Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Akademii Górniczo-Hutniczej im. St. Staszica
w Krakowie

Zarządzanie zasobami gridowymi z użyciem parawirtualizacji

Rozprawa doktorska

Jacek Kosiński

Promotor: prof. dr hab. inż. Krzysztof Zieliński

Kraków, Styczeń 2009

Serdecznie dziękuję wszystkim osobom, które wspierały mnie w czasie pisania tej pracy.

W szczególności chciałbym podziękować mojemu promotorowi Panu prof. dr hab. inż. Krzysztofowi Zielińskiemu za wszechstronną pomoc oraz wnikliwe i konstruktywne uwagi.

Pragnę także podziękować za pomoc kolegom z Grupy Systemów Rozproszonych Katedry Informatyki AGH.

Chciałbym także okazać wdzięczność mojej Rodzinie oraz Przyjaciołom za cierpliwość i zrozumienie wykazywane dla mojego zaangażowania w tę rozprawę.

Jacek Kosiński

Spis treści

Spis rysunków

Spis tabel

Kod źródłowy

Rozdział 1

Wprowadzenie

Pojęcie wirtualizacji, w znaczeniu technicznym, pojawiło się już na samym początku rozwoju technik komputerowych. Wszystkie próby tworzenia zunifikowanych interfejsów, rozwiązań sprzętowych ukrywających złożoność używanych komponentów, interfejsu programistycznego API (ang. *Application Programming Interface*) — powodowały opracowanie pewnej abstrakcji rzeczywistości, upraszczającej tworzenie sprzętu i oprogramowania. Zagadnienia wirtualizacji wpisują się ściśle w paradygmaty współczesnego projektowania i użytkowania systemów komputerowych, włączając w to projektowanie zorientowane obiektowo, Web-serwisy czy technologie komponentowe.

Sieć Internet jest również dobrze rozpoznawanym przykładem wirtualizacji, jako że udostępnia usługi uniwersalnego medium komunikacyjnego, umożliwiając uzyskanie dostępu do rozproszonych zasobów lub aplikacji z dowolnego miejsca.

Systemy Grid, jako jedne z ważniejszych narzędzi współczesnego przetwarzania rozproszonego, wprowadzają pewien poziom wirtualizacji, ponieważ Grid można określić jako kolekcję rozproszonych zasobów obliczeniowych udostępnionych poprzez sieć LAN (ang. *Local Area Network*) lub WAN (ang. *Wide Area Network*) i widzianych z poziomu użytkownika jako duży, pojedynczy system obliczeniowy.

Techniki wirtualizacji sprzętu komputerowego umożliwiają uproszczenie mechanizmów wykorzystywanych do zarządzania zasobami. W podejściu klasycznym system operacyjny (najbardziej uprzywilejowany element) ma nieograniczony dostęp do zasobów komputera, na którym jest uruchomiony. Wprowadzając technikę wirtualizacji, dostarczającą iluzji wirtualnego komputera VM (ang. *Virtual Machine*), możliwe jest precyzyjne sterowanie konsumpcją zasobów dostępnych dla systemu operacyjnego oraz pośrednio dla wszystkich aplikacji w nim uruchomionych.

Budując środowiska obliczeniowe z wirtualnych maszyn VM, uzyskuje się dostęp do mechanizmów, które pozwalają na precyzyjne sterowanie dostępem oraz konsumpcją zasobów. Dzięki temu, można tak dobrać rozmieszczenie i konfigurację wirtualnych węzłów obliczeniowych VWN (ang. *Virtual Worker Node*) wykonujących aplikacje o zróżnicowanych wymaganiach, aby maksymalnie wykorzystać dostępne zasoby (w szczególności moc obliczeniową, która w klasycznych instalacjach Grid, często składających się z heterogenicznych konfiguracji sprzętowych, nie zawsze może być efektywnie wykorzystywana).

Dzięki takiemu podejściu, twórcy aplikacji obliczeniowych mogą określić parametry środowiska wykonania — wirtualnego Gridu, które najlepiej będzie odpowiadać potrzebom danej aplikacji. Wirtualny Grid może być zatem rozumiany jako pewna forma określenia wymogów czy opisu zasobów (zarówno obliczeniowych jak i komunikacyjnych), dostarczona wraz z aplikacją w celu stworzenia środowiska umożliwiającego jej poprawne i efektywne działanie. Specyfikacja ta określa wymagane przez aplikację zasoby, ich konfigurację i inne

elementy, na które dzięki zastosowaniu wirtualizacji węzłów obliczeniowych, infrastruktura zarządzania będzie miała wpływ, a których dostępność i odpowiednie przygotowanie stanowi kluczowy wkład w osiągnięcie przez aplikację odpowiedniego poziomu wykorzystania infrastruktury fizycznej.

Cechy systemu gridowego, takie jak zmienna w czasie dostępność, rozproszenie geograficzne, komponenty należące i zarządzane przez różne organizacje, różne wymagania, polityki bezpieczeństwa i zarządzania zasobami, a także komunikacja za pomocą heterogenicznych technologii sieciowych — mogą być efektywnie zapewnione za pomocą koncepcji wirtualnego Gridu. Zbudowanie infrastruktury obliczeniowej w oparciu o wirtualne komponenty, jak zostanie wykazane, będzie umożliwiało osiągnięcie wizji Gridu jako elastycznego, adaptowalnego [107] narzędzia dla obliczeń na żądanie dla szerokiej liczby aplikacji.

1.1. Definicje

Przedstawienie tezy, celu oraz zakresu pracy wymaga określenia definicji podstawowych pojęć używanych w dalszej części niniejszej dysertacji. Pojęcia te dotyczą głównych elementów dziedziny tworzonego rozwiązania i są nimi: „Grid", „(para)wirtualizacja", „zarządzanie zasobami", „wirtualne organizacje" a także takie wskaźniki jak „efektywność zarządzania zasobami" czy „efektywność wykonania aplikacji".

Środowiska typu Grid

Idea przetwarzania opartego na rozproszonej geograficznie infrastrukturze obliczeniowej pojawiła się już na początku lat 90-tych ubiegłego wieku. Przyczyniły się do tego prace nad tworzeniem środowisk ewaluacji oprogramowania takich jak CASA [63]. Wspomniane prace koncentrowały się na połączeniu zasobów kilku super-komputerowych centrów naukowych USA za pomocą gigabitowej sieci komputerowej. Pojęcie „Grid" zostało wprowadzone do literatury naukowej dopiero za sprawą opublikowanej w 1998 roku książki „*The grid: Blueprint for a New Computing Infrastructure*" [36]. Autorzy I. Foster i C. Kesselman, zaproponowali użycie pojęcia Grid przez analogię przedstawionej koncepcji przetwarzania do budowy infrastruktury dystrybucji energii elektrycznej[1]. Ta pierwsza definicja określała Grid jako: medium dostarczające usługi ciągłego dostępu do mocy obliczeniowej. W dalszych publikacjach wspomnianych autorów zostały określone właściwości i charakterystyki, jakie powinna posiadać nowo proponowana architektura programowa. W pozycji „*What is the grid? A Three Point Checklist*" [35] zostały wymienione warunki jakie system Grid powinien spełniać, a są nimi [35]:

1. Obsługa zasobów nie może podlegać scentralizowanemu mechanizmowi kontroli.
2. Konieczne jest stosowanie uniwersalnych i otwartych standardów i protokołów.
3. Infrastruktura powinna udostępniać zaawansowane usługi QoS (ang. *Quality of Service*).

W tradycyjnym systemie komputerowym zasoby są rozdzielane z wykorzystaniem mechanizmów dostępnych w systemie operacyjnym. Mechanizmy te zakładają, iż dostęp i mo-

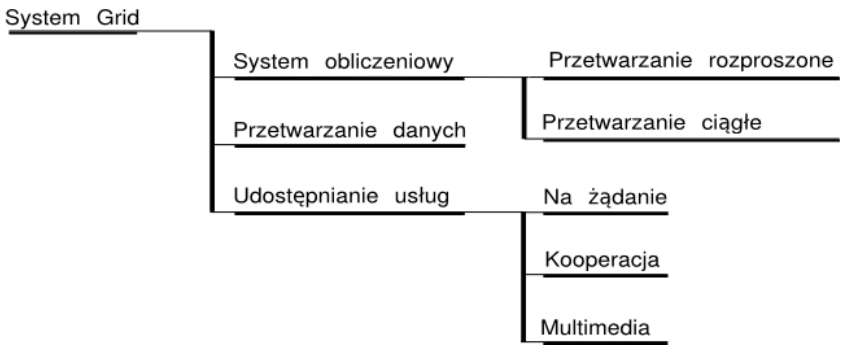
[1] Z angielskiego: *Electric power GRID.*

nitorowanie zużycia są obsługiwane w sposób scentralizowany. W systemie Grid multipleksacja zasobów musi obejmować wiele niezależnych domen administracyjnych i nie może wykorzystywać na scentralizowanego modelu. Fundamentalną cechą organizacji przetwarzania w oparciu o koncepcję Grid jest transparentna eksploatacja rozproszonych zasobów poprzez wielu użytkowników połączonych siecią rozległą WAN.

Ta główna cecha rozważanej architektury przetwarzania pozwala na przedstawienie pierwszej, podstawowej definicji środowiska Grid:

Definicja 1.1 (Grid) *System Grid to dynamiczne środowisko, bazujące na współdzielonych zasobach i wykorzystujące na masową skalę oraz w sposób zdecentralizowany rozproszone komponenty.*

Od czasu zaproponowania koncepcji Grid nastąpił znaczący rozwój tej architektury przetwarzania rozproszonego [10, 15, 49, 89]. Dynamicznie zmieniają się również wykorzystywane przy budowie tych środowisk technologie, zastosowanie czy sposób organizacji obliczeń wielkiej skali. Taksonomię współcześnie stosowanych środowisk Grid przedstawia rysunek 1.1.



Rysunek 1.1. Taksonomia stosowanych obecnie architektur typu Grid [53]

Szczególnie ze względu na mnogość wizji wykorzystania jej koncepcji przetwarzania, trudno się obecnie doszukiwać jedynej ogólnie przyjętej definicji systemu Grid. Poprzez analizę różnych (często uzupełniających się) definicji systemu Grid dostępnych w literaturze, można podjąć próbę określenia ścisłego zestawu cech, które system ten powinien posiadać, bądź których posiadanie przez środowisko przetwarzania pozwoli nazwać je „Gridem". A zatem cechy współczesnego Gridu to:

- **Środowisko dużej skali**: Grid musi poprawnie obsługiwać zarówno pojedyncze zasoby jak i ich duże ilości.
- **Rozproszenie geograficzne**: zasoby Gridu mogą być rozmieszczone w odległych geograficznie lokalizacjach.
- **Heterogeniczność zasobów**: zarówno zasoby sprzętowe jak i programowe mogą być silnie zróżnicowane.

- **Współdzielenie zasobów**: zasoby Gridu fizycznie należą do wielu różnych instytucji i organizacji, zezwalającym obcym użytkownikom na ich wykorzystanie.
- **Rozproszona władza administracyjna**: wiele instytucji i organizacji to równocześnie wiele polityk administracji i bezpieczeństwa określonych dla udostępnionej infrastruktury.

Dodatkowo oprócz powyższych cech, środowiska Grid powinny spełniać następujące wymagania:

- **Koordynacja zasobów**: zasoby muszą być skoordynowane aby możliwe było ich równoczesne wykorzystanie.
- **Przezroczysty dostęp**: zasoby Gridu powinny być widziane z poziomu użytkownika jako pojedynczy, system komputerowy.
- **Wsparcie dla QoS**: procedura udostępniania usług musi mieć możliwość określenia warunków użycia oraz wymagań QoS.
- **Zunifikowany dostęp**: Grid powinien być budowany z wykorzystaniem standardowych usług, protokołów i interfejsów komunikacyjnych, umożliwiając tym samym ukrycie heterogeniczności zasobów oraz zwiększenie efektywności tworzenia aplikacji.
- **Ciągła dostępność**: system musi udostępniać zasoby w sposób ciągły, poprzez adaptację do zmian w dynamicznym środowisku, w którym występują awarie i modyfikację konfiguracji zasobów. Środowisko musi osiągnąć maksymalną wydajność możliwą do uzyskania z aktualnie dostępnych zasobów.

Praca [14] jest próbą usystematyzowania pojęcia Grid, autorzy poprzez analizę cech wynikających z definicji dostępnych w literaturze tematu, proponują (według autora, najbardziej pełną) definicję Gridu jako:

> **Definicja 1.2** (Grid) *Rozproszona geograficznie, sprzętowa i programowa infrastruktura przetwarzania dużej skali, zbudowana z heterogenicznych zasobów (w tym zasobów komunikacyjnych), udostępnionych i współdzielonych przez wiele organizacji, koordynowanych w celu osiągnięcia przezroczystego wsparcia dla przetwarzania szerokiego zakresu aplikacji [14].*

Wirtualne Organizacje

Ogólne definicje środowisk typu Grid nie ograniczają w żaden sposób możliwych zastosowań zbudowanej w oparciu o tę koncepcję infrastruktury przetwarzania. W środowiskach naukowych, częstym jest scenariusz połączenia wysiłków zespołów naukowych, w celu rozwiązania złożonego problemu. Możemy mówić wtedy o tzw. „Wirtualnej Organizacji" czyli nieistniejącej w rzeczywistości instytucji powołanej w celu rozwiązywania problemów naukowych dużej skali wykorzystując do tego celu przetwarzanie w oparciu o zasoby środowisk typu grid.

W typowym scenariuszu duże centra naukowe są odpowiedzialne za udostępnianie zasobów używanych przez pewną grupę osób. W podejściu Grid definicja Wirtualnej Organizacji sprowadza się do określenia zestawu użytkowników, którzy za pomocą centralnego serwisu zlecają zadania oraz do określenia zestawu reguł, według których obsługiwany jest przydział poszczególnych zasobów infrastruktury. Wirtualna Organizacja (VO) może być zatem określona jako:

Definicja 1.3 (Wirtualna Organizacja) *Wirtualna Organizacja jest to dynamiczna kolekcja rozproszonych zasobów wraz z zestawem reguł określających w precyzyjny sposób ich współdzielenie. Zasoby te są wykorzystywane przez zmieniającą się grupę użytkowników, należących do jednej lub wielu fizycznych organizacji.*

Politykę zarządzania współdzieleniem zasobów można wyrazić jako zestaw reguł typu zezwól/zabroń bazujących na informacjach uzyskanych za pomocą usług kontroli wykorzystania zasobów oraz serwisach identyfikacji grup/użytkowników. Polityka funkcjonowania VO powinna adresować każdy aspekt działania VO, dla którego organizacja ta została powołana i może być podzielona na trzy następujące kategorie[2]:

- reguły zarządzania Wirtualną Organizacją jako całością,
- polityka zarządzania zasobami,
- polityka zarządzania użytkownikami.

Zarządzanie Wirtualną Organizacją wymaga użycia systemu monitorowania infrastruktury umożliwiającego zebranie informacji na temat poziomu wykorzystania zasobów. Informacje te są zwykle przetwarzane przez element środowiska (system zarządzania) odpowiedzialny za przestrzeganie reguł polityki. Odbywa się to poprzez podejmowanie określonych akcji na podstawie weryfikacji zdefiniowanych wcześniej warunków.
Przykładowe polityki funkcjonowania VO dotyczące dystrybucji zasobów:

- Zgodnie z regułą $1/N$ — czyli równe rozłożenie obciążenia pomiędzy zasoby należące do VO.
- Dostęp do zasobów innych członków VO na podstawie wartości pewnej metryki, której wartość zależy od poziomu wykorzystania zasobów udostępnionych lokalnie dla danej Wirtualnej Organizacji.[3]

[2]Przyjęte założenia (sekcja 3.3) oraz zakres funkcjonalny proponowanego w niniejszej pracy rozwiązania sprowadza zakres polityki VO jedynie do kwestii związanych z zarządzaniem zasobami.

[3]Czyli odwzorowanie zasady: „dostaniesz tyle ile sam udostępnisz" — polityka „*you-get-what-you-give*" (YGwYG).

Wirtualizacja i parawirtualizacja

Znaczenie przymiotnika wirtualny wiąże się ze sposobami kreowania pewnej wyimaginowanej rzeczywistości. Przykładami mogą tu być takie określenia jak „wirtualny świat" czy „wirtualna rzeczywistość" określające techniki mające na celu oszukanie obserwatora, poprzez dostarczenie pewnej iluzji rzeczywistego obiektu. Techniki wirtualizacji stosowane w niniejszej pracy mają za zadanie właśnie takie oszukiwanie [6, 18]. Oszukiwany jest w tym przypadku system operacyjny (oraz działające w jego obrębie aplikacje) poprzez dostarczenie nierozróżnialnej iluzji wyłącznego dostępu do fizycznego sprzętu.

Parawirtualizacja jest jednak pewnym osłabieniem tego podejścia, gdyż w tym przypadku określonym modyfikacjom musi ulec również element „oszukiwany". A zatem system operacyjny musi być odpowiednio dostosowany, by móc wykorzystywać wirtualne środowisko w którym jest uruchomiony. To dostosowanie oraz pełna świadomość faktu, iż wykonanie odbywa się na wirtualnym sprzęcie powodują, że podejścia tego nie można nazywać pełną wirtualizacją. Niewątpliwą zaletą tego rozwiązania jest osiągana wydajność wirtualnych maszyn zbliżona do wydajności uzyskiwanej bez stosowania technik wirtualizacji [5, 103]. Parawirtualizacja to zatem:

> **Definicja 1.4** (Parawirtualizacja) *Parawirtualizacja lub wirtualizacja lekka (ang.* LightWeight Virtualization*) — technika wirtualizacji, która zapewniając pewną iluzję rzeczywistości, wymaga aby wykorzystujące ją elementy zostały odpowiednio dostosowane i były świadome wykorzystywania mechanizmów wirtualizacji.*

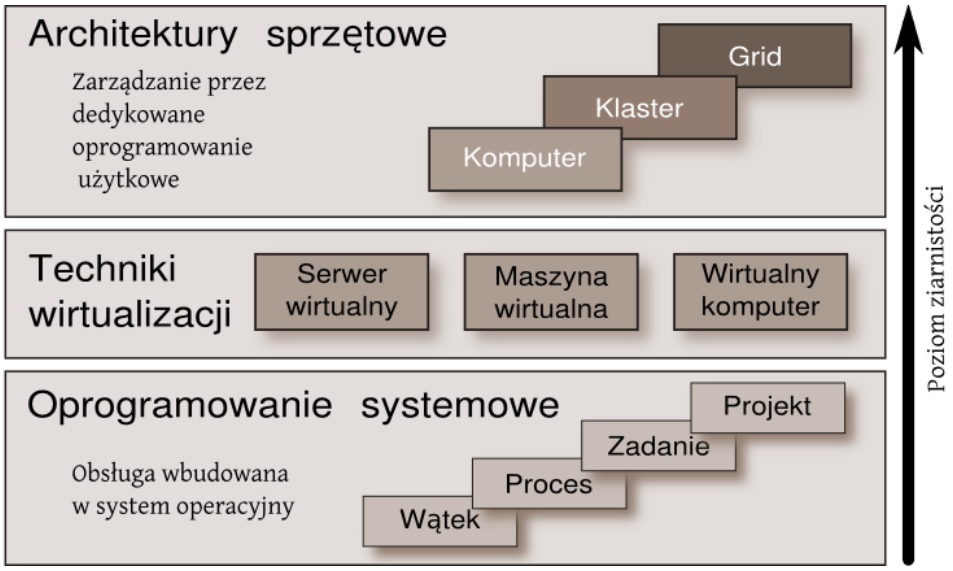
Parawirtualizację realizuje się przez implementację specjalnego API, które używane jest przez zmodyfikowane systemy operacyjne uruchomione w wirtualnych maszynach. Poprzez ten interfejs system operacyjny ma dostęp do zasobów fizycznych, jednak wywołania te są weryfikowane przez element nadzorczy (ang. *hypervisor*), którego celem jest zapewnienie izolacji VM.

Wirtualny Grid

W odniesieniu do rysunku 1.2 przedstawiającym hierarchię poziomów wirtualizacji, również technologia Grid wprowadza pewien poziom abstrakcji. Abstrakcję tą uzyskuje się poprzez przykrycie złożoności metod dostępu, rozproszenia i heterogeniczności zasobów, zestawem serwisów udostępniających usługi skoordynowanego i zunifikowanego dostępu do zasobów.

„Wirtualny Grid" jest rezultatem wykorzystania specyfikacji wymagań aplikacji (dostarczonych z aplikacją bądź abstrahowanych przez system) w celu konstrukcji wymaganego dla niej środowiska wykonawczego.
Definicja wirtualnego Gridu przyjęta w tej pracy to:

> **Definicja 1.5** (Wirtualny Grid) *Wirtualny Grid jest rozumiany jako forma określenia wymogów czy opisu zasobów, dostarczona przez aplikację w celu stworzenia środowiska umożliwiającego jej poprawne i efektywne działanie.*



Rysunek 1.2. Poziomy wirtualizacji w systemach informatycznych

Wirtualny Grid jest zatem rezultatem tworzenia wyidealizowanej infrastruktury przetwarzania tzw. środowiska wykonawczego na podstawie specyfikacji określonej dla konkretnej aplikacji. Specyfikacja określa wymagane zasoby, ich konfigurację i inne elementy, na które dzięki zastosowaniu np. technik wirtualizacji aplikacja będzie miała pośrednio wpływ, a które stanowią kluczowy wkład w osiągnięcie przez nią odpowiedniego poziomu wykorzystania zasobów fizycznych.

Zasoby i zarządzanie zasobami

Zasoby to elementy systemu informatycznego, wykorzystywane do realizacji zadań, do których system ten został stworzony. W przypadku środowisk Grid, pojęcie zasobu, odnosić może się zarówno do zasobów infrastruktury (środowiska) realizujących przetwarzanie (ang. *computing resources*) jak i zestawu usług, udostępnianych przez komponenty programowe środowiska dla aplikacji. Zasoby używane do przetwarzania zadań, to zasoby typowego systemu komputerowego takie jak: moc obliczeniowa, pamięć operacyjna, przestrzeń dyskowa oraz zasoby infrastruktury sieciowej, takie jak np. przepustowość komunikacji. Z punktu widzenia aplikacji (konsumenta zasobów) są to elementy systemu wymagane dla jej poprawnego i efektywnego działania.

Definicja 1.6 (Zasób) *Zasoby są to elementy systemu informatycznego, bez dostępności których na określonym poziomie niemożliwe byłoby wykonywanie zadań przetwarzania zleconych przez użytkowników tego systemu.*

Wsparcie systemów Grid dla przetwarzania dużej liczby aplikacji o zróżnicowanych wymaganiach powoduje konieczność posiadania odpowiedniego systemu zarządzania zasobami RMS (ang. *Resource Management System*). System ten musi mieć możliwość wpły-

wania na konfigurację zasobów, ich dostępność oraz parametry procesu udostępniania zasobów dla aplikacji [4, 16]. Wpływ ten określamy mianem zarządzania zasobami.

Definicja 1.7 (Zarządzanie) *Zarządzanie zasobami jest to proces, którego rezultatem jest zmiana dostępności bądź stanu zarządzanego zasobu. Wpływ na zasób może być bezpośredni lub osiągnięty pośrednio za pomocą zmiany właściwości otoczenia obiektu lub warunków jego pracy.*

System zarządzania zasobami powinien móc wykonywać decyzje powodujące maksymalizację metryk zdefiniowanych przez użytkowników dla aplikacji posiadających takie wymagania. Wymóg spełnienia w/w cech powoduje, iż znacząco rośnie złożoność mechanizmów przydziału zasobów, gdyż konieczne jest rozwiązywanie wielokryterialnych problemów optymalizacyjnych.

Efektywność zarządzania i efektywność wykonania

Proces zarządzania zasobami, to możliwość wykonania określonych operacji regulujących powiązanie pomiędzy zasobem a jego konsumentem. Efektywność zarządzania może być więc rozumiana jako zapewnienie możliwości wykonania operacji zarządzania, wraz z równoczesnym dążeniem do minimalizacji złożoności i narzutu dodatkowych mechanizmów umożliwiających realizację tej funkcjonalności.

O efektywności zarządzania możemy mówić zatem jako o pewnym kompromisie pomiędzy uzyskanymi dzięki zarządzaniu rezultatami a kosztami poniesionymi na ten cel. Rezultaty zarządzania to oczekiwane modyfikacje wartości określonych wskaźników/metryk opisujących stan zarządzanych obiektów. Koszt zarządzania wynika głównie z intruzywności implementacji, czyli dodatkowego obciążenia generowanego przez komponenty dodane w celu realizacji zarządzania, zwiększenia stopnia złożoności systemu, zwiększenia awaryjności, itp.

Na potrzeby niniejszej pracy, za efektywne zarządzanie przyjmuje się takie podejście, które cechuje się bogatym zestawem możliwości określenia jakie zasoby będą udostępnione, na jakich warunkach, na jak długo oraz czy będzie można zagwarantować pewien poziom ich dostępności. Efektywne zarządzanie jest możliwe pod warunkiem, że zasoby są wyposażone w odpowiednie mechanizmy umożliwiające ich zarządzanie. Konieczne staje się więc odpowiednie wyposażenie zasobów w komponenty wykonujące polecenia systemu zarządzania.

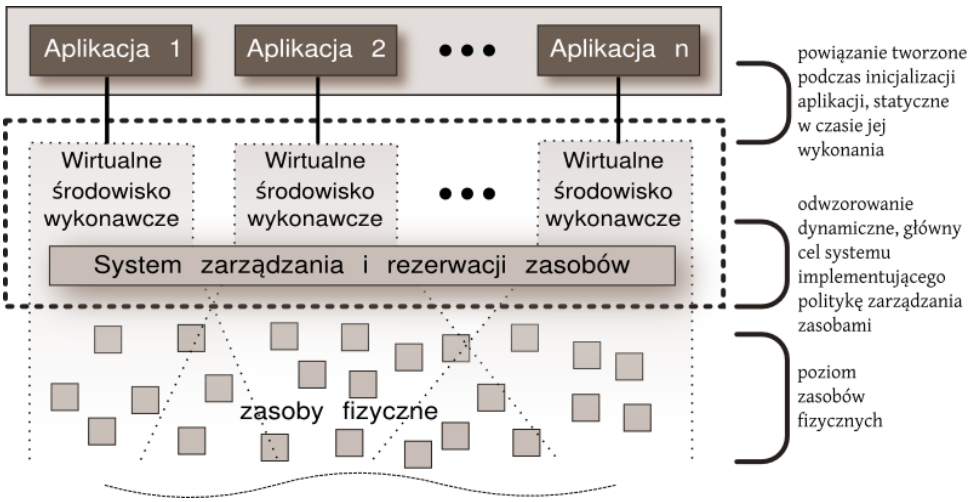
Podsumowując, można przyjąć następującą definicję efektywności zarządzania zasobami:

Definicja 1.8 (Efektywność zarządzania) *Sposób realizacji zarządzania zasobami jest efektywny jeśli równocześnie z udostępnieniem szerokiego zestawu możliwości zarządzania umożliwiających osiągnięcie określonego celu minimalizowany jest niekorzystny wpływ na środowisko i aplikacje elementów dodanych w celu realizacji tych możliwości.*

1.2. Cel rozprawy

Zastosowanie technik wirtualizacji jest naturalną ewolucją istniejących systemów zarządzania zasobami w środowiskach rozproszonych. Niniejszą pracę można zatem traktować jako odpowiedź na pytanie: czy poziom wirtualizacji jaki oferowany jest przez dostępne technologie tworzenia wirtualnych komputerów oraz wirtualnej infrastruktury komunikacyjnej daje się skutecznie wykorzystać do organizacji przetwarzania w środowiskach typu Grid? Odpowiedź na to pytanie będzie przedstawiona poprzez ukazanie jakie pozytywne skutki (w jakich sytuacjach) dają zastosowane koncepcje polegające na tym, że środowisko wykonawcze aplikacji (wirtualny Grid) jest tworzone na żądanie na podstawie dostarczonej wraz z aplikacją specyfikacji i założenia, iż środowisko to będzie przydzielone do aplikacji na zasadach wyłączności.

Wykonanie aplikacji w wirtualnym otoczeniu (środowisku uruchomieniowym) powoduje, iż uzyskuje się dwuetapowe odwzorowanie zasobów. Pierwsze powiązanie jest realizowane między zasobami fizycznymi a zasobami wirtualnymi, drugie powiązanie to przydzielenie zasobów wirtualnych dla aplikacji (rysunek 1.3). Dzięki temu znacznie upraszcza się alokację zasobów do aplikacji, gdyż operując na zasobach wirtualnych a nie na fizycznych, specyfikacja wymogów aplikacji może być konstruowana na znacznie wyższym poziomie abstrakcji. Rozwiązanie to umożliwia rozdzielenie podstawowej funkcjonalności (biznesowej) aplikacji od zarządzania zasobami co ułatwia konstrukcję, utrzymanie, a także dostosowanie mechanizmów zarządzania zasobami do zmian stanu aplikacji i faz jej wykonania.



Rysunek 1.3. Dwuetapowy model powiązań zasobów do aplikacji

Również dzięki temu, iż wykorzystywane są techniki wirtualizacji możliwe będzie łatwe dostarczenie funkcjonalności, która w oparciu o reguły działania stworzone na podstawie zadanej polityki VO będzie dokonywała dynamicznej modyfikacji wspomnianego wcześniej powiązania pomiędzy zasobami fizycznymi a wirtualnymi.

Podstawowym celem niniejszej pracy jest wykazanie, iż poprzez zastosowanie technik parawirtualizacji, możliwe jest stworzenie systemu zarządzania zasobami, który dzięki zapewnieniu elastycznych mechanizmów alokacji zasobów oraz możliwości określenia polityki

ich rozdziału umożliwi zwiększenie szeroko rozumianej wydajności działania dla wybranych aplikacji rozproszonych w środowisku typu Grid.

1.3. Teza pracy

Większość obecnych prac mających na celu poprawę mechanizmów zarządzania zasobami w środowiskach Grid koncentruje się na zapewnieniu podobnej funkcjonalności w tzw. warstwie pośredniczącej (ang. *middleware*). Ponieważ istnieją pewne ograniczenia [4, 16, 71] takiego sposobu realizacji przydziału i zarządzania zasobami, w niniejszej pracy została zaproponowana koncepcja wykorzystania technik wirtualizacji jako lekkiego mechanizmu umożliwiającego sterowanie wykorzystaniem zasobów.

Przedstawione koncepcje prowadzą do przyjęcia na potrzeby realizacji niniejszej pracy następujących założeń określających:

- **poziom wirtualizacji** — praca koncentruje się na możliwościach tworzenia wirtualnych komputerów i wirtualnych sieci dostępnych dzięki wykorzystaniu technik parawirtualizacji. Tak przyjęty poziom wirtualizacji umożliwia stworzenie środowiska wykonania, które nie wymaga modyfikacji i dostosowania aplikacji,
- **reprezentacja zasobów** — obszar działania systemu zarządzania stanowią zasoby udostępniane przez wirtualne komputery i wirtualną sieć. Są to zasoby reprezentowane przez system, nad którymi będzie miał on kontrolę poprzez elementy środowiska umożliwiające alokację zasobów dla aplikacji,
- **zakres polityki wirtualnej organizacji** — wirtualny Grid jest tworzony w ramach zasobów danej wirtualnej organizacji, a zatem polityka funkcjonowania VO będzie dotyczyła aspektów zarządzania zasobami obliczeniowymi. Są to reguły dynamicznego przydzielania zasobów dla wirtualnego Gridu z puli zasobów VO oraz reguły realizacji dostępu w przypadku wykorzystania tych samych współdzielonych zasobów przez wiele wirtualnych Gridów równocześnie.

Wykorzystanie w praktyce implementacji zaproponowanej w pracy koncepcji będzie pozwalało na efektywne zarządzanie zasobami wirtualnego Gridu zbudowanego w oparciu o komponenty składające się z emulowanych (wirtualnych) maszyn. Oparcie budowy Gridu o wirtualizację pozwoli na elastyczną rekonfigurację poszczególnych węzłów (poprzez zmianę parametrów „fizycznych" takich jak liczba procesorów, dostępna pamięć operacyjna czy właściwości komunikacji sieciowej) oraz dzięki użyciu techniki migracji wirtualnej maszyny, na efektywniejsze wykorzystanie zasobów puli fizycznych maszyn — bez przerywania pracy poszczególnych komponentów wirtualnego Gridu.

Nadrzędnym założeniem pracy jest zatem konieczność przekonania się na ile zaproponowany poziom wirtualizacji nadaje się do realizacji koncepcji wirtualnego Gridu (poprzez weryfikację praktyczną) oraz czy poziom ten pomaga w realizacji zarządzania zasobami dla aplikacji uruchomionej w środowisku rozproszonym. Równie istotną rolą tej pracy jest także próba odpowiedzi na pytanie czy koncepcja powiązania aplikacji z dostosowanym do jej wymogów środowiskiem uruchomieniowym w postaci wirtualnego Gridu jest skutecznym mechanizmem, który pozwala na efektywne dzielenie zasobów dostępnych w ramach fizycznego Gridu.

Tezę pracy stanowi stwierdzenie, że:

Parawirtualizacja stanowi skuteczne narzędzie kreowania wirtualnych Gridów oraz zarządzania ich zasobami zgodnie z wymaganiami określonymi przez polityki funkcjonowania wirtualnych organizacji.

Teza ta zostanie w niniejszej pracy zweryfikowana poprzez budowę odpowiedniego środowiska tworzenia wirtualnych Gridów oraz eksperymentalną ocenę jego wpływu na charakterystyki wykonania szeregu aplikacji, w oparciu o odpowiednio dobrane strategie zarządzania zasobami.

1.4. Zakres rozprawy

Niniejsza rozprawa przedstawia proces konstrukcji systemu umożliwiającego efektywne zarządzanie zasobami w środowiskach Grid. Autor skupił się na zapewnieniu przezroczystości działania systemu, widzianej z poziomu istniejącego oprogramowania warstwy pośredniczącej, wykorzystywanej w rozproszonych środowiskach obliczeniowych. Dzięki temu założeniu, został osiągnięty brak integracji z warstwą *middleware*, co powoduje iż możliwe będzie zapewnienie poprawnej pracy implementacji proponowanego modelu z dowolną architekturą systemu kolejkowego czy innych komponentów specyficznych dla danego środowiska przetwarzania. Część opisowa pracy pomija zatem przedstawienie istniejących koncepcji architektur środowisk Grid, a używane pojęcie „Grid" odnosi się do infrastruktury przetwarzania rozproszonego bez jakichkolwiek założeń na temat stosowanego oprogramowania warstwy pośredniczącej.

Zarządzanie zasobami w zaproponowanym rozwiązaniu odbywa się na podstawie możliwości wprowadzonych dzięki zastosowaniu techniki wirtualizacji, działającej na poziomie systemu operacyjnego OS (ang. *Operating System*) i sieci komputerowej. Polityki zarządzania zasobami bazują zatem na regułach współdzielenia zasobów, wyizolowanych poprzez wirtualizację tego poziomu i nie uwzględniają analogicznych mechanizmów dostępnych w warstwie pośredniczącej *middleware*. Tak przyjęty model zasobów jest wystarczający dla systemu, którego głównym celem jest maksymalizacja wydajności działania rozproszonych aplikacji obliczeniowych. Dlatego przedstawiona w niniejszej pracy implementacja polityki zarządzania Wirtualną Organizacją sprowadza się jedynie do aspektów związanych z zarządzaniem zasobami obliczeniowymi i komunikacyjnymi, z pominięciem funkcjonalności umożliwiającej np. obsługę użytkowników czy harmonogramowanie dostępu do specjalistycznych urządzeń.

1.5. Osiągnięcia autora

Do najważniejszych osiągnięć pracy — zdaniem jej autora — należą:

- przeprowadzenie krytycznej analizy istniejących koncepcji zastosowania technik wirtualizacji systemu operacyjnego oraz sieci komputerowej w środowiskach typu Grid,
- praktyczna ewaluacja właściwości mechanizmów udostępnianych poprzez zastosowanie technik wirtualizacji, ze szczególnym naciskiem na oszacowanie wprowadzanych przez nie narzutów,

- zaproponowanie architektury systemu zarządzania zasobami zarówno obliczeniowymi jak i komunikacyjnymi z wykorzystaniem mechanizmów dostępnych dzięki zastosowaniu parawirtualizacji,
- implementacja, w oparciu o proponowany model, systemu kreowania wirtualnych Gridów i zarządzania ich konfiguracją,
- zaproponowanie i implementacja ogólnego zestawu interfejsów dla elementów systemu odpowiedzialnych za zarządzanie zwirtualizowanymi zasobami,
- implementacja aplikacji służącej do kontroli i wizualizacji aktualnego stanu systemu,
- wybór nowoczesnych narzędzi inżynierii oprogramowania i praktyczne wykazanie ich przydatności do budowy efektywnego i stabilnego rozwiązania,
- wykonanie praktycznych testów implementacji systemu pod kątem efektywności jego działania oraz ujemnych cech całości rozwiązania w odniesieniu do rezultatów uzyskiwanych w klasycznych środowiskach Grid,

Autor wykazał przydatność zastosowania technik parawirtualizacji w środowiskach obliczeniowych, które mimo przedstawionych w pracy ograniczeń, umożliwiają tworzenie wirtualnych Gridów oraz skuteczne zarządzanie ich zasobami. Tym samym została wykazana prawdziwość postawionej wcześniej tezy pracy.

1.6. Struktura pracy

Niniejsza rozprawa została podzielona na następujące rozdziały:

Rozdział 1 przedstawia motywację podjęcia prac, wprowadza definicję kluczowych pojęć a także zawiera tezę, cel i zakres rozprawy.

Rozdział 2 zawiera przegląd istniejących standardów oraz technologii związanych z problemami budowy wirtualnych Gridów, wirtualizacji systemów komputerowych oraz technik i metod tworzenia wirtualnej sieci. W rozdziale tym została również przedstawiona dyskusja możliwych zastosowań technik wirtualizacji. Informacje te stanowią tło do dalszych rozważań i badań zawartych w pozostałych rozdziałach niniejszej dysertacji.

Rozdział 3 zawiera dyskusję problemu efektywnego zarządzania zasobami w środowiskach Grid oraz przedstawia funkcjonalne i niefunkcjonalne wymagania, które system zarządzania powinien spełniać.

Rozdział 4 przedstawia formalny model systemu zarządzania zasobami w środowiskach Grid. Rozdział ten zawiera również rozważania na temat różnych koncepcji budowy takich systemów.

Rozdział 5 poświęcony został w całości przedstawieniu koncepcji budowy systemu zarządzania zasobami w środowiskach Grid. Opis ten przestawiony został w odniesieniu do wyróżnionych warstw funkcjonalnych architektury systemu.

Rozdział 6 zawiera szczegóły rozwiązań programowych użytych przy implementacji systemu.

Rozdział 7 opisuje ewaluację opracowanej koncepcji systemu oraz weryfikację spełnienia przyjętych w rozdziale 4 wymagań. Zostały w nim przedstawione wyniki jakościowej i ilościowej oceny zaproponowanej architektury oraz efektywności jej implementacji.

Rozdział 8 stanowi podsumowanie i wnioski z pracy oraz propozycje dalszego rozwoju.

Dodatki zawierają bibliografię, rozwinięcia używanych w pracy skrótów, definicje zawartych w pracy pojęć oraz skorowidz. Zostały przedstawione także przykładowe polityki przydziału zasobów oraz interfejsy ważniejszych komponentów systemu.

Rozdział 2

Technologie leżące u podstaw pracy

Rozdział ten stanowi tło dla dalszych prac i rozważań. Przedstawia aktualny stan prac związanych z technikami wirtualizacji oraz systemami Grid.

Przedstawione w niniejszym rozdziale informacje rozpoczyna przegląd istniejących koncepcji i rozwiązań z zakresu technik wirtualizacji. Zostały opisane ich możliwości ze szczególnym naciskiem na cechy i funkcje, których zastosowanie w systemach Grid byłoby pożądane. Wybór opisanych rozwiązań był podyktowany koniecznością zapewnienia wsparcia dla platformy Linux oraz dostępnością implementacji na licencji gwarantującej dostęp do źródeł projektu.[1] Rozdział zawiera podsumowanie możliwości tych technologii.

Z punktu widzenia wykorzystania systemów wirtualizacji w środowiskach Grid istotne są aspekty związane z wirtualizacją sieci komputerowej. Dlatego szczególną uwagę zwrócono na opis technik budowy wirtualnej infrastruktury sieciowej umożliwiającej komunikację wirtualnych maszyn.

2.1. Wirtualizacja systemów komputerowych

Naturalna ewolucja i konieczność pokonywania coraz to nowych barier technologicznych w procesie produkcji układów półprzewodnikowych, doprowadziła w obecnych czasach do pewnego paradoksalnie niezwykłego zjawiska. Współczynnik oferowanej mocy obliczeniowej [51] do ceny urządzenia obniżył się do tego stopnia, że łączenie w klastry masowo produkowanych komputerów bywa często alternatywą dla zakupu jednego superkomputera. Co więcej, można dzięki posiadanej mocy oraz wobec faktu, iż przy pracy z typowymi aplikacjami nie jest ona w pełni wykorzystywana [87], pokusić się o chęć stworzenia z puli niewykorzystanych zasobów wydzielonego wirtualnego komputera.

Separacja zasobów sprzętowych

Idea separacji zasobów jest bardzo często wykorzystywana. Prawie od zawsze istniała taka możliwość w instalacjach superkomputerów. Technologia ta pozwala aby na jednym komputerze było równocześnie uruchomionych wiele systemów operacyjnych. Pierwsze prace nad wirtualizacją platformy zostały rozpoczęte w latach sześćdziesiątych ubiegłego wieku przez połączenie wysiłków IBM (ang. *International Business Machines*) i MIT (ang. *Mas-*

[1] System Linux jest systemem operacyjnym najczęściej używanym do organizacji obliczeń rozproszonych [24], a dostęp do źródeł oprogramowania ułatwia rozwiązywanie wielu problemów, nawet w przypadku braku kompletnej dokumentacji projektu.

sachusetts Institute of Technology) nad projektem M44/44X.[2] Niedługo później IBM zaczął oferować możliwość tworzenia i zarządzania tzw. pseudo-komputerami w linii swoich systemów operacyjnych. Jednak w większości przypadków, jak w opracowanym przez IBM dla systemów klasy Mainframe — LPAR (ang. *Logical Partition*)[3] czy w rozwiązaniach firmy HP (ang. *Hewlett-Packard*) oraz technologii promowanej przez SGI (ang. *Silicon Graphics, Inc.*) — NUMA (ang. *Non-Uniform Memory Access*), była to koncepcja budowy oddzielnych instancji za pomocą autonomicznych elementów infrastruktury (płyt procesorów, płyt pamięci czy układów we/wy).

Podejście to równoważne konfiguracji fizycznie rozdzielonych serwerów znakomicie sprawdzało się dla systemów, dla których wymagano bardzo wysokiego poziomu bezpieczeństwa[4], gdyż oferowało kompletną separację działania każdej instancji, mimo iż złożone one były z podzespołów jednego komputera. Dodatkowo rozwiązanie to pociągało za sobą konieczność instalacji wielu kopii systemu operacyjnego po jednej dla każdej instancji „wirtualnego" komputera. Koncepcja ta posiadała szereg wad wynikających z ograniczeń, które wprowadzała: można je tworzyć tylko na specjalnie do tego celu zaprojektowanych architekturach, liczba możliwych instancji jest sprzętowo ograniczona, nie możliwa jest równomierna dystrybucja obciążenia, rozwiązanie jest nieopłacalne ekonomicznie jeśli istnieje potrzeba stworzenia wielu wirtualnych instancji.

Wirtualizacja programowa

Lata 80-te i 90-te ubiegłego wieku przyniosły jednak wyhamowanie zastosowań wirtualizacji. Pojawiły się nowoczesne systemy operacyjne umożliwiające równoczesne uruchamianie wielu procesów. Równoczesny spadek cen popularnego sprzętu komputerowego spowodował odejście od architektur typu MainFrame. Wszystko to wraz z brakiem wsparcia do sprzętowej wirtualizacji w masowo produkowanym sprzęcie sprawiło, iż zaczęto postrzegać wirtualizację jako historyczną ciekawostkę. Lata 90-te przyniosły jednakże odwrócenie tych tendencji. Ośrodki naukowe coraz częściej spoglądały na wirtualizację jako na technologię umożliwiającą eliminację problemów pojawiających się przy budowie i użytkowaniu dużych instalacji obliczeniowych. Systemy klasy MPP (ang. *Massively Parallel Processing*) były trudne w oprogramowaniu i nie można było stosować w nich istniejących systemów operacyjnych.

Coraz częściej stosowanym podejściem do wirtualizacji zaczęła być metoda bazująca na oprogramowaniu, czyli budowa instancji wirtualnego komputera na warstwie emulacji. Istnieje szereg podejść umożliwiających za pomocą dedykowanego oprogramowania udostępnianie iluzji rzeczywistego komputera. Dzięki czemu możliwe jest uruchomienie w wirtualnym komputerze dowolnego oprogramowania użytkowego bez koniecznych do wykonania, przy innych tego typu rozwiązaniach, adaptacji na poziomie kodu źródłowego. Dostępne implementacje wirtualizacji możemy sklasyfikować według następujących rozłącznych kategorii:

[2] M44/44X był eksperymentalnym systemem komputerowym opracowanym przez centrum naukowe firmy IBM — *Thomas J. Watson Resarch Center*, w stanie Nowy Jork. System ten bazował na projekcie IBM 7044 i umożliwiał symulację wielu maszyn 7044 za pomocą rozwiązań sprzętowo-programowych.

[3] Rozszerzonych przez technologię *Dynamic LPAR* dla procesorów POWER4 i POWER5, umożliwiającą na stworzenie 254 instancji LPAR dla pojedynczego wieloprocesorowego serwera (maksymalnie 10 instancji na procesor) z możliwością dynamicznego przenoszenia zasobów.

[4] LPAR firmy IBM posiada certyfikat EAL5 (ang. *Evaluation Assurance Level 5*).

- Pełna wirtualizacja lub emulacja — oprogramowanie emuluje (udaje) kompletny sprzęt pozwalając na uruchomienie niezmodyfikowanego systemu operacyjnego (w wielu instancjach równocześnie), również możliwe jest uruchamianie oprogramowania skompilowanego dla innych architektur procesorów (poprzez dynamiczną rekompilację kodu na daną architekturę). Do przykładów implementacji zaliczyć można: Virtual PC, Qemu, Virtual Iron, VMware Workstation, Mac-on-Linux, Win4Lin Pro.

- Parawirtualizacja lub wirtualizacja lekka (*LightWeight Virtualization*) — wirtualna maszyna nie emuluje sprzętu, lecz w zamian implementuje obsługę specjalnego API, które jest używane przez zmodyfikowane systemy operacyjne uruchomione w wirtualnych maszynach.[5] API to służy do wykonywania operacji związanych z podziałem fizycznych zasobów pomiędzy maszyny wirtualne. Oprogramowanie dokonujące rozdziału zasobów (implementacja wspomnianego wcześniej API nazywana jest często *hypervisor* (lub *hypercall* w implementacji Xen). Podejście to znacznie mniej obciąża zasoby fizycznego komputera, gdyż oprogramowanie nie implementuje kompletnej iluzji sprzętu a jedynie zapewnia rozłączny dostęp do niektórych fizycznych komponentów emulując tylko niektóre. Przykładami implementacji są: Xen, Parallels Workstation, Enomalism, VMware ESX Server i Win4Lin 9x.

- Wirtualizacja poziomu systemu operacyjnego[6] — podejście to umożliwia uruchomienie wydzielonego środowiska współdzielącego zasoby systemu operacyjnego komputera (np. jest używane to samo jądro), na którym jest uruchomione. Rozwiązania takie można spotkać zarówno w systemach komercyjnych — przykładem jest architektura Zone w systemie Solaris 10, jak i systemach operacyjnych z rodziny „*Open Source*" — Linux-VServer, Virtuozzo, OpenVZ czy BSD-Jail.

- Maszyny wirtualne języków programowania — stworzone jako nieodłączna część środowisk programistycznych języków, których kompilacja jest dokonywana do tak zwanego „kodu pośredniego". Aby spełnić wymóg przenośności aplikacji napisanych w tych językach, musiało zostać stworzone oprogramowanie dokonujące tłumaczenia kodu pośredniego na instrukcje specyficzne dla danego systemu operacyjnego. Oprogramowanie to, tzw. „wirtualna maszyna", wykonuje interpretację[7] kodu pośredniego tworząc iluzję działania w wydzielonym środowisku. Przykładami języków stosujących koncepcję maszyny wirtualnej są: stworzona przez firmę Sun Microsystems — Java, oraz środowisko firmy Microsoft — .NET (język *C♯*).

Równoczesne uruchomienie kilku instancji systemu operacyjnego stwarza jednak szereg wymagań, których zapewnienie będzie gwarantowało efektywne działanie danej implementacji VM. Po pierwsze maszyny wirtualne muszą być izolowane — nie dopuszczalna jest

[5] Szczegółowy opis działania tej techniki wirtualizacji zostanie przedstawiony w dalszej części rozdziału na przykładach implementacji Xen (sekcja 2.3.5) i KVM (ang. *Kernel Virtual Machine*) (sekcja 2.3.4).

[6] W ten sposób bywają też często określane techniki pełnej wirtualizacji i parawirtualizacji. Jest to dopuszczalne, gdyż obie techniki udostępniają iluzję wirtualnego komputera (na poziomie zwykłego użytkownika systemu operacyjnego) gwarantując możliwość uruchamiania niezmodyfikowanych programów użytkowych.

[7] Języki interpretatywne są z reguły wolniejsze od oprogramowania kompilowanego do kodu natywnego dla danej platformy/systemu operacyjnego. Dlatego też w maszynach wirtualnych została dodana możliwość kompilacji kodu pośredniego do kodu natywnego, która przyśpiesza wykonanie kosztem przenośności.

sytuacja kiedy uruchomiony proces działający na jednej może negatywnie wpływać na działanie lub wydajność procesów z pozostałych wirtualnych maszyn. Problem ten jest szczególnie istotny gdy poszczególne maszyny są zarządzane przez użytkowników nie mających nawzajem do siebie zaufania. Kolejnym problemem jest konieczność zapewnienia wsparcia dla możliwości uruchamiania wielu różnych systemów operacyjnych aby móc zapewnić środowiska dla heterogenicznych aplikacji. Trzecim aspektem, którego uwzględnienie należy rozważyć przy tworzeniu implementacji VM jest narzut jaki będzie wprowadzać czyli jaki będzie spadek wydajności w stosunku do klasycznego podejścia.

Dalsze prace nad rozwojem technik wirtualizacji będą z pewnością koncentrować się nad zwiększeniem możliwości tych środowisk. Funkcje takie jak bezpieczeństwo (izolacja), migracja czy odtworzenie poprzedniego stanu są trudne w implementacji w obecnych systemach operacyjnych. Możliwość zapewnienia tej funkcjonalności w warstwie emulacji pozwoli, zachowując wsparcie dla istniejących aplikacji, uzyskanie jeszcze większej elastyczności użytkowania systemów komputerowych.

2.1.1. Izolacja zasobów wirtualnego komputera

Wzrost możliwości systemów operacyjnych, możliwy dzięki rozwojowi architektur sprzętowych, spowodował iż oprogramowanie systemowe stało się złożone i skomplikowane. Aby zredukować ewentualne straty związane ze złamaniem zabezpieczeń czy awarią systemu, administratorzy zaczęli stosować nowy model przetwarzania: jedna aplikacja — jeden fizyczny komputer. Podejście takie znacznie zwiększyło wymagania, zarówno sprzętowe jak i ilość prac koniecznych do utrzymania całej infrastruktury. Koncepcja przeniesienia wykonania aplikacji do wydzielonej wirtualnej maszyny oraz konsolidacja VM, tak aby wykorzystywały małą pulę fizycznych zasobów spowoduje znaczne zwiększenie efektywności wykorzystania całej dostępnej infrastruktury. Podejście to nie powoduje odejścia od koncepcji rozdzielenia przestrzeni wykonania dla kluczowych aplikacji a poprzez konsolidację zasobów dodatkowo zostają zmniejszone koszty związane z administracją całego systemu.

Separacja zasobów wirtualnego komputera, od zasobów komputera na którym jest on uruchomiony powinna być całkowita. Oprogramowanie, które jest uruchomione na komputerze wirtualnym nie powinno mieć dostępu do przestrzeni adresowej, stosu TCP/IP (ang. *Transmission Control Protocol/Internet Protocol*), mechanizmów komunikacji IPC (ang. *Inter-process Communication*) innych wirtualnych komputerów a także komputera fizycznego, na którym dana instancja wirtualnej maszyny VM jest uruchomiona. Separacja ta jest bardzo ważna, gdyż koncepcja separacji jest często wykorzystywana przez firmy z sektora ISP (ang. *Internet Service Provider*) jako mechanizm udostępniania serwisów, dzięki czemu można stworzyć wiele wirtualnych komputerów na jednej silnej maszynie, oszczędzając na kosztach obsługi, serwisowania itp. Każda wirtualna maszyna może być zarządzana przez inną organizację posiadającą pełne prawa administratora. Mechanizmy separacji nadają się również znakomicie do realizacji testów oprogramowania, tak aby łatwo można było realizować np. wielokrotne prześledzenie procesu instalacji a ewentualne problemy nie zakłócałyby działania komputera macierzystego.

Wiele różnych implementacji wirtualizacji było testowanych ze względu na narzut jaki wprowadzają w stosunku do systemu operacyjnego uruchomionego natywnie. Informacje te nie mówią jednak nic na temat właściwości wprowadzanej wraz z wirtualizacją izolacji dostępu do zasobów dla maszyny wirtualnej. Większość implementacji wirtualizacji pozwala na definiowanie parametrów ograniczających konsumpcję zasobów przez poszczególne VM.

Ograniczenia te umożliwiają stworzenie izolowanego środowiska, którego działanie pozostanie niezakłócone bez względu na generowane obciążenie pochodzące od innych VM działających w obrębie tego samego fizycznego hosta.

Poziom dostępnej izolacji zależy silnie od zastosowanej w danej implementacji techniki wirtualizacji. I tak systemy wirtualizacji sprzętowej umożliwiają uzyskanie stuprocentowej separacji, podczas gdy w rozwiązaniach programowych obserwowany jest wpływ innych VM powodujący degradację wydajności środowiska. Również przy porównaniu technik wirtualizacji programowej, lepszy poziom izolacji jest uzyskiwany w implementacjach pełnej wirtualizacji w odniesieniu do parawirtualiazacji. Obie techniki umożliwiają jednak osiągnięcie możliwości kontynuowania pracy VM w przypadku gdy któraś z wirtualnych maszyn ulegnie awarii.

W większości wykonywanych testów izolacji [23, 88] brano pod uwagę cztery główne rodzaje aktywności mające wpływ na uruchomione równolegle instancje VM:

- duże obciążenie CPU (ang. *Central Processing Unit*),
- duże wykorzystanie pamięci,[8]
- intensywne korzystanie z lokalnego I/O (ang. *Input/Output*),
- generowanie dużej ilości komunikacji sieciowej.

Istniejąca w programowych technikach separacji zasobów degradacja może być minimalizowana za pomocą dostępnych w systemach operacyjnych mechanizmów. Zastosowanie techniki dwupoziomowego przydziału zasobów lub bezpiecznego dostępu do zasobów [39] umożliwia uzyskanie poziomu izolacji dostępnego w wirtualizacji sprzętowej.

Dzięki izolacji można zagwarantować dla procesów systemowych uruchomionych wewnątrz systemu operacyjnego, dostępność zasobów na wymaganym poziomie. Zasoby, których poziom będzie zagwarantowany umożliwiają uzyskanie przez aplikację określonego poziomu wykonania, niezmiennego przez cały czas jej działania bez względu na obciążenie generowane przez inne zadania współdzielące tę samą infrastrukturę sprzętową.

Odrębny problem wirtualizacji stanowi zapewnienie izolacji również na poziomie komunikacji sieciowej. Istotne jest aby komunikacja sieciowa pochodząca od różnych VM również była separowana. Oznacza to, iż wirtualne komputery komunikujące się zwykle przez jeden fizyczny interfejs sieciowy komputera macierzystego, nie będą miały dostępu do informacji przesyłanych przez inne VM.[9]

W tym przypadku ważne jest aby parametry komunikacji sieciowej mogły podlegać, podobnym jak w przypadku izolacji zasobów obliczeniowych, gwarancjom. Gwarancje te, których określenie będzie możliwe dzięki zastosowaniu technik wirtualizacji, będą miały wpływ na komunikację sieciową realizowaną jedynie w obrębie fizycznego węzła. Różnorodność stosowanych w praktyce technologii budowy sieci i ich właściwości powodują, iż gwarancja parametrów QoS dla komunikacji sieciowej pomiędzy VM nie zawsze jest możliwa.[10]

[8] Przez wykorzystanie pamięci, rozumie się zarówno fakt przydzielenia dużej ilości stron dla VM jak i częste wykonywanie operacji modyfikujących zajmowane obszary pamięci.

[9] Możliwe rozwiązania tego problemu przedstawia sekcja 2.2 zawierająca opis technik budowy wirtualnych topologii sieciowych używanych do komunikacji VM.

[10] Brak wsparcia wynikał będzie z braku możliwości gwarancji jakości usługi dla technologii sieciowych używanych do transmisji danych na ścieżce łączącej maszyny wirtualne.

2.1.2. Migracja maszyny wirtualnej

Migracja poziomu systemu operacyjnego[11], pomiędzy dwoma fizycznymi węzłami, jest jedną z ważniejszych możliwości dostępnych dzięki zastosowaniu wirtualizacji. Przeniesienie uruchomionego systemu operacyjnego bez zatrzymywania[12] jest kluczowym narzędziem w takich zastosowaniach jak obsługa przeciążenia poprzez dynamiczne rozłożenie obciążenia hostów, wykonywanie prac związanych z utrzymaniem i serwisowaniem sprzętu, czy konsolidacja serwisów.

Migracja całego systemu operacyjnego wraz ze wszystkimi uruchomionymi aplikacjami pozwala na uniknięcie wielu problemów pojawiających się przy realizacji migracji na poziomie pojedynczych procesów. Zastosowanie migracji poziomu VM pozwala na wyeliminowanie problemu ciągłej dostępności zasobów[13] [67], które muszą być zdalnie dostępne na hoście, z którego procesy były przeniesione. W przypadku migracji poziomu VM, host z którego nastąpiło przeniesienie wykonania przestaje być potrzebny i może być wyłączony.

Dodatkowo podczas migracji całej maszyny konieczne jest przeniesienie zawartości pamięci[14] używanej zarówno przez aplikacje jak i pamięć przydzieloną dla jądra. Powoduje to konsystentne przeniesienie wraz z VM informacji na temat np. otwartych plików czy aktywnych połączeń sieciowych, dzięki temu oprogramowanie sieciowe, korzystające z zasobów przenoszonej maszyny nie musi być restartowane.

Kolejną zaletą migracji na poziomie VM jest separacja uprawnień, która dostępna jest dzięki zastosowaniu wirtualizacji. Uprawnienia administratora działającego w obrębie danej maszyny nie są wymagane aby móc przeprowadzić jej poprawną migrację. Równoczesne zastosowanie wirtualizacji i migracji umożliwia znaczne usprawnienie zarządzania zasobami.[15]

Implementacje migracji

Migracja poziomu procesów była ważnym tematem badań od początku lat 80-tych [27, 75]. Jednak rezultaty tych prac nie miały szerokiego zastosowania w typowych, masowo wykorzystywanych aplikacjach. Do słabej popularności tych rozwiązań przyczyniła się konieczność zapewnienia dostępności zasobów węzła, z którego następowało przenoszenie OS. Dodatkowo, ujemną cechą migracji na poziomie procesów był spadek wydajności przenoszonych zadań. Rozwiązania takie jak np. *Sprite* [73] umożliwiły przenoszenie procesów kosztem przekazywania niektórych wywołań systemowych do węzła, na którym proces został pierwotnie uruchomiony.

Problemów zależności dla przenoszonych procesów nigdy nie udało się do końca rozwiązać w żadnej z dostępnych implementacji migracji na poziomie procesów. Nawet nowoczesne środowiska uruchomieniowe dla mobilnych aplikacji jak np. Java ME[16] czy .NET[17]

[11] Lub inaczej migracja VM czy migracja poziomu OS, to przeniesienie wykonania uruchomionej instancji systemu operacyjnego wraz ze wszystkimi aktualnie uruchomionymi programami.

[12] Występuje przerwa w działaniu, lecz w zależności od konfiguracji przenoszonego systemu powinna ona nie przekraczać kilkudziesięciu milisekund i wtedy może zostać niezauważona.

[13] Przykładem zależności mogą być deskryptory otwartych plików, segmenty pamięci dzielonej, semafory i inne lokalne struktury jądra używane przez procesy.

[14] Pamięć ta jest kopiowana etapami.

[15] Co zostanie pokazane w dalszej części niniejszej pracy.

[16] `http://java.sun.com/javame/index.jsp`

[17] `http://www.microsoft.com/net/overview.aspx`

zakładają mobilność urządzenia, gdyż przenoszenie aplikacji nawet pomiędzy różnymi instancjami maszyny wirtualnej np. JVM (ang. *Java Virtual Machine*) jest skomplikowane w realizacji.

Dopiero migracja poziomu systemu operacyjnego pozwala na eliminację zależności pomiędzy używanymi w procesie migracji maszynami. Migracja poziomu OS została wykorzystana w projekcie *Collective* [82] jako narzędzie wspomagające mobilnych użytkowników pracujących na różnych komputerach w różnym czasie. Podejście to zawierało pewne optymalizacje i umożliwiało przeniesienie instancji OS poprzez połączenie sieciowe o niskiej przepustowości (typowo było to ADSL (ang. *Asymmetric Digital Subscriber Line*)). Także prace w projektach „*Internet Suspend/Resume*" [52] oraz *µDenali* [99] zakładały zatrzymanie systemu operacyjnego na czas przenoszenia.

Migracja systemu operacyjnego „na żywo" była używana w systemie *NomadBios* [45]. Także nowoczesne techniki wirtualizacji takie jak VMware (rozwiązanie *VMotion*) czy projekt Xen umożliwiają przeniesienie systemu operacyjnego z minimalnym czasem zatrzymania. Przedstawione rozwiązania różnią się zastosowanymi technikami[18] usprawniającymi proces migracji powodując skrócenie czasu jego wykonania, zmniejszenie ilości koniecznych do przesłania danych, czy ograniczając wykorzystanie zasobów przez proces wykonujący migrację.

Migracja systemu operacyjnego uruchomionego w wirtualnym komputerze możliwa jest dzięki temu, iż w systemie działa oprogramowanie (nadzorca ang. *supervissor*) uruchomione na wyższym poziomie uprawnień w dostępie do zasobów komputera fizycznego. Nadzorca ten, dzięki temu ma dostęp do zawartości przestrzeni adresowej systemu operacyjnego VM. Dostęp ten umożliwia przeprowadzenie migracji, ponieważ migracja VM technicznie sprowadza się do synchronizacji zawartości pamięci używanej przez maszynę wirtualną pomiędzy dwoma fizycznymi komputerami.

Warunkiem koniecznym do przeprowadzenia migracji jest brak lokalnych zależności przenoszonego systemu w postaci lokalnie wykorzystywanej przestrzeni dyskowej, obszaru wymiany (ang. *swap space*) czy urządzeń I/O, których niedostępność na docelowym węźle uniemożliwi kontynuowanie pracy przenoszonej maszyny. Konieczna również jest zgodność (w przypadku parawirtualizacji) zarówno architektur sprzętowych jak i programowych (rodzaju i wersji systemu operacyjnego) fizycznych maszyn pomiędzy, którymi następuje migracja VM. Migracja wymaga też, aby host docelowy dysponował wolnymi zasobami[19] na poziomie wymaganym do kontynuacji działania przenoszonej maszyny.

Przeniesienie zawartości pamięci

Przeniesienie działającego systemu z jednego hosta na drugi może być realizowane na wiele sposobów, jednak w każdym przypadku zarówno czas trwania całego procesu jak i czas wyłączenia systemu powinny być jak najmniejsze. Są to jednak parametry zależne i nie możliwa jest jednoczesna minimalizacja obu. Proces przeniesienia może być podzielony na następujące rozłączne etapy [20]:

Etap „kopiowania wyprzedzającego" . Podczas gdy wirtualna maszyna kontynuuje

[18] Ze względu na komercyjny charakter produktu VMware oraz brak szczegółowej dokumentacji implementacji — brak jest informacji na temat efektywności rozwiązania *VMotion*.

[19] Takimi jak np. pamięć operacyjna, przestrzeń dyskowa, właściwości komunikacji sieciowej, itp.

pracę, podczas gdy niektóre strony pamięci są kopiowane przez sieć. Podczas wykonywania tego etapu może nastąpić obniżenie wydajności przenoszonej maszyny.

Etap „zatrzymania i kopiowania” . Maszyna wirtualna jest zatrzymana, zawartość pamięci jest kopiowana.[20] Działanie tego etapu jest widziane z poziomu użytkowników wykorzystujących daną VM jako wstrzymanie jej działania.

Etap „kopiowania na żądanie” . Maszyna jest przeniesiona i jeśli próbuje uzyskać dostęp do zawartości strony, która nie została przesłana — strona ta jest kopiowana. Podobnie jak przy etapie „kopiowania wyprzedzającego” może nastąpić (typowo większe) obniżenie wydajności.

Praktyczne implementacje działają z wykorzystaniem jednego lub maksymalnie dwóch z podanych wyżej etapów migracji. Przykładowo rozwiązania [82, 99] wykorzystują jedynie etap „**zatrzymania i kopiowania**” poprzez zatrzymanie maszyny i kopiowanie zawartości całej jej pamięci.

Inne możliwe podejście to kopiowanie na żądanie. W podejściu tym podczas krótkiego etapu zatrzymania, następuje kopiowanie jedynie stron pamięci zajętych przez struktury danych jądra. Następnie działanie VM jest wznowione i pozostałe strony są przesyłane przez sieć przy pierwszym odczycie. Podejście to [104] powoduje osiągnięcie krótkich czasów przerwy w działaniu kosztem wydłużenia czasu całego procesu migracji. Dodatkowo występuje znaczny spadek wydajności działania maszyny wirtualnej do czasu gdy większość stron nie zostanie przeniesiona.

Trzecim i najbardziej efektywnym podejściem jest zastosowane w implementacji projektu Xen [6] rozwiązanie, wykorzystujące wielokrotne wykonanie etapu „**kopiowania wyprzedzającego**” przed trwającym zwykle krótko etapem „**zatrzymania i kopiowania**”. Kopiowanie w cyklach polega na kopiowaniu w cyklu n jedynie stron pamięci zmienionych od czasu zakończenia cyklu $n-1$.[21] Dodatkowo możliwa jest optymalizacja tego procesu poprzez identyfikację dla każdej maszyny zestawu często modyfikowanych stron WWS (ang. *Writable Working Set*), których przesłanie w każdym cyklu byłoby nieefektywne. Jednak i w tym podejściu może być (przez użytkowników) odczuwalna degradacja wydajności działania przenoszonej maszyny spowodowana wzrostem obciążenia sieci i procesora przez trwający proces migracji.

Migracja lokalnych zasobów

Przeniesienie zawartości pamięci przez bezpośrednie kopiowanie nie jest jedyną czynnością wymaganą podczas przeprowadzania migracji VM. Również obsługa połączeń sieciowych oraz przestrzeni dyskowej musi być wzięta pod uwagę.

Otwarte połączenia sieciowe powinny móc być zachowane przez przenoszoną wirtualną maszynę bez zastosowania mechanizmu przekazywania z wykorzystaniem oryginalnego węzła. W topologiach, gdzie węzły pomiędzy którymi następuje migracja znajdują się w tej samej sieci LAN, wystarczy prosty mechanizm aktualizacji tablic przełączania *switch*y

[20] Na tym etapie migracji nie wszystkie strony muszą być przesłane.
[21] W pierwszym cyklu są przesłane wszystkie strony.

i tablic protokołu ARP (ang. *Address Resolution Protocol*).[22] Migracja pomiędzy węzłami z różnych sieci LAN jest możliwa z użyciem dodatkowych mechanizmów ukrywających strukturę sieci pomiędzy komputerami.[23]

Najczęstszym obejściem problemu dostępności identycznej konfiguracji przestrzeni dyskowej, na obu węzłach pomiędzy którymi wykonywana jest migracja, jest zastosowanie technologi NAS (ang. *Network Attached Storage*). Połączenie migracji pamięci wraz z migracją przestrzeni dyskowej jest możliwe aczkolwiek powoduje znaczny wzrost ilości kopiowanych danych. Optymalizacja tego procesu zakłada wykorzystanie obrazów dysków używanych z zastosowaniem metody CoW (ang. *Copy-on-Write*) wykorzystującej oddzielną przestrzeń. Podczas migracji nie następowałoby przesłanie całego systemu plików a jedynie (niewielkich objętościowo)[24] zmian w stosunku do dostępnego na obu węzłach obrazu OS.

Narzuty migracji

Efektywność migracji zależy głównie od kilku podstawowych czynników. Główny czynnik to rozmiar przydzielonej dla danej VM ilości pamięci operacyjnej. Jednak ilość przesyłanych danych, ze względu na kopiowanie iteracyjne będzie większa.[25] Nie bez znaczenia jest efektywność komunikacji sieciowej, której wydajność określa całkowity czas procesu przeniesienia. Zastosowany w implementacji Xen mechanizm identyfikacji często modyfikowanych stron pamięci, pozwala na znaczne zmniejszenie czasu migracji oraz precyzyjne oszacowanie czasu przerwy. Zastosowanie mechanizmu identyfikacji WWS w systemie Xen pozwala średnio na czterokrotne [20] skrócenie czasu przerwy w działaniu VM.

Skrócenie czasu zatrzymania VM poprzez identyfikację WWS jest możliwe, gdyż większość uruchomionych w systemie operacyjnym aplikacji posiada zwykle mały zestaw bardzo często modyfikowanych stron.[26] Czas pomiędzy poszczególnymi modyfikacjami tych stron jest mniejszy niż czas potrzebny do przesłania ich przez sieć. Dlatego też jedyną opcją jest ich przesłanie w ostatnim cyklu (podczas zatrzymania maszyny wirtualnej).

2.2. Wirtualizacja sieci komputerowej

Zastosowanie wirtualizacji poziomu systemu operacyjnego pozwala na eliminację wielu problemów związanych z przydziałem zasobów, które pojawiają się przy tworzeniu rozproszonych środowisk obliczeniowych. Elementy składowe tych środowisk oraz wykonywane aplikacje wymagają jednak do poprawnego działania dostępności efektywnej i dwukierunkowej komunikacji sieciowej. Kluczowym zadaniem jest zapewnienie możliwości komunikacji wirtualnym maszynom tworzącym infrastrukturę typu Grid a rozproszonych na wiele fizycznych hostów.

[22] Np. poprzez wysłanie odpowiedzi protokołu ARP tzw. *Unsolicited ARP*. Jednakże mechanizm ten nie będzie działał poprawnie w topologiach, w których zastosowano wysokopoziomowe mechanizmy bezpieczeństwa jak np. rozwiązanie firmy Cisco Systems — *Dynamic ARP Inspection* [19].

[23] Opis tych mechanizmów znajduje się w sekcji 2.2.

[24] Większość implementacji mechanizmu CoW dla urządzeń blokowych tworzy tzw. pliki rzadkie (ang. *sparse file*), których rozmiar kilkakrotnie przekracza zajmowaną na dysku przestrzeń — dlatego mechanizm przesyłania tych danych musi mieć implementację obsługi tego typu plików.

[25] Może się bowiem zdarzyć, iż ze względu na częste modyfikacje, pewne strony pamięci będą musiały być przesyłane wielokrotnie. Efekt ten występuje szczególnie w systemach bez identyfikacji WWS.

[26] Zwykle są to strony przechowujące stos, czy też często modyfikowane zmienne lokalne.

Silne rozproszenie fizycznych węzłów, budowanie infrastruktury z elementów należących i administrowanych przez różne organizacje oraz złożoność topologii i konfiguracji sieci Internet używanej często jako uniwersalne medium komunikacyjne — powoduje trudności w zapewnieniu efektywnej komunikacji wirtualnym węzłom. Dlatego podjętych było wiele prac mających na celu stworzenie środowiska, umożliwiającego elastyczne tworzenie topologii sieciowych budowanych z wirtualnych maszyn [40–42, 48, 57, 59, 65, 94].

Stosowanie technik ograniczających wykorzystanie dostępnej przestrzeni adresowej[27] jak NAT (ang. *Network Address Translation*) czy DHCP (ang. *Dynamic Host Configuration Protocol*) oraz coraz częstsze umieszczanie na styku z siecią Internet urządzeń typu Firewall powoduje trudności przy zapewnieniu komunikacji typu każdy-z-każdym. Dostępność jedynie ograniczonej funkcjonalności nawiązywania połączeń przy braku możliwości ich odbierania, powoduje odejście od oryginalnej koncepcji identyfikacji każdego węzła sieci Internet jako potencjalnie użytecznego elementu udostępniającego zasoby. Istnieją wprawdzie protokoły umożliwiające obejście ograniczeń technologii NAT/Firewall (jak np. [80, 86]) ale ich zastosowanie wymaga modyfikacji istniejących bibliotek systemowych oraz aplikacji sieciowych.

Techniki wirtualizacji sieci dla środowisk Grid mają za zadanie dostarczenie aplikacji jej natywnego środowiska sieciowego, bez względu na kształt i stopień skomplikowania wykorzystywanej topologii fizycznej. Przedstawione wcześniej komplikacje wprowadzane przez NAT/Firewall mogą zostać wyeliminowane poprzez dodanie nowej warstwy abstrakcji udostępniającej uniwersalne usługi komunikacji. Sieci wirtualne umożliwiają uzyskanie pełnej dwukierunkowej osiągalności rozproszonym wirtualnym komputerom za pomocą stosu protokołów TCP/IP.

Poprzez wprowadzenie dodatkowych mechanizmów modyfikacji komunikacji, możliwe będzie budowanie z puli wirtualnych komputerów dowolne wirtualne topologie sieciowe.[28] Topologie te pozwolą na tworzenie modelowych systemów rozproszonych oraz umożliwią efektywne uruchamianie aplikacji w tych systemach. Jednak konfiguracja komputerów wirtualnych, tak aby można było je grupować w oddzielne sieci, aby sieci te można było łączyć za pomocą routerów (również wirtualnych) i aby możliwe było stworzenie topologii wraz z pełną konfiguracją QoS[29] jest zadaniem złożonym. Złożoność ta wynika głównie z faktu wykorzystywania przez poszczególne implementacje wirtualizacji różnych niekompatybilnych metod zapewnienia komunikacji sieciowej.

2.2.1. Lokalna komunikacja sieciowa

Umożliwienie lokalnej komunikacji sieciowej dla VM jest możliwe na wiele sposobów, a wybór konkretnego zależy od dostępności danego rozwiązania w stosowanej implementacji wirtualizacji.[30] Dla platformy Linux są dostępne m.in. następujące rozwiązania:

[27] Z uwagi na wyczerpywanie się przestrzeni adresowej protokołu IPv4 (ang. *Internet Protocol version 4*).

