

Henryk KRAWCZYK
Politechnika Gdańska

OPTYMALNY DOBÓR INTELIGENTNYCH USŁUG W SCENARIUSZU DZIAŁANIA APLIKACJI CHMUROWEJ

Streszczenie. W pracy przeanalizowano problem składania aplikacji chmurowej z dostępnych usług, budowanych przy wykorzystaniu sztucznej inteligencji. Zaprezentowano platformę CAISE do wytwarzania takich usług oraz metodę ich doboru dla dobrze wyspecyfikowanej aplikacji. Problem ten sprowadzono do zadania optymalizacyjnego, dla którego dobrym rozwiązaniem jest algorytm mrówkowy. Proponowane kryterium optymalizacji uwzględnia zarówno jakość usług jak też ich efektywne wykonanie na platformie CAISE wykorzystując dynamiczne mechanizmy dostępne w K8s.

OPTIMAL SELECTION OF INTELLIGENT SERVICES IN A CLOUD APPLICATION OPERATIONAL SCENARIO

Summary. This paper analyzes the problem of composing a cloud application from available services built using artificial intelligence. It presents the CAISE platform for developing such services and a method for selecting them for well specified applications. The problem is formulated as an optimization task, for which the ant colony algorithm provides an effective solution. The proposed optimization criterion takes into account both service quality and efficient execution on the CAISE platform, leveraging dynamic mechanisms available in Kubernetes (K8s) scenario.

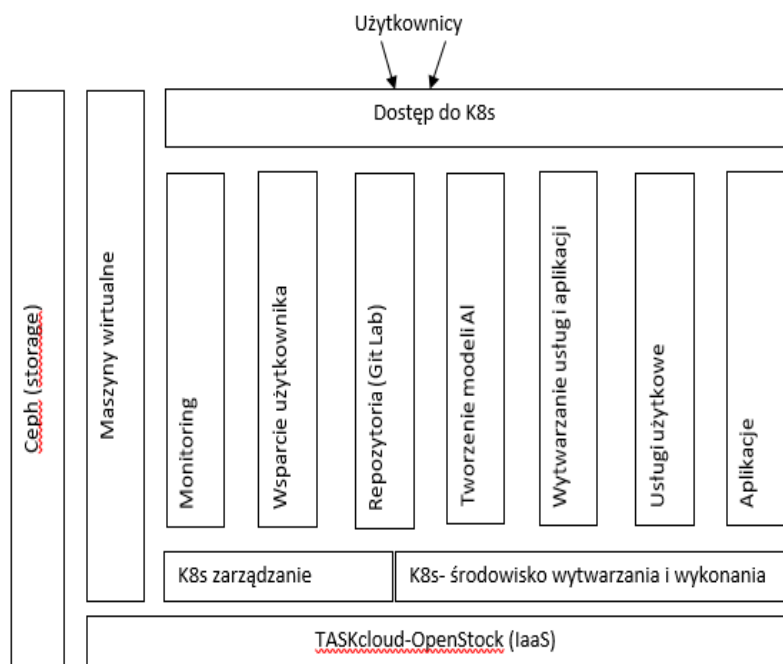
1. Wprowadzenie

Tradycyjne aplikacje (monolityczne) są opracowywane z reguły w postaci pojedynczego kodu, który jest implementowany i testowany zazwyczaj przez jeden zespół projektowy (deweloperski). Akceptowalny kod jest przekazywany zespołowi operacyjnemu w celu wdrożenia na wskazanym serwerze w postaci aplikacji użytkowej. Jeśli okaże się że aplikacja posiada błędy lub wymaga zmian to jest ponownie przekazywana zespołowi projektowemu, który rozwiązuje zasygnalizowane wady, wprowadza poprawki i przekazuje nową wersję zespołowi operacyjnemu w celu ponownego wdrożenia. Taki cykl postępowania może pojawiać się wielokrotnie. W przypadku złożonych aplikacji, w ten sposób budowane są jej podstawowe komponenty, które następnie są integrowane, testowane i wdrażane jako spójna aplikacja [14].

W przypadku aplikacji chmurowych wytwarzanie, testowanie i wdrażanie aplikacji jest realizowane przy wykorzystaniu możliwości warstwy PaaS (ang. *Platform*

as a Service). Deweloperzy oferują różnego typu takie platformy, a w nich odpowiednie usługi systemowe wspomagające proces wytwarzania i wdrażania aplikacji (np. Google App Engine). Pojedyncza aplikacja instalowana na serwerze, jest kompilowana z wielu małych elementów oprogramowania, które są zwane mikrousługami. Są one umieszczane w kontenerach zainstalowanych na różnych serwerach w chmurze. Mogą być oferowane nawet przez różnych dostawców. Komunikują się za pośrednictwem szybkich, bezpiecznych sieci, współdziałając ze sobą w celu wykonania wymaganych funkcji. Może istnieć wiele wariantów mikrousług o podobnej funkcjonalności, ale różnych parametrach jakościowych [2]. Użytkownik bądź projektant decyduje jaką mikrousługę z dostępnego zbioru wykorzystać w swojej aplikacji

Takie podejście ma wyraźne zalety w stosunku do podejścia monolitycznego. Mikrousługi mogą być tworzone, testowane i wdrażane niezależnie, co przyspiesza proces wytwarzania oprogramowania. Jeśli w danej mikrousłudze zostanie wykryta usterka, to może ona być testowana, ulepszana i wymieniona, bez wpływu na pozostałą część aplikacji. Jeśli mikrousługa jest przeciążona z powodu nieoczekiwanie zwiększonego użycia zasobów, jej kontener może być replikowany, tworząc co najmniej jedną identyczną instancję tej mikrousługi, zainstalowaną na innym serwerze, a istniejące obciążenie może być między tymi instancjami rozłożone. Pozwala to na eliminację wąskich gardeł wydajnościowych. Poza tym deweloperzy, rozwijający daną mikrousługę, mają możliwość wyboru języków programowania, narzędzi wytwarzania, czy architektur procesorów oraz środowiska wykonania, co ułatwia jej wytworzenie.



