POLITECHNIKA GDAŃSKA
Wydział Elektroniki, Telekomunikacji i Informatyki

Jerzy Proficz

Zarządzanie zasobami obliczeniowymi w klastrowym środowisku przetwarzania strumieni multimedialnych

Rozprawa doktorska

Promotor:
prof. dr hab. inż. Henryk Krawczyk, prof. zw.
Wydział Elektroniki, Telekomunikacji i Infrormatyki
Politechnika Gdańska

Gdańsk, 2012

INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

MAYDAY EURO 2012

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ ROZWOJU REGIONALNEGO

Dziękuję prof. Henrykowi Krawczykowi, mojemu promotorowi, za wiarę, cierpliwość i wysiłek jaki włożył w powstanie tej rozprawy.

Dziękuję zespołowi projektu Mayday Euro 2012, a szczególnie inż. Bartłomiejowi Dacy, inż. Adamowi Glinianowiczowi i inż. Piotrowi Sawickiemu za pomoc podczas pracy nad tą rozprawą.

Streszczenie

Zaprezentowano zintegrowany model zarządzania multimedialnym systemem rozproszonym, przetwarzającym strumienie danych, jak również opisano koncepcję jego implementacji w platformie KASDADA. Zdefiniowano charakterystyki wydajnościowe i wiarygodnościowe oraz określono klasę przetwarzanych algorytmów analizy strumieni. Sformułowano tezy rozprawy: (1) Przy akceptowanej wiarygodności przetwarzania strumieni, obciążenie węzła dla analizowanej klasy algorytmów i dla danego typu przetwarzanego strumienia wzrasta nieliniowo wraz z liczbą przetwarzanych strumieni, przy czym wzrost ten można oszacować tzw. funkcją korekty podającą przyrost obciążenia w stosunku do jego wzrostu liniowego. (2) Dla zadanej klasy algorytmów analizy strumieni oraz dla zaproponowanego algorytmu alokacji zadań na węzły klastra (wykorzystującego funkcję korekty), skalowalność klastra utrzymuje się na stałym poziomie przy wzroście liczby jednocześnie przetwarzanych zadań i strumieni, pod warunkiem proporcjonalnego do niego wzrostu liczby węzłów obliczeniowych zaangażowanych w to przetwarzanie. Zaprezentowano wyniki badań eksperymentalnych potwierdzających te tezy. Dla pojedynczego węzła obliczeniowego dokonano oceny możliwości przetwarzania strumieni multimedialnych, wyznaczono i zweryfikowano funkcję korekty. Dla całego klastra przedstawiono heurystyczny algorytm alokacji (zmodyfikowany BFD) uwzględniający powyższą funkcję i wykazano przewagę zaproponowanego algorytmu nad innymi dotychczas stosowanymi algorytmami.

Abstract

The integrated model of the distributed multimedia system processing data streams and its implementation in the KASKADA platform were presented. The desired efficiency and dependability characteristics as well as the related data stream algorithm class were defined. The following dissertation claims were formulated: (1) For the acceptable dependability level of data stream processing, the computation node utilization for the analyzed algorithm class and for a specific stream type increases non-linearly to the number of the processed streams, we assume the deviation from the linear increase can be estimated by the so called correction function. (2) For the concerned algorithm class and for the proposed task-to-node allocation algorithm (utilizing the correction function), the computation cluster scalability remains at the same level with the increase of processed streams and tasks, under the condition of proportional increase of the involved computation cluster nodes. The results of the experimental research, proving the claims were presented. For the single node, its multimedia processing capabilities were evaluated, as well as the correction function was determined and verified. For the whole cluster, a new heuristic allocation algorithm (modified BFD) concerning this function was described and its advantage over usually used heuristics was shown.

Spis treści

Wykaz oznaczeń....8
Wykaz skrótów....10
1 Wstęp....13
2 Problematyka zarządzania systemami komputerowymi....15
2.1 Rozwój systemów komputerowych....15
2.2 Zarządzanie danymi....19
2.3 Zarządzanie zasobami....22
2.4 Zarządzanie komunikacją....23
2.5 Zarządzanie obliczeniami....25
2.6 Systemy czasu rzeczywistego....28
2.7 Przetwarzanie strumieni multimedialnych....30
2.8 Pojęcie skalowalności....33
2.9 Tezy rozprawy....35
3 Zintegrowane zarządzanie platformą KASKADA....38
3.1 Koncepcja zintegrowanego zarządzania....38
3.2 Porównanie platformy KASKADA z istniejącymi systemami rozproszonymi....43
3.3 Zarządzanie na platformie KASKADA....44
3.4 Przykład wykorzystania platfromy KASKADA....48
3.5 Konfiguracja środowiska przetwarzania....52
3.6 Wybór modelu przetwarzania równoległego....53
3.7 Zarządzanie wykonywaniem aplikacji Faces....57
3.8 Przebieg działania aplikacji Faces....58
4 Charakterystyka węzła obliczeniowego....62
4.1 Badane charakterystyki węzła obliczeniowego....62
4.1.1 Standardowe benchmarki węzła obliczeniowego....62
4.1.2 Struktura danych....64
4.1.3 Model węzła obliczeniowego....65
4.1.4 Procedura oceny charakterystyk wydajnościowych i wiarygodnościowych węzła obliczeniowego....68
4.1.5 Oczekiwane obciążenia węzła....73
4.2 Wyznaczenie funkcji korekty dla strumieni PAL....74
4.2.1 Strumień PAL: Przekaźnik....74
4.2.2 Strumień PAL: Zegar....75
4.2.3 Strumień PAL: Detekcja twarzy, na co 32-giej klatce....76
4.2.4 Strumień PAL: Detekcja twarzy, na co 16-tej klatce....77
4.2.5 Wyznaczenie funkcji korekty dla strumieni PAL....78
4.3 Weryfikacja funkcji korekty dla strumieni PAL....78
4.3.1 Strumień PAL: Detekcja twarzy, na co 4-tej klatce....79
4.3.2 Strumień PAL: Wykrywanie krawędzi....79
4.3.3 Strumień PAL: Maska tła....80
4.3.4 Strumień PAL: Scalenie obrazów....80
4.3.5 Obraz PAL: Statyczna detekcja twarzy, na co 32-giej klatce....81
4.4 Wyniki badań wydajnościowych i wiarygodnościowych – strumienie HD i audio....82
4.4.1 Strumień HD: Przekaźnik....82
4.4.2 Strumień HD: Zegar....83
4.4.3 Strumień audio: Przekaźnik....83
4.4.4 Strumień audio: Filtr częstotliwości....84
4.5 Podsumowanie badań pojedynczego węzła....85

5 Zarządzanie zadaniami na platformie KASKADA....87
5.1 Model klastra obliczeniowego dla platformy KASKADA....87
5.1.1 Alokacja węzłów dla zadań obliczeniowych....88
5.1.2 Algorytmy alokacji dla zadań bezstrumieniowych....90
5.1.3 Algorytm alokacji dla zadań strumieniowych....92
5.1.4 Alokacja MBFD dla aplikacji Faces....95
5.1.5 Skalowalność klastra obliczeniowego....96
5.1.6 Usługi benchmarkowe dla oceny algorytmów alokacji....98
5.2 Wyniki badań algorytmu alokacji: MBFD....102
5.2.1 MBFD: Detekcja twarzy na co 4-tej klatce z zapisem do pliku....102
5.2.2 MBFD: Detekcja twarzy na każdej klatce z dekompozycją danych....103
5.2.3 MBFD: Detekcja twarzy na co 32-giej klatce z generacją zdarzeń....104
5.2.4 MBFD: Scalanie strumieni z wpisywaniem czasu....105
5.3 Wyniki badań algorytmu alokacji: BFD....106
5.3.1 BFD: Detekcja twarzy na co 4-tej klatce z zapisem do plików....106
5.3.2 BFD: Detekcja twarzy na każdej klatce z dekompozycją danych....107
5.3.3 BFD: Detekcja twarzy na co 32-giej klatce z generacją zdarzeń....108
5.3.4 BFD: Scalanie strumieni z wpisywaniem czasu....109
5.4 Analiza porównawcza algorytmów alokacji....110
5.4.1 Porównanie algorytmu alokacji MBFD z LB....110
5.4.2 Porównanie algorytmu alokacji MBFD z BFD....110
6 Uwagi końcowe....112
6.1 Wykazanie tez rozprawy....112
6.2 Możliwości platformy KASKADA....114
6.3 Kierunki dalszych prac badawczych....116
Referencje....119

Wykaz oznaczeń

C	zbiór węzłów obliczeniowych klastra
c_i	i-ty węzeł klastra, $c_i \in C$
si	strumień wejściowy, $si=(si^1, si^2, \ldots si^{\lvert si\rvert})$
si^i	i-ty element strumienia wejsciowego
so	strumień wyjściowy, $so=(so^1, so^2, \ldots so^{\lvert so\rvert})$
so^i	i-ty element strumienia wyjściowego
H	zbiór typów strumieni, $H=\{hd, pal, aud, nil\}$
h	typ strumienia, $h \in H$
T	zbiór zadań
t_i	zadanie i, $t_i \in T$
a_i	algorytm wykonywany przez zadanie t_i
SI_i	zakładany zbiór strumieni wejściowych zadania t_i
SO_i	pożądany zbiór strumieni wyjściowych zadania t_i
SO_i^R	uzyskany zbiór strumieni wyjściowych dla wykonania zadania t_i
γ_i	obciążenie węzła spowodowane wyłącznie zadaniem t_i
$\gamma(c, T)$	obciążenie węzła c przez zadania ze zbioru T
$\tau r(c, T)$	suma wszystkich przedziałów czasu aktywności, w których rdzenie węzła c wykonywały obliczenia dla zadań ze zbioru T
$\gamma'(c, T)$	aproksymowane obciążenie węzła c zadaniami ze zbioru T
$v(t_i)$	typ strumieniowy zadania t_i
$\varrho_S(t_i)$	stopień strumieniowy zadania t_i
$\theta(si)$	typ strumienia si
τ_{si}	moment startu zadania t_i
τ_{fi}	moment zakończenia zadania t_i
τ_E	czas obserwacji obliczeń
$\eta(i)$	funkcja korekty obciążenia dla i strumieni
$\omega^\tau(c_i)$	zbiór zadań wykonywanych na węźle c_i, w danym momencie τ
Θ	zbiór wszystkich zadań jakie mogą być uruchomione na klastrze obliczeniowym
p	skala systemu, mierzona liczbą jednostek obliczeniowych (np. rdzeni, węzłów), w zależności od poziomu rozważań
$\xi(p, n)$	przyspieszenie obliczeń na p jednostkach obliczeniowych dla danych o rozmiarze n
$\tau(p, n)$	czas wykonania obliczeń na p jednostkach obliczeniowych dla danych o rozmiarze n
$\varepsilon(p)$	metryka skalowalności systemu o skali p

$\iota(p)$	produktywność obliczeń na systemie o skali p
$\lambda(p)$	przepustowość systemu o skali p
$f(p)$	znormalizowana jakość obliczeń na systemie o skali p
$\kappa(p)$	koszt systemu o skali p
$\tau(p)$	czas wykonania danej operacji/wołania dla zadanej skali p
τ'	oczekiwany czas wykonania danej operacji/wołania
Π_i	zbiór okresów aktywności zadania t_i
π^j_i	czas trwania j-tej aktywności realizacji zadania t_i
$\psi^\tau(C)$	funkcja fragmentacji klastra C, w momencie τ
$\boldsymbol{N}$	zbiór liczb naturalnych
$\boldsymbol{R}$	zbiór liczb rzeczywistych
$\mu^\tau(t_i)$	funkcja alokacji węzła dla zadania t_i, przeprowadzana w momencie τ
$M^\tau(C, T)$	zbiór wszystkich możliwych alokacji zadań ze zbioru T na węzły klastra C w momencie τ, $\mu^\tau \in M^\tau(C, T)$
$\mu_C{}^\tau(t_i)$	funkcja alokacji węzła dla zadania bezstrumieniowego t_i, przeprowadzana w momencie τ
$\varphi(c, t, T)$	strata danych wyjściowych dla zadania t podczas przetwarzania zbioru zadań T na węźle c
$\varphi(c, T)$	średnia strata danych wyjściowych podczas przetwarzania zbioru zadań T na węźle c
$\varphi^\tau(C)$	strata danych podczas przetwarzania zadań w momencie τ na klastrze C
L	maksymalna akceptowana średnia strata danych
α_X	współczynnik oceny algorytmu alokacji X
$k(p)$	funkcja zwiększająca indeks p w procedurze pomiaru charakterystyk klastra
R^2	kwadrat współczynnika korelacji Pearsona
$\|X\|$	liczebność zbioru X

Wykaz skrótów

3D	ang. Three-Dimensional
3DTV	ang. Three-Dimensional TeleVision
ACID	ang. Atomic, Consistent, Izolated, Durable
ACL	ang. Access Control List
ASR	ang. Automatic Speech Recognition
AVI	ang. Audio Video Interleave
BF	ang. Best Fit
BFD	ang. Best Fit Descending
BPP	ang. Bin Packing Problem
CI	Centrum Informatyczne
CMS	ang. Connection Management System
CORBA	ang. Common Request Broker Architecture
CPM	ang. Call Processing System
CPU	ang. Central Processing Unit
CRC	ang. Cyclic Redundancy Check
EDF	ang. Earliest Deadline First
EJB	ang. Enterprise Java Beans
ESB	ang. Enterprise Service Bus
FF	ang. First Fit
FFD	ang. First Fit Descending
FIFO	ang. First In First Out
fps	ang. Frames Per Second
Gb	GigaBit
GIF	ang. Graphics Interchange Format
GPGPU	ang. General-Purpose Graphics Processing Unit
GUI	ang. Graphics User Interface
HD	ang. High Definition
HQL	ang. Hibernate Query Language
HTML	ang. HyperText Markup Language
HTTP	ang. HyperText Transfer Protocol
IEC	ang. International Electrotechnical Commission
IEF	ang. Intelligent Enterprise Framework
ISO	ang. International Organization for Standardization
JEE	ang. Java Enterprise Edition

JMS	ang. Java Message System
JPEG	ang. Joint Photographic Experts Group
KASKADA	Kontekstowa Analiza Strumieni Danych z Kamer Dla Aplikacji Definiujących Alarmy
Kb	KiloBit
LB	ang. Load Balancing
LITMUS	ang. LInux Testbed for MUltiprocessor Scheduling in Real-Time systems
MB	MegaBajt
MBFD	ang. Modified Best Fit Descending
MIMD	ang. Many Instructions Multiple Data
MISD	ang. Many Instructions Single Data
MMOG	ang. Massively Multiplayer Online Game
MPEG	ang. Moving Picture Experts Group
MPU	ang. Multi-core Processing Unit
MSMQ	ang. Microsoft Message Queuing
MPI	ang. Message Passing Interface
MSP-ML	ang. Multimedia Stream Processing Modeling Language
NAS	ang. Numerical Aerodynamic Simulation
NP	ang. Nondeterministic Polynomial
ODF	ang. Open Document Format
OQL	ang. Object Query Language
PAL	ang. Phase Alternating Line
PBDS	ang. Pull-Based Distributed Scheduling
PC	ang. Personal Computer
PDF	ang. Portable Document Format
PFlop	ang. Peta Floating Point OPeration
PNG	ang. Portable Network Graphics
POV-Ray	ang. Persistence of Vision RAYtracer
PVM	ang. Parallel Virtual Machine
QoS	ang. Quality of Service
RAM	ang. Radom Access Memory
RMA	ang. Rate-Monotonic priority Assignment
RMI	ang. Remote Method Invocation
ROI	ang. Region Of Interest
ROIA	ang. Real-time Online Interactive Application
RPC	ang. Remote Procedure Call

RTF	ang. Rich Text Format
RTSP	ang. Real Time Streaming Protocol
RUP	ang. Rational Unified Process
SIMD	ang. Single Instruction Multiple Data
SISD	ang. Single Instruction Single Data
SOWI	System Ochrony Własności Intelektualnej
SQL	ang. Structured Query Language
SOA	ang. Service Oriented Architecture
SOAP	ang. Simple Object Access Protocol
SSP	ang. Subset Sum Problem
TASK	Trójmiejska Akademicka Sieć Komputerowa
TB	TeraBajt
TCP/IP	ang. Transmission Control Protocol/Internet Protocol
TIFF	ang. Tagged Image File Format
UDDI	ang. Universal Description, Discovery and Integration
UNICORE	ang. UNiform Interface to COmputing REsources
UPS	ang. Uninterruptible Power Supply
UIMA	ang. Unstructured Information Management Architecture
URL	ang. Uniform Resource Locator
VBPP	ang. Variable size Bin Packing Problem
VPN	ang. Virtual Private Network
WSDL	ang. WebService Description Language
XML	ang. eXtensible Markup Language

1 Wstęp

Zagrożenie atakami terrorystycznymi, działalnością przestępczą i innymi niebezpiecznymi zdarzeniami spowodował szybki rozwój systemów monitoringu wideo i audio, jest to szczególnie zauważalne we wszystkich obiektach użyteczności publicznej takich jak lotniska, stacje kolejowe i metra, stadiony sportowe czy nawet na zwykłych ulicach i parkach miejskich. Organizowane wydarzenia masowe, przedstawienia, koncerty czy zawody sportowe są bardzo narażone na na wszystkie problemy bezpieczeństwa wynikające ze zgromadzenia w jednym miejscu wielu tysięcy ludzi. Powodują one, że wymagane jest wykorzystanie licznych służb ochrony oraz mechanizmów zapewnienia analizy aktualnego stanu bezpieczeństwa. Powoduje to popyt na coraz bardziej złożone systemy automatycznego rozpoznawania niebezpieczeństwa na podstawie sygnałów gromadzonych przez systemy monitoringu.

Wyższa jakość otrzymywanych danych umożliwia rozwój nowych bardziej skomplikowanych algorytmów do automatycznego rozpoznawania problemów bezpieczeństwa oraz ich analizę. Strumienie wideo wysokiej rozdzielczości (HD) są używane nie tylko do przewidywania ruchu tłumów, ale także do rozpoznawania ich nastroju czy do obserwacji pojedynczych ludzi. Z drugiej strony możemy zaobserwować stały rozwój centrów komputerowych, a wraz z nimi klastrów obliczeniowych, obecnie przekraczających granicę 10PFlop [top500]. Tak duża moc obliczeniowa w połączeniu z szybkimi sieciami rozległymi opartymi na technologii światłowodowej oraz z wielkimi wolumenami pamięci masowej mogą być wykorzystane do stworzenia systemu umożliwiającego przetwarzanie dużej liczby strumieni multimedialnych.

Platforma KASKADA jest właśnie takim systemem wspierającym tworzenie i wykonywanie aplikacji przetwarzających strumienie multimedialne w czasie rzeczywistym. Jej rozwój wymagał wykorzystania nowych procedur zarządzania przetwarzaniem równoległym, w tym komunikacją, obliczeniami, zasobami i danymi. Autor rozprawy jest koordynatorem zespołu realizującego tą platformę, odpowiedzialnym za równo za stronę techniczną przedsięwzięcia jak i za zarządzanie zespołem deweloperów ją rozwijającą. Platforma powstała w ramach projektu MAYDAY EURO 2012* realizowanego przez Politechnikę Gdańską przy udziale innych uczelni: Gdańskiego Uniwersytetu Medycznego, Politechniki Warszawskiej i Akademii Górniczo-Hutniczej.

W następnym rozdziale został przedstawiony przegląd problematyki zarządzania systemami komputerowymi, ze szczególnym uwzględnieniem systemów przetwarzania strumieni multimedialnych w czasie rzeczywistym. Zdefiniowano koncepcję zarządzania takimi systemami oraz przedstawiono najważniejsze elementy składowe: jednostkę obliczeniową, oprogramowanie, dane i użytkowników. Zaprezentowano typowe warstwowe architektury: prostą dla pojedynczego komponentu oraz złożoną dla współpracujących komponentów rozproszonych. Uwzględniono podstawowe domeny zarządzania systemami: dane – dotyczące zarówno strumieni wejściowych jak i wynikowych; obliczenia – obejmujące alokację i szeregowanie zadań obliczeniowych, w szczególności dla systemów czasu rzeczywistego; komunikację – wewnętrzną jak i zewnętrzną w tym z użytkownikami; zasoby – obejmujące infrastrukturę sprzętową. W rozdziale tym przedstawiono dwie główne tezy rozprawy.

W rozdz. 3 zaprezentowano zintegrowany model zarządzania systemem komputerowym, jak również opisano koncepcję jego implementacji w platformie KASDADA. Przedstawiono architekturę platformy i zrealizowane mechanizmy zarządzania przetwarzaniem zadań multimedialnych. Przeanalizowano aplikację detekcji twarzy w strumieniu wideo w celu

* Politechnika Gdańska, „MAYDAY EURO 2012” Superkomputerowa platforma kontekstowej analizy strumieni danych multimedialnych do identyfikacji wyspecyfikowanych obiektów lub niebezpiecznych zdarzeń. Projekt współfinansowany z Europejskiego Funduszu Rozwoju Regionalnego i Budżetu Państwa w ramach Programu Operacyjnego Innowacyjna Gospodarka.

zilustrowania podstawowych problemów równoleglizacji oraz optymalnego wykorzystania zasobów. W szczególności najwięcej uwagi poświęcono optymalizacji procedur zarządzania wykorzystaniem węzłów obliczeniowych. Opisano konfigurację środowiska przetwarzania jak również dokonano jego porównania z innymi istniejącymi rozproszonymi systemami obliczeniowymi.

W rozdz. 4 rozważono i wykazano pierwszą tezę rozprawy poprzez ocenę pojedynczego węzła obliczeniowego platformy KASKADA w kontekście możliwości wykorzystania jego zasobów jako części klastra komputerowego. Opisano architekturę węzła, strukturę strumieni danych oraz kategorie zadań obliczeniowych wykonywanych na węźle. Zaprezentowano procedurę oceny charakterystyk wydajnościowych i wiarygodnościowych węzła, ze względu na rozmiar strumieni multimedialnych oraz złożoność zadań wykonujących obliczenia. Przedstawiono wyniki eksperymentów wydajnościowych oraz podano zasady wykorzystywania węzłów w przypadku równoległego przetwarzania strumieni multimedialnych.

W rozdz. 5 rozważono i wykazano drugą tezę rozprawy, poprzez opracowanie procedury zarządzania zadaniami na platformie KASKADA z wykorzystaniem wielu węzłów. Zaprezentowano heurystyczne algorytmy alokacji zadań bez analizy strumieni multimedialnych: FF (ang. first fit), FFD (ang. first fit descending), BF (ang. best fit), BFD (ang. best fit descending) i SSP (ang. subset sum problem). Zaproponowano oryginalny algorytm alokacji będący ulepszoną wersją algorytmu BFD: MBFD (ang. modified best fit descending) umożliwiający alokację zadań przetwarzających strumienie multimedialne. Przedstawiono wyniki pomiarów wydajności i wiarygodności platformy KASKADA z wykorzystaniem algorytmów BFD i MBFD.

Ostatni rozdział zawiera podsumowanie najważniejszych tematów poruszanych w rozprawie oraz wnioski wynikających z prac naukowych z nią związanych, jak również planowane dalsze prace jakie mogą być przedsięwzięte w przyszłości.

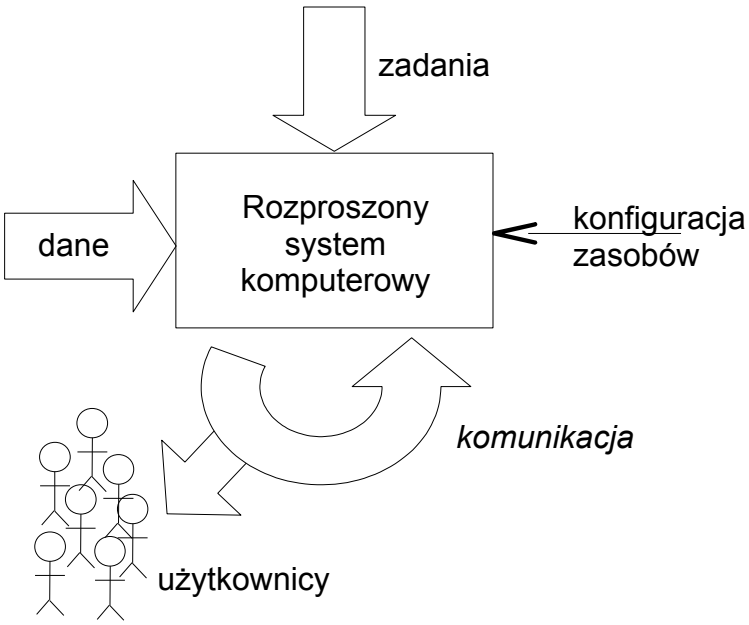
Wiele zaproponowanych rozwiązań zostało zweryfikowanych na krajowych i międzynarodowych konferencjach. W [krawczyk09] zaprezentowano architekturę i projekt platformy KASKADA przetwarzającej strumienie multimedialne z użyciem klastra komputerowego, natomiast w [krawczyk10a] skupiono się na architekturze i scenariuszach wykonywania usług. W [proficz10] kontynuowano rozwój scenariuszy analizy z bardziej praktycznego punktu widzenia oraz wykorzystania wytworzonych narzędzi. W [bobkowska11] przedstawiono ocenę języka MSP-ML – wykorzystywanego na platformie do definicji usług złożonych. W [krawczyk10b] przedstawiono problematykę związaną z alokacją zasobów obliczeniowych. W [krawczyk10a] i [krawczyk10c] zostały przedstawione aspekty równoległości obliczeń w ramce (ang. framework) platformy KASKADA – z punktu widzenia ułatwienia konstrukcji zadań obliczeniowych. Natomiast w [krawczyk11] zaprezentowano podstawowe strategie zarządzania całej platformy, ze szczególnym uwzględnieniem interakcji usług i zdarzeń. Podsumowanie doświadczeń zdobytych podczas tworzenia całej platformy znajduje się w [krawczyk12].

2 Problematyka zarządzania systemami komputerowymi

Przedstawiono problemy zarządzania systemami komputerowymi, ze szczególnym uwzględnieniem multimedialnych systemów rozproszonych, przetwarzających strumienie danych. Zdefiniowano koncepcję zarządzania takim systemem oraz uwzględniono podstawowe domeny zarządzania systemami: dane, obliczenia, komunikację i zasoby. Zaproponowano procedury zarządzania tymi domenami jak również zasygnalizowano potrzebę podejścia zintegrowanego o określonych własnościach. Takie rozwiązanie zaimplementowano dla platformy KASKADA dla której określono klasę przetwarzanych algorytmów oraz sformułowano dwie główne tezy rozprawy.