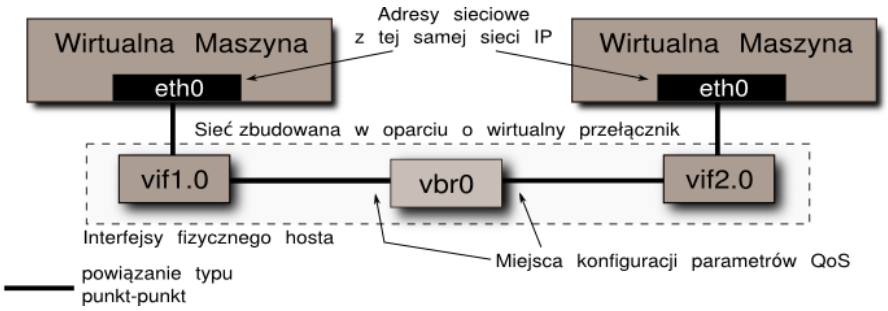
[28] W większości przypadków wystarcza topologia sieci lokalnej LAN, jednak aby móc zapewnić takie właściwości jak wsparcie dla QoS czy filtrację komunikacji wymagane będzie zastosowanie np. routingu, podziału na podsieci, itp.

[29] Czyli symulacja ograniczeń podobnych do ograniczeń na CPU i pamięć operacyjną definiowanych w momencie uruchamiania komputera wirtualnego.

[30] Listę dostępnych technik w różnych popularnych implementacjach wirtualizacji przedstawia sekcja 2.3.6.

- Komunikacja interfejsem typu „*ethertap*" — czyli dołączenie interfejsu wirtualnego[31] jako nasłuchującego na fizycznym interfejsie komputera fizycznego. W tym przypadku konieczne jest użycie wolnego adresu z sieci, do której podłączony jest komputer fizyczny. Komunikacja pozostałych fizycznych maszyn z wirtualną odbywa się bezpośrednio i komputery te nie są w stanie rozróżnić czy komunikacja odbywa się z komputerem wirtualnym czy też nie. W podejściu tym często wykorzystywany jest mechanizm Proxy-ARP [17].

- Komunikacja za pomocą powiązanych interfejsów wirtualnych — każdy wirtualny interfejs posiada swój odpowiednik na liście interfejsów komputera fizycznego, z którym jest podłączony bezpośrednio łączem typu punkt-punkt (rysunek 2.1). Komunikacja z innymi węzłami jest możliwa tylko dzięki zastosowaniu dodatkowych mechanizmów (np. technik routingu lub przełączania oraz translacji adresów NAT — co powoduje, że komunikacja ta widziana ze zdalnych urządzeń, pochodzi z fizycznego komputera). Podejście to oferuje największe możliwości konfiguracyjne, gdyż stosując routing, przełączanie i NAT możemy budować dowolne topologie sieciowe z użyciem wirtualnych komputerów. Dodatkowo, możliwe jest zastosowanie dla interfejsów typu punkt-punkt dowolnych ustawień takich jak filtracja (Firewall) czy kolejkowanie i kształtowanie ruchu (QoS).

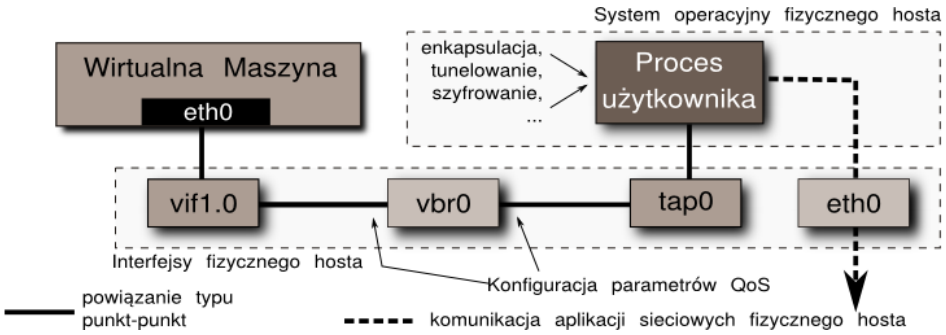


Rysunek 2.1. Tworzenie wirtualnej sieci z wykorzystaniem techniki przełączania

- Użycie interfejsów logicznych `tap/tun` — jądro systemu operacyjnego Linux udostępnia sterowniki dla dwóch rodzajów wirtualnych interfejsów sieciowych. Pierwszy z nich — interfejs typu `tap` symuluje interfejs Ethernet i operuje na warstwie 2 modelu OSI/ISO (ang. *Open Systems Interconnection/International Organization for Standardization*). Użycie tego interfejsu umożliwia odbieranie informacji w postaci kompletnych ramek Ethernet. Z kolei interfejs `tun` pozwala na symulację urządzenia działającego na warstwie 3 (sieciowej) poprzez zastosowanie komunikacji poziomu pakietów IP (ang. *Internet Protocol*). Typowe zastosowanie interfejsu `tap` to stworzenie sieci w oparciu o mechanizm przełączania, podczas gdy urządzenie `tun` nadaje się do implementacji routingu.

  Wszystkie dane przekazane do wirtualnych urządzeń `tap/tun` są przekazywane do uruchomionego w przestrzeni użytkownika programu, który dokonał wcześniejszego dołączenia do w/w urządzenia (rysunek 2.2)[32].

[31] Interfejs ten dla oprogramowania komputera wirtualnego jest widziany jako zwykły interfejs fizyczny.
[32] Obsługa urządzeń `tap/tun` odbywa się przez standardowy Unixowy mechanizm abstrakcji sprzętu czyli



Rysunek 2.2. Tworzenie wirtualnej sieci z wykorzystaniem technik tunelowania komunikacji

- Komunikacja za pomocą mechanizmów IPC — komunikacja ta jest możliwa tylko w niektórych przypadkach i jest rzadko stosowana ze względu na ograniczenia, które wprowadza.[33]
- Oprogramowanie sieciowe działające w przestrzeni użytkownika — interfejsy `tun/tap` umożliwiają odbieranie i wysyłanie pakietów obsługiwane przez oprogramowanie działające w trybie użytkownika. Urządzenia te są widziane jako interfejsy sieciowe działające w trybie punkt-punkt, które zamiast odbierać informacje z fizycznego medium, odbierają je z oprogramowania użytkownika, z kolei wszystko co jest wysyłane przez ten interfejs jest przekazywane do oprogramowania obsługującego dany interfejs.

  Obsługa urządzeń `tun/tap` za pomocą oprogramowania umożliwia stworzenie wirtualnej sieci poprzez odbieranie danych z maszyny wirtualnej i wysyłania ich za pomocą fizycznego interfejsu (rysunek 2.2).
- Routing pomiędzy interfejsami fizycznymi i wirtualnymi, a także interfejsami pomocniczymi używanymi do implementacji polityki zarządzania ruchem sieciowym (rysunek 2.3).

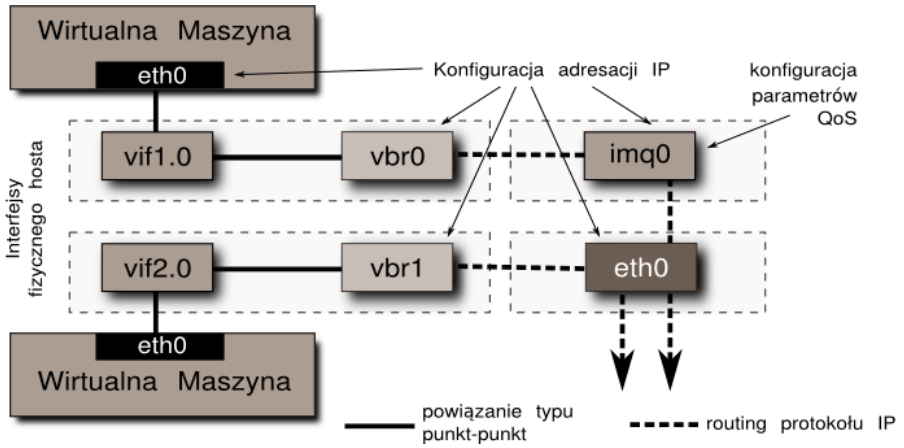
Dzięki dostępności w/w rozwiązań budowanie sieci lokalnych z użyciem maszyn wirtualnych uruchomionych w obrębie tego samego komputera (rysunek 2.1) nie stanowi problemu. Jednak technologie te nie pozwalają na realizację komunikacji po sieci lokalnej wirtualnych komputerów uruchomionych na różnych węzłach znajdujących się w różnych sieciach LAN. Przekazywanie pomiędzy VM ruchu *broadcast* czy *multicast* w takiej wirtualnej sieci lokalnej, bez wprowadzenia technik tunelowania i enkapsulacji oraz mechanizmów wspierających automatyczne tworzenie tych połączeń nie będzie możliwe.

Techniki separacji ruchu

Innym istotnym problemem jest utrzymanie na poziomie sieciowym separacji, którą daje nam wirtualny komputer. Separacja ta znika w przypadku używania istniejących rozwiązań

tzw. plik urządzenia, który obsługiwany jest przez zwykłe systemowe funkcje `read()/write()`.

[33]Większość mechanizmów komunikacyjnych IPC (z wyjątkiem gniazd) wymaga aby komunikujące się procesy/wątki działały w obrębie jednego systemu operacyjnego. Dodatkowo interfejs programistyczny IPC jest silnie zależny od rodziny systemu operacyjnego, utrudniając tworzenie przenośnych aplikacji.



Rysunek 2.3. Tworzenie wirtualnej sieci z wykorzystaniem routingu w warstwie sieciowej

zapewniających dostęp do sieci dla wirtualnych komputerów. W większości przypadków aby taką separację osiągnąć wymagane jest ograniczenie pewnych rodzajów komunikacji (jeżeli urządzenia znalazłyby się w różnych sieciach lokalnych) lub należy wprowadzić wsparcie ze strony infrastruktury sieciowej.

Technologie i rozwiązania sieciowe, których zastosowanie mogłoby umożliwić uzyskanie rozdzielenia ruchu sieciowego dla VM uruchomionych w obrębie jednego fizycznego węzła i komunikujących się przez pojedynczy fizyczny interfejs sieciowy przedstawia tabela 2.1[34].

Przy ocenie możliwości dostępnych metod separacji komunikacji należy odnieść się do funkcjonalności wymaganej w konkretnym zastosowaniu technik wirtualizacji. Jeżeli wymagany jest wysoki poziom bezpieczeństwa komunikacji i zapewnienie poufności transmitowanych danych to najlepszymi metodami będą techniki wydzielonych interfejsów fizycznych, sieci wirtualne w warstwie 2 OSI/ISO oraz enkapsulacja za pomocą technologii wspierających szyfrowanie jak np. protokół IPSec (ang. *Internet Protocol Security*) czy tunelowanie SSH (ang. *Secure SHell*). Z punktu widzenia wydajności komunikacji istotny będzie narzut w postaci dodatkowych danych związanych z enkapsulacją oraz dodatkowy czas konieczny do nawiązania połączenia. W tej kategorii najbardziej efektywnymi metodami są techniki tagowania oraz sieci wirtualne. Największy narzut wprowadzany jest przez metody wykorzystujące koncepcje P2P (ang. *Peer To Peer*) [40, 41] oraz enkapsulacja za pomocą protokołu TCP (ang. *Transmission Control Protocol*). Zwiększenie niezawodności oraz osiągalności wirtualnych maszyn szczególnie w sieciach rozległych WAN, wspierają rozwiązania działające w warstwach 5–7 czyli takie rozwiązania jak sieci MPLS (ang. *Multiprotocol Label Switching*), czy technologie VRF (ang. *Virtual Routing and Forwarding*)/MBGP (ang. *Multiprotocol BGP*).

Przy rozważaniu zastosowania technik separacji komunikacji sieciowej, należy również uwzględnić możliwość zdalnego dostępu do konsoli (lub interfejsu graficznego) wirtualnego komputera. Dostęp ten powinien być również możliwy w trybie wydzielonym, czyli być możliwy bez względu na stan i konfigurację interfejsów sieciowych VM. Istniejące implementacje wirtualizacji jak np. Xen mają możliwość udostępniania interfejsu admini-

[34] Tabela ta przedstawia możliwe rozwiązania uszeregowane zgodnie z warstwami modelu referencyjnego OSI/ISO.

Warstwa OSI	Technologia/metoda	Właściwości
Warstwa fizyczna	wydzielone interfejsy fizyczne	Wyposażenie fizycznego węzła w odpowiednią liczbę interfejsów sieciowych tak aby możliwe było powiązanie VM z interfejsem relacją jeden-do-jeden.
Warstwa łącza-danych	sieci wirtualne VLAN (ang. *Virtual LAN*)	Zastosowanie metody separacji ruchu generowanego przez wirtualne komputery na styku pomiędzy fizycznym interfejsem sieciowym a siecią LAN. Metody umożliwiają uzyskanie separacji w obrębie sieci w warstwie 2. Najczęściej będzie to wykorzystanie sieci wirtualnych, lub zwykłe przesyłanie unicastu czy broadcastu (tunelowanie) w warstwie łącza danych. Zapewnienie możliwości wydzielonej komunikacji w obrębie systemu autonomicznego (AS (ang. *Autonomic System*)) w warstwie 3.
	protokoły 802.1q/p (lub ISL (ang. *Inter-Switch Link*))	
Warstwa sieciowa	tagowanie w warstwie 3 (IPv4 ToS (ang. *Type of Service*)/IP Precedence, IPv6 (ang. *Internet Protocol version 6*) flow-label, czy etykiety MPLS [79])	
	enkapsulacja IPSec	
	routing unicastowy lub/i multicastowy	
Warstwa transportowa	enkapsulacja TCP/UDP (ang. *User Datagram Protocol*) w wersji unicastowej (TCP) i muliticastowej (UDP)	Dystrybucja informacji o adresacji (identyfikatorze sieci lokalnej) w celu umożliwienia dołączenia wirtualnego węzła z poza lokalnego systemu autonomicznego.
Warstwy sesji, prezentacji i aplikacji	sieci nakładkowe VRF wykorzystywane rozszerzenie wielo-protokołowości BGP-4 (ang. *Border Gateway Protocol 4*) (MBGP) [7, 77]	
	techniki dystrybucji etykiet MPLS [79]	
	protokoły wykorzystujące koncepcje sieci P2P	

Tabela 2.1. Technologie separacji komunikacji sieciowej

stratora w oparciu o technologię RFB (ang. *Remote Frame Buffer*) (implementacja VNC (ang. *Virtual Network Computing*)). Połączenie to jest możliwe nawet z węzłami, których komunikacja sieciowa nie została skonfigurowana, gdyż odbywa się poprzez przekierowanie graficznej konsoli i udostępnienie jej przez oprogramowanie uruchomione na hoście fizycznym.[35]

[35]Komunikacja ta może być zastąpiona przez protokół X (ang. *X Window System*) wraz z wprowadzonymi przez np. firmę NoMachine rozszerzeniami, które zmniejszają ilość koniecznych do przesyłania informacji poprawiając równocześnie interaktywność sesji.

2.2.2. Komunikacja w sieci rozległej

Obecne techniki wirtualizacji sieci dla środowisk Grid [40, 48, 65, 94] wykorzystują koncepcję sieci nakładkowych (ang. *overlay network*). W sieciach tych, zbudowanych za pomocą technik tunelowania, istnieje wydzielona przestrzeń adresowa[36], a administrator odpowiedzialny jest za konfigurację routingu oraz wykonywanie czynności związanych z dodawaniem i usuwaniem wirtualnych węzłów. Zarządzanie konfiguracją węzłów oraz konieczność ręcznego administrowania routingiem powodują, że rozwiązania te nie są skalowalne.

Istnieją również techniki umożliwiające adaptację topologii sieciowych [95], jednak wymagają one obecności centralnego serwisu koordynującego modyfikacje tras. Techniki wirtualizacji sieci zastosowane w projektach *VNET* [94] oraz *VIOLIN* [48] umożliwiają stworzenie niezawodnej infrastruktury komunikacyjnej z zastosowaniem redundantnych połączeń, kosztem wydłużenia czasu potrzebnego na dołączenie bądź odłączenie węzła z infrastruktury.

Zaproponowane w [40, 41] rozwiązania zakładają wykorzystanie koncepcji P2P jako narzędzia tworzenia wirtualnej infrastruktury komunikacyjnej. Zastosowanie P2P pozwala na zwiększenie:

- skalowalności — zarządzanie siecią P2P jest skalowalne, gdyż routing w sieciach P2P jest samokonfigurowalny, zdecentralizowany i adaptowalny do zdarzeń dołączania i odłączania węzłów,
- niezawodności — możliwa jest adaptacja ścieżek transmisji pakietów, tak aby możliwe było ominięcie węzłów, które uległy awarii.
- dostępności — zastosowanie wirtualizacji sieci pozwala na eliminację konieczności stosowania mechanizmów obchodzenia niedogodności technik NAT/Firewall, podejście to jest zdecentralizowane i nie wymaga dedykowanych serwisów.

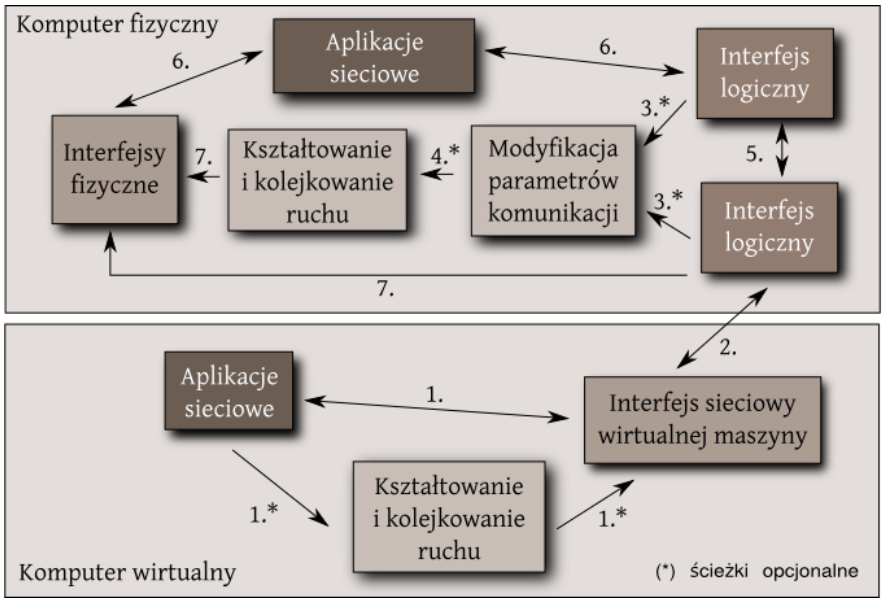
2.2.3. Zarządzanie komunikacją w wirtualnej sieci

Zarządzanie komunikacją sieciową może być sprowadzone do dwóch aspektów. Pierwszy dotyczy operacji tworzenia i dynamicznej zmiany kształtu topologii sieci wirtualnej, drugi to możliwość wpływu na właściwości samej komunikacji (jak np. gwarancje QoS). Modyfikacja wirtualnych topologii sieciowych odbywa się poprzez zmianę konfiguracji komponentów odpowiedzialnych za określenie ścieżki przekazywania komunikacji sieciowej pomiędzy maszynami wirtualnymi.

Wsparcie dla QoS dla ruchu sieciowego to możliwość określania wartości parametrów opisujących komunikację poprzez modyfikację konfiguracji zbioru wirtualnych połączeń sieciowych oraz możliwość zagwarantowania, że dla konkretnego wirtualnego połączenia parametry te będą spełnione, przez cały czas jego trwania. Sama komunikacja pomiędzy VM przekazywana jest przez wiele elementów i dlatego sposoby wpływania na jej parametry jest możliwa na wiele sposobów.

Dla komunikacji sieciowej realizowanej w obrębie sieci wirtualnej zbudowanej z połączeń wirtualnych maszyn, możemy wydzielić następujące etapy transmisji (rysunek 2.4):

[36] Np. z wykorzystaniem adresacji prywatnej [78].



Rysunek 2.4. Etapy zarządzania komunikacją sieciową wirtualnych maszyn

1. Komunikacja pomiędzy aplikacją sieciową a interfejsem sieciowym w systemie operacyjnym wirtualnej maszyny — etap ten odpowiada typowej sytuacji komunikacji oprogramowania sieciowego w systemie operacyjnym i nie różni się od analogicznej sytuacji dla fizycznego węzła. Na tym etapie możliwa jest modyfikacja konfiguracji dostępnych w systemie operacyjnym mechanizmów filtracji i kształtowania ruchu.[37]

2. Komunikacja pomiędzy interfejsem sieciowym wirtualnej maszyny a logicznym interfejsem z przestrzeni interfejsów sieciowych węzła fizycznego[38] — dla każdego interfejsu wirtualnego jest stworzony dedykowany interfejs logiczny umożliwiający obsługę ruchu sieciowego wirtualnego węzła. Oprogramowanie zarządzające nie ma dostępu do tych danych.[39]

3. Opcjonalny etap modyfikacji parametrów jakościowych komunikacji — wykorzystanie interfejsów logicznych zmieniających parametry transmisji. Mechanizmy te umożliwiają, poprzez zmianę parametrów komunikacji, symulację komunikacji zachodzącej np. w sieciach rozległych.[40] Zastosowanie tych mechanizmów umożliwia symulowanie łącza o właściwościach sieci WAN, podczas gdy np. komunikujące się przez niego maszyny wirtualne działają w obrębie jednego fizycznego węzła.

[37] Ponieważ jednak, wiązałoby się to z koniecznością posiadania odpowiedniego poziomu uprawnień dla każdej maszyny wirtualnej oraz z faktu, iż identyczne możliwości są zazwyczaj dostępne z poziomu fizycznego węzła — możliwości te nie są zwykle wykorzystywane do kształtowania ruchu w wirtualnych sieciach.

[38] Lub innym analogicznym mechanizmem umożliwiającym przekazanie ruchu pomiędzy węzłem wirtualnym a fizycznym.

[39] Dla implementacji Xen jest to komunikacja typu punkt-punkt implementowana za pomocą buforów cyklicznych, których działanie zostało przedstawione w sekcji 2.3.5.

[40] W sieciach tych występuje zwykle znacznie większy współczynnik strat, większe są opóźnienia komunikacji, częściej możliwa jest zmiana kolejności odbieranych pakietów, itp.

4. Opcjonalny etap kształtowania i kolejkowania ruchu — dodanie kolejnego zestawu interfejsów logicznych, przez które będzie przekazywany ruch sieciowy VM umożliwia implementację parametrów QoS oraz metod kształtowania ruchu.[41] Interfejsy te nie posiadają adresacji a są jedynie używane jako miejsca przyłączenia, wcześniej zdefiniowanych, definicji polityk określających parametry komunikacji.

5. Komunikacja pomiędzy interfejsami logicznymi fizycznego węzła — komunikacja ta jest obsługiwana za pomocą mechanizmów sieciowych dostępnych w systemie operacyjnym. Dzięki temu możliwe jest łączenie (wszystkich rodzajów) interfejsów w bardziej złożone topologie, poprzez budowanie (z puli interfejsów) wirtualnego przełącznika w warstwie 2, czy implementację routingu, translacji adresów NAT, filtrowania ruchu itp.

6. Komunikacja za pomocą aplikacji sieciowych uruchomionych w przestrzeni użytkownika — dzięki temu możliwe jest wysyłanie i odbieranie komunikacji z wirtualnej sieci za pomocą oprogramowania użytkowego. Przykładem różnych rodzajów oprogramowania mogą być implementacje sieci wirtualnych w technologii VPN (ang. *Virtual Private Network*), obsługa sieci nakładkowych czy obsługa różnych rodzajów tunelowania komunikacji. Najczęściej aplikacje te będą przekazywać ruch pochodzący z sieci wirtualnej dalej do odpowiednich odbiorców za pomocą sieci fizycznej. To w konfiguracji danej aplikacji będzie leżało stworzenie sieci wirtualnej nad fizyczną siecią rozległą WAN, gdyż przekazywana przez nią komunikacja będzie z punktu widzenia fizycznego węzła pochodziła z przestrzeni użytkownika. Możliwa zatem będzie komunikacja pomiędzy wirtualnymi maszynami całkowicie odseparowana od komunikacji fizycznych maszyn. Możliwe będzie również zwiększenie prywatności komunikacji przez zastosowanie szyfrowania czy kompresji zmniejszającej ilość koniecznych do przesłania danych.

2.2.4. Migracja w wirtualnej sieci

Migracja umożliwia wirtualnym węzłom przeniesienie wykonania pomiędzy komputerami zlokalizowanymi w różnych punktach sieci. Migracja ta może być wykonywana z zatrzymaniem pracy wirtualnego komputera lub bez przerywania jego działania oraz otwartych połączeń sieciowych.[42]

Zastosowanie migracji wirtualnych maszyn powoduje konieczność zapewnienia niezmienności kształtu wirtualnej topologii sieciowej przy zmianie topologii fizycznej. Przeniesienie VM pomiędzy węzłami znajdującymi się w różnych sieciach LAN powoduje konieczność dostosowania ustawień mechanizmów tunelowania/enkapsulacji, których konfiguracja odpowiada kształtowi topologii logicznej. Zwykle wymaga się aby proces ten przebiegał przeźroczyście dla komunikujących się węzłów w czasie możliwie najkrótszym.

Migracja wirtualnej maszyny powoduje zmianę tzw. adresów przyłączenia, czyli adresów[43] interfejsu fizycznego węzła, na którym wykonywana jest dana instancja VM. W procesie tym możemy wyróżnić dwa przypadki:

[41] Mechanizmy te to np. interfejs pośredniczący IMQ (ang. *Linux Intermediate Queueing Device*).
[42] Właściwości oraz działanie procesu migracji zawarte jest w sekcji 2.1.2.
[43] Należy tutaj uwzględnić zmianę adresów w warstwie drugiej jak i trzeciej modelu OSI/ISO.

1. Zmianie ulega adres warstwy łącza danych oraz część hosta (adres sieci pozostaje bez zmian) w adresie IP. Przypadek ten występuje jeśli migracja jest wykonywana pomiędzy węzłami w tej samej sieci lokalnej.

2. Modyfikacji ulegają oba adresy (adres IP jak i adres MAC (ang. *Media Access Control*)) — jest to migracja VM pomiędzy fizycznymi węzłami znajdującymi się w różnych sieciach LAN.

Zachowanie kształtu topologii wirtualnej jest dokonywane poprzez odpowiednie zmiany w konfiguracji (przedstawionych w sekcji 2.2.1) mechanizmów[44] określających początkowy kształt wirtualnej sieci. W przypadku migracji wirtualnej maszyny pomiędzy fizycznymi węzłami znajdującymi się w różnych sieciach protokołu IP [45] konieczna jest aktualizacja adresów punktu przyłączenia wirtualnego węzła (czyli styku pomiędzy maszyną wirtualną a siecią fizyczną). Konieczna jest zatem do przeprowadzenia procedura umożliwiająca aktualizację informacji o fizycznej lokalizacji maszyny wirtualnej oraz mechanizm transmisji danych do tej lokalizacji. W większości przypadków[46] stosowane są tutaj technologie dynamicznych sieci VPN, w których konfiguracja połączeń jest modyfikowana na podstawie informacji o zmianie adresu sieciowego na skutek zmiany fizycznego węzła, na którym działała dana wirtualna maszyna.

Propozycje technik budowy sieci wirtualnych z wirtualnych maszyn [40–42, 48, 54, 57, 65, 82, 94, 95] zazwyczaj nie adresują problemu zachowania kształtu topologii logicznej przy migracji wirtualnych węzłów. Po pierwsze wynika to z faktu, iż popularne techniki wirtualizacji stosowane w tych rozwiązaniach nie wspierają migracji lub jest ona nie uwzględniana. Drugi problem to występowanie znacznych narzutów zarówno na ilość koniecznych do przesłania danych (wynika to z zastosowanych technik tunelowania ruchu) jak i na czas konieczny do wznowienia komunikacji z przenoszoną maszyną.

Możliwość tworzenia dowolnych topologii logicznych dla sieci łączących wirtualne węzły jest ważną cechą. Należy jednak tę możliwość uzupełnić o wsparcie dla zapewnienia niezmienności jej kształtu bez względu na przenoszenie wykonania wirtualnych maszyn. To przenoszenie, czyli zmiana kształtu topologii fizycznej nie może wpływać na kształt topologii logicznej oraz nie powinno powodować długiego czasu nieosiągalności komunikacji wirtualnej maszyny.

2.2.5. Wirtualna sieć Ethernet

Zastosowaną w niniejszej pracy technologią budowy wirtualnych sieci dla komunikacji VM jest implementacja VDE (ang. *Virtual Distributed Ethernet*)[47]. Projekt ten umożliwia stworzenie wirtualnej sieci LAN, do której należeć mogą wirtualne maszyny uruchomione na różnych fizycznych hostach. Architektura VDE oraz komponenty służące do tworzenia sieci wirtualnych są wzorowane na technikach budowy współczesnych sieci Ethernet. W skład sieci VDE wchodzą:

[44] Tylko te mechanizmy, których konfiguracja może być modyfikowana dynamicznie mogą wspierać obsługę komunikacji sieciowej podczas migracji wirtualnych maszyn.

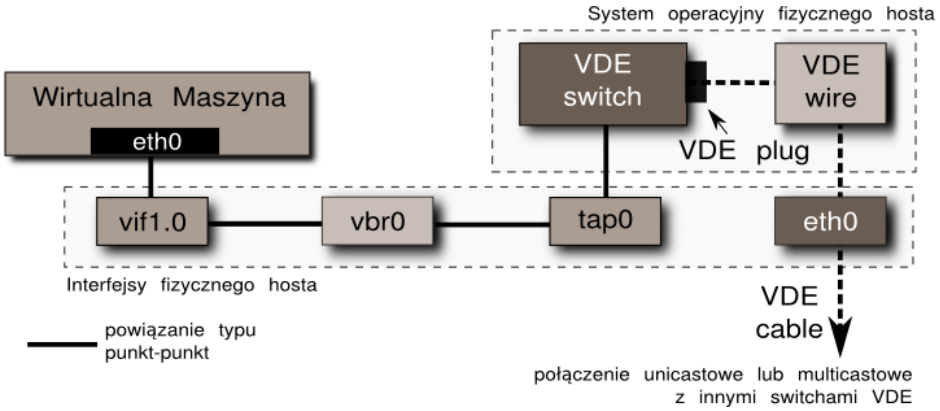
[45] Opis migracji w obrębie pojedynczej sieci LAN zawarty jest w sekcji 2.1.2.

[46] Przykłady alternatywnych technologii dystrybucji informacji o adresach dostępowych dla wirtualnego węzła przedstawia tabela 2.1.

[47] Projekt VDE: `http://vde.sourceforge.net/`.

- **Przełącznik (ang. *switch*) VDE** jest to element odpowiedzialny za przekazywanie komunikacji pomiędzy wirtualnymi maszynami, posiada porty, do których można podłączyć VM, inne przełączniki uruchomione lokalnie oraz wirtualne połączenia (ang. VDE *wire*).
- **Wtyczka (ang. *plug*) VDE** jest to program uruchomiony w przestrzeni użytkownika, na którego standardowe wejście jest przekazywany ruch z przełącznika. Dane kierowane na standardowe wyjście są przesyłane do wirtualnej sieci.
- **Połączenie (ang. *wire*) VDE** jest to dowolne oprogramowanie[48] umożliwiające komunikację strumieniową przez sieć. Używane jest do łączenia przełączników VDE uruchomionych na różnych fizycznych hostach.
- **Kabel (ang. *cable*) VDE** służy do łączenia komponentów VDE, składa się z dwóch wtyczek VDE i jednego połączenia VDE.

Schemat budowy rozproszonej wirtualnej sieci z wykorzystaniem komponentów projektu VDE przedstawia rysunek 2.5.



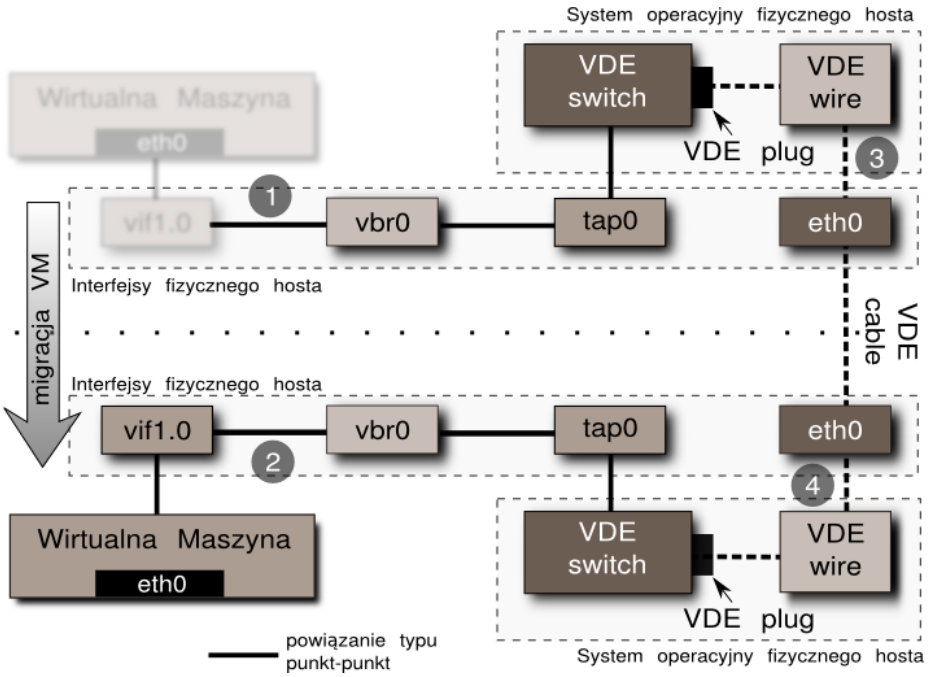
Rysunek 2.5. Schemat budowy wirtualnej rozproszonej sieci Ethernet z wykorzystaniem komponentów projektu VDE

Obsługa migracji VM w wirtualnej sieci zbudowanej z wykorzystaniem technologii VDE wymaga aktualizacji powiązania pomiędzy wirtualnym interfejsem (reprezentującym interfejs Ethernet maszyny wirtualnej w obrębie puli interfejsów fizycznego hosta) a przełącznikiem VDE. W praktyce sprowadza się to do rekonfiguracji listy interfejsów wchodzących w skład grupy zawierającej interfejs `tap` reprezentujący przełącznik VDE (czyli do usunięcia powiązania (1), dodania powiązania (2) oraz rekonfiguracji połączenia (3) i (4) — rysunek 2.6).

2.3. Implementacje wirtualizacji dla platformy Linux

Sekcja ta przedstawia opis różnych implementacji technik wirtualizacji. Ze względu na ograniczoną wielkość niniejszej rozprawy, opis ten koncentruje się jedynie na rozwiązaniach

[48] Takie jak np. `netcat`, SSH, `iptunnel` i inne.



Rysunek 2.6. Obsługa migracji VM w wirtualnej sieci zbudowanej z wykorzystaniem komponentów projektu VDE

dostępnych dla platformy Linux udostępnionych na licencji[49] umożliwiającej dostęp do kodu źródłowego.

Głównym celem tworzenia i implementacji technik wirtualizacji jest umożliwienie wykorzystania zasobów współdzielonej maszyny. Cel ten można było osiągnąć bez konieczności wprowadzania wirtualizacji, korzystając jedynie z mechanizmów udostępnianych przez system operacyjny OS. Jednak umożliwienie użytkownikom instalowania aplikacji oraz zapewnienie klasycznymi metodami izolacji czy ochrony wykonania jest złożonym procesem głównie z powodu skomplikowanej konfiguracji interakcji różnych komponentów pochodzących z niezależnych aplikacji.

Tradycyjne systemy operacyjne zwykle nie posiadają mechanizmów izolacji wydajnościowej uruchomionych wewnątrz nich zadań. Priorytety procesów, wykorzystanie pamięci, obciążenie sieci czy dostęp do danych masowych jednego z zadań zwykle negatywnie wpływa na wydajność działania pozostałych. Jedną z metod adresujących problem izolacji procesów jest dodanie możliwości kontroli wykorzystania zasobów w implementacji systemu operacyjnego.

Powstały rozwiązania uzupełniające większą kontrolę dostępu do zasobów poprzez definicję tzw. kontenerów dla zasobów (ang. *resource containers*). Głównym problemem rozwiązań takich jak Linux/RK [72], QLinux [93] czy SILK [8] jest poprawne zdefiniowanie reguł identyfikacji przynależności procesów do kontenera z uwagi na złożoność interakcji różnych komponentów aplikacji. Złożoność algorytmów systemu operacyjnego, stosowa-

[49] Takich jak licencje aprobowane przez OSI (ang. *Open Source Initiative*) jak np. GPL (ang. *GNU General Public License*), LGPL (ang. *GNU Lesser General Public License*) czy licencja BSD (ang. *Berkeley Software Distribution*).

nie buforowania, stronicowania, korzystanie z przestrzeni wymiany powoduje efekt zwany przesłuchem QoS (ang. *QoS crosstalk*) [96].

Wirtualizacja OS jest procesem multipleksacji zasobów na poziomie całego systemu operacyjnego. Umożliwia wprowadzenie izolacji wydajnościowej dla istniejących aplikacji bez konieczności ich modyfikacji. Ceną tego rozwiązania jest większy narzut jaki wprowadza, zarówno w kategoriach inicjalizacji środowiska[50] jak i w postaci narzutu powodującego degradację wydajności aplikacji.

Wirtualizacja wprowadza wysoki poziom elastyczności. Użytkownik może precyzyjnie określić konfigurację środowiska działania aplikacji, a niekorzystny wpływ pomiędzy wirtualnymi systemami praktycznie nie istnieje, zaś identyfikowanie procesów i określenie ograniczeń konsumpcji zasobów jest w wielu przypadkach trywialne. Dlatego wirtualizacja odgrywa coraz większą rolę we współczesnych systemach operacyjnych, a co za tym idzie dostępnych jest coraz więcej różnych jej implementacji.

2.3.1. UML — User Mode Linux

UML (ang. *User Mode Linux*) [26, 60] jest jedną z pierwszych szeroko stosowanych implementacji wirtualizacji dla platformy Linux. Oprogramowanie to oryginalnie było tworzone i udostępnione jako opcjonalne rozszerzenie źródeł jądra Linux od wersji 2.2.x. W wersji jądra 2.6.0, UML został zintegrowany z systemem Linux i dlatego przy tworzeniu wirtualnych maszyn UML nie ma konieczności modyfikacji systemu operacyjnego hosta. Oryginalnie, projekt tworzony był dla architektury `x86`, ale obecnie dostępne są wersje UML na procesory rodziny PowerPC oraz IA-64, oraz architekturę `x86-64`. Obecnie trwają prace nad wykorzystaniem w projekcie UML mechanizmów wspierających wirtualizację dostępnych w nowoczesnych procesorach architektury `x86`.

Działanie wirtualizacji UML opiera się na rozszerzeniu źródeł jądra Linux o nową architekturę sprzętową. Kompilacja systemu Linux dla tej architektury powoduje umieszczenie binarnej wersji jądra w kontenerze ELF (ang. *Executable and Linking Format*), czyli stworzenie pliku wykonalnego zawierającego kompletne jądro systemu operacyjnego. Uruchomienie maszyny wirtualnej sprowadza się więc do wykonania z odpowiednimi argumentami[51] wspomnianego wcześniej pliku ELF.

W przeciwieństwie do innych technik wirtualizacji takich jak np. Xen czy OpenVZ, mechanizmy UML nie wirtualizują systemu operacyjnego hosta, na którym działają instancje UML. Jest to główną zaletą tego rozwiązania gdyż dzięki temu możliwe jest uruchamianie wirtualnych maszyn w dowolnym systemie operacyjnym (dystrybucji) z rodziny Linux. Nie jest również wymagana zgodność wersji jądra pomiędzy VM a hostem, na którym dana wirtualna maszyna jest wykonywana. Dodatkowo do wykonania VM nie są konieczne uprawnienia administratora systemu.

Główne zastosowania UML to tworzenie środowisk uruchomieniowych aplikacji, czy udostępnianie usług na współdzielonej infrastrukturze. Istnieją również projekty używające rozwiązania UML do tworzenia infrastruktury bazowej dla symulacji działania systemów rozproszonych [25].

[50] Czas startu OS w porównaniu do czasu operacji `fork()`/`vfork()` dla procesów.

[51] Argumenty te są używane do specyfikacji podstawowej konfiguracji tworzonej maszyny wirtualnej i określają między innymi położenie obrazu zawierającego główny system plików, konfigurację sieci, itp.

Zasada działania UML

Początkowo działanie UML opierało się na następujących założeniach [60]:

- każdy proces działający w obrębie maszyny wirtualnej UML posiadał odpowiadający mu wpis w tablicy procesów systemu operacyjnego hosta,
- w systemie był uruchomiony specjalny wątek — *Tracing Thread*, który odpowiadał za śledzenie wywołań systemowych procesów UML,
- wywołanie systemowe powodowało przejście procesu do wykonania kodu jądra UML, które to było mapowane do górnego obszaru przestrzeni adresowej każdego procesu.

Takie podejście powodowało, iż uruchomione jądro UML było dostępne w przestrzeni adresowej wszystkich procesów działających w obrębie danej maszyny wirtualnej. Co więcej obszar ten domyślnie był w trybie do zapisu, dzięki czemu możliwe było złamanie izolacji wykonania poszczególnych maszyn wirtualnych oraz dostęp do procesów hosta, na którym dane maszyny wirtualne zostały uruchomione.

Nowy tryb działania UML powodował eliminację w/w problemów poprzez rozdzielenie przestrzeni adresowych jądra i procesów VM działających w trybie użytkownika. Tryb ten nazwany SKAS (ang. *Separate Kernel Address Space*) powodował dodatkowo około 50 % zwiększenie wydajności [60] dzięki eliminacji obsługi sygnałów używanych do śledzenia wywołań systemowych w trybie *Tracing Thread*.

Właściwości UML

Zaletą tego rozwiązania jest niewątpliwie brak koniecznych modyfikacji systemu operacyjnego hosta, duże możliwości konfiguracyjne oraz znaczna liczba metod organizacji komunikacji sieciowej wirtualnych maszyn.

Do głównych wad tego podejścia należy zaliczyć znacznie mniejszą wydajność[52] niż w przypadku rozwiązań implementujących parawirtualizację oraz możliwe zagrożenie bezpieczeństwa występujące w trybie *Tracing Thread* a związane z dostępem do zawartości pamięci innych wirtualnych maszyn. Projekt UML nie udostępnia również możliwości migracji wirtualnych maszyn pomiędzy fizycznymi węzłami.

2.3.2. OpenVZ — *Open Virtuozzo*

Implementacja OpenVZ jest techniką wirtualizacji działającą na poziomie systemu operacyjnego. Działanie OpenVZ umożliwia uruchomienie na pojedynczej maszynie wielu izolowanych instancji systemu operacyjnego (kontenerów[53]). Cechą odróżniającą to podejście od takich technik jak Xen czy VMware jest wymóg aby systemem operacyjnym hosta oraz kontenera był Linux.

OpenVZ stanowi udostępnioną na licencji GPL część komercyjnego produktu „*Parallels Virtuozzo Containers*" rozwijanego przez firmę Parallels, Inc. Produkt ten składa się ze zmodyfikowanej wersji jądra systemu Linux oraz dodatkowego oprogramowania narzędziowego do zarządzania cyklem życia kontenerów.

[52] Wydajność ta z pewnością zostanie zwiększona dzięki planowanemu wsparciu obsługi wirtualizacji bezpośrednio przez procesor.

[53] Określanych również często jako prywatny serwer wirtualny VPS (ang. *Virtual Private Server*) czy wirtualne środowisko VE (ang. *Virtual Environment*).

Zasada działania OpenVZ

Działanie implementacji OpenVZ oparte jest na obsłudze kontenerów. Każdy kontener jest wydzielonym elementem posiadającym:

- system plików, biblioteki, aplikacje, specjalne systemy plików jak `/proc` czy `/sys`,
- konfiguracje uprawnień, użytkowników i grupy, konto administratora (root),
- przestrzeń identyfikatorów procesów (dostęp tylko do procesów uruchomionych w obrębie danego kontenera),
- wirtualne interfejsy sieciowe, konfiguracje routingu, filtracji i kształtowania ruchu,
- urządzenia fizyczne (porty, interfejsy sieciowe, karty rozszerzeń itp.) przydzielone do danego kontenera z puli urządzeń fizycznego hosta,
- obiekty komunikacji IPC.

Zarządzanie konfiguracją dostępu do zasobów przez poszczególne kontenery może być realizowane przez modyfikacje trzech komponentów: limitów alokacji przestrzeni dyskowej, regulację przydziału czasu procesora oraz pasma komunikacji I/O a także zestawu kilkunastu parametrów i ograniczeń ustawianych niezależnie dla każdego kontenera. Konfiguracje te mogą być modyfikowane w czasie działania danego kontenera.

Dostęp do dysku jest zarządzany podobnie jak przy standardowym mechanizmie *Quota*, możliwe jest określenie maksymalnej liczby bloków i plików, które może używać dany kontener. Zarządzanie dostępem do procesora odbywa się poprzez przydział slotu czasu na podstawie ustalonego dla kontenera priorytetu (proporcjonalnie do jego wartości). Podział pomiędzy procesy wewnątrz kontenera jest obsługiwany przez standardowe mechanizmy systemu operacyjnego. Możliwe są również ograniczenia „twarde" takie jak np. nie więcej niż 15 % czasu procesora. Podobnie odbywa się również zarządzanie dostępem do pasma komunikacyjnego I/O.

Pozostałe cechy kontenerów związane z konsumpcją zasobów są określane przez zestaw około dwudziestu parametrów. Parametry te powalają kontrolować dostęp do pamięci, użycie buforów, alokację struktur jądra itp. Mechanizm ten pozwala też na śledzenie zużycia zasobów oraz ustalanie poziomu i monitorowanie nadużyć.

Właściwości OpenVZ

OpenVZ pozwala na obsługę do 64 procesorów i do 64 GB pamięci RAM (ang. *Random Access Memory*). Rozwiązanie to cechuje się niskimi narzutami wynoszącymi około $1-3$ % w stosunku do wykonania natywnego. OpenVZ nie ogranicza ilości równocześnie uruchomionych kontenerów, dzięki temu możliwe jest równoczesne wykonanie setek kontenerów[54] używając współczesnego sprzętu komputerowego.

OpenVZ umożliwia także przeprowadzenie migracji uruchomionego kontenera. Proces ten wymaga zamrożenia wykonania i zapisanie stanu kontenera na dysk. Plik ten po przesłaniu na inną maszynę pozwala na wznowienie działania kontenera. Występująca przerwa

[54] Ograniczenie występuje na poziomie dostępnej ilości pamięci operacyjnej.

w działaniu kontenera wynika z czasu koniecznego na skopiowanie zawartości pliku przechowującego jego stan. Przerwa ta jest zależna od rozmiaru kopiowanych danych i wynosi typowo kilka sekund.

Technologia ta jest często używaną implementacją konsolidacji serwerów, ponieważ możliwy jest łatwy dostęp do przestrzeni dyskowej i procesów poszczególnych kontenerów. Dzięki temu można sprawnie przeprowadzać operacje np. aktualizacji czy instalacji oprogramowania, gdyż szybkie i efektywne staje się wykonywanie tych samych operacji na wielu kontenerach równocześnie[55].

2.3.3. Projekt Linux-VServer

Projekt Linux-VServer jest kolejną implementacją wirtualizacji dla platformy Linux działającej na poziomie systemu operacyjnego. Zasoby komputera fizycznego jak pamięć, czas procesora, interfejsy sieciowe, są dzielone poprzez tworzenie tak zwanych partycji. Mechanizm ten jest podobny do znanego z systemów BSD, operującego na poziomie systemu plików, rozwiązania BSD-Jail.

Oprogramowanie Linux-VServer jest tworzone jako rozszerzenie funkcjonalności jądra systemu Linux i dystrybuowane jest w oparciu o licencję GPL. Są dostępne dwie wersje; stabilna (2.2.x) i rozwojowa (2.3.x) obie dla źródeł jądra w wersji 2.6. Dodatkowo, dostępna jest również wersja Linux-VServer zawierająca integrację rozszerzeń bezpieczeństwa jądra Linux tworzonych w ramach projektu GRSecurity[56].

Proces uruchamiania instancji systemu operacyjnego VPS wymaga stworzenia nowej partycji, zmiany kontekstu wykonania (za pomocą odpowiedniego narzędzia) a następnie wykonania procesu odpowiedzialnego za inicjalizację startu systemu operacyjnego (typowo jest to proces `init`). Zamknięcie systemu to zabicie wszystkich procesów działających w ramach danego kontekstu (partycji). Oprogramowanie to umożliwia dodatkowo tworzenie systemu plików dla partycji poprzez zestaw twardych dowiązań[57] działających w trybie CoW. Powoduje to znaczne oszczędności w ilości miejsca zajmowanego na dysku przez wiele identycznych partycji.

Właściwości Linux-VServer

Główne zalety implementacji wirtualizacji Linux-VServer:

- brak narzutów związanych z wirtualizacją wywołań systemowych, gdyż wszystkie partycje współdzielą ten sam system operacyjny,
- brak konieczności tworzenia obrazów systemów plików dla poszczególnych partycji, gdyż partycje mogą współdzielić wspólną przestrzeń w obszarze systemu plików hosta,
- możliwość współdzielenia mechanizmów cache dla systemu plików pomiędzy partycjami a system operacyjnym hosta,

[55] W systemach Xen czy VMware wymagane jest w tym przypadku niezależne logowanie się na każdą maszynę wirtualną i powtarzanie tych samych operacji wielokrotnie.

[56] `http://www.grsecurity.net`.

[57] Dowiązania te są specjalnie oznaczane z wykorzystaniem mechanizmu atrybutów w systemie plików i gdy następuje jego modyfikacja dowiązanie to jest zamieniane na kopię oryginalnego pliku.

- obsługa sieci bazuje na izolacji, brak obciążenia dodatkowymi narzutami zawiązanymi z enkapsulacją komunikacji,
- procesy uruchomione w obrębie partycji są widziane jako zwykłe procesy działające w systemie hosta, dzięki temu możliwe jest pełniejsze wykorzystanie zasobów (szczególnie w systemach SMP (ang. *Symmetric Multi-Processing*) gdzie możliwe jest równoczesne wykonanie procesów z wielu partycji).

Ujemne cechy rozwiązania:

- podobnie jak w systemie OpenVZ konieczne są modyfikacje systemu operacyjnego hosta,
- brak wsparcia dla migracji i możliwości zapisania i odtworzenia stanu partycji, system operacyjny hosta jest pojedynczym punktem awarii dla wszystkich uruchomionych partycji,
- brak możliwości zarządzania komunikacją sieciową (routing, filtracja ruchu) na poziomie partycji,
- wywołania systemowe związane ze sprzętem nie są wirtualizowane, nie są wirtualizowane również specjalne systemy plików: `/proc` czy `/sys`,
- brak możliwości alokacji pasma I/O dla poszczególnych partycji.

2.3.4. KVM — *Kernel Based Virtualization Driver*

KVM jest dostępną w jądrze[58] systemu Linux implementacją pełnej wirtualizacji dla platform opartych na architekturze x86. Implementacja ta działa z wykorzystaniem dostępnych w procesorach rozszerzeń wspierających wirtualizację.[59]

Implementacja Linux KVM składa się z zestawu modułów ładowanych do jądra a zawierających:

`kvm.ko` — główną implementację infrastruktury wirtualizacji — obsługa wirtualizacji niezależna od rodzaju procesora,

`kvm-intel.ko` **lub** `kvm-amd.ko` — dodatkowe implementacje obsługi wirtualizacji specyficzne dla architektury procesora.

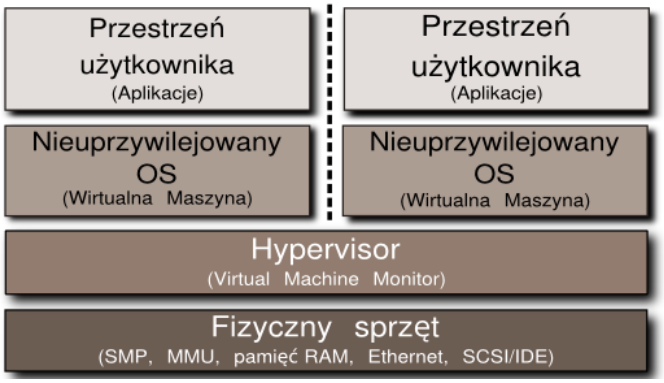
KVM do działania potrzebuje również zmodyfikowanej wersji oprogramowania Qemu[76]. Użycie KVM pozwala na uruchamianie niezmodyfikowanych wersji systemów operacyjnych z rodziny Linux oraz Windows.

[58] Począwszy od wersji systemu 2.6.20, implementacja KVM została dołączona do głównej gałęzi jądra. Dla pozostałych (wcześniejszych) wersji jest dostępna w postaci rozszerzenia.

[59] Rozszerzenia te dla procesorów firmy Intel — IVT (ang. *Intel Virtualization Technology*) oraz dla procesorów firmy AMD (ang. *Advanced Micro Devices*) — SVM (ang. *Secure Virtual Machine*) lub AMD-V (ang. *AMD-Virtualization*).

Zasada działania KVM

Wykorzystanie wsparcia dla wirtualizacji wbudowanego w nowe procesory[60] rodziny x86 pozwoliło znacznie uprościć implementację pełnej wirtualizacji dla tej platformy. Zastosowany schemat wirtualizacji używany również w innych implementacjach zakłada wykorzystanie tzw. modelu z nadzorcą (ang. *hypervisor*).



Rysunek 2.7. Architektura wirtualizacji z nadzorcą

Nadzorca *hypervisor* w tym podejściu jest odpowiedzialny za udostępnienie i multipleksację dostępu do sprzętu dla wielu równocześnie uruchomionych wirtualnych maszyn. Dodatkowo wymagane jest aby był odpowiedzialny za przydział i zarządzanie pamięcią oraz kolejkowanie procesów. W alternatywnych rozwiązaniach operacje te są delegowane do systemu uruchomionego w trybie uprzywilejowanym (rysunek 2.7).

Użycie, jako system uprzywilejowany, jądra systemu Linux daje dostęp do kompletnej i efektywnej implementacji obsługi zarządzania pamięcią i kolejkowania procesów dla szerokiej ilości platform i architektur.[61] W tym podejściu każdy wirtualny komputer jest widziany z poziomu systemu operacyjnego fizycznego węzła jako pojedynczy proces i może być zarządzany przez podsystem kolejkowania procesów systemu Linux.[62] Integracja oprogramowania KVM bezpośrednio z jądrem systemu Linux pozwala na automatyczne dostosowanie systemu wirtualizacji do zmian wprowadzonych do źródeł jądra.[63]

Standardowy proces systemu Linux może być wykonywany w dwóch trybach: tryb jądra i tryb użytkownika. Użycie rozwiązania KVM dodaje trzeci tryb: tryb nieuprzywilejowany, który posiada własne uruchomione jądro oraz załadowane moduły. Podział pracy pomiędzy różne tryby (rysunek 2.8) wygląda następująco [47]:

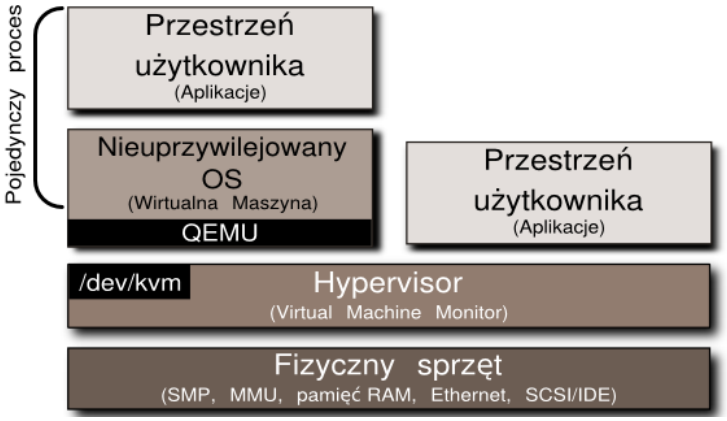
Tryb nieuprzywilejowany — wykonanie kodu niezwiązanego z operacjami I/O.

[60] Rozszerzenie Intel-VT zostało wprowadzone w pierwszym kwartale 2005 roku i dostępne jest w procesorach linii Pentium 4 (modele 6x1 i 6x2), Pentium D 9x0, Xeon 3xxx/5xxx/7xxx, Core Duo oraz Core 2 Duo, natomiast rozszerzenie SVM — procesory linii K8 (Athlon 64) oraz we wszystkich nowych procesorach wspierających AMD-V.

[61] Włączając w to np. systemy wieloprocesorowe czy złożone architektury NUMA.

[62] Procesy te mogą być zarządzane z użyciem standardowych narzędzi obsługi procesów w systemach rodziny Linux.

[63] Szczególnie istotne jest dostosowanie KVM do wykorzystania niedawno wymienionego podsystemu kolejkowania [68].



Rysunek 2.8. Architektura systemu KVM

Tryb jądra — przełączanie do trybu nieuprzywilejowanego oraz obsługa powrotów z trybu nieuprzywilejowanego związanych z obsługą I/O lub instrukcji specjalnych.

Tryb użytkownika — wykonanie operacji I/O.

Zarządzanie KVM

Ponieważ uruchomiona wirtualna maszyna (VM) jest widziana z poziomu jądra jako pojedynczy proces, wszystkie standardowe narzędzia systemu Linux do obsługi procesów mogą być stosowane do zarządzania maszynami KVM. Usunięcie, wstrzymanie czy wznowienie wykonania VM może być dokonane za pomocą standardowego polecenia `kill`. Prawa dostępu i bezpieczeństwo jest obsługiwane przez standardowe mechanizmy systemu operacyjnego.[64]

Pozostałe czynności administracyjne takie jak obsługa I/O (tworzenie i obsługa obrazów dysków, użycie mechanizmu CoW, migracja) są wykonywane za pomocą narzędzi z pakietu Qemu [76].

Implementacja KVM nie zawiera oprogramowania umożliwiającego zdalne zarządzanie wirtualizacją, aczkolwiek jest ono możliwe poprzez wykorzystanie narzędzi[65] (niezależnych od KVM) realizujących zdalne wywołanie poleceń systemu operacyjnego.

2.3.5. Xen — Virtual Machine Monitor

W klasycznych implementacjach pełnej wirtualizacji, udostępniany przez VMM (ang. *Virtual Machine Monitor*) wirtualny sprzęt jest identyczny z fizycznym, umożliwiając uruchamianie niezmodyfikowanych systemów operacyjnych. Jednak dla architektur typu x86 gdzie wsparcie dla wirtualizacji nigdy nie było planowane, takie podejście ma szereg wad[66].

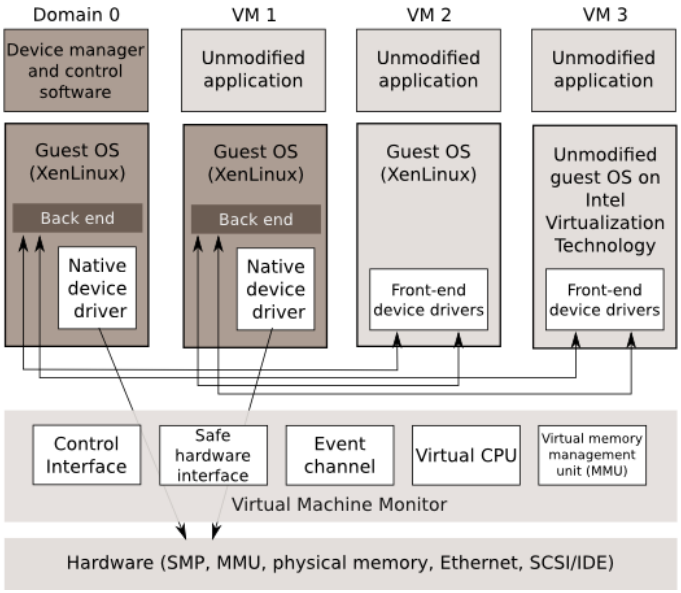
[64] Maszyna wirtualna działa z prawami użytkownika, który ją uruchomił — nie musi to być administrator (`root`).

[65] Np. za pomocą standardowego oprogramowania obsługi protokołu SSH.

[66] Największe problemy z wirtualizacją dla platformy x86 sprawia wirtualizacja dostępu do pamięci oraz konieczność dynamicznego instrumentowania kodu, aby możliwe było przechwytywanie przez VMM instrukcji uprzywilejowanych CPU.

Problem braku wsparcia dla wirtualizacji był rozwiązywany poprzez programową realizację brakujących mechanizmów, jednak powodowało to znaczny wzrost złożoności implementacji, który z kolei przyczyniał się do zwiększenia narzutu na wykonanie aplikacji.

Projekt Xen — *High performance resource-managed virtual machine monitor* [5, 6, 18, 20] powstał na uniwersytecie w Cambridge i jest obecnie najbardziej dynamicznie rozwijaną platformą VM stworzoną dla systemu Linux. Zaproponowane w implementacji Xen podejście do wirtualizacji polega na stworzeniu wyidealizowanej architektury (podobnej, ale nie identycznej z architekturą fizycznego węzła), na którą systemy Linux, BSD czy Windows muszą zostać przeniesione. Implementacja ta pozwala na współdzielenie konwencjonalnego sprzętu, pomiędzy wieloma systemami operacyjnymi, w bezpieczny (izolowany) sposób umożliwiający zarządzanie zasobami bez znacznego obniżenia wydajności.[67]



Rysunek 2.9. Architektura systemu Xen [6]

Zasada działania parawirtualizacji Xen

Implementacja parawirtualizacji udostępnia specjalny interfejs dla systemu operacyjnego. Interfejs ten jest wykorzystywany przez system operacyjny do komunikacji z VMM. Implementacja Xen uwzględnia trzy główne obszary zarządzania zasobami: obsługa pamięci, procesor oraz urządzenia I/O (tabela 2.2).

Zarządzanie pamięcią w systemie Xen

Zarządzanie pamięcią jest najbardziej skomplikowaną częścią architektury Xen, zarówno pod względem mechanizmów implementowanych przez *hypervisor* jak i nakładu pracy związanego z przeniesieniem systemu operacyjnego. Złożoność ta wynika głównie z faktu,

[67] Mierzony spadek wydajności rzędu kilku procent.

Zarządzanie pamięcią	Segmentacja Stronicowanie
Procesor (CPU)	Ochrona Wyjątki Wywołania systemowe Przerwania Czas
Urządzenia I/O	Interfejsy sieciowe Kontrolery dysków Grafika

Tabela 2.2. Interfejs parawirtualizacji dla architektury x86

iż architektura x86 nie posiada mechanizmów służących do programowego zarządzania organizacją dostępu do pamięci.

W przeciwieństwie do architektur typu RISC (ang. *Reduced Instruction Set Computer*) gdzie w większości przypadków możliwa jest programowa obsługa zarządzania stronami pamięci TLB (ang. *Translation Lookaside Buffer*)[68] architektura x86 nie posiada obsługiwanej programowo tablicy TLB. Zamiast tego, odwołania dla których nie istnieje wpis w TLB są automatycznie obsługiwane przez procesor. Efektywne działanie systemu zarządzania pamięcią dla architektury x86 wymaga aby translacje adresów stron były obsługiwane za pomocą sprzętowego TLB, jednak przy przełączeniu kontekstu dla adresacji wykonywane jest całkowite opróżnienie zawartości TLB.

Te właściwości architektury x86 spowodowały, iż twórcy systemu Xen przyjęli, że za zarządzanie pamięcią odpowiedzialny będzie system operacyjny VM z jedynie minimalną interwencją VMM konieczną do zapewnienia bezpieczeństwa i izolacji przestrzeni adresowych VM. Dodatkowo aby nie obniżać wydajności, konieczne było również umieszczenie kodu VMM w pierwszych 64 MB[69] każdej przestrzeni adresowej VM. Dzięki tej replikacji kodu unika się konieczności opróżniania zawartości TLB przy przełączaniu kontekstu pomiędzy wirtualną maszyną a VMM.

W sytuacji gdy VM potrzebuje nowej strony pamięci, rezerwuje i alokuje stronę z własnej przestrzeni adresowej a następnie rejestruje ją w Xen. Wszystkie kolejne aktualizacje tablicy stron muszą być weryfikowane przez *hypervisor* co zapewnia izolację VM, a równocześnie powoduje, iż zarządzanie pamięcią następuje w sposób efektywny. W podobny sposób weryfikowana jest przez Xen obsługa sprzętowej tablicy deskryptorów SDC (ang. *Segment-Descriptor Cache*).

Wirtualizacja procesora

Wirtualizacja procesora wymaga zmiany pewnych założeń, gdyż system operacyjny nie jest już najbardziej uprzywilejowanym elementem programowym całego systemu. Aby za-

[68] TLB jest pewnym rodzajem pamięci adresowanej kontekstowo CAM (ang. *Content-Addressable Memory*) używanej przez kontroler zarządzania pamięcią do przyśpieszenia translacji adresów wirtualnych na sprzętowe przy odwołaniach do pamięci. TLB jest zlokalizowana w procesorze. Jeśli wpis nie znajduje się w TLB — przeszukiwana jest (znacznie wolniejsza) pamięć stron w celu odnalezienia mapowania.

[69] Obszar ten zwykle nie jest wykorzystywany i dlatego zajęcie go przez *hypervisor* nie łamie ABI (ang. *Application Binary Interface*) istniejących aplikacji.

pewnić ochronę VMM przed niepożądanymi działaniami systemu operacyjnego oraz chronić wykonanie wirtualnych maszyn pomiędzy sobą — system operacyjny VM musi być zmodyfikowany tak aby mógł działać na niższym poziomie uprawnień. Efektywna wirtualizacja w architekturze x86 jest możliwa ponieważ wspiera ona sprzętowo cztery różne poziomy uprawnień. Podobnie jak w przypadku implementacji KVM nastąpiło przeniesienie systemu operacyjnego z poziomu 0 (najbardziej uprzywilejowanego) i zmodyfikowanie go, tak aby możliwe było jego działanie na poziomie 1. Poziom 0 został wykorzystany przez *hypervisor*, a poziom 3 bez zmian jest wykorzystywany przez aplikacje. Instrukcje uprzywilejowane[70] są wirtualizowane (parawirtualizowane) poprzez wymóg weryfikacji[71] i wykonania ich przez *hypervisor* systemu Xen.

Wyjątki są obsługiwane poprzez rejestrację tablicy uchwytów dla każdego z typów wyjątku. Uchwyty te kierują obsługę do odpowiedniej procedury walidacji Xen. Same procedury obsługi są prawie identyczne z tymi stworzonymi dla oryginalnej architektury x86. Wywołania systemowe oraz zdarzenia notyfikujące o braku strony pamięci (ang. *page fault*) są najczęstszymi wyjątkami w systemie i dlatego ich złożona obsługa mogłaby powodować degradację wydajności działania VM. Dlatego w implementacji Xen, system operacyjny VM rejestruje tzw. szybką wersję procedury obsługi wyjątku, weryfikowaną przez Xen jedynie przy instalacji jej w sprzętowej tablicy wyjątków. Wykonanie tej procedury nie wymaga przejścia procesora przez najwyższy poziom uprawnień — poziom 0, co skraca czas obsługi wyjątku.

Urządzenia wejścia/wyjścia

W przeciwieństwie do rozwiązań pełnej wirtualizacji gdzie implementowany jest kompletny zestawu procedur służących do emulacji fizycznego sprzętu, Xen udostępnia jedynie minimalną abstrakcję urządzeń. Obecność *hypervisor*'a zlokalizowanego bezpośrednio nad sprzętem powoduje, iż pomiędzy systemem VM a urządzeniami znajduje się dodatkowy element odpowiedzialny za multipleksację i ochronę przesyłanych danych. Przesyłanie informacji pomiędzy urządzeniami I/O a domeną musi odbywać się w sposób wydajny a jednocześnie umożliwiać elastyczną konfigurację zarządzania zasobami oraz zgłaszanie i obsługę zdarzeń.

Implementacja sterowników do urządzeń znajduje się jedynie w systemie operacyjnym uruchomionym w trybie uprzywilejowanym. Komunikacja pomiędzy systemem VM a urządzeniem odbywa się za pomocą zestawu buforów cyklicznych — *ring*. Ring jest kołową kolejką deskryptorów alokowanych przez domeny i dostępną z poziomu Xen. Deskryptory te nie przechowują danych a jedynie odwołania do obszarów pamięci.[72] Implementacja Xen wykorzystuje do przesyłania danych I/O pamięć współdzieloną dostępną zarówno z poziomu VM jak i *hypervisor*'a. Dostęp do *ring*u[73] odbywa się za pomocą dwóch par wskaźników (oznaczających początek i koniec ważnych danych dla producenta i konsumenta). Jądro systemu VM chcąc przeprowadzić komunikację z urządzeniem umieszcza w tym bu-

[70] Przykładowo instalowanie nowej tablicy stron.

[71] Każda próba wykonania uprzywilejowanej instrukcji przez VM jest automatycznie blokowana przez procesor.

[72] Obszary te są przydzielone z przestrzeni pojedynczego VM do komunikacji z dostępnymi w jego konfiguracji urządzeniami I/O. Zawartość tych obszarów nie jest dostępna dla innych VM.

[73] Liczba pierścieni cyklicznych zależy od liczby używanych aktualnie urządzeń I/O przez wszystkie maszyny wirtualne.

forze odpowiednie żądanie, za którego obsłużenie odpowiedzialny jest Xen. W technice tej nie ma jednak wymogu obsługi żądań w określonym porządku, gdyż VM przydziela każdemu żądaniu unikalny identyfikator używany przez Xen do oznaczenia odpowiedzi. VM może również umieścić w pierścieniu kilka żądań i dopiero po zebraniu ich odpowiedniej ilości wysłać notyfikację obsługi do VMM. Obsługa ilości żądań umieszczonych w pierścieniu w jednym cyklu może być również ograniczona, dzięki temu możliwe jest określenie parametrów jakościowych obsługi urządzeń I/O.

Struktura pierścieni cyklicznych do komunikacji I/O jest wystarczająco prosta aby móc obsłużyć większość sposobów komunikacji z urządzeniami, dodatkowo sposób ten jest wydajny, zapewnia konieczną izolację oraz ochronę procedur obsługi urządzeń.

2.3.6. Porównanie technik wirtualizacji

Porównanie cech użytkowych implementacji wirtualizacji dostępnych dla platformy Linux zostało przedstawione w tabeli 2.3.

Przedstawienie niniejszych cech oraz praktyczna weryfikacja udostępnianych funkcjonalności pozwoliło na wybór techniki wirtualizacji zastosowanej w niniejszej pracy. Według autora najważniejszymi cechami, które przesądziły o wyborze to przede wszystkim bogate możliwości konfiguracji[74] i regulacji poziomu konsumpcji zasobów przez poszczególne VM. Równie ważna była możliwość pełnego zarządzania środowiskiem wirtualizacji poza lokalną konsolą administracyjną w trybie zdalnym.

Ze względu na ograniczenie rozmiaru niniejszej pracy z dostępnych rozwiązań dla platformy Linux zostały przedstawione tylko rozwiązania najbardziej popularne, najczęściej wykorzystywane w praktyce oraz posiadające cechy istotne dla praktycznej realizacji opisywanego w dalszej części pracy systemu.

2.4. Podsumowanie

Zastosowanie wirtualizacji w systemach Grid oferuje szereg korzyści, które m.in. umożliwią osiągnięcie funkcjonalności, których wcześniejsze próby implementacji okazały się skomplikowane i nieefektywne.

Podsumowując, wykorzystanie wirtualizacji umożliwi uzyskanie następujących cech i własności:

- możliwość tworzenia usług i serwisów na żądanie, wraz z wcześniejszym określeniem ich charakterystyk,
- możliwość przydziału i zarządzania zasobami za pomocą określonych wcześniej polityk,
- wprowadzenie efektywnego mechanizmu zarządzania zasobami Grid,
- uproszczenie stopnia skomplikowania sieci, poprzez obniżenie poziomu abstrakcji oraz umożliwienie komunikacji za pomocą protokołów, których wsparcie w istniejących instalacjach jest znikome,

[74] Takie jak obsługa wielu metod organizacji komunikacji sieciowej, zarządzania przestrzenią dyskową, obsługa migracji itp.

Projekt	Twórcy	Platforma	System VM	System hosta	Licencja
Xen	University of Cambridge, Intel, AMD	x86, AMD64[a]	NetBSD, Linux, Solaris	Linux, Solaris, Windows XP & 2003 Server[b], Plan 9	GPL
User Mode Linux	Autor: Jeff Dike (i inni)	x86, x86-64, PowerPC	Linux	Linux	GPL wersja 2
Sun xVM	Sun Microsystems	x86-64, SPARC	brak[c]	Windows XP & 2003 Server[d], Linux, Solaris	GPL wersja 3
OpenVZ	Projekt społecznościowy, wpierany przez SWsoft	Intel x86, AMD64, IA-64, PowerPC64, SPARC/64	Linux	Linux	GPL
KVM	Qumranet [47]	Procesory Intel/AMD[e]	Linux	Linux, Windows	GPL wersja 2
QEMU	Autor: Fabrice Bellard (i inni)	x86, AMD64, IA-64, PowerPC, Alpha, SPARC 32 i 64, ARM, S/390, M68k	Windows, Linux, Mac OS X, Solaris, FreeBSD, OpenBSD, BeOS	Linux	GPL/LGPL
Linux-VServer	Projekt społecznościowy	x86, AMD64, IA-64, Alpha, PowerPC/64, PA-RISC/64, SPARC/64, ARM, S/390, SH/66, MIPS	Linux	Linux	GPL wersja 2

Tabela 2.3. Zestawienie właściwości technik wirtualizacji dostępnych dla platformy Linux

[a] Trwają prace nad architekturami PowerPC i IA-64
[b] Wymagana jest wersja 3.0 oraz procesor ze wsparciem dla wirtualizacji.
[c] Solaris udostępniający sterowniki dla systemów nieuprzywilejowanych.
[d] Tylko wersja na dla architektury x86-64.
[e] Wymagane wsparcie dla wirtualizacji w CPU.

- wsparcie dla mechanizmów QoS przy przetwarzaniu zadań w systemach Grid,
- wprowadzenie izolacji środowiska uruchomieniowego dla zadań i aplikacji,
- bardziej efektywne wykorzystanie współdzielonych zasobów, dzięki zastosowaniu statystycznej multipleksacji różnych wymagań aplikacji uruchomionych w obrębie Gridu.

Wprowadzenie wirtualizacji może dodać wiele pożądanych właściwości do istniejących i przyszłych instalacji Grid.

Rozdział 3

Przedstawienie problemu i sformułowanie wymagań

Obecnie, koncepcja przetwarzania rozproszonego bazująca na modelu Grid, została szeroko przyjęta przez środowiska naukowo-badawcze oraz organizacje komercyjne. Kluczowym zadaniem jest więc tworzenie i obsługa systemów zarządzania zasobami Grid, które to umożliwiają maksymalizację efektywności działania oraz wykorzystania infrastruktury dla szerokiej grupy aplikacji.