Rys. 1 Architektura platformy CAISE

W projekcie CAISE (ang. *Cloud Artificial Intelligence Service Engineering*) [7,8] realizowanym przez Politechnikę Gdańską w ramach europejskiego programu IPCEI-CIS, wykorzystano chmurę TASKcloud zbudowaną w CI TASK już ponad 10 lat temu. Na jej podstawie wytworzono platformę typu PaaS o tej samej nazwie co nazwa projektu, której podstawowa architektura jest przedstawiona na rysunku 1. Platforma

CAISE oferuje dwie kategorie usług chmurowych: użytkowe i systemowe. Usługi użytkowe dotyczą analizy treści trzech rodzajów dokumentów: tekstowych, obrazowych oraz hybrydowych zawierających różnego typu teksty jak i obrazy. Pozwalają na rozpoznanie i analizę różnego typu obiektów umieszczonych w dokumentach cyfrowych, określenie ich kategorii, a także zawartej w nich treści, i na tej podstawie podejmowanie odpowiednich decyzji. Na przykład na podstawie treści zawartej w dokumencie cyfrowym określa się jakie czynności powinny być podjęte, (np. gdzie ten dokument przekierować), jakie dalsze działania wskazać, bądź jakie kolejne dokumenty wygenerować. Wykorzystuje się do tego sztuczną inteligencję, koncentrując się na dokumentach pochodzących ze sektora publicznego, w szczególności medycyny, oświaty czy wymiaru sprawiedliwości.

Drugi rodzaj usług - usługi systemowe wspierają podstawową funkcjonalność platformy związaną z wytwarzaniem i wykonaniem usług użytkowych wraz z monitoringiem zdarzeń i pomiarem zużycia energii elektrycznej. Pozwalają na realizację czynności związanych z repozytoriami: danych, usług użytkowych oraz modeli sztucznej inteligencji. Kreują również środowisko dostrajania modeli do potrzeb sektora publicznego. Dzięki wykorzystaniu komputerów dużej mocy obliczeniowej (HPC) zastosowane są nowe rozwiązania przyspieszające realizację procesu uczenia czy analizy dużych zbiorów danych. Wykorzystane przy tym technologie dotyczą OpenStack oraz Kubernetes, które nadają się do realizacji usług na poziomie IaaS (usługi systemowe) oraz PaaS (usługi użytkowe). Z kolei integracja takich komponentów technologicznych jak Jupiter, MLflow czy Argo Workflow wspomaga realizację i wytwarzanie usług oraz umożliwia przeprowadzanie treningów i strojenie modeli. Poza tym usługi monitorowania funkcjonowania platformy wykorzystują takie komponenty technologiczne jak: Prometheus, Grafana czy Grafana Alloy. W projekcie dodatkowo zakłada się współpracę z firmami zagranicznymi uczestnikami programu IPCEI-CIS, jak również polskimi firmami IT pracującymi w sektorze publicznym. Firmy polskie wyłonione w postępowaniu przetargowym dokonują walidacji platformy CAISE poprzez realizację demonstratorów przypadków użycia, biznesowo istotnych dla tego sektora. Wytwarzane demonstratory wykorzystują właśnie dostępne na platformie CAISE usługi użytkowe. Komponują z nich odpowiedni potok czynności bądź bardziej złożony model reprezentowany przez graf wywoływanych usług. Co więcej różni dostawcy mogą oferować alternatywne zestawy usług do wykonania. Wówczas pojawia się problem odpowiedniego doboru takich usług, tak by budowany demonstrator spełniał określone wymagania użytkowników. Takie zadanie optymalizacyjne może być realizowane również na platformie CAISE i ten problem jest poniżej dyskutowany.

Istnieje więc pewna analogia budowy aplikacji użytkowych z dostępnych usług chmurowych do kreowania usług użytkowych z mikrousług. Jednak wykorzystywane mechanizmy w tych obu przypadkach są na ogół różne, gdyż funkcjonują na różnych poziomach abstrakcji. W artykule zostaną przeanalizowane możliwości rozwiązania tego problemu dla aplikacji kreowanych poprzez usługi użytkowe. Z uwagi na dynamikę działania chmury istotne jest uwzględnianie bieżącego stanu zaangażowania jej poszczególnych zasobów (węzłów). Z kolei każda z usług powinna charakteryzować się odpowiednią jakością, jak też odpowiednimi wartościami dotyczącymi czasów wykonania oraz zużycia zasobów. W następnym rozdziale przedstawiono możliwości

oprogramowania Kubernetes do rozwiązania tego problemu. W rozdziale 3 zaprezentowano możliwości uwzględnienia wymagań jakościowych w wyborze usług alternatywnych. W rozdziale 4 zaprezentowano definicję problemu doboru usług i podano metodę rozwiązania w oparciu o algorytmu mrówkowy. We wnioskach końcowych wskazano na dalsze udoskonalenie przedstawionej propozycji.

