Polska Akademia Nauk
Instytut Podstawowych Problemów Techniki
oraz
Uniwersytet Jagielloński
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Rozprawa doktorska

Inteligentne systemy monitoringu procesów harmonogramowania w rozproszonych środowiskach dużej skali w ujęciu wieloagentowym

mgr inż. Daniel Grzonka

Promotor: dr hab. Joanna Kołodziej, prof. PK
Promotor pomocniczy: dr Agnieszka Jakóbik

Politechnika Krakowska
Instytut Informatyki

Kraków, 2018

Streszczenie

W ostatnich latach nowoczesne rozproszone środowiska obliczeniowe, takie jak chmury, dynamicznie zyskują na znaczeniu. Wzrost popularności tego typu środowisk niesie ze sobą wiele wyzwań, wśród nich efektywne wykorzystanie zasobów, zagwarantowanie odpowiedniego poziomu oferowanych usług czy nadzór nad działaniem całego środowiska. Procesy harmonogramowania zadań są jednymi z procesów mających niebagatelny wpływ na prawidłowe funkcjonowanie takich środowisk.

Głównym celem niniejszej pracy jest opracowanie modeli inteligentnych systemów monitorujących procesy harmonogramowania niezależnych zadań w rozproszonych środowiskach obliczeniowych dużej skali. Rolą takich systemów jest nie tylko nadzór nad procesem, ale również aktywna ingerencja w jego przebieg. Zaproponowany system zbudowany jest z pięciu typów agentów, które wspierają proces harmonogramowania zadań poprzez bezpośrednią ingerencję w przebieg algorytmu ewolucyjnego, wsparcie w podejmowaniu decyzji o rozpoczęciu generowania harmonogramu oraz monitoring wykonania zadań i korektę oczekiwanych czasów realizacji harmonogramów.

Ponadto w ramach pracy opracowano formalny model dynamiki rozproszonego środowiska obliczeniowego, zaproponowano nową reprezentację populacji dla ewolucyjnego procesu harmonogramowania, rozszerzono modele zadań i jednostek obliczeniowych, zaproponowano notację do opisu systemów monitoringu procesów zachodzących w środowiskach rozproszonych oraz rozszerzono istniejące taksonomie systemów monitoringu.

Obliczenia numeryczne przeprowadzono w ramach dedykowanego środowiska symulacyjnego. Uzyskane wyniki dowodzą, że możliwe jest usprawnienie procesu generowania harmonogramów niezależnych zadań obliczeniowych poprzez działania inteligentnego systemu monitoringu opartego o paradygmat agentowy. Dzięki zastosowaniu zaproponowanych modeli m.in. przyspieszono proces generowania harmonogramów, zredukowano czas wykonywania całych pakietów zadań, a także zwiększono średnią wydajność infrastruktury i niezawodność predykcji czasów realizacji zadań.

Abstract

Intelligent agent-based monitoring systems of task scheduling for distributed high-performance environments

Over the last few years modern distributed computing environments, such as clouds, become more and more important. The increase in the popularity of this type of environment brings many challenges, among them the effective usage of resources, guaranteeing the appropriate quality of services, or supervision over the operation of the entire environment. Task scheduling processes have a significant impact on the proper functioning of such environments.

The main goal of this study is to develop models of intelligent systems that monitor the processes of independent tasks scheduling in distributed large-scale computing environments. The role of such systems is not only the supervision of the process, but also its support. The proposed system is built of five types of agents that support the process of scheduling tasks through direct interference with evolutionary scheduling algorithm, support in making decisions on starting schedule generation, monitoring the performance of tasks and correcting the expected times of task completion.

Furthermore, as part of the thesis, a formal model for the dynamics of the distributed computing environment was developed, a new population representation for the evolutionary algorithm used in the scheduling process was proposed, the models of tasks and computing units were expanded, a notation was added to the description of monitoring systems of processes occurring in distributed environments and the pre-existing monitoring taxonomies were extended.

The numerical experiments were carried out as part of a dedicated simulation environment. The obtained results prove that it is possible to improve the process of generating schedules of independent tasks through the agent-based intelligent monitoring system. Due to the proposed models, i.a. the scheduling generation process was accelerated, the time of executing whole task batch was reduced, as well as the average efficiency of the computing infrastructure and the reliability of prediction of task execution times were increased.

Podziękowania

Chciałbym wyrazić najserdeczniejsze podziękowania osobom, które przyczyniły się do powstania niniejszej rozprawy:

- promotorowi pracy, Pani Profesor Joannie Kołodziej za opiekę merytoryczną, możliwość rozwoju naukowego i okazaną cierpliwość podczas prac nad rozprawą;
- promotorowi pomocniczemu, Pani Doktor Agnieszce Jakóbik za owocną współpracę, wielogodzinne dyskusje i wiarę w powodzenie realizowanych prac;
- Panu Doktorowi Rostislavowi Razumchikowi oraz Pani Doktor Grażynie Suchackiej za poświęcony czas, liczne konsultacje i cenne uwagi;
- współpracownikom z Instytutu Informatyki Politechniki Krakowskiej za owocne dyskusje i przemyślenia;
- przyjaciołom, znajomym, studentom i uczniom, a przede wszystkim mojej rodzinie za nieustanne wsparcie i wiarę, które dawały mi siłę i motywację do działania.

"A co człowiek sieje,
to i żąć będzie"
(Ga 6, 8)

Najbliższym…

Spis treści

Rozdział 1

Wstęp

1.1. Wprowadzenie i motywacja podjęcia tematu pracy

Rozproszone środowiska obliczeniowe od wielu lat są intensywnie rozwijane i coraz częściej wykorzystywane w różnych aspektach codziennego funkcjonowania. Wśród nich na szczególną uwagę zasługują gridy i chmury obliczeniowe. W ostatnim czasie głównie chmury zmieniają sposób zaspokajania potrzeb związanych m.in. z infrastrukturą komputerową oraz oprogramowaniem. Ideą chmury obliczeniowej jest przeniesienie całego ciężaru świadczenia usług informatycznych na usługodawcę i umożliwienie stałego, zdalnego dostępu do wykupionych zasobów. Wykorzystanie systemów rozproszonych stało się bardzo popularnym i przede wszystkim efektywnym podejściem w rozwiązywaniu złożonych problemów obliczeniowych w wielu dziedzinach nauki. Wzrost popularności tego typu środowisk niesie ze sobą wiele wyzwań, wśród nich efektywne wykorzystanie zasobów, zapewnienie odpowiedniego poziomu bezpieczeństwa, personalizację usług czy nadzór nad działaniem całego środowiska. Wymienione problemy wymagają często nowych, złożonych, a nieraz i inteligentnych rozwiązań.

Jednocześnie można zauważyć trend związany z automatyzacją jak największej liczby procesów zachodzących w każdym obszarze działania systemów dużej skali. Podejście takie często zwalnia z konieczności manualnego podejmowania decyzji, a nawet wykonywania nadzoru przez człowieka. Systemy coraz częściej charakteryzują się możliwością podejmowania inteligentnych i autonomicznych działań. Jednym z zyskujących popularność w ostatnich

dziesięcioleciach algorytmów sztucznej inteligencji są autonomiczne systemy wieloagentowe. W rozumieniu niektórych autorów (por. [88]), inteligencja obliczeniowa definiowana jest właśnie poprzez użycie pojęcia agenta. Agent to system, który podejmuje autonomiczne działania na podstawie danych dotyczących okoliczności i założonych celów, a także ma zdolność przystosowywania się do zmieniającego się środowiska.

Paradygmat agentowy, poprzez swój charakter, może być interesującym rozwiązaniem stosowanym w systemach monitoringu – w szczególności monitoringu systemów rozproszonych. Monitoring taki nie ograniczałby się tylko do zbierania danych statystycznych, ale miałby realny wpływ – poprzez inteligentne działania – na sposób funkcjonowania środowiska.

Jednym z ważniejszych procesów zachodzących w rozproszonych środowiskach obliczeniowych jest proces harmonogramowania (nazywany również procesem szeregowania) zadań. Sprowadza się on do utworzenia harmonogramu przydziału zadań do jednostek obliczeniowych. Harmonogramowanie jest niezbędnym elementem każdego rozproszonego systemu – pozwala na odpowiednie wykorzystanie dostępnych zasobów i może być kluczowym zagadnieniem w funkcjonowaniu środowiska obliczeniowego, jakim jest chmura obliczeniowa. Tak istotne zagadnienie, szczególnie w nowoczesnych systemach opartych o ideę wirtualizacji (gdzie zdolności obliczeniowe systemu mogą być wysoce zmienne), niewątpliwie wymaga innowacyjnych rozwiązań monitoringu i aktywnego wsparcia procesu. Odpowiednim narzędziem wspomagania tak złożonych procesów decyzyjnych w środowiskach rozproszonych mogą być wcześniej wspomniane inteligentne systemy wieloagentowe.

1.2. Cel i teza pracy

Głównym celem niniejszej rozprawy jest opracowanie i implementacja innowacyjnych modeli agentowych systemów monitorujących i wspomagających procesy harmonogramowania w rozproszonych środowiskach obliczeniowych dużej skali. Zaproponowany system złożony jest z wielu rozproszonych elementów – agentów, które pozwalają zaliczyć go do klasy rozwiązań z dziedziny inteligencji obliczeniowej. System ten ma możliwość ingerencji

w procesy ewolucyjne, których zadaniem jest wygenerowanie harmonogramu zadań. Ponadto w ramach systemu wykorzystano sztuczne sieci neuronowe (SSN) oraz analizę stanów monitorowanego środowiska.

Do celów pobocznych pracy można zaliczyć rewizję istniejących taksonomii systemów monitoringu, formalny opis dynamiki obliczeniowego systemu rozproszonego pod kątem realizacji procesu szeregowania zadań, definicję nowej reprezentacji populacji dla algorytmu ewolucyjnego wykorzystanego w procesie harmonogramowania, zaproponowanie bardziej realistycznych modeli zadań i jednostek obliczeniowych, propozycję notacji dla definicji przeznaczenia systemów monitoringu rozproszonych środowisk obliczeniowych a także propozycję integracji systemu monitorującego z rozproszonym środowiskiem obliczeniowym.

Określone wcześniej motywacje, a także powyższe cele pozwoliły na sformułowanie następującej tezy rozprawy doktorskiej:

> Agentowe wspomaganie monitoringu procesów harmonogramowania i realizacji harmonogramów w rozproszonych systemach obliczeniowych dużej skali pozwoli na usprawnienie procesu generowania harmonogramów niezależnych zadań obliczeniowych. Zastosowanie inteligentnych metod może przyczynić się do generowania bardziej optymalnych harmonogramów w krótszym czasie. Dodatkowo, inteligentny monitoring pozwoli na efektywną, wszechstronną analizę i poprawę parametrów funkcjonalnych (takich jak równoważenie obciążenia czy terminowość realizacji harmonogramów) monitorowanych systemów obliczeniowych.

Rozdział 2

Ogólny opis problemu

Monitoring rozproszonych środowisk obliczeniowych dużej skali, ze względu na złożoność systemów jak i oczekiwania użytkowników, jest szerokim zagadnieniem. Systemy takie w dużej mierze są systemami heterogenicznymi – zbudowanymi z jednostek o różnych charakterystykach i przeznaczeniach. Na właściwą pracę rozproszonego środowiska obliczeniowego (jak np. chmury obliczeniowej czy gridu) wpływ ma szereg czynników i procesów. Jednym z kluczowych z punktu widzenia wydajności systemu jest proces harmonogramowania (inaczej: szeregowania) wykonywanych zadań obliczeniowych. Sam proces harmonogramowania zadań w środowiskach dużej skali jest problemem NP-zupełnym, a więc wymaga specjalnych metod generujących rozwiązanie.

Tytułowy inteligentny system monitoringu rozumiany jest jako system obserwujący złożone, rozproszone środowisko, który zbiera dane na temat poszczególnych obszarów mających bezpośredni wpływ na proces harmonogramowania oraz, za pomocą inteligentnych narzędzi informatycznych, w sposób aktywny i niezależny wspiera ich funkcjonowanie. Rozważanym problemem harmonogramowania jest harmonogramowanie niezależnych zadań obliczeniowych w rozproszonych środowiskach dużej skali. Kompleksowy system monitoringu dla tego problemu może obejmować szereg obszarów, takich jak proces gromadzenia zadań do wykonania, ustalanie momentu rozpoczęcia procesu generowania harmonogramu, ingerencja w parametry generowania, weryfikacja realizowanych harmonogramów czy analiza stanów jednostek obliczeniowych oraz zbieranie danych statystycznych [47, 48, 49]. Jeśli weźmie się pod uwagę dodatkowe kryteria, które również mają istotny wpływ na proces harmono-

gramowania, jak np. zużycie energii, bezpieczeństwo, gwarancja zapisów umowy o gwarantowanym poziomie świadczenia usług (ang. *Service Level Agreement, SLA*), rozliczanie czy personalizacja usług, to monitoring może być kluczowym elementem działania rozproszonego środowiska.

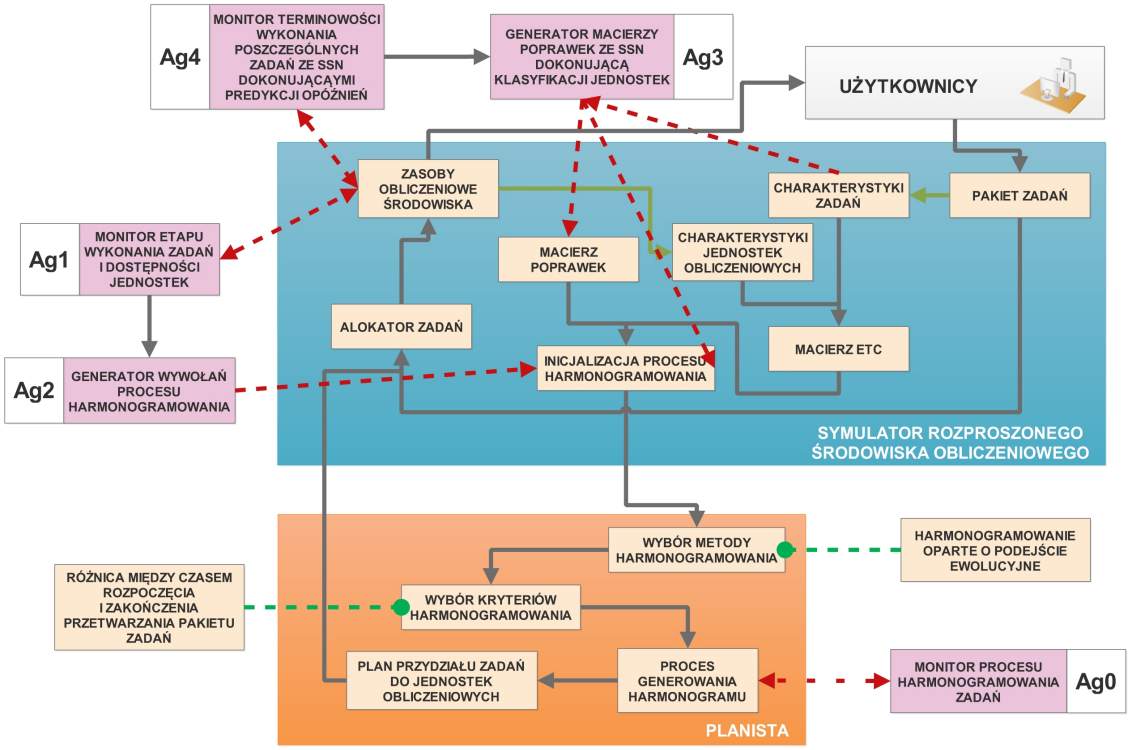
Wobec powyższych uwag, zagadnienie monitorowania działania systemów rozproszonych nabiera zupełnie innego znaczenia. Odpowiednim narzędziem wspomagania złożonych procesów decyzyjnych w powyższych zagadnieniach mogą być inteligentne systemy wieloagentowe (ang. *multi-agent system, MAS*). Teza ta znajduje potwierdzenie m.in. w [25], gdzie autor wskazuje na zarządzanie rozproszonymi systemami jako potencjalne pole do zastosowań paradygmatu agentowego – ze szczególnym naciskiem na procesy harmonogramowania oraz równoważenia obciążenia.

W ramach niniejszej rozprawy opracowano i zaimplementowano model inteligentnego systemu monitoringu procesu harmonogramowania i obszarów mających bezpośredni wpływ na ten proces. System opiera się o podejście wieloagentowe, w którym inteligentne agenty, poprzez realizację swoich strategii, oddziałują na stany środowiska.

Z perspektywy postawionego problemu, inteligentny system monitoringu może być zdefiniowany jako system obserwujący złożone procesy w rozproszonych środowiskach, który zbiera dane na temat poszczególnych obszarów oraz za pomocą inteligentnych narzędzi informatycznych w sposób aktywny i niezależny wspiera ich funkcjonowanie. Cechy, jakimi charakteryzuje się inteligentny system monitoringu, zbieżne są z własnościami, jakie niesie ze sobą wykorzystanie paradygmatu agentowego. Odpowiednio zaprojektowany system agentowy może w pełni spełnić oczekiwania postawione wobec systemu monitorującego.

Rysunek 2.1 przedstawia przepływ informacji w systemie rozproszonym, ze szczególnym naciskiem na moduł harmonogramowania zadań. Diagram uwzględnia również rozproszony system monitoringu oraz jego oddziaływanie na środowisko.

Najogólniej problem ujmując, zadaniem rozpatrywanego środowiska obliczeniowego jest realizacja zleconych zadań obliczeniowych. Rysunek 2.1 przedstawia ogólną ideę działania środowiska. Użytkownicy zlecają wykonanie zadań, które formują się w pakiet – docelowo



Rysunek 2.1: Przepływ pracy w systemie rozproszonym z uwzględnieniem systemu monitorującego.

zadania zostaną wykonane na zasobach obliczeniowych środowiska. Na podstawie pakietu zadań konstruowane są ich charakterystyki, które wraz z charakterystykami jednostek obliczeniowych służą do budowy macierzy oczekiwanych czasów realizacji zadań (jest to oszacowanie czasu wykonania każdego zadania pakietu na każdej jednostce dostępnej w zasobach obliczeniowych środowiska). Następnie inicjalizowany jest proces generowania harmonogramów przez planistę. Planista dokonuje wyboru metody harmonogramowania. Rozważaną w niniejszej pracy metodą jest harmonogramowanie oparte o algorytm ewolucyjny. Następnie wybierane jest kryterium, względem którego harmonogramy będą optymalizowane. Najpopularniejszym kryterium harmonogramowania pakietów zadań jest różnica między czasem rozpoczęcia i zakończenia przetwarzania pakietu zadań. Po dokonaniu wyboru metody i kryterium realizowany jest proces, w ramach którego generowany jest harmonogram, czyli plan przydziału zadań do jednostek obliczeniowych. Gotowy harmonogram przesyłany jest do środowiska, w którym zostaje zrealizowany. Poszczególne zadania zostają przypisane i wykonane na jednostkach obliczeniowych zgodnie z przyjętym harmonogramem. Wyniki wykonania zadań przekazywane są do zleceniodawców.

W opisanym procesie udział bierze również system monitoringu, którego model jest przedmiotem rozprawy. W systemie monitoringu wyszczególniono pięć typów agentów – każdy typ realizuje powierzone mu zadania. Agenty wyposażone są w systemy decyzyjne oparte o wartości progowe parametrów, stany poszczególnych elementów systemu czy odpowiedzi uzyskane ze sztucznych sieci neuronowych. Poprzez nadzór nad procesem harmonogramowania, ingerencję w jego przebieg, a także analizę wykonania harmonogramów, system monitorujący aktywnie wspiera działanie środowiska.

Pierwszy z agentów – monitor procesu harmonogramowania zadań – monitoruje przebieg procesu ewolucji rozwiązania na przestrzeni epok. Jego rola sprowadza się do inteligentnego operatora selekcji (a zarazem realizatora strategii przeżywalności osobników). W przypadku gdy generowane harmonogramy nie charakteryzują się lepszym przystosowaniem, agent podejmuje decyzję o zmniejszeniu liczby osobników, które podlegają krzyżowaniu, zwiększając jednocześnie liczbę osobników, które zostają przeniesione do nowej populacji.

Dwa kolejne typy agentów, czyli monitor dostępności jednostek obliczeniowych oraz generator wywołań procesu harmonogramowania, współpracują w celu minimalizacji czasu bezczynności jednostek obliczeniowych. Każda z jednostek obliczeniowych monitorowana jest przez agenta w celu uzyskania informacji o jej dostępności lub aktualnym stanie realizacji harmonogramu. Agent inicjujący proces harmonogramowania przeprowadza referendum wśród agentów monitorujących jednostki. Na podstawie udzielonych informacji wyznacza termin rozpoczęcia kolejnego procesu harmonogramowania.

Pozostałe dwa typy agentów wyposażone są w sztuczne sieci neuronowe. Pierwszy typ, monitor terminowości wykonania poszczególnych zadań dla każdej jednostki obliczeniowej, sprawdza terminowość wykonania zadań zgodnie z przyjętymi szacunkami. Na podstawie zebranych danych oraz charakterystyk przetwarzanych zadań uczy sztuczną sieć neuronową predykcji opóźnień dla każdego z zadań. Drugi typ agenta konsultuje oczekiwane czasy realizacji zadań z każdym z agentów monitorujących opóźnienia. W wyniku konsultacji tworzona jest tzw. macierz poprawek, która koryguje wartości oszacowań z macierzy ETC.

Opisany powyżej system monitoringu wspiera działanie rozproszonego środowiska na różnych etapach jego działania. Nie ogranicza się tylko do nadzoru, ale przede wszystkim podejmuje działania, które mają pozytywny wpływ na wydajność środowiska i czas generowania harmonogramów.

Zakres niniejszej rozprawy jest następujący:

W rozdziale 3 przybliżono ogólny problem monitoringu rozproszonych środowisk obliczeniowych ze szczególnym uwzględnieniem chmury obliczeniowej, dokonano przeglądu i analizy istniejących systemów monitoringu, przedstawiono istniejące taksonomie systemów monitoringu chmur oraz zaproponowano rozszerzenie jednej z nich. Na końcu rozdziału dokonano formalnej definicji modelu monitoringu.

Rozdział 4 poświęcony jest systemom agentowym. W ramach pracy opisano koncepcję agenta i jego środowiska działania, dokonano przeglądu istniejących architektur agentowych oraz opisano zależności występujące w systemach wieloagentowych.

Rozdział 5 opisuje problem harmonogramowania zadań. W ramach tego rozdziału przedstawiono istniejące klasyfikacje problemów harmonogramowania oraz metody ich rozwiązywania – szczególny nacisk został położony na harmonogramowanie niezależnych zadań obliczeniowych w trybie pakietowym. Następnie zdefiniowano problem i kryteria harmonogramowania oraz zaproponowano tzw. profile modeli zadań i jednostek obliczeniowych. W ostatnim podrozdziale przedstawiono ideę ewolucyjnego rozwiązywania problemu harmonogramowania zadań.

Rozdział 6 przedstawia oryginalne opracowania i modele dotyczące procesu harmonogramowania i jego monitoringu. Rozdział zawiera opis reprezentacji populacji dla problemu harmonogramowania niezależnych zadań obliczeniowych w trybie pakietowym, formalny opis dynamiki rozważanego środowiska obliczeniowego, opis procesu generowania harmonogramów, model wieloagentowego systemu monitoringu, a także formalne definicje sposobu działania zaproponowanego modelu.

W rozdziale 7 przedstawiono rezultaty przeprowadzonych badań, w ramach których zbadano wpływ zaproponowanych modeli na przebieg procesu harmonogramowania i jakość

uzyskiwanych rozwiązań. W ramach rozdziału przetestowano wszystkie zaproponowane typy agentów występujące w modelu.

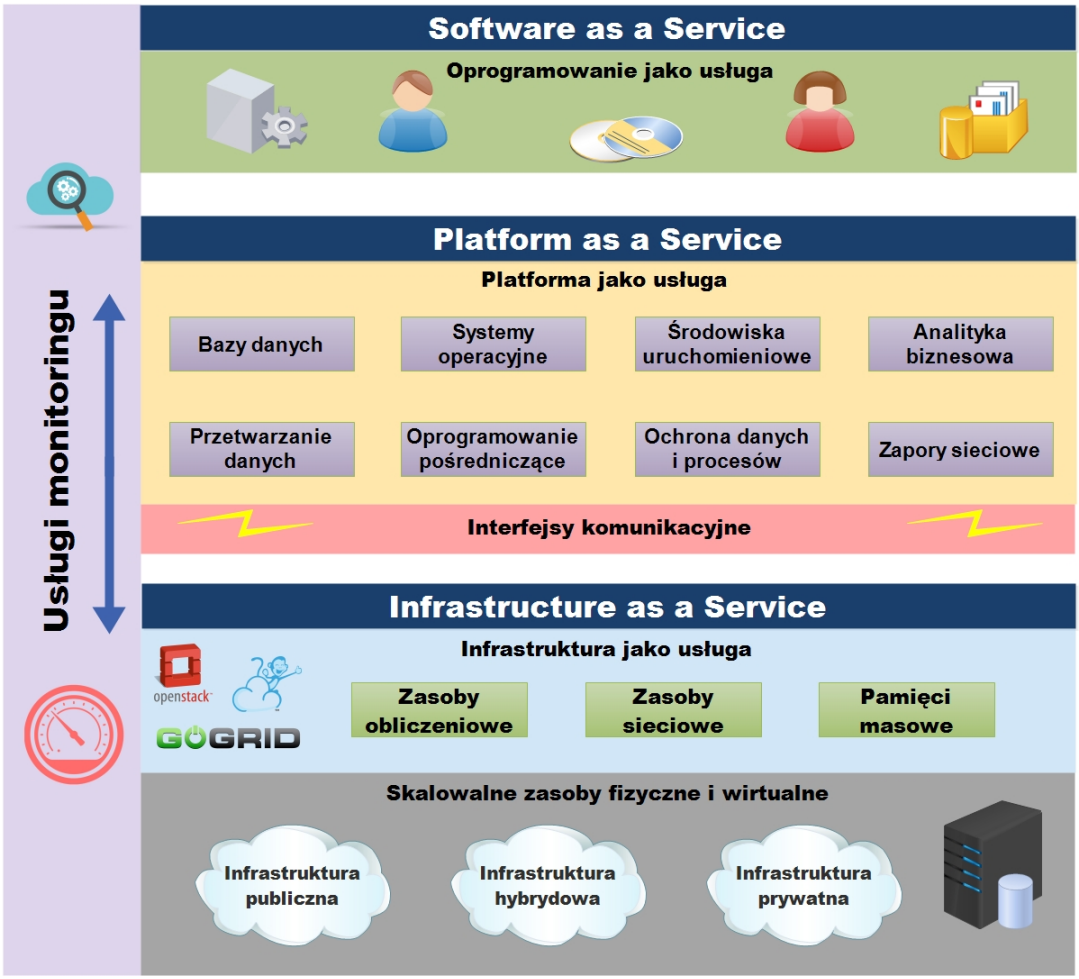
Rozdział 8 zawiera podsumowanie niniejszej pracy, w tym najważniejsze wnioski i wkład własny autora w problematykę monitorowania i wsparcia procesów harmonogramowania zadań.

Rozdział 3

Monitoring rozproszonych systemów obliczeniowych

Efektywne wykorzystanie rozproszonych środowisk obliczeniowych dużej skali, ich zarządzanie, a także zapewnienie odpowiedniego poziomu oferowanych usług wymaga zaawansowanych technik monitoringu. Monitoring takich środowisk jest trudnym wyzwaniem, które wymusza zastosowanie odpowiednich rozwiązań (narzędzi i paradygmatów). Jednocześnie odgrywa on kluczową rolę w praktycznie każdym obszarze związanym z planowaniem oraz użytkowaniem rozproszonego środowiska.

W uproszczeniu, system monitoringu może być opisany za pomocą procesu składającego się z następujących trzech etapów: (i) pomiaru stanów środowiska, (ii) kompleksowej analizy tych pomiarów oraz (iii) podjęcia odpowiednich decyzji na podstawie wyników analizy [118]. Najbardziej podstawowe narzędzia monitoringu (jak np. *df*, *top* czy *uptime* systemów Unix'owych [5]) ograniczają się tylko do pomiarów oraz analizy stanów systemu operacyjnego oraz sprzętu komputerowego lub wykonaniu wcześniej zdefiniowanych reguł (np. wyrażenia typu *cron* [3], które pozwalają na uruchomienie zadań zgodnie ze zdefiniowanym harmonogramem). Wszystkie te narzędzia inicjowane oraz konfigurowane są przez użytkownika. Biorąc pod uwagę dużą złożoność oraz skalę rozproszonych systemów obliczeniowych, jak również ich dynamiczny charakter, pojawia się konieczność możliwie maksymalnej automatyzacji procesów monitoringu. Automatyzacja ta ma na celu redukcję zaangażowania użytkowników – zarówno administratorów, jak i klientów. Podjęte przez system monitorujący



Rysunek 3.1: Idea chmury obliczeniowej wraz z usługami monitoringu obejmującymi wszystkie aspekty działania środowiska.

automatyczne działania pozwolą na wsparcie działania środowiska.

Obecnie moduły monitorujące są nieodłącznymi elementami zaawansowanych rozproszonych środowisk obliczeniowych. Przykładem takiego środowiska może być chmura obliczeniowa, której idea została przedstawiona na rysunku 3.1. Potrzeba monitoringu dotyczy prawie każdego aspektu działania środowiska rozproszonego. Do najpopularniejszych obszarów możemy zaliczyć planowanie zasobów fizycznych, zapewnienie odpowiedniego poziomu bezpieczeństwa, detekcję błędów i awarii, przydział zadań czy monitoring wydajności jednostek obliczeniowych. W przypadku chmur obliczeniowych ten zakres ulega poszerzeniu o jeszcze bardziej szczegółowe aspekty – głównie związane z ideą wirtualizacji i globalnego rozproszenia zasobów. Należy do nich zaliczyć m.in.: planowanie odpowiedniej infrastruktury wirtualnej, dużą zmienność wydajności maszyn wirtualnych, konieczność realizacji zapisów

umowy o gwarantowanym poziomie świadczenia usług (SLA), wielopoziomową abstrakcję architektury środowiska czy opóźnienia związane z tworzeniem i rekonfiguracją infrastruktury wirtualnej [118]. Dodatkowo, w przypadku środowisk chmurowych mamy do czynienia z ograniczonym zaufaniem użytkowników do działania systemu [72]. Nakłada to dodatkowe obowiązki na dostawcę usług związane z monitorowaniem środowiska, w szczególności obszarów związanych z naliczaniem opłat czy bezpieczeństwem przechowywania i transmisji danych [38]. Mając na względzie szeroki zakres obszarów wymagających monitoringu, prawidłowa klasyfikacja problemu monitorowania wymaga zarówno przeglądu istniejących rozwiązań, jak i odpowiedniej taksonomii systemów monitoringu.

3.1. Przegląd wybranych systemów monitoringu

W literaturze dostępnych jest kilka prac, będących przeglądem dostępnych rozwiązań monitoringu chmur obliczeniowych, m.in.: [14, 15, 38, 76, 118]. W ramach niniejszej rozprawy ograniczono się do rozwiązań najbardziej bliskich zaproponowanej idei agentowego monitorowania procesów harmonogramowania zadań, a także systemów najbardziej popularnych.

Amazon Web Services (AWS) jest obecnie największą i najbardziej popularną platformą oferującą usługi w chmurach. Platforma ta udostępnia szereg elastycznych i przystępnych cenowo zasobów, narzędzi oraz usług. Do najbardziej popularnych można zaliczyć usługi obliczeniowe Amazon EC2 oraz AWS Lambda, usługi przechowywania danych Amazon S3 oraz Amazon EBS czy usługi DNS realizowane przez Amazon Route 53. W sumie platforma AWS dostarcza ponad 90 różnych usług. Tak rozbudowana oferta wymaga kompleksowego monitoringu wszystkich aspektów działania platformy. Amazon CloudWatch jest to komercyjny serwis wchodzący w skład chmury Amazon Web Services pozwalający na zbieranie danych statystycznych, monitorowanie plików dziennika, ustawianie alarmów oraz automatyczne reakcje na zmiany związane z zasobami. Moduł ten pozwala zobrazować aktualny stan systemu (zużycie zasobów, wydajność aplikacji) dla najpopularniejszych usług: Amazon EC2, Amazon EBS, Elastic Load Balancers oraz Amazon RDS DB. Monitorowanie zasobów odbywa się w niemal rzeczywistym czasie. Dane takie jak wykorzystanie procesora, opóźnienia

czy liczba żądań są automatycznie zbierane dla wszystkich monitorowanych zasobów. System daje również możliwość definicji własnych metryk monitoringu poszczególnych zasobów, jak np. wykorzystanie pamięci, rozmiary transakcji czy wskaźniki błędów. Amazon CloudWatch dostarcza także narzędzia pozwalające na definiowanie reguł (wyrażenia typu *cron*), które są samoczynnie wykonywane zgodnie z ustalonym harmonogramem. Każda taka akcja musi zostać zdefiniowana przez użytkownika. Szczegółowe informacje na temat sposobu implementacji i działania Amazon CloudWatch są ukryte przed użytkownikami platformy. Należy również zauważyć, że narzędzie to nie pozwala na monitorowanie infrastruktury łączącej ze sobą platformy rożnych usługodawców (ang. *multi-cloud*).

Platforma AWS oferuje również rodzinę usług Amazon EC2 Container Service (Amazon ECS), która zarządza kontenerami usług oraz wirtualnymi maszynami. W ramach Amazon ECS udostępniona jest możliwość harmonogramowania zadań w oparciu o:

(i) automatyczny harmonogram usług (ang. *Service Scheduler*) – harmonogramowanie to opiera się na algorytmach równoważących obciążenie (ang. *load balancing*) dostępnych zasobów;

(ii) ręczne uruchamianie zadań (ang. *Manually Running Tasks*) – oparte o metody dostarczone przez dedykowane API, które użytkownikowi pozostawiają decyzję o wyborze strategii przypisania przychodzącego zadania (domyślnie jest to przydział losowy);

(iii) uruchamianie zadań w oparciu o harmonogram zbudowany z reguł typu cron (ang. *Running Tasks on a cron-like Schedule*) – pozwala użytkownikowi na zdefiniowanie reguł wykonania określonych zadań (np. co określony interwał lub w odpowiedzi na zdarzenie);

(iv) niestandardowe moduły harmonogramowania (ang. *Custom Schedulers*) – Amazon ECS pozwala tworzyć własne moduły odpowiedzialne za przydział zadań, które realizują zgodnie z implementacją swoje cele;

(v) przypisywanie zadań w oparciu o Amazon ECS Task Placement – moduł ten pozwala na przypisanie zadań uwzględniając ich wymagania (np. moc obliczeniową i pamięć),

a także strategie przypisania (losowa, równomierna na podstawie określonego atrybutu, minimalizująca liczbę instancji w użyciu) oraz ograniczenia.

Procesy te korzystają z pomiarów systemu monitorującego, lecz sam system monitoringu nie oferuje aktywnego wsparcia procesów harmonogramowania. Dodatkowo, oficjalna dokumentacja nie wspomina o inteligentnych narzędziach biorących udział w procesach harmonogramowania oraz weryfikacji wykonania harmonogramów.

Innym, istotnym z punktu widzenia niniejszej pracy, systemem monitoringu środowisk rozproszonych jest MonALISA (MONitoring Agents using a Large Integrated Services Architecture) [71]. Jest to zestaw narzędzi do monitorowania, kontroli oraz optymalizacji działania systemów rozproszonych oparty na paradygmacie agentowym. Główny nacisk rozwiązania został położony na rozproszenie modułów agentowych i ich zdolności komunikacyjne oparte o popularną architekturę sieciową Jini (obecnie Apache River). Rolą agentów jest zbieranie i analiza informacji wykorzystywanych w procesie optymalizacji doboru ścieżek przetwarzania zadań oraz transferu danych. Przetwarzanie informacji odbywa się w sposób rozproszony, dzięki czemu monitorowane środowisko może być wydajnie zarządzane w czasie rzeczywistym. Zaproponowane agenty skupiają się głównie na zbieraniu informacji o poszczególnych węzłach obliczeniowych, połączeniach sieciowych oraz wydajności wykonywanych usług. Mają one również możliwość interakcji z innymi usługami w celu dostarczenia rezultatów monitoringu, natomiast nie implementują mechanizmów pozwalających na podejmowanie bardziej złożonych działań. System oferuje możliwość współpracy z popularnymi narzędziami monitoringu jak Ganglia, Nagios, MRTG czy Hawkeye.

D. Zubok i wsp. w swojej pracy [129] z 2016 roku zaproponowali model agentowego systemu monitorującego chmurę obliczeniową z dynamicznie zmieniającą się konfiguracją środowiska. Autorzy proponują system składający się z trzech typów agentów, które monitorują odpowiednio: wydajność systemu, przychodzące zadania oraz obecny stan zasobów. Pierwszy typ dokonuje pomiarów wydajności systemu w równych odstępach czasowych korzystając z metody średniej kroczącej. Drugi rodzaj agentów monitoruje częstotliwość przychodzenia zadań. Trzeci typ agentów zbiera informacje o aktualnym poziomie dostępnych zasobów

(czasu obliczeniowego, pamięci oraz wolnej przestrzeni dyskowej). W pracy zwrócono szczególną uwagę na problem ciągłego odpytywania (ang. *constant polling*), czyli sprawdzania stanu zasobów obliczeniowych. Podejście takie jest obciążające zarówno dla jednostek obliczeniowych jak i jednostki zarządzającej. W związku z powyższym autorzy proponują aktualizację stanów zasobów na żądanie, realizowaną poprzez system agentowy. Zaproponowane agenty decydują, czy posiadana wiedza o stanie środowiska jest aktualna oraz czy zachodzące zmiany są wystarczająco duże. Na podstawie tej wiedzy oraz z góry ustalonych progów, podejmowane są odpowiednie działania, np. przekierowanie strumienia zadań. Rozwiązanie to nie posiada mechanizmów inteligentnych – opiera się na badaniu stanów środowiska oraz współistnieniu wartości progowych.

Astrolabe [116] jest to system zarządzania informacjami pochodzącymi z rozproszonego środowiska. Jego celem jest zbieranie danych na temat stanów poszczególnych komponentów środowiska, ich szybka aktualizacja oraz agregacja atrybutów opisujących środowisko. System dostarcza możliwości lokalizacji zasobów, a także oferuje skalowalny sposób śledzenia stanu środowiska w miarę upływu czasu oraz rozbudowy środowiska. Autorzy rozwiązania szczególny nacisk położyli na możliwość skalowania systemu monitoringu i jego adaptację do monitorowanego środowiska. Komunikacja w systemie Astrolabe opiera się na modelu równorzędnym (ang. *peer-to-peer*) i wykorzystuje pewne formy zapytań SQL stosowane w agregacji. System, według autorów, ma możliwość obsługi środowiska zbudowanego z tysięcy, a nawet miliona węzłów z czasem propagacji sięgającym dziesiątek sekund. Autorzy rozwiązanie opierają o uproszczone pojęcie jednego typu agenta, który odpowiada za cały proces monitoringu jednostki obliczeniowej.

W pracy [34] M. Dhingry i wsp. zaproponowano platformę programistyczną (ang. *framework*) zbierającą ogólne dane dotyczące zużycia zasobów. Platforma ta składa się z czterech elementów: (i) interfejsu użytkownika, który pozwala na wybór metryk monitoringu stosownie do wymagań; (ii) agenta wirtualnej maszyny, który zbiera dane dotyczące użycia zasobów przypisanej sobie wirtualnej maszyny; (iii) agenta nadzorcy, który monitoruje całą fizyczną jednostkę, w której działają wirtualne maszyny oraz (iv) kolektora metryk, który zbiera zestawy przesłanych przez agenty danych. W porównaniu do innych, podobnych roz-

wiązań, autorzy wyróżniają pomiary infrastruktury wirtualnej (agenty wirtualnych maszyn) oraz infrastruktury fizycznej (agenty nadzorców).

A. Meera i S. Swamynathan w pracy [74] zaproponowali system monitoringu środowisk chmurowych oparty o podejście agentowe. Każda maszyna wirtualna wyposażona jest w nadzorującego jej stan agenta (*VmRM agent*). Zadaniem agentów jest pomiar wykorzystania mocy obliczeniowej procesora oraz pamięci operacyjnej. System monitorujący operuje na poziomie modelu *Infrastructure as a Service* (IaaS, z ang. infrastruktura jako usługa), dokonując pomiarów odpowiednich zwirtualizowanych zasobów. Zebrane dane zostają przesłane do centralnego systemu monitorującego składającego się z dwóch komponentów: kolektora oraz modułu raportującego. Kolektor odpowiada za odbiór i przechowywanie pomiarów, natomiast moduł raportujący informuje administratora o stanie wirtualnej infrastruktury oraz przesyła dane do panelu sterowania systemu monitoringu. Autorzy pracy nie wspominają o inteligentnych mechanizmach agentów ani rozbudowanych procesach decyzyjnych.

Wiele miejsca zostało poświęconego problemowi monitorowania sieci. Jednym z popularnych rozwiązań jest standard sFlow opisany dokumentem RFC 3176 [83]. Opiera się on na trzech podstawowych własnościach:

(i) możliwości pomiarów ruchu w sieci, gromadzenia, przechowywania i analizowania danych o ruchu – daje to możliwość zarządzania rozległą siecią z jednego miejsca;

(ii) skalowalności, która pozwala na monitorowanie połączeń bez wpływu na wydajność sprzętu sieciowego oraz bez znacznego obciążenia sieci;

(iii) niskim koszcie implementacji – rozwiązanie to zostało wdrożone zarówno w prostych przełącznikach L2 jak i w zaawansowanych routerach bez konieczności zwiększania pamięci podręcznej.

sFlow opiera się na architekturze agentowej – każde urządzenie jest wyposażone w lekkiego agenta, który ma zdolność ciągłego monitoringu wszystkich interfejsów jednocześnie. Zebrane dane (w postaci dedykowanego datagramu) są niezwłocznie wysyłane do centralnego kolektora, gdzie są analizowane. Na podstawie otrzymanych informacji kolektor jest w stanie

odtworzyć dokładny, aktualny obraz przepływu danych w sieci.

Kolejnym przykładem zestawu narzędzi monitoringu chmur jest New Relic Digital Intelligence Platform [4], który udostępnia wszechstronne rozwiązania dostarczane w modelu *Software-as-a-Service* (SaaS, z ang. oprogramowanie jako usługa). Platforma New Relic składa się z siedmiu rozszerzalnych modułów, do zadań których zalicza się monitorowanie zasobów oraz działań użytkowników, powiadamianie o zmianach stanów, analiza danych w czasie rzeczywistym, wizualizacja stanu środowiska i inne.

Monitoring opiera się na działaniu dedykowanego agenta dostępnego w siedmiu technologiach programistycznych: Go, Java, .NET, Node.js, PHP, Python oraz Ruby. Agent przesyła zebrane dane do centrum sterowania, które odpowiednio je przetwarza i wizualizuje. System kładzie szczególny nacisk na analizowanie wydajności aplikacji, czasu odpowiedzi, żądań przesyłanych do usług, jak również dokonuje pomiarów standardowych metryk: zużycia CPU, pamięci operacyjnej czy przestrzeni dyskowej. New Relic umożliwia współpracę z popularnymi platformami chmurowymi, takimi jak Amazon AWS, Azure, OpenStack czy HP Cloud Services.

Innym, podobnym rozwiązaniem jest IDERA Uptime Cloud Monitor (poprzednio CopperEgg) [54]. Jest to system pozwalający na monitorowanie całej infrastruktury chmury, a także – bazując na aktualnych pomiarach i historycznych danych – identyfikację potencjalnych problemów. Dostarczany jest w formule SaaS, w której dokonuje się opłat tylko za wykorzystywane funkcjonalności. Opiera się na otwartym API, które gwarantuje skalowalność i łatwość wdrożenia – również w komercyjnych rozwiązaniach chmurowych, bazodanowych, serwerach WWW i innych. Integracja z szeroką listą systemów pozwala na rozszerzenie metryk i obszarów dokonywanych pomiarów.

Monitorowanie zasobów przez Uptime Cloud Monitor opiera się na działaniu lekkiego agenta, który – podobnie jak w przypadku New Relic – dokonane pomiary przekazuje do centralnej jednostki przetwarzającej i wizualizującej otrzymane dane.

Warto również wspomnieć o innym narzędziu firmy IDERA – SQL Enterprise Job Manager [53], który oferuje wsparcie dla harmonogramowania zadań wykonywanych w ra-

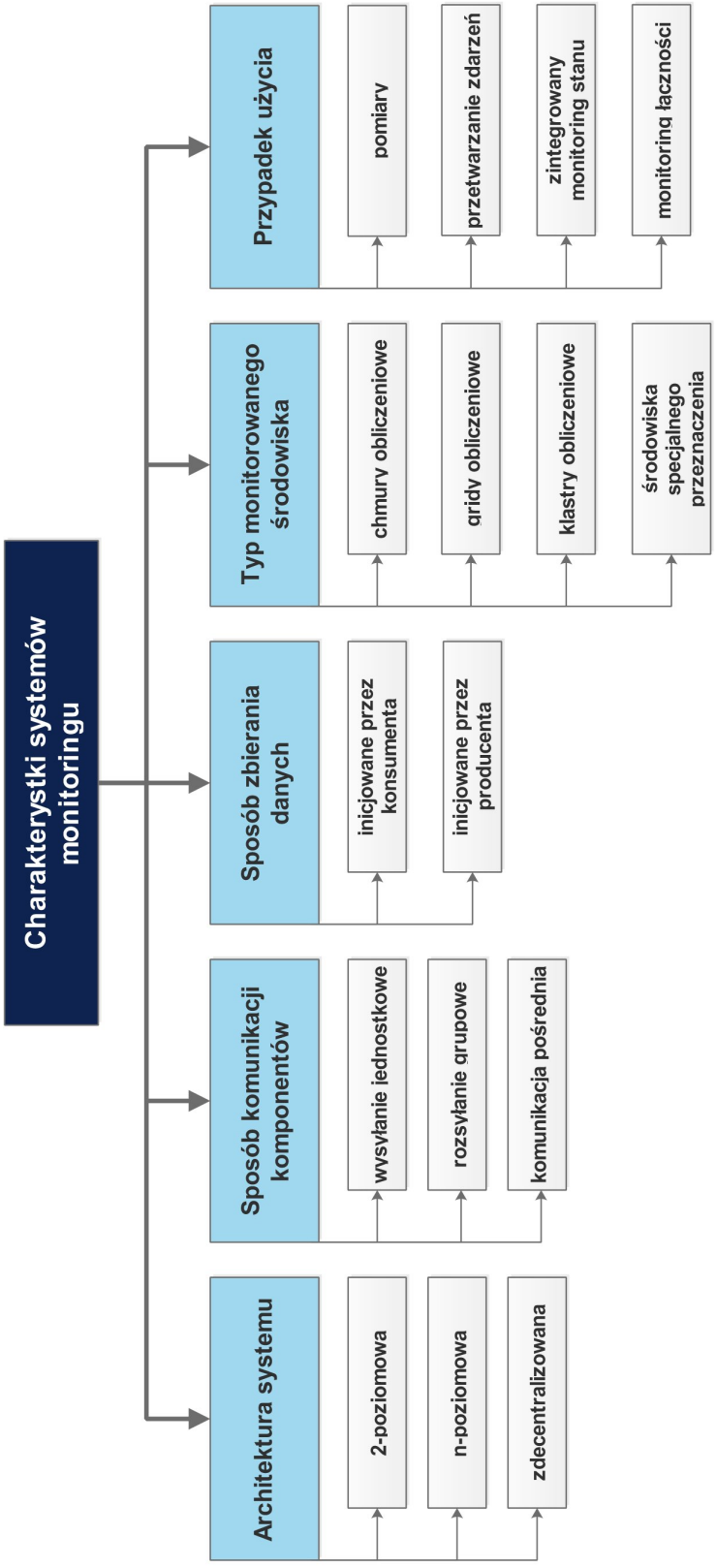
mach oprogramowania MS SQL Server. Są to zadania administracyjne związane z działaniem i utrzymaniem baz SQL Server (np. kopie zapasowe). Oprogramowanie – podobnie jak Uptime Cloud Monitor – działa w oparciu o system agentowy, który przygotowuje i nadzoruje wykonanie harmonogramów.

Powyżej przedstawiono systemy monitoringu oparte o paradygmat agentowy, a także rozwiązania, które dokonują pomiarów wykorzystywanych w procesie harmonogramowania zadań. Innymi, popularnymi i bardzo rozbudowanymi narzędziami monitoringu są m.in. Ganglia [73], Nagios [20], Opsview [122], CloudHarmony [1] czy SPAE Server Monitoring [9], jednak nie poruszają one problemu harmonogramowania zadań oraz systemów agentowych. Powyższy przegląd pokazuje, że zastosowanie lekkiego systemu agentowego, działającego w tle i mającego minimalny wpływ na pracę zasobów, jest częstym rozwiązaniem. Jednocześnie należy zwrócić uwagę na pomijanie (lub też minimalne wsparcie) procesów harmonogramowania zadań przez takie systemy. Agenty, które implementowane są w przytoczonych rozwiązaniach zwykle nie operują inteligentnymi mechanizmami, jak również rozbudowaną komunikacją. W związku z tym pozostaje szerokie pole do rozwoju systemów monitorujących rozproszone środowiska obliczeniowe.

3.2. Przegląd dostępnych taksonomii monitoringu

Przegląd dostępnych rozwiązań oraz analiza zarówno architektury jak i funkcjonalności pozwala na sprecyzowanie klas systemów monitoringu. W dotychczas opublikowanych pracach naukowych zaproponowano dwie taksonomie systemów monitoringu chmur obliczeniowych. Obie prezentują odmienne podejścia.

W pierwszej pracy [118] autorzy J. S. Ward i A. Barker opierają zaproponowaną taksonomię na pięciu cechach, które charakteryzują systemy monitorujące. Są to: (i) architektura systemu (ang. *architecture*), (ii) sposób komunikacji poszczególnych komponentów (ang. *communication*), (iii) sposób zbierania danych (ang. *collection*), (iv) typ monitorowanego środowiska (ang. *origin*) oraz (v) przypadek użycia (ang. *use case*). Schemat taksonomii przedstawiony jest na rysunku 3.2.



Rysunek 3.2: Taksonomia systemów monitoringu zaproponowana w pracy J. S. Ward i A. Barker: [118].

(i) Architektura systemu może być zcentralizowana lub zdecentralizowana. W architekturze scentralizowanej możemy wyróżnić architekturę 2- lub N-poziomową. Architektura 2-poziomowa zakłada współistnienie dwóch typów komponentów: producentów i konsumentów. Producenci dokonują pomiarów monitorowanych elementów, natomiast konsumenci gromadzą dane uzyskane przez producentów, jak również dokonują ich analizy. Architektura N-poziomowa zakłada dodatkowo współistnienie pośrednich elementów, które również gromadzą i wstępnie analizują zebrane przez producentów dane, a następnie przekazują je w określonym kierunku do warstwy wyższej.

