81 & 0,79 & 1,0 \\ 1,0 & 1,0 & 1,0 & 1,0 \end{bmatrix} \tag{5.9}$$

Zaprezentowane wyniki eksperymentów potwierdziły, że macierz komplementarności nie musi być symetryczna. Świadczy to o tym, że działanie workloadu na-

leżącego do klasy c_k wpływa w innym stopniu na działanie workloadu należącego do klasy c_l niż działanie workloadu należącego do klasy c_l na działanie workloadu należącego do klasy c_k. Na podstawie macierzy komplementarności (5.9) możemy stwierdzić, że najmniej korzystną jest alokacja workloadów z klasy OLTP z workloadami z klasy Science, gdyż efektywność wykonania workloadu OLTP spada do 60%. W związku z niesymetrycznością tej macierzy, efektywność wykonania workloadu Science spada do 94%. Dla macierzy komplementarności możemy określić minimalną i maksymalną wartość. Minimalną wartością jest 0 w momencie kiedy wykonanie workloadu nie jest możliwe (przestaje działać prawidłowo) pod wpływem innego workloadu działającego na tym samym węźle. Natomiast maksymalną wartością jest $1,0$ (sytuacja analogiczna do klasy idle) kiedy to działanie innego workloadu na tym samym węźle nie wpływa na działanie rozważanego workloadu. Aby rozszerzyć macierz komplementarności na pozostałe 3 klasy workloadów należałoby wybrać dla nich odpowiednie symulatory oraz właściwe parametry efektywności i na drodze eksperymentów możliwe byłoby wyznaczenie wartości w macierzy dla pozostałych klas. Należałoby również uwzględnić w eksperymentach pary workloadów stworzone z klas, które zostały już zweryfikowane w niniejszej rozprawie doktorskiej z pozostałymi workloadami. Niestety takie eksperymenty są czasochłonne zaś przeprowadzone symulacje dla wybranych klas potwierdzają zauważone powyżej tendencje.

5.3.3 Algorytm alokacji workloadów uwzględniający cechę komplementarności

W praktyce cecha komplementarności może zostać wykorzystana przy wyborze węzła chmurowego, na którym ma zostać uruchomiony zgłaszany do wykonania workload. To oznacza, że wybór węzła, na którym będzie uruchomiony ten workload zależy od miary komplementarności tzn. tam gdzie alokowane workloady mniej będą wpływać na siebie (będą bardziej komplementarne), co wpłynie na efektywniejsze działanie nowo alokowanego workloadu, a w konsekwencji zminimalizuje spadek efektywności działających na węźle pozostałych workloadów.

Wdrożenie takiej dodatkowej funkcji lub algorytmu wykorzystującego proponowaną macierz komplementarności może wymagać różnego nakładu pracy w zależności od środowiska implementacji systemu zarządzania chmurą obliczeniową. W niniejszym rozdziale zaprezentowano taką propozycję dla systemu OpenStack. Z założenia algorytm ma realizować funkcję oceniania węzłów obliczeniowych pod kątem możliwości alokacji nowego workloadu. Wpisuje się to w ideę funkcji oceniających, których działanie zostało opisane w rozdziale 5.2. Poniżej przedstawiono implementację funkcji oceniającej wspierającej algorytm alokacji systemu OpenStack. Funkcja oceniająca została nazwana *complementarityNodeWeigher* (CNW). Do implementacji i wdrożenia funkcji oceniającej niezbędne jest wykorzystanie klasyfikatora opracowanego w rozdziale 4.5.2. Klasyfikator zostanie wykorzystany do rozpoznania klasy funkcjonalnej workloadu, jeżeli nie została ona podana przez użytkownika przy uruchomieniu tego workloadu.

W wyniku działania funkcji oceniającej CNW otrzymujemy ocenę dopasowania węzła obliczeniowego zdefiniowanego na wejściu funkcji do wykonania workloadu zdefiniowanego przez użytkownika chmury w przesłanym żądaniu. Pseudokod implementacji funkcji oceniającej prezentuje Listing 3.

W przedstawionym pseudokodzie funkcją oceniającą jest funkcja CNW. Dodatkowo przedstawiono funkcję weryfikującą klasę funkcjonalną workloadu *getCategory*. Funkcja oceniająca CNW w pierwszej kolejności weryfikuje czy na węźle jest uruchomiony jakikolwiek inny workload (linia 13). Alokacja na pustym węźle jest najlepszym rozwiązaniem, gdyż nie będzie występowało zjawisko wzajemnego oddziaływania. stąd ocena pustego węzła wynosi 1 (linia 14).

Następnie tworzony jest zbiór składający się z wszystkich zaalokowanych workloadów na ocenianym węźle oraz nowego workloadu do alokacji (linia 17). Na utworzonym zbiorze wywoływana jest funkcja zwracająca wszystkie możliwe pary workloadów (linia 18). Inicjalizowana jest również pusta tablica, w której będą zbierane wartości współczynnika komplementarności dla kolejnych par (linia 19).

Kolejnym krokiem jest zebranie wartości macierzy korelacji dla każdej z par workloadów (linie 21:27). Najpierw dla każdego workloadu z pary odczytywana jest

Listing 3 Pseudokod implementacji funkcji oceniającej CNW

```
1: alpha ← macierz komplementarności α
2: classifier ← klasyfikator workloadów
3: vm ← maszyna wirtualna reprezentująca workload
4: function getCategory(vm ← maszyna realizująca workload) {
5:   if vm.category is not empty then
6:     return vm.category;
7:   else
8:     return classifier.classify(vm);
9:   end if
10: }
11:
12: function complementarityNodeWeigher(n←node, vm ←new vm specification)
    {
13:   if length(node.workloads()) == 0 then
14:     return 1.0;
15:   end if
16:
17:   workloadsSet = [vm] + node.workloads()
18:   workloadsCombinations = combinations(workloadSet, 2)
19:   partialScore = array()
20:
21:   for all item : workloadsCombinations do
22:       w1category = getCategory(item[0])
23:       w2category = getCategory(item[1])
24:
25:       partialScore.append(alpha[w1category, w2category])
26:       partialScore.append(alpha[w2category, w1category])
27:   end for
28:   score = average(partialScore)
29:   return score;
30: }
```

jego klasa funkcjonalna, a następnie do utworzonej wcześniej tablicy dopisywane są wartości współczynnika α. Dla każdej pary workloadów dodawane są dwie wartości α, gdyż należy uwzględnić wzajemne oddziaływanie workloadów. Ostatnim krokiem jest wyliczenie średniej wartości współczynnika komplementarności ze wszystkich wartości częściowych (linia 28) i zwrócenie wartości oceny węzła.

Do powyższego opisu należy dodać wyjaśnienie dotyczące klasyfikacji nowego workloadu. Jeżeli użytkownik przy uruchamianiu nie zdefiniował klasy funkcjonalnej workloadu to zostanie podjęta próba określenia klasy za pomocą klasyfikatora (linie 5-8). Klasyfikator ze względu na brak wartości metryk zbieranych w trakcie pracy workloadu, również nie będzie w stanie ocenić jego klasy. Z tego powodu klasyfikator przypisze nowy workload do klasy Idle. Klasa Idle zakłada, że workload nie wpływa na inne workloady działające na węźle stąd zostanie założona najbardziej optymistyczna z możliwych klas. Określenie rzeczywistej klasy workloadu będzie możliwe przy ewentualnej migracji po zebraniu przez agenty monitorujące odpowiednich metryk (patrz Tabela 4.4).

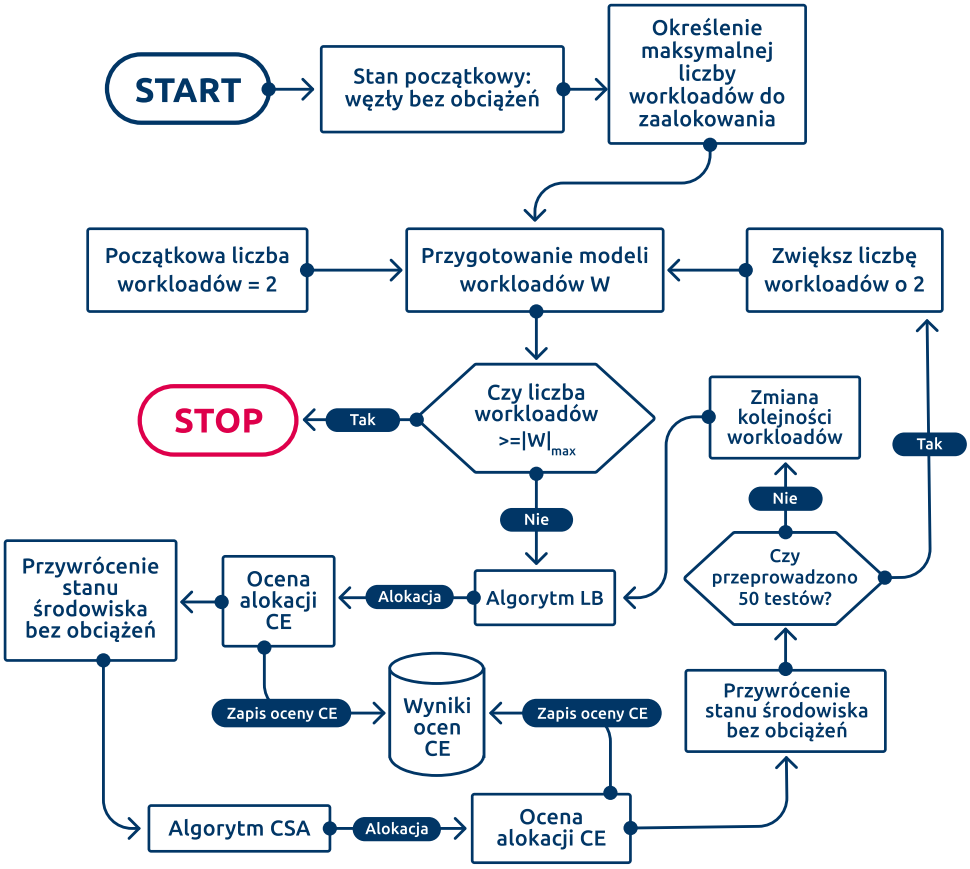
Po dokonaniu oceny wszystkich węzłów przez proponowany algorytm, mechanizm zarządzania systemem OpenStack wybierze węzeł z najwyższą ocenę. Innymi słowy wybierze węzeł, na którym po zaalokowaniu nowego workloadu spadek średniej efektywności wykonania workloadów na wybranym węźle będzie najmniejszy.