Zaproponowana w rozdziale 1 definicja klasycznego Gridu, przedstawia Grid jako dynamiczne środowisko, bazujące na współdzielonych zasobach i wykorzystujące na masową skalę rozproszone komponenty. To klasyczne podejście wnosi jednak szereg ograniczeń w działaniu aplikacji i możliwościach środowiska. Aplikacja nie jest zdolna do wyszukania, wyboru i pozyskania zasobów, których wykorzystanie, z punktu widzenia efektywności jej działania byłoby optymalne. Lista ewentualnych konfiguracji byłaby zbyt duża dla efektywnej selekcji. Dodatkowo zasoby Gridu są rozproszone, współdzielone pomiędzy różnymi organizacjami oraz posiadają heterogeniczne właściwości i konfiguracje. Także ich aktualna dostępność nie może być zagwarantowana. Wszystkie te czynniki powodują trudności lub całkowitą niemożność uzyskania odpowiedniego zestawu konfiguracji zasobów.

Większość osób zaangażowanych w badania nad infrastrukturą Grid jest przekonanych, że obecne koncepcje tych środowisk są jeszcze w początkowym stadium rozwoju. Przewidywany jest wzrost zarówno wykorzystania środowisk Grid jak i ich rozbudowa, tak iż w przyszłości będą składać się z milionów niezależnych urządzeń [43]. Rozwój ten spowoduje, że obecnie stosowane klasyczne mechanizmy zarządzania zasobami [53], osiągną swoje możliwości graniczne.

Stanowi to uzasadnienie do przyjętego w niniejszej pracy podejścia do zarządzania zasobami środowisk Grid, uwzględniającego wykorzystanie koncepcji Wirtualnego Gridu, tworzonego na żądanie na podstawie specyfikacji określonej wraz z konfiguracją aplikacji. Koncepcja ta pozwoli na stworzenie efektywnego, skalowalnego mechanizmu zarządzania zasobami w środowiskach rozproszonych.

3.1. Charakterystyka istniejących systemów Grid

Dokonując analizy architektur systemów zarządzania używanych w obecnych środowiskach Grid [11, 61, 97] oraz w proponowanych rozwiązaniach dla przyszłych zastosowań tej idei przetwarzania [28, 84, 85], możemy wyróżnić następujące [83] kolekcje zasobów:

1. Zasoby obliczeniowe — fizyczne węzły udostępniające takie zasoby jak: CPU, pamięć

operacyjna i inne elementy sprzętowe komputera.

2. Zasoby dyskowe — przestrzeń dyskowa oraz elementy infrastruktury komputerowej i komunikacyjnej związane z przechowywaniem i udostępnianiem danych.

3. Sieć komputerowa — infrastruktura sieciowa, włączając w to interfejsy sieciowe węzłów obliczeniowych oraz urządzenia komunikacyjne używane przez węzły.

4. Usługi — obsługa infrastruktury, raportowanie, wirtualne organizacje.

5. Oprogramowanie — komponenty (wraz z zależnościami), rozszerzenia, obsługa wersji oraz licencji.

6. Dane — dostępność danych dla wykonywania obliczeń, ich lokalizacja, oraz takie atrybuty jak spójność czy poufność.

Odpowiednie wykorzystanie zasobów obliczeniowych i komunikacyjnych oraz danych z wyżej wymienionych klas, jest w obecnie stosowanych środowiskach postrzegane jako główny cel funkcjonowania systemów zarządzania RMS. Chcąc jednak zapewnić odpowiednie zarządzanie danymi, konieczne jest również odpowiednie wsparcie dla obsługi infrastruktury, ich przechowywania i udostępniania. Także ze względu na rozproszony charakter węzłów obliczeniowych i przetwarzanych przez nie danych, obsługa zasobów sieciowych jest ważna zarówno dla zapewnienia dostępu do informacji jak i wymogów komunikacyjnych [106] dla różnych komponentów programowych tworzących zadanie obliczeniowe. Zadania interaktywne, np. wykonujące ciągłą wizualizację przetwarzanych operacji mogą wymagać określenia parametrów definiujących komunikację sieciową (zapotrzebowanie na pasmo komunikacyjne, maksymalne opóźnienie itp.).

Analiza rodzajów zasobów wskazuje, że niektóre z nich będą mniej skomplikowane w zarządzaniu, podczas gdy zarządzanie pozostałymi będzie wymagało większej złożoności procedur obsługi. Złożoność ta wynika z konieczności koordynacji równocześnie wykonywanych operacji dla różnych typów zasobów.

3.2. Główne założenia pracy

Jednym z najważniejszych zadań systemów zarządzania w środowiskach rozproszonych jest umożliwienie skoordynowanego i efektywnego zarządzania zasobami obliczeniowymi i komunikacyjnymi poprzez zastosowanie technik wirtualizacji. Działania takie pozwalają na pełniejsze wykorzystanie dostępnej infrastruktury oraz na zmniejszenie kosztów, poprzez redukcję wykorzystania zasobów bez zmniejszenia poziomu QoS dla usługi. Efektywne zarządzanie zasobami na żądanie z wykorzystaniem wirtualizacji jest możliwe poprzez użycie takich funkcji jak dynamiczne instancjonowanie elementów infrastruktury i alokacja zasobów w zależności od fazy działania aplikacji bądź zmieniających się warunków.[1] Do realizacji tych celów musi istnieć zarządzalne i elastyczne środowisko uruchomieniowe, które będzie dostosowane do wymogów danej aplikacji, pozwalając na osiągnięcie odpowiedniego poziomu jakości wykonania, gwarantując odpowiedni poziom bezpieczeństwa oraz izolację od innych równocześnie wykonywanych zadań.

[1] Większość instalacji boryka się z problemem przydziału współdzielonych zasobów tak aby był możliwy zagwarantowany poziom wydajności aplikacji (QoS dla aplikacji).

Zastosowanie wirtualizacji w tej koncepcji odgrywa kluczową rolę. Pula fizycznych węzłów może być zastąpiona za pomocą wirtualnych maszyn pełniących rolę kontenerów uruchomieniowych aplikacji. Kontenery te mogą być zarządzane (wykonywanie operacji tworzenia, usuwania i modyfikacji konfiguracji przydziału zasobów) za pomocą zdalnego interfejsu. Dla instalacji, w których możliwe jest uruchamianie wielu aplikacji równocześnie, zastosowanie grupowania zasobów jest istotne. Pozwala to na określenie minimalnego (gwarantowanego) poziomu jakości wykonania. Co więcej, zastosowanie technik izolacji, pozwala na wyeliminowanie sytuacji konsumpcji przez aplikację większej niż przydzielonej dla niej ilości zasobów, umożliwiając efektywne działanie innych zadań.

3.2.1. Dostosowanie środowiska do charakterystyki aplikacji

System decyzyjny wykonujący operacje przydziału zasobów powinien móc gromadzić i przetwarzać informacje na temat zapotrzebowania na zasoby dla aktualnie uruchomionych zadań. Informacje te powinny pozwolić na realizację efektywnego przydziału zadań do zasobów, a także na podejmowanie trafnych decyzji na temat ich konfiguracji. Klasyfikacja aplikacji[2] pozwala również na estymację zapotrzebowania, co z kolei pozwoli na wcześniejsze zlecenie (czasochłonnych) operacji ponownej konfiguracji zasobów.

Identyfikacja klasy aplikacji bądź fazy jej wykonania jest możliwa na podstawie wartości parametrów wydajnościowych zebranych za pomocą systemu monitorowania. Problemem jest określenie, które spośród dużej ilości udostępnionych przez system monitorowania parametrów, powinny być użyte w procesie regulacji dostępu do zasobów. Co więcej klasyfikacja ta powinna umożliwiać dostosowanie sposobu zbierania informacji do ciągle zmieniającego się stanu aplikacji czy fazy jej wykonania.

Moduł decyzyjny powinien umożliwiać realizacje powyższych metod poprzez zbieranie i porównywanie informacji na temat aktualnego poziomu wykorzystania zasobów w odniesieniu do przydzielonych limitów.

3.2.2. Izolacja aplikacji i wsparcie dla jakości usługi

Podstawowym założeniem koncepcyjnym systemu zarządzania zasobami jest izolacja aplikacji. Aplikacja, do której wykonania będą wykorzystywane elementy infrastruktury Grid zostaje dostarczona wraz z definicją wymagań. Przydzielenie zasobów dla tej aplikacji odbywać się będzie za pomocą koncepcji wirtualnego Gridu. Wirtualny Grid jest tutaj rozumiany jako rozproszony kontener, w którym nastąpi wykonanie aplikacji. Alokacja zasobów dla aplikacji będzie możliwa poprzez konfigurację dostępu do zasobów fizycznych (obliczeniowych, komunikacyjnych) dla wirtualnego Gridu. Wpływ na poziom wykorzystania zasobów przez aplikację jest możliwy bez konieczności modyfikacji aplikacji; jej działania, konfiguracji itp.

Kolejnym istotnym założeniem przyjętym podczas projektowania systemu jest koncepcja powiązania aplikacji z wydzielonym środowiskiem uruchomieniowym relacją jeden-do-jeden. Każda równocześnie uruchomiona aplikacja będzie posiadała własną przestrzeń dostępu do zasobów (sekcja 2.1.1) co spowoduje, iż aplikacje nie będą miały bezpośredniego

[2]Klasyfikacja ta w podstawowym ujęciu opierać się może na identyfikacji zapotrzebowania na czterech głównych płaszczyznach działania. Elementy te to wykorzystanie CPU systemu (aplikacje wykonujące dużą liczbę obliczeń), wykorzystanie pamięci operacyjnej, obciążenie operacjami I/O (aplikacje przetwarzające duże ilości danych) oraz wykorzystanie sieci.

wpływu na siebie, oraz iż możliwe będzie sterowanie dostępem do zasobów dla każdej aplikacji niezależnie. Izolacja ta poza oczywistą zaletą indywidualnej regulacji parametrów, pozwala również na zwiększenie poziomu bezpieczeństwa infrastruktury. Aplikacje będą uruchomione niezależnie, nie będą miały dostępu do danych, obszaru roboczego i komunikacji sieciowej (sekcja 2.2.1) innych aplikacji.

Wsparcie dla QoS jest możliwe, gdyż wirtualne środowisko może udostępniać właściwości grupowania heterogenicznych, rozproszonych zasobów w jeden kontener. Parametry i konfiguracja środowiska mogą gwarantować dostępność zasobów oraz ich minimalny wymagany poziom[3]. Dzięki temu możliwe jest opisanie właściwości aplikacji warunkami, na których spełnienie może być zawarty kontrakt gwarancji jakości usługi SLA (ang. *Service Level Agreement*) pomiędzy dostarczycielem usług obliczeniowych a klientem.

Środowisko wykonawcze dla aplikacji będzie tworzone na żądanie. Przestrzeń ta będzie budowana za pomocą zestawu wirtualnych maszyn VM o określonych przez system zarządzania parametrach i właściwościach. Aplikacja, dzięki zastosowaniu technik wirtualizacji, nie będzie świadoma, iż jest uruchomiona pod kontrolą systemu zarządzania zasobami.

3.2.3. Tworzenie wirtualnych topologii sieciowych

Tworzenie wirtualnych węzłów VM na żądanie, w celu określenia konfiguracji zasobów dla aplikacji musi być uzupełnione o wsparcie do modelowania infrastruktury komunikacyjnej. Stosując techniki wirtualizacji pojawia się możliwość zarówno dynamicznego instancjonowania elementów infrastruktury obliczeniowej VWN jak i kontroli kształtu oraz konfiguracji topologii sieciowej, która będzie wykorzystywana przez uprzednio dodane węzły.

Złożoność istniejących (fizycznych) topologii sieciowych tworzących sieć Internet, brak masowego wdrożenia nowych standardów i protokołów[4] oraz zastosowanie technik ograniczenia wykorzystania dostępnej przestrzeni adresowej[5], utrudniają tworzenie środowiska, którego głównym celem jest zapewnienie komunikacji wirtualnym węzłom.

Aby zniwelować wymienione właściwości, system musi udostępniać usługi tworzenia wirtualnej sieci[6] zbudowanej nad istniejącą siecią Internet[7]. Sieć ta powinna posiadać w pełni konfigurowalną topologię oraz powinna umożliwiać egzekwowanie narzuconych wcześniej parametrów komunikacji. Ponieważ elementy infrastruktury środowisk Grid są silnie rozproszone, a co za tym idzie należą do różnych sieci protokołu IP, to przy budowie sieci wirtualnej konieczne jest zastosowanie technik enkapsulacji bądź tunelowania. Sieć wirtualna powinna posiadać istotne, z punktu widzenia aplikacji uruchamianych w środowiskach Grid, cechy takie jak: wsparcie dla protokołów, których wykorzystanie bezpośrednie (bez stosowania np. technik tunelowania) na całej ścieżce w istniejącej sieci Internet jest niemożliwe oraz uproszczenie konfiguracji protokołu IP dla elementów infrastruktury.

[3] Możliwości technik wirtualizacji w obszarze gwarancji dostępu do zasobów zostały przedstawione w sekcji 2.3.6.

[4] Brak dostępności routingu multicastowego dużej skali, znikoma popularność protokołu IPv6 oraz nie zawsze dostępna możliwość bezpiecznej komunikacji protokołami IPSec czy SSL (ang. *Secure Sockets Layer*).

[5] Mechanizmy translacji adresów NAT wykorzystywane do podmiany adresów z przestrzeni prywatnej [78] oraz techniki dynamicznego przydziału adresacji np. protokół DHCP.

[6] Termin „sieć wirtualna" został tutaj użyty na określenie wydzielonej infrastruktury komunikacyjnej. W szczególności technologie użyte do budowy tej infrastruktury mogą być inne od tych używanych przy tworzeniu „wirtualnych sieci prywatnych" — VPN. Sieci VPN są z reguły przedłużeniem (fizycznych) topologii korporacyjnych, ze wsparciem dla weryfikacji tożsamości użytkowników oraz szyfrowaniem danych.

[7] Sieć Internet jest tutaj wykorzystywana jako uniwersalne medium komunikacyjne.

Analogicznie jak przy wirtualizacji systemu operacyjnego, dodatkowy poziom abstrakcji dla komunikacji sieciowej jakim jest tunelowanie ruchu wprowadza narzut powodujący zwiększenie ilości koniecznych do przesłania danych. Sytuację dodatkowo komplikuje konieczność wykorzystania tych sieci dla środowisk przetwarzania rozproszonego, których kluczową cechą jest wydajność. Dlatego stworzony system komunikacji, oprócz wcześniej wymienionych cech, powinien wprowadzać jak najmniejsze narzuty, zarówno na komunikację jak i na operacje (takie jak np. dodawanie lub migracja węzłów) związane z tworzeniem topologii. Cechy te powodują iż stworzenie infrastruktury, umożliwiającej budowanie sieci wirtualnych o dowolnych topologiach, ze wsparciem dla QoS oraz migracji VMów jest złożonym problemem.

3.2.4. Adaptowalność do zmiennych wymagań

Budowa systemów, które adaptują się do zmieniających wymagań jest zadaniem niezwykle trudnym. Ciągłym zmianom ulegają struktury organizacyjne w firmach, oprogramowanie, technologie których używają, oraz, być może najważniejszy z aspektów — klient używający ich produktów. Dlatego potrzeba tworzenia systemów adaptowalnych wydaje się coraz bardziej uzasadniona. Motorem napędowym w tworzeniu systemów adaptowalnych jest szczególnie postęp w dziedzinach związanych z wszechobecnym przetwarzaniem (ang. *ubiquitous computing*) [81], a szczególnie obszar związany z urządzeniami mobilnymi, który zaciera granice pomiędzy miejscem, sposobem i czasem korzystania z zasobów obliczeniowych [66].

Adaptowalność może być definiowana jako zdolność przystosowywania się aplikacji do zmieniających się ograniczeń środowiska oraz zmieniających się wymagań użytkowników bądź innych aplikacji [31]. Głównym więc celem adaptowalności jest osiągnięcie jakiegoś akceptowalnego minimum poziomu funkcjonalności czy jakość działania określona zazwyczaj przez specyfikacje QoS [31]. Wniosek, który się nasuwa można sformułować następująco: adaptowalność jest pożądaną cechą, gdyż przedłuża żywotność oprogramowania czyniąc go bardziej elastycznym, a z punktu widzenia programistów pozwala oszczędzić czas i zasoby. Najbardziej ogólny podział technik adaptacyjnych oprogramowania to:

adaptacja parametryczna polegająca na dostosowywaniu odpowiednich parametrów do zmieniających się warunków środowiska.[8]

adaptacja kompozycyjna polegająca na zmianie struktury i algorytmów oprogramowania, dostosowując go do nieprzewidzianych wcześniej warunków. Techniki adaptacji kompozycyjnej można zaś skategoryzować (uszeregowane w kolejności łatwości implementacji) następująco:

- **adaptacja architektury** — dostosowanie całej aplikacji do nowych warunków wymaga często zmian na poziomie projektu systemu,
- **adaptacja kodu** — modyfikacji ulegają konkretne fragmenty kodu odpowiedzialne za dostarczenie nowej funkcjonalności,
- **adaptacja komponentowa** — z tym, że nie jest modyfikowany kod implementujący funkcjonalność komponentów, a modyfikacja aplikacji odbywa się poprzez

[8] Przykładem może być mechanizm okna przesuwnego transmisji TCP, uzależniający szybkość transmisji od aktualnie dostępnego pasma komunikacyjnego.

rozszerzenia komponentów oraz ich rekonfigurację. Tego typu podejście idealnie wspiera programowanie aspektowe [50, 70] poprzez separację aspektów funkcjonalnych od niefunkcjonalnych aplikacji i nanoszenie modyfikacji jedynie w aspektach niefunkcjonalnych (np. zarządzanie zasobami, protokoły komunikacyjne, persystentność, bezpieczeństwo, itd.) [64].

Osobny aspekt technik adaptacji stanowi adaptowalność autonomiczna. Adaptacja jest tutaj połączona z technikami opartymi o zdobywanie wiedzy i pozwala systemom dostosowywać swoje zachowanie do nowych warunków, wymogów na podstawie wiedzy o zmianach w środowisku. Techniki te są zakładają, że system będzie sam gromadził wiedzę na temat działania środowiska, która następnie będzie wykorzystywana do podejmowania decyzji. Jest to technika skomplikowana, zarówno koncepcyjnie (wymaga szczegółowej znajomości dziedziny problemu) jak i implementacyjne (mechanizmy gromadzenia i wykorzystania wiedzy). Do tej kategorii zaliczają się również systemy odzwierciedlające zmiany w modelach znajdujących się w repozytorium.

Powyższe podziały odpowiadają na pytanie w jaki sposób dokonać adaptacji. Można zadać sobie również pytanie kiedy dokonać adaptacji. Według tego podziału adaptacja dzieli się na statyczną (zakodowana w programie, dostosowana np. poprzez deskryptory plików, moduły ładowane w trakcie wykonywania programu) i dynamiczną (modyfikacja parametrów, modyfikacja całej struktury programu) odbywającą się w trakcie działania systemu. Zarówno statyczna jak i dynamiczna adaptacja może odbywać się bezpośrednio w samej aplikacji jak i w warstwie pośredniczącej. Oba podejścia mają odpowiedni wpływ na zachowanie aplikacji, jednak w drugim przypadku istnieją ograniczenia tego co może podlegać adaptacji.

3.3. Analiza wymagań

Model architektury systemu zarządzania zasobami w środowiskach Grid silnie zależy od przyjętych na wstępie wymagań, które tworzony system powinien spełniać. Wymogi te zostały podzielone na klasy: cechy funkcjonalne — określenie możliwości systemu widzianych z poziomu jego użytkowników oraz właściwości i wymagania niefunkcjonalne. Do wymagań niefunkcjonalnych zalicza się ograniczenia w działaniu systemu wynikające z konieczności dostosowania go do istniejących rozwiązań, konkretnych warunków działania czy z potrzeby współpracy z innymi środowiskami. Wymagania niefunkcjonalne zostały również określone i przedstawione w celu zagwarantowania, iż nowo tworzony system będzie oparty o najnowsze trendy i zalecenia projektowe dla środowisk rozważanego typu.

3.3.1. Wymagania ogólne dla systemów zarządzania zasobami

Wymagania ogólne dla systemu możemy przedstawić poprzez zdefiniowanie zestawu podstawowych usług i serwisów jakie system zarządzania powinien posiadać. Dzięki temu można będzie przedstawić proces zarządzania zasobami jako kolekcję serwisów o określonych wymogach i właściwościach.

Poniższa lista usług wchodzących w skład systemu zarządzania zasobami dla nowoczesnych architektur Grid została zaproponowana przez grupę GSA-RG (ang. *Grid Scheduling*

Architecture Research Group)[9] [34]:

Wykrywanie zasobów — działanie tej usługi polega na lokalizacji zasobów spełniających określone parametry. Wyszukiwanie to powinno również pozwalać na wykrywanie zmian związanych z dostępnością zasobów.

Dostęp do informacji na temat zasobów — właściwości dostępnych zasobów powinny być określone za pomocą odpowiednio dobranego zestawu parametrów. Parametry te to zarówno wartości statyczne[10] opisujące infrastrukturę jak i dynamiczne[11] przedstawiające jej obciążenie.

Monitorowanie stanu zadań i środowiska — podczas uruchamiania zadań oraz podczas ich wykonania konieczne jest śledzenie stanu aplikacji i środowiska. System monitorowania powinien udostępniać usługi notyfikacji o zdarzeniach istotnych z punktu widzenia procesu zarządzania zasobami.

Przydział zasobów dla zadań — ważne jest aby wyszukiwanie i przydział zasobów dla zadań nie wymagał ingerencji użytkownika. Przydział ten powinien być dokonywany automatycznie i powinien uwzględniać wymagania aplikacji. Ze względu na zmiany wymogów aplikacji podczas wykonania, powiązanie zasobów powinno być czasowo weryfikowane i w miarę potrzeb modyfikowane.

Obsługa rezerwacji/limitów/kontraktów SLA — większość użytkowników zakłada, iż zgłoszone przez nich zadania będą mogły korzystać ze wszystkich aktualnie dostępnych zasobów (ang. *best effort*). Niekiedy jednak występują sytuacje, w których wymagane jest wsparcie ze strony infrastruktury dla limitów wykorzystania, wynikających z rezerwacji czy konieczności spełnienia warunków umów SLA. Limity te można określić jako parametry QoS dla infrastruktury (zarówno obliczeniowej jak i komunikacyjnej).

Zarządzanie wykonaniem zadań — czyli możliwość wykonywania takich operacji jak zlecanie i przerywanie zadań, czy wykonywanie i odwoływanie rezerwacji. Jest to główny (z poziomu użytkowników) obszar działania systemów RMS w środowiskach Grid.

Zliczanie i raportowanie — w wielu sytuacjach wymagana jest możliwość mierzenia poziomu zużycia zasobów. Informacje te mogą być wykorzystane jako dane wejściowe pozwalające opracować modele biznesowe działania infrastruktury.

Przedstawione ogólne wymagania odnośnie serwisów i usług powinny być wzięte pod uwagę na etapie projektowania dowolnego systemu zarządzania zasobami w środowiskach Grid. Te oraz opisane w następnych sekcjach szczegółowe wymagania funkcjonalne i niefunkcjonalne, zostaną wykorzystane do określenia zakresu funkcjonalności systemu zarządzania RMS z wykorzystaniem technik wirtualizacji.

[9] Grupa ta wchodzi w skład organizacji GGF (ang. *Global Grid Forum*).

[10] Wartości niezmienne w czasie jak np. zegar i liczba rdzeni CPU, przepustowość interfejsu sieciowego, itp.

[11] Takie jak np. aktualne obciążenie procesora, pamięci czy liczba przesłanych bajtów przez dany proces obliczeniowy, itp.

3.3.2. Wymagania funkcjonalne

Wymagania funkcjonalne zdefiniowane zostały dla dwóch klas użytkowników systemu. Klasy te to „zwykły użytkownik" — mogący jedynie zlecać zadania oraz zbierać wyniki ich działania, oraz „administrator" — wykonujący funkcje związane z obsługą systemu i infrastruktury.

Zakłada się, że system RMS powinien spełniać następujące wymagania funkcjonalne istotne z punktu widzenia jego użytkowników:

1. Niskie narzuty na zarządzanie zasobami

Obciążenie generowane przez działanie komponentów systemu RMS powinno być jak najmniejsze. Równocześnie możliwie niski powinien być narzut wprowadzony przez zastosowane techniki sterowania dostępem i konfiguracją zasobów. Wprowadzane obciążenie relatywne oraz narzut na uruchomione aplikacje będzie zależał od przyjętych wcześniej założeń dla klas zastosowań środowiska. Dla aplikacji w dużym stopniu korzystających z zasobów proporcje pomiędzy efektywnością działania w środowisku natywnym oraz pod kontrolą systemu zarządzania zasobami będą odpowiednio mniejsze.

2. Możliwość uruchamiania wielu aplikacji równocześnie

System powinien dostarczać możliwość współdzielenia dostępnej puli zasobów dla różnych, równocześnie uruchomionych aplikacji. Dzięki temu możliwe będzie osiągnięcie wyższych wartości współczynników charakteryzujących wykorzystanie infrastruktury. Parametry te, są szczególnie istotne z punktu widzenia administratora. Współdzielenie infrastruktury powinno być możliwe wraz z elastycznym mechanizmem umożliwiającym definiowanie poziomów dostępności zasobów dla każdej aplikacji niezależnie. Definicje te powinny być respektowane przez podsystem kontroli przydziału zasobów w całym czasie działania aplikacji.

3. Izolacja uruchomionych zadań

Równoczesne uruchomienie niezależnych zadań, wykorzystujących tą samą fizyczną infrastrukturę może powodować szereg niedogodności. Jedną z nich jest wymóg konieczności zagwarantowania, poprzez zastosowanie odpowiednich technik i procedur, braku wzajemnych bezpośrednich i pośrednich interakcji aplikacji. Spełnienie tego wymogu jest możliwe na wiele sposobów, jednak nie wszystkie z nich gwarantują izolację na odpowiednio wysokim poziomie. Kluczowym zadaniem jest zastosowanie odpowiednio dobranych rozwiązań wprowadzających izolację wykonania, tak aby dowolne operacje[12] z różnych aplikacji nie zakłócały się nawzajem.

4. Możliwość zatrzymywania procesów obliczeniowych

Czyli możliwość zawieszenia na dowolnie długi czas działania aplikacji (całej, bądź tylko niektórych jej komponentów). Potrzeba taka może wynikać z konieczności chwilowego zwolnienia zasobów na skutek koniecznych do wykonania prac administracyjnych lub np. wy-

[12]Dopuszczalne są sytuacje, w których aplikacje ze względu na awarie czy błędne konfiguracje będą wymagać więcej zasobów ponad dostępny dla nich poziom.

mogów przyśpieszenia wykonania dla innych bardziej kluczowych zadań. Stan aplikacji po wznowieniu wykonania powinien być identyczny ze stanem aplikacji przed zatrzymaniem. Dotyczy to zarówno stanu samej aplikacji (np. jej aktualnie otwartych połączeń sieciowych czy zawartości przetwarzanych plików) jak i konfiguracji środowiska udostępniającego zasoby dla danej aplikacji.[13]

5. Umożliwienie zawieszenia działania infrastruktury

Zawieszenie działania infrastruktury jest uzupełnieniem funkcjonalności, dla przedstawionej wcześniej, możliwości zatrzymywania procesów obliczeniowych. Zatrzymywanie procesów obliczeniowych umożliwia jedynie częściowe zwolnienie zasobów[14]. Zawieszenie infrastruktury będzie wykonywane poprzez przeprowadzenie operacji zapisu stanu całego środowiska uruchomieniowego dla danej aplikacji, a następnie wykonania procedury jego usunięcia. Zwolnione w ten sposób zasoby będą mogły być wykorzystane przez inne zadania, lub możliwe będzie całkowite ich odłączenie spod kontroli systemu zarządzania (np. poprzez czasowe wyłączenie infrastruktury).

6. Możliwość precyzowania zapotrzebowania aplikacji na zasoby

Aplikacja może być dostarczona przez użytkownika wraz z opisem charakteryzującym jej wymogi odnośnie konfiguracji i dostępności zasobów. Charakterystyka ta określa pewną konfigurację środowiska obliczeniowego, którego dostępność spowoduje bardziej efektywne działanie aplikacji. Specyfikacja ta powinna móc określać zarówno konfigurację zasobów obliczeniowych, wpływających na efektywność wykonywanych operacji przetwarzania jak i zasoby, i kształt topologii sieciowej koniecznej do zapewnienia komunikacji dla rozproszonych komponentów aplikacji.

7. Wsparcie dla autonomicznej i ręcznej konfiguracji środowiska

Podejmowane przez system autonomiczne decyzje odnośnie dopasowania konfiguracji środowiska do wymogów aplikacji powinny być możliwe do modyfikacji przez użytkownika. Działania te pozwolą na poprawienie ewentualnych błędów konfiguracji przydziału zasobów i pozwolą na osiągnięcie przez aplikację lepszej wydajności.

8. Dostępność graficznego narzędzia do zarządzania

Aktualny stan i dostępność zasobów dla aplikacji powinien być dostępny dla użytkownika poprzez graficzne narzędzie monitorujące. Oprogramowanie to będzie miało za zadanie śledzenie konfiguracji i efektywności wykonania dla danej aplikacji, wprowadzenie i modyfikację specyfikacji opisującej jej wymogi oraz modyfikację (dostępnych dla użytkownika) reguł działania systemu.

[13] Spełnienie tego wymogu nie będzie możliwe dla aplikacji czasu rzeczywistego lub takich, których poprawne działanie zależy od wyników pomiarów czasu, dokonywanych w trakcie ich działania.

[14] Uzyskuje się wtedy dostęp do np. czasu procesora, podczas gdy takie elementy jak pamięć czy zasoby komunikacyjne będą nadal wykorzystywane.

Wymagania funkcjonalne poziomu operatora

Wymaga się aby system zarządzania zasobami spełniał następujące wymagania funkcjonalne ważne z punktu widzenia jego administratorów:

9. Wsparcie do zarządzania w heterogenicznych konfiguracjach systemowych

Działanie systemu powinno wspierać różne platformy sprzętowe i programowe środowisk przetwarzania rozproszonego. Założenie to jest rozumiane jako brak ograniczeń na poziomie modelu systemu wynikających z przyjętej architektury, czy zastosowanych koncepcji budowy środowiska, które to ograniczałyby obszar jego zastosowań. W implementacji systemu mogą wystąpić jednak ograniczenia będące skutkiem braku wsparcia dla konkretnej architektury systemów komputerowych czy platform oprogramowania w wykorzystanych narzędziach i rozwiązaniach.[15]

Środowisko musi mieć również możliwość efektywnego działania w sytuacji obecności zasobów o niejednorodnych charakterystykach. Wymóg ten wynika z faktu, iż w większości przypadków nie można oczekiwać homogeniczności zasobów używanych we współczesnych środowiskach obliczeniowych.

10. Możliwość określenia parametrów QoS

Działanie systemu będzie sprowadzało się do zapewnienia efektywnego podziału infrastruktury komunikacyjnej i obliczeniowej pomiędzy równocześnie uruchomione aplikacje. Podział ten powinien być dokonany w taki sposób aby możliwe było osiągnięcie odpowiedniego poziomu wykonania aplikacji udostępniającej swoje usługi dla użytkowników systemu. Użytkownik powinien móc zatem określić minimalny akceptowalny przez niego poziom jakości usługi. Specyfikacja ta dostarczona wraz z aplikacją posłuży do wyznaczenia istotnych z punktu widzenia użytkownika parametrów jej środowiska uruchomieniowego, których zapewnienie na określonym w specyfikacji poziomie będzie mogło być przez system zagwarantowane. Gwarancje jakości wykonania QoS będą mogły być określone autonomicznie[16] lub półautonomicznie[17] na podstawie charakterystyki aplikacji oraz określonej polityki działania systemu. Parametry QoS powinny zatem określać:

- jakość usług oferowanych przez aplikację,
- właściwości środowiska uruchomieniowego,
- właściwości komunikacji sieciowej.

11. Łatwe dostosowanie do istniejących rozwiązań

Opracowany model środowiska powinien zakładać niezależność zarówno od rodzaju jak i konkretnej implementacji używanego oprogramowania warstwy pośredniczącej

[15] Przykładem może tutaj być ograniczenie platform sprzętowych, na których dostępna jest dana implementacja technik (para)wirtualizacji.

[16] System sam dobiera (na podstawie własnej polityki działania) parametry istotne dla określonej klasy aplikacji.

[17] Zasoby określone przez użytkownika są jedynie weryfikowane pod kątem dostępności i jeśli będzie to możliwe będą one zagwarantowane na określonym poziomie.

(ang. „*middleware*") systemów grid. Niezależność ta wynikać będzie z faktu umieszczenia systemu RMS pomiędzy infrastrukturą (zasobami) a warstwą *middleware*. Koncepcja ta pozwoli na łatwą integrację proponowanego rozwiązania z istniejącą infrastrukturą oraz z oprogramowaniem do jej obsługi.

12. Elastyczna definicja reguł działania systemu

W systemie zarządzania dużą rolę odgrywał będzie moduł decyzyjny, którego działanie polegać będzie na wykonywaniu akcji dokonujących modyfikacji podziału zasobów. Decyzje te będą podejmowane na podstawie aktualnego stanu aplikacji, jej wymogów oraz ustalonych na podstawie polityki funkcjonowania systemu ograniczeń. Odpowiednie dobranie algorytmów działania dla tego komponentu będzie kluczowe z punktu widzenia efektywności całego środowiska. Reguły, na podstawie których będą podejmowane decyzje powinny umożliwiać elastyczne definiowanie wszystkich aspektów działania podsystemu przydziału zasobów oraz powinny zapewniać możliwość dostosowania do różnorodnych wymagań aplikacji.

Ze względu na różnorodny charakter aplikacji wykonywanych w środowiskach rozproszonych oraz zmienności ich wymagań w czasie, stworzenie uniwersalnego a zarazem optymalnego algorytmu przydziału zasobów może okazać się skomplikowanym zadaniem. Dlatego konieczna będzie możliwość określenia dla konkretnej aplikacji procedur optymalizacji środowiska dostosowanych do jej wymogów i charakterystyki. Procedury te zostałyby użyte do obsługi przydziału zasobów tylko dla tej jednej, konkretnej aplikacji.

Należy również przewidzieć sytuację, w której konieczna będzie aktualizacja (poprzez uzupełnienie lub całkowitą wymianę) reguł modułu decyzyjnego. Operacja ta powinna mieć możliwość wykonania bez konieczności zatrzymywania pracy zarządzanej aplikacji lub przerywania działania środowiska. Dostosowanie to powinno być możliwe z poziomu uprawnień użytkownika mającego dostęp do operacji związanych z zarządzaniem daną aplikacją.

13. Automatyzacja procesu instalacji środowiska

Początkowa instalacja i konfiguracja środowiska powinna być maksymalnie uproszczona poprzez zastosowanie następujących elementów i rozwiązań:

- przygotowanie zestawu skryptów instalacyjnych oraz odpowiednio skonfigurowanych obrazów mediów instalacyjnych dla głównych komponentów systemu,
- włączenie możliwości instalacji systemu operacyjnego zawierającego odpowiednie oprogramowanie[18] jak i opcji uruchomienia tej samej funkcjonalności w konfiguracji bez-stanowej (ang. *state-less*)[19],
- rozdzielenie funkcjonalności zależnej od sprzętu umożliwi realizację procedur dynamicznego ładowania i uruchamiania wymaganych komponentów w zależności od rodzaju używanych zasobów.

[18] Jak np. implementację obsługi (para)wirtualizacji.

[19] System operacyjny wraz z oprogramowanie zostanie załadowany i uruchomiony z pamięci operacyjnej (brak pamięci masowej).

14. Wysoki poziom bezpieczeństwa infrastruktury

Wymagane jest zapewnienie odpowiedniego poziomu bezpieczeństwa systemu, tak aby zminimalizować ryzyko ewentualnych problemów związanych z nieautoryzowanym dostępem. System powinien zatem implementować podstawowe metody weryfikacji tożsamości i uprawnień użytkowników. Powinny być również zastosowane mechanizmy zabezpieczające infrastrukturę środowiska przed zagrożeniami wynikającymi z dopuszczenia wykonywania potencjalnie niebezpiecznego kodu aplikacji obliczeniowych. Mechanizmy te pozwolą na izolację wykonania aplikacji i ograniczenie jej ewentualnego szkodliwego działania.

W wyborze zastosowanych technologii i w trakcie tworzenia projektu architektury systemu należy rozważyć również ryzyko związane z monopolizacją dostępu do zasobów (przez dowolny z komponentów systemu). Monopolizacja ta może być spowodowana poprzez ataki na odmowę usługi typu DoS (ang. *Denial-of-Service*) przeprowadzane na elementy infrastruktury lub przez awarię któregoś z komponentów. Zabezpieczenie komunikacji sieciowej, dla sieci wirtualnej tworzonej na żądanie dla aplikacji, wymagać będzie nierzadko dodatkowych mechanizmów obsługujących szyfrowanie ruchu i uwierzytelnienie komunikujących się elementów. Mechanizmy te będą wymagane w sytuacji gdy jako medium komunikacyjne wykorzystywana jest publiczna sieć Internet.

15. Dostępność graficznego narzędzia zarządzania środowiskiem

Analogicznie jak dla użytkownika, również dla administratora konieczna będzie aplikacja obrazująca aktualny stan i dostępność zasobów dla wszystkich aktualnie przetwarzanych aplikacji. Oprogramowanie to, podobnie jak w wersji dla zwykłego użytkownika, będzie miało za zadanie śledzenie konfiguracji i efektywności działania środowiska oraz umożliwi wprowadzenie i modyfikację polityk zarządzania zasobami i reguł działania systemu.

3.3.3. Wymagania niefunkcjonalne

Przyjęcie wymienionych niżej wymagań ma na celu zagwarantowanie, iż nowo tworzone rozwiązanie będzie zgodne z szeroko przyjętymi i uznanymi zaleceniami dla systemów tego typu.

Pożądane jest aby system spełniał następujące wymagania niefunkcjonalne:

1. Wykorzystanie istniejących standardów

Rozbudowa systemów informatycznych o nowe funkcje i właściwości jest ograniczona między innymi poprzez brak wykorzystania ogólnie przyjętych standardów i notacji. Zastosowanie sprawdzonych i popularnych standardów przy przedstawieniu architektury, sposobu realizacji implementacji umożliwia szybkie zrozumienie zasad działania projektowanego systemu.

Poprzez wybór odpowiednich narzędzi i rozwiązań, zastosowanie sprawdzonych zaleceń i koncepcji oraz odpowiednie zaprojektowanie architektury, pozwoli na stworzenie środowiska o komponentowej budowie upraszczającej jego dalszy rozwój i dostosowanie do nowych wymagań.

2. Wykorzystanie koncepcji MDA przy opracowaniu architektury

MDA (ang. *Model Driven Architecture*) jest metodologią tworzenia oprogramowania, która zakłada automatyczne tworzenie oprogramowania na podstawie graficznego modelu systemu. Metodologia ta nie precyzuje żadnych narzędzi, technologii oraz języka programowania. Dzięki temu możliwe jest zaprojektowanie systemu, który w fazie implementacji może być zrealizowany z użyciem różnych technologii.

3. Zastosowanie paradygmatu SOA

Ze względu na rosnącą złożoność systemów informatycznych, obserwowany jest odwrót od koncepcji monolitycznych architektur. Wymóg szybkiej i efektywnej implementacji złożonych systemów powoduje, iż dominujące przy ich budowie stają się technologie komponentowe. Architektura SOA (ang. *Service Oriented Architecture*) [13, 30, 74] jest to koncepcja, według której oprogramowanie będzie tworzone z użyciem dynamicznie budowanych powiązań pomiędzy interfejsami oferentów usługi a komponentami, które z tych usług korzystają. W koncepcji tej, główny nacisk położony jest na odpowiednie zdefiniowanie właściwości usługi (implementującej moduł oprogramowania realizujący wymagane przez użytkownika funkcje biznesowe). Usługi określone są zazwyczaj na wyższym poziomie abstrakcji, umożliwiając tym samym osiągnięcie niezależności od platformy programistycznej. Niezależność od różnych technologii implementacji usługi uzyskuje się również przez użycie uniwersalnego protokołu komunikacyjnego.

4. Skalowalność i efektywność działania

Zakłada się, iż obciążenie systemu[20] będzie zależne od ilości zarządzanych zasobów oraz ilości przetwarzanych aplikacji. Wymóg zapewnienia skalowalności systemu to konieczność zapewnienia mechanizmów umożliwiających efektywne działanie w sytuacji znacznego wzrostu ilości przetwarzanych zadań. Cecha ta osiągnięta zostanie poprzez zaproponowanie hierarchicznej koncepcji zarządzania związanej z podziałem zasobów. Wyznaczenie grup zasobów będzie możliwe poprzez odpowiednie określenie granic na podstawie metryk opłacalności ich wykorzystania dla danej aplikacji. Zasoby rozdzielone na takie, których wykorzystanie umożliwi zwiększenie efektywności działania aplikacji oraz takie, dla których koszt użycia będzie zbyt wysoki, pozwoli na zmniejszenie ilości przetwarzanych przez system informacji oraz na uproszczenie reguł decyzyjnych związanych z zarządzaniem zasobami.

5. Wykorzystanie narzędzi *Open Source*

Wykorzystanie darmowego oprogramowania zarówno w postaci narzędzi programistycznych używanych podczas tworzenia systemu i testowania jak i w formie bibliotek i oprogramowania systemowego jest priorytetem. Oprogramowanie to w większości przypadków jest dostępne na wolnej licencji gwarantującej swobodny dostęp do kodu źródłowego. Spełnienie tego zalecenia pozwoli na:

- redukcję kosztów związanych z licencjonowaniem używanego oprogramowania,

[20] Jest to dodatkowe obciążenie infrastruktury wnoszone przez system a związane z konsumpcją zasobów przez poszczególne komponenty środowiska.

- umożliwienie wprowadzania własnych poprawek i rozszerzeń do kodu źródłowego,
- łatwa do uzyskania kompatybilność ze stosowanym obecnie oprogramowaniem tworzonym dla środowisk Grid.

Dodatkowo wymóg ten jest zgodny z zaleceniami odnośnie stosowania oprogramowania FLOSS (ang. *Free Libre/Open Source Software*) w wielu projektach[21] tworzących rozwiązania dla środowisk Grid.

3.3.4. Klasy zastosowań systemu

Przyjęcie określonego modelu architektury i działania systemu będzie również determinowało klasę jego zastosowań. Wykorzystanie wirtualizacji, modelowania infrastruktury sieciowej oraz technik umożliwiających dynamiczne instancjonowanie i adaptację środowiska wprowadza dodatkowy narzut. Równocześnie trafność decyzji modyfikujących proces wykonania aplikacji silnie zależy od ilości i jakości zgromadzonych danych wejściowych. Gromadzenie kompletnych danych opisujących działanie aplikacji, zachowanie środowiska i wpływu sterowania na efektywność wykonywanych zadań wymaga odpowiednio długiego czasu obserwacji.

Te oraz przedstawione w sekcji 3.3 wymagania pozwalają na określenie następujących właściwości aplikacji i środowiska, dla których system będzie miał zastosowanie:

- Czas działania aplikacji powinien być odpowiednio długi — umożliwi to zbieranie dokładniejszych informacji na temat przebiegu jej wykonania, a także na śledzenie wpływu adaptacji i dostosowania infrastruktury do założonych wcześniej wymagań (określonych w definicji polityki adaptacji).
- Aplikacje powinny znacząco wykorzystywać zasoby infrastruktury — zadania, dla których wykorzystanie infrastruktury jest duże mogą więcej skorzystać na procesach adaptacji. Również wspomniane wcześniej dodatkowe narzuty związane z funkcjonowaniem infrastruktury będą miały mniejszy udział w całkowitym czasie wykonania aplikacji.

3.4. Ogólna koncepcja realizacji systemu

Problem, którego dotyczy niniejsza praca to zaprojektowanie i implementacja środowiska zarządzania zasobami infrastruktury Grid o cechach podanych w tym rozdziale. Przyjęta w niniejszej pracy koncepcja realizacji systemu zakłada opracowanie jego modelu, a następnie implementację i praktyczną weryfikację.

Aby możliwa była konstrukcja systemu spełniającego podane w sekcji 3.3 zarówno formalne jak i nieformalne wymogi, konieczne będzie rozwiązanie problemów technicznych występujących na każdym, niżej wymienionym etapie konstrukcji:

1. Faza modelowania. Wykonanie tego etapu ma na celu identyfikację celu działania i głównych funkcji systemu. Etap ten zostanie zrealizowany poprzez stworzenie opisu

[21] Lista ważniejszych z nich to: EDG (ang. *EU DataGrid Project* IST-2000-25182) [22], CG (ang. *EU CrossGrid Project* IST-2001-32243) [21] czy EGEE (ang. *Enabling Grids for E-sciencE Project* IST-2003-508833) [29].

głównych elementów architektury za pomocą określenia realizowanej przez nie funkcjonalności. Ważna na tym etapie jest również identyfikacja powiązań i zależności pomiędzy komponentami środowiska. Faza modelowania jest również istotna ponieważ od jakości jej wykonania silnie zależą przyszłe możliwości środowiska oraz spełnienie bądź nie, przedstawionych w sekcji 3.3 wymagań.

2. Faza implementacji. Na tym etapie następuje implementacja środowiska zarządzania na podstawie modelu działania opracowanego w fazie pierwszej. Ważne jest aby większa złożoność infrastruktury, dodatkowy narzut i konieczne do wykonania prace integracyjne miały jak najmniejszy udział w stosunku do wprowadzonej przez system nowej funkcjonalności. Głównymi problemami, które należy pokonać przy implementacji to:

   - dobór odpowiednich narzędzi i środowisk programistycznych, użycie rozwiązań programowych wspierających tworzenie adaptowalnych, komponentowych i skalowalnych systemów zarządzania,
   - odpowiedni dobór technik wirtualizacji. Zastosowanie wirtualizacji jako techniki umożliwiającej elastyczne konfigurowanie dostępu do zasobów jest szeroko stosowane. Systemy Grid są jednak środowiskami rozproszonymi, przed którymi stawia się główne wymagania zapewnienia efektywnego wykonywania przetwarzania. Dlatego od własności techniki wirtualizacji zależy czy koncepcja jej wykorzystania, jako kluczowego mechanizmu działania systemu zarządzania zasobami, nie będzie wprowadzała znaczącej degradacji wydajności środowiska.

3. Faza praktycznej weryfikacji. Zaproponowane koncepcje, architektura oraz rozwiązania programowe takie jak zastosowane technologie i implementacje komponentów systemu zostaną poddane praktycznej weryfikacji. Weryfikacja zarówno formalnych jak i nieformalnych założeń, zostanie przeprowadzona w oparciu o prototypową implementację systemu. Głównymi problemami tego etapu tworzenia środowiska są:

   - odpowiednie przygotowanie i konfiguracja sprzętowej oraz programowej infrastruktury testowej,
   - opracowanie zestawu aplikacji testowych o zróżnicowanych właściwościach i wymaganiach oddających typowe zastosowania dla środowisk Grid,
   - opracowanie scenariuszy testów pozwalających na wiarygodną ocenę właściwości środowiska oraz identyfikację jego potencjalnie nieefektywnych działań.

Zastosowanie takiego podejścia pozwala na stworzenie spójnego i funkcjonalnego rozwiązania. Niniejsza praca jest zatem propozycją rozwiązania problemu konstrukcji systemu, spełniającego i posiadającego wszystkie wymagania oraz cechy wymienione w sekcji 3.3.

3.5. Podsumowanie

Opracowanie koncepcji i stworzenie systemu zarządzania zasobami w środowiskach Grid z wykorzystaniem wirtualizacji, stwarza szereg wyzwań. Najważniejszym z nich będzie

niewątpliwie opracowanie mechanizmów umożliwiających realizację autonomicznego zarządzania zasobami poprzez wykorzystanie koncepcji systemów adaptowalnych i samozarządzalnych (ang. *self-management*).

Przy wyborze technologii, dla praktycznej realizacji przyjętej koncepcji zarządzania zasobami w środowiskach Grid, istotne jest odpowiednie wyważenie pomiędzy możliwościami dostępnymi dzięki wykorzystaniu technik wirtualizacji (jako mechanizmu umożliwiającego elastyczne zarządzanie zasobami) a dodatkowym narzutem przez nie wprowadzanym.

Zdaniem autora wstępne zebranie cech i (niekiedy kolidujących ze sobą) wymagań jest konieczne dla podjęcia trafnych decyzji odnośnie modelu systemu i technologii jego praktycznej implementacji. Zamieszczone w tym rozdziale informacje posłużą zatem jako założenia do opracowania modelu systemu przedstawionego w rozdziale 4 oraz jako wskazówki dla wyboru technik jego praktycznej realizacji (rozdział 6).

Rozdział 4

Model systemu zarządzania zwirtualizowanymi zasobami Grid

Systemy Grid, jak większość systemów rozproszonych, do sprawnego i efektywnego działania wymagają obecności odpowiedniego systemu zarządzania. Zarządzanie zadaniami, aplikacjami, bezpieczeństwem, elementami podsystemu gromadzenia danych oraz infrastrukturą sieciową jest potencjalnie skomplikowanym zadaniem. Złożoność ta wynika głównie z konieczności zapewnienia obsługi dużej liczby niejednorodnych zasobów oraz z ich heterogeniczności. Dodatkowo skomplikowanie systemu wzrasta również z powodu rozproszenia zarządzania pomiędzy różne domeny administracyjne.

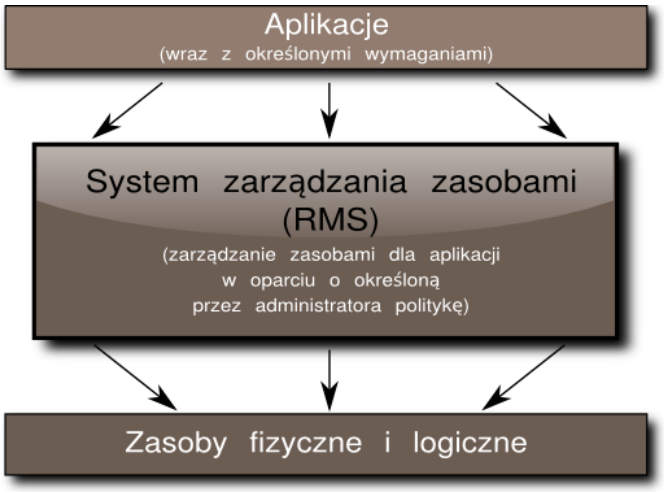
W rozdziale tym zostanie zaproponowany model systemu tworzenia infrastruktury obliczeniowej i zarządzania jej zasobami w oparciu o zwirtualizowane komponenty. Jako pierwsze zostaną przedstawione definicje ogólnych terminów związanych z zarządzaniem w systemach informatycznych. Następnie zostanie przedstawiony model podstawowy (statyczny) wirtualnego Gridu, umożliwiający w sposób formalny przedstawienie zależności pomiędzy elementami środowiska. Model ten zostanie następnie rozszerzony o właściwości umożliwiające realizację procedur adaptowalności do zmian wymagań aplikacji w czasie jej działania oraz do zmian w konfiguracji oraz dostępności zasobów. W dalszej części rozdziału na podstawie modelu dynamicznego zostanie opracowana i przedstawiona ogólna architektura systemu zarządzania zasobami wykorzystująca koncepcję wirtualnego Gridu.

4.1. Model systemu

Wszystkie aplikacje w trakcie działania wymagają dostępu do zasobów bądź obecności zasobów na wymaganym poziomie, tak aby móc wykonywać zlecone zadania. Wymagania aplikacji (czyli pośrednio również zapotrzebowanie na zasoby) mogą się zmieniać w zależności od zmieniających się wymagań (np. oczekiwanej wydajności, jakości zwracanych rezultatów, itp.) stawianych przed samą aplikacją. Przydzielanie zasobów w klasycznych systemach obliczeniowych odbywa się zwykle w fazie uruchomienia aplikacji i jest niezmienne w trakcie jej wykonania. Z tego powodu w środowiskach tych mogą wystąpić sytuacje (statycznego) przydziału zasobów do zadań, których rezultatem będzie nieefektywne działanie aplikacji oraz słabe wykorzystanie infrastruktury.

System zarządzania zasobami RMS pełni rolę pośrednika pomiędzy aplikacją a zasobami infrastruktury i jest odpowiedzialny za powiązanie obu komponentów czyli przydzielenie zasobów[1] dla aplikacji (rysunek 4.1).

[1]W projektowanym systemie, zgodnie z przyjętymi we wstępie założeniami odnośnie reprezentacji za-



Rysunek 4.1. Klasyczny system zarządzania zasobami

Wprowadzenie dodatkowego poziomu abstrakcji (rysunek 4.2), umożliwiającego dynamiczne[2] grupowanie zasobów w kontenery, których zmiany konfiguracji pozwolą na dostosowanie (optymalizację) przydziału zasobów w czasie wykonania aplikacji. Dzięki temu możliwe jest poprawienie efektywności działania całego środowiska. Zastosowanie technik wirtualizacji pozwala na elastyczne zarządzanie zasobami w trakcie działania aplikacji, bez przerywania jej pracy, co jest trudne w realizacji w klasycznych systemach RMS.

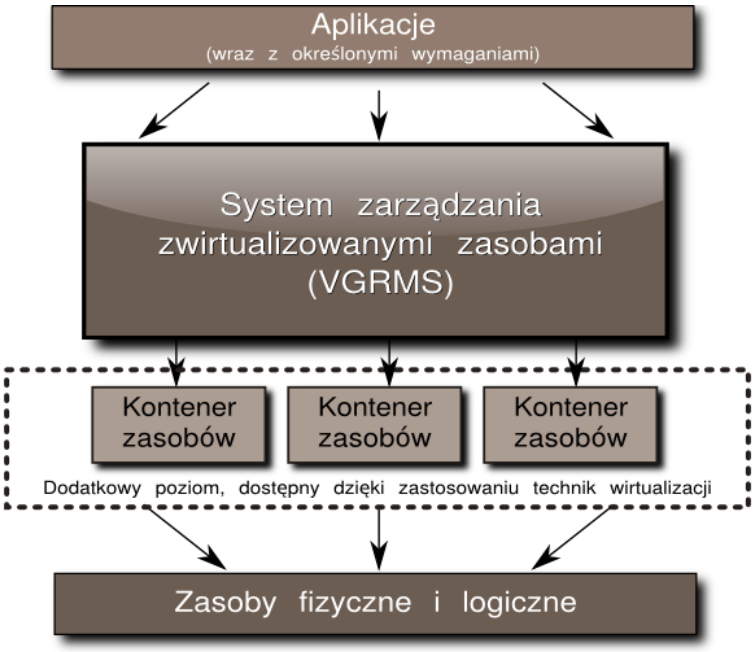
W rozdziale tym proponuje się model systemu, na podstawie którego zostanie stworzona jego architektura. Takie podejście do konstrukcji systemu (ang. *model-driven*) jest często używane przy tworzeniu systemów w środowiskach rozproszonych. Stworzenie szczegółowego modelu pozwoli dodatkowo na:

- teoretyczną weryfikację niektórych aspektów działania systemu zanim zostanie on stworzony,
- stworzenie podstaw umożliwiających łatwe i spójne przedstawienie szczegółowych aspektów działania środowiska,
- obliczenie, bądź oszacowanie niektórych parametrów pracy systemu, których uwzględnienie w implementacji pozwoli uzyskać większą efektywność działania oprogramowania zarządzania zasobami,
- opracowanie i wstępną weryfikację podstawowych algorytmów koniecznych do zapewnienia wymaganej funkcjonalności implementacji systemu.

Budowa modelu oraz zaproponowanie architektury systemu VRMS (ang. *Virtual Resources Management System*) pozwoli na zarządzanie alokacją zasobów wykonywaną w czasie działania aplikacji, dzięki czemu możliwa będzie implementacja adaptowalnego systemu zarządzania zasobami dla rozproszonych środowisk obliczeniowych.

sobów, pod pod pojęciem zasobu rozumiane są zarówno zasoby obliczeniowe, jak i komunikacyjne.

[2]Dynamiczne, czyli możliwe do modyfikacji, zmienne w czasie.



Rysunek 4.2. System zarządzania zwirtualizowanymi zasobami

4.1.1. Wykorzystywane pojęcia i określenia

Przedstawiając model systemu należy na wstępie sprecyzować podstawowe pojęcia, które będą używane do jego budowy. Określenie tych pojęć jest uzupełnieniem definicji zawartych w rozdziale 1 takich jak definicje zasobu (definicja 1.6), zarządzania zasobami (definicja 1.7) czy efektywności zarządzania (definicja 1.8) w odniesieniu do przyjętej reprezentacji zasobów w oparciu o koncepcję wirtualnego Gridu (definicja 1.5).
Do ważnych pojęć używanych w opisie modelu systemu zarządzania zasobami zalicza się:

Interfejs zarządzania jest to zestaw określonych interfejsów umożliwiających zarządcy interakcję z zarządzanym obiektem. Typowe akcje zarządzania to: uruchamianie i zatrzymywanie usług, zebranie informacji na temat stanu komponentu, itp.

Zarządzany obiekt jest to komponent udostępniający zasoby (definicja 1.6) oraz interfejs zarządzania, poprzez który może być zarządzany. W systemie zarządzania zasobami wyróżnia się następujące, możliwe typy zarządzanych obiektów:

- **obiekt fizyczny** (np. węzeł obliczeniowy, interfejs sieciowy, procesor) lub **obiekt logiczny** (np. system plików, wirtualna sieć, wirtualny węzeł),
- **obiekt pojedynczy** (np. węzeł, połączenie sieciowe, urządzenie sieciowe) lub **obiekt złożony** (np. klaster węzłów, topologia sieciowa),
- **obiekt ulotny** (np. zadanie obliczeniowe, węzeł wirtualny) lub **obiekt trwały** (np. fizyczny komputer).

Zarządca rozpoczyna/inicjuje akcje zarządzania, może to być osoba operująca na obiektach za pośrednictwem odpowiedniej konsoli administracyjnej (implementującej obsługę interfejsów zarządzania), bądź komponent programowy kontrolujący zarządzane obiekty w sposób automatyczny/autonomiczny.

Zarządzanie pośrednie rozumiane w tej pracy, jest to proces kontrolowania stanu obiektu poprzez wpływanie na jego otoczenie[3], jako odpowiedź na zmianę zewnętrznych lub wewnętrznych warunków działania danego obiektu.

Metody zarządzania są to ogólne różne sposoby wpływania na stan zasobu. Dla środowisk zarządzania związanych z systemami IT (ang. *Information Technology*) mogą to być operacje takie jak:

- rezerwacja, *brokering*, szeregowanie,
- instalacja, wdrożenie, *provisioning*,
- agregacja, grupowanie serwisów,
- monitorowanie wydajności, dostępności,
- kontrola, operacje start, stop, usunięcie.

Do pojęcia „metody zarządzania" zalicza się zadania związane z zarządzaniem ale nie mechanizmy, których zarządzanie wymaga do poprawnego działania, lub z których korzysta (np. lokalizacja serwisów).

Model zasobów jest to abstrakcyjna reprezentacja zestawu zarządzanych elementów, definiująca ich schemat (hierarchię oraz zależności pomiędzy obiektami) a także ich charakterystykę (atrybuty i możliwe operacje zarządzania).

Notacja używana przy opisie modelu

Model działania i interakcji komponentów systemu został przedstawiony w oparciu o notację, której określenia zostały przedstawione w tabeli 4.1.

Symbol	Znaczenie
$\mathbf{PH}$, $\mathbf{P}$	oznaczenie elementu będącego zbiorem
h_i, $l_{i,j}, param$	elementy zbiorów
r, t	funkcje operujące na elementach zbiorów
$Type$, $Name$	zmienna, typ wyliczany
$\ll const \gg$	wartość stała
$param_{k(1)}, \ldots, param_{k(vm)}$	k oznacza funkcję porządkującą postaci $k : \mathbf{N} \rightarrow \mathbf{N}$
$U_{VM}(\mathbf{A}, \mathbf{B})$	klasa funkcji przyjmujących za argument element z $\mathbf{A}$ i zwracających element z $\mathbf{B}$

Tabela 4.1. Notacja używana przy opisie modelu systemu

[3]Przykładowo sterowanie działaniem np. procesów, poprzez ograniczanie dostępu do zasobów środowiska, w których są one uruchomione. Metoda stosowana dla obiektów, których bezpośrednia kontrola z różnych przyczyn jest niemożliwa.

4.1.2. Model statyczny wirtualnego Gridu

Podstawowe elementy VG (ang. *Virtual Grid*) tworzące infrastrukturę wykonania aplikacji rozproszonych, możemy przedstawić za pomocą modelu mnogościowego określającego składowe poszczególnych komponentów a także interakcje pomiędzy nimi.

Przedstawiona w rozdziale 1 definicja wirtualnego Gridu zakłada powiązanie aplikacji z zasobami za pomocą dedykowanego środowiska. Środowisko to umożliwia zasłonięcie przed aplikacją szczegółów związanych z działaniem mechanizmów alokacji zasobów. Przedstawiając model działania systemu należało zatem wyjść od znalezienia i określenia możliwych interakcji komponentów realizujących grupowanie zasobów. Definicje te pozwolą na sformułowanie pierwszej statycznej wersji modelu wirtualnego Gridu, gdyż nie będą one uwzględniać możliwych zmian w czasie odnośnie wymagań aplikacji oraz modyfikacji konfiguracji zasobów.

Komponenty, których definicje oraz powiązania zostaną przestawione (za pomocą notacji mnogościowej) w tym modelu to:

- Elementy udostępniające zasoby obliczeniowe infrastruktury, czyli zbiór fizycznych węzłów — **PH** (4.1),
- Wirtualne kontenery zasobów takie jak wirtualne komputery — **VM** (4.5) i wirtualna sieć — **VN** (4.11),
- Statyczny Wirtualny Grid — VG_S (4.21) czyli element grupujący zasoby wraz z określeniem wymaganych reguł.

Grupowanie zasobów do kontenerów

Mając do dyspozycji pewną infrastrukturę obliczeniową możemy określić ją za pomocą zbioru **PH** niezależnych urządzeń h_i (komputerów) o określonych i niezmiennych w czasie właściwościach:

$$\mathbf{PH} := \{h_i : \ i = 1, \ldots, I\} \tag{4.1}$$

Właściwości te w niniejszym modelu będą reprezentowane za pomocą zestawu parametrów (4.2) określających zasoby.[4]

$$\begin{aligned} \mathbf{P} &:= \{param_j : j = 1, \ldots, J\} && (4.2)\\ \text{param} &:= (Name,\ Type,\ Value) && (4.3)\\ Type &:= \{\ll required \gg, \ll optional \gg, \\ &\quad \ll unspec \gg, \ll unknown \gg\} \end{aligned}$$

gdzie:

[4]Parametry te to np. właściwości sprzętowe takie jak typ czy zegar procesora lub ilość dostępnej pamięci oraz parametry związane z oprogramowaniem, czyli atrybuty opisujące konfigurację systemu operacyjnego i zainstalowanego oprogramowania.

$Name$ — symboliczna nazwa jednoznacznie identyfikującą dany parametr,
$Type$ — oznaczenie operacji określenia właściwości danej zmiennej (tabela 4.2),
$Value$ — ilościowe określenie danego parametru wraz z dopuszczalną wartością pustą (*Null*),
P — zbiór wszystkich możliwych parametrów.

Modyfikator	Znaczenie
$\ll optional \gg$	parametr jest opcjonalny
$\ll required \gg$	parametr jest wymagany
$\ll unspec \gg$	wartość parametru jest nieokreślona
$\ll unknown \gg$	wartość parametru jest nieznana

Tabela 4.2. Modyfikatory znaczenia parametrów konfiguracyjnych

Również definicja maszyny wirtualnej **VM** może być przedstawiona jako zestaw parametrów i odpowiadających im wartości[5] określających dostępność oraz sposób wykorzystania zasobów przez daną instancję maszyny wirtualnej — **VM**:

Nazwa parametru	Typ parametru
identification/uuid	statyczny
resident_on host	statyczny
kernel file	statyczny
memory/dynamic_max size	dynamiczny
memory/dynamic_min size	dynamiczny
virtual block device	statyczny
virtual network interface	statyczny
name/description	dynamiczny
VCPU/params	dynamiczny
VCPU/max number	dynamiczny
monitoring metrics	statyczny
VCPU/at_startup number	statyczny

Tabela 4.3. Przykładowa lista parametrów konfiguracyjnych VM

$$\mathbf{VM} := \left\{ param_{k(1)}, \ldots, param_{k(vm)} \right\} \tag{4.4}$$

$$\mathbf{VZ} := \{ \mathbf{VM} \in 2^{\mathbf{P}} : k = 1, \ldots, 2^J \} \tag{4.5}$$

Zbór **VZ** reprezentuje wszystkie możliwe zestawy parametrów opisujących maszyny wirtualne.

Mając dwa komponenty systemu w postaci zasobów fizycznych (4.1) oraz zestawu konfiguracji wirtualnych maszyn (4.5) jako kontenerów grupujących i określających dostępność

[5] Parametry te określają konfigurację alokacji zasobów dla danej wirtualnej maszyny a także jej aktualny stan.

zasobów, konieczne jest określenie funkcji umożliwiającej odwzorowanie elementów tych zbiorów — czyli procedury r dokonującej rozmieszczenia wykonania wirtualnych maszyn na fizycznym sprzęcie:

$$r : \mathbf{VZ} \rightarrow 2^{\mathbf{PH}}, \quad r(\mathbf{VM}) := \{h_i \in PH : \ Cap(h_i) \overset{(1)}{\equiv} Req(\mathbf{VM})\} \tag{4.6}$$

gdzie:

$r(\mathbf{VM})$ — funkcja tworząca zbiór maszyn fizycznych, dla których dostępność i konfiguracja zasobów spełnia wymagania maszyny wirtualnej $\mathbf{VM}$,
$Cap(h_i)$ — zasoby (4.7) maszyny fizycznej h_i,
$Req(\mathbf{VM})$ — wymagania (4.8) maszyny wirtualnej $\mathbf{VM}$,
$\overset{(1)}{\equiv}$ — oznacza relację (procedurę) weryfikacji dostępności zasobów fizycznego węzła h_i na poziomie przedstawionym w specyfikacji wirtualnej maszyny $\mathbf{VM}$.

$$Cap : \mathbf{PH} \rightarrow 2^{\mathbf{P}}, \quad Cap(h_i) := \left\{param_{p(1)}, \ldots, param_{p(h_i)}\right\} \tag{4.7}$$

gdzie:

$p(h_i)$ — liczba parametrów opisujących węzeł h_i.

$$Req : \mathbf{VZ} \rightarrow 2^{\mathbf{P}}, \quad Req(\mathbf{VM}) := \left\{param_{p(1)}, \ldots, param_{p(\mathbf{vm})}\right\} \tag{4.8}$$

Działanie funkcji r sprowadza się do wyboru z pośród dostępnych elementów infrastruktury tylko tych, które spełniają wymagania danej maszyny wirtualnej. Przez $\mathbf{PH}^{(i)}$ oznaczony zostanie zbiór fizycznych elementów infrastruktury spełniających wymagania maszyny wirtualnej $\mathbf{VM}$, czyli:

$$\mathbf{PH}^{(i)} := r(\mathbf{VM}) \tag{4.9}$$

Dodatkowo konieczne jest określenie funkcji s (4.10), która dla zbioru $\mathbf{PH}^{(i)}$ wybierze jeden fizyczny element infrastruktury h_i, na którym dana maszyna zostanie stworzona i uruchomiona.

$$s \ : \ \mathbf{PH}^{(i)} \rightarrow \mathbf{PH}^{(i)}, \quad s\left(\mathbf{PH}^{(i)}\right) := h_j \ : \ h_j \in \mathbf{PH}^{(i)} \tag{4.10}$$

Różne możliwe algorytmy rozmieszczenia maszyny wirtualnej są możliwe poprzez zmianę działania funkcji s.

Grupowanie zasobów wirtualnej sieci

Wirtualna sieć $\mathbf{VN}$ (4.11) czyli zestaw wirtualnych połączeń sieciowych definiujący topologię dla komunikacji w obrębie VG składa się z następujących elementów:

$$\mathbf{VN} := (\mathbf{D}, \mathbf{L}) \tag{4.11}$$

gdzie:

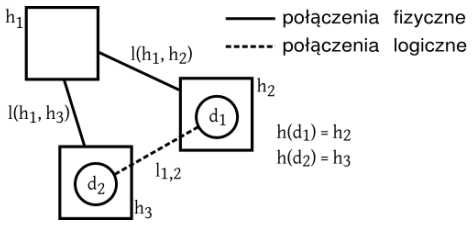
D — zbiór (wirtualnych) urządzeń sieciowych (elementów infrastruktury) oraz urządzeń końcowych (wirtualnych komputerów).
L — zbiór wirtualnych połączeń (4.14).

$$d := \left\{param_{d(1)}, \ldots, param_{d(d_i)}\right\} \tag{4.12}$$

$$\mathbf{D} := \{d \in 2^{\mathbf{P}} : m = 1, \ldots, 2^J\} \tag{4.13}$$

Element d oznacza wirtualne urządzanie sieciowe (wirtualny switch, wirtualny router), którego funkcjonalność i wymagania określone są poprzez zestaw parametrów (4.12).

$$\mathbf{L} \subseteq \{l_{i,j} = (d_i, d_j) : d_i, d_j \in \mathbf{D}\} \tag{4.14}$$



Rysunek 4.3. Przykład topologii fizycznej wirtualnego Gridu

Wirtualne połączenie posiada również określony dla niego zestaw parametrów charakteryzujący jego właściwości:

$$l_{i,j} := \left\{param_{l(1)}, \ldots, param_{l(l_{i,j})}\right\} \tag{4.15}$$

Nazwa parametru	**Typ parametru**
lista interfejsów fizycznych	statyczny
parametry interfejsów fizycznych	statyczny
lista wirtualnych interfejsów	dynamiczny
połączenia pomiędzy interfejsami	dynamiczny
rodzaj powiązania interfejsów	dynamiczny
metoda separacji komunikacji	dynamiczny
konfiguracja adresacji IP	dynamiczny
parametry QoS połączeń	dynamiczny
konfiguracje routigu i NAT	dynamiczny
rozmieszczenie wirtualnych routerów	dynamiczny
(...)	(...)

Tabela 4.4. Przykładowa lista parametrów konfiguracyjnych VN

Pełne określenie wirtualnej topologii wymaga dodatkowo zdefiniowania reguł rozmieszczenia wirtualnych komponentów w obrębie fizycznych zasobów. Wybór elementu infrastruktury, podobnie jak przy instancjonowaniu wirtualnych węzłów wymagać będzie okre-

ślenia funkcji n (4.16) weryfikującej dostępność zasobów danego węzła pod kątem wymagań wirtualnego komponentu topologii.

$$n : \mathbf{D} \to 2^{\mathbf{PH}}, \quad n(d_i) := \{h_j \in \mathbf{PH} : \ Cap(h_j) \overset{(2)}{\equiv} Req(d_i)\} \tag{4.16}$$

gdzie:

$n(d_i)$ — funkcja tworząca zbiór maszyn fizycznych, dla których dostępność i konfiguracja zasobów spełnia wymagania elementu d_i,
$Cap(h_j)$ — zasoby maszyny fizycznej h_j,
$Req(d_i)$ — wymagania elementu topologii sieciowej d_i.
$\overset{(2)}{\equiv}$ — oznacza relację weryfikacji dostępności zasobów fizycznego węzła h_j na poziomie przedstawionym w specyfikacji komponentu d_i.

Dodatkowo określić należy funkcję weryfikującą zasoby ścieżki fizycznej (łączącej dwa komputery fizyczne), pod kątem wymagań tworzonego nad nią połączenia logicznego. Weryfikacja ta wymagana jest w sytuacji gdy połączenie logiczne jest realizowane pomiędzy wirtualnymi węzłami uruchomionymi na dwóch różnych[6] fizycznych węzłach.

$$\begin{aligned} &w : \mathbf{D} \to 2^{\mathbf{PH}}, \\ &\quad w(d_i) := \left\{ h_i \in PH : \bigwedge_{\substack{d_j \in D : l_{i,j} \in l(d_i) \\ h_k = h(d_i) \\ h_l = h(d_j)}} Cap\Big(l(h_k, h_l)\Big) \overset{(3)}{\equiv} Req(l_{i,j}) \right\} \end{aligned} \tag{4.17}$$

gdzie:

$w(d_i)$ — funkcja tworząca zbiór maszyn fizycznych, które mogą obsługiwać element d_i powodując iż możliwe będzie spełnienie wymagań wszystkich wirtualnych połączeń z tego elementu,
$Cap\Big(l(h_k, h_l)\Big)$ — zasoby ścieżki fizycznej pomiędzy elementami h_k i h_l,
$Req(l_{i,j})$ — wymagania połączenia logicznego l,
$l(d_i)$ — zbiór połączeń logicznych (4.18) wykorzystywanych przez element d_i,
$\overset{(3)}{\equiv}$ — oznacza relację weryfikacji dostępności zasobów ścieżki $l(h_k, h_l)$ na poziomie przedstawionym w specyfikacji połączenia logicznego $l_{i,j}$.

$$\begin{aligned} l : \mathbf{D} \to 2^{\mathbf{L}}, \quad l(d_i) &:= \left\{ \bigcup_{d_j \in \mathbf{D}} l_{i,j} = (d_i, d_j) : l_{i,j} \in L \right\} \\ l(d_i) &\subseteq L \end{aligned} \tag{4.18}$$

[6] Przyjęto bowiem, iż zasoby komunikacji sieciowej w obrębie jednego fizycznego węzła są nieskończone — dlatego weryfikacja ich dostępności nie jest konieczna.

Przez $PH^{l(i)}$ oznaczony zostanie zbiór fizycznych elementów infrastruktury, spełniających wymagania wirtualnego elementu d_i, czyli:

$$\mathbf{PH}^{l(i)} := n(d_i) \cup w(d_i) \tag{4.19}$$

Dodatkowo konieczne jest określenie funkcji e (4.20), która dla zbioru $PH^{l(i)}$ wybierze jeden fizyczny komponent infrastruktury h_i, na którym dany element wirtualnej topologii zostanie stworzony i uruchomiony.

$$e : \mathbf{PH}^{l(i)} \rightarrow PH, \quad e\Big(\mathbf{PH}^{l(i)}\Big) := h_j \ : \ h_j \in \mathbf{PH}^{l(i)} \tag{4.20}$$

Statyczny Wirtualny Grid

Wirtualny Grid możemy przedstawić jako złożenie (4.21) zbioru maszyn fizycznych (4.1), zbioru maszyn wirtualnych (4.5), funkcji dokonujących rozmieszczenia wirtualnych komponentów (4.10) i elementów topologii sieciowej (4.20) określonej za pomocą definicji (4.11) wraz z zestawem funkcji wykonujących weryfikacji rozmieszczenia elementów sieciowych. Określenie takie będzie definiowało wirtualny Grid jako statyczną infrastrukturę, w której rozmieszczenie i konfiguracje elementów nie będzie zmieniane podczas działania aplikacji.[7]

$$VG_S \ := \ \Big\{\mathbf{PH},\ \mathbf{VZ},\ r,\ s,\ \mathbf{VN},\ n,\ w,\ e\Big\} \tag{4.21}$$

Powyższy zestaw definicji przestawia w sposób uproszczony zależności pomiędzy elementami składowymi wirtualnego Gridu. Zestaw ten pozwala na stworzenie architektury środowiska, której opis został przedstawiony w sekcji 5.2 a szczegóły implementacyjne zawiera rozdział 6.