Architektura zdecentralizowana zbudowana jest z komponentów realizujących te same zadania, które są realizowane w przypadku architektury zcentralizowanej, z tą różnicą, że komponenty te mają możliwość komunikacji i przesyłu danych w różnych kierunkach w zależności od umiejscowienia, zachowań i innych czynników.

J. S. Ward i A. Barker w swojej pracy utożsamiają komponenty systemu monitorującego z agentami, które realizują funkcje monitoringu.

(ii) Kolejną cechą jest sposób komunikacji pomiędzy komponentami systemu monitorującego. Wyróżnione są trzy typy komunikacji: (a) wysyłanie jednostkowe (ang. *unicast*), (b) rozsyłanie grupowe (ang. *multicast*) oraz (c) komunikacja pośrednia (ang. *middleware*). W przypadku komunikacji typu jednostkowego komponenty komunikują się metodą 1 do 1 (ang. *one-to-one*). Rozsyłanie grupowe zakłada przesyłanie informacji do grupy zdefiniowanych odbiorców (np. za pomocą adresu IP typu *multicast*). Ostatni typ komunikacji zakłada współistnienie oprogramowania pośredniczącego umożliwiającego komunikację (takiego jak np. kolejki, brokerzy wiadomości, dedykowane usługi lub systemy operacyjne).

(iii) Przesyłanie danych w systemach monitorujących może być inicjowane zarówno przez konsumenta (ang. *pull*), jak i producenta (ang. *push*). W przypadku inicjalizacji przesyłania przez producenta, to on podejmuje decyzję kiedy następuje odpowiedni moment przesłania zebranych pomiarów. Natomiast w przypadku drugiego scenariusza to konsument wysyła żądanie przesłania zebranych danych.

(iv) Zaproponowana taksonomia uwzględnia szereg rozproszonych środowisk, w których może działać system. Autorzy zaliczają do nich: chmury, klastry, gridy oraz wyspecjalizowane rozproszone środowiska dla przedsiębiorstw (ang. *enterprise computing*). Pominięte w taksonomii zostały natomiast systemy typu przetwarzanie we mgle (ang. *fog computing*) czy Internet rzeczy (ang. *Internet of Things*).

(v) Ostatnią cechą opisującą system monitoringu są przypadki użycia. W pracy zostały zdefiniowane cztery główne operacje systemu monitoringu:

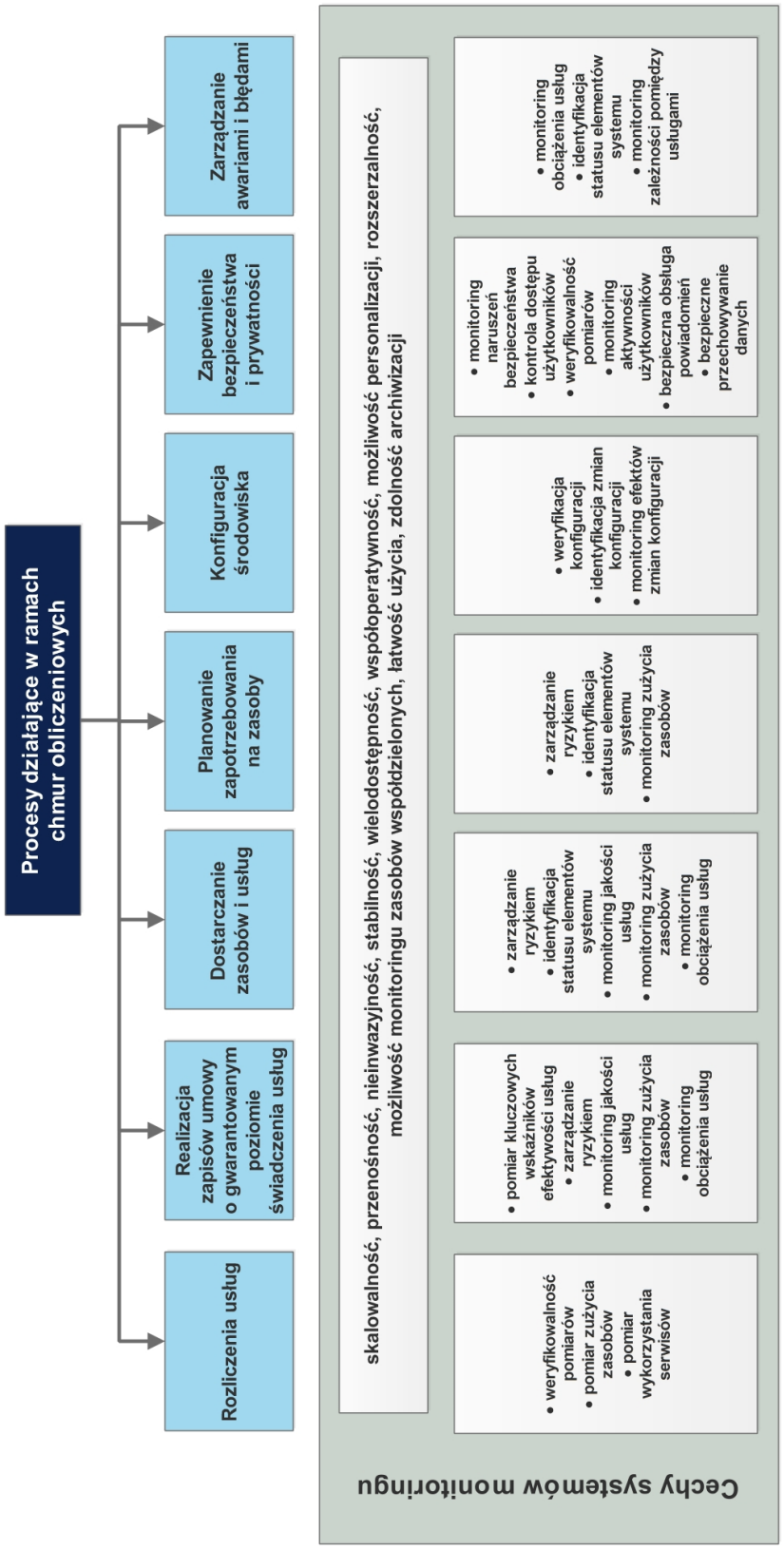
   (a) pomiary (ang. *metric collection*) – najbardziej podstawowa funkcjonalność każdego systemu monitorującego. Polega na zbieraniu danych statystycznych na temat wydajności systemu oraz użyciu zasobów; nie obejmuje kompleksowej analizy zebranych danych.

   (b) przetwarzanie zdarzeń (ang. *event processing*) – polega na ciągłej analizie zachodzących zdarzeń. Zwykle narzędzia przetwarzają całe strumienie zdarzeń na podstawie zdefiniowanych przez użytkowników instrukcji. Ten typ wymaga większych zasobów obliczeniowych ze względu na przetwarzanie strumieni w czasie rzeczywistym.

   (c) zintegrowany monitoring stanu (ang. *health monitoring*) – podobnie jak w przypadku pomiarów, ten typ również dostarcza informacje opisujące stan danej usługi. Dodatkowo, monitoring zintegrowany zakłada interakcję z usługami, np. komunikację.

   (d) monitoring łączności (ang. *network monitoring*) – są to operacje skupione głównie na pomiarach związanych z działaniem sieci – jej wydajności, urządzeń oraz usług sieciowych (np. detekcja ataków sieciowych czy problemów z trasowaniem pakietów).

Inne podejście do klasyfikacji systemów monitorujących środowiska chmurowe zostało przedstawione w pracy K. Fatema i wsp.: [38]. Autorzy tej pracy oparli taksonomię o procesy związane z funkcjonowaniem i zarządzaniem rozproszonych środowisk obliczeniowych, a także cechy narzędzi monitorujących. Rysunek 3.3 obrazuje zaproponowaną taksonomię.



Rysunek 3.3: Taksonomia systemów monitoringu zaproponowana w pracy K. Fatema i wsp.: [38].

W pierwszej kolejności zdefiniowano cechy systemów monitoringu, które są najistotniejsze w kontekście projektowania oraz wyboru narzędzi monitorujących. Możemy do nich zaliczyć:

(i) skalowalność (ang. *scalability*) – chmury obliczeniowe są środowiskami rozproszonymi, mogącymi składać się z tysięcy węzłów obliczeniowych. Mając na uwadze tą cechę, system monitoringu powinien również cechować się możliwością skalowalności, tak, aby móc nadzorować i zbierać dane bez opóźnień.

(ii) przenośność (ang. *portability*) – chmury często są środowiskami heterogenicznymi (zarówno pod kątem platform jak i usług), stąd systemy monitorujące powinny być niezależne od platformy, na której działają.

(iii) nieinwazyjność (ang. *non-intrusiveness*) – działanie systemu monitorującego powinno być w jak najmniejszym stopniu zauważalne, innymi słowy: ilość zasobów potrzebnych do działania narzędzia monitorującego (szczególnie mając na uwadze mnogość monitorowanych obszarów i zasobów) powinna być możliwie jak najmniejsza.

(iv) stabilność (ang. *robustness*) – chmury są bardzo dynamicznymi środowiskami, dlatego odpowiedni monitoring powinien nie tylko zapewniać możliwość detekcji zmian zachodzących w monitorowanych obszarach, ale także umiejętność dostosowywania się do zmieniającego środowiska i ciągłego działania.

(v) wielodostępność (ang. *multi-tenancy*) – usługi oferowane w ramach chmur obliczeniowych opierają się o ideę wielodostępu, czyli możliwości jednoczesnego korzystania z zasobów przez wielu użytkowników. Podobną cechą powinien charakteryzować się system monitorujący – udostępniać ogólnodostępne dane wszystkim użytkownikom systemu, a informacje chronione odpowiednio izolować i udostępniać tylko klientom upoważnionym.

(vi) współoperatywność (ang. *interoperability*) – jest to cecha reprezentująca umiejętność współpracy i wymiany danych pomiędzy środowiskami rozproszonymi. Obecnie popularność zdobywają tzw. federacje chmur (ang. *cloud federation*), będące rozszerzeniem

możliwości chmury jednego dostawcy poprzez odpłatny dostęp do zasobów innego dostawcy. Nowoczesne systemy monitoringu powinny być w stanie wymieniać informacje pomiędzy wszystkimi komponentami środowiska, również komponentami dostarczanymi z zewnątrz.

(vii) możliwość personalizacji (ang. *customizability*) – wiele usług oferowanych w ramach przetwarzania w chmurze może być odpowiednio przystosowanych do konkretnych potrzeb użytkownika końcowego. Jest to wyzwanie również dla systemów monitorujących, które podobnie powinny dać możliwość wyboru pożądanych metryk monitoringu.

(viii) rozszerzalność (ang. *extensibility*) – wraz z szybkim rozwojem technologii chmurowych, szczególnie w obszarze zarządzania, pojawia się potrzeba rozwoju również systemów monitorowania. Narzędzia monitoringu powinny być rozszerzalne oraz mieć możliwość dostosowywania się do nowych wersji środowisk, np. poprzez wprowadzanie nowych metryk monitoringu.

(ix) możliwość monitoringu zasobów współdzielonych (ang. *shared resource monitoring*) – chmura opiera się na technologii wirtualizacji zasobów fizycznych, dzięki czemu uzyskuje się izolowane środowiska, czyli wirtualne maszyny. Możliwość monitoringu zasobów współdzielonych jest podstawową cechą systemów monitoringu. Systemy takie mają w efekcie za zadanie zapobiegać współzawodniczeniu wirtualnych maszyn (czy konkretnych aplikacji) o zasoby fizyczne.

(x) łatwość użycia (ang. *usability*) – jest to cecha bardzo istotna z punktu widzenia użytkownika. Systemy monitoringu powinny być możliwie jak najbardziej intuicyjne, a nawet być w stanie podejmować odpowiednie działania bez nadmiernej ingerencji użytkownika.

(xi) zdolność archiwizacji (ang. *archivability*) – głównym celem systemu monitorującego jest zbieranie informacji. Funkcjonalność ta jest nierozłącznie związana ze zdolnością do ich przechowywania. Historyczne dane dotyczące działania środowiska rozproszonego mogą być bardzo użyteczne, a czasem wręcz niezbędne, w procesie rzetelnej analizy i identyfikacji przyczyn problemów (szczególnie występujących na przestrzeni czasu).

Wszystkie powyższe cechy dotyczą każdego aspektu działania monitoringu chmur obliczeniowych. Ponadto autorzy wyszczególnili dodatkowe cechy, które odnoszą się do wybranych procesów zachodzących w chmurach. Należą do nich następujące możliwości: weryfikowalność pomiarów (ang. *verifiable measuring*), pomiar zużycia zasobów (ang. *resource usage metering*), pomiar wykorzystania serwisów (ang. *service usage metering*), pomiar kluczowych wskaźników efektywości usług (ang. *service KPI monitoring*), monitoring jakości usług (ang. *QoS monitoring*), monitoring zużycia zasobów (ang. *resource usage monitoring*), zarządzanie ryzykiem (ang. *risk assessment*), identyfikacja statusu elementów systemu (ang. *component status identification*), monitoring obciążenia usług (ang. *service load monitoring*), weryfikacja konfiguracji (ang. *configuration verification*), identyfikacja zmian konfiguracji (ang. *configuration drift identification*), monitoring efektów zmian konfiguracji (ang. *configuration effect monitoring*), monitoring naruszeń bezpieczeństwa (ang. *security breach monitoring*), kontrola dostępu użytkowników (ang. *user access control*), monitoring aktywności użytkowników (ang. *user activity monitoring*), bezpieczna obsługa powiadomień (ang. *secured notification*), bezpieczne przechowywanie danych (ang. *secured storage*), monitoring zależności pomiędzy usługami (ang. *service dependency monitoring*).

W ramach pracy [38] wyróżniono również siedem głównych procesów działających w ramach chmury obliczeniowej. Są to: (i) rozliczenia usług (ang. *accounting and billing*), (ii) realizacja zapisów umowy o gwarantowanym poziomie świadczenia usług (ang. *SLA management*), (iii) dostarczanie zasobów i usług (ang. *resource/service provisioning*), (iv) planowanie zapotrzebowania na zasoby (zarówno fizyczne jak i wirtualne) (ang. *capacity planning*), (v) konfiguracja środowiska (ang. *configuration management*), (vi) zapewnienie bezpieczeństwa i prywatności (ang. *security and privacy assuring*), (vii) zarządzanie awariami i błędami (ang. *fault management*). Są to procesy, które wymagają szczególnego nadzoru i wsparcia ich działania.

(i) Monitoring rozliczania usług: Środowiska oparte na świadczeniu usług wymagają wiarygodnych metod rejestracji użycia zasobów i ich rozliczenia. Dokładna księgowość uzależniona jest od zdolności dokonywania rzetelnych pomiarów użycia wirtualnych zasobów i aplikacji oraz przechowywania tych danych. Dodatkowo, odpowiedni moni-

toring i analiza wyników może pomóc w identyfikacji potencjalnych zagrożeń, jak np. wstrzyknięcia zadań [57].

(ii) Monitoring realizacji zapisów umowy o gwarantowanym poziomie świadczenia usług: Umowa o gwarantowanym poziomie świadczenia usług (SLA) określa warunki związane z zakupionymi przez klienta usługami, w tym jakość usług (ang. *Quality of Service, QoS*), ceny, polityki bezpieczeństwa czy umowne kary. Monitoring realizacji zapisów tej umowy ma zasadnicze znaczenie w jej realizacji. Do zadań systemu monitorującego należą pomiary parametrów jakości dostarczanych usług (np. wydajność wirtualnych maszyn, zużycie CPU, dostępna przestrzeń dyskowa), weryfikacja zapisów SLA z aktualnym stanem środowiska czy kontrola bezpieczeństwa.

(iii) Monitoring dostarczania zasobów i usług: Świadczenie usług i dostarczanie zwirtualizowanych zasobów wymaga odpowiedniego nadzoru, który pozwoli na optymalną alokację fizycznych zasobów. Jest to niezbędny proces w zapewnieniu elastycznego i wydajnego działania środowiska. Fizyczne zasoby mogą być udostępniane na dwa sposoby: statyczny i dynamiczny. Statyczne udostępnianie zakłada tworzenie wirtualnych maszyn o z góry określonym rozmiarze. Charakterystyki wirtualnych maszyn nie ulegają zmianom. W przypadku dynamicznego udostępniania wydajność wirtualnych maszyn może być automatycznie skalowana do bieżących potrzeb (wahań obciążenia pracą). W obu scenariuszach niezbędny jest monitoring, który pozwoli na pomiar zużycia zasobów potrzebnych do prawidłowego działania usług i zgłoszenia zapotrzebowania na dodatkowe zasoby (lub zwolnienie nadmiarowych). Dodatkowo, monitoring taki wspomaga ocenę ryzyka realizacji zapisów SLA oraz świadczenia usług o odpowiedniej jakości (QoS).

(iv) Monitoring planowania zapotrzebowania na zasoby: Odpowiednia analiza zdolności obliczeniowej infrastruktury (zarówno fizycznej jak i wirtualnej) jest szczególnie istotna w procesie planowania i tworzenia środowiska. Monitoring planowania zapotrzebowania na zasoby pozwala na zbudowanie odpowiedniej infrastruktury wirtualnej pozwalającej zaspokoić zapotrzebowanie na zasoby konieczne do zapewnienia odpowiedniej jakości usług jak i bezpieczeństwa związanego z awariami sprzętu. Możliwość pomiarów zużycia

zasobów umożliwia przewidywanie potrzeby wykorzystania większej ilości zasobów lub też określenia stopnia ich marnotrawienia.

(v) Monitoring konfiguracji środowiska: Zarządzanie konfiguracją środowiska pozwala sterować jego zachowaniem. Monitorowanie tego obszaru jest szczególnie istotne ze względu na dynamiczny charakter środowiska (np. zmiany infrastruktury fizycznej i wirtualnej). Zasoby mogą być w każdej chwili dodawane lub usuwane, co wymaga rekonfiguracji w czasie rzeczywistym. System zarządzania konfiguracją musi być w stanie zweryfikować nowe zestawy konfiguracyjne i zidentyfikować możliwe zmiany zachodzące w wyniku wprowadzenia nowych ustawień. W tym obszarze monitoring wspiera walidację nowych konfiguracji poprzez sprawdzenie efektów ich wprowadzenia.

(vi) Monitoring zapewnienia bezpieczeństwa i prywatności: Zagadnienia bezpieczeństwa są jednymi z najistotniejszych problemów w systemach chmurowych. Odpowiedni monitoring jest niezbędnym elementem pozwalającym na identyfikację wykraczających poza normy, czy nawet nieuprawnionych procesów. Jednocześnie narzędzie monitorujące musi być wiarygodne zarówno pod względem dokonywanych pomiarów jak i bezpieczeństwa.

(vii) Monitoring zarządzania awariami i błędami: Jedną z własności chmur obliczeniowych jest odporność na awarie i błędy. W przypadku awarii fizycznej infrastruktury środowisko jest automatycznie odtwarzane na innym sprzęcie. Ciągły monitoring pozwala na identyfikację, a nawet predykcję mających nastąpić awarii. Ze względu na złożoną budowę środowiska, awarie mogą dotyczyć wielu różnych obszarów, np. sprzętu, sieci, usług. Wszystkie te obszary powinny być objęte odpowiednim monitoringiem.

Powyżej szczegółowo przedstawiono dwie taksonomie systemów monitoringu chmur obliczeniowych – środowisk, które ze względu na swój charakter w sposób szczególny wymagają zaawansowanych technik monitoringu. W literaturze dostępne są także taksonomie systemów gridowych: [128], jak również taksonomia poświęcona autonomicznym systemom zarządzania aplikacjami w środowiskach przetwarzania sieciowego [89].

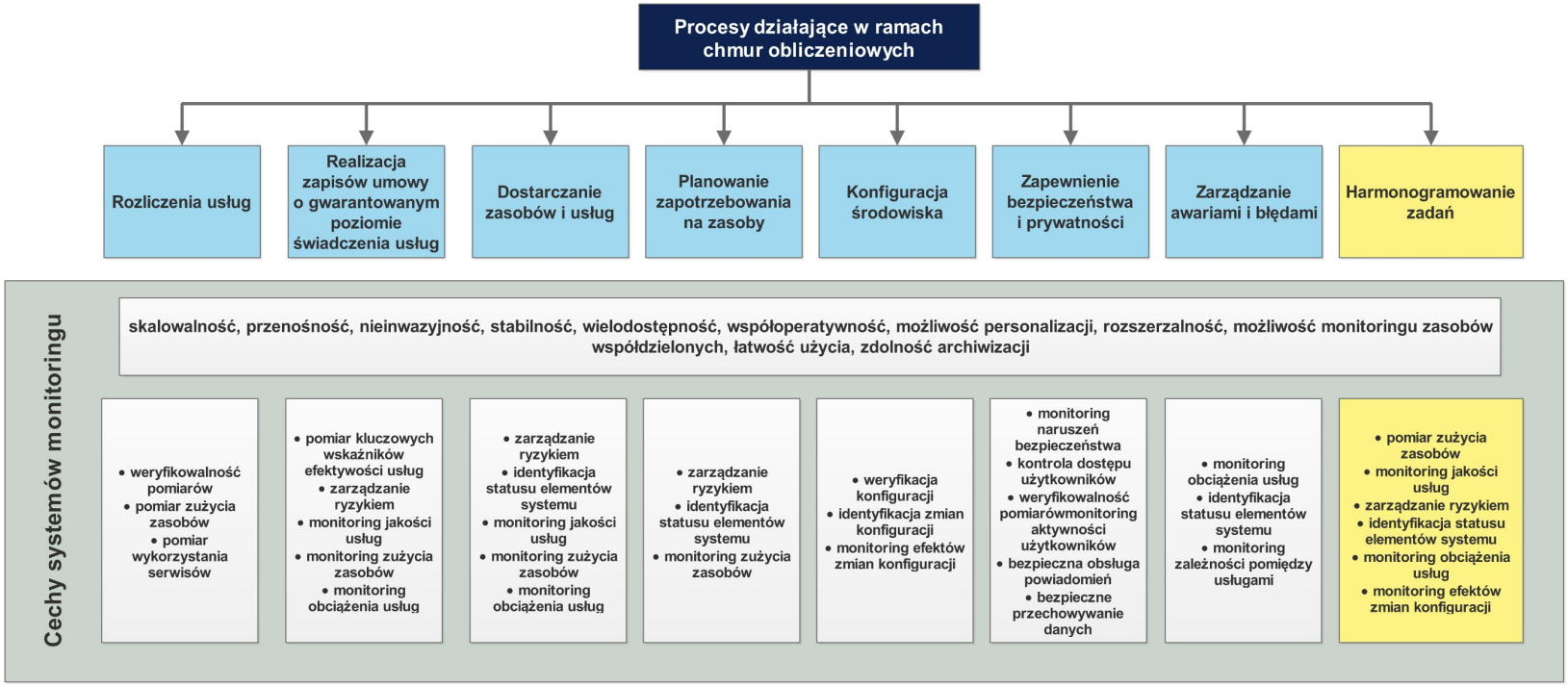
3.3. Rozszerzenie taksonomii

W pracach omówionych w poprzednim podrozdziale autorzy nie uwzględnili w taksonomiach problemu monitoringu procesu harmonogramowania zadań (ang. *task scheduling*). Jest to proces obecny szczególnie na platformach dostarczanych przez największych dostawców usług w obszarze środowisk chmurowych, jak np. Amazon Web Services, Google Cloud Platform czy Microsoft Azure. Monitoring tego procesu, jak i jego aktywne wsparcie, może mieć znaczący wpływ na optymalne użycie dostępnych wirtualnych zasobów.

Proces harmonogramowania zadań szczególnie istotny jest w rozproszonych środowiskach obliczeniowych. Chmury obliczeniowe co prawda dostarczają mechanizmów tworzenia spersonalizowanych maszyn wirtualnych czy też skalowania już istniejących, ale wiąże się to z dużym nakładem czasowym oraz okresową niedostępnością niektórych maszyn. Dodatkowo, takie podejście może być nieefektywne np. pod względem kosztów czy zużycia energii.

W związku z powyższym, proces harmonogramowania, który należy do problemów NP-zupełnych, może być kluczowym zagadnieniem w optymalnym działaniu środowiska. Jeśli weźmiemy pod uwagę dodatkowe wymagania użytkowników (jak np. spełnienie polityk bezpieczeństwa), problem ten staje się jeszcze bardziej złożony. Monitoring samego procesu, jak i otrzymanych rezultatów (pod względem spełnienia kryteriów) harmonogramowania, jest istotnym elementem działania środowiska rozproszonego. W związku z powyższym, proces harmonogramowania zadań powinien zostać ujęty w taksonomii poświęconej monitoringowi procesów zachodzących w chmurach obliczeniowych.

Harmonogramowanie zadań można ująć w taksonomii w dwojaki sposób - jako element jednego z już wyodrębnionych procesów (np. dostarczania zasobów i usług) lub w postaci nowego, odrębnego procesu. W niniejszej pracy rozszerzono taksonomię zaproponowaną przez K. Fatemę i wsp. [38] o dodatkowy proces. Rozszerzenie to zostało uwzględnione na rysunku 3.4.



Rysunek 3.4: Propozycja rozszerzenia taksonomii systemów monitoringu zaproponowanej w pracy K. Fatema i wsp.: [38].

Proces ten może być scharakteryzowany przez 11 wspólnych cech dla wszystkich wyszczególnionych procesów zachodzących w środowisku chmurowym, tj. skalowalność, przenośność, nieinwazyjność, stabilność, wielodostępność, współoperatywność, możliwość personalizacji, rozszerzalność, możliwość monitoringu zasobów współdzielonych, łatwość użycia oraz zdolność archiwizacji. Dodatkowo monitoring procesu harmonogramowania zadań powinien charakteryzować się możliwościami takimi jak: pomiar zużycia zasobów, monitoring jakości usług, zarządzanie ryzykiem, identyfikacja statusu elementów systemu, monitoring obciążenia usług czy monitoring efektów zmian konfiguracji. Harmonogramowanie zadań może mieć kluczowy wpływ na wydajność chmury obliczeniowej jak i inne kryteria działania środowiska (np. zużycie energii, koszty usług).

3.4. Definicja modelu monitoringu

Do tej pory nie zaproponowano formalnego zapisu modeli monitoringu procesów działających w rozproszonych środowiskach obliczeniowych. Na podstawie analizy taksonomii przedstawionych w niniejszej pracy, a także wzorując się na notacji zastosowanej przy definicji problemu harmonogramowania w książce [86], podjęto próbę formalnej definicji modelu systemu monitorującego. Zaproponowany model definiuje system monitoringu w sposób ogólny (na wysokim poziomie abstrakcji).

Model systemu monitorującego procesy w rozproszonych środowiskach dużej skali można zdefiniować za pomocą następującej notacji:

$$\alpha|\beta|\gamma \tag{3.1}$$

gdzie: α – model środowiska rozproszonego, β – monitorowany proces, γ – cele monitoringu.

Możemy wyróżnić kilka popularnych modeli środowiska rozproszonego (α), m.in.:

- chmura obliczeniowa (ang. *cloud computing*),
- prztwarzanie sieciowe (ang. *grid computing*),
- klaster obliczeniowy (ang. *cluster computing*),

- mgła obliczeniowa (ang. *fog computing*).

Monitorowane procesy (β) uzależnione są od wybranego środowiska. W niniejszej pracy szczególny nacisk został położony na chmury obliczeniowe. Zaproponowana taksonomia (rys. 3.4) podaje następujące procesy zachodzące w chmurach obliczeniowych:

- rozliczenia usług,
- realizacja zapisów umowy o gwarantowanym poziomie świadczenia usług,
- dostarczanie zasobów i usług,
- planowanie zapotrzebowania na zasoby (zarówno fizyczne jak i wirtualne),
- konfiguracja środowiska,
- zapewnienie bezpieczeństwa i prywatności,
- zarządzanie awariami i błędami,
- harmonogramowanie zadań.

Ostatni element definicji modelu monitoringu to cele monitoringu (γ). Analizując zadania stawiane nowoczesnym systemom monitoringu (por. [13, 14, 38, 74]), do celów monitoringu można zaliczyć m.in.:

- zbieranie danych statystycznych,
- sporządzanie raportów,
- wsparcie monitorowanego procesu,
- identyfikacja awarii i przekroczeń norm,
- powiadamianie o nieterminowej realizacji zadań,
- detekcja naruszeń bezpieczeństwa i prób ataków,
- audyt wydajności infrastruktury,

- analiza i optymalizacja przepływu danych,
- nadzór nad warstwą sieciową rozproszonego środowiska,
- nadzór nad ilością dostępnych zasobów,
- oszacowanie zapotrzebowania na zasoby,
- weryfikacja wpływu zmian konfiguracji na rozproszone środowisko.

Powyższa lista prezentuje jedynie przykładowe cele systemu monitoringu i może być rozwijana w zależności od stawianych wobec systemu oczekiwań. Stosując zaproponowaną notację można dokonać klasyfikacji systemu względem jego zastosowania. Rozważany w pracy system ukierunkowany jest w sposób szczególny na środowiska chmurowe (co jednak nie wyklucza możliwości jego stosowania w rozproszonych środowiskach obliczeniowych innego typu). Głównym przedmiotem monitoringu jest proces harmonogramowania zadań - w niniejszej pracy są to niezależne zadania obliczeniowe przetwarzane pakietowo. Do stawianych przed systemem celów można zaliczyć w szczególności wsparcie monitorowanego procesu, powiadamianie o nieterminowej realizacji zadań, audyt wydajności infrastruktury, zbieranie danych statystycznych czy weryfikację wpływu zmian konfiguracji na rozproszone środowisko.

Rozdział 4

Wprowadzenie do systemów wieloagentowych

W niniejszym rozdziale przedstawiono formalną definicję agenta, a także opisano jego model. Opis ten uwzględnia zarówno cechy pojedynczej jednostki, jak i własności pozwalające na współdziałanie grupy agentów w ramach systemu wieloagentowego osadzonego w pewnym środowisku.

Systemy wieloagentowe (ang. *multiagent systems*) to systemy zbudowane ze współpracujących bytów zwanych agentami. Agenty cechują się dwiema własnościami: możliwością wykonywania autonomicznych akcji oraz możliwością interakcji z innymi agentami [125]. Ten bardzo nieprecyzyjny opis dobrze odzwierciedla to, co obecnie kryje się pod pojęciem agenta.

Idea programowania agentowego należy do tzw. rozproszonej sztucznej inteligencji (ang. *distributed artificial intelligence*), zajmującej się rozwojem rozproszonych rozwiązań złożonych problemów, które wymagają podejścia inteligentnego. Pierwsze prace poświęcone idei systemów wieloagentowych pojawiły się w latach 80-tych. Idea ta zdobyła popularność w połowie lat 90-tych i od tego momentu zainteresowanie nią intensywnie wzrasta. Wynika ono z przekonania, że paradygmat agentowy jest odpowiednim narzędziem pozwalającym na wykorzystanie możliwości, jakie dają otwarte rozproszone systemy (ang. *massive open distributed systems*) [30, 125]. Koncepcja agenta była również często adaptowana w równoległych środowiskach obliczeniowych, jak np. system do rozpraszania zadań w strukturach wieloprocesorowych (w tym również gridów i chmur obliczeniowych) [28, 30, 79].

4.1. Koncepcja autonomicznego agenta

Pomimo kilku dekad prac nad koncepcją agentów, wciąż szeroko dyskutowany jest problem definicji pojedynczego agenta. Najbardziej popularną definicję sformułował M. Wooldridge [125] (tłumaczenie: [30]):

Agent jest systemem komputerowym, który jest osadzony w pewnym środowisku, i który jest zdolny do autonomicznych akcji w celu zrealizowania zleconych, określonych celów.

Podobną definicję podają Franklin i Graesser [40] (tłumaczenie: [65]):

Agent to system, który usytuowany jest w pewnym środowisku, i którego jednocześnie jest częścią. Agent obserwuje to środowisko oraz działa w nim, w czasie, według własnego planu, wpływając na to, co będzie mógł zaobserwować w przyszłości.

Wiele prac poświęconych paradygmatowi agentowemu ([30, 40, 65, 121, 125] podkreśla autonomiczność agentów, a więc cechę, która pozwala na podejmowanie samodzielnych decyzji. Autonomiczność pozwala agentowi na zachowanie pełnej kontroli nad swoim wewnętrznym stanem oraz podejmowanymi akcjami [65]. Jest to kluczowa własność kształtująca charakter agenta, który często w literaturze jest spotykany pod pojęciem autonomicznego agenta. Cecha autonomiczności agenta nie jest jednak wyróżniającą dla tego pojęcia – podobnie obiekty w ujęciu programistycznym mogą cechować się autonomicznością. Cechą, która pozwala na odróżnienie agentów od innych bytów jest zdolność obserwacji otaczającego środowiska, w tym również agentów sąsiadujących. Pozwala to na podejmowanie decyzji na podstawie stanu otaczającego środowiska i stanów pozostałych agentów (w tym celu agent może posiadać zdolność do komunikacji międzyagentowej za pośrednictwem środowiska lub komunikacji bezpośredniej) [30].

M. Wooldridge w [121] autonomicznego agenta określa mianem agenta inteligentnego. Definiuje go jako agenta, który w sposób zdecydowany działa w dynamicznie zmieniającym się, nieprzewidywalnym środowisku, w którym istnieje wysokie prawdopodobieństwo niepowodzenia akcji agentów. Kluczowym problemem w obliczu którego staje agent (czy też cały system wieloagentowy) jest podjęcie odpowiednich akcji, które powinny zostać wyko-

nane w celu realizacji zleconych zadań (ang. *delegated objectives*). Zadania te powinny być realizowane dzięki trzem cechom agentów:

- działanie celowe (ang. *proactiveness*) – inteligentne agenty są zdolne do przejęcia inicjatywy w celu realizacji zleconych celów;
- reaktywność (ang. *reactivity*) – inteligentne agenty są zdolne postrzegać środowisko w którym są ulokowane, a także reagować we właściwym czasie na zmiany zachodzące w środowisku;
- zdolność współpracy (ang. *social ability*) – inteligentne agenty są zdolne do interakcji z innymi agentami (a w niektórych przypadkach z człowiekiem) w celu realizacji zleconych celów.

Niektórzy autorzy ([45, 59, 65, 110]) dodają inne cechy, takie jak:

- mobilność (ang. *mobility*) – agent może być zdolny do przemieszczania się w obrębie swojego środowiska;
- umiejętność adaptacji/uczenia się (ang. *adaptability, learning ability*) – agent może być wyposażony w mechanizmy pozwalające na uczenie się, a także powinien mieć zdolność adaptacji do zmieniających się warunków;
- racjonalność (ang. *rationality*) – agent realizując zlecone cele korzysta z posiadanej wiedzy, w szczególności nie działając sprzecznie ze zleconym celem;
- usytuowanie (ang. *situatedness*) – agent znajduje się w pewnym środowisku, jest jego częścią, ma możliwość jego postrzegania oraz podejmowania akcji, które mogą zmieniać to środowisko.

Ponadto M. Wooldridge zwraca uwagę na trzy aspekty wyróżniające agenta w stosunku do obiektu będącego podstawą paradygmatu programowania obiektowego. Są to następujące różnice [121]:

(i) agenty silniej uosabiają pojęcie autonomii niż obiekty, w szczególności same decydują o tym, czy wykonać żądanie przesłane przez innego agenta;

(ii) agenty – w odróżnieniu od obiektów – cechują się reaktywnością, proaktywnością i zdolnością do współpracy;

(iii) system wieloagentowy jest z założenia wielowątkowy – każdy agent operuje co najmniej jednym wątkiem, nieustannie realizując swoje zadania.

W literaturze podjęto kilka prób klasyfikacji agentów, m.in. w [40, 80, 107]. Najbardziej popularną jest stosunkowo prosta taksonomia zaproponowana przez Franklin i Graesser w [40]. Autorzy w pierwszym rzędzie wyszczególniają trzy klasy autonomicznego agenta:

(i) agent biologiczny (ang. *biological agent*) – reprezentuje wszystkie organizmy żywe;

(ii) robot (ang. *robotic agent*) – klasa reprezentująca autonomiczne roboty mające zdolność percepcji oraz podejmowania decyzji;

(iii) agent komputerowy (ang. *computational agent*) – reprezentuje agenty wykorzystywane w komputerowych symulacjach istot żywych (ang. *artificial life agents*) oraz oprogramowania agentowego (ang. *software agents*).

4.2. Środowisko działania agenta

Istotną cechą agenta, uwzględnioną w przytoczonych definicjach, jest jego usytuowanie w środowisku. S. Russell i P. Norvig w [93] zaproponowali taksonomię środowisk, w których przebywają i działają agenty (rys. 4.1). Autorzy opisują środowisko bazując na następujących charakterystykach:

(i) W pełni postrzegalne – częściowo postrzegalne (ang. *fully observable – partially observable*).

Jeśli agent jest w stanie uzyskać pełną informację o środowisku w każdym momencie czasowym, wtedy środowisko można określić jako w pełni postrzegalne przez agenta.

Środowisko jest w pełni widoczne, jeśli wszystkie aspekty, które są istotne dla wyboru działań agenta, mogą być monitorowane. W pełni obserwowalne środowiska zwalniają agenta z obowiązku utrzymywania stanu wewnętrznego opisującego środowisko. Środowisko może być częściowo postrzegalne z powodu braku odpowiednich mechanizmów agenta lub też polityk bezpieczeństwa.

(ii) Z pojedynczym agentem – z systemem wieloagentowym (ang. *single agent – multiagent*).

Różnica pomiędzy środowiskiem z pojedynczym agentem a systemem wieloagentowym jest stosunkowo jasna. W przypadku środowisk, w których działa grupa agentów wspólnie realizująca określone cele mamy do czynienia ze środowiskiem, w którym zaimplementowany jest system wieloagentowy. Środowiska, w których działa pojedynczy agent są zwykle środowiskami ograniczającymi się do realizacji prostszych celów.

(iii) Deterministyczne – stochastyczne (ang. *deterministic – stochastic*).

Jeśli następny stan środowiska jest całkowicie zależny od stanu obecnego oraz działań wykonanych przez agenta, to mówimy, że środowisko jest deterministyczne, w przeciwnym przypadku mamy do czynienia ze środowiskiem stochastycznym. W praktyce środowiska deterministyczne często są traktowane jak środowiska stochastyczne. Jest to spowodowane dużą złożonością procesów zachodzących w środowiskach deterministycznych i brakiem możliwości obserwowania wszystkich aspektów wpływających na stan środowiska.

(iv) Epizodyczne – sekwencyjne (ang. *episodic – sequential*).

W środowiskach epizodycznych, działania (a zarazem doświadczenia) agenta zachodzą w atomowych epizodach. W każdym takim epizodzie, na podstawie zachodzącej w nim percepcji, agent wykonuje jedno działanie. Epizody są niezależne od działań podejmowanych w poprzednich epizodach. Środowiska epizodyczne, w przeciwieństwie do środowisk sekwencyjnych, nie wymagają od agenta brania pod uwagę przyszłych możliwych stanów środowiska.

(v) Statyczne – dynamiczne (ang. *static – dynamic*).

Jeśli środowisko może się zmieniać niezależnie od działań agenta, szczególnie w trakcie podejmowania decyzji dotyczących działań agenta, to takie środowisko określamy dynamicznym dla tego agenta. W przypadku środowisk statycznych, stan środowiska ulega zmianie tylko w wyniku działań agenta. Środowiska statyczne nie wymagają rozbudowanych narzędzi percepcji, które nieustannie dokonują pomiarów w środowisku. Czas w środowiskach dynamicznych odgrywa kluczową rolę.

(vi) Dyskretne – ciągłe (ang. *discrete – continuous*).

Atrybut dyskretności/ciągłości dotyczy czterech własności: stanu środowiska, sposobu obsługi czasu, a także percepcji oraz akcji wykonywanych przez agenta. Środowiska ze skończoną liczbą stanów, skokowym działaniem w czasie, określonymi momentami percepcji oraz wykonywania akcji przez agenty nazywamy środowiskami dyskretnymi. W środowiskach ciągłych wymienione własności są ciągłe. Często systemy agentowe działają w środowiskach mieszanych.

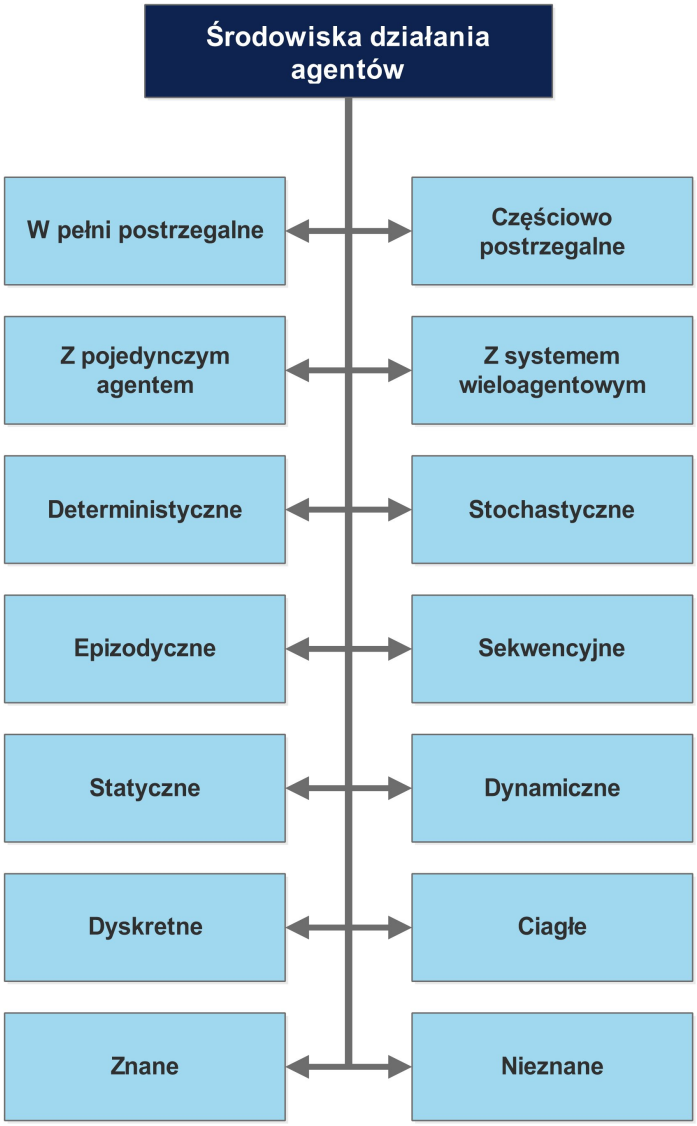
(vii) Znane – nieznane (ang. *known – unknown*).

Cecha ta odnosi się bezpośrednio do wiedzy agenta i nie jest tożsama z postrzegalnością środowiska. Dotyczy znajomości reguł działania środowiska. Jeśli agent od początku posiada dokładną wiedzę na temat procesów zachodzących w środowisku, wtedy takie środowisko określane jest jako znane. W przypadku kiedy agent (nawet jeśli jest zdolny do pełnego postrzegania) nie zna reguł działania środowiska, musi się tych reguł nauczyć, aby podejmować odpowiednie decyzje.

Druga klasyfikacja została opracowana przez K. Cetnarowicza w [29]. Autor proponuje podział środowiska wokół agenta z punktu widzenia jego dostępności. Wyszczególniono pięć obszarów:

(i) środowisko dalekie – reprezentuje dane, które są w postrzeganiu bezpośrednim lub pośrednim agenta (np. poprzez kontakt z innymi agentami lub przemieszczenie się);

(ii) środowisko bliskie – środowisko będące w zasięgu mechanizmów percepcji agenta;



Rysunek 4.1: Klasyfikacja środowiska działania agentów na podstawie jego cech [93].

Rysunek 4.2: Klasyfikacja środowiska działania agentów z punktu widzenia jego dostępności [29].

(iii) otoczenie – część środowiska bliskiego będąca w zasięgu mechanizmów sprawczych agenta;

(iv) sąsiedztwo – część otoczenia, na którą agent może oddziaływać w większym stopniu niż inne agenty;

(v) środowisko własne – część sąsiedztwa, która może być modyfikowana tylko przez agenta – właściciela lub za jego zgodą.

Autor wyszczególnia również dwa składniki występujące w środowisku:

(i) zasoby – są to elementy nieposiadające zdolności inicjatywnych agenta, ale mogące ulegać zmianom według własnego ustalonego algorytmu;

(ii) agenty – są częścią składową środowiska odpowiednio postrzeganą przez pozostałych agentów ulokowanych w środowisku.

4.3. Typy agentów

Prace nad paradygmatem agentowym zaowocowały próbami formalizacji architektury agenta. Zaproponowano kilka metod budowy agentów opisanych różnymi (bardziej lub mniej) formalnymi terminami. Literatura podaje kilka modeli i architektur agentowych. Do popularnych podejść można zaliczyć następujące typy [29, 35, 65, 125]:

(i) Agent-0

Agent-0 [99] jest to jedna z pierwszych propozycji agentowych architektur. Agenty w tej architekturze podejmują decyzje w oparciu o tzw. stany świadomości (ang. *mental states*). Stany świadomości zbudowane są na podstawie trzech komponentów: wiedzy (ang. *belief*), zobowiązania (ang. *obligation*), zdolności (ang. *capability*).

(ii) Agent BDI (ang. *Beliefs-Desires-Intentions*)

Architektura BDI [90] opiera się o teorie pragmatycznego wnioskowania. Model ten wywodzi się z nauk kognitywistycznych, które poświęcone są obserwacjom i analizom działania zmysłów, mózgu i umysłu. Agent ten składa się z trzech elementów: przekonanie (ang. *belief*) – reprezentuje informacje o środowisku agenta; pragnienie (ang. *desire*) – reprezentuje motywacyjny stan agenta; zamiar (ang. *intention*) – reprezentuje cele agenta (cele te nie mogą się wzajemnie wykluczać).

(iii) Agent logiczny (ang. *logic-based/deliberate agent*)

Agenty logiczne [69, 125] bazują na symbolicznym podejściu sztucznej inteligencji. Zachowania inteligentne agentów generowane są na podstawie symbolicznej reprezentacji (logicznych reguł) środowiska i celów agentów. Agenty podejmują decyzje w wyniku logicznej dedukcji. Wadą tego podejścia jest wysoka złożoność obliczeniowa.

(iv) Agent reaktywny (ang. *reactive/behavioural/situated agent*)

Idea agenta reaktywnego [22, 23] opiera się na założeniu, że inteligencja jest wytworem oddziaływań pomiędzy agentem, a środowiskiem. Inteligentne zachowanie powstaje w wyniku interakcji wielu prostych zachowań. Zbiór takich zachowań, realizowanych jako niezależne podsystemy, tworzy reaktywnego agenta. Podejście to stoi w opozycji do agenta logicznego.

(v) Agent hybrydowy (ang. *hybrid agent*)

Agent hybrydowy [125] zbudowany jest w oparciu o architekturę warstwową. Hierarchia współpracujących ze sobą warstw prezentuje odpowiednie podsystemy składające się

na całość agenta. Zwykle agent zbudowany jest z dwóch warstw, odpowiednio modelujących zachowania reaktywne oraz proaktywne. Warstwy mogą być ułożone w sposób horyzontalny (ang. *horizontal layering*) lub wertykalny (ang. *vertical layering*). W architekturach horyzontalnych każda warstwa otrzymuje zestaw danych wejściowych, przetwarza je i generuje swój wynik. W przypadku architektur wertykalnych dane odbierane są przez jedną warstwę, a następnie (po przetworzeniu) przesyłane do warstwy wyższej.

(vi) Agent zewnętrzny

Idea agenta zewnętrznego została zaproponowana przez G. Dobrowolskiego w [35]. Autor opracował model agenta i systemu agentowego, sformalizował pojęcia działania agenta i systemu, przedstawił cechy typowe dla systemów agentowych oraz sformułował problem integralności funkcjonalnej. Duży nacisk został położony na własności komunikacyjne agentów – autor sformalizował typ agenta komunikującego się i opracował problem komunikacji oparty o protokoły komunikacyjne. W ramach pracy określono również formalne granice systemu agentowego.

(vii) Inteligentny agent (ang. *intelligent agent*)

Inteligentny agent [121, 125] jest jednym z najbardziej popularnych formalnych zapisów agentów. Jest to wysoce abstrakcyjna architektura zaproponowana przez M. Wooldridge. Autor definiuje zarówno agenta, środowisko, jak i funkcje reprezentujące różnego rodzaju działania agenta w środowisku.

(viii) M-agent

M-agent [29, 30] reprezentuje architekturę bardziej szczegółową w stosunku do inteligentnego agenta. Model ten został zaproponowany przez K. Cetnarowicza. Autor dogłębnie definiuje zarówno środowisko jak i samego agenta, a także relacje pomiędzy agentami a środowiskiem oraz wzajemne relacje agentów. Poza ogólnym modelem M-agenta, wyszczególnione zostały dodatkowo profile agentowe: intelektualny, energetyczny oraz wieloprofilowy.

W dalszej części rozdziału zostaną przedstawione szczegółowo dwa modele agentów: inteligentny agent oraz M-agent. Pierwszy model, jeden z najbardziej rozpowszechnionych w literaturze, prezentujący podejście wysoce abstrakcyjne, drugi charakteryzujący szerszy zakres składowych pojedynczego agenta i – w przypadku profilu intelektualnego agenta – kładący nacisk na własność inteligencji.

4.3.1. Inteligentny agent

Formalny zapis abstrakcyjnej architektury inteligentnych agentów został zaproponowany przez M. Wooldridge'a [120, 125].

Jak zostało wcześniej podkreślone, agenty są usytuowane w konkretnym środowisku, i stanowią jego część. Mają możliwość interakcji ze środowiskiem poprzez wykonywanie określonych akcji. Środowisko reaguje na te akcje poprzez zmianę swojego stanu. W pracy [125] przedstawiono poniższe pojęcia i definicje dotyczące środowiska oraz działających w nim agentów.

Załóżmy, że skończony zbiór:

$$E = \{e, e', \ldots\} \tag{4.1}$$

reprezentuje zbiór stanów środowiska, w jakim może się ono znajdować. W każdym momencie czasowym środowisko znajduje się w jednym ze stanów e. Możliwość działania agentów w środowisku reprezentowana jest przez skończony zbiór akcji:

$$Ac = \{\alpha, \alpha', \ldots\} \tag{4.2}$$

Środowisko pierwotnie znajduje się w swoim początkowym stanie e_0. Agent wybiera odpowiednią akcję na podstawie aktualnego stanu środowiska. Jako rezultat wykonania akcji środowisko może odpowiedzieć wieloma teoretycznie możliwymi stanami. Jednakże tylko jeden z tych stanów może być stanem aktualnym. Agent nie zna stanu środowiska z góry przed wykonaniem akcji. Po zmianie stanu środowiska ze stanu e_0 agent jest zdolny do

wyboru kolejnej akcji na podstawie aktualnego stanu e_1. Wykonanie kolejnej akcji powoduje odpowiednią zmianę stanu środowiska.