W ramach prac związanych z niniejszą rozprawą doktorską została utworzona prototypowa implementacja algorytmu alokacji z wykorzystaniem funkcji CNW o nazwie *Complementarity Scheduling Algorithm* (CSA). Działanie algorytmu CSA porównano z najczęściej wykorzystywanym w chmurach obliczeniowych algorytmem równoważenia obciążenia LB pomiędzy węzły chmurowe. Algorytm LB jest również wykorzystywany w chmurze TASKcloud. Rezultat porównania algorytmów przedstawiono w następnym rozdziale.

5.3.4 Analiza działania algorytmu CSA

Do oceny działania algorytmu alokacji CSA z wykorzystaniem funkcji oceniającej CNW przeprowadzono odpowiednie eksperymenty i dokonano porównania jego wyników z alokacją wykonaną za pomocą algorytmu LB.

Na potrzeby eksperymentu przygotowano program, który symulował działanie algorytmu z CSA oraz algorytmu LB. Schemat przeprowadzania eksperymentów prezentuje Rysunek 5.4



Rysunek 5.4: Schemat realizacji eksperymentów porównawczych dla algorytmów CSA i LB (źródło: własne)

W programie zamodelowano środowisko chmurowe, w którym możliwe było zadanie liczby węzłów w tym środowisku oraz liczby wirtualnych zasobów, którymi węzeł dysponuje. Zaimplementowano działanie algorytmu CNA (zgodnie z pseudokodem zaprezentowanym w Listingu 3), algorytm LB (zgodnie z działaniem przed-

stawionym w rozdziale 5.3.3). Przyjęto funkcję oceniającą efektywność wykonania wszystkich workloadów uruchomionych w modelowanym środowisku po ich alokacji (*environmentComplementaryScorer*, ECS). Funkcja ECS została zaimplementowana z wykorzystaniem funkcji CNW. W funkcji ECS dla każdego węzła wykonywana jest ocena efektywności zgodnie z CNW a następnie z otrzymanych wartości wyliczana jest średnia arytmetyczna.

Za pomocą funkcji ECS były oceniane zarówno alokacje wykonane algorytmem z CNW jaki i algorytmem LB. Wykorzystana funkcja CNW zwraca ocenę z przedziału $[0, 1]$, która interpretujemy jako procent efektywności wykonania workloadów na węźle. Wynikiem prezentowanym w dalszej części była różnica w ocenie efektywności algorytmu CSA oraz algorytmu LB. Uzyskana wartość niesie informację o ile bardziej efektywne jest wykonanie workloadów w dokonanej alokacji z wykorzystaniem algorytmu CSA niż za pomocą algorytmu LB. Wartość tę oznaczymy jako CE (ang. *complementarity effecitveness*).

Przeprowadzono eksperymenty dla trzech zamodelowanych środowisk o następującej liczbie węzłów chmurowych: $[8, 32, 64]$. Przyjęto następujące założenia:

- Pojedynczy węzeł obliczeniowy dysponuje 48 wirtualnymi zasobami.
- Workloady mogą mieć rozmiary maszyny: $2, 4, 6$ lub 8 jednostek wirtualnych zasobów.
- Rozmiar workloadu był dobierany losowo z przyjętego zakresu na etapie przygotowania symulacji.
- Maksymalna liczba workloadów do uruchomienia w środowisku była wyznaczana zgodnie ze wzorem 5.10

$$|W|_{max} = \frac{|N| * 48}{8} * wspolczynnik\_wypenienia \tag{5.10}$$

We wzorze występuje tzw. współczynnik wypełnienia, który ma na celu zwiększenie obciążenia chmury przez workloady. Zakładając równomierny podział rozmiarów maszyn na każdy z rozmiarów (czyli 25% maszyn o rozmiarze odpowiednio $2, 4, 6, 8$) bez zastosowania współczynnika wypełnienia statystycznie

tylko $62,5\%$ zasobów chmury byłoby wykorzystane. Przyjmując, że w środowisku powinno zostać około 10% zasobów wolnych przyjęto wartość wskaźnika wypełnienia równą $1,4$ ($\frac{90\%}{62.5\%} = 1,44 \approx 1,4$).

- Proces alokacji na zadanym zbiorze workloadów przeprowadzono 50 razy a otrzymane wyniki uśredniono.
- Ze względu na sekwencyjną przetwarzani workloadów przez algorytmy alokacji przed każdym testem losowano kolejność workloadów w zbiorze.

Pojedynczy eksperyment przebiegał zgodnie z procedurą przedstawioną w Listingu 4.

Wyniki symulacji dla środowiska z 8 węzłami chmurowymi przedstawia wykres na Rysunku 5.5. Wykres liniowy ilustruje o ile większą efektywność wykonania średnio workloady po alokacji za pomocą algorytmu wykorzystującego CNW w porównaniu z alokacją wykonaną przez algorytm LB. Wykres kolumnowy przedstawia procentową zajętość zasobów chmury obliczeniowej.

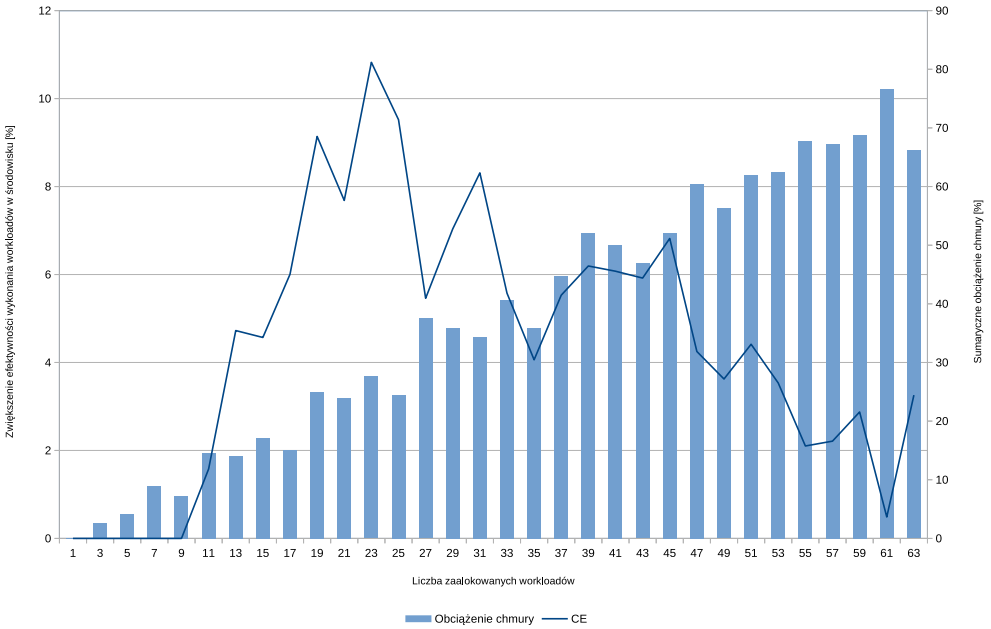
Wyniki symulacji dla innej liczby węzłów prezentują wykresy przedstawione na Rysunkach 5.6 oraz 5.7 odpowiednio dla środowiska z 32 oraz 64 węzłami chmurowymi.

Na wszystkich wykresach możemy zaobserwować zerowe zwiększenie efektywności do momentu, w którym alokowana jest liczba workloadów mniejsza lub równa liczbie wszystkich węzłów w środowisku chmury obliczeniowej. Wynika to z faktu, że zarówno algorytm LB jaki i algorytm CSA preferują alokację workloadu na pustym węźle. Dla liczby workloadów powyżej liczby węzłów w środowisku (odpowiednio powyżej 8, 32 i 64 workloadów) następuje zwiększenie wartości metryki CE w przedziale od 20% do 70% całkowitego obciążenia chmury, jest ona powyżej wartości 4%. Dla środowisk o większej liczbie węzłów, w zakresie $30\% - 70\%$ obciążenia, metryka CE wynosi powyżej 8% Powyżej 70% obciążenia wartość metryki spada, co można wyjaśnić ograniczonymi możliwościami wyboru węzła, na którym ma zostać zaalokowany workload ze względu na mniejszą liczbę dostępnych zasobów.

Zaprezentowane w rozdziale symulacje pokazują, że dla środowisk chmurowych,

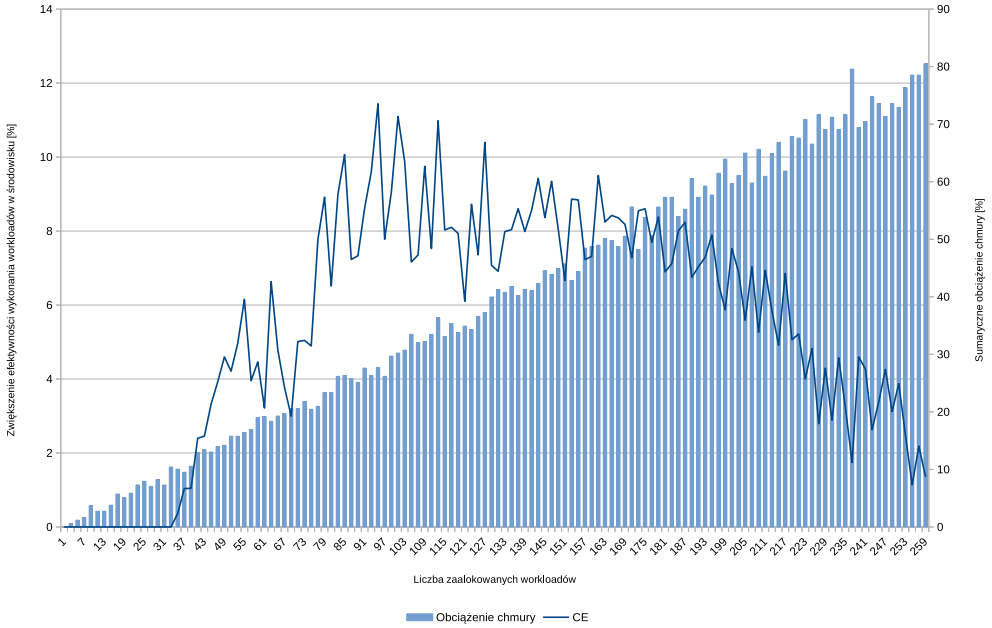
Listing 4 Pseudokod programu oceniającego zwiększenie efektywności workloadów zaalokowanych za pomocą algorytmu CSA w porównanie z alokacją dokonaną za pomocą algorytmu LB

```
1: |N| ← liczba węzłów w środowisku
2: N = initNodes(|N|)
3: maxWorkloadCount = |N|*48/8 * 1.4
4: ce = array()
5: for all workloadCount in range(1, maxWorkloadCount, 2) do
6:    W = array()
7:    for all i in range(1, workloadCount) do
8:       W.append(Workload(resources=power(2, rand(1,3)), category=rand(1,3))
9:    end for
10:   lbScore = array()
11:   csaScore = array()
12:   for all i in range(1, 50) do
13:      shuffle(w)
14:      lbAllocation = lb(W,N)
15:      lbScore.append(ecs(lbAllocation))
16:      clear(N) // wyzerowanie stanu środowiska
17:      csaAllocation = csa(W,N)
18:      csaScore.append(ecs(csaAllocation))
19:      clear(N)
20:   end for
21:   lbFinalScore = average(lbScore)
22:   csaFinalScore = average(csaScore)
23:   ce.append(|W|, csaFinalScore-lbFinalScore)
24: end for
25: return ce
```