2.1 Rozwój systemów komputerowych

Systemy komputerowe realizują obliczenia zgodnie ze wskazanym przez użytkownika programem. Dodatkowo specjalistyczne oprogramowanie i/lub sprzęt wspomaga zarządzanie procesem obliczeń. Na ogół tą rolę pełnią systemy operacyjne, które również zapewniają komunikację z użytkownikiem. Na rys. 1 przedstawiono koncepcję przetwarzania w rozproszonych systemach komputerowych. Istotny jest dobór odpowiedniej konfiguracji zasobów systemu do realizowanych zadań, zapewnienie dostępu do niezbędnych danych, a także umożliwienie komunikacji pomiędzy elementami systemu.



Rysunek 1: Ogólny schemat rozproszonego systemu komputerowego.

Możemy przyjąć, że system komputerowy, składa się z czterech najważniejszych elementów:

- jednostek przetwarzających złożonych z jednego lub więcej procesorów, pamięci oraz urządzeń zewnętrznych takich jak: pamięć masowa, karta sieciowa, drukarka, skaner itd.
- oprogramowania wykonującego obliczenia (zadania obliczeniowe) oraz oprogramowania zarządzającego procesami obliczeń przy przyjętej konfiguracji zasobów i dostępnych mechanizmach komunikacyjnych,
- danych wejściowych oraz produkowanych wyników obliczeń, składowanych na archiwizujących urządzeniach zewnętrznych lub przesyłanych to systemów zewnętrznych,

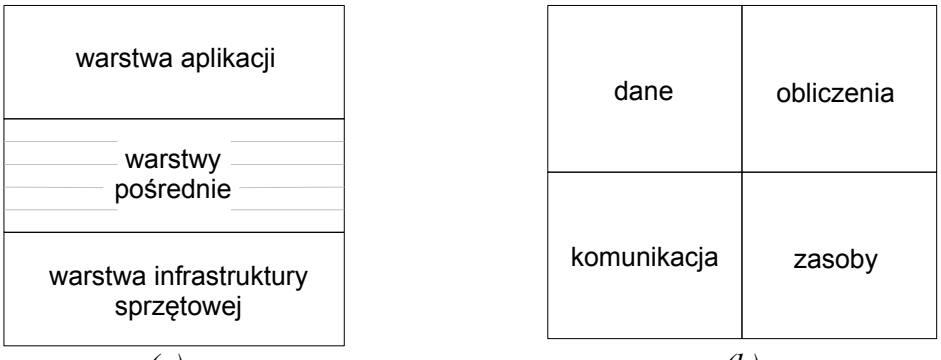
- użytkowników zlecających wykonywanie konkretnych zadań i ingerujących w proces obliczeń w taki sposób by osiągnąć zamierzone cele.

Współczesne systemy komputerowe charakteryzują się możliwością wykonywania zadań w różny sposób:

- współbieżny – gdy zadania przetwarzane są z przełączaniem czasu na jednym procesorze,
- równoległy – gdy wiele zadań jest przetwarzanych jednocześnie na wielu węzłach,
- rozproszony – gdy określone elementy zadania są realizowane w sieci na różnych serwerach połączonych siecią przewodową lub bezprzewodową, wykorzystujących odpowiednie protokoły komunikacyjne, dla zapewnienia integracji obliczeń,
- zespołowy – gdy system rozproszony wspomaga realizację przedsięwzięcia wykonywanego przez odpowiednio dobrany zespół ludzki, gdzie istotna staje się zarówno interakcja typu człowiek-komputer, jak też komputer-komputer, w trakcie realizacji tego przedsięwzięcia.

Na ogół zarządzanie definiuje się jako zestaw działań (obejmujący planowanie i podejmowanie decyzji, organizowanie, przewodzenie, tj. kierowanie ludźmi, i kontrolowanie) skierowanych na zasoby organizacji (ludzkie, finansowe, rzeczowe i informacyjne) i wykonywanych z zamiarem osiągnięcia celów organizacji w sposób sprawny i skuteczny [gryffin98]. Z punktu widzenia systemu komputerowego, powyższy zestaw działań będzie obejmował planowanie oraz podejmowanie decyzji dotyczących sterowania i kontrolowania zasobów, zarządzania danymi, zadaniami, a także zapewnienia skutecznej komunikacji.

W zależności od rodzaju przetwarzania systemy mogą być w różny sposób zorganizowane (np. pojedynczy komputer, sieć węzłów tworząca klaster [bader01], siatka [bertis01, chin05], chmura [knorr08]), a także w różny sposób zarządzane. Co więcej od organizacji systemu zależy też implementacja strategii zarządzania. Innymi słowy organizacja i architektura systemu zapewnia użycie dostępnych i odpowiednich zasobów do wykonania różnego typu funkcji związanych z obliczeniami (przetwarzanie, komunikacja, prezentacja, przechowywanie). Angażowanie zasobów oraz koordynacja ich pracy zależy już od strategii zarządzania. Do niej należy pobieranie zadań do przetwarzania, alokacja ich na odpowiednich węzłach obliczeniowych, śledzenie postępu realizacji obliczeń i obsługa sytuacji wyjątkowych.



(a) *(b)*

Rysunek 2: System komputerowy: (a) model warstwowy systemu komputerowego, (b) podstawowe domeny zarządzania.

System komputerowy możemy prezentować na różnym poziomie szczegółowości. Najbardziej obecnie popularną metodą opisu systemu jest struktura warstwowa. W ogólnym przypadku system składa się z co najmniej trzech warstw, patrz rys. 2a.

Najniższą warstwę stanowi infrastruktura sprzętowa, która dostarcza najbardziej elementarne funkcje systemu, takie jak wykonywanie rozkazów procesora czy sterowanie urządzeniami

zewnętrznymi. Dane są w niej reprezentowane np. jako bajty w pamięci RAM, obliczenia są reprezentowane przez wykonywane rozkazy procesorów, komunikacja odbywa się zwykle przez szynę danych, a zarządzanie zasobami dotyczy podstawowych funkcji systemu, takich jak zasilanie czy chłodzenie procesora. Na poziomie systemu rozproszonego wyróżniamy wiele węzłów (serwerów) komunikujących się za pomocą sieci komputerowej.

Ze sprzętem wiąże się również odpowiednie oprogramowanie zawierające sterowniki i programy bezpośrednio nim zarządzające (ang. firmware). W nowoczesnym systemie – węźle obliczeniowym każdy komponent posiada funkcje organizacyjno-sterujące, typowymi przykładami są płyta główna, karta graficzna, twardy dysk, karta sieciowa czy napęd optyczny. Ze względu na wysoką zależność tego typu oprogramowania od używanego sprzętu, jest ono zwykle dostarczane przez producenta danego produktu.

Warstwa pośrednia umożliwia organizację obliczeń na poziomie procesów czy zadań. W tej warstwie zwykle jest zlokalizowany system operacyjny. Z tą warstwą kojarzone są dane jako np. pliki, wpisy do rejestru itp. Obliczenia są opisywane jako procesy lub wątki, natomiast komunikacja opiera się na wykorzystaniu mechanizmów przesyłania komunikatów czy pamięci współdzielonej. Jako charakterystyka zasobów zwykle rozumiane są takie parametry jak np. czas dostępu do procesora czy wielkość dostępnej pamięci, a zarządzanie realizacją procesów uwzględnia wykorzystanie zadań, np. priorytety. Dla serwerów sieciowych najczęściej używanym systemem operacyjnym są różne odmiany Unixa: 64.3% z ogromną przewagą jego darmowej wersji Linuxa: 63.7% oraz Microsoft Windows: 35.7% [qsucc] (04.2012). Natomiast po stronie klienta sieciowego, Microsoft Windows jest wykorzystywany przez 90.9%, Apple 7.6%, a Linux 0.9% [atint10] (08.2010).

W przypadku systemów komputerowych składających się z większej liczby węzłów obliczeniowych często stosuje się kolejne warstwy pośrednie sterujące współpracą między nimi. Jako przykłady można wymienić proste rozwiązania z przesyłaniem wiadomości MPI [lusk97], JMS [jms], MSMQ [msmq]; systemy z wywoływaniem procedur i obiektów: RPC [rpc], RMI [rmi], CORBA [corba] oraz usług SOAP [soap], a także systemy wirtualnych maszyn z przesyłaniem wiadomości: PVM [pvm] i pamięcią współdzieloną: VSMP [vsmp].

Najwyższą warstwą systemu komputerowego stanowią aplikacje dostarczające funkcjonalności użytkowej oraz umożliwiające współpracę z użytkownikiem, zwykle poprzez graficzny interfejs użytkownika (GUI, ang. graphical user interface). Dane, na tym najwyższym poziomie abstrakcji reprezentowane są jako wpisy w formularzach, dokumenty tekstowe, bądź pliki z zawartością multimedialną itp. Obliczenia zwykle wykonywane są w tle, w stosunku do obsługi użytkownika, a postęp realizacji zadania jest reprezentowany graficznie za pomocą odpowiednich wskaźników. Komunikacja z kolei wykorzystuje mechanizmy typowe dla działań rzeczywistych: pocztę (e-mail) czy rozmowy (chat). Podobnie zasoby w tym urządzenia zewnętrzne, takie jak skaner czy cyfrowy aparat fotograficzny są reprezentowane przez foldery bądź ikony umieszczone na pulpicie.

Obecnie najczęściej stosowanym modelem współpracy w warstwie aplikacji jest model klient-serwer. Na poziomie interfejsu użytkownika zwykle wykorzystuje się przeglądarkę internetową (lekki klient) albo specjalne dedykowane aplikacje (ciężki/bogaty klient). W tym pierwszym przypadku najczęściej stosowane przeglądarki to: Firefox 43.5%, Internet Explorer 27.5% i Chrome 22.4% [refsnes10] (12.2010). Natomiast w przypadku drugim, dostarczane jest specyficzne oprogramowanie w postaci niezależnej aplikacji (ang. stand alone application), instalowanej na komputerze użytkownika.

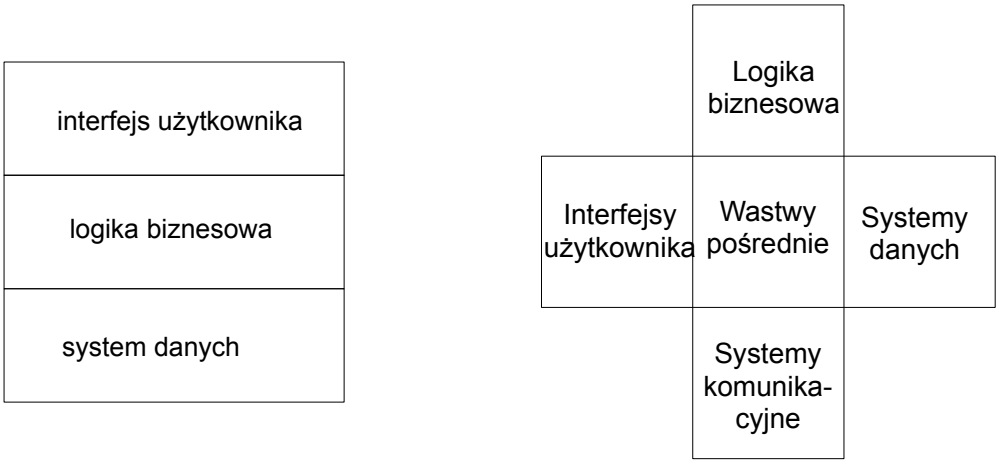
Rys. 2b przedstawia główne domeny zarządzania wykorzystywane w korelacji z warstwami z rys. 2a. Domena dane, obejmuje typowe wartości zaczynając od liczbowych, bądź tekstowe, poprzez struktury złożone (rekordy, klasy itp.) aż do strumieni danych multimedialnych. Domena obliczeń dotyczy operacji wykonywanych na danych różnego typu. Domena komunikacji obejmuje wymianę

informacji pomiędzy komponentami obliczeń, zaś domena zasoby reprezentuje infrastrukturę umożliwiającą działanie całego systemu. Należy podkreślić, że każda z tych domen w każdej z warstw systemu komputerowego (rys. 2a) może być realizowana w różny sposób. I tak dla infrastruktury sprzętowej komunikacja dokonuje się poprzez szynę danych, natomiast na poziomie współpracy z użytkownikiem poprzez graficzny interfejs (GUI). Natomiast w przeglądarce internetowej pomiędzy rozproszonymi komponentami aplikacji może być realizowana poprzez protokół TCP/IP.

Każda aplikacja reprezentująca zadanie zlecone przez użytkownika może też się składać z trzech warstw, patrz rys. 3a. Warstwa bazy danych odpowiedzialna jest za przechowywanie i zarządzanie danymi. Warstwa logiki biznesowej umożliwia implementacje i wykonywanie odpowiednich algorytmów obsługujących żądania użytkownika w oparciu o dane wprowadzone przez użytkownika lub otrzymane z warstwy niższej. W warstwie interfejsu użytkownika znajdują się mechanizmy umożliwiające bezpośrednią współpracę z użytkownikiem, w celu odbioru i interpretacji jego żądań oraz przedstawienia wyników przetwarzania warstwy logiki biznesowej z wykorzystaniem warstwy danych.

Biorąc pod uwagę współpracujące ze sobą aplikacje rozproszone i internetowe, ich architektura jest znacznie rozbudowana, patrz rys. 3a i 3b. Warstwy pośrednie umożliwiają kooperację wielu aplikacji i wykorzystywanie różnych komponentów: interfejsów użytkownika – poprzez re-użycie gotowych skryptów czy okienek i wprowadzenie jednolitego ich wyglądu, logiki biznesowej – zwykle udostępnianej jako repozytorium usług, systemów danych – zawierających dane współdzielone oraz systemów komunikacyjnych – ułatwiających współpracę aplikacji, zarówno w trybie synchronicznym jak i asynchronicznym [krawczyk09].

W celu usystematyzowania problemów zarządzania przyjmujemy następujące założenia. Zarządzanie dotyczy czterech domen: danych, obliczeń, zasobów i komunikacji, patrz rys. 2b.



Rysunek 3: Architektura warstwowa aplikacji: (a) prosta – dla pojedynczej aplikacji, (b) złożona – dla aplikacji rozproszonych i internetowych.

W przypadku danych polega na odpowiednim ich przygotowaniu i przekazywaniu adresatom np. zadaniom obliczeniowym, istotna jest aktualność i rzetelność dostarczanych danych. W przypadku obliczeń (tj. wykonania wymaganych zadań) stosuje się następujące kryteria jakościowe: wydajność, w tym skalowalność i wiarygodność. W zarządzaniu zasobami ważną rolę odgrywają takie cechy jak bezpieczeństwo, odporność na awarie, koszty zużycia energii. Zarządzanie komunikacją obejmuje między innymi sterowanie zdarzeniami, dla podejmowanie właściwych decyzji, jak również informowanie użytkownika o specyficznych sytuacjach zachodzących w środowisku systemu komputerowego, dzięki czemu wyniki obliczeń mogą na czas trafić do

odpowiedniego adresata.

2.2 Zarządzanie danymi

System komputerowy (patrz rys. 1) przetwarza dane wejściowe według zadanych algorytmów, analizuje ich poprawność, modyfikuje do wymaganych standardów i reprezentuje wyniki jako dane wyjściowe. Dane klasyfikujemy ze względu na ich wielkość oraz typ i specyfikę ich przechowywania. System komputerowy powinien dostarczać mechanizmów umożliwiających ich wyszukiwanie oraz operowanie na nich jak również zapewniać ich transformację w celu zapewnienia wymaganej formy prezentacji.

Istnieje wiele standardów reprezentacji danych, te same dane mogą być przedstawione i przetwarzane w wielu formatach. Przykładowo dla dokumentów tekstowych, możemy wyróżnić formaty: XML, HTML, RTF itd., dla obrazów statycznych: JPEG, GIF, PNG, TIFF itd., mamy też formaty hybrydowe: PDF, ODF (ISO/IEC 26300) oraz formaty multimedialne: AVI, MPEG, MKV.

Dane jednostkowe – są określane jako proste parametry przekazywane między komponentami systemu – np. poprzez aplikacje klienta lub bezpośrednio przez użytkownika. Zwykle są one przedstawiane jako parametry wywołania określonej funkcji – zdalnej lub lokalnej albo definiowane poprzez interfejs użytkownika. Oprogramowanie dla tego typu danych, np. zawarte w systemie operacyjnym, umożliwia zwykle wstępną walidację i parsowanie argumentów.

Dane składowane – są to dane o dużym rozmiarze, często ze skomplikowanymi zależnościami między elementami składowymi. Najczęściej są one przechowywane na dyskach jako pliki (dokumenty tekstowe, obrazy itp.) lub w specjalizowanych bazach danych (dane ze złożoną strukturą): relacyjnych czy obiektowych. Specjalistyczne oprogramowanie bazodanowe umożliwia wykonywanie zapytań w celu odnalezienia poszukiwanych danych o znanych zależnościach, lub o pewnych właściwościach.

Najpopularniejszym obecnie językiem zapytań jest SQL [sql92] (ang. structured query language), który wykorzystuje w tym celu relacje między encjami danych składowanych w tablicach i związanych ze sobą relacjami. Inne języki takie jak OQL [cattel00] (ang. object query language), EQL [eql] (ang. EJB [ejb] query language) czy HQL (ang. Hibernate [elliot04, hiber] query language) traktują dane jako obiekty, których relacje wzbogacone są o typowo obiektowe zależności, takie jak dziedziczenie czy polimorfizm [booch93].

Strumienie danych stanowią dane o dużej wielkości, napływające w czasie ich generowania on-line, z takich źródeł jak różnego typu czujniki i systemy monitorujące. Ich rozmiar oraz specyfika przetwarzania może utrudnić ich przechowywanie. Wymagane są dla nich duże i wydajne systemy archiwizacyjne, ale powinny być one przekazywane do odpowiednich zadań obliczeniowych na bieżąco. System komputerowy powinien zatem wspierać przetwarzanie tego typu danych poprzez dostarczenie mechanizmów ich konwersji, dekodowania oraz wykonywania zapytań na podstawie ich zawartości (ang. Data Stream Management [bai08]). Przykładem strumienia danych jest strumień danych multimedialnych z kamer wideo czy urządzeń audio.

Zarządzanie danymi skupia się na dostarczeniu mechanizmów dostępu, tworzenia, modyfikacji i usuwania, z uwzględnieniem zadanych kryteriów optymalizacji. Przy dostępie do danych, system może dokonać selekcji źródła danych, co jest wymagane z uwagi na zwiększenie niezawodności lub wydajności i jest możliwe gdy posiadamy kilka kopii danych. Odczyt danych jest związany z kontrolą ich poprawności i w przypadku wystąpieniu błędu, komponent zarządzający powinien powtórzyć odczyt lub wygenerować zdarzenie – wyjątek do warstwy wyższej/użytkownika.

Zarządzanie gromadzeniem danych wymaga analizy i modyfikacji struktury danych w przypadku dodawania nowego typu/klasy danych, bądź alokacji większej ilości pamięci operacyjnej lub

masowej – w zależności od sposobu przechowywania danych. Zarządzanie modyfikacją i usuwaniem danych musi brać pod uwagę takie problemy jak sposób dostępu równoległy/rozproszony, niezbędne blokady danych przy operacjach zapisu oraz odświeżanie buforów pamięci podręcznej. Typowym rozwiązaniem tego typu problemów jest wprowadzenie mechanizmu ACID, które zapewniają, że każda transakcja jest niepodzielna (ang. Atomic), dane przez nią przetwarzane są spójne (ang. Consistent), izolowane (ang. Izolated) oraz pozostają na trwale (ang. Durable) zapisane w pamięci masowej [haerder83].

Osobną kategorią problemów zarządzania danymi jest zagadnienie optymalizacji mechanizmów tworzenia, odczytu, modyfikacji i usuwania. Typowym przykładem jest indeksowanie plików przez system operacyjny w celu szybszego ich przeszukiwania, innym są specjalne indeksy tworzone w relacyjnej bazie danej w celu przyspieszenia działania najczęściej wykonywanych zapytań SQL [sql92].

Warto podkreślić, że typowe przechowywanie danych nie jest całkowicie odporne na błędy systemu. Istnieje wiele sposobów zapobiegania i odtwarzania danych utraconych wskutek awarii sprzętu, błędu w oprogramowaniu, ataku wirusa lub hakera, czy też pomyłki użytkownika. Do metod zapobiegających błędom zaliczamy tworzenie kopii zapasowej oraz replikację danych na wielu woluminach. Wyróżniamy dwa podstawowe podejścia przy wykonywaniu kopii zapasowej: składowanie całościowe – kiedy wszystkie dane są zapisywane, oraz składowanie przyrostowe, kiedy zapisywana jest tylko różnica dla nowo dodanych/modyfikowanych danych. Najczęściej stosowanym podejściem jest podejście mieszane, kiedy w regularnych odstępach czasu wykonuje się kopie całościowe, a następnie częstsze kopie przyrostowe.

Innym sposobem zwiększenia niezawodności składowania danych jest, duplikowanie woluminów z zapisanymi danymi, np. z użyciem macierzy dyskowej. W tym przypadku, dane są zapisywane wielokrotnie na osobnych urządzeniach pamięci masowych, najczęściej dyskach twardych, dzięki czemu w przypadku awarii jednego z nich, są one nadal dostępne. Taką funkcję na ogół posiadają systemy archiwizacji.

Bez względu na rodzaj danych wyróżniamy kilka ich charakterystycznych cech:

- Rozmiar – określenie wielkości danych, najczęściej używane jednostki to bajty dla danych składowanych, dla strumieni danych może być określony przez czas trwania (w sekundach) oraz wykorzystane pasmo sieciowe (MB/s).
- Struktura – określa budowę wewnętrzną danych, pod względem zależności między danymi, dane mogą być kontekstowe lub bezkontekstowe, powiązane ze sobą, hierarchiczne itp.
- Ziarnistość – określa poziom szczegółowości opisu danych, może to postać binarna lub zdefiniowana struktura. Wyraża się przez rozmiar niezależnych elementów danych – np. dla strumienia danych PAL ziarnistość pojedynczej klatki obrazu wynosi 1MB, dla typowego strumienia audio, pojedyncza próbka zawiera 2 bajty.
- Format – określa w jaki sposób mają być opisane i interpretowane dane binarne, tekstowe lub mieszane. Należy rozpoznać tu wiele standardów ich opisu: XML, ODF, MPEG itd.
- Entropia – określa ilość informacji przypadającą na jednostkę rozmiaru danych, przy czym dane mogą być skompresowane, kodowane, wzbogacone o kod CRC itp.
- Spójność – określa zgodność opisu danych i możliwości ich wykorzystania przy realizacji transakcji, równoległego dostępu czy zapewnienia integralności.

Dane mogą być oceniane za pomocą wielu charakterystyk jakościowych, np. w [wang96] przedstawione zostało aż 179 różnych miar jakościowych. W dalszych pracach można znaleźć propozycję zgrupowania tych miar, istotną analizę przedstawiono w [wand96], gdzie zdefiniowano

następujące kategorie cech jakościowych:

- dokładność (ang. accuracy, precision) – reprezentuje jak dane w systemie reprezentują rzeczywisty stan, który opisują,
- wiarygodność (ang. reliability) – określa ją kilka parametrów (1) prawdopodobieństwo zapobieganiu błędom, (2) zgodność i rzetelność informacji wyjściowych lub (3) jak bardzo dane odpowiadają założonym charakterystykom,
- aktualność i powszechność (ang. timeliness and currency) – określają czy dane są nadal ważne i czy można je uzyskać w określonym czasie,
- kompletność (ang. completness) – określa zdolność systemu przetwarzającego dane do przedstawienia wszystkich znaczących stanów opisywanej rzeczywistości,
- zgodność (ang. consistency) – określa czy dane, ich zakres wartości oraz fizyczna reprezentacja są takie same – powtarzalne, dla danej sytuacji.

Innym ważnym aspektem zarządzania danymi jest kontrola dostępu, której celem jest ograniczanie wykonywania możliwych czynności, jakie mogą być wykonane bezpośrednio przez użytkownika lub przez program/aplikację działającą na jego zlecenie. Typowym sposobem modelowania jest przedstawienie zależności między encjami danych, lub uogólniając: obiektami systemu, a użytkownikami jako macierzy dostępu [sandhu94].

W tabeli 1 została przedstawiona macierz dostępu dla przykładowych strumieni danych multimedialnych. W wierszach wymienieni są użytkownicy, kolumny odpowiadają kolejnym obiektom, natomiast w poszczególnych komórkach przedstawione są możliwe akcje jakie konkretny użytkownik może wykonać na danym obiekcie. Na przykład użytkownik Jan w odniesieniu do kamery na lotnisku, może odczytywać dane on-line, może je zachować w archiwum, jak również może edytować parametry konfiguracyjne kamery.

Tabela 1: Macierz dostępu dla przykładowych danych strumieniowych.

użytkownik/obiekt	kamera-dworzec	kamera-lotnisko	mikrofon-lotnisko
Adam	odczyt, edycja		
Michał	odczyt	odczyt	odczyt
Jan		odczyt, edycja, rejestracja	oczyt, rejestracja

Sami użytkownicy również traktowani są jako obiekty – np. administrator może modyfikować ich dane lub prawa dostępu. Ponieważ nie tylko ludzie mogą wykonywać konkretne operacje na obiektach, dlatego użytkownika lub program (aplikację, proces, usługę) go reprezentujący nazywa się tematem (ang. subject). Wyróżniamy następujące typowe sposoby implementacji macierzy dostępu [sandhu94]:

- Lista kontroli dostępu (ang. Access Control List, ACL) – z każdym obiektem skojarzona jest lista wskazująca dla każdego tematu w systemie jakie operacje może on wykonać na kolejnych obiektach. Takie listy odpowiadają kolejnym kolumnom macierzy dostępu.
- Możliwości (ang. capabilities) – z każdym tematem skojarzona jest lista wskazująca, jakie operacje na jakich obiektach może on wykonać. Takie listy odpowiadają kolejnym wierszom macierzy dostępu.
- Relacje autoryzacji (ang. authorization relations) – wprowadzony jest zbiór relacji między kolejnymi tematami i obiektami, nie jest on na stałe przypisany ani do tematów, ani do dostępnych obiektów.

2.3 Zarządzanie zasobami

Przez zasoby systemu rozumiemy niezbędne środki do realizacji zadania obliczeniowego przedstawianego na poziomie procesu, wątku, wykonywanej usługi. Tego typu środki stanowią sprzęt komputerowy, system operacyjny oraz warstwy pośrednie. Możliwości przetwarzania zależą bezpośrednio od architektury używanego sprzętu oraz osadzonego na nim systemu operacyjnego/warstwy pośredniej. Ze względu na równoległość przetwarzania, wyróżniamy następujące architektury pojedynczych komputerów [flynn72]:

- SISD (ang. single instruction single data), gdzie pojedyncza instrukcja jest wykonywana przez procesor na pojedynczej porcji danych,
- SIMD (ang. single instruction multiple data), gdzie wektor różnych danych jest przetwarzany równolegle przez tą samą instrukcję,
- MISD (ang. many instructions single data), gdzie pojedyncza dana jest przetwarzana przez różne instrukcje,
- MIMD (ang. many instructions multiple data), gdzie wiele procesorów wykonuje różne instrukcje na różnych danych.