2. Możliwości Kubernetes

Usługa chmurowa to proces obliczeniowy, który wykonuje konkretne zadanie dla użytkownika. Ma ona najczęściej architekturę dwu lub trójwarstwową, wśród których wyróżnia się warstwę wizualizacji kształtującą interfejs użytkownika (ang. *frontend*) oraz logikę biznesową i zarządzanie danymi (ang. *backend*). Logika wykonywana na zdalnym serwerze jest rozwijana często w postaci zestawu mikroserwisów. Generalnie operacje związane z cyklem życia usługi chmurowej (wdrożonej w środowisku Kubernetes) są następujące: programista tworzy obraz aplikacji i plik konfiguracyjny wdrożenia (ang. *helm chart*), obraz trafia do rejestru (np. Docker Hub). Na tej podstawie Kubernetes (w skrócie K8s) [12] pobiera obraz i tworzy pody (uruchomione kontenery), rozkłada je na maszyny wirtualne (węzły, nody) tworząc tzw. klaster, a następnie monitoruje i utrzymuje działanie takich usług.

Zaletą działania usługi w chmurze obliczeniowej jest dynamiczne dopasowanie zasobów do zmieniającego się obciążania. Zmienna liczba żądań użytkowników (wywołanie różnych usług czy złożonych aplikacji – demonstratorów) jest zamieniana na pracę odpowiedniej liczby równoległe działających podów (kontenerów), po rozdzielaniu ich obrazów na dostępne węzły (VM) platformy CAISE zgodnie z opisem umieszczonym w pliku konfiguracyjnym lub przy zastosowaniu kryteriów uwzględniających dynamiczne zmiany zaangażowania węzłów. Kryterium rozdziału usług polega na ich asynchronicznym uruchamianiu, w zależności od dostępnych zasobów platformy oraz dynamicznym równoważeniu obciążenia węzłów.

Umieszczenie usługi w kontenerze oznacza stworzenie obrazu środowiska uruchomieniowego, który zawiera wszystko, co jest potrzebne do jej działania, w tym kod usługi, opis sposobu uruchomienia, biblioteki, zmienne środowiskowe i narzędzia systemowe. Wartości zmiennych środowiskowych przekazywane są do kontenera w momencie jego uruchomienia i mogą być następnie zmieniane. Pozwalają również na parametryzację środowiska wykonawczego usług. Kontenery zapewniają spójność programistyczną, testową i produkcyjną, ułatwiając tym samym tworzenie i wdrażanie usług w różnych środowiskach chmurowych. Podobnie zapewniają spójny i wydajny sposób izolacji i uruchamiania aplikacji złożonej z takich usług. To jest podstawowa cecha konteneryzacji platformy, dzięki której uzyskano spójność i przenośność środowisk chmurowych. W ten sposób każda usługa czy aplikacja zapakowana w kontener będzie działać identycznie w dowolnym środowisku wykonania Kubernetes, np. na komputerze deweloperskim, serwerze testowym czy produkcyjnym.

Dodatkowo K8s jako środowisko orkiestracji usług, nieustannie monitoruje stan wykonywanych usług. Jeśli kontener, w którym działa kluczowa mikrousluga, ulegnie awarii, to K8s automatycznie dokonuje restartu, a następnie może przenieść działające w nim usługi na inne węzły i uruchomić je szybko na nowo, w sposób całkowicie zautomatyzowany. K8s pozwala też na dynamiczne dostosowywanie się do

zmieniającego się obciążenia. W zależności od aktywności użytkowników, na podstawie zdefiniowanych metryk (np. użycia zasobów), automatycznie dobierze liczbę działających kopii usługi wykorzystując inteligentny mechanizm planowania (ang. *scheduler*). W efekcie odpowiednie rozmieszczenie usług na dostępnych serwerach prowadzi np. do zminimalizowania zużycia mocy obliczeniowej i wielkości pamięci, co w konsekwencji oznacza mniejsze koszty wykonania usług.

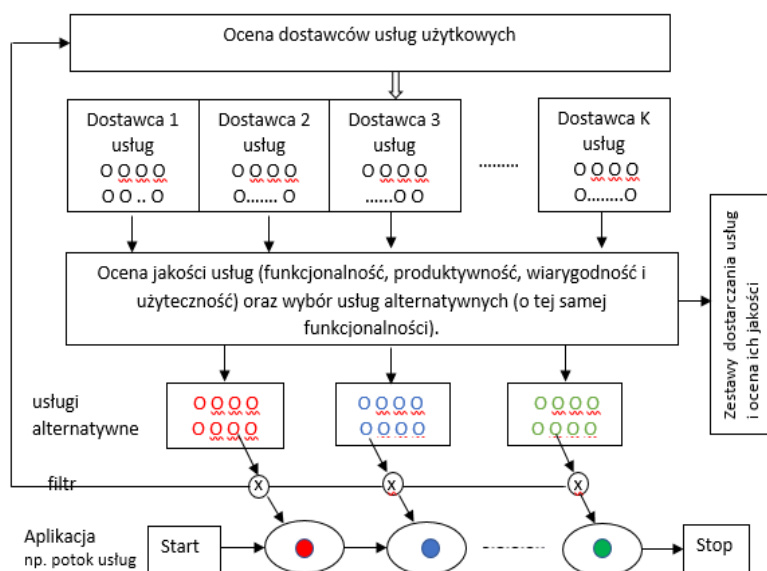
W K8s za alokację kontenerów (a dokładniej całych podów) na odpowiednich węzłach (ang. *nodes*) odpowiada komponent planisty (ang. *scheduler*). Kubernetes nie używa jednego, sztywnego algorytmu. zamiast tego działa w oparciu o tzw. Scheduling Framework – architekturę wtyczek (ang. *plugins*), które realizują proces wyboru węzła w dwóch głównych fazach: filtrowania (weryfikacja, kto spełnia wymagania) oraz punktowania (ocena, kto jest najlepszy wg zadanej funkcji kosztu). Algorytm planowania zależy od typu wykonywanej usługi. Wyróżniamy dwa podstawowe przypadki, gdy aplikacja jest tylko potokiem usług bądź jest reprezentowana przez bardziej złożoną strukturę (graf wykonania usług), gdzie poprzednio wykonanie zadania wskazuje na następne zadania do wykonania. Najczęściej stosowanym algorytmem alokacji zadań algorytm HEFT (ang. *Heterogeneous Earliest Finish Time*), który liczy czas zakończenia na każdym węźle i minimalizuje czas zakończenia aplikacji [12]. Inny algorytm to NSGA-II (ang. *Non-dominated Sorting Genetic Algorithm*) (Pareto DAG scheduling [11]) uwzględnia kilka kryteriów optymalizacji (np. czas wykonania, wielkość kosztu czy zużytej energii).