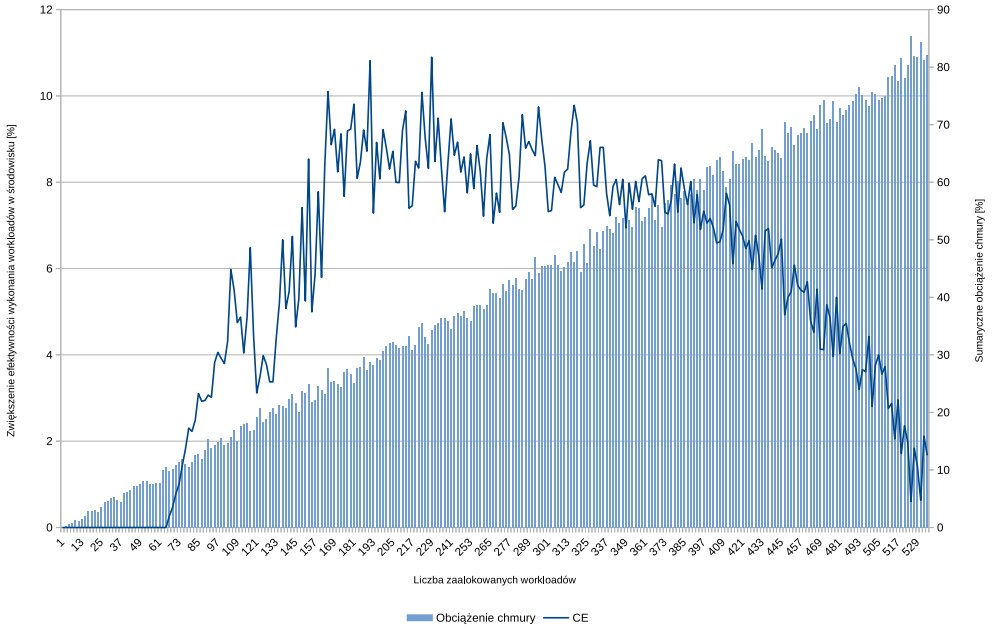


Rysunek 5.5: Procentowe zwiększenie efektywności wykonania workloadów (CE) przy alokacji algorytmem CSA w porównaniu do algorytmu LB dla środowiska 8 węzłowego (źródło: własne)

w których zastosowano algorytm alokacji CSA wykorzystujący funkcję oceny CNW, czyli funkcję oceny wyboru węzła wykorzystującą komplementarność klas workloadów średnia efektywność wykonania workloadów wzrośnie o około 8%. Tym samym potwierdzono tezę, że znajomość klasy funkcjonalnej workloadu i jej wykorzystanie w procesie alokacji na zasobach chmury obliczeniowej może pozytywnie wpłynąć na efektywność wykonania workloadów. Taki wzrost jest jest istotny, gdyż oznacza (patrz rozdział 3) lepsze wykorzystanie zasobów oraz zmniejszone zużycie energii elektrycznej. Co więcej tego typu podejście może być w łatwy sposób zaimplementowane i wdrożone w środowisku OpenStack, co również podnosi jego praktyczną przydatność.



Rysunek 5.6: Procentowe zwiększenie efektywności wykonania workloadów (CE) przy alokacji algorytmem CSA w porównaniu do algorytmu LB dla środowiska 32 węzłowego (źródło: własne)



Rysunek 5.7: Procentowe zwiększenie efektywności wykonania workloadów (CE) przy alokacji algorytmem CSA w porównaniu do algorytmu LB dla środowiska 64 węzłowego (źródło: własne)

Rozdział 6

Wnioski końcowe

W rozprawie doktorskiej zaprezentowano architekturę i mechanizmy funkcjonowania chmury obliczeniowej zrealizowanej na podstawie otwartego oprogramowania. Podkreślono złożoność takiego systemu oraz konieczność przeprowadzenia niezbędnych eksperymentów w celu zweryfikowania przyjętych hipotez badawczych. Główną hipotezą było zwiększenie efektywności działania workloadów w chmurze obliczeniowej poprzez wprowadzenie i wykorzystanie klas funkcjonalnych worklodów (reprezentujących usługi) oraz określenie ich macierzy komplementarności.

Proces budowy, wdrożenia oraz rozwoju chmury obliczeniowej TASKcloud jest zadaniem złożonym i wymagającym szczegółowej wiedzy o poszczególnych komponentach. Na instancji TASKcloud przeprowadzono szereg eksperymentów, których wyniki zostały przedstawione w niniejszej rozprawie doktorskiej. Wykorzystano również rezultaty eksperymentów przeprowadzonych m.in. przez autora rozprawy w projekcie Komponent Rekomendacji dla Inteligentnych Chmur Obliczeniowych w środowisku przygotowanym przez Intel Technology Poland co pozwoliło na zwiększenie rozmiaru danych do analiz. W oparciu o wyniki eksperymentów przeprowadzono analizę zaproponowanej klasyfikacji funkcjonalnej oraz zaproponowano klasyfikator do identyfikacji workloadów nieznanej klasy. Następnie wykorzystując zaproponowane klasy określono cechę komplementarności worklodów i określono macierz komplementarności klas funkcjonalnych. Jednocześnie wykazano skuteczność algorytmu CSA, którego wykorzystanie zwiększa efektywność wykonania workloadów.

Do najważniejszych osiągnięć rozprawy należy zaliczyć:

1. Zaprojektowanie i wdrożenie chmury obliczeniowej TASKcloud w Centrum Informatycznym TASK do przeprowadzania eksperymentów naukowych. Autor rozprawy doktorskiej jest jednym z dwóch głównych autorów opracowanego rozwiązania.

2. Wyznaczenie 7 klas funkcjonalnych workloadów działających w chmurach obliczeniowych wraz z potwierdzeniem możliwości identyfikacji klasy funkcjonalnej workloadu na podstawie wykorzystywanych przez workload zasobów obliczeniowych.

3. Zaproponowanie cechy komplementarności określającej wzajemny wpływ par klas workloadów na efektywność ich wykonania oraz podanie procedury wyznaczenia macierzy komplementarności.

4. Implementacja algorytmu wykorzystującego cechę komplementarności w celu wykazania zysku z jego wykorzystania w porównaniu do powszechnie używanego algorytmu równoważenia obciążenia.

5. Przeprowadzenie eksperymentów potwierdzających tezę pracy w środowisku otwartego oprogramowania OpenStack (co czyni je wiarygodnym dla różnych wdrożeń chmurowych opartych o to rozwiązanie).

Teza rozprawy została potwierdzona empirycznie w rozdziale 5.3.4, gdzie wykazano, że zaproponowana klasyfikacja workloadów oraz wykorzystanie cechy komplementarności pozwalają na zwiększenie efektywności wykonywania workloadów w chmurze obliczeniowej o ok. 8%.

Zaproponowano modyfikację algorytmów dostępnych w oprogramowaniu OpenStack. Jak udowodniono pozwoliła ona na zwiększenie efektywności wykorzystania zasobów obliczeniowych chmury. W środowisku chmurowym zachodzą złożone procesy, których częściowo nie jesteśmy w stanie modelować i często nie potrafimy teoretycznie wytłumaczyć. Co więcej korzystanie z coraz bardziej skomplikowanych algorytmów wymusza zużycie coraz większych mocy obliczeniowych dlatego jedną

z zasad informatyki jest zasada *simplicity*, która zaleca by złożone problemy były upraszczane. Zaproponowane podejście jak i główny algorytm CSA, prosty w swym działaniu dopasowuje się do tej zasady.

Zaproponowana cecha komplementarności jak i zbudowany klasyfikator ukazuje pewną ścieżkę badawczą, którą z pewnością należałoby rozszerzać. Potencjalnymi kierunkami badań jest gromadzenie zbiorów danych z rzeczywistych workloadów wykonywanych w różnych chmurach obliczeniowych. na tej podstawie istnieje możliwość przeprowadzenia dalszej weryfikacji przeprowadzonych badań. Dotyczy to również kompleksowej weryfikacji modelu workloadów dla różnych klas funkcjonalnych. Kolejnym kierunkiem dalszych analiz może być przeprowadzenie klasyfikacji w oparciu o charakterystyki wielkości wykorzystywanych zasobów na bieżąco a nie tak jak zaprezentowano w niniejszej rozprawie w oparciu o średnie wykorzystanie w czasie konkretnych eksperymentów. Innym aczkolwiek nie najmniej ważnym kierunkiem zidentyfikowanym na etapie dotychczasowych badań byłaby próba przeniesienia modelu z workloadów chmurowych opartych o maszyny wirtualne (model IaaS) na model bazujący na kontenerach, czyli model PaaS. Ciekawym podejściem byłaby weryfikacja klas funkcjonalnych w oparciu o metryki uzyskane z agentów monitorujących zasoby wykorzystywane przez kontenery. W przypadku kontenerów jest zdecydowanie mniejszy narzut na wykorzystywane zasoby spowodowany wirtualizacją, co mogłoby pozwolić na budowę i wykorzystanie bardziej dokładnego klasyfikatora. Tego typu problemy są istotne dla rozwoju metod zarządzania zasobami chmury obliczeniowej i mieszczą się w dalszych planach badawczych autora rozprawy doktorskiej jak i Centrum Informatycznego TASK.

Dodatek A

Metryki gromadzone w fazie I eksperymentów

Lista metryk wraz z opisem gromadzona przez agenty monitorujące w I fazie eksperymentów.