Wiele pojedynczych komputerów może zostać połączonych za pomocą sieci w system rozproszony, na różny sposób tworząc architektury rozproszone:

- klaster [bader01] (ang. cluster computing) – gdzie homogeniczne węzły obliczeniowe połączone są szybką, jednolitą siecią komputerową – zwykle dominuje scentralizowane zarządzanie, z wykorzystaniem systemu kolejkującego zadania do wykonania,
- siatka komputerowa [bertis01] (ang. grid computig), którą stanowią heterogeniczne węzły obliczeniowe, rozproszone geograficznie i połączone Internetem, widziane przez użytkownika jako zaoby obliczeniowe,
- chmura komputerowa [buyya11] (ang. cloud computing) to przetwarzanie skoncentrowane na dostarczaniu różnego typu usług, a nie tylko mocy obliczeniowej. Jest to infrastruktura usług (ang. Infrastructure as a Service) dzięki której użytkownik otrzymuje na zamówienie właściwe oprogramowanie, włączając w to licencje i obsługę techniczną,
- niebo komputerowe [keahey09] (ang. sky computing) oznacza przetwarzanie w wielu chmurach, przy założeniu dostępu do chmur pochodzących od różnych dostawców wraz ze standardami i mechanizmami integracji poszczególnych różnych komponentów.

Tanenbaum [tanenbaum97] zwraca uwagę na następujące cechy istotne podczas projektowania rozproszonych systemów operacyjnych. Należą do nich: przezroczystość, niezawodność, wydajność i skalowalność.

Przezroczystość umożliwia uzyskanie funkcjonalności jednolitego systemu dostarczającego funkcjonalność dla uruchamianych programów. Wyróżniamy następujące rodzaje przezroczystości:

- położenia – zadania obliczeniowe mają dostęp do zasobów w taki sposób, że ich fizyczne położenie nie ma znaczenia,
- wędrówki – referencje (nazwy) umożliwiają dostęp do zasobów bez względu na którym węźle zadania są zlokalizowane,
- zwielokrotniania – w przypadku tworzenia różnych kopii tych samych zasobów, konkretne zadanie obliczeniowe ma dostęp do jednej z nich,
- współbieżności – dla tego rodzaju przezroczystości, zadania jednego użytkownika nie

interferują z zadaniami drugiego – np. poprzez wzajemnie wykluczanie w dostępie do konkretnego zasobu,

- równoległości – potencjalnie jest możliwe wykonywanie zadań sekwencyjnych w sposób równoległy bez ich modyfikacji przez programistę/projektanta; obecnie jest to trudne do realizacji.

Elastyczność jest bezpośrednio związana z budową jądra systemu operacyjnego, lub uogólniając z budową warstwy pośredniej. Można założyć dwa skrajne podejścia: jądro monolityczne, gdzie wszystkie funkcje znane są a priori, lub mikro-jądro, gdzie dostarczono tylko podstawową funkcjonalność (komunikacja między zadaniowa/procesowa, zarządzanie pamięcią, zarządzanie zadaniami/procesami oraz niskopoziomowe operacje wejścia/wyjścia), natomiast pozostałe oprogramowanie systemowe jest dostarczone w postaci dodatkowych modułów. Przy powyższych założeniach, zmieniając system na bardziej monolityczny tracimy elastyczność, ale możemy zyskać na wydajności.

Niezawodność jest cechą jakościową określającą prawdopodobieństwo poprawnej pracy systemu w danym momencie. W rozproszonych systemach operacyjnych warstwy pośrednie umożliwiają zwielokrotnienie zasobów sprzętowych w celu zapewnienia wyższej niezawodności, w tym także odporności na błędy. W takim przypadku uszkodzony komponent może być zastąpiony przez działającą inną kopię.

Wydajność jest cechą określającą maksymalną liczbę obliczeń możliwą do wykonania w danym przedziale czasu na danym systemie. Jej rozwinięciem jest skalowalność, określająca możliwość zapewnienia wyższej wydajności poprzez rozbudowę zasobów systemu komputarowego. Dla systemów równoległych/ rozproszonych wyróżniamy dwa rodzaje skalowalności:

- pionową – kiedy system może zapewnić wyższą wydajność poprzez rozbudowanie pojedynczego węzła obliczeniowego: np. poprzez dodanie pamięci operacyjnej, szybszego procesora lub dysku,
- poziomą – kiedy wyższą wydajność może być zapewniona przez dodanie nowych węzłów obliczeniowych.

Zarządzanie konfiguracją umożliwia wpływanie na powyższe cechy systemu rozproszonego, bez konieczności zmiany kodu źródłowego oprogramowania systemowego, poprzez zmianę konkretnych parametrów, w zależności od konkretnych wymagań. Przykładem może być zwiększenie niezawodności, poprzez wykorzystanie dodatkowej wersji zadania obliczeniowego, co również prowadzi do zmniejszenia wydajności systemu.

Zarządzanie zasobami systemu komputerowego, w tym jego konfiguracją, może być wykonywane w sposób scentralizowany lub rozproszony. Scentralizowane systemy zarządzania zwykle są bardziej wydajne – nie wymagają dodatkowej komunikacji między rozproszonymi komponentami. Algorytmy scentralizowanego zarządzania posiadają globalną informację o stanie systemu, natomiast nie są one tak niezawodne jak również są słabiej skalowalne (upadek węzła zarządzającego dezorganizuje pracę systemu) niż systemy rozproszone.

2.4 Zarządzanie komunikacją

Przez komunikację rozumiemy zbiór mechanizmów umożliwiających przekazywanie danych pomiędzy poszczególnymi elementami systemu komputerowego, w zależności od poziomu abstrakcji: pomiędzy zadaniami, procesami, węzłami obliczeniowymi czy podsystemami jak również wymianę informacji na zewnątrz – z użytkownikami i innymi systemami komputerowymi. Z punktu widzenia oprogramowania wyróżniamy dwa podstawowe sposoby komunikacji i synchronizacji [el-rewini04]:

- pamięć współdzielona – zadania mają dostęp do wspólnego obszaru pamięci, jak również do związanych z nią mechanizmów synchronizacji, np. sekcje krytyczne, semafory, monitory itp.
- przesyłanie wiadomości – gdzie zadania komunikują się przez przez sieć, wykorzystując pewne mechanizmy synchronizacji np. bariera, oczekiwanie blokujące, itp.

W przypadku pamięci współdzielonej kod programu może być wspólny dla kilku zadań, natomiast dla systemów z przesyłaniem wiadomości wymagane są mechanizmy zdalnego wywoływania. Najprostszym rozwiązaniem jest zdalne wywoływanie procedur RPC, gdzie argumenty wywołania procedury są kodowane (ang. marshaling), wysyłane siecią, odkodowane (ang. unmarshaling), następnie jest wykonywana odpowiednia procedura/funkcja, wynik jest kodowany, przesyłany i odkodowywany przez wołającego.

Rozwinięciem tego podejścia jest zdalne wykonywanie metod zdalnych obiektów. Przyjęte standardy CORBA [corba], RMI [rmi], EJB [ejb] itd. umożliwiają utworzenie, rejestrację i dostęp do obiektu znajdującego się na odległej maszynie, jak również wywoływanie ich metod w podobny sposób jak zdalnych procedur RPC [rpc].

Obecnie najbardziej popularnym sposobem komunikacji, w celu wykonywania obliczeń na odległym komputerze jest architektura zorientowana na usługi – SOA [welke10, erl09] (ang. service oriented architecture) umożliwiająca rejestrację, wyszukiwanie, ocenę i wykonywanie potrzebnej funkcjonalności (usługi) z użyciem szyny usługowej ESB [chappell04] (ang. enterprice service bus), przy wykorzystaniu różnych protokołów komunikacyjnych np. SOAP [soap].

Każde zlecenie komunikacyjne wysłania-odbioru (ang. send-receive) może być wykonane w sposób synchroniczny – kiedy przepływ sterowania jest wstrzymywany na czas jego wykonania, albo asynchroniczny – w przypadku przeciwnym. Standard MPI [lusk97] jest dobrym przykładem wykorzystania różnych możliwości wykorzystania powyższych mechanizmów i wspiera następujące *tryby komunikacji* [el-rewini04]:

- *wysyłanie standardowe* (ang. standard send) ma miejsce kiedy przepływ sterowania w zadaniu wysyłającym jest blokowany, aż do momentu kiedy dane są skopiowane do bufora komunikacyjnego albo do bufora systemu operacyjnego po stronie wysyłającego,
- *odbiór blokujący* (ang. blocking receive) zachodzi kiedy przepływ sterowania w zadaniu odbierającym jest wstrzymany dopóki dane nie zostaną otrzymane,
- *wysyłanie buforowane* (ang. buffered send) jest podobne do standardowego, z tą różnicą, że system gwarantuje możliwość użycia bufora komunikacyjnego, tak że nie ma oczekiwania na zwolnienie bufora systemu operacyjnego,
- *wysyłanie synchroniczne* (ang. synchronous send) zachodzi kiedy przepływ sterowania jest zatrzymany po stronie wysyłającej i odbierającej, aż do momentu przesłania danych,
- *wysyłanie na gotowość* (ang. ready send) ma miejsce kiedy przepływ sterowania jest wznawiany równolegle z odbiorem wiadomości, jednak bez oczekiwania na jego zakończenie,
- *komunikacja nieblokująca* (and. nonblocking communication) zachodzi kiedy zadanie inicjuje operację wysyłania/odbioru, a przepływ sterowania nie jest zatrzymywany, aż do momentu kiedy zadanie zdecyduje się zakończyć operację.

Implementacja powyższych rozwiązań wykorzystuje szereg standardowych mechanizmów i wzorców projektowych, umożliwiających łatwą wymianę danych, także między osobnymi systemami, m.in. w celu ich integracji. Wyróżniamy tu dwa podstawowe *tryby wywoływania*

zdalnego, umożliwiające zmianę relacji zależności [krawczyk09]:

- request-response – kiedy system odbierający żądanie wykonania funkcjonalności (usługi, funkcji, metody, procedury itd.) i zwracający odpowiedź jest znany systemowi zlecającemu wraz ze szczegółami interfejsu, formatu przesyłanych danych itd.
- publish-subscribe – kiedy system uzgadnia wykonanie żądania z systemem zlecającym i potrafi zidentyfikować wołającego, wraz ze szczegółami jego interfejsu, formatu przesyłanych danych itd., a następnie po rejestracji oczekuje na publikację zlecenia i wykorzystać go w określonych sytuacjach.

Z punktu widzenia komunikacji rozległych systemów rozproszonych (ang. large-scale distributed systems) możemy wyróżnić następujące rodzaje połączeń komunikacyjnych [haddad11]:

- point-to-point – kiedy całość komunikacji zachodzi pomiędzy nadawcą i pojedynczym odbiorcą,
- multicast – kiedy nadawca wysyła jednocześnie do kilku wybranych odbiorców,
- broadcast – kiedy nadawca wysyła jednocześnie do wszystkich uczestników danej części sieci (podsieci).

Zarządzanie komunikacją polega na takim wyborze parametrów wysyłania i odbioru, aby uzyskać żądany poziom jakości przesyłania danych. Przykładami typowych parametrów wykorzystywanych do optymalizacji są wielkość buforów wysyłania i odbioru, rodzaj routing'u, czasy (ang. timeuot) retransmisji danych przy braku potwierdzenia odbioru itp.

Typowymi charakterystykami jakościowymi systemów komputerowych ze względu na komunikację są:

- przepływność łączą określona przez szybkość z jaką strumień danych jest przekazywany między poszczególnymi komponentami systemu (np. węzłami obliczeniowymi),
- opóźnienie łącza, czas jaki mija między wysłaniem pierwszego bajtu danych, a jego odbiorem,
- utrata danych, procentowy współczynnik ilości utraconych danych do ich całkowitej wielkości,
- cechy przesyłanych danych: rozmiar, struktura, format itd. (patrz rozdz. 2.2).

2.5 Zarządzanie obliczeniami

W nowoczesnych systemach komputerowych obliczenia są wykonywane przez procesor, ich głównym zadaniem jest przetwarzanie danych wejściowych lub danych wynikowych obliczeń. Sekwencja wykonywanych obliczeń jest określana jako przepływ sterowania, który, w zależności od użytej architektury systemu komputerowego (rozdz. 2.3), może współdziałać na wspólnej pamięci współdzielonej – wtedy nazywany jest wątkiem, lub może być odizolowany i korzystać z własnej przestrzeni adresowej – nazywany jest wówczas procesem. Jako zadanie rozumiemy zbiór kilku wątków współdzielących pamięć i współpracujących z innymi zadaniami poprzez mechanizmy współpracy rozproszonej, np. przekazywanie komunikatów, lub zdalne wywoływanie procedur (rozdz. 2.4).

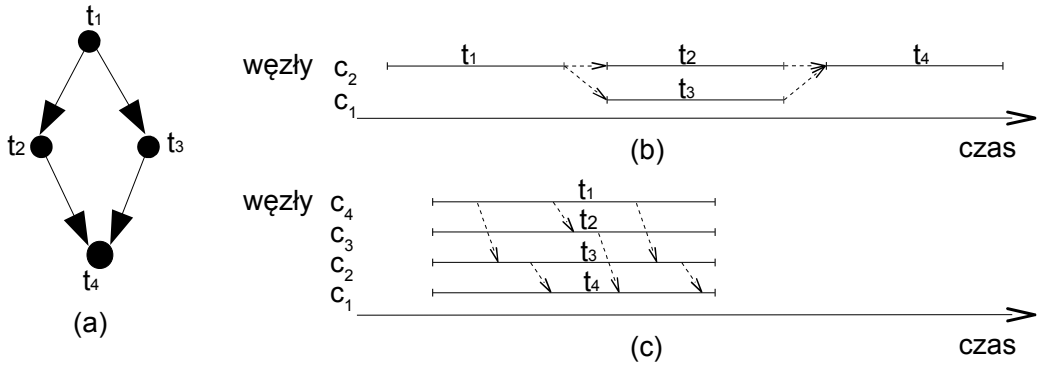
Rozróżniamy kilka najczęściej stosowanych modeli obliczeń rozproszonych:

- master-slave – gdzie wyróżniamy zadanie główne (master) rozwiązujące problem, które dokonuje jego podziału (funkcjonalnego lub danych) na podproblemy, przesyła je do

wykonania przez podzadania: slaves i następnie scala otrzymane od nich wyniki,

- klient-serwer – gdzie klient zwraca się do serwera w celu wykonania odpowiednich obliczeń (np. usługi). W przypadku dostępu do wielu serwerów klient może wykorzystać mechanizmy pośrednie, takie jak wyszukiwanie odpowiednich usług z wykorzystaniem katalogu np. UDDI [uddi],
- klient-agent-serwer – model klient-serwer rozszerzony o warstwę agentów, które są pośrednikiem między klientami a serwerami i umożliwiają łatwą rozbudowę, migrację obliczeń rozproszonych ze szczególnym uwzględnieniem symulacji [troitzsch09, wooldridge09],
- workflow – gdzie dane wejściowe są przesyłane między poszczególnymi jednostkami przetwarzającymi konkretne zadania, tworzącymi potok danych i obliczeń,
- peer-to-peer – gdzie obliczenia są wykonywane w sposób symetryczny przez równoprawnych partnerów współpracujących do osiągnięcia celu [haddad11],
- mieszany – z użyciem kilku z powyższych rozwiązań.

Tak więc zarządzanie wykonaniem zbioru zadań w równoległych bądź rozproszonych systemach komputerowych zgodnie z przyjętym modelem obliczeń polega na doborze potrzebnych zasobów obliczeniowych. Jeśli wykonywane zadania są ze sobą powiązane relacją poprzedzania – podczas ich wykonywania musi być zachowana odpowiednia kolejność ich zakończenia tzn. przynajmniej jedno z planowanych zadań z kolejki oczekujących może zostać uruchomione dopiero po wykonaniu innego konkretnego zadania. Wówczas przydział zasobów obliczeniowych dla poszczególnych zadań w odpowiedniej kolejności nazywamy **szeregowaniem** (ang. scheduling) [el-rewini04]. Kiedy zadania są między sobą niezależne, w sensie następstwa, a jedynie komunikują się między sobą to wystarczająca jest **alokacja** zadań (ang. task allocation) na węzłach obliczeniowych [el-rewini04]. Należy podkreślić, że w wielu pracach powyższe pojęcia używane są jako synonimy, np. [vidyarthi10].



Rysunek 4: Zadania i ich wykonanie: (a) graf zadań, (b) wykonanie z wykorzystaniem algorytmu szeregowania, (c) wykonanie z wykorzystaniem algorytmu alokacji.

Na rys. 4a został przedstawiony przykład grafu 4 zadań. Strzałki łączące te zadania mogą być rozważane jako relacja następstwa, kiedy np. przetwarzany jest pojedynczy obraz, wtedy to zadanie t_1 musi zostać zakończone zanim zacznie się zadanie t_2 i t_3, podobnie zadanie t_4 może być uruchomione po zakończeniu t_2 i t_3. Dla takich założeń w celu wykonania wszystkich zadań konieczne jest użycie algorytmu szeregowania zadań, którego przykładowy wynik działania został przedstawiony na rys. 4b. Natomiast, jeżeli założymy że strzałki na rys. 4a oznaczają kierunek komunikacji, kiedy np. przetwarzany jest cały strumień (klatek) obrazów, to wszystkie zadania t_1- t_4 powinny być uruchomione jednocześnie. Dla takich założeń konieczne jest wykorzystanie

algorytmu alokacji, którego którego przykładowy wynik działania został przedstawiony na rys. 4c.

Zarówno szeregowanie jak i alokacja jednego zbioru zadań może być wykonana na wiele sposobów, dlatego też można wprowadzić różne kryteria optymalizacji. Najbardziej klasycznymi kryterium jest minimalizacja czasu wykonania i minimalizacja kosztów wykonania. Dokładne algorytmy umożliwiające ich realizację są na ogół NP-trudne [vidyarthi10]. Dlatego najczęściej dla szeregowania używane heurystyki wykorzystują kolejkę FIFO zaś dla alokacji równoważenie obciążenia (ang. load-balancing, [watts98, culler99]).

Pomimo tych ograniczeń, wiele systemów komputerowych umożliwia alokację zadań wraz z gwarancją wypełnienia dodatkowych wymagań jakościowych lub funkcjonalnych, takich jak spełnienie jednego z kryteriów: utrzymanie opóźnienia przetwarzania (ang. latency) na odpowiednio niskim poziomie [cucinotta10], minimalizacji zużycia energii elektrycznej dla systemów wbudowanych [hyeran10, lindberg12], minimalizacji rzeczywistego czasu wykonania zbioru zadań w obciążonym środowisku rozproszonym w stosunku do czasu wykonania bez udziału innych zadań [benoit10]. Ponadto wspomniane algorytmy alokacji i szeregowania zadań obliczeniowych są często uzupełniane o dodatkowe wymagania np. uwzględniające minimalizację zużycia zasobów sieciowych [xiao10], czy też zwiększenie niezawodności pracy węzłów [pezoa10], jak również mogą być modyfikowane dla specyficznych zagadnień obliczeniowych, np. podziału danych wejściowych [czarnul02], czy też uwzględnienia rozproszonego charakteru wykonywanej aplikacji [brudlo95].

Oprócz wykorzystania powyższych kryteriów algorytmy szeregowania i alokacji dzieli się na statyczne, kiedy to dane zadanie jest przypisywane do pojedynczego węzła i tam pozostaje, do chwili zakończenia swojego działania. W takim przypadku potrzebne są pewne informacje *a priori* na temat uruchamianych zadań, takie jak obciążenie obliczeniowe, sieciowe czy użycie pamięci. Podejście to jest typowe dla systemów czasu rzeczywistego, dla których zakończenie obsługi danego zdarzenia jest istotne (patrz rozdz. 2.6). Alokacja dynamiczna umożliwia uruchamianie zadań, które w późniejszym czasie mogą zostać przeniesione na inny węzeł, co wymaga wykorzystania mechanizmów migracji [vidyarthi10]. W takim przypadku nie jest wymagana znajomość parametrów zadań przed ich uruchomieniem. Tego typu informacje mogą być zaobserwowane już w trakcie wykonywania zadań, a ewentualne korekty rozmieszczenia zadań mogą być wprowadzone w trakcie obliczeń.

Problemy monitorowania i obsługi sytuacji wyjątkowych w trakcie obliczeń mogą być rozpatrywane na każdym z poziomów modelu warstwowego systemu komputerowego. Dla infrastruktury stosuje się specjalistyczne urządzenia umożliwiające pomiar zadanych parametrów, np. temperatury procesora lub napięcia w sieci elektrycznej, do której podłączone są zasilacze awaryjne (UPS). Dzięki takim rozwiązaniom możliwa jest obsługa sytuacji wyjątkowych w dwóch trybach: na poziomie warstwy, gdzie odpowiednie działania są podejmowane bezpośrednio przez uszkodzone/zagrożone urządzenie np. wyłączenie przegrzewającego się procesora lub jest propagowane do warstwy wyższej, np. poinformowanie systemu operacyjnego o braku zasilania w sieci w celu zamknięcia systemu lub przejścia w tryb oszczędzania energii.

W przypadku warstwy pośredniej informacje o sytuacjach wyjątkowych mogą pochodzić z warstw niższych, lub jako wynik działania oprogramowania diagnostycznego, np. sprawdzającego połączenie sieciowe, obecność wirusów w nowo-zainstalowanym dysku, lub mierzącego zużywaną moc obliczeniową przez wątki danego procesu. Podobnie jak w przypadku warstwy infrastruktury, zdarzenie wyjątkowe może być obsługiwane w warstwie, w której wystąpiło np. poprzez poddanie podejrzanego pliku kwarantannie, bądź przekazane do warstwy wyższej. W systemach współbieżnych i rozproszonych typowymi metodami przekazywania komunikatów o powstałej sytuacji wyjątkowej z warstwy niższej do wyższej są:

- Sygnały – kiedy normalny przebieg przetwarzania jest przerywany, a informacja

o powstałym problemie jest obsługiwana przez specjalną procedurę (ang. handler), typowym systemem, w którym stosuje się powyższe rozwiązanie jest system UNIX [posix].

- Wyjątki – specjalne obiekty zwracane zamiast oczekiwanej odpowiedzi, np. dla protokołu SOAP [soap], stosowanego w architekturze orientowanej na usługi.
- Kody błędów – kiedy wykonywana funkcja/usługa kończy się bez przerywania normalnego toku przetwarzania, a jedynie jej wynik wskazuje iż wystąpił błąd informując o jego typie, są stosowane np. w protokole HTTP [fielding99].

Monitorowanie obliczeń w warstwie aplikacji użytkowych jest zwykle związane z bezpośrednią interakcją z użytkownikiem, dlatego często są w niej wykorzystywane odpowiednie elementu interfejsu użytkownika, np. wykresy obrazujące wykorzystanie zasobów. Należy podkreślić, że w tej warstwie następuje końcowa obsługa wykrytych sytuacji wyjątkowych – użytkownik jest ostatnim elementem w łańcuchu ich propagacji. Sama aplikacja może zdecydować, czy monitorowany problem ma zostać przedstawiony użytkownikowi, np. poprzez odpowiedni komunikat przerywający pracę, czy może powinno zostać uruchomione dodatkowe oprogramowanie systemowe, np. systemowy kreator rozwiązania problemów lub program antywirusowy.

Wszystkie podstawowe charakterystyki jakościowe systemów komputerowych mają swoje odzwierciedlenie w obliczeniach:

- elastyczność, która oznacza możliwość rozbudowy i konfiguracji systemu,
- funkcjonalność, która podaje co właściwie i w jakim zakresie jest wykonywane podczas obliczeń,
- użyteczność, która definiuje grupę cech związanych z poziomem dostępności funkcji systemu dla użytkowników,
- wydajność, która określa cechy związane z szybkością obliczeń oraz
- wiarygodność, która wyznacza jak akceptowalne są te obliczenia: dokładne, rzetelne, odpowiadające rzeczywistości, kompletne.

Z punktu widzenia niniejszej rozprawy kluczowymi charakterystykami są wydajność obliczeń wyrażana przez takie cechy jak złożoność, efektywność, skalowalność (patrz rozdz. 2.8) oraz często jej przeciwstawna wiarygodność określana przez takie cechy jak: dokładność, aktualność czy rzetelność oferowanych wyników.

2.6 Systemy czasu rzeczywistego

Przez system czasu rzeczywistego rozumiemy system komputerowy, w którym zadane obliczenia muszą być wykonane zgodnie z wcześniej zdefiniowanymi ograniczeniami czasowymi. A co za tym idzie w tego typu systemach, oprócz poprawności wykonania zadanej funkcjonalności liczy się także czas w jakim otrzymywane są wyniki [butazzo05].

Wyróżniamy dwa rodzaje systemów czasu rzeczywistego [leontyev10]:

- twarde (ang. hard) systemy czasu rzeczywistego – kiedy zdefiniowane ograniczenia czasowa muszą być zawsze zachowane oraz
- miękkie (ang. soft) systemy czasu rzeczywistego – kiedy zdefiniowane ograniczenia czasowe mogą być czasami niezachowane.

Ze względu na swoją specyfikę, dla systemu czasu rzeczywistego przyjmuje się następujące założenia jeśli chodzi o wykonywane zadania [liu73]:

Z1. Zadanie czasu rzeczywistego jest to ciąg prac (ang. jobs) wykonywanych okresowo ze stała częstotliwością żądań ich wykonania.

Z2. Maksymalny czas obsługi (ang. deadline) danej pracy jest zawsze krótszy lub równy okresowi żądań zgłaszanych przez zadanie czasu rzeczywistego.

Z3. Zadania czasu rzeczywistego są niezależne, tzn. obliczenia w jednym zadaniu są niezależne od wcześniejszego wykonania obliczeń innych zadań. Założenie to może być złagodzone w pewnych specyficznych warunkach [liu73].

Z4. Czas wykonywania obliczeń dla kolejnych prac danego zadania czasu rzeczywistego jest stały i krótszy od maksymalnego czasu obsługi (ang. deadline).

Z5. W systemie mogą występować także zadania nieokresowe – nie spełniające założenia Z1, jednak nie mogą one wymagać spełnienia ograniczeń czasu rzeczywistego.

Ponadto należy zauważyć niejawne założenie występujące w wielu pracach nad równoległymi systemami czasu rzeczywistego (np. w [qi11]), gdzie pomimo iż system umożliwia wykonywanie wielu zadań to jednocześnie:

Z6. Zadania czasu rzeczywistego są wykonywane w sposób sekwencyjny spełniając założenie Z2.