4.1.3. Model dynamiczny wirtualnego Gridu

Przedstawiony w sekcji 4.1.2 model wirtualnego Gridu uwzględnia jedynie statyczną konfigurację komponentów. Operacje związane z tworzeniem wirtualnego Gridu (wraz z wirtualną topologią sieciową) nie uwzględniają adaptowalności infrastruktury w czasie wykonania aplikacji dla zmieniających się wymagań.

Dynamiczny wirtualny Grid, oznacza iż tworzące go komponenty i ich interakcje mogą zmieniać się w czasie. Zmienne w czasie mogą być zarówno zasoby obliczeniowe (**PH**), komunikacyjne, ich dostępność a także konfiguracje kontenerów zasobów takich jak maszyny wirtualne (**VM**) i wirtualna sieć (**VN**). Dynamiczny wirtualny Grid, wymaga również wprowadzenia pojęcia adaptowalności. Adaptowalność jest to zdolność do nienadzorowanej, autonomicznej modyfikacji konfiguracji elementów wchodzących w skład modelu statycznego. Wprowadzenie do modelu adaptowalności wymaga określenia dodatkowych cech. Cechy te wynikają z modelu podstawowego (statycznego) i określają:

[7] Rozszerzenie tego modelu o elementy umożliwiające realizacje adaptowalności dla zmieniających się zasobów oraz wymogów aplikacji, wraz z powiązaniem modelu z aplikacją użytkownika zawiera sekcja 4.1.3. W sekcji 7.1.3 zawarte są definicje metryk związanych z modelem, służących do jakościowej i ilościowej oceny jego implementacji.

- **akcje** — czyli zbiór możliwych do wykonania operacji związanych z dziedziną działania systemu. Dla systemu zarządzania zasobami z wykorzystaniem wirtualizacji będzie to zestaw operacji modyfikujących konfigurację alokacji zasobów dla aplikacji rozproszonych. Listę możliwych akcji dla modelu VG_S przedstawia tabela 4.6.
- **zdarzenia** — jest to lista możliwych do wystąpienia sytuacji, dla których konieczne będzie wykonanie określonej akcji. Dla modelu VG_S będą to zdarzenia związane z modyfikacją właściwości komponentów czy zmianą dostępności i poziomu zasobów. Listę zdarzeń określonych dla modelu VG_S przedstawia tabela 4.5.

Możliwe zdarzenia	**Występowanie**
zmiana stanu aplikacji	zdarzenie informujące o np. zakończeniu czy wstrzymaniu przetwarzania całej aplikacji bądź któregoś z jej komponentów,
zmiana wymogów dla aplikacji	modyfikacja określonych na początku działania aplikacji założeń odnośnie jakościowych i ilościowych własności zwracanych przez aplikację rezultatów,
zmiana wymagań aplikacji	zmiana zapotrzebowania na zasoby w trakcie działania aplikacji związana bądź ze zmianą wymogów, bądź ze zmianą stanu (fazy działania) samej aplikacji,
zmiana poziomu zasobów	obliczeniowych oraz komunikacji sieciowej
modyfikacja topologii fizycznej	topologia fizyczna sieci jest modyfikowana głównie pośrednio przez akcję związane z relokacją kontenerów zasobów
zmiana dostępności zasobów	czyli odnotowanie faktu dodania bądź usunięcia elementów infrastruktury dostarczających zasoby
zmiana definicji polityki określającej funkcjonowanie systemu	konfiguracja faz adaptacji, monitoringu i wykonania operacji zarządzania

Tabela 4.5. Zbiór możliwych zdarzeń określonych dla modelu VG_S

Możliwe akcje	**Działanie**
modyfikacja wirtualnego Gridu	akcja ta dopuszcza takie operacje dla VG jak stworzenie, usunięcie oraz wstrzymanie lub wznowienie wykonania
modyfikacja konfiguracji alokacji zasobów	czyli rekonfiguracja przydziału zasobów w obrębie fizycznego węzła lub z przeniesieniem (migracją) wykonania
modyfikacja reguł, algorytmów działania systemu	dopuszczalne są również rekonfiguracje parametrów pracy systemu takich jak definicje procedur adaptacji, monitorowania, metryk wydajnościowych itp.

Tabela 4.6. Zbiór możliwych akcji określonych dla modelu VG_S

Zaproponowane w sekcji 4.1.2 składowych VG jako (4.21) powoduje iż procedury adaptacji infrastruktury VG dotyczą tylko mechanizmów tworzenia i konfiguracji wirtualnych węzłów VM oraz topologii sieciowych VN (ang. *Virtual Network*) wraz z uwzględnieniem

możliwego wpływu na działanie całego środowiska. Instancja VG jest tworzona dynamicznie dla każdej aplikacji[8] dzięki temu możliwe będzie określenie procesów decyzyjnych, których działanie umożliwi ścisłe dopasowanie środowiska do wymogów aplikacji. Dopasowanie to spowoduje, iż możliwe będzie zwiększenie wydajności działania aplikacji lub umożliwienie działania wielu aplikacji równocześnie bez znacznej degradacji ich wydajności.

Definicje możliwych akcji

Zarządzanie zasobami w prezentowanym modelu może odbywać się za pomocą dwóch technik. Pierwszą z nich stanowią modyfikacje udostępnionych przez warstwę wirtualizacji parametrów definiujących właściwości zasobów udostępnianych dla aplikacji. Modyfikacje te będą operowały na zasobach dostępnych w ramach pojedynczego fizycznego węzła. Z drugiej strony, dzięki zastosowaniu możliwości przeniesienia wykonania maszyny wirtualnej, możliwa jest zmiana powiązania (określonego przy tworzeniu wirtualnego Gridu) pomiędzy maszyną fizyczną a wirtualną. Dzięki temu możemy bardziej efektywnie wpływać na wykorzystanie zasobów fizycznych, gdyż możliwe będzie dokładniejsze dopasowanie miejsca wykonania do wymagań aplikacji.

Akcje wykonywane przez system możemy podzielić zatem na trzy grupy:

- akcje wykonujące modyfikacje parametrów — czyli takie jak zmiana właściwości[9] wirtualnych zasobów,
- akcje związane z modyfikacją stanu obiektów — czyli wykonywanie operacji wpływających na stan obiektu takich jak stworzenie obiektu, przeniesienie (migracja) czy wstrzymanie wykonania,
- akcje związane z modyfikacją parametrów systemu — takich jak stosowane algorytmy, monitorowane parametry itp.

Przedstawione w tabeli 4.3 przykładowe parametry konfiguracyjne VM udostępniane przez oprogramowanie zarządzania wirtualizacją, zostały podzielone na dwie grupy. Parametry statyczne są to ustawienia, które mogą być modyfikowane tylko przed uruchomieniem lub po włączeniu VM. Zmienianie ich wartości w czasie wykonania jest niedozwolone lub nie ma wpływu na działanie wirtualnego węzła. Parametry te są używane na etapie tworzenia środowiska uruchomieniowego dla aplikacji. Parametry dynamiczne mogą być modyfikowane przez system zarządzania w dowolnym czasie bez względu na aktualny stan VM. Dostosowanie wartości tych parametrów pozwala na sterowanie konfiguracją zasobów przydzielonych w fazie tworzenia środowiska.

Infrastruktura sieciowa, czyli zasoby związane z komunikacją również mogą być modyfikowane. I analogicznie jak w przypadku VM niektóre parametry nie mogą być modyfikowane po stworzeniu i początkowej konfiguracji właściwości topologii. Przykładową listę parametrów VN zawiera tabela 4.4.

[8] Aby przyśpieszyć ten proces, wirtualny Grid może być również stworzony z wcześniej przygotowanych i prekonfigurowanych elementów.

[9] Modyfikowane parametry to dla wirtualnej sieci i wirtualnych węzłów odpowiednie parametry dynamiczne. Przykład tych parametrów zawierają tabele 4.4 i 4.3.

Operacje modyfikowania parametrów sprowadza się do określenia funkcjonalności, która dla zbioru parametrów opisujących dany obiekt zwróci ich nowy zestaw. Dopuszczalne są takie operacje jak modyfikacja wartości zmiennej, zmiana jej typu, usunięcie lub dodanie nowej. Operacje modyfikacji parametrów mogły być wykonywane dla obiektów z modelu podstawowego, które posiadają w swojej specyfikacji parametry dynamiczne. Dla modelu podstawowego modyfikowane mogą być zatem ustawienia:

- wirtualnej maszyny **VM** (4.4) czyli kontenera zasobów obliczeniowych,
- wirtualnej sieci VN (4.11) (poprzez zmianę parametrów wirtualnego urządzenia sieciowego d (4.12) oraz połączeń logicznych l (4.15)).

Definicje możliwych zdarzeń

W przypadku konstrukcji systemów zarządzania w oparciu o model zdarzeniowy zakłada się konieczność określenia:

- zbioru dopuszczalnych zdarzeń dla systemu — czyli sytuacji, w której nastąpiła zmiana istotnych parametrów pracy środowiska,
- funkcjonalności generatora zdarzeń — funkcji zwracającej listę zdarzeń występujących w danej chwili t.

Przez $EV^{(VG_S)}$ (4.22) zostanie oznaczony zbiór wszystkich możliwych zdarzeń związanych z zarządzaniem zasobami.[10]

$$EV^{(VG_S)} := \{event_n : n = 1, \dots E\} \tag{4.22}$$

Dynamiczny model wirtualnego Gridu

Model dynamiczny powstał przez dodatnie do modelu podstawowego cech przedstawiających dynamiczne interakcje komponentów. Zdefiniowanie zbioru możliwych akcji oraz zdarzeń, wynikających z modelu statycznego pozwoli na przedstawienie systemu zarządzania zasobami.

Model dynamiczny wirtualnego Gridu jest zatem złożeniem:

- definicji modelu (statycznego) podstawowego (4.21),
- zbioru możliwych akcji (modyfikacji) dla komponentów modelu podstawowego $ACT^{(VG_S)}$,
- zbioru możliwych zdarzeń, których występowanie wynika z dopuszczalnych zmian w parametrach opisujących model podstawowy (4.22),

Dynamiczny wirtualny Grid będzie zatem określony następująco:

$$VG_D := \begin{cases} VG_S & \text{dla } t = t_0 \\ \left\{VG_S(t),\ ACT^{(VG_S)},\ EV^{(VG_S)}\right\} & \text{dla } t > t_0 \end{cases} \tag{4.23}$$

Definicja ta zostanie użyta przy opisie modelu systemu zarządzania zasobami z wykorzystaniem koncepcji wirtualnego Gridu przedstawionego w sekcji 4.1.4.

[10] Listę przykładowych zdarzeń dla modelu podstawowego zawiera tabela 4.5.

4.1.4. Model systemu zarządzania zasobami

Przedstawienie modelu systemu polega na określeniu dla zaproponowanego w sekcji 4.1.3 modelu dynamicznego wirtualnego Gridu odpowiedniej funkcjonalności umożliwiającej tworzenie i adaptację środowiska do przedstawionych wraz z aplikacją wymogów. Adaptacja konfiguracji VG do wymagań aplikacji tak aby możliwe było osiągnięcie przez nią maksymalnej wydajności dzięki zapewnieniu optymalnej konfiguracji środowiska jest możliwa przy dwóch fazach działania systemu.

Faza pierwsza dotyczy adaptacji na etapie tworzenia VG. W fazie tej następuje wyszukanie i przydzielenie zasobów na podstawie dostarczonej wraz z aplikacją specyfikacji. Specyfikacja ta jest podstawą dla początkowej konfiguracji zasobów, oraz podlega ograniczeniom związanym z dostępnością zasobów oraz możliwościami infrastruktury. W fazie tej po stworzeniu VG następuje uruchomienie aplikacji.

Drugim elementem, w którym adaptacja jest konieczna, jest faza wykonania. Na tym etapie system modyfikuje dynamiczne parametry środowiska dostosowując do zmieniających się w czasie wymagań aplikacji. Wymagania te nie są określone na etapie uruchamiania aplikacji. System poprzez analizę informacji dostępnych z podsystemu monitorowania określa wymogi aplikacji. Podobnie jak w fazie wdrożenia infrastruktury modyfikowane są parametry wirtualnych węzłów i wirtualnych sieci. Dużą rolę w zapewnieniu odpowiedniego poziomu dostępności zasobów, dzięki zastosowaniu wirtualizacji odgrywa migracja VM. Technika ta pozwala na przeniesienie bez zatrzymywania aktualnego stanu VM pomiędzy dwoma fizycznymi węzłami.

W obu fazach działania infrastruktury do podejmowania działań zaangażowane są informacje o aktualnym stanie aplikacji i środowiska (dostępne z monitoringu) oraz informacje o wymaganiach aplikacji dostarczone przez użytkownika. Wymagania aplikacji mogą być modyfikowane na podstawie predykcji stanu i zapotrzebowania na zasoby w kolejnych etapach wykonania.

Aplikację $\mathbf{A}$, na której operować będzie tworzony model systemu zarządzania, możemy przedstawić jako zbiór uruchomionych niezależnie komponentów a_i:

$$\mathbf{A} := \{a_i : i = 1, \ldots, I\} \tag{4.24}$$

Dostarczona wraz z aplikacją specyfikacja powstała przez połączenie definicji wymogów określonych dla poszczególnych komponentów aplikacji:

$$\mathbf{P}^{(\mathbf{A})} : \mathbf{A} \to 2^{\mathbf{P}}, \quad \mathbf{P}^{(\mathbf{A})}(t) := \left\{ \bigcup_{a_i \in \mathbf{A}} Req\,(a_i, t) \right\} \tag{4.25}$$

gdzie:

$Req\,(a_i, t)$ — specyfikacja wymogów komponentu a_i dla aplikacji $\mathbf{A}$ w czasie t.

$$Z_A := \left(\mathbf{A},\ \mathbf{P}^{(\mathbf{A})} \right) \tag{4.26}$$

Zadanie Z_A przedstawione jako wejście dla infrastruktury tworzenia wirtualnego Gridu to aplikacja (4.24) wraz z określoną dla niej specyfikacją wymagań (4.25). Dążenie do spełnienie w/w wymagań w trakcie działania aplikacji będzie umożliwiać osiągnięcie przez nią oczekiwanego poziomu wykonania.

Określenie adaptowalności w fazie tworzenia VG

Adaptacja fazy tworzenia to kreacja wirtualnego Gridu na podstawie specyfikacji dostarczonej wraz z aplikacją. Adaptowalność ta została oznaczona przez $adaptability^{(VG_S)}$, gdyż jest ona wykonywana w fazie tworzenia VG czyli określa funkcjonalność z modelu podstawowego VG_S (4.21).
Określenie adaptowalności dla modelu VG_S, składa się na następujące operacje:

1. Określenie zbioru maszyn fizycznych i ich topologii sieciowej — czyli wyznaczenie poziomu oraz dostępności zasobów.

2. Określenie specyfikacji i uruchomienie wirtualnych maszyn (zasoby obliczeniowe) na podstawie wymogów komponentów a_i dla aplikacji **A**.

3. Stworzenie elementów topologii sieciowej (urządzeń sieciowych oraz połączeń logicznych) na podstawie specyfikacji ścieżek komunikacyjnych komponentów dostarczonej wraz z aplikacją.

4. Zdefiniowanie postaci funkcji s (4.10) oraz e (4.10) dokonujących początkowego rozmieszczenia maszyn logicznych[11] w obrębie infrastruktury fizycznej.

Tworzenie instancji VG rozpoczyna się od zidentyfikowania zbioru maszyn fizycznych **PH** poprzez zastosowanie usługi wykrywania i lokalizacji zasobów (funkcja dis).

Następnie określamy funkcję g_M, która dla komponentów aplikacji a_i określa specyfikację wirtualnych maszyn **VM**.

Tworzenie wirtualnych węzłów VM na żądanie, w celu określenia konfiguracji zasobów dla aplikacji musi być uzupełnione o wsparcie do modelowania infrastruktury komunikacyjnej. Stosując techniki wirtualizacji posiadany możliwość zarówno dynamicznego instancjonowania elementów infrastruktury obliczeniowej jak i kontroli kształtu oraz konfiguracji topologii sieciowej, która będzie wykorzystywana przez uprzednio dodane węzły.

Funkcja g_N odpowiedzialna jest za określenie konfiguracji połączeń sieciowych — wirtualnej topologii sieciowej, na podstawie specyfikacji aplikacji.

Warunkiem koniecznym, którego spełnienie określa iż możliwe jest stworzenie instancji VG jest aby, dla każdego komponentu aplikacji a_i istniało niepuste przecięcie zbiorów fizycznych węzłów spełniających wymagania wirtualnych komponentów środowiska stworzonych na podstawie specyfikacji elementu a_i.

Adaptowalność dla modelu VG_S będzie zatem złożeniem następujących elementów:

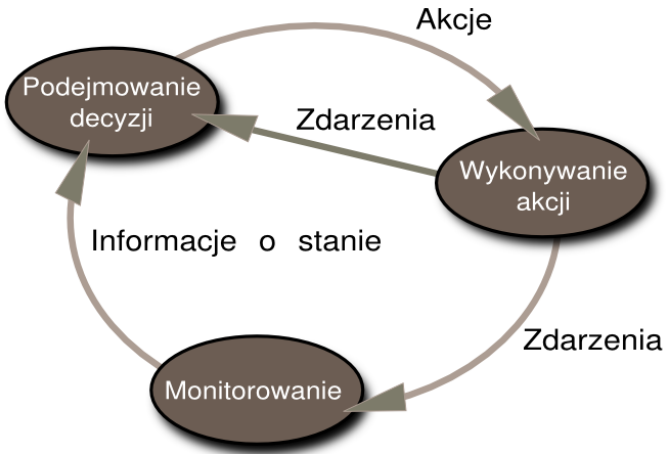
$$adaptability^{(VG_S)} := \{dis,\ g_M,\ g_N\} \tag{4.27}$$

Adaptowalność w czasie wykonania aplikacji

Wprowadzenie definicji adaptowalności systemu w czasie wykonania aplikacji wiąże się z koniecznością określenia tzw. pętli sterowania. Pętla ta będzie określać kolejność oraz zależności pomiędzy operacjami wykonywanymi przez system. Operacje te mają za zadanie dostosowanie warunków środowiska do aktualnych wymogów aplikacji (zmiennych w czasie) z użyciem odpowiednio zdefiniowanych reguł.

[11] Określenie tych funkcji wymaga definicja modelu podstawowego VG_S (4.21).

W opisie modelu dynamicznego VG zostały przedstawione definicje określające zbiory zdarzeń oraz akcji przetwarzanych przez system sterowania zasobami z użyciem tego modelu. Możliwe akcje oraz zdarzenia wynikały z przyjętego modelu podstawowego i związane były z wymaganiami[12] założonymi dla systemów tej klasy. Opis działania systemu zarządzania zasobami zostanie wprowadzony poprzez przedstawienie właściwości komponentów, których działanie operować będzie na modelu zdarzeniowym.



Rysunek 4.4. Pętla zarządzania dla modelu systemu

Podstawowe operacje (rysunek 4.4), które mogą być wykonywane w obrębie systemu i służą do sterowania właściwościami środowiska wykonawczego aplikacji, to:

1. Monitorowanie aplikacji i infrastruktury — jest to śledzenie zmian kluczowych parametrów pracy systemu, oraz generowanie zdarzeń przenoszących informację o wykrytych zmianach.

2. Podejmowanie decyzji optymalizujących konfigurację zasobów — w kroku tym na podstawie zebranych danych oraz określonych wcześniej wymogów, a także przewidywań czy oszacowań dotyczących stanu aplikacji (i jej komponentów) oraz środowiska, są podejmowane decyzje, których rezultatem jest lista koniecznych do wykonania akcji.

3. Wykonywanie akcji — przeprowadzanie określonych w poprzednim kroku operacji.

Argumentami wejściowymi dla systemu jest polityka — czyli zbiór reguł i wymogów określających działanie systemu w podstawowych jego aspektach, zasoby infrastruktury fizycznej oraz aplikacja wraz z definicją wymagań. Dodatkowo wykorzystywane zasoby mogą podlegać pewnym ograniczeniom, które wynikać będą z faktu dostosowania reguł działania środowiska do sytuacji, w której infrastruktura fizyczna jest współdzielona.

Wprowadzenie akcji wynikających i związanych ze zmianą stanu komponentów systemu pozwala na uzyskanie właściwości adaptowalności środowiska wykonawczego aplikacji ($monitor^{(VG_D)}$).

[12] Wymogi te zostały szczegółowo przedstawione w sekcji 3.3.

Moduł decyzyjny dla modelu VG_D (4.23) jest funkcją $optimize^{(VG_D)}$, która dla każdego elementu zbioru zachodzących aktualnie zdarzeń *Events* (4.22) zwraca podzbiór koniecznych do wykonania akcji *Actions*.

Faza wykonania ($execute^{(VG_D)}$) pobiera zbiór operacji, których wykonanie ma na celu zmianę konfiguracji zasobów. Wykonywanie akcji wiąże się również ze zwróceniem nowych zdarzeń, dla których wymagane będzie obsłużenie przez moduł decyzyjny .

Złożenie tych funkcjonalności określa adaptowalność dokonywaną w trakcie wykonania aplikacji dla modelu dynamicznego VG.

$$adaptability^{(VG_D)} := \left\{ monitor^{(VG_D)},\ optimize^{(VG_D)},\ execute^{(VG_D)} \right\} \tag{4.28}$$

System zarządzania zasobami

System zarządzania VGRMS (ang. *Virtual Grid Resources Management System*) przedstawiony zostanie jako złożenie następujących funkcjonalności i definicji:

- definicji dla modelu dynamicznego wirtualnego Gridu (4.23),
- określonej dla modelu podstawowego VG_S procedury adaptowalności fazy tworzenia środowiska ($adaptability^{(VG_S)}$),
- definicji polityki funkcjonowania środowiska,
- aplikacji wraz z określeniem wymagań (4.26),
- procedur wchodzących w skład definicji adaptowalności fazy wykonania aplikacji (4.28).

Złączenie powyższych elementów pozwoli na przedstawienie adaptowalności wykorzystania zasobów wirtualnego Gridu w czasie wykonania aplikacji **A**. Definicja systemu będzie więc miała postać:

$$VGRMS := \left\{ Z_A,\ VG_D,\ adaptability^{(VG_D)},\ Policy \right\} \tag{4.29}$$

Powyższa definicja oraz przedstawione wcześniej zależności posłuży do zaproponowania i przedstawienia modelu systemu zarządzania zasobami z wykorzystaniem koncepcji wirtualnego Gridu.

4.2. Podsumowanie

Przedstawiony w niniejszym rozdziale model, jest próbą formalnego opisu zależności jakie występują w środowisku, którego celem jest wykorzystanie techniki dynamicznego (zmiennego w czasie) grupowania zasobów do tworzenia przestrzeni wykonawczej aplikacji. Model systemu dopuszcza modyfikacje właściwości przestrzeni wykonawczej tak aby (możliwie najlepiej) odpowiadała ona wymogom aplikacji.

Przedstawienie definicja systemu zarządzania VGRMS autor rozpoczął od określenia modelu środowiska wykonawczego aplikacji czyli wirtualnego Gridu. Następnie w odniesieniu do statycznego podejścia, określone zostały możliwe modyfikacje, zmiany wymagań

i inne operacje zachodzące w czasie, których uwzględnienie wymagane było w modelu dynamicznym VG. System VGRMS jest to, upraszczając, definicja sposobu realizacji sterowania, którego celem jest realizacja założonej w rozdziale 3 adaptacji parametrycznej wirtualnego Gridu do wymagań aplikacji.

Przedstawiona w kolejnym rozdziale architektura komponentów systemu zarządzania zasobami jest odwzorowaniem opisanych w niniejszym rozdziale zależności i bezpośrednim ich następstwem.

Rozdział 5

Architektura systemu VGRMS

Rozdział ten zawiera opis architektury systemu zarządzania zasobami w środowiskach Grid. Proponowana architektura jest koncepcją budowy środowiska, powstałą w oparciu o model zaprezentowany w rozdziale 4.

Przedstawiając system, autor postanowił rozdzielić jego opis na dwie części. Pierwsza z nich dotyczy identyfikacji i przedstawienia podstawowych komponentów środowiska oraz ich wzajemnych interakcji. Część druga (zawarta w rozdziale 6) zawiera szczegóły implementacyjne oraz opis technologii i narzędzi wykorzystanych na tym etapie. W tym kontekście opis ten jest zgodny z podejściem PIM (ang. *Platform Independant Model*)[1], czyli pozwala na przedstawienie funkcjonalności komponentów środowiska, bez odniesienia do konkretnej technologii.

Opis architektury systemu został poprzedzony informacjami przedstawiającymi podstawowe założenia i koncepcje zastosowane w trakcie prac projektowych. Założenia te umożliwiły konstrukcję systemu zgodnego z określonymi w sekcji 3.3 wymogami oraz takiego, którego implementacja pozwoli na praktyczną weryfikację, postawionej na wstępie tezy pracy.

5.1. Podstawowe założenia projektowe

Zgodnie z zaproponowanymi w dokumencie [16] oraz organizację *Grid Forum* [33] koncepcjami modelowania architektur systemów zarządzania dla środowisk Grid, przyjmuje się odpowiedni podział funkcjonalny głównych komponentów środowiska. Podział ten zakłada rozdzielenie komponentów pasywnych (zarządzanych) od komponentów aktywnych (zarządzających). Jednak w systemach hierarchicznych występują sytuacje, w których relacja zarządzania jest relacją względną. Zdarza się tak dla tych elementów środowiska, które zarówno wpływają na stan elementów pasywnych jak i same podlegają zarządzaniu przez elementy działające na wyższym poziomie hierarchii.

Zależności pomiędzy tak określonymi klasami komponentów oraz możliwe operacje, przedstawia rysunek 5.1.

5.1.1. Komponenty pasywne systemu

Elementy pasywne zostały zdefiniowane na etapie tworzenia modelu środowiska zarządzania. W opisie architektury systemu, model systemu zarządzania zwirtualizowanymi zasobami przedstawiony w rozdziale 4 stanowi uzupełnienie tych informacji gdyż precy-

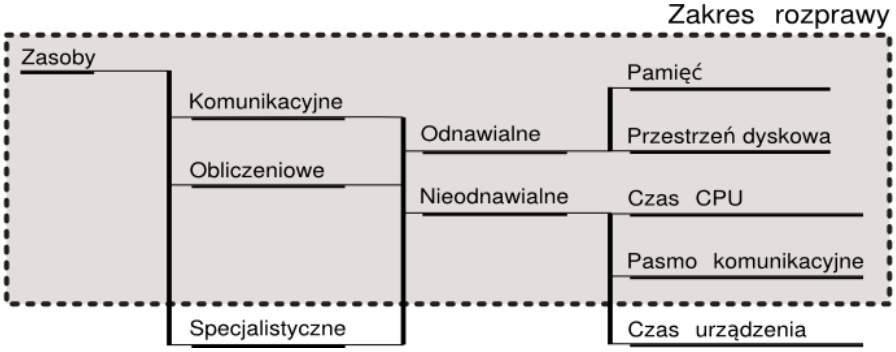
[1] Podejście to zostało przedstawione w rozdziale 3.

Rysunek 5.1. Możliwe powiązania komponentów aktywnych i pasywnych w systemach zarządzania zasobami

zuje właściwości tych komponentów (możliwe operacje, sposób określania, wykorzystywane przez system atrybuty, itp.).

W opisywanym systemie zarządzania zasobami, wyróżnione zostały zatem następujące komponenty pasywne:

- **Zasoby** — są to elementy środowiska używane przez aplikacje (konsumentów zasobów) do realizacji określonych funkcjonalności i usług.[2] Zasoby mogą być odnawialne lub nieodnawialne oraz mogą być dedykowane lub współdzielone. Przyjętą w niniejszej pracy ogólną taksonomię zasobu, przedstawia rysunek 5.2.



Rysunek 5.2. Taksonomia zasobów w środowiskach przetwarzania rozproszonego

- **Aplikacje** — to zbiór podstawowych zadań przetwarzania oraz zbiór powiązań między nimi. Powiązania określają zależności poszczególnych komponentów aplikacji. Zbiór powiązań komponentów aplikacji może być pusty.[3]
- **Zadania** — czyli podstawowe elementy wykorzystujące zasoby, zadania mogą być zarówno obliczeniowe jak i komunikacyjne. Zadania posiadają wymogi odnośnie zasobów koniecznych dla ich poprawnego wykonania.
- **Środowisko wykonawcze aplikacji** — czyli zbiór zasobów oraz zbiór usług, z którego korzysta aplikacja podczas wykonania. Środowisko to wiąże z aplikacją określone

[2] W modelu systemu zarządzania (rozdział 4) zasób reprezentowany jest przez definicje: zasobów obliczeniowych **PH** (4.1) oraz zasobów komunikacyjnych **VN** (4.11).

[3] Definicja aplikacji **A** i jej składowych została przedstawiona w modelu jako (4.24).

zasoby systemu. Jest to komponent pasywny systemu reprezentujący dynamiczne powiązania pomiędzy zadaniami a zasobami przez nie aktualnie wykorzystywanymi.

5.1.2. Komponenty aktywne systemu

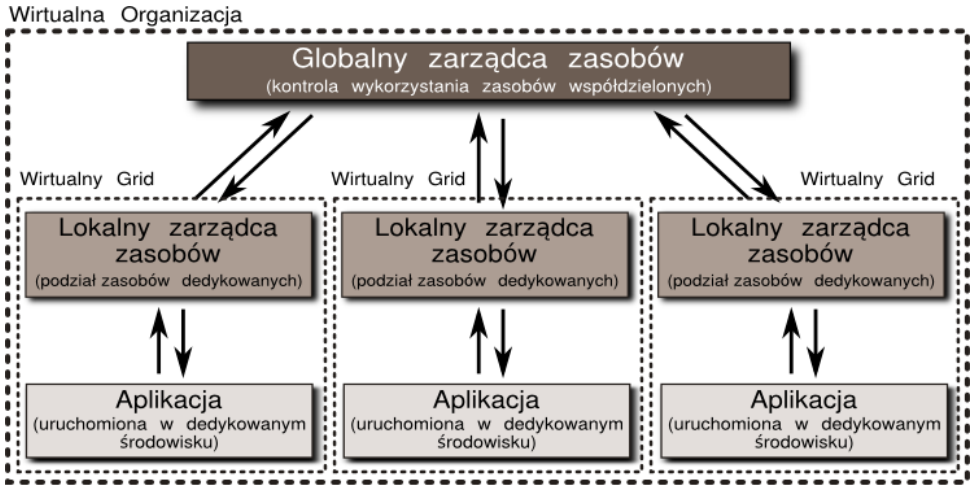
Architektura systemu zarządzania zasobami, prezentowanego w tym rozdziale, została skonstruowana w oparciu o aktualne trendy w technikach budowy systemów rozproszonych. Jednym z ważniejszych założeń jest dekompozycja funkcjonalności środowiska na mniejsze komponenty udostępniające funkcjonalność pojedynczej usługi. Do klasy komponentów aktywnych wyróżnionych w przedstawianej architekturze systemu zaliczają się:

- **Moduł wyszukiwania i lokalizacji** — umożliwia odnajdowanie i lokalizację zasobów, które mogą być wykorzystywane przez system. Zasięg operacji wyszukiwania tworzy tzw. obszar wyszukiwania.
- **Moduł tworzenia przestrzeni uruchomieniowej** — odpowiedzialny za początkowy przydział zasobów koniecznych do uruchomienia aplikacji. Jego działanie polega na wykonywaniu początkowej weryfikacji dostępności zasobów na poziomie wymaganym do prawidłowego uruchomienia aplikacji. Moduł ten tworzy wirtualny Grid i alokuje dla niego zasoby.
- **Monitory zasobów i aplikacji** — monitorowanie wykonania aplikacji, jej aktualnych wymagań oraz zbieranie informacji na temat poziomu wykorzystania używanych przez nie zasobów. Dane te pozwalają przewidywać zapotrzebowanie na zasoby oraz na podejmowanie decyzji o ewentualnych zamianach ich rozdziału.
- **Moduł zarządzania zasobami aplikacji** — odpowiedzialny za realizację strategii działania dla środowiska wykonawczego dla dostarczonej przez użytkownika aplikacji z określonymi dla niej wymogami. Moduł ten funkcjonuje i rozdziela zasoby zgodnie z określonymi przez administratora regułami. Zawiera implementację logiki działania opracowanej na postawie wymogów aplikacji, podejmuje i wykonuje decyzje dotyczące powiązań zasobów do wirtualnego Gridu w ramach danej lokalnej przestrzeni wyszukiwania. Działanie tego modułu określa lokalną domenę zarządzania zasobami.
- **Agent nadzorczy VO** — regulacja dostępu do zasobów wykonywana na poziomie wirtualnej organizacji, reguluje wykorzystanie zasobów pomiędzy środowiskami uruchomieniowymi aplikacji w ramach współdzielonych zasobów danej wirtualnej organizacji.
- **Monitory środowiska** — czyli sensory zbierające informacje na temat stanu i poprawności działania poszczególnych urządzeń systemu. Weryfikowane są takie parametry jak temperatura, działanie wewnętrznych podzespołów, liczniki błędów sygnalizujących możliwe awarie, itp.
- **Serwisy informacyjne**[4] — dostarczające usługi składowania i udostępniania danych zbieranych i wykorzystywanych w trakcie działania systemu przez pozostałe komponenty.

[4]Takie jak np. bazy danych, serwery LDAP (ang. *Lightweight Directory Access Protocol*) [105] i inne.

5.1.3. Hierarchiczna koncepcja zarządzania

Najczęściej [53] wykorzystywaną koncepcją budowy systemów zarządzania zasobami w środowiskach Grid jest podejście hierarchiczne. Wynika to głównie z konieczności wsparcia do zarządzania w sytuacji silnego rozproszenia i heterogeniczności zasobów. Koncepcja hierarchiczna zakłada podział domeny działania systemu RMS na obszary. Podział ten może wynikać np. z geograficznego rozmieszczenia zarządzanych zasobów, ich typów, przynależności do domeny administracyjnej, właściwości komunikacji sieciowej, itp. Do obsługi danego obszaru zostaje stworzony odpowiedni lokalny zarządca. Zarządca ten nie jest autonomiczny, gdyż jego działanie podlega kontroli na poziomie globalnego zarządcy zasobów (obsługa globalnej domeny zasobów).



Rysunek 5.3. Hierarchiczna koncepcja zarządzania zasobami w systemie VGRMS

Podejście hierarchiczne (rysunek 5.3) do zarządzania powala na uniknięcie wielu problemów związanych ze skalowalnością oraz zmniejszenie ogólnej złożoności systemu zarządzania. Koncepcja ta, ze względu na konieczność spełnienia przedstawionych w sekcji 3.3.3 założeń odnośnie skalowalności i efektywności działania, zastosowana została również w niniejszej pracy. Jednak podział na obszary zarządzania i wydzielenie poziomów zarządzania wiąże się z koniecznością zapewnienia możliwości tworzenia środowiska uruchomieniowego (wirtualnego Gridu) dostosowanego do wymogów aplikacji. Zwiększenie wydajności działania aplikacji rozproszonych jest głównym celem działania proponowanego rozwiązania. Dlatego, to wymogi odnośnie zasobów dostarczone wraz z aplikacją, stanowią podstawę do określenia zarówno lokalnej domeny zarządzania[5], reguł określających rozdział zasobów oraz listy dozwolonych operacji modyfikacji podziału zasobów dla aplikacji.

Zasoby zarządzane na poziomie lokalnym mogą być współdzielone, to znaczy że w systemie może istnieć inny lokalny zarządca mający do nich dostęp. Współdzielenie to, dla aplikacji o zróżnicowanych wymaganiach będzie powodować zwiększenie współczynników charakteryzujących poziom wykorzystania infrastruktury. Jednak w sytuacji, gdy równocześnie działające aplikacje mają przeciwstawne, kolidujące ze sobą wymagania konieczne

[5]Poprzez wybranie z puli dostępnych tylko tych zasobów, które konieczne będą dla optymalnego wykonania danej aplikacji.

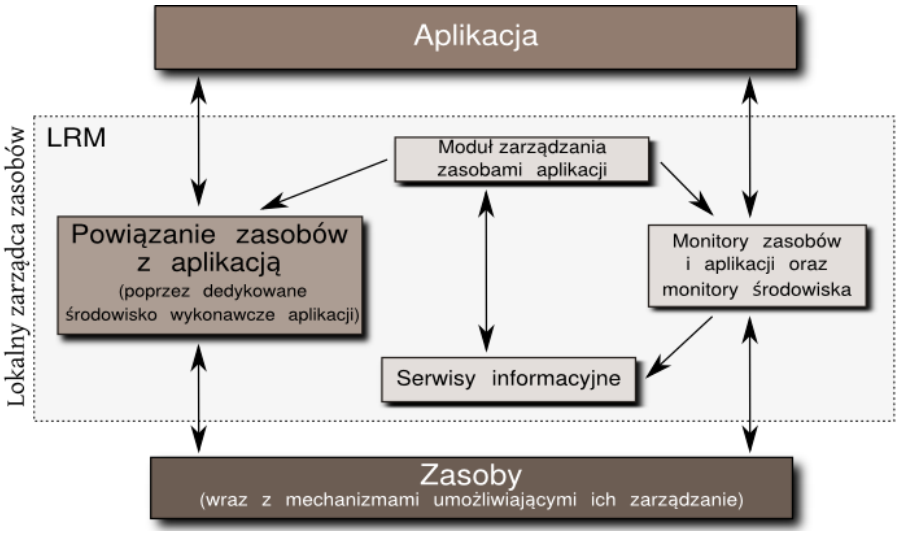
będzie wymuszenie przez system przestrzegania określonych ograniczeń w dostępie do zasobów[6]. Za kontrolę wykorzystania zasobów na poziomie danej wirtualnej organizacji, odpowiedzialny jest tzw. globalny zarządca zasobów.
Podsumowanie poziomów zarządzania i ich reguł zawiera tabela 5.1.

Poziom zarządzania	Reguły zarządzania	Zasoby
lokalna domena zasobów (wirtualny Grid)	tworzone na podstawie wymagań aplikacji (określonych przez użytkownika lub oszacowanych przez system)	współdzielone
globalna domena zasobów (wirtualna organizacja)	określone przez politykę funkcjonowania danej VO (administratora systemu)	dedykowane

Tabela 5.1. Hierarchia poziomów zarządzania systemu VGRMS

Lokalny zarządca zasobów

Proces tworzenia i zarządzania zasobami wirtualnego Gridu wymaga do poprawnego działania funkcjonalności komponentu LRM (ang. *Local Resource Manager*). LRM jest to system decyzyjny realizujący politykę zarządzania zasobami VG. Komponenty składowe oraz ich powiązania w obrębie lokalnego zarządcy zasobów przedstawia rysunek 5.4.

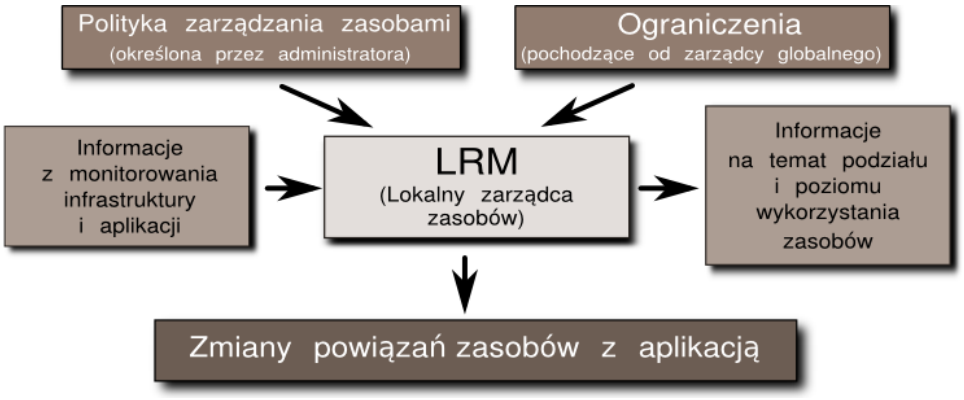


Rysunek 5.4. Komponenty składowe lokalnego zarządcy zasobów

Możliwe interakcje LRM z pozostałymi komponentami systemu przedstawia rysunek 5.5. Wejściem dla tego systemu jest „Polityka zarządzania zasobami" czyli specyfikacja

[6]Reguły współdzielenia są określone na podstawie polityki funkcjonowania VO — mogą więc zakładać dowolny podział, nie zawsze będzie to podział sprawiedliwy.

reguł wykorzystania i współdzielenia zasobów określona przez użytkownika infrastruktury. Stworzony przez LRM wirtualny Grid rezerwuje dla aplikacji fizyczne zasoby. W trakcie wykonania aplikacji LRM monitoruje infrastrukturę VG maksymalizując określoną wcześniej funkcję celu. Jak wspomniano przy opisie modelu wirtualnego Gridu, modyfikacje VG dotyczą parametrów dynamicznych opisujących VM oraz wirtualną sieć. Modyfikacja konfiguracji VG jest wykonywana przez cały czas działania aplikacji[7], a po jej zakończeniu wykorzystywane przez dany VG zasoby są zwalnianie.



Rysunek 5.5. Interakcje komponentu LRM

Działanie LRM może być określone poprzez specyfikację takich właściwości jak:

- lista parametrów VM, które będą modyfikowane (lista ta zawiera parametry dynamiczne VM) przykładowo zakładamy, że modyfikowane mogą być tylko ustawienia ilości i konfiguracji procesorów,
- lista parametrów VN, które będą modyfikowane (lista ta jest podzbiorem parametrów dynamicznych VN) np. zmieniane będą jedynie ustawienia ograniczenia szerokości pasma dla komunikacji węzłów,
- określenie kiedy i które parametry mogą być modyfikowane np. zezwolenie na zmianę parametru X w czasie t,
- definicję funkcji celu, której maksymalizacja będzie powodowała zwiększenie wydajności działania aplikacji,
- określenie algorytmu adaptacji zastosowanej techniki,
- zastosowanie informacji na temat śledzenia działania aplikacji przy podejmowaniu decyzji,
- wykorzystanie dodatkowych możliwości jak np. migracja VM, rekonfiguracja kształtu topologii sieciowej, itp.,

[7]Rekonfiguracja jest dokonywana poprzez modyfikowanie przedstawionych w tabelach 4.3 i 4.4 parametrów dynamicznych VM i VN.

Dodatkowo, biorąc pod uwagę rodzaje akcji wykonywane przez LRM można określić przykładowe klasy działania LRM:

- zarządca z migracją — przykładowe określenie funkcji celu na maksymalizację wykorzystania CPU fizycznych węzłów, system decyzyjny będzie wykonywał akcje przenoszenia VM wewnątrz infrastruktury VG tak aby średnie obciążenie wszystkich procesorów infrastruktury było równe,
- zarządca bez migracji,
- zarządca z modyfikacją topologii,
- zarządca bez modyfikacji topologii, z modyfikacją parametrów QoS komunikacji.

Działanie globalnego zarządcy zasobów w środowisku współdzielonym

Po stworzeniu VG, wykorzystującego elementy współdzielonej infrastruktury, jego aktywność musi być kontrolowana. Dążąc do stworzenia optymalnego środowiska wykonania dla aplikacji, LRM stara się zaspakajać wymagania przydziału zasobów. Dlatego w swoim działaniu LRM opiera się na pewnych ograniczeniach. Ograniczenia te wynikają z dwóch źródeł. Pierwsze zależne jest od maksymalnych możliwości zasobów fizycznych (parametry fizycznych węzłów oraz konfiguracja i wydajność sieci). Drugie ograniczenie wynika z faktu, iż dana infrastruktura może być wykorzystywana do obsługi wielu wirtualnych Gridów o kolidujących ze sobą wymaganiach. Dlatego system zarządzania wyróżnia także rolę Globalnego zarządcy zasobów GRM (ang. *Global Resource Manager*), który odpowiedzialny będzie za dysponowanie kontenerami zasobów zgodnie z określoną polityką pomiędzy rywalizujące VG.

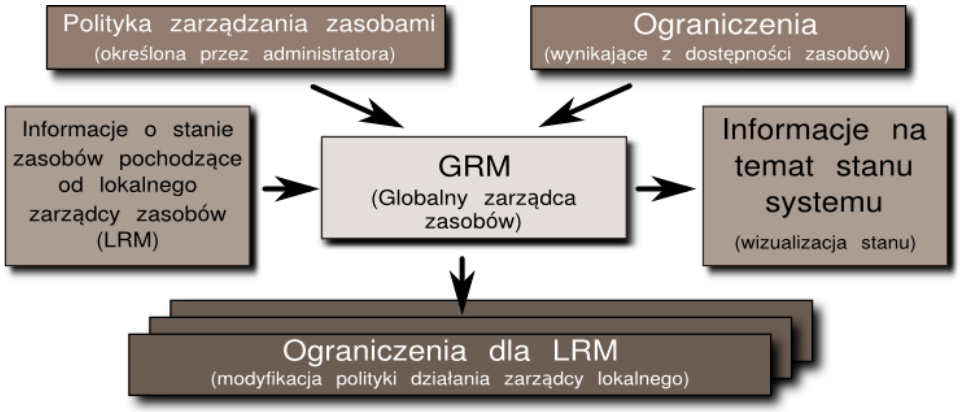
Efektem wprowadzenia GRM jest hierarchiczny model składający się z dwóch zarządców, mających przeciwstawne zadania. LRM dąży do zwiększenia wydajności aplikacji przez przydział i zmianę konfiguracji zasobów, natomiast GRM odpowiada za sprawiedliwy przydział zasobów oraz minimalizowanie ujemnego wpływu różnych VG współdzielących ten sam sprzęt. Decyzje te są podejmowane na podstawie analizy danych pochodzących z monitorowania VG oraz definicji polityki przydziału zasobów fizycznych.

W przypadku rywalizacji pomiędzy VG zarządca globalny będzie przydzielał limity, których przestrzeganie przez VG będzie gwarantowało odpowiedni poziom usługi dla pozostałych uruchomionych równocześnie VG.

Podsumowując, do głównych zadań komponentu GRM należy sterowanie wieloma LRM działającymi na danej fizycznej, współdzielonej infrastrukturze za pomocą przekazywanych do nich ograniczeń. Sterowanie to ma zapewnić z jednej strony zagwarantowany minimalny poziom usługi QoS dla aplikacji uruchomionej w obrębie VG a z drugiej maksymalizację wykorzystania współdzielonych zasobów.

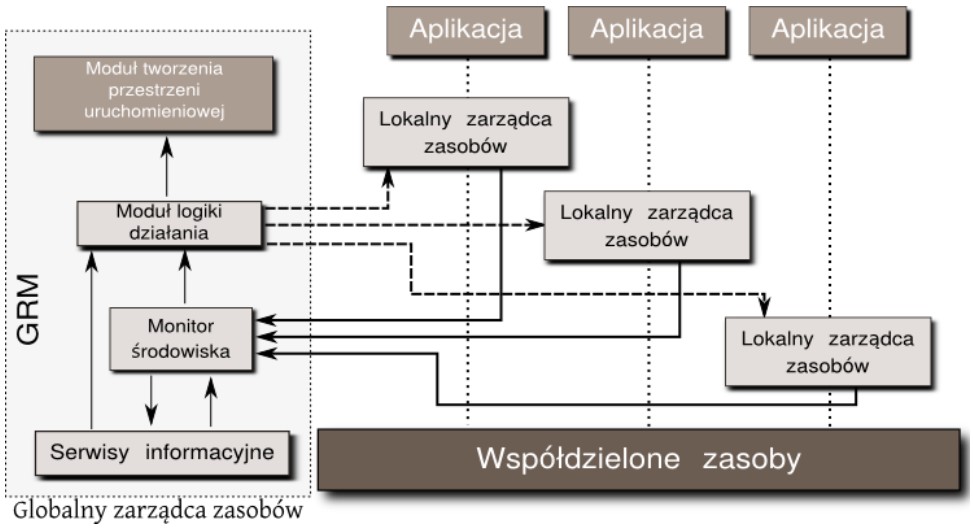
Wirtualny Grid pozwala na zdefiniowanie konfiguracji zasobów dla aplikacji oraz na elastyczną modyfikację tej konfiguracji w zależności od dynamicznie zmieniających się wymagań, stanu aplikacji oraz dostępności i poziomu wykorzystania zasobów.

Przyjęta koncepcja Wirtualnego Gridu nie precyzuje w jaki sposób przydzielone aplikacji zasoby będą przez nią wykorzystywane, zamiast tego określa jakie i ile elementów infrastruktury zostanie przydzielone na czas wykonania aplikacji. Koncepcja ta określa również



Rysunek 5.6. Interakcje globalnego zarządcy zasobów

sposób zarządzania zasobami przydzielonymi aplikacji. Użytkownicy określają konfigurację Wirtualnego Gridu poprzez specyfikację wymogów dla aplikacji, które następnie przekazane są do modułu tworzenia przestrzeni uruchomieniowej, który instancjonuje dany zestaw zasobów udostępniając interfejs umożliwiający dostęp do ich parametrów, lokalizacji, itp. Informacje te są z kolei przetwarzane przez podsystem wykonania, który wykonuje aplikacje. System monitorowania śledzi proces wykonania aplikacji, zgłaszając zdarzenia możliwej adaptacji konfiguracji do zmiennych warunków poziomu wykorzystania zasobów lub etapu wykonania aplikacji.



Rysunek 5.7. Komponenty składowe globalnego zarządcy zasobów

Podsumowując system zarządzania zasobami będzie posiadał strukturę hierarchiczną. Poszczególne poziomy hierarchii zarządzania będą określone poprzez sprecyzowanie dozwolonych operacji związanych z regulacją dostępu do zasobów dla aplikacji. Inne będą też, określone przez administratora, polityki działania dla każdego poziomu zarządzania.

5.2. Warstwowa architektura systemu

System zarządzania zasobami Grid powinien mieć skalowaną i elastyczną architekturę, umożliwiającą łatwe rozszerzanie i implementację nowych funkcjonalności, poprzez zapewnienie wsparcia dla abstrakcji zasobów, dynamicznej konfiguracji komponentów oraz lokalizacji i wykrywania elementów infrastruktury.

W dalszej części niniejszego rozdziału, szczegóły architektury systemu zarządzania zasobami VGRMS zostaną przedstawione w oparciu o określenie zestawu warstw funkcjonalnych. Podejście to zakłada identyfikację podstawowych funkcji systemu i określenie ich w hierarchicznej strukturze zależności. Każda warstwa w tym opisie zawiera określoną funkcjonalność udostępnianą warstwom bezpośrednio przyległym. Podejście to w dobry sposób strukturalizuje system i sprawdza się w wielu systemach informatycznych.

Projektowany system zarządzania zasobami wirtualnego Gridu posiada architekturę złożoną z pięciu warstw. Warstwy te wynikają z przyjętej koncepcji zastosowania wirtualizacji jako techniki zarządzania i są nimi: „warstwa zasobów fizycznych", „warstwa wirtualizacji", „warstwa ekspozycji" oraz warstwy zarządzania i prezentacji (rysunek 5.8).

Rysunek 5.8. Warstwy funkcjonalne systemu zarządzania zasobami

Warstwy te oraz komponenty do nich należące zostaną omówione w kolejnych sekcjach niniejszego rozdziału.

5.2.1. Warstwa zasobów fizycznych

Najniższą warstwę implementowanego systemu stanowi warstwa heterogenicznych zasobów. Zasoby te są dostarczone przez urządzenia i elementy infrastruktury stanowiące zazwyczaj wyposażenie typowych ośrodków obliczeniowych. Urządzeniami tymi są:

- fizyczne komputery takie jak wszelkiego rodzaju serwery czy komputery użytkowe klasy PC (ang. *Personal Computer*) lub popularne ze względu na gęstość upakowania komputery wykonane w obudowach typu *Blade*. Komputery te mogą tworzyć takie struktury jak klastry czy farmy,

- urządzenia specjalistyczne używane w ośrodkach obliczeniowych do zadań związanych z obsługą przetwarzania aplikacji. Przykładami tych urządzeń mogą być np. macierze dyskowe, urządzenia udostępniania danych masowych za pomocą sieci komputerowej[8], itp.,

- infrastruktura komunikacyjna czyli sieć komputerowa, która wykorzystywana jest do łączenia węzłów obliczeniowych (poszczególnych komputerów). Infrastruktura składa się zarówno z urządzeń aktywnych takich jak routery, switche, bezprzewodowe punkty dostępowe oraz elementów pasywnych takich jak np. okablowanie, bez których infrastruktura nie mogłaby poprawnie funkcjonować,

- komponenty programowe, takie jak system operacyjny, biblioteki systemowe, oprogramowanie obsługi wirtualizacji.

Nie wszystkie urządzenia czy komputery mogą być użyte jako elementy dostarczające zasobów w proponowanym systemie[9]. Konieczne jest tu posiadanie cech, które wynikają z zapewnienia wsparcia dla obsługi wirtualizacji, takich jak konkretna platforma sprzętowa czy wersja systemu operacyjnego.

5.2.2. Warstwa wirtualizacji

W rozproszonych środowiskach obliczeniowych jakimi są środowiska Grid, użytkownicy nie posiadają dostępu do poszczególnych urządzeń wykorzystywanej infrastruktury. Dlatego z punktu widzenia użytkownika parametry, takie jak lokalizacja, rozmieszczenie, sposób zarządzania poszczególnych zasobów nie są istotne. Jednak z punktu widzenia administratora pełna funkcjonalność i sprawność systemu zarządzania jest możliwa do osiągnięcia, pod warunkiem że wszystkie wykorzystywane zasoby i elementy składowe infrastruktury będą zarządzalne[10] oraz będą dostępne dedykowane narzędzia i oprogramowanie do ich obsługi.

Zasoby, również w proponowanym systemie, nie będą wykorzystywane (zarządzane) bezpośrednio. Dostęp bezpośredni ze względu na charakter i heterogeniczność zasobów powodowałby ograniczenie możliwości ich zarządzania[11]. Dlatego zasoby fizyczne zostały obudowane warstwą wirtualizacji, która wprowadza dodatkowy poziom abstrakcji umożliwiający lepsze ich zarządzanie.

Warstwa wirtualizacji poprzez udostępnienie odpowiednich mechanizmów tworzy abstrakcję zasobów. Aby to było możliwe zasoby muszą być w określony sposób instrumentowane. Instrumentacja ta polega na dostarczeniu funkcjonalności umożliwiającej dynamiczne grupowanie zasobów poprzez:

- tworzenie i zarządzanie wirtualnymi komputerami (grupującymi zasoby obliczeniowe),

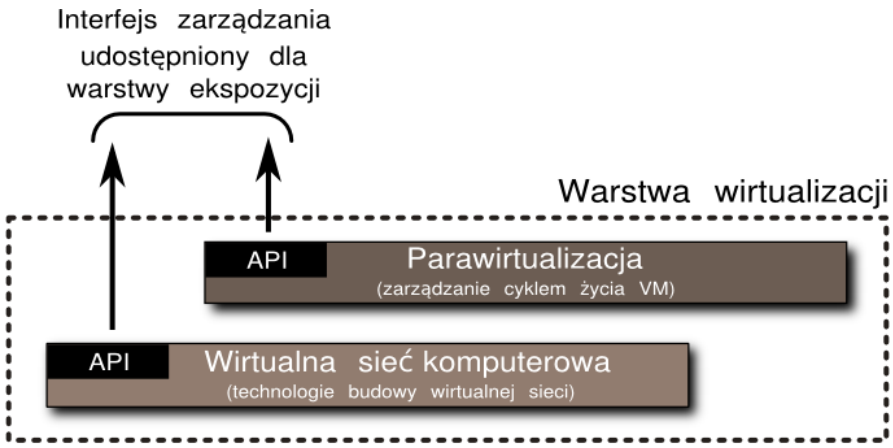
[8] Takie jak np. kontrolery protokołu iSCSI (ang. *Internet SCSI*) czy SAS (ang. *Serial Attached SCSI*) over Ethernet.

[9] Przyjęta na potrzeby niniejszej pracy taksonomia zasobów przedstawiona została na rysunku 5.2.

[10] Pod tym pojęciem rozumie się fakt posiadania interfejsu umożliwiającego (najczęściej zdalne) zarządzanie danym zasobem lub elementem.

[11] Ograniczenia te wynikają głównie z braku możliwości elastycznej kontroli wykorzystania zasobów dla aplikacji za pomocą narzędzi systemu operacyjnego.

- tworzenie i zarządzanie wirtualnymi sieciami (analogicznie dla zasobów obliczeniowych grupujących zasoby komunikacyjne).



Rysunek 5.9. Komponenty składowe warstwy wirtualizacji

Wymogiem komponentów z pozostałych warstw systemu zarządzania VGRMS jest aby funkcjonalność wprowadzona przez warstwę wirtualizacji była dostępna zdalnie poprzez sieć komputerową co umożliwi zdalną współpracę z elementami warstwy ekspozycji (rysunek 5.9).
Do głównych zadań warstwy wirtualizacji należy:

- tworzenie wirtualnych maszyn,
- dostarczenie usługi dynamicznej modyfikacji konfiguracji maszyn wirtualnych (zmiany powiązania, poziomu wykorzystania zasobów fizycznych). Realizowane jest to poprzez zmianę parametrów dynamicznych VM,
- tworzenie wirtualnych sieci,
- umożliwienie dynamicznej modyfikacji parametrów wirtualnych sieci.

Do zadań przedstawionej warstwy wirtualizacji należy również dostarczenie infrastruktury umożliwiającej dystrybucję oprogramowania systemowego dla instancji VM. Infrastruktura ta musi zapewniać dostępność dostarczonych przez użytkownika obrazów dysków zawierających oprogramowanie systemowe, biblioteki obliczeniowe, oprogramowanie obliczeniowe oraz dane wejściowe używane w procesie obliczeniowym.

5.2.3. Warstwa ekspozycji

Funkcjonalność warstwy wirtualizacji jest na tyle złożona, iż obsługa jej bezpośrednio przez warstwę zarządzania byłaby skomplikowana. Dlatego w projektowanym systemie wyróżniono elementy, które dodane zostały w celu przystosowania i ujednolicenia mechanizmów warstwy wirtualizacji, tak aby uprościć interfejs dla realizacji podstawowej funkcji systemu VGRMS jaką jest zarządzanie zasobami. Komponenty te w przedstawionym opisie

zostały umieszczone w warstwie ekspozycji, której nazwa wynika z funkcjonalności jaką jest udostępnienie możliwości wirtualizacji w postaci umożliwiającej realizację zarządzania zasobami.
Do głównych zadań komponentów wchodzących w skład warstwy ekspozycji zalicza się:

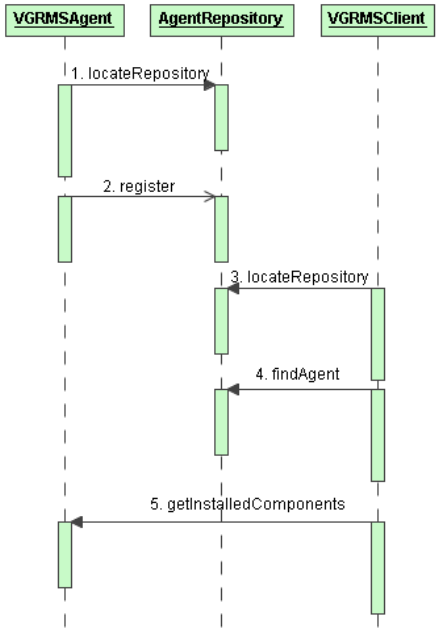
- umożliwienie zarządzania zasobami poprzez wykorzystanie możliwości udostępnionych przez warstwę wirtualizacji,
- udostępnienie usługi wyszukiwania i lokalizacji zasobów (a dokładniej komponentów systemu VGRMS realizujących zarządzanie zasobami), oraz rejestrację i wyszukiwanie pozostałych komponentów,
- monitorowanie poziomu wykorzystania zasobów oraz stanu infrastruktury,
- zarządzanie cyklem życia komponentów należących do pozostałych warstw systemu (takich jak np. komponenty warstwy zarządzania czy prezentacji),

Warstwa ekspozycji jest najbardziej złożoną z wszystkich warstw funkcjonalnych wyodrębnionych w architekturze systemu VGRMS. Dlatego przedstawienie właściwości komponentów należących do tej warstwy zostało podzielone na następujące komponenty:

- **repozytorium agentów** — czyli rejestr agentów dostarczający możliwość wyszukiwania komponentów systemu VGRMS. Każdy host używany przez system VGRMS musi posiadać uruchomiony komponent `AgentVGRMS`. Funkcją `AgentVGRMS` jest rejestracja referencji do siebie oraz informacji o innych komponentach uruchomionych w obrębie węzła systemu VGRMS.
- **komponenty fizycznego węzła** — elementy których działanie konieczne jest aby system mógł wykorzystać zasoby hosta,
- **komponent obsługi maszyny wirtualnej** — element dzięki funkcjonalności którego system może zarządzać instancjami maszyn wirtualnych,
- **komponent obsługi wirtualnej sieci** — element dzięki funkcjonalności którego system może tworzyć wirtualne topologie sieciowe dla komunikacji wirtualnych maszyn.

Repozytorium agentów

W projektowanym systemie zakłada się istnienie repozytorium agentów, z którego klient (czyli dowolny inny komponent systemu) będzie mógł otrzymać informacje o tym na jakich fizycznych węzłach zainstalowane zostały komponenty wchodzące w skład VGRMS (zgodnie z zasadą, że jeśli na danym węźle występuje agent VGRMS, to węzeł ten zawiera komponenty budujące system VGRMS). Uzyskana w ten sposób informacja pozwoli na odpytanie węzła o to jakie komponenty zarządzania zostały na nim zainstalowane. Zaletą istnienia repozytorium jest brak potrzeby posiadania wiedzy na temat rozmieszczenia komponentów w danym wirtualnym Gridzie a metoda ich automatycznego wyszukania i lokalizacji jest skalowalna. Na rysunku 5.10 przedstawiono kroki jakie klient systemu musi wykonać aby móc korzystać z komponentów VGRMS zainstalowanych na poszczególnych węzłach.



Rysunek 5.10. Proces wyszukiwania komponentów systemu VGRMS

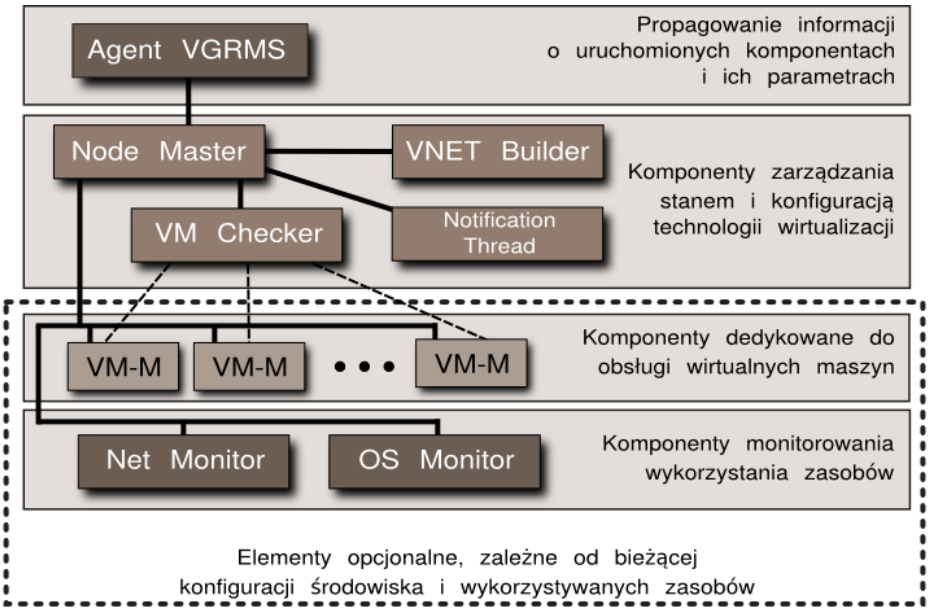
Pożądane jest aby repozytorium agentów było łatwo lokalizowane. Najlepiej jeśli wyszukiwanie repozytorium odbywać się będzie w sposób dynamiczny, tak aby klienci VGRMS nie byli świadomi miejsca, w którym zostało ono zainstalowane. Dodatkowe cechy jakimi powinno się ono odznaczać to:

- możliwość opisu agenta poprzez dodatkowe atrybuty, wg. których klienci będą mogli przeszukiwać repozytorium. Atrybuty takie będą opisywały właściwości sprzętowe węzła (np. ilość dostępnej pamięci RAM, parametry CPU, liczba kart sieciowych, itp), na którym agent jest uruchomiony. Znajomość tych parametrów pozwoli klientowi na efektywniejsze wyszukanie węzła o zadanych parametrach,

- wsparcie dla notyfikacji o rejestracji bądź wyrejestrowaniu się agenta. Poza prostymi zdarzeniami o tym, że nowy węzeł jest osiągalny (bądź zniknął), w systemie powinna istnieć możliwość definiowania wzorców, które muszą być spełnione aby zdarzenie zostało wygenerowane jeśli dopasowanie do wzorca zostanie spełnione. Zaletą takiej cechy będzie odciążenie klienta od konieczności periodycznego sprawdzania dostępności interesującego go agenta.

Komponenty fizycznego węzła

Rysunek 5.11 przedstawia komponenty składowe warstwy ekspozycji systemu VGRMS uruchamiane na fizycznych węzłach. Liniami przerywanymi zaznaczono elementy opcjonalne. Na komponenty hosta z warstwy ekspozycji składają się:



Rysunek 5.11. Komponenty składowe warstwy ekspozycji działające na hoście fizycznym w systemie VGRMS

- `AgentVGRMS` — umożliwia uzyskanie referencji do środowiska VGRMS zainstalowanego na danym węźle, przez co klient będzie miał dostęp do pozostałych komponentów znajdujących się na danym hoście.

- `NodeMaster` — odpowiedzialny za zarządzanie tworzeniem i usuwaniem komponentów warstwy ekspozycji `VM-M` i `VNetBuilder` oraz (opcjonalnych) komponentów warstwy zarządzania znajdujących się na danym węźle.

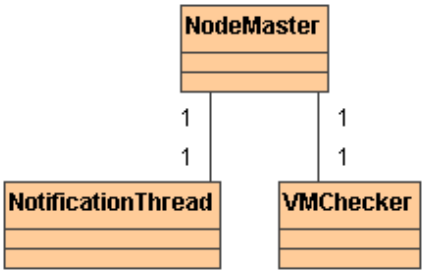
- `NotificationThread` — odpowiedzialny za odbiór zdarzeń z warstwy wirtualizacji i propagowanie ich do odpowiednich komponentów.

- `VMChecker` — komponent odpowiedzialny za periodyczne sprawdzanie stanu wirtualnych maszyn na danym węźle fizycznym i ewentualną weryfikację tego stanu z odpowiadającymi im komponentami warstwy ekspozycji (czyli sprawdzenie poprawności odwzorowania instancji VM do odpowiadającego jej komponentu ekspozycji). Istnienie takiego komponentu podyktowane jest faktem, iż VM mogą być tworzone (usuwane) również narzędziami zewnętrznymi (jak np. za pomocą tekstowej konsoli zarządzania systemu Xen) lub mogą wystąpić sytuacje awaryjne (powodujące usunięcie VM bez wygenerowania odpowiedniego zdarzenia). W przypadkach tych nie zostaną dla nich automatycznie stworzone (usunięte) komponenty z warstwy ekspozycji, a zatem dodatkowy mechanizm musi wykryć taką sytuację i podjąć odpowiednie działania. Sytuacja taka może również wystąpić gdy zdarzenie pochodzące z warstwy wirtualizacji nie zostanie odebrane przez komponent `NotificationThread`, dlatego zapewnienie redundantnego sposobu weryfikacji[12] jest tutaj uzasadnione.

[12]Mechanizm ten jest często stosowany również w protokołach sieciowych routingu dynamicznego, gdzie

- `VNetBuilder` — komponent, którego zadaniem jest tworzenie topologii sieciowej umożliwiającej (najczęściej odseparowaną) komunikację wirtualnych maszyn.[13]
- `VM-M` — (element opcjonalny) oznacza komponent ekspozycji dedykowany do obsługi maszyny wirtualnej. Komponent ten występuje na danym węźle fizycznym jedynie w przypadku gdy w jego obrębie działa maszyna wirtualna. Dzięki obecności tego elementu możliwe jest wykonywanie operacji na danej instancji VM w sposób autonomiczny przez pozostałe elementy warstwy zarządzania systemu VGRMS.

Zgodnie z rysunkiem 5.11 w skład komponentów zarządzania stanem i konfiguracją fizycznego węzła wchodzą trzy obowiązkowe elementy: `NodeMaster`, `NotificationThread` oraz `VMChecker`. Zależności pomiędzy nimi pokazano na rysunku 5.12.



Rysunek 5.12. Relacje pomiędzy głównymi komponentami zarządzania stanem i konfiguracją fizycznego węzła

W trakcie inicjalizacji węzła VGRMS[14] jest uruchomiony komponent `NodeMaster`, którego zadaniami, istotnymi z punktu widzenia klienta korzystającego z usług warstwy ekspozycji, są:

- tworzenie, usuwanie i migracja VM, pobranie aktualnej listy i parametrów wirtualnych maszyn[15],
- tworzenie i usuwanie komponentów warstwy zarządzania, pobranie aktualnej listy i parametrów komponentów warstwy zarządzania.

Komponenty `NotificationThread` oraz `VMChecker` spełniają zadania pomocnicze (omówione na początku sekcji) w stosunku do `NodeMaster` i mają charakter uzupełniający.

Komponent obsługi maszyny wirtualnej `VM-M`

Komponent obsługi `VM-M` jest to komponent powiązany z maszyną wirtualną, który udostępnia interfejs, dzięki czemu możliwe jest zdalne wpływanie na stan i parametry VM.

mimo przesyłania potwierdzeń komunikacji o zmianach w topologii, występują również pełne okresowe weryfikacje zgodności informacji o stanie połączeń [69].

[13] Działanie tego komponentu zostanie przedstawione w dalszej części niniejszej sekcji.

[14] Np. podczas startu systemu operacyjnego.

[15] Operacji tych ze względu na konieczność obsługi propagacji zdarzeń nie może realizować komponent `VM-M`.

Podstawową funkcjonalnością, której posiadanie wymusiło dodanie tego komponentu jest konieczność dostosowania specyficznego interfejsu zarządzania daną implementacją wirtualizacji do sposobu zarządzania wynikającego z przyjętej koncepcji reprezentacji zasobów (obliczeniowych i komunikacyjnych) w systemie VGRMS.

Parametry, których obsługę implementuje komponent zarządzania VM można podzielić na dwie grupy:

- parametry obsługiwane przez implementację wirtualizacji — których obsługa przez komponent `VM-M` jest konieczna ze względu, iż parametry te nie są zachowywane przy operacjach takich jak restart systemu operacyjnego VM czy migracja VM (np. ograniczenia czasu dostępu do procesora lub pamięci),
- parametry i metody specyficzne dla systemu VGRMS — obsługa funkcjonalności koniecznej do realizacji przyjętego dla systemu sposobu regulacji przydziału zasobów (np. typ VM, powiązanie z wirtualnym Gridem).

W funkcjonalności przedstawianego komponentu wymagane są również procedury monitorowania stanu oraz parametrów opisujących konsumpcję zasobów przez daną maszynę wirtualną. Metody te pozwalają na dostęp do takich parametrów jak np. typ danej instancji VM, rodzaj i wersja systemu operacyjnego, ilość i obciążenie wirtualnych procesorów w systemie, itp. Parametry te są używane w większości przypadków przez komponenty zarządzania do realizacji polityki przydziału zasobów.

Komponent `VNetBuilder`

Na potrzeby niniejszego systemu konieczne było opracowanie mechanizmów umożliwiających budowanie i zarządzanie siecią komputerową wirtualnych maszyn. Mechanizm ten obsługiwany jest przez komponent `VNetBuilder`. Odpowiedzialny on jest za stworzenie (i ewentualne aktualizacje) topologii sieciowej, której opis przechowywany jest przez element `VG-M`. Proces ten wymaga modyfikacji parametrów interfejsów sieciowych fizycznych węzłów, dlatego element ten musi być uruchomiony na każdym węźle, którego zasoby są wykorzystywane w wirtualnym Gridzie dla którego tworzona jest topologia sieciowa.

Komponent `VNetBuilder` odpowiedzialny jest również za modyfikację topologii sieciowej wirtualnego Gridu. Modyfikacje te mogą wynikać ze zmiany kształtu topologii logicznej (modyfikacja przez administratora) lub ze zmiany kształtu topologii fizycznej łączącej fizyczne węzły (np. przez migrację wirtualnych maszyn).

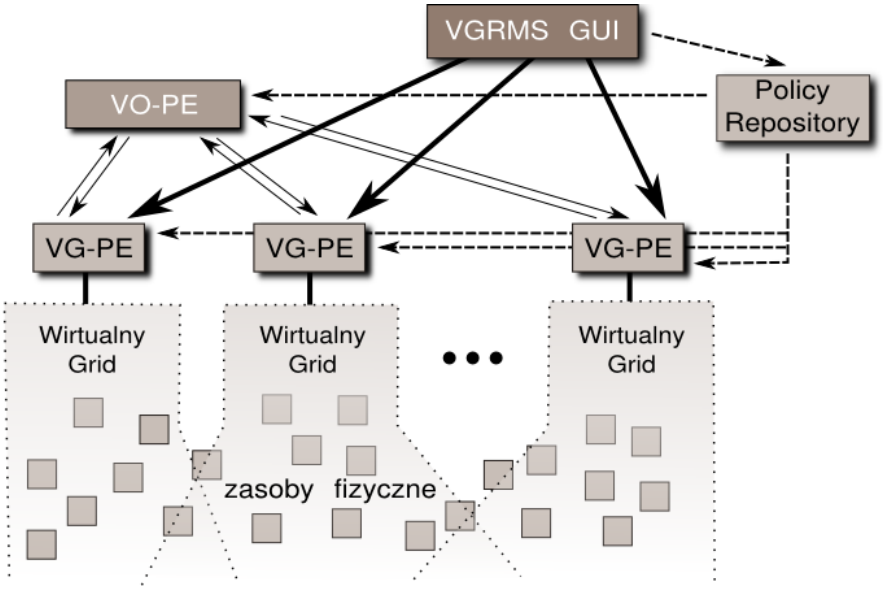
Realizacja funkcjonalności komponentu `VNetBuilder` opiera się na modyfikacji konfiguracji mechanizmów sieciowych dla fizycznego hosta. Zmieniana jest konfiguracja zarówno parametrów interfejsów (za pomocą poleceń systemowych) oraz topologia wirtualnej sieci (modyfikacja konfiguracji oprogramowania dokonującego tunelowania, enkapsulacji komunikacji wirtualnych maszyn).