Cały ten proces można określić jako ruch (lub też historię) agenta i zapisać w postaci sekwencji przeplatającej stany środowiska oraz akcje agenta:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_{u-1}} e_u \tag{4.3}$$

Niech:

- $\mathcal{R} = r, r', \ldots$ będzie zbiorem wszystkich możliwych skończonych ruchów agenta (nad E i Ac),
- $\mathcal{R}^{Ac}$ będzie podzbiorem $\mathcal{R}$ reprezentującym ruchy, które kończą się akcją,
- $\mathcal{R}^{E}$ będzie podzbiorem $\mathcal{R}$ reprezentującym ruchy, które kończą się stanem środowiska.

Funkcja transformacji stanu (ang. *a state transformer function*) odwzorowuje wykonanie akcji na stan środowiska. Można ją sformułować w następujący sposób:

$$\tau : \mathcal{R}^{Ac} \nrightarrow \mathbf{2}^{E} \tag{4.4}$$

Zatem funkcja transformatora stanu jest funkcją częściową [126], która odwzorowuje ruch agenta (kończący się akcją) na zbiór potęgowy możliwych stanów środowiska.

Należy zwrócić uwagę, że powyższe rozważania zakładają, że środowisko jest zależne od historii wykonywanych akcji i stanów środowiska. To oznacza, że uprzednio wykonane akcje również odgrywają rolę w ustalaniu aktualnego stanu środowiska. Ponadto rozważane środowisko może być niedeterministyczne, co wiąże się z niepewnością wyniku wykonania akcji w pewnych stanach.

Jeśli $\tau(r) = \emptyset$ ($r \in \mathcal{R}^{Ac}$), to brak jest możliwych następnych stanów środowiska dla ruchu r. W takiej sytuacji ruch zostaje zakończony.

Formalnie środowisko może zostać zdefiniowane za pomocą trójki:

$$Env =< E, e_0, \tau > \tag{4.5}$$

gdzie: E – skończony zbiór stanów środowiska, e_0 – początkowy stan środowiska, τ – funkcja transformacji stanu.

W najprostszej formie, agent Ag może być reprezentowany przez funkcję odwzorowującą ruchy, które kończą się stanem środowiska na akcje agenta. Funkcję taką można zapisać jako:

$$Ag : \mathcal{R}^E \rightarrow Ac. \tag{4.6}$$

W sposób intuicyjny można opisać działanie tej funkcji jako podjęcie przez agenta decyzji odnośnie wyboru akcji do wykonania bazując na jego doświadczeniach. Doświadczeniem agenta jest sekwencja stanów środowiska i akcji (czyli ruchy), z którymi agent w swojej historii miał do czynienia i które są mu znane.

Parę agent – środowisko będziemy określać mianem systemu. Każdy system jest powiązany ze zbiorem możliwych ruchów. Zbiór ruchów agenta Ag w środowisku Env oznaczamy jako $\mathcal{R}(Ag, Env)$. Dla uproszczenia, rozważamy tylko ruchy skończone ($\tau(r) = \emptyset$).

Wobec powyższego, sekwencja:

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots) \tag{4.7}$$

przedstawia ruch agenta Ag w środowisku $Env =< E, e_0, \tau >$ jeśli:

1. e_0 jest stanem początkowym środowiska Env;
2. $\alpha_0 = Ag(e_0)$;
3. $\forall u > 0, e_u \in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1})) \wedge \alpha_u = Ag((e_0, \alpha_0, \ldots, e_u))$.

Mówimy, że dwa agenty Ag_1 i Ag_2 znajdujące się w środowisku Env są równoważne behawioralnie (ang. *behaviourally equivalent*) wtedy i tylko wtedy, gdy $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$.

Agent, który nie odwołuje się do swojego doświadczenia, podczas wyboru akcji nazywany jest agentem czysto reaktywnym (ang. *purely reactive agent*). Decyzję o wyborze akcji opiera jedynie na bieżącym stanie środowiska, bez odniesienia do historii środowiska. Formalnie takiego agenta można zdefiniować za pomocą funkcji:

$$Ag : E \rightarrow Ac. \tag{4.8}$$

Agenty mają swoją pewną wewnętrzną strukturę, która zazwyczaj jest wykorzystywana do rejestrowania informacji o stanie środowiska i jego historii. Niech I będzie zbiorem wszystkich wewnętrznych stanów agenta. Proces podejmowania decyzji będzie wtedy co najmniej częściowo opierał się na tej informacji. Takie agenty określane są mianem agentów ze stanem (ang. *agents with state*).

Agent powinien mieć możliwość uzyskiwania informacji o otaczającym środowisku. W tym celu należy zdefiniować następującą funkcję pozwalającą na postrzeganie stanu środowiska:

$$see : E \rightarrow Per, \tag{4.9}$$

gdzie Per to niepusty zbiór percepcji.

Wtedy funkcję wyboru akcji można zdefiniować następująco:

$$action : I \rightarrow Ac. \tag{4.10}$$

Funkcja ta odwzorowuje stany wewnętrzne agenta w jego akcje.

Funkcję, która aktualizuje stan wewnętrzny agenta, można zapisać w następującej formie:

$$next : I \times Per \rightarrow I. \tag{4.11}$$

Funkcja ta odwzorowuje stany wewnętrzne agenta i jego percepcję w aktualne stany wewnętrzne.

Zasadę działania agentów ze stanem można opisać w następujący sposób. Agent rozpoczyna działanie w stanie początkowym i_0. Obserwuje stan swojego środowiska (e) i generuje

percepcję $see(e)$. Stan wewnętrzny agenta zostaje zaaktualizowany poprzez wykonanie funkcji $next(i_0, see(e))$. Następnie agent dokonuje wyboru akcji: $action(next(i_0, see(e)))$. Akcja zostaje wykonana, a agent rozpoczyna nowy cykl: obserwacja środowiska, aktualizacja stanu, wybór i wykonanie akcji.

4.3.2. M-agent

K. Cetnarowicz w swoich pracach [29, 30] proponuje inny model agenta, nazwany M-agentem. Autor proponuje poniższy formalny zapis architektury agentowej.

Środowisko v, w którym umieszczone są i działają agenty można zdefiniować jako trójkę:

$$v = (E, A, C), \tag{4.12}$$

gdzie: E – zbiór wszystkich nieaktywnych elementów systemu (element aktywny to autonomiczny element zdolny do działania i wprowadzania zmian w środowisku) określany przez charakter problemu, dla którego definiowany jest system agentowy, A – aktualny zbiór agentów działających i znajdujących się w środowisku, C – relacja pomiędzy przestrzenią E a agentami ze zbioru A.

Zbiór środowisk rozważanych w systemie będzie oznaczany przez $\mathcal{V}$.

Agent a może zostać zdefiniowany za pomocą ósemki:

$$a = (M, \mathcal{Q}, \mathcal{S}, I, X, m, q, s), \tag{4.13}$$

gdzie:

m – model środowiska v, w którym znajduje się agent (zgodnie z taksonomią środowiska przedstawioną w podrozdziale 4.2 jest to otoczenie agenta, czyli część środowiska bliskiego będąca w zasięgu mechanizmów sprawczych agenta).

$\mathcal{M}$ – zbiór modeli (m) środowisk, które mogą skupiać się wokół agenta ($m \in \mathcal{M}$). Modele te znajdują się w zakresie wiedzy agenta. M – wiedza agenta (pewnego rodzaju doświadczenie)

służąca do budowania modeli m. s – strategia określająca działanie agenta, będąca funkcją:

$$s : \mathcal{M} \to \mathcal{M}, \qquad m' = s(m). \tag{4.14}$$

$\mathcal{S}$ – zbiór możliwych strategii agenta ($s \in \mathcal{S}$). q – cel agenta wyrażony w postaci funkcji pozwalającej na ocenę działania agenta:

$$q : \mathcal{M} \times \mathcal{M} \to \mathfrak{R}, \qquad q(m, m') \in \mathfrak{R}, \tag{4.15}$$

gdzie:

$\mathfrak{R}$ – zbiór liczb rzeczywistych;

m – model środowiska w którym agent aktualnie się znajduje;

m' – model środowiska docelowego, czyli takiego, który agent chce osiągnąć za pomocą realizacji celu q,

$\mathcal{Q}$ – zbiór celów agenta ($q \in \mathcal{Q}$) ze zdefiniowanym porządkiem.

I – operator obserwacji otoczenia agenta określony formułą:

$$m = I(v, M), \qquad m \in \mathcal{M}. \tag{4.16}$$

W wyniku obserwacji środowiska v i wiedzy M powstaje model środowiska m.

X – operator realizacji wybranej strategii s w otoczeniu v określony formułą:

$$v' = X(s, m, v), \qquad m \in \mathcal{M}, \qquad s \in \mathcal{S}, \qquad v, v' \in \mathcal{V}, \tag{4.17}$$

gdzie:

v' – środowisko powstałe w wyniku realizacji strategii s wybranej na podstawie analizy modelu m w ramach środowiska v.

Algorytm działania M-agenta może zostać opisany w następujący sposób:

1. Inicjalizacja agenta a wg formuły 4.13 w środowisku v (wzór: 4.12). Ustalenie początkowego stanu relacji C między agentem a przestrzenią E.

2. Obserwacja środowiska v i konstrukcja modelu m za pomocą operacji I danej wzorem

4.16.

3. Wybór strategii optymalnej s^* – czyli strategii najlepiej realizującej cel q. Wybór strategii optymalnej dokonywany jest na podstawie oceny, w jakiej mierze zastosowanie strategii s^* pozwala na realizację celu q, co można zapisać jako:

$$s^* = \arg\max_{s \in \mathcal{S}} q(m, s(m)). \tag{4.18}$$

4. Realizacja wybranej strategii s^* w środowisku v za pomocą operatora X danego wzorem 4.17. Przejście do punktu 2.

K. Cetnarowicz w [29] zaproponował dwa profile modelu M-agenta: profil intelektualny oraz profil energetyczny.

Profil intelektualny reprezentuje agenta, którego działania mają na celu realizację podstawowych, zleconych zadań. Są to działania dążące do rozwiązania zadanego problemu.

Profil energetyczny odnosi się do tej grupy agentów, której podstawowym celem jest "przetrwanie" w środowisku (możliwość egzystencji i działania).

Profil agenta odgrywa kluczową rolę przy podejmowaniu przez agenta decyzji odnośnie realizowanych strategii. Struktura agenta nie musi ograniczać się do wyżej zaproponowanych profili, co więcej, agent może być jednostką wieloprofilową.

Z punktu widzenia niniejszej pracy, wartościową jest idea M-agenta o profilu intelektualnym. W [29] K. Cetnarowicz proponuje rozszerzenie modelu agenta danego wzorem 4.13 w poniższy sposób:

$$a = (M, \mathcal{Q}, \mathcal{S}, I, X, L, m, q, s), \tag{4.19}$$

gdzie:

L – operator uczenia się agenta. Operator $L = (L_{\mathcal{M}}, L\mathcal{S})$ oddziałuje zarówno na zbiór modeli $\mathcal{M}$, jak i zbiór strategii $\mathcal{S}$.

W wyniku działania operatora $L_{\mathcal{M}}$ zostaje określona nowa konfiguracja (zbiór) modeli:

$$\mathcal{M}' = L_{\mathcal{M}}(\mathcal{M}, m', m'') \tag{4.20}$$

oraz w wyniku działania operatora $L_{\mathcal{S}}$ zostaje określona nowa konfiguracja (zbiór) strategii:

$$\mathcal{S}' = L_{\mathcal{S}}(\mathcal{S}, m', m''), \tag{4.21}$$

gdzie:

m' – model przewidywanego środowiska otrzymanego po realizacji strategii s;

m'' – model otrzymanego środowiska po realizacji strategii s.

Modele m' i m'' mogą różnić się na skutek posiadania przez agenta niepełnej informacji o środowisku.

Pozostałe oznaczenia są zgodne z definicją agenta danego wzorem 4.13.

4.3.3. Inne modele agentów

W literaturze można znaleźć wiele formalnych definicji agentów. Są one mniej lub bardziej szczegółowe. K. Cetnarowicz w [30] zwraca szczególną uwagę na różne pojmowanie pojęcia agenta, a co za tym idzie – brak jednolitego wzorca pozwalającego na odróżnianie agenta od innych pojęć. Taki stan rzeczy istotnie komplikuje zarówno badania nad systemami agentowymi, jak również proces ich projektowania. Powszechnie stosowaną praktyką jest proponowanie takiej definicji agentów, która nie narzuca zbytnich ograniczeń na modelowany system i jednocześnie jest w stanie realizować postawione funkcjonalności.

G. Dobrowolski w [35] proponuje agenta (Λ) składającego się z trzech elementów:

$$\Lambda = \{A, S, F \subset S \times A \times S\}, \tag{4.22}$$

gdzie: A jest skończonym zbiorem akcji agenta, S jest skończonym zbiorem stanów wewnętrznych agenta, a F to trójczłonowa relacja określająca możliwe następstwa stanów i akcji agenta. Definicja ta następnie rozbudowywana jest pod kątem reaktywności agenta, jego strategii, możliwości komunikacji czy samoorganizacji.

M. Kisiel-Dorohinicki w [65] definiuje agenta (ag) poprzez następujące elementy:

$$ag = \{id, tp, dat_1, \ldots, dat_n\}, \tag{4.23}$$

gdzie: *id* jest unikalnym identyfikatorem agenta, *tp* oznacza typ agenta (w zależności od typu, agent posiada określone dane i może wykonywać określone akcje), a $dat_{i,i=1,\ldots,n}$ reprezentuje posiadane przez agenta dane. Akcje agentów zdefiniowane są jako osobny zbiór.

W pracy [24] A. Byrskiego system agentowy opisany jest za pomocą krotki:

$$MAS \equiv \langle Loc, Top, \lambda, \omega, Act \rangle, \tag{4.24}$$

gdzie: *Loc* to zbiór lokalizacji, do których mogą być przypisane agenty, $Top \subset Loc^2$ oznacza topologię określającą połączenia pomiędzy poszczególnymi lokalizacjami, λ to funkcja opisująca stan systemu poprzez przyporządkowanie stanów agentów do lokalizacji, ω jest funkcją decyzyjną, za pomocą której agent na podstawie swoich obserwacji oraz zasobów wybiera akcję, natomiast *Act* to zbiór par - warunku koniecznego do wykonania akcji oraz samej akcji.

Agent (ag) z powyższego systemu opisany jest przez stan dany wektorem:

$$ag = (id, \sigma, o, r), \tag{4.25}$$

gdzie: *id* to unikalny w skali systemu identyfikator agenta, σ oznacza rozwiązanie problemu reprezentowanego przez agenta, o to obserwacje wykonane przez agenta, natomiast r reprezentuje zgromadzone przez agenta zasoby.

J. Sobieska-Karpińska i M. Hernes w [100] pomijają formalną definicję agenta i skupiają się na definicji wiedzy agenta o monitorowanym środowisku.

4.4. Systemy wieloagentowe

W poprzednich podrozdziałach główny akcent został położony na pojedyncze agenty oraz środowisko ich działania. Systemy wieloagentowe z założenia zbudowane są ze zbioru agentów – często różnych typów. Agenty te pozostają w pewnych relacjach pomiędzy sobą. Prace [29, 33] wymieniają następujące klasy relacji pomiędzy agentami operującymi w tym samym środowisku (relacje agenta mogą należeć do jednej z poniższych klas):

(i) Współistnienie (ang. *cohabitation*) – celem agenta jest wykonanie powierzonego zadania, które jest on w stanie wykonać samodzielnie. Cecha ta jednak nie wyklucza możliwych konfliktów między agentami, które mogą być spowodowane ich dynamicznym zachowaniem.

(ii) Kooperacja (ang. *cooperation*) – w celu wykonania zadanego zadania agent musi współpracować z innymi agentami. Agent nie jest w stanie wykonać zadania samodzielnie lub też jest w stanie wykonać go w sposób mniej efektywny.

(iii) Współpraca (ang. *collaboration*) – pewne cele mogą być zrealizowane samodzielnie przez kilka (lub nawet wszystkie) spośród agentów. Głównym problemem jest wybór jednego z agentów, który wykona zadanie. W celu wyłonienia agenta potrzebna jest sprawna współpraca pomiędzy agentami.

(iv) Działanie rozproszone (ang. *distribution*) – pewne globalne cele mogą być osiągnięte jedynie przez grupę agentów. Głównym wyzwaniem dla tej klasy relacji jest dekompozycja globalnego zadania na poszczególne, współpracujące agenty. Ten rodzaj zachowań można zaobserować w systemach rozproszonej sztucznej inteligencji.

Poza pewnymi relacjami w jakich pozostają agenty, istotne są również inne własności systemów wieloagentowych. N. R. Jennings i wsp. w [60] tak charakteryzują system wieloagentowy:

- każdy z agentów systemu wieloagentowego posiada niepełną wiedzę lub niepełne możliwości wykonania zadania,
- w systemie brak jest globalnego sterowania,
- działanie agentów jest asynchroniczne,
- dane wykorzystywane przez agenty są rozproszone.

M. Kisiel-Dorohinicki w [65] zwraca uwagę, że powyższe cechy dostarczają nowych możliwości rozwiązywania złożonych problemów, które dotychczas były poza zasięgiem możliwości istniejących rozwiązań lub też rozwiązywania zadań w sposób bardziej efektywny

(prostszy, szybszy lub tańszy). Ponadto cechy systemów wieloagentowych wskazują na możliwość rozwiązywania zagadnień, w których posiadane dane są niepełne, bądź też są silnie rozproszone i współdzielone z innymi systemami. Niestety, podejście to stawia również wiele złożonych wyzwań w obszarze inżynierii oprogramowania, jak np. dekompozycja problemów, komunikacja międzyagentowa czy unikanie chaotycznego działania autonomicznych agentów. Rozważania w tych obszarach zaowocowały nowym obszarem określanym mianem agentowej inżynierii oprogramowania (ang. *agent-based software engineering*) [124].

K. Cetnarowicz w [30] podaje trzy elementy, które należy określić w ramach modelowania systemu wieloagentowego:

(i) Środowisko działania agentów. Środowisko musi uwzględniać cechy rozwiązywanego problemu.

(ii) Typy i zbiór agentów przebywających i działających w środowisku. Agenty powinny być zgrupowane zgodnie z ich typami. Należy również wyszczególnić cechy wspólne oraz własności charakterystyczne dla poszczególnych grup.

(iii) Relacje zachodzące pomiędzy agentami a środowiskiem oraz wzajemne relacje pomiędzy agentami. Podczas definiowania relacji należy zwrócić szczególną uwagę na ograniczone możliwości agentów.

Proces tworzenia modelu systemu nie jest łatwym zadaniem i może wymagać pewnego "utożsamienia" się z rolą modelowanego obiektu [31]. Analogiczne podejście zostało zaproponowane w [29, 30] – analiza i projektowanie agentowe należy wykonywać z punktu widzenia agenta.

Formalna definicja systemu agentowego zaproponowanego w ramach niniejszej pracy bazuje na modelach przedstawionych w tym rozdziale, jednocześnie realizując wyżej opisane właściwości systemu wieloagentowego i implementując funkcjonalności konieczne do realizacji postawionych celów. Proponowany formalny opis systemu agentowego znajduje się w rozdziale 6.4.

Rozdział 5

Harmonogramowanie zadań

Problem harmonogramowania zadań (ang. *task scheduling problem*), nazywany również problemem szeregowania zadań, jest jednym z kluczowych zagadnień w obszarze działania rozproszonych środowisk obliczeniowych. Jak podaje Internetowy Słownik Języka Polskiego [11], harmonogramowanie jest to *planowanie prac poprzez przydzielanie zasobów pracy i narzędzi do zadań oraz ustalenie rozkładu tych zadań w czasie* (określenie to występuje również w innych źródłach słownikowych – por. [87]). W kontekście rozproszonych systemów obliczeniowych ogólny problem harmonogramowania zadań obejmuje problem przypisywania zadań do odpowiednich zasobów oraz problem zarządzania ich wykonywaniem [61, 63, 67, 114]. Skuteczne harmonogramowanie zadań obliczeniowych wykonywanych na dostępnych zasobach (zwykle heterogenicznych) ma kluczowe znaczenie w osiągnięciu wysokiej wydajności i odpowiedniego obciążenia infrastruktury w systemach równoległych i rozproszonych.

Problem harmonogramowania nie ogranicza się jedynie do maksymalizacji wydajności rozproszonej infrastruktury; problem ten dotyka również innych obszarów i aspektów działania środowiska, takich jak np. bezpieczeństwo, minimalizacja zużycia energii, terminy wykonania zadań, personalizacja działania środowiska czy zależności występujące pomiędzy zadaniami. Wszystkie te wyzwania podkreślają, jak istotnym procesem w działaniu środowisk rozproszonych jest harmonogramowanie zadań.

Problemowi harmonogramowania zadań poświęcono wiele miejsca w literaturze naukowej na przestrzeni ostatnich kilkudziesięciu lat. W ramach niniejszego rozdziału zostaną

przybliżone podstawowe taksonomie problemu, jego formalny zapis oraz definicja problemu harmonogramowania niezależnych zadań obliczeniowych wraz z popularnymi kryteriami harmonogramowania. Następnie zdefiniowane zostaną reprezentacje zadań oraz jednostek obliczeniowych wraz z propozycją ich rozszerzenia, a także model macierzy ETC. Rozdział kończy opis ewolucyjnego podejścia do generowania harmonogramów wykonania zadań.

5.1. Klasyfikacja problemów harmonogramowania zadań oraz metod ich rozwiązywania

Literatura proponuje kilka taksonomii problemów harmonogramowania zadań oraz metod ich rozwiązywania dla rozproszonych środowisk obliczeniowych [18, 27, 36, 37, 61, 92, 98]. W niniejszym rozdziale przedstawiono trzy z nich, które zostały skonstruowane w oparciu o: (i) sposób przetwarzania zadań, (ii) zależności pomiędzy zadaniami, oraz (iii) rodziny algorytmów rozwiązujących problem harmonogramowania zadań.

H. D. Karatza w swoich pracach [62, 78, 85, 102, 103, 105, 106] skupia się na zadaniach, które mogą być wykonywane równolegle. Na podstawie sposobu przetwarzania zadań, autorka wyróżnia pięć klas problemów harmonogramowania:

(i) harmonogramowanie w trybie pakietowym (ang. *batch/bag-of-tasks scheduling*) – skupia problemy z zadaniami wzajemnie niezależnymi, które mogą być przetwarzane równolegle;

(ii) harmonogramowanie grup zadań (ang. *gang scheduling/coscheduling*) – zadania w tej klasie często wymieniają między sobą informacje. Mogą być one przetwarzane równolegle, przy czym wymagają możliwości cyklicznej wzajemnej komunikacji;

(iii) harmonogramowanie w oparciu o skierowany graf acykliczny (ang. *DAG scheduling*) – reprezentuje problemy, w których występują zadania z istotną kolejnością wykonania, a ich wykonanie wymaga realizacji w ramach konkretnych węzłów systemu. Tego typu zadania określane są mianem przepływu pracy (ang. *workflow*);

(iv) harmonogramowanie oparte o terminy wykonania (ang. *real–time scheduling*) – obejmuje problemy, w których zadania scharakteryzowane są przez ostateczne terminy wykonania i uzyskania rezultatów;

(v) harmonogramowanie zadań obarczonych prawdopodobieństwem niepowodzenia (ang. *fault–tolerant scheduling*) – klasa ta dotyczy zadań, podczas wykonywania których istnieje wysokie prawdopodobieństwo wystąpienia błędu, który uniemożliwi pełną realizację zadania.

Rozważane problemy harmonogramowania mogą jednocześnie należeć do kilku wyżej zaproponowanych klas – przykładowo, grupa komunikujących się ze sobą zadań, które mają wysokie prawdopodobieństwo wystąpienia błędów oraz wymóg uzyskania rezultatów w określonym czasie (por. [102, 105]) mogą zostać zaklasyfikowane do (ii), (iv) oraz (v) jednocześnie.

Inną klasyfikację przedstawia R. Annette i wsp. w [18]. Autorzy proponują prostą taksonomię metod harmonogramowania w oparciu o dwie właściwości: (i) występujące pomiędzy zadaniami zależności oraz (ii) zmienność charakterystyk środowiska i zadań. Na podstawie tych własności do problemu przypisywane są odpowiednie grupy algorytmów generujących jego rozwiązania.

W pracy wyszczególniono zadania niezależne, w przypadku których kolejność wykonania nie ma znaczenia (ang. *independent tasks*) oraz zadania pomiędzy którymi występują zależności, a kolejność wykonania jest istotna (ang. *dependent tasks*). Ponadto algorytmy harmonogramowania mogą być zaklasyfikowane jako statyczne lub dynamiczne. W przypadku harmonogramowania statycznego przyjmuje się, że charakterystyka zarówno zadań jak i maszyn jest wcześniej znana i niezmienna podczas całego procesu. Harmonogramowanie dynamiczne dopuszcza brak aktualnej wiedzy zarówno na temat wymagań zadania, jak i charakterystyk zasobów.

W efekcie wyłoniono cztery klasy, którym przyporządkowano grupy algorytmów rozwiązujących problem harmonogramowania. Są to:

(i) problemy z niezmiennymi parametrami oraz zadaniami niezależnymi – problemy te

mogą być rozwiązywane przez przeszukiwanie losowe (ang. *random search*) oraz heurystyki (ang. *heuristic*);

(ii) problemy ze zmiennymi parametrami oraz zadaniami niezależnymi – problemy te mogą być rozwiązywane przez algorytmy działające w trybach on–line oraz pakietowym (ang. *batch mode*);

(iii) problemy z niezmiennymi parametrami oraz zadaniami zależnymi – problemy te mogą być rozwiązywane przez przeszukiwanie losowe oraz heurystyki;

(iv) problemy ze zmiennymi parametrami oraz zadaniami zależnymi – problemy te mogą być rozwiązywane przez algorytmy działające w trybie on–line oraz zmodyfikowane algorytmy tradycyjne (ang. *modified traditional algorithms*).

Podobny podział zaproponowali M. Kalra i S. Singh w pracy [61], ograniczając się do algorytmów opartych o metaheurystyki. Autorzy podkreślają złożoność problemu harmonogramowania i wskazują na metaheurystyki jako odpowiednie narzędzie do uzyskiwania suboptymalnych rozwiązań tego typu problemów. W pracy wyszczególniono pięć metod i przypisano im klasy problemów harmonogramowania w rozwiązywaniu których znajdują zastosowanie. Są to:

(i) algorytm mrówkowy (ang. *ant colony optimization*): harmonogramowanie zadań, harmonogramowanie zadań z równoważeniem obciążenia, harmonogramowanie zadań pod kątem realizacji umowy o gwarantowanym poziomie świadczenia usług, harmonogramowanie zadań pod kątem zużycia energii;

(ii) algorytm genetyczny (ang. *genetic algorithm*): harmonogramowanie zadań, harmonogramowanie zadań z równoważeniem obciążenia, harmonogramowanie zadań pod kątem realizacji umowy o gwarantowanym poziomie świadczenia usług, harmonogramowanie zadań pod kątem zużycia energii;

(iii) optymalizacja stadna cząsteczek (ang. *particle swarm optimization*): harmonogramowanie zadań, harmonogramowanie zadań z równoważeniem obciążenia, harmonogra-

mowanie zadań pod kątem realizacji umowy o gwarantowanym poziomie świadczenia usług, harmonogramowanie zadań pod kątem zużycia energii;

(iv) algorytm ligi mistrzów (ang. *league championship algorithm*): harmonogramowanie zadań;

(v) algorytm nietoperza (ang. *bat algorithm*): harmonogramowanie zadań, harmonogramowanie zadań pod kątem realizacji umowy o gwarantowanym poziomie świadczenia usług.

Przedstawione klasyfikacje główny nacisk kładą na te cechy problemów harmonogramowania, które pozwolą dobrać odpowiednie narzędzie generujące jak najlepsze rozwiązanie w rozsądnym czasie.

5.2. Definicja problemu harmonogramowania

Zgodnie z terminologią zaproponowaną w [86], problem harmonogramowania zadań można przedstawić za pomocą trójki:

$$A|B|C, \tag{5.1}$$

gdzie:

- A – warstwa dostępnych zasobów sprzętowych i typ architektury środowiska, w ramach którego zostaną wykonane zadania;
- B – sposób przetwarzania zadań oraz ograniczenia z tym związane;
- C – kryteria harmonogramowania.

Dla centralnie zarządzanych środowisk, które mogą przetwarzać zadania równolegle, problem harmonogramowania można zdefiniować w następujący sposób (por. [67]):

$$Rm, CM|\{(batch/on-line), (independent/dependent/workflow), (static/dynamic)\}|(objectives), \quad (5.2)$$

gdzie:

- Rm – środowisko heterogenicznych zasobów sprzętowych, gdzie zadania mogą być przetwarzane w sposób równoległy;
- CM – środowisko zarządzane centralnie;
- $batch/on--line$ -– zadania przetwarzane pakietowo albo w trybie on–line,
- $independent/dependent/workflow$ – zadania niezależne, zadania pomiędzy którymi występują zależności oraz przepływ pracy;
- $static/dynamic$ — tryb statyczny/dynamiczny, w którym charakterystyki jednostek nie ulegają/ulegają zmianom w trakcie procesu harmonogramowania;
- $objectives$ – kryteria harmonogramowania.

Popularnym problemem harmonogramowania zadań, będącym przedmiotem rozważań w niniejszej pracy, jest harmonogramowanie niezależnych zadań w trybie pakietowym (ang. *independent batch scheduling*) [18, 49, 66, 67, 78]. W problemie tym zadania są gromadzone w ramach pakietów oraz niezależnie przetwarzane na dostępnych zasobach. Na podstawie wyżej przedstawionej terminologii, problem ten można zapisać następująco:

$$Rm, CM|\{batch, independent, (static/dynamic)\}|(objectives). \quad (5.3)$$

Harmonogramowanie niezależnych zadań w trybie pakietowym może zostać zrealizowane w sześciu krokach [68]:

(i) uzyskanie charakterystyk dostępnych jednostek;

(ii) uzyskanie charakterystyk oczekujących zadań;

(iii) uzyskanie informacji o lokalizacji danych potrzebnych do wykonania zadań;

(iv) przygotowanie pakietu zadań oraz harmonogramu ich wykonania na dostępnych jednostkach;

(v) alokacja zadań do jednostek obliczeniowych;

(vi) monitorowanie wykonania zadań

W niniejszej pracy przyjęto, że wszystkie potrzebne dane do wykonania zadania znajdują się na jednostkach obliczeniowych lub też ich pobranie nie wpłynie na czas wykonania zadania. Istnieje specjalna grupa problemów, nazywana harmonogramowaniem zorientowanym na dane (ang. *data-aware task scheduling*), która w sposób szczególny uwzględnia zagadnienia związane z lokalizacją danych potrzebnych w trakcie wykonywania zadań (por. [70, 111, 117]).

5.3. Kryteria harmonogramowania

Problemy harmonogramowania zadań mogą być rozważane pod kątem wielu kryteriów optymalizacyjnych. Najpopularniejszym z nich jest różnica między czasem rozpoczęcia i zakończenia sekwencji (całego pakietu) zadań (ang. *makespan*). To, jak i inne popularne kryteria mogą być zdefiniowane następująco [57, 61, 67]:

- różnica między czasem rozpoczęcia i zakończenia wykonywania wszystkich zadań pakietu – najbardziej popularne kryterium oparte o czas. Wskazuje na czas wykonania ostatniego zadania z puli. Kryterium to można zapisać w następujący sposób:

$$C_{\max} = \min_{S \in Schedules} \left\{ \max_{\substack{i \in Machines, \\ j \in Tasks_i}} \sum C_j \right\}, \tag{5.4}$$

gdzie C_j oznacza czas wykonywania zadania j, $Tasks_i$ oznacza zbiór wszystkich zadań przypisanych do wykonania przez jednostkę i, $Machines$ to zbiór wszystkich jednostek obliczeniowych, a $Schedules$ to zbiór wszystkich możliwych harmonogramów.

- suma czasów wykonania wszystkich zadań z puli (ang. *flowtime*):

$$F = \min_{S \in Schedules} \left\{ \sum_{j \in Tasks} C_j \right\}. \tag{5.5}$$

- opóźnienie (ang. *lateness*) – definiuje maksymalny czas jaki upłynął pomiędzy ostatecznym terminem wykonania zadania (ang. *deadline*), a faktycznym czasem zakończenia zadania. Maksymalne opóźnienie może być sformułowane następująco:

$$Lat_{\max} = \max_{j \in Tasks} Lat_j, \tag{5.6}$$

gdzie Lat_j oznacza opóźnienie w wykonaniu zadania j i może być obliczone za pomocą wzoru:

$$Lat_j = C_j - d_j,$$

gdzie C_j oznacza faktyczny czas ukończenia zadania j, a d_j – żądany ostateczny termin wykonania zadania.

- koszt (ang. *economic cost*) – opisuje całkowity koszt, jaki poniesie użytkownik za wykorzystane zasoby:

$$Cost_{total} = \min_{S \in Schedules} \left\{ \sum_{i \in Machines} (Cost_i \cdot T_i) \right\}, \tag{5.7}$$

gdzie $Cost_i$ oznacza koszt korzystania z maszyny i przez jednostkę czasową, a T_i to liczba jednostek czasowych korzystania z maszyny i; $Machines$ oznacza zbiór jednostek obliczeniowych.

- całkowite zużycie energii (ang. *the total energy consumption*) – definiuje sumaryczne zużycie energii podczas wykonania całego pakietu zadań:

$$E_{total} = \left\{ \sum_{i \in Machines} E_i \right\}, \tag{5.8}$$

gdzie E_i oznacza sumaryczne zużycie energii przez jednostkę i w celu wykonania wszystkich przypisanych do niej zadań z pakietu.

Często problem harmonogramowania rozważany jest pod kątem optymalizacji wielo-

kryterialnej. Przykładowo, w procesie harmonogramowania pod kątem zużycia energii powinien być również brany pod uwagę czas wykonania zadań [55, 56].

Przedstawione powyżej kryteria stanowią jedynie przykłady najpopularniejszych funkcji celu. Innymi warunkami harmonogramowania mogą być polityki bezpieczeństwa, przepustowość, dostępność danych czy priorytety zadań.

5.4. Model zadania i jednostki obliczeniowej

W rozważanym problemie harmonogramowania zadań mamy do czynienia z dwoma grupami obiektów: zadania (ang. *tasks*) oraz zasoby (ang. *resources*). W przypadku środowisk typowo obliczeniowych, modele często ograniczają się do charakterystyk opisujących zdolności i zapotrzebowania obliczeniowe (a nawet samych czasów realizacji zadań), pomijając tym samym charakterystyki związane z pamięcią operacyjną czy przestrzenią dyskową (por. [78, 104, 109]). W takim podejściu zasoby ograniczają się do jednostki obliczeniowej, nazywanej maszyną (ang. *machine*), a zadania do zbioru wykonywanych przez procesor operacji. Pod pojęciem maszyny/jednostki obliczeniowej może kryć się maszyna wirtualna, pojedynczy komputer, smartfon czy nawet niewielki klaster obliczeniowy, natomiast zadanie reprezentuje wykonywany program, usługę lub nawet zestaw składający się z wielu usług i programów.

Jedna z najpopularniejszych notacji opisująca zadanie i jednostkę obliczeniową została opisana w [67]. Niech:

- n – liczba zadań w pakiecie;
- m – liczba jednostek obliczeniowych.

Biorąc pod uwagę jedynie atrybuty związane ze zdolnościami obliczeniowymi procesora, zadania oraz jednostki mogą zostać scharakteryzowane za pomocą liczby operacji lub instrukcji wykonanych przez procesor. Zatem niech:

- Zadanie j będzie opisane przez zapotrzebowanie obliczeniowe (ang. *workload*) wyrażone

w operacjach zmiennoprzecinkowych (ang. *floating point operations, FLO*) oznaczane symbolem wl;

- Jednostka obliczeniowa i będzie opisana przez zdolność obliczeniową (ang. *computing capacity*), którą definiuje się jako liczbę operacji zmiennoprzecinkowych wykonywanych przez jednostkę w czasie jednej sekundy (ang. *floating point operations per second, FLOPS*) oznaczoną symbolem cc.[1]

Zbiór zapotrzebowań obliczeniowych zadań niech będzie opisany przez wektor $WL = [wl_1, \ldots, wl_n]$, a zbiór zdolności jednostek obliczeniowych przez wektor $CC = [cc_1, \ldots, cc_m]$.

Zadania w tym modelu są zadaniami niezależnymi od siebie. Liczba operacji opisująca zadanie może być określona m.in. na podstawie specyfikacji dostarczonej przez użytkownika, zebranych statystyk czy odczytów z dedykowanych rejestrów procesora (ang. *hardware performance counters*). Zdolności obliczeniowe maszyn mogą być oszacowane na podstawie wyników uzyskanych przez testy wydajnościowe (ang. *benchmarks*), np. przy wykorzystaniu bibliotek służących do obliczeń wysokiej wydajności BLAS/LAPACK.

5.4.1. Model macierzy ETC

Zaprezentowany model pozwala na oszacowanie czasu potrzebnego do wykonania zadań na każdej z jednostek obliczeniowych. S. Ali i wsp. w pracy [16] zaproponowali model macierzy ETC (ang. *Expected Time to Compute*), która reprezentuje oczekiwane czasy wykonania zadań na poszczególnych maszynach. Czas wykonania zadania j na jednostce i może być oznaczony jako $ETC[j][i]$ i opisany jako stosunek zapotrzebowania obliczeniowego zadania (wl_j) do zdolności obliczeniowej maszyny (cc_i) [67]:

$$ETC[j][i] = \frac{wl_j}{cc_i} \tag{5.9}$$

Wszystkie możliwe wartości $ETC[j][i]$ składają się na pełną macierz ETC, którą można zapisać: $ETC = [ETC[j][i]]_{n \times m}$. Elementy w wierszu macierzy estymują czasy wyko-

[1] W pracy zarówno liczba operacji opisująca zadania jak i zdolności obliczeniowe jednostek zostały poprzedzone przedrostkiem "giga". Zapotrzebowania oraz zdolności obliczeniowe mogą być opisane za pomocą innych miar, np. liczby instrukcji

nań danego zadania na każdej z maszyn, natomiast elementy w kolumnie to czasy wykonań każdego z zadań na danej maszynie.

Jak podaje [67], macierz ETC może być scharakteryzowana przez trzy parametry:

1. Różnorodność (niejednorodność) zasobów;
2. Różnorodność zadań;
3. Spójność macierzy.

Różnorodność maszyn przejawia się istotnymi różnicami w wartościach elementów wierszy macierzy ETC, natomiast różnorodność zadań powoduje wysoką zmienność czasów w ramach kolumn macierzy. Ostatnia z cech – spójność macierzy, dotyczy zachowania zależności przedstawionej we wzorze 5.9. Mówimy, że macierz jest spójna, jeśli dla każdej z par jednostek obliczeniowych i oraz $\hat{i}$ ($i \neq \hat{i}$) spełniony jest warunek: jeśli czas wykonania zadania j jest krótszy na i niż na $\hat{i}$, to wszystkie zadania mogą zostać wykonane szybciej na i niż na $\hat{i}$.

Na podstawie macierzy ETC można wyznaczyć czas wykonania wszystkich przydzielonych zadań ct (ang. *completion time*) dla każdej z jednostek obliczeniowych i:

$$ct_i = \sum_{j \in Tasks} ETC[j][i]. \tag{5.10}$$

Najwyższa wartość ($\max_{i \in Machines} ct_i$) będzie równa różnicy między czasem rozpoczęcia i zakończenia wykonywania wszystkich zadań pakietu (por. wzór 5.4).

5.4.2. Rozszerzenie modelu zadania i jednostki obliczeniowej

Przedstawiony powyżej model cechuje się dość istotną wadą. Współczesne urządzenia (komputery, smartfony, dedykowane karty graficzne) wyposażone są w procesory składające się nawet z kilkunastu rdzeni (a w przypadku procesorów graficznych – z kilku tysięcy). Ponadto wyspecjalizowane jednostki obliczeniowe mogą zawierać od kilkudziesięciu do nawet kilkuset procesorów będących w stanie przetwarzać złożone zadanie równolegle. Analogicznie,

wykonywane programy składają się z tych części, które muszą być wykonywane w sposób sekwencyjny (na jednym rdzeniu procesora) oraz tych, które mogą zostać zrównoleglone. Powyższy model nie uwzględnia tego typu charakterystyk jednostek obliczeniowych oraz zadań.

Dla potrzeb niniejszej pracy został zaproponowany rozszerzony model bazujący na Prawie Amdahla. W 1967 roku Gene Amdahl w pracy [17] zaproponował wzór na maksymalne możliwe przyspieszenie (ang. *speedup*) związane ze zrównolegleniem obliczeń. Prawo to pozwala oszacować teoretyczny maksymalny wzrost szybkości obliczeń przy użyciu wielu rdzeni procesora. Wzór na przyspieszenie można zapisać w następujący sposób:

$$S = \frac{T_1}{T_{cn}} = \frac{1}{F + \frac{1-F}{cn}}, \tag{5.11}$$

gdzie:

- T_1 – czas wykonania programu na jednym rdzeniu procesora (w pełni sekwencyjnie);
- T_{cn} – czas wykonania programu w sposób zrównoleglony na cn rdzeniach procesora;
- F – proporcja programu, która nie może zostać zrównoleglona (część sekwencyjna programu);
- cn – liczba rdzeni przeznaczonych do wykonania programu.

Biorąc pod uwagę możliwości zrównoleglania obliczeń, zadania oraz jednostki mogą zostać scharakteryzowane następująco:

- Zadanie $j = (wl^s, wl^p)$ – opisane przez liczbę operacji zmiennoprzecinkowych, które muszą być wykonane w sposób sekwencyjny (oznaczane symbolem wl^s) oraz przez liczbę operacji, które mogą zostać zrównoleglone (oznaczane symbolem wl^p).
- Jednostka obliczeniowa $i = (cc^c, cn)$ – opisana przez zdolność obliczeniową pojedynczego rdzenia (ang. *core*) procesora, wyrażona w wykonanych operacjach zmiennoprzecinkowych w czasie jednej sekundy (oznaczana symbolem cc^c) oraz przez liczbę rdzeni (oznaczonych symbolem cn).

Korzystając z zaproponowanych wyżej modeli, czas wykonania zadania j na jednostce i może przybrać formę elementu macierzy ETC, która uwzględnia możliwość przetwarzania równoległego (ang. *parallel processing, PP*) i może zostać sformułowany następująco:

$$ETC_{PP}[j][i] = \frac{wl_j^s}{cc_i^c} + \frac{wl_j^p}{cn_i \cdot cc_i^c}. \tag{5.12}$$

Powyższe podejście nie uwzględnia szeregu czynników, takich jak narzuty związane z potrzebą utworzenia wątków/procesów, problem równoważenia obciążenia (ang. *load balancing*) czy możliwa heterogeniczność zasobów obliczeniowych w ramach jednej jednostki. Ta ostatnia z wad może być w prosty sposób wyeliminowana poprzez definiowanie jednostek obliczeniowych w postaci wektora zmiennej długości, opisującego każdą, atomową podjednostkę obliczeniową (np. rdzeń procesora CPU, rdzeń procesora GPU).

Pomimo wspomnianych wad, w porównaniu do powszechnych w literaturze modeli, zaproponowane rozwiązanie w bardziej dokładny i realistyczny sposób odwzorowuje zarówno zadania, jak i jednostki w kontekście wykonywania obliczeń równoległych. Ponadto zaproponowane zmiany nie stoją w sprzeczności z intuicyjnym pojęciem jednostki obliczeniowej/maszyny. Zrównoleglenie obliczeń może być wykonywane za pomocą powszechnie znanych rozwiązań, jak OpenMP (ang. *Open Multi-Processing*) czy MPI (ang. *Message Passing Interface*).

Aby rozróżnić różnego rodzaju modele zadań i jednostki, dostosowane pod kątem przypadku użycia, w ramach niniejszej pracy proponuje się wprowadzenie pojęcia tzw. *profili*. Poszczególne profile identyfikują charakterystyki zadań i jednostek, na podstawie których przeprowadzany jest proces harmonogramowania. W problemie harmonogramowania niezależnych zadań możemy wyszczególnić m.in. następujące profile:

1. Profil podstawowy (ang. *basic profile*), w którym zadanie opisane jest przez zapotrzebowanie obliczeniowe wl, a jednostka obliczeniowa przez zdolność obliczeniową cc. Profil ten szerzej przedstawiony został w podrozdziale 5.4.

2. Profil przetwarzania równoległego (ang. *parallel computation profile*), który jest oparty

na prawie Amdahla. Profil ten wyróżnia część sekwencyjną oraz równoległą zadania, a w przypadku jednostki obliczeniowej podaje liczbę rdzeni i ich moc obliczeniową. Jest to profil zdefiniowany w niniejszym podrozdziale.

3. Profil energetyczny (ang. *energy profile*), który rozszerza profil podstawowy o charakterystyki związane ze zużyciem energii jednostek obliczeniowych zarówno w stanie bezczynności, jak i wykonywania obliczeń [55, 56, 58]. Jednostka obliczeniowa i zdefiniowana jest przez trójkę: (cc, P^{busy}, P^{idle}), gdzie cc – zdolność obliczeniowa jednostki (w GFLOPS), P^{busy} – moc potrzebna do zapewnienia stanu wykonywania obliczeń (w dżulach), P^{idle} – moc potrzebna do zapewnienia stanu bezczynności (w dżulach). Na podstawie tych charakterystyk jesteśmy w stanie oszacować energię zużytą przez jednostkę i:

$$E_i = (P_i^{busy} * t_i^{busy} + P_i^{idle} * t_i^{idle}), \tag{5.13}$$

gdzie t_i^{busy} – to czas (wyrażony w sekundach), w którym jednostka i znajduje się w stanie wykonywania obliczeń, a t_i^{idle} to czas (wyrażony w sekundach), w którym jednostka znajduje się w stanie bezczynności. Natomiast całkowitą energię E_{total} potrzebną do wykonania pakietu zadań możemy określić jako sumę energii zużytej przez wszystkie jednostki obliczeniowe:

$$E_{total} = \sum_{i=1}^{m} E_i. \tag{5.14}$$

4. Profil z poziomem bezpieczeństwa (ang. *secure profile*) rozszerza charakterystyki zadań o parametr pożądanego przez zadanie poziomu bezpieczeństwa sd (ang. *security demand*), jak i poziomu bezpieczeństwa oferowanego przez jednostkę tl (ang. *trust level*) [48, 49, 58]. Parametry te przyjmują wartości z zakresu [0, 1] i odnoszą się do konfiguracji środowiska zadań, stosowanych uwierzytelnień, zabezpieczeń konfiguracji, a w efekcie podatności zadania na niepowodzenia w przypadku braku realizacji oczekiwanego poziomu bezpieczeństwa. W profilu tym zadanie j definiowane jest jako (wl, sd), natomiast jednostka i przyjmuje postać (cc, tl). Podejście to pozwala na wprowadzenie macierzy prawdopodobieństwa niepowodzenia wykonania zadań (ang. *machine failure*

probability matrix) danej wzorem [48, 49]:

$$P_f[j][i] = \begin{cases} 0 & , \quad sd_j \leqslant tl_i \\ 1 - e^{-\alpha(sd_j - tl_i)} & , \quad sd_j > tl_i \end{cases}, \tag{5.15}$$

gdzie α jest interpretowany jako współczynnik niepowodzenia i jest globalnym parametrem modelu.

Parametry *sd* oraz *tl* mogą być również wykorzystane w celu estymacji potrzebnego czasu do zapewnienia żądanego poziomu bezpieczeństwa [58]. W efekcie można wprowadzić rozszerzony model macierzy ETC o narzut czasowy związany z zapewnieniem bezpieczeństwa (ang. *Security Biased Expected Time to Compute*):

$$SBETC[j][i] = wl_j/cc_i + b(sd_j, wl_j, tl_i, cc_i). \tag{5.16}$$

Powyższą taksonomię można dowolnie rozszerzać – w zależności od potrzebnych charakterystyk zadań i maszyn (np. o charakterystyki związane z możliwością przetwarzania danych), a także dokonywać łączeń profili.

5.5. Podejście ewolucyjne w rozwiązywaniu problemu harmonogramowania zadań

Problem harmonogramowania zadań należy do grupy problemów NP–zupełnych [41, 61, 67, 115]. Złożoność problemu może być poszerzana na skutek narzucanych charakterystyk i ograniczeń. Możemy do nich zaliczyć [67]: (i) kryteria harmonogramowania (optymalizacja jednokryterialna vs. wielokryterialna), (ii) typ środowiska (środowisko statyczne vs. dynamiczne), (iii) sposób przetwarzania zadań oraz (iv) zależności pomiędzy zadaniami. W związku z powyższym wymagane są rozwiązania, które są w stanie sprostać zarówno dynamice systemu jak i różnorodności wymagań narzuconych na system. Niestety, dla takich problemów nie są znane algorytmy generujące optymalne rozwiązanie w czasie wielomianowym. Rozwiązania bazujące na wyszukiwaniu wyczerpującym (ang. *exhaustive search*) są wręcz niewykonalne z powodu bardzo wysokiego kosztu obliczeniowego, zarówno wynikają-

cego z generowania jak i oceny wszystkich możliwych harmonogramów [61, 127]. Ponieważ w większości środowisk rozproszonych czas gra kluczową rolę, istotne jest szybkie podejmowanie decyzji dotyczących przydziału zasobów. Proces harmonogramowania zadań i wykonania wygenerowanego harmonogramu w sposób zgodny z przewidywaniami jest priorytetowym elementem działania środowiska i powinien być wykonywany w sposób niezwłoczny, tak, aby nie powodować dodatkowych opóźnień.

Aby sprostać powyższym wymaganiom, dotychczas zaproponowano wiele skutecznych metod opartych o algorytmy heurystyczne i metaheurystyczne – co wyraźnie można zaobserwować w przytoczonych taksonomiach. W ciągu ostatnich lat metody te zyskały dużą popularność, którą zawdzięczają swojej wydajności i efektywności w rozwiązywaniu dużych, złożonych problemów, do których można zaliczyć problem harmonogramowania zadań [67, 78, 96, 97, 109].

Biorąc pod uwagę dokonany przegląd w dziedzinie harmonogramowania zadań, uzasadnionym wydaje się być wybór algorytmów ewolucyjnych jako metody generowania suboptymalnych rozwiązań. Idea algorytmów ewolucyjnych czerpie inspirację z procesów zachodzących w naturze. Obliczenia ewolucyjne opierają się na symulacji procesu ewolucji, który dzięki odpowiedniej reprezentacji problemu oraz dedykowanym operatorom pozwala na heurystyczne przeszukiwanie przestrzeni potencjalnych rozwiązań problemu [39]. Algorytmy ewolucyjne swoją historią sięgają roku 1957, kiedy to Alex Fraser zaproponował sposób symulacji procesu genetycznego [95]. Przez długi czas rozwijane były niezależnie od siebie, w związku z czym powstało wiele odmiennych podejść różniących się szczegółami. Do najpopularniejszych można zaliczyć programowanie ewolucyjne, strategie ewolucyjne, algorytmy genetyczne czy programowanie genetyczne. W wyniku ich rozwoju, a także procesu wzajemnego upodobniania, podejścia te są rzadko spotykane w swoich pierwotnych formach. Dlatego, aby uniknąć problemu klasyfikacji nowych rozwiązań, ustalono wspólną nazwę obejmującą niniejszy nurt: algorytmy ewolucyjne (ang. *evolutionary algorithms*) – czasami spotykane pod nazwą obliczenia ewolucyjne (ang. *evolutionary computations*) [19].