Tabela A.1: Lista metryk gromadzonych w I fazie eksperymentów

Parametr	Opis
/proc/stat	
stat::system::user-time	Time spent in user mode.
stat::system::nice-time	Time spent in user mode with low priority (nice).
stat::system::system-time	Time spent in system mode.
stat::system::idle-time	Time spent in the idle task. This value should be USER_HZ times the second entry in the /proc/uptime pseudo-file.
stat::system::iowait-time	Time waiting for I/O to complete.
stat::system::irq-time	Time servicing interrupts.
stat::system::softirq-time	Time servicing softirqs.
stat::system::steal-time	Stolen time, which is the time spent in other operating systems when running in a virtualized environment
stat::system::guest-time	Time spent running a virtual CPU for guest operating systems under the control of the Linux kernel.
stat::system::guest-nice-time	Time spent running a niced guest (virtual CPU for guest operating systems under the control of the Linux kernel).

kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
/proc/<pid>/stat	
stat::process::minor-faults	The number of minor faults the process has made which have not required loading a memory page from disk.
stat::process::minor-faults-children	The number of minor faults that the process's waited-for children have made.
stat::process::major-faults	The number of major faults the process has made which have required loading a memory page from disk.
stat::process::major-faults-children	The number of major faults that the process's waited-for children have made.
stat::process::user-time	Amount of time that this process has been scheduled in user mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)). This includes guest time, guest_time (time spent running a virtual CPU, see below), so that applications that are not aware of the guest time field do not lose that time from their calculations.
stat::process::system-time	Amount of time that this process has been scheduled in kernel mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).
	kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
stat::process::user-time-children	Amount of time that this process's waited-for children have been scheduled in user mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)). (See also times(2).) This includes guest time, cguest_time (time spent running a virtual CPU, see below).
stat::process::system-time-children	Amount of time that this process's waited-for children have been scheduled in kernel mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).
stat::process::threads	Number of threads in this process (since Linux 2.6). Before kernel 2.6, this field was hard coded to 0 as a placeholder for an earlier removed field.
stat::process::block-io-delays	Aggregated block I/O delays, measured in clock ticks (centiseconds).
stat::process::guest-time	Guest time of the process (time spent running a virtual CPU for a guest operating system), measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).
stat::process::guest-time-children	Guest time of the process's children, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).
/proc/loadavg	
stat::system::run-queue-length-1min	the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 1 minute
kontynuacja na następnej stronie	

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
stat::system::run-queue-length-5min	the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 5 minutes
stat::system::run-queue-length-15min	the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 15 minutes
/proc/meminfo	
memory::system::free-memory-size	The sum of LowFree+HighFree.
memory::system::buffers-size	Relatively temporary storage for raw disk blocks that shouldn't get tremendously large (20MB or so).
memory::system::memory-cache-size	In-memory cache for files read from the disk (the page cache). Doesn't include SwapCached.
memory::system::swap-cache-size	Memory that once was swapped out, is swapped back in but still also is in the swap file. (If memory pressure is high, these pages don't need to be swapped out again because they are already in the swap file. This saves I/O.)
memory::system::active-memory-size	Memory that has been used more recently and usually not reclaimed unless absolutely necessary.

kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
memory::system::inactive-memory-size	Memory which has been less recently used. It is more eligible to be reclaimed for other purposes.
/proc/<pid>/statm	
memory::process::total-pages	total program size (same as VmSize in /proc/[pid]/status)
memory::process::resident-set-pages	resident set size (same as VmRSS in /proc/[pid]/status)
memory::process::shared-pages	shared pages (i.e., backed by a file)
memory::process::code-pages	text (code)
memory::process::data-pages	data + stack
/proc/<pid>/io	
io::process::characters-read	The number of bytes which this task has caused to beread from storage. This is simply the sum of bytes which this process passed to read(2) and similar system calls. It includes things such as terminal I/O and is unaffected by whether or not actual physical disk I/O was required (the read might have been satisfied from pagecache).
io::process::characters-written	The number of bytes which this task has caused, or shall cause to be written to disk. Similar caveats apply here as with rchar.

kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
io::process::read-syscalls	Attempt to count the number of read I/O operations—that is, system calls such as read(2) and pread(2).
io::process::write-syscalls	Attempt to count the number of write I/O operations—that is, system calls such as write(2) and pwrite(2).
io::process::bytes-read	Attempt to count the number of bytes which this process really did cause to be fetched from the storage layer. This is accurate for block-backed filesystems.
io::process::bytes-written	Attempt to count the number of bytes which this process caused to be sent to the storage layer.
/sys/class/hwmon	
energy::system::cpu-temperature	/sys/class/hwmon/hwmon0/temp1_input
Performance Application Programming Interface (PAPI)	
cpu::process::unhalted-core-cycles	ix86arch::UNHALTED_CORE_CYCLES count core clock cycles whenever the clock signal on the specific core is running (not halted)
	kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
cpu::process::instructions-retired	ix86arch::INSTRUCTION_RETIRED count the number of instructions at retirement. For instructions that consists of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction
cpu::process::llc-references	ix86arch::LLC_REFERENCES count each request originating from the core to reference a cache line in the last level cache. The count may include speculation, but excludes cache line fills due to hardware prefetch
cpu::process::llc-misses	ix86arch::LLC_MISSES count each cache miss condition for references to the last level cache. The event count may include speculation, but excludes cache line fills due to hardware prefetch
cpu::process::branch-instructions-retired	ix86arch::BRANCH_INSTRUCTIONS_RETIRED count branch instructions at retirement. Specifically, this event counts the retirement of the last micro-op of a branch instruction

kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
cpu::process::mispredicted-branch -instructions-retired	ix86arch::MISPREDICTED_BRANCH_RETIRED count mispredicted branch instructions at retirement. Specifically, this event counts at retirement of the last micro-op of a branch instruction in the architectural path of the execution and experienced misprediction in the branch prediction hardware
cpu::process::bus-cycles	perf::PERF_COUNT_HW_BUS_CYCLES
cpu::process::stalled-cycles-frontend	perf::PERF_COUNT_HW_STALLED_CYCLES_FRONTEND
cpu::process::stalled-cycles-backend	perf::PERF_COUNT_HW_STALLED_CYCLES_BACKEND
cpu::process::clock	perf::PERF_COUNT_SW_CPU_CLOCK
cpu::process::task-clock	perf::PERF_COUNT_SW_TASK_CLOCK
cpu::process::context-switches	perf::PERF_COUNT_SW_CONTEXT_SWITCHES
cpu::process::cpu-migrations	perf::PERF_COUNT_SW_CPU_MIGRATIONS
cpu::process::cache-l1d-loads	perf::L1-DCACHE-LOADS L1 cache load accesses
cpu::process::cache-l1d-load-misses	perf::L1-DCACHE-LOAD-MISSES L1 cache load misses
cpu::process::cache-l1d-stores	perf::L1-DCACHE-STORES L1 cache store accesses
cpu::process::cache-l1d-store-misses	perf::L1-DCACHE-STORE-MISSES L1 cache store misses
kontynuacja na następnej stronie	

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
cpu::process::cache-l1d-prefetches	perf::L1-DCACHE-PREFETCHES L1 cache prefetch accesses
cpu::process::cache-l1d-prefetch-misses	perf::L1-DCACHE-PREFETCH-MISSES L1 cache prefetch misses
cpu::process::cache-l1i-loads	perf::L1-ICACHE-LOADS L1I cache load accesses
cpu::process::cache-l1i-load-misses	perf::L1-ICACHE-LOAD-MISSES L1I cache load misses
cpu::process::cache-llc-loads	perf::LLC-LOADS Last level cache load accesses
cpu::process::cache-llc-load-misses	perf::LLC-LOAD-MISSES Last level cache load misses
cpu::process::cache-llc-stores	perf::LLC-STORES Last level cache store accesses
cpu::process::cache-llc-store-misses	perf::LLC-STORE-MISSES Last level cache store misses
cpu::process::cache-llc-prefetches	perf::LLC-PREFETCHES Last level cache prefetch accesses
cpu::process::cache-llc-prefetch-misses	perf::LLC-PREFETCH-MISSES Last level cache prefetch misses
cpu::process::dtlb-loads	perf::DTLB-LOADS Data TLB load accesses
cpu::process::dtlb-load-misses	perf::DTLB-LOAD-MISSES Data TLB load misses
cpu::process::dtlb-stores	perf::DTLB-STORES Data TLB store accesses
cpu::process::dtlb-store-misses	perf::DTLB-STORE-MISSES Data TLB store misses
cpu::process::itlb-loads	perf::ITLB-LOADS Instruction TLB load accesses
cpu::process::itlb-load-misses	perf::ITLB-LOAD-MISSES Instruction TLB load misses

kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
cpu::process::branch-loads	perf::BRANCH-LOADS Branch load accesses
cpu::process::branch-load-misses	perf::BRANCH-LOAD-MISSES Branch load misses
cpu::process::node-loads	perf::NODE-LOADS Node load accesses
cpu::process::node-load-misses	perf::NODE-LOAD-MISSES Node load misses
cpu::process::node-stores	perf::NODE-STORES Node store accesses
cpu::process::node-store-misses	perf::NODE-STORE-MISSES Node store misses
cpu::process::node-prefetches	perf::NODE-PREFETCHES Node prefetch accesses
cpu::process::node-prefetch-misses	perf::NODE-PREFETCH-MISSES Node prefetch misses
cpu::process::uops-retired	UOPS_RETIRED Cycles Uops are being retired (Precise Event)
cpu::process::memory-stores-retired	MEM_INST_RETIRED:STORES Instructions retired which contain a store (Precise Event)
cpu::process::memory-loads-retired	MEM_INST_RETIRED:LOADS Instructions retired which contain a load (Precise Event)
cpu::process::thread-active-cycles	THREAD_ACTIVE Cycles thread is active
cpu::process::io-transactions	IO_TRANSACTIONS I/O transactions
cpu::process::offcore-requests	OFFCORE_REQUESTS:ANY Offcore requests

kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
cpu::process::misaligned-memory-accesses	MISALIGN_MEMORY Misaligned accesses
cpu::process::lsd-overflow	LSD_OVERFLOW Number of loops that cannot stream from the instruction queue.
cpu::process::machine-clear-cycles	MACHINE_CLEARS:CYCLES Cycles machine clear is asserted
cpu::process::offcore-responses	OFFCORE_RESPONSE_0:ANY_REQUEST:ANY_RESPONSE Offcore response 0 Request: combination of all requests umasks Response: combination of all response umasks
cpu::process::offcore -responses-cache	OFFCORE_RESPONSE_0:ANY_REQUEST:LOCAL_CACHE Offcore response 0 Request: combination of all requests umasks Response: any local (core and socket) caches
	kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
cpu::process::offcore -responses-local-dram	OFFCORE_RESPONSE_0:ANY_REQUEST: LOCAL_DRAM_AND_REMOTE_CACHE_HIT Offcore response 0 Request: combination of all requests umasks Response: counts L3 Miss: local home requests that missed the L3 cache and were serviced by local DRAM or a remote cache
cpu::process::offcore -responses-remote-dram	OFFCORE_RESPONSE_0:ANY_REQUEST:REMOTE_DRAM Offcore response 0 Request: combination of all requests umasks Response: counts L3 Miss: remote home requests that missed the L3 cache and were serviced by remote DRAM
cpu::process::offcore -responses-non-dram	OFFCORE_RESPONSE_0:ANY_REQUEST:NON_DRAM Offcore response 0 Request: combination of all requests umasks Response: Non-DRAM requests that were serviced by IOH
libvirt	
kontynuacja na następnej stronie	