Możliwości K8s są jednak ciągle ograniczone. Ciągłe istnieje problem jak alokować i zarządzać efektywnie zasobami chmurowymi dla różnych implementacji architektur platform chmurowych, w tym także platformy CAISE. Rozmaitość rozwiązań platform chmurowych oraz chmurowych sieci i modeli obliczeń, złożoność procesów zarządzania stale rośnie, co wymaga również przygotowania nowych algorytmów zarządzania. Dodatkową kwestią, rozpatrywaną w tym artykule, jest budowa aplikacji z alternatywnych usług o różnej jakości, z których każda, charakteryzuje się różnymi czasami wykonania czy wielkością wymaganych zasobów chmurowych.

3. Propozycja uwzględniania parametrów jakości dostawców lub oferowanych usług

Przyjęto założenie, że aplikacja wytwarzana w środowisku K8s jest zbudowana z odpowiednich usług dostępnych na platformach chmurowych. Istnieją dwa podejścia do wyboru takich usług (rys. 2). Pierwsze podejście sprowadza się do uwzględnienia tylko dostawców usług. Wówczas uwzględnia się jedynie usługi najlepiej ocenianego dostawcy, zakładając, że jego usługi są najwyższej jakości. Podstawą do takiej oceny mogą być wymagania funkcjonalne czy jakościowe dostępne na poziomie proponowanych umów do zawarcia pomiędzy dostawcą z użytkownikiem (SLA). Co więcej oceny mogą się różnić w zależności od potrzeb klienta czy rodzaju poszukiwanych usług. Warto zaznaczyć, że proces wyboru jest dynamiczny, ponieważ pojawiają się nowi dostawcy oferujący wiele nowych usług w chmurze o podobnych funkcjonalnościach. Dokonując oceny można skorzystać z opracowywanych rankingów publikowanych w Internecie bądź wykorzystać narzędzia umożliwiające porównywania

wiarygodności i jakości różnych dostawców. Oprócz miar w pełni mierzalnych, takich jak kompletność funkcjonalna, wiarygodność dostarczania poprawnych wyników, czy skalowalność usług dla konkretnych wielkości zasobów obliczeniowych, ważne są także inne czynniki jakościowe, takie jak zaufanie, kompetencje i reputacja dostawcy, które również są uwzględniane w tego typu analizach [5]. Istnieje też możliwość oceny wielkości ryzyka związania się z solidnym dostawcą. Trudnością w takiej ocenie dostawców i ich usług jest brak ujednoliconych standardów opisu wymagań, co utrudnia często dokonanie poprawnego wyboru. Zaletą takiego podejścia jest prosty filtr eliminujący innych dostawców, co przekłada się na brak konieczności dokonywania zmian w oferowanych przez K8s algorytmów zarządzania usługami.



Rys. 2. Koncepcja wyboru usług do tworzonej aplikacji

Drugie podejście, które uwzględnia w swoim wyborze jakość usług użytkowych dostarczanych przez różnych dostawców, wymaga złożonych metod oceny jakości każdej usługi. W tym podejściu można też uwzględnić ranking dostawców, ale wchodzi on w skład oceny jakości usług. Na ogół pojedyncze usługi mogą być oceniane zarówno przez różnych dostawców, czy przez różnych użytkowników tych usług, na podstawie ich własnych doświadczeń. Takie rozwiązanie wymaga już włączenia opracowanych mechanizmów oceny jakości usług do algorytmów zarządzania oferowanych przez K8s. Jak podano w rozdziale 1 platforma CAISE proponuje wiele usług użytecznych oraz oferuje możliwości oceny każdej wytworzonej usługi [9]. Przyjęty model oceny jakości usług dla platformy CAISE obejmuje 4 następujące atrybuty:

1. Funkcjonalność która opisana jest poprzez wartość F1 reprezentującą jakość wykorzystanego w tej usłudze modelu sztucznej inteligencji.
2. Produktywność jest określana zarówno przez czas jej wytwarzania jak też średni czas jej wykonywania dla różnych zbiorów danych, jak też wielkość zużytych zasobów w K8s, (tzn.: wykorzystanie CPU i pamięci RAM przez dany pod), czy aktualny stopień wyskalowania danej usługi (np. poprzez liczbę replik).
3. Wiarygodność wyrażana jest przez dostępność i pewność realizowanych funkcjonalności w zadanym przedziale czasu. Opisać ją możemy poprzez stosunek

liczby wyeliminowanych błędów do liczby wszystkich pojawiających się problemów w tym czasie, co jest rejestrowane np. przez sondy K8s.