Algorytmy ewolucyjne bazują na terminologii zapożyczonej z genetyki. Wszystkie organizmy żywe posiadają specyficzny dla siebie materiał genetyczny, który zawiera informacje

o nich samych. Materiał ten może być przekazywany kolejnym pokoleniom w procesie reprodukcji. Cechy osobników zapisane są w genach, które tworzą chromosomy. W wyniku procesu krzyżowania powstają nowe osobniki bazujące na materiale genetycznym swoich rodziców, ale różniące się od nich. Każdy z osobników cechuje się pewnym stopniem przystosowania do życia w środowisku – osobniki najlepiej przystosowane mają największe szanse na przeżycie i przekazanie swoich genów kolejnemu pokoleniu. Powyższa idea wykorzystywana jest do rozwiązywania zadań optymalizacji [94].

Ogólne podejście ewolucyjne zastosowane w problemie generowania harmonogramów niezależnych zadań obliczeniowych zostało sformułowane zgodnie z algorytmem 1 (por. [67]). Sprecyzowany model monitorowanego środowiska, ze szczególnym uwzględnieniem procesu harmonogramowania, jak i również model systemu agentowego, zostały przedstawione w kolejnym rozdziale.

Algorytm 1 Ogólny algorytm ewolucyjny generacji harmonogramu niezależnych zadań

1: **function** GenerujHarmonogram-AlgorytmEwolucyjnyOgólny
2: $i = 0$;
3: Wygeneruj początkową populację P^0 harmonogramów S;
4: Dokonaj oceny P^0;
5: **while** (**not** *warunekStopu*) **do**
6: Wybierz podzbiór T^i przeznaczony do reprodukcji; $T^i := Select(P^i)$;
7: Dokonaj krzyżowania par osobników z podzbioru T^i z prawdopodobieństwem $\mathcal{P}_c$; $P_c^i := Cross(T^i, \mathcal{P}_c)$;
8: Dokonaj mutacji osobników populacji P_c^i z prawdopodobieństwem $\mathcal{P}_m$; $P_m^i := Mutate(P_c^i, \mathcal{P}_m)$;
9: Dokonaj oceny P_m^i;
10: $P^i \leftarrow P_m^i$;
11: $i \leftarrow i + 1$;
12: **end while**
13: **return** Najlepszy harmonogram S z populacji P^i;
14: **end function**

Cykl ewolucji może kończyć się wówczas, gdy wartość funkcji przystosowania jest wystarczająco duża, został osiągnięty limit cyklów lub stwierdzono, że stan populacji bazowej świadczy o stagnacji algorytmu [19].

Rozdział 6

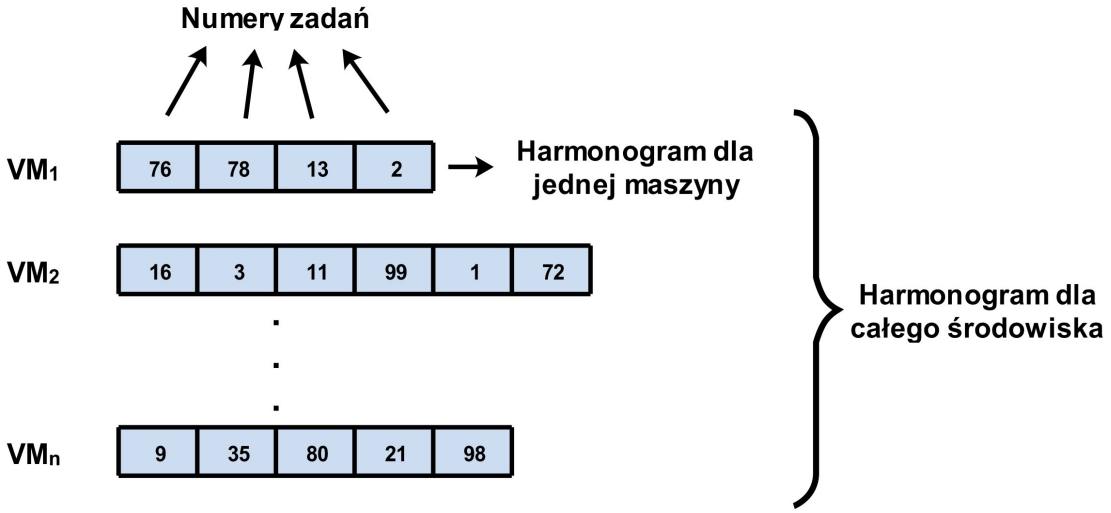
Model środowiska rozproszonego i systemu monitorującego

W niniejszym rozdziale sformułowano model dynamiki rozproszonego środowiska, w którym zachodzi proces harmonogramowania, model samego procesu harmonogramowania niezależnych zadań obliczeniowych, zaproponowano reprezentację populacji dla procesu ewolucyjnego, a także przedstawiono formalny zapis agentowego systemu monitorującego. Większość elementów przedstawionych w tym rozdziale to rozwiązania nowatorskie.

6.1. Reprezentacja populacji

Kluczowym elementem procesu ewolucyjnego jest odpowiednio zamodelowana populacja osobników. Każdy z osobników składa się z chromosomów, zbudowanych z poszczególnych genów. De Jong w pracy [32] z 1988 roku przedstawił dwa rywalizujące ówcześnie ze sobą podejścia do realizacji procesu ewolucyjnego: Michigan oraz Pitt. Podejście Pitt reprezentuje powszechnie znany sposób reprezentacji populacji, gdzie każdy osobnik odpowiada jednemu rozwiązaniu problemu. Natomiast równolegle prowadzono prace nad podejściem Michigan, w którym to każdy z osobników reprezentuje tylko część rozwiązania problemu – pełne rozwiązanie niesie cała populacja [75]. W niniejszej pracy zaproponowano oryginalną reprezentację problemu harmonogramowania opierającą się o podejście Michigan.

Ogólna struktura populacji została przedstawiona na rys. 6.1. Można ją scharaktery-



Rysunek 6.1: Reprezentacja populacji dla podejścia ewolucyjnego w rozwiązywaniu problemu harmonogramowania niezależnych zadań.

zować w kilku następujących punktach:

1. Pojedynczy gen reprezentuje zadanie z pakietu zadań.
2. W populacji znajduje się w sumie n genów.
3. Zestaw zadań (genów) tworzy chromosom.
4. Chromosom jest harmonogramem dla pojedynczej jednostki obliczeniowej.
5. W populacji występuje m osobników.
6. Każdy chromosom składa się przynajmniej z jednego genu.
7. Każdy osobnik składa się z jednego chromosomu.
8. Zadania w całej populacji są unikalne (nie powtarzają się).
9. Allele, czyli wartości poszczególnych genów, są ściśle związane z charakterystykami jednostek obliczeniowych, do których zadania (geny) zostaną przypisane – wartości te wyznaczane są na podstawie wzoru 5.12.

Postać funkcji przystosowania zależna jest od kryteriów harmonogramowania przedstawionych w podrozdziale 5.3. Przyjętą funkcją oceny populacji w ramach niniejszej pracy

jest różnica między czasem rozpoczęcia i zakończenia wykonywania wszystkich zadań pakietu dana wzorem 5.4. Celem algorytmu jest znalezienie populacji minimalizującej wartość tej funkcji. Populacja początkowa generowana jest poprzez losowy, równomierny przydział zadań. Krzyżowanie osobników dokonywane jest na losowo wybranej parze rodziców (jeden rodzic z części lepiej przystosowanej, drugi z części gorzej przystosowanej). Operacja krzyżowania wykonywana jest na podstawie wylosowanego punktu krzyżowania, poprzez zamianę ze sobą genów po prawej stronie. Operator mutacji jest pomijany.

6.2. Formalny opis dynamiki środowiska

Zaproponowany system monitoringu działa w ramach rozproszonego środowiska, którego celem jest realizacja powierzonych zadań obliczeniowych. Istotnym elementem środowiska jest moduł generowania harmonogramów – tzw. planista (ang. *scheduler*). Z punktu widzenia przetwarzania pakietów zadań istotne jest zdefiniowanie podstawowych założeń, jakie środowisko, a także powierzone zadania, muszą spełnić. Ma to bezpośredni wpływ na sposób funkcjonowania środowiska w czasie. W niniejszym podrozdziale przedstawiono dynamikę systemu w kontekście decyzji związanych z przetwarzaniem zadań, a także ograniczenia, jakie zostały nałożone na system oraz zadania.

System działa zgodnie z następującymi ograniczeniami i założeniami:

1. Liczba jednostek obliczeniowych m jest skończona i parzysta.
2. Liczba zadań obliczeniowych n jest znacznie większa od liczby jednostek obliczeniowych m, $n \gg m$.
3. Czas generowania harmonogramu, w porównaniu do czasu wykonywania zadań jest znikomy.
4. Czas rekonfiguracji maszyn wirtualnych (zwiększenia zdolności obliczeniowych jednostki) jest znaczący.
5. Zadania są przetwarzane pakietowo (tryb wsadowy).

6. Charakterystyki pakietów zadań nie cechują się nadmierną wartością odchylenia standardowego.

7. Sumaryczna liczba wszystkich jednostek obliczeniowych (w stanie bezczynności, jak i zajętych wykonywaniem obliczeń) jest stała.

8. Każda z jednostek obliczeniowych ma przypisane co najmniej jedno zadanie.

9. Zadania są niezależne.

10. Każde zadanie jest wykonywane tylko raz, na jednej jednostce obliczeniowej.

11. Wszystkie zadania muszą zostać przypisane i wykonane.

Ponadto jednym z podstawowych założeń systemu jest jego umiejscowienie w czasie i dyskretyzacja realizacji działań zachodzących w systemie. Przy takim założeniu wszystkie możliwe decyzje podejmowane (czy też widoczne dla obserwatora) są w dyskretnych momentach czasowych, oznaczanych przez t_k, gdzie $t_1 < t_2 < \cdots < t_k < \ldots$ oraz $t_1 = 0$, natomiast k ($k \geqslant 1$) jest numerem wygenerowanego momentu. Podejście to pozwala na manipulację obciążeniem wynikającym z działania systemu monitorującego. Wpływ wyboru momentów czasowych t_k na wydajność systemu został przeanalizowany w pracach [47, 57, 58]. Momenty te mogą być generowane w sposób losowy, na podstawie równomiernego rozkładu, na podstawie decyzji podjętych przez system agentowy, jak również w oparciu o wyznaczone cele optymalizacyjne. Czas pomiędzy momentami będziemy oznaczać przez τ_k, gdzie $\tau_k = t_k - t_{k-1}$, $k \geqslant 2$. W praktyce momenty czasowe będą tożsame ze stanem, w którym zakończyliśmy proces formowania pakietu zadań i rozpoczynamy proces harmonogramowania, a następnie wykonania zadań. A więc poprzez k możemy również identyfikować pakiet zadań.

Jednocześnie należy *a priori* przyjąć, że czas generacji harmonogramu wraz z nawiązaniem odpowiednich połączeń w systemie agentowym oraz rozesłania zadań do wykonania (oznaczany przez ξ_k) spełnia warunek: $\tau_k > \xi_k$ dla każdego k. To założenie wymaga, aby wartości τ_k nie były zbyt małe w odniesieniu do rozmiaru przetwarzanego pakietu zadań.

Pakiety formowane są z nadsyłanych przez użytkowników zadań i przetwarzane jeden po drugim. Formowanie pakietu kończy się w obranym momencie czasowym k, i w tym samym momencie rozpoczyna się proces harmonogramowania (zakłada się, że w każdym momencie czasowym k istnieje niepusty pakiet przesłanych zadań). Pierwszy pakiet zadań jest gotowy już w momencie czasowym $t_1 = 0$. Rozmiar (liczba zadań) k-tego pakietu będzie oznaczany przez n_k. W wyniku procesu harmonogramowania zadań otrzymujemy informację na temat przydziału zadań do jednostek obliczeniowych. Dla j-tego zadania w k-tym pakiecie informację taką będziemy oznaczać przez $s_{j,k}$. Każda informacja $s_{j,k}$ przyjmuje jedną wartość ze skończonego zbioru $\{1, 2, \ldots, m\}$. Informacje dla całego k-tego pakietu (będące harmonogramem) możemy zapisać w postaci wektora $S_k = (s_{1,k}, \ldots, s_{n_k,k})$, gdzie $k \geqslant 1$, a wymiar jest równy $1 \times n_k$.

Podobnie, zgodnie z założeniem w którym każda jednostka ma przypisane co najmniej jedno zadanie, możemy zdefiniować wektor decyzji dla każdej z m jednostek obliczeniowych. Decyzją będziemy nazywać stan, w którym zadanie jest przypisane do jednostki. Wektor decyzji przydziału zadań z pakietu k dla jednostki obliczeniowej i możemy zapisać w następujący sposób:

$$D_{i,k} = (\mathbb{1}\{s_{1,k} = i\}, \ldots, \mathbb{1}\{s_{n_k,k} = i\}) = \sum_{j=1}^{n_k} E_j \cdot \mathbb{1}\{s_{j,k} = i\} = (d_{i,1,k}, \ldots, d_{i,j,k}), \qquad (6.1)$$

gdzie $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n_k$, natomiast $\mathbb{1}\{a = b\}$ oznacza funkcję charakterystyczną (ang. *indicator function*), taką, że $\mathbb{1}\{x = y\} = 1$, jeśli $x = y$ oraz $\mathbb{1}\{x = y\} = 0$, jeśli $x \neq y$. Przez E_j oznaczony jest wektor o wymiarze $1 \times n_k$, w którym tylko element na pozycji j jest równy 1, a wszystkie pozostałe równe 0, tj.:

$$E_j = (\underbrace{0, \ldots, 0}_{j-1}, 1, \underbrace{0, \ldots, 0}_{n-j}), \ 1 \leqslant j \leqslant n_k.$$

Zgodnie z powyższym, wektor $D_{i,k}$ składa się jedynie z elementów równych zero lub jeden. Jeśli j-ty element wektora jest równy 1, to oznacza, że j-te zadanie zostało przydzielone do jednostki i. W przeciwnym razie zadanie zostanie wykonane na innej jednostce.

Na podstawie wszystkich wektorów decyzji (dla każdej z jednostek obliczeniowych),

może zostać wyznaczony wektor reprezentujący harmonogram całego pakietu k:

$$S_k = \mathcal{S}_m(D_{1,k}, \ldots, D_{m,k}) = 1 \cdot D_{1,k} + 2 \cdot D_{2,k} + \cdots + m \cdot D_{m,k} = (s_{1,k}, \ldots, s_{n_k,k}), \quad (6.2)$$

gdzie $\mathcal{S}_m(\cdot)$ jest operatorem budującym harmonogram dla m jednostek obliczeniowych na podstawie m wektorów decyzji.

W niniejszej pracy wykorzystywany jest profil zadania i jednostki, który uwzględnia możliwości przetwarzania równoległego. Profil ten szczegółowo został opisany w podrozdziale 5.4.2. Zgodnie z profilem, każde zadanie w pakiecie jest scharakteryzowane poprzez zapotrzebowania obliczeniowe $wl^s_{j,k}$ oraz $wl^p_{j,k}$ wyrażone w GFLO (ang. *giga floating point operations*). Pierwszy parametr opisuje część zadania, która musi być wykonana sekwencyjnie (na jednym rdzeniu procesora), natomiast drugi opisuje część, która może zostać zrównoleglona. Dla k-tego pakietu, który zawiera n_k zadań, wektor zapotrzebowań obliczeniowych reprezentujący operacje sekwencyjne można zapisać w formie:

$$WL^s_k = [wl^s_{1,k}, \ldots, wl^s_{n_k,k}]. \quad (6.3)$$

Analogicznie, wektor zapotrzebowań obliczeniowych reprezentujący operacje mogące być wykonane równolegle:

$$WL^p_k = [wl^p_{1,k}, \ldots, wl^p_{n_k,k}]. \quad (6.4)$$

Należy przyjąć, że mediana i wariancja dla wektorów WL^s_k i WL^p_k są skończone, a rozkład nie jest długoogonowy (ang. *heavy-tailed distribution*). Takie założenie pozwoli uniknąć sytuacji, w których jednostka obliczeniowa zostanie zablokowana przez niewspółmiernie duże zadanie.

Podobnie można zapisać charakterystyki jednostek obliczeniowych. Zakładając, że każdy z rdzeni w ramach jednostki obliczeniowej charakteryzuje się taką samą zdolnością obliczeniową oraz że zdolności te nie ulegają zmianie w czasie, a każda z jednostek jest zawsze dostępna (to znaczy, że nie uległa np. awarii), możemy sformułować wektor liczby rdzeni:

$$CN = [cn_1, \ldots, cn_m], \quad (6.5)$$

oraz wektor odpowiadających im zdolności obliczeniowych:

$$CC^c = [cc_1^c, \ldots, cc_m^c]. \tag{6.6}$$

Analogicznie, na podstawie wzorów 5.10 i 5.12, czas jaki należy poświęcić na wykonanie zadań z pakietu k dla poszczególnych jednostek obliczeniowych, może być podany w formie wektora:

$$CT_k = [ct_{1,k}, \ldots, ct_{m,k}]. \tag{6.7}$$

Wartości wektora CT_k można wyznaczyć na podstawie wygenerowanego harmonogramu przedstawionego w postaci wektora decyzji (D_k), a także na podstawie wektorów opisujących charakterystyki zadań i jednostek obliczeniowych:

$$ct_{i,k} = \frac{D_{i,k} \cdot (WL_k^s)^T}{cc_i^c} + \frac{D_{i,k} \cdot (WL_k^p)^T}{cn_i \cdot cc_i^c}. \tag{6.8}$$

W niniejszej rozprawie zakłada się ciągłość pracy systemu w czasie, tj. wysyłanie przez użytkowników zadań, ich gromadzenie, formowanie pakietów, następnie harmonogramowanie i wykonywanie. Czynności te powtarzane są tak długo, jak zachodzi potrzeba. Proces formowania pakietu kończy się w momencie czasowym t_k. Przy założeniu, że $\tau_{k+1} \geqslant \xi_k + \max CT_k$, wszystkie zadania wykonają się przed rozpoczęciem procesowania nowego pakietu zadań. Problem staje się bardziej złożony, jeśli dopuszczamy sytuacje, w których czas realizacji harmonogramu wykracza poza wygenerowany moment czasowy t_k. W związku z powyższym należy wprowadzić pojęcie zaległości w wykonywaniu zadań (ang. *backlog*). Niech zatem $u_{i,k}$ będzie zaległością w wykonywaniu zadań dla jednostki obliczeniowej i w momencie czasowym t_k. Przy założeniu, że pierwszy pakiet zadań wykonywany jest w okresie τ_2, to zaległości dla wszystkich jednostek w momencie $k = 1$ będą równe zero. Dla $k \geqslant 2$ zaległość możemy wyznaczyć wzorem:

$$u_{i,k} = \max\left(0, u_{i,k-1} + ct_{i,k-1} + \xi_{k-1} - \tau_k\right), \ 1 \leqslant i \leqslant m. \tag{6.9}$$

Korzystając z powyższych notacji, czas potrzebny na wykonanie całego pakietu zadań

k (por. wzór 5.4), przy założeniu, że $\tau_{k+1} \geqslant \xi_k + \max CT_k$, można zapisać jako:

$$C_{max_k} = \max_{1 \leqslant i \leqslant m} ct_{i,k} + \xi_k. \tag{6.10}$$

Uwzględniając możliwe zaległości w wykonywaniu zadań z poprzednich pakietów, wzór ten przyjmie postać:

$$C_{max_k} = \max_{1 \leqslant i \leqslant m} (u_{i,k} + ct_{i,k}) + \xi_k. \tag{6.11}$$

6.3. Formalny opis procesu generacji harmonogramu

W niniejszym podrozdziale przedstawiono w sposób formalny proces generacji harmonogramu dla pojedynczego pakietu zadań. Proces ten rozpoczyna się w każdym momencie t_k i z założenia nie kończy się później niż w momencie t_{k+1} ($k \geqslant 1$). Korzystając z notacji przedstawionej w poprzednich podrozdziałach, w tym podrozdziale pominięto indeks k określający numer sekwencyjny przetwarzanego pakietu zadań. Przedstawione rozważania dotyczą pojedynczego pakietu zadań.

W związku z powyższym, dla jednostki obliczeniowej i wektor reprezentujący harmonogram dla całego pakietu można zapisać jako $S = (s_1, \ldots, s_n)$, gdzie elementy $s_j \in \{1, 2, \ldots, m\}$, a wymiar wektora jest równy liczbie zadań w pakiecie: $1 \times n$. Proces harmonogramowania rozważany jest jako proces iteracyjny ze skończoną, z góry określoną liczbą iteracji I_{max}. Założenie to wynika z charakteru rozważanego problemu – harmonogramowanie należy do problemów NP-zupełnych – i wyboru podejścia ewolucyjnego jako metody pozwalającej na wygenerowanie suboptymalnego rozwiązania. Reprezentacja populacji została omówiona w podrozdziale 6.1.

Z matematycznego punktu widzenia, proces generowania harmonogramu może być zapisany w różny sposób. W ramach niniejszej pracy proces ten postrzegany jest jako powtarzające się zastosowanie procedury, która działając na elementach wektora S generuje wektor reprezentujący nowy harmonogram. Oznaczmy tą procedurę za pomocą operatora

$\mathcal{F}$:

$$\mathcal{F} : (s_1, \ldots, s_n) \rightarrow (s'_1, \ldots, s'_n), \tag{6.12}$$

gdzie $s_j, s'_j \in \{1, 2, \ldots, m\},\ 1 \leqslant j \leqslant n, \forall(k,l); k,l = 1, \ldots, n : s_k \neq s_l, s'_k \neq s'_l$.

Odwzorowanie $\mathcal{F}$ jest więc zestawem działań, które po zastosowaniu kolejno przekształcają wektor informacji o przydziale zadań do jednostek obliczeniowych, na nowy harmonogram: $\mathcal{F}(s_1, \ldots, s_n) = (s'_1, \ldots, s'_n)$. Inaczej mówiąc, odwzorowanie $\mathcal{F}$ jest opisem pojedynczej iteracji (epoki) w procesie ewolucyjnym algorytmu użytego do generacji harmonogramu. Algorytm zawsze rozpoczyna od wygenerowania losowego, wstępnego harmonogramu S (i odpowiadających mu wektorów decyzji). W ramach niniejszej pracy, generowanie wstępnego harmonogramu można zdefiniować następująco:

$$S = \mathcal{T}(\underbrace{1, \ldots, m, \ldots, 1, \ldots, m}_{\lfloor \frac{n}{m} \rfloor m}, 1, \ldots, n - \lfloor \frac{n}{m} \rfloor m), \tag{6.13}$$

gdzie operator $\mathcal{T}$ w sposób losowy zmienia pomiędzy sobą wartości elementów zbioru (ang. *shuffling*).

Na podstawie wektora decyzji, a także charakterystyk zadań i jednostek obliczeniowych oraz informacji o zaległości w wykonywaniu zadań możemy oszacować czas potrzebny do ukończenia wszystkich przypisanych zadań (w tym określanych jako zaległości). Jest to informacja pozwalająca na ocenę jakości harmonogramu dla każdej jednostki i. Czas ten możemy zapisać w postaci wektora $V = (v_1, \ldots, v_m)$, a poszczególne elementy można obliczyć w następujący sposób:

$$v_i = ct_i + u_i = \frac{D_i \cdot (WL^s)^T}{cc_i^c} + \frac{D_i \cdot (WL^p)^T}{cn_i \cdot cc_i^c} + u_i, \tag{6.14}$$

gdzie $1 \leqslant i \leqslant m$.

W powyższym wzorze pominięto czas poświęcony na generację harmonogramu (ξ), a także komunikację agentów oraz rozesłanie zadań do wykonania – jest to informacja niedostępna dla planisty (a więc nie jest brana pod uwagę w procesie generowania harmonogramów).

Planista, po wygenerowaniu wstępnego harmonogramu, dokonuje jego oceny za pomocą funkcji przystosowania:

$$Fitness(S) = max(V). \tag{6.15}$$

Zadaniem procesu ewolucji jest minimalizacja wartości funkcji przystosowania.

Po wygenerowaniu i ocenie wstępnego harmonogramu rozpoczyna się proces ewolucji. Wynik po pierwszej epoce (iteracji) można zapisać jako $\mathcal{F}(s_1, \ldots, s_n)$, po drugiej jako $\mathcal{F}^2(s_1, \ldots, s_n)$, itd. Sekwencja akcji potrzebnych do wykonania odwzorowania $\mathcal{F}$ (czyli jednej epoki algorytmu ewolucyjnego) może zostać opisana w postaci następujących kroków:

1. **Dobór osobników do reprodukcji**

   Proces doboru osobników w pary, które następnie będą podlegać reprodukcji, rozpoczyna się od uszeregowania wektora V, zbudowanego na podstawie wektora decyzji D. Przez V^{sort} oznaczmy wektor powstały w wyniku działania operatora $\mathcal{R}(\cdot)$, który szereguje elementy w porządku rosnącym, na rzecz wektora V. Formalnie, możemy zapisać to jako:

   $$V^{sort} = \mathcal{R}(v_1, \ldots, v_m) = (v_1^{sort}, \ldots, v_m^{sort}). \tag{6.16}$$

   Następnie elementy wektora V dzielimy na dwa rozłączne zbiory: $\mathcal{A}$ i $\mathcal{B}$. Oba zbiory są równoliczne (zgodnie z założeniami przedstawionymi w podrozdziale 6.2, liczba jednostek obliczeniowych jest parzysta), a liczba elementów każdego z nich jest równa $m/2$. Zbiory te są konstruowane w następujący sposób:

   $$\mathcal{A} = \left\{ i : v_i < v^{sort}_{\frac{m}{2}+1}, 1 \leqslant i \leqslant m \right\}, \ \mathcal{B} = \left\{ i : v_i > v^{sort}_{\frac{m}{2}}, 1 \leqslant i \leqslant m \right\} = \{1, 2, \ldots, m\} \setminus \mathcal{A}. \tag{6.17}$$

   Innymi słowy, każdy element zbioru $\mathcal{A}$ jest mniejszy lub równy najmniejszemu elementowi zbioru $\mathcal{B}$.

   Elementy zbioru $\mathcal{A}$ oznaczmy jako $\{a_1, \ldots, a_{m/2}\}$, natomiast elementy zbioru $\mathcal{B}$ jako $\{b_1, \ldots, b_{m/2}\}$.

   W procesie reprodukcji krzyżowaniu podlegają jednostki obliczeniowe reprezentowane przez pary elementów – jeden element wylosowany ze zbioru $\mathcal{A}$ i jeden element ze

zbioru $\mathcal{B}$. Proces wyboru można zapisać w następujący sposób:

$$a = \sum_{j=1}^{\frac{m}{2}} a_j \cdot \mathbb{1}\left\{\frac{j-1}{\frac{m}{2}} \leqslant \varepsilon < \frac{j}{\frac{m}{2}}\right\}, \quad b = \sum_{j=1}^{\frac{m}{2}} b_j \cdot \mathbb{1}\left\{\frac{j-1}{\frac{m}{2}} \leqslant \varepsilon < \frac{j}{\frac{m}{2}}\right\}, \tag{6.18}$$

gdzie a i b reprezentują jednostki wylosowane do procesu reprodukcji, a ε jest zmienną losową o rozkładzie równomiernym z przedziału $[0, 1)$.

Następnie ze zbiorów $\mathcal{A}$ oraz $\mathcal{B}$ usuwane są wylosowane jednostki a i b.

2. **Operacja krzyżowania i mutacji**

Operacja krzyżowania sprowadza się do wymiany genów (zadań) pomiędzy osobnikami wyłonionymi w poprzednim punkcie. Wykonywane jest krzyżowanie jednopunktowe, a punkt podziału wyznaczany jest w sposób losowy. Wymianie podlegają geny znajdujące się po prawej stronie od wylosowanego punktu krzyżowania. W rezultacie zmianie ulegają wektory decyzji (por. wzór 6.1) dla jednostek a oraz b, które możemy wyznaczyć w następujący sposób:

$$D_a = \sum_{j=1}^{n} E_j \cdot \mathbb{1}\left\{d_{a,j} = a\right\} \cdot \mathbb{1}\left\{\sum_{k=1}^{j} \mathbb{1}\{d_{a,k} = a\} \leqslant N_a \cdot \varepsilon + 1\right\} + \\ + \sum_{j=1}^{n} E_j \cdot \mathbb{1}\left\{d_{b,j} = b\right\} \cdot \left(1 - \mathbb{1}\left\{\sum_{k=1}^{j} \mathbb{1}\{d_{b,k} = b\} \leqslant N_b \cdot \varepsilon + 1\right\}\right), \tag{6.19}$$

$$D_b = \sum_{j=1}^{n} E_j \cdot \mathbb{1}\left\{d_{b,j} = b\right\} \cdot \mathbb{1}\left\{\sum_{k=1}^{j} \mathbb{1}\{d_{b,k} = b\} \leqslant N_b \cdot \varepsilon + 1\right\} + \\ + \sum_{j=1}^{n} E_j \cdot \mathbb{1}\left\{d_{a,j} = a\right\} \cdot \left(1 - \mathbb{1}\left\{\sum_{k=1}^{j} \mathbb{1}\{d_{a,k} = a\} \leqslant N_a \cdot \varepsilon + 1\right\}\right), \tag{6.20}$$

gdzie ε przyjmuje taką samą wartość jak we wzorze 6.18, natomiast

$$N_a = \sum_{j=1}^{n} \mathbb{1}\left\{d_{a,j} = a\right\}, \; N_b = \sum_{j=1}^{n} \mathbb{1}\left\{d_{b,j} = b\right\},$$

są równe całkowitej liczbie zadań przypisanych odpowiednio do jednostki a oraz b.

Inaczej ujmując, proces krzyżowania polega na losowym doborze punktu krzyżowania wektorów decyzji (osobno dla jednostki a oraz osobno dla jednostki b) i wymiany

elementów wektorów. Warto zauważyć, że punkt krzyżowania nigdy nie będzie znajdować się przed pierwszym genem, w związku z czym pierwszy gen nie może podlegać wymianie. Podejście to pozwala na uniknięcie sytuacji, w której któraś z jednostek obliczeniowych nie będzie miała przypisanego żadnego zadania.

Aby zadania przypisane na początku harmonogramów mogły podlegać migracji do innych jednostek, w niniejszej pracy zaproponowano operację mutacji polegającą na zmianie kolejności występowania zadań w harmonogramach jednostek. Działanie tego operatora nie ma bezpośredniego pływu na wartość funkcji przystosowania, ale daje możliwość migracji zadań w późniejszych epokach. Operator mutacji działa analogicznie do operatora $\mathcal{T}$ przedstawionego we wzorze 6.13. Działanie operatora $\mathcal{T}$ można zapisać jako $\mathcal{T}_{\mathcal{P}}(D_i)$, gdzie D_i to wektor decyzji wybranej jednostki (por. 6.18), a $\mathcal{P}$ ($\mathcal{P} \in [0, 1]$) określa prawdopodobieństwo wystąpienia mutacji.

Punkty 1 i 2 algorytmu wykonywane są tak długo, jak zbiory $\mathcal{A}$ oraz $\mathcal{B}$ są zbiorami niepustymi. Po wylosowaniu ostatniej pary osobników dokonuje się wymiany genów (krzyżowania), a następnie przechodzi do ostatniego punktu algorytmu.

3. **Generowanie nowego harmonogramu**

   Na podstawie zmodyfikowanych wektorów decyzji D_i wyznaczany jest wektor reprezentujący nowy harmonogram dla całego pakietu zadań. Proces tworzenia harmonogramu na podstawie wektorów decyzji wszystkich jednostek obliczeniowych został sformułowany za pomocą równania 6.2.

Proces ten został przedstawiony w postaci algorytmu 2.

6.4. Model wieloagentowego systemu monitorującego

W niniejszym podrozdziale zaproponowano formalny model wieloagentowego systemu monitorującego. W rozdziale 4 przedstawiono kilka propozycji architektur agentowych, a także wskazano na brak jednoznacznych wymagań dotyczących definicji agenta (por. [30]). Autorzy wspomnianych wcześniej prac poświęconych systemom agentowym proponują albo

Algorytm 2 Algorytm generowania harmonogramów

```
1: function GenerujHarmonogram-PodejścieEwolucyjne
2:     i = 0
3:     Wygeneruj populację początkową S^0 reprezentującą harmonogram; S ← S^0;
4:     Dokonaj oceny rozwiązania za pomocą funkcji przystosowania (wzór 6.15);
5:     while i < I_max do
6:         Wygeneruj nowy harmonogram: S^{i+1} ← F^{i+1}(S^i);
7:         Dokonaj oceny rozwiązania za pomocą funkcji przystosowania (wzór 6.15);
8:         if nowy harmonogram jest lepszy then
9:             S ← S^{i+1};
10:        end if
11:        i ← i + 1;
12:    end while
13:    return Harmonogram S;
14: end function
```

definicje abstrakcyjne (charakteryzujące się wysoką elastycznością i niewielką szczegółowością), albo wpisujące się w problem, który został powierzony systemowi agentowemu. W niniejszej pracy, bazując na przeprowadzonych analizach modeli, zaproponowano model odpowiadający funkcjonalnościom systemu monitorującego.

G. Dobrowolski w [35] wskazuje na dekompozycję jako narzędzie analizy systemów agentowych. Proces ten pozwala na przejście od specyfikacji globalnego celu systemu do specyfikacji działań poszczególnych agentów. Szczególnie istotna z punktu widzenia niniejszej pracy jest dekompozycja funkcjonalna, która pozwala na zdefiniowanie poszczególnych akcji pojedynczych agentów, określenie zakresu współdziałania pomiędzy agentami, a także identyfikację spójnych celów, czyli złożonych funkcjonalności całego systemu agentowego.

Akcje agentów definiowane są zwykle w postaci zbiorów elementów (por. [25, 26, 35, 65, 125]) lub za pomocą zbioru par: warunku koniecznego zaistnienia akcji oraz akcji agenta (por. [24]).

Z punktu widzenia funkcjonalności modelowanego systemu, istotnymi elementami agentów oraz całego systemu są: typy agentów, lokalizacje agentów, strategie realizujące spójne cele, możliwe do wykonania akcje, stany środowiska oraz operatory pozwalające na postrzeganie oraz oddziaływanie na środowisko. Zgodnie z powyższym, można zaproponować

system wieloagentowy zapisany w następującej postaci:

$$MAS = \{AG, ID, TP, LOC, ES, ACT, ST\}, \tag{6.21}$$

gdzie:

AG – zbiór agentów należących do systemu wieloagentowego;

ID – zbiór unikalnych identyfikatorów agentów;

TP – zbiór wszystkich typów agentów;

LOC – zbiór lokalizacji, w których mogą przebywać agenty;

ES – zbiór wszystkich możliwych stanów środowiska;

ACT – zbiór akcji, które mogą być wykonywane przez agentów w ramach środowiska;

ST – zbiór strategii realizowanych przez agentów.

Podstawowym elementem systemu jest agent (ag), który składa się z następujących elementów:

$$AG \ni ag = \{id, tp, k, \gamma, \beta, \delta, \alpha\}, \tag{6.22}$$

gdzie:

$id \in ID$ – unikalny identyfikator agenta w skali całego systemu;

$tp \in TP$ – typ agenta;

$k \in K$ – aktualna wiedza agenta;

γ – funkcja obserwacji (percepcji) pozwalająca na monitorowanie stanu środowiska – na podstawie aktualnego stanu środowiska agent będzie w stanie budować swoją wiedzę:

$$\gamma : ES \rightarrow \mathcal{M}(K), \tag{6.23}$$

gdzie: K to zbiór reprezentujący wszelką możliwą wiedzę, jaką agenty są w stanie uzyskać, a $\mathcal{M}$ oznacza przestrzeń miar probabilistycznych nad zbiorem.

β – funkcja wyboru strategii, która na podstawie typu agenta i posiadanej wiedzy zwraca identyfikator strategii:

$$\beta : TP \times K \rightarrow \mathbf{Z}; \tag{6.24}$$

δ – funkcja decyzyjna, która na podstawie realizowanej strategii dokonuje wyboru akcji:

$$\delta : ST \rightarrow \mathcal{M}(ACT); \tag{6.25}$$

α – funkcja działania, która na podstawie wybranej akcji wykonuje ją na środowisku zmieniając jego stan:

$$\alpha : ACT \rightarrow \mathcal{M}(ES). \tag{6.26}$$

W ramach modelu zaproponowanego w niniejszej rozprawie wyszczególniono pięć typów agentów, $TP = \{Ag0, Ag1, Ag2, Ag3, Ag4\}$.

1. $Ag0$ – monitoruje proces generowania harmonogramu oparty o podejście ewolucyjne i ingeruje w proces selekcji i przeżywalności osobników (w systemie jest obecny tylko jeden agent typu $Ag0$).

2. $Ag1$ – informuje o aktualnym etapie wykonania zadań i gotowości jednostek obliczeniowych do przyjęcia nowych.

3. $Ag2$ – na podstawie informacji uzyskanych od agentów typu Ag1 generuje moment inicjalizacji procesu harmonogramowania zadań (w systemie jest obecny tylko jeden agent typu $Ag2$).

4. $Ag3$ – za pomocą sztucznej sieci neuronowej monitoruje terminowość wykonywania poszczególnych zadań przypisanych do jednostki obliczeniowej i dokonuje predykcji opóźnień.

5. $Ag4$ – konsultuje oczekiwane czasy wykonania zadań z agentami typu $Ag3$ i na podstawie wygenerowanych przez nich predykcji opóźnień generuje macierz poprawek (w systemie jest obecny tylko jeden agent typu $Ag4$).

Agenty mogą być umiejscowione w jednej z trzech lokalizacji, które podlegają moni-

toringowi[1] oraz na które agenty bezpośrednio oddziałują poprzez wykonywane akcje:

$$LOC = \{compunit, scheduler, envmgmt\}, \tag{6.27}$$

gdzie *compunit* reprezentuje jednostkę obliczeniową, *scheduler* – planistę, a *envmgmt* – obszar zarządzania rozproszonym środowiskiem (jest to miejsce, gdzie m.in. generowana jest macierz ETC oraz inicjalizowany jest proces harmonogramowania – por. rys. 2.1). Agent w zaproponowanym modelu może znajdować się w jednej lokalizacji przez cały okres działania systemu. Agent typu $Ag0$ znajduje się w obrębie planisty, agenty $Ag2$ i $Ag4$ w obrębie modułu zarządzania rozproszonym środowiskiem, natomiast agenty $Ag1$ i $Ag3$ działają przy każdej z monitorowanych jednostek obliczeniowych.

Strategie zależne są od typu agenta. Określają sposób funkcjonowania agenta w środowisku i sposób realizacji celów. Strategie przyjmują postać zbioru:

$$ST = \{st_0, st_1, st_2, st_3, st_4, st_5\}. \tag{6.28}$$

Zbiór ten można zapisać w postaci sumy podzbiorów strategii dedykowanych agentom określonego typu:

$$ST = ST_{Ag0} \cup \ldots \cup ST_{Ag4}. \tag{6.29}$$

W związku z powyższym, strategie mogą zostać pogrupowane względem typów agentów:

$Ag0$:

st_0 – ograniczanie liczby par osobników poddawanych reprodukcji na podstawie przyjętego parametru Φ;

$Ag1$:

st_1 – monitoring aktualnego stanu wykonania zadań wraz z przekazywaniem uzyskanych informacji agentowi typu $Ag2$;

$Ag2$:

st_2 – generacja nowego momentu inicjalizacji procesu harmonogramowania zadań na pod-

[1] Agenty mogą również znajdować się w innych lokalizacjach, które umożliwiają im realizację postawionych celów. W tej pracy założono, że agenty znajdują się bezpośrednio w lokalizacjach podlegających monitoringowi.

stawie przyjętych parametrów θ_1 oraz θ_2;

$Ag3$:

st_3 – monitoring terminowości wykonania poszczególnych zadań wraz z przekazywaniem uzyskanych informacji agentowi typu $Ag4$ oraz uczenie sztucznej sieci neuronowej predykcji opóźnień;

st_4 – predykcja opóźnień dla konsultowanych zadań;

$Ag4$:

st_5 – konsultacja czasów wykonania zadań i budowa macierzy poprawek.

W zaproponowanym modelu akcje agentów są powiązane ze strategią realizowaną przez agenta. Zbiór akcji przedstawia się następująco:

$$ACT = \{getimpr, incrsel, getstage, informstage, gentick, \\ startsched, getdel, trainpred, predict, consult, gencorr\}. \quad (6.30)$$

Akcje te można pogrupować pod względem realizowanych strategii i zapisać w formie sumy podzbiorów akcji wykonywanych w ramach danej strategii:

$$ACT = ACT_{st_0} \cup \ldots \cup ACT_{st_5} \quad (6.31)$$

st_0:

- $getimpr$ – w ramach tej akcji agent wykorzystuje funkcję obserwacji do monitoringu stanu środowiska i budowy swojej wiedzy o jakości generowanych harmonogramów;
- $incrsel$ – zwiększa o 1 liczbę par osobników, które zostają przeniesione do następnej epoki i nie biorą udziału w procesie reprodukcji;

st_1:

- $getstage$ – w ramach tej akcji agent wykorzystuje funkcję obserwacji do monitoringu stanu środowiska i budowy swojej wiedzy o etapie wykonania zadań;
- $informstage$ – informuje agenta typu $Ag2$ o aktualnym stanie wykonania zadań;

st_2:

- $gentick$ – na podstawie informacji uzyskanych od agentów typu $Ag1$ agent generuje mo-

ment czasowy, w którym rozpocznie się proces harmonogramowania;

- *startsched* – agent inicjuje proces harmonogramowania w wygenerowanym wcześniej momencie czasowym;

st_3:

- *getdel* – w ramach tej akcji agent wykorzystuje funkcję obserwacji do monitoringu stanu środowiska i budowy swojej wiedzy o opóźnieniach generowanych przez poszczególne zadania;

- *trainpred* – agent na podstawie swoich doświadczeń (wiedzy) uczy sieć neuronową predykcji opóźnień;

st_4:

- *predict* – agent dokonuje predykcji opóźnień na podstawie przesłanych przez agenta typu $Ag4$ informacji o zadaniach;

st_5:

- *consult* – agent przesyła do agentów typu Ag3 w celu konsultacji charakterystyki zadań dla poszczególnych jednostek obliczeniowych celem uzyskania informacji o możliwych opóźnieniach w ich realizacji;

- *gencorr* – agent na podstawie predykcji opóźnień otrzymanych od agentów typu $Ag3$ generuje macierz poprawek.

Działanie każdego z agentów rozpoczyna się wraz z momentem jego stworzenia w systemie i trwa tak długo, jak działa środowisko. W dalszej części przedstawiono szczegółowy sposób działania systemu agentowego w ramach rozproszonego środowiska obliczeniowego.

6.5. Sterowanie harmonogramowaniem przez system monitorujący

6.5.1. Model oparty o agentowe wsparcie ewolucyjnego procesu generowania harmonogramów

Głównym zadaniem agenta Ag0 jest nadzorowanie i bezpośrednie wsparcie procesu ewolucyjnego generowania harmonogramu. Agent monitoruje proces budowy harmonogramu

i może podjąć odpowiednie działania, jeśli uzna, że w wyniku procesu ewolucji otrzymywane rozwiązania nie spełniają oczekiwanych norm jakościowych w zadanym czasie. W takim przypadku agent ingeruje w proces generowania harmonogramu poprzez ograniczenie liczby osobników (jednostek obliczeniowych) przeznaczonych do reprodukcji. Działanie to może być utożsamiane z realizacją strategii przeżywalności osobników w algorytmie ewolucyjnym i jednocześnie oddziaływaniem na proces doboru osobników do reprodukcji (selekcji).

Z punktu widzenia przetwarzania pakietowego zadań, najmniej pożądanymi harmonogramami mogą być te, które charakteryzują się wysoką różnicą pomiędzy najdłuższym i najkrótszym czasem potrzebnym do ukończenia wszystkich przypisanych zadań do jednostek obliczeniowych. Innymi słowy, jeśli to możliwe, harmonogram powinien gwarantować również odpowiednie rozłożenie pracy pomiędzy jednostkami obliczeniowymi (ang. *load balancing*), a więc minimalizować różnicę $v_m^{sort} - v_1^{sort}$ (por. 6.16). Takie podejście (przy przyjętych w rozdziale 6.2 ograniczeniach) będzie wspierać generowanie harmonogramów charakteryzujących się stosunkowo korzystnym czasem ich realizacji.

Działanie agenta Ag0 sprowadza się do ingerencji w proces formowania zbiorów $\mathcal{A}$ i $\mathcal{B}$. Agent wyklucza po jednej jednostce obliczeniowej ze zbiorów $\mathcal{A}$ oraz $\mathcal{B}$. W przypadku zbioru $\mathcal{A}$ jest to jednostka cechująca się najwyższą wartością v, a w przypadku $\mathcal{B}$ – odpowiednio – najniższą. Jeśli agent, po dokonaniu obserwacji i oceny (po pewnym czasie) uzna, że w wyniku procesu ewolucji nie są generowane lepsze (tzn. cechujące się lepszą wartością przystosowania) rozwiązania, ponawia procedurę wykluczenia osobników poddawanych operacji krzyżowania.

Monitoring procesu harmonogramowania przez agenta można w sposób formalny przedstawić poprzez modyfikację procedur tworzenia zbiorów $\mathcal{A}$ i $\mathcal{B}$ w ramach działania operatora $\mathcal{F}$. Dla odróżnienia, niech $\mathcal{F}_\kappa$ będzie operatorem opisującym proces harmonogramowania wspierany przez system agentowy, a $\kappa \in \mathbf{Z}^+, \kappa \leqslant \frac{m}{2}$ (początkowo $\kappa = 0$) będzie liczbą określającą, ile razy agent wykonał procedurę wykluczenia osobników (w ramach jednej procedury agent wyklucza dwoje osobników). Dwa rozłączne zbiory $\mathcal{A}$ i $\mathcal{B}$ mogą być skonstruowane w następujący sposób:

$$\mathcal{A} = \left\{ i : v_i < v^{sort}_{\frac{m}{2}+1-\kappa}, 1 \leqslant i \leqslant m \right\}, \ \mathcal{B} = \left\{ i : v_i > v^{sort}_{\frac{m}{2}+\kappa}, 1 \leqslant i \leqslant m \right\}. \tag{6.32}$$

Proces wyboru par do reprodukcji można natomiast sformułować w następujący sposób:

$$a = \sum_{j=1}^{\frac{m}{2}-\kappa} a_j \cdot \mathbb{1} \left\{ \frac{j-1}{\frac{m}{2}-\kappa} \leqslant \varepsilon < \frac{j}{\frac{m}{2}-\kappa} \right\}, \quad b = \sum_{j=1}^{\frac{m}{2}-\kappa} b_j \cdot \mathbb{1} \left\{ \frac{j-1}{\frac{m}{2}-\kappa} \leqslant \varepsilon < \frac{j}{\frac{m}{2}-\kappa} \right\}, \tag{6.33}$$

gdzie a i b reprezentują jednostki wylosowane do procesu reprodukcji, a ε jest zmienną losową o rozkładzie równomiernym z przedziału $[0, 1)$.

Warto zauważyć, że poprzez eliminację wszystkich par osobników (jednostek obliczeniowych) z procesu ewolucji, system agentowy ma możliwość zakończenia procesu generowania harmonogramu przed osiągnięciem wymaganej liczby epok I_{max}. Sytuacja ta zależy od częstości generowania lepszego harmonogramu oraz parametru określającego, po ilu epokach nieprzynoszących lepszego rozwiązania należy zmniejszyć liczbę par dopuszczonych do reprodukcji (zwiększyć przeżywalność osobników ze starej populacji). Parametr ten oznaczmy jako Φ ($\Phi \in (0, 100]$), gdzie Φ jest wartością procentową, która po przemnożeniu przez maksymalną liczbę iteracji podzieloną przez liczbę par osobników określa próg dla zmiany (inkrementacji) wartości parametru κ.

Proces harmonogramowania uwzględniający ingerencję ze strony systemu agentowego został przedstawiony w ramach algorytmu 3.

6.5.2. Model oparty o monitoring stanu jednostek obliczeniowych

Model oparty o monitoring stanu jednostek obliczeniowych polega na realizacji przez agentów odpowiedniej strategii generowania momentów czasowych t_k ($k \geqslant 1$), które inicjują procesy harmonogramowania. Każda z jednostek obliczeniowych monitorowana jest przez agenta typu Ag1 w celu uzyskania informacji o dostępności jednostki lub aktualnym stanie realizacji harmonogramu. Agent inicjujący proces harmonogramowania (typu Ag2) przeprowadza swojego rodzaju referendum wśród agentów monitorujących jednostki (typu Ag1). Agenty jednostek formułują informację zwrotną, określającą czas potrzebny do ukończe-

Algorytm 3 Algorytm generowania harmonogramów wspomagany agentowo

function GenerujHarmonogram-PodejścieEwolucyjne
  $\kappa = 0$, $i = 0$
  Wygeneruj populację początkową S^0 reprezentującą harmonogram; $S \leftarrow S^0$;
  Dokonaj oceny rozwiązania za pomocą funkcji przystosowania (wzór 6.15);
  while $i < I_{max}$ lub $\kappa < m/2$ **do**
    Wygeneruj nowy harmonogram: $S^{i+1} \leftarrow \mathcal{F}_{\kappa}^{i+1}(S^i)$;
    Dokonaj oceny rozwiązania za pomocą funkcji przystosowania;
    if nowy harmonogram jest lepszy **then**
      $S \leftarrow S^{i+1}$;
    end if
    if brak poprawy rozwiązania przez $\lfloor \Phi \cdot I_{max} \rfloor$ epok **then**
      $\kappa \leftarrow \kappa + 1$
    end if
    $i \leftarrow i + 1$;
  end while
  return Harmonogram S;
end function

nia wykonywania zadań lub informację o dostępności jednostki obliczeniowej. Na podstawie udzielonych informacji agent Ag2 wyznacza termin rozpoczęcia kolejnego procesu harmonogramowania. Jako wiarygodne źródło informacji o stanie realizacji harmonogramu przez konkretne jednostki obliczeniowe uważa się monitorujące je agenty typu Ag1.

Poniższe rozważania dotyczą sekwencji generowania harmonogramów, stąd indeks k ponownie identyfikuje przetwarzany pakiet zadań oraz rozpatrywany moment czasowy. Czas potrzebny do komunikacji pomiędzy agentami został uznany za nieistotny i pominięty.