5.2.4. Warstwa zarządzania

Główną funkcjonalnością warstwy zarządzania jest przetwarzanie informacji o stanie środowiska i obsługa procedury przydziału lub/i rezerwacji zasobów dla aplikacji rozproszonej. Tworzenie środowiska wykonawczego aplikacji realizowane przez komponenty warstwy zarządzania można podzielić na następujące etapy:

1. Rejestracja komponentów środowiska — takich jak fizyczne zasoby, wyposażone w oprogramowanie umożliwiające zarządzanie.
2. Przetwarzanie początkowej specyfikacji wirtualnego Gridu.
3. Konfiguracja połączeń sieciowych za pomocą mechanizmów wirtualnych sieci.
4. Modyfikacja konfiguracji infrastruktury (automatyczna lub za pomocą akcji wywołanych przez operatora) na podstawie danych dostarczonych przez podsystem śledzenia poziomu wykorzystania zasobów.
5. Zwolnienie zasobów, usunięcie rezerwacji.

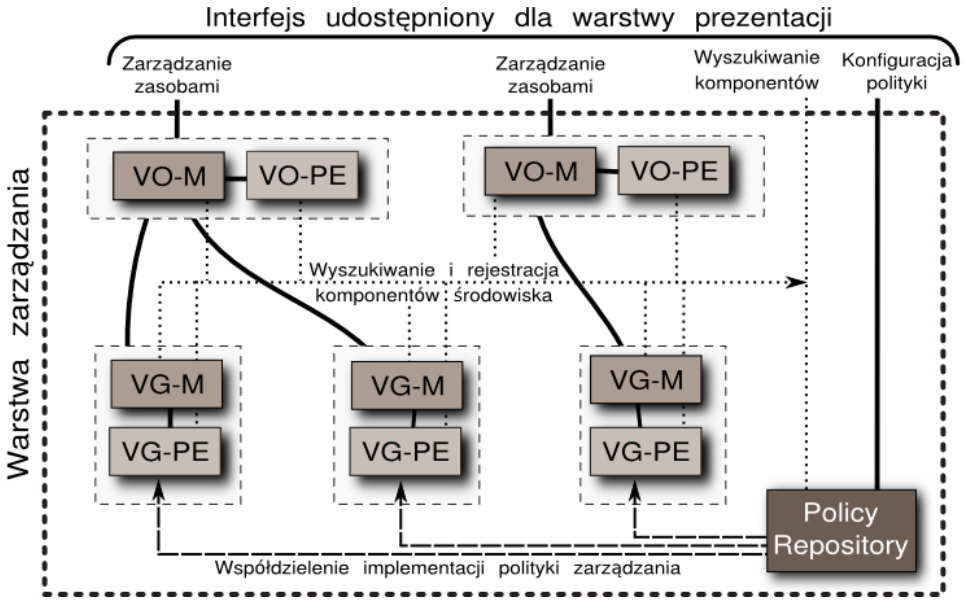
Rysunek 5.13 przedstawia odwzorowanie przyjętej we wstępie tego rozdziału hierarchicznej koncepcji zarządzania za pomocą powiązań silników regułowych działających na różnych poziomach zarządzania. Zgodnie z rysunkiem, element `VO-PE` może wpływać na działanie silników regułowych poszczególnych wirtualnych Gridów. Zarządzanie na dowolnym poziomie może być kontrolowane i modyfikowane za pomocą graficznej konsoli zarządzania przez administratora systemu.



Rysunek 5.13. Hierarchia komponentów zarządzania systemu VGRMS

Rysunek 5.14 przedstawia komponenty warstwy zarządzania i ich powiązania. Komponenty te to:

- `VO-M` (*VO Manager*) oraz `VO-PE` (*VO Policy Engine*) — działanie tych elementów w stosunku do wirtualnej organizacji jest analogiczne do komponentów zarządzania wirtualnym Gridem, które to zostaną dokładniej przedstawione.
- `VG-M` (*VG Manager*) — oznacza komponent zarządzania konfiguracją i stanem wirtualnego Gridu. Pojedyncza instancja tego komponentu reprezentuje jeden wirtualny



Rysunek 5.14. Komponenty warstwy zarządzania systemu VGRMS

Grid. Komponent ten posiada informacje o konfiguracji wirtualnego Gridu oraz implementuje operacje związane z zarządzaniem stanem wirtualnego Gridu. System wymaga, iż komponent ten będzie uruchomiony na jednym z węzłów wchodzących w skład danej instancji wirtualnego Gridu.

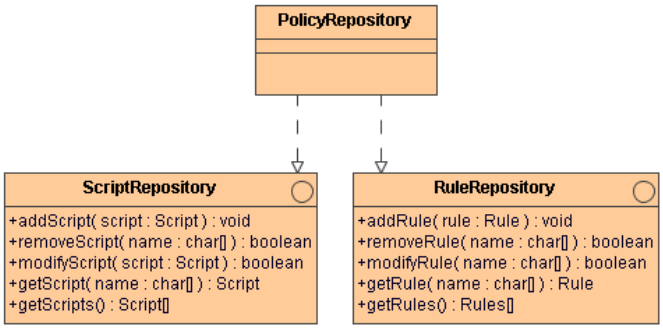
- `VG-PE` (*VG Policy Engine*) — element oznaczający silnik regułowy (ang. *Policy Engine*). Na rysunku 5.14 element ten został oznaczony wspólną linią przerywaną z `VG-M`, co oznacza iż element ten jest ściśle zintegrowany z `VG-M` i zawsze utworzenie (usunięcie) `VG-M` powoduje utworzenie (usunięcie) komponentu `VG-PE`. Komponent ten jest odpowiedzialny za wykonanie polityki zarządzania zasobami wirtualnego Gridu, która to została narzucona przez administratora systemu.

- `PolicyRepository` — globalne repozytorium polityk zarządzania, komponent którego zadaniem jest przechowywanie i udostępnianie reguł w oparciu o które budowana będzie polityka przydziału zasobów. Istnienie centralnego repozytorium polityk umożliwia współdzielenie konfiguracji dla wielu elementów `VG-PE` lub `VO-PE`.

Repozytorium polityk zarządzania (`PolicyRepository`)

W systemie zakłada się istnienie centralnego repozytorium polityk zarządzania, w którym gromadzone będą wszystkie definiowane polityki dla VG, tak aby można było je wykorzystywać w innych VG bez konieczności ponownej ich deklaracji. Takie rozwiązanie pozwala na współdzielenie złożonych reguł pomiędzy wieloma instancjami wirtualnych Gridów (wirtualnych organizacji) zarejestrowanych w systemie. Dodatkowo dzięki centralnemu repozytorium możliwe będzie proste wykonywanie operacji, które zdalnie spowodują zmiany na liście wykorzystywanych reguł dla elementu `PE`. Repozytorium polityk udostępnia podstawowe operacje CRUD (ang. *Create, Read, Update, Delete*) dla reguł i skryptów

(rysunek 5.15) implementujących polityki zarządzania, które można w nim przechowywać.



Rysunek 5.15. Funkcjonalność udostępniana przez repozytorium polityk zarządzania zasobami

Reprezentacja informacji o stanie systemu

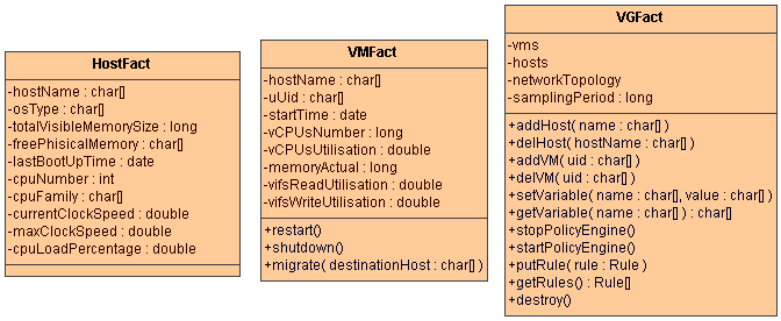
W trakcie działania systemu zbierane są informacje o stanie komponentów systemu. Fakty to informacje dotyczące aktualnego stanu danego komponentu systemu, parametrach dostępnych zasobów i ich wykorzystaniu będące niezbędne do podejmowania akcji poprzez silnik regułowy. W systemie VGRMS wyróżniono trzy rodzaje faktów:

1. Fakty systemowe — pierwszą grupę stanowią fakty, których cyklem życia zarządza system VGRMS. Fakty te są tworzone, usuwane i aktualizowane automatycznie na podstawie zdarzeń opisujących zmiany stanu komponentów systemu. Do faktów tej grupy należą:

   - `HostFact` — reprezentacja informacji na temat parametrów fizycznych hosta, a także poziomu wykorzystania zasobów obliczeniowych. Fakt ten jest wykorzystywany do zarządzania zasobami na poziomie wirtualnego Gridu (rysunek 5.16),
   - `VMFact` — reprezentacja informacji opisujących stan i konfigurację maszyny wirtualnej. Fakt ten pozwala także na dostęp do interfejsu zarządzania VM (rysunek 5.16),
   - `VGFact` — reprezentacja informacji o zasobach wirtualnego Gridu. Informacje te są wykorzystywane do zarządzania zasobami na poziomie wirtualnej organizacji (rysunek 5.16).

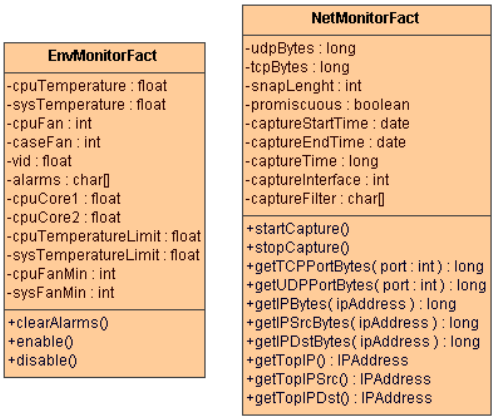
2. Fakty dynamiczne — tworzone i usuwane na żądanie za pomocą reguł zaimplementowanych w silniku regułowym. Fakty te używane są jeśli dana heurystyka przydziału zasobów wymaga do działania informacji przez nie dostarczanych. Przykładem faktów tej grupy są:

   - `EnvMonitorFact` — monitorowanie parametrów środowiska takich jak np. temperatura komponentów fizycznych, błędy sprzętu, itp. (rysunek 5.17),



Rysunek 5.16. Reprezentacja informacji o aktualnym stanie węzła, wirtualnej maszyny i wirtualnego Gridu w postaci faktów przetwarzanych przez silnik regułowy

- `NetMonitorFact` — monitorowanie poziomu wykorzystania zasobów komunikacyjnych (rysunek 5.17),



Rysunek 5.17. Reprezentacja informacji pochodzących z monitorowania komunikacji sieciowej w postaci faktów

3. Fakty tymczasowe — istnieje możliwość tworzenia dodatkowych faktów, których postacią i sposobem wykorzystania zarządza administrator. Fakty te najczęściej służą do przechowywania informacji tymczasowych, wyników obliczeń, danych historycznych, wartość zagregowanych itp.

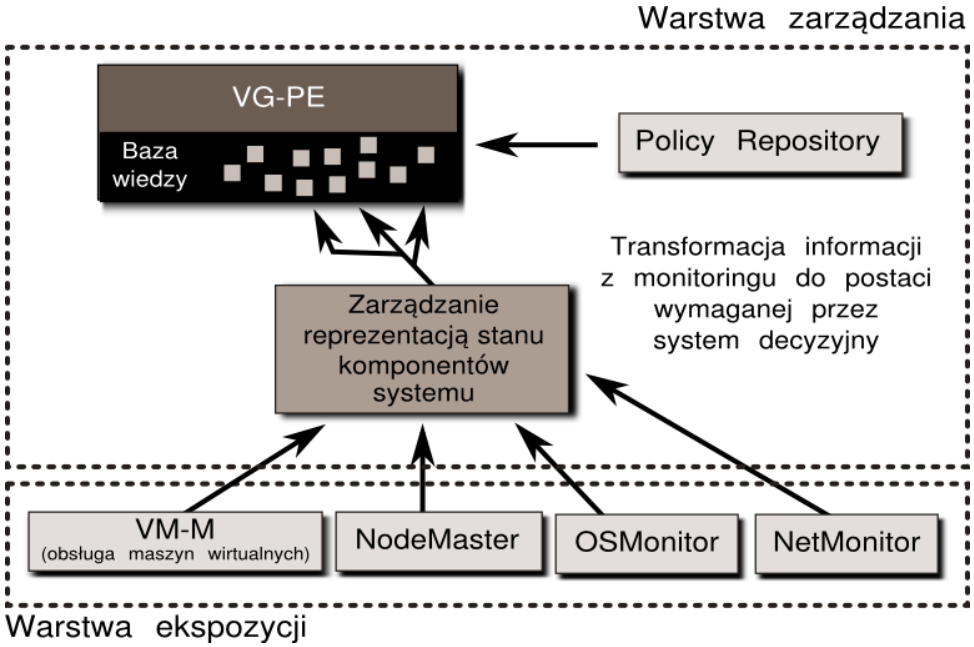
Wyłączenie zarządzania faktami `EnvMonitorFact` oraz `NetMonitor` spod kontroli systemu powoduje, iż można tworzyć te fakty tylko wtedy kiedy są wymagane, odciążając zasoby w sytuacjach gdy monitorowanie danych parametrów nie jest konieczne. Dodatkowo istnieje możliwość parametryzowania działania obiektów zbierających dane (np. dla komunikacji sieciowej daje to możliwość określenia, iż tylko na jednym hoście (interfejsie) będą zbieranie informacje).

Poza informacjami o stanie zasobu, który jest reprezentowany przez dany fakt, fakt posiada również metody, których wywołanie może nastąpić z poziomu silnika regułowego.

Rozwiązanie to pozwala na zarządzanie stanem obiektu, którego dany fakt jest reprezentacją. Metody te są odpowiedzialne za komunikację i wywołanie odpowiednich metod na obiektach reprezentowanych przez dany fakt. Dla faktów opisujących np. stan wirtualnych maszyn będą to takie metody, które pozwolą na zdalne wyłączenie, restart czy migrację VM. Metody, które będą mogły być wywoływane na faktach reprezentujących fizyczne węzły, będą pozwalały na dynamiczne tworzenie dodatkowych komponentów takich jak np. monitory komunikacji sieciowej czy parametrów środowiska.

Konfiguracja silnika regułowego systemu może także być tworzona w oparciu o wykorzystanie interpretatora języków skryptowych. Pozwoli to na łatwą implementację złożonych akcji[16], wykonywanych jako rezultat działania danej reguły.

Rysunek 5.18 przedstawia proces transformacji informacji zbieranych od elementów warstwy ekspozycji i umieszczanie ich w postaci faktów w bazie wiedzy silnika regułowego.



Rysunek 5.18. Transformacja informacji o stanie infrastruktury do postaci faktów

Poprzez zaimplementowaną w komponencie `VG-M` funkcjonalność dostępną z poziomu silnika regułowego możliwe będzie zarządzanie konfiguracją (wirtualnej) topologii sieciowej wykorzystywanej do komunikacji w obrębie wirtualnego Gridu. Mechanizm ten jest dopełnieniem funkcjonalności zarządzania zasobami obliczeniowymi (poprzez zmianę parametrów wirtualnych maszyn) o możliwość zarządzania zasobami komunikacyjnymi (zmiana parametrów wirtualnej topologii sieciowej).

Komponenty zarządzania wirtualnym Gridem

Komponentami zarządzania wirtualnym Gridem są `VG-M` oraz `VG-PE` (rysunek 5.14).

[16]Takich jak np. wysyłanie informacji o sytuacjach wyjątkowych za pomocą poczty elektronicznej czy generowanie raportów itp.

Tworzenie wirtualnego Gridu jest pierwszą operacją, której wykonanie umożliwi zarządzanie zasobami w systemie VGRMS.[17] Aby zatem utworzyć wirtualny Grid należy wyspecyfikować:

- nazwę (unikalna w obrębie danej wirtualnej organizacji),
- opis (używany przez administratora do identyfikacji VG),
- listę węzłów — hosty wchodzące w skład wirtualnego Gridu. Lista ta może być modyfikowana w czasie cyklu życia wirtualnego Gridu, warunkiem koniecznym jest podanie przynajmniej jednego fizycznego hosta,
- (opcjonalny) zestaw filtrów, które będą używane do automatycznego dodawania do listy węzłów tych elementów, które spełniają zadane kryteria,
- listę wirtualnych maszyn — w czasie cyklu życia wirtualnego Gridu maszyny wirtualne mogą być dodawane i można je też usuwać. Maszyny wirtualne mogą również migrować, ale tylko w obrębie węzłów wchodzących w skład danego wirtualnego Gridu,
- definicję topologii sieciowej używanej do komunikacji wirtualnych maszyn wchodzących w skład wirtualnego Gridu[18].

Rysunek 5.19 przedstawia interakcje komponentów w trakcie procesu tworzenia wirtualnego Gridu, podczas którego powstaje obiekt `VG-M`, określający parametry konfiguracyjne danego wirtualnego Gridu oraz silnik regułowy `VG-PE` (*VG PolicyEngine*), który w stanie początkowym inicjalizowany jest faktami dotyczącymi hostów i wirtualnych maszyn należących do danego wirtualnego Gridu.

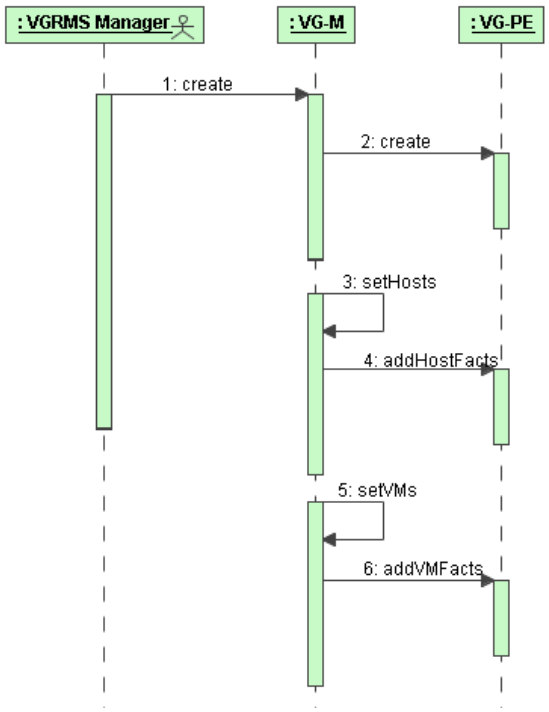
Jak pokazano na rysunku 5.19 po utworzeniu VG, silnik regułowy `VG-PE` załadowany jest faktami dotyczącymi aktualnego stanu wirtualnego Gridu. W tym momencie nie posiada on żadnych polityk zarządzania, w związku z czym nie podejmuje żadnych akcji. Liczba faktów, które w stanie początkowym znajdują się w `VG-PE` to liczba węzłów tworzących VG powiększona o liczbę VM znajdujących się na każdym z tych węzłów.

Zarządzanie konfiguracją wirtualnych maszyn wykonywane przez `VG-M` ogranicza się wyłącznie do aktualizowania listy VM i list fizycznych węzłów oraz udostępniania metod związanych z odczytem/zapisem zmiennych, które ustawiane są w trakcie inicjalizowania wirtualnego Gridu. Z poziomu `VG-M` nie ma możliwości wpływania na stan wirtualnych maszyn. Te operacje możliwe są na poziomie zarządzania polityką VG, czyli poprzez silnik regułowy `VG-PE`. Wszelkie zmiany w systemie VG, otrzymuje on poprzez zdarzenia, na które odpowiednio reaguje aktualizując zmienne. `VG-M` ma jednak możliwość tworzenia topologii sieciowych z urządzeń, które wchodzą w jego skład. Rodzaj urządzenia określony jest poprzez typ VM [19].

[17] Stworzenie wirtualnej organizacji, do której będzie przypisany wirtualny Grid, jest opcjonalne. System jest przygotowany do tworzenia wirtualnych organizacji jednak eksperymenty w tym zakresie nie były prowadzone.

[18] Definicja topologii jest opcjonalną, tzn. w przypadku gdy nie zostanie wyspecyfikowania, system przyjmie domyślą topologię pojedynczej sieci LAN.

[19] Typy te zostały przedstawione w kodzie źródłowym komponentu zarządzania VM A.3.



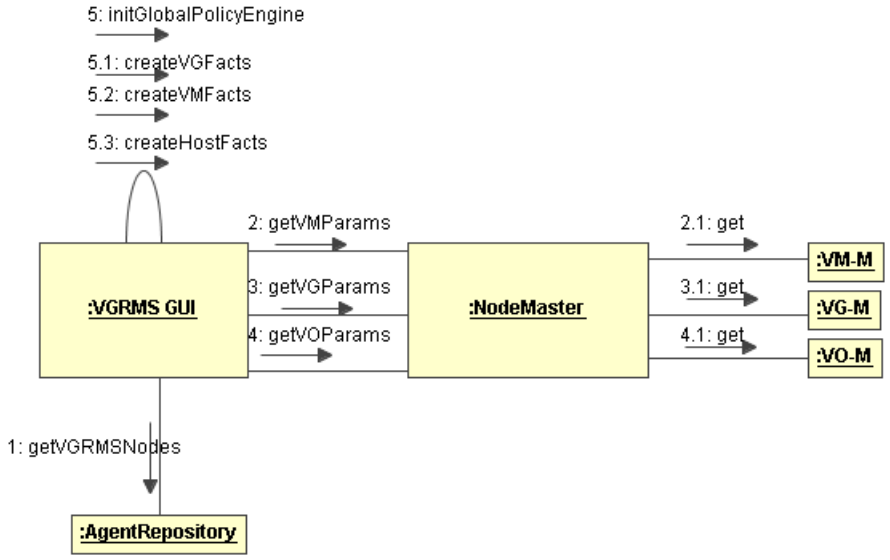
Rysunek 5.19. Interakcje komponentów systemu przy tworzeniu wirtualnego Gridu

5.2.5. Warstwa prezentacji

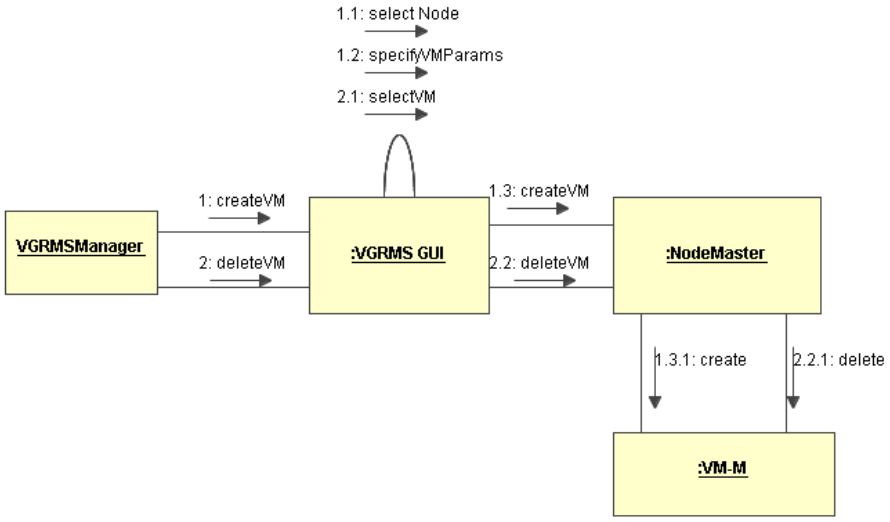
Do warstwy prezentacji należą komponenty odpowiedzialne za udostępnianie końcowemu użytkownikowi możliwości wpływania na stan pozostałych elementów systemu. Realizowane jest to poprzez dostarczenie interfejsu graficznego w postaci konsoli systemu VGRMS. Każda konsola może być powiązana z jedną bądź więcej instancjami komponentów reprezentujących i zarządzających wirtualną organizacją i wchodzącymi w jej skład wirtualnymi Gridami.

Komponenty warstwy prezentacji systemu VGRMS dostęp do pozostałych komponentów systemu uzyskują poprzez usługi wyszukiwania udostępniane z warstwy ekspozycji. Dostęp do informacji o stanie komponentów (reprezentujących zasoby, logikę zarządzania itp.) uzyskiwany jest przez odpytanie komponentu `AgentRepository` (rysunek 5.20).
Do funkcjonalności udostępnionej przez elementy warstwy wirtualizacji zalicza się:

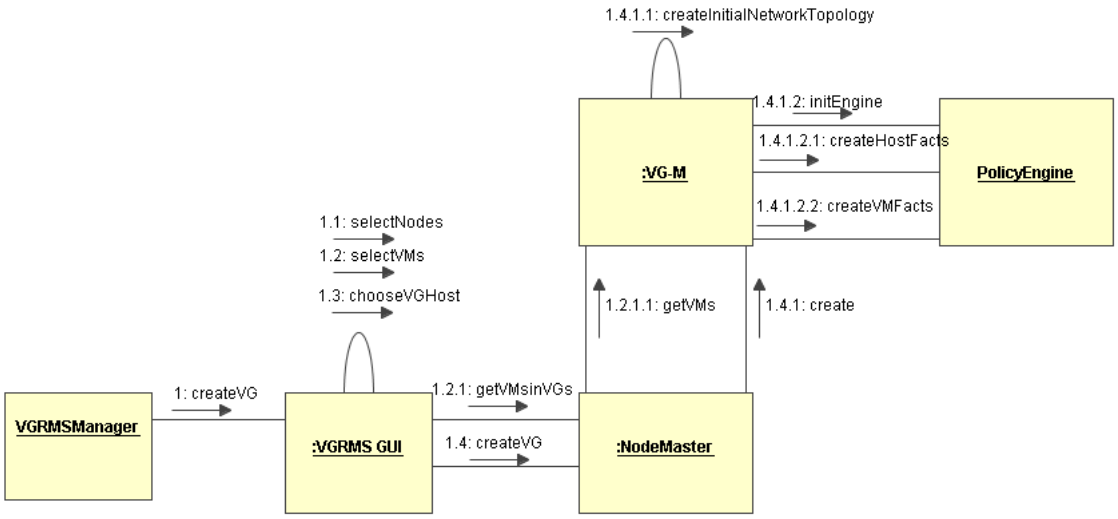
- udostępnienie kreatorów dla początkowych konfiguracji wirtualnych maszyn, wirtualnych Gridów oraz wirtualnych organizacji,
- umożliwienie zarządzania konfiguracją programową wirtualnej maszyny (za pomocą konsoli graficznej dla systemu operacyjnego VM). Rysunek 5.21 przedstawia podstawowe operacje (tworzenia i usuwania) maszyny wirtualnej wykonywane przez administratora,
- umożliwienie zarządzania konfiguracją wirtualnego Gridu (rysunek 5.22),



Rysunek 5.20. Sekwencja interakcji komponentów systemu podczas inicjalizacji konsoli zarządzania



Rysunek 5.21. Zarządzanie cyklem życia maszyny wirtualnej przez administratora z użyciem konsoli

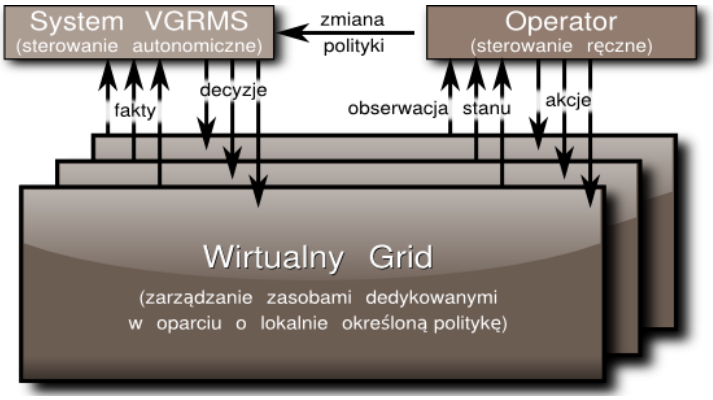


Rysunek 5.22. Tworzenie wirtualnego Gridu wykonywane przez administratora z użyciem konsoli

- umożliwienie zarządzania konfiguracją wirtualnej organizacji,
- umożliwienie określenia kształtu topologii sieciowej wirtualnego Gridu za pomocą graficznego edytora,
- dostarczenie interfejsu umożliwiającego definicje polityk zarządzania dla wirtualnego Gridu oraz wirtualnej organizacji,
- dostarczenie graficznej reprezentacji stanu systemu poprzez wizualizację (w postaci dynamicznie aktualizowanych wykresów) informacji pochodzących z monitorowania poziomu wykorzystania zasobów,
- umożliwienie ciągłego działania komponentów systemu po wyłączeniu klienta graficznego.

Przedstawiona funkcjonalność udostępniana będzie z wykorzystaniem architektury ciężkiego klienta. Konfiguracje i działanie pozostałych komponentów będą kontrolowane z poziomu konsoli tylko w trakcie jej działania, natomiast działanie systemu związane z regulacją przydziału zasobów będzie mogło być kontynuowane po jej wyłączeniu. Sekwencja 5.20 ukazuje proces dołączenia konsoli do systemu VGRMS. W trakcie tej procedury są pobierane referencje do komponentów `NodeMaster` działających na fizycznych hostach i dalej do pozostałych elementów systemu.

Możliwości administratora korzystającego z konsoli zarządzania wynikają z przyjętego rozwiązania, zgodnie z którym może on wykonywać zarówno operacje modyfikacji konfiguracji dystrybucji zasobów jak i modyfikować zasady autonomicznej ich regulacji (rysunek 5.23). Zaznaczyć należy, że jest to reprezentacją wprowadzonej w modelu (rozdział 4) pętli zarządzania, której wykonanie może odbywać się zarówno autonomicznie (domknięcie pętli sprzężenia z użyciem elementów warstwy zarządzania systemu VGRMS) jak i przez



Rysunek 5.23. Rodzaje zarządzania przydziałem zasobów w systemie VGRMS

operatora (domknięcie pętli sprzężenia poprzez sterowanie ręczne), który obserwując stan zasobów, może podejmować decyzje o zmianach ich konfiguracji.

Na rysunku 5.23 oprócz wspomnianego wcześniej domknięcia pętli sterowania związanego z autonomicznym i ręcznym zarządzaniem, istnieje jeszcze domknięcie związane z modyfikacją przez operatora polityki przydziału zasobów. Modyfikacja ta może być dokonywana na podstawie obserwacji stanu systemu i wykonywana w czasie działania aplikacji.

5.3. Konfiguracje systemu VGRMS

System VGRMS jest systemem rozproszonym. Jego komponenty mogą być różnie alokowane do elementów infrastruktury fizycznej. Komponenty w przedstawionym systemie komunikują się za pomocą sieci komputerowej, ich rozmieszczenie w obrębie zarządzanej infrastruktury udostępniającej zasoby jest praktycznie dowolne[20] Rysunek 5.24 przedstawia przykładowe rozmieszczenie komponentów zarządzania[21] w obrębie infrastruktury fizycznej wykorzystującej do komunikacji sieć rozległą WAN. Na prezentowanym przykładzie zostały stworzone dwa wirtualne Gridy należące do jednej wirtualnej organizacji.

5.4. Podsumowanie

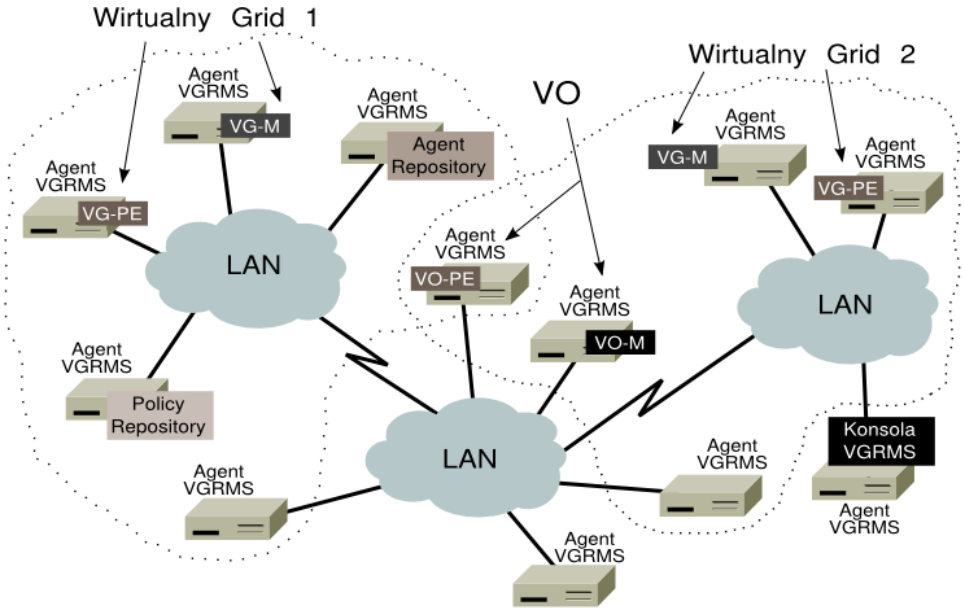
Niniejszy rozdział przedstawił architekturę systemu zarządzania zasobami w środowiskach Grid. Architektura ta została opisana w oparciu o zaproponowany podział na warstwy funkcjonalne, do których należały poszczególne komponenty systemu.

Stworzenie zestawu komponentów o ściśle określonej funkcjonalności współpracujących ze sobą za pomocą zdalnych interfejsów spowodowało, iż możliwe było spełnienie wcześniej przedstawionych wymagań[22], które nowo tworzony system powinien spełniać.

[20] Wymagane jest jedynie, aby na każdym fizycznym węźle działał komponent `NodeMaster` oraz `AgentVGRMS`.

[21] W rzeczywistości w systemie brak jest zależności wymuszających jakiekolwiek rozmieszczenie komponentów warstwy zarządzania.

[22] Przedstawienie i dyskusję wymagań stawianych dla systemu zarządzania zasobami zawiera rozdział 3.



Rysunek 5.24. Rozmieszczenie komponentów systemu VGRMS w środowisku rozproszonym

W kolejnym rozdziale zostaną przedstawione szczegóły implementacyjne i rozwiązana użyte podczas praktycznej realizacji systemu VGRMS.

Rozdział 6

Implementacja systemu

Rozdział ten przedstawia szczegóły implementacyjne systemu VGRMS Zawarte informacje dotyczą ogólnych rozwiązań i technologii użytych w trakcie tworzenia implementacji, a także przedstawiają szczegóły działania poszczególnych komponentów środowiska.

Dalsze części rozdziału zawierają informacje na temat sposobu realizacji głównych funkcji takich jak optymalizacja konfiguracji przestrzeni uruchomieniowej aplikacji, automatyczne dostosowanie do zmian konfiguracji zasobów oraz monitorowanie środowiska i aplikacji.

Rozdział zamyka opis możliwości graficznego narzędzia do obsługi poszczególnych komponentów systemu oraz przedstawienie typowych scenariuszy działania procesów optymalizacji wraz z opisem wykorzystywanych przez nie algorytmów oraz sposobów ich realizacji.

6.1. Technologie implementacji elementów systemu VGRMS

Projektowanie w rozdziale 5 systemu VGRMS, odbywało się zgodnie z koncepcją MDA, której koncepcja pokrótce została przedstawiona w rozdziale 3. Tego typu podejście zagwarantowało niezależność od środowiska implementacyjnego.

Ze względu na charakterystykę systemu, którego głównym celem jest zarządzanie zasobami w oparciu o złożone heterogeniczne mechanizmy, autor postanowił stworzyć prototyp systemu w technologii JMX (ang. *Java Management Extensions*) [90–92]. Technologia ta pozwoliła na realizację i unifikację różnych metod zarządzania rozproszonymi obiektami za pomocą jednolitego i przejrzystego interfejsu programistycznego. Dodatkowym argumentem przemawiającym za wyborem JMX API jest fakt, iż JVM od wersji 1.5 jest wyposażona w implementację tej specyfikacji. Komponenty systemu, zaproponowane w rozdziale 5, zostały odwzorowane do elementów technologii JMX.

Poziom instrumentacji zasobu sprowadza się do konstrukcji obiektu Java reprezentującego interfejs zarządzający. Interfejs zarządzający udostępnia aplikacjom klienckim atrybuty zasobu oraz możliwe operacje jakie na danym zasobie mogą być wykonane. Reprezentowany w ten sposób zasób nazywany jest MBean'em, (ang. *Managed Bean*) i jest podstawowym elementem zarządzanej infrastruktury. Centralnym komponentem poziomu serwera jest `MBeanServer`, w którym tworzone są i rejestrowane MBean'y. Klienci chcący zarządzać zasobami, muszą najpierw uzyskać do nich dostęp (odnaleźć odpowiedniego MBean'a) poprzez `MBeanServer`. Dostęp ten może odbywać się w sposób lokalny lub w sposób zdalny, za pomocą tzw. konektorów (łączników) i adapterów protokołów. Elementy te działają w warstwie agenta i usług rozproszonych, i odpowiadają za zdalną komunikację z wykorzystaniem protokołów takich jak HTTP (ang. *HyperText Transfer Protocol*), RMI (ang. *Remote Method Invocation*), Burlap, Hessian, SOAP (ang. *Simple Object*

Access Protocol) (konektory) i standardów zarządzania, takich jak WBEM (ang. *Web-Based Enterprise Management*) i SNMP (ang. *Simple Network Management Protocol*) (adaptery)[2, 9, 58].

6.1.1. Implementacja głównych komponentów systemu

W sekcji tej zostaną przedstawione rozwiązania implementacyjne wykorzystane podczas realizacji głównych komponentów systemu. Komponentem, który wymaga przedstawienia w pierwszej kolejności, jest repozytorium agentów dostarczające usługi wyszukiwania i rejestracji dla pozostałych elementów systemu.

Repozytorium agentów

Specyfikacja JMX opisuje mechanizmy umożliwiające ogłaszanie i znajdowanie agentów JMX z wykorzystaniem istniejących technologii dedykowanych do tych celów. Wymienione zostały w niej m.in. protokół SLP (ang. *Service Location Protocol*) [44], LDAP [105] i Jini [98]. Biorąc pod uwagę właściwości, jakimi repozytorium agentów powinno się cechować (sekcja 5.2.3) autor zdecydował się na wybór technologii Jini, jako że technologia ta posiada bardzo bogate możliwości co zdecydowanie ułatwia implementację tego komponentu.

Technologia Jini pozwala na tworzenie rozproszonych środowisk, opartych o usługi, które są adaptowalne do zmian zachodzących w sieci. Podstawowym elementem systemu jest *Lookup Service* umożliwiający usługom na dołączanie do środowiska, a klientom na wyszukiwanie zarejestrowanych usług. Ważną zaletą Lookup Service jest posiadanie trzech protokołów znajdowania: „*unicast request*", „*multicast request*" oraz „*multicast announcement*". Szczególnie istotne są protokoły rozgłaszania oparte o komunikację *multicast*, co wprowadza do środowiska dużą elastyczność jako, że klienci nie muszą znać lokalizacji *Lookup Service*, a mogą dynamicznie je odnaleźć. Przyjęto, że w implementowanym systemie repozytorium agentów będzie reprezentowane poprzez *Jini Lookup Service*. Wynika z tego, że agent VGRMS będzie usługą Jini rejestrowaną w *Lookup Service*. Proces rejestracji agenta oraz wyszukania węzłów VGRMS opisany jest poniższą procedurą:

- Agent VGRMS posiada jeden lub więcej konektorów JMX (w szczególności posiada konektor RMI).
- Dla każdego konektora, który ma zostać udostępniony klientom, tworzona jest usługa Jini i rejestrowana w *Lookup Service*. Usługa taka posiada referencję (*stub*) do konektora JMX. Dodatkowo podczas rejestracji podawane są atrybuty opisujące MBean'y (np. VM o danej konfiguracji, fizyczny host posiadający wymagane zasoby, itp.), które są dostępne poprzez ten konektor. Parametry te mogą być użyte jako filtry w procesie wyszukiwania usługi lub zasobu.
- Klient VGRMS używając protokołu *multicast discovery*, odnajduje w systemie repozytoria agentów (a dokładniej *Lookup Service*) i wyszukuje w nim konektory spełniające zadane kryteria.[1]

[1] Operacja ta jest ograniczona zasięgiem ruchu *multicast* co powoduje ograniczenie możliwości wyszukiwania w obrębie klastra (sieci LAN). Wyszukiwanie globalne możliwe jest poprzez stworzenie hierarchii *Lookup Service*.

- Posiadając konektor JMX, klient może bezpośrednio łączyć się z `MBeanServer`'em, a co za tym idzie z poszczególnymi MBean'ami na danym węźle VGRMS.

6.1.2. Komponenty węzła VGRMS

Zgodnie z przyjętymi założeniami implementacyjnymi, komponenty węzła VGRMS są reprezentowane poprzez JMX MBean. W momencie inicjalizowania węzła VGRMS tworzony jest na nim `MBeanServer`, a w nim rejestrowany jest `NodeMaster`. Kolejne komponenty węzła VGRMS, które również są MBean'ami — `VM-M`, `VG-M`, są tworzone i rejestrowane w `MBeanServer` poprzez `NodeMaster`. Na węźle instalowane są dodatkowe MBean'y służące do zbierania informacji o stanie systemu — `NetMonitor`, wykorzystujący bibliotekę Jpcap[2] do analizy pakietów komunikacji sieciowej oraz `OSMonitor` pobierający informacje o stanie systemu operacyjnego i udostępniający je poprzez interfejs JMX. `OSMonitor` korzysta z danych udostępnianych przez systemy monitorujące oparte o standard CIM (ang. *Common Information Model*) [3]. Standard ten w niniejszej pracy wykorzystywany był jedynie jako jedno z możliwych źródeł informacji o systemie operacyjnym hosta. Możliwości CIM związane z kontrolą i zarządzaniem nie były wykorzystywane.

Komponent `NodeMaster`

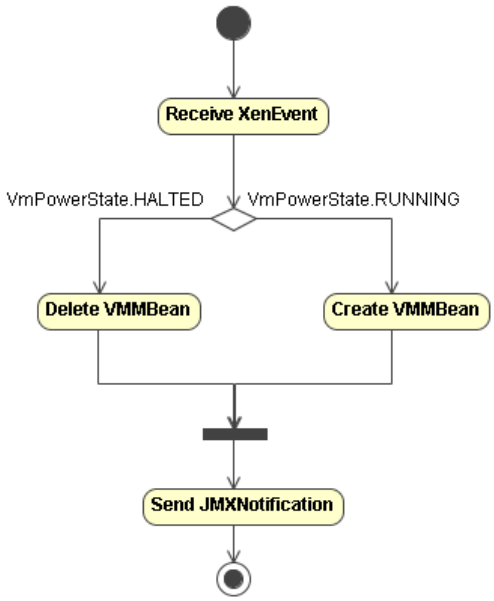
NodeMaster rejestrowany jest w `MBeanServer` pod nazwą (ang. *ObjectName* [58]) „`bean:name=nodeMaster`". Komponent ten implementuje interfejs `NotificationPublisherAware`, stając się propagatorem zdarzeń JMX. Zdarzeniami w systemie, o których powiadamia `NodeMaster` są informacją o rezultacie zadań (np. stworzenie `VM`), które wykonuje ten komponent (omówione w sekcji 5.2.3).

Podczas inicjalizacji komponentu `NodeMaster`, tworzone są komponenty pomocnicze `NotificationThread` i `VMChecker`. `NotificationThread` jest wątkiem, który oczekuje na zdarzenia przychodzące ze środowiska Xen (rejestruje się tylko na zdarzenia dotyczące klasy `com.xensources.xenapi.VM`), weryfikuje ich znaczenie i podejmuje odpowiednie działania. Implementacja Xen rozróżnia trzy rodzaje zdarzeń: `EventOperation.ADD`, `EventOperation.DEL`, `EventOperation.MOD`. Obecna implementacja Xen generuje dla klasy `VM` zdarzenia typu `EventOperation.MOD` zarówno w przypadku dodania, usunięcia czy migracji VM. Dodatkowo, migracja wirtualnej maszyny to stan, który realizowany jest poprzez dwa zdarzenia: usunięcie jej z węzła źródłowego, a następnie stworzenie na węźle docelowym. Gdy `NotificationThread` otrzyma zdarzenie ze środowiska Xen, to to jaką akcję podejmie zależy od wartości pola `VmPowerState`. Pole to przyjmuje m.in. wartości `HALTED`, `RUNNING`. Dodatkowo zdarzenie zawiera `uid` maszyny wirtualnej, która spowodowała wygenerowanie danego zdarzenia. Dwie akcje (rysunek 6.1) jakie należy wykonać, po otrzymaniu zdarzenia to stworzenie/usunięcie `VM-M`'a zarządzającego wirtualną maszyną i rejestracja/wyrejestrowanie w/z `MBeanServer`, a następnie wygenerowanie odpowiedniego zdarzenia JMX, w celu poinformowania pozostałych komponentów systemu VGRMS.

Notyfikacje JMX zawierają informację o typie operacji jaka została wykonana, `uid` maszyny wirtualnej, której dotyczy dana operacja oraz adres hosta, na którym zdarzenie

[2] Jpcap: `http://http://netresearch.ics.uci.edu/kfujii/jpcap/doc/`.

[3] W środowisku uruchomieniowym wykorzystano darmową implementację standardu CIM dostępną w pakiecie OpenPegasus (`http://www.openpegasus.org/`).



Rysunek 6.1. Przechwytywanie zdarzeń środowiska Xen i podejmowanie odpowiednich akcji przez komponenty zarządzania stanem węzła

zostało wygenerowane. Na podstawie tych informacji dowolny odbiorca (węzeł, a w szczególności aplikacja kliencka) w systemie VGRMS z łatwością zaktualizuje dane na temat stanu środowiska.

Komponent pomocniczy `VMChecker` dziedziczy z klasy `javax.swing.Timer` jako, że jego zadania są wykonywane co określony interwał czasu. Komponent ten weryfikuje:

- czy dla każdej maszyny wirtualnej (listę wirtualnych maszyn pobiera z *hypervisor* systemu Xen) środowiska Xen istnieje MBean zarządzający (sprawdza poprzez `ObjectName`, w sekcji „Komponent `VM-M`" opisano schemat generowania `ObjectName` dla `VM-M`). Jeśli nie istnieje to go tworzy i rejestruje w `MBeanServer`, a następnie generuje notyfikację JMX o pojawieniu się nowego komponentu. Sytuacja, w której MBean nie istniał może wystąpić gdy wirtualna maszyna została stworzona poza systemem VGRMS, np. poprzez polecenie tekstowej konsoli systemu Xen (komenda `xm create`),

- czy dla każdego `VM-M` istnieje wirtualna maszyna środowiska Xen. Jeśli nie istnieje, wyrejestrowywuje go z `MBeanServer`. Sytuacja taka może wystąpić, gdy wirtualna maszyna zostanie usunięta ze środowiska Xen, ale zdarzenie z tej warstwy z jakiegoś powodu zaginie i nie dotrze do warstwy zarządzania komponentami węzła VGRMS.

Powyższe operacje stanowią pewien rodzaj realizacji protokołu zapewniającego konsystentność konfiguracji i rozmieszczenia MBean'ów zarządzania VM (`VM-M`) dla aktualnego stanu wirtualnych maszyn. Konieczność implementacji tego elementu wniknęła głównie z praktycznych obserwacji zawodności komunikacji zdarzeniowej systemu JMX oraz z konieczności pozostawienia możliwości zarządzania stanem VM z poziomu konsoli tekstowej

systemu Xen[4].

Rysunek 6.2 przedstawia propagację zdarzenia o usunięciu wirtualnej maszyny. Warto zaznaczyć, że zdarzenie o usunięciu maszyny wirtualnej jest generowane dopiero w momencie faktycznego usunięcia jej z systemu Xen. Nie wystarczy usunąć np. MBean'a zarządzającego `VM-M`, bo odpowiednie mechanizmy po wykryciu braku MBean'a dla VM odtworzą dla niej MBean'a.

Komponent `VM-M`

Rysunek 6.3 przedstawia komponenty realizujące funkcjonalność `VM-M`. Na fizycznym węźle zainstalowane jest oprogramowanie realizujące techniki wirtualizacji w postaci systemu Xen. System Xen został przez twórców wyposażony w *Xen Management API* Xen-API (ang. *Xen Application Programming Interface*) — interfejs umożliwiający zdalne konfigurowanie i zarządzanie wirtualnymi systemami Grid obsługiwanymi przez oprogramowanie Xen. Interfejs ten został oparty o zdalne wywołanie metod — RPC (ang. *Remote Procedure Call*) przesyłających informacje za pomocą notacji XML (ang. *eXtensible Mark-up Language*).

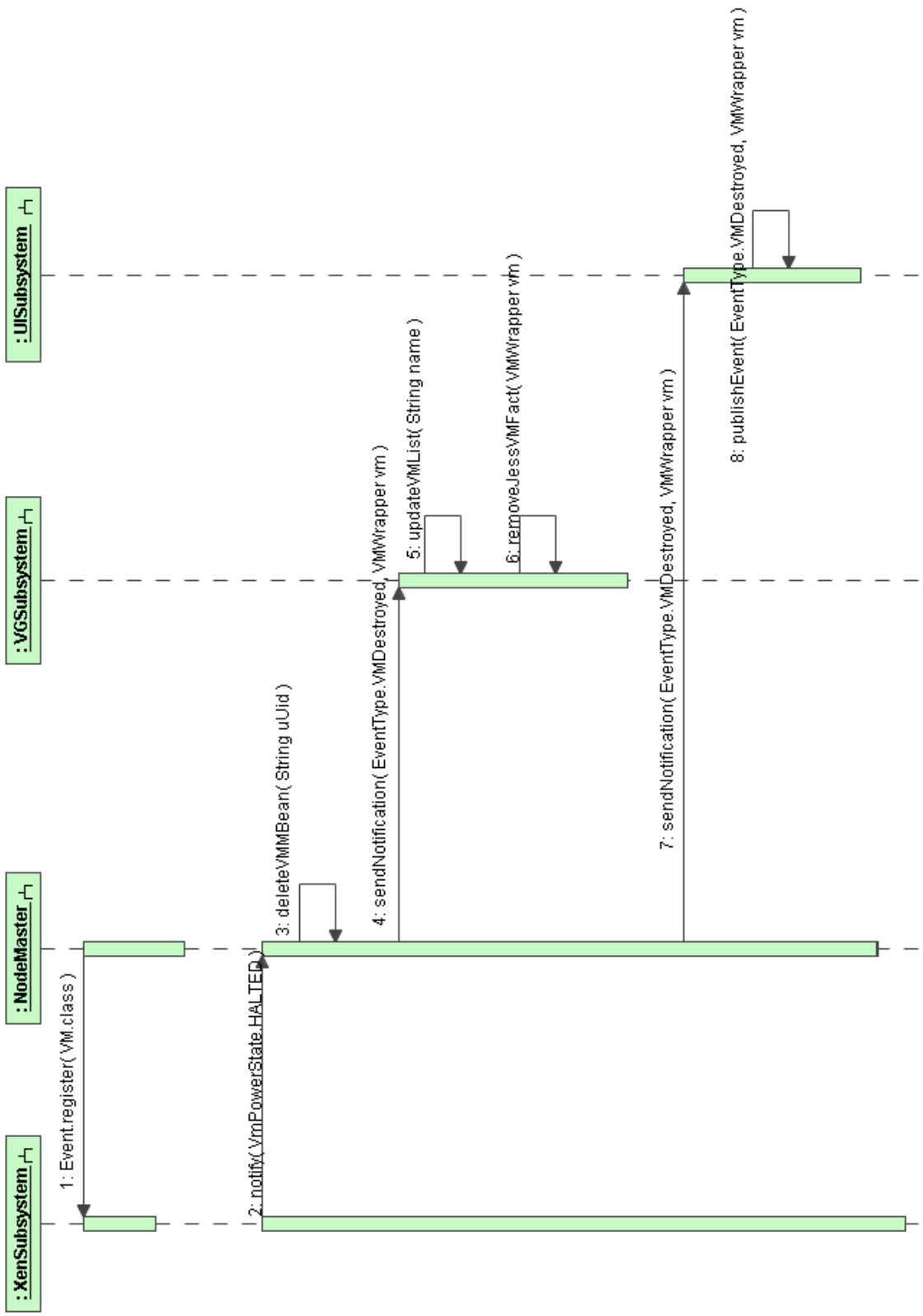
Powiązanie pomiędzy protokołem XML-RPC (ang. *XML — Remote Procedure Call*) [100] a komponentem JMX jest możliwe dzięki wykorzystaniu biblioteki obsługi tego protokołu [37], której implementacja dostarczona jest przez organizację *Apache Foundation* oraz rozwijanego[5] wraz z projektem Xen interfejsu zarządzającego [101] dla języka Java. Lista klas zdefiniowanych przez specyfikację Xen-API dostępna jest w tabeli 6.1.

Schemat operacji tworzenia maszyny wirtualnej przedstawia rysunek 6.4. Proces interakcji komponentów środowiska rozpoczyna się od rozpoczęcia sesji pomiędzy elementem `NodeMaster` a interfejsem zarządzania systemem Xen. W kroku tym następuję uwierzytelnienie oraz stworzenie sesji zarządzania protokołu XML-RPC. Klient za pomocą wywołania interfejsu zarządzającego implementowanego przez komponent `NodeMaster` tworzy wirtualny komputer. Parametry potrzebne do stworzenia instancji wirtualnej maszyny są przekazywane jako argumenty funkcji `createVM()`. Informacje te wykorzystywane są jako argumenty dla szeregu wywołań mających na celu przygotowanie środowiska działania VM. Ostatecznie, uruchomienie VM jest dokonywane za pomocą przekazanego do systemu Xen wywołania `create(vmRecord)`. Zwrócenie informacji o poprawnym zakończeniu procedury tworzenia maszyny wirtualnej powoduje rozpoczęcie procesu tworzenia i rejestrowania `VM-M`.

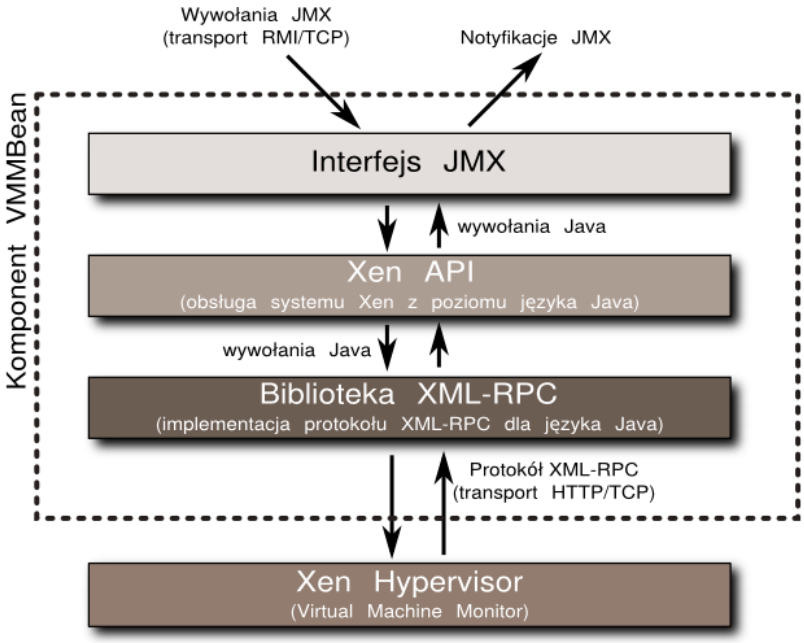
Nazwa pod jaką `VM-M` jest rejestrowany w `MBeanServer`, konstruowana jest zgodnie ze wzorcem: `"bean:name="+ uUid + "_MBean"`, gdzie UUID (ang. *Universally Unique Identifier*) jest unikalnym identyfikatorem generowanym przez Xen w momencie uruchamiania VM. Metody interfejsu (zawarte w załączniku A), które komponent ten udostępnia, w większości są wywołaniem odpowiednich metod Xen-API wraz z odpowiednią konwersją ich argumentów.

[4]Użycie konsoli systemu Xen było wymagane w sytuacjach, w których z różnych przyczyn, awarii ulegało działanie *hypervisor*a i system VGRMS tracił dla danego węzła możliwość zarządzania stanem wirtualnych maszyn.

[5]Projekt ten przestał być rozwijany około kwietnia 2007 roku (wersja API 0.9), a że względu że po tej dacie wykonano szereg prac związanych z modyfikacjami API zarządzania projektu Xen — oryginalny kod musiał być dostosowany do nowej wersji (1.0) specyfikacji.



Rysunek 6.2. Propagacja zdarzenia pomiędzy komponentami VGRMS o usunięciu z systemu wirtualnej maszyny



Rysunek 6.3. Komponenty programowe modułu `VM-M`.

Komponent `VG-M`

Tworząc wirtualny Grid z węzłów fizycznych, które wchodzą w skład wirtualnego Gridu, wybierany jest jeden, na którym zostanie stworzony komponent odpowiedzialny za zarządzanie danym VG. Jako, że reguły wyboru takiego węzła nie są istotne z punktu widzenia wydajności działania systemu, jest on wybierany drogą losową. Można by jednak zdefiniować bardziej inteligentne mechanizmy wyznaczania węzła-zarządcy biorąc pod uwagę, np. parametry fizyczne hosta, przepustowość komunikacji do pozostałych węzłów, itp.

Po wybraniu węzła-zarządcy, następuje wywołanie metody `createVG` komponentu `NodeMaster`, której rezultatem jest m.in. stworzenie i zarejestrowanie `VG-M`. Dodatkowo tworzony jest plik konfiguracyjny (6.1) zawierający informacje o początkowej topologii sieciowej[6] (definiującej jedynie użyteczne interfejsy sieciowe bez informacji o połączeniach).

Metodę `createVG` kończy oczywiście wygenerowanie odpowiedniej notyfikacji JMX o fakcie stworzenia nowego VG.

Z chwilą stworzenia `VG-M`, komponent ten przejmuje funkcje obsługi wirtualnego Gridu. `NodeMaster` może jedynie usunąć wirtualny Grid (metoda `removeVG`), bądź zwrócić listę aktualnych VG zainstalowanych na danym węźle (metoda `getVGs`).

`VG-M` posiadając listę węzłów wchodzących w jego skład (maszyny wirtualne tworzące ten VG, na pewno znajdują się na tych węzłach) musi być odbiorcą zdarzeń JMX generowanych przez zarządców `NodeMaster` tych węzłów. Wynika to z faktu, iż musi być informowany np. o migracji wirtualnej maszyny wchodzącej w skład danego wirtualnego Gridu.

[6]Topologia ta może być zmieniona za pomocą graficznego narzędzia do edycji połączeń wirtualnej sieci, którego rezultatem jest modyfikacja konfiguracji topologii początkowej zapisana w formacie XML.

Nazwa klasy	Opis
session	Sesja zarządzania systemem Xen
task	Zadania asynchroniczne o długich czasach trwania
event	Rejestracja i obsługa zdarzeń asynchronicznych
pool	Obsługa grupowania fizycznych hostów
VM	Maszyna wirtualna
VM_metrics	Metryki i parametry skojarzone z VM
VM_guest_metrics	Informacje zgłaszane przez VM
host	Fizyczny host
host_crashdump	Informacje o awariach fizycznego węzła
host_metrics	Metryki i parametry skojarzone z hostem
host_cpu	Fizyczne procesory
network	Wirtualna sieć
VIF	Interfejs sieci wirtualnej
VIF_metrics	Metryki interfejsów wirtualnych
PIF	Interfejs fizyczny
PIF_metrics	Metryki interfejsów fizycznych
SM	Zarządzanie repozytorium obrazów
SR	Repozytorium obrazów
VDI	Obraz wirtualnego dysku
VBD	Wirtualne urządzenie blokowe
VBD_metrics	Metryki skojarzone z klasą VBD
PBD	Urządzenia blokowe fizycznego hosta
crashdump	Informacje dostępne po awaryjnym zamknięciu VM
VTPM	Wirtualne urządzenie TPM
console	Konsola administracyjna
user	Zarządzanie użytkownikami
debug	Informacje diagnostyczne dla testów

Tabela 6.1. Klasy interfejsu zarządzania systemem Xen

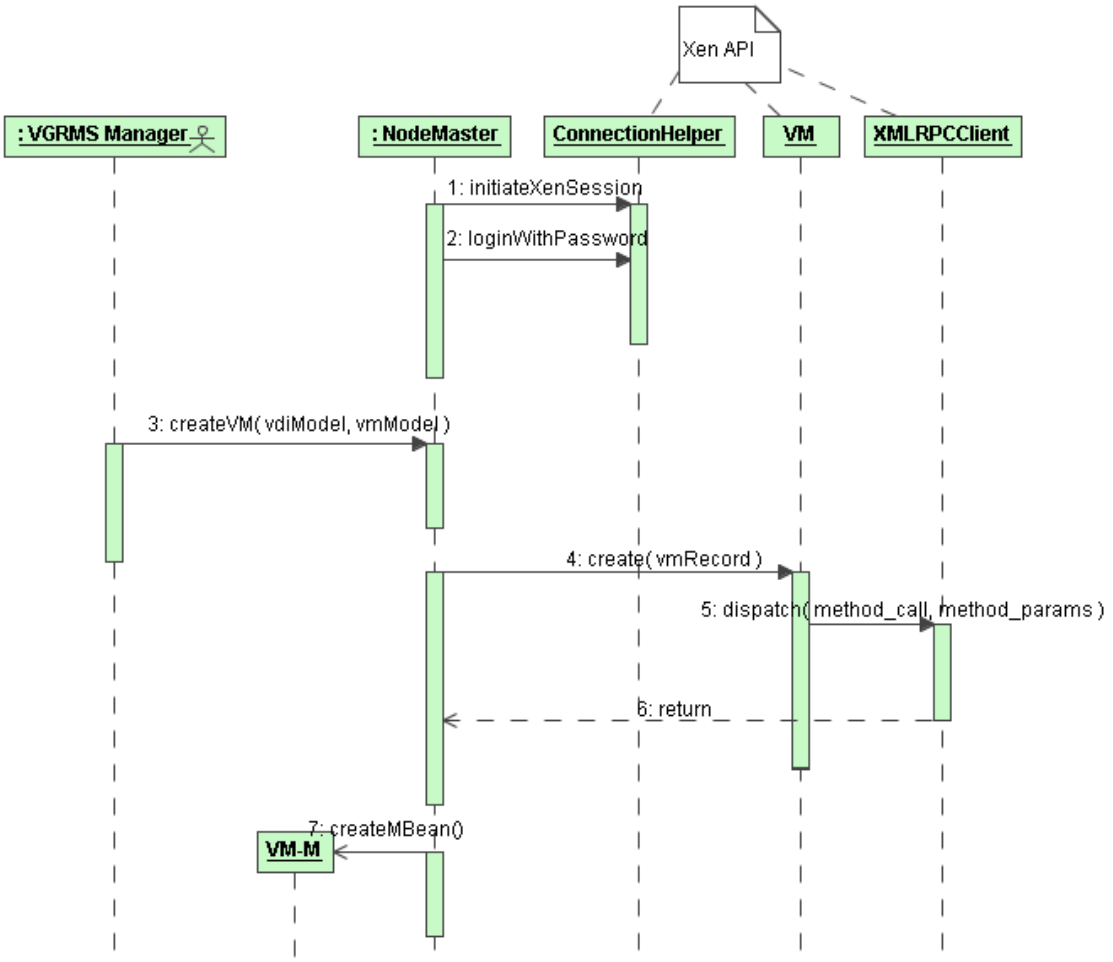
Silnik regułowy — PolicyEngine

Silnik regułowy, czyli moduł implementujący logikę działania środowiska został zaimplementowany w oparciu o komercyjną[7] implementację specyfikacji JSR (ang. *Java Specification Request*): JSR-94 („*Java*^{TM} *Rule Engine API* " — system regułowy dla platformy Java). Jess jest to silnik regułowy (ang. *rule engine*) oraz środowisko skryptowe[8] stworzone w oparciu o platformę Java.

Platforma Jess została wyposażona w rozbudowany język budowy i reprezentacji reguł (wzorowany na języku Lisp), a całe środowisko jest zwartą, lekką i jedną z najwydajniejszych [1, 102] implementacji silnika regułowego dla platformy Java. Język skryptowy Jess pozwala na bezpośredni dostęp do API języka Java, umożliwiając łatwą integrację zarówno

[7] Twórcy oprogramowania dla zastosowań akademickich wymagają jedynie podpisania licencji bez konieczności wnoszenia opłat.

[8] Oprogramowanie to opracowane zostało przez Sandia National Laboratories, Livermore, CA [56] (firmę należącą do konsorcjum Lockheed Martin).



Rysunek 6.4. Tworzenie maszyny wirtualnej przez `NodeMaster`.

z pozostałą częścią aplikacji jak i dostęp do bibliotek Java zawierających implementację wymaganej funkcjonalności[9].

Implementacja Jess wykorzystuje do przetwarzania reguł rozszerzoną wersję algorytmu Rete[32]. Algorytm ten jest podstawą większości silników regułowych, gdyż umożliwia efektywne rozwiązywanie problemu dopasowania typu wiele-do-wielu[32]. Rozszerzenia Jess dotyczą możliwości stosowania dopasowania wstecznego[10], API filtrów dla przeszukiwania bazy wiedzy oraz bezpośredni dostęp do metod i atrybutów klas Java. Język Jess pozwala na tworzenie obiektów Java, bezpośrednie wywoływanie ich metod oraz na implementację interfejsów bez konieczności kompilacji kodu języka Java.

Zastosowanie silnika regułowego w implementowanym module systemu podyktowane było koniecznością zapewnienia możliwie najbardziej elastycznego mechanizmu implemen-

[9]Przykładowo obsługa złożonych struktur danych, pakiety statystyczne, obsługa czasu, interfejs GUI (ang. *Graphical User Interface*), itp.

[10]Dopasowanie wsteczne (ang. *backwards chaining*) umożliwia poszukiwanie i tworzenie przesłanek (faktów) wymaganych do spełnienia warunków dla danej reguły.

```
<?xml version="1.0" encoding="utf-8" ?>
<topology>
  <settings fileType="vgrid-network" />
  <devices>
    <device id="1" name="VM1" type="FIREWALL" visible="true">
      <description>Provides access to outside network</description>
      <interfaces>
        <ifc>eth0</ifc>
        <ifc>eth1</ifc>
      </interfaces>
      (...)
    </device>

    <device id="2" name="VM2" type="WORKERNODE" visible="true">
      <description>Worker Node 1 (Scientific Linux 4.3)</description>
      <interfaces>
        <ifc>eth0</ifc>
      </interfaces>
      (...)
    </device>

    (...)
  </devices>
</topology>
```

Kod źródłowy 6.1. Początkowa konfiguracja topologii sieciowej wirtualnego Gridu

tującego wybrany algorytm zarządzania zasobami. Koncepcja reguł pozwala na osiągnięcie następujących założeń funkcjonalnych:

- oddzielenie logiki działania systemu od jego implementacji,
- możliwość zmiany algorytmu zarządzania oraz jego parametrów bez konieczności ponownej kompilacji kodu źródłowego aplikacji,
- istniejące reguły realizujące daną strategię optymalizacji nie muszą być modyfikowane przy rozszerzaniu ich zestawu.

Silnik regułowy jest obiektem powiązanym z wirtualnym Gridem, dlatego przyjęto, że nie będzie dla niego tworzony dodatkowy MBean, a jego funkcjonalność będzie zawarta w zdalnym interfejsie `VG-M`. `VG-M` będzie je zaś przekazywał do odpowiednich metod komponentu `PolicyEngine`.

Na rysunku 5.19 (rozdział 5), będącym diagramem sekwencji tworzenia wirtualnego gridu, pokazano dwie metody wywoływane na `PolicyEngine` w trakcie tworzenia wirtualnego Gridu: `addHostFacts`, `addVMFacts`. Specyfika implementowanego systemu wymusza aby fakty miały możliwość komunikacji z odpowiednimi MBean'ami, z których pobierają wartości. Listing 6.2 zawiera konstruktor tworzący fakt (w tym przypadku `VMFact`, analogicznie tworzony jest `HostFact`). To, z którym MBean'em fakt ma się łączyć wynika ze schematu tworzenia `JMXServiceURL` oraz `ObjectName`.

Ponieważ silnik regułowy w przypadku dodawania polityk zarządzania (metodą `addRule`) i przetwarzania ich musi operować na zaktualizowanych faktach, wprowadzono

```
public JessVMFact(VMWrapper vm) {
    super(vm.getName(), vm.getHostName());
    setUUid(vm.getUUid());

    try {
        masterObjectName = new ObjectName("bean:name=nodeMaster");
    } catch (Exception e) {
        e.printStackTrace();
    }
    connect();
}

private void connect() {
    try {
        serviceURL = new JMXServiceURL("service:jmx:rmi://" + hostName
        objectName = new ObjectName("bean:name=" + uUid + "\_MBean");
          + "/jndi/rmi://" + hostName + ":" + port + "/xen");

        connector = JMXConnectorFactory.connect(serviceURL);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Kod źródłowy 6.2. Konstruktor tworzący fakt — `VMFact`

mechanizm odświeżania kolejnych parametrów w oparciu o interwał czasu, którego wartość ustala administrator z poziomu konsoli zarządzającej. Czyli co określony interwał, wszystkie parametry faktu (np. `VCPUsUtilisation`, `VIFsReadUtilisation`, itd. (rysunek 5.16) są odświeżane poprzez wywołanie odpowiednich metod na `VM-M` (z poziomu silnika regułowego Jess). Niestety nie jest możliwe, aby w tym systemie wykorzystane fakty były notyfikowane o zmianie wartości parametrów (mimo, iż Jess posiada wsparcie w postaci `PropertyChangeListener`) i dopiero wtedy gdy taka sytuacja zaistnieje, odpowiednie polityki były przeliczane. Wynika to ze specyfiki monitorowanych zasobów [108], w których zmiany są ciągłe w czasie (np. obciążenie procesora, ilość wysłanych pakietów, wykorzystanie pamięci operacyjnej, itp.) w odróżnieniu od zdarzeń dyskretnych.

Poza metodami dodającymi fakty (`addVMFacts`, `addHostFacts`), `PolicyEngine` posiada metody usuwające fakty: `removeVMFact(VMWrapper vm)`, `removeHostFact(String hostName)`. Metody te są wykonywane po otrzymaniu z `NodeMaster` notyfikacji JMX dotyczącej usunięcia maszyny wirtualnej z sytemu Xen lub fizycznego hosta z konfiguracji wirtualnego Gridu.

Za tworzenie `VGFact` opisującego stan i parametry wirtualnego Gridu odpowiedzialna jest konsola administracyjna systemu VGRMS. Fakty te tworzone są automatycznie po stworzeniu wirtualnego Gridu i są używane przy implementacji polityki zarządzania na poziomie wirtualnej organizacji.

Cyklem życia pozostałych faktów (`OSMonitorFact` i `NetMonitorFact`)[11] zarządzają reguły silnika regułowego.

[11] Implementacja Jess umożliwia dynamiczne tworzenie obiektów Java i instancjowanie ich w postaci faktów.

Repozytorium polityk zarządzania

Repozytorium polityk jest reprezentowane w postaci JMX MBean, zainstalowanym w dowolnej lokalizacji (w systemie wymagana jest minimum jedna instancja tego komponentu). `PolicyRepository` implementuje interfejs `NotificationPublisherAware`, stając się propagatorem zdarzeń JMX. Zdarzenia te są istotne dla aplikacji klienckich, tak aby administratorzy mieli dostęp do aktualnej listy zdefiniowanych polityk. Polityki zarządzania, czyli reguły Jess oraz skrypty np. języka jython, są przechowywane w repozytorium w systemie plików. Każda reguła czy skrypt są reprezentowane w postaci pojedynczego pliku XML. Listing 6.3 przedstawia przykładową regułę zapisaną w repozytorium.

```
<org.dsrg.rule.jess.JessRule>
  <name>set-default-weigts-and-caps</name>
  <salience></salience>
  <depends-on></depends-on>
  <conflicts-with></conflicts-with>
  <description>Test 1 (QoS)</description>
  <conditions>(JessVMFact (name ?vm))</conditions>
  <actions>(assert (vcpu-params (vm ?vm) (weight 256) (cap 0)))</actions>
</org.dsrg.rule.jess.JessRule>
```

Kod źródłowy 6.3. Przykładowa postać reguły zapisana w repozytorium polityk zarządzania

6.1.3. Zarządzanie wirtualną topologią sieciową

Tworzenie topologii sieciowej rozpoczyna się w momencie stworzenia komponentu `VG-M`, który to odpowiedzialny jest za zarządzanie wirtualną siecią. Tworzenie kształtu topologii sieciowej, zarządzanie jej konfiguracją poprzez definiowanie np. ograniczeń przepustowości, w systemie Linux odbywa się za pomocą zestawu poleceń systemowych[12], dokonujących odpowiednich zmian[13] w strukturach jądra.

Przechowywana w konfiguracji `VG-M` definicja topologii sieciowej jest odwzorowana do sekwencji poleceń systemowych. Komponent `VG-M` ma zaimplementowaną logikę „transformacji" konfiguracji z formatu XML (kod źródłowy 6.4) do sekwencji poleceń systemowych z odpowiednimi argumentami. Wykonanie tych poleceń w obrębie fizycznych węzłów wchodzących w skład VG spowoduje stworzenie wirtualnej sieci. Kod źródłowy 6.5 zawiera przykładową listę poleceń tworzących wirtualną sieć (z wykorzystaniem technologii przełączania) składająca się z dwóch wirtualnych maszyn wykonywanych na dwóch różnych fizycznych węzłach połączonych siecią LAN.

Jako technika budowania sieci wirtualnej obejmującej zasięgiem sieć składająca się z maszyn fizycznych znajdujących się w różnych sieciach LAN została wykorzystana implementacja wirtualnego przełącznika Ethernet stworzona w ramach projektu[14] VDE. Polega to na tworzeniu na węzłach wirtualnych *switch*y, które będą łączone za pomocą tuneli[15].

[12] Takich jak np. polecenia `ifconfig`, `route`, `brctl` czy `tc`.

[13] Niektóre ustawienia mogą być modyfikowane za pomocą zmian zawartości plików specjalnych w systemie plików `/proc`.

[14] VDE: `http://sourceforge.net/projects/vde/`.

[15] Rodzaj tunelowania ruchu jest dowolny, gdyż *switch* VDE komunikuje się z oprogramowaniem tunelującym poprzez standardowe wejście/wyjście.