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
stat::domain::cpu-time	virDomainGetCPUStats > cpu_time cpu usage (sum of both vcpu and hypervisor usage) in nanoseconds, as a ullong
stat::domain::user-time	virDomainGetCPUStats > user_time cpu time charged to user instructions in nanoseconds, as a ullong
stat::domain::system-time	virDomainGetCPUStats > system_time cpu time charged to system instructions in nanoseconds, as a ullong
memory::domain::balloon-size	virDomainMemoryStats > VIR_DOMAIN_MEMORY_STAT_ACTUAL_BALLOON Current balloon value (in kB).
memory::domain::resident-set-size	virDomainMemoryStats > VIR_DOMAIN_MEMORY_STAT_RSS Resident Set Size of the process running the domain.
io::domain::bytes-read	virDomainBlockStatsFlags > rd_bytes The total number of read bytes of the block device.
kontynuacja na następnej stronie	

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
io::domain::read-requests	virDomainBlockStatsFlags > rd_operations The total read requests of the block device.
io::domain::read-time	virDomainBlockStatsFlags > rd_total_times The total time spend on cache reads in nano-seconds of the block device.
io::domain::bytes-written	virDomainBlockStatsFlags > wr_bytes The total number of write bytes of the block device.
io::domain::write-requests	virDomainBlockStatsFlags > wr_operations The total write requests of the block device.
io::domain::write-time	virDomainBlockStatsFlags > wr_total_times The total time spent on cache writes in nano-seconds of the block device.
io::domain::flush-requests	virDomainBlockStatsFlags > flush_operations The total flush requests of the block device.
io::domain::flush-time	virDomainBlockStatsFlags > flush_total_times The total time spent on cache flushing in nano-seconds of the block device.
network::domain::bytes-received	virDomainInterfaceStats > rx_bytes

kontynuacja na następnej stronie

Tabela A.1 – kontynuacja z poprzedniej strony

Parametr	Opis
network::domain::packets-received	virDomainInterfaceStats > rx_packets
network::domain::bytes-sent	virDomainInterfaceStats > tx_bytes
network::domain::packets-sent	virDomainInterfaceStats > tx_packets

Dodatek B

Metryki gromadzone w fazie II eksperymentów

Lista metryk wraz z opisem gromadzona przez agenty monitorujące w II fazie eksperymentów.

Tabela B.1: Lista metryk gromadzonych w II fazie eksperymentów

Parametr	Opis
/proc/stat	
system.processes.running	Number of processes in runnable state.
system.processes.blocked	Number of processes blocked waiting for I/O to complete.
system.context.switch.rate	The number of context switches that the system underwent.
system.interrupt.rate	The total of all interrupts serviced including unnumbered architecture specific interrupts.
processor.time.user	Time spent in user mode.
processor.time.nice	Time spent in user mode with low priority (nice).
processor.time.system	Time spent in system mode.
processor.time.idle	Time spent in the idle task. This value should be USER_HZ times the second entry in the /proc/uptime pseudo-file.
processor.time.iowait	Time waiting for I/O to complete.
processor.time.irq	Time servicing interrupts.
processor.time.softirq	Time servicing softirqs.
processor.time.steal	Stolen time, which is the time spent in other operating systems when running in a virtualized environment.

kontynuacja na następnej stronie

Tabela B.1 – kontynuacja z poprzedniej strony

Parametr	Opis
processor.time.guest	Time spent running a virtual CPU for guest operating systems under the control of the Linux kernel.
processor.time.guest.nice	Time spent running a niced guest (virtual CPU for guest operating systems under the control of the Linux kernel).
/proc/meminfo	
memory.used	MemTotal - MemFree MemTotal - Total usable RAM (i.e., physical RAM minus a few reserved bits and the kernel binary code). MemFree - The sum of LowFree+HighFree.
memory.cached	In-memory cache for files read from the disk (the page cache). Doesn't include SwapCached.
swap.used	SwapTotal – SwapFree SwapTotal - Total amount of swap space available. SwapFree - Amount of swap space that is currently unused.
swap.cached	Memory that once was swapped out, is swapped back in but still also is in the swap file. (If memory pressure is high, these pages don't need to be swapped out again because they are already in the swap file. This saves I/O.)

kontynuacja na następnej stronie

Tabela B.1 – kontynuacja z poprzedniej strony

Parametr	Opis
memory.active	Memory that has been used more recently and usually not reclaimed unless absolutely necessary.
memory.inactive	Memory which has been less recently used. It is more eligible to be reclaimed for other purposes.
/proc/vmstat	
memory.page.in.rate	Number of pageins.
memory.page.out.rate	Number of pageouts.
memory.page.fault.rate	Number of page faults.
memory.page.faults.major.rate	Number of major page faults.
memory.page.free.rate	Number of page frees.
memory.page.activate.rate	Number of page activations.
memory.page.deactivate.rate	Number of page deactivations.
Intel Performance Counter Monitor	
processor.instructions.retired.rate	The number of retired instructions. getInstructionsRetired()

kontynuacja na następnej stronie

Tabela B.1 – kontynuacja z poprzedniej strony

Parametr	Opis
processor.cycles.reference.rate	The number of reference clock cycles while clock signal on the core is running. The reference clock operates at a fixed frequency, irrespective of core frequency changes due to performance state transitions. See Intel(r) Software Developer's Manual for more details. getRefCycles()
processor.frequency.relative	Average core frequency also taking Intel Turbo Boost technology into account. getRelativeFrequency()
processor.frequency.active.relative	Average core frequency when not in powersaving C0-state (also taking Intel Turbo Boost technology into account). getActiveRelativeFrequency()
processor.cache.l2.hit.rate	Number of L2 cache hits. getL2CacheHits()
processor.cache.l2.miss.rate	Number of L2 cache misses. getL2CacheMisses()
processor.cache.l3.hit.rate	Total number of L3 cache hits. getL3CacheHits()
processor.cache.l3.hit.snoop.rate	Number of L3 cache hits where snooping in sibling L2 caches had to be done. getL3CacheHitsSnoop()
processor.cache.l3.miss.rate	Number of L3 cache misses. getL3CacheMisses()

kontynuacja na następnej stronie

Tabela B.1 – kontynuacja z poprzedniej strony

Parametr	Opis
memory.read.rate	Number of bytes read from DRAM memory controllers. getBytesReadFromMC()
memory.write.rate	Number of bytes written to DRAM memory controllers. getBytesWrittenToMC()
memory.io.rate	Number of bytes of read/write requests from all IO sources. getIORequestBytesFromMC()
processor.power	Joules consumed by processor (excluding DRAM). getConsumedJoules()
memory.power	Joules consumed by DRAM. getDRAMConsumedJoules()
libvirt	
storage.read.rate	Count of read bytes.
storage.read.time	Total time read operations took.
storage.read.operations.rate	Count of read operations.
storage.write.rate	Count of written bytes.
storage.write.time	Total time write operations took.
storage.write.operations.rate	Count of write operations.

kontynuacja na następnej stronie

Tabela B.1 – kontynuacja z poprzedniej strony

Parametr	Opis
storage.flush.time	Total time flush operations took.
storage.flush.operations.rate	Count of flush operations.
network.receive.rate	Bytes received.
network.receive.packets.rate	Packets received.
network.send.rate	Bytes sent.
network.send.packets.rate	Packets sent.

Bibliografia

[1] Mohammed Abdullahi, Md Asri Ngadi i in. „Symbiotic organism search optimization based task scheduling in cloud computing environment". W: *Future Generation Computer Systems* 56 (2016), s. 640–650.

[2] Mohiuddin Ahmed, Raihan Seraj i Syed Mohammed Shamsul Islam. „The k-means Algorithm: A Comprehensive Survey and Performance Evaluation". W: *Electronics* 9.8 (2020). ISSN: 2079-9292.

[3] Klaithem Al Nuaimi i in. „A survey of load balancing in cloud computing: Challenges and algorithms". W: *2012 second symposium on network cloud computing and applications*. IEEE. 2012, s. 137–142.

[4] Ahmed Ali-Eldin i in. „Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control". W: *Proceedings of the 3rd Workshop on Scientific Cloud Computing*. ScienceCloud '12. Delft, The Netherlands: Association for Computing Machinery, 2012, s. 31–40. ISBN: 9781450313407. DOI: 10.1145/2287036.2287044.

[5] Sahar Arshad i in. „A survey of Cloud computing variable pricing models". W: *Evaluation of Novel Approaches to Software Engineering (ENASE), 2015 International Conference on*. IEEE. 2015, s. 27–32.

[6] AR. Arunarani, D. Manjula i Vijayan Sugumaran. „Task scheduling techniques in cloud computing: A literature survey". W: *Future Generation Computer Systems* 91 (2019), s. 407–415. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.09.014.

[7] AI Awad, NA El-Hefnawy i HM Abdel_kader. „Enhanced particle swarm optimization for task scheduling in cloud computing environments". W: *Procedia Computer Science* 65 (2015), s. 920–929.

[8] Nidhi Bansal, Amit Awasthi i Shruti Bansal. „Task scheduling algorithms with multiple factor in cloud computing environment". W: *Information Systems Design and Intelligent Applications: Proceedings of Third International Conference INDIA 2016, Volume 1*. Springer. 2016, s. 619–627.

[9] Nidhi Bansal i in. „Cost performance of QoS Driven task scheduling in cloud computing". W: *Procedia Computer Science* 57 (2015), s. 126–130. ISSN: 18770509. DOI: 10.1016/j.procs.2015.07.384.

[10] Rabindra K Barik i in. „Performance analysis of virtual machines and containers in cloud computing". W: *2016 international conference on computing, communication and automation (iccca)*. IEEE. 2016, s. 1204–1210.

[11] Michael A Benjamin, Robert A Rigby i D Mikis Stasinopoulos. „Generalized autoregressive moving average models". W: *Journal of the American Statistical association* 98.461 (2003), s. 214–223.

[12] Sushil Bhardwaj, Leena Jain i Sandeep Jain. „Cloud computing: A study of infrastructure as a service (IAAS)". W: *International Journal of engineering and information Technology* 2.1 (2010), s. 60–63.

[13] Leon Bottou. „Large-Scale Machine Learning with Stochastic Gradient Descent". W: *Proc. of COMPSTAT* (sty. 2010). DOI: 10.1007/978-3-7908-2604-3_16.

[14] Rodrigo N. Calheiros i in. „Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS". W: *IEEE Transactions on Cloud Computing* 3.4 (2015), s. 449–458. DOI: 10.1109/TCC.2014.2350475.