Oznaczmy wektor uporządkowanych zaległości w wykonywaniu zadań w momencie czasowym t_k przez V_k^{sort} (por. 6.16). Strategia generowania momentów czasowych przez system agentowy opiera się na dwóch parametrach:

(i) θ_1, $\theta_1 \in \{1, 2, \ldots, m\}$ – określa liczbę jednostek obliczeniowych, które muszą być w stanie bezczynności[2], aby wygenerować nowy moment czasowy;

(ii) θ_2, $\theta_2 \geqslant 0$ – określa wartość graniczną zaległości w wykonywaniu zadań, która pozwoli na włączenie jednostek obliczeniowych do zbioru jednostek, które będą brały

[2]Stan bezczynności rozumiany jest jako brak aktualnie przypisanych zadań obliczeniowych do realizacji. W modelu tym zakłada się, że zadania realizowane są zgodnie z oszacowaniem pochodzącym z macierzy ETC, jednak w rzeczywistym środowisku nie ma przeciwwskazań, aby agenty typu Ag1 miały wpływ na wartości v_i.

udział w procesie harmonogramowania i przydziału zadań z nowego pakietu (innymi słowy, parametr θ_2 określa maksymalny czas oczekiwania na jednostki, które wkrótce zrealizują wszystkie aktualnie przypisane zadania).

System agentowy generuje nowy moment t_k w oparciu o następującą relację:

$$t_k = t_{k-1} + \max\Big\{v^{sort}_{\theta_1,k-1}, v^{sort}_{\theta_1+1,k-1}\mathbb{1}\{v^{sort}_{\theta_1+1,k-1} - v^{sort}_{\theta_1,k-1} \leqslant \theta_2\}, \ldots, v^{sort}_{m,k-1}\mathbb{1}\{v^{sort}_{m,k-1} - v^{sort}_{\theta_1,k-1} \leqslant \theta_2\}\Big\}, \; k \geqslant 1. \quad (6.34)$$

Zgodnie z realizowaną strategią, dokładnie (a przy założeniu, że jednostki obliczeniowe mogą ukończyć swoją pracę w tym samym momencie – co najmniej) θ_k elementów wektora V^{sort}_k jest równych 0, natomiast $\sum_{j=\theta_1+1}^{m}\mathbb{1}\{v^{sort}_{j,k-1} - v^{sort}_{\theta_1,k-1} \leqslant \theta_2\}$ komponentów mieści się w zakresie $(0, \theta_2 + v^{sort}_{\theta_1,k-1}]$. Zatem przez $\mathcal{I}_k$ oznaczmy zbiór jednostek obliczeniowych, których zaległości w wykonywaniu zadań nie są większe niż $\theta_2 + v^{sort}_{\theta_1,k-1}$ w momencie t_k. Zbiór ten można zdefiniować następująco:

$$\mathcal{I}_k = \{i : v_{ki} \leqslant \theta_2 + v^{sort}_{\theta_1,k-1}, \; 1 \leqslant i \leqslant m\} \quad (6.35)$$

Liczba elementów zbioru będzie równa $(\sum_{j=1}^{m}\mathbb{1}\{v^{sort}_{j,0} - v^{sort}_{\theta_1,0} \leqslant \theta_2\}) \in \mathbf{2N}$. Liczebność zbioru musi być parzysta – w przeciwnym razie ze zbioru usuwany jest element dodany jako ostatni.

W nowo wygenerowanym momencie czasowym t_k, w procesie harmonogramowania udział wezmą tylko te jednostki obliczeniowe, które znajdują się w zbiorze $\mathcal{I}_k$ (wszystkie one będą w momencie t_k bezczynne). Oznacza to, że nie wszystkie jednostki obliczeniowe otrzymają zadania z nowego, k-tego pakietu, a więc wektory decyzji $D_{i,k}$ będą zdefiniowane tylko dla $i \in \mathcal{I}_k$. Pozostałe $m - \sum_{j=1}^{m}\mathbb{1}\{v^{sort}_{j,0} - v^{sort}_{\theta_1,0} \leqslant \theta_2\}$ wektorów decyzyjnych, będą wektorami zerowymi. W związku z powyższym, wektory decyzji dla poszczególnych jednostek przyjmą postać:

$$D_{i,k} = \begin{cases} \sum_{j=1}^{n_k} E_j \cdot \mathbb{1}\{s_{j,k} = i\}, & i \in \mathcal{I}_k, \\ \mathbf{0}, & i \notin \mathcal{I}_k, \end{cases} \qquad 1 \leqslant i \leqslant m. \quad (6.36)$$

Wektor reprezentujący harmonogram wykonania wszystkich zadań z pakietu k można wyznaczyć zgodnie ze wzorem 6.2. Pozostałe formuły opisujące dynamikę systemu w czasie również nie ulegną zmianie.

6.5.3. Model oparty o monitoring generowanych opóźnień

W modelu tym zakłada się nieterminową realizację harmonogramu przez poszczególne jednostki obliczeniowe – to znaczy, że czasy wykonania poszczególnych zadań różnią się od tych, które są oczekiwane (por. 5.12). Rozwiązanie to opiera się na działaniu dedykowanej dla każdej z jednostek obliczeniowych niewielkiej sztucznej sieci neuronowej (SSN), której zadaniem jest predykcja możliwych opóźnień generowanych w trakcie wykonywania zadań. W rzeczywistych środowiskach obliczeniowych czasy wykonania różnych zadań obliczeniowych – pomimo podobnego, a nawet takiego samego zapotrzebowania obliczeniowego zadania i takiej samej zdolności obliczeniowej jednostki – mogą się znacząco różnić. Zjawisko takie jest jeszcze bardziej powszechne w środowiskach opartych na wirtualizacji. Ponadto przedstawione w niniejszej pracy charakterystyki jednostek obliczeniowych pomijają cechy takie jak pojemność i szybkość pamięci operacyjnej, szybkość zapisu i odczytu z nośników danych, istnienie pamięci kieszeniowych i wiele innych, które mają znaczący wpływ na zdolności obliczeniowe. Próba uwzględnienia tego typu czynników może znacznie zwiększyć złożoność problemu, a nawet uniemożliwić wystarczająco wierne scharakteryzowanie zadania i jednostki. W celu zminimalizowania ewentualnych opóźnień wynikających z niedoskonałości modeli, zaproponowano sztuczną sieć neuronową, która na podstawie wektora wejściowego, składającego się z charakterystyki zadania dokonuje predykcji opóźnienia jego wykonania.

Powyższe funkcjonalności realizowane są przez dwa typy agentów: Ag3 i Ag4. Każda z jednostek obliczeniowych monitorowana jest przez agenta typu Ag3. Agent ten wyposażony jest w zdolność kontroli czasów wykonania poszczególnych zadań. Na podstawie oszacowanych czasów wykonań zadań, wyznacza on ewentualne opóźnienie dla każdego z aktualnie przypisanych zadań. Ponadto agent ten wyposażony jest w sztuczną sieć neuronową, która na podstawie charakterystyk zadań dokonuje predykcji opóźnień. Każdy z agentów typu Ag3 używa oddzielnej sieci, w związku z czym w całym środowisku znajduje się tyle sieci,

ile jednostek obliczeniowych.

Drugi typ agenta, Ag4, zlokalizowany jest w obrębie modułu zarządzania środowiskiem rozproszonym. Jego zadaniem jest konsultacja oczekiwanych czasów wykonania z agentami typu Ag3. Agent Ag4 przesyła do każdego z agentów Ag3 wektor charakterystyk wszystkich zadań (odbywa się to w kolejnym momencie czasowym t_k), te natomiast odsyłają predykcję opóźnień dla każdego z zadań. W ten sposób uzyskane dane formułowane są w macierz korekcji czasów TC (ang. *time correction matrix*) o wymiarach $n \times m$, będącą odpowiednikiem macierzy ETC.

Każdy agent typu Ag3 dokonuje predykcji na podstawie sztucznej sieci neuronowej uczonej przez tylko te charakterystyki i opóźnienia, które dotyczą zadań przesłanych do danej jednostki obliczeniowej. Rozwiązanie zostało oparte o architekturę perceptronu wielowarstwowego zbudowanego z neuronów McCullocha-Pittsa. SSN, którą posługują się agenty typu Ag3, uczy się przewidywać opóźnienia na podstawie danych z monitoringu odpowiadającej jej jednostki obliczeniowej. Następnie dokonywana jest predykcja opóźnień dla każdego zadania z pakietu. SNN rozwiązuje więc problem ekstrapolacji [51].

Modele zaproponowane w niniejszej pracy bazują na profilu zadania i jednostki, który uwzględnia możliwości przetwarzania równoległego (por. podrozdział 5.4.2). Sieci neuronowe uczone są w oparciu o wartości zapotrzebowania obliczeniowego zadania. W niniejszym modelu założono *a priori*, że opóźnienia rozkładają się w takim samym stopniu podczas wykonywania operacji sekwencyjnych, jak i równoległych. Dlatego pod pojęciem zapotrzebowania obliczeniowego zadania j realizowanego na jednostce obliczeniowej i należy rozumieć sumę liczby operacji wykonywanych sekwencyjnie z liczbą operacji wykonywanych równolegle podzieloną przez liczbę rdzeni monitorowanej jednostki. Zapotrzebowanie to można zapisać jako:

$$wl_{i,j}^{ann} = wl_j^s + \frac{wl_j^p}{cn_i}, \tag{6.37}$$

gdzie wl_j^s to liczba operacji zmiennoprzecinkowych, które muszą być wykonane w sposób sekwencyjny, wl_j^p to liczba operacji zmiennoprzecinkowych, które mogą zostać zrównoleglone, a cn_i to liczba rdzeni jednostki obliczeniowej.

Zamodelowana sieć jest siecią jednokierunkową, wielowarstwową, będącą modelem regresyjnym, uczoną algorytmem Levenberga-Marquardta. Uczenie przeprowadzane jest za pomocą metod nadzorowanych, tzn. poprzez zbiory uczące. W celu zbudowania odpowiedniego zbioru, zapewniającego wystarczającą ilość informacji do przeprowadzenia procesu optymalizacji wag sieci, zebrano dane w ramach monitoringu wcześniej realizowanych harmonogramów. Należy zauważyć, że liczebność zbioru uczącego powiększa się z każdym wykonanym zadaniem. W związku z powyższym, początkowo sieć uczona jest w oparciu o zebrane dane archiwalne, a następnie douczana po każdym wykonaniu zadania, kiedy to agent uzyskuje kolejny wzorzec uczący.

Cały proces można zapisać w sposób formalny – przedstawione rozważania na temat budowy i uczenia sieci dotyczą pojedynczego modułu monitorującego jednostkę. Niech $\mathcal{L}$ oznacza liczbę warstw SSN (bez warstwy wejściowej). W zakresie każdej ℓ-tej ($1 \leqslant \ell \leqslant \mathcal{L}$) warstwy znajduje się N_ℓ neuronów oznaczonych jako X_1^ℓ, X_2^ℓ, …, $X_{N_\ell}^\ell$, z których każdy wyposażony jest w funkcję aktywacji f^ℓ – wspólną dla wszystkich neuronów w warstwie. Neurony te otrzymują sygnał z neuronów warstwy poprzedniej, tj. o numerze $\ell - 1$. Przykładowo, neuron X_s^ℓ otrzymuje sygnał z każdego neuronu warstwy $\ell - 1$. Wagę tego sygnału oznaczamy jako $w_{r,s}^\ell$, gdzie r oznacza numer neuronu z warstwy poprzedniej, a s numer neuronu docelowego. Dla każdej warstwy otrzymujemy macierz o wymiarach $N_{\ell-1} \times N_\ell$, której elementami są wszystkie wagi ℓ-tej warstwy:

$$\mathbf{W}^\ell = [w_{r,s}^\ell]_{s=1,\ldots,N_\ell}^{r=1,\ldots,N_{\ell-1}}. \tag{6.38}$$

Ponadto każdy neuron X_s^ℓ posiada tzw. bias b_s^ℓ o funkcji aktywacji a_s^ℓ. Biasy dla każej warstwy można zapisać w postaci N_ℓ-elementowego wektora:

$$\mathbf{b}^\ell = [b_s^\ell]_{s=1,\ldots,N_\ell}. \tag{6.39}$$

Oznaczmy przez n_s^ℓ sygnał docierający do neuronu X_s^ℓ. Ma on postać ważonej sumy

sygnałów pochodzących z neuronów poprzedniej warstwy oraz biasu:

$$n_s^\ell = \sum_{r=1}^{N_{\ell-1}} a_r^{\ell-1} w_{r,s}^\ell + b_s^\ell, \tag{6.40}$$

gdzie $1 \leqslant s \leqslant N_\ell$. Aktywacja neuronu X_s^ℓ przyjmuje wtedy postać:

$$a_s^\ell = f^\ell(n_s^\ell) = f^\ell(\sum_{r=1}^{N_{\ell-1}} a_r^{\ell-1} w_{r,s}^\ell + b_s^\ell), \tag{6.41}$$

czyli stanowi sygnał wejściowy przekształcony przez funkcję aktywacji.

Warstwę wejściową, odbierającą sygnały wejściowe, możemy oznaczyć jako warstwę o numerze 0. Niech $\mathbf{x}$ będzie wektorem wejścia do sieci. Zgodnie z przyjętymi wcześniej oznaczeniami, będzie to wektor zbudowany z N_0 elementów. Funkcja aktywacji w warstwie wejściowej przyjmie postać: $a_r^{(0)} = x_r, 1 \leqslant r \leqslant N_0$. Ostatnia warstwa sieci (o numerze $\mathcal{L}$) jest warstwą wyjściową.

Po poprawnym nauczeniu sieci, generuje ona informację, która stanowić będzie wniosek z absorpcji wiedzy z całego zbioru danych uczących. Niech $\mathbf{y}$ reprezentuje wektor wyjściowy sieci o $N_\mathcal{L}$ elementach. Elementy wektora wyjściowego przyjmują postać:

$$y_s = a_s^\mathcal{L}, \tag{6.42}$$

gdzie $1 \leqslant s \leqslant N_\mathcal{L}$.

Sygnał wejściowy, przechodząc przez kolejne warstwy, aktywuje ich neurony, aż do warstwy wyjściowej. Dzięki temu odpowiedź sieci na każdy z wektorów uczących zależy od wartości wszystkich wag sieci.

W zaproponowanym modelu użyto sieci o dwóch warstwach ukrytych. Jest to najmniejsza liczba warstw, dla której SSN jest tzw. uniwersalnym aproksymatorem [21]. Oznacza to, że tak zbudowana sieć umożliwia aproksymację dowolnej funkcji ciągłej określonej na zbiorze domkniętym z zadaną dokładnością.

Wektor wejściowy dla sieci zarządzanej przez agenta przyjmuje postać zapotrzebowa-

nia obliczeniowego zadania j realizowanego w ramach jednostki obliczeniowej i:

$$\mathbf{x} = wl_{i,j}^{ann}. \tag{6.43}$$

Natomiast wektor wyjściowy tej sieci ma postać predykcji wartości opóźnienia (oznaczanego przez o), jakie mogłoby powstać podczas wykonywania zadania j:

$$\mathbf{y} = o_j. \tag{6.44}$$

Przy tak przyjętych wektorach wejścia i wyjścia: $N_0 = N_{\mathcal{L}} = 1$.

Zbiór uczący Λ przyjmuje postać par:

$$\Lambda = \{(\mathbf{x}_l, \mathbf{y}_l)\}, \tag{6.45}$$

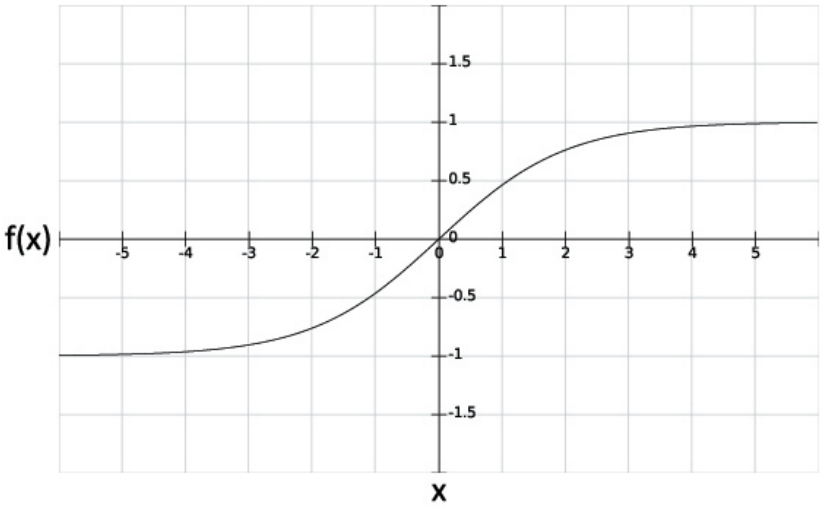
gdzie $1 \leqslant l \leqslant L$, a L to liczba elementów zbioru uczącego Λ. Zbiór ten składa się z zapotrzebowań obliczeniowych $wl_{i,j}^{ann}$ wszystkich zadań j wykonanych na jednostce obliczeniowej i oraz odpowiadających im opóźnieniom w ich wykonaniu. Po przekazaniu sieci każdego zadania następuje korekta wag. Liczba podanych zadań zmienia się w czasie i zależy od realizowanych harmonogramów poszczególnych pakietów zadań. Przez cykl uczenia należy rozumieć podanie sieci wzorów uczących związanych ze wszystkimi zadaniami danego pakietu przypisanymi do jednostki obliczeniowej.

Funkcja aktywacji dla wszystkich neuronów w warstwie pierwszej ukrytej została przyjęta w postaci sigmoidalnego tangensa hiperbolicznego (ang. *hyperbolic tangent sigmoid function*) [81] – rys. 6.2:

$$f_{\text{tansig}}(x) = \frac{1 - e^{-x}}{1 + e^{-x}} = \tanh(x/2), \tag{6.46}$$

natomiast funkcja aktywacji w warstwie wyjściowej przyjmuje postać funkcji liniowej.

Jakość odwzorowania zbioru uczącego oraz testującego sprawdzano wyznaczając współczynnik korelacji pomiędzy pożądanymi wartościami wyjścia a wartościami uzyskanymi w pro-



Rysunek 6.2: Funkcja aktywacji neuronów w pierwszej warstwie ukrytej.

cesie predykcji:

$$R^2 = \frac{\sum_{p=1}^{n}(y_p - \bar{y})(\hat{y_p} - \bar{\hat{y}})}{\sqrt{\sum_{p=1}^{n}(y_p - \bar{y})^2}\sqrt{\sum_{p=1}^{n}(\hat{y_p} - \bar{\hat{y}})^2}}, \tag{6.47}$$

gdzie:

- $\hat{y}$ – rzeczywista wartość opóźnienia,
- y – predykcja opóźnienia wygenerowana przez sieć,
- $\bar{y}$ – średnia arytmetyczna zbioru liczb $y_1, \ldots, y_n$,
- $\bar{\hat{y}}$ – średnia arytmetyczna zbioru liczb $\hat{y_1}, \ldots, \hat{y_n}$,
- n – liczba elementów zbioru uczącego/testującego.

Drugim sprawdzanym parametrem określającym jakość uczenia oraz testowania był błąd średniokwadratowy (ang. *mean squared error*) obliczony dla całego zbioru danych:

$$MSE = \frac{1}{L+T+V}\sum_{p=1}^{L+T+V}(y_p - \hat{y_p})^2, \tag{6.48}$$

gdzie: L – liczba elementów zbioru uczącego, T – liczba elementów zbioru testowego, V – liczba elementów zbioru walidującego.

Jeśli błąd średniokwadratowy został wyznaczony tylko dla zbioru danych uczących, mówimy o błędzie średniokwadratowym uczenia (ang. *mean squared error learning*):

$$MSEL = \frac{1}{L}\sum_{p=1}^{L}(y_p - \hat{y_p})^2, \tag{6.49}$$

a jeśli tylko i wyłącznie dla zbioru testującego – o błędzie średniokwadratowym testowania (ang. *mean squared error testing*):

$$MSET = \frac{1}{T}\sum_{p=1}^{T}(y_p - \hat{y_p})^2. \tag{6.50}$$

Dodatkowo, pewien procent zbioru danych został wyłączony z procesu poprawek wag, stając się zbiorem walidującym zdolności sieci do generalizacji. Błąd średniokwadratowy dla zbioru walidacji (ang. *mean squared error validation*) obliczamy analogicznie:

$$MSEV = \frac{1}{V}\sum_{p=1}^{V}(y_p - \hat{y_p})^2. \tag{6.51}$$

Uczenie sieci opiera się na algorytmie Levenberga-Marquardta [51, 91]. Taka metoda uczenia zakłada asocjację informacji pochodzącej z kolejnych wzorców uczących, propagowaną w postaci poprawek wartości wag sieci neuronowych. Zbiór par uczących jest prezentowany sieciom po jednym elemencie. Każdorazowo liczona jest różnica pomiędzy pożądaną odpowiedzią na dany wektor wejściowy, a odpowiedzią sieci na ten wektor – czyli błąd E dla jednej pary danych uczących:

$$E(x_l, y_l, \mathbf{W}^1, \ldots, \mathbf{W}^{\mathcal{L}}, \hat{y}_l) = ||y_l - \hat{y}_l||^2. \tag{6.52}$$

Wielkość błędu wyznacza wartość, o jaką zmieniane są wagi sieci, po prezentacji wzorca uczącego, który generuje wyjście $\mathbf{y}$. Błąd ten zależy od wszystkich wag i biasów sieci neuronowej. Proces uczenia sieci polega na zastosowaniu wybranej metody numerycznej minimalizującej $MSEL$, przy jednoczesnej obserwacji $MSEV$. Uczenie jest przerywane, gdy osiągnięto zadowalającą wielkość błędu $MSEL$.

Kolejne wartości wag oraz biasów sieci wyznacza się korzystając z metody najwięk-

szego spadku:

$$w_{r,s}^{\ell}(\beta+1) = w_{r,s}^{\ell}(\beta) - \alpha \frac{\partial E}{\partial w_{r,s}^{\ell}(\beta)}, \tag{6.53}$$

$$b_{s}^{\ell}(\beta+1) = b_{s}^{\ell}(\beta) - \alpha \frac{\partial E}{\partial b_{s}^{\ell}(\beta)}, \tag{6.54}$$

gdzie $\alpha(\alpha > 0)$ jest tzw. współczynnikiem uczenia, a β jest numerem powtórzenia kolejnej procedury uczenia.

Aby rozpocząć proces uczenia najpierw ustalamy topologię sieci, tzn. liczbę warstw ukrytych (tutaj dwie), oraz liczbę neuronów w warstwach. Następnie, po ustaleniu początkowej wartości współczynnika uczenia α, inicjalizujemy rzeczywistymi wartościami losowymi wartości wag i biasów sieci. Następnie powtarzamy kroki 1-5 do otrzymania odpowiednich wartości błędu $MSEL$, wartości współczynnika R .

1. Wybieramy losowy wzorzec uczący x_l $(1 \leqslant l \leqslant L)$ ze zbioru uczącego Λ.
2. Wyznaczamy $y(x_l)$ – wyjście z sieci, które jest odpowiedzią sieci na wylosowany wzorzec uczący x_l.
3. Sprawdzamy pożądaną odpowiedź $\hat{y}_l$ na wzorzec uczący.
4. Wyznaczamy błąd $E(x_l, y(x_l), \mathbf{W}^1(\beta), \ldots, \mathbf{W}^{\mathcal{L}}(\beta), \hat{y}_l)$ dla wzorca uczącego.
5. Wyznaczamy nowe wagi: $\mathbf{W}^1(\beta+1), \ldots, \mathbf{W}^{\mathcal{L}}(\beta+1)$ oraz biasy: $\mathbf{b}^1(\beta+1), \ldots, \mathbf{b}^{\mathcal{L}}(\beta+1)$.
6. Wyznaczamy wartości błędów $MSEL$ i $MSEV$ oraz wartość współczynnika R^2.

Po jednokrotnej prezentacji wszystkich wzorców uczących, czyli przeprowadzeniu powyższej procedury dla $l = 1, \ldots, L$ (1 epoka uczenia), wzorce te prezentujemy ponownie, przechodząc do kolejnej epoki uczenia. Jeśli uzyskujemy niezadowalające wyniki, dokonujemy zmiany liczby neuronów w warstwach ukrytych.

Poprawnie nauczona sieć neuronowa dokonuje predykcji opóźnień generowanych przez monitorowaną jednostkę na podstawie charakterystyk zadania. Informacje te wykorzystywane są w procesie budowy macierzy poprawek TC. Jest to macierz o wymiarach $n \times m$, gdzie n to liczba zadań, a m to liczba jednostek obliczeniowych. Agent typu Ag4 tuż przed

rozpoczęciem procesu harmonogramowania, kiedy wektor charakterystyk zadań jest już uformowany, przesyła go do każDego agenta typu Ag3, który monitoruje jednostkę, która ma brać udział w procesie przydziału nowych zadań. Każdy z agentów Ag3 wyznacza zapotrzebowanie obliczeniowe dla każdego zadania zgodnie ze wzorem 6.37. Następnie, korzystając z gotowego modelu regresyjnego – sztucznej sieci neuronowej, dokonuje predykcji czasów opóźnień. Predykcje te formułowane są w wektor, który przesyłany jest do agenta typu Ag4. Agent Ag4 z otrzymanych wektorów konstruuje macierz TC, której elementy są postaci:

$$tc_{i,j} = y_i(wl_{i,j}^{ann}), \tag{6.55}$$

gdzie $y_i(wl_{i,j}^{ann})$ stanowi odpowiedź sieci monitorującej jednostkę obliczeniową i na dane wejściowe w postaci zapotrzebowania obliczeniowego $wl_{i,j}^{ann}$.

Tak przygotowana macierz służy do korekty wartości macierzy ETC, która wykorzystywana jest w celu oszacowania czasu realizacji harmonogramów. Uwzględniając korekty oczekiwanych czasów wykonania zadań, modyfikacji ulegnie wzór 6.8, który wyznacza czas, jaki należy poświęcić na wykonanie wszystkich zadań przypisanych do jednostki i (we wzorze pominięto oznaczenie pakietu zadań):

$$ct_i = \frac{D_i \cdot (WL^s)^T}{cc_i^c} + \frac{D_i \cdot (WL^p)^T}{cn_i \cdot cc_i^c} + D_i \cdot (TC[i])^T. \tag{6.56}$$

Podobnej modyfikacji ulegnie wzór 6.14, który wyznacza czas potrzebny do ukończenia wszystkich przypisanych zadań (w tym określanych jako zaległości), a więc pozwala na ocenę jakości harmonogramu dla każdej jednostki i:

$$v_i = ct_i + u_i = \frac{D_i \cdot (WL^s)^T}{cc_i^c} + \frac{D_i \cdot (WL^p)^T}{cn_i \cdot cc_i^c} + u_i + D_i \cdot (TC[i])^T. \tag{6.57}$$

Pozostałe wzory nie ulegną zmianom.

Rozdział 7

Wyniki badań

W ramach niniejszego rozdziału zaprezentowano wyniki badań przeprowadzonych z wykorzystaniem zaproponowanego systemu monitoringu. Pierwszy podrozdział poświęcony jest charakterystykom danych testowych oraz parametrom środowiska. W kolejnych trzech podrozdziałach zaprezentowano wyniki testów zaproponowanych w rozdziale 6 modeli monitoringu procesu harmonogramowania zadań obliczeniowych. Piąty podrozdział został poświęcony obszarom, w których zaproponowane modele monitoringu mogą znaleźć zastosowanie. Testy przeprowadzono z wykorzystaniem dedykowanego symulatora, zbudowanego zgodnie z diagramem przepływu pracy (rys. 2.1). Implementację symulatora opisano w ostatnim podrozdziale.

7.1. Charakterystyki danych testowych oraz parametry środowiska

W celu przeprowadzenia badania wpływu zaproponowanego systemu monitoringu na proces harmonogramowania niezależnych zadań, przygotowano zestawy danych testowych w postaci charakterystyk zadań i jednostek obliczeniowych. Bazując na niżej przytoczonych badaniach, dokonano wyboru parametrów rozkładów wykorzystanych w procesie generowania danych testowych. Rozkłady wartości charakterystyk jednostek zgodne są z rozkładem normalnym oznaczanym jako:

$$\mathcal{N}(\mu;\, \sigma^2), \tag{7.1}$$

gdzie μ to wartość oczekiwana, a σ^2 to wariancja. Natomiast wartości zapotrzebowań obliczeniowych opisane są za pomocą rozkładu wykładniczego:

$$Exp(\lambda), \tag{7.2}$$

gdzie λ to odwrotność parametru skali, równa odwrotności wartości oczekiwanej.

J. Kołodziej w [67] przeprowadza symulacje rozważając cztery scenariusze, w których liczba jednostek obliczeniowych wynosi od 32 do 256, a liczba zadań od 512 do 4096. Zdolności obliczeniowe jednostek wygenerowane zostały za pomocą rozkładu normalnego $\mathcal{N}(5000; 875)$, podobnie jak zapotrzebowanie obliczeniowe $\mathcal{N}(250000000; 43750000)$. Analogicznie dane generuje M. Szmajduch w [111], gdzie dla zadań przyjmuje rozkład jak powyżej, a dla jednostek obliczeniowych $\mathcal{N}(1000; 175)$. Symulowane środowisko składa się z 64 jednostek, na których zrealizowano pakiet składający się z 1024 zadań. P. Świtalski i F. Seredyński w [109] rozważają scenariusze, w których środowisko składa się z 4 do 48 jednostek, a każdy pakiet zawiera od 100 do 500 zadań. Natomiast w pracy [42] J. Gąsior i F. Seredyński symulują środowisko składające się z 8 do 64 jednostek, które realizują zadania przesłane przez 16 niezależnych użytkowników systemu. Każdy z użytkowników generuje zestaw 1000 zadań. J. Tchórzewski i wsp. w [112] oszacowali liczbę operacji zmiennoprzecinkowych potrzebnych do wykonania rozmycia gaussowskiego. Rozważane w pracy obrazy miały rozmiar od 200×200 px do 2000×2000 px. Liczba operacji potrzebna do realizacji zadań wynosiła od 180 281 484 do 5 027 151 620.

Rzeczywiste charakterystyki jednostek obliczeniowych udostępniane są na wielu stronach poświęconych testom porównawczym procesorów [2, 6, 10, 12]. Charakterystyki takie można znaleźć również na stronie programu badawczego Asteroids@home [8]. W projekcie w sumie zaangażowanych jest 213 633 komputerów, o łącznej mocy obliczeniowej 3 080,32 TFLOPS[1]. Moc obliczeniowa pojedynczego rdzenia waha się od 0,16 do 6,22 GFLOPS. Testy wydajnościowe zostały wykonane za pomocą oprogramowania Whetstone, jednego z najpopularniejszych narzędzi do oceny wydajności komputerów.

[1]Dane na 11 maja 2018r.

Podobne pomiary wykonywane są w ramach oprogramowania Geekbench [2]. Narzędzie to pozwala na testowanie wydajności procesora i pamięci RAM pod kątem szybkości i wydajności przetwarzania różnych zestawów testowych, m.in. SGEMM (operacje typu macierz * macierz – poziom trzeci procedur BLAS) czy SFFT (szybka transformata Fouriera dla macierzy rzadkich). Producent oprogramowania udostępnia dodatkową stronę z pomiarami dla konkretnych jednostek obliczeniowych. Wiele z pomiarów zostało dokonanych na wirtualnych maszynach, m.in. największego usługodawcy rozwiązań chmurowych – firmy Amazon. Dla testu SFFT usługi uzyskiwały osiągi od 8,16 GFLOPS do 14,4 GFLOPS dla jednego rdzenia, natomiast dla SGEMM – od 37,5 GFLOPS do 171 GFLOPS.

Szerokie testy wykonali M. Mohammadi i T. Bazhirov, których rezultaty zostały opublikowane w pracy [77]. Autorzy, korzystając z testów LINPACK, dokonali ewaluacji zasobów obliczeniowych oferowanych przez największych dostawców usług typu cloud computing. Testom podlegały usługi zbudowane nawet z kilkudziesięciu węzłów obliczeniowych. W ramach usług dostarczano od 16 do 1536 rdzeni. Przeanalizowano ofertę czterech usługodawców: Amazon Web Services, Microsoft Azure, Rackspace oraz IBM SoftLayer, a także zbadano osiągi superkomputera NERSC Edison. Wydajność jednego rdzenia wahała się od 2,4 GFLOPS do 38,1 GFLOPS.

Podobne badanie zostało wykonane przez S. Ostermanna i wsp. [82]. Autorzy ograniczyli się do usług oferowanych w ramach platformy Amazon EC2. Testy zostały przeprowadzone za pomocą narzędzi LINPACK. Osiągi przebadanych rozwiązań wahały się od 0,78 GFLOPS do 3,47 GFLOPS w przeliczeniu na jeden rdzeń.

Na podstawie przeprowadzonej analizy, w ramach niniejszej pracy zdecydowano się przyjąć wartości zdolności obliczeniowych dla pojedynczego rdzenia z zakresu od 2 do 10 GFLOPS. Wygenerowany rozkład dla zdolności obliczeniowych jest zgodny z rozkładem normalnym. W pracy rozważane są dwa rozproszone środowiska – pierwsze, składające się z 40 jednostek, i drugie, składające się z 200 jednostek. Pełna charakterystyka jednostek znajduje się w tabeli 7.1. Przyjętym modelem zadania i jednostki obliczeniowej jest model uwzględniający możliwość zrównoleglenia wykonywanych operacji, opisany w podrozdziale 5.4.2.

Tablica 7.1: Charakterystyki jednostek obliczeniowych.

	Zestaw I	Zestaw II
Liczba jednostek	40	200
Miara cc^c	$GFLOPS$	
Rozkład wartości cc^c	$\mathcal{N}(6;\ 4)$	
Wartość oczekiwana	6	
Odchylenie standardowe	2	
Wartość minimalna	2	
Wartość maksymalna	10	
Liczba rdzeni (cn):		
1	15%	
2	30%	
4	40%	
8	10%	
16	5%	

Tablica 7.2: Charakterystyki zadań.

	Pakiet I	Pakiet II
Liczba zadań	4 000	20 000
Miara	$GFLO$	
Rozkład wartości wl^s	$Exp(0,0005)$	
Wartość oczekiwana wl^s	2 000	
Odchylenie standardowe wl^s	2 000	
Wartość minimalna wl^s	1 000	
Wartość maksymalna wl^s	8 000	
Rozkład wartości wl^p	$Exp(0,0002)$	
Wartość oczekiwana wl^s	5 000	
Odchylenie standardowe wl^s	5 000	
Wartość minimalna wl^s	1 000	
Wartość maksymalna wl^s	20 000	

Najczęściej pojawiającym się w literaturze sposobem generowania zapotrzebowania obliczeniowego zadań jest generowanie za pomocą rozkładu wykładniczego (por. [50, 119, 123]). W ramach niniejszej pracy rozpatrywane są dwa pakiety różniące się liczbą zadań: 4 000 lub 20 000. Pełne charakterystyki znajdują się w tabeli 7.2.

Dane testowe generowane są za pomocą dedykowanego generatora opisanego bliżej w podrozdziale 7.6. Każdy ze scenariuszy testowych został powtórzony 100-krotnie. Maksymalna liczba epok w każdym z rozważanych przypadków wynosiła 10 000. Prawdopodobieństwo wystąpienia mutacji wynosiło 5%. Harmonogramy były optymalizowane pod kątem czasu wykonania wszystkich zadań z pakietu.

Tablica 7.3: Charakterystyka komputera na którym wykonano symulacje.

Procesor	Intel(R) Core(TM) i7-4710HQ CPU @ 2,50GHz
Liczba rdzeni/wątków	4/8
Architektura	x86_64
Pamięć operacyjna	16GiB SODIMM DDR3 Synchronous 1600 MHz (0,6 ns)
Dysk	256GB ADATA SP600
System operacyjny	Windows 8.1, 64-bit

Symulacje zostały wykonane na komputerze osobistym, którego charakterystyka przedstawiona jest w tabeli 7.3.

7.2. Agent typu Ag0

W pierwszej kolejności dokonano testów działania agenta typu Ag0. Agent ten, poprzez realizację jednej ze strategii, ma możliwość ograniczania liczby osobników przeznaczonych do reprodukcji, a więc zastępuje rolę operatora selekcji. Szczegółowy opis modelu znajduje się w podrozdziale 6.5.1.

W ramach niniejszej pracy rozważanych jest siedem dynamicznych strategii ingerencji w proces harmonogramowania zadań. Każda ze strategii definiuje liczbę epok, jaka musi zostać wykonana w procesie ewolucyjnym od momentu uzyskania najlepszego rozwiązania, po której następuje ograniczenie zakresu par osobników podlegających krzyżowaniu. Agent za każdym razem podejmuje decyzję o wykluczeniu jednej pary osobników. Liczba epok wyznaczana jest zgodnie z poniższą zależnością:

$$I_\kappa = \Phi \cdot \frac{I_{max}}{\frac{m}{2}}, \tag{7.3}$$

gdzie Φ ($\Phi \in (0, 100]$) jest wartością procentową, zależną od realizowanej strategii, która po przemnożeniu przez maksymalną liczbę iteracji (I_{max}) podzieloną przez liczbę par osobników ($\frac{m}{2}$), określa liczbę iteracji konieczną do zmiany (inkrementacji) wartości parametru κ. We wszystkich przeprowadzonych testach maksymalna liczba epok w procesie ewolucyjnym wynosiła 10 000.

Zaproponowane strategie zostały przedstawione w tabeli 7.4. Wszystkie zostały prze-

Tablica 7.4: Strategie agenta typu Ag0.

Nazwa strategii	Wartość parametru Φ	Wartość I_κ dla zestawu I	Wartość I_κ dla zestawu II
D10	10%	50	10
D30	30%	150	30
D50	50%	250	50
D60	60%	300	60
D90	90%	450	90
D100	100%	500	100
D150	150%	750	150
UNS	Proces harmonogramowania niewspierany		

Tablica 7.5: Scenariusze testowe.

	Scenariusz I	Scenariusz II	Scenariusz III	Scenariusz IV
Liczba zadań	4 000	20 000	4 000	20 000
Liczba jednostek obliczeniowych	40	40	200	200
Liczba epok	10 000			
Prawdopodobieństwo mutacji	5%			
Liczba powtórzeń	100			

testowane w ramach zdefiniowanych scenariuszy testowych z tabeli 7.5. Otrzymane rezultaty realizacji poszczególnych strategii zostały porównane z wynikami otrzymanymi w ramach testów procesu harmonogramowania niewspieranego przez system agentowy. Strategia, w której agent nie ingeruje w proces harmonogramowania została oznaczona symbolem UNS.

W pierwszej kolejności zbadano wpływ monitoringu i ingerencji agenta w proces szeregowania na jakość otrzymanych harmonogramów. Rozważanym w niniejszej pracy problemem jest harmonogramowanie niezależnych zadań obliczeniowych w trybie pakietowym. W klasycznym problemie tego typu jakość harmonogramów utożsamiana jest z czasem wykonania wszystkich zadań z pakietu. Im krótszy czas wykonania całego pakietu, tym harmonogram jest uznawany za lepszy. Wpływ zastosowanych strategii można zaobserwować na wykresach 7.1, 7.2 oraz 7.3.

Najlepsze rezultaty otrzymano dla strategii D50 i D60 (rys. 7.2), czyli tych, w ramach których zmiana liczby krzyżowanych par osobników następowała po 250 – 300 (dla zestawu I) lub 50 – 60 (dla zestawu II) epokach niegenerujących lepszego rozwiązania.

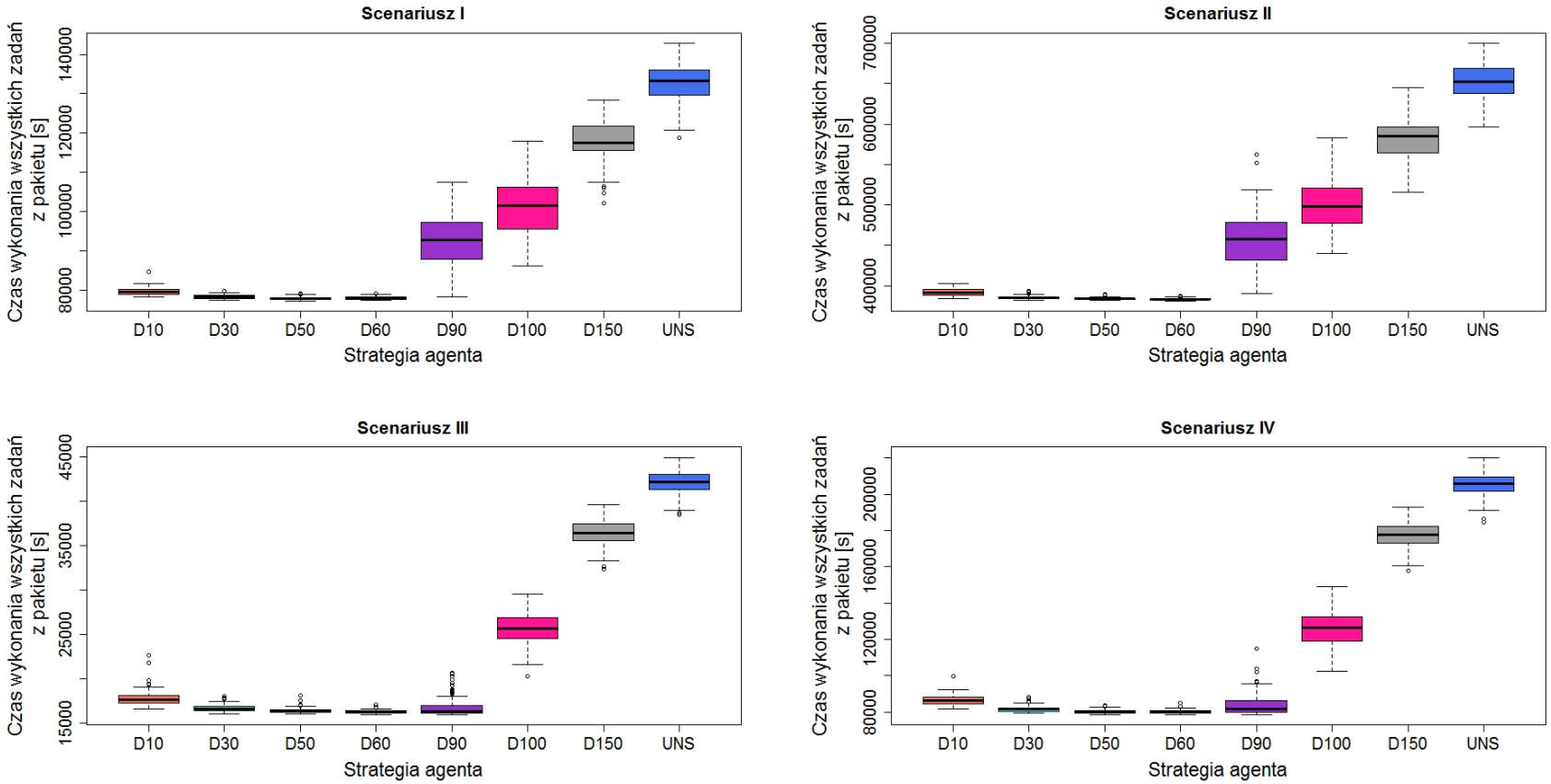
Porównując strategie D10, D30, D50 i D60 z pozostałymi strategiami (rys. 7.1), można

dostrzec znaczne różnice w rozproszeniu wartości – uzyskane czasy dla wspomnianych strategii są bardziej skumulowane wokół mediany. Natomiast dla pozostałych strategii wartości są rozproszone zarówno w kwartylach 1 i 3 oraz w przedziałach międzykwartylowych. Na tej podstawie można wywnioskować, że zastosowanie monitoringu wraz z realizacją odpowiedniej strategii eliminacji osobników z procesu krzyżowania (i ich przeżywalności) pozwala na stabilizację ewolucyjnego procesu generowania harmonogramów. Harmonogramy takie – dla określonych, z góry znanych charakterystyk – będą cechować się bardziej przewidywalnym czasem potrzebnym na wykonanie całego pakietu zadań.

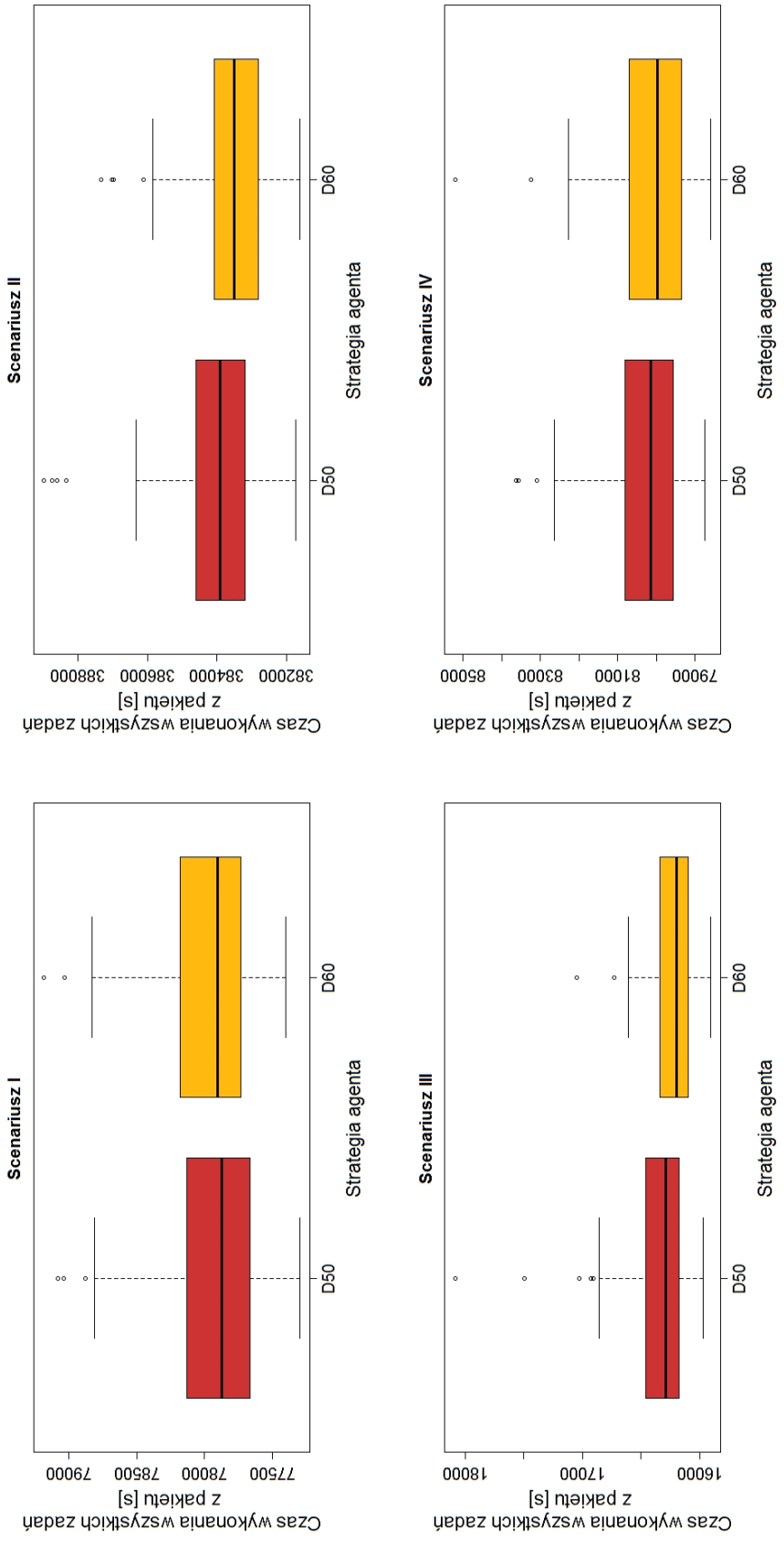
Najważniejszą korzyścią, jaką można zaobserwować na podstawie zaprezentowanych wykresów jest istotna redukcja czasów wykonania pakietów zadań. Każda z zaproponowanych strategii uzyskała lepszy średni wynik w porównaniu do harmonogramowania bez nadzoru agentowego (wykres 7.1). W przypadku środowisk składających się z większej liczby jednostek, zysk, czyli różnica średnich czasów realizacji pakietów zadań, z zastosowania wsparcia procesu harmonogramowania jest większy. Dla scenariusza I najlepsze rezultaty osiągnęła strategia D50 – dla której poprawa w stosunku do szeregowania nienadzorowanego (UNS) wynosi 41,36%. Zbliżony wynik uzyskano dla strategii D60 – 41,33%. Pozostałe strategie również odnotowały znaczące poprawy: D10 – 40,11%, D30 – 41,08%, D90 – 30,34%, D100 – 23,88%, D150 – 11,47%. Podobne procentowo rezultaty uzyskano dla scenariusza II. W przypadku tego scenariusza, najskuteczniejsza okazała się strategia D60 – 41,21% (o 0,07% lepsza od strategii D50).