Zgodnie z [vidyarthi10] jako alokację zadań w systemie czasu rzeczywistego rozumiemy przydział elementów realizujących zadane zadanie do węzłów obliczeniowych systemu komputerowego, jest ona też nazywana szeregowaniem (ang. scheduling), por. rozdz. 2.5 . W systemach czasu rzeczywistego wyróżniamy dwa rodzaje alokacji ze względu na priorytety [liu73]:

1. statyczne – kiedy priorytet jest nadawany na stałe na początku wykonywania zadania, przy jednoczesnym założeniu, że zadanie o wyższym priorytecie ma zawsze pierwszeństwo przez zdaniem, który ma niższy priorytet, przykładem algorytmu alokacji opartego na statycznych priorytetach jest RMA (ang. rate-monotonic prioryty assignment), gdzie priorytet zależy od częstotliwości wykonywanych obliczeń prze dane zadanie [liu73],
2. dynamiczne – kiedy priorytet zadania może się zmieniać w czasie jego wykonywania, lub wyższy priorytet nie zawsze gwarantuje pierwszeństwo w dostępie do zasobów, np. EDF (ang. earliest deadline first), gdzie zasoby są alokowane dla zadania, którego termin wykonania jest najbliższy,
3. hybrydowe – kiedy niektóre zadania mają priorytet przydzielony statycznie i jest on zawsze respektowany przez algorytm alokacji, a inne mogą mieć dynamiczne – zmieniające się w zależności od warunków wykonania.

W równoległych systemach czasu rzeczywistego możemy wyróżnić następujące algorytmy alokacji [leontyev03]:

1. dzielone, kiedy każdy procesor posiada swoją osobną kolejkę zadań żądających wykonania kolejnych periodycznych obliczeń i
2. globalne, kiedy obliczenia danego zadania mogą być wykonywane na różnych procesorach.

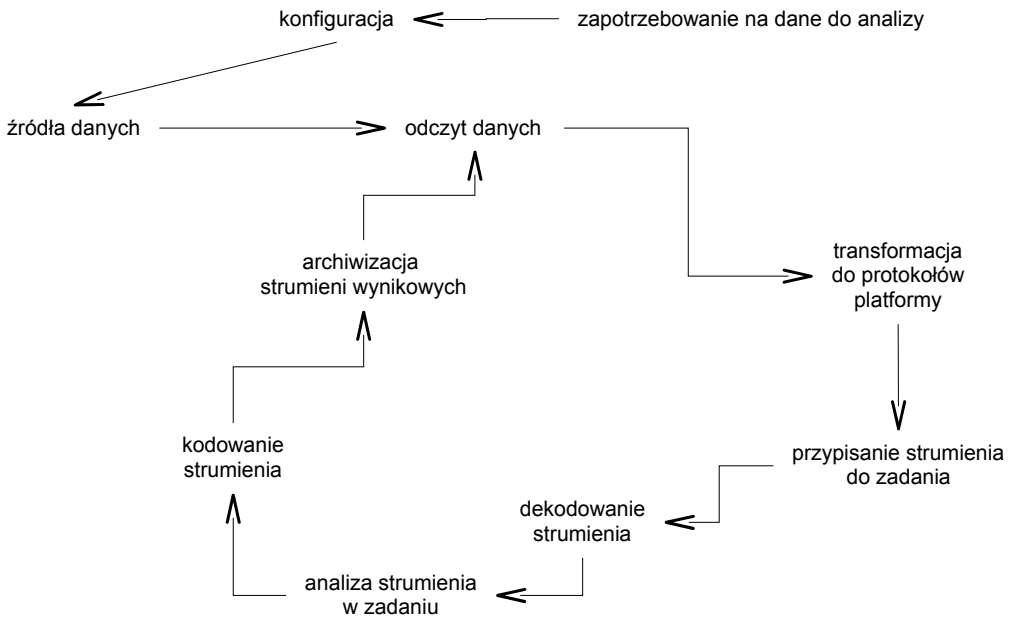
Jak można zauważyć, dla systemów czasu rzeczywistego wszystkie cztery domeny powinny być zarządzane w specyficzny sposób: dane wejściowe i wyjściowe muszą być przesyłane na czas, zaś zarządzanie komunikacją musi gwarantować odpowiedni poziom jakości dostarczenia wiadomości, natomiast obliczenia przeprowadzane na danych muszą mieć zarezerwowaną odpowiednią wielkość zasobów aby zdążyć z wykonaniem przed wymaganym terminem zakończenia.

2.7 Przetwarzanie strumieni multimedialnych

Przez strumień multimedialny rozumiemy ciąg **elementów strumienia:** klatek obrazu dla strumieni wideo albo próbek dźwięku dla strumień audio. Strumienie wideo umożliwiają przedstawienie ruchomych obrazów zarejestrowanych kamerą i mogą się różnić między sobą częstotliwością rejestracji obrazów oraz ich rozdzielczością. Podobnie strumieni audio mogą mieć różne częstotliwości próbkowania jak również liczbą poziomów kwantyzacji próbkowania.

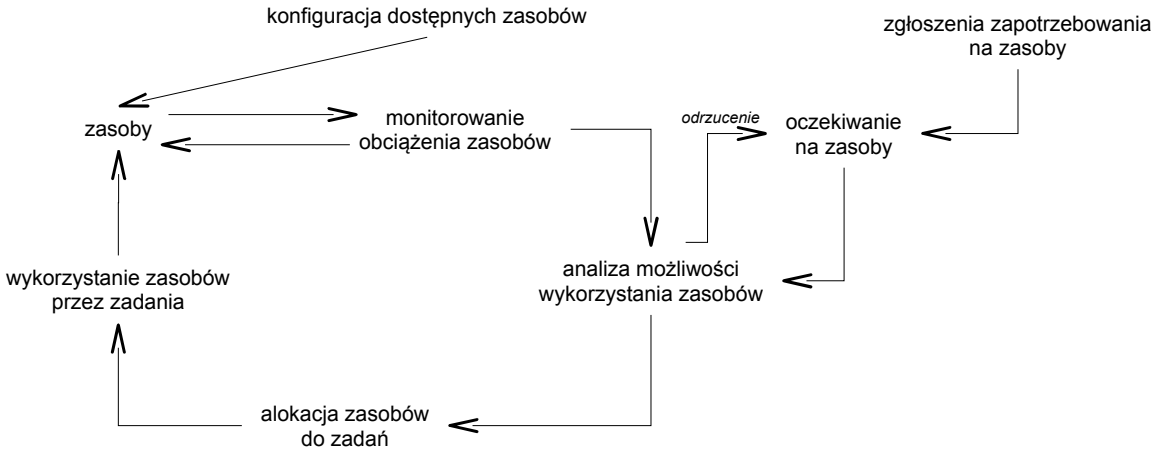
Jest wiele zastosowań systemów automatycznego przetwarzania strumieni audio/wideo. Wśród głównych celów ich budowy można wymieć następujące [bovik09]:

- Kodowanie i dekodowanie (ang. encoding and decoding) – umożliwia przetworzenie obrazów i dźwięków do postaci spakowanych i z powrotem do oryginalnej, dzięki temu zajmują mniej miejsca na dysku i wykorzystują mniejsze pasmo sieciowe podczas transmisji.
- Odnawianie (ang. restoration) – umożliwia odtworzenie oryginalnego stanu filmu lub ścieżki dźwiękowej na podstawie uszkodzonej kopii, jest to szczególnie istotne w przypadku przechowywania danych w formie analogowej i późniejszej ich digitalizacji.
- Wzbogacanie (ang. ehancement) – umożliwia dodanie pewnych elementów obrazu i dźwięku do istniejących nagrań, typowym przykładem jest wprowadzanie kolorów do czarno-białych filmów.
- Stabilizacja obrazów (ang video stabilization) – umożliwia rozpoznanie stałych części w ruchomych scenach wideo, w tym eliminacja wpływu trzęsienia i obrotów kamerą, jak również odnajdowanie maski tła i poruszających się obiektów.
- Składanie obrazów (ang. video mosaicing) – umożliwia tworzenie pojedynczego obrazu statycznego ze złożenia wielu obrazów ze strumienia wideo, dzięki czemu obejmuje on większą powierzchnię i może posiadać lepsze charakterystyki jakościowe np. wyższą rozdzielczość w porównaniu z oryginałem wideo.
- Streszczanie wideo (ang. video summarization) – umożliwia tworzenie streszczenia filmu, automatyczne indeksowanie oraz na tej podstawie odtwarzanie wybranych fragmentów strumienia.
- Obserwacja wideo (ang. video surveillance) – umożliwia wykrycie i śledzenie niebezpiecznych obiektów, osób i całych sytuacji. Jest to szczególnie ważne dla ochrony takich obiektów jak lotniska, stacje kolejowe/metra czy zabezpieczenie ruchu ulicznego.
- Rozpoznawanie twarzy (ang. face recognition) – jest związane z obserwacją wideo i umożliwia detekcję twarzy na obrazie wideo, a następnie przypisanie do niej konkretnych danych personalnych.
- Automatyczne rozpoznawanie mowy (ASR, ang. automatic speech recongnition) – umożliwia automatyczną konwersję zarejestrowanych słów na tekst pisany. Może ono być wspomagane poprzez równoczesną analizę mimiki twarzy osoby mówiącej.



Rysunek 5: Podstawowa procedura zarządzania danymi dla przetwarzania strumieni multimedialnych w czasie rzeczywisym.

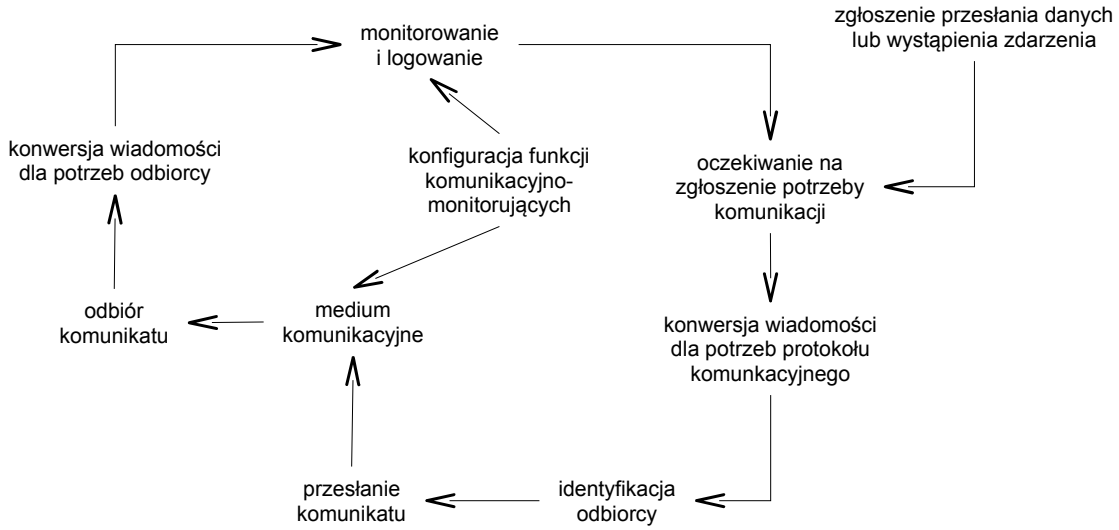
Systemy komputerowe rozwiązujące powyżej przedstawione problemy są trudne do realizacji, a obecnie stosowane algorytmy są niedoskonałe i wymagają dużej mocy obliczeniowej. Dodatkowym, często wykorzystywanym wymaganiem jest ich wykonywanie w czasie rzeczywistym, kiedy to płynący strumień multimedialny musi być na bieżąco analizowany i modyfikowany. Uwzględniając powyższe założenia można przedstawić podstawowe procedury zarządzania poszczególnymi domenami dotyczącymi przetwarzania strumieni multimedialnych w czasie rzeczywistym. Takie zadanie powstało przy realizacji platformy KASKADA (patrz rozdz. 3). Odpowiednio skonfigurowane źródła **danych** rejestrują strumienie audio/wideo (patrz rys. 5), które następnie są odczytywane przez realizującą przetwarzanie platformę (system komputerowy). Następnie ich format musi być dostosowany do wykorzystywanych protokołów przesyłania danych, a cały strumień przypisany do odpowiadającego mu zadania, po czym jest on dekodowany, analizowany i modyfikowany przez zadanie (obliczenia) a następnie ponownie kodowany i archiwizowany. Tak zarejestrowany strumień może być ponownie odczytany do wykorzystania



Rysunek 6: Podstawowa procedura zarządzania zasobami dla przetwarzania strumieni multimedialnych w czasie rzeczywisym.

przez obliczenia off-line (już bez wymogów czasu rzeczywistego).

Przez **zasoby** rozumiemy sprzęt komputerowy, np. węzły klastra obliczeniowego służące do wykonywania obliczeń na strumieniach multimedialnych. Na rys. 6 przedstawiono typową procedurę zarządzania zasobami obliczeniowymi. Odpowiednio skonfigurowane zasoby są ciągle monitorowane ze względu na ich obciążenie. Z drugiej strony system oczekuje na zgłoszenia zapotrzebowania na zasoby i w przypadku odbioru takiego żądania, na podstawie aktualnego stanu wykorzystania zasobów analizuje możliwość dostarczenia odpowiednich zasobów. Jeśli jest to możliwe alokuje zasoby dla zadań (obliczeń). Po wykonaniu obliczeń, wykorzystane zasoby są zwracane do systemu, gdzie mogą być ponownie przydzielane.



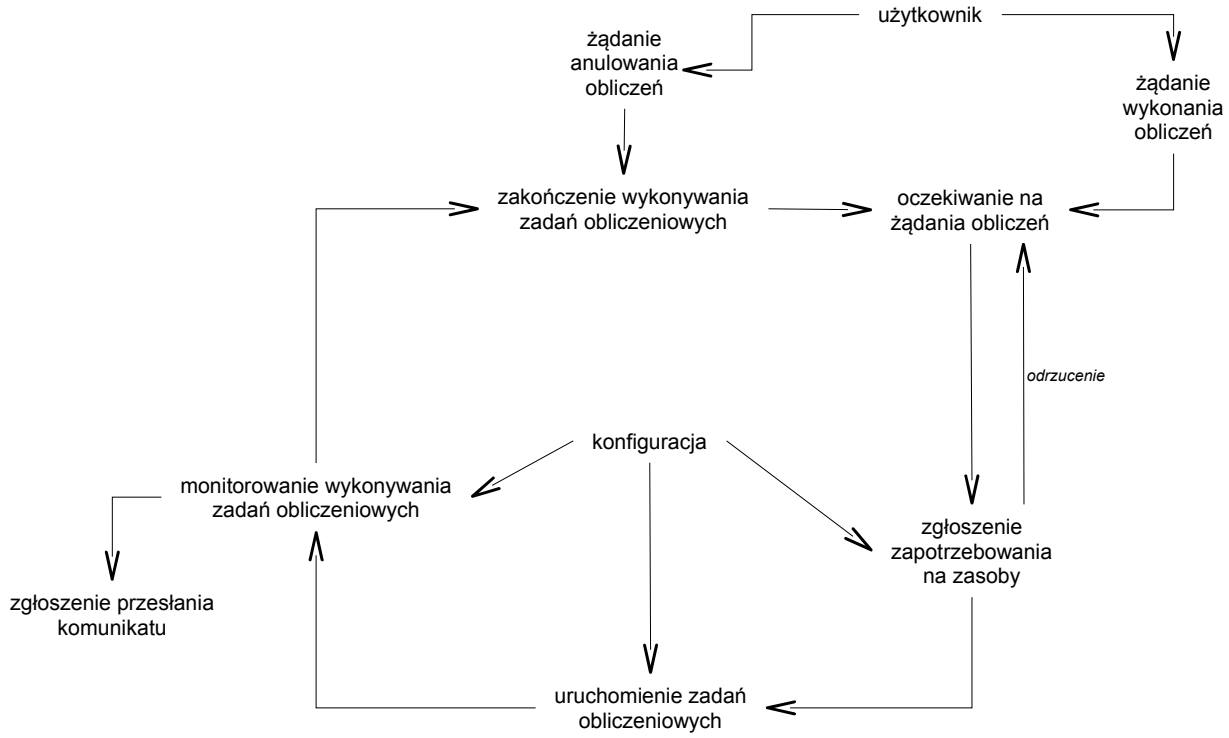
Rysunek 7: Podstawowa procedura zarządzania komunikacją dla przetwarzania strumieni multimedialnych w czasie rzeczywisym.

Typowym sposobem wymiany informacji pomiędzy komponentami systemu rozproszonego, jakimi są systemy przetwarzające strumienie multimedialne w czasie rzeczywistym jest **komunikacja** poprzez sieć komputerową. Na rys. 7 przedstawiona została typowa procedura zarządzania komunikacją w takim systemie. System po odpowiednim skonfigurowaniu, oczekuje na zgłoszenia potrzeby komunikacji od zadań obliczeniowych, które przekazują między sobą odpowiednie komunikaty zawierające dane ze strumieni lub informacjami o wystąpieniu danego zdarzenia. Następnie następuje konwersja danych dla potrzeb wykorzystywanego protokołu komunikacyjnego oraz identyfikacja odbiorcy, do którego przesyłany jest dany komunikat. Na podstawie konfiguracji systemu wybierane jest odpowiednie łącze komunikacyjne, z którego po stronie odbiorcy następuje odbiór komunikatu, po którym następuje konwersja danych, tak aby były one dostosowane do potrzeb odbiorcy i w zależności od konfiguracji logowanie i monitorowanie, którego działanie może wpłynąć na wysyłanie kolejnego komunikatu (np. ograniczenie ze względu na zbyt duże wykorzystanie pasma sieciowego).

Zakładamy, że w systemie przetwarzania strumieni multimedialnych czasu rzeczywistego **obliczenia** będą realizowane przez zadania obliczeniowe. Na rys. 8 przedstawiono procedurę zarządzania obliczeniami, gdzie system komputerowy oczekuje na żądania wykonania obliczeń i po otrzymaniu takiego zgłoszenia od użytkownika, na podstawie zadanej konfiguracji wykonuje analizę wymagań na zasoby. Następnie jeśli odpowiednie zasoby są dostępne uruchamia zadania obliczeniowe i je realizuje, jednocześnie monitorując ich przebieg, aż do momentu zakończenia obliczeń. Wówczas zajęte zasoby są zwalniane i wykorzystane do obsługi kolejnych żądań.

Należy podkreślić, że poszczególne procedury, opisane nawet na tak wysokim poziomie ogólności są równolegle realizowane i przenikają się wzajemnie. To implikuje, że obliczenia na strumieniach

multimedialnych w czasie rzeczywistym muszą wykorzystywać **zintegrowany system zarządzania** opisanymi domenami, co zostało zrealizowane na platformie KASKADA i opisane w rozdz. 3 niniejszej rozprawy.



Rysunek 8: Podstawowa procedura zarządzania obliczeniami dla przetwarzania strumieni multimedialnych w czasie rzeczywisym.

2.8 Pojęcie skalowalności

Cechą zwykle wykorzystywaną podczas oceny systemu równoległego w dłuższym horyzoncie czasowym jest skalowalność (por. rozdz. 2.5). Intuicyjnie skalowalność rozumiemy jako zdolność systemu do zwiększenia wydajności wraz ze wzrostem ilości zasobów jakimi dysponuje. Formalnie, skalowalność jest wyrażana przy użyciu pojęcia przyspieszenia (ang. speedup), które jest opisane poniższą funkcją [kuck96]:

$$\xi(p,n)=\frac{\tau(1,n)}{\tau(p,n)} \quad (1)$$

Gdzie p jest liczbą zainstalowanych jednostek obliczeniowych (np. procesorów, rdzeni, węzłów obliczeniowych), a n jest rozmiarem danych wejściowych. Skalowalność jest oceniana dla danego środowiska, w którym wykonywane są obliczenia, w tym dla wybranej strategii zarządzania i konkretnej aplikacji, usługi lub zadania obliczeniowego. $\tau(p, n)$ jest zmierzonym czasem wykonania obliczeń dla danej liczby jednostek obliczeniowych i rozmiaru danych. W przypadku *stałego rozmiaru danych n*, przyspieszenie dla systemu idealnie skalowalnego jest równe: $\xi(p)=p$, natomiast dla systemu wysoko skalowalnego (ang. high-performance scalability) $p/2<\xi(p)<p$. Podobnie dla stałej *liczby jednostek obliczeniowych p* i zmiennego rozmiaru danych wejściowych n, system jest wysoko skalowalny kiedy $p/2<\xi(n)<p$. W [kuck96] zdefiniowano kolejne przedziały, dalszych poziomów skalowalności: średnio wysoka, średnia, progowa i nieakceptowalna.

W dalszych rozważaniach skupiamy się na systemach rozproszonych przetwarzających strumienie multimedialne w czasie rzeczywistym, na które wymagania użytkowe dopuszczają możliwość występowania pewnych opóźnień, a nawet pewien poziom straty danych [krawczyk11] – kiedy to na skutek zbyt dużego obciążenia część danych nie zostaje przetworzona i przesłana jako strumień

wyjściowy. Z drugiej strony, ze względu na wymóg przetwarzania w czasie rzeczywistym, wyniki muszą być dostarczane jak najszybciej i całkowity czas przetwarzania nie może znacznie odbiegać od czasu transmisji strumienia wejściowego. Dlatego definicja z równania (1) nie może być zastosowane w sposób bezpośredni – w takim przypadku przyspieszenie byłoby zawsze równe w przybliżeniu: 1 ($\xi(p, n) \approx 1$).

Skalowalność systemu czasu rzeczywistego była rozważana w [branden08], gdzie pięć różnych algorytmów alokacji zasobów było zaimplementowane w środowisku LITMUS (ang. LInux Testbed for MUltiprocessor Scheduling in Real-Time systems) i wykorzystane do testów platformy sprzętowej Sun Niagara. Analiza skalowalności została przeprowadzona dla stałej liczby logicznych rdzeni procesora równej 32 w zależności od liczby uruchomionych zadań. Wyniki były przedstawione za pomocą wskaźnika planowalności (ang. schedulability) określającego stosunek liczby działań wykonanych na czas do wszystkich wykonanych działań, w porównaniu ze współczynnikiem użytkowania (ang. utilization cap) – mierzonego jako średnie obciążenie obliczeniowe testowanego procesora.

Inny przykład oceny skalowalności został przedstawiony w [gorlatch09]. Opisuje on „cyber-infrastrukturę" (ang. cyberinfrastructure) czasu rzeczywistego, składającą się zarówno ze sprzętu jak i odpowiedniego oprogramowania służących do uruchamiania aplikacji ROIA (ang. Real-time Online Interactive Application) głównie gier MMOG (ang. Massively Multiplayer Online Game). W przytaczanym artykule, założono że platforma jest skalowalna, gdy może zapewnić warunki wykonania w czasie rzeczywistym aplikacji ROIA wykorzystujących mechanizmy zrównoleglenia i rozproszenia obliczeń. Jest to możliwe dla rosnącej liczby jednocześnie obsługiwanych użytkowników, poprzez zwiększenie liczby serwerów wykonujących ich żądania. Wyniki badań skalowalności zostały zaprezentowane jako średni czas użycia procesorów dla danej liczby serwerów w zależności od liczby współbieżnie obsługiwanych użytkowników.

Yu Tang w [yutang10] zaprezentował algorytm PBDS (ang. Pull-Based Distributed Scheduling) dokonujący alokacji zasobów obliczeniowych dla komputera klastrowego wykonującego zadania czasu rzeczywistego. Opisany test skalowalności został wykonany dla zwiększającej się liczby wykorzystywanych węzłów obliczeniowych oraz dla niskiego i wysokiego poziomu obciążenia wejściowego sieci, wyrażonego w Kb/s. Wyniki zostały badań skalowalności przedstawiono jako średni czas odpowiedzi systemu (w sekundach) oraz jako całkowitą przepustowość systemu w Kb/s w zależności od liczby węzłów w klastrze oraz dla poszczególnych poziomów obciążenia wejściowego.

Ocena skalowalności platformy ALIFE, wykonującej syntezę obrazów (ang. rendering) w czasie rzeczywistym została przedstawiona w [rogmans08]. Głównym celem platformy jest optymalizacja wydajności tworzenia obrazów używanych w aplikacjach 3DTV używających monitorów autostereoskopowych. Zaproponowane rozwiązanie oparte jest na wykorzystaniu specjalnych kart graficznych GPGPU (ang. General-Purpose Graphics Processing Units) w konfiguracji potokowej (ang. pipeline). Przyspieszenie jest wyrażone jako liczba wygenerowanych obrazów w czasie (fps) w zależności od rosnącej liczny jednostek wykorzystywanych w potoku (jednostki obliczeniowe).

Wspólną cechą opisywanych powyżej metod oceny skalowalności jest wykorzystanie wybranej charakterystyki wydajnościowej, np. czasu odpowiedzi, liczby generowanych obrazów w czasie czy współczynnika określającego procent wykonania zadań we właściwym czasie jako substytut przyspieszenia ξ. Następnie, podobnie jak w definicji [kuck96], taką charakterystykę przedstawiono jako funkcję liczby jednostek obliczeniowych (np. węzłów) lub rozmiaru problemu (współczynnik użytkowania lub rozmiar danych wejściowych podany w definicji [kuck96]). Skalowalność, sensu stricto nie była bezpośrednio oceniania, z wyjątkiem ogólnych stwierdzeń typu „skalowalność jest prawie liniowa" w [gorlatch09].

Bardziej ogólny model określania skalowalności już na etapie analizy wymagań dla systemu

rozproszonego zaproponowała L. Duboc w [duboc10, duboc08]. Określiła ona skalowalność jako zdolność do utrzymania satysfakcjonujących wyników działania w sensie jakościowym, na poziomie, który zadowala jego użytkowników (udziałowców, ang. stakeholders), kiedy charakterystyki związane z wymaganiami oraz projektem systemu są lepsze niż zakładano podczas jego konstrukcji. Tak bardzo ogólna definicja została z sukcesem wykorzystana i wdrożona w platformie IEF i opisana w [duboc12].

Bardziej precyzyjną definicję, zaproponował Jogalekar w [jogalekar00]. Przedstawił on metrykę skalowalności opartą na produktywności $\iota(p)$ (ang. productivity):

$$\iota(p)=\frac{\lambda(p)f(p)}{\kappa(p)} \quad (2)$$

Gdzie p jest liczbą zainstalowanych jednostek obliczeniowych (np. procesorów, rdzeni, węzłów obliczeniowych), $\lambda(p)$ określa uzyskaną przepustowość (np. liczbę przetwarzanych transakcji/sek), $f(p)$ jest funkcją jakości określającą jak dobrze przetwarzanie (np. odpowiedni średni czas odpowiedzi lub akceptowalny poziom straty danych) jest wykonywane, a $\kappa(p)$ określa koszt (np. liczba wykorzystanych jednostek obliczeniowych). Sama metryka skalowalności definiowana jest następującym wzorem:

$$\varepsilon(p)=\frac{\iota(p)}{\iota(1)}=\frac{\lambda(p)f(p)\kappa(1)}{\lambda(1)f(1)\kappa(p)} \quad (3)$$

Wykorzystując powyższą metrykę Jogaleker zakłada, że system jest skalowalny jeśli wraz ze wzrostem jego skali (zmienna p) iloraz jego przepustowości $\lambda(p)$ i zadanej funkcji jakości $f(p)$ „dotrzymuje kroku" kosztom zwiększenia skali $\kappa(p)$, w przypadku idealnym będzie stały i równa 1, a w rzeczywistości będzie niezmienna w zadanym przedziale p. W dalszej części rozprawy wykorzystujemy powyższą metrykę do porównywania skalowalności systemu przetwarzania strumieni multimedialnych w czasie rzeczywistym dla różnych algorytmów alokacji.