[15] Yang Cao, CheulWoo Ro i JianWei Yin. „Comparison of job scheduling policies in cloud computing". W: *Future Information Communication Technology and Applications: ICFICE 2013* (2013), s. 81–87.

[16] Goncalo Carvalho i in. „Edge computing: current trends, research challenges and future directions". W: *Computing* 103 (5 maj 2021), s. 993–1023. ISSN: 0010-485X. DOI: 10.1007/s00607-020-00896-5.

[17] Deji Chen, A.K. Mok i Tei-Wei Kuo. „Utilization bound revisited". W: *IEEE Transactions on Computers* 52.3 (2003), s. 351–361. DOI: 10.1109/TC.2003.1183949.

[18] Shi Chen, Jie Wu i Zhihui Lu. „A cloud computing resource scheduling policy based on genetic algorithm with multiple fitness". W: *2012 IEEE 12th International Conference on Computer and Information Technology*. IEEE. 2012, s. 177–184.

[19] Vidyanand Choudhary. „Comparison of software quality under perpetual licensing and software as a service". W: *Journal of Management Information Systems* 24.2 (2007), s. 141–165.

[20] Essential Characteristics of Cloud Computing. https://comparecloud.in. [on-line; dostęp: 14 grudnia 2022].

[21] Paweł Czarnul. *Model of a computational node in a cloud evaluation of hardware metrics*. Spraw. tech. 1-26. CI TASK, 2015.

[22] Manoranjan Dash, Hua Liu i Jun Yao. „Dimensionality reduction of unsupervised data". W: *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*. IEEE. 1997, s. 532–539.

[23] Nicola Dimitri. „Pricing cloud IaaS computing services". W: *Journal of Cloud Computing* 9 (1 grud. 2020), s. 14. ISSN: 2192-113X. DOI: 10.1186/s13677-020-00161-2.

[24] Youwei Ding i in. „Energy efficient scheduling of virtual machines in cloud with deadline constraint". W: *Future Generation Computer Systems* 50 (2015), s. 62–74.

[25] Yvonne Dittrich. „What does it mean to use a method? Towards a practice theory for software engineering". W: *Information and Software Technology* 70 (2016), s. 220–231.

[26] Ziqian Dong, Ning Liu i Roberto Rojas-Cessa. „Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers". W: *Journal of Cloud Computing* 4.1 (2015), s. 1–14.

[27] Hancong Duan i in. „Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems". W: *Future Generation Computer Systems* 74 (2017), s. 142–150.

[28] „Dynamic service migration and workload scheduling in edge-clouds". W: *Performance Evaluation* 91 (2015). Special Issue: Performance 2015, s. 205–228. ISSN: 0166-5316. DOI: https://doi.org/10.1016/j.peva.2015.06.013.

[29] Erik Elmroth i in. „Accounting and Billing for Federated Cloud Infrastructures". W: IEEE, sierp. 2009, s. 268–275. ISBN: 978-0-7695-3766-5. DOI: 10.1109/GCC.2009.37.

[30] Hadi Esmaeilzadeh i in. „Power Limitations and Dark Silicon Challenge the Future of Multicore". W: *ACM Trans. Comput. Syst.* 30.3 (sierp. 2012). ISSN: 0734-2071. DOI: 10.1145/2324876.2324879.

[31] I. Foster i in. „Cloud Computing and Grid Computing 360-Degree Compared". W: *2008 Grid Computing Environments Workshop*. 2008, s. 1–10. DOI: 10.1109/GCE.2008.4738445.

[32] Armando Fox i in. „Above the clouds: A Berkeley view of cloud computing". W: *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28.13 (2009), s. 2009.

[33] Archana Ganapathi i in. „Statistics-driven workload modeling for the Cloud". W: *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. 2010, s. 87–92. DOI: 10.1109/ICDEW.2010.5452742.

[34] Jiechao Gao, Haoyu Wang i Haiying Shen. „Machine Learning Based Workload Prediction in Cloud Computing". W: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, s. 1–9. DOI: 10.1109/ICCCN49398.2020.9209730.

[35] Jose Maria Garcia i in. „A Flexible Billing Life Cycle for Cloud Services Using Augmented Customer Agreements". W: *IEEE Access* 9 (2021), s. 44374–44389. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3066443.

[36] Zhenhuan Gong, Xiaohui Gu i John Wilkes. „PRESS: PRedictive Elastic ReSource Scaling for cloud systems". W: *2010 International Conference on Network and Service Management*. 2010, s. 9–16. DOI: 10.1109/CNSM.2010.5691343.

[37] Sumit Goyal. „Public vs private vs hybrid vs community-cloud computing: a critical review". W: *International Journal of Computer Network and Information Security* 6.3 (2014), s. 20–29.

[38] Albert Greenberg i in. „Towards a next Generation Data Center Architecture: Scalability and Commoditization". W: *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*. PRESTO '08. Seattle, WA, USA: Association for Computing Machinery, 2008, s. 57–62. ISBN: 9781605581811. DOI: 10.1145/1397718.1397732.

[39] Punit Gupta i Satya Prakash Ghrera. „Load and fault aware honey bee scheduling algorithm for cloud infrastructure". W: *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014: Volume 2*. Springer. 2015, s. 135–143.

[40] Jiangshui Hong i in. „An Overview of Multi-cloud Computing". W: *Web, Artificial Intelligence and Network Applications*. Red. Leonard Barolli i in. Cham: Springer International Publishing, 2019, s. 1055–1068. ISBN: 978-3-030-15035-8. DOI: 10.1007/978-3-030-15035-8_103.

[41] Seyedmehdi Hosseinimotlagh, Farshad Khunjush i Rashidaldin Samadzadeh. „SEATS: smart energy-aware task scheduling in real-time cloud computing". W: *The Journal of Supercomputing* 71 (2015), s. 45–66.

[42] Yicheng Huang, An Vu Tran i Ye Wang. „A Workload Prediction Model for Decoding Mpeg Video and Its Application to Workload-Scalable Transcoding". W: *Proceedings of the 15th ACM International Conference*

on Multimedia. MM '07. Augsburg, Germany: Association for Computing Machinery, 2007, s. 952–961. ISBN: 9781595937025. DOI: 10.1145/1291233.1291443.

[43] Sadeka Islam i in. „Empirical prediction models for adaptive resource provisioning in the cloud". W: *Future Generation Computer Systems* 28 (1 sty. 2012), s. 155–162. ISSN: 0167739X. DOI: 10.1016/j.future.2011.05.027.

[44] RK Jena. „Multi objective task scheduling in cloud environment using nested PSO framework". W: *Procedia Computer Science* 57 (2015), s. 1219–1227.

[45] Brendan Jennings i Rolf Stadler. „Resource management in clouds: Survey and research challenges". W: *Journal of Network and Systems Management* 23.3 (2015), s. 567–619.

[46] Hejun Jiao i in. „Immune optimization of task scheduling on multidimensional QoS constraints". W: *Cluster Computing* 18 (2015), s. 909–918.

[47] Hai Jin i in. „Towards optimized fine-grained pricing of IaaS cloud platform". W: *IEEE Transactions on cloud Computing* 3.4 (2015), s. 436–448.

[48] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[49] Ashish Joshi i in. „Improved Security and Privacy in Cloud Data Security and Privacy: Measures and Attacks". W: *2022 International Conference on Fourth Industrial Revolution Based Technology and Practices (ICFIRTP)*. 2022, s. 230–233. DOI: 10.1109/ICFIRTP56122.2022.10063186.

[50] Ritu Kapur. „A cost effective approach for resource scheduling in cloud computing". W: *2015 International Conference on Computer, Communication and Control (IC4)*. IEEE. 2015, s. 1–6.

[51] Mayanka Katyal i Atul Mishra. „A comparative study of load balancing algorithms in cloud computing environment". W: *arXiv preprint arXiv:1403.6918* (2014).

[52] Yousef Khalidi. „Building a Cloud Computing Platform for New Possibilities". W: *Computer* 44.3 (2011), s. 29–34. DOI: 10.1109/MC.2011.63.

[53] George Kousiouris i in. „Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in Cloud platforms". W: *Future Generation Computer Systems* 32 (2014). Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures, s. 27–40. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2012.05.009.

[54] Zhanibek Kozhirbayev i Richard O. Sinnott. „A performance comparison of container-based technologies for the Cloud". W: *Future Generation Computer Systems* 68 (mar. 2017), s. 175–182. ISSN: 0167739X. DOI: 10.1016/j.future.2016.08.025.

[55] Anne Krampe, Joachim Lepping i Wiebke Sieben. „A Hybrid Markov Chain Model for Workload on Parallel Computers". W: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: Association for Computing Machinery, 2010, s. 589–596. ISBN: 9781605589428. DOI: 10.1145/1851476.1851563.

[56] H. Krawczyk. *Efektywne wykorzystanie zasobów chmury obliczeniowej*. Gdańsk: Wydawnictwo CI TASK, 2017. ISBN: 978-83-947070-0-2.

[57] Henryk Krawczyk i Piotr Orzechowski. „Algorytm mrówkowy do zarzadzania zasobami sprztowymi chmury obliczeniowej w przypadku różnych kategorii usług". W: unknown, 2022, s. 225–245.

[58] Joseph B Kruskal i Myron Wish. *Multidimensional scaling*. T. 11. Sage, 1978.

[59] Dinesh Kumar i Zahid Raza. „A PSO based VM resource scheduling model for cloud computing". W: *2015 IEEE international conference on computational intelligence & communication technology*. IEEE. 2015, s. 213–219.

[60] Sajib Kundu i in. „Modeling Virtualized Applications Using Machine Learning Techniques". W: *SIGPLAN Not.* 47.7 (mar. 2012), s. 3–14. ISSN: 0362-1340.

[61] Sajib Kundu i in. „Modeling virtualized applications using machine learning techniques". W: *ACM SIGPLAN Notices*. T. 47. 7. ACM. 2012, s. 3–14.

[62] Ewnetu Bayuh Lakew i in. „A Synchronization Mechanism for Cloud Accounting Systems". W: IEEE, wrz. 2014, s. 111–120. ISBN: 978-1-4799-5841-2. DOI: 10.1109/ICCAC.2014.11.

[63] Atul Vikas Lakra i Dharmendra Kumar Yadav. „Multi-objective tasks scheduling algorithm for cloud computing throughput optimization". W: *Procedia Computer Science* 48 (2015), s. 107–113.

[64] C. L. Liu i James W. Layland. „Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". W: *Journal of the ACM* (sty. 1973). DOI: 10.1145/321738.321743.

[65] Haikun Liu i in. „Performance and Energy Modeling for Live Migration of Virtual Machines". W: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. HPDC '11. San Jose, California, USA: ACM, 2011, s. 171–182. ISBN: 978-1-4503-0552-5. DOI: 10.1145/1996130.1996154.