Scenariusz III i IV odnoszą się do znacznie większych środowisk, zbudowanych z 200 jednostek. Również w tych scenariuszach wyraźnie widoczna jest różnica pomiędzy harmonogramowaniem wspieranym przez agenta typu Ag0 a harmonogramowaniem nienadzorowanym. Podobnie jak w przypadku poprzednich scenariuszy, najlepsze rezultaty (dla obu scenariuszy) uzyskano w ramach strategii D60 – średni czas planowego wykonania pakietu wyniósł odpowiednio 16 233 oraz 80 147,31 sekund. Warto również zwrócić uwagę na rezultaty strategii D90, które są znacznie lepsze niż uzyskane dla scenariuszy I i II. W odniesieniu do szeregowania nienadzorowanego, poszczególne strategie zanotowały poprawy: D10 – 57,81%, D30 – 60,40%, D50 – 61,10%, D60 – 61,38%, D90 – 60,09%, D100 – 39,14%, D150

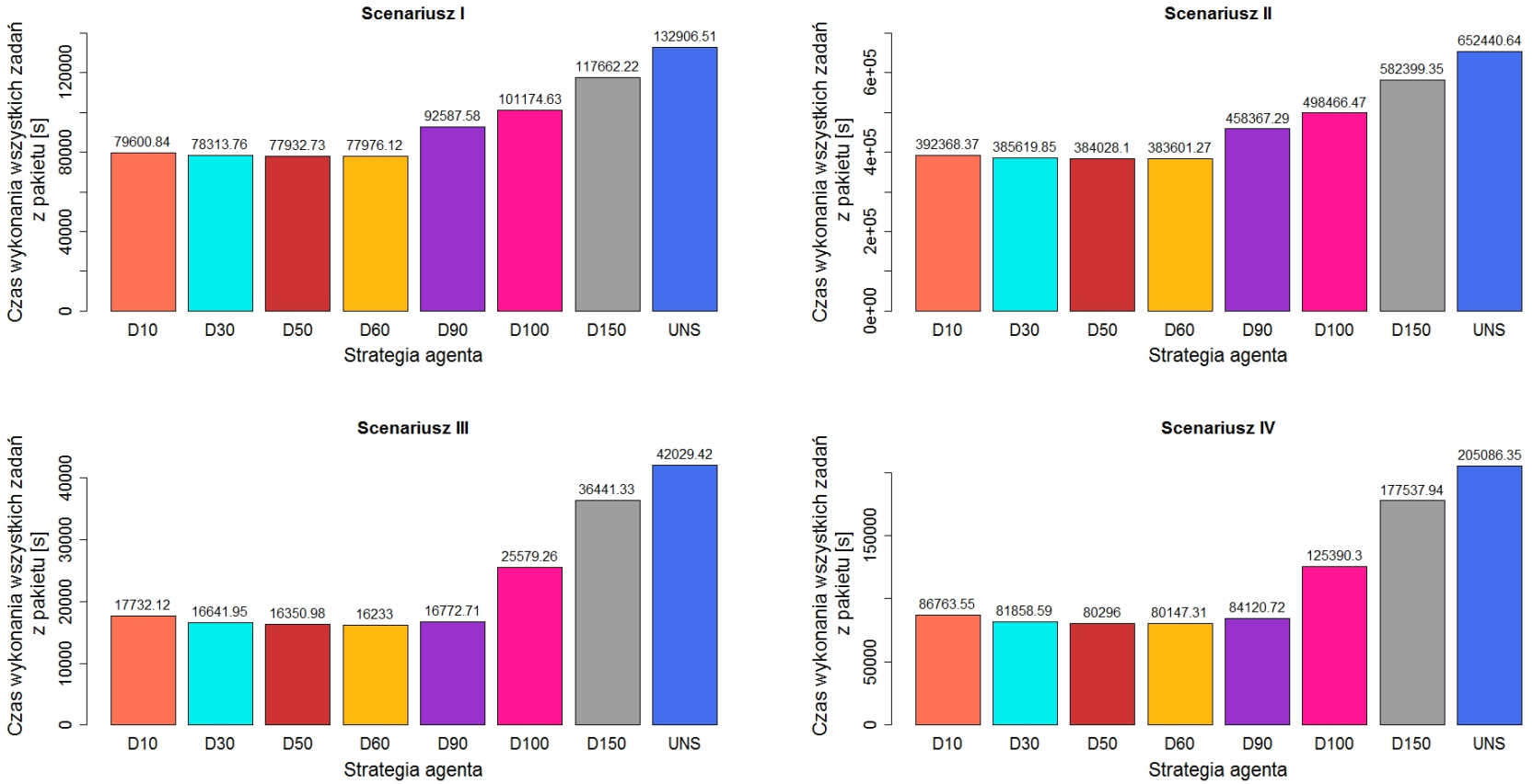
– 13,30%. W porównaniu do rezultatów uzyskanych w ramach scenariusza I i II, korzyści wynikające z zastosowania monitoringu są jeszcze wyższe (nawet o 20% dla strategii D60 i blisko o 30% dla D90).



Rysunek 7.1: Wykresy pudełkowe czasów wykonania wszystkich zadań z pakietu dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.



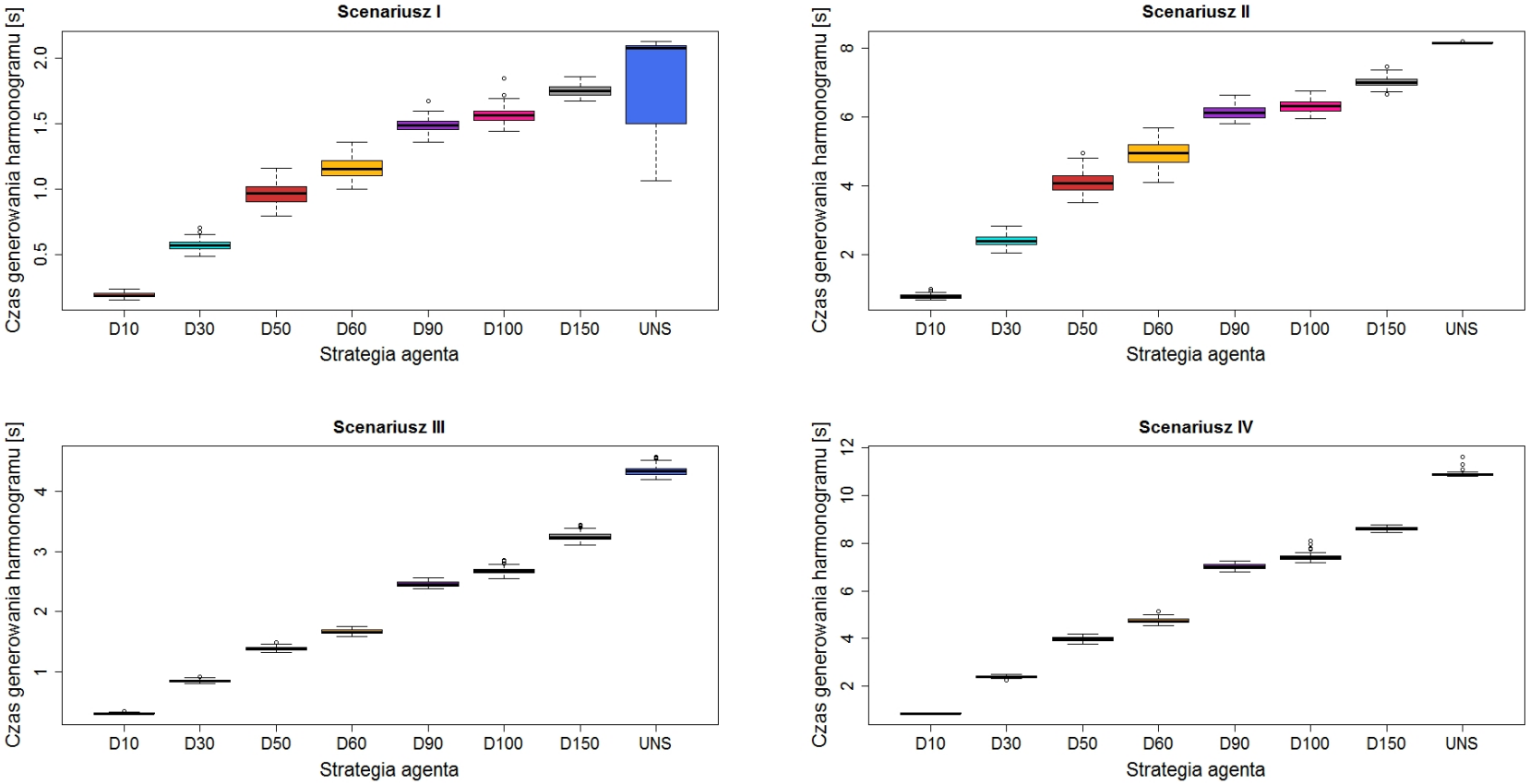
Rysunek 7.2: Wykresy pudełkowe czasów wykonania wszystkich zadań z pakietu dla dwóch najlepszych strategii agenta typu Ag0.



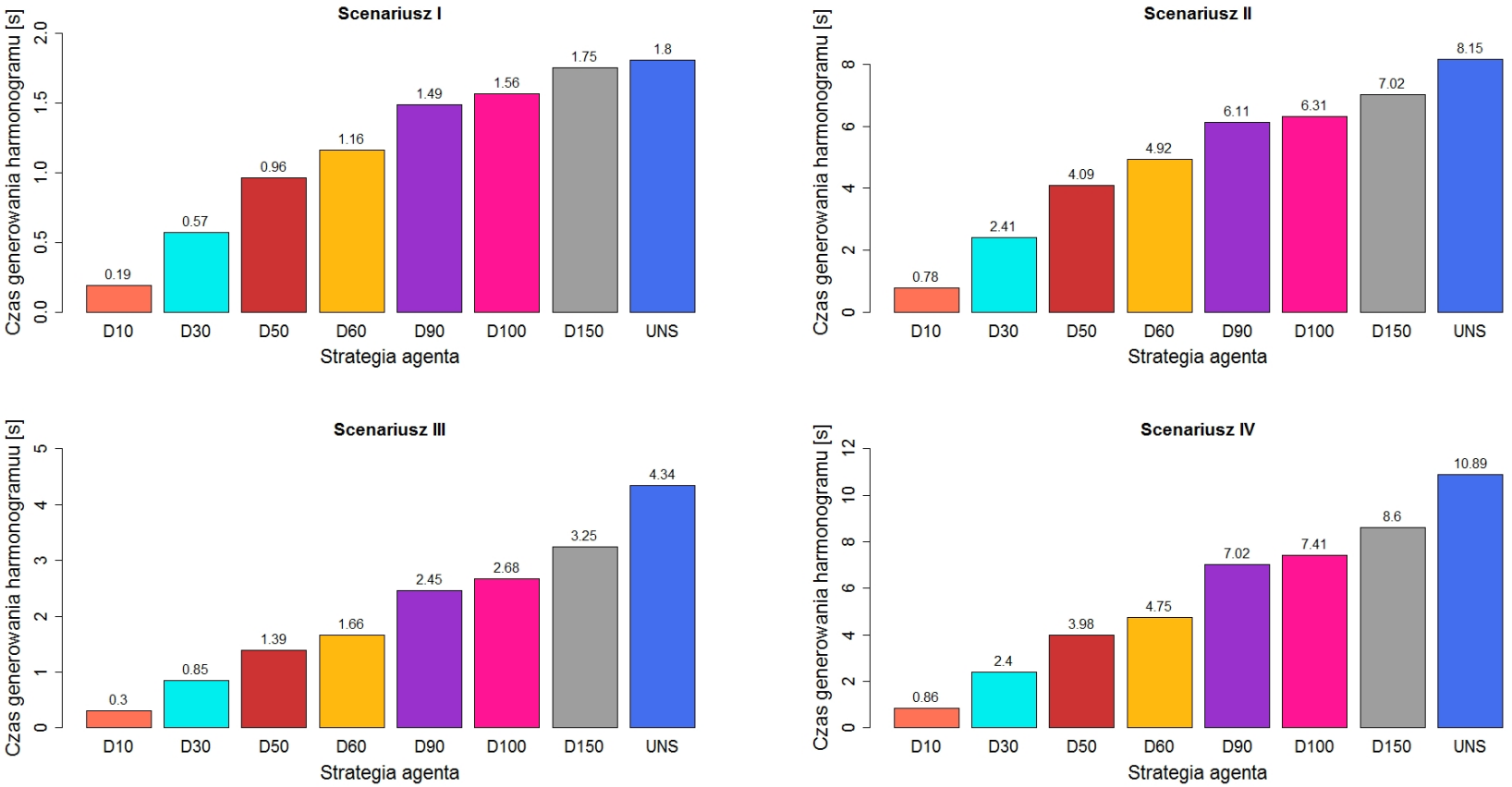
Rysunek 7.3: Średnie czasy wykonania wszystkich zadań z pakietu dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.

Kolejnym istotnym pomiarem był czas poświęcony na generowanie harmonogramu dla pojedynczego pakietu zadań. Wyniki zostały przedstawione na rysunkach 7.4 oraz 7.5. Najlepsze rezultaty uzyskano dla strategii D10, natomiast najgorsze dla harmonogramowania bez wsparcia (UNS). Strategia D10 powoduje zmniejszenie liczby osobników podlegających reprodukcji po 50 (dla zestawu I) lub 10 (dla zestawu II) epokach nieprzynoszących lepszego rozwiązania. Dzięki zmniejszaniu liczby krzyżowanych osobników algorytm ewolucyjny charakteryzuje się niższą złożonością obliczeniową. Ponadto w ramach realizacji swojej strategii agent może zmniejszyć maksymalną liczbę epok I_{max}, co skutkuje szybszym zakończeniem poszukiwania rozwiązania. Zysk ze stosowania strategii D10 sięga aż 93% (dla scenariusza III) oraz 67,96% i 61,76% odpowiednio dla strategii D50 i D60 (również dla scenariusza III). W przypadku scenariuszy I i II, zysk dla najlepszych strategii pod względem jakości generowanego rozwiązania jest nieco niższy i mieści się w zakresie od 35 do 50%. Należy jednak odnotować fakt, że dużą niestabilnością w zakresie czasu generowania rozwiązania dla scenariusza I charakteryzuje się szeregowanie nienadzorowane (UNS), gdzie pierwszy kwartyl i dolny przedział międzykwartylowy obejmują bardzo szeroki zakres wartości – odmiennie natomiast trzeci kwartyl i górny przedział międzykwartylowy, które mieszczą się w niewielkim zakresie wartości. W pozostałych scenariuszach czas generowania harmonogramów jest stabilny.

Warto również zauważyć niewielki czas generowania harmonogramów dla wszystkich przeprowadzonych testów. Rezultaty wszystkich rozważanych scenariuszy dla strategii D10 nie przekroczyły 1 sekundy. Nawet scenariusz IV, charakteryzujący się niezwykle dużą liczbą rozważanych jednostek i zadań obliczeniowych (odpowiednio 200 i 20 000), dla strategii D10 uzyskał średni czas 0,86 sekundy, a najlepsza strategia pod względem jakości generowanego rozwiązania (D60), na jego uzyskanie poświęciła średnio 3,9 sekundy więcej. Porównując te czasy z czasem wykonywania wszystkich zadań z pakietu, jest to jedynie nieznaczny ułamek. Uzyskane wyniki świadczą o wysokiej sprawności zaproponowanego podejścia do szeregowania zadań.



Rysunek 7.4: Wykresy pudełkowe czasów generowania harmonogramu dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.



Rysunek 7.5: Średnie czasy generowania harmonogramu dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.

Istotnym zagadnieniem w systemach obliczeniowych jest tzw. równoważenie obciążania (ang. *load balancing*), mające na celu racjonalizację przydziału zadań. Zaproponowana implementacja algorytmu ewolucyjnego w ramach prostej techniki równoważenia obciążenia zapewnia przydział co najmniej jednego zadania do każdej z dostępnych jednostek obliczeniowych. W ramach niniejszej pracy przeanalizowano wpływ strategii na sposób wykorzystania dostępnej infrastruktury poprzez pomiar czasów bezczynności poszczególnych jednostek obliczeniowych. Rezultaty badań zostały przedstawione w tabeli 7.6 oraz na rysunkach 7.6, 7.7 i 7.8.

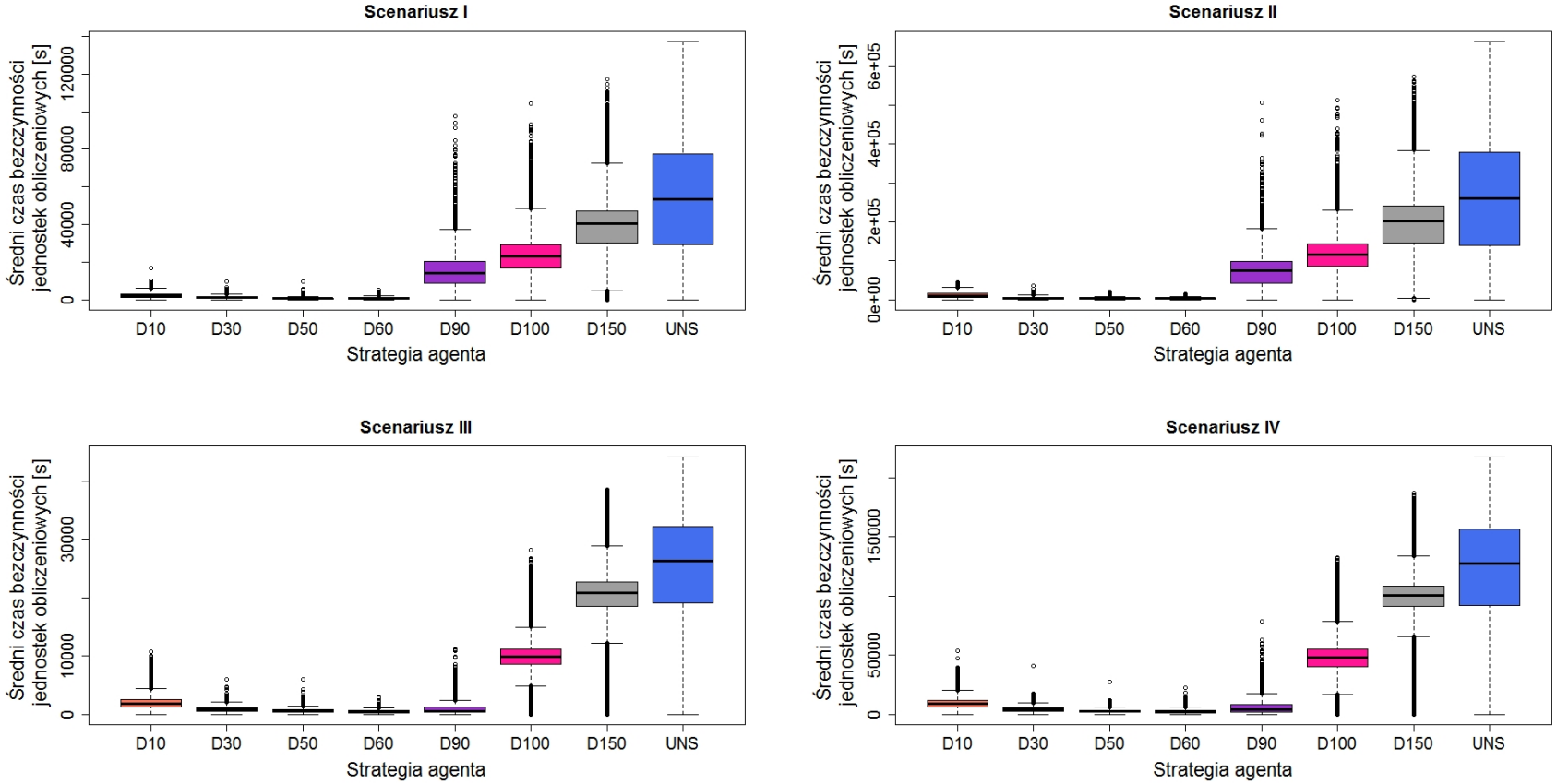
Podobnie jak w przypadku pomiarów dotyczących czasu wykonania wszystkich zadań pakietu, tak i w obszarze równoważenia obciążenia najlepsze rezultaty otrzymano dla strategii D50 i D60. Harmonogramy przygotowane w ramach realizacji obu strategii cechują się najniższym czasem bezczynności (rys. 7.6). Porównując strategię D60 do harmonogramowania niewspieranego agentowo, czas bezczynności uległ skróceniu, odpowiednio dla scenariuszy I, II, III i IV, aż o 98,76%, 99,01%, 97,91% i 97,75%.

Należy jednak odnotować dużą rozpiętość wartości dla przypadków mniej efektywnych, tj. D90, D100, D150 i UNS. W szczególności, podejście nienadzorowane cechuje się wartościami zarówno bliskimi najlepszym strategiom, jak i niezwykle odległymi. Może to świadczyć o stabilizującym wpływie ingerencji agenta na proces generowania harmonogramów.

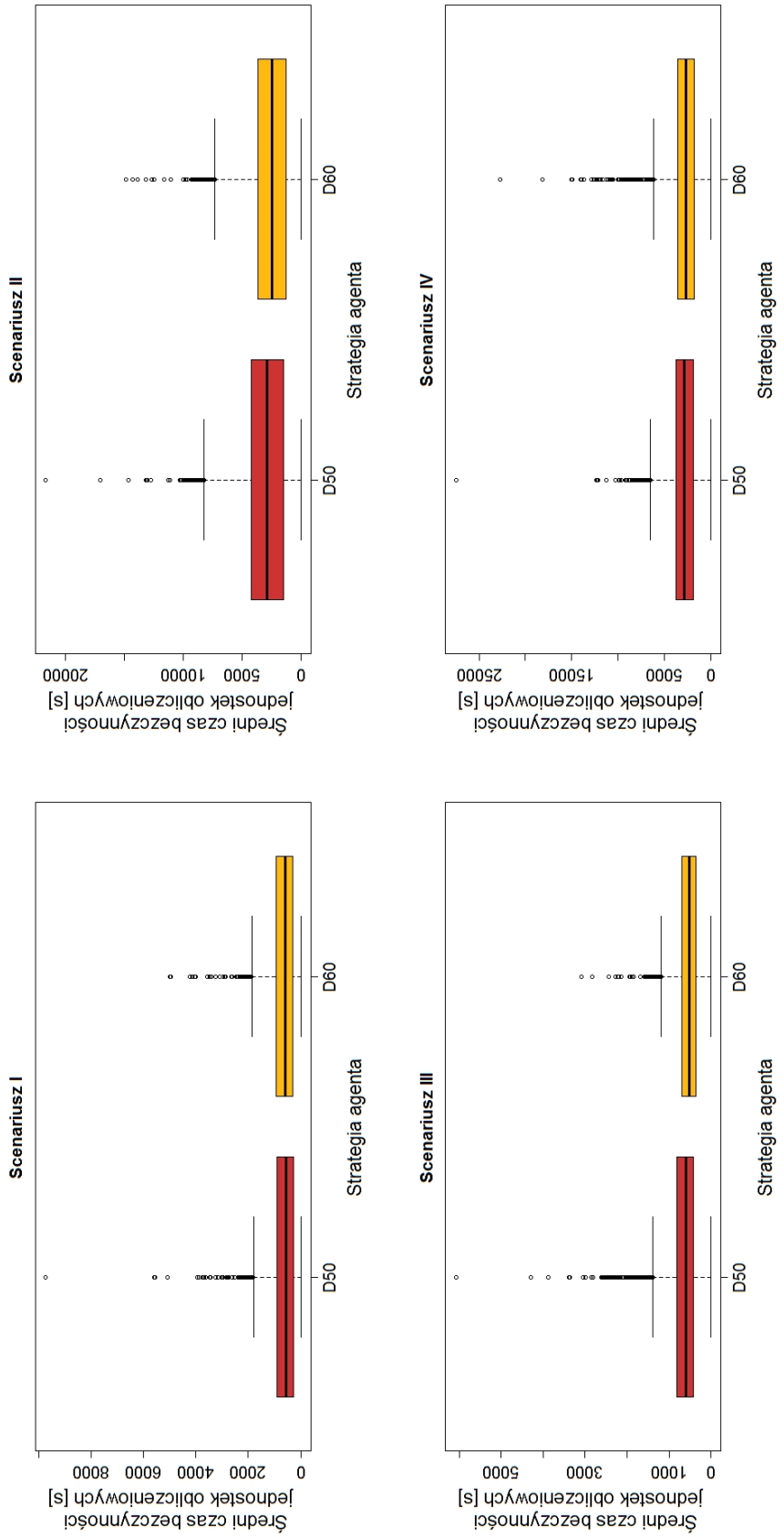
Porównując wyniki średnich czasów bezczynności jednostek obliczeniowych w stosunku do średnich czasów wykonania wszystkich zadań pakietu, również można dostrzec korzystny wpływ procesu monitoringu i wsparcia szeregowania zadań. Dla strategii D60, jednostki obliczeniowe pozostawały bezczynne średnio przez 2,04% czasu realizacji wszystkich zadań z pakietu, gdzie dla harmonogramowania nienadzorowanego wynik ten wyniósł średnio 50%. Najlepsze rezultaty uzyskano dla scenariuszy charakteryzujących się niewielką liczbą jednostek obliczeniowych. Pełne wyniki zostały przedstawione w tabeli 7.6.

Tablica 7.6: Stosunek czasu bezczynności jednostek obliczeniowych do czasu wykonania wszystkich zadań pakietu dla strategii agenta typu Ag0 i bez wsparcia agentowego (UNS).

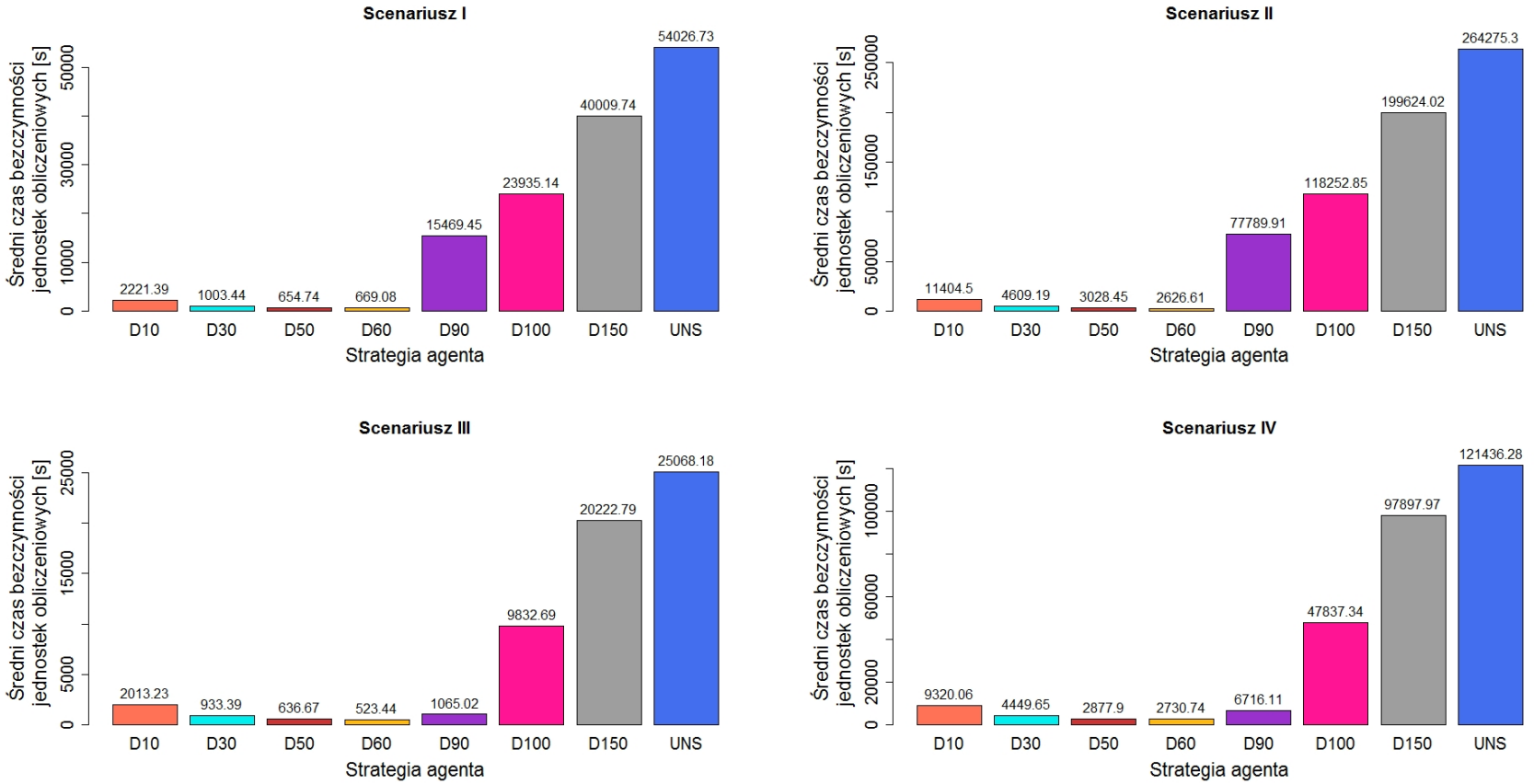
Nazwa strategii	Scenariusz I	Scenariusz II	Scenariusz III	Scenariusz IV
D10	2,79%	2,91%	11,35%	10,74%
D30	1,28%	1,20%	5,61%	5,44%
D50	0,84%	0,79%	3,89%	3,58%
D60	0,86%	0,68%	3,22%	3,41%
D90	16,71%	16,97%	6,35%	7,98%
D100	23,66%	23,72%	38,44%	38,15%
D150	34,00%	34,28%	55,49%	55,14%
UNS	40,65%	40,51%	59,64%	59,21%



Rysunek 7.6: Wykresy pudełkowe średnich czasów bezczynności jednostek obliczeniowych dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.



Rysunek 7.7: Wykresy pudełkowe średnich czasów bezczynności jednostek obliczeniowych dla dwóch najlepszych strategii agenta typu Ag0.



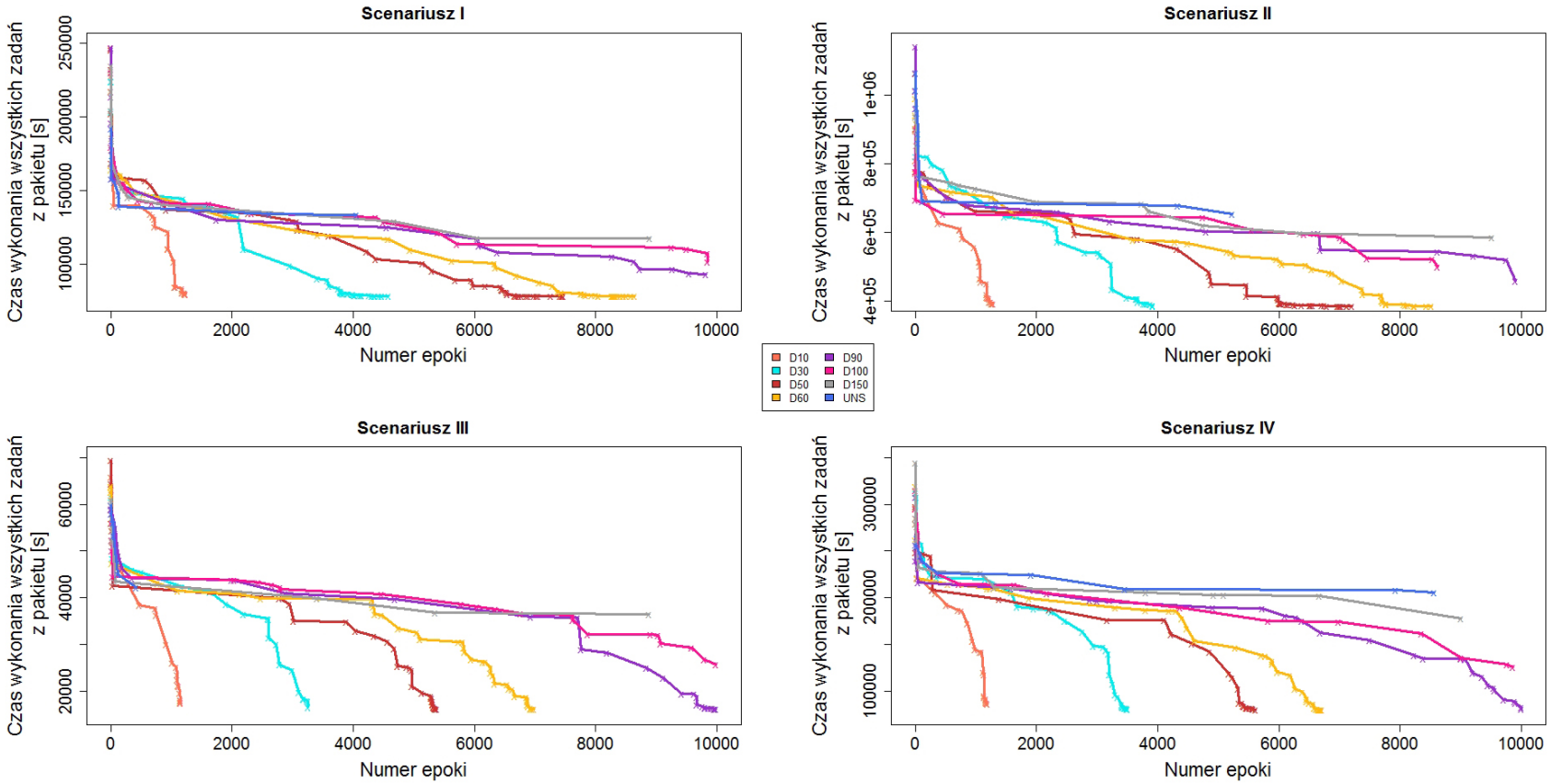
Rysunek 7.8: Średnie czasy bezczynności jednostek obliczeniowych dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.

Ostatnie z przeprowadzonych pomiarów dotyczą poprawy jakości generowanych harmonogramów wraz z upływem epok oraz samej liczby epok. Wykres obrazujący generację najlepszego rozwiązania w zależności od epoki znajduje się na rys. 7.9. W celu przedstawienia niniejszych zmian wybrano przeciętne rozwiązania (medianę) pod względem czasu wykonania wszystkich zadań z pakietu dla każdej z realizowanych strategii w każdym scenariuszu. Na wykresie można zaobserwować szybszą poprawę generowanych rozwiązań dla strategii charakteryzujących się niską wartością parametru Φ (np. dla strategii D10). Im wartość parametru Φ jest większa, tym proces uzyskiwania lepszego rozwiązania jest bardziej rozłożony na przestrzeni epok, a dynamika mniejsza.

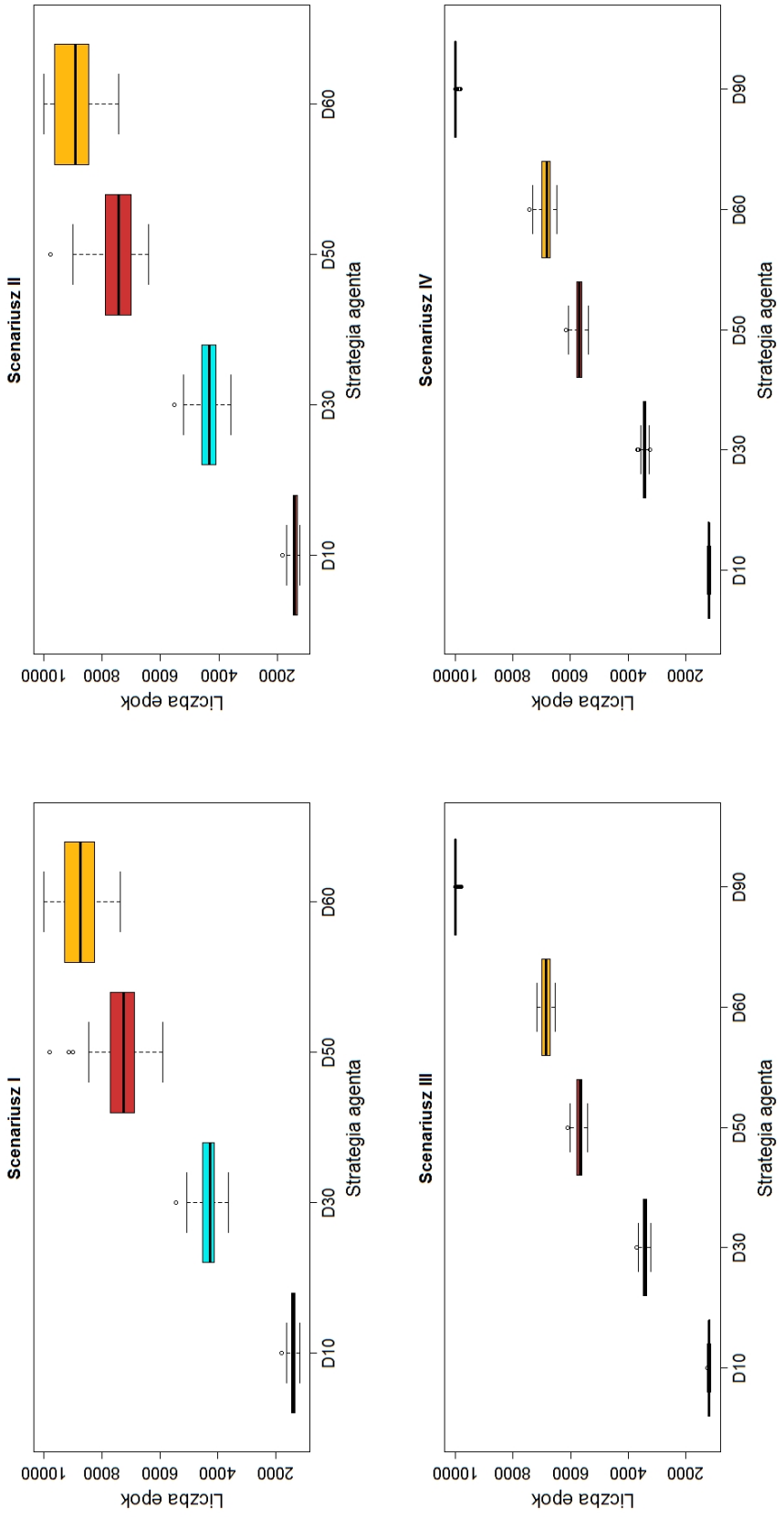
Zgodnie z przyjętymi strategiami, zwiększanie parametru κ następuje do osiągnięcia maksymalnej liczby epok lub do wyłączenia wszystkich osobników z procesu reprodukcji. Wykresy 7.10 i 7.11 przedstawiają odpowiednio rozkład liczby epok oraz średnią liczbę epok dla wybranych strategii agenta typu Ag0. Na wykresach pominięto strategie, w przypadku których liczba epok zawsze wynosiła I_{max}. Zgodnie z przewidywaniami, algorytm ewolucyjny najszybciej został zakończony dla strategii D10; strategie D90 (dla scenariusza I i II), D100, D150 oraz harmonogramowanie nienadzorowane nie kończyły procesu ewolucji przed osiągnięciem epoki I_{max}.

Ostatni z wykresów, rysunek 7.12 prezentuje średni numer epoki, w której uzyskano najlepsze rozwiązanie w całym procesie ewolucji. W przypadku większości strategii dynamicznych, numer epoki najlepszego rozwiązania bliski jest całkowitej liczbie epok z wykresu 7.11. W przypadku szeregowania niewspieranego agentowo, przez blisko połowę epok algorytm nie wygenerował lepszego harmonogramu.

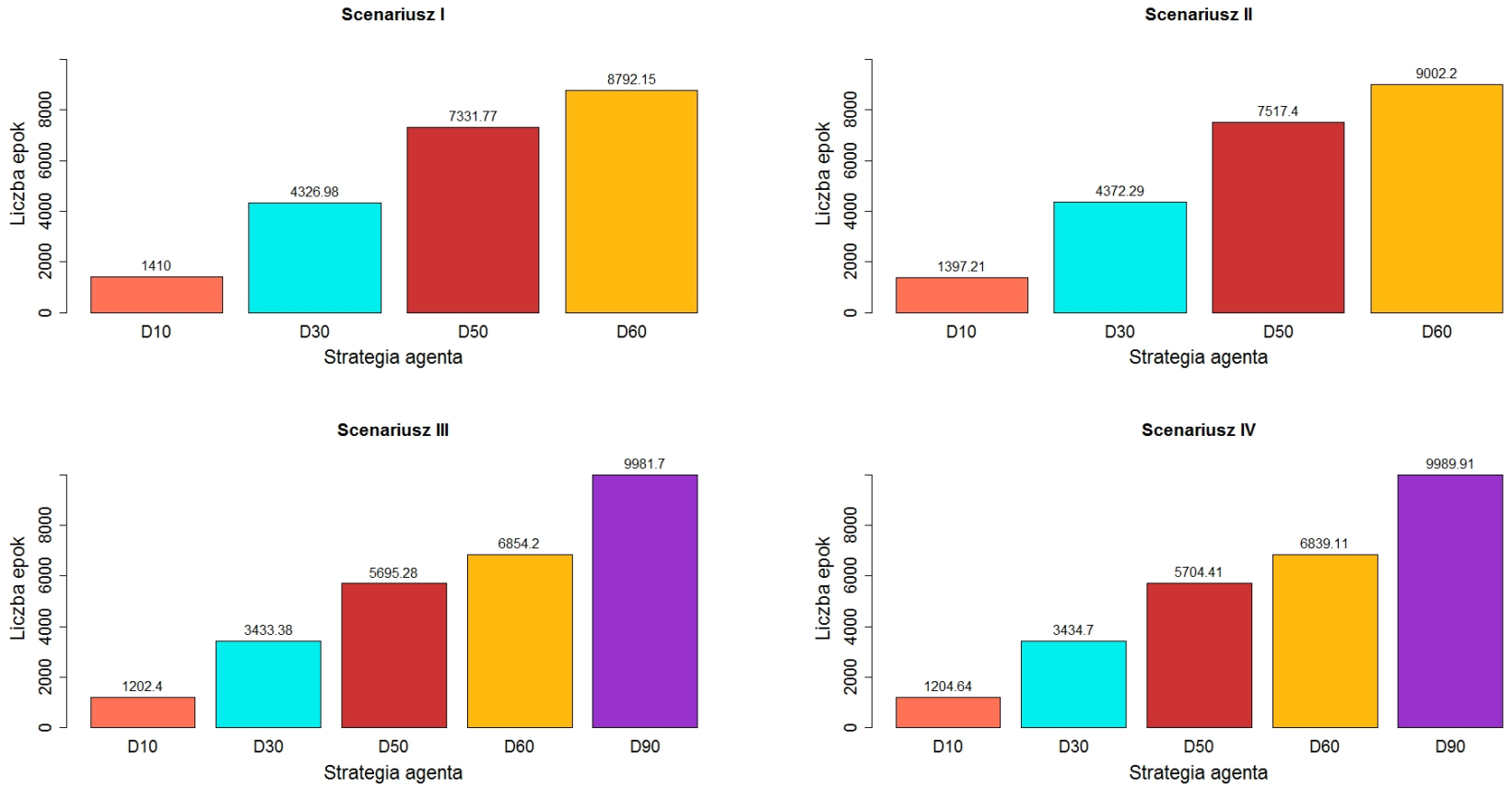
Analizując przedstawione wykresy można dostrzec, że w większości przypadków harmonogramowania nienadzorowanego, proces ewolucyjny może zostać zakończony znacznie wcześniej. Ponadto zasadnym wydaje się być zwiększenie liczby epok w ostatnich stadiach, tj. kiedy liczba osobników dopuszczonych do procesu reprodukcji jest bliska minimum, dla strategii charakteryzujących się niską wartością parametru Φ (tj. od 10 do 60).



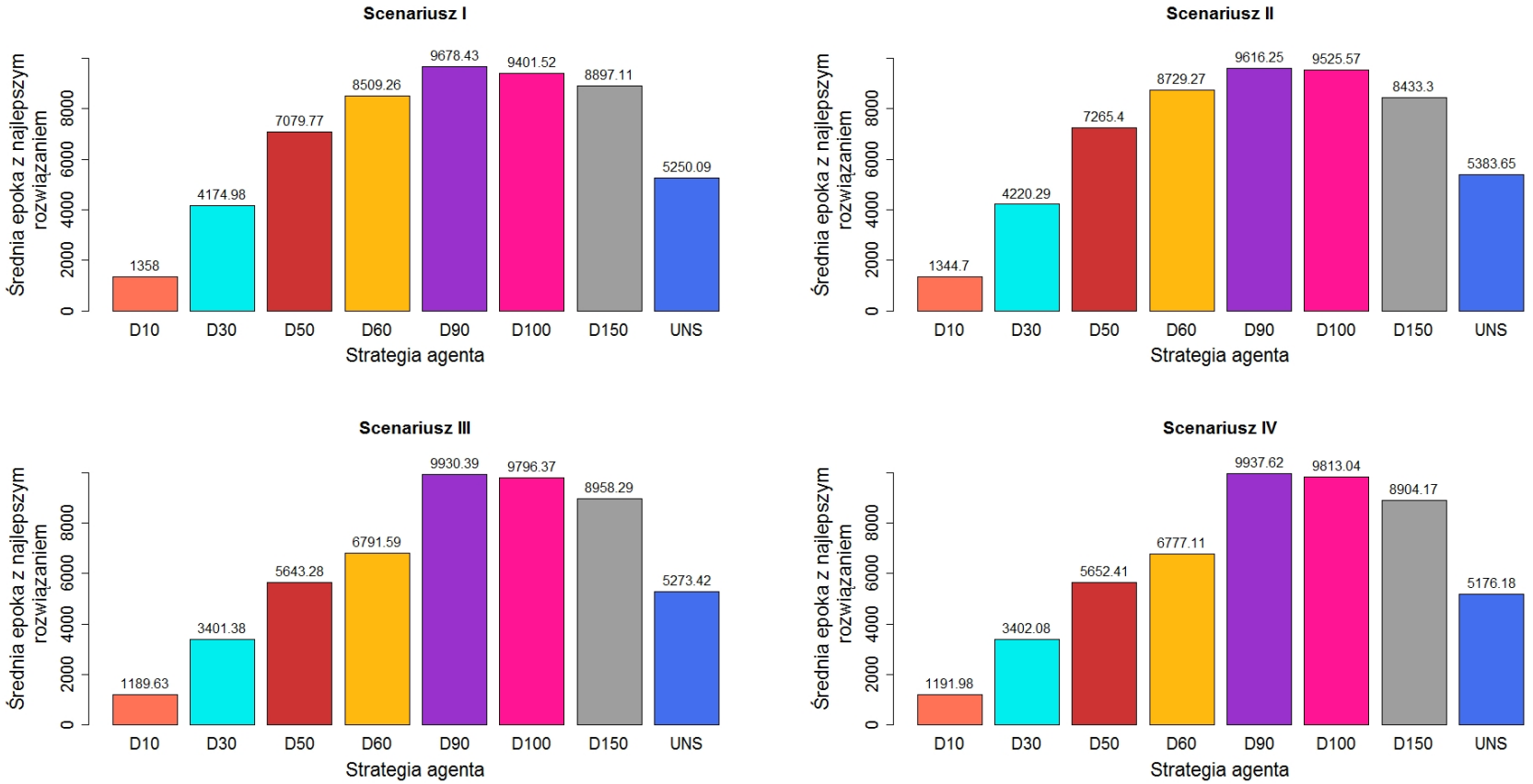
Rysunek 7.9: Wykresy obrazujące zmiany najlepszego rozwiązania pod względem czasów wykonania wszystkich zadań z pakietu dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.



Rysunek 7.10: Wykresy pudełkowe liczby epok dla wybranych strategii agenta typu Ag0.



Rysunek 7.11: Średnie liczby epok dla wybranych strategii agenta typu Ag0.



Rysunek 7.12: Średnie epoki, w których wygenerowano najlepsze rozwiązanie dla każdej ze strategii agenta typu Ag0 oraz harmonogramowania bez wsparcia agentowego.

7.3. Agenty typu Ag1 i Ag2

Agenty typu Ag1 i Ag2 mają za zadanie zarządzanie procesem harmonogramowania poprzez jego wywoływanie w możliwie jak najbardziej korzystnym momencie. Problem harmonogramowania niezależnych zadań obliczeniowych w trybie pakietowym rozważany jest zwykle w ramach realizacji pojedynczych pakietów zadań. W rzeczywistości, w wielu przypadkach systemy obliczeniowe (szczególnie systemy typu cloud computing) działają w sposób ciągły. W związku z tym pakiety przetwarzane są sekwencyjnie, jeden po drugim. Może to powodować nieoptymalne wykorzystanie infrastruktury.

W ramach zaprezentowanego modelu rozważany jest problem generowania momentów rozpoczęcia procesu harmonogramowania w kontekście ciągłości pracy środowiska. Ideą modelu jest monitoring postępów realizacji zleconych harmonogramów oraz bezczynności poszczególnych jednostek obliczeniowych. Zadania te realizują agenty typu Ag1. Agent typu Ag2 komunikuje się z agentami typu Ag1 i dokonuje zapytania o aktualny status realizacji harmonogramu. Na podstawie zebranych informacji podejmuje decyzję o rozpoczęciu (bądź też nie) kolejnego procesu harmonogramowania dotychczas zebranych zadań obliczeniowych. Decyzje podejmowane są na podstawie dwóch parametrów: θ_1 oraz θ_2. Parametr θ_1 określa liczbę jednostek obliczeniowych, które muszą być w stanie bezczynności, aby wygenerować nowy moment czasowy (jednoznaczny z zakończeniem procesu formowania pakietu zadań i rozpoczęciem procesu generowania harmonogramu), natomiast parametr θ_2 określa maksymalny czas oczekiwania na jednostki, które wkrótce zrealizują wszystkie aktualnie przypisane zadania i zostaną włączone do grona jednostek określonych przez θ_1. Szczegóły dotyczące działania agentów zostały przedstawione w podrozdziale 6.5.2.

Testy niniejszego modelu zostały przeprowadzone w ramach dedykowanego symulatora. Zasymulowane środowiska obliczeniowe zgodne są z charakterystykami przedstawionymi w tabeli 7.1. W badaniach rozważane były dwa pakiety scharakteryzowane w 7.2 (z liczbą zadań zależną od liczby jednostek obliczeniowych) w ramach procesu harmonogramowania niewspieranego agentowo. Liczba epok, prawdopodobieństwo mutacji oraz liczba powtórzeń zostały dobrane zgodnie z tabelą 7.5.

Na potrzeby testów dobór parametru θ_1 został uzależniony od procenta liczby jednostek obliczeniowych działających w środowisku. Zależność taką można zapisać w postaci następującej formuły:

$$\theta_1 = \Omega_1 \cdot m, \tag{7.4}$$

gdzie $\Omega_1 \in (0, 100])$ jest wartością procentową, która po przemnożeniu przez liczbę wszystkich jednostek obliczeniowych w środowisku (m), określa wartość parametru θ_1.

Podobnie parametr θ_2 został wyznaczony na podstawie procenta średniego obciążenia 1 GFLOPS mocy obliczeniowej wyliczonego na podstawie zapotrzebowania obliczeniowego zadań z obecnie formowanego pakietu oraz dostępnej w tym momencie[2] mocy obliczeniowej. Wartość parametru θ_2 można obliczyć w następujący sposób:

$$\theta_2 = \Omega_2 \cdot \frac{\sum_{j=1}^{n} wl_j^s}{\sum_{i=1}^{m} cc_i^c} + \frac{\sum_{j=1}^{n} wl_j^p}{\sum_{i=1}^{m} cn_i \cdot cc_i^c}, \tag{7.5}$$

gdzie wl_j^s to liczba operacji zmiennoprzecinkowych, które muszą być wykonane w sposób sekwencyjny, wl_j^p to liczba operacji zmiennoprzecinkowych, które mogą zostać zrównoleglone, cn_i to liczba rdzeni jednostki obliczeniowej, cc_i^c to moc obliczeniowa pojedynczego rdzenia jednostki i, a $\Omega_2 \in (0, 100])$ jest wartością określającą procent średniego obciążenia 1 GFLOPS mocy obliczeniowej, będący wartością parametru θ_2.

Przyjęto założenie, że w momencie czasowym $t_1 = 0$ wszystkie jednostki znajdują się w stanie bezczynności i nie posiadają żadnych zaległości w wykonywaniu zadań. Ponadto w momencie czasowym t_1 gotowy jest pełen pakiet zadań zgodny z charakterystyką przedstawioną w tabeli 7.2, a planista rozpoczyna proces generowania harmonogramu. Kolejne momenty czasowe wyznaczane są na podstawie decyzji agenta w oparciu o wartości parametrów θ_1 oraz θ_2. Liczba zadań w pakiecie jest wprost proporcjonalna do liczby dostępnych (czyli w stanie bezczynności) jednostek obliczeniowych, a więc dana wzorem:

$$n = \Omega_1 \cdot n_{max}, \tag{7.6}$$

gdzie n_{max} to liczba zadań z charakterystyki pakietu.

[2]Poprzez dostępną moc obliczeniową należy rozumieć moc jednostek obliczeniowych w stanie bezczynności, natomiast moment czasowy określany jest na podstawie wartości parametru θ_1.

Tablica 7.7: Charakterystyki scenariuszy dla agenta typu Ag2.