```
(...)
<device id="1" name="vg1-vm1" type="WORKERNODE" visible="true">
    <interfaces>
        <ifc>veth2.0</ifc>
    </interfaces>
    <connections>
        <conn to="1" on="veth2.0" type="BMA" label="" x="3" y="2"/>
    </connections>
</device>

<device id="2" name="vg1-vm2" type="WORKERNODE" visible="true">
    <interfaces>
        <ifc>veth3.0</ifc>
    </interfaces>
    <connections>
        <conn to="2" on="veth3.0" type="BMA" label="" x="11" y="6"/>
    </connections>
</device>
(...)
```

Kod źródłowy 6.4. Przykładowa topologia sieciowa przechowywana przez komponent VG-M

```
# host 1
brctl addbr vg1-vm1-vg1-vm2
brctl addif vg1-vm1-vg1-vm2 veth2.0  # logiczny interfejs VM1
brctl addif vg1-vm1-vg1-vm2 eth0     # fizyczny interfejs hosta

# host 2
brctl addbr vg1-vm1-vg1-vm2
brctl addif vg1-vm1-vg1-vm2 veth3.0  # logiczny interfejs VM2
brctl addif vg1-vm1-vg1-vm2 eth0     # fizyczny interfejs hosta
```

Kod źródłowy 6.5. Przykładowe polecenia tworzenia wirtualnej sieci zlecane przez komponent VG-M

Konfigurację poleceń tworzących wirtualną sieć w przypadku gdy maszyny fizyczne znajdują się w różnych sieciach LAN przedstawia kod źródłowy 6.6.

Obsługa migracji VM powoduje, iż zmianie ulega kształt topologii fizycznej wykorzystywanej do przesyłania ruchu dla sieci wirtualnej. Migracja wykonania VM wymaga jedynie aby w konfiguracji sieci docelowego hosta fizycznego dostępny był wirtualny *switch* reprezentujący wirtualną sieć LAN, do której należy przenoszona maszyna. Aktualizacja konfiguracji sieci po (lub w trakcie) migracji wymaga dodania logicznego interfejsu sieciowego do listy interfejsów wirtualnego *switch*'a (linia 5 w przykładzie 6.6) i usunięcia tego interfejsu z konfiguracji odpowiedniego *switch*'a na hoście źródłowym (co robione jest automatycznie po usunięciu VM).

Istnieje jeszcze określony zestaw poleceń powodujących usunięcie konfiguracji wirtualnej sieci oraz powodujący zwolnienie zasobów sieciowych. Polecenia te są wykonywane podczas modyfikacji topologii wirtualnej sieci lub po usunięciu VG.

```
# host 1
vde_switch -tap tap0 -daemon
brctl add vg1-vm1-vg1-vm2
brctl addif vg1-vm1-vg1-vm2 tap0
brctl addif vg1-vm1-vg1-vm2 veth2.0  # dostep VM1 do switcha VDE
vde_plug "dpipe vde_plug = ncat -u -e vde_plug host1 4567"  # tunel UDP do
    hosta 2
```

Kod źródłowy 6.6. Polecenia tworzenia wirtualnej sieci z zastosowaniem tunelowania komunikacji

6.2. Graficzny klient systemu VGRMS

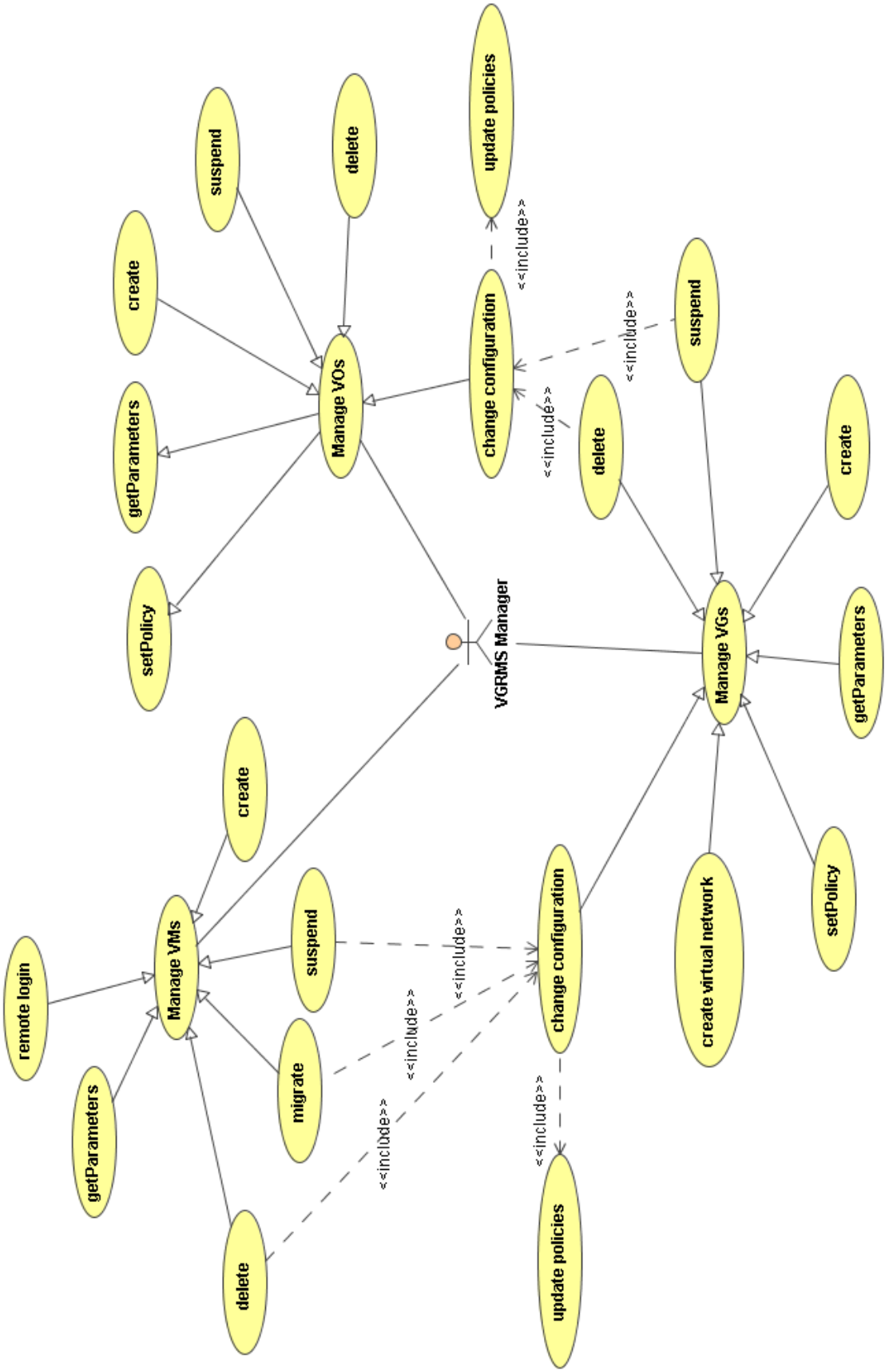
Konsola graficzna systemu VGRMS powstała celem uproszczenia zarządzania systemem VGRMS. Konsola ta umożliwia modyfikację polityk działania, śledzenie wykonania aplikacji, dostęp do danych z monitorowania poziomu wykorzystania zasobów. Rysunek 6.5 przedstawia możliwe operacje wykonywane za pomocą konsoli administracyjnej.
Konsola graficzna (rysunek 6.6) przeznaczona do administrowania systemem VGRMS została podzielona na dwie części:

1. Sekcja szybkiego dostępu znajdująca się po lewej stronie przedstawia widoki w postaci drzewa zależności. Z każdym elementem drzewa związane jest menu kontekstowe, które umożliwia wykonanie akcji na danym elemencie (np. dla VM menu zawiera pozycje: przenieś na inny węzeł, zmień typ, tp.). Dostępne w tej części widoki to:

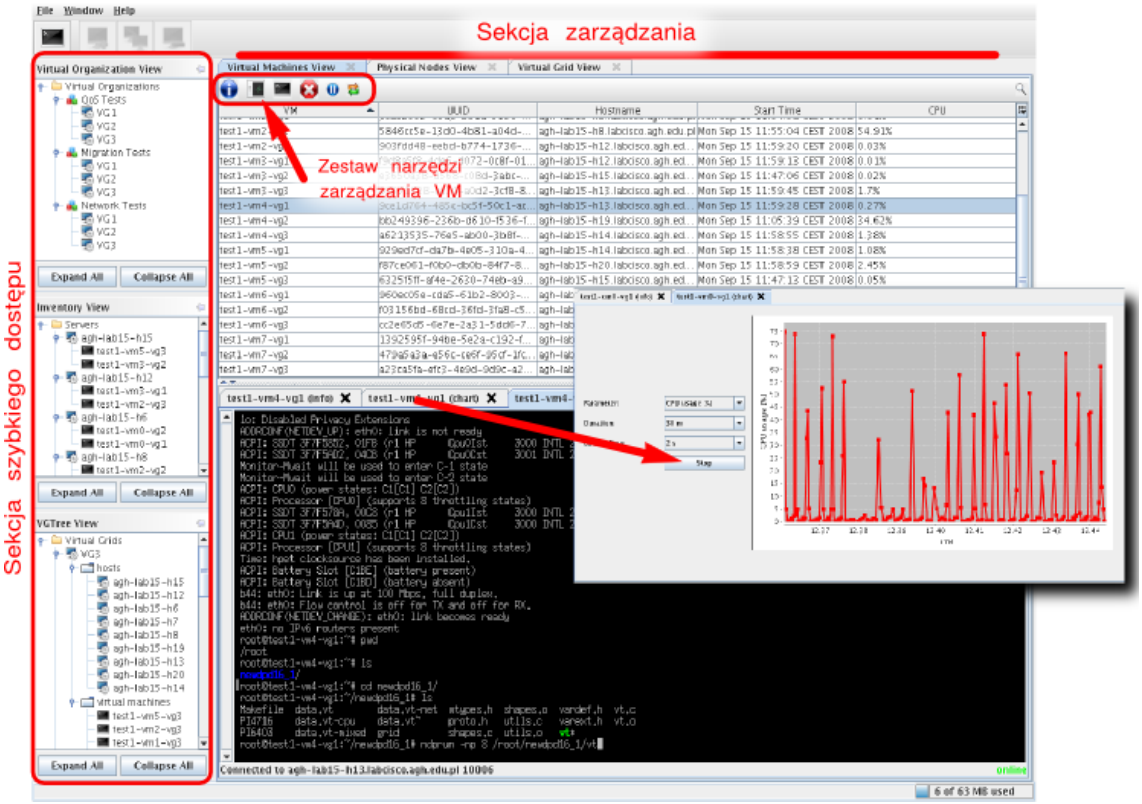
   - widok wirtualnych organizacji — hierarchiczne powiązanie pomiędzy instancjami wirtualnych Gridów a wirtualną organizacją do której należą,
   - widok fizycznych węzłów — zawiera wszystkie dostępne węzły, w liściach których znajdują się zainstalowane na nich wirtualne maszyny,
   - widok wirtualnych Gridów — zawiera wszystkie dostępne wirtualne Gridy, w liściach których jest gałąź dotycząca listy węzłów oraz gałąź dotycząca listy wirtualnych maszyn.

2. Sekcja zarządzania znajdująca się po prawej stronie okna aplikacji przedstawia widoki w postaci tabelarycznej. Dodatkowo nad każdą tabelą znajduje się pasek narzędzi zapewniający łatwy dostęp do funkcjonalności danych komponentów. W przypadku gdy funkcjonalność wymaga otworzenia dodatkowego okna (np. konsola do zarządzania, wykresy, itp.) są one agregowane w postaci zakładek tuż pod tabelą. Dostępne w tej części widoki to:

   - widok wirtualnych maszyn (rysunek 6.6) — domyślnie tabela podzielona została na trzy kolumny, wyświetlając dane o nazwie VM, nazwie węzła, na którym się znajduje oraz sumarycznym obciążeniu CPU maszyny wirtualnej. Użytkownik ma jednak możliwość dodania większej ilości kolumn (np. aktualnie zajmowana pamięć). Dostępny pasek narzędzi zawiera przyciski umożliwiające (kolejno) na zaznaczonej wirtualnej maszynie wykonanie:



Rysunek 6.5. Diagram możliwych operacji zarządzania dostępnych z poziomu konsoli administracyjnej

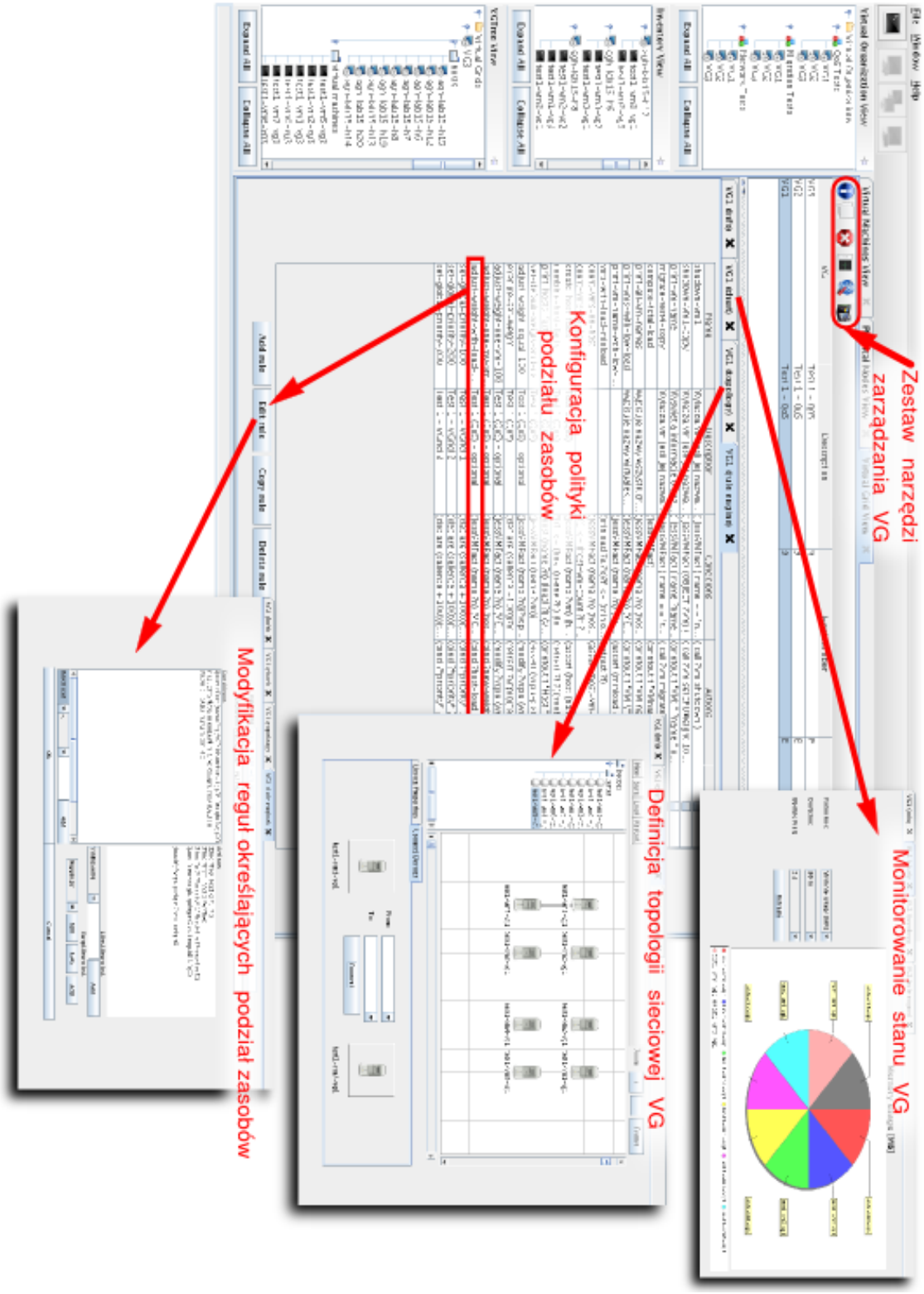


Rysunek 6.6. Konsola systemu VGRMS — zarządzanie wirtualnymi maszynami

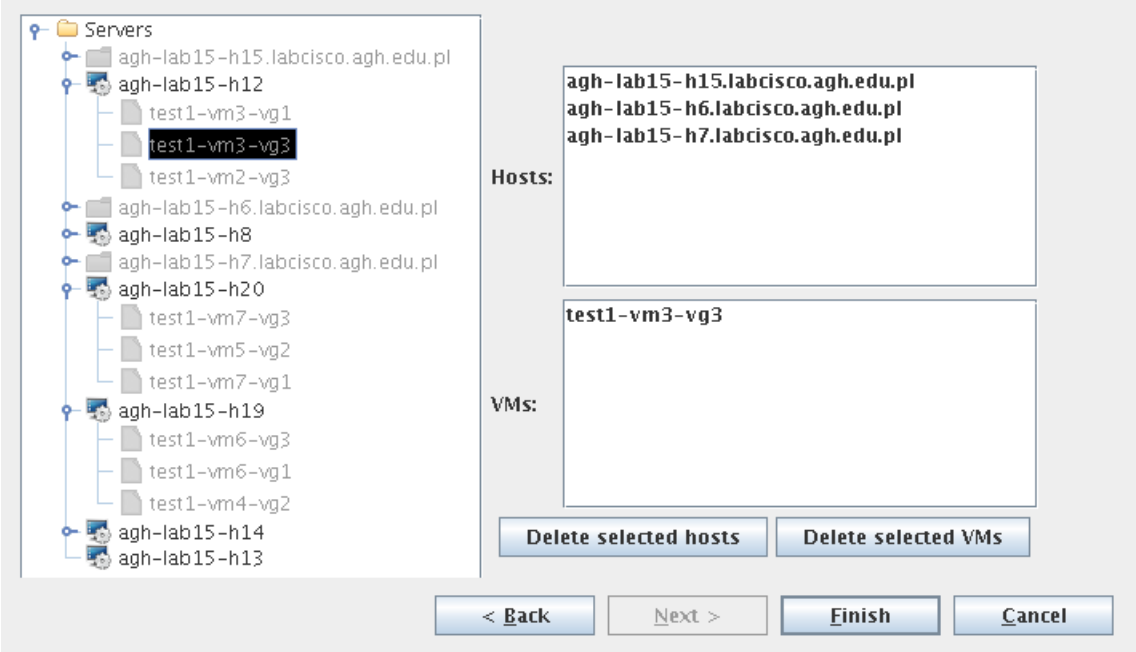
  - uzyskanie szczegółowych informacji na temat instancji VM,
  - pokazanie wykresów związanych z różnymi parametrami działania VM. Wykresy te są na bieżąco aktualizowane, a próbkowanie odbywa się z określoną przez użytkownika częstotliwością,
  - dostęp do tekstowej konsoli systemowej za pomocą protokołów telnet lub SSH,
  - usunięcie VM,
  - zatrzymanie VM,
  - restart VM,
- widok fizycznych węzłów — domyślnie tabela podzielona została na trzy kolumny (użytkownik ma możliwość dodawania nowych kolumn), wyświetlając dane o nazwie węzła, ilości zainstalowanych wirtualnych maszyn oraz sumaryczne obciążenie zasobów wnoszone przez maszyny wirtualne. Dostępny pasek narzędzi zawiera przyciski umożliwiające (kolejno) na zaznaczonym fizycznym węźle wykonanie:
  - uzyskanie szczegółowych informacji,
  - pokazanie wykresów związanych z różnymi parametrami. Wykresy te są na bieżąco aktualizowane, a próbkowanie odbywa się z określoną przez użytkownika częstotliwością,

  - dostęp do konsoli systemowej za pomocą protokołów telnet lub SSH,

- widok wirtualnych Gridów (rysunek 6.7) — domyślnie tabela podzielona została na cztery kolumny (użytkownik ma możliwość dodawania kolumn), wyświetlając dane o nazwie wirtualnego Gridu, jego opis, ilość węzłów wchodzących w jego skład oraz ilość wirtualnych maszyn, która do niego należy. Dostępny pasek narzędzi zawiera przyciski umożliwiające (kolejno) na zaznaczonym wirtualnym Gridzie wykonanie:
  - uzyskanie szczegółowych informacji,
  - stworzenie nowego wirtualnego Gridu. Metodą przeciągnij-i-upuść użytkownik wybiera komponenty (maszyny wirtualne i fizyczne węzły) wchodzące w jego skład. Panel wyboru węzłów fizycznych i wirtualnych maszyn przedstawia rysunek 6.8. Niektóre elementy drzewa po lewej stronie są nieaktywne. Są te elementy, których nie można wybrać. W praktyce oznacza to, iż elementy te zostały już wybrane i przeniesione do bieżącej konfiguracji, bądź w przypadku wirtualnych maszyn może oznaczać to, iż wykorzystane zostały w konfiguracji innego wirtualnego Gridu. Dla fizycznych hostów nieaktywne są też te, których parametry nie spełniają minimalnych wymagań określonych w konfiguracji VG,
  - zatrzymanie VG,
  - pokazanie wykresów związanych z różnymi parametrami. Wykresy te są na bieżąco aktualizowane, a próbkowanie odbywa się z określoną przez użytkownika częstotliwością. Wykresy generowane są w odniesieniu do poziomu zasobów dostępnych w ramach całego wirtualnego Gridu,
  - tworzenie topologii sieciowej przy użyciu interfejsu graficznego pozwalający w sposób wizualny stworzyć topologię sieciową wirtualnego Gridu (wyszczególniona na rysunku 6.7).
    Topologia sieciowa jest tworzona za pomocą interfejsu pozwalającego na graficzne łączenie dostępnych w ramach danego wirtualnego Gridu elementów (wirtualnych maszyn). Użyte symbole dla oznaczenia funkcji danej VM (router, firewall, switch, itp.) wybierane są na podstawie określonego wcześniej typu VM (kod źródłowy A.3). Przy definiowaniu połączenia administrator ma możliwość wyboru interfejsów VM, które zostaną połączone łączem typu punkt-punkt. Tworzenie sieci LAN wymaga zatem dołączenia za pomocą kilku połączeń do posiadającej wiele interfejsów maszyny wirtualnej pełniącej funkcję *switch*'a.
    Możliwe jest późniejsze, ręczne doprecyzowanie konfiguracji sieciowej wirtualnego Gridu poprzez możliwość połączenia się za pomocą konsoli systemowej (powłoki) i wydanie odpowiednich poleceń.
  - uruchomienie konsoli zarządzania silnikiem polityk. W tym panelu administrator uzyskuje dostęp do repozytorium polityk oraz do konfiguracji silnika regułowego w danym wirtualnym Gridzie. Konsola widoczna jest na rysunku 6.7 w aktywnej zakładce sekcji zarządzania.
    Reguły dostępne w konfiguracji `PolicyRepository` są przedstawione w postaci tabelarycznej. Administrator może po uprzednim wybraniu aktywnej reguły dokonać jej modyfikacji a także kontrolować jej wykorzystanie w sil-



Rysunek 6.7. Konsola systemu VGRMS — zarządzanie wirtualnymi Gridami



Rysunek 6.8. Tworzenie wirtualnego Gridu za pomocą kreatora konfiguracji

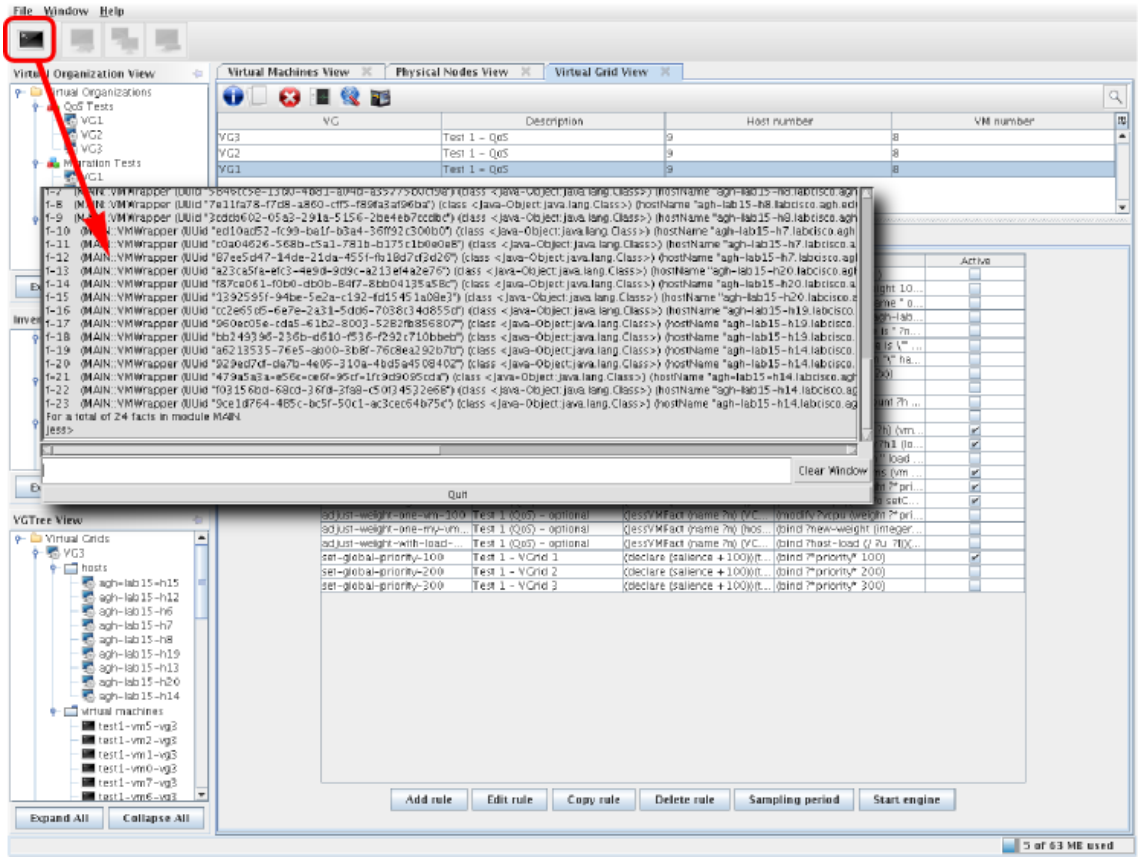
nikach regułowych (`VG-PE`) poszczególnych wirtualnych Gridów. Lista dostępnych reguł jest aktualizowana na podstawie notyfikacji generowanych przez komponent `PolicyRepository`.

Konsola graficzna systemu VGRMS pozwala także na grupowanie wirtualnych Gridów poprzez stworzenie VO oraz zarządzanie polityką przydziału zasobów na poziomie tak określonej wirtualnej organizacji. Po stronie konsoli zarządzania VGRMS dostępny jest dodatkowy komponent zarządzania czyli globalny Silnik Regułowy GPE (ang. *Global Policy Engine*) (konsola ta została przedstawiona na rysunku 6.9). Silnik ten jest dostępny dla administratora, który ma możliwość konstrukcji reguł posiadających dostęp do informacji o stanie wszystkich elementów systemu VGRMS. Możliwe jest również, dzięki bezpośredniemu dostępowi z poziomu silnika do klas wykorzystywanych do obsługi interfejsu graficznego, opracowanie wysokopoziomowej logiki zarządzania GUI jak np. wyświetlanie okien z ostrzeżeniami po wystąpieniu określonych warunków, wyłączanie okien dla nie aktywnych elementów itp.

Istotną funkcjonalnością konsoli edycji polityki VO jest możliwość wpływania na konfigurację reguł działania silników regułowych wszystkich wirtualnych Gridów. Możliwe jest to poprzez wykonywanie operacji modyfikacji reguł i zmiennych, dodawanie definicji zmiennych, itp. poprzez zdalne wywołanie odpowiednich metod komponentów `PolicyEngine` znajdujących się na poszczególnych wirtualnych Gridach.

6.3. Organizacja implementacji

Oprócz omówionych w rozdziale 6.1 podstawowych technologii użytych do implementacji systemu VGRMS, mających zdecydowany wpływ na funkcjonalność systemu, wykorzystano



Rysunek 6.9. Konsola zarządzania polityką przydziału zasobów w ramach VO

dodatkowe biblioteki wspierające proces tworzenia oprogramowania. Do implementacji komponentów aplikacji wykorzystano szkielet (ang. *framework*) Spring[16]. Jedno z jego założeń to możliwość tworzenia aplikacji przy użyciu plików konfiguracyjnych, z wykorzystaniem technik odwrócenia sterowania IoC (ang. *Inversion of Control*) [38] a w szczególności wstrzykiwania zależności DI (ang. *Dependancy Injection*) [38], co znacznie ułatwia proces tworzenia aplikacji, a szczególnie tych fragmentów, które są powtarzalne, bądź jasno zdefiniowane.

Po stronie serwerowej Spring został wykorzystany między innymi do integracji z JMX. Poniżej znajduje się uproszczony kod źródłowy (bez dodatkowych parametrów inicjalizacyjnych bean'ów) fragmentu pliku konfiguracyjnego tworzącego infrastrukturę JMX dla części serwerowej systemu VGRMS.
Uproszczenia jakie zostały wniesione w powyższym przykładzie dzięki zastosowaniu Spring'a to:

- automatyzacja tworzenia komponentu `MBeanServer` (linie 1–2),
- automatyczna rejestracja dowolnego bean'a jako JMX MBean. Linia 3 to stworzenie komponentu nodeMaster, natomiast linie 5–12 powodują eksport komponentu do

[16]Spring: `http://www.springframework.org/`.

```
1  <bean id="mbeanServer"
       class="org.springframework.jmx.support.MBeanServerFactoryBean" />
2
3  <bean id="nodeMaster" class="org.dsrg.xen.NodeMaster">
4
5  <bean id="exporter" class="org.springframework.jmx.export.MBeanExporter"
       lazy-init="false">
6      <property name="beans">
7          <map>
8              <entry key="bean:name=nodeMaster" value-ref="nodeMaster"/>
9          </map>
10     </property>
11    <property name="server" ref="mbeanServer"/>
12 </bean>
13
14 <bean id="registry"
       class="org.springframework.remoting.rmi.RmiRegistryFactoryBean">
15     <property name="port" value="13132"/>
16 </bean>
17
18 <bean id="serverConnector"
       class="org.springframework.jmx.support.ConnectorServerFactoryBean">
19     <property name="objectName" value="connector:name=rmi"/>
20     <property name="serviceUrl"
           value="service:jmx:rmi://${HNAME}/jndi/rmi://${HNAME}:13132/xen"/>
21 </bean>
```

Kod źródłowy 6.7. Konfiguracja części serwerowej systemu VGRMS dla infrastruktury JMX

JMX poprzez rejestrację w `MBeanServer`,

- aby udostępnić MBean'y poprzez zdalne konektory JSR-160 wystarczy odpowiednia deklaracja w pliku konfiguracyjnym. Linie 14–16 to utworzenie rejestru RMI (`rmiregistry`), w liniach 18–21 zostaje utworzony konektor RMI, który w trakcie wykonania aplikacji zostaje uruchomiony i wyeksponowany.

Również po stronie klienckiej wykorzystano zalety Spring'a, a szczególnie podprojektu spring-richclient (RCP)[17], umożliwiającego łatwe tworzenie zaawansowanych interfejsów graficznych przy użyciu plików konfiguracyjnych. Spring RCP posiada wsparcie dla aplikacji składających się z wielu widoków (w przypadku systemu VGRMS są to m.in. widoki: dostępnych węzłów — `InventoryView`, dostępnych wirtualnych gridów — `VGView`, dostępnych wirtualnych organizacji — `VOView`, czy też widoki zawierające bardziej szczegółowe dane i umożliwiające wykonywanie skomplikowanych operacji; `PhysicalHostTableView`, `VGTableView`, `VMTableView`), okien, posiadających wsparcie dla mechanizmów *docking*[18]. W odpowiednim pliku definiuje się przyciski znajdujące się na pasku menu czy narzę-

[17] Spring Richclient: `http://spring-rich-c.sourceforge.net/`.

[18] Przesuwanie metodą przeciągnij-i-upuść, maksymalizacja rozmiaru w obrębie aplikacji, odłączenie od głównego okna aplikacji, minimalizacja okna do postaci przycisku a także do edycji etykiet i statusu okna, w prosty sposób definiowane są skomplikowane formularze, z automatyczną walidacją wprowadzanych danych czy podpowiedziami.

dzi, a także tekst pojawiający się na elementach UI (ang. *User Interface*). Bardzo dobrze zdefiniowane są mechanizmy zarządzania akcjami czy obsługa zdarzeń.

Wszystkie powyższe i wiele innych zalet czynią z Spring RCP środowisko, które strukturalizuje i porządkuje aplikacje graficzne oraz zdecydowanie upraszcza proces rozszerzania aplikacji poprzez dodawanie kolejnych elementów GUI. System korzysta również z biblioteki XStream[19], służącej do zapisu obiektów POJO (ang. *Plain Ordinary/Old Java Object*) do postaci XML i na odwrót. Do postaci XML zapisywane są np. reguły Jess'a w centralnym repozytorium (patrz sekcja 6.1.2) czy topologia sieciowa wirtualnego Gridu (patrz sekcja 6.1.3). Z wykorzystaniem tej biblioteki niepotrzebne staje się parsowanie plików XML, czy też konstruowanie ich z łańcuchów znaków, zastępują te czynności metody `toXML(object, file)` oraz `fromXML(file, object)`.

6.3.1. Propagacja zdarzeń w systemie

Stworzony system oparty został o model zdarzeniowy, który jest ściśle związany z zastosowanymi technologiami implementacyjnymi. Zmiany w systemie generują zdarzenia, na które mogą rejestrować się pozostałe komponenty wchodzące w skład systemu[20].
Wyróżniono trzy rodzaje zdarzeń generowanych w systemie:

- zdarzenia generowane przez Xen-API, zdarzenia te zostały przedstawione w sekcji 6.1.2. Odbiorcą zdarzeń generowanych przez podsystem Xen jest `NodeMaster`,
- zdarzenia generowane przez warstwę pośredniczącą JMX. Zdarzenia te są generowane przez `NodeMaster` jako propagacja zdarzeń otrzymanych z warstwy Xen oraz w sytuacji informowania o zmianie stanu (stworzenie/usunięcie) wirtualnego Gridu zainstalowanego na danym węźle (referencja do rozdziału gdzie to opisano szczegółowo). Odbiorcami zdarzeń tego rodzaju są komponenty zdalne budujące system VGRMS a w szczególności aplikacje klienckie,
- zdarzenia generowane w ramach aplikacji klienckiej. Ten rodzaj zdarzeń wynika ze specyfiki architektury implementacyjnej Spring, w której poszczególne komponenty budujące aplikację komunikują się ze sobą przy użyciu zdarzeń propagowanych przez `ApplicationEventMulticaster`[21]. Zdarzenia te są generowane jako propagacja zdarzeń otrzymanych z warstwy pośredniczącej JMX. Dla celów VGRMS stworzono dwa rodzaje zdarzeń propagowanych wewnątrz aplikacji klienckiej:
  - zdarzenia pochodzące od obiektu `NodeMaster` — są one propagowane w postaci `VGRMEvent` (listing 6.8), do wszystkich widoków aplikacji klienckiej. Tym samym wszystkie widoki mogą zaktualizować swoje dane co sprowadza się do przebudowania drzewa bądź reorganizacji tabeli,
  - zdarzenia pochodzące od obiektu `PolicyRepository` — są one propagowane w postaci `PolicyRepoEvent` (listing 6.9), do widoku wyświetlającego wszystkie zdefiniowane w repozytorium polityki. Tym samym aktualizowana jest tabela wyświetlająca polityki (rysunek 6.7).

[19] XStream: `http://xstream.codehaus.org/`.
[20] Niektóre fragmenty mechanizmu zdarzeniowego zostały opisane przy okazji omawiana szczegółów implementacyjnych poszczególnych komponentów.
[21] Jest to klasa z biblioteki Spring.

```
public enum Type {
    VMCreated, VMDestroyed, VGCreated, VGDestroyed
}
```

Kod źródłowy 6.8. Możliwe typy zdarzeń przenoszone przez VGRMEvent

```
public enum Type {
    RuleAdded, RuleDeleted, ScriptAdded, ScriptDeleted
}
```

Kod źródłowy 6.9. Możliwe typy zdarzeń przenoszone przez PolicyRepoEvent

6.4. Podsumowanie

Opisane w niniejszym rozdziale szczegóły implementacyjne systemu VGRMS zostały ograniczone jedynie do najistotniejszych elementów.

W ramach tworzonego systemu zostały także zaimplementowane (w formie reguł dla silników regułowych) różne algorytmy zarządzania zasobami[22] wykorzystywane do realizacji testów różnych elementów systemu, wykonywanych na etapie tworzenia oprogramowania. Część tych reguł, wykorzystywanych w przedstawionej w rozdziale 7 praktycznej ewaluacji została zamieszczona w dodatku B.

Elementem zupełnie pominiętym były także narzędzia i skrypty stworzone przez autora i używane np. w celach automatyzacji procesu dystrybucji wersji binarnych komponentów systemu w obrębie infrastruktury sprzętowej.

[22] Prace te pokazały również jakie są praktyczne możliwości zastosowania systemu, co z kolei pomogło w opracowaniu scenariuszy praktycznej jego ewaluacji.

Rozdział 7

Testy systemu

W niniejszym rozdziale zostały przedstawione rezultaty prac, mających na celu praktyczną ewaluację proponowanego rozwiązania, a także weryfikację przyjętych w rozdziale 3 wymagań. Przedstawione koncepcje zostały zaimplementowane w prototypowym systemie, który został uruchomiony na dedykowanej infrastrukturze sprzętowej. Zostały przedstawione wyniki jakościowej i ilościowej oceny zaproponowanej architektury oraz efektywności jej implementacji.

W tym celu autor zaproponował metodologię testów, kryteria ewaluacji oraz opracował i wykonał środowisko testowe, w ramach którego zostały przeprowadzone eksperymenty umożliwiające praktyczną weryfikację i ocenę właściwości opracowanego środowiska.

Właściwe prace związane z ewaluacją środowiska zostały poprzedzone pracami wynikającymi z konieczności uzyskania informacji na temat właściwości wykorzystywanych technologii (szczególnie techniki wirtualizacji). Informacje te pozwoliły na wstępne oszacowanie kluczowych parametrów pracy, takich jak np. koszt migracji, narzuty, czy poziom separacji wykonania VM oraz na odpowiednie dobranie algorytmów rozdziału zasobów wykorzystywanych w systemie zarządzania.

Na potrzeby przeprowadzenia pomiarów stworzony został również zestaw narzędzi używanych do automatyzacji procedury wykonywania testów oraz zestaw skryptów służących do zbierania i wstępnej analizy wyników.

Rozdział zamyka podsumowanie zawierające omówienie najistotniejszych wniosków z przeprowadzonych badań w odniesieniu do zaproponowanej tezy niniejszej rozprawy.

7.1. Metodologia testów

Podczas opracowania metodologii testów autor skoncentrował się na możliwości wykonania praktycznych testów implementacji systemu pod kątem efektywności jego działania oraz ujemnych cech całości rozwiązania w odniesieniu do możliwości uzyskiwanych w klasycznych środowiskach Grid.

Badania zostały podzielone na trzy niezależne części. Pierwsza z nich związana była z ewaluacją własności środowiska w wydzielonej infrastrukturze laboratoryjnej. Infrastruktura ta została zaprojektowana i zbudowana aby można było ocenić szczególnie istotne skutki działania środowiska takie jak np. możliwości regulacji przydziału zasobów wraz z izolacją aplikacji, użycie migracji do realizacji dynamicznej alokacji zasobów czy możliwości konstrukcji odpowiednich polityk optymalizacji. W infrastrukturze laboratoryjnej obciążenie było generowane przez odpowiednio dobrany zestaw wykonujących się aplikacji[1]

[1]W niektórych testach działanie aplikacji było specjalnie zakłócane (np. przez nierównomierny rozdział

rozproszonych.

W laboratoryjnym środowisku testowym wydajność przedstawionego rozwiązania oceniana będzie w dwóch zależnych kontekstach. Pierwszy z nich to kontekst użytkownika, związany z oceną wydajności np. skróceniem czasu wykonania aplikacji czyli czasu pomiędzy rozpoczęciem wykonania a chwilą, w której następuje zwrócenie rezultatów. Drugi zestaw parametrów to współczynniki opisujące wykorzystanie zasobów, właściwości rezerwacji (gwarancje parametrów QoS) czy izolacji — parametry istotne z poziomu administratora systemów informatycznych wykorzystywanych do obliczeń rozproszonych.

Kolejnym celem niniejszych badań jest weryfikacja koncepcji powiązania aplikacji z dostosowanym do jej wymogów środowiskiem uruchomieniowym w postaci wirtualnego Gridu oraz na ile jest to skuteczny mechanizm pozwalający na efektywne dzielenie zasobów obliczeniowych i komunikacyjnych.

Według autora tak zaproponowana metodologia badań (oraz weryfikacja praktyczna systemu) dała możliwość przekonania się na ile zaproponowany poziom wirtualizacji nadaje się do realizacji koncepcji wirtualnego Gridu wykorzystywanej do realizacji zarządzania zasobami dla aplikacji uruchomionej w środowisku rozproszonym.

7.1.1. Konfiguracje testowe

Niniejsza sekcja przedstawia informacje na temat wykorzystywanej infrastruktury sprzętowej, programowej a także właściwości aplikacji testowych używanych podczas ewaluacji środowiska VGRMS. Dodatkowo zamieszczone zostały rysunki topologii sieciowych zarówno fizycznych (połączenia w obrębie urządzeń środowiska testowego) jak i wirtualnych używanych w obrębie wirtualnego Gridu.

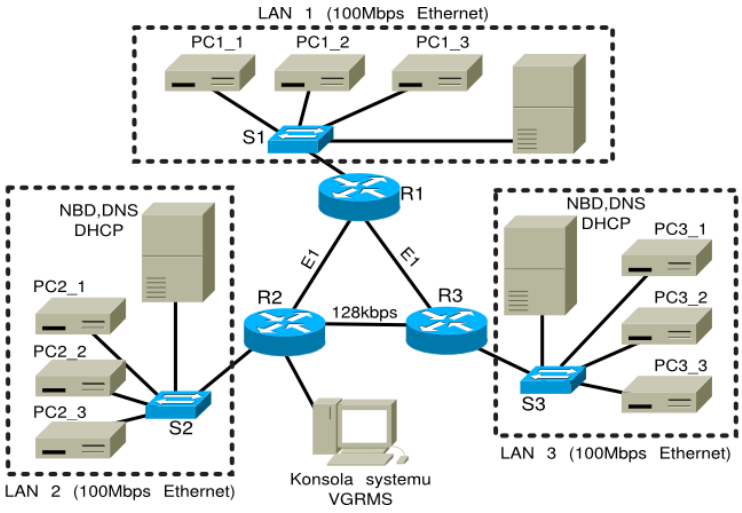
Infrastruktura sprzętowa

Implementacja systemu VGRMS została zainstalowana na dedykowanej infrastrukturze sprzętowej. Infrastruktura ta składała się z zestawu komputerów klasy PC połączonych siecią komputerową. Scenariusz zakładał przetestowanie proponowanego rozwiązania w warunkach zbliżonych do często obserwowanego sposobu udostępniania zasobów w środowiskach Grid. Dlatego środowisko testowe zostało zbudowane w oparciu o dedykowaną infrastrukturę sprzętową, której zasoby zostały podzielone na trzy niezależne klastry obliczeniowe, a następnie połączone za pomocą urządzeń sieciowych, których konfiguracja odpowiadała parametrom komunikacji dostępnych w sieciach rozległych WAN. Diagram połączeń sieciowych infrastruktury oraz ich właściwości został przedstawiony na rysunku 7.1.

Całość testów przeprowadzona była w „Laboratorium zaawansowanych technologii sieciowych" należącym do Wydziału Elektrotechniki, Automatyki, Informatyki i Elektroniki AGH. Wyposażenie laboratorium stanowi zestaw 30 komputerów PC o identycznych konfiguracjach[2] oraz sprzęt sieciowy firmy Cisco Systems służący do budowania topologii komunikacyjnych.

ilości przetwarzanych informacji przez poszczególne procesy) aby można było ukazać pozytywne właściwości zastosowania wirtualizacji w takich sytuacjach.

[2]Parametry sprzętowe wszystkich wykorzystywanych komputerów PC były jednakowe i zostały przedstawione w tabeli 7.1.



Rysunek 7.1. Topologia sieciowa ewaluacyjnej infrastruktury laboratoryjnej

Komponent	Opis
Procesor	Intel® Celeron® 2.8 GHz, 256 kB L2 *cache*
Płyta główna, *chpiset*	ASUS P5-P800, Intel® 865PE
Pamięć operacyjna	od 1 GB do 4 GB (w konfiguracjach: 1 × 1 GB, DDR-400, 2 × 1 GB, DDR-400, 2 × 500 MB, DDR-400)
Dysk twardy	Samsung SP0812N, 80 GB, ATA-133, 7200 rpm, 8 MB *cache*
Interfejs sieciowy	Marvell Technology Group Ltd. 88E8001 Gigabit Ethernet

Tabela 7.1. Konfiguracja sprzętowa komputerów PC środowiska testowego

Infrastruktura programowa

Implementacją parawirtualizacji w wykorzystanej infrastrukturze jest Projekt Xen. [3]. Pozostałe komponenty środowiska (odpowiednie wersje) zostały dobrane na podstawie praktycznej ewaluacji przeprowadzonej przez autora, przed lub w trakcie pisania niniejszej pracy.

Konfiguracja programowa głównych elementów infrastruktury laboratoryjnej została zebrana w tabeli 7.2.

Tabela 7.3 zawiera programową konfigurację środowiska wirtualnego Gridu. Konfiguracja ta została przygotowana w formie zestawu obrazów głównych partycji systemowych, których wykonanie odbywa się w obrębie maszyny wirtualnej VM.
Rysunek 7.2 przedstawia topologię logiczną Gridu zbudowanego w oparciu o wirtualne maszyny VM.
Prace, których rezultatem było wykonanie konfiguracji programowej laboratoryjnego środowiska testowego obejmowały:

- Instalację i konfigurację systemu operacyjnego na fizycznych węzłach, wraz z opro-

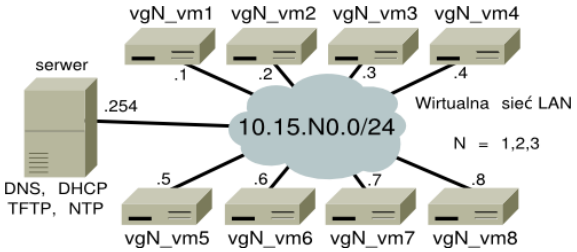
[3] W oparciu o stabilną linię 3.1.x. Wersja ta została wybrana, gdyż w fazie budowy prototypu systemu obserwowane były często występujące problemy stabilności otwartej implementacji projektu Xen, w wydaniach należących do serii 3.2.x.

Komponent	Opis
System operacyjny	Slackware Linux, wersja 12.1+ (2008-07-29)
Implementacja parawirtualizacji	Projekt Xen, wersja 3.1.4 (2008-04-25)
Organizacja dysków sieciowych	Projekt NBD, wersja 2.9.11 (2008-05-01)
Logiczne partycje	Projekt LVM, wersja 2.02.09 (2006-08-17)
Obsługa zapisu CoW	LVM (ang. *Logical Volume Manager*) Snapshots
Implementacja CIM	OpenPegasus, wersja 2.7.1 (2008-04-17)
Monitory środowiska	Projekt Lm_sensors, wersja 3.0.2 (2008-05-18)
Komunikacja sieciowa	Projekt VDE, wersja 2.2.2 (vde2) (2008-07-08)
Wirtualna maszyna Java	SUN JDK (ang. *Java Development Kit*) 1.6.0_02-b5

Tabela 7.2. Konfiguracja programowa elementów środowiska testowego

Komponent	Opis
System operacyjny	Scientific Linux SL, wersja 4.5
EGEE Grid *middleware*	gLite, wersja 3.1
Serwer kolejkowy	Torque, wersja 1.0.1p5
Scheduler systemu kolejkowego	Maui, (dostępny w pakiecie gLite)
Biblioteka MPI	MPICH, wersja 1.2.7p1 / MPICH2, wersja 1.0.7

Tabela 7.3. Konfiguracja programowa maszyn VM tworzących wirtualny Grid



Rysunek 7.2. Logiczna topologia połączeń w obrębie wirtualnego Gridu

gramowaniem obsługi wirtualizacji.

- Instalacja dodatkowych komponentów systemowych takich jak oprogramowanie monitorowania środowiska, obsługa logicznych woluminów dyskowych, itp.
- Instalacja i konfiguracja oprogramowania stworzonego w ramach prac nad tworzeniem systemu zarządzania zasobami[4].
- Stworzenie obrazów systemu operacyjnego wirtualnych maszyn VM i instalacja w ich obrębie oprogramowania typu „*middleware*" dla środowisk Grid.
- Instalacja i konfiguracja oprogramowania używanego do automatyzacji wykonywania testów oraz do zbierania i przetwarzania wyników pomiarów.

[4]Oprogramowanie to zostało przedstawione w rozdziale 6.

Aplikacje testowe

System zarządzania zasobami został przetestowany poprzez pomiar parametrów opisujących wykorzystanie zasobów oraz parametry wykonania trzech testowych aplikacji. Aplikacje te zostały tak dobrane aby można było przetestować środowisko zarządzania w sytuacjach znacznego obciążenia zarówno zasobów obliczeniowych jak i komunikacyjnych.[5]

Aplikacja	Właściwości	Użyta w teście
Aplikacja A_1	Intensywnie korzystająca z zasobów obliczeniowych (CPU, Pamięć)	4
Aplikacja A_2	Intensywnie wykorzystująca zasoby komunikacyjne	2
Aplikacja A_3	Aplikacja o zróżnicowanych (mieszanych) wymaganiach	1, 3
Aplikacja A_4	Aplikacja, w której obciążenie zasobów jest niesymetryczne	5

Tabela 7.4. Właściwości aplikacji testowych używanych podczas ewaluacji systemu

Aplikacją testową była implementacja algorytmu oddziałujących cząstek w wersji równoległej dla komputerów z pamięcią rozproszoną wykorzystującą środowisko MPI (ang. *Message Passing Interface*). Idea symulacji metodą cząstek polega na podziale symulowanego układu na fragmenty zwane cząstkami oraz zadaniu sposobu oddziaływania między nimi. Oddziaływanie to jest najczęściej opisane przy pomocy potencjału. Symulacja polega na wygenerowaniu konfiguracji początkowej (położeń i prędkości cząstek) oraz obserwacji ewolucji układu przez zadaną z góry liczbę kroków czasowych. Cząstki zamknięte są w odizolowanym od wpływów zewnętrznych pudle obliczeniowym. W każdym kroku czasowym dokonuje się: obliczenia sił oddziaływania pomiędzy cząstkami i zbudowaniu układu równań ruchu zgodnie z II zasadą dynamiki Newtona, rozwiązanie układu równań ruchu prowadzące do wyznaczenie położeń i prędkości cząstek w kolejnym kroku czasowym oraz obliczenia wartości parametrów makroskopowych charakteryzujących układ. W metodach tych najistotniejszym zagadnieniem jest efektywne wyznaczenie sąsiadów każdej cząstki dla obliczenia wartości sił oddziaływania. Zastosowany algorytm zależy od charakteru potencjału oddziaływania. Dla potencjałów długozasięgowych stosuje się sumowanie Ewalda, metodę *Particle Mesh Ewald*, metodę *Particle–Particle–Particle–Mesh* lub *Fast Multipole Method*. Dla potencjałów krótkozasięgowych wprowadza się tzw. promień obcięcia i najczęściej stosuje metodę cel połączonych Hockneya. W implementacji testowej oddziaływanie między cząstkami opisuje model dyssypatywnej dynamiki cząstek DPD (ang. *Dissipative Particle Dynamics*) z liniowym, krótkozasięgowym potencjałem odpychającym.

Metoda ta jest powszechnie stosowana do symulacji zjawisk zachodzących w mezoskali. Symulowane zjawisko to separacja faz. Konfiguracja początkowa polega na przypadkowym rozmieszczeniu w pudle obliczeniowym cząstek dwóch typów. W trakcie symulacji cząstki każdego typu zajmują spójne obszary przestrzeni.

Różne parametry wykonania aplikacji uzyskano głównie poprzez zmianę ilości kroków czasowych (czas obliczeń) czy zmianę sposobu odwzorowania fragmentów przestrzeni pudła

[5]Przy doborze aplikacji testowych zostały uwzględnione przyjęte na wstępie założenia odnośnie klas zastosowań środowiska zarządzania zasobami, przedstawione w sekcji 3.3.4.

obliczeniowego na zbiór wirtualnych maszyn (komunikacja sieciowa). W teście dla którego wymagane było niejednorodne rozłożenie obciążenia został użyty parametr powodujący grawitacyjne osiadanie cząstek co z kolei spowodowało ich nierównomierne rozmieszczenie.

7.1.2. Scenariusze przeprowadzenia pomiarów

Sekcja ta przedstawia pięć scenariuszy weryfikujących właściwości systemu VGRMS. Każdy scenariusz zawiera opis wymagań oraz przedstawienie algorytmów jakie będą wykorzystane i zaimplementowane za pomocą konfiguracji modułu obsługi logiki przydziału zasobów. Kod źródłowy ważniejszych reguł wykorzystywanych podczas testów został umieszczony w dodatku B.

Zaproponowane scenariusze testów można podzielić na dwie kategorie, pierwszą stanową testy demonstrujące możliwości systemu w zakresie przydziału zasobów. Drugą kategorie stanowią testy wydajnościowe mające na celu wykazanie możliwości w zakresie poprawy działania aplikacji.

Eksperyment 1 — Zadane rozmieszczenie VM

Scenariusz ten zakłada wykorzystanie możliwości automatyzacji rozmieszczenia wykonania VM, w zależności od przyjętych kryteriów.
Scenariusz testów składa się następujących kroków:

- system tworzy maszyny wirtualne wirtualnego Gridu w sposób losowy wybierając węzeł na którym dana VM zostanie uruchomiona,
- uruchomiona zostaje aplikacja generująca obciążenie (aplikacja A_3),
- zgodnie z przyjętą polityką następuje korekcja rozmieszczenia wykonania wirtualnych maszyn.

Przyjęta polityka rozmieszczenia w obrębie wirtualnego Gridu zakłada następujące warunki:

1. Minimalna liczba VM uruchomionych w obrębie fizycznego węzła — 1.
2. Maksymalna liczba VM uruchomionych w obrębie fizycznego węzła — 3.
3. Wykorzystanie infrastruktury (dostępnych fizycznych węzłów) nie większe niż 35 % dostępnego czasu procesora wszystkich maszyn fizycznych.
4. Wykorzystanie pamięci fizycznego węzła nie może przekroczyć 67 %.

W trakcie wykonania danego eksperymentu zostanie zmierzony czas stworzenia VG, poziom wykorzystania zasobów podczas uruchomienia VG, a także czas wykonania i zużycie zasobów w trakcie działania procesu zmiany rozmieszczenia wykonania VM. Dodatkowo zostanie także zmierzony również narzut generowany przez wszystkie komponenty systemu.

Realizacja algorytmu działania procedury modyfikacji rozmieszczenia opiera się na prostej zasadzie — najpierw zostaje stworzony zestaw faktów dla wszystkich możliwych

konfiguracji migracji,VM [6] a następnie eliminowane są te, dla których parametry hosta źródłowego spełniają przyjęte założenia (politykę rozmieszczenia) oraz usuwane są migracje tej samej maszyny wirtualnej (duplikaty). Wszystkie pozostałe procesy migracji zostają wykonane. To podejście, w sytuacji gdy nie istniałoby takie rozmieszczenie maszyn, które dla danej infrastruktury spełniałoby postawione założenia powodowałoby wykonywanie migracji VM w nieskończoność — dlatego do warunku migracji został prowadzony element ograniczający maksymalną liczbę migracji dla każdej maszyny wirtualnej.

Eksperyment 2 — Grupowanie VM na podstawie ścieżek komunikacyjnych

Eksperyment ma na celu ukazanie możliwości poprawy działania aplikacji, grupując wirtualne maszyny, na których działają te aplikacje na podstawie informacji pochodzących z monitorowania komunikacji sieciowej. Wykonanie dwóch VM w obrębie jednego fizycznego węzła, maszyn silnie ze sobą komunikujących się powoduje, iż komunikacja ich będzie miała praktycznie nieograniczoną przepustowość[7]. Optymalizacja ta będzie szczególnie istotna dla aplikacji rozproszonych podczas działania, których występuje konieczność wymiany dużych ilości danych.

Scenariusz eksperymentu zakłada zmierzenie parametrów wykonania aplikacji testowej w sytuacji gdy rozmieszczenie wykonujących jej wirtualnych maszyn z punktu widzenia wydajności komunikacji sieciowej powoduje ograniczenie jej wydajności. Przypadek taki będzie miał miejsce gdy maszyny wirtualne, których komunikacja sieciowa jest znacząco duża będą działały w obrębie węzłów fizycznych połączonych siecią komputerową o niskiej przepustowości.

Następnie zostaną zmierzone są parametry wykonania aplikacji w sytuacji gdy system będzie dynamicznie zmieniał rozmieszczenie wirtualnych maszyn tak aby wirtualne maszyny, pomiędzy którymi jest znaczna komunikacja, wykonywały się w obrębie jednej maszyny fizycznej.

Algorytm działania procesu optymalizacji zakłada stworzenie dla każdej maszyny wirtualnej monitora komunikacji sieciowej. Monitor taki zbiera informacje statystyczne[8] o parametrach komunikacji sieciowej danej wirtualnej maszyny. Jego działanie umożliwia znalezienie między innymi adresu sieciowego VM, z którą wymieniane jest najwięcej danych. Dzięki możliwości zastosowania filtrów (ograniczenia interfejsów sieciowych zbierania danych i filtrów monitowanego ruchu) w statystykach tych możliwe będzie pominięcie np. komunikacji generowanej podczas migracji VM.
Algorytm działania systemu regułowego zakłada wykonanie następujących kroków:

1. Stworzenie dla każdego interfejsu sieciowego maszyny wirtualnej instancji modułu monitorowania komunikacji sieciowej (reguła „*create-net-monitor*" — kod źródłowy w dodatku B.10).

2. Wybranie dla każdego fizycznego węzła maszyny wirtualnej generującej największą komunikację i stworzenie dla niej tymczasowej instancji faktu migracji[9] (reguła

[6] Czyli dla każdej wirtualnej maszyny i każdego fizycznego hosta — wszystkie możliwe migracje.

[7] W praktyce będzie to szybkość przesyłania danych poprzez pamięć współdzieloną wraz z narzutem powodowanym kosztem dodatkowych wywołań systemowych.

[8] Interfejs komponentu używanego do monitorowania komunikacji sieciowej przedstawia kod źródłowy dodatek A.10.

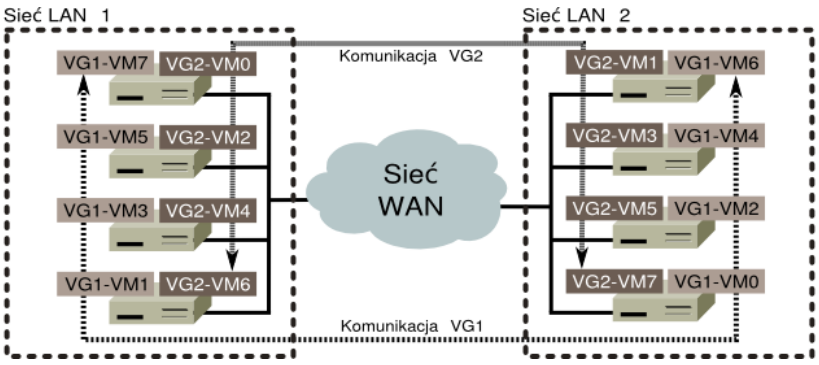
[9] Definicję faktu opisującego parametry migracji zawiera kod źródłowy B.5.

„*migrate-net-communication*" — kod źródłowy w dodatku B.10).

3. Usunięcie duplikujących się migracji, oraz migracji z zamienionymi fizycznymi węzłami (reguła „*cancel-cross-migrations*" — kod źródłowy w dodatku B.10).

4. Wykonanie wszystkich pozostałych migracji, (opcjonalnie) usunięcie monitorów sieciowych (reguły obsługi migracji B.6).

Metoda ta pozwala na zwiększenie efektywności komunikacji, jednak obciążenie sieci wynikające z transmisji danych podczas wykonywania migracji również może być znaczące. Możliwe jest wprowadzenie logiki, która mierząc parametry przeprowadzonych migracji będzie aktualizowała informacje statystyczne (bazę wiedzy) wykorzystywane do podejmowania bardziej trafnych decyzji.



Rysunek 7.3. Początkowe rozmieszczenie wykonania wirtualnych maszyn dla scenariusza z migracją na podstawie monitorowania komunikacji

Rysunek 7.3 przedstawia początkowe rozmieszczenie wirtualnych maszyn, tworzących wirtualne Gridy w obrębie fizycznych maszyn. W teście tym będą wykonane dwie instancje aplikacji intensywnie korzystającej z zasobów komunikacyjnych.

Eksperyment 3 — Awaria fizycznego węzła

Możliwość monitorowania fizycznych parametrów infrastruktury (jak np. temperatura komponentów sprzętowych, uszkodzenia fizyczne, liczniki błędów itp.) pozwala na łatwe uzupełnienie logiki działania systemu o funkcjonalność pozwalającą na zwiększenie poziomu niezawodności procesów obliczeniowych. Można zatem dodać reguły, które śledząc parametry środowiska[10] pozwolą na przewidzenie i przeciwdziałanie awariom.

W eksperymencie tym będzie mierzona temperatura procesora[11], a system będzie reagował w sytuacjach wzrostu jej powyżej dopuszczalnej wartości. Jako akcja podejmowana przy wykryciu przekroczenia progu dozwolonych temperatur CPU, będzie wykonana migracja wszystkich działających w jego obrębie wirtualnych maszyn.

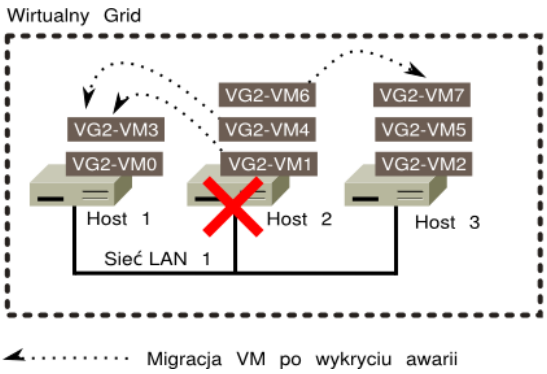
[10] Interfejs komponentu stosowanego do monitorowania parametrów środowiskowych fizycznych węzłów przedstawia kod źródłowy A.11.

[11] Zdarzają się bowiem często prozaiczne awarie układu chłodzenia procesora, bądź kluczowych elementów płyty głównej komputera powodujące jego zatrzymanie (bądź nawet trwałe uszkodzenie).

Symulacja awarii układu chłodzenia CPU została wykonana poprzez odłączenie na czas około 15 sekund zasilania dla wentylatora procesora w pojedynczym węźle wykonującym przetwarzanie dla trzech wirtualnych maszyn. Wyłączenie to nastąpiło po upłynięciu 50 % typowego czasu wykonania aplikacji testowej (A_3).
Operacje wykonywane przy monitoringu infrastruktury i przy wykryciu sytuacji zagrożenia przerwaniem obliczeń:

1. Stworzenie faktu reprezentującego parametry hosta uszkodzonego (reguła „*mark-failed-host*" — kod źródłowy B.8).
2. Zablokowanie migracji, w których docelowym fizycznym hostem jest węzeł uszkodzony (reguła „*cancel-migration-to-failed-host*" — kod źródłowy B.9).
3. Stworzenie tymczasowych faktów migracji wszystkich VM z uszkodzonego węzła na pozostałe dostępne w obrębie danego VG (reguła „*migrate-from-failed-host*" — kod źródłowy B.9).

Rysunek 7.4 przedstawia rozmieszczenie wykonania VM przed i po wykryciu i reakcji na uszkodzenie jednego z węzłów.



Rysunek 7.4. Rozmieszczenie wykonania wirtualnych maszyn w obrębie VG dla scenariusza z awarią jednego fizycznego węzła

Reguły realizujące powyższy scenariusz stanowią niejako uzupełnienie pozostałych reguł stworzonych na potrzeby innych eksperymentów. Brak jest jakichkolwiek interakcji i zależności pomiędzy nowo dodanymi regułami a istniejącą logiką systemu. W systemie pozostają zatem w użyciu (są wymagane) reguły obsługi migracji[12] (wykonywania, weryfikacji dostępności zasobów, usuwania duplikatów itp.).

Eksperyment 4 — Zastosowanie gwarancji QoS

Celem tego eksperymentu jest ukazanie możliwości systemu w zakresie gwarancji jakości wykonania, poprzez przydział zasobów dla równocześnie wykonujących się trzech instancji wirtualnych Gridów współdzielących daną infrastrukturę sprzętową.

Scenariusz zakłada uruchomienie trzech identycznych aplikacji (każda w obrębie dedykowanego wirtualnego Gridu). Każda z trzech aplikacji została przydzielona do jednej

[12] Reguły te przedstawione są w źródłach B.6.

z trzech klas określających poziom dostępności zasobów. Między klasami występuje zależność, a sytuacja ta odpowiada przypadkowi gdy operator udostępnia swoją infrastrukturę równocześnie dla klientów z różnymi parametrami kontraktów SLA.

Scenariusz tego eksperymentu zakłada wykonanie i przeprowadzenie następujących etapów (etapy te będą powtarzane dla każdej z trzech aplikacji testowych niezależnie):

- Zbadanie parametrów równoczesnego wykonania trzech instancji aplikacji w środowisku natywnym — pomiary czasu odpowiedzi, poziomu wykorzystania infrastruktury obliczeniowej i komunikacyjnej w sytuacji gdy aplikacje działają z bezpośrednim wykorzystaniem zasobów maszyn fizycznych. W tym przypadku nie ma zewnętrznej kontroli wykorzystania zasobów, za przydział zasobów dla każdej aplikacji odpowiada jedynie system operacyjny.
- Uruchomienie każdej aplikacji w obrębie dedykowanego wirtualnego Gridu — dla potrzeb każdej aplikacji na podstawie jej wymagań zostanie stworzone wirtualne środowisko wykonawcze. Przydział zasobów będzie sterowany przez system w oparciu o ustaloną politykę na podstawie przyjętych gwarancjach QoS dla trzech klas wykonania.

Podział zasobów (obliczeniowych, komunikacyjnych) dla klas będzie zgodny z następującą proporcją (7.1).

$$\text{Klasa I} \ : \ \text{Klasa II} \ : \ \text{Klasa III} \quad \equiv \quad 1 \ : \ 2 \ : \ 3 \tag{7.1}$$

Przebieg eksperymentu zakładał zbudowanie wirtualnego Gridu dla każdej aplikacji niezależnie. Zasoby w dedykowanym środowisku wykonania będą powiązane z zasobami fizycznymi w sposób statyczny, tj. podczas trwania obliczeń wirtualne komputery nie będą w żaden sposób przemieszczane w obrębie infrastruktury fizycznej. Na potrzeby przyjętego scenariusza została opracowana polityka zarządzania zasobami, która zaimplementowana została za pomocą zestawu reguł dla silnika decyzyjnego systemu zarządzania zasobami VGRMS. W scenariuszu tym, migracja VM nie będzie wykorzystywana.

Podział zasobów zgodnie z przyjętą (7.1) proporcją wymaga aby system dokonał regulacji wykorzystania zasobów poprzez modyfikacje następujących parametrów:

- zarządzanie zasobami obliczeniowymi — poprzez modyfikację czasu dostępu do procesora CPU dla maszyny wirtualnej,
- zarządzanie zasobami komunikacyjnymi — poprzez modyfikację parametrów komunikacji sieciowej wymienianej pomiędzy VM.

W proponowanym systemie, zarządzanie czasem procesora możliwe jest dzięki wykorzystaniu możliwości wpływania na parametry (tabela 7.5) algorytmu odpowiadającego za zmianę kontekstu wykonania (przełączenia procesora do wykonania innej maszyny wirtualnej) zaimplementowanego w nadzorcy (ang. *supervisor*) systemu Xen.

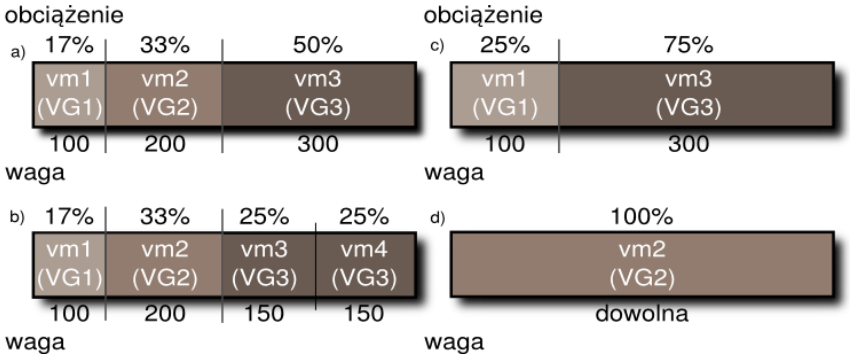
Konfiguracja parametrów wszystkich trzech, uruchomionych równocześnie wirtualnych Gridów była identyczna i została przedstawiona na listingu 7.1.

Dystrybucja obciążenia pomiędzy wirtualne maszyny powinna odpowiadać założonej proporcji. Rysunek 7.5 przedstawia możliwe przypadki rozmieszczenia wykonania wirtualnych maszyn należących do trzech wirtualnych Gridów w obrębie pojedynczej maszyny

Parametr	Działanie
Waga (ang. *weight*)	Wartość całkowita z zakresu od 1 do 65535 określająca proporcje pomiędzy czasem wykonania wirtualnych maszyn.
Limit górny (ang. *cap*)	Górne ograniczenie dostępu do procesora, mierzone w procentach obciążenia pojedynczego procesora fizycznego. Możliwe są wartości powyżej 100 dla systemów wieloprocesorowych (np. architektura SMP).

Tabela 7.5. Parametry konfiguracyjne algorytmu przydziału czasu procesora w systemie Xen

fizycznej. Wartości wag wynikają z proporcji przyjętej w założeniach i zakładają, iż każda wirtualna maszyna generuje maksymalne obciążenie procesora.



Rysunek 7.5. Możliwe przypadki rozmieszczenia wykonania wirtualnych maszyn w obrębie fizycznego węzła dla testu nr 4

$$\frac{\text{waga przyjęta dla VG}}{\text{liczba VM danego VG działających w obębie hosta}} \tag{7.2}$$

Takie statyczne przypisanie wag (zgodnie ze wzorem 7.2) nie będzie adekwatne w sytuacjach gdy obciążenie wirtualnych maszyn będzie zmienne (rysunek 7.6). Przykład ten pokazuje, iż założona proporcja 1 : 3 nie będzie zachowana, gdyż niewykorzystane przez maszynę vm3 cykle procesora będą przydzielone dla pozostałych maszyn w proporcji 1 : 3. Efektywne wykorzystanie zasobów przez dwie instancje wirtualnego Gridu będzie wynosiło zatem: 1 : 2.14.

Odpowiednia modyfikacja proporcji wag wirtualnych maszyn pozwoli na korektę poziomu wykorzystania zasobów (drugi przypadek z rysunku 7.6), dzięki czemu możliwe będzie zachowanie przyjętego poziomu QoS. Ta korekcja jest możliwa do przeprowadzenia przez silnik regułowy `VG-PE` gdyż zmiany dotyczą wirtualnych maszyn danego wirtualnego Gridu, i nie ma konieczności modyfikacji wag pozostałych VM (należących do innych VG).

Ostateczny algorytm działania przydziału zasobów podczas testu gwarancji QoS został zaimplementowany w postaci zestawu reguł (kod źródłowy B.4) używanych przez moduł decyzyjny do modyfikacji parametrów wirtualnego Gridu wpływających na podział zasobów. Zestaw ten był stosowany dla wszystkich trzech wirtualnych Gridów równocześnie

```
<?xml version="1.0" encoding="UTF-8"?>
<vgrid>
  <name>vg1-test1</name>
  <description></description>

  <phosts number="8" max_number="8" min_number="8">
    <filters></filters>
  </phosts>

  <rules></rules>

  <vnodes number="8">
    <vnode name="WorkerNode_1">
      <label>label</label>
      <memory>256</memory>
      <vcpu>1</vcpu>
      <vif>1</vif>

      <kernel>
        <file>/boot/vmlinux-2.6.18-xenU</file>
        <root>/dev/hda1</root>
        <args>ro 3</args>
        <ramdisk></ramdisk>
      </kernel>

      <interface>
        <type>ethernet</type>
        <name>eth0</name>
        (...)
        <script>dhcp</script>
      </interface>

      <device>
        <type>disk</type>
        <uri>file://dev/nbd1</uri>
        <name>/dev/hda1</name>
      </device>
    </vnode>
    (...)