2.9 Tezy rozprawy

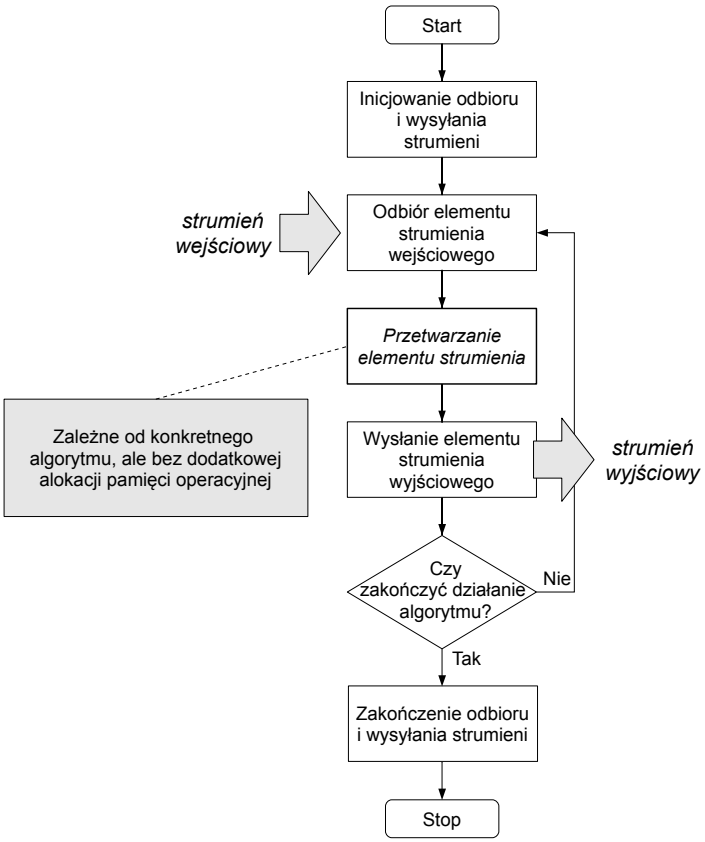
Celem rozprawy jest przedstawienie i weryfikacja przyjętej koncepcji zarządzania przetwarzaniem strumieni multimedialnych na klastrze obliczeniowym, przy zachowaniu miękkich (ang. soft) założeń przetwarzania czasu rzeczywistego. W szczególności sprowadza się to do przedstawienia algorytmów alokacji zasobów obliczeniowych dla zadań przetwarzających strumienie multimedialne oraz ich wdrożenia na platformie KASKADA.

W dalszej części rozprawy przyjmując procedury zarządzania z rozdz. 2.7 będziemy badali zachowanie wybranej **klasy algorytmów** przetwarzania strumieni multimedialnych. Klasa ta charakteryzuje się wspólnym dla wszystkich algorytmów przebiegiem działania przedstawionym na rys. 9. Przetwarzanie kolejnych elementów strumienia wejściowego (klatki obrazu lub grupy próbek audio) odbywa się w pętli, która jest wykonywana jednokrotnie dla każdego odebranego elementu. W pierwszym kroku pętli następuje odczyt elementu strumienia, alokacja pamięci operacyjnej do jego przechowywania i odkodowania. W kolejnym kroku wykonywana jest analiza i modyfikacja elementu strumienia, zakładamy przy tym, że w tym czasie **nie następuje kolejna alokacja pamięci,** a jedynie realizowane są obliczenia. W trzecim kroku pętli następuje zakodowanie i wysłanie zmodyfikowanego elementu strumienia, jak również zwolnienie pamięci. W następnym kroku następuje decyzja o kontynuacji pętli: kiedy istnieją kolejne elementy w strumieniu wejściowym do przetwarzania lub gdy nie minął założony czas przetwarzania. W ostatnim kroku następuje zakończenie odbioru i wysyłania elementów strumieni wejściowych i wyjściowych. Tak więc każde zadanie obliczeniowe analizuje strumień multimedialny według schematu z rys. 9. Przykłady zastosowań takich zadań zawarto w rozdz. 2.7 .

Należy podkreślić, że tak rozumiane zadania czasu rzeczywistego wykonywane przez platformę

KASKADA nie spełniają większości założeń z rozdz. 2.6 . W szczególności Z2 nie jest spełnione, przejściowo obliczenia mogą być wykonywane dłużej niż okres przychodzących żądań, np. wykonywanie obliczeń tylko dla co 10 klatki obrazu. W takim przypadku platforma powinna buforować kolejne zgłoszenia. Z3 nie jest spełnione, gdyż zadania są wykonywane potokowo, a wynik obliczeń jednego zadania jest wykorzystywany przez inne zadanie. Z4 nie jest spełnione, podobnie jak dla Z2, bowiem niektóre żądania mogą wymagać dłuższego czasu wykonania. Z6 również nie jest spełnione gdyż platforma umożliwia wykorzystywanie wielowątkowości na poziomie zadania obliczeniowego. To świadczy o oryginalności zaprezentowanego podejścia.

Wiarygodność przetwarzania (rozumiana jako akceptowalna strata przetwarzanych danych) zależy od liczby jednocześnie analizowanych strumieni multimedialnych oraz od wielkości obciążenia obliczeniowego tego węzła wynikającego z wykonania na tym węźle tego typu analizy dla pojedynczego strumienia.



Rysunek 9: Podstawowy schemat działania badanej klasy algorytmów.

W celu dokonania weryfikacji i oceny zaproponowanego algorytmu alokacji zadań dotyczących analizy strumieni wykonano szereg eksperymentów, które pozwoliły wykazać następujące tezy:

1. Przy akceptowanej wiarygodności przetwarzania strumieni, obciążenie węzła dla analizowanej klasy algorytmów i dla danego typu przetwarzanego strumienia wzrasta nieliniowo wraz z liczbą przetwarzanych strumieni, przy czym wzrost ten można oszacować tzw. funkcją korekty podającą przyrost obciążenia w stosunku do jego wzrostu liniowego.
2. Dla zadanej klasy algorytmów analizy strumieni oraz dla zaproponowanego algorytmu alokacji zadań na węzły klastra (wykorzystującego funkcję korekty), skalowalność klastra określona wzorem (3) utrzymuje się na stałym poziomie przy wzroście liczby jednocześnie przetwarzanych zadań i strumieni, pod warunkiem proporcjonalnego do niego wzrostu liczby węzłów obliczeniowych zaangażowanych w to przetwarzanie.

Powyższe tezy są rozpatrywane w rozdziałach 4 i 5 po przedstawieniu wykorzystywanego środowiska obliczeń zrealizowanego w ramach projektu MAYDAY EURO 2012*.

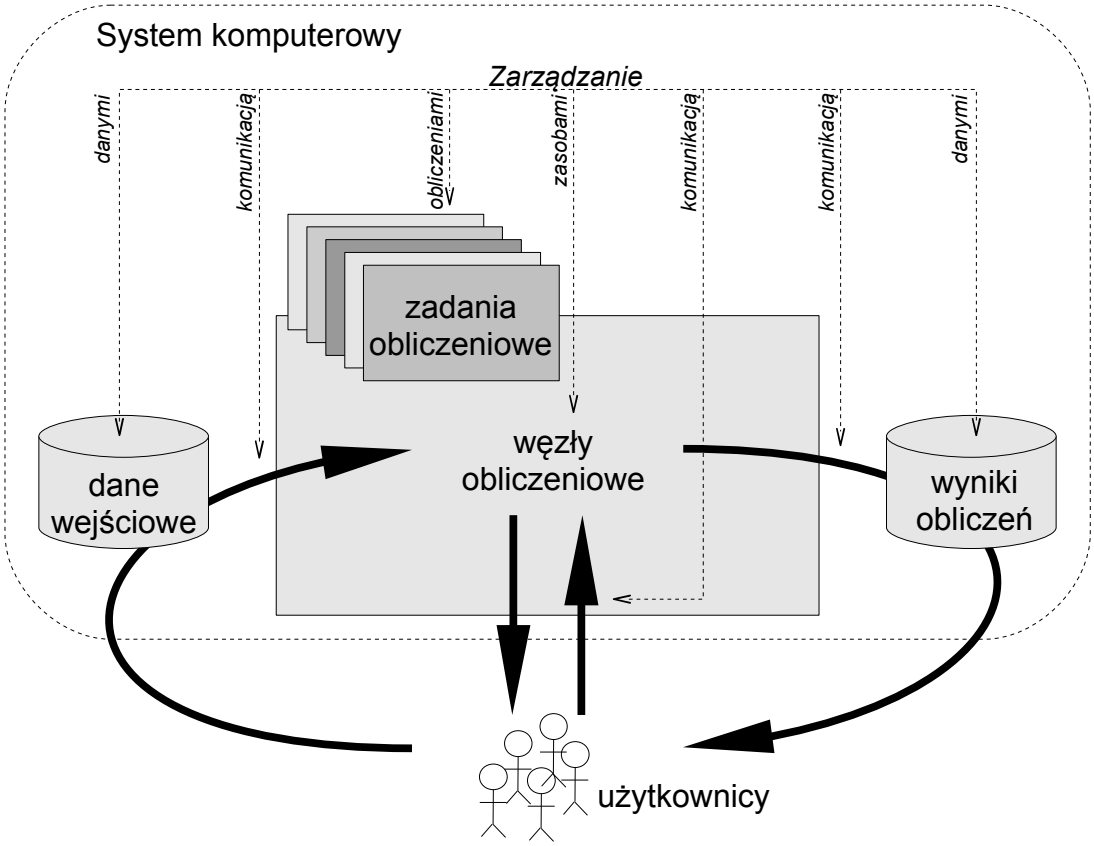
* Politechnika Gdańska, „MAYDAY EURO 2012" Superkomputerowa platforma kontekstowej analizy strumieni danych multimedialnych do identyfikacji wyspecyfikowanych obiektów lub niebezpiecznych zdarzeń. Projekt współfinansowany z Europejskiego Funduszu Rozwoju Regionalnego i Budżetu Państwa w ramach Programu Operacyjnego Innowacyjna Gospodarka.

3 Zintegrowane zarządzanie platformą KASKADA

Zaprezentowano zintegrowany model zarządzania systemem komputerowym, jak również opisano koncepcję jego implementacji w platformie KASDADA. Przedstawiono architekturę platformy i zrealizowane mechanizmy zarządzania przetwarzaniem zadań multimedialnych. Przeanalizowano aplikację detekcji twarzy w strumieniu wideo w celu zilustrowania podstawowych problemów równoleglizacji oraz optymalnego wykorzystania zasobów. Opisano konfigurację środowiska przetwarzania jak również dokonano jego porównania z innymi istniejącymi rozproszonymi systemami obliczeniowymi. Zaproponowane środowisko będzie przedmiotem szczegółowych badań w następnych rozdziałach rozprawy doktorskiej.

3.1 Koncepcja zintegrowanego zarządzania

Opisane w poprzednim rozdziale metody zarządzania obliczeniami zakładają, że uruchamiane zadania oprócz najważniejszych funkcji obliczeniowych, same zarządzają danymi wejściowymi, decydują o sposobie komunikacji, jak również odpowiadają za dostarczanie wyników do odbiorcy. Wadą takiego założenia jest brak możliwości globalnego zarządzania danymi i komunikacją przez co system komputerowy, w przypadku przetwarzania on-line, może dokonać nieoptymalnego, lub nawet niedopuszczalnego z punktu widzenia wymagań użytkownika przydziału zasobów. Typowym przykładem może być wzajemne zagłodzenie grupy zadań obliczeniowych, wymagających częstego i szybkiego dostępu do danych na dysku twardym, w przypadku ich alokacji na tym samym węźle obliczeniowym.



Rysunek 10: Idea integracji zarządzania danymi, komunikacją, obliczeniami i zasobami.

W niniejszej rozprawie zaproponowaliśmy rozszerzone podejście do zarządzania zasobami z punktu widzenia obliczeń, danych oraz komunikacji (patrz rys. 10). Oprócz klasycznego przydziału

węzłów obliczeniowych kolejno wykonywanym zadaniom, uwzględniona jest specyfika danych ze szczególnym uwzględnieniem komunikacji międzyzadaniowej. Tego typu rozwiązanie zostało zaimplementowane na platformie programistycznej umożliwiającej tworzenie, uruchamianie i wykonywanie algorytmów przetwarzania strumieni multimedialnych o nazwie KASKADA. Platforma KASKADA (Kontekstowa Analiza Strumieni Danych z Kamer Dla Aplikacji Definiujących Alarmy) [krawczyk10a] została wdrożona na klastrze Galera znajdującym się w Centrum Informatycznym TASK w ramach projektu MAYDAY EURO 2012*.

W przyjętym rozwiązaniu podstawowym elementem wykonującym **obliczenia** pozostaje **zadanie obliczeniowe**. Jest to zaimplementowany i uruchomiony kod reprezentujący odpowiedni algorytm analizy strumienia, realizowany za pomocą jednego lub więcej wątków uruchomianych na jednym węźle obliczeniowym i wykorzystujących pamięć współdzieloną. Współpraca między zadaniami uruchomionymi w ramach usługi złożonej grupującej zadania, odbywa się na poziomie wymiany komunikatów. Dostępne usługi kreują funkcjonalność platformy, z których komponowane mogą być różnego typu aplikacje bezpośrednio przez użytkowników bądź deweloperów.

Zasobami wykorzystywanymi przez platformę są węzły obliczeniowe komputera klastrowego. Wszystkie wątki pojedynczego zadnia muszą być umieszczone na tym samym węźle. Każdy węzeł posiada określoną wielkość zasobów: liczbę rdzeni obliczeniowych, wielkość pamięci RAM oraz różne typy urządzeń wejścia-wyjścia: twardy dysk, karty sieciowe itp.

Platforma KASKADA zorientowana jest na optymalizację wykorzystania **danych** multimedialnych. Zadanie przetwarzające strumień audio lub wideo otrzymuje ciąg próbek dźwięku lub klatek obrazu i analizuje je zgodnie ze swoim algorytmem. Przy czym może być wykorzystywane do przetwarzania **on-line**, kiedy strumień przekazywany jest bezpośrednio z urządzeń (np. kamer lub sprzętu medycznego), jak również **off-line**, gdy wykorzystywane jest nagranie zapamiętane w archiwum strumieni [bovik09].

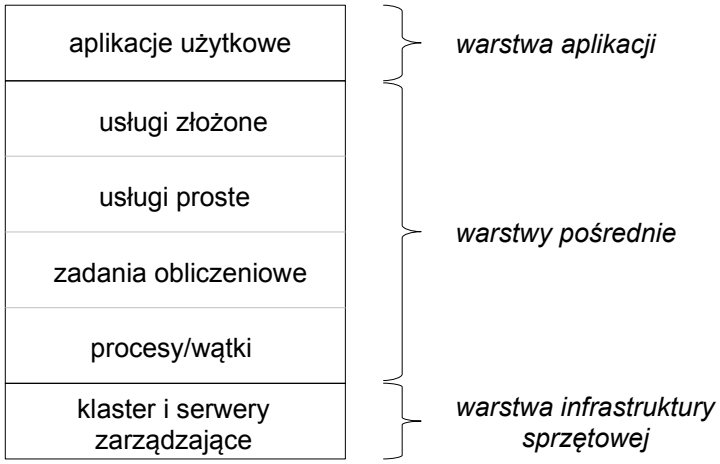
Mechanizmy **komunikacji** dostępne na platformie KASKADA obejmują przekazywanie danych wewnątrz platformy między zadaniami oraz między platformą, a aplikacjami użytkownika. Komunikacja wewnętrzna jest całkowicie zarządzana przez komponenty platformy, dotyczy to zarówno przetwarzanych strumieni jak i wiadomości informujących o występujących zdarzeniach w analizowanym strumieniu. Komunikacja z aplikacjami użytkowymi wykonywana jest w dwóch trybach: synchronicznym podczas uruchamiania usługi – kiedy aplikacja użytkowa, na zlecenie użytkownika, przekazuje serwerowi zarządzania przetwarzaniem rozproszonym właściwe parametry uruchomienia usług, a co za tym idzie, zadań oraz w trybie asynchronicznym kiedy przekazywany jest użytkownikowi wynikowy strumień danych multimedialnych lub zdarzeń.

Na rys. 11 został przedstawiony model warstwowy platformy KASKADA, jest on praktyczną realizacją warstwowego modelu opisanego w rozdz. 2.1 . W najwyższej warstwie znajdują się **aplikacje użytkowe** wykorzystywane bezpośrednio przez użytkowników, umożliwiają one dostarczając interfejs użytkownika, wykorzystanie funkcjonalności dostępnej na platformie, poprzez wywołanie odpowiedniej usługi: złożonej lub prostej.

Kolejna warstwa reprezentuje **usługi złożone** reagujące bezpośrednio na zlecenia aplikacji użytkowych według przyjętych scenariuszy działań [proficz10]. Scenariusze, tego typu podają, które z usług prostych będą wykonywane, w jaki sposób mają się komunikować i jakiej kolejności przekazywać między sobą dane. Scenariusz usługi złożonej może być przedstawiony jako acykliczny graf skierowany, gdzie wierzchołki odpowiadają poszczególnym usługom prostym, natomiast krawędzie określają kierunek przepływu strumieni między nimi.

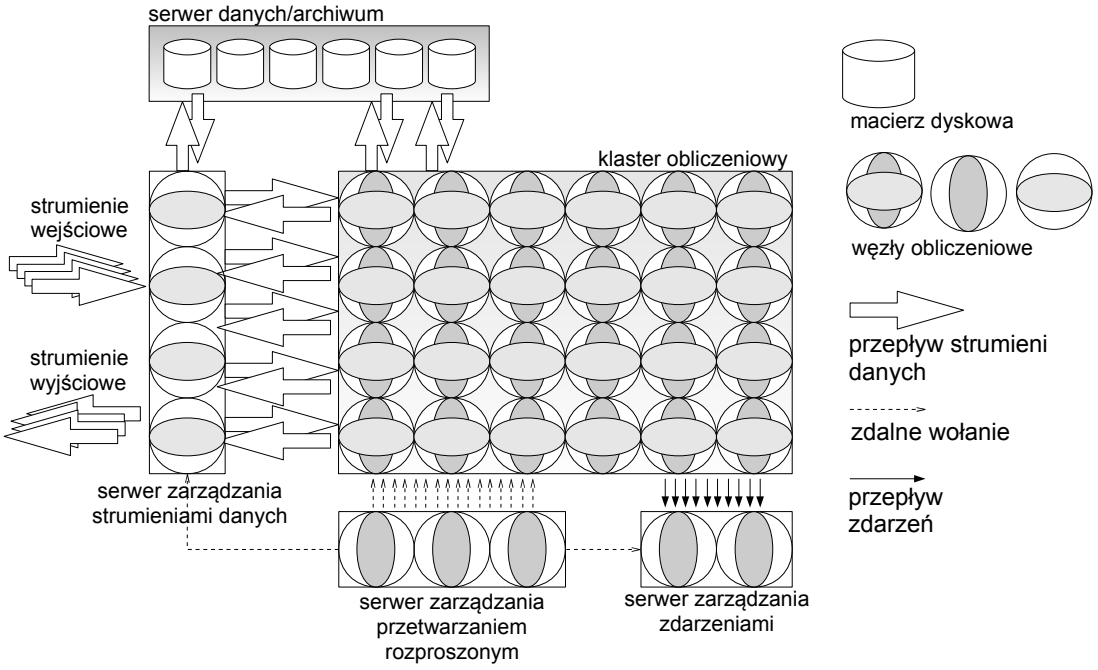
* Politechnika Gdańska, „MAYDAY EURO 2012” Superkomputerowa platforma kontekstowej analizy strumieni danych multimedialnych do identyfikacji wyspecyfikowanych obiektów lub niebezpiecznych zdarzeń. Projekt współfinansowany z Europejskiego Funduszu Rozwoju Regionalnego i Budżetu Państwa w ramach Programu Operacyjnego Innowacyjna Gospodarka.

Niższą warstwę stanowią **usługi proste** będące elementami usług złożonych i reprezentujące konkretną funkcjonalność związaną z przetwarzaniem strumieni multimedialnych. Na podstawie zadanych przez użytkownika parametrów jakościowych platforma dokonuje wyboru najodpowiedniejszego kodu spośród dostępnych zadań realizujących zadaną funkcjonalność usługi, a następnie uruchamia zadanie go wykonujące. Co więcej dla zapewnienia odpowiednio wysokiej wiarygodności przetwarzania platforma może zadecydować o uruchomieniu dwóch lub więcej zadań jednocześnie i dobrać właściwy wynik metodą np. głosowania. W przypadku wielu możliwości spełniających zadane wymagania jakościowe i funkcjonalne, wybierane jest rozwiązanie o najniższym koszcie obliczeniowym.



Rysunek 11: Model warstwowy platformy KASKADA.

Głównym elementem kolejnej warstwy są **zadania obliczeniowe**. Ich kod implementuje konkretne algorytmy analizy strumieni posiadające różne charakterystyki jakościowe i funkcjonalne. Zadania uruchomione na węzłach obliczeniowych działają pod kontrolą monitora platformy – wspomagającego start, przebieg i zakończenie wykonywanych zadań obliczeniowych, jak również kontrolującego komunikację z innymi komponentami platformy (np. dyspozytorem).

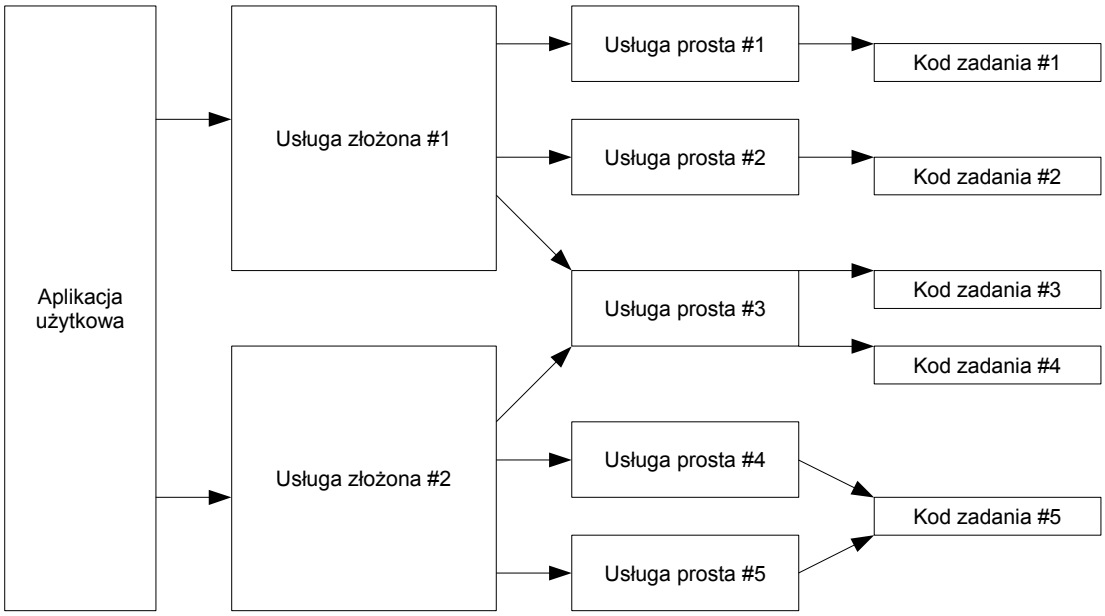


Rysunek 12: Infrastruktura sprzętowa platformy KASKADA.

Warstwa **procesów i wątków** jest zarządzana przez systemy operacyjne poszczególnych węzłów

obliczeniowych. Każdy proces posiada swój program, przydzieloną pamięć operacyjną oraz inne zasoby, np. pliki na dysku. Każde zadanie z warstwy wyższej jest reprezentowane przez pojedynczy proces umieszczony na konkretnym węźle. W ramach procesu może działać kilka niezależnych wątków wykorzystujących przypisane mu zasoby i komunikujących się za pomocą mechanizmów pamięci współdzielonej.

Najniższą warstwę platformy KASKADA stanowi infrastruktura sprzętowa oparta o **klaster obliczeniowy** oraz dodatkowe serwery stanowiące też elementy klastra, patrz rys. 12. Serwer zarządzania przetwarzaniem rozproszonym jest odpowiedzialny za zarządzanie wykonywanymi zadaniami na dostępnych węzłach klastra. Wykorzystuje on mechanizmy zdalnego wołania w celu uruchomienia oraz monitorowania obliczeń na różnym poziomie abstrakcji. Serwer danych składuje strumienie multimedialne i inne dane archiwalne, które mogą być następnie wykorzystane do przetwarzania off-line. Serwer zarządzania strumieniami danych umożliwia wybór odpowiedniego źródła danych: urządzeń zewnętrznych albo archiwum danych, jak również jest odpowiedzialny za dystrybucję strumieni wynikowych oraz redystrybucję strumieni z archiwum. Serwer zarządzania zdarzeniami umożliwia kontrolę platformy nad komunikacją wewnątrz systemu jak również koordynuje odbiór i przesyłanie zdarzeń na zewnątrz platformy.



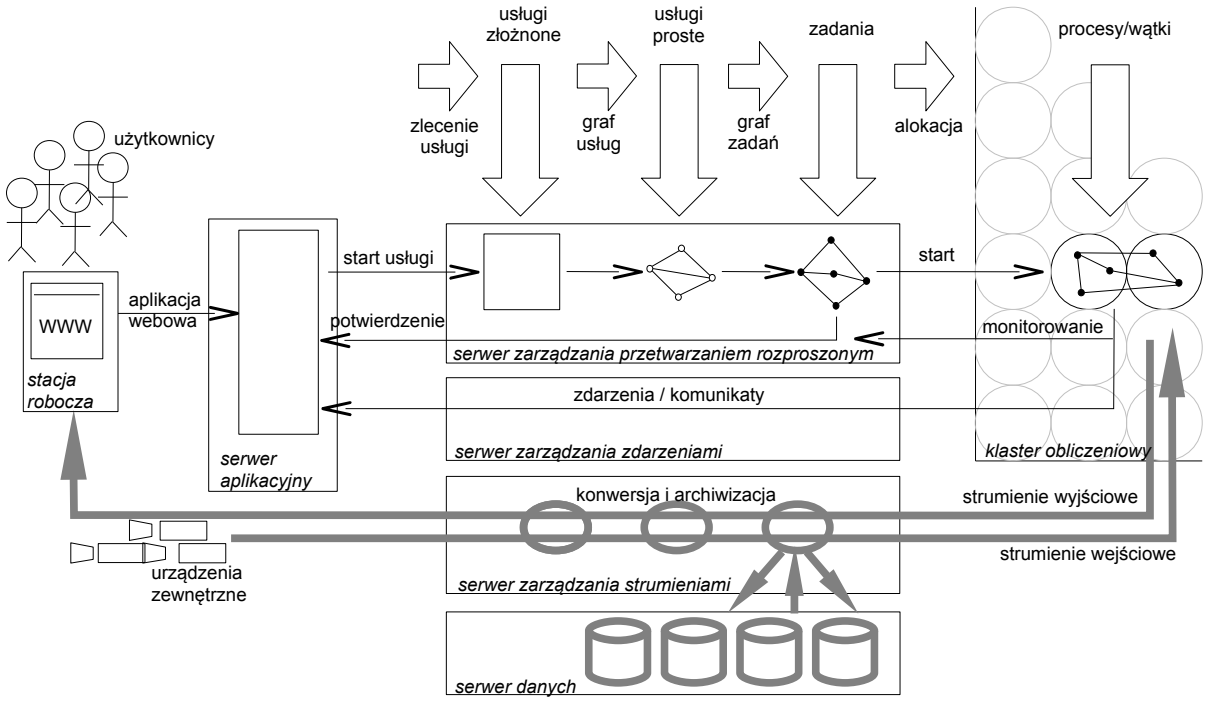
Rysunek 13: Przykład powiązań między komponentami aplikacji na platformie KASKADA.