[66] Haikun Liu i in. „Performance and energy modeling for live migration of virtual machines". W: *Cluster computing* 16.2 (2013), s. 249–264.

[67] Tania Lorido-Botran i in. „An unsupervised approach to online noisy-neighbor detection in cloud data centers". W: *Expert Systems with Applications* 89 (2017), s. 188–204. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.07.038.

[68] J. M. López, J. L. Dıaz i D. F. Garcıa. „Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems". W: *Real-Time Syst.*

28.1 (paź. 2004), s. 39–68. ISSN: 0922-6443. DOI: 10.1023/B:TIME.0000033378.56741.14.

[69] Hao Lu i in. „Optimal pricing of multi-model hybrid system for PaaS cloud computing". W: *Cloud and Service Computing (CSC), 2012 International Conference on*. IEEE. 2012, s. 227–231.

[70] Paweł Lubomski, Andrzej Kalinowski i Henryk Krawczyk. „Multi-level Virtualization and Its Impact on System Performance in Cloud Computing". W: *Computer Networks*. 2016.

[71] Juntao Ma i in. „A novel dynamic task scheduling algorithm based on improved genetic algorithm in cloud computing". W: *Wireless Communications, Networking and Applications: Proceedings of WCNA 2014*. Springer. 2016, s. 829–835.

[72] Syed Hamid Hussain Madni i in. „Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges and opportunities". W: *Journal of Network and Computer Applications* 68 (czer. 2016), s. 173–200. ISSN: 10848045. DOI: 10.1016/j.jnca.2016.04.016.

[73] Simon J. Malkowski i in. „Automated Control for Elastic N-Tier Workloads Based on Empirical Modeling". W: *Proceedings of the 8th ACM International Conference on Autonomic Computing*. ICAC '11. Karlsruhe, Germany: Association for Computing Machinery, 2011, s. 131–140. ISBN: 9781450306072. DOI: 10.1145/1998582.1998604.

[74] Geetanshu Mangal i in. „Flexible Cloud Computing by Integrating Public-Private Clouds Using OpenStack". W: *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. 2015, s. 146–152. DOI: 10.1109/CCEM.2015.26.

[75] Mohammad Masdari, Sayyid Shahab Nabavi i Vafa Ahmadi. „An overview of virtual machine placement schemes in cloud computing". W: *Journal of Network and Computer Applications* 66 (maj 2016), s. 106–127. ISSN: 10848045. DOI: 10.1016/j.jnca.2016.01.011.

[76] Ahmed Mihoob, Carlos Molina-Jimenez i Santosh Shrivastava. „A Case for Consumer150;centric Resource Accounting Models". W: IEEE, lip. 2010, s. 506–512. ISBN: 978-1-4244-8207-8. DOI: 10.1109/CLOUD.2010.44.

[77] Asit K. Mishra i in. „Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters". W: *SIGMETRICS Perform. Eval. Rev.* 37.4 (mar. 2010), s. 34–41. ISSN: 0163-5999. DOI: 10.1145/1773394.1773400.

[78] Shivangi Nigam i Abhishek Bajpai. „An Optimal Resource Provisioning Algorithm for Cloud Computing Environment". W: *Oriental journal of computer science and technology* 10 (2 czer. 2017), s. 371–384. ISSN: 09746471. DOI: 10.13005/ojcst/10.02.17.

[79] Michał Nykiel. „Optymalizacja kosztu działania aplikacji na urzadzeniach mobilnych wspomaganych przez chmur obliczeniowa". Prac. dokt. Politechnika Gdańska, 2018.

[80] Isaac Odun-Ayo, Blessing Udemezue i Abiodun Kilanko. „Cloud Service Level Agreements and Resource Management". W: *Advances in Science, Technology and Engineering Systems Journal* 4 (sty. 2019), s. 228–236. DOI: 10.25046/aj040230.

[81] Piotr Orzechowski. „Complementary oriented allocation algorithm for cloud computing". W: *TASK Quarterly* 21.4 (2017), s. 395–403.

[82] Piotr Orzechowski i Henryk Krawczyk. „General provisioning strategy for local specialized cloud computing environments". artykuł przyjęty na konferencję: 18th International Conference on Dependability of Computer Systems. 2023.

[83] Piotr Orzechowski i in. „Automatic Discovery of IaaS Cloud Workload Types". W: *Proceedings of 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016)*. 2016.

[84] Piotr Orzechowski i in. „Categorization of Cloud Workload Types with Clustering". W: *Proceedings of the International Conference on Signal, Networks, Computing, and Systems*. Red. Daya K. Lobiyal i in. New Delhi: Springer India, 2017, s. 303–313. ISBN: 978-81-322-3592-7.

[85] Jeng-Shyang Pan i in. „Interaction artificial bee colony based load balance method in cloud computing". W: *Genetic and Evolutionary Computing: Proceeding of the Eighth International Conference on Genetic and Evolutionary Computing, October 18-20, 2014, Nanchang, China*. Springer. 2015, s. 49–57.

[86] Jerzy Proficz. „Zarzadzanie zasobami obliczeniowymi w klastrowym środowisku przetwarzania strumieni multimedialnych". Prac. dokt. Politechnika Gdańska, 2012.

[87] Deepak Puthal i in. „Cloud Computing Features, Issues, and Challenges: A Big Picture". W: IEEE, sty. 2015, s. 116–123. ISBN: 978-1-4799-7548-8. DOI: 10.1109/CINE.2015.31.

[88] Xuan Qi, Dakai Zhu i Hakan Aydin. „Cluster scheduling for real-time systems: utilization bounds and run-time overhead". W: *Real-Time Systems* 47 (3 maj 2011), s. 253–284. ISSN: 0922-6443. DOI: 10.1007/s11241-011-9121-1.

[89] Kimmo E. E. Raatikainen. „Cluster Analysis and Workload Classification". W: *SIGMETRICS Perform. Eval. Rev.* 20.4 (maj 1993), s. 24–30. ISSN: 0163-5999. DOI: 10.1145/155775.155781.

[90] Indukuri R Krishnam Raju i in. „Deadline aware two stage scheduling algorithm in cloud computing". W: *Indian Journal of Science and Technology* 9.4 (2016), s. 1–10.

[91] Omer Rana. „The Costs of Cloud Migration". W: *IEEE Cloud Computing* 1.01 (maj 2014), s. 62–65. ISSN: 2372-2568. DOI: 10.1109/MCC.2014.24.

[92] Jia Rao i in. „Self-adaptive Provisioning of Virtualized Resources in Cloud Computing". W: *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '11. San Jose, California, USA: ACM, 2011, s. 129–130. ISBN: 978-1-4503-0814-4. DOI: 10.1145/1993744.1993790.

[93] Jia Rao i in. „VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration". W: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC '09. Barcelona, Spain: ACM, 2009, s. 137–146. ISBN: 978-1-60558-564-2. DOI: 10.1145/1555228.1555263.

[94] KR Remesh Babu i Philip Samuel. „Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud". W: *Innovations in Bio-Inspired Computing and Applications: Proceedings of the 6th International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA 2015) held in Kochi, India during December 16-18, 2015*. Springer. 2016, s. 67–78.

[95] J.W. Rittinghouse i J.F. Ransome. *Cloud Computing: Implementation, Management, and Security*. CRC Press, 2009. DOI: https://doi.org/10.1201/9781439806814.

[96] May Al-Roomi i in. „Cloud computing pricing models: a survey". W: *International Journal of Grid and Distributed Computing* 6.5 (2013), s. 93–106.

[97] Supreet Sahi i V.S.Dhaka V.S.Dhaka. „A Review on Workload Prediction of Cloud Services". W: *International Journal of Computer Applications* 109 (sty. 2015), s. 1–4. DOI: 10.5120/19213-0911.

[98] Vyas Sekar i Petros Maniatis. „Verifiable resource accounting for cloud computing services". W: ACM, paź. 2011, s. 21–26. ISBN: 9781450310048. DOI: 10.1145/2046660.2046666.

[99] Mohamed Abu Sharkh, Abdelkader Ouda i Abdallah Shami. „A resource scheduling model for cloud computing data centers". W: *2013 9th In-*

ternational Wireless Communications and Mobile Computing Conference (IWCMC). IEEE. 2013, s. 213–218.

[100] Bhanu Sharma i in. „Pricing Cloud Compute Commodities: A Novel Financial Economic Model". W: IEEE, maj 2012, s. 451–457. ISBN: 978-1-4673-1395-7. DOI: 10.1109/CCGrid.2012.126.

[101] Sukhpal Singh i Inderveer Chana. „QRSF: QoS-aware resource scheduling framework in cloud computing". W: *The Journal of Supercomputing* 71 (2015), s. 241–292.

[102] Zafar U. Singhera. „A Workload Model for Topic-Based Publish/Subscribe Systems". W: *Companion to the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*. OOPSLA Companion '08. Nashville, TN, USA: Association for Computing Machinery, 2008, s. 703–712. ISBN: 9781605582207. DOI: 10.1145/1449814.1449824.

[103] A. Snavely i in. „A Framework for Performance Modeling and Prediction". W: *SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. 2002, s. 21–21. DOI: 10.1109/SC.2002.10004.

[104] George Suciu, Elena G. Ularu i Razvan Craciunescu. „Public versus private cloud adoption - A case study based on open source cloud platforms". W: *2012 20th Telecommunications Forum (TELFOR)*. 2012, s. 494–497. DOI: 10.1109/TELFOR.2012.6419255.

[105] Yifan Sun i in. „Summarizing CPU and GPU design trends with product data". W: *arXiv preprint arXiv:1911.11313* (2019).

[106] Sandeep G Sutar, Pallavi J Mali i Amruta Y More. „Resource utilization enhancemnet through live virtual machine migration in cloud using ant colony optimization algorithm". W: *International Journal of Speech Technology* 23.1 (2020), s. 79–85.

[107] Julian Szymański. *Opracowanie koncepcji komponentu rekomendacji z wykorzystaniem nowych modeli i algorytmów*. Spraw. tech. 1-22. CI TASK, 2015.

[108] David Talby, Dror G. Feitelson i Adi Raveh. „A Co-Plot Analysis of Logs and Models of Parallel Workloads". W: 17.3 (lip. 2007), 12–es. ISSN: 1049-3301. DOI: 10.1145/1243991.1243993.

[109] Wenting Tang i in. „Modeling and generating realistic streaming media server workloads". W: *Computer Networks* 51.1 (2007), s. 336–356. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2006.05.003.