Ω_1:	25%, 50%, 75%, 100%
Ω_2:	0%, 5%, 10%
Jednostki obliczeniowe:	Zestaw I i II
Zadania:	Pakiet I i II

Tablica 7.8: Wartości parametru θ_1.

Wartość parametru Ω_1	Zestaw I	Zestaw II
25%	10	50
50%	20	100
75%	30	150
100%	40	200

Doświadczenia zostały przeprowadzone w oparciu o czterdzieści scenariuszy różniących się wartościami parametrów θ_1, θ_2, Ω_1 i Ω_2, pakietami zadań oraz zestawami jednostek obliczeniowych. Każda sekwencja składała się z czterech procesów harmonogramowania. Charakterystyki scenariuszy zostały przedstawione w tabelach 7.7 oraz 7.8.

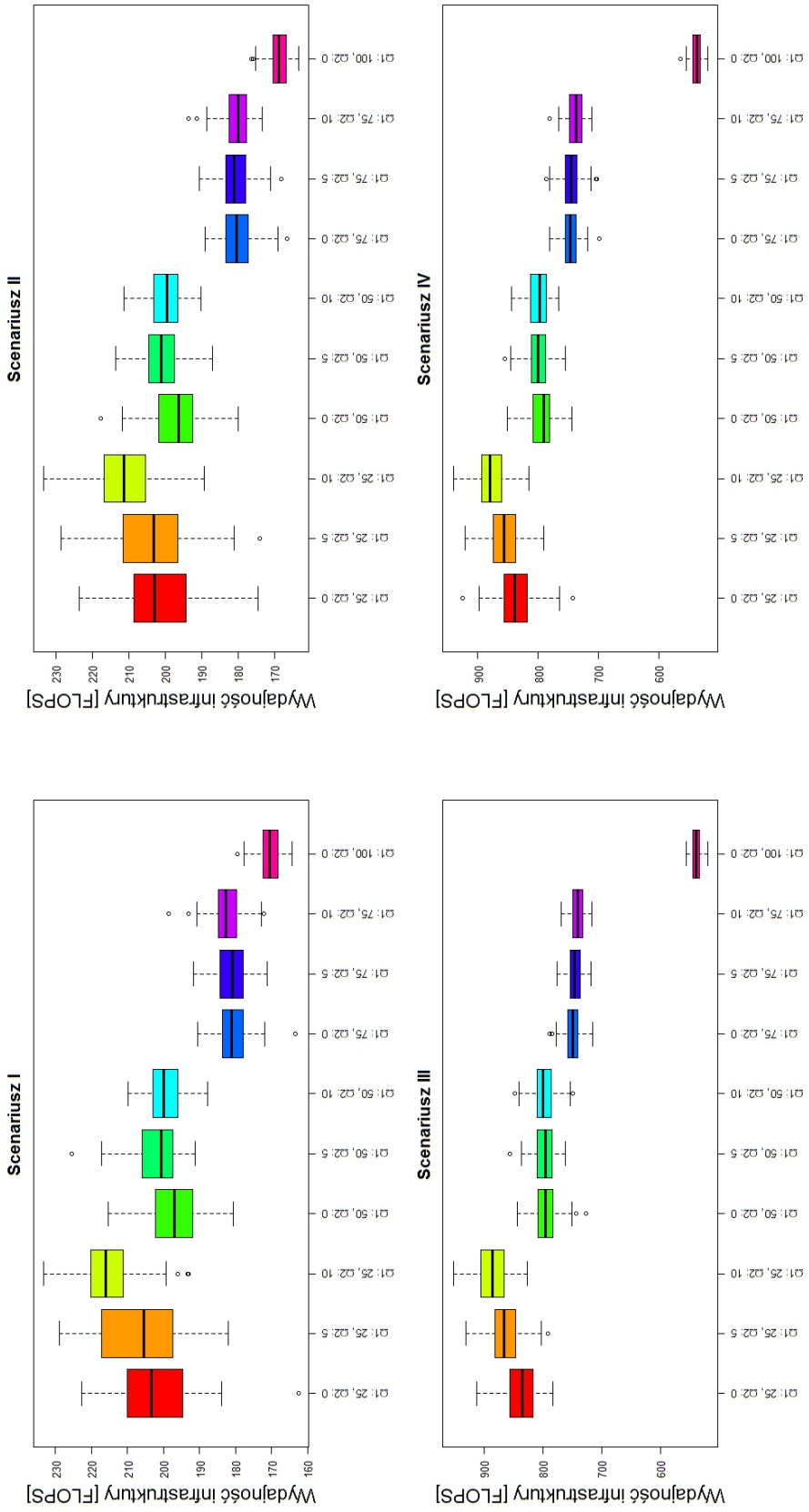
Wartość parametru θ_2 została odpowiednio dostosowana do rozważanego zestawu jednostek obliczeniowych. W przypadku gdy parametr θ_2 jest równy liczbie wszystkich jednostek w środowisku, wówczas mamy do czynienia z przetwarzaniem pakietów w sposób sekwencyjny, jeden po drugim. Wartość parametru θ_2 została wyznaczona w oparciu o wartości parametru Ω_2 oraz aktualnego średniego obciążenia 1 GFLOPS mocy obliczeniowej, stąd parametr ten nie został wyznaczony *a priori* i uwzględniony w tabeli. Każdy pomiar został powtórzony 100-krotnie. Wyniki przedstawiono na rysunkach 7.13 i 7.14. Każdy scenariusz został przetestowany w oparciu o dziesięć kombinacji ustawień parametrów Ω_1 i Ω_2 (por. tab. 7.7). Parametry te zostały umieszczone pod każdym z wykresów. Ustawienie $\Omega_1 = 100$ równoważne jest z harmonogramowaniem nienadzorowanym przez agenty typu Ag1 i Ag2.

W pierwszej kolejności porównano wygenerowane harmonogramy pod względem liczby operacji zmiennoprzecinkowych wykonanych w trakcie jednej sekundy w ramach całej dostępnej infrastruktury, czyli określono średnią wydajność całej infrastruktury. Celem zastosowanych strategii był dobór takich momentów czasowych, które pozwolą w lepszy sposób wykorzystać dostępną infrastrukturę.

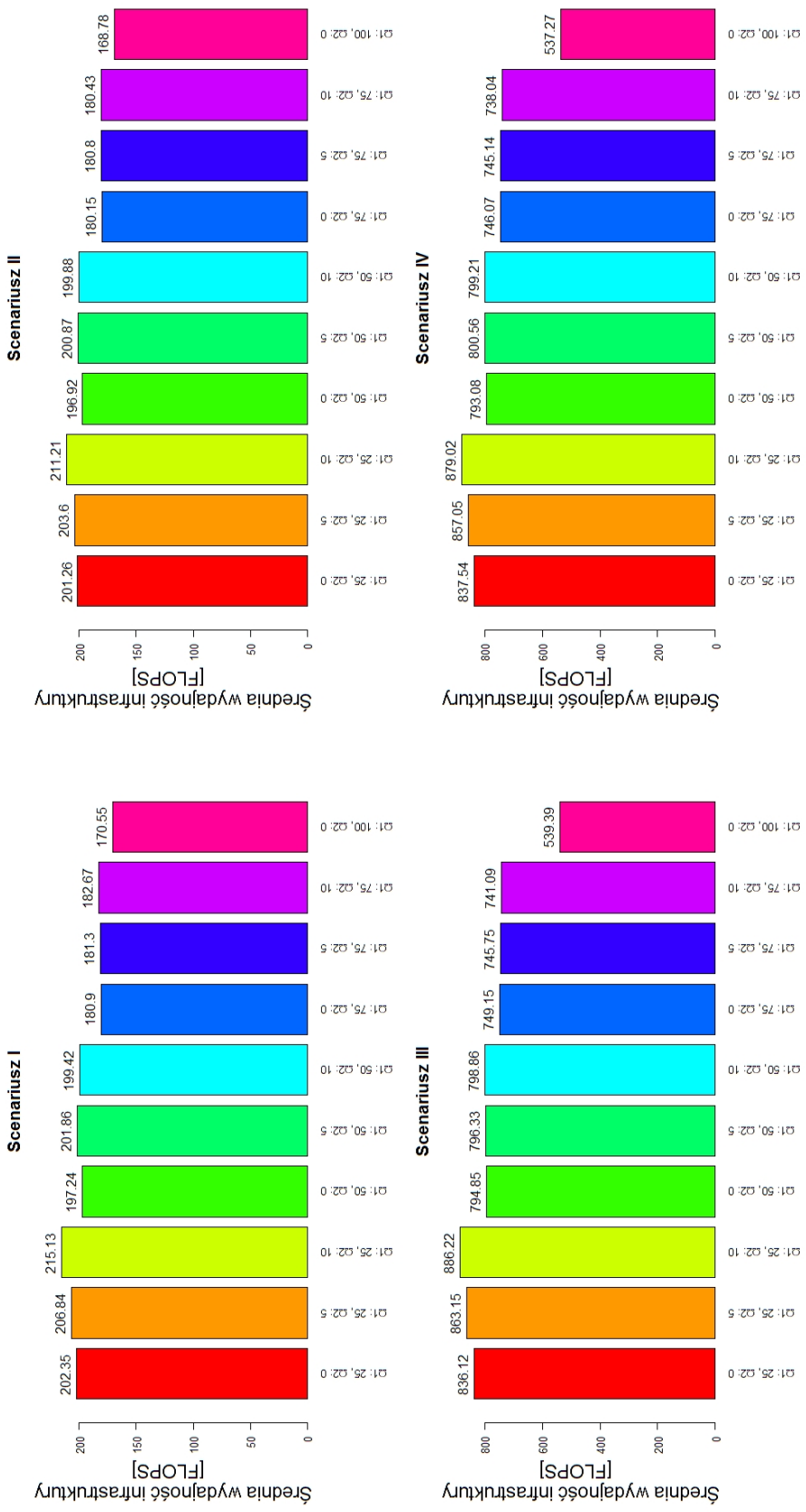
Najlepsze rezultaty pod względem średniej liczby przetworzonych operacji zmienno-przecinkowych w trakcie jednej sekundy dla każdego ze scenariuszy uzyskano dla strategii trzeciej, w której kolejny proces harmonogramowania rozpoczynano w momencie, gdy co najmniej 25% jednostek obliczeniowych było w stanie bezczynności oraz zakładano możliwość oczekiwania na poziomie 10% średniego obciążenia 1 GFLOPS mocy obliczeniowej (por. wzór 7.5). Oczekiwanie na dodatkowe jednostki (względem pierwszej strategii) pozwoliło na poprawę wydajności nawet o 6,32% (dla scenariusza I), natomiast w porównaniu do strategii nienadzorowanej, poprawa wynosiła: dla scenariusza I – 26,14%, dla scenariusza II – 25,14%, dla scenariusza III – 64,3%, dla scenariusza IV – 63,61%. Wyraźnie widać znaczącą poprawę w przypadku środowisk dużej skali.

Zauważyć należy również większy rozrzut wartości dla przypadków wspieranych przez agenty. Z jednej strony świadczy to o mniejszej przewidywalności generowanych harmonogramów, z drugiej natomiast może świadczyć o szansie na wypracowanie rozwiązań pozwalających na uzyskiwanie lepszych rezultatów.

Przedstawione wyniki wskazują również na lepsze wykorzystanie mocy obliczeniowej w przypadku mniejszych infrastruktur - środowiska składające się z 40 jednostek obliczeniowych charakteryzowały się około 20% lepszą wydajnością.



Rysunek 7.13: Wykresy pudełkowe prezentujące wydajność infrastruktury dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego.

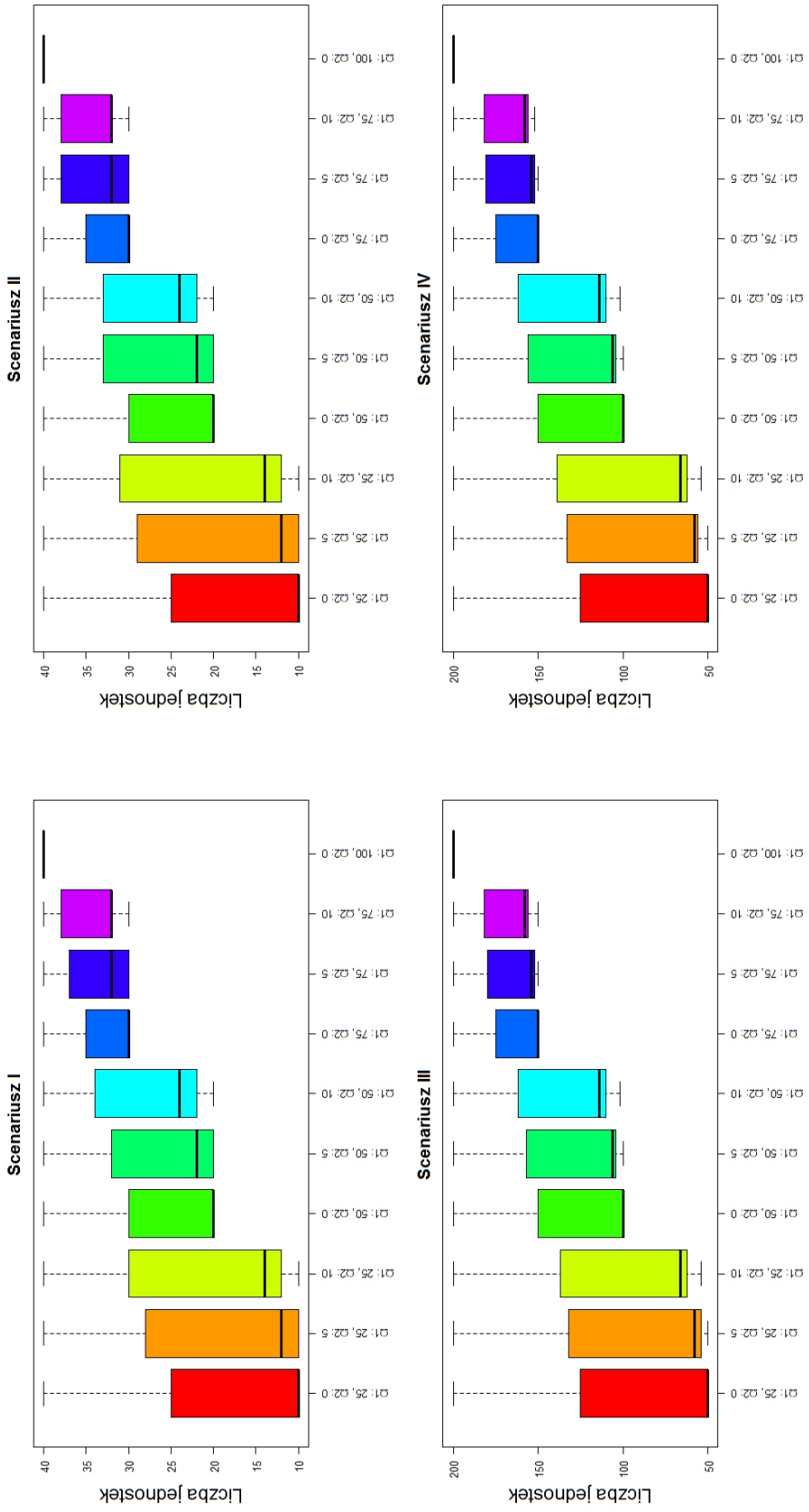


Rysunek 7.14: Średnia wydajność infrastruktury dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego.

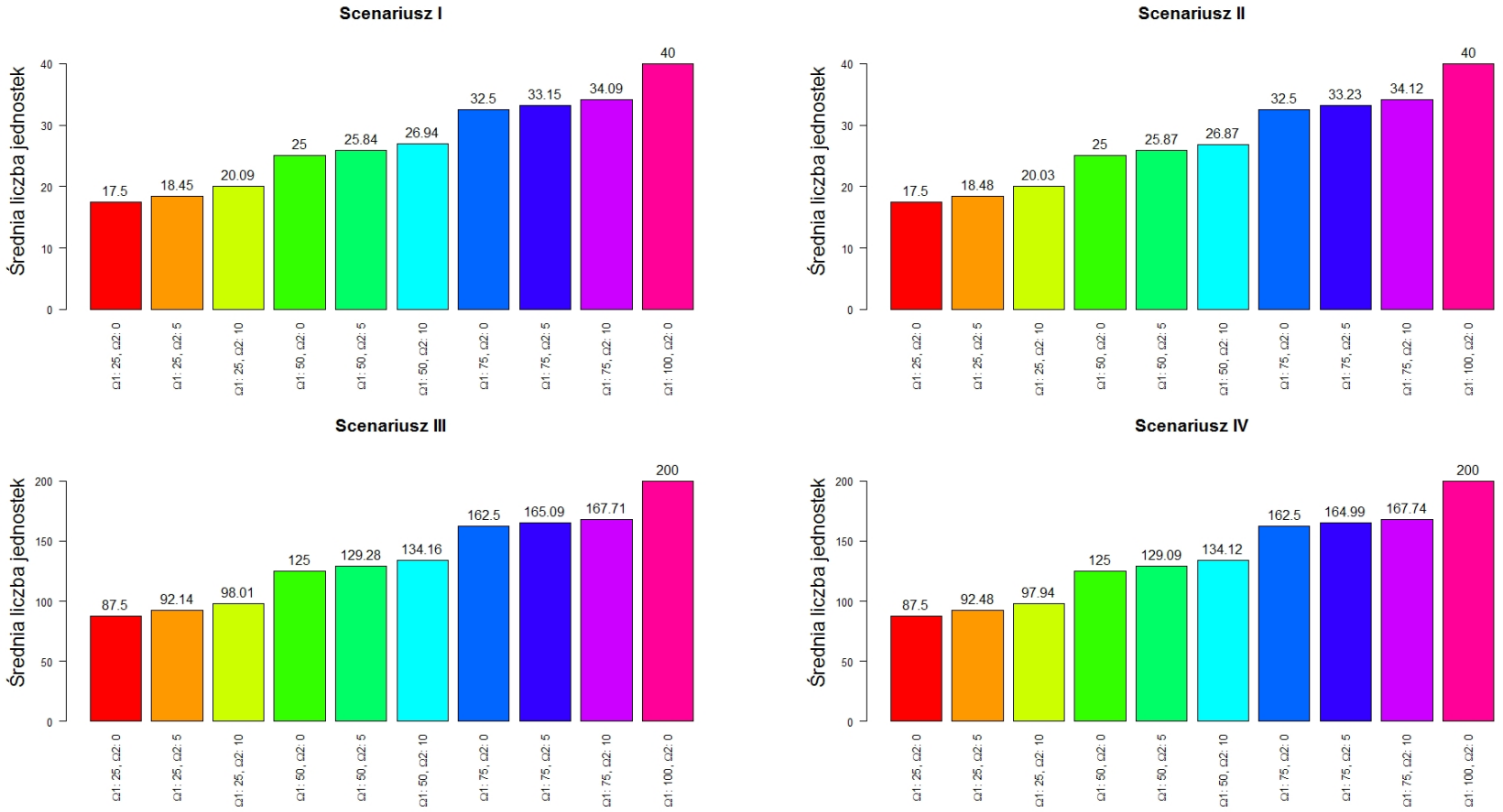
Efektywność zaproponowanego modelu w sposób istotny zależy od liczby dostępnych jednostek, które biorą udział w procesie realizacji harmonogramu. Liczba ta nie zależy jedynie od parametru θ_1, ale również czasu od jaki, jesteśmy w stanie odczekać na dodatkowe jednostki. W ramach przeprowadzonych badań czas ten został uzależniony zarówno od zapotrzebowania obliczeniowego pakietu, jak i mocy obliczeniowej aktualnie wolnych jednostek (por. wzór 7.5). W związku z tym dokładne wartości parametrów modelu mogły być wyznaczone niedługo przed rozpoczęciem procesu szeregowania.

Rysunki 7.15 i 7.16 przedstawiają średnią liczbę jednostek zaangażowanych w proces realizacji harmonogramu. Wykresy te uwzględniają również pierwszy z sekwencji czterech procesów harmonogramowania, kiedy liczba dostępnych jednostek jest równa m (odpowiednio 40 dla scenariuszy I i II, oraz 200 dla III i IV).

Pierwsza, czwarta, siódma i dziesiąta strategia zdefiniowane są jedynie przez wartość θ_1. Pozostałe rozwiązania wykorzystują strategię oczekiwania na dodatkowe jednostki obliczeniowe. W przypadku scenariuszy I i II, oczekiwanie na dodatkowe jednostki zwiększyło średnio liczbę jednostek od 0.65 do 2.59. Największy wzrost odnotowano dla strategii drugiej (Ω_1: 25, Ω_2: 10). Dla scenariuszy III i IV, liczba jednostek wzrosła średnio od 2.49 do 10.51. Ponownie największy przyrost jednostek zanotowano dla strategii drugiej.



Rysunek 7.15: Wykresy pudełkowe prezentujące liczbę jednostek biorących udział w sekwencji procesów harmonogramowania dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego.

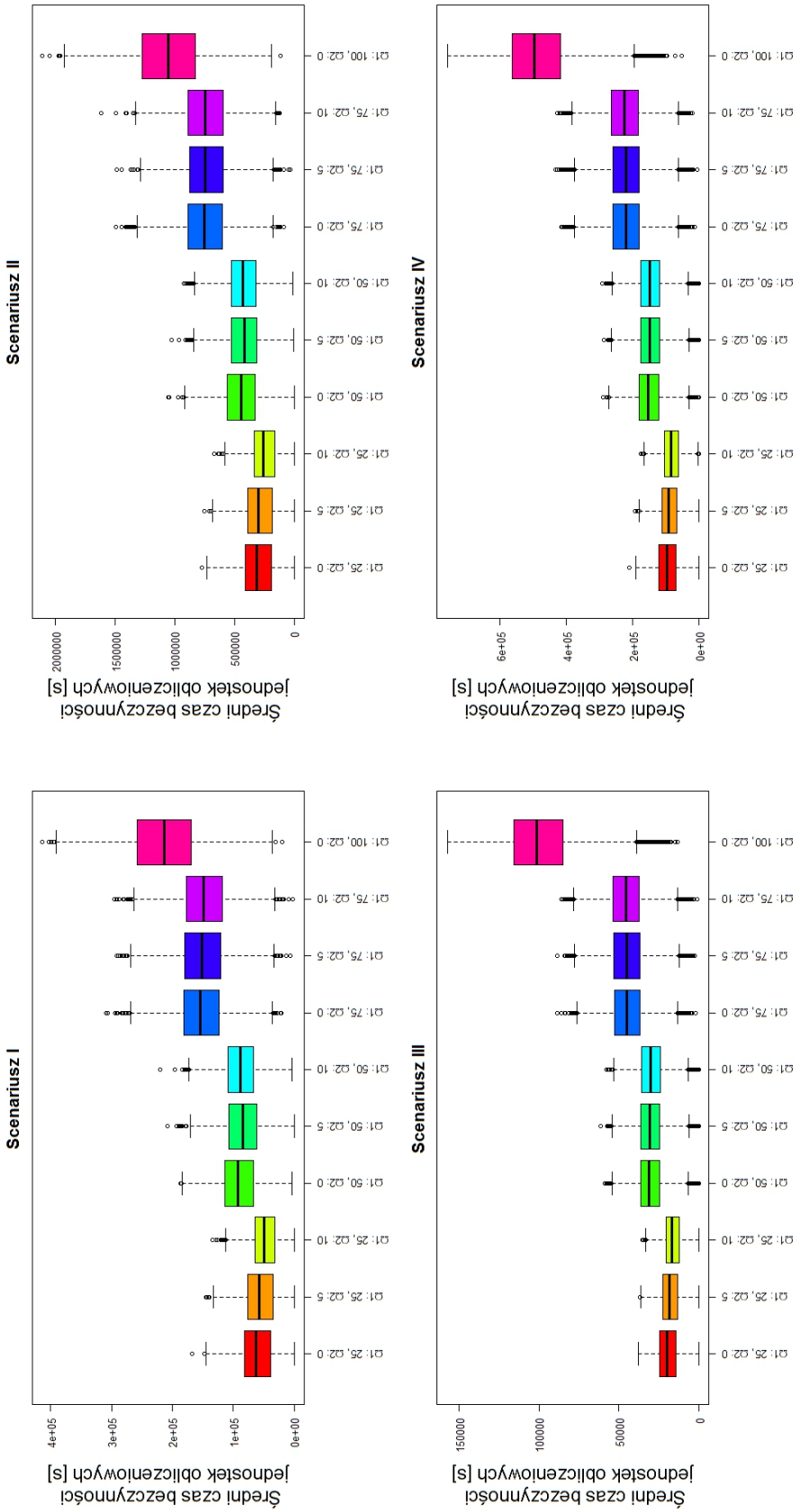


Rysunek 7.16: Średnia liczba jednostek biorących udział w sekwencji procesów harmonogramowania dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego.

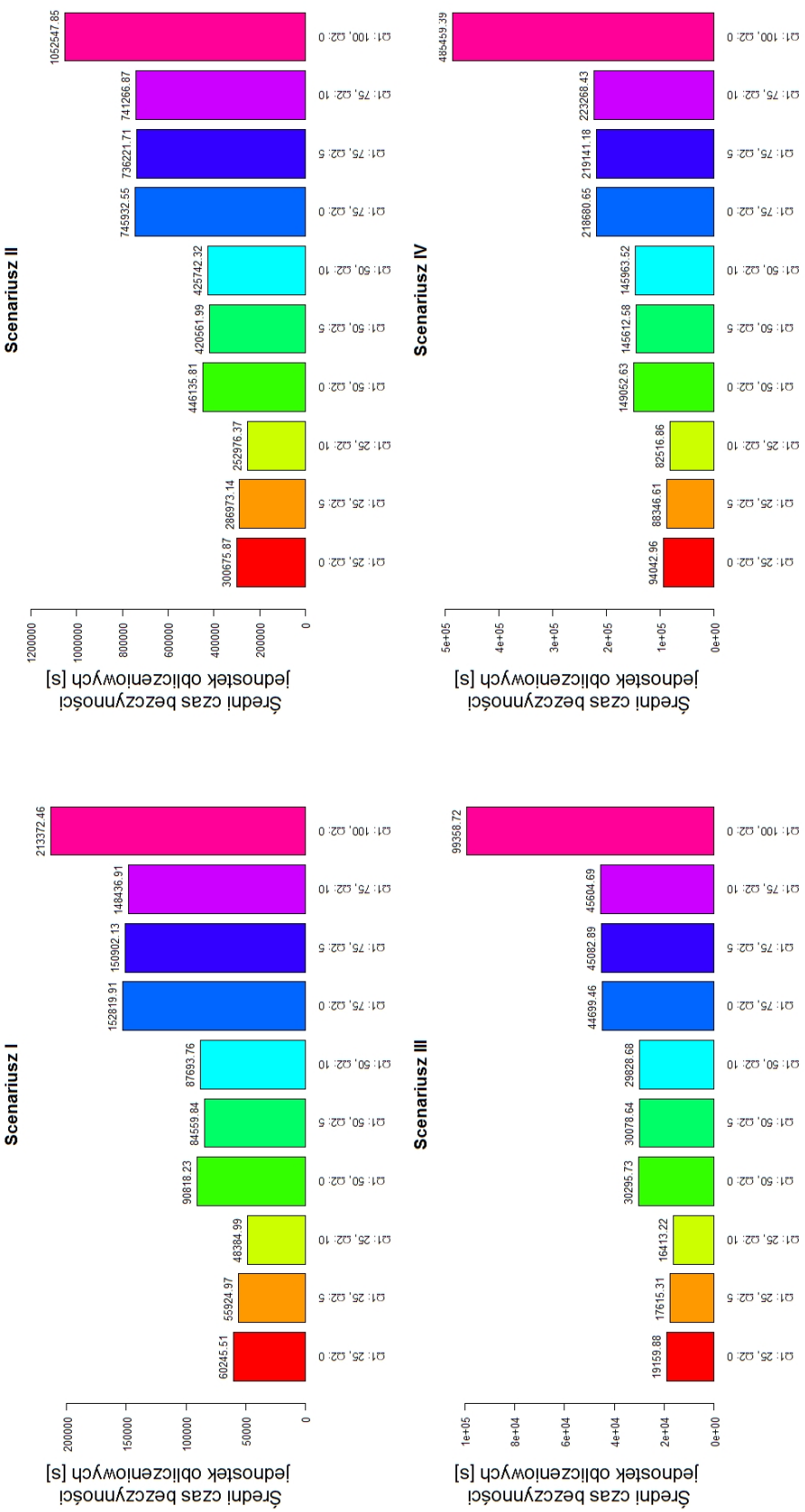
Ostatnie trzy rysunki (7.17, 7.18, 7.19) przedstawiają średnie czasy bezczynności jednostek obliczeniowych. Rys. 7.17 przedstawia rozkład wartości bezczynności pojedynczych jednostek. Najwyższymi wartościami charakteryzuje się strategia nienadzorowana, natomiast dla wszystkich scenariuszy strategia trzecia uzyskuje najlepsze rezultaty rozłożone na najmniejszym zakresie wartości. Strategia trzecia najbardziej minimalizuje czas bezczynności jednostek.

Rysunek 7.18 przedstawia wartości uśrednione czasów bezczynności jednostek. Ponownie można zauważyć, że zastosowane strategie najlepsze rezultaty uzyskują dla środowisk dużej skali (scenariusz III i IV). Jednak najistotniejsze rezultaty w kontekście czasów bezczynności zostały zaprezentowane na rysunku 7.19. Przedstawia on czasy bezczynności względem czasu wykonania wszystkich zadań. Wykres ten obrazuje właściwy zysk uzyskany dzięki realizacji strategii agentów typu Ag1 i Ag2. Dla strategii I i II średnio każda z jednostek była bezczynna przez ok. 40% czasu realizacji harmonogramów generowanych bez wsparcia agentów. Najlepsza ze strategii minimalizowała ten czas o około 13 – 14%, pozostałe od około 2,6% do 11%.

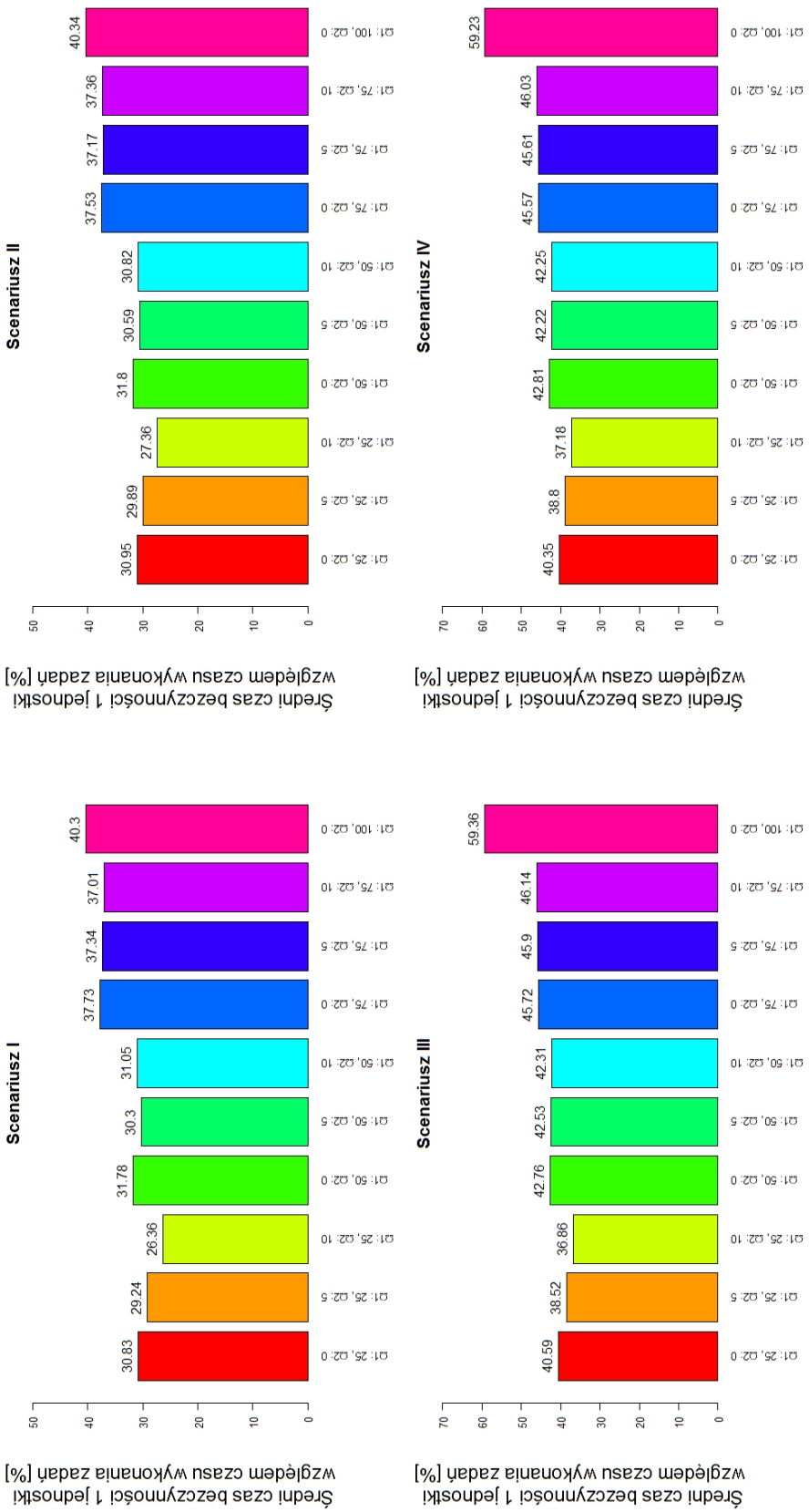
W przypadku scenariuszy III i IV zysk wynikający z zastosowania nadzoru agentowego był jeszcze większy. W przypadku strategii nienadzorowanej średni czas bezczynności osiągał blisko 60% czasu realizacji sekwencji harmonogramów. Podobnie jak w poprzednich scenariuszach, najlepsze rezultaty osiągnięto dla strategii drugiej (Ω_1: 25, Ω_2: 10) – czas bezczynności równy 36,86% i 37,18% odpowiednio dla strategii III i IV. Osiągnięto w ten sposób poprawę na poziomie 22 – 22,5%. Pozostałe strategie pozwoliły na zmniejszenie średniego czasu bezczynności pojedynczej jednostki od 13,2% do 20,84%.



Rysunek 7.17: Wykresy pudełkowe średnich czasów bezczynności jednostek obliczeniowych dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego.



Rysunek 7.18: Średnie czasy bezczynności jednostek obliczeniowych dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego.



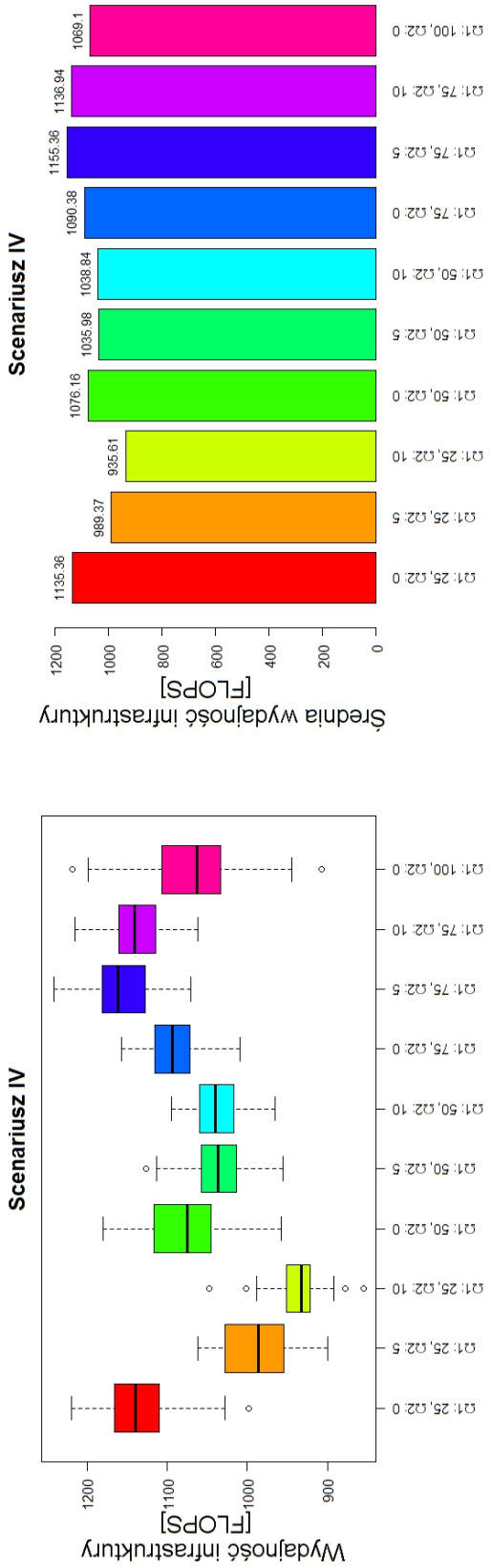
Rysunek 7.19: Średnie czasy bezczynności jednostek obliczeniowych względem czasu wykonania wszystkich pakietów zadań dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego.

Zaprezentowany model, w przeciwieństwie do poprzedniego, jest niezależny od reprezentacji populacji czy nawet metody generowania rozwiązań. Zastosowanie strategii realizowanych przez agenty Ag1 i Ag2 daje dobre rezultaty w przypadku generowania harmonogramów charakteryzujących się wysokimi czasami bezczynności jednostek obliczeniowych. Wstępne testy przeprowadzone ze współudziałem wsparcia ze strony agenta Ag0 (dla najlepszej strategii, tj. D60) nie wykazały znaczącego wpływu działania niniejszego modelu. Spowodowane jest to niską wartością czasów bezczynności jednostek obliczeniowych dla strategii D60.

Wyższe wartości czasów bezczynności jednostek obliczeniowych mogą występować w przypadku, gdy w puli zadań znajdować się będzie odpowiednia (niewielka) liczba zadań charakteryzująca się niewspółmiernie dużym zapotrzebowaniem obliczeniowym. W celu dokonania symulacji takiego przykładu, każde co pięćsetne zadanie zostało powiększone stukrotnie (zarówno zapotrzebowanie obliczeniowe sekwencyjne jak i równoległe). Testy zostały przeprowadzone w ramach strategii D60 agenta typu Ag0. Wpływ działania strategii realizowanych przez agenty Ag1 i Ag2 możliwy jest do zaobserwowania w przypadku scenariuszy, w których infrastruktura środowiska składa się z 200 jednostek obliczeniowych - w szczególności dla scenariusza IV charakteryzującego się dużą liczbą przetwarzanych zadań.

Rysunek 7.20 przedstawia zdolność obliczeniową infrastruktury na jedną sekundę uzyskaną w trakcie wykonywania wszystkich pakietów zadań w ramach scenariusza IV. Najlepsze rezultaty uzyskano dla strategii ósmej, w której kolejny proces harmonogramowania rozpoczynano w momencie, gdy co najmniej 75% jednostek obliczeniowych było w stanie bezczynności oraz zakładano możliwość oczekiwania na poziomie 5% średniego obciążenia 1 GFLOPS mocy obliczeniowej. Dla takiego rozkładu charakterystyk zadań najgorsze rezultaty uzyskano dla strategii trzeciej (nie uwzględniając ostatniej strategii, która równoważna jest z brakiem działań agentów), która poprzednim przykładzie generowała najlepsze rezultaty.

Zastosowanie wsparcia ze strony agentów typu Ag1 i Ag2 w przypadku scenariusza IV, w którym liczba zadań jest pięciokrotnie większa, pozwoliło na wyraźny wzrost wydajności obliczeniowej infrastruktury w porównaniu do wszystkich pozostałych eksperymentów.

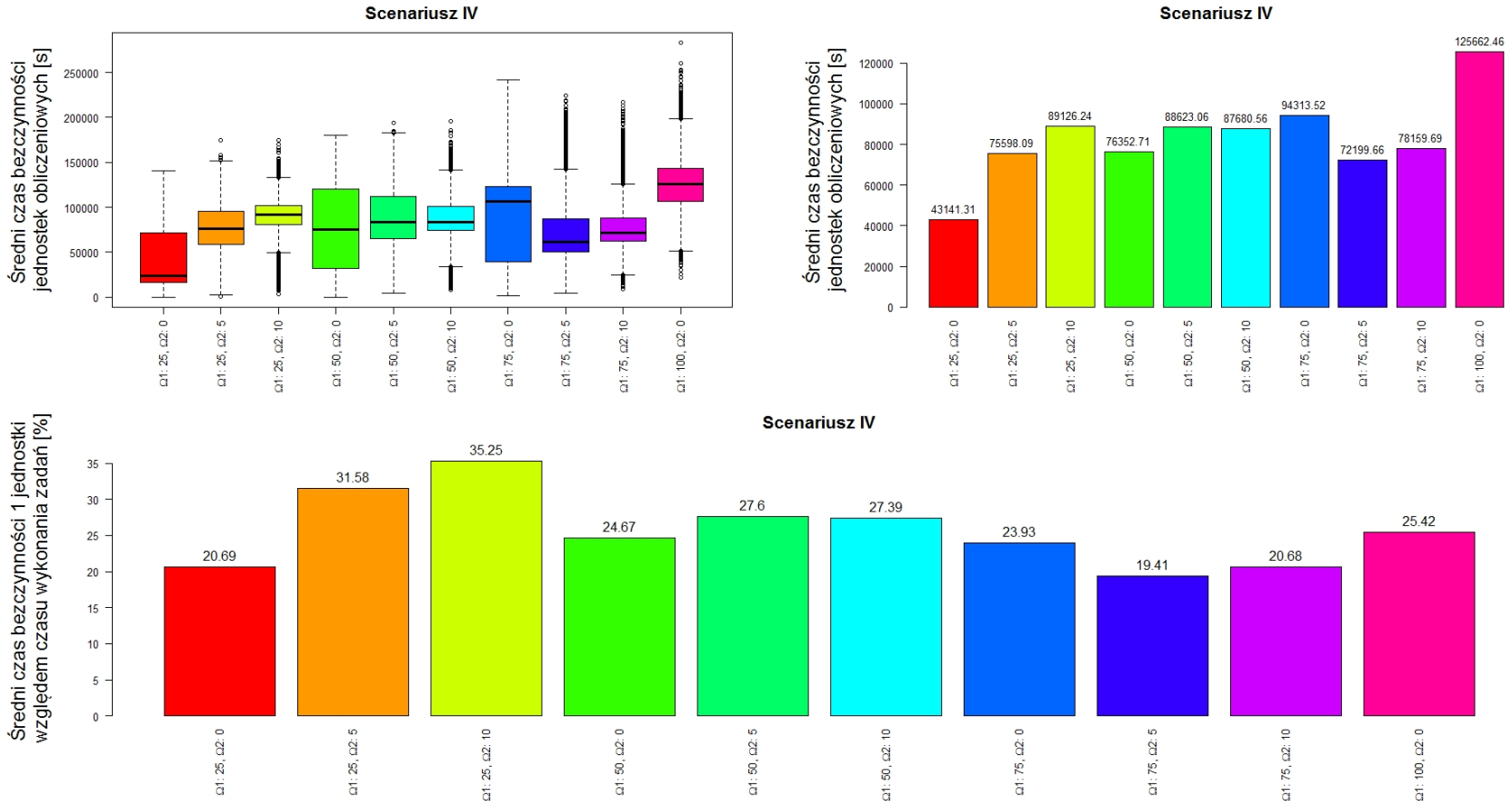


Rysunek 7.20: Wykresy prezentujące wydajność infrastruktury dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego dla scenariusza IV w przykładzie z zaburzonym rozkładem charakterystyk zadań.

Inaczej przedstawiają się pomiary czasów bezczynności jednostek - przedstawione na rys. 7.21 (dwa pierwsze wykresy). W przypadku scenariusza IV różnice pomiędzy poszczególnymi przypadkami testowymi są nawet dwukrotne. Ponadto można zaobserwować duży rozrzut wartości dla powtórzeń eksperymentów. Najlepsze rezultaty uzyskano dla strategii pierwszej.

Ostatni z wykresów - rys. 7.21 (wykres dolny) - przedstawia czas bezczynności w odniesieniu do całkowitego czasu realizacji wszystkich pakietów. Najlepsze rezultaty uzyskano dla ósmej strategii agentów Ag1 i Ag2. Średnio każda jednostka była bezczynna przez 19,41% czasu wykonywania obliczeń.

Uzyskane wyniki dowodzą, że odpowiedni dobór momentu rozpoczęcia procesu harmonogramowania i realizacji zadań ma istotne znaczenie w przypadku przetwarzania pakietowego. Klasyczne podejście, które zakłada zakończenie wykonywania pełnego pakietu zadań przed rozpoczęciem przetwarzania kolejnego może być nieefektywne. Wsparcie agentowe pozwala znacznie ograniczyć czas bezczynności infrastruktury obliczeniowej.



Rysunek 7.21: Wykresy prezentujące średni czas bezczynności jednostek obliczeniowych dla każdej ze strategii agenta typu Ag2 oraz harmonogramowania bez wsparcia agentowego dla scenariusza IV w przykładzie z zaburzonym rozkładem charakterystyk zadań.

7.4. Agenty typu Ag3 i Ag4

Zastosowanie SSN w ramach aktywności agenta Ag4 miało na celu umożliwienie przewidywania rzeczywistych opóźnień, jakie generowane są poprzez poszczególne jednostki obliczeniowe podczas wykonywania zadań. Agent ten otrzymuje wiedzę o wartościach parametru $wl_{i,j}$ (por. 6.37) i oczekiwanym czasie wykonania dla każdego z zadań oraz, dzięki mechanizmom monitoringu, informację o rzeczywistym czasie realizacji zadania. Na tej podstawie, agent może uczyć sieć neuronową predykcji równowartości opóźnienia w postaci liczby operacji, a następnie oszacować średnie opóźnienie w sekundach jakie może wystąpić dla zadań o określonym nakładzie obliczeniowym.

Wytrenowane sieci mogą być wykorzystane do korekcji oczekiwanych czasów wykonania zadań określonych przez macierz ETC. W tym celu agent Ag3 przesyła wszystkim agentom typu Ag4 charakterystyki całego pakietu zadań. Każdy z agentów Ag4 dokonuje oszacowania ewentualnych opóźnień dla poszczególnych zadań i odsyła informację zwrotną do agenta Ag3. Na podstawie zebranych danych agent ten formułuje macierz poprawek w stosunku do macierzy ETC.

W pracy przedstawiono wyniki testów dla trzech losowo wybranych jednostek obliczeniowych (kolejność zgodna z tabelą 7.9 – rosnąco, według zdolności obliczeniowych).

Opóźnienia wybranych jednostek generowano zgodnie z następującymi scenariuszami:

1. Jednostka nr 40 generowała średnie opóźnienie wynoszące 0 GFLO, z odchyleniem standardowym wynoszącym 2% liczby operacji przetwarzanego zadania – parametr ten był stały dla wszystkich zadań.

2. Jednostka nr 19 generowała średnie opóźnienie wynoszące 0 GFLO, z odchyleniem standardowym wynoszącym 1% liczby operacji przetwarzanego zadania dla zadań mniejszych (tj. poniżej 2000 GFLOPS) oraz z odchyleniem standardowym 10% dla zadań większych (tj. powyżej 2000 GFLOPS).

3. Jednostka nr 33 generowała średnie opóźnienie stałe równoważne wykonaniu 1000 GFLOPS (zgodnie ze zdolnością obliczeniową jednostki), z odchyleniem standardowym

Tablica 7.9: Testowane jednostki obliczeniowe oraz symulowane opóźnienia.

numer jednostki	*cc*	*cn*	$\mathcal{N}(0;\,\sigma)$	wejście SSN	wyjście SSN
40	2,30	4	$\mathcal{N}(0;\,0,02 * wl_{i,j})$	$[wl^{ann}_{40,j}]_{j=1,\ldots,20000}$	o_j
19	4,92	2	$\mathcal{N}(0;\,0,1 * wl_{i,j})$ lub $\mathcal{N}(0;\,0,01 * wl_{i,j})$	$[wl^{ann}_{19,j}]_{j=1,\ldots,20000}$	o_j
33	9,03	16	$\mathcal{N}(1000;\,0,1 * wl_{i,j})$	$[wl^{ann}_{33,j}]_{j=1,\ldots,20000}$	o_j

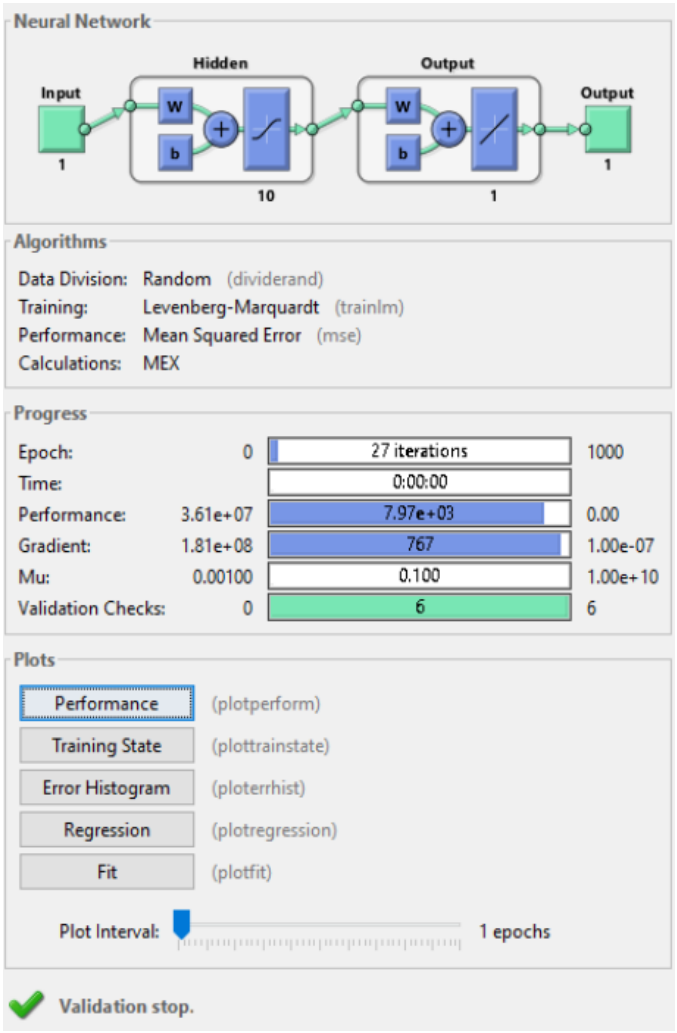
10% liczby operacji przetwarzanego zadania – parametr ten był stały dla wszystkich zadań.

Estymowane opóźnienia zależne były od wielkości zadania (określonej przez zapotrzebowanie obliczeniowe) oraz zdolności obliczeniowej danej jednostki. Wybrane scenariusze testowe zostały dobrane tak, aby odzwierciedlić narzuty związane m.in. z komunikacją pomiędzy procesami przetwarzanymi na różnych rdzeniach w przypadku wykonywania zadań wielowątkowych. Im większą zdolnością obliczeniową charakteryzowała się jednostka, tym większe generowała opóźnienia (w scenariuszu 2 również zależnie od wielkości zadania). Jednostka charakteryzująca się największą mocą obliczeniową z założenia generowała stały błąd, odzwierciedlając sytuację, która może wynikać z nieprawidłowo przeprowadzonej procedury określenia jej zdolności obliczeniowej.