Na rys. 13 został przedstawiony przykład powiązań między poszczególnymi komponentami aplikacji na platformie KASKADA, z zachowaniem założeń architektury warstwowej. Przykładowa aplikacja składa się z następujących komponentów:

- jednej aplikacji użytkowej, odpowiedzialnej za bezpośrednią współpracę z użytkownikiem, może ona być realizowana w dowolny sposób np. jako aplikacja webowa,
- dwóch usług złożonych, reprezentujących złożoną funkcjonalność platformy,
- pięciu usług prostych, z których każda reprezentuje prostą funkcjonalność platformy, jedna z tych usług (#3) jest wspólna dla obu usług złożonych przykładowej aplikacji,
- pięć różnych kodów zadań będących implementacją pięciu różnych algorytmów przetwarzania strumieni, usługi proste #1 i #2 wykorzystują inny kod zadania każda (najbardziej typowe rozwiązanie), usługa #3 w zależności od wymagań jakościowych wykorzystuje kod zadania #3 bądź #4, natomiast usługi #4 i #5 wykorzystują jeden wspólny kod zadania #5.

Na rys. 14 została przedstawiona procedura zarządzania usługami multimedialnymi na platformie KASKADA. Poniżej zostały opisane kolejne kroki, w których platforma obsługuje i zarządza wykonywaniem usług:

1. **Odbiór od użytkownika zlecenia uruchomienia usługi.** Użytkownik uruchamia aplikację i wybiera wymaganą funkcjonalność przetwarzania lub analizy strumieni multimedialnych lub bezpośrednio przez interfejs graficzny platformy wysyła żądanie startu wybranej usługi, opisanej za pomocą dokumentu WSDL [wsdl]. Serwer zarządzania przetwarzaniem rozproszonym odbiera zlecenie wykonując odpowiednią usługę sieciową, identyfikuje odpowiadającą jej usługę złożoną i przeprowadza walidację formatu danych wejściowych. Platforma odczytuje dane usługi złożonej z repozytorium usług, dokonuje weryfikacji danych pod względem zakresów parametrów liczbowych, poprawności typów wyliczeniowych, a także dodatkowo wykonywane jest parsowanie parametrów jakościowych.
2. **Uruchomienie usługi złożonej.** Na podstawie danych z repozytorium usług, następuje pobranie właściwego scenariusza wykonania usługi złożonej. Na jego podstawie tworzona jest lista odpowiadających mu usług prostych, identyfikowane są wejściowe strumienie danych, określane miejsca dostarczenia wyjściowych strumieni danych oraz weryfikowane parametry wywołania kolejnych usług prostych.



Rysunek 14: Przyjęta koncepcja zintegrowanego zarządzania na platformie KASKADA.

3. **Uruchomienie usług prostych.** Każda usługa realizuje konkretną funkcjonalność związaną z jednym lub więcej strumieniami danych. Na podstawie zadanych parametrów jakościowych wybierany jest odpowiedni kod zadania realizujący daną usługę lub w przypadku wysokich wymagań jakościowych, jak wspomniano wcześniej dwa lub więcej kody jednocześnie.
4. **Start zadań obliczeniowych.** Po zarezerwowaniu zasobów obliczeniowych i sieciowych, wybrane zadania są uruchamiane bezpośrednio na węzłach klastra obliczeniowego. Wymagane strumienie oraz parametry startowe są podawane na wejście procesów realizujących poszczególne zadania, jak również konfigurowane są kanały wysyłania

wyników ich działania. Pojedynczy proces danego zadania obsługuje cały strumień, a nie tylko jego pojedynczy element (patrz rys. 9).

5. **Wykonywanie i monitorowanie zadań obliczeniowych.** Podczas wykonywania funkcjonalności usług platforma śledzi działanie wszystkich odpowiadających im zadań obliczeniowych. Sprawdzany jest ich bieżący stan, dostępność i poziom wykorzystania potrzebnych zasobów. Serwer zarządzania zdarzeniami selekcjonuje i przesyła do użytkownika komunikaty generowane przez zadania. W przypadku wystąpienia sytuacji wyjątkowej, takiej jak np. przekroczenie zadeklarowanej mocy obliczeniowej, platforma podejmuje odpowiednie czynności, np. zakończenie zadania, czy związanej z nim usługi oraz informuje o tym użytkownika.
6. **Zakończenie działania usługi.** Wyróżniamy trzy możliwości zakończenia usługi: (1) w wyniku decyzji użytkownika, kiedy bezpośrednio wysyła żądanie zakończenia jej działania, lub pośrednio po upływie zadeklarowanego czasu realizacji usługi; (2) w wyniku prawidłowego zakończenia pracy przez usługę, kiedy to jej zadania sygnalizują koniec przetwarzania i wysyłają wyniki końcowe, np. wykrycie poszukiwanego zdarzenia, obiektu lub zakończenie transmisji; (3) w wyniku błędu któregoś z zadań, np. poprzez niespodziewane jego zakończenie lub wystąpienie problemu wewnątrz platformy, np. awaria węzła obliczeniowego, na którym działało zadanie.

3.2 Porównanie platformy KASKADA z istniejącymi systemami rozproszonymi

Przedstawiona powyżej koncepcja zarządzania została zrealizowana i przetestowana na platformie KASKADA. Jest to podejście scentralizowane dysponujące dostępem do informacji o globalnym stanie, wykorzystujące homogeniczny klaster komputerowy oraz przetwarzanie w czasie rzeczywistym dużej liczby strumieni multimedialnych. Porównanie głównych różnić z innymi rozwiązaniami zostało przedstawione w tabeli 2.

Większość rozwiązań jest bardzo ogólnych, tzn. dotyczy dowolnego modelu przetwarzania np. Globus [foster96], UNICORE [breuer03], BeesyCluster [czarnul02]. Rozwiązanie przyjęte na platformie KASKADA charakteryzuje się swoją specyfiką wynikającą z przetwarzania konkretnej kategorii danych (strumieni multimedialnych), jest otwarte na różny poziom jakości przetwarzania oraz uwzględnia istnienie usług wspomagających przetwarzanie. Platforma wspiera też tworzenie nowych usług i implementację algorytmów poprzez dostarczenie mechanizmów obróbki strumieni testowych – ułatwiających rozwój i uruchamianie nowo-tworzonych komponentów. Natomiast przy przetwarzaniu strumieni „na żywo”, umożliwia ich archiwizację, dystrybucję i konwersję, dzięki czemu zapewnia unifikację podstawowego szablonu wspierającego wykorzystywaną klasę algorytmów (por. rozdz. 2.9) w ramach specjalnego framwork'u: ramki KASKADA. Dzięki temu twórca algorytmu koncentruje się tylko na jego podstawowych elementach, pozostawiające sprawy techniczne i pomocnicze odpowiednim funkcjom platformy. Innym, wyróżniającym mechanizmem, tej platformy, jest obsługa zdarzeń wewnętrznych i związanej z nimi komunikacji. Z reguły systemy gridowe muszą wspierać heterogeniczność środowisk w których działają, natomiast KASKADA działa na jednolitym, homogenicznym klastrze co upraszcza pewne mechanizmy i czyni ją bardziej wydajną. Istotną cechą wspólną powyższych rozwiązań i platformy KASKADA jest wykorzystanie usług sieciowych (ang. webservices), jak również wdrożona funkcjonalność zapewniająca odpowiedni poziom bezpieczeństwa.

Przykład innej platformy przetwarzającej strumienie multimedialne został przedstawiony w [taoyu09]. Podobnie jak KASKADA akceptuje ona model aplikacji jako acykliczny graf skierowany, który następnie jest interpretowany i uruchamiany w ramach framework'u UIMA

[uima], natomiast same obliczenia wykonywane są przez silnik bazy danych PostgreSQL [postgres] jako procedury składowane. W przypadku platformy KASKADA obliczenia wykonywane są przez zadania uruchomione na klastrze komputerowym, co eliminuje związanie się z jednym rodzajem bazy danych, a tym samym z mechanizmami dotyczącymi wydajności i skalowalności obliczeń wykonywanych w warstwie danych.

Inna propozycja przetwarzania strumieni multimedialnych została przedstawiona w [eide03]. Autorzy zdefiniowali hierarchiczny model przetwarzania, składający się z elementów odpowiadających poszczególnym etapom analizy strumieni wideo: strumieniowanie – umożliwiające przesyłanie obrazu w całości, lub podzielonego na obszary, filtrowanie – umożliwiające przystosowanie obszaru obrazu do dalszej obróbki, wydobycie cech – odpowiedzialne za ustalenie konkretnych właściwości obszaru obrazu, a także klasyfikację – końcowy etap decydujący o wynikach analizy. Wykorzystując taki model, zaproponowano automatyczne mechanizmy doboru jakości przetwarzania (rozdzielczość czasowa i dokładność) w zależności od wydajności (rozmiar danych wejściowych, wyrażony jako liczba przetwarzanych obrazów w czasie) dla zadanej konfiguracji sprzętowej wykorzystywanego systemu rozproszonego. W przeciwieństwie do platformy KASKADA, powyższe rozwiązanie wymaga dostarczenia rozbudowanych metadanych w postaci systemu oceniającego jakość danej konfiguracji a priori, jeszcze przed uruchomieniem przetwarzania, jak również budowy aplikacji według sztywnych reguł wynikających z przyjętego modelu przetwarzania.

Tabela 2: Porównanie platformy KASKADA z innymi systemami rozproszonymi: UNICORE/Globus/BeesyCluster [breuer03, foster96, czarnul02], VAP [taoyu09] i RTVCA [eide03].

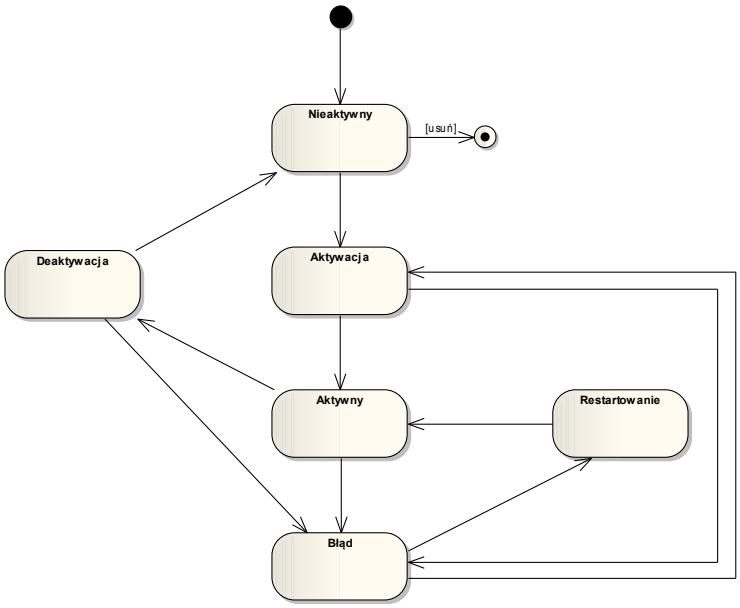
Charakterystyka	UNICORE/ Globus/ BeesyCluster	VAP	RTVCA	KASKADA
Wsparcie dla przetwarzania czasu rzeczywistego strumieni multimedialnych	-	+	+	+
Wykorzystanie architektury zorientowanej na usługi (SOA)	+	-	-	+
Akceptowanie grafowego modelu aplikacji	-	+	+	+
Uwzględnienie jakości usług (QoS)	+	-	+	+
Wsparcie dla archiwizacja strumieni	-	+	-	+

3.3 Zarządzanie na platformie KASKADA

Zgodnie z procedurą zarządzania usługami multimedialnymi na platformie KASKADA (rozdz. 3.1) podczas obsługi żądania wykonania usługi następuje szereg czynności zarządzających. Pierwszym rozważanym mechanizmem jest **odbiór i archiwizacja strumieni multimedialnych** w repozytorium danych. Wszystkie strumienie on-line z urządzeń zewnętrznych są odbierane przez węzły serwera zarządzania strumieniami. Posiada on własną bazę danych, w której dla każdego strumienia zapisane są następujące informacje: nazwa – będąca jednocześnie jego identyfikatorem używanym przez klientów platformy, opis słowny, port dla protokołu KBIN, adres URL urządzenia, czas archiwizacji strumienia oraz typ.

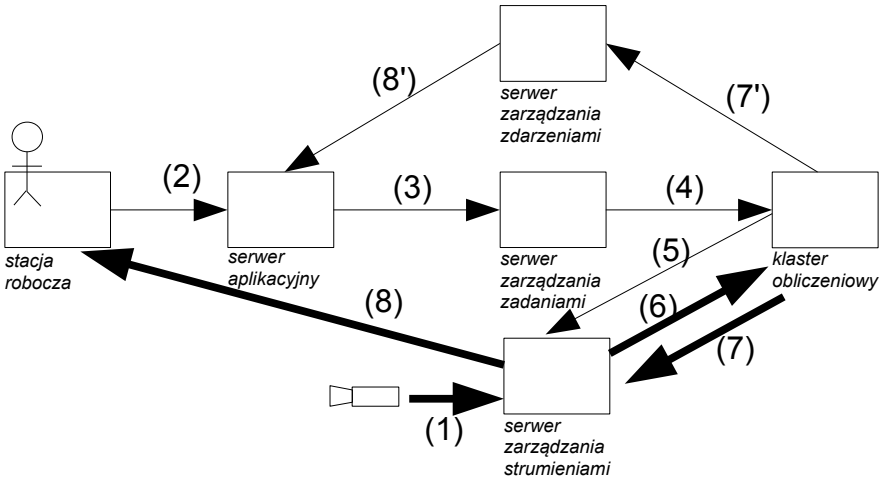
Każdy z takich strumieni posiada flagę opisującą jego aktualny status. Po wprowadzeniu nowego urządzenia (kamery lub mikrofonu) znajduje się ono w stanie nieaktywnym, następnie administrator może zainicjować aktywację odbioru danych – wtedy to stan tego urządzenia zmienia się na aktywację i jeżeli odbiór strumienia po protokole RTSP rozpocznie się poprawnie, następuje przejście do stanu: aktywny, w przeciwnym przypadku do stanu: błąd. Stan aktywny trwa tak długo, aż administrator nie zdecyduje się deaktywować tegp strumienia lub nie wystąpi inny błąd. Strumień w czasie deaktywacji zostaje zamknięty, a połączenie z urządzeniem zakończone. Ze stanu błędu strumień może wyjść automatycznie – kiedy to platforma KASKADA próbuje

zrestartować odbiór strumienia lub ręcznie przez administratora inicjującego ponowną aktywację, patrz rys. 15.



Rysunek 15: Maszyna stanów odbieranego strumienia multimedialnego na platformie KASKADA.

Kiedy strumień jest w stanie aktywnym jego elementy są archiwizowane w repozytorium danych. Wykorzystywane są do tego standardowe format danych: MKV, Matroska [matroska]. Strumień jest przechowywany w postaci półgodzinnych plików i jego archiwizowane dane są dostępne dla użytkowników poprzez węzły serwera zarządzania strumieniami. Czas przechowywania danych jest określony w dniach i jest definiowany osobno dla każdego urządzenia. Strumienie archiwalne mogą zostać wykorzystane do utworzenia strumieni testowych – umożliwiających powtarzalne badanie zachowania testowanych zadań obliczeniowych. Dane archiwalne nieoznaczone jako strumienie testowe, starsze niż przyjęty okres czasu przechowywania są usuwane.

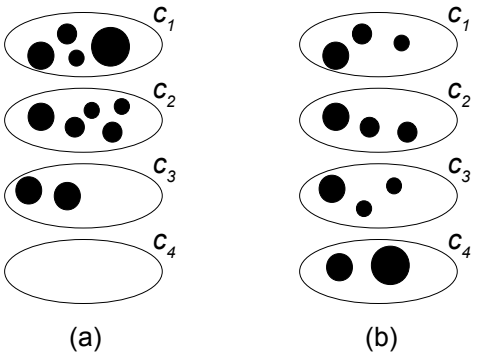


Rysunek 16: Schemat uruchomienia usługi na platformie KASKADA: (1) odbiór strumienia, (2) wołanie aplikacji, (3) żądanie usługi, (4) uruchomienie zadań, (5) żądanie strumienia wejściowego, (6) przesyłanie strumienia wejściowego, (7) przesyłanie strumienia wynikowego, (7') przesyłanie komunikatów zdarzeń, (8) przesyłanie strumienia wynikowego do użytkownika, (8') przesyłanie komunikatów zdarzeń do aplikacji webowej.

Rozważmy **mechanizm odbioru żądania wykonania usługi** oraz inicjowanie przetwarzania przez

usługę. Na rys. 16 przedstawiono kolejne kroki uruchomienia usługi z punktu widzenia komunikacji między poszczególnymi serwerami platformy. Zakładamy, że w kroku (1) przetwarzany strumień wejściowy jest aktywny: odbierany i archiwizowany przez serwer zarządzania strumieniami. Zgodnie z procedurą opisaną w rozdz. 3.1 , aplikacja webowa, wykorzystywana bezpośrednio przez użytkownika (krok 2), wysyła żądanie wykonania usługi złożonej (krok 3). Po otrzymaniu żądania wykonania usługi serwer zarządzania przetwarzaniem rozproszonym, ustala jaki zbiór zadań jest potrzebny aby zrealizować daną usługę i w kroku (4) uruchamia je na kolejnych węzłach obliczeniowych. W celu optymalizacji liczby wykorzystanych węzłów użyty jest specjalny algorytm alokacji. Na każdym z węzłów następuje uruchomienie odpowiednich zadań, ich kod ładowany jest z dysku sieciowego. Następnie, w kroku (5) uruchamiane zadania zgłaszają się do serwera zarządzania strumieniami w celu uzyskania dostępu do strumienia wejściowego. W kroku (6) odpowiedni strumień jest przekazywany do zadań, przetwarzany i w zmodyfikowanej postaci strumienia wynikowego zwracany do serwera zarządzania strumieniami (krok 7), a dalej przekazywany do stacji roboczej użytkownika. Równolegle, w przypadku wykrycia zdarzeń w analizowanym strumieniu multimedialnym, w kroku (7') wysyłane są komunikaty w formacie XML do serwera zarządzania zdarzeniami, który dokonuje ich filtrowania i w przypadku zajścia takiej potrzeby w kroku (8') przekazuje je do aplikacji webowej na serwerze aplikacyjnym.

Kolejnym interesującym mechanizmem wykorzystywanym na platformie KASKADA jest planowanie przypisania zadań obliczeniowych działających na strumieniach multimedialnych do zasobów obliczeniowych. Jak to zostało przedstawione w rozdz. 2.5 w przypadku zarządzania zadaniami, dla których istnieje relacja poprzedzania, tj. przetwarzania danego zadania musi się zakończyć przed uruchomieniem kolejnego zadania, stosujemy algorytmy szeregowania. Na przykład dla przetwarzania pojedynczej klatki obrazu, kiedy zadanie #1 dokonujące jej rozpakowania, zadanie #2 zajmujące się analizą i modyfikacją oraz zadanie #3 dotyczące jej ponownej kompresji powinny być wykonywane po kolei. Na ogół te czynności realizuje się na pojedynczym węźle.



Rysunek 17: Przykłady dwóch alokacji jedenastu zadań na czterech węzłach wykonanych przez dwa różne algorytmy alokacji: (a) wykorzystujący minimalną liczbę węzłów, (b) rozkładający obciążenie w sposób równomierny. Średnica kół reprezentuje wielkość obciążenia węzła przez dane zadanie.

Kiedy zadania obsługują całe strumienie danych multimedialnych, np. ciąg klatek obrazu to ich działanie nie kończy się wraz z przetworzeniem pojedynczej klatki. Dodatkowo w czasie przetwarzania są przekazywane między nimi komunikaty i dlatego wymagają one jednoczesnego uruchomienia w czasie inicjowania usługi. Każde zadanie jest pewną zamkniętą całością i do zarządzania przydziałem zasobów powinien być wykorzystywany **algorytm alokacji** [el-rewini04]. Na rys. 17 przedstawiono przykłady dwóch różnych alokacji tego samego zbioru zadań, dokonanych dla różnych kryteriów alokacji. Opis, analiza i ocena zaproponowanego algorytmu alokacji wykorzystanego na platformie KASKADA został umieszczony w rozdz. 5 . Zgodnie z

założeniami z rozdz. 2.9 przyjmujemy, że każde zadanie pracuje według następującego wzorca: odbiór elementu danych (np. klatki obrazu), przetwarzanie, wysłanie elementu danych, oczekiwanie, odbiór elementu danych itd. (patrz rys. 9) Z tego powodu można przyjąć, że średnie obciążenie węzła (wyznaczane dla czasu większego niż okres przesyłanych elementów strumienia) jest stałe.

W celu śledzenia obliczeń pojawia się potrzeba zastosowania kolejnego mechanizmu wykorzystywanego w platformie KASKADA: **monitorowanie**. Na każdym węźle obliczeniowym uruchamiany jest rezydentny proces monitorujący, którego zadaniem jest:

- startowanie zadań na węźle,
- kontrola komunikacji i obciążenia obliczeniowego na danym węźle generowanego przez każde z uruchomionych na nim zadań,
- zakończenie działania zadań na węźle,
- kontrola poprawności połączenia węzła z serwerami zarządzającymi.

Istotną cechą podsystemu monitorowania i algorytmu alokacji jest ich adaptacja do różnego przyrostu obciążenia zadań przetwarzających strumienie multimedialne. Na rys. 18 zostały przedstawione dwie hipotetyczne alokacje tego samego zadania, posiadającego ten sam algorytm analizy strumieni, dla tych samych danych wejściowych, na dwóch homogenicznych węzłach. W przypadku (a) zadanie jest wykonywane jako jedyne na danym węźle oraz przypadek (b) kiedy towarzyszą temu zadaniu inne zadania przetwarzające strumienie multimedialne. Zauważmy, że obciążenie obliczeniowe węzłów tym samym zadaniem jest różne: 30% dla przypadku (b) w porównaniu z 25% dla przypadku (a). Taka tendencja została zauważona przy testowaniu obciążenia węzłów na platformie KASKADA. Algorytm alokujący oraz monitor muszą brać pod uwagę to zjawisko i odpowiednio jet korygować. Obszerniejsze badania tego problemu oraz propozycja odpowiedniej funkcji korekty zostały przedstawione w rozdz. 4 .

Rysunek 18: Alokacje tego samego zadania t_1 na dwóch rożnych węzłach: (a) na pustym węźle (bez innych zadań), (b) na węźle częściowo zajętym przez inne zadania.

Platforma KASKADA zawiera wszystkie powyższe procedury zarządzania, które zostały zaprojektowane, zaimplementowane i przetestowane dla rzeczywistych warunków pracy przy wykorzystaniu klastra Galera znajdującego się w CI TASK na Politechnice Gdańskiej [task]. Rys. 19 przedstawia główne komponenty przyjętego rozwiązania:

- Menedżer platformy (Platform Manager) – aplikacja webowa odpowiedzialna za zarządzanie innymi komponentami platformy, dostarczająca interfejs użytkownika, odbierająca i przetwarzająca żądania wykonania usług i alokację.
- Usługa (Service) reprezentuje dostępną funkcjonalność możliwą do wykorzystania przez aplikacje użytkowe. Zgodnie z modelem z rys. 11 usługa złożona jest realizowana przez usługi proste, natomiast usługi proste przez zadania obliczeniowe.
- Zadanie obliczeniowe (Computation Task) jest komponentem, który umożliwia wykonywanie konkretnego algorytmu przetwarzania strumieni multimedialnych. Samo

zadanie koncentruje się na organizacji procesu przetwarzania jednego lub wielu strumieni multimedialnych.

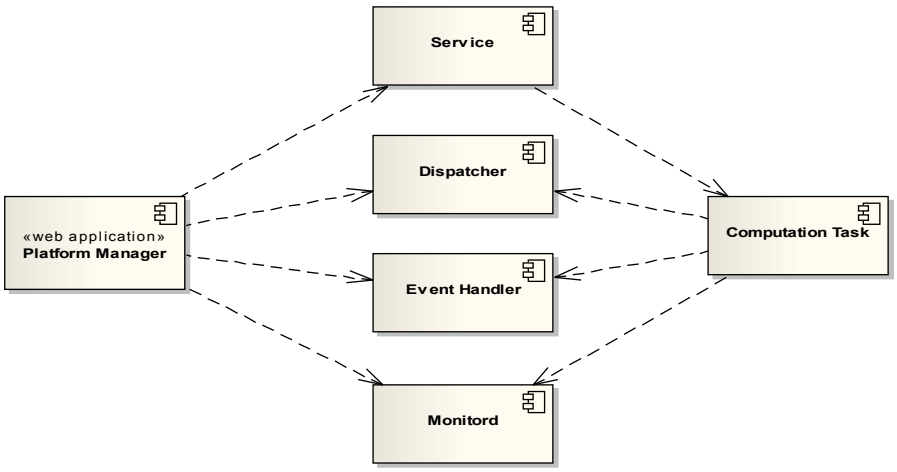
- Dyspozytor (Dispatcher) – komponent odpowiedzialny za odbiór i archiwizację strumieni multimedialnych pozyskiwanych z zewnątrz platformy jak i generowanych przez zadania obliczeniowe.
- Obsługa zdarzeń (Event Handler) – komponent odpowiedzialny za obsługę komunikatów zawierających informacje o zdarzeniach wewnętrznych wykrytych przez zadania obliczeniowe, jak również o zdarzeniach wygenerowanych wewnętrznie przez platformę.
- Rezydentny proces monitorujący (Monitord) – komponent umożliwiający monitorowanie przebiegu obliczeń i dotyczy poszczególnych zadań i węzłów obliczeniowych.