  </vnodes>
</vgrid>
```

Kod źródłowy 7.1. Konfiguracja parametrów wirtualnego Gridu na potrzeby eksperymentu nr 1

(różne były tylko domyślne wartości wag modyfikowane na poziomie globalnym zarządzania za pomocą reguł 7.2).

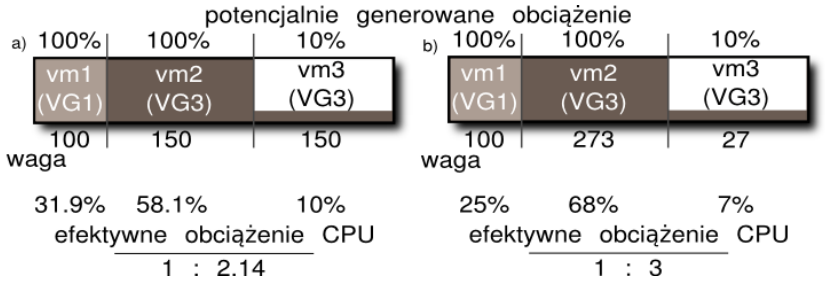
```
(defrule vg1-static-weight-100 "set default weight (100) for VG1"
  (JessVGFact (vgName VG1) (OBJECT ?o)) => (call ?o addRule set-vg-global-weight-100))

(defrule vg2-static-weight-200 "set default weight (200) for VG2"
  (JessVGFact (vgName VG2) (OBJECT ?o)) => (call ?o addRule set-vg-global-weight-200))

(defrule vg3-static-weight-300 "set default weight (300) for VG3"
  (JessVGFact (vgName VG3) (OBJECT ?o)) => (call ?o addRule set-vg-global-weight-300))
```

Kod źródłowy 7.2. Zestaw reguł określających dystrybucję zasobów w obrębie VO podczas testu nr 4

Reguły związane z rozdziałem wag w obrębie poszczególnych wirtualnych Gridów zostały przedstawione w dodatku B.

Eksperyment 5 — Wykorzystanie migracji VM

Dodatkowo w systemie zostaną przeprowadzone testy działania infrastruktury w sytuacji wykonania aplikacji rozproszonej, w której wykonaniu występują przypadki nieefektywnego rozdziału obciążenia. Aplikacja taka np. przy dystrybucji zadań (danych obliczenio-



Rysunek 7.6. Przypadek niezgodnego z założeniami przydziału zasobów dla testu nr 4

wych) nie uwzględnia aktualnego poziomu dostępności zasobów tworząc sytuację, w której część węzłów obliczeniowych jest przeciążona, podczas gdy zasoby pozostałych są niewykorzystane.

Działanie systemu pozwoli w tym przypadku, poprzez wykorzystanie mechanizmów migracji na uwzględnienie aktualnego obciążenia węzłów oraz na skrócenie czasu wykonania aplikacji przez eliminację sytuacji niepełnego wykorzystania dostępnych zasobów.

Scenariusz ten zakłada wykonanie aplikacji obliczeniowych w sytuacji, gdy część węzłów jest obciążona za pomocą sztucznie wygenerowanego, zmiennego obciążenia oraz algorytmy dystrybucji zadań nie uwzględniają rzeczywistego czasu przetwarzania poszczególnych etapów.
Wyróżniono tutaj następujące przypadki:

- wykonywana jest aplikacja nie uwzględniająca istniejącego obciążenia infrastruktury przy wewnętrznym rozdziale zadań,
- wykonywane są równocześnie dwie aplikacje o ortogonalnych wymaganiach, np. aplikacja obliczeniowa i aplikacja przetwarzająca duże ilości danych (generująca obciążenie infrastruktury komunikacyjnej),
- wykonanie każdej z trzech aplikacji w sytuacji, gdy część węzłów jest obciążona za pomocą sztucznie wygenerowanego, zmiennego obciążenia.

Celem nadrzędnym tego eksperymentu jest wykazanie możliwości systemu w zakresie poprawy parametrów działania aplikacji, widoczną głównie poprzez skrócenie czasu odpowiedzi lub przy zachowaniu tego samego czasu wykonania aplikacji na możliwości równoczesnego wykonania większej ich liczby. Ten drugi przypadek możliwy jest dzięki poprawie współczynników określających poziom wykorzystania zasobów.

Dystrybucja obciążenia może być statyczna (dynamiczna dystrybucja obciążenia polega głównie na zwiększeniu poziomu wykorzystania zasobów poprzez zmianę parametrów wykonania procesów w trakcie ich działania. W pracach naukowych [3, 12, 46] poszukiwane są głównie techniki i algorytmy wykonujące dystrybucję obciążenia w celu osiągnięcia określonego stanu systemu. Zadanie przyjętych algorytmów sprowadza się do zapewnienia w infrastrukturze, składającej się ze zbioru fizycznych maszyn udostępniających zasoby, wymogu spełnienia następujących warunków:

- w trakcie działania aplikacji wszystkie węzły mają mniej więcej takie same obciążenie,

- brak jest w infrastrukturze zasobów niewykorzystywanych podczas gdy inne są w tym czasie w użyciu.

Wykorzystywane i prezentowane w literaturze algorytmy zazwyczaj podchodzą do problemu dystrybucji wykonania poprzez podział problemu na mniejsze przypadki, które mogą być obsługiwane niezależnie (dystrybucja obciążenia w środowiskach rozproszonych). Rezultatem badań w tym obszarze są następujące możliwe podejścia:

- Techniki równoważenia obciążenia (ang. *load-balancing*) — celem, których jest wyrównanie obciążenia generowanego przez procesy. W tym przypadku możliwe jest wyszukanie rozmieszczenia o zadanych właściwościach [3] poprzez migrację zadań przy znanym koszcie migracji.

- Techniki współdzielenia obciążenia (ang. *load-sharing*) — można osłabić poprzednie podejście przyjmując jedynie, że niedopuszczalną sytuacją jest występowanie w środowisku zasobów niewykorzystanych. Migracja zadań wykonywana jest wtedy tylko pomiędzy zasobami przeciążonymi a zasobami całkowicie niewykorzystywanymi.

- Dystrybucja bez wywłaszczania (ang. *non-preemptive load distributing*) — wykorzystywane są informacje statystyczne pochodzące z monitoringu wcześniej zakończonych zadań w celu określenia miejsca wykonania dla nowego zadania. Jest to głównie metoda stosowana dla krótkotrwałych zadań, dla których koszt migracji odgrywa znaczącą rolę.

Wybór algorytmu podejmowania decyzji powinien zależeć od kosztu migracji. Przy dużym koszcie migracji narzut związany z wykonaniem operacji poszukiwania kandydata i docelowego miejsca wykonania może być zaniedbany. Dodatkowo koszt związany z degradacją wykonania procesów przy błędnie wykonanej migracji może być w tym przypadku znaczny. Implementacja każdego z tych trzech podejść jest możliwa za pomocą reguł polityki przydziału zasobów w systemie VGRMS.

Na potrzeby niniejszego testu został zaproponowany algorytm dokonujący modyfikacji rozmieszczenia wirtualnych maszyn w obrębie infrastruktury sprzętowej na podstawie kryteriów równoważenia obciążenia.

7.1.3. Metryki wydajnościowe

Definicje metryk opisujących wydajność rozproszonych środowisk obliczeniowych można podzielić na dwie grupy. Podział ten wynika z uwzględnienia różnych oczekiwań względem infrastruktury z poziomu dwóch rozłącznych grup jej użytkowników. I tak parametry związane z czasem przetwarzania aplikacji (7.3), czyli czasem pomiędzy chwilą rozpoczęcia przetwarzania a momentem zakończenia obliczeń (zwrócenia rezultatów), są istotne głównie z punktu widzenia użytkowników Gridu. Natomiast parametry określające stopień wykorzystania infrastruktury (7.10) to ważne z punktu widzenia administratora czynniki charakteryzujące właściwości warstwy pośredniej (*middleware*).

Metryki klasyczne[13] związane z przetwarzaniem aplikacji

Czas odpowiedzi infrastruktury R_A (7.3) dla aplikacji **A** (4.24) definiujemy jako różnicę czasów pomiędzy rozpoczęciem przetwarzania aplikacji a momentem jej zakończenia:

$$R_A = \Delta T := T_{end} - T_{start} \tag{7.3}$$

W przypadku, gdy przetwarzana aplikacja składa się z kilku uruchomionych niezależnie komponentów (4.24) należy rozdzielić czas wykonania całości przetwarzania od czasu wykonania poszczególnych składowych. Czas odpowiedzi (7.4) mierzony jest wtedy jako czas pomiędzy rozpoczęciem wykonywania pierwszego komponentu a czasem zakończenia przetwarzania ostatniego komponentu tej samej aplikacji:

$$R_A^{(i,j)} \quad := \quad T_{end}^{a_j} - T_{start}^{a_i} \quad \text{i} \quad a_i,\ a_j \in \mathbf{A}, \tag{7.4}$$

gdzie:

a_i — komponent aplikacji **A**, którego wykonanie zostało rozpoczęte jako pierwsze,
a_j — komponent aplikacji **A**, który został zakończony jako ostatni,

czyli:

$$R_A^{(i,j)} = R_A \tag{7.5}$$

Możemy również określić sumaryczny czas wykonania wszystkich komponentów, który określa jak długo działałaby aplikacja, jeżeli całe jej przetwarzanie byłoby wykonane sekwencyjnie[14]:

$$\overline{R_A} := \sum_{a_i \in A} R_{a_i}, \quad \text{gdzie:} \quad R_{a_i} = \Delta T_{a_i} := T_{end}^{a_i} - T_{start}^{a_i} \tag{7.6}$$

Zysk zrównoleglenia (7.7) to z kolei parametr określający w jakim stopniu możliwe jest skrócenie czasu wykonania aplikacji przy jej współbieżnym wykonaniu:

$$Z_A := \frac{\overline{R_A}}{R_A^{(i,j)}} * 100\ \% \tag{7.7}$$

Zdefiniowanie i wyznaczenie w/w parametrów pozwoli na oszacowanie właściwości systemu zarządzania zasobami widzianych z poziomu użytkowników. Parametry te opisują o ile zmieni się czas wykonania aplikacji w zależności od przyjętych parametrów pracy algorytmów zarządzania zasobami.

[13] Proponowane w tej sekcji metryki odnoszą się do klasycznej interpretacji infrastruktury Grid, czyli nie uwzględniają wpływu żadnych technik związanych z wirtualizacją infrastruktury. Odniesienie do środowiska, w którym zasoby są wirtualizowane, wprowadzają metryki wydajnościowe zaproponowane w dalszej części sekcji.

[14] Co naturalnie, nie dla każdej aplikacji jest możliwe.

Metryki obciążenia infrastruktury

Obciążenie infrastruktury, czyli efektywne wykorzystanie dostępnych zasobów jest kolejnym istotnym parametrem. Parametr ten określa stopień wykorzystania infrastruktury lub ile jest jeszcze niewykorzystanej np. mocy obliczeniowej czy dostępnej przepustowości sieciowej.

Dla aplikacji **A** (4.24) składającej się z niezależnych komponentów uruchomionych w obrębie zbioru fizycznych maszyn **PH** (4.1) określamy następujące parametry[15]:

$E_{a_i}^{(c_j)}$ — efektywny czas procesora c_j zużyty dla wykonania pojedynczego komponentu a_i aplikacji **A**,
$p(h_k)$ — zbiór procesorów dostępnych w obrębie fizycznego węzła h_k,
$h(a_i)$ — funkcja (7.8) zwracająca fizyczny węzeł, na którym jest uruchomiony komponent a_i.

$$h : \mathbf{A} \to \mathbf{PH},$$
$$h(a_i) := \{h_j \in \mathbf{PH} : a_i \text{ jest przetwarzane przez } h_j\} \quad (7.8)$$

Wykorzystanie pojedynczego węzła h przez aplikację **A** określamy zatem jako:

$$u(h,\ \mathbf{A}) := \frac{\sum_{a_i \in A} \sum_{c_j \in p(h)} E_{a_i}^{(c_j)}}{R_A}, \quad (7.9)$$

a współczynnik wykorzystania całej dostępnej infrastruktury **PH** przez aplikację **A** zostanie określony następująco:

$$U(\mathbf{PH},\ \mathbf{A}) := \frac{\sum_{h_k \in \mathbf{PH}} u(h_k, \mathbf{A})}{\#\mathbf{PH}}, \quad (7.10)$$

gdzie:

$\#\mathbf{PH}$ — liczba elementów zbioru **PH** (liczba dostępnych fizycznych węzłów).

Współczynnik ten możemy również obliczyć biorąc pod uwagę jedynie obciążenie węzłów wykorzystywanych przez aplikację. Parametr ten (7.11) będzie mógł lepiej charakteryzować obciążenie generowane przez aplikacje, które nie korzystają z wszystkich dostępnych zasobów infrastruktury.

$$U'(\mathbf{PH},\ \mathbf{A}) := \frac{\sum_{h_k \in H(\mathbf{A})} u(h_k, \mathbf{A})}{\#H(\mathbf{A})}, \quad (7.11)$$

gdzie:

[15] „Czas efektywny” rozumiany jest jako czas procesora przydzielony przez system operacyjny do obsługi aplikacji jedynie w tzw. trybie użytkownika. Wartość ta nie uwzględnia czasu spędzonego przez procesor w trybie systemowym, czyli czasu poświęconego na obsługę przez system operacyjny zdarzeń generowanych przez aplikację.

$H(\mathbf{A})$ — funkcja (7.12) zwracająca zbiór fizycznych węzłów h_k, które używane są podczas wykonania aplikacji $\mathbf{A}$,
$\#H(\mathbf{A})$ — liczba elementów zbioru $H(\mathbf{A})$ (liczba węzłów fizycznych używanych przez aplikację $\mathbf{A}$).

$$H : \mathbf{A} \to 2^{\mathbf{PH}}, \quad H(\mathbf{A}) := \left\{ \bigcup_{a_i \in \mathbf{A}} h(a_i) \right\} \tag{7.12}$$

Analogicznie do obciążenia zasobów obliczeniowych, możemy zdefiniować zestaw parametrów określających wykorzystanie infrastruktury komunikacyjnej. Dla aplikacji $\mathbf{A}$ należy określić zbiór jednokierunkowych transmisji sieciowych[16] $\mathbf{F_A}$ (7.13) poszczególnych jej komponentów a_i, czyli:

$$\mathbf{F_A} := \{ f_{i,j} : a_i \text{ i } a_j \text{ komunikowały się} \wedge a_i, a_j \in \mathbf{A} \wedge h(a_i) \neq h(a_j) \}, \tag{7.13}$$

gdzie:

$f_{i,j}$ — element oznaczający jednokierunkową komunikację komponentów a_i oraz a_j, dokonaną w trakcie działania aplikacji $\mathbf{A}$.

Pasmo zajęte przez komunikację komponentów a_i i a_j określamy zatem następująco:

$$B_{i,j} := \frac{b(f_{i,j}) + b(f_{j,i})}{R_A} \quad \text{dla:} \quad f_{i,j}, f_{j,i} \in \mathbf{F_A}, \tag{7.14}$$

gdzie:

$b(f_{i,j})$ — liczba bitów informacji przesłanych od komponentu a_i do a_j.

Dla topologii fizycznej $\mathbf{PN}$ (7.15), składającej się z zestawu połączeń pomiędzy fizycznymi węzłami $\mathbf{PH}$ określone zostały następujące funkcje:

$p^{(n)}(h_i,\ h_j)$ — funkcja (7.16) zwracająca zbiór połączeń sieciowych dla n-tej ścieżki łączącej fizyczne elementy infrastruktury h_i i h_j,
$\widetilde{b}^{(n)}(h_i,\ h_j)$ — przepustowość (7.17) n-tego połączenia pomiędzy h_i a h_j — czyli przepustowość najwolniejszego odcinka na danej ścieżce,
$\widetilde{B}(h_i,\ h_j)$ — efektywna przepustowość (7.18) infrastruktury sieciowej łączącej węzły h_i i h_j.

$$\mathbf{PN} \subseteq \{ l_{i,j} = (h_i, h_j) : h_i, h_j \in \mathbf{PH} \} \tag{7.15}$$

$$p : \mathbf{PH} \times \mathbf{PH} \to 2^{\mathbf{L}},$$
$$p^{(n)}(h_i,\ h_j) = \{ l_{m,n} : l_{m,n} \in \text{ n-tej ścieżki pomiędzy } h_i \text{ a } h_j \} \tag{7.16}$$

[16] Komunikacja sieciowa komponentów aplikacji uruchomionych w obrębie jednego fizycznego węzła nie jest uwzględniana, gdyż nie powoduje ona obciążenia zasobów topologii komunikacyjnej.

$$\widetilde{b}^{(n)}(h_i,\ h_j) := \min_{l_{m,n} \in p^{(n)}(h_i,h_j)} b(l_{m,n}) \tag{7.17}$$

$$\widetilde{B}(h_i,\ h_j) := \sum_{n=1}^{N} \widetilde{b}^{(n)}(h_i, h_j) \tag{7.18}$$

A zatem obciążenie generowane podczas komunikacji dwóch niezależnych komponentów aplikacji **A** będzie ilorazem wykorzystanego pasma i szerokości pasma dostępnego dla komunikacji fizycznych węzłów, na których odbywa się wykonanie tych komponentów.

$$N_A := \frac{1}{\#F_A} \sum_{f_{i,j} \in F_A} \frac{B_{f_{i,j}}}{\widetilde{B}(h_i, h_j)} \quad \text{oraz} \quad \bigwedge_{a_i \in A} h_i = h(a_i) \tag{7.19}$$

Efektywna komunikacja sieciowa przesłana podczas wykonania aplikacji będzie wyrażona następująco:

$$X_A := R_A \sum_{f_{i,j} \in F_A} B_{f_{i,j}} \tag{7.20}$$

7.2. Ocena zastosowanych technologii

Działanie systemu VGRMS, jego możliwości oraz ograniczenia silnie wynikają z właściwości techniki wirtualizacji, która zastosowana była jako mechanizm umożliwiający regulację dostępu do zasobów. Technika parawirtualizacji której implementacja stanowi projekt Xen była głównie testowana [5, 6] pod kątem poziomu wprowadzanego narzutu na wykonanie aplikacji uruchomionej w obrębie maszyny wirtualnej Xen. W literaturze tematu, niedostatecznie szczegółowo były przedstawiane aspekty związane z właściwościami migracji, kluczowej zdaniem autora z punktu widzenia możliwości oferowanych przez system VGRMS. Dlatego autor postanowił zamieścić przed wynikami pomiarów działania systemu VGRMS także informacje o możliwościach i narzutach związanych z wykonywaniem procesu przeniesienia wykonania VM pomiędzy dwoma fizycznymi hostami.

7.2.1. Wydajność i optymalizacja procesu migracji

Ocena jakościowa i ilościowa narzutu wprowadzanego przez proces migracji wirtualnych maszyn systemu Xen jest istotna z punktu widzenia kryteriów działania algorytmu dystrybucji obciążenia. Kluczowe jest również określenie wprowadzanego obciążenia, generowanego przez proces migracji na działanie przenoszonego systemu VM. Obciążenie to jest wnoszone przez oprogramowanie migracji działające na poziomie domeny kontrolnej Xen.

Właściwości generowanego przez proces migracji obciążenia należy rozpatrywać w dwóch zależnych kategoriach:

- obciążenie procesora przez oprogramowanie dokonujące identyfikacji i kopiowania stron pamięci używanych przez przenoszoną maszynę (zarówno po stronie fizycznego hosta źródłowego jak i maszyny odbierającej dane),
- obciążenia infrastruktury sieciowej przez połączenia przesyłające zawartość pamięci VM.

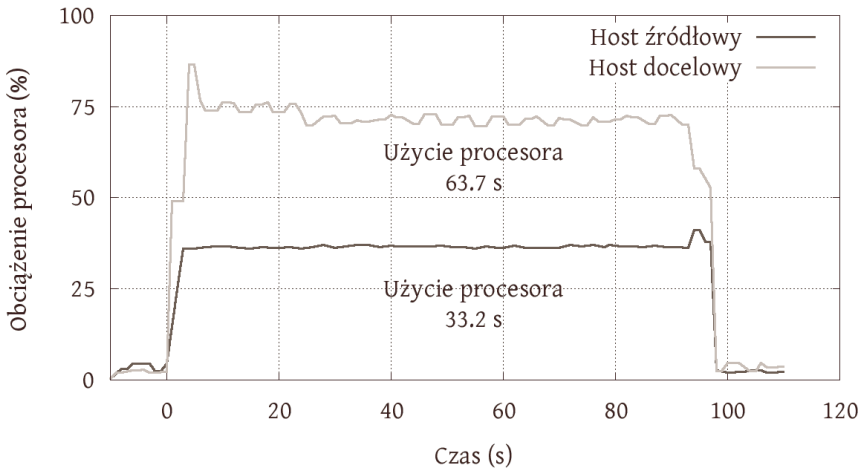
W celu oszacowania w/w narzutów zostało stworzone środowisko testowe składające się z czterech fizycznych węzłów[17]. Wszystkie elementy zostały połączone za pomocą pojedynczej sieci LAN. Maszyna wirtualna była przenoszona pomiędzy dwoma węzłami, a pozostałe hosty były odpowiedzialne za udostępnianie przestrzeni dyskowej w technologii NAS[18] dla migrowanej maszyny oraz za sterowanie samym procesem migracji.

Ponieważ efektywność migracji zależy głównie od trzech czynników — rozmiaru pamięci operacyjnej przydzielonej dla przenoszonej VM, obciążenia generowanego przez VM oraz intensywności korzystania z pamięci[19] — scenariusz testu składał się z następujących przypadków:

- migracji maszyny wirtualnej nieobciążonej — brak jakiejkolwiek aktywności poza procesami systemowymi. Obciążenie sumaryczne wirtualnych procesorów nie przekraczało w tym przypadku 1 %. Wykres obciążenia domeny kontrolnej dla tego testu przedstawia rysunek 7.7.

- migracji maszyny wirtualnej intensywnie wykorzystującej CPU — zostało uruchomionych $N + 1$ procesów (gdzie N to liczba wirtualnych procesorów w systemie w zakresie od 1 do 4) wykorzystujących w 100 % dostępny czas procesora. Obciążenie domeny 0 przedstawia wykres na rysunku 7.8.

- migracji maszyny wirtualnej, której uruchomione oprogramowanie intensywnie korzystało z operacji I/O i z pamięci operacyjnej. Obciążenie procesora podczas migracji dla tego przypadku przedstawia rysunek 7.9.

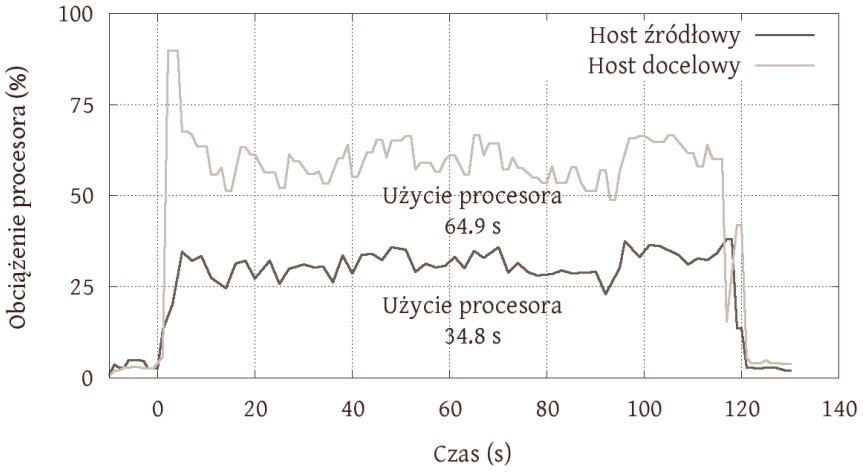


Rysunek 7.7. Obciążenie procesora podczas migracji VM nieobciążającej CPU

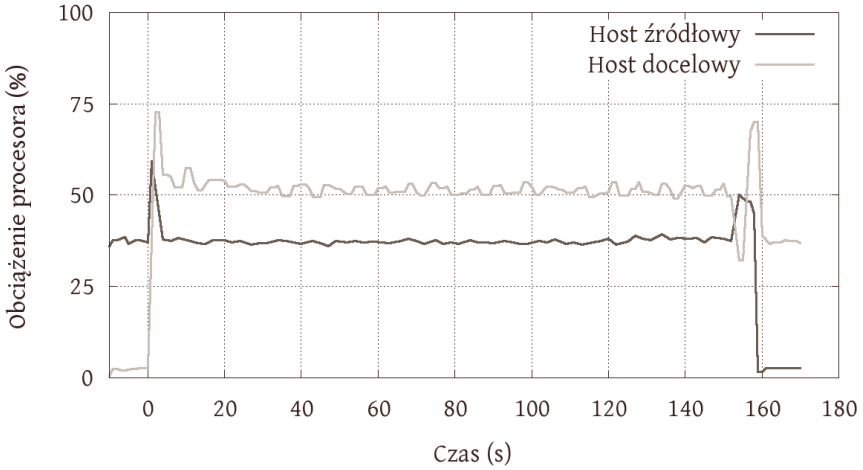
[17]Parametry fizyczne komputerów użytych podczas testów znajdują się w sekcji 7.1.1 na stronie 136.

[18]Zastosowanym oprogramowaniem realizującym usługi zdalnego (sieciowego) udostępniania blokowych urządzeń dyskowych był pakiet NBD (ang. *Network Block Device*) w wersji 2.9.9.

[19]Dokładniej częstość wykonywania operacji modyfikacji zawartości przesłanych już stron pamięci. Każda taka operacja powoduje konieczność ponownego przesłania zawartości strony w kolejnym kroku.



Rysunek 7.8. Obciążenie procesora podczas migracji VM obciążającej CPU



Rysunek 7.9. Obciążenie procesora podczas migracji węzła intensywnie wykorzystującego operacje I/O

Powyższe czynniki wpływają na ilość koniecznych do przesłania danych. Na sumaryczny czas migracji danego rodzaju obciążenia wpływa również wydajność (przepustowość) komunikacji sieciowej oraz efektywność działania oprogramowania identyfikującego i przesyłającego zawartość stron pamięci. Podane czynniki są od siebie niezależne, chcąc zatem zwiększyć efektywność procesu migracji (skrócić jego czas trwania) można:

- zwiększyć przepustowość dla komunikacji sieciowej — pozwoli to na skrócenie czasu migracji w sytuacji gdy oprogramowanie jest w stanie szybciej generować dane niż je przesyłać,
- poprzez zwiększenie wydajności procesora lub przez zwiększenie przydziału ilości czasu procesora w przypadku gdy czas ten był ograniczany. Możliwe jest również, gdy w systemie są dostępne dodatkowe procesory, przeniesienie obsługi domeny 0 na

dedykowany wydzielony procesor,

- zmniejszyć ilość koniecznych do przesłania danych — nawet w przypadku gdy obciążenie pamięci dla przenoszonej maszyny nie jest duże, proces migracji przenosi również zawartość stron nieużywanych, które zwykle nie zawierają[20] żadnych danych i są wypełnione zerami. Dane te można poddać procesowi kompresji, który pozwoli na kilkukrotne zmniejszenie ilości koniecznych do przesłania informacji kosztem większego obciążenia procesora.

Wszystkie testy zostały wykonane dla różnych wartości rozmiaru pamięci VM (w zakresie od 64 MB do 4 GB[21]) przydzielonej dla migrowanego systemu. Zmierzone wartości czasu procesu migracji oraz obciążenia sieci dla wybranych wielkości pamięci VM zostały zawarte w tabeli 7.6.

	Czas (sec)		
Pamięć VM	brak obciążenia	obciążenie CPU	obciążenie RAM
256 M	25.64	29.82	46.52
512 M	48.66	57.64	91.54
1 G	95.00	112.03	184.91
2 G	186.39	221.67	354.47
4 G	359.31	429.61	686.12

Tabela 7.6. Czas migracji względem rozmiaru pamięci i rodzaju obciążenia VM

Rysunek 7.10 przedstawia zależność pomiędzy rozmiarem pamięci przydzielonej dla VM a łącznym czasem migracji. Dla maszyny nieobciążonej oraz dla VM intensywnie wykorzystującej procesor zależność ta jest z dużą dokładnością liniowa. Przybliżając można oszacować czas migracji na podstawie wzoru:

$$T_m = T_c * m + T_0, \tag{7.21}$$

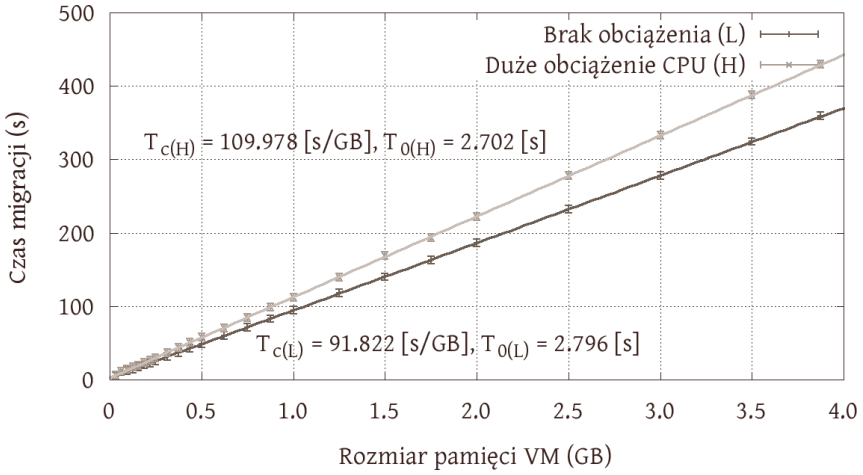
gdzie:

T_m — czas migracji,
m — rozmiar pamięci operacyjnej VM,
T_c, T_0 — współczynniki zależne od wydajności fizycznych węzłów, parametrów sieci oraz konfiguracji systemu Xen. Oszacowane wartości tych parametrów przedstawione zostały na wykresie 7.10.

W przypadku przenoszenia wykonania maszyny nieobciążonej zależność liniowa jest uzasadniona ustaleniem się czasu migracji do wartości wynikającej z czasu dla transmisji

[20] Strona używana przez proces a następnie zwolniona, będzie posiadała nieużywane już dane. Możliwe jest jednak dodanie, w funkcjonalności systemu zarządzania pamięci, odpowiednich rozszerzeń wykonujących czyszczenie (zerowanie) zwalnianych stron. Funkcjonalność ta nie będzie wprowadzała znaczącego dodatkowego narzutu, gdyż większość procesorów posiada optymalizowane instrukcje do wypełniania bloków pamięci stałą wartością.

[21] Maksymalny rozmiar pamięci przydzielonej dla VM wynosił 3968 MB z powodu zajęcia 128 MB przez system operacyjny domeny 0 (*supervisor*).



Rysunek 7.10. Czas migracji w zależności od rozmiaru pamięci VM

danych. Na rysunku 7.7 widać, że zarówno obciążenie generowane przez proces migracji (dla nieobciążonej VM) nie osiąga wartości maksymalnej. Maksymalne jest natomiast obciążenie interfejsu sieciowego, które wynosi[22]:

$$\frac{1\ GB * 8}{T_c} = \frac{8\ Gb}{91.552 \frac{s}{GB}} = 93.79 \frac{Mb}{s} \tag{7.22}$$

W przypadku gdy komunikacja sieciowa pozwala na przesyłanie danych szybciej niż są one generowane, dochodzi do całkowitego obciążenia procesora przez proces odbierający informacje.[23]

Czas migracji zależy zatem od następujących czynników:

- rozmiaru pamięci przenoszonej maszyny,
- czasu wykonania procesu odbierającego oraz wysyłającego dane,
- przepustowości sieci.

$$T_n = max\left(T_g(m_n),\ T_k(m_n)\right), \tag{7.23}$$

gdzie:

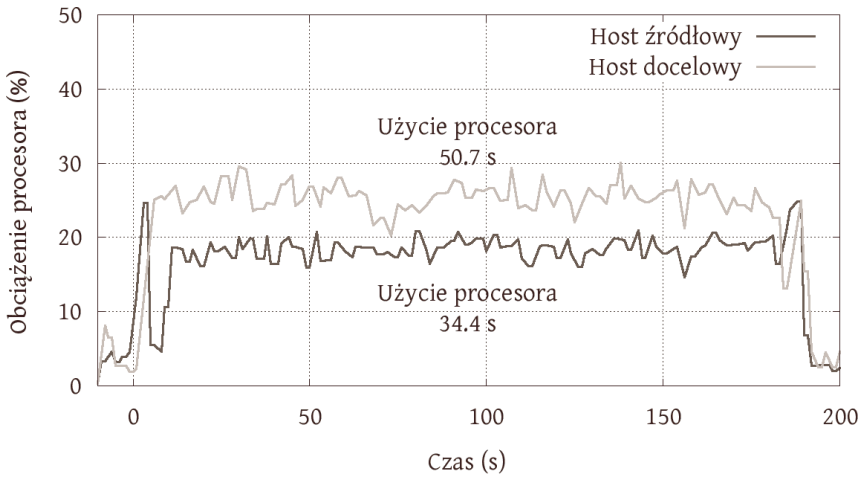
T_n — czas migracji n–tej maszyny wirtualnej,
m_n — rozmiar pamięci przenoszonej maszyny,
T_g — czas identyfikacji i generowania danych,
T_k — czas potrzebny do przesłania wygenerowanych w czasie T_g danych.

[22] Biorąc jedynie pod uwagę dane warstwy 4 modelu OSI/ISO. Oszacowanie to pomija narzut związany z nagłówkami warstw 2–4 oraz synchronizacją ramek Ethernet, wynoszący typowo ok 4.5 %.

[23] Proces identyfikujący i wysyłający informacje, podobnie jak w przypadku migracji dla wolnej sieci, będzie generował tylko ok. 50 % obciążenia procesora.

Chcąc ograniczać zasoby potrzebne do przeprowadzenia procesu migracji można wykorzystać trzy czynniki. Pierwszy z nich to przepustowość komunikacji — ograniczając obciążenie procesora hostów pomiędzy, którymi następuje migracja oraz ilość danych przesłania, zwiększając czas przerwy w działaniu VM i ogólny czas migracji. Drugą metodą jest modyfikacja parametrów wydajnościowych procesora obsługującego domenę 0, umożliwiając tym samym redukcję wpływu migracji na wydajność pozostałych wirtualnych maszyn uruchomionych w obrębie tego samego fizycznego węzła. Możliwe jest również zastosowanie mechanizmów pozwalających na zmniejszenie ilości przesyłanych informacji. Włączając kompresję przesyłanych danych możliwe jest znaczne ograniczenie czasu migracji kosztem większego obciążenia procesora.

Rysunek 7.11 przedstawia obciążenie procesorów domen 0, pomiędzy którymi następuje migracja, w przypadku gdy za pomocą mechanizmów systemu Xen, procesory obsługujące domeny 0 mają dostęp tylko do 25 % czasu procesora fizycznego. W sytuacji gdy przenoszona domena nie generowała obciążenia nastąpił wzrost ogólnego czasu migracji o około 68 %.



Rysunek 7.11. Obciążenie procesora dla migracji z ograniczeniem wykorzystania CPU

W systemie Xen są dostępne dwie implementacje algorytmów przydziału czasu procesora dla uruchomionych VM. Pierwsza z nich to implementacja używająca dwóch parametrów „*Weight*" oraz „*Cap*". Procent procesora jest obliczany na podstawie wzoru 7.24.

$$T_n = min\left(\frac{W_n}{\sum_{i=1}^{N} W_i}, \frac{1}{C_n}\right) * 100\%, \tag{7.24}$$

gdzie:

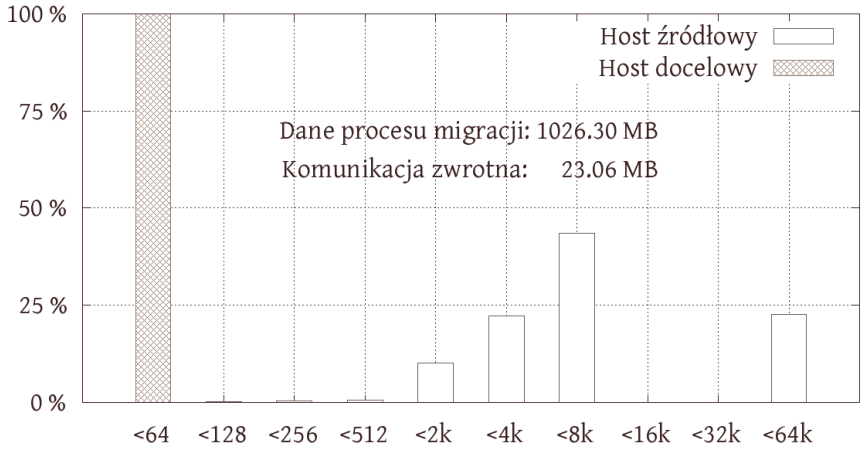
T_n — czas procesora dla wirtualnej maszyny n (procentowy udział czasu fizycznych procesorów, z których mocy obliczeniowych może korzystać domena),

W_n — waga, wartość umożliwiająca określenie priorytetów — podział czasu będzie zgodny z proporcją wynikająca z ilorazu wag,

C_n — wartość parametru „*Cap*", którego ustawienie na wartość z przedziału $(1, 100\rangle$ powoduje ustalenie górnego twardego limitu czasu na wartość $\frac{1}{Cap} * 100\%$.

Synchronizacja pamięci przenoszonego węzła, podczas migracji wymaga skopiowania (etapami) całego obszaru dostępnego dla systemu operacyjnego. Nawet w przypadku gdy część tej pamięci nie jest wykorzystywana przez uruchomione procesy czy system, obszar ten też będzie przesyłany. Ponieważ przy starcie domeny kontrolnej systemu Xen następuje zerowanie całej zawartości pamięci przydzielanej później dla wirtualnych węzłów, strony pamięci, które nie były wykorzystywane zawierają same zera.

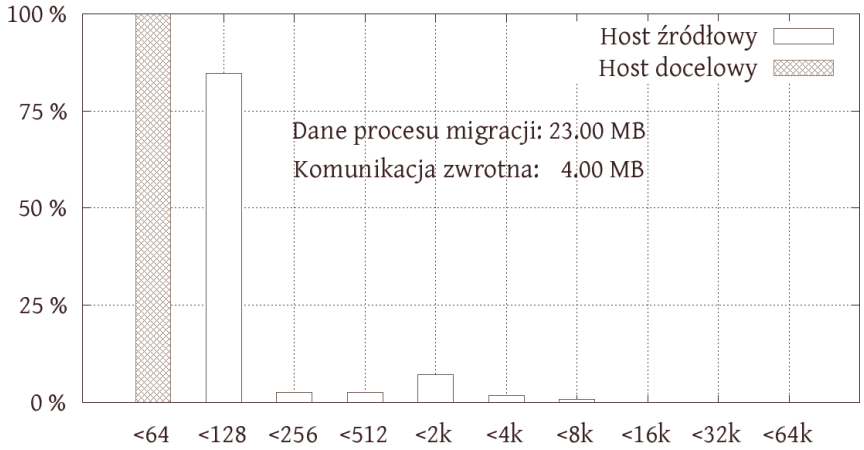
Wprowadzając kompresję informacji, przesyłanych podczas synchronizacji zawartości pamięci, można drastycznie zmniejszyć ilość danych do transmisji. Wykres 7.12 przedstawia histogram procentowego udziału ilości pakietów IP, w zależności od rozmiaru wysyłanych porcji danych, podczas migracji maszyny wirtualnej VM o rozmiarze pamięci 1 GB. Widać na nim znaczny udział pakietów IP o rozmiarach 4 kb i 8 kb. Wynika to z faktu operowania przy migracji na pamięci o organizacji w 4 kb strony. Komunikację zwrotną stanowią prawie wyłącznie pakiety przenoszące potwierdzenia protokołu TCP.



Rysunek 7.12. Rozmiar ramek protokołu IP w procesie migracji

Tunelowanie komunikacji sieciowej pozwala na użycie kompresji przesyłanych danych. Rysunek 7.13 przedstawia rozkład rozmiaru pakietów IP dla tunelowanej komunikacji z zastosowaniem oprogramowania *stunnel*[24] dla tego samego przypadku migracji.

[24]Zastosowany algorytm kompresji pochodził z biblioteki `zlib`.



Rysunek 7.13. Rozmiar ramek protokołu IP w procesie migracji dla kompresji algorytmem *Deflate*

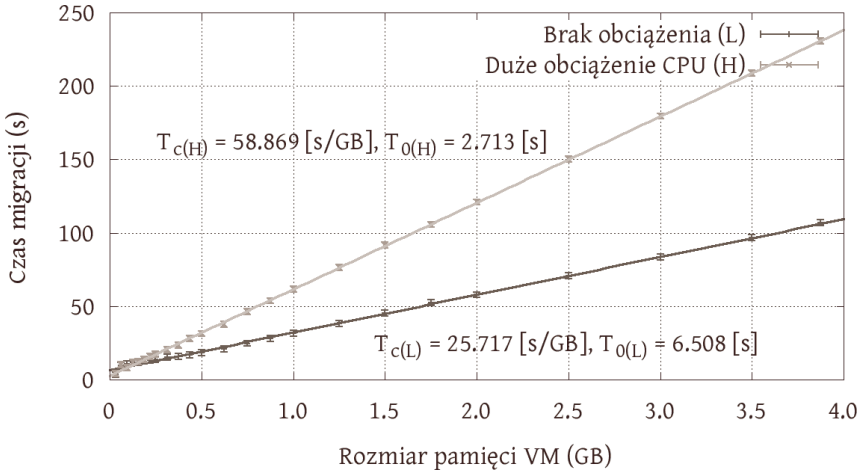
	Proces migracji	
Parametr	**bez kompresji**	**z kompresją**
Czas migracji	93.69 s	56.75 s
Liczba wysłanych pakietów	61251	65917
Liczba wysłanych bajtów	1026.30 MB	23.00 MB
Średni rozmiar pakietu	17569.49 B	365.84 B
Przepustowość wysyłania	93.53 Mbps	3.47 Mbps
Średnie obciążenie procesora	37.64 %	97.62 %
Liczba odebranych pakietów	366300	63590
Liczba odebranych bajtów	23.06 MB	4.00 MB
Średni rozmiar pakietu	66.00	66.01
Przepustowość odbierania	2.06 Mbps	0.598 Mbps
Średnie obciążenie procesora	73.38 %	57.91 %

Tabela 7.7. Porównanie parametrów migracji po zastosowaniu kompresji przesyłanych danych

7.3. Wyniki praktycznej weryfikacji właściwości systemu

Tabela 7.8 przedstawia wartości parametrów opisujących wykonanie aplikacji testowych w środowisku natywnym. Wartości te potwierdzają założenia odnośnie własności aplikacji testowych oraz stanowią wartości porównawcze używane przy testach wykonania z wykorzystaniem możliwości środowiska zarządzania zasobami wirtualnego Gridu.

We wszystkich pomiarach do zbierania wyników stosowane były narzędzia, których działanie powodowało dodatkowe obciążenie infrastruktury. Obciążenie procesora fizycznych węzłów wynikające z monitorowana wirtualnych maszyn wynosiło 3 – 5 %, natomiast pasmo komunikacyjne zajęte przez transmisje danych pomiarowych dochodziło do około 2.1 kbps. Obciążenie wprowadzane przez komponenty systemu zarządzania było zmierzone dla każdego testu niezależnie i jest przedstawione w rezultatach poszczególnych badań.



Rysunek 7.14. Czas migracji względem rozmiaru pamięci i rodzaju obciążenia VM dla sieci Ethernet 1Gbps

	Aplikacje testowe		
Parametr	A_1	A_2	A_3
Czas odpowiedzi — R_A	3820.5 s	3657.3 s	2807.4 s
Czas wykonania komponentów — $\overline{R_A}$	6120.0 s	1116.2 s	3633.4 s
Zysk zrównoleglenia — Z_A	1.602	0.305	1.323
Liczba wątków liczących	8.0	8.0	8.0
Wykorzystanie infrastruktury — U	6.67 %	1.27 %	5.52 %
Efektywne wykorzystanie infrastruktury — U'	20.02 %	3.81 %	16.55 %
Wykorzystanie infrastruktury komunikacyjnej — X_A	901.2 MB	2.64 GB	1.32 GB
Zajętość pasma komunikacyjnego — N_A	7.871 %	15.17 %	10.11 %

Tabela 7.8. Zestawienie parametrów opisujących wykonanie aplikacji testowych w środowisku natywnym

7.3.1. Eksperyment 1 — zadane rozmieszczenie VM

Pełna ocena właściwości i rezultatów działania systemu VGRMS (których rezultaty zawarte są w kolejnych sekcjach) wymaga aby dostępne były pomiary typowych operacji wykonywanych przez środowisko. Pomiary operacji tworzenia i usuwania wirtualnych Gridów oraz narzutów systemu działającego bez uruchomionych aplikacji obliczeniowych stanowią istotny punkt odniesienia przy ocenie pozostałych wyników pomiarów. Wyniki te pozwolą także do określenia stopnia skalowalności systemu przy wzroście ilości zarządzanych fizycznych węzłów czy aplikacji obliczeniowych.

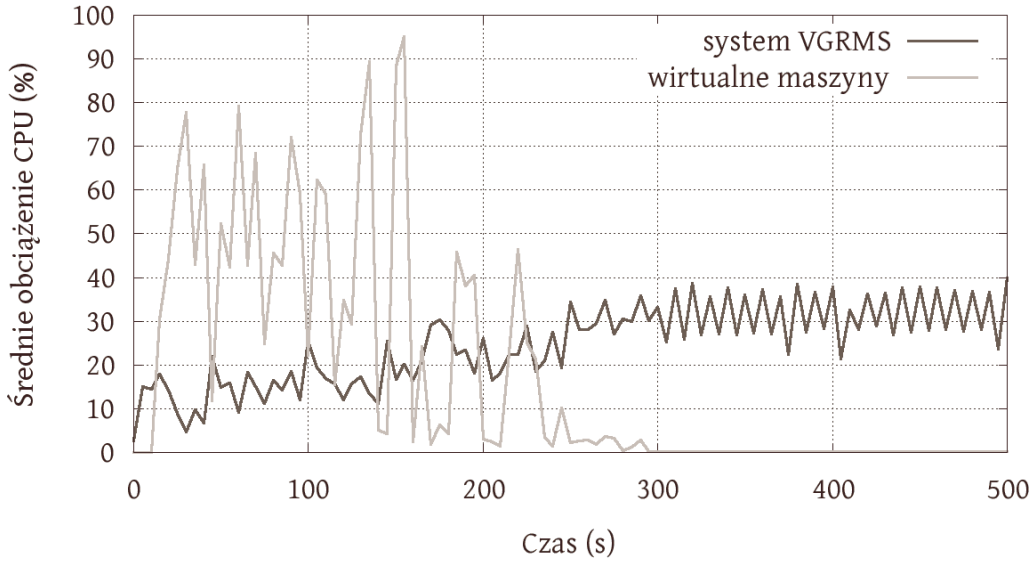
Tworzenie i usuwanie wirtualnych Gridów

Tabela 7.9 przedstawia wyniki pomiarów użycia procesora i infrastruktury komunikacyjnej podczas wykonywania procedury stworzenia równoczesnego od jednej do trzech instancji wirtualnego Gridu. W skład każdej instancji VG wchodziło 8 wirtualnych maszyn. Pomiary zostały wykonane na infrastrukturze składającej się z 9 komputerów PC, której topologia przedstawiona była na rysunku 7.1.

	Liczba instancji VG		
Parametr	**1**	**2**	**3**
Czas tworzenia VG	104.1 s	200.6 s	297.3 s
Zużycie czasu CPU (VM)	87.4 s	173.8 s	260.2 s
Zużycie czasu CPU (VGRMS)	176.7 s	350.5 s	523.9 s
Komunikacja sieciowa (VGRMS)	160.10 MB	311.46 MB	463.64 MB
Czas usuwania VG	21.9 s	44.9 s	64.5 s
Zużycie czasu CPU (VM)	60.8 s	121.1 s	180.7 s
Zużycie czasu CPU (VGRMS)	55.7 s	103.2 s	150.6 s
Komunikacja sieciowa (VGRMS)	15.7 MB	27.1 MB	38.39 MB

Tabela 7.9. Zestawienie parametrów opisujących proces tworzenia i usuwania instancji VG

Wykres 7.15 przedstawia sumaryczne obciążenie generowane podczas uruchamiania wirtualnych maszyn podczas tworzenia trzech instancji VG czyli w praktyce uruchomienia 24 wirtualnych maszyn.



Rysunek 7.15. Obciążenie procesora podczas wykonywania operacji tworzenia trzech instancji VG

Czas potrzebny na stworzenie topologii sieciowej wirtualnego Gridu wynosił zwykle

około 2–3 sekund, jednak efektywnie komunikacja sieciowa była możliwa dopiero po około 15 sekundach. Dodatkowa zwłoka związana była z mechanizmami sieciowymi takimi jak aktualizacje tablic przełączania oraz np. działaniem protokołu *spanning-tree* dla wirtualnych przełączników.

Obciążenie generowane przez komponenty systemu

Obciążenie to należy rozdzielić na obciążenie wnoszone przez komponenty działające na fizycznych węzłach używanych do wykonywania przetwarzania aplikacji oraz na obciążenie generowane przez konsolę zarządzania (obsługa VO)

Tabela 7.10 przedstawia parametry opisujące działanie systemu przy braku aplikacji obliczeniowych. Realizowanymi operacjami było tu głównie zbieranie i przetwarzanie informacji o stanie zasobów i komponentów systemu. Pomiary te zostały przeprowadzone dla dwóch ilości fizycznych węzłów, w każdym przypadku system był uruchomiony przez 1 godzinę.

	Liczba fizycznych węzłów	
Parametr	**4**	**8**
Czas trwania testu	3600.0 s	
Zużyty czas CPU (VM)	17.31 s	17.28 s
Zużyty czas CPU (VGRMS)	3823.9 s	7637.7 s
Zużyty czas CPU (konsola)	147.41 s	281.3 s
Komunikacja sieciowa (VM)	12.80 MB	12.22 MB
Komunikacja sieciowa (VGRMS)	394.53 MB	458.14 MB
Komunikacja sieciowa (konsola)	32.5 MB	61.4 MB

Tabela 7.10. Zestawienie parametrów opisujących stworzenie instancji VG

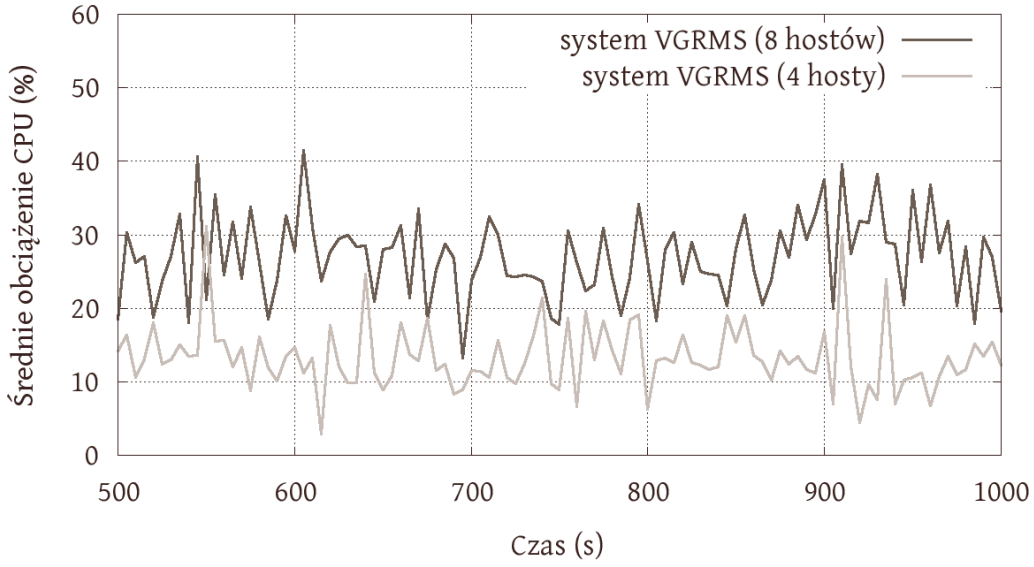
Wykres 7.16 przedstawia zależność obciążenia wnoszonego przez komponenty systemu od czasu.

Obsługa migracji bez wykonana aplikacji obliczeniowych

Wykonywana jest tutaj migracja wirtualnych maszyn tak aby ich rozmieszczenie było zgodne z przyjętymi założeniami (przedstawionymi przy opisie scenariusza tego eksperymentu). Przenoszone maszyny wirtualne, w czasie testu, nie wykonywały żadnych aplikacji obliczeniowych. Początkowe rozmieszczanie VM zostało tak dobrane aby konieczne było wykonanie jak największej ilości migracji koniecznych dla spełnienia założonych warunków rozmieszczenia. W trakcie testu wykonane zostało 9 migracji VM. Parametry opisujące działanie systemu w tym scenariuszu zostały zebrane w tabeli 7.11.

Wykres 7.17 przedstawia zależność obciążenia wnoszonego przez komponenty systemu w trakcie realizacji zadanych migracji VM.

Uzyskane wyniki pokazują, iż jest to w przybliżeniu zależność liniowa. Kolejne rezultaty przedstawiają właściwości procesu migracji wykonywanego z dużą częstością. Uzyskane wyniki korespondują z pomiarami czasu migracji wykonanego w środowisku bez kontroli systemu VGRMS.



Rysunek 7.16. Obciążenie procesora wnoszone przez działanie systemu VGRMS w teście nr 1

Parametr	Wartość
Czas trwania testu	542.6 s
Zużyty czas CPU (VM)	55.49 s
Zużyty czas CPU (VGRMS)	2171.0 s
Zużyty czas CPU (konsola)	48.8 s
Komunikacja sieciowa (VM)	1.70 MB
Komunikacja sieciowa (VGRMS)	2412.1 MB
Komunikacja sieciowa (konsola)	11.05 MB

Tabela 7.11. Zestawienie parametrów opisujących działanie systemu podczas optymalizacji rozmieszczenia VM

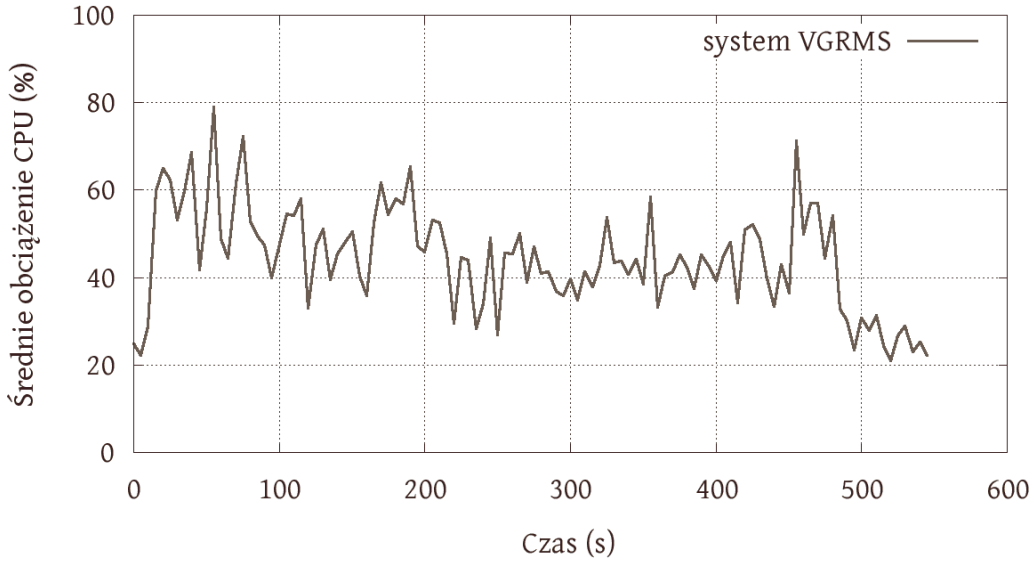
7.3.2. Eksperyment 2 — Optymalizacja na podstawie ścieżek komunikacyjnych

Wyniki tego eksperymentu ukazują działanie aplikacji w dwóch przypadkach. Pierwszy z nich to działanie bez kontroli systemu VGRMS w sytuacji gdy rozmieszczenie wykonania VM powoduje ograniczenie przepustowości[25] dla komunikacji sieciowej wymienianej podczas realizacji aplikacji testowej. Przypadek drugi polega na uruchomieniu aplikacji[26] pod kontrolą systemu VGRMS, który to z wykorzystaniem migracji VM dokona poprawy rozmieszczenia wirtualnych maszyn.

Eksperyment ten pokazuje czy opłacalne jest z punktu widzenia użytkownika (efektem oczekiwanym jest skrócenie czasu odpowiedzi aplikacji) przeniesienie wykonania wirtual-

[25] Część wirtualnych maszyn dostępna jest przez stosunkowo wolną sieć WAN.

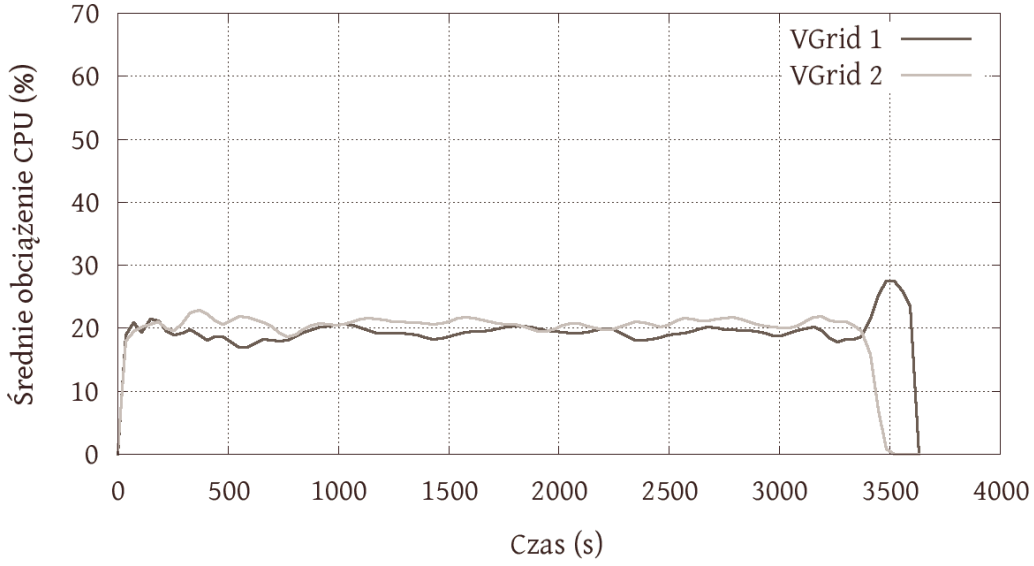
[26] W eksperymencie tym była wykorzystana aplikacja A_3 (posiadająca średnie zapotrzebowanie na zasoby komunikacyjne).



Rysunek 7.17. Obciążenie zasobów obliczeniowych podczas wykonywania zmian rozmieszczenia VM w teście nr 1

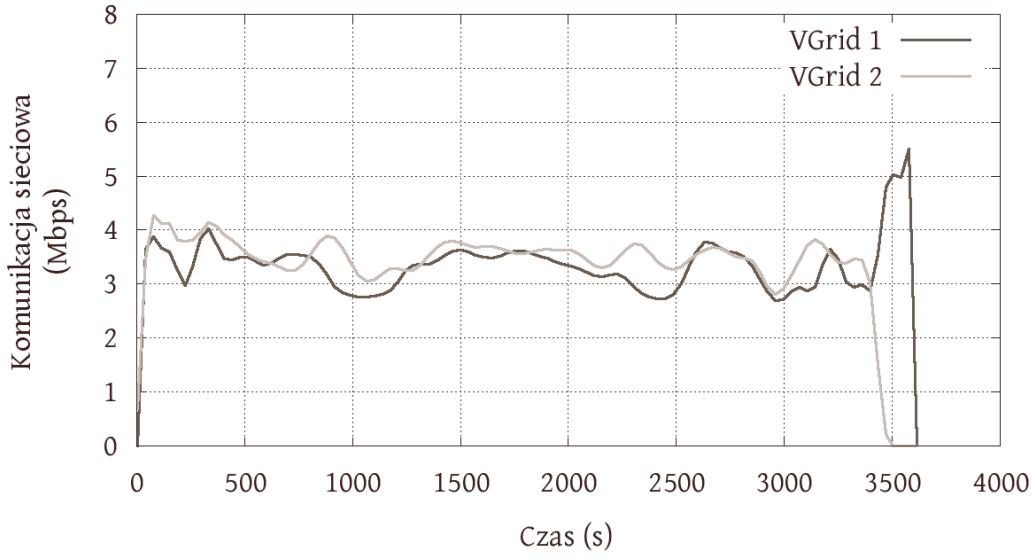
nych maszyn tak aby komunikacja pomiędzy nimi była jak najbardziej wydajna.

Wykres 7.18 przedstawia sumaryczne obciążenie zasobów obliczeniowych generowane przez oba uruchomione równocześnie wirtualne Gridy. Działanie systemu w tym przypadku było zaniechane, dlatego wykres ten nie uwzględnia obciążenia generowanego przez komponenty systemu.



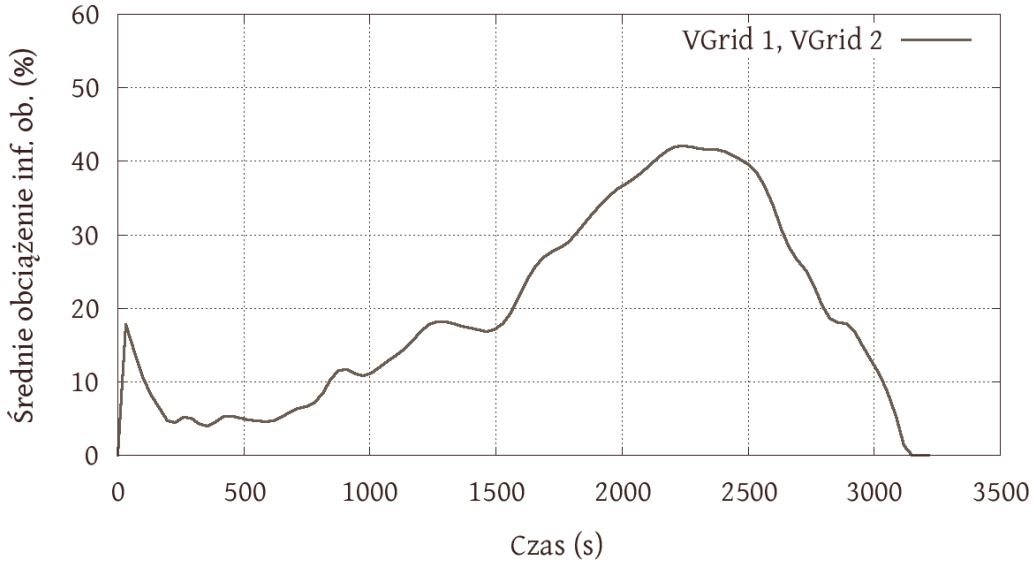
Rysunek 7.18. Obciążenie infrastruktury obliczeniowej przez dwie równocześnie działające instancje VG w teście nr 2

Parametry wykonania aplikacji testowych pod kontrolą systemu przedstawia wy-



Rysunek 7.19. Obciążenie infrastruktury komunikacyjnej przez dwie równocześnie działające instancje VG w teście nr 2

kres 7.20.



Rysunek 7.20. Obciążenie CPU wnoszone przez dwie instancje VG oraz przez system VGRMS dla testu nr 2

Na wykresie 7.20 widać, iż obciążenie wnoszone przez aplikacje testowe zależy od operacji wykonywanych przez system VGRMS. W fazie początkowej, w której wykonywane są migracje dostęp do zasobów jest ograniczony i dlatego aplikacja osiąga wydajność mniejszą niż w sytuacji wykonania bez optymalizacji. Po zakończeniu procesów migracji obserwowany jest znaczny wzrost wydajności dzięki temu, iż pomiędzy komponentami aplikacji

znikło wąskie gardło w postaci sieci WAN. Sumarycznym rezultatem jest skrócenie czasu odpowiedzi[27] aplikacji.

W tabeli 7.12 zestawione zestawienie zmian parametrów opisujących wykonanie aplikacji testowych w sytuacji gdy system grupował wykonanie VM na podstawie ścieżek komunikacyjnych.

Parametr	**Optymalizacja wyłączona**	**Optymalizacja włączona**
Czas odpowiedzi — R_A	3429.7 s 3622.6 s	3085.9 s 3110.2 s
Czas wykonania komponentów — $\overline{R_A}$	3633.0 s 3634.6 s	3634.1 s 3632.8 s
Zysk zrównoleglenia — Z_A	1.06 1.01	1.17 1.16
Liczba wątków liczących Liczba fizycznych węzłów (użytych)	8.0 6.0 (6)	8.0 6.0 (3–6)
Wykorzystanie dostępnej infrastruktury — U	17.64 % 16.76 %	19.06 % 19.04 %
Efektywne wykorzystanie infrastruktury — U'	17.64 % 16.76 %	22.72 % 22.70 %
Wykorzystanie infrastruktury komunikacyjnej — X_A	1323.22 MB 1323.18 MB	1323.27 MB 1323.19 MB
Zajętość pasma komunikacyjnego — N_A	64.71 %	81.01 %
Zużycie czasu CPU przez system VGRMS Komunikacja generowana przez system Tunel pomiędzy sieciami LAN[28]	5464.2 s 1289.65 MB 485.64 MB	6866.5 s 4647.58 MB 1259.91 MB

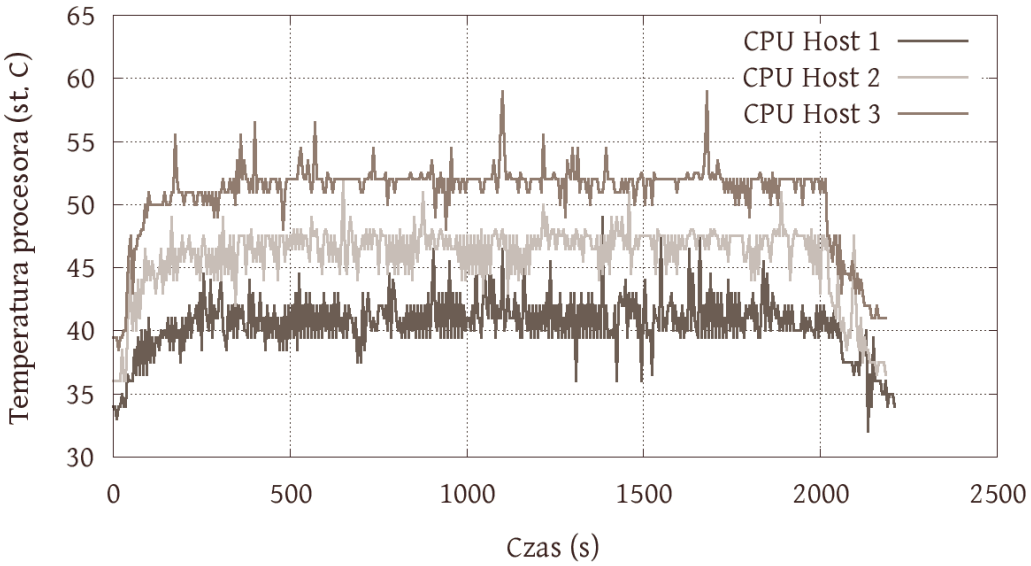
Tabela 7.12. Porównanie parametrów opisujących równoczesne działanie dwóch instancji aplikacji A_3 przy użyciu rozmieszczenia VM na podstawie ścieżek komunikacyjnych

W eksperymencie tym decyzje o migracji VM były podejmowane były na podstawie ścieżek komunikacyjnych. Test ten pokazuje właściwości związane z dynamicznych wykorzystaniem dodatkowych modułów monitorowania infrastruktury takich jak monitor komunikacji sieciowej oraz monitor środowiska. W obu przypadkach zostały zmierzone i przedstawione narzuty systemu VGRMS. Monitorowanie ścieżek komunikacyjnych umożliwiło podjęcie trafnych decyzji o migracji wirtualnych maszyn, których wykonanie skutkowało skróceniem czasu odpowiedzi aplikacji (tabela 7.12). Wynik ten uzyskany został poprzez zwiększenie (kilkukrotne) wykorzystania infrastruktury komunikacyjnej.

[27] Zysk ten byłby bardziej znaczący dla aplikacji o długim czasie wykonania.

7.3.3. Eksperyment 3 — Awaria fizycznego węzła

Celem tego eksperymentu było ukazanie możliwości systemu w zakresie wykrywania i reagowania na awarie infrastruktury. Na wykresie 7.21 pokazany został wpływ obciążenia na temperaturę wewnętrznych komponentów komputera.

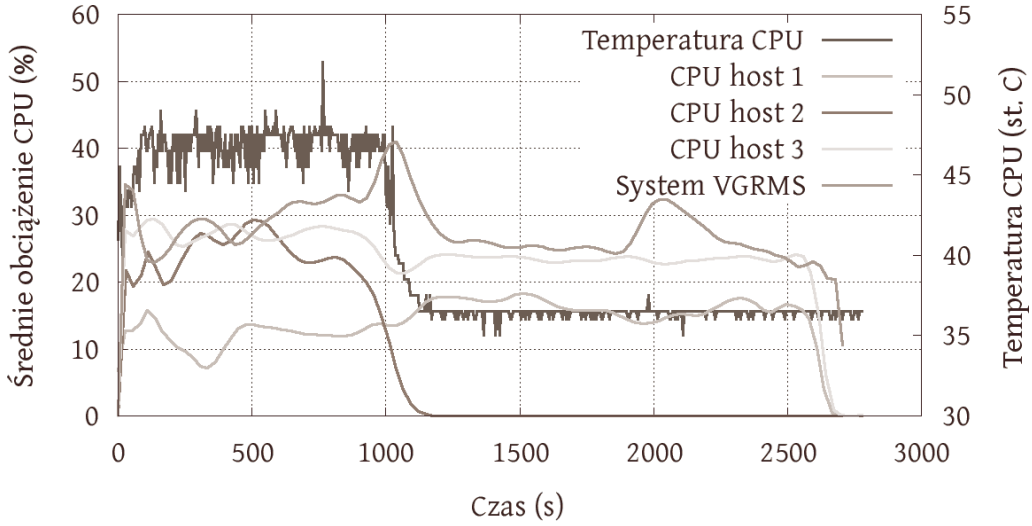


Rysunek 7.21. Zmiana temperatury procesora podczas wykonywania aplikacji testowej dla scenariusza z awarią układu chłodzenia procesora

W wykres 7.22 przedstawia obciążenie trzech fizycznych węzłów w trakcie wykonania aplikacji testowej w sytuacji gdy awarii uległ wentylator jednego z procesorów. Obciążenie wnoszone przez komponenty systemy VGRMS zostało ukazane za pomocą wykresu pełnego. Obciążenie to związane jest z obsługą procesu migracji wirtualnych maszyn z uszkodzonego fizycznego węzła. Zmniejszenie obciążenia jednego z fizycznych węzłów (poprzez migrację VM wykonujących obliczenia na inne) spowodowało powrót wartości temperatury procesora do wartości początkowych. Samo wykonanie aplikacji było w tym przypadku dłuższe od typowego gdyż po czasie ok. 1000 sekund dostępne było mniej fizycznych węzłów (zasobów) w danym wirtualnym Gridzie.

Tabela 7.13 zawiera podsumowanie uzyskanych w teście 3 wyników. Porównane są sytuacje wykonania aplikacji, w sytuacji wystąpienia awarii oraz parametry dla wykonania niezakłóconego.

W teście z awarią fizycznego węzła migracja umożliwiła kontynuowanie obliczeń mimo iż awarii uległ jeden z węzłów. Przeniesienie wykonania VM na pozostałe sprawne maszyny wydłużyło czas działania aplikacji (mniej dostępnych zasobów) ale spowodowało iż wykonanie obliczeń pozostało niezakłócone. Dzięki temu np. powtórzenie wykonania aplikacji nie było konieczne (co niekiedy dla czasochłonnych aplikacji jest znaczną oszczędnością czasu i zasobów).



Rysunek 7.22. Obciążenie fizycznych hostów generowane przez aplikację oraz system dla scenariusza z awarią węzła

Parametr	Bez awarii	Z awarią węzła
Czas odpowiedzi — R_A	2042.36 s	2621.31 s
Czas wykonania komponentów — $\overline{R_A}$	3620.6 s	3623.1 s
Zysk zrównoleglenia — Z_A	1.773	1.382
Liczba wątków liczących	8.0	8.0
Wykorzystanie dostępnej infrastruktury — U	59.09 %	46.07 %
Efektywne wykorzystanie infrastruktury — U'	59.09 %	57.85 %
Wykorzystanie infrastruktury komunikacyjnej — X_A	901.22 MB	901.31 MB
Zajętość pasma komunikacyjnego — N_A	3.70 %	2.88 %
Obciążenie wnoszone przez system	1661.7 s	2183.9 s
Komunikacja generowana przez system	50.72 MB	322.41 MB

Tabela 7.13. Porównanie parametrów opisujących działanie aplikacji A_3 w sytuacji wystąpienia awarii oraz dla wykonania niezakłóconego

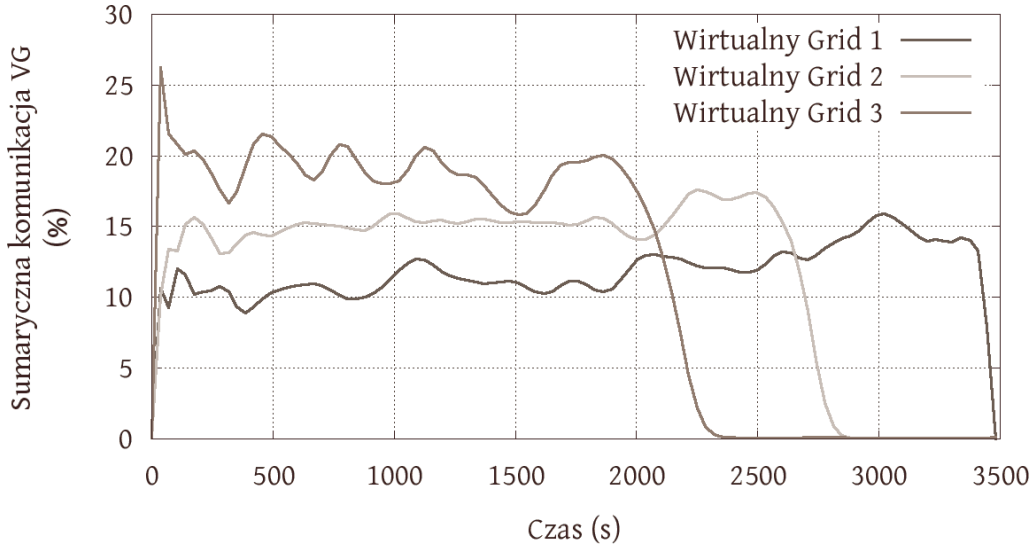
7.3.4. Eksperyment 4 — Zastosowanie gwarancji QoS

Celem eksperymentu było wykazanie możliwości działania systemu w zakresie gwarancji QoS. Wykres 7.23 przedstawia sumaryczne obciążenie wirtualnych maszyn z trzech wirtualnych Gridów w trakcie wykonania poszczególnych aplikacji w przypadku gdy ograniczeniom podlegało zużycie czasu procesora[29].

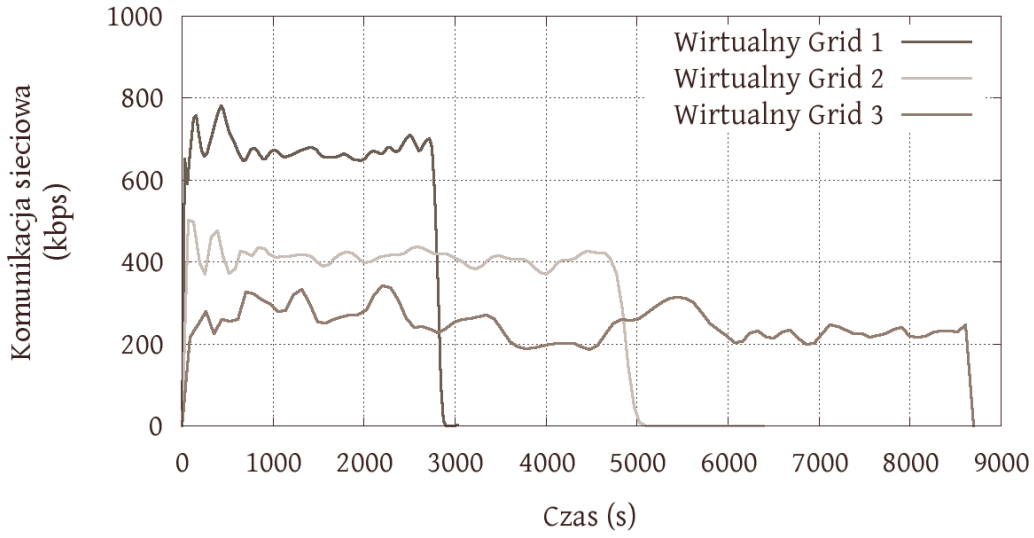
Kolejny wykres (7.24) przedstawia rozkład w czasie obciążenia infrastruktury sieciowej w przypadku gdy ograniczeniom podlegały również zasoby komunikacyjne przydzielone dla danych wirtualnych Gridów zgodnie z przyjętą proporcją (7.1).

Wartości metryk opisujących wykonanie aplikacji zebranych podczas wykonywania ba-

[29] Wykorzystano kod źródłowy B.3 oraz kod źródłowy B.4 zawarty w dodatku.



Rysunek 7.23. Sumaryczne obciążenie wirtualnych Gridów podczas testów gwarancji QoS dla aplikacji A_3



Rysunek 7.24. Obciążenie infrastruktury komunikacyjnej podczas testów gwarancji QoS dla aplikacji A_3

dań przedstawia tabela 7.14.

Testy możliwości w zakresie gwarancji QoS oraz optymalizacji wykorzystania infrastruktury z zastosowaniem algorytmu wyrównania obciążenia ukazują główne możliwości systemu VGRMS. Możliwości tez związane są z wykonywaniem regulacji w dostępie do zasobów w taki sposób aby spełnić przyjęte wymagania.

Test gwarancji QoS jest przypadkiem w którym system dokonuje regulacji przydziału zasobów dla aplikacji na podstawie zdefiniowanych klas jakości usługi. Wykorzystany przez autora algorytm bazował na dynamicznej aktualizacji współczynników określających do-

Parametr	Wirtualny Grid		
	VG 1	**VG 2**	**VG 3**
Ograniczenie zasobów obliczeniowych			
Czas odpowiedzi — R_A	2186.0 s	2731.0 s	3447.5 s
Czas wykonania komponentów — $\overline{R_A}$	3716.4 s	3716.3 s	3717.1 s
Efektywne wykorzystanie infrastruktury — $U'(PH,\ A)$	13.47 %	16.67 %	17.01 %
Proporcje	1.000	1.249	1.577
Obciążenie generowane przez VGRMS		7467.3 s	
Wykorzystanie infrastruktury komunikacyjnej — X_A	1.32 GB	1.32 GB	1.32 GB
Zajętość pasma komunikacyjnego — N_A	9.31 %	9.31 %	9.31 %
Komunikacja generowana przez VGRMS		146.5 MB	
Ograniczenie zasobów komunikacyjnych			
Czas odpowiedzi — R_A	2817.3 s	4881.2 s	8683.1 s
Czas wykonania komponentów — $\overline{R_A}$	3717.1 s	3716.5 s	3716.3 s
Efektywne wykorzystanie infrastruktury — $U'(PH,\ A)$	16.48 %	9.52 %	5.35 %
Proporcje	1.000	1.733	3.082
Obciążenie generowane przez VGRMS		7130.6 s	
Wykorzystanie infrastruktury komunikacyjnej — X_A	1.32 GB	1.32 GB	1.32 GB
Zajętość pasma komunikacyjnego — N_A	93.8 %	54.1 %	30.4 %
Komunikacja generowana przez VGRMS		135.4 MB	

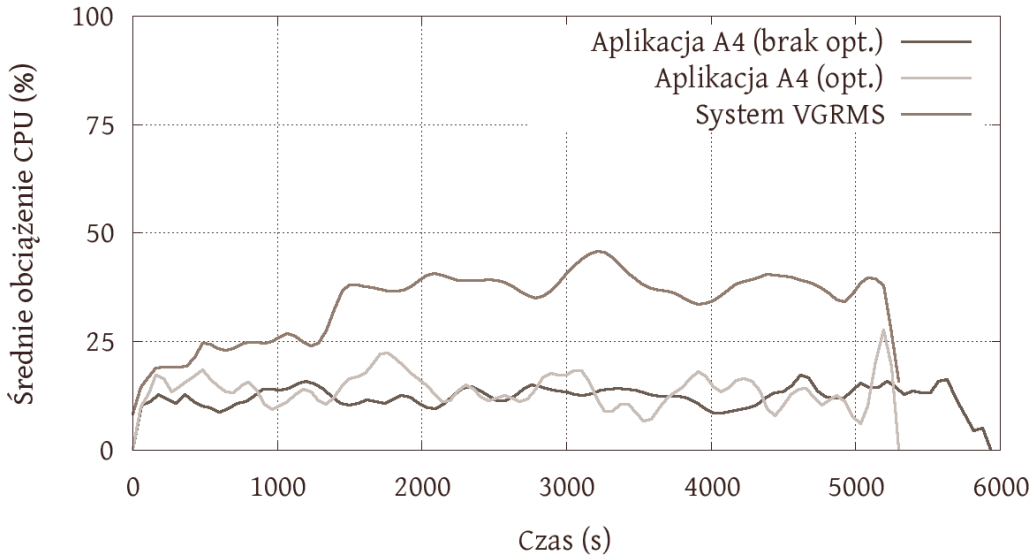
Tabela 7.14. Porównanie parametrów opisujących wykonanie aplikacji A_3 z narzuconymi ograniczeniami QoS

stępny poziom zasobów na podstawie aktualnego na nie zapotrzebowania od wszystkich działających równolegle aplikacji. Zebrane wyniki pokazały, iż możliwe jest spełnienie gwarancji określających dostęp do zasobów współdzielonej infrastruktury obliczeniowej. Widoczne jest to na wykresie 7.23 który przedstawia średnie obciążenie procesora dla trzech wirtualnych Gridów, które w czasie trwania eksperymentu było zbliżone do przyjętej proporcji.

7.3.5. Eksperyment 5 — Wykorzystanie migracji VM

Badania te miały na celu wykazanie efektów jakie daje daje możliwość przenoszenia wirtualnych maszyn w trakcie wykonania. W tym celu, w systemie oprócz aplikacji wykorzystywanych w teście gwarancji QoS, wykorzystane zostaną również aplikacje, których generowane obciążenie będzie nierównomiernie rozłożone.

Wykres 7.25 przedstawia średnie obciążenie zasobów podczas wykonania aplikacji testowej. Dodatkowo wykres ten przedstawia wykonanie aplikacji oraz obciążenie infrastruktury podczas wykonania algorytmu dążącego do równego rozłożenie obciążenia fizycznych hostów.

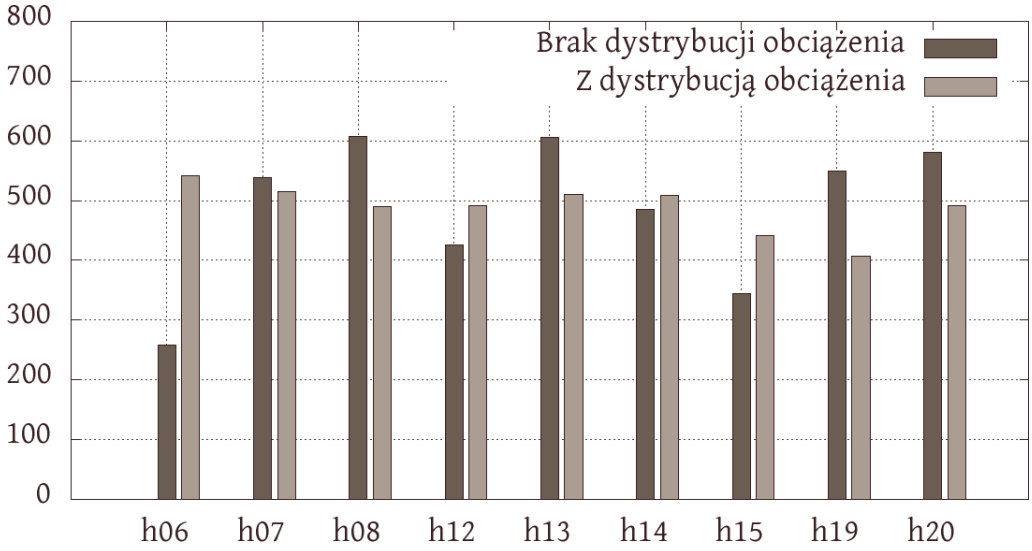


Rysunek 7.25. Obciążenie generowane przez aplikację testową A_4 bez i z kontrolą wykorzystania zasobów przez systemu VGRMS

Porównanie właściwości aplikacji testowej dla której był użyty algorytm rozłożenia obciążenia (rysunek 7.26) poprzez określenie poziomu obciążenia każdego z węzłów infrastruktury obliczeniowej.

Tabela 7.15 przedstawia parametry wykonania aplikacji testowej.

Test z algorytmem rozłożenia obciążenia pozwala zweryfikować czy dodatkowe narzuty związane z zastosowaniem technik parawirtualizacji oraz wykonywaniem procesów migracji mimo wszystko pozwolą na zwiększenie efektywności wykorzystania infrastruktury obliczeniowej. Zostały tutaj wykorzystane aplikacja w których komponenty generowały niejednorodne obciążenie. Prosta optymalizacja polegająca na grupowaniu na fizycznych



Rysunek 7.26. Rozłożenie użycia zasobów dla aplikacji testowej A_4 przed i po użyciu algorytmów dystrybucji obciążenia

	Algorytm dystrybucji	
Parametr	**wyłączony**	**włączony**
Czas odpowiedzi — R_A (s)	5936.2 s	5264.6 s
Czas wykonania komponentów — $\overline{R_A}$ (s)	4397.2 s	4398.7 s
Efektywne wykorzystanie infrastruktury — $U'(PH,\ A)$	12.35 %	13.92 %
Obciążenie generowane przez komponenty systemu	8312.9 s	10758.6 s
Wykorzystanie infrastruktury komunikacyjnej — X_A	1052.31 MB	1052.37 MB
Zajętość pasma komunikacyjnego — N_A	1.48 %	1.68 %
Komunikacja generowana przez komponenty systemu	173.61 MB	2335.18 MB

Tabela 7.15. Porównanie parametrów opisujących wykonanie aplikacji A_4 z zastosowaniem algorytmu dystrybucji obciążenia

węzłach komponentów o niskim zapotrzebowaniu na zasoby a rozdzielaniu komponentów z zapotrzebowaniem dużym, co pozwoliło na uzyskanie zamierzonych rezultatów.

7.4. Podsumowanie

Zaproponowana w niniejszym rozdziale metodologia badań oraz zrealizowane scenariusze testów i zamieszczone rezultaty ukazują praktyczne aspekty funkcjonowania prototypowej implementacji systemu VGRMS. Celem tych badań było przekonanie się czy przyjęta w tezie pracy koncepcja zastosowania technik parawirtualizacji jako efektywnego sposobu realizacji regulacji dostępu do zasobów oraz koncepcja powiązania aplikacji obliczeniowej z zasobami poprzez dedykowane środowisko wykonawcze — sprawdza się praktyce.

Przedstawione w rozdziale badania właściwości systemu VGRMS można podzielić na

dwie grupy. Pierwszą z nich stanowią testy mające za zadanie zademonstrować możliwości systemu takie jak tworzenie i zatrzymywanie wirtualnych Gridów oraz proste działania na ich konfiguracji[30]. W tym celu zaproponowane zostały trzy scenariusze testów. Pierwszy z nich ukazuje narzuty systemu VGRMS czyli czas oraz zużycie zasobów potrzebnych na stworzenie i uruchomienie zestawu wirtualnych maszyn oraz połączenie ich za pomocą wirtualnej sieci. Autor starał się oszacować również skalowalność systemu czyli zmierzyć jak zmienia się obciążenie generowane przez system wraz ze wzrostem ilości zarządzanych fizycznych węzłów, maszyn wirtualnych i wirtualnych Gridów.

Wszystkie testy wykonane były z wykorzystaniem odpowiednio dobranych przez autora aplikacji. Jednak w systemie możliwa jest implementacja niemal dowolnej heurystyki przydziału zasobów dostosowanej do właściwości konkretnej aplikacji.

Zamieszczone wyniki praktycznej ewaluacji środowiska pozwalają na stwierdzenie, iż możliwe było skonstruowanie systemu, który implementując przyjęte koncepcje spełniał określone na wstępie założenia. Zaprezentowane wyniki potwierdzają również poprawność wyboru technologii i koncepcji zastosowanych podczas tworzenia implementacji systemu VGRMS.

[30]Możliwe jest także dzięki tym scenariuszom ukazanie łatwości z jaką administrator może implementować za pomocą zestawu reguł politykę działania wirtualnego Gridu.

Rozdział 8

Podsumowanie

Rozprawa prezentuje podejście do zarządzania zasobami środowisk rozproszonych z wykorzystaniem technik wirtualizacji. Opracowana i zaimplementowana architektura środowiska stanowi oryginalne rozwiązanie. Prace przedstawione w niniejszej rozprawie stanowią potwierdzenie słuszności przyjętych na wstępie założeń odnośnie efektywności zarządzania zasobami dla aplikacji uruchomionych w dedykowanym środowisku wykonawczym — wirtualnym Gridzie.

Praktyczna implementacja architektury systemu oraz jego ewaluacja demonstruje właściwości opracowanego modelu zarządzania zasobami. Autor wykonał działającą instalację dokonując uzupełnienia dostępnych technologii o dodatkowe komponenty, konieczne dla realizacji wspomnianych wyżej wymogów. Sprawne działanie systemu, jego stabilność oraz posiadanie funkcjonalności umożliwiającej poprawę parametrów jakościowych opisujących działanie aplikacji rozproszonych stanowi dla autora sporą satysfakcję.

8.1. Weryfikacja tezy pracy

Celem niniejszej pracy było zbadanie w jaki sposób dostępne mechanizmy wirtualizacji można zastosować do budowy zwirtualizowanych środowisk Grid. Istotnym zagadnieniem była również weryfikacja na ile zastosowanie technik wirtualizacji jest skutecznym mechanizmem pozwalającym na realizację wybranych strategii przydziału zasobów w środowiskach rozproszonych. Aby móc odpowiedzieć na te pytania autor opracował model systemu, który zakładał uzupełnienie odpowiednich komponentów dla istniejących rozwiązań umożliwiających realizację zakładanej funkcjonalności. Model ten został praktycznie zweryfikowany z wykorzystaniem prototypowej implementacji systemu zarządzania zasobami VGRMS. Zostały zaproponowane i zrealizowane scenariusze ewaluacyjne, które na przygotowanej infrastrukturze testowej pozwoliły przekonać się o praktycznych właściwościach przyjętej na wstępie pracy koncepcji.

Przedstawione w rozdziale 7 wyniki uzyskane z praktycznej ewaluacji pokazują dla aplikacji testowych, na ile możliwe jest dynamiczne dostosowanie konfiguracji środowiska wykonawczego tak aby odpowiadało ono wymaganiom aplikacji. Zdaniem autora uzyskane rezultaty, zarówno funkcjonalne (w postaci unikalnych możliwości, których zapewnienie w klasycznych instalacjach jest trudne, bądź nieefektywne) oraz zysk widoczny w poprawie wartości metryk opisujących wykonanie aplikacji, stanowią o słuszności podjęcia się praktycznej realizacji zaproponowanej tematyki.

Przyjęte na wstępie założenia w postaci określonego poziomu wirtualizacji, sposobu reprezentacji zasobów i możliwości wymaganych podczas konstrukcji polityki przydziału zasobów w hierarchicznym modelu zarządzania, zostały potwierdzone praktycznie. Dodat-

kowo potwierdziła się koncepcja powiązania aplikacji z dedykowanym środowiskiem wykonawczym — wirtualnym Gridem, co skutkowało wieloma przedstawionymi w rozdziale 7 zaletami.

Podsumowując wykorzystanie implementacji zaproponowanej w pracy koncepcji przyniosło wymagane rezultaty i spełnienie przedstawionych w rozdziale 3 wymagań. A zatem autor wykazał, iż poprzez zastosowanie technik wirtualizacji możliwe było opracowanie skutecznego narzędzia budowy wirtualnych Gridów, których zasoby mogą być przydzielane na podstawie określonych polityk, co dowodzi spełnienia tezy rozprawy.

8.2. Nowatorstwo pracy

Zaproponowane rozwiązanie wprowadza nowe podejście do koncepcji zarządzania zasobami w środowiskach rozproszonych, które w odniesieniu do klasycznych rozwiązań posiada wiele istotnych cech. Większość podejść prezentowanych w literaturze tematu stosuje jedynie jedną z technik: wirtualizację zasobów obliczeniowych bądź wirtualizację sieci komputerowej. Zaproponowane rozwiązanie łączy wirtualizację zasobów obliczeniowych oraz wirtualizację systemów komunikacyjnych, dzięki temu uzyskuje się pełną wirtualizację zasobów w środowiskach Grid. Oparcie prezentowanego rozwiązania o wirtualizację obu technik równocześnie umożliwiło stworzenie spójnego systemu zgodnego z przyjętymi kryteriami.

Koncepcja systemu zaproponowana w niniejszej pracy polegała na uzupełnieniu możliwości jakie niesie zastosowanie wirtualizacji (regulacja wykorzystania zasobów, izolacja wykonania) o funkcjonalność (dodaną poprzez dobudowanie odpowiednich komponentów) umożliwiającą autonomiczne wykorzystanie tych możliwości. W pracy zaproponowano także mechanizm dynamicznego zarządzania opracowanymi w systemie VGRMS komponentami. Zarządzanie to może być realizowane zarówno w sposób autonomiczny pod kontrolą silników regułowych bądź manualnie za pomocą graficznego interfejsu użytkownika.

System VGRMS jest w pełni przygotowany do działania zgodnie z ograniczeniami narzuconymi na przydział zasobów wynikający z realizacji wirtualnego Gridu w ramach wirtualnych organizacji. W tym sensie uzyskane rozwiązanie jest kompleksowe i wpisuje się w szerszy obraz idei realizacji obliczeń w dużych systemach rozproszonych współdzielonych przez wiele instytucji. W systemach tych obecnie trudno jest uzyskać gwarancje alokacji zasobów na określonym poziomie. Praca pokazuje sposób uzyskania gwarancji jakości wykonania dla aplikacji, co w aspekcie wykorzystania technik wirtualizacji stanowi interesujące rozwiązanie.

Pozostałe ważne cechy systemu to możliwość zdefiniowania prawie dowolnej heurystyki podziału zasobów, separacja logiki zarządzania zasobami od aplikacji użytkownika oraz brak konieczności dostosowywania aplikacji. Rozwiązania opracowane w zakresie wirtualizacji infrastruktury mogą być z powodzeniem wykorzystane w nowoczesnych systemach oprogramowania zrealizowanych zgodnie z paradygmatem SOA.

Poprawność wykorzystania nowoczesnych narzędzi i środowisk programistycznych, a także użycie koncepcji separacji funkcjonalności poszczególnych komponentów środowiska zostało potwierdzone praktyczną weryfikacją funkcjonalności zgodnej z przyjętymi założeniami. W obecnych instalacjach Grid techniki wirtualizacji nie odgrywają jeszcze znaczącej roli, jednak zdaniem autora, sytuacja ta nie będzie trwała długo, gdyż technologia ta z powodzeniem może być stosowana jako metoda rozwiązywania licznych problemów

związanych ze skalowalnością stosowanych obecnie rozwiązań przydziału zasobów implementowanych w warstwach pośredniczących.

8.3. Braki systemu

W zaimplementowanym systemie z pewnością możliwe i potrzebne są prace związane z optymalizacją i usprawnieniem jego działania zarówno jeśli chodzi o poziom wykorzystywanych przez sam system zasobów jak i czas wykonywania poszczególnych operacji oraz stabilność całości rozwiązania. W środowisku laboratoryjnym występowały problemy związane z małą skalowalnością niektórych przyjętych rozwiązań.[1]

W przedstawionym rozwiązaniu nie został położony silny nacisk na zapewnienie odpowiedniego poziomu bezpieczeństwa. Konieczne są zatem prace związane z uzupełnieniem mechanizmów uwierzytelnienia użytkowników i autoryzacji ich działań, a także zapewnieniem poufności komunikacji danych[2]. Prace te są konieczne w sytuacji gdy system będzie wykorzystywany w środowisku produkcyjnym.

8.4. Możliwości dalszego rozwoju

Podczas wykonywania prac związanych z praktyczną oceną możliwości systemu, autor zaproponował środowisko testowe, aplikacje i scenariusze testów tak, aby przedstawić możliwości jakie daje proponowane rozwiązanie. W szczególności stosowane algorytmy optymalizacji zaimplementowane w postaci polityki przydziału zasobów dla modułu decyzyjnego nie były zaawansowane. Dlatego zdaniem autora możliwe jest jeszcze przeprowadzenie wiele ciekawych eksperymentów związanych z rozbudową logiki działania środowiska.

Wprowadzenie mechanizmów pozwalających na bardziej złożoną analizę i przetwarzanie informacji o stanie pozwoliłoby na konstrukcję bardziej optymalnych heurystyk procesów optymalizacji. Jednak zmniejszenie prawdopodobieństwa podjęcia błędnych decyzji optymalizacji rozmieszczenia zasobów wymaga dokładnych danych opisujących aktualny stan ich wykorzystania. Wykorzystanie technologii semantycznych serwisów [55] pozwoliłoby na stworzenie zasobów, zbudowanych wraz z logiką regulującą ich wykorzystanie.

Wykorzystywane w systemie techniki reprezentacji stanu i podejmowania decyzji bazują na informacjach opisujących aktualny stan środowiska reprezentowanych w postaci faktów dla systemu regułowego. Takie podejście wystarczało w ewaluowanych strategiach przydziału zasobów, jednak w niektórych przypadkach pożądane byłoby rozszerzenie zbioru faktów o informacje uzyskane na podstawie przetwarzania strumieni zdarzeń. Istnieją implementacje[3] pozwalające na wykorzystanie w implementacji systemu technologii zaawansowanego przetwarzania zdarzeń CEP (ang. *Complex Event Processing*) [62] za pomocą specjalizowanego języka EQL (ang. *Event Query Language*). Technologia ta pozwala na przetwarzanie zdarzeń i identyfikację złożonych symptomów z wielu równoległych stru-

[1] Pożądane było np. ograniczenie wykorzystania zasobów przez system VGRMS, które jest proporcjonalne do ilości obsługiwanych fizycznych węzłów.

[2] Przykładem komunikacji poufnych jest transmisja obrazów dysków czy migracja (kopiowanie) zawartości pamięci VM poprzez siec rozległą WAN.

[3] Takie jak np. Esper: (`http://esper.codehaus.org/`) czy NESper upraszczające konstrukcję systemów i aplikacji przetwarzających znaczne ilości generowanych zdarzeń i komunikatów.

mieni komunikatów, które następnie mogłyby być reprezentowane jako przesłanki wnioskowania.

Inne kierunki możliwego rozwoju to uzupełnienie funkcjonalności interfejsu użytkownika poprzez dodanie możliwości automatyzacji często wykonywanych a równocześnie czasochłonnych operacji jak np. aktualizacja oprogramowania poszczególnych VM.

Autor jest przekonany, iż zaimplementowany system zarządzania zasobami stanowi solidną podstawę dla dalszych prac związanych z rozbudową istniejących możliwości i dodaniem nowych funkcjonalności, celem stworzenia bardziej uniwersalnego rozwiązania.

Dodatek A

Kod źródłowy głównych komponentów systemu

A.1. Komponenty zarządzania stanem i konfiguracją wirtualnych maszyn

A.1.1. Obsługa parawirtualizacji środowiska Xen