4. Użyteczność usług z kolei jest wyrażana poprzez opinie użytkowników co do ich przydatności do realizacji różnego typu aplikacji. Do oceny użyteczności można wykorzystać kwestionariusz PSSUQ [13]. Umożliwia pomiar odczuwanej satysfakcji użytkowników w trakcie wykorzystania tej usługi w ich aplikacjach.

Powyższe atrybuty są wyrażane przez odpowiednie metryki, które z kolei są wyliczane na podstawie odpowiednio przygotowanych procedur, a ich wartości są znormalizowane do przedziału $<0,1>$. W przypadku pomiarów innych wielkości rzeczywistych ich pozyskane wartości można odnieść do wartości maksymalnych. Kompleksową wartość jakości aplikacji, złożonej z konkretnych usług określa się, przy wysokich wymaganiach jakościowych, jako najniższą wartość metryki jakości otrzymaną dla wykorzystanych usług. Na ogół jednak do oceny przyjmuje się średnią wartości wszystkich metryk opisujących usługi wchodzące w skład ocenianej aplikacji. Praktycznie wartości liczbowe uzyskane przy ocenie usług można odnieść do konkretnego poziomu jakości, określanego w sposób następujący:

$$\text{Jakość } (Q) = \begin{cases} \text{niska} & Q \in [0.0, 0.3) \\ \text{dostateczna} & Q \in [0.3, 0.5) \\ \text{średnia} & Q \in [0.5, 0.7) \\ \text{wysoka} & Q \in [0.7, 0.9) \\ \text{bardzo wysoka} & Q \in [0.9, 1.0] \end{cases} \quad (1)$$

W środowisku chmurowym parametry jakościowe podane w specyfikacji (z przedziału $<0,1>$) są często tylko wartościami deklarowanymi. W praktyce mogą one ulegać wahaniom, co wiąże się z dynamiczną eksploatacją chmury obliczeniowej. Na przykład produktywność wybranej usługi oceniono na 0.85, ale gdy serwer tej usługi zostaje przeciążony to ten parametr jakości może spaść np. do 0.60. Wówczas powinna zostać dokonana zmiana na taką usługę alternatywną, która w tym momencie ma lepsze parametry, zapewniając wykonanie aplikacji z wymaganą jakością. Podobnie kiedy ruch użytkowników maleje, np. w nocy, można wykorzystać usługi o niższej produktywności, ale wciąż spełniające minimalne wymagania jakościowe na całą aplikację, wynikające z SLA sformułowanego przez użytkownika dla tej aplikacji. Strategia zarządzania jakością wykonywanych aplikacji powinna więc działać zarówno w sposób reaktywny (reagować po wykryciu spadku jakości), jak i proaktywny reaktywny (przewidywać takie sytuacje na podstawie obserwowanych trendów jej zachowania w przeszłości). W tym drugim przypadku wielką rolę odgrywa sztuczna inteligencja, która pozwala sformułować wzorce takich zachowań [3]. Można wykorzystać obie strategie działania, wówczas zwiększamy tzw. odporność (ang. *resilience*) wykorzystywanej platformy chmurowej. Pozwala na to zestaw Python + K8s, gdyż poszczególne usługi w potoku będą najczęściej osobnymi podami, a cała aplikacja może być sterowana przez centralny mikroservis dostępny w Pythonie i K8s.

4. Metoda optymalnego doboru usług dla aplikacji chmurowej o wysokiej jakości

Zakładamy, że problem doboru usług w scenariuszu działania aplikacji chmurowej sprowadza się do określenia odpowiedniej kompozycji dostępnych usług. Konieczna

jest podana architektura aplikacji (graf powiązań), w której wyspecyfikowano wymagania na poszczególne usługi. Właśnie taki konkretny opis dla wymaganej usługi konfrontuje się z opisami usług alternatywnych dostępnych w repozytorium platformy CAISE lub w innych otwartych repozytoriach. W przypadku braku odpowiednich alternatyw, istnieje możliwość budowy wymaganej usługi na tej platformie i następnie dołączenia go do repozytorium usług, co zwiększa też możliwości użytkowe samej platformy. Budowaną aplikację można zaprojektować i zaimplementować w całości i następnie ją wykonać. Można też wpleść problem doboru usług na bieżąco, tzn. w trakcie wykonywania aplikacji, inaczej w trakcie wywoływania kolejnych usług w niej występujących. To ostatnie podejście zapewnia reaktywną i proaktywną strategię zarządzania aplikacją. Oczywiście algorytm wyboru odpowiedniej usługi musi być efektywny czasowo, by nie obciążać aplikacji zbyt dużym dodatkowym czasem jej wykonania.

Zapewnienie dostępu do niezbędnych usług oraz możliwości ich efektywnej alokacji oraz wykonania jest dużym wyzwaniem [6]. Jak sygnalizowano powyżej niezwykle trudne jest przewidzenie a priori wszystkich usług, które mogą okazać się potrzebne dla nowo tworzonych, zwłaszcza innowacyjnych aplikacji. Na ogół dostawcy oferują proste, specyficzne usługi, które powinny być odpowiednio adaptowane do typowych potrzeb. Wybór optymalnego zestawu właściwych usług ze zbiorów dostępnych usług alternatywnych o różnej jakości jest dodatkowym wyzwaniem. Co więcej oferowane usługi alternatywne przez różnych dostawców mogą charakteryzować się odmiennymi parametrami jakości, tak że podanie jednej optymalnej kompozycji jest prawie niemożliwe. Rozpatrywanie tego zagadnienia [4] w kategoriach problemu optymalizacyjnego prowadzi do problemu klasy NP-trudnej, co wynika nie tylko z dużej liczby usług oferowanych przez różnych chmurowych dostawców, ale też ze złożoności struktury samej aplikacji.