Komponenty programowe zostały przetestowane, wdrożone i udokumentowane w ramach kolejnych iteracji rozbudowy platformy KASKADA.

3.4 Przykład wykorzystania platfromy KASKADA

Dla ilustracji problematyki rozprawy doktorskiej rozważmy następujący przykład aplikacji *Faces* realizowanej na platformie: w monitorowanym budynku względy bezpieczeństwa wymagają aby twarze wszystkich osób wchodzących były zapamiętywane w postaci zdjęć. Przy wszystkich wejściach zainstalowane są kamery wideo, z których strumienie multimedialne przekazywane są do platformy KASKADA. Tam powinna nastąpić lokalizacja każdej twarzy w strumieniu wideo, zaznaczenie oraz skopiowanie takich obrazów ze strumienia, a następnie archiwizacja w postaci zdjęć cyfrowych.

Aplikacja *Faces* dla wybranych kamer rozmieszczonych w danym obszarze geograficznym analizuje strumienie wideo pochodzące z tych kamer i rozpoznaje w czasie rzeczywistym twarze wszystkich zarejestrowanych osób. Przyjmuje się, że w jednej zarejestrowanej klatce obrazu strumienia wideo może znajdować się kilka twarzy. W wyniku działania aplikacji powstaje baza danych zdjęć wykrytych twarzy oraz multimedialny strumień wejściowy zamieniony w strumień wyjściowy zawierający zaznaczone obszary ROI (ang. Region Of Interest) reprezentujące wykryte twarze. Aplikacja może być rozbudowana o identyfikację różnych twarzy, o kontrolę osób znajdujących się aktualnie w budynku itp. Ograniczymy się jednak tylko do detekcji twarzy ze strumienia.



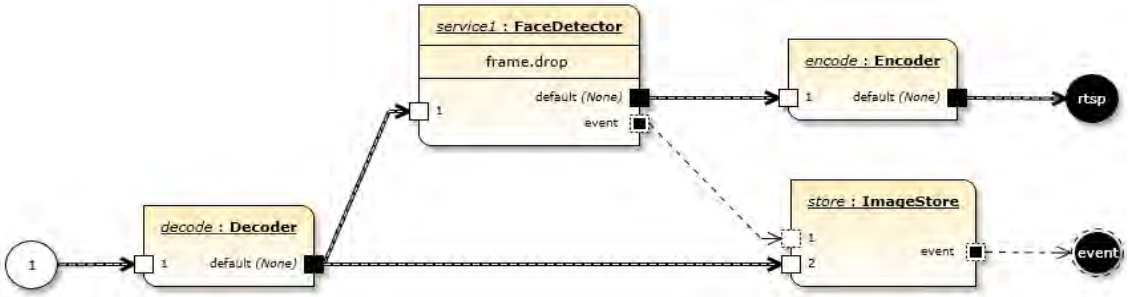
Rysunek 19: Główne komponenty programowe platformy KASKADA.

Warstwa najwyższa aplikacji stanowi przeglądarkę z odpowiednim graficznym interfejsem

użytkownika pozwalającym na wybór monitorowanych strumieni, czas ich obserwacji, uruchamianie dostępnych usług, wybór bazy danych i miejsca archiwizacji wyników przetwarzania oraz ich ponownego przeglądania. Warstwa usług składa się z jednej usługi złożonej, która analizuje pojedynczy strumień i zapewnia wykrycie twarzy jak i zaznaczenie odpowiadających im obszarów (ROI), będących prostokątami różnej wielkości. Wykorzystuje ona cztery usługi proste:

- *Decoder* – dekodowanie strumienia wejściowego,
- *FaceDetector* – detekcja twarzy (identyfikacja ROI),
- *ImageStore* – zapis twarzy (ROI) w bazie danych,
- *Encoder* – kodowanie strumienia wyjściowego.

W związku z tym scenariusz usługi złożonej jest przedstawiony na rys. 20.



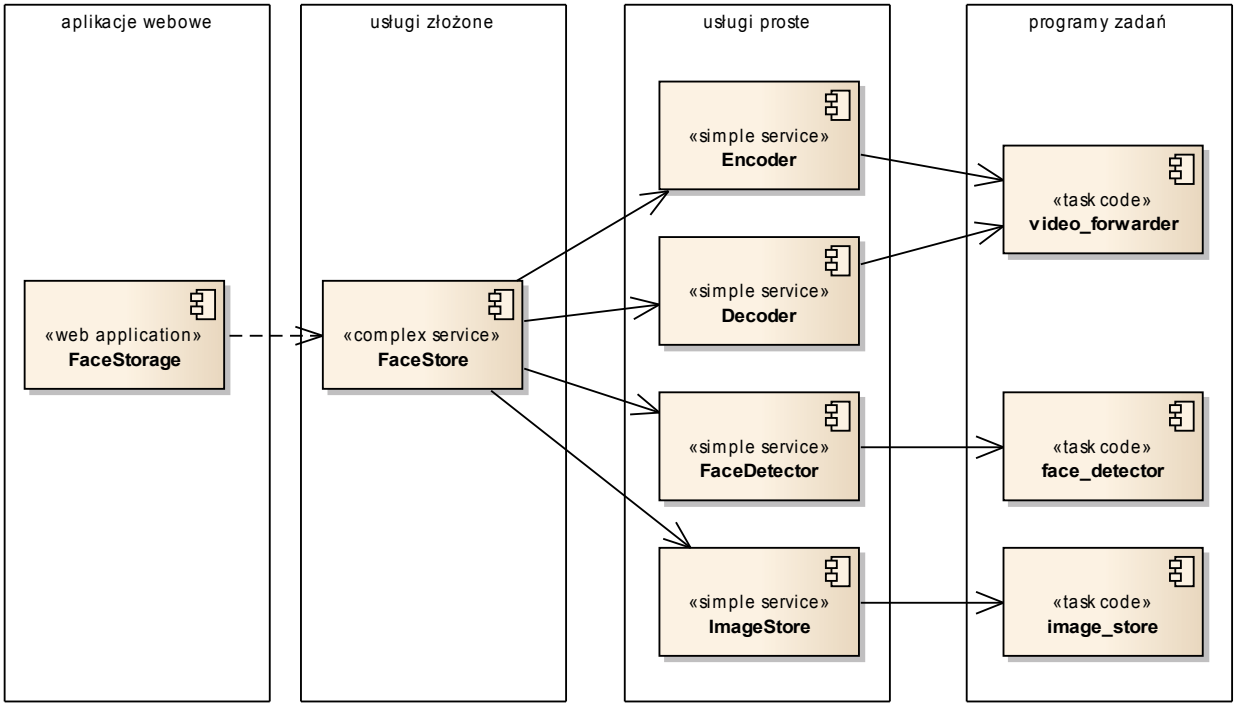
Rysunek 20: Przykład usługi złożonej opisanej w języku MSP-ML.

W opisywanym przykładzie każda usługa prosta jest realizowana przez pojedyncze uruchomione zadanie, które wiążę dany kod zadania i implementowany przez niego algorytm z wybranym strumieniem. W przypadku ogólnym, kiedy istniałoby kilka alternatywnych kodów zadań realizujących funkcjonalność zadanej usługi prostej, wybór zostałby dokonany na podstawie wymagań jakościowych podanych przez użytkownika. Pojedyncze zadanie jest wykonywane na jednym węźle jako jeden wielowątkowy proces zarządzany przez system operacyjny Linux. Platforma KASKADA przy pomocy procedur alokacji decyduje o rozmieszczeniu zadań na węzłach. Zadanie przetwarza cały strumień multimedialny i jego wykonanie trwa od uruchomienia usługi aż do momentu kiedy minie założony z góry czas przetwarzania, bądź użytkownik aplikacji zadecyduje o odłączeniu strumienia i zakończeniem związanych z nią zadań. W czasie realizacji zadanych zadań, mogą być zgłaszane nowe, które są dynamicznie alokowane na węzły.

Realizacja oprogramowania spełniającego powyższe założenia wymaga następujących komponentów (patrz rys. 21):

1. Aplikacja webowa jest odpowiedzialna za zarządzanie procesem biznesowym na najwyższym poziomie. Umożliwia ona bezpośrednią komunikację z użytkownikiem końcowym, dostarczając mu interfejsu graficznego, dzięki któremu będzie on mógł zarządzać rejestracją twarzy oraz mieć dostęp do archiwum z zapisanymi zdjęciami. Z punktu widzenia platformy zadaniem aplikacji jest inicjowanie i zatrzymywanie przetwarzania – uruchamianie i kończenie działania usługi rejestracji twarzy na odpowiednich strumieniach wideo – odpowiadających konkretnym kamerom umieszczonych przy wejściach do budynków.
2. Usługa złożona jest odpowiedzialna za wykonanie usług prostych, inicjację i kontrolę komunikacji między nimi. Pojedyncza aplikacja może uruchomić wiele takich usług jednocześnie, co w przypadku rejestracji zdjęć twarzy oznacza jedną usługę dla każdej kamery umieszczonej przy monitorowanym wejściu do budynku. Struktura takiej usługi jest zapisywana w postaci dokumentu XML i może być graficznie przedstawiona za pomocą

notacji MSP-ML, patrz rys. 20.



Rysunek 21: Struktura komponentów aplikacji Faces dla platformy KASKADA w języku UML.

3. Usługa prosta dekodowania strumienia wideo jest standardową usługą dostarczaną przez platformę KASKADA. Umożliwia ona rozpakowanie strumienia dostarczonego przez kamerę i przekształcenie go w sekwencję niezależnych obrazów przekazywaną kolejnym usługom zgodnie ze strukturą usługi złożonej.
4. Usługa prosta detekcji twarzy – jej zadaniem jest odnalezienie twarzy na kolejnych klatkach obrazu strumienia wideo za pomocą kaskady klasyfikatorów cech Haar'a [viola01] zaimplementowanej w bibliotece OpenCV [opencv], przesłanie wiadomości o tym zdarzeniu do usługi zapisu zdjęć, a następnie zaznaczenie ramki wokół znalezionego obiektu i wysłanie tak przetworzonego strumienia multimedialnego do usługi kodowania.
5. Usługa zapisu zdjęć odbiera wiadomości o wykrytych twarzach, odnajduje je na strumieniu wideo i kopiuje je w postaci statycznych zdjęć do archiwum. Wyniki działania tej usługi nie obejmują transmisji strumienia wynikowego, za to posiada ona dwa wejścia jedno wykorzystujące strumień multimedialny, drugie wiadomości ze zdarzeniami wykrycia twarzy.
6. Usługa kodowania strumienia wideo umożliwia przekształcenie wejściowego, nieskompresowanego strumienia wideo w zakodowany strumień H.264 [h264]. Podobnie jak usługa dekodowania jest ona standardową usługą dostarczaną w repozytorium platformy KASKADA.

W powyższym przykładzie, każda z usług *FaceDetector* i *ImageStore* jest realizowana przez jeden kod zadania implementujący konkretny algorytm przetwarzania strumienia multimedialnego lub wiadomości, natomiast usługi *Decoder* i *Encoder* wykorzystują jeden wspólny kod zadania: *video_forwarder*. Ponieważ sama platforma dostarcza kod zadań dla konkretnych usług standardowych deweloper wykonujący oprogramowanie dla powyższego przykładu, powinien zaimplementować tylko algorytmy zadań dla usług detekcji twarzy na obrazach strumienia wideo

oraz zapisu zdjęć wykrytych obiektów do archiwum (*FaceDetector* i *ImageStore*).

Na platformie KASKADA kod zadań jest implementowany w języku C++ [stroustrup00] z udziałem dedykowanego framework'u – ramki KASKADA – specjalnego zbioru bibliotek ułatwiającego i unifikującego implementację algorytmów przetwarzania strumieni multimedialnych, wraz ze wsparciem dla przetwarzania równoległego i komunikacji strumieniowej. Wykorzystuje ona dobrze znane biblioteki: Boost [boost, mayers05] do organizacji kodu klas, ich dziedziczenia i manipulacji na obiektach oraz GStreamer [gstreamer] do dekodowania i kodowania strumieni multimedialnych z wykorzystaniem standardu H.264 [h264] oraz obsługi protokołów zewnętrznego RTSP [schulzrinne98] i wewnętrznego KBIN.

W przedstawianym przykładzie, kod zadań: *face_detector* i *image_store* implementuje jedynie metody *processDataObject()* klasy dziedziczącej po *KASKADAStreamAlgorithm*, cała reszta szczegółów implementacyjnych, w tym pętla odbierająca poszczególne elementy strumieni wejściowych jest zaszyta w bibliotekach ramki KASKADA. Sam kod zadań oprócz wspomnianych bibliotek Ramki KASKADA wykorzystują dodatkową bibliotekę OpenCV [opencv, bradski08] zawierającą implementacje algorytmów analizy i manipulacji obrazami, np. skalowanie czy rozpoznawania obrazów.

Tak wytworzone komponenty są instalowane na poszczególnych serwerach platformy KASKADA: aplikacja na niezależnym serwerze aplikacyjnym JEE, usługi na serwerze zarządzania przetwarzaniem rozproszonym, a kod zadań na klastrze obliczeniowym (por. rys. 22). Początkowo uruchamiana jest tylko aplikacja webowa, oczekująca na zlecenia użytkownika.

Użytkownik wykorzystując interfejs graficzny aplikacji wybiera kamery dla których ma nastąpić archiwizowanie zdjęć twarzy, po czym aplikacja uruchamia usługi złożone dla każdej z wybranych kamer. W tym momencie kontrolę nad przetwarzaniem strumieni przejmuje sama platforma. Ponadto użytkownik może wykorzystać aplikację webową do przeglądania składowych zdjęć oraz do zakończenia monitorowania poszczególnych kamer, co przekłada się na zlecenia zakończenia działania poszczególnych instancji usługi złożonej.

Opisane powyżej wielokrotne uruchomienie usługi złożonej powoduje, kolejne uruchomianie usług prostych przetwarzających strumień multimedialny z konkretnej kamery. Serwer zarządzania przetwarzaniem rozproszonym jest odpowiedzialny za taki przydział zasobów aby komunikacja i obliczenia mogły być wykonywane w czasie rzeczywistym. Dla każdej działającej usługi prostej z opisywanego przykładu uruchamiane jest pojedyncze zadanie obliczeniowego na klastrze obliczeniowym. To właśnie do nich przekazywane są strumienie danych zawierające obrazy wideo. Dlatego też procedura alokacji węzłów obliczeniowych musi brać pod uwagę nie tylko obciążenie procesorów generowane przez obliczenia wykonywanych zadań, ale także ich wymagania komunikacyjne wynikające z przesyłania dużych ilości danych.

Rozważmy teraz poszczególne etapy przetwarzania strumienia multimedialnego pomijając procedurę zrządzania usługami złożonymi (patrz rozdz. 3.1), bierzemy pod uwagę tylko warstwę zadań obliczeniowych. Pamiętając jednocześnie, że dla każdej z kamer generujących strumienie takie przetwarzanie wykonywane jest niezależnie:

1. Strumień jest transmitowany przez kamerę z wykorzystaniem protokołu RTPS poprzez sieć TCP/IP.
2. Strumień jest odbierany przez serwer zarządzania strumieniami, gdzie następuje jego archiwizacja i transmisja do zadania dekodowania.
3. Zadanie dekodowania rozpakowuje kolejne obrazy strumienia wideo i przesyła je do zadań detekcji twarzy i zapisu zdjęć.
4. Zadanie detekcji twarzy odnajduje twarze na poszczególnych obrazach przetwarzanego

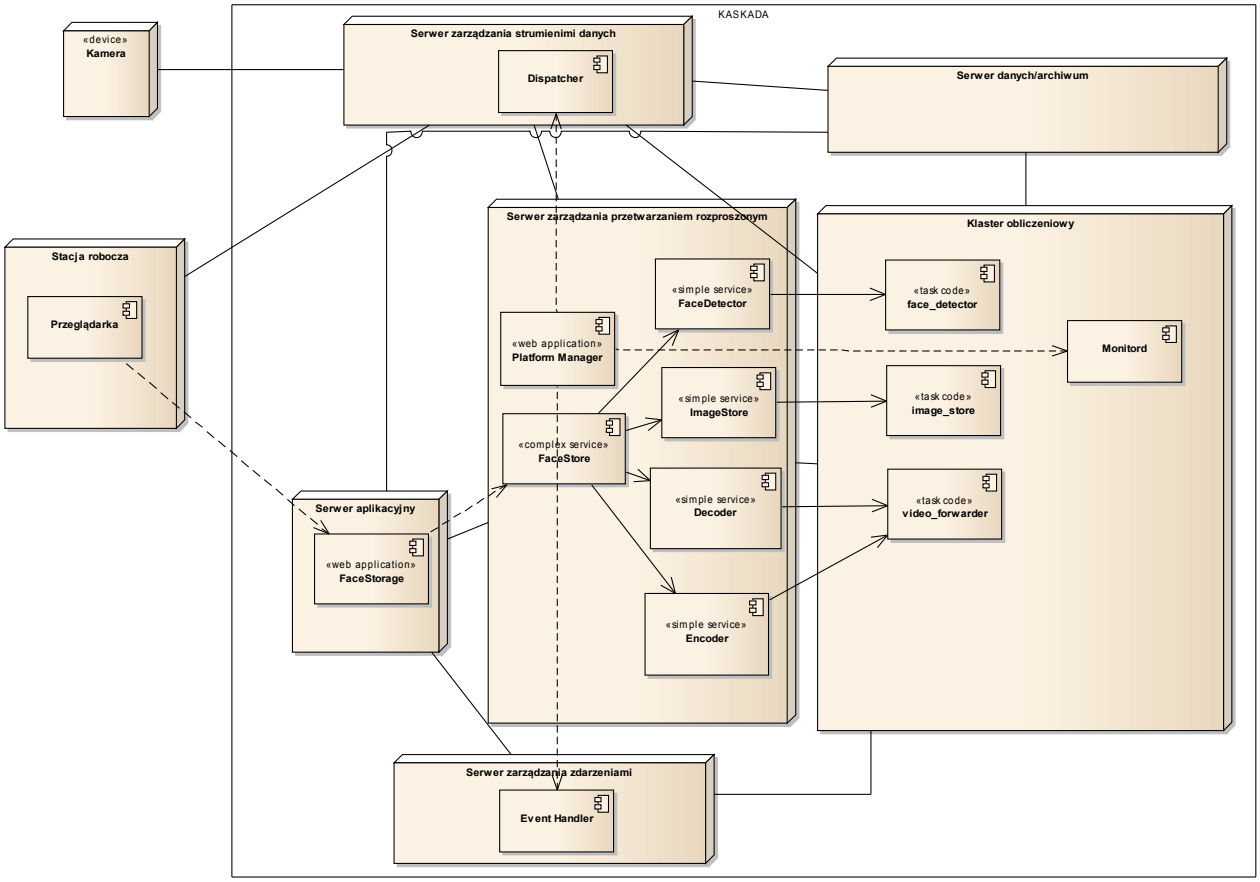
strumienia i wysyła wiadomości do zadania zapisu zdjęć, modyfikuje także strumień wideo – oznaczając rozpoznane obiekty i wysyła je do zadania kodującego jako swój strumień wyjściowy.

5. Zadanie zapisywania zdjęć odbiera strumień wideo i wykorzystując dane z wiadomości odnajduje wykryte twarze i zapisuje je w postaci statycznych obrazów jpeg do archiwum zdjęć.
6. Zadanie kodowania odbiera nieskompresowany strumień wideo od zadania detekcji twarzy i koduje go za pomocą kodeka H.264 jako strumień wyjściowy, który jest transmitowany z powrotem do serwera zarządzania strumieniami.
7. Serwer zarządzania strumieniami umożliwia transmisję już zakodowanego strumienia do użytkownika bądź zapisuje go w archiwum.

Dla wyżej wymienionych zadań należy rozpatrywać sposób przetwarzania, odpowiednią konfigurację platformy, a także strategię zarządzania przetwarzaniem w tym alokację zadań do węzłów. Rozpatrzymy to w kolejnych podrozdziałach.

3.5 Konfiguracja środowiska przetwarzania

Na rys. 22 został przedstawiony sposób wdrożenia aplikacji *Faces* na platformie KASKADA. Użytkownik wykorzystuje przeglądarkę internetową na swojej stacji roboczej, która przedstawia graficzny interfejs użytkownika aplikacji webowej osadzonej na serwerze aplikacyjnym. Aplikacja webowa bezpośrednio uruchamia usługę złożoną *FaceStore* komunikując się z Menedżerem



Rysunek 22: Rozmieszczenie komponentów platformy KASKADA oraz aplikacji Faces na poszczególnych serwerach.

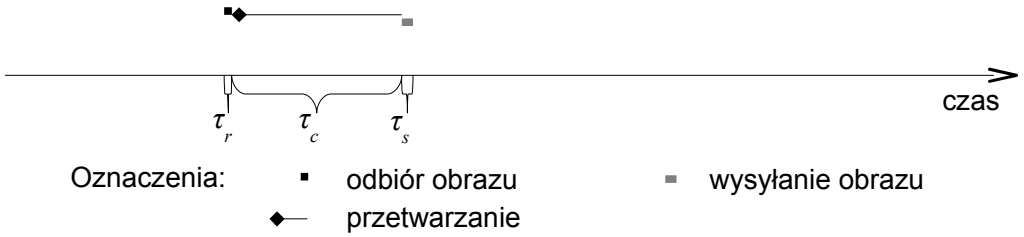
Platformy (Platform Manager) osadzonymi na serwerze zarządzania przetwarzaniem rozproszonym. Komponent platformy Dyspozytor (Dispacher) jest osadzony na serwerze zarządzania strumieniami multimedialnymi skąd może zarządzać archiwum znajdującym się na serwerze danych, natomiast komponent Obsługi zdarzeń (Event Handler) znajduje się na serwerze zarządzania zdarzeniami. Kod zadań obliczeniowych aplikacji jak również kontrolujący je komponent platformy: Rezydentny proces monitorujący (Monitord) są zainstalowane na klastrze obliczeniowym.

Strumień wideo pochodzący z kamery, jest odbierany przez serwer zarządzania strumieniami oraz archiwizowany na serwerze danych/archiwum. Równolegle jest na bieżąco przekazywany do zadania dekodującego, który zgodnie ze strukturą usługi złożonej, zdefiniowaną w MSP-ML'u (rys. 20) przekazuje go dalej. Strumień wynikowy jest przesyłany z powrotem do serwer zarządzania strumieniami, który umożliwia jest przekazanie do bezpośrednio do użytkownika, nawet z pominięciem serwera aplikacyjnego. Otrzymane „fotografie" twarzy – fragmenty obrazów z kolejnych klatek strumienia, są przechowywane w systemie plików na serwera danych i są dostępne dla użytkownika poprzez aplikację webową.

Podczas eksperymentów dotyczących oceny wykonywania tego typu aplikacji zostało wykorzystane środowisko klastra Galera z Centrum Informatycznego TASK na Politechnice Gdańskiej [task]. Serwery: aplikacyjny, zarządzania strumieniami danych, zarządzania przetwarzaniem rozproszonym zostały utworzone wykorzystując po jednym węźle obliczeniowym, natomiast klaster obliczeniowy wykorzystywał kolejne węzły Galery, których liczba zmieniała się w zależności od potrzeb. Ponadto serwer danych działał z wykorzystaniem szybkiego systemu plików LUSTRE, a całość została połączona szybką siecią Infiniband.

3.6 Wybór modelu przetwarzania równoległego

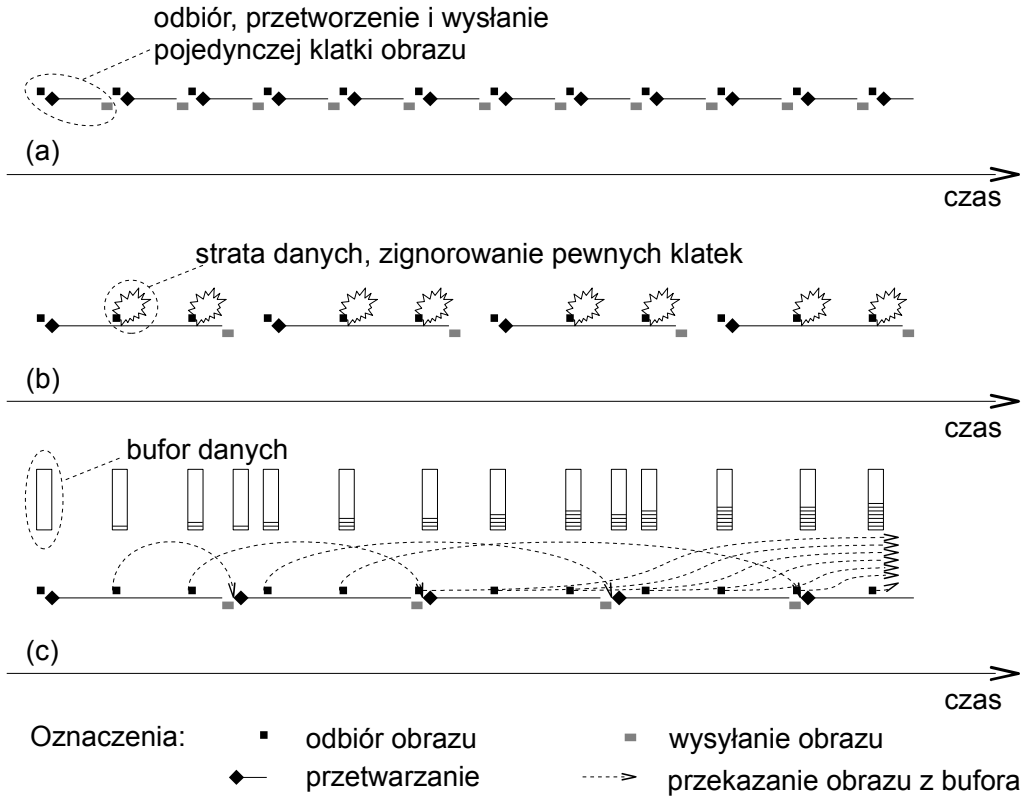
Rozważmy typowy model przetwarzania danych: na początku odbierane są dane wejściowe, np. analizowana klatka obrazu, rys. 23. Jak łatwo zauważyć, na początku ładowane są dane – klatka obrazu (w przedziale czasu: τ_r), następnie wykonywane obliczenia – analiza i przetwarzanie danych zgodnie z założeniami algorytmu (przedział czasu: τ_c), aż w końcu otrzymany w wyniku zmodyfikowaną klatkę obrazu, która jest wysyłana (przedział czasu: τ_s) do miejsca przeznaczenia.



Rysunek 23: Sposób przetwarzania pojedynczej klatki obrazu, τ_r – przedział czasu w jakim jest wykonywana operacja odbioru danych wejściowych: klatki obrazu, τ_c – przedział czasu w jakim jest klatka obrazu jest przetwarzana, τ_s – przedział czasu w jakim jest wykonywana operacja wysłania danych wyjściowych: przetworzonej klatki obrazu.