```
1  public interface NodeMaster {
2
3      public void migrate(String vmLabel, String dest);
4      public void create(VDIModel vdiModel, VMModel vmModel, VBDModel vbdModel);
5      public void destroy(String uUid);
6
7      public String[] getVmsLabel();
8      public ArrayList<VMWrapper> getVMs();
9      public ArrayList<VGWrapper> getVGs();
10
11     public void create(VGModel vgModel) throws MalformedObjectNameException,
12         NullPointerException, NotCompliantMBeanException,
13         InstanceAlreadyExistsException, MBeanRegistrationException;
14     public void destroyVG(String name) throws InstanceNotFoundException,
15         MBeanRegistrationException, MalformedObjectNameException,
16         NullPointerException;
17 }
```

Kod źródłowy A.1. Interfejs komponentu zarządzającego domeną kontrolną systemu Xen (`NodeMaster`)

```
1  public interface XenJMXMediator {
2
3      public void createVMMBean(String uUid) throws BadServerResponse,
4          NoConnectionOnThisThreadException, XmlRpcException,
5          MalformedObjectNameException, NullPointerException,
6          InstanceAlreadyExistsException, MBeanRegistrationException,
7          NotCompliantMBeanException;
8
9      public void deleteVMMBean(String uUid) throws MalformedObjectNameException,
10         NullPointerException, InstanceNotFoundException,
11         MBeanRegistrationException;
12
13     public String mBeanExists(String uUid) throws MalformedObjectNameException,
14         NullPointerException, AttributeNotFoundException,
15         InstanceNotFoundException, MBeanException, ReflectionException;
16
17     public boolean vmExists(String uuid);
18     public String[] getVmsUuid();
19     public String getHostName();
20     public String getServerAgentID();
21 }
```

Kod źródłowy A.2. Interfejs pośredniczący pomiędzy warstwą Xen a JMX (`XenJMXMediator`)

A.1.2. Komponenty zarządzania wirtualną maszyną

```
public interface VMM {

    public enum Type {
        ACCESSELEMENT ("AccessElement"), FIREWALL ("Firewall"),
        NODE ("Node"), ROUTER ("Router"), SERVER ("Server"),
        STORAGEELEMENT ("StorageElement"), SWITCH ("Switch"),
        WORKERNODE ("WorkerNode");

        private String name;

        private Type(String name) {
            this.name = name;
        }
    };

    public String getUuid();
    public String getLabel();

    public Type getType();
    public void setType(Type type);

    public ArrayList <String> getVIFs();
    public Double getVIFsReadUtilisation();
    public Double getVIFsWriteUtilisation();

    public Long getVCPUsNumber();
    public Map <Long, Double> getVCPUsUtilisation();
    public Map <Long, Set<String>> getVCPUsFlags();
    public Map <String, String> getVCPUsParams();

    public Long getMemoryActual();
    public Long getMemoryDynamicMax();
    public Long getMemoryDynamicMin();
    public Long getMemoryStaticMax();
    public Long getMemoryStaticMin();

    public Date getLastUpdated();
    public Date getStartTime();
    public Set <String> getState();

    public Long getCPUWeight();
    public void setCPUWeight(Long weight);

    public Long getCPUCap();
    public void setCPUCap(Long cap);

    public Long getConsolePort();
}
```

Kod źródłowy A.3. Interfejs komponentu zarządzania pojedynczą instancją wirtualnej maszyny (`VMM`)

```
public class VMWrapper implements Serializable {
    private static final long serialVersionUID = -8825486039756434321L;

    public enum State {RUNNING, DESTROYING, MIGRATING, DESTROYED}

    State state = State.RUNNING;
    protected String name;
    protected String uUid;
    protected String hostName;

    public VMWrapper(String name) {
        this.name = name;
    }
    public VMWrapper(String name, String hostName) {
        this.name = name;
        this.hostName = hostName;
    }
    public String getUUid() {
        return uUid;
    }
    public void setUUid(String uid) {
        uUid = uid;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
30      public String getHostName() {
31          return hostName;
32      }
33      public void setHostName(String hostName) {
34          this.hostName = hostName;
35      }
36      public State getState() {
37          return state;
38      }
39      public void setState(State state) {
40          this.state = state;
41      }
42      @Override
43      public String toString() {
44          return name;
45      }
46  }
```

Kod źródłowy A.4. Serializowalny obiekt reprezentujący podstawowe informacje o wirtualnej maszynie wykorzystywany w komunikacji zdalnej (`VMWrapper`)

A.1.3. Komponenty zarządzania wirtualnym Gridem

```
1   public interface VGM {
2       public String getName();
3       public String getDescription();
4       public String[] getHosts();
5       public String[] getVMs();
6
7       public void addVM(String name);
8       public boolean removeVM(String name);
9       public void addHost(String name);
10      public boolean removeHost(String name);
11
12      public String getTopology();
13      public void setTopology(String topology);
14
15      public List<String> getActiveRules();
16      public void addRule(JessRule jessRule);
17      public void deleteRule(String name);
18      public void startEngine();
19      public void stopEngine();
20      public void setSamplingPeriod(Integer milis);
21      public Integer getSamplingPeriod();
22  }
```

Kod źródłowy A.5. Interfejs komponentu zarządzającego pojedynczą instancją wirtualnego Gridu (`VGM`)

```
1   public class VGWrapper implements Serializable {
2
3       private static final long serialVersionUID = 6997154517911467641L;
4
5       String name;
6       String hostName;
7
8       public VGWrapper(String name, String hostName) {
9           super();
10          this.name = name;
11          this.hostName = hostName;
12      }
13
14      public String getHostName() {
15          return hostName;
16      }
17      public void setHostName(String hostName) {
18          this.hostName = hostName;
19      }
20      public String getName() {
21          return name;
22      }
23      public void setName(String name) {
24          this.name = name;
25      }
26  }
```

Kod źródłowy A.6. Serializowalny obiekt reprezentujący podstawowe informacje o wirtualnym Gridzie wykorzystywany w komunikacji zdalnej (`VGWrapper`)

A.2. Komponenty zarządzania polityką działania systemu

```
1  public interface PolicyRepository {
2      public void addScript(JythonScript script);
3      public void removeScript(String name);
4      public void modifyScript(JythonScript script);
5      public ArrayList<JythonScript> getScripts();
6      public String getScript(String name);
7      public void addRule(JessRule rule);
8      public boolean removeRule(String name);
9      public void modifyRule(JessRule rule);
10     public ArrayList<JessRule> getRules();
11     public JessRule getRule(String name);
12 }
```

Kod źródłowy A.7. Interfejs komponentu obsługi repozytorium polityk (`PolicyRepository`)

```
1  public class JythonScript implements Serializable {
2
3      private static final long serialVersionUID = 1569261696290632849L;
4
5      private String name;
6      private String content;
7
8      public JythonScript (String name, String content) {
9          this.name = name;
10         this.content = content;
11     }
12     public String getName() {
13         return name;
14     }
15     public void setName(String name) {
16         this.name = name;
17     }
18     public String getContent() {
19         return content;
20     }
21     public void setContent(String content) {
22         this.content = content;
23     }
24 }
```

Kod źródłowy A.8. Reprezentacja skryptów języka Jython (`JythonScript`)

```
1  public class JessRule implements Serializable {
2      private static final long serialVersionUID = 761568621392546686L;
3
4      private String name;
5      private String description;
6      private String conditions;
7      private String actions;
8
9      public String getName() {
10         return name;
11     }
12     public void setName(String name) {
13         this.name = name;
14     }
15     public String getDescription() {
16         return description;
17     }
18     public void setDescription(String description) {
19         this.description = description;
20     }
21     public String getConditions() {
22         return conditions;
23     }
24     public void setConditions(String conditions) {
25         this.conditions = conditions;
26     }
27     public String getActions() {
28         return actions;
29     }
30     public void setActions(String actions) {
31         this.actions = actions;
32     }
33 }
```

Kod źródłowy A.9. Reprezentacja reguł środowiska Jess (`JessRule`)

A.3. Monitorowanie wykorzystania zasobów

```
1  public interface NetMonitor {
2      public List <String> getInterfaceList();
3
4      public void setCaptureInterface(int newInterface);
5      public int getCaptureInterface();
6
7      public void setPromiscuous(boolean promiscuous);
8      public boolean getPromiscuous();
9
10     public void setCaptureFilter(String newCaptureFilter);
11     public String getCaptureFilter();
12
13     public void setSnapLenght(int newSnapLenght);
14     public int getSnapLenght();
15
16     public NetMonitoringThread startCapture();
17
18     public void getMap(NetMonitoringThread netMonitoringThread);
19 }
```

Kod źródłowy A.10. Interfejs komponentu używanego do monitorowania komunikacji sieciowej (`NetMonitor`)

```
1  public interface OSMonitor {
2
3      public void connect();
4      public void connect(String login, String password, String nameSpace, String address);
5
6      public Vector getInstancesDescription();
7      public int getNumber();
8
9      public String getName();
10     public BigInteger getTotalVisibleMemorySize();
11     public BigInteger getTotalSwapSpaceSize();
12     public BigInteger getTotalVirtualMemorySize();
13     public BigInteger getFreePhisicalMemory();
14     public BigInteger getFreeVirtualMemory();
15     public Date getLastBootUpTime();
16     public Date getLocalDateTime();
17     public long getMaxNumberOfProcesses();
18     public BigInteger getMaxProcessMemorySize();
19     public long getNumberOfProcesses();
20     public String getOSType();
21     public String getOSVersion();
22
23     public String getCPUStatus(Integer processorNumber);
24     public int getCurrentClockSpeed(Integer processorNumber);
25     public int getMaxClockSpeed(Integer processorNumber);
26     public String getFamily(Integer processorNumber);
27     public int getLoadPercentage(Integer processorNumber);
28     public String getRole(Integer processorNumber);
29 }
```

Kod źródłowy A.11. Interfejs komponentu używanego do monitorowania parametrów fizycznego węzła (`OSMonitor`)

Dodatek B

Reguły modułu decyzyjnego

Niniejszy dodatek przedstawia przykładowe reguły systemu decyzyjnego używane podczas testów ewaluacji implementacji systemu zarządzania zasobami. Rezultaty tych badań zostały zamieszczone w rozdziale 7.

B.1. Przydział zasobów dla wirtualnych maszyn

```
1  (defglobal ?*priority* = 256)
2
3  (deftemplate host "dynamic host information"
4    (slot name)
5    (slot load (default 0.0))
6    (slot weight (default 0))
7    (multislot vms)
8  )
9
10 (deftemplate vcpu-params "vcpus scheduler params"
11   (slot vm)
12   (slot weight (default 256))
13   (slot cap (default 0))
14 )
```

Kod źródłowy B.1. Deklaracje faktów tymczasowych i zmiennych globalnych

```
1  (defrule create-host-facts
2    (JessVMFact (name ?vm) (hostName ?h) (CPUWeight ?w) (VCPUsUtilisation ?l))
3   =>
4    (assert (host (name ?h) (vms ?vm) (weight ?w) (load ?l)))
5  )
6
7  (defrule combine-host-facts
8    ?h1 <- (host (name ?h) (load ?l1) (weight ?w1) (vms ?v1))
9    ?h2 <- (host (name ?h) (load ?l2) (weight ?w2) (vms ?v2&~?v1))
10  =>
11   (retract ?h2)
12   (modify ?h1 (load (+ ?l1 ?l2)) (weight (+ ?w1 ?w2)) (vms ?v1 ?v2))
13 )
```

Kod źródłowy B.2. Reguły języka Jess związane z obliczaniem obciążenia fizycznych hostów

```
1  (defrule set-vg-global-weight-100
2    (declare (salience +100))
3    (test (neq ?*priority* 100))
4   =>
5   (bind ?*priority* 100)
6  )
7
8  (defrule set-default-weght-and-cap
9    (JessVMFact (name ?vm))
10  =>
11   (assert (vcpu-params (vm ?vm) (weight 256) (cap 0)))
12 )
13
14 (defrule execute-set-weight
15   (declare (salience -100))
16   ?vcpu <- (vcpu-params (vm ?name) (weight ?w))
```

```
17    (JessVMFact (name ?name) (CPUWeight ?jw&:(neq ?jw w2)) (OBJECT ?o))
18   =>
19    (retract ?vcpu)
20    (call ?o setCPUWeight ?w)
21  )
```

Kod źródłowy B.3. Reguły języka Jess odpowiedzialne za ustalanie domyślnych wartości parametrów CPU

B.1.1. Przydział zasobów zgodnie z przyjętą polityką QoS

```
1   (defrule adjust-weights-equal
2     (JessVMFact (name ?n))
3     ?vcpu <- (vcpu-params (vm ?n) (weight ~?*priority*))
4    =>
5     (modify ?vcpu (weight ?*priority*))
6   )
7
8   (defrule adjust-weight-one-vm
9     (JessVMFact (name ?n) (hostName ?h))
10    (JessHostFact (hostName ?h) (VMNumer 1))
11    ?vcpu <- (vcpu-params (vm ?n) (weight ~?*priority*))
12   =>
13    (modify ?vcpu (weight ?*priority*))
14  )
15
16  (defrule adjust-weight-vm-count
17    (JessVMFact (name ?n) (hostName ?h))
18    (JessHostFact (hostName ?h) (VMNumer ~1))
19    (host (vms $?v&:(member$ ?n $?v)))
20    ?vcpu <- (vcpu-params (vm ?n))
21   =>
22    (bind ?new-weight (integer(/ ?*priority* (length$ $?v))))
23    (modify ?vcpu (weight ?new-weight))
24  )
25
26  (defrule adjust-weight-with-load
27    (JessVMFact (name ?n) (hostName ?h) (CPUWeight ?w) (VCPUsUtilisation ?u))
28    (JessHostFact (hostName ?h) (VMNumer ~1))
29    (host (vms $?v&:(member$ ?n $?v)) (weight ?hw) (load ?l))
30    ?vcpu <- (vcpu-params (vm ?n))
31   =>
32    (bind ?p (* ?*priority* (/ (/ ?u ?l) (/ ?w ?hw))))
33    (bind ?new-weight (integer(/ ?p (length$ $?v))))
34    (modify ?vcpu (weight ?new-weight))
35  )
```

Kod źródłowy B.4. Zestaw reguł określających metodę podziału zasobów podczas testu właściwości QoS

B.2. Zarządzanie i obsługa migracji wirtualnych maszyn

```
1   (deftemplate migration "migration␣parameters"
2     (slot srcHost)
3     (slot dstHost)
4     (slot vm)
5     (slot priority)
6     (slot status)
7   )
```

Kod źródłowy B.5. Deklaracje faktów tymczasowych i zmiennych globalnych dla reguł obsługi migracji

```
1   (defrule create-migration-facts
2     (JessVmFact (name ?vm) (hostName ?src))
3     (JessHostFact (hostName ?dst~?src))
4    =>
5     (assert (migration (srcHost ?src) (dstHost ?dst) (vm ?vm)))
6   )
```

```
7
8  (defrule remove-double-migrations
9    ?migration <- (migration (vm ?vm) (priority ?pio1))
10   (migration (vm ?vm) (priority ?pio2))
11   (test (> ?pio1 ?pio2))
12 =>
13   (retract ?migration)
14 )
15
16 (defrule check-vm-requirements
17   (migration (vm ?name) (dstHost ?host))
18   (JessHostFact (name ?host) (freeMemory ?freemem))
19   (JessVmFact (name ?name) (memory ?memory))
20   (test (> ?memory ?freemem))
21 =>
22   (retract ?migration)
23 )
24
25 (defrule execute-migration
26   (declare (salience -100))
27   ?migration <- (migration (vm ?name) (dstHost ?host))
28   (JessVmFact (name ?name) (hostName ~?host) (OBJECT ?o))
29   (JessHostFact (name ?host))
30 =>
31   (retract ?migration)
32   (call ?o migrate ?host)
33 )
```

Kod źródłowy B.6. Zestaw reguł dla obsługi tworzenia faktów opisujących migrację

```
1  ; Regula powodujaca zatrzymanie migracji
2  (defrule stop-all-migrations
3    ?migration <- (migration)
4  =>
5    (retract ?migration)
6  )
7
8  ; Lista gotowych do wykonania migracji
9  (defrule print-all-migrations
10   (declare (salience -50))
11   ?migration <- (migration (vm ?vm) (srcHost ?src) (dstHost ?dst))
12 =>
13   (printout t "VM: " ?vm " will migrate from: " ?src " to: " ?dst crlf)
14 )
```

Kod źródłowy B.7. Zestaw reguł diagnostycznych dla faktów opisujących migrację

B.2.1. Rozmieszczenie VM w oparciu o monitorowanie parametrów środowiskowych

```
1  (deftemplate failed-host
2    (slot name)
3    (slot reason)
4    (slot timestamp)
5  )
6
7  (defrule create-env-monitor
8    (JessHostFact (?name ?n))
9  =>
10   (bind ?envmon (new JessEnvMonitor ?n))
11   (assert ?envmon)
12 )
13
14 (defrule mark-failed-host
15   (JessEnvMonior (host ?h) (cpuTemperature ?cputemp) (cpuTempMax ?cputempmax))
16   (test (> ?cputemp ?cputempmax))
17 =>
18   (assert (failed-host (name ?h) (reason "overtemp")))
19 )
```

Kod źródłowy B.8. Reguły obsługi modułu monitorowania parametrów środowiska

```
(defrule migrate-from-failed-host
  (JessVmFact (hostName ?h) (name ?vm))
  (failed-host (name ?h))
  (JessHostFact (name ?dst&~?h))
 =>
  (assert (migration (dstHost ?dst) (srcHost ?h) (vm ?vm)))
)

(defrule cancel-migration-to-failed-host
  ?migration <- (migration (dstHost ?h))
  (failed-host (name ?h))
 =>
  (retract ?migration)
)
```

Kod źródłowy B.9. Zarządzanie migracją VM w oparciu o informacje o uszkodzonych węzłach

B.2.2. Rozmieszczenie VM w oparciu o monitorowanie komunikacji sieciowej

```
(defrule create-net-monitor
  (JessVmFact (name ?n) (hostName ?h) (id ?id))
 =>
  (bind ?net-monitor (new JessNetMonitor ?h ?n eth?id.0))
  (assert ?net-monotor)
)

(defrule migrate-net-communication
  (JessVmFact (name ?n))
  (JessHostFact (hostName ?h))
  ?moniotr <- (JessNetMonitor (host ?h) (runtime > 10))
 =>
  (bind ?topip (call ?monitor getTopDstIP))
  (assert (migrate (srcHost ?h) (dstHost ?topip) (vm ?vm)))
)

(defrule resolve-ip-address
  ?migrate <- (migrate (dstHost ?ip))
  (JessVmFact (ip ?ip) (hostName ?h))
 =>
  (modify (?migrate (dstHost ?h)))
)

(defrule cancel-cross-migrations
  ?migrate <- (migrate (srchost ?src) (dsthost ?dst))
  (migrate (srchost ?dst) (dsthost ?src))
 =>
  (retract ?migration)
)
```

Kod źródłowy B.10. Reguły obsługi migracji wykorzystujące informacje z monitorowania komunikacji VM

Bibliografia

[1] Jakub Adamczyk, Rafał Chojnacki, Marcin Jarząb, Krzysztof Zieliński. Rule Engine Based Lightweight Framework for Adaptive and Autonomic Computing. Opublikowane w *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part I*, strony 355–364. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-69383-3. doi:http://dx.doi.org/10.1007/978-3-540-69384-0˙41.

[2] AdventNet, Inc., Pleasanton, CA, USA. *SNMP Adaptor for JMX. Release 5, Software documentation*, 2003. URL `http://www.adventnet.com/products/snmpadaptor/AdventNetSNMPAdaptorForJMX.pdf`.

[3] Gagan Aggarwal, Rajeev Motwani, An Zhu. The load rebalancing problem. Opublikowane w *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, strony 258–265. ACM, New York, NY, USA, 2003. ISBN 1-58113-661-7. doi:http://doi.acm.org/10.1145/777412.777460.

[4] Arshad Ali, Ashiq Anjum, Atif Mehmood, Richard McClatchey, Ian Willers, Julian J. Bunn, Harvey B. Newman, Michael Thomas, Conrad Steenberg. A Taxonomy and Survey of Grid Resource Planning and Reservation Systems for Grid Enabled Analysis Environment. *The Computing Research Repository (CoRR)*, 27(1), styczeń 2004.

[5] Padma Apparao, Srihari Makineni, Don Newell. Characterization of network processing overheads in Xen. Opublikowane w *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, strona 2. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2873-1. doi:http://dx.doi.org/10.1109/VTDC.2006.3.

[6] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield. Xen and the Art of Virtualization. Opublikowane w *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, strony 164–177. ACM, New York, NY, USA, 2003. ISBN 1-58113-757-5. doi:http://doi.acm.org/10.1145/945445.945462.

[7] T. Bates, R. Chandra, D. Katz, Y. Rekhter. Multiprotocol Extensions for BGP-4. RFC 4760 (Draft Standard), styczeń 2007. URL `http://www.ietf.org/rfc/rfc4760.txt`.

[8] Andy Bavier, Thiemo Voigt, Mike Wawrzoniak, Larry Peterson, Per Gunningberg. SILK: Scout Paths in the Linux Kernel. raport techniczny, Department of Information Technology, Uppsala University, luty 2002. URL `http://citeseer.ist.psu.edu/bavier01silk.html`.

[9] Kazimierz Bałos. *Self-configurable Service with Elements of Adaptability for Monitoring of Infrastructure and Applications in a Grid Environment.* praca doktorska, Akademia Górniczo-Hutnicza, Wydział EAIiE, Katedra Informatyki, Kraków, Polska, lipiec 2007.

[10] Judy I. Beiriger, Hugh P. Bivens, Steven L. Humphreys, Wilbur R. Johnson, Ronald E. Rhea. Constructing the ASCI Computational Grid. *hpdc*, 00:193, 2000. ISSN 1082-8907. doi:http://doi.ieeecomputersociety.org/10.1109/HPDC.2000.868650.

[11] John Bent, Venkateshwaran Venkataramani, Nick LeRoy, Alain Roy, Joseph Stanley, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Miron Livny. *Grid Resource Management*. Kluwer Academic Publishers, 2003.

[12] Veeravalli Bharadwaj, Debasish Ghose, Thomas G. Robertazzi. Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems. *Cluster Computing*, 6(1):7–17, 2003. ISSN 1386-7857. doi:http://dx.doi.org/10.1023/A:1020958815308.

[13] Joseph Bih. Service Oriented Architecture (SOA) a new paradigm to implement dynamic e-business solutions. *Ubiquity*, 7(30):1–1, 2006. doi:http://doi.acm.org/10.1145/1159402.1159403.

[14] Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, Eduardo Gómez-Sanchez. Grid Characteristics and Uses: a Grid Definition. Opublikowane w *Proceedings of the First European Across Grids Conference*, strony 291–298. luty 2003.

[15] S. Brunett, Karl Czajkowski, Steven Fitzgerald, Ian T. Foster, Andrew E. Johnson, Carl Kesselman, Jason Leigh, Steven Tuecke. Application Experiences with the Globus Toolkit. Opublikowane w *HPDC*, strony 81–. 1998.

[16] Rajkumar Buyya, Steve J. Chapin, David C. DiNucci. Architectural Models for Resource Management in the Grid. Opublikowane w *GRID*, strony 18–35. 2000. URL `http://citeseer.ist.psu.edu/402856.html`.

[17] S. Carl-Mitchell, J. S. Quarterman. Using ARP to implement transparent subnet gateways. RFC 1027, październik 1987. URL `http://www.ietf.org/rfc/rfc1027.txt`.

[18] David Chisnall. *The Definitive Guide to the Xen Hypervisor (Prentice Hall Open Source Software Development Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007. ISBN 013234971X.

[19] Cisco Systems. *Cisco Dynamic ARP Inpection*. URL `http://www.cisco.com/web/psa/technologies`.

[20] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, Andrew Warfield. Live Migration of Virtual Machines. Opublikowane w *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, strony 273–286. USENIX Association, Berkeley, CA, USA, maj 2005.

[21] Projekt CrossGrid. CrossGrid. Strona główna projektu, grudzień 2007. URL `http://www.crossgrid.org`. IST-2001-32243.

[22] Project DataGrid. Datagrid. Strona główna projektu, listopad 2007. URL `http://www.eu-datagrid.org`. IST-2000-25182.

[23] Todd Deshane, Demetrios Dimatos, Gary Hamilton, Madhujith Hapuarachchi, Wenjin Hu, Michael McCabe, Jeanna Neefe Matthews. Performance Isolation of a Misbehaving Virtual Machine with Xen, VMware and Solaris Containers. Opublikowane w *Proceedings of USENIX 2006 Conference*. 2006.

[24] Aseem Deshpande. The Role of Linux in Grid Computing. *Linux Journal*, styczeń 2004.

[25] Jeff Dike. User-mode Linux. Opublikowane w *Proceedings of the 5th annual conference on Linux Showcase & Conference*, strony 2–2. USENIX Association, Berkeley, CA, USA, 2001.

[26] Jeff Dike. *User Mode Linux(R) (Bruce Perens Open Source)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006. ISBN 0131865056.

[27] Fred Douglis, John K. Oustrhout. Transparent Proces Migration: Design alternatives and the Spirite implementation. *Software — Practice and Experience*, 21(8):757–785, 1991.

[28] Alain Drotz, Ralf Gruber, Vincent Keller, Michela Thiémard, Ali Tolou, Trach-Minh Tran, Kevin Cristiano, Pierre Kuonen, Philipp Wieder, Oliver Wäldrich, Wolfgang Ziegler, Pierre Manneback, Uwe Schwiegelshohn, Ramin Yahyapour, Peter Kunszt, Sergio Maffioletti, Marie-Christine Sawley, Christoph Witzig. Application-oriented scheduling for HPC Grids. raport techniczny TR-0070, Institute on Resource Management and Scheduling, CoreGRID — Network of Excellence, luty 2007. URL `http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0070.pdf`.

[29] Projekt EGEE. Enabling Grids for E-sciencE (EGEE). Strona główna projektu, marzec 2006. URL `http://public.eu-egee.org`. IST-2003-508833.

[30] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Techne Group, Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 2004. ISBN 0-13-142898-5.

[31] Michael W. Evans, John J. Marciniak. *Software quality assurance & management*. Wiley--Interscience, New York, NY, USA, 1987. ISBN 0-471-80930-6.

[32] Charles L. Forgy. Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[33] Organizacja Grid Forum. Open grid forum. Strona główna projektu, styczeń 2008. URL `http://www.gridforum.org`.

[34] The Global Grid Forum. Grid Scheduling Architecture Research Group (GSA-RG). Strona domowa projetku, styczeń 2008. URL `https://forge.gridforum.org/projects/gsa-rg`.

[35] Ian Foster. What is the Grid? — a three point checklist. *GRIDtoday*, 1(6), lipiec 2002. URL `http://www.gridtoday.com/02/0722/100136.html`.

[36] Ian Foster, Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, listopad 1998. ISBN 1558604758.

[37] Apache Software Foundation. Apache XML-RPC — Java implementation of XML-RPC protocol over HTTP. Strona domowa projektu, maj 2007. URL `http://ws.apache.org/xmlrpc`.

[38] Martin Fowler. Inversion of Control Containers and the Dependancy Injection Pattern. raport techniczny, ThoughtWorks, styczeń 2004. URL `http://martinfowler.com/articles/injection.html`.

[39] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, M. Wiliamson. Save Hardware Access with the Xen Virtual Machine Monitor. Opublikowane w *Proceedings of 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*. październik 2004.

[40] Arijit Ganguly, A. Agrawal, P. O. Boykin, Renato J. Figueiredo. IP over P2P: Enabling Self-configuring Virtual IP Networks for Grid Computing. Opublikowane w *Proceedings of 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. Rodos, Grecja, kwiecień 2006.

[41] Arijit Ganguly, A. Agrawal, P. O. Boykin, Renato J. Figueiredo. WOW: Self-Organizing Wide Area Overlay Networks of Virtual Workstations. Opublikowane w *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC)*. czerwiec 2006.

[42] Arijit Ganguly, D. Wolinsky, P. O. Boykin, Renato J. Figueiredo. Decentralized Dynamic Host Configuration in Wide-Area Overlays of Virtual Workstations. Opublikowane w *Proceedings of the Workshop on Large-Scale and Volatile Desktop Grids (PCGrid)*. marzec 2007.

[43] NGG Expert Group. Future for European Grids: GRIDs and Service Oriented Knowledge Utilities — Vision and Research Directions 2010 and Beyond. raport techniczny, Next Generation Grid, styczeń 2006. URL `http://www.semanticgrid.org/NGG3/`.

[44] E. Guttman. Vendor Extensions for Service Location Protocol, Version 2. RFC 3224, styczeń 2002. URL `http://www.ietf.org/rfc/rfc3224.txt`.

[45] Jacob G. Hansen, Asger K. Henriksen. *Nomadic operating systems*. praca magisterska, Dept. of Computer Science, University of Copenhagen, Denmark, 2002.

[46] Mor Harchol-Balter, Allen B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Trans. Comput. Syst.*, 15(3):253–285, 1997. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/263326.263344.

[47] Qumranet Inc. KVM: Kernel-based Virtualization Driver. White Paper, październik 2006. URL `http://www.qumranet.com/wp/kvm_wp.pdf`.

[48] Xuxian Jiang, Dongyan Xu. VIOLIN: Virtual Internetworking on Overlay Infrastructure. Opublikowane w *Parallel and Distributed Processing and Applications*, wolumen 3358, strony 937–946. Springer, LNCS, grudzień 2004. ISSN 0302-9743.

[49] William E. Johnston, Dennis Gannon, Bill Nitzberg. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. Opublikowane w *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, strona 34. IEEE Computer Society, Washington, DC, USA, 1999. ISBN 0-7695-0287-3.

[50] Gregor Kiczales, Mira Mezini. Aspect-oriented programming and modular reasoning. Opublikowane w *ICSE '05: Proceedings of the 27th international conference on Software engineering*, strony 49–58. ACM, New York, NY, USA, 2005. ISBN 1-59593-963-2. doi: http://doi.acm.org/10.1145/1062455.1062482.

[51] Jacek Kitowski. *Współczesne systemy komputerowe*. CCNS, Sp. z o.o., Kraków, 2000. ISBN 83-909958-3-2.

[52] M. Kożuch, M. Satyanarayanan. Internet Suspend/resume. Opublikowane w *Proceedings of the IEEE Workshop on Mobile Computing Systems and Aplications*. 2002.

[53] Klaus Krauter, Rajkumar Buyya, Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software — Practice and Experience*, 32(2):135–164, luty 2002. URL `http://citeseer.ist.psu.edu/krauter01taxonomy.html`.

[54] Ivan V. Krsul, A. Ganguly, J. Zhang, José A. B. Fortes, Renato J. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. Opublikowane w *Proceedings of Supercomputing Conference*. listopad 2004.

[55] Dominik Kuropka, Mathias Weske. Implementing a Semantic Service Provision Platform — Concepts and Experiences. *Special Issue on Service Oriented Architectures and Web Services of Journal Wirtschaftsinformatik*, (1/2008):16–24, 2008.

[56] Sandia National Laboratories. Jess, the Rule Engine for the Java™ Platform. Online. URL `http://herzberg.ca.sandia.gov`.

[57] John R. Lange, Peter A. Dinda. Transparent network services via a virtual traffic layer for virtual machines. Opublikowane w *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, strony 23–32. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-673-8. doi:http://doi.acm.org/10.1145/1272366.1272370.

[58] Aleksander Laurentowski, Krzysztof Zieliński. JMX — Środowisko konstrukcji nowoczesnych systemów monitorowania i zarządzania. *Telenet Forum*, strony 40–43, kwiecień 2002.

[59] Huang Li-can, Wu Zhao-hui, Pan Yun-he. Virtual and Dynamic Hierarchical Architecture: an overlay network topology for discovering grid services with high performance. Opublikowane w *Journal of Zhejiang University SCIENCE*, wolumen 5 z *5*, strony 539–549. Hangzhou, Chiny, czerwiec 2004. ISSN 1009-3095.

[60] User Mode Linux. The User-mode Linux Kernel Home Page. Online. URL `http://user-mode-linux.sourceforge.net`.

[61] Miron Livny, Rajesh Raman. High-throughput resource management. Opublikowane w Ian Foster, Carl Kesselman, redaktorzy, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[62] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[63] P. Lyster, L. Bergman, Stanfill Li, P., Crippe D., Blom B., C. R., Pardo, D. Okaya. CASA gigabit supercomputing network: CALCRUST three-dimensional real-time multi-dataset rendering. Opublikowane w *Proceedings of Supercomputing '92*. Minneapolis, MN, USA, listopad 1992.

[64] Vania Marangozova, Daniel Hagimont. Availability through Adaptation: a Distributed Application Experiment and Evaluation. *European Research Seminar on Advances in Distributed Systems*, maj 2001. SIRAC Project.

[65] Tsugawa Maurício, José A. B. Fortes. A Virtual Network (ViNe) Architecture for Grid Computing. Opublikowane w *Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS-2006)*, strona 10. kwiecień 2006.

[66] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, Betty H. C. Cheng. Composing Adaptive Software. *Computer*, 37(7):56–64, 2004. ISSN 0018-9162. doi:http://dx.doi.org/10.1109/MC.2004.48.

[67] Dejan S. Milojičić, Fred Douglis, Yves Paindaveine, Richard Wheeler, Songnian Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, 2000. ISSN 0360-0300. doi:http://doi.acm.org/10.1145/367701.367728.

[68] Ingo Molnar. Linux: The Completely Fair Scheduler. raport techniczny, Red Hat Inc., kwiecień 2007. URL `http://kerneltrap.org/node/8059`.

[69] J. Moy. Ospf version 2. RFC 2328 (Standard), kwiecień 1998. URL `http://www.ietf.org/rfc/rfc2328.txt`.

[70] Gail C. Murphy, Robert J. Walker, Elisa L. A. Baniassad, Martin P. Robillard, Albert Lai, Mik A. Kersten. Does aspect-oriented programming work? *Commun. ACM*, 44(10):75–77, 2001. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/383845.383862.

[71] Jarek Nabrzyski, Jennifer M. Schopf, , Jan Weglarz, redaktorzy. *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishing, MA, USA, 2003.

[72] Shuichi Oikawa, R. Rajkumar. Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior. Opublikowane w *IEEE Real Time Technology and Applications Symposium*, strony 111–120. czerwiec 1999. URL `http://citeseer.ist.psu.edu/oikawa99portable.html`.

[73] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, B. B. Welch. The Sprite network operating system. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 21(2), 1988. ACM CR 8905-0314.

[74] Paul Patrick. Impact of SOA on enterprise information architectures. Opublikowane w *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, strony 844–848. ACM, New York, NY, USA, 2005. ISBN 1-59593-060-4. doi:http://doi.acm.org/10.1145/1066157.1066263.

[75] Michael L. Powell, Barton P. Miller. Process migration in DEMOS/MP. Opublikowane w *Proceedings of the ninth ACM Symposium on Operating System Principles*, strony 110–119. ACM Press, 1983.

[76] *QEMU Emulator User Documentation*, październik 2007. URL `http://fabrice.bellard.free.fr/qemu/qemu-tech.html`.

[77] Y. Rekhter, T. Li, S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), styczeń 2006. URL `http://www.ietf.org/rfc/rfc4271.txt`.

[78] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), luty 1996. URL `http://www.ietf.org/rfc/rfc1918.txt`.

[79] Y. Rekhter, E. Rosen. Carrying Label Information in BGP-4. RFC 3107 (Proposed Standard), maj 2001. URL `http://www.ietf.org/rfc/rfc3107.txt`.

[80] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy. STUN — Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), marzec 2003. URL `http://www.ietf.org/rfc/rfc3489.txt`.

[81] Daniel M. Russell, Mark Weiser. The Future of Integrated Design of Ubiquitous Computing in Combined Real & Virtual Worlds. Opublikowane w *CHI '98: CHI 98 conference summary on Human factors in computing systems*, strony 275–276. ACM, New York, NY, USA, 1998. ISBN 1-58113-028-7. doi:http://doi.acm.org/10.1145/286498.286756.

[82] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, Mendel Rosenblum. Optimizing the migration of virtual computers. *SIGOPS Oper. Syst. Rev.*, 36(SI):377–390, grudzień 2002. ISSN 0163-5980. doi:http://doi.acm.org/10.1145/844128.844163.

[83] Uwe Schwiegelshohn, Philipp Wieder, Ramin Yahyapour. Resource Management for Future Generation Grids. Opublikowane w Vladimir Getov, Domenico Laforenza, Alexander Reinefeld, redaktorzy, *Proceedings of the Workshop on Future Generation Grids*, strony 99–112. Springer, Dagstuhl, Germany, listopad 2004. ISBN 0-387-27935-0. CoreGRID Technical Report TR-0005.

[84] Uwe Schwiegelshohn, Ramin Yahyapour, Philipp Wieder. Resource Management for Future Generation Grids. raport techniczny TR-0005, Institute on Resource Management and Scheduling, CoreGRID — Network of Excellence, maj 2005. URL `http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0005.pdf`.

[85] Jan Seidel, Oliver Wäldrich, Wolfgang Ziegler, Philipp Wieder, Ramin Yahyapour. Using SLA for resource management and scheduling — a survey. raport techniczny TR-0096, Institute on Resource Management and Scheduling, CoreGRID — Network of Excellence, sierpień 2007. URL `http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0096.pdf`.

[86] D. Senie. Network Address Translator (NAT)-Friendly Application Design Guidelines. RFC 3235 (Informational), styczeń 2002. URL `http://www.ietf.org/rfc/rfc3235.txt`.

[87] Roger Smith. Grid Computing: A Brief Technology Analysis. *CTO Network Library*, lipiec 2005.

[88] Stephen Soltesz, Marc E. Fiuczynski, Larry Peterson, Michael McCabe, Jeanna Matthews. Virtual Doppelganger: On the Performance, Isolation, and Scalability of Para- and Paene-Virtualized Systems. *Proceeding of Eurosys*, 2006.

[89] Rick Stevens, Paul Woodward, Tom DeFanti, Charlie Catlett. From the I-WAY to the National Technology Grid. *Commun. ACM*, 40(11):50–60, 1997. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/265684.265692.

[90] Sun Microsystems, Santa Clara, CA, USA. *JSR003: Java Management Extension (JMX) Maintenance Release 2*, 2004. URL `http://jcp.org/aboutJava/communityprocess/mrel/jsr003/index2.html`.

[91] Sun Microsystems, Santa Clara, CA, USA. *Java Management Extensions (JMX) Specification, Version 1.4, JSR 160*, 2006. URL `http://jcp.org/en/jsr/detail?id=160`. Jmx-1_4-mrel3-spec.pdf.

[92] Sun Microsystems, Santa Clara, CA, USA. *Java Management Extensions (JMX) Specification, Version 2.0, JSR 255*, luty 2008. URL `http://jcp.org/en/jsr/detail?id=255`.

[93] Vijay Sundaram, Abhishek Chandra, Pawan Goyal, Prashant Shenoy, Jasleen Sahni, Harrick Vin. Application performance in the QLinux multimedia operating system. Opublikowane w *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, strony 127–136. ACM, New York, NY, USA, 2000. ISBN 1-58113-198-4. doi:http://doi.acm.org/10.1145/354384.354448.

[94] A. Sundararaj, P. Dinda. Towards virtual networks for virtual machine grid computing. Opublikowane w *Proceedings of the 3rd USENIX Virtual Machine Research and Technology Symposium*. San Jose, CA, USA, maj 2004.

[95] Ananth I. Sundararaj, Ashish Gupta, Peter A. Dinda. Dynamic topology adaptation of virtual networks of virtual machines. Opublikowane w *LCR '04: Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, strony 1–8. ACM, New York, NY, USA, październik 2004. doi:http://doi.acm.org/10.1145/1066650.1066665.

[96] D. L. Tennenhouse. Layered multiplexing considered harmful. Opublikowane w *Proceedings of the 1-st International Workshop on High-Speed Networks*, strony 143–148. listopad 1989.

[97] Nicola Tonellotto, Philipp Wieder, Ramin Yahyapour. A Proposal for a Generic Grid Scheduling Architecture. Opublikowane w Sergei Gorlatch, Marco Danelutto, redaktorzy, *Proceedings of the Integrated Research in Grid Computing Workshop*, strony 337–346. University di Pisa, listopad 2005. CoreGRID Technical Report TR-0015.

[98] Jim Waldo. *The Jini Specifications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000. ISBN 0201726173.

[99] Andrew Whitaker, Richard S. Cox, Marianne Shaw, Steven D. Gribble. Constructing services with interposable virtual hardware. Opublikowane w *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04)*. 2004.

[100] Dave Winer. XML-RPC Specification. raport techniczny, UserLand Software, Inc., czerwiec 1999. URL `http://www.xmlrpc.com/spec`.

[101] The Xen API Project. Strona domowa projektu, 2007. URL `http://wiki.xensource.com/xenwiki/XenApi`. Informacje (wraz z przykładową implementacją dla języka Java) na temat projektu Xen API.

[102] Charles Young. Microsoft's Rule Engine Scalability Results — A comparison with Jess and Drools. raport techniczny, Solidsoft, wrzesień 2005. URL `http://geekswithblogs.net/cyoung/articles/54022.aspx`.

[103] Lamia Youseff, Rich Wolski, Brent Gorda, Chandra Krintz. Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems. Opublikowane w *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, strona 1. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2873-1. doi:http://dx.doi.org/10.1109/VTDC.2006.4.

[104] E. Zayas. Attacking the process migration bottleneck. Opublikowane w *Proceeding of the eleventh ACM Symposium on Operating systems principles*, strony 13–24. ACM Press, 1987.

[105] K. Zeilenga. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. RFC 4510, czerwiec 2006. URL `http://www.ietf.org/rfc/rfc4510.txt`.

[106] Honghai Zhang, Kate Keahey, William Allcock. Providing Data Transfer with QoS as Agreement-Based Service. Opublikowane w *Proceedings of the IEEE International Conference on Services Computing*, strony 344–353. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2225-4.

[107] Krzysztof Zieliński. Adaptowalne i reflektywne warstwy pośredniczące jako podstawa autonomicznych systemów komputerowych. *Czasopismo Pomiary, Automatyka i Kontrola*, marzec 2007.

[108] Krzysztof Zieliński, Sławomir Zieliński. A concept of Monitoring Grid Infrastructure with Java and Jiro Technologies. Opublikowane w *Proceedings of 1st Cracow Grid Workshop*. Kraków, Polska, listopad 2001.

Skorowidz

T

U

V

W

X

Z

Definicje

Efektywność zarządzania Sposób realizacji zarządzania zasobami jest efektywny jeśli równocześnie z udostępnieniem szerokiego zestawu możliwości zarządzania umożliwiających osiągnięcie określonego celu minimalizowany jest niekorzystny wpływ na środowisko i aplikacje elementów dodanych w celu realizacji tych możliwości. **8**

Grid Rozproszona geograficznie, sprzętowa i programowa infrastruktura przetwarzania dużej skali, zbudowana z heterogenicznych zasobów (w tym zasobów komunikacyjnych), udostępnionych i współdzielonych przez wiele organizacji, koordynowanych w celu osiągnięcia przezroczystego wsparcia dla przetwarzania szerokiego zakresu aplikacji [14]. **4**

Grid System Grid to dynamiczne środowisko, bazujące na współdzielonych zasobach i wykorzystujące na masową skalę oraz w sposób zdecentralizowany rozproszone komponenty. 1–3, **3**, 4–7, 9–12, 15, 23, 24, 29, 45, 47, 49, 51, 52, 54, 55, 59, 62–65, 83, 86, 91, 92, 115, 135–138, 143, 148, 175, 176

Parawirtualizacja Parawirtualizacja lub wirtualizacja lekka (ang. *LightWeight Virtualization*) — technika wirtualizacji, która zapewniając pewną iluzję rzeczywistości, wymaga aby wykorzystujące ją elementy zostały odpowiednio dostosowane i były świadome wykorzystywania mechanizmów wirtualizacji. **6**, 9, 12, 17, 19

Wirtualna Organizacja Wirtualna Organizacja jest to dynamiczna kolekcja rozproszonych zasobów wraz z zestawem reguł określających w precyzyjny sposób ich współdzielenie. Zasoby te są wykorzystywane przez zmieniającą się grupę użytkowników, należących do jednej lub wielu fizycznych organizacji. 4, 5, **5**, 9–11, 50, 85, 87, 129, 162

Wirtualny Grid Wirtualny Grid jest rozumiany jako forma określenia wymogów czy opisu zasobów, dostarczona przez aplikację w celu stworzenia środowiska umożliwiającego jej poprawne i efektywne działanie. 1, 2, 6, **6**, 7, 10, 12, 49, 51, 65, 67, 69, 74–79, 81, 82, 85–90, 94, 98–100, 103–105, 107, 108, 117, 120, 121, 124, 127, 129, 132, 136, 137, 140, 142, 144–146, 159–161, 164, 167, 168, 175, 176

Xen *High performance resource-managed virtual machine monitor*. Projekt powstał na uniwersytecie w Cambridge i jest obecnie najbardziej dynamicznie rozwijaną platformą VM stworzoną dla systemu Linux. Wirtualizacja w tym podejściu została osiągnięta poprzez stworzenie wyidealizowanej abstrakcyjnej architektury, architektury

na która systemy Linux, BSD czy Windows mogą zostać przeniesione. 17, 21–23, 27, 30, 35, 36, 38, 42–45, 96, 113–115, 121, 132, 137, 138, 144, 152, 155, 157, 158

Zarządzanie Zarządzanie zasobami jest to proces, którego rezultatem jest zmiana dostępności bądź stanu zarządzanego zasobu. Wpływ na zasób może być bezpośredni lub osiągnięty pośrednio za pomocą zmiany właściwości otoczenia obiektu lub warunków jego pracy. **8**

Zasób Zasoby są to elementy systemu informatycznego, bez dostępności których na określonym poziomie niemożliwe byłoby wykonywanie zadań przetwarzania zleconych przez użytkowników tego systemu. **7**

Wykaz skrótów

ABI *Application Binary Interface*. 43

ADSL *Asymmetric Digital Subscriber Line*. 21

AMD *Advanced Micro Devices*. 39

AMD-V *AMD-Virtualization*. 39, 40

API *Application Programming Interface*. 1, 6, 17, 111, 115, 118, 119

ARP *Address Resolution Protocol*. 23

AS *Autonomic System*. 28

BGP-4 *Border Gateway Protocol 4*. 28

BSD *Berkeley Software Distribution*. 34, 38, 42

CAM *Content-Addressable Memory*. 43

CEP *Complex Event Processing*. 177

CG *EU CrossGrid Project* IST-2001-32243. 62

CIM *Common Information Model*. 113

CoW *Copy-on-Write*. 23, 38, 41, 138

CPU *Central Processing Unit*. 19, 24, 41, 43, 46, 49, 51, 55, 89, 95, 124, 139, 142–144, 153

CRUD *Create, Read, Update, Delete*. 100

DHCP *Dynamic Host Configuration Protocol*. 24, 52

DI *Dependancy Injection*. 130

DoS *Denial-of-Service*. 60

DPD *Dissipative Particle Dynamics*. 139

EAL5 *Evaluation Assurance Level 5*. 16

EDG *EU DataGrid Project* IST-2000-25182. 62

EGEE *Enabling Grids for E-sciencE Project* IST-2003-508833. 62, 138

ELF *Executable and Linking Format*. 35

EQL *Event Query Language*. 177

FLOSS *Free Libre/Open Source Software*. 62

GGF *Global Grid Forum*. 55

GPE *Global Policy Engine*. 129

GPL *GNU General Public License*. 34, 36, 38

GRM *Global Resource Manager*. 89

GSA-RG *Grid Scheduling Architecture Research Group*. 55

GUI *Graphical User Interface*. 119, 129, 132

HP *Hewlett-Packard*. 16

HTTP *HyperText Transfer Protocol*. 111

I/O *Input/Output*. 19, 21, 37, 39–45, 51, 153

IBM *International Business Machines*. 15, 16

IMQ *Linux Intermediate Queueing Device*. 31

IoC *Inversion of Control*. 130

IP *Internet Protocol*. 25, 28, 32, 52, 72, 158

IPC *Inter-process Communication*. 18, 26, 37

IPSec *Internet Protocol Security*. 27, 28, 52

IPv4 *Internet Protocol version 4*. 24, 28

IPv6 *Internet Protocol version 6*. 28, 52

iSCSI *Internet SCSI*. 92

ISL *Inter-Switch Link*. 28

ISP *Internet Service Provider*. 18

IT *Information Technology*. 68

IVT *Intel Virtualization Technology*. 39

JDK *Java Development Kit*. 138

JMX *Java Management Extensions*. 111–115, 117, 121, 122, 130–132

JSR *Java Specification Request*. 118, 131

JVM *Java Virtual Machine*. 21, 111

KVM *Kernel Virtual Machine*. 17, 39–41, 44

LAN *Local Area Network*. 1, 22–24, 26, 28, 31, 32, 104, 112, 122, 123, 127, 153

LDAP *Lightweight Directory Access Protocol*. 85, 112

LGPL *GNU Lesser General Public License*. 34

LPAR *Logical Partition*. 16

LRM *Local Resource Manager*. 87–89

LVM *Logical Volume Manager*. 138

MAC *Media Access Control*. 32

MBGP *Multiprotocol BGP*. 27, 28

MDA *Model Driven Architecture*. 61, 111

MIT *Massachusetts Institute of Technology*. 16

MPI *Message Passing Interface*. 139

MPLS *Multiprotocol Label Switching*. 27, 28

MPP *Massively Parallel Processing*. 16

NAS *Network Attached Storage*. 23, 153

NAT *Network Address Translation*. 24, 25, 29, 31, 52, 72

NBD *Network Block Device*. 153

NUMA *Non-Uniform Memory Access*. 16, 40

OS *Operating System*. 11, 20, 21, 23, 34, 35

OSI *Open Source Initiative*. 34

OSI/ISO *Open Systems Interconnection/International Organization for Standardization*. 25, 27, 31, 156

P2P *Peer To Peer*. 27–29

PC *Personal Computer*. 91, 136, 161

PIM *Platform Independant Model*. 83

POJO *Plain Ordinary/Old Java Object*. 132

QoS *Quality of Service*. 2, 4, 19, 24, 25, 29, 31, 35, 47, 50, 52, 53, 55, 58, 72, 89, 136, 144, 145, 168, 169, 171

RAM *Random Access Memory*. 37, 95

RFB *Remote Frame Buffer*. 28

RISC *Reduced Instruction Set Computer*. 43

RMI *Remote Method Invocation*. 111, 112, 131

RMS *Resource Management System*. 7, 50, 55, 56, 59, 65, 66, 86

RPC *Remote Procedure Call*. 115

SAS *Serial Attached SCSI*. 92

SDC *Segment-Descriptor Cache*. 43

SGI *Silicon Graphics, Inc.*. 16

SKAS *Separate Kernel Address Space*. 36

SLA *Service Level Agreement*. 52, 55, 144

SLP *Service Location Protocol*. 112

SMP *Symmetric Multi-Processing*. 39, 145

SNMP *Simple Network Management Protocol*. 112

SOA *Service Oriented Architecture*. 61, 176

SOAP *Simple Object Access Protocol*. 112

SSH *Secure SHell*. 27, 33, 41, 126, 127

SSL *Secure Sockets Layer*. 52

SVM *Secure Virtual Machine*. 39, 40

TCP *Transmission Control Protocol*. 27, 28, 53, 158

TCP/IP *Transmission Control Protocol/Internet Protocol*. 18, 24

TLB *Translation Lookaside Buffer*. 43

ToS *Type of Service*. 28

UDP *User Datagram Protocol*. 28

UI *User Interface*. 132

UML *User Mode Linux*. 35, 36

UUID *Universally Unique Identifier*. 115

VDE *Virtual Distributed Ethernet*. 32, 33, 122

VE *Virtual Environment*. 36

VG *Virtual Grid*. 69, 71, 75, 76, 78–82, 87–89, 100, 104, 117, 122, 123, 127, 140, 143, 145, 161

VGRMS *Virtual Grid Resources Management System*. 81, 82, 91, 93–98, 101, 104, 105, 107–109, 111–115, 121, 124, 129–133, 136, 140, 144, 148, 152, 160, 162, 163, 165–167, 169, 172, 173, 175–177

VLAN *Virtual LAN*. 28

VM *Virtual Machine*. 1, 6, 17–24, 26–29, 31–33, 35, 36, 41–45, 52, 53, 75, 76, 78, 79, 88, 89, 93, 96–98, 101, 103–105, 112–115, 123, 124, 126, 127, 135, 137, 138, 140, 141, 143–145, 152, 153, 155–158, 162, 163, 166, 167, 177, 178

VMM *Virtual Machine Monitor*. 41–45

VN *Virtual Network*. 75, 76, 88

VNC *Virtual Network Computing*. 28

VPN *Virtual Private Network*. 31, 32, 52

VPS *Virtual Private Server*. 36, 38

VRF *Virtual Routing and Forwarding*. 27, 28

VRMS *Virtual Resources Management System*. 66

VWN *Virtual Worker Node*. 1, 52

WAN *Wide Area Network*. 1, 3, 27, 30, 31, 108, 136, 163, 166, 177

WBEM *Web-Based Enterprise Management*. 112

WWS *Writable Working Set*. 22, 23

X *X Window System*. 28

Xen-API *Xen Application Programming Interface*. 115, 132

XML *eXtensible Mark-up Language*. 115, 117, 122, 132

XML-RPC *XML — Remote Procedure Call*. 115