[110] Gerald Tesauro i in. „On the use of hybrid reinforcement learning for autonomic resource allocation". W: *Cluster Computing* 10 (3 sierp. 2007), s. 287–299. ISSN: 1386-7857. DOI: 10.1007/s10586-007-0035-6.

[111] Theo Theunissen i Uwe Van Heesch. „Specification in Continuous Software Development". W: *Proceedings of the 22nd European Conference on Pattern Languages of Programs*. EuroPLoP '17. Irsee, Germany: Association for Computing Machinery, 2017. ISBN: 9781450348485. DOI: 10.1145/3147704.3147709.

[112] Mustafa M Tikir i in. „A genetic algorithms approach to modeling the performance of memory-bound computations". W: *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. 2007, s. 1–12. DOI: 10.1145/1362622.1362686.

[113] Leslie G. Valiant. „A bridging model for multi-core computing". W: *Journal of Computer and System Sciences* 77.1 (2011). Celebrating Karp's Kyoto Prize, s. 154–166. ISSN: 0022-0000. DOI: https://doi.org/10.1016/j.jcss.2010.06.012.

[114] Luis M. Vaquero i in. „A Break in the Clouds: Towards a Cloud Definition". W: *SIGCOMM Comput. Commun. Rev.* 39.1 (grud. 2008), s. 50–55. ISSN: 0146-4833. DOI: 10.1145/1496091.1496100.

[115] Michel Verleysen i Damien François. „The curse of dimensionality in data mining and time series prediction". W: *Computational Intelligence and Bioinspired Systems*. Springer, 2005, s. 758–770.

[116] Samuel Williams, Andrew Waterman i David Patterson. „Roofline". W: *Communications of the ACM* 52 (4 kw. 2009), s. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785.

[117] M. Witkowski i in. „Practical power consumption estimation for real life HPC applications". W: *Future Generation Computer Systems* 29.1 (2013). Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures, s. 208–217. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2012.06.003.

[118] Shiliang Wu, Hans Wortmann i Chee-wee Tan. „A pricing framework for software-as-a-service". W: *Innovative Computing Technology (INTECH), 2014 Fourth International Conference on*. IEEE. 2014, s. 152–157.

[119] Xing Wu, Wu Zhang i Dou Wanchun. „Pricing as a service: personalized pricing strategy in cloud computing". W: *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*. IEEE. 2012, s. 1119–1124.

[120] Ning Xie, Xuejie Zhang i Jixian Zhang. „A truthful auction-based mechanism for virtual resource allocation and pricing in clouds". W: *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. 2017, s. 578–582. DOI: 10.1109/CompComm.2017.8322611.

[121] Ying Yidu Xiong i Yan Yan Wu. „Cloud computing resource schedule strategy based on pso algorithm". W: *Applied Mechanics and Materials*. T. 513. Trans Tech Publ. 2014, s. 1332–1336.

[122] Lei Xu i in. „A Scalable Accounting Solution for Prepaid Services in Cloud Systems". W: IEEE, czer. 2012, s. 81–89. ISBN: 978-1-4673-3049-7. DOI: 10.1109/SCC.2012.40.

[123] Zhengqiu Yang i in. „Study on cloud resource allocation strategy based on particle swarm ant colony optimization algorithm". W: *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*. T. 01. 2012, s. 488–491. DOI: 10.1109/CCIS.2012.6664453.

[124] Dewei Yi i in. „New Driver Workload Prediction Using Clustering-Aided Approaches". W: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.1 (2019), s. 64–70. DOI: 10.1109/TSMC.2018.2871416.

[125] Jian Zhang i R.J. Figueiredo. „Application classification through monitoring and learning of resource consumption patterns". W: *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*. 2006. DOI: 10.1109/IPDPS.2006.1639378.

[126] Weizhan Zhang i in. „Workload modeling for virtual machine-hosted application". W: *Expert Systems with Applications* 42.4 (2015), s. 1835–1844. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/j.eswa.2014.10.012.

[127] Xiaoyi Zhang i Yongsheng Wang. „Research on prepaid account financing model based on embedded system and Internet of Things". W: *Microprocessors and Microsystems* 82 (kw. 2021), s. 103935. ISSN: 01419331. DOI: 10.1016/j.micpro.2021.103935.

[128] Xinkui Zhao i in. „Workload Classification Model for Specializing Virtual Machine Operating System". W: *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE. 2013, s. 343–350.

[129] Liyun Zuo i in. „A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing". W: *Ieee Access* 3 (2015), s. 2687–2699.

Referencje sieć Internet

[web1] The Performance Application Programming Interface (PAPI). https://icl.utk.edu/papi/. [dostęp: 3 kwietnia 2023].

[web2] *Ansible role docker*. https://projects.task.gda.pl/ansible-roles/docker. [dostęp: 14 kwietnia 2023].

[web3] *Ansible role iptables*. https://projects.task.gda.pl/ansible-roles/iptables. [dostęp: 14 kwietnia 2023].

[web4] *Ansible role networking*. https://projects.task.gda.pl/ansible-roles/networking. [dostęp: 14 kwietnia 2023].

[web5] *Ansible role ssh*. https://projects.task.gda.pl/ansible-roles/ssh. [dostęp: 14 kwietnia 2023].

[web6] *Ansible role system*. https://projects.task.gda.pl/ansible-roles/system. [dostęp: 14 kwietnia 2023].

[web7] *Ansible role system*. https://projects.task.gda.pl/ansible-roles/lldp. [dostęp: 14 kwietnia 2023].

[web8] *Ansible role system*. https://projects.task.gda.pl/ansible-roles/filesystem. [dostęp: 14 kwietnia 2023].

[web9] *Ansible software website*. https://www.ansible.com/. [dostęp: 12 marca 2023].

[web10] Benchmark. *Cloudsuite*. https://ecocloud.epfl.ch/research/sustainable-it-infrastructure/cloudsuite/. [dostęp: 3 kwietnia 2023].

[web11] Benchmark. *OLTPBench*. https://github.com/oltpbenchmark/oltpbench. [dostęp: 26 luty 2023].

[web12] Benchmark. *PUMA*. https://engineering.purdue.edu/~puma/pumabenchmarks.htm. [dostęp: 1 maja 2023].

[web13] High Performance Conjugate Gradients (HPCG) Benchmark. https://www.hpcg-benchmark.org/. [dostęp: 1 maja 2023].

[web14] HPC Challenge Benchmark. https://hpcchallenge.org/hpcc/. [dostęp: 1 maja 2023].

[web15] *Borg cluster traces from Google*. https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md. [dostęp: 4 kwietnia 2023].

[web16] Benjamin Cane. *Troubleshooting High I/O Wait in Linux*. http://bencane.com/2012/08/06/troubleshooting-high-io-wait-in-linux/. [dostęp: 15 stycznia 2023].

[web17] *Centrum Doskonałości Naukowej Infrastruktury Wytwarzania Aplikacji*. https://niwa.gda.pl/. [dostęp: 3 kwietnia 2023].

[web18] *Centrum Informatyczne Trójmiejskiej Akademickiej Sieci Komputerowej*. https://task.gda.pl/. [dostęp: 3 kwietnia 2023].

[web19] *Ceph-Ansible*. https://github.com/ceph/ceph-ansible. [dostęp: 13 kwietnia 2023].

[web20] *Ceph: an open-source, distributed storage system*. https://ceph.io/. [dostęp: 12 marca 2023].

[web21] *Cobbler*. https://cobbler.github.io/. [dostęp: 14 kwietnia 2023].

[web22] Darren Hoch. *Extreme Linux Performance Monitoring Part II*. http://www.ufsdump.org/papers/io-tuning.pdf. [dostęp: 16 marca 2021].

[web23] Jarrod. *Root Users*. http://www.rootusers.com/my-top-3-linux-commands-for-logging-problems/. [dostęp: 15 stycznia 2023].

[web24] *Kolla-Ansible*. https://www.openstack.org/software/releases/zed/components/kolla-ansible. [dostęp: 3 kwietnia 2023].

[web25] *Komponent Rekomendacji dla Inteligentnych Chmur Obliczeniowych*. https://krico.gda.pl/. [dostęp: 3 kwietnia 2023].

[web26] *MaaS*. https://maas.io/. [dostęp: 14 kwietnia 2023].

[web27] Memtier benchmark. https://github.com/RedisLabs/memtier_benchmark. [dostęp: 1 maja 2023].

[web28] CloudSim: A Framework For Modeling, Simulation Of Cloud Computing Infrastructures i Services. https://github.com/Cloudslab/cloudsim. [online; dostęp 16.05.2023].

[web29] National Institute of Standards and Technology. *The NIST Definition of Cloud Computing*. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. [Online; przeglądano 20-02-2017]. 2011.

[web30] *Open-Source Cloud Solutions: a License for Achieving Multi-Cloud Interoperability*. https://www.infopulse.com/blog/open-source-cloud-solutions. [dostęp: 3 kwietnia 2023].

[web31] *OpenStack Components*. https://www.openstack.org/software/releases/zed/components. [dostęp: 13 kwietnia 2023].

[web32] *OpenStack software website*. https://www.openstack.org/software/. [dostęp: 12 marca 2023].

[web33] *OpenStack Tempest*. https://www.openstack.org/software/releases/zed/components/tempest. [dostęp: 14 kwietnia 2023].

[web34] Felix Richter. *Big Three Dominate the Global Cloud Market*. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. [dostęp: dostęp: 3 maja 2023].

[web35] sysbench: Scriptable database i system performance benchmark. https://github.com/akopytov/sysbench. [dostęp: 1 maja 2023].

[web36] *Terraform software website*. https://www.terraform.io/. [dostęp: 12 marca 2023].

[web37] Siege: an http load tester i benchmarking utility. https://github.com/JoeDog/siege. [dostęp: 1 maja 2023].

[web38] thattommyhall. *Measuring Disk Usage in Linux (%iowait vs IOPS)*. http://www.thattommyhall.com/2011/02/18/iops-linux-iostat/. [dostęp: 15 stycznia 2023].

[web39] *Top Open Source Cloud Platforms and Solutions*. https://computingforgeeks.com/top-open-source-cloud-platforms-and-solutions/. [dostęp: 3 kwietnia 2023].

[web40] Bhavin Turakhia. *Understanding CPU Utilization and Optimization*. http://careers.directi.com/display/tu/Understanding+CPU+Utilization+and+Optimization. [dostęp: 12 lipca 2020].

[web41] WPMark Twenty Twelve. http://mtekk.us/archives/wordpress/wpmark-twenty-twelve/. [dostęp: 1 maja 2023].

[web42] *Vault*. https://www.vaultproject.io/.

[web43] Stephanie Vozza. *Redefining Workloads in Cloud Environments*. https://www.nutanix.com/theforecastbynutanix/technology/rethinking-cloud-workloads. [dostęp: dostęp: 3 maja 2023].