Testy wykonano dla wszystkich zadań z pojedynczego pakietu, tzn. dla 20 000 zadań, które podzielono na zbiór uczący (70% pakietu, 14 000 zadań), zbiór walidujący (15% pakietu, 3 000 zadań) oraz zbiór testowy (15% pakietu, 3 000 zadań). Zaproponowano sieć dwuwarstwową o architekturze 1-10-1 (zbudowaną z jednej warstwy ukrytej), zaprezentowaną na rysunku 7.22.

Uczenie SSN dla jednostki nr 40 przeprowadzono standardową wersją algorytmu Levenberg -– Marquardt [51]. Otrzymane wartości błędów uczenia – MSEL (6.49), walidacji – MSEV (6.51) oraz testowania – MSET (6.50) przedstawione zostały na rysunku 7.23 i wyniosły odpowiednio (w zaokrągleniu) 7969, 8536 oraz 7765.

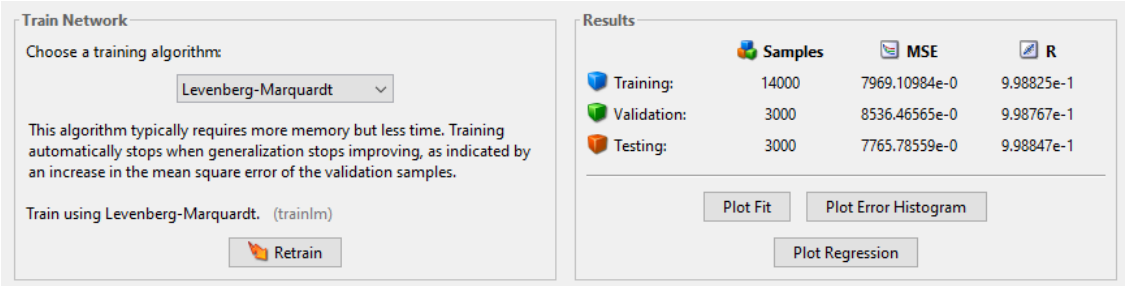
Otrzymane wartości współczynnika R^2 (6.47), zostały przedstawione na rysunku 7.24. Wykresy kolejno przedstawiają dopasowanie dla zbiorów: uczącego, walidującego, testowego oraz wszystkich danych. Wartości współczynnika są bliskie idealnemu dopasowaniu dla każ-



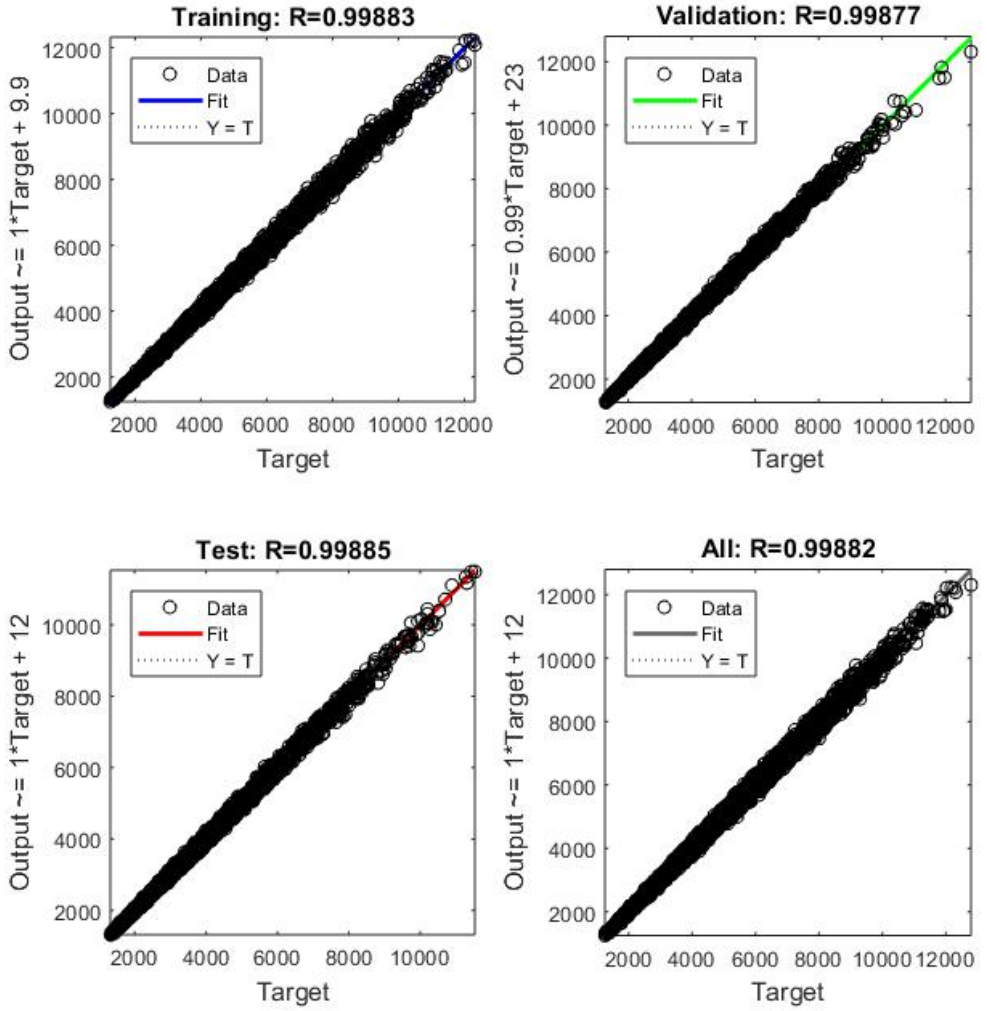
Rysunek 7.22: Charakterystyka SSN oraz kryterium zatrzymania algorytmu uczenia dla jednostki nr 40.

dego zestawu danych. Na tej podstawie możemy wywnioskować, że zamodelowana sieć w sposób bardzo wierny odwzorowała zasymulowany rozkład opóźnień (w przypadku przeprowadzonych testów – rozkład normalny).

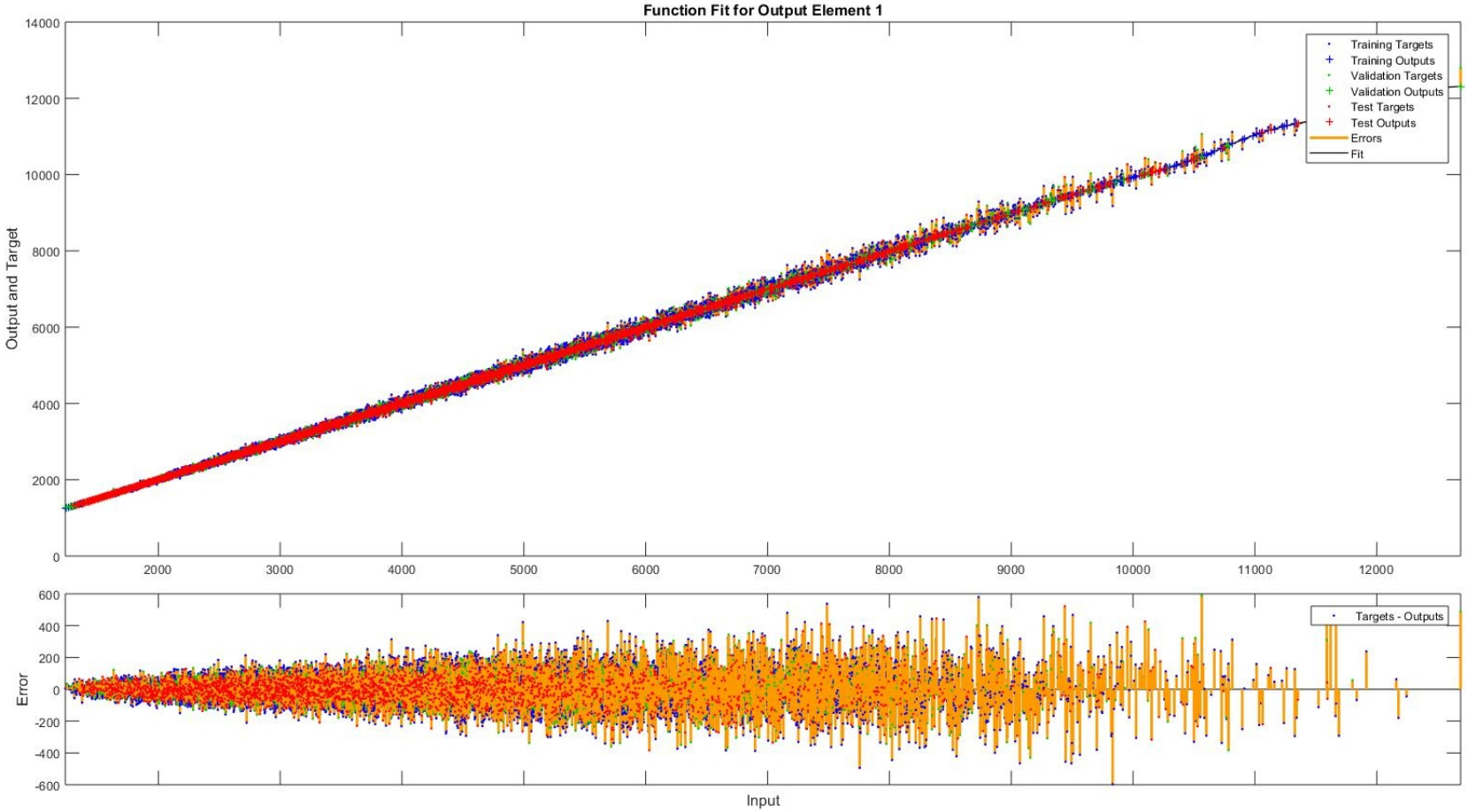
Na rysunku 7.25 zaznaczone zostały obciążenia poszczególnych zadań użyte podczas harmonogramowania oraz obciążenia rzeczywiste, które zostały wyznaczone przez SSN w procesie uczenia, walidacji i testowania. Otrzymano bardzo dobre dopasowanie dla zbioru uczącego i testowego. Tak przygotowana SSN służy agentowi AG4 do przewidywania, jaki będzie rzeczywisty czas wykonania zadania przez tę jednostkę obliczeniową.



Rysunek 7.23: Wyniki uczenia oraz testowania SSN dla jednostki nr 40.



Rysunek 7.24: Dopasowanie danych wyjściowych oraz pożądanych odpowiedzi sieci na wzorce uczące dla jednostki nr 40.



Rysunek 7.25: Wyniki modelowania przez SSN wartości rzeczywistego nakładu obliczeniowego zadań, zgodnego z charakterystyką opóźnień generowanych przez jednostkę nr 40.

Tablica 7.10: Testowane jednostki obliczeniowe oraz symulowane opóźnienia.

numer jednostki	R zbiór uczący	R walidujący	R testowy
40	0, 98825	0, 98770	0, 98047
19	0, 98759	0, 98200	0, 97023
33	0, 98632	0, 98159	0, 9658

Uczenie dla jednostki nr 19 oraz 33 również przeprowadzono algorytmem Levenberg–Marquardt, z wykorzystaniem takiej samej architektury sieci neuronowej. Otrzymane wartości współczynnika dopasowania R dla wszystkich testowanych jednostek przedstawiono w tabeli 7.10.

7.5. Obszary zastosowań zaproponowanych modeli monitoringu

Zaproponowane w niniejszej pracy modele nie są powiązane z rozwiązaniem żadnego konkretnego problemu inżynierskiego, stanowią natomiast formę wsparcia dla funkcjonowania środowisk, w ramach których takie problemy mogą być rozwiązywane. Do wspieranych rozproszonych systemów obliczeniowych można zaliczyć m.in. przetwarzanie sieciowe, przetwarzanie w chmurze czy systemy hybrydowe. Są to systemy zbudowane z infrastruktury heterogenicznej, różniącej się charakterystykami dostępnych zasobów. W niniejszej pracy, w ramach procesu harmonogramowania zadań, rozważano jedynie zasoby obliczeniowe scharakteryzowane przez zdolności obliczeniowe rdzeni jednostek. Zastosowane podejście pakietowe daje wiele możliwości odpowiedniego rozdysponowania zadań na dostępną infrastrukturę. W tym celu potrzebne są z góry wiadome szacunkowe wymagania potrzebne do realizacji zadań – w przypadku tej rozprawy, są to zapotrzebowania obliczeniowe zdefiniowane przez liczbę operacji zmiennoprzecinkowych składających się na realizację pojedynczego zadania.

Zaproponowany system monitoringu nie jest zależny jedynie od problemu harmonogramowania, ale również – częściowo – od algorytmu harmonogramowania. Planista generuje harmonogram korzystając z algorytmu ewolucyjnego. Podejście to jest oparte o model macierzy *ETC* oraz specjalną reprezentację populacji wzorowaną na reprezentacji typu Michigan.

Pierwszy z zaproponowanych agentów (agent typu Ag0) jest ściśle powiązany z reprezentacją populacji, natomiast działania agenta typu Ag4 wspierają model *ETC*. Nie oznacza to jednak, że częściowo model nie może być wykorzystany w innych wariantach modułu harmonogramowania. Strategie realizowane przez agenty typu Ag1 i Ag2 nie zależą od algorytmu generowania harmonogramów, a predykcja opóźnień przez agenty typu Ag3 może wpływać również na korektę charakterystyk jednostek w innych rozwiązaniach.

Stosując w pełni lub częściowo zaproponowane modele monitoringu można bezpośrednio wspierać pracę środowiska obliczeniowego (z korzyścią zarówno dla usługodawcy jak i usługobiorcy) poprzez lepsze wykorzystanie dostępnej infrastruktury. Korzyści te będą najbardziej odczuwalne w przypadku zadań, dla których istnieje możliwość oszacowania ich złożoności obliczeniowej (o czym szerzej wspomniano na początku podrozdziału). W oszacowaniu liczby operacji zmiennoprzecinkowych potrzebnych do wykonania zadania pomóc mogą dostępne narzędzia. Jednym z nich może być biblioteka LINPAKC, stworzona w latach 70. w celu rozwiązywania problemów algebry liniowej. Jej różne implementacje pozwalają zarówno mierzyć zdolność obliczeniową jednostki, jak i liczbę operacji składających się na zadanie. Jedna z takich implementacji dostępna jest w środowisku MATLAB[3]. Innym narzędziem pozwalającym na dokonanie podobnych pomiarów jest PAPI (Performance API). Pozwala ono na dostęp do liczników wydajności sprzętu dostępnych na większości nowoczesnych mikroprocesorach [113]. Innym, bardzo popularnym narzędziem pozwalającym na dokonywanie pomiarów związanych z wydajnością sprzętu komputerowego oraz liczbą operacji potrzebną do wykonania zadania, jest *perf*[4]. Jest to rozbudowane oprogramowanie dedykowane pod systemy Linux, składające się z kilku modułów. Jednym z modułów jest *perf stat*, który ma dostęp do liczników zdarzeń w rejestrach procesora i dostarcza funkcjonalności zbierania i analizowania danych dotyczących wydajności i zapotrzebowań obliczeniowych.

Powyższe rozwiązania wymagają wcześniejszego, monitorowanego wykonania zadania. Może to pozwolić na obliczenie liczby operacji dla zadań o mniejszej skali i na tej podstawie oszacowanie liczby operacji dla większych zadań. Podobne analizy zostały przedstawione

[3]https://uk.mathworks.com/company/newsletters/articles/matlab-incorporates-lapack.html
[4]http://www.brendangregg.com/perf.html

Tablica 7.11: Liczba operacji niezbędna do przeprowadzenia operacji zmiennoprzecinkowych dla popularnych obliczeń wektorowo–macierzowych, gdzie: $\alpha \in R$ – skalar, $a, b \in R^N$ – wektory, $A \in R^{N \times M}$, $C \in R^{N \times K}$ – macierze, $D \in R^{N \times N}$ – macierz diagonalna, $L \in R^{N \times N}$ – dolna macierz trójkątna, $P \in R^{N \times N}$ – macierz dodatnio określona, K, N, M – wymiary wektorów i macierzy.

Typ obliczeń	Liczba operacji (FLOPs)
Mnożenie wektora przez skalar: αa	N
Mnożenie macierzy przez skalar: αA	MN
Iloczyn skalarny wektorów: $a^T b$	$2N - 1$
Iloczyn diadyczny wektorów: ab^T	MN
Mnożenie macierzy przez wektor: Ab	$2MN - M$
Mnożenie macierzy przez macierz: AC	$2MNK - MK$
Rozkład Choleskiego: $P = LDL^T$	$N^3/3 + N^2 - 7/3N + 1$
Rozkład Choleskiego (wersja Gaxpy): $P = LL^T$	$N^3/3 + N^2/2 + N/6$
Macierz Gramma: $A^H A$	$MN^2 + MN - N^2/2 - N/2$

w pracy [112] na przykładzie rozmycia gaussowskiego.

W przypadku, gdy nie ma możliwości skorzystania z automatycznych narzędzi, a problem sprowadza się do obliczeń wektorowo–macierzowych, istnieje możliwość oszacowania liczby operacji na podstawie złożoności algorytmu. R. Hunger w pracy [52] dokonał oszacowania liczby operacji zmiennoprzecinkowych dla popularnych obliczeń wektorowo–macierzowych i opartych o nie algorytmów. Przykładowe działania i algorytmy zostały przedstawione w tabeli 7.11.

Zarówno informacje o złożoności algorytmów jak i przywołane narzędzia mogą pomóc w oszacowaniu liczby operacji w takich zagadnieniach jak metody symulacji zagadnień technicznych (np. metoda elementów skończonych), cyfrowe przetwarzanie obrazów, kryptografia, przetwarzanie dźwięków i wiele innych.

Jednym z klasycznych problemów algebry liniowej jest rozwiązywanie układów równań algebraicznych, które to znajduje zastosowanie w szeregu metod numerycznych, np. metodzie elementów skończonych czy metodzie różnic skończonych. Tego typu zadanie może być dobrym przykładem, w którego realizacji może pomóc zaproponowany w pracy system monitoringu. Często spotykanym algorytmem stosowanym do rozwiązywania niektórych układów równań liniowych jest metoda gradientu sprzężonego (ang. *conjugate gradient method, CG*). Pozwala ona rozwiązywać te układy, których macierz jest symetryczna i dodatnio określona.

Duże, rzadkie układy równań tego typu powstają podczas numerycznego rozwiązywania równań różniczkowych cząstkowych lub problemów optymalizacyjnych.

Metoda gradientu sprzężonego należy do metod iteracyjnych, w których to kolejne przybliżone rozwiązania wyznaczane są na podstawie poprzednich. Wykonywanie iteracji rozpoczyna się od przybliżenia początkowego x_0. Końcowy wynik zostaje osiągnięty poprzez zadaną *a priori* dokładność albo określoną liczbę iteracji. Korzyści wynikające z takiego podejścia szczególnie widoczne są dla problemów dużej skali, czyli takich, gdzie układy składają się z tysięcy, a nawet milionów równań. Algorytmy te często występują w parze z metodami przygotowania wstępnego (ang. *preconditioning*), które mogą znacznie podwyższyć szybkość procedur iteracyjnych [84].

Metoda gradientu sprzężonego jest najbardziej wydajną i najlepiej poznaną metodą przestrzeni Kryłowa, która w swoim założeniu odnosi się do układów $Ax = b$, gdzie A jest macierzą kwadratową stopnia n, która jest rzeczywista, symetryczna i dodatnio określona [64, 84]. Idea oraz algorytm metody gradientu sprzężonego zostały przedstawione w wielu pozycjach naukowych, m.in. [44, 46, 64].

W przypadku, gdy macierz A jest źle uwarunkowana, uzyskanie rozwiązania metodą iteracyjną może być bardzo czasochłonne. Rozwiązaniem może być wykorzystanie metod uwarunkowania wstępnego. Pozwalają one na przekształcenie macierzy A tak, aby polepszyć rozkład wartości własnych i obniżyć wskaźnik uwarunkowania. To natomiast ma bezpośredni wpływ na zbieżność metod iteracyjnych [46, 101].

Jedną z popularnych metod uwarunkowania wstępnego jest niepełną faktoryzacją Choleskiego (ang. *incomplete Cholesky factorization*). Ma ona zastosowanie dla macierzy, które są rzeczywiste, symetryczne i dodatnio określone. Macierz taka (A) ma dokładnie jeden rozkład na czynniki $A = LL^T$, gdzie L jest macierzą trójkątną dolną o elementach dodatnich na diagonali [64]. Rozkład taki można uzyskać wykorzystując algorytm dekompozycji (faktoryzacji) Choleskiego (ang. *Cholesky factorization*). Jednak w przypadku metod warunkowania wstępnego nie stosuje się pełnej faktoryzacji, a jedynie częściową - niepełną faktoryzację Choleskiego. Zastosowanie pełnej faktoryzacji Choleskiego oznaczałoby rozwią-

zanie zadanego układu równań. Algorytmy oraz opis niepełnej faktoryzacji Choleskiego zostały zaprezentowane m.in. w [44, 46, 108].

Omówiony wyżej problem rozwiązywania układów równań liniowych za pomocą metody gradientu sprzężonego z uwarunkowaniem wstępnym typu niepełnej faktoryzacji Choleskiego może być przykładem zadania scharakteryzowanego za pomocą liczby operacji zmiennoprzecinkowych potrzebnych do jego wykonania. W celu uzyskania charakterystyk takiego zadania dokonano implementacji wspomnianych algorytmów. Obliczenia wykonywane były w sposób sekwencyjny, na jednym rdzeniu procesora. W celu oszacowania liczby operacji posłużono się wcześniej wspominanym programem *perf* systemu Linux. Jednostka obliczeniowa, w ramach której wykonano zadanie wyposażona jest w procesor AMD Phenom(tm) II X4 945. Nowoczesne procesory posiadają sprzętowe liczniki, które są w stanie monitorować szereg procesów zachodzących w ich ramach (ang. *performance monitoring counter, PMC*). Jednym z takich liczników jest *RETIRED_X87_OPS*, w który wyposażone są m.in. procesory rodziny AMD64 Fam10h Shanghai. Rejestr ten zlicza poprawnie wykonane operacje zmiennoprzecinkowe [43].

W celu dokonania pomiaru liczby operacji potrzebnych do realizacji zadania, w pierwszej kolejności zidentyfikowano kod rejestru *RETIRED_X87_OPS – r1C0*. W tym celu posłużono się biblioteką *libpfm4* rozwijaną w ramach projektu *perfmon2* [7], którego celem jest wsparcie konfiguracji dla programu *perf*.

Przykładem testowym jest macierz o wymiarach 7400x7400, spełniająca wymagania narzucone przez metodę gradientu sprzężonego i algorytm faktoryzacji, a więc jest to macierz, która jest rzeczywista, symetryczna i dodatnio określona. Macierz cechuje się bardzo niskim wskaźnikiem gęstości równym 0,2% elementów niezerowych, a także bardzo wysokim wskaźnikiem uwarunkowania, wynoszącym 17787,5. Przykład ten został wygenerowany w środowisku MATLAB metodą generującą macierze Wathena.

Pomiaru dokonano za pomocą polecenia systemu Linux: *perf stat -e r1C0 ./PCG*, gdzie *r1C0* to kod licznika, a *./PCG* nazwa monitorowanego programu. Rozwiązanie przykładu testowego na jednym rdzeniu procesora zajęło 23 minuty i 37 sekund. Liczba wyko-

nanych operacji zmiennoprzecinkowych wyniosła 994 334 167 349 (w przybliżeniu 994,33 GFLO). Na podstawie liczby wykonanych operacji, a także czasu poświęconego za realizację zadania można określić zdolność obliczeniową pojedynczego rdzenia, która w przypadku testowanej jednostki wyniosła 0,70 GFLOPS.

Przedstawiony wyżej przykład obrazuje jedynie przykładowe zadanie mogące być częścią większego pakietu. Podejście polegające na określaniu przybliżonej liczby potrzebnych operacji szczególne korzyści niesie w przypadku zadań wielokrotnie powtarzanych lub takich, w których zachodzą stabilne relacje pomiędzy skalą rozwiązywanego problemu, a liczbą operacji.

7.6. Implementacja modeli i dane zbierane przez system monitoringu

Implementacja modelu została zrealizowana w ramach dedykowanego symulatora procesów harmonogramowania i wykonania zadań obliczeniowych. Całe oprogramowanie powstało w ramach niniejszej rozprawy oraz wcześniejszych badań, których rezultaty zostały przedstawione w pracach [47, 56, 57, 58]. Implementacja oparta jest na mechanizmach języka C++11, a także częściowo z użyciem framework'a FastFlow, biblioteki POSIX oraz funkcji środowiska MATLAB. Wersja symulatora wykorzystana w ramach niniejszej pracy implementuje system wieloagentowy oraz planistę opartego o podejście ewolucyjne, z funkcją optymalizacji ukierunkowaną na minimalizację czasu realizacji całego pakietu zadań.

Symulator powstał w środowisku Microsoft Visual Studio 2017, natomiast kod źródłowy programu został zoptymalizowany i skompilowany za pomocą kompilatora Microsoft C/C++ w wersji 19.10.25017 dla platform 64-bitowych.

W ramach działania systemu agentowego i przeprowadzanych symulacji generowane są obszerne statystyki, które częściowo zostały zaprezentowane w poprzednich podrozdziałach. W wyniku monitoringu zbierano szereg danych, takich jak:

- parametry systemu (liczba i charakterystyki jednostek obliczeniowych, parametry θ_1,

θ_2, κ, realizowane strategie, prawdopodobieństwo mutacji),

- generowane momenty czasowe,
- liczba i charakterystyki zadań obliczeniowych,
- wygenerowane ostateczne harmonogramy,
- czasy generowania harmonogramów,
- oczekiwane i rzeczywiste czasy wykonania zadań dla każdej z jednostek,
- zmiany oczekiwanych czasów realizacji harmonogramu na przestrzeni epok,
- liczba epok,
- numer epoki z najlepszym rozwiązaniem,
- czasy bezczynności jednostek,
- czasy zaległości w wykonywaniu zadań dla każdej jednostki,
- czasy realizacji sekwencji harmonogramów,
- momenty ograniczenia liczby par osobników podlegających reprodukcji,
- sumaryczna liczba wykonanych operacji zmiennoprzecinkowych zadań,
- wartości średnie czasów i epok,
- przewidywane opóźnienia w wykonywaniu zadań,
- wyniki uczenia sztucznych sieci neuronowych.

Rozdział 8

Wnioski i zakończenie

Głównym celem niniejszej pracy było opracowanie i implementacja innowacyjnych modeli inteligentnych systemów monitoringu procesów harmonogramowania niezależnych zadań w rozproszonych środowiskach obliczeniowych dużej skali. Zaproponowane modele oparte są o paradygmat wieloagentowy, natomiast monitorowany proces generowania harmonogramów realizowany jest w oparciu o podejście ewolucyjne.

W ramach rozprawy zaproponowano system wieloagentowy zbudowany z pięciu typów agentów. Cały system w kompleksowy sposób wspiera proces harmonogramowania, począwszy od bezpośredniej ingerencji w przebieg algorytmu ewolucyjnego, poprzez wsparcie w podejmowaniu decyzji o rozpoczęciu generowania harmonogramu, skończywszy na monitoringu wykonania zadań i korekcie oczekiwanych czasów realizacji harmonogramów.

Zadaniem pierwszego z typów agentów – Ag0 jest monitoring procesu ewolucji rozwiązania (harmonogramu) na przestrzeni epok oraz ingerencja w ten proces. Rola agenta sprowadza się do inteligentnego operatora selekcji, a zarazem realizatora strategii przeżywalności osobników. Przetestowano siedem strategii wsparcia harmonogramowania oraz harmonogramowanie niemonitorowane w ramach czterech scenariuszy. Najlepsze rezultaty otrzymano dla strategii D60, zgodnie z którą po określonej liczbie epok (w ramach niniejszej pracy 60 lub 300), w których nie otrzymano lepszego rozwiązania, agent ograniczał liczbę par krzyżowanych osobników. Zastosowanie tej strategii pozwoliło na uzyskanie harmonogramów charakteryzujących się niższym czasem wykonania całego pakietu zadań, co jest główną miarą jako-

ści dla problemu harmonogramowania pakietów zadań. Dodatkowo, wykazano, że wsparcie procesu szeregowania pozwoliło na uzyskanie harmonogramów charakteryzujących się lepszym zrównoważeniem obciążenia i krótszym czasem bezczynności jednostek obliczeniowych. Wszystkie z przetestowanych strategii pozwoliły na znaczne obniżenie czasów generowania rozwiązań.

Zadaniem kolejnych agentów systemu, tj. agentów typu Ag1 i Ag2, jest zarządzanie procesem harmonogramowania poprzez jego wywoływanie w odpowiednich momentach przy przetwarzaniu całych sekwencji pakietów zadań. Decyzja o rozpoczęciu nowego procesu harmonogramowania podejmowana jest na podstawie informacji o liczbie dostępnych jednostek obliczeniowych oraz szacowanym czasie dostępności nowych jednostek. Badania symulacyjne zostały przeprowadzone w oparciu o 40 scenariuszy i porównane z szeregowaniem niewspieranym agentowo. Testy przeprowadzono pod kątem średniej liczby przetworzonych operacji zmiennoprzecinkowych w trakcie jednej sekundy. Najlepsze rezultaty uzyskano dla strategii, w której kolejny proces harmonogramowania rozpoczynano w momencie, gdy co najmniej 25% jednostek obliczeniowych było w stanie bezczynności oraz zakładano możliwość oczekiwania na nowe jednostki na poziomie 10% średniego obciążenia 1 GFLOPS mocy obliczeniowej. Podobnie, najlepsze rezultaty pod kątem wykorzystania infrastruktury środowiska uzyskano dla tej strategii. Powyższe rezultaty uzyskano w przypadku, gdy proces harmonogramowania nie był wspierany przez agenta typu Ag0. Jednak dodatkowe badania pozwoliły na wykazanie pozytywnego wpływu łączenia strategii agenta Ag0 ze strategiami agentów Ag1 i Ag2 w przypadku zaburzenia rozkładu charakterystyk zadań dla największego z rozważanych scenariuszy. W zasymulowanym przykładzie najlepsze rezultaty uzyskała strategia, w której co najmniej 75% jednostek obliczeniowych było w stanie bezczynności oraz zakładano możliwość oczekiwania na nowe jednostki na poziomie 5% średniego obciążenia. Uzyskane wyniki zobrazowały jak duży wpływ na wydajność środowiska może mieć moment rozpoczęcia kolejnego procesu harmonogramowania zadań i ich wykonania, a także jak istotne w tym problemie mogą być charakterystyki pakietów zadań.

Monitoring opóźnień generowanych przez zadania i korekta oczekiwanych czasów ich wykonania realizowane są przez agenty typu Ag3 i Ag4. Agenty typu Ag3 wyposażone zo-

stały w niewielkie sztuczne sieci neuronowe, które na podstawie informacji o zapotrzebowaniu obliczeniowym zadania i terminowości wykonania przeszłych zadań były w stanie dokonać predykcji opóźnienia generowanego przez nowe zadanie. Głównym celem agenta Ag4 jest konstrukcja macierzy poprawek względem macierzy ETC. W ramach pracy przedstawiono wyniki dla trzech jednostek obliczeniowych różniących się rozkładem opóźnień. Dla wszystkich rozważanych scenariuszy uzyskano bardzo dobre dopasowania dla zbiorów uczącego i testowego.

Wyniki przeprowadzonych doświadczeń pozwalają na potwierdzenie postawionej tezy rozprawy doktorskiej, mówiącej, że możliwe jest usprawnienie procesu generowania harmonogramów niezależnych zadań obliczeniowych poprzez zastosowanie inteligentnych metod takich jak systemy wieloagentowe.

Wśród elementów nowatorskich rozprawy należy wymienić:

- opracowanie szczegółowego modelu inteligentnego systemu monitorującego i wspierającego procesy generowania harmonogramów w ujęciu wieloagentowym,
- opracowanie modelu dynamiki obliczeniowego środowiska rozproszonego ze szczególnym uwzględnieniem procesu harmonogramowania,
- zaproponowanie definicji nowej reprezentacji populacji dla algorytmu ewolucyjnego wykorzystanego w procesie harmonogramowania,
- rozszerzenie modeli zadań i jednostek obliczeniowych o aspekt wykonywania operacji w sposób równoległy oraz zaproponowanie profili dla modeli zadań i jednostek,
- zaproponowanie notacji dla definicji przeznaczenia systemów monitoringu rozproszonych środowisk obliczeniowych,
- rozszerzenie istniejących taksonomii monitoringu systemów rozproszonych o procesy harmonogramowania zadań.

Opracowane w ramach rozprawy modele zostały zaimplementowane w języku programowania C++11 oraz, częściowo, z wykorzystaniem środowiska MATLAB.

Bibliografia

[1] CloudHarmony. Dostępne w Internecie: https://cloudharmony.com.

[2] Geekbench 4 - Cross-Platform Benchmark. Dostępne w Internecie: https://www.geekbench.com.

[3] Linux man pages. Dostępne w Internecie: https://linux.die.net/man/.

[4] New Relic Digital Intelligence Platform. Dostępne w Internecie: https://newrelic.com.

[5] newbie: System Monitoring commands. Dostępne w Internecie: http://unixgeeks.org/security/newbie/unix/man9/misc3.html.

[6] PassMark CPU Benchmarks. Dostępne w Internecie: https://www.cpubenchmark.net/cpu_list.php.

[7] perfmon2 - the hardware-based performance monitoring interface for Linux. Dostępne w Internecie: http://perfmon2.sourceforge.net/.

[8] Projekt Asteroids@home. Dostępne w Internecie: https://asteroidsathome.net.

[9] Security Performance Availability Engine Server Monitoring. Dostępne w Internecie: https://shalb.com/en/spae/.

[10] SETI@HOME CPU performance. Dostępne w Internecie: https://setiathome.berkeley.edu/cpu_list.php.

[11] Słownik SJP.PL. Dostępne w Internecie: https://sjp.pl/.

[12] TOP500 Supercomputer Sites. Dostępne w Internecie: `https://top500.org`.

[13] G. Aceto, A. Botta, W. de Donato, and A. Pescapè. Cloud monitoring: Definitions, issues and future directions. In *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, pages 63–67, Nov 2012.

[14] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. Survey cloud monitoring: A survey. *Comput. Netw.*, 57(9):2093–2115, June 2013.

[15] Khalid Alhamazani, Rajiv Ranjan, Karan Mitra, Fethi Rabhi, Prem Prakash Jayaraman, Samee Ullah Khan, Adnene Guabtni, and Vasudha Bhatnagar. An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. *Computing*, 97(4):357–377, Apr 2015.

[16] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Sahra Ali, and Debra Hensgen. Task execution time modeling for heterogeneous computing systems. In *Proceedings of the 9th Heterogeneous Computing Workshop*, HCW '00, pages 185–199. IEEE Computer Society, 2000.

[17] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, 1967.

[18] Ruby Annette, Aisha Banu. W, and Shriram Shriram. A taxonomy and survey of scheduling algorithms in cloud: Based on task dependency. 82:20–26, 11 2013.

[19] Jarosław Arabas. *Wykłady z algorytmów ewolucyjnych.* Wydawnictwo WNT, 2nd edition, 2004.

[20] Wolfgang Barth. *Nagios: System and network monitoring.* No Starch Press, 2008.

[21] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., 2006.

[22] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, pages 569–595. Morgan Kaufmann Publishers Inc., 1991.

[23] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, February 1991.

[24] A. Byrski. Immunologiczny mechanizm selekcji w agentowych obliczeniach ewolucyjnych. 2006.

[25] A. Byrski. *Agent-based Metaheuristics in Search and Optimisation*, volume 268 of *Rozprawy. Monografie*. AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, 2013.

[26] A Byrski, R Schaefer, M Smołka, and C Cotta. Asymptotic guarantee of success for multi-agent memetic systems. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 61(1):257–278, 2013.

[27] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, 1988.

[28] K. Cetnarowicz. Koncepcja strategii rozdziału zadań w zdecentralizowanych strukturach wieloprocesorowych. *Zeszyty Naukowe AGH, Elektrotechnika*, 8(3-4):621—-629, 1989.

[29] K. Cetnarowicz. *Problemy projektowania i realizacji systemów wieloagentowych*, volume 80 of *Rozprawy. Monografie*. AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, 1999.

[30] K. Cetnarowicz. *Paradygmat agentowy w Informatyce: koncepcje, podstawy i zastosowania*. Akademicka Oficyna Wydawnicza EXIT, 2012.

[31] P. Coad and J. Nicola. *Object-Oriented Programming*. Yourdon Press, 1993.

[32] Kenneth De Jong. Learning with genetic algorithms: An overview. *Machine Learning*, 3(2):121–138, 1988.

[33] Y. Demazeau and J.-P. Müller. Decentralized artificial intelligence. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I. : Proc. of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Cambridge, England*, pages 3–13. North-Holland, 1990.

[34] M. Dhingra, J. Lakshmi, and S. K. Nandy. Resource usage monitoring in clouds. In *2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 184–191, Sept 2012.

[35] G. Dobrowolski. *Technologie agentowe w zdecentralizowanych systemach informacyjno-decyzyjnych*, volume 107 of *Rozprawy. Monografie*. AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, 2002.

[36] Fangpeng Dong. A taxonomy of task scheduling algorithms in the grid. 17:439–454, 12 2007.

[37] Fangpeng Dong and Selim G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, 2006.

[38] Kaniz Fatema, Vincent C. Emeakaroha, Philip D. Healy, John P. Morrison, and Theo Lynn. A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, 74(10):2918 – 2933, 2014.

[39] M. Flasiński. *Wstęp do sztucznej inteligencji*. Wydawnictwo Naukowe PWN, 2011.

[40] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35. Springer-Verlag, 1997.

[41] M. R. Garey, D. S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[42] Jakub Gąsior and Franciszek Seredyński. Decentralized job scheduling in the cloud based on a spatially generalized prisoner's dilemma game. *International Journal of Applied Mathematics and Computer Science*, 25(4):737–751, 2015.

[43] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. In *International Conference on Green Computing*, pages 135–146, 2010.

[44] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[45] Richard Goodwin. Formalizing properties of agents. Technical report, 1993.

[46] Daniel Grzonka. Application of high-performance techniques for solving linear systems of algebraic equations. *Journal of Telecommunications and Information Technology*, 4:85–91, 2013.

[47] Daniel Grzonka, Agnieszka Jakóbik, Joanna Kołodziej, and Sabri Pllana. Using a multi-agent system and artificial intelligence for monitoring and improving the cloud performance and security. *Future Generation Computer Systems*, 86:1106 – 1117, 2018.

[48] Daniel Grzonka, Joanna Kolodziej, and Jie Tao. Using artificial neural network for monitoring and supporting the grid scheduler performance. In *28th European Conference on Modelling and Simulation, ECMS 2014, Brescia, Italy, May 27-30, 2014*, pages 515–522, 2014.

[49] Daniel Grzonka, Joanna Kołodziej, Jie Tao, and Samee Ullah Khan. Artificial neural network support to monitoring of the evolutionary driven security aware scheduling in computational distributed environments. *Future Generation Computer Systems*, 51:72–86, 2015.

[50] Mor Harchol-Balter, Mark E Crovella, and Cristina D Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, 1999.

[51] Simon Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA, 2009.

[52] Raphael Hunger. *Floating point operations in matrix-vector calculus*. Munich University of Technology, Inst. for Circuit Theory and Signal Processing Munich, 2005.

[53] IDERA. SQL Enterprise Job Manager. Dostępne w Internecie: `https://www.idera.com/productssolutions/sqlserver/sql-server-agent-job`.

[54] IDERA. Uptime Cloud Monitor. Dostępne w Internecie: `https://www.idera.com/infrastructure-monitoring-as-a-service`.

[55] A. Jakóbik, D. Grzonka, J. Kołodziej, A. E. Chis, and H. Gonzalez-Velez. Energy Efficient Scheduling Methods for Computational Grids and Clouds. *Journal of Telecommunications and Information Technology*, 1:56–64, 2017.

[56] Agnieszka Jakóbik, Daniel Grzonka, and Joanna Kołodziej. Security supportive energy aware scheduling and scaling for cloud environments. In *European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary, May 23-26, 2017, Proceedings.*, pages 583–590, 2017.

[57] Agnieszka Jakóbik, Daniel Grzonka, Joanna Kołodziej, and Horacio Gonzalez-Velez. Towards secure non-deterministic meta-scheduling for clouds. In *30th European Conference on Modelling and Simulation, ECMS 2016, Regensburg, Germany, May 31 - June 03, 2016. Proceedings.*, pages 596–602, 2016.

[58] Agnieszka Jakóbik, Daniel Grzonka, and Francesco Palmieri. Non-deterministic security driven meta scheduler for distributed cloud organizations. *Simulation Modelling Practice and Theory*, 76:67 – 81, 2017. High-Performance Modelling and Simulation for Big Data Applications.

[59] P. C. Janca and D. Gilbert. *Practical Design of Intelligent Agent Systems*, pages 73–89. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[60] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, January 1998.

[61] Mala Kalra and Sarbjeet Singh. A review of metaheuristic scheduling techniques in cloud computing. *Egyptian Informatics Journal*, 16(3):275 – 295, 2015.

[62] Helen D. Karatza. *Performance Evaluation and Analysis of Large Scale Distributed Systems: Issues, Trends, Problems and Solutions.* cHiPSet COST Action Summer School (September 21-23, 2016, Bucharest, Romania).

[63] David Karger, Cliff Stein, and Joel Wein. Algorithms and theory of computation handbook. chapter Scheduling Algorithms. Chapman & Hall/CRC, 2010.

[64] David Kincaid and Ward Cheney. *Numerical Analysis: Mathematics of Scientific Computing.* Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 1991.

[65] M. Kisiel-Dorohinicki. *Agentowe architektury populacyjnych systemów inteligencji obliczeniowej*, volume 269 of *Rozprawy. Monografie.* AGH Uczelniane Wydawnictwa Naukowo-Dydaktyczne, 2013.

[66] J. Kołodziej, S. U. Khan, and F. Xhafa. Genetic algorithms for energy-aware scheduling in computational grids. In *2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 17–24, 2011.

[67] Joanna Kołodziej. *Evolutionary Hierarchical Multi-Criteria Metaheuristics for Scheduling in Large-Scale Grid Systems.* Springer Publishing Company, Incorporated, 2012.

[68] Joanna Kołodziej, Samee Ullah Khan, Lizhe Wang, Aleksander Byrski, Nasro Min-Allah, and Sajjad Ahmad Madani. Hierarchical genetic-based grid scheduling with energy optimization. *Cluster Computing*, 16(3):591–609, 2013.

[69] Kurt Konolige. *A Deduction Model of Belief.* Morgan Kaufmann Publishers Inc., 1986.

[70] Tevfik Kosar and Mehmet Balman. A new paradigm: Data-aware scheduling in grid computing. *Future Generation Computer Systems*, 25(4):406 – 413, 2009.

[71] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu, and C. Stratan. Monalisa: An agent based, dynamic service

system to monitor, control and optimize distributed systems. *Computer Physics Communications*, 180(12):2472 – 2498, 2009. 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures.

[72] Theo Lynn, Philip D. Healy, Richard McClatchey, John P. Morrison, Claus Pahl, and Brian Lee. The case for cloud service trustmarks and assurance-as-a-service. In *CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 8-10 May, 2013*, pages 110–115, 2013.

[73] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[74] A. Meera and S. Swamynathan. Agent based resource monitoring system in iaas cloud environment. *Procedia Technology*, 10:200 – 207, 2013. First International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013.

[75] Z. Michalewicz. *Algorytmy genetyczne+struktury danych=programy ewolucyjne*. WNT, 2nd edition, 1999.

[76] Mohd Hairy Mohamaddiah, Azizol Abdullah, Shamala Subramaniam, and Masnida Hussin. A survey on resource allocation and monitoring in cloud computing. *International Journal of Machine Learning and Computing*, 4(1):31, 2014.

[77] Mohammad Mohammadi and Timur Bazhirov. Comparative benchmarking of cloud computing vendors with high performance linpack. *Computing Research Repository*, 2017.

[78] Ioannis A. Moschakis and Helen D. Karatza. Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing. *Journal of Systems and Software*, 101:1 – 14, 2015.

[79] E. Nawarecki and K. Cetnarowicz. A concept of the decentralized multi-agent RT system. In *Proceedings of the International Conference Real Time'95*, pages 167–171, 1995.

[80] Hyacinth S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11:205–244, 1996.

[81] S. Osowski. *Sieci neuronowe do przetwarzania informacji*. Oficyna Wydawnicza Politechniki Warszawskiej, 2000.

[82] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of ec2 cloud computing services for scientific computing. In Dimiter R. Avresky, Michel Diaz, Arndt Bode, Bruno Ciciani, and Eliezer Dekel, editors, *Cloud Computing*, pages 115–131. Springer Berlin Heidelberg, 2010.

[83] Sonia Panchen, Peter Phaal, and Neil McKee. Inmon corporation's sFlow: A method for monitoring traffic in switched and routed networks. RFC 3176, 2001.

[84] Manolis Papadrakakis. Solving large-scale linear problems in solid and structural mechanics. 1993.

[85] Z. C. Papazachos and H. D. Karatza. Performance evaluation of gang scheduling in a two-cluster system with migrations. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8, 2009.

[86] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[87] E. Polański. *Wielki słownik ortograficzny PWN z zasadami pisowni i interpunkcji*. Wydawnictwo Naukowe PWN, iv, dodruk poprawiony edition, 2017.

[88] David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, Inc., New York, NY, USA, 1997.

[89] Mustafizur Rahman, Rajiv Ranjan, Rajkumar Buyya, and Boualem Benatallah. A taxonomy and survey on autonomic management of applications in grid computing environments. *Concurrency and Computation: Practice and Experience*, 23(16):1990–2019, 2011.

[90] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Principles of Knowledge Representation and Reasoning. Proceedings of the second International Conference*, pages 473–484. Morgan Kaufmann, 1991.

[91] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, volume 1, pages 586–591, 1993.

[92] Maria Alejandra Rodriguez and Rajkumar Buyya. A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. *Concurrency and Computation: Practice and Experience*, 29(8), 2017.

[93] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition, 2009.

[94] Leszek Rutkowski. *Metody i techniki sztucznej inteligencji*. Wydawnictwo Naukowe PWN, 2nd edition, 2009.

[95] A S. Fraser. Simulation of genetic systems by automatic digital computers i. introduction. 10:484 – 491, 1957.

[96] Franciszek Seredynski. Coevolutionary game theoretic multi-agent systems. In Zbigniew W. Raś and Maciek Michalewicz, editors, *Foundations of Intelligent Systems*, pages 356–365. Springer Berlin Heidelberg, 1996.

[97] Franciszek Seredynski. Competitive coevolutionary multi-agent systems: The application to mapping and scheduling problems. *Journal of Parallel and Distributed Computing*, 47(1):39 – 57, 1997.

[98] Shyna Sharma, Amit Chhabra, and Sandeep Sharma. Comparative analysis of scheduling algorithms for grid computing. In *2015 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2015, Kochi, India, August 10-13, 2015*, pages 349–354, 2015.

[99] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51 – 92, 1993.

[100] J. Sobieska-Karpińska and M. Hernes. Consensus determining algorithm in multiagent decision support system with taking into consideration improving agent's knowledge. In *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1035–1040, 2012.

[101] Huaguang Song. Preconditioning techniques analysis for cg method.

[102] G. L. Stavrinides and H. D. Karatza. Performance evaluation of gang scheduling in distributed real-time systems with possible software faults. In *2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pages 1–7, 2008.

[103] G. L. Stavrinides and H. D. Karatza. Scheduling different types of applications in a saas cloud. In *Proceedings of the 6th International Symposium on Business Modeling and Software Design (BMSD'16), At Rhodes, Greece*, pages 144–151, 2016.

[104] Georgios L Stavrinides and Helen D Karatza. Scheduling real-time bag-of-tasks applications with approximate computations in saas clouds. *Concurrency and Computation: Practice and Experience*, pages 1–12.

[105] Georgios L. Stavrinides and Helen D. Karatza. Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations. *SIMULATION*, 85(8):525–536, 2009.

[106] Georgios L. Stavrinides and Helen D. Karatza. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations.

Journal of Systems and Software, 83(6):1004 – 1014, 2010. Software Architecture and Mobility.

[107] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, June 2000.

[108] Made Suarjana and Kincho H. Law. A robust incomplete factorization based on value and space constraints. *International Journal for Numerical Methods in Engineering*, 38(10):1703–1719, 1995.

[109] Piotr Switalski and Franciszek Seredynski. Scheduling parallel batch jobs in grids with evolutionary metaheuristics. *Journal of Scheduling*, 18(4):345–357, 2015.

[110] Katia Sycara, Anandee Pannu, Mike Williamson, Dajun Zeng, and Keith Decker. Distributed intelligent agents. *IEEE Expert: Intelligent Systems and Their Applications*, 11(6):36–46, December 1996.

[111] Magdalena Szmajduch. Data and task scheduling in distributed computing environments. *Journal of Telecommunications and Information Technology*, (4):71–78, 2014.

[112] Jacek Tchórzewski, Ana Respício, and Joanna Kołodziej. ANN-based secure task scheduling in computational clouds. In *32nd European Conference on Modelling and Simulation, ECMS 2018, Wilhelmshaven, Germany, May 22 - 25, 2018. Proceedings.*, pages 468–474, 2018.

[113] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[114] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[115] J. D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384 – 393, 1975.

[116] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, May 2003.

[117] K. Wang, X. Zhou, T. Li, D. Zhao, M. Lang, and I. Raicu. Optimizing load balancing and data-locality with data-aware scheduling. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 119–128, 2014.

[118] Jonathan Stuart Ward and Adam Barker. Observing the clouds: a survey and taxonomy of cloud monitoring. *Journal of Cloud Computing*, 3(1):24, 2014.

[119] Richard R Weber. On the optimal assignment of customers to parallel servers. *Journal of Applied Probability*, 15(2):406–413, 1978.

[120] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.

[121] Gerhard Weiss. *Multiagent Systems*. The MIT Press, 2013.

[122] Alan Wijntje. *Monitoring with Opsview*. Packt Publishing Ltd, 2013.

[123] Wayne Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14(1):181–189, 1977.

[124] M. Wooldridge. Agent-based software engineering. *IEE Proceedings - Software Engineering*, 144(1):26–37, Feb 1997.

[125] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2009.

[126] Michael Wooldridge and Paul E. Dunne. *The Computational Complexity of Agent Verification*, pages 115–127. Springer Berlin Heidelberg, 2002.

[127] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. *Workflow Scheduling Algorithms for Grid Computing*, pages 173–214. Springer Berlin Heidelberg, 2008.

[128] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163 – 188, 2005.

[129] D. A. Zubok, T. V. Kharchenko, A. V. Maiatin, and M. V. Khegai. A multi-agent approach to the monitoring of cloud computing system with dynamically changing configuration. In *2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, pages 410–416, April 2016.

Spis tablic

Spis rysunków