Ogólny schemat wyboru usług jest podany na rysunku 3. Jak wyżej zaakcentowano, głównym problemem jest opracowanie metody wyboru usługi spośród alternatywnie dostępnych, takiej która spełnia zarówno założone specyfikacje dotyczące aplikacji, jak i wymagania przedstawione w SLA. Ten problem możemy zdefiniować w następujący sposób, przyjmując następujące oznaczenia:

$U = \{u_1, u_2, \dots, u_m\}$ – zbiór usług alternatywnych do wykonania analizowanej usługi.

$Q = \{q_1, q_2, \dots, q_n\}$ – zbiór parametrów jakościowych określających każdą z usług alternatywnych, $q_i \in \langle 0, 1 \rangle$.

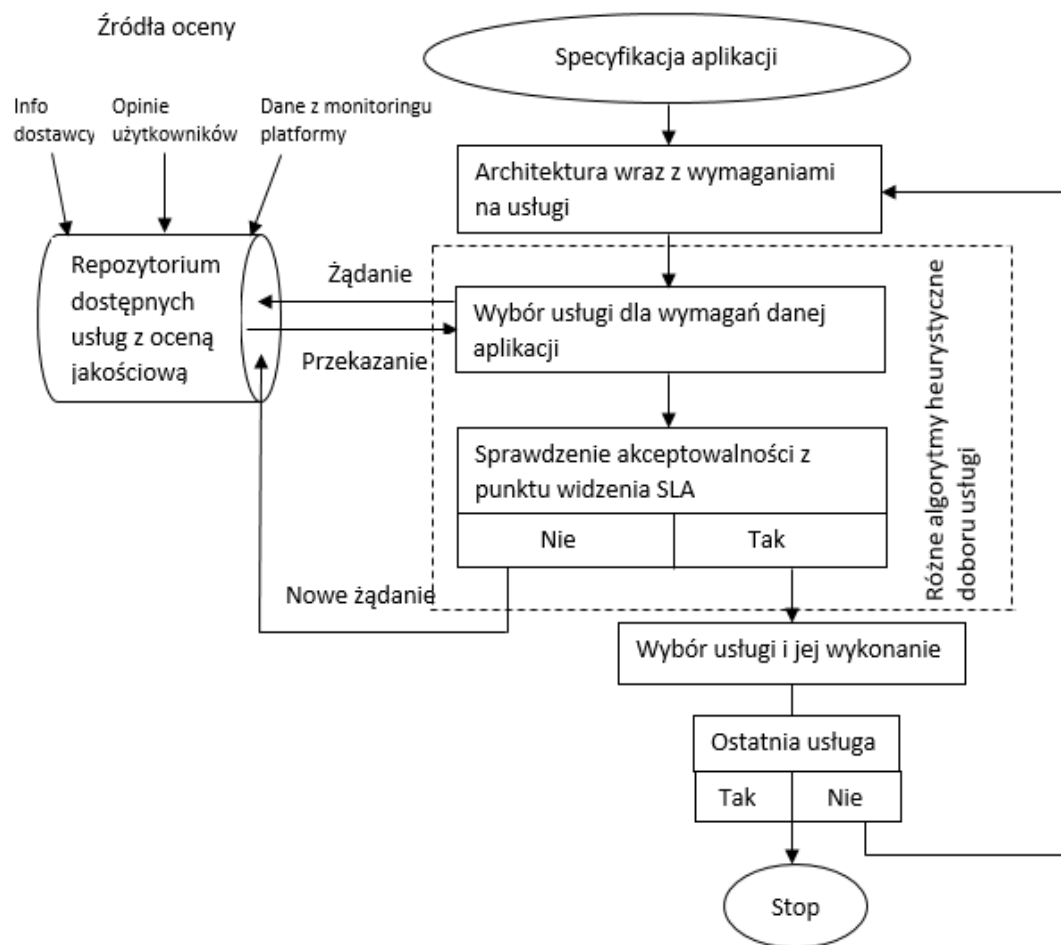
$Z = \{z_1, z_2, \dots, z_n\}$ – zbiór zasobów będących w dyspozycji chmury obliczeniowej.

δ – wektor bieżącego zaangażowania zasobów chmury, np. $\delta = [\delta_1, \delta_2, \dots, \delta_n]$, $\delta_i \in \langle 0, 1 \rangle$.

ψ – macierz wykorzystania zasobów niezbędnych dla wykonania alternatywnych usług ze zbioru U :

$\psi = [\psi_{ij}]$, gdzie $i = 1, 2, \dots, m; j = 1, 2, \dots, n$; zaś:

$$\psi_{ij} = \begin{cases} 1 & \text{gdy usługi } u_i \text{ jest przypisana do zasobu } z_j \\ 0 & \text{w przeciwnym przypadku} \end{cases} \quad (2)$$



Rys. 3. Schemat realizacji aplikacji o wymaganej jakości

Macierz $\gamma = [\gamma_{ij}]$ określa stopień dotychczasowego zaangażowanie zasobów przed wykonaniem usług, ze zbioru U , $\gamma_{ij} \in \langle 0, 1 \rangle$.

T – macierz czasów wykonania usług ze zbioru U przy wykorzystaniu zasobów ψ ze zbioru Z :

$$T = [t_{ij}] \quad (3)$$

gdzie t_{ij} jest czasem wykonania usługi u_i na dostępnym zasobie z_j , tzn. gdy $\gamma_{ij} \leq 1 - \delta_j$. W przeciwnym przypadku, czas wykonania usługi t_{ij} zwiększa się zgodnie ze wzorem:

$$t'_{ij} = (\delta_j + \gamma_{ij}) \cdot t_{ij} \cdot q_i \quad (4)$$

gdzie uwzględniono, tzw. uogólniony czas wykonania usługi uwzględniający jej jakość, zakładając, że jest mniejszy dla usługi o niższej jakości, a większy dla usługi o wyższej jakości. Oczywiście zmienną q_i można zastąpić funkcją $f(q_i)$, która określa w jaki sposób jakość usługi jest związana z czasem jej przetwarzania i wielkością wymaganych zasobów.

Czas realizacji aplikacji A na zbiorze Z określono jako $\tau(A, Z)$, maksymalne zaangażowanie zasobów chmurowych w wykonanie aplikacji A oznaczono jako $\delta(A)$, która nie powinno przekraczać δ_o , zaś jakość odpowiadająca aplikacji $Q(A)$ powinna

być wyższa od wartości granicznej q_o . Zatem zadanie optymalizacyjne można sformułować następująco:

$$\begin{aligned} \tau(A, Z) &= \min \\ \delta(A) &\leq \delta_o \\ Q(A) &> q_o \end{aligned} \quad (5)$$

Wielkości δ_o oraz q_o są wartościami dopuszczalnymi dla całej aplikacji wynikającymi z jej specyfikacji. Zakładając, że dla wykonania aplikacji UA dokonano wyboru usług ua_k , ze zbiorów usług alternatywnych, tzn.: $UA = \{ua_1, ua_2, \dots, ua_K\}$, dla $k = 1, 2, \dots, K$. Warto podkreślić, że wzorze (4) preferuje się usługi o mniejszej jakości, ale w ograniczeniach wzoru (5) cała aplikacja powinna zapewniać odpowiednią jakość, co eliminuje to rozwiązanie. Nie możemy więc wybierać usług tylko o najniższej jakości. Czas realizacji aplikacji na zbiorze zasobów Z określa się w sposób następujący:

$$\tau(A, Z) = \sum_{k=1}^K \sum_{j=1}^n t'_{ij} \cdot \psi_{kj} \quad (6)$$

Szukamy więc takiego ψ dla którego $\tau(A, Z)$ wykorzysta nie większą niż δ_o wielkość zasobów oraz jakość aplikacji będzie większa od wymaganej wartości δ_o . Dodatkowo zakładamy, że każda usługa jest lokowana na różne zasoby $\sum_{k=1}^K \psi_{ij} = 1$, które umożliwiają jej w miarę szybkie wykonanie tzn.: $\gamma_{ij} \leq 1 - \delta_j$. To oznacza, że aplikacja musi adaptować się do istniejących zasobów, dynamicznie w czasie rzeczywistym. To oznacza również, że aplikacja potrafi samodzielnie i natychmiastowo zmienić swoją konfigurację (np. podmienić jedną z usług alternatywnych na inną) w trakcie swojego działania, bez przestojów i bez pomocy zewnętrznej.

Jednym z możliwych algorytmów do wykorzystania jest algorytm mrówkowy [10]. Kolonia mrówek w celu zdobycia pożywienia wykorzystuje proste instynkty mrówek, które pozwalają je znaleźć pożywienie (w naszym przypadku odpowiedni czas $\tau(A, Z)$) nawet jeśli znajduje się bardzo daleko od ich zasiedlenia. Mrówisko wypuszcza poruszających się mniej więcej losowo zwiadowców, którzy mają pewne poczucie kierunków i gdy natrafią na coś interesującego wracają do kolonii, znacząc swoją trasę feromonami. Pozostawione na trasie feromony przyciągają więcej mrówek, które zaczynają się kierować mniej więcej wyznaczoną przez feromony trasą i transportują pożywienie do miejsca pobytu, zaznaczając swoją drogę powrotną również feromonami. Większa liczba feromonów przyciąga większą liczbę mrówek, tym samym trasy krótsze są częściej wykorzystywane, a te najdłuższe są znacznie częściej ignorowane. Dodatkowo, feromony parują z czasem, co coraz bardziej zwiększałoby różnicę w ich natężeniu pomiędzy różnymi trasami, sprawiając że bardziej preferowana trasa stawałaby się jeszcze bardziej popularna. Zatem dzięki feromonom mrówki same się organizują i wybierają zalecane trasy, które wskazują sposób zrealizowania założonego celu, tzn. zdobycie odpowiedniej wielkości pożywienia, jak najmniejszym kosztem. Tego typu zachowania mrówek wykorzystano właśnie przy wyborze optymalnego rozwiązania dla problemu wyznaczonego wzorem (5).

5. Wnioski końcowe

Problematyka doboru usług dla aplikacji chmurowej staje się coraz bardziej popularna z uwagi na wzrost dostępności różnego typu usług w środowiskach chmurowych. Zapewnienie zdolności do adaptacji, dynamiki przetwarzania usług oraz efektywnego zarządzania jakością w środowiskach chmurowych pozostaje nadal

kluczowym wyzwaniem. W pracy zaproponowano proste zadanie optymalizacyjne uwzględniające jakość oferowanych usług jak i dynamikę ich przetwarzania na zasobach chmurowych. Przedstawiony problem może być bardziej rozbudowywany. Na przykład czas realizacji pojedynczej usługi może być określony poprzez trzy składowe: czas ulokowania jej na zasobach wirtualnych chmury obliczeniowej, czas transmisji danych do tej maszyny wirtualnej oraz czas jej wykonania na wskazanych zasobach fizycznych węzła chmury. W przypadku minimalizacji zużytej energii tego typu czasy muszą być pomnożone przez odpowiednie współczynniki zużycia energii. Dla zapewnienia wymaganej niezawodności działania w ograniczeniach jakościowych można uwzględnić zarówno intensywność błędów pojawiających się w maszynach wirtualnych jak też intensywność uszkodzeń całych węzłów. Można także wprowadzić procedury zwielokrotnienia wykonania wybranych usług alternatywnych na różnych węzłach oraz procedury migracji maszyn wirtualnych na inne zasoby fizyczne chmury w przypadku uszkodzenia węzła lub nadmiernego jego obciążenia. W tym ostatnim przypadku należy odpowiednio zmodyfikować wzrost czasów wykonania usług (patrz wzór (4)) o czas uruchomienia nowego węzła, jeśli nie jest aktywny oraz o czas transmisji maszyn wirtualnych pomiędzy dwoma węzłami, tym rozpatrywanym i tym docelowym, a także czas wyboru i posadowienia wskazanej usługi na węźle docelowym. Przedstawione kryteria optymalizacji wymagają jednak pozyskania dodatkowych danych związanych zarówno, z wykorzystaniem infrastruktury chmurowej, jak też efektywnych metod oceny usług alternatywnych. Dane te mogą być dostarczone dzięki specjalnym eksperymentom wykonywanych na badanej chmurze obliczeniowej, albo obserwacji zachowania się danej platformy chmurowej. W pracy wykorzystano pewne doświadczenie wynikające z procesu walidacji platformy CAISE, w tym ocenę jakości dostarczanych przez nią usług. Na tej podstawie przeanalizowano możliwości K8s oraz sformułowano powyższe zadanie optymalizacyjne dla dynamicznie zmieniających się zasobów obliczeniowych. Dalsze interesujące kierunki rozwoju są przedstawione w pracy [1], gdzie rozpatruje się nawet wykorzystanie systemów samoadaptacyjnych do zapewnienia wymagań użytkowników platform chmurowych. To podejście wymaga jednak oddzielnego potraktowania.

Podziękowania: Badania były częściowo wspierane przez projekt pt. *Chmurowa platforma (CAISE) do wytwarzania uniwersalnych usług inteligentnych dla różnych obszarów zastosowań*, nr KPOD.05.10-IW.10-0005/24, w ramach europejskiego programu IPCEI-CIS, finansowanego ze środków KPO.

LITERATURA

1. Apostolos A., Kousiouris G.: An Overview on the Landscape of Self-Adaptive Cloud Design and Operation Patterns: Goals, Strategies, Tooling, Evaluation, Future Internet 2025, 17(10), 434; <https://doi.org/10.3390/fi17100434>
2. Burns B., Beda J., Hightower K., Kubernetes: Up & Running, O'Reilly Media, 2022.
3. [Huyen Chip](#), AI Engineering: Building Applications with Foundation Models, O'Reilly Media, 2025

4. Dahan F.: An innovative approach for QoS-aware web service composition using whale optimization algorithm. *Sci Rep* **14**, 22622 (2024). <https://doi.org/10.1038/s41598-024-73414-8>
5. Kumar G. S., Versteeg S., Buyya R.: SMICloud: A Framework for Comparing and Ranking Cloud Services, 4th IEEE International Conf. on Utility and Cloud Computing, Dec. 2011, pp. 210 – 218.
6. Jula A., Sundararajan E., Othman Z.: Cloud computing service composition: A systematic literature review. *Expert Systems with Applications*, 41(4), 3809-3824. Elsevier BV, 2014, 10.1016/j.eswa.2013.12.017
7. Krawczyk H., Weichbroth P.: The CAISE Platform: A Modern Framework for the Development of Intelligent Cloud Services, 30th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, KES 2026, wyd. Elsevier, 2026.
8. Krawczyk H., Wiszniewski B., Orzechowski P., Ziółkowski T.: An open cloud native AI platform for public sector digitization, accepted for PIPAM Int. Conference, 2026.
9. Krawczyk H., Wiszniewski B.: Resilient Software Development of Intelligent Services in Cloud Environments. *Advances in Dependable Systems and Networks – Proceedings of the Twentieth International Conference on Dependability of Computer Systems DepCoS-RELCOMEX*, June 30 – July 3, 2025, Brunów, Poland, Springer.
10. Krawczyk H., Orzechowski P.: Algorytm mrówkowy do zarządzania zasobami sprzętowymi chmury obliczeniowej w przypadku różnych kategorii usług, KASKBook, wydawnictwo Politechniki Gdańskiej, 2022.
11. Ma H., Zhang Y., Sun S. *et al.*: A comprehensive survey on NSGA-II for multi-objective optimization and applications. *Artif Intell Rev* 56, 15217–15270 (2023). <https://doi.org/10.1007/s10462-023-10526-z>
12. Samadi Y., Zbakh M., Tadonki C.: E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm for Task Scheduling based on Load Balancing in Cloud Computing," *2018 International Conference on High Performance Computing & Simulation (HPCS)*, Orleans, France, 2018, pp. 601-609, doi: 10.1109/HPCS.2018.00100.
13. Sauro J., Lewis James R.: Quantifying the User Experience: Practical Statistics for User Research, Morgan Kaufmann, 2016.
14. Sommerville J.: Inżynieria oprogramowania, PWN, 2020.