W przypadku przetwarzania strumienia multimedialnego na jednym węle o danej częstotliwości f pojawiania się obrazów na sekundę (fps, ang. frames per second), aby wykonać analizę na bieżąco – z zachowaniem wymagań czasu rzeczywistego, algorytm przetwarzania musi zakończyć obliczenia przed odbiorem kolejnej klatki obrazu (rys. 24a), czyli $\tau_r+\tau_c+\tau_s \leq 1/f$. W przeciwnym przypadku, zbyt długie obliczenia mogłoby spowodować stratę danych (rys. 24b). Możliwym rozwiązaniem jest buforowanie, jednak przy ciągłym przekraczaniu czasu przetwarzania w stosunku do częstotliwości, spowodowałoby poważne opóźnienie w przetwarzaniu, a w przypadku długich

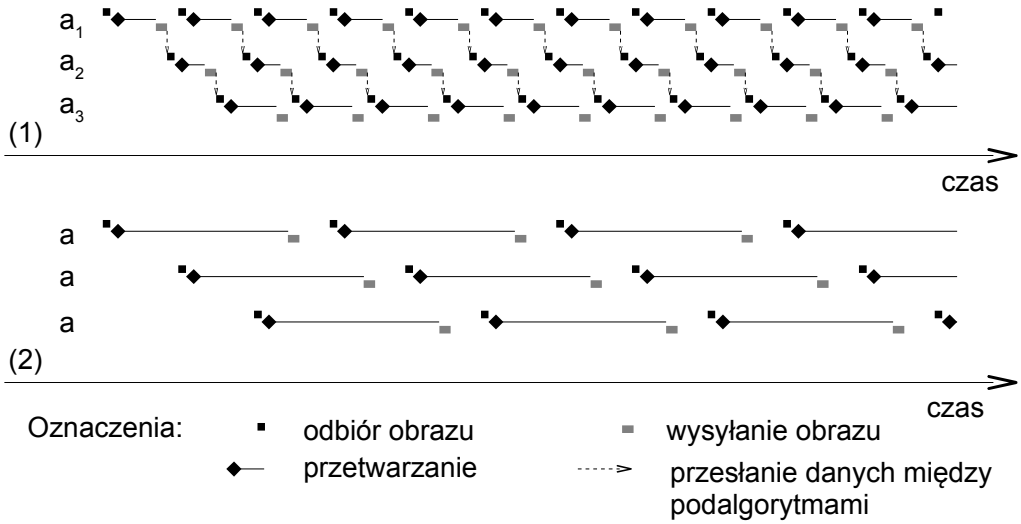
strumieni multimedialnych także przepełnienie bufora (rys. 24c).



Rysunek 24: Sekwencyjne przetwarzanie strumienia obrazów w czasie rzeczywistym na jednym węźle: (a) bez straty danych, (b) ze stratą danych, (c) z buforowaniem danych.

Skutecznym rozwiązaniem powyższego problemu jest zastosowanie przetwarzania równoległego. Możliwe jest wykorzystanie dwóch podejść, z rys. 25:

- dekompozycja funkcjonalna – polegająca na podziale algorytmu obliczeń na podalgorytmy i wykorzystanie przetwarzania potokowego (1) oraz

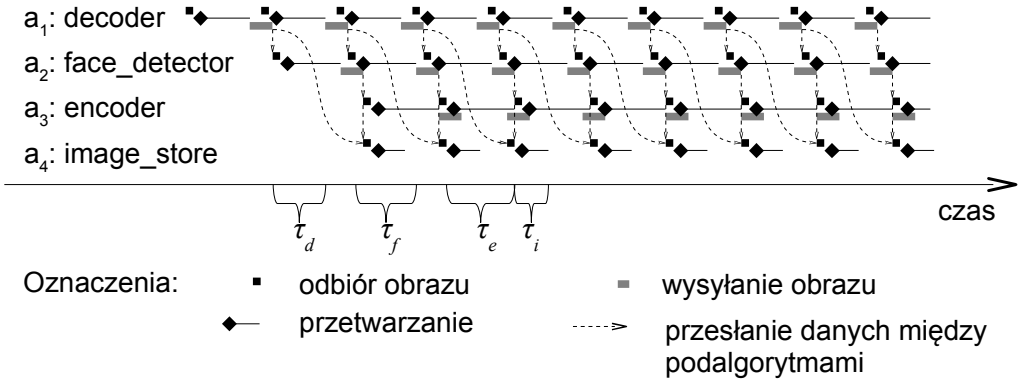


Rysunek 25: Równoległe przetwarzanie pojedynczego strumienia obrazów w czasie rzeczywistym: (1) dekompozycja funkcjonalna algorytmu: a *na podalgorytmy* a_1, a_2 i a_3*, (2) dekompozycja danych i wykonywanie na nich całego algorytmu:* a.

- dekompozycja danych – kiedy dane, jedna lub więcej klatek obrazu są przekazywane do przetwarzających je równolegle zadań realizujących jednakowe algorytmy (2).

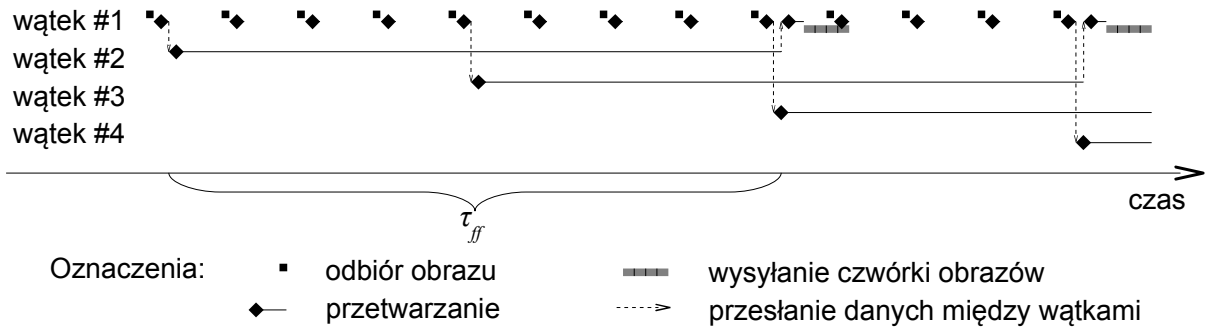
W rozważanym przykładzie wykorzystujemy obydwie metody zrównoleglenia. Dzięki dekompozycji funkcjonalnej wydzielone zostały usługi dekodowania strumienia wejściowego, detekcji twarzy, zapisu do bazy oraz kodowania obrazów strumienia wyjściowego. Poniżej podano czasy wykonywania obliczeń dla poszczególnych zadań (por. rys. 26):

- *decoder (video_forwarder)* τ_d = 81ms,
- *face_detector* τ_f = 40ms (po wewnętrznym zrównolegleniu: dekompozycji danych),
- *encoder (video_forwarder)* τ_e = 46ms,
- *image_store* τ_i = 15ms.



Rysunek 26: Równoległe przetwarzanie strumienia obrazów przez usługę złożoną (potencjalnie na wielu węzłach): FaceStore, z wykorzystaniem dekompozycji funkcjonalnej algorytmu na 4 podalgorytmy zadań obliczeniowyh (t_1-t_4) wykonujących programy: decoder (video_forwarder), face_detector, encoder (video_forwarder) i image_store (ze względu na przejrzystość pominięto opóźnienie wprowadzane przez zadanie face_detector), przedziały czasu przetwarzania przez algorytmy: τ_d – decoder, τ_f – face_detector, τ_e – encoder, τ_i – image_store.

Należy zauważyć, że dekompozycja została wykonana na poziomie zadań, które mogą być wykonywane na oddzielnych węzłach klastra, dlatego pomiędzy poszczególnymi zadaniami należy przesyłać dane w postaci komunikatów (paradygmat *message-passing*), co powoduje dodatkowy narzut komunikacyjny (wysyłanie i odbiór danych).

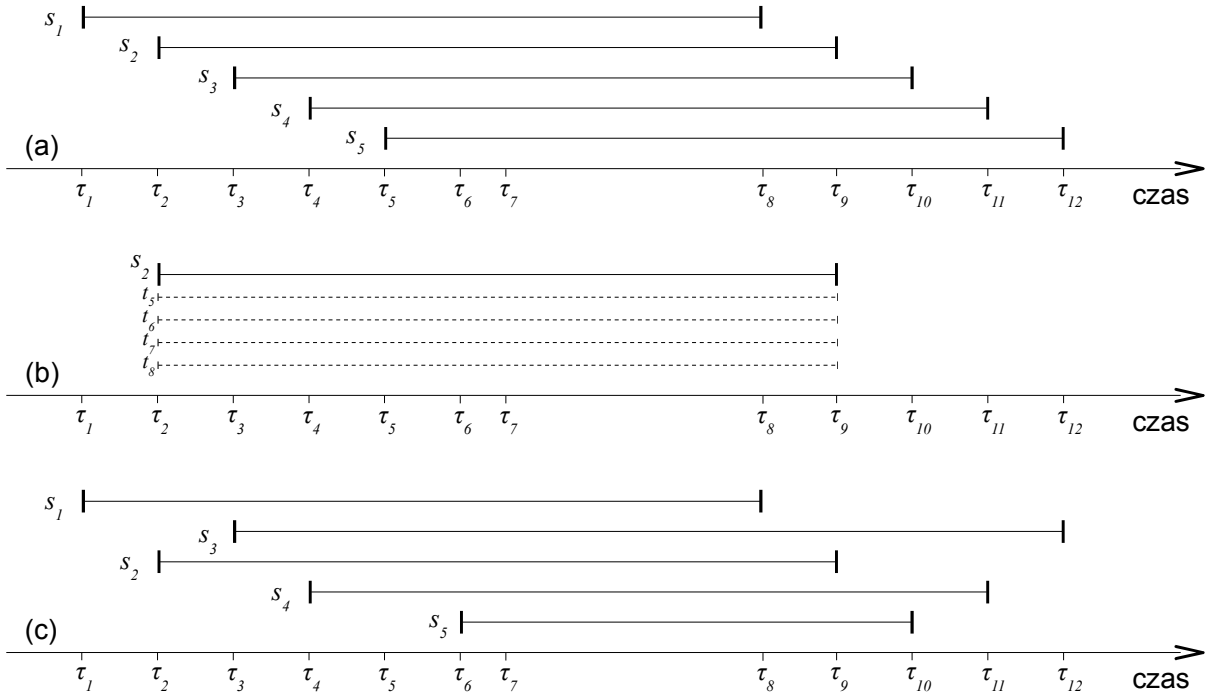


Rysunek 27: Równoległe przetwarzanie strumienia obrazów przez zadanie realizujące program face_detector na jednym węźle z wykorzystaniem dekompozycji danych na 4 wątki: 1 zarządzający (#1) i 3 przetwarzające (#2-4). Przedział czasu τ_{ff} – detekcja twarzy na pojedynczej klatce obrazu.

Dekompozycja danych została wykorzystana natomiast wewnątrz zadania rozpoznawania twarzy

(*FaceDetector*), gdzie poszczególne klatki strumienia wideo są rozdzielane na cztery wątki wykonujące algorytm rozpoznawania twarzy. Działają one na jednym węźle, z wykorzystaniem pamięci współdzielonej (paradygmat *shared-memory*). Dzięki temu podejściu nie ma narzutów czasowych związanych z przekazywaniem danych.

Na rys. 27 został przedstawiony przebieg działania w czasie zadania wykonującego kod *face_detector*. Pierwszy wątek, zarządzający zajmuje się odbiorem i wysyłaniem danych, natomiast pozostałe 3 wątki wykonują obliczenia: analizę klatek obrazu. Czas wykonywani analizy pojedynczej klatki obrazu (τ_{ff} = 400ms) jest znacznie dłuższy od okresu odbioru klatek obrazów (50ms), dlatego zdecydowano się na wykorzystanie 3 wątków oraz na wykonywanie analizy tylko na co czwartej klatce obrazu. Wadą powyższego rozwiązania jest wprowadzenie opóźnienia (8 klatek czyli 400ms) oraz nierównomierne odstępy między wysyłanymi klatkami (klatki wysyłane są w grupach po cztery).

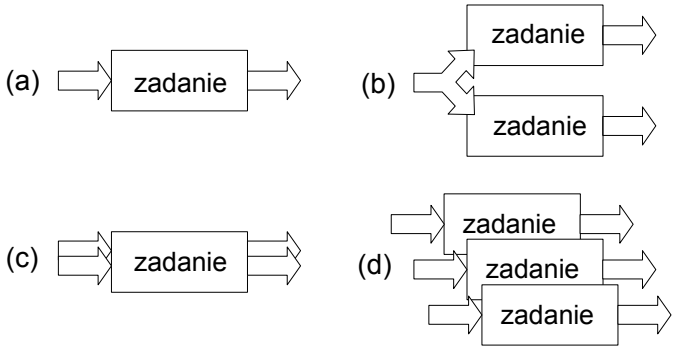


Rysunek 28: Równoległe przetwarzanie usług: (a) usługi złożone o jednolitym czasie wykonania, (b) zadania usług prostych dla pojedynczej usługi złożonej, (c) usługi złożone o różnych czasach wykonania.

Rozważmy teraz model równoległości, przy większej ziarnistości, z punktu widzenia wywoływania usług złożonych. Dla analizowanego przykładu uruchomienie usług odbyło się sekwencyjnie z aplikacji webowej, co spowodowało, że kolejne usługi były wywoływane z pewnym opóźnieniem (rys. 28a). Jednak usługi proste i odpowiadających im zadania: t_5-t_8 na rys. 28b realizujące daną usługę złożoną s_2 były uruchamiane równolegle, tak aby stworzyć działający potok. Ponadto, należy zauważyć, że dla usług złożonych, czas oraz kolejność wykonywania może być różna w zależności od potrzeb użytkownika, tak jak zostało zaprezentowane na rys. 28c.

Na rys. 29 zostały przedstawione modele przetwarzania strumieni z punktu widzenia dystrybucji danych. Najprostszym podejściem jest przetwarzanie pojedynczego strumienia przez jedno zadanie (rys. 29a), następnie ten sam pojedynczy strumień może być powielany i przetwarzany równolegle przez wiele zadań (rys. 29b), można także wykorzystać model kiedy wiele strumieni jest przetwarzanych przez jedno zadanie (rys. 29c) oraz przypadek najbardziej złożony kiedy wiele strumieni jest wykorzystywanych przez wiele zadań (rys. 29d). Ten ostatni przypadek został

przyjęty w omawianym przykładzie.



Rysunek 29: Modele przetwarzania z punktu widzenia dystrybucji danych: (a) jeden strumień – jedno zadanie, (b) jeden strumień – wiele zadań, (c) wiele strumieni – jedno zadanie, (d) wiele strumieni – wiele zadań.

3.7 Zarządzanie wykonywaniem aplikacji Faces

Jak już sygnalizowano **obliczenia** na platformie KASKADA są reprezentowane przez zadania, które w rozważanym przykładzie, działają według następującego kodu (implementacja algorytmów przetwarzania strumieni): *video_forwarder* dla usług *Encoder* i *Decoder*, *face_detector* dla usługi *FaceDetector*, oraz *image_store* dla usługi ImageSore (patrz rys. 21). Każde z takich zadań działa w pętli, powtarzanej według szablonu: (i) odbiór obrazu, (ii) analiza obrazu, (iii) modyfikacja obrazu, (iv) wysłanie obrazu. Całość przetwarzania jest określona przez scenariusz usługi złożonej (rys. 20).

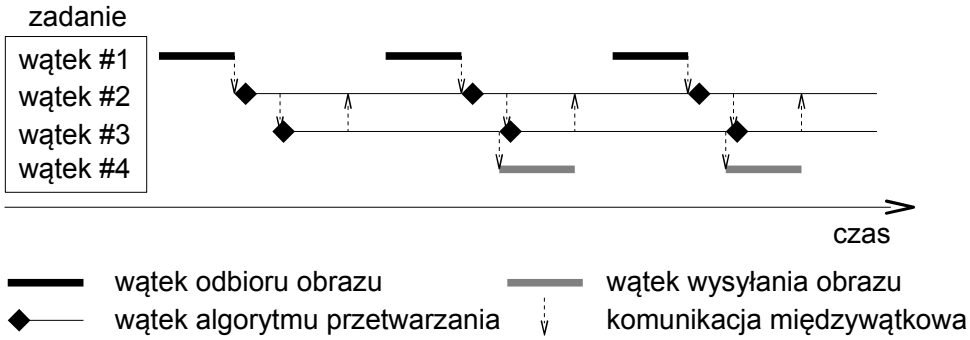
Przesyłane **dane** – strumienie multimedialne (obrazy) oraz komunikaty zdarzeń są skojarzone z zadaniami – każde zadanie może przyjmować wiele strumieni wejściowych oraz generować wiele strumieni wyjściowych. Na przykład zadanie $t_1=(Decoder, \{si_1\}, \{so_1, so_2\}, 0{,}0375)$, dostaje na wejście jeden strumień: si_1, a generuje na wyjściu dwa strumienie: so_1, so_2 (wiele strumieni – jedno zadanie, por. rys.29c). Strumień si_1 jest strumieniem typu PAL, co implikuje typ strumieniowy zadania $v(t_1)=PAL$ i ponieważ jest jedynym strumieniem wejściowym to stopień strumieniowy zadania $\varrho_S(t_i)=1$.

Innym parametrem, oprócz typu algorytmu i strumieni danych, określającym zadanie, jest obciążenie węzła obliczeniowego powodowane jego wykonywaniem dla zadanych danych bez udziału innych zadań. Ponieważ zakładamy użycie homogenicznego klastra obliczeniowego, obciążenie to jest identyczne dla wszystkich węzłów, a jego pomiar jest wykonywany doświadczalnie na wyizolowanym węźle, przy znormalizowaniu wartości obciążenia do przedziału (0, 1>, gdzie 1 oznacza całkowite obciążenie wszystkich rdzeni węzła.

Współpraca zadań możliwa jest dzięki **komunikacji** zewnętrznej obejmujący multimedialny strumień wejściowy, wyjściowy i komunikaty dotyczące zdarzeń oraz dzięki komunikacji wewnętrznej – między realizowanymi zadaniami. Platforma KASKADA dostarcza mechanizmów przesyłania danych, które są przezroczyste dla programisty i dotyczą odbioru i wysyłania obrazów i są wykonywane przez specjalny *framework* dostarczony wraz z platformą zwany ramką KASKADA. Należy podkreślić, że dzięki zastosowaniu mechanizmu wątków komunikacja odbywa się równolegle z obliczeniami. Na przykładzie z rys. 30 wątki #1 i #4 są kontrolowane właśnie przez ramkę KASKADA i są odpowiednio odpowiedzialne za odbiór i wysyłanie klatek obrazów, natomiast wątki #2 i #3 są typowymi wątkami obliczeniowymi przeprowadzającymi analizę tych strumieni.

Zadania w zależności od liczby przesyłanych danych potrzebują odpowiednich **zasobów** aby mogły być wykonywane. W naszym przypadku jest to wspomniany klaster obliczeniowy C składający się ze zbioru homogenicznych węzłów, np. $C = \{c_1, c_2, c_3\}$, por. rys. 22. Każde z zadań może być

osadzone na jednym z jego węzłów, a każdy węzeł może obsługiwać jedno lub więcej zadań. Stan węzła zależy od działających na nim zadań w danej chwili τ, których zbiór jest określany za pomocą funkcji $\omega^{\tau}(c_i)$, np. $\omega^{\tau}(c_1) = \{t_1, t_2\}$. Zadania realizowane na platformie KASKADA mają określony stopień i typ strumieniowy, jak również ich wykonywane obciąża węzeł c_1, np.: $\gamma(c_1, \omega^{\tau}(c_1)) = \gamma(c_1, \{t_1, t_2\}) = 0{,}78$.



Rysunek 30: Przykład przetwarzania równoległego w czterowątkowym zadaniu obliczeniowym, z uwzględnieniem komunikacji międzywątkowej, na platformie KASKADA.

Węzły klastra zostały przebadane w celu określenia wpływu przetwarzania dużej liczby zadań wykorzystujących odpowiednią liczbą strumieni multimedialnych, jak to opisano w rozdz. 4 . Ustalono, że obciążenie węzła wzrasta nieliniowo dla zwiększającej się liczby zadań przetwarzających strumienie i tak dla nowych zadań t_3 i t_4: $\gamma(c_1, \{t_1, t_2, t_3, t_4\}) > \gamma(c_1, \{t_1, t_2\}) + \gamma(c_1\{t_3, t_4\})$.

Dla pojedynczego typu strumieniowego zadań (*PAL*) oszacowano obciążenie oraz ustalono, że zadania obsługujące strumienie HD powinny być wykonywane pojedynczych węzłach (jedno zadanie *hd* – jeden węzeł), a zadania audio (*aud*) oraz zadanie nieprzetwarzające strumieni multimedialnych (*nil*) mogą być umieszczane zarówno z zadaniami typu *pal* jak i *hd*. Funkcja korekty $\eta_h(n)$ wyznaczająca przyrost obciążenia została wyznaczona oraz zweryfikowana w eksperymentach w rozdz. 4 .

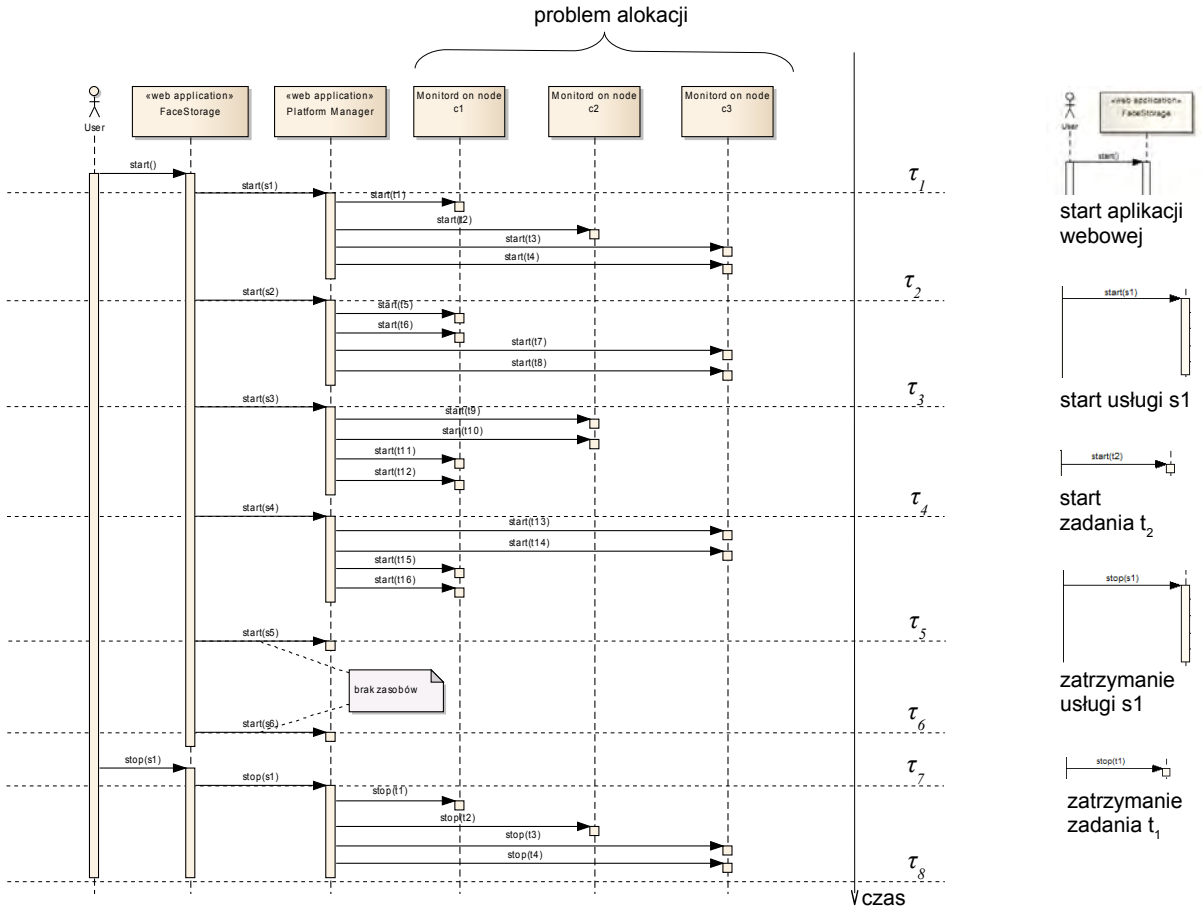
3.8 Przebieg działania aplikacji Faces

Użytkownicy platformy KASKADA zgłaszają swoje żądania realizacji usług poprzez aplikacje użytkowe, robią to w sposób asynchroniczny: rozpoczęcie przetwarzania może nastąpić w dowolnym momencie, tak samo jak zlecenie zakończenia jednej z aktualnie wykonywanych usług. Z każdą działającą usługą skojarzone są zadania wykonywane na węzłach obliczeniowych. Platforma KASKADA dobiera je odpowiednio w zależności od wymaganej funkcjonalności i strumieni wejściowych. Przykładowe zadania wykonujące obliczenia dla aplikacji Faces zostały przedstawione w tabeli 3.

Tabela 3: Przykładowe zadania uruchamiane przez aplikację Faces.

nazwa kodu zadania	usługa złożona	identyfikator uruchomionego zadania	definicja zadania	*stopień strumieniowy zadania: $\varrho_S(t)$*	*uwagi*
video_forwarder	s1	t_1	(video_forwarder, {si_1}, {so_1, so_2}, 0,0375)	1	dekoder wideo
face_detector	s1	t_2	(face_detector, {si_2}, {so_3}, 0,3)	1	
image_store	s1	t_3	(image_store, {si_3}, ø, 0,0125)	1	koder wideo
video_forwarder	s1	t_4	(video_forwarder, {si_4}, {so_4}, 0,2)	1	
video_forwarder	s2	t_5	(video_forwarder, {si_5}, {so_5, so_6}, 0,0375)	1	dekoder wideo
face_detector	s2	t_6	(face_detector, {si_6}, {so_7}, 0,3)	1	
image_store	s2	t_7	(image_store, {si_7}, ø, 0,0125)	1	koder wideo
video_forwarder	s2	t_8	(video_forwarder, {si_8}, {so_8}, 0,2)	1	

Na rys.31 został przedstawiony scenariusz uruchomienia czterech usług złożonych *FaceStore*: *s1-s4*, próby uruchomienia usług *s5* i *s6*, a następnie zatrzymania usługi *s1*. Każda z tych usług realizowana jest przez cztery zadania, odpowiadające jej usługom prostym. W tabeli 3 zostały przedstawione parametry tych zadań dla dwóch pierwszych usług, obciążenia przez nie generowane wahają się od 0,0125 do 0,3, wszystkie zadania mają po jednym strumieniu wejściowym oraz od 0 do 2 strumieni wyjściowych. Zakładamy użycie typowego dla systemów rozproszonych algorytmu alokacji zapewniającego równoważenie obciążenia (ang. load-balancing) [watts98], w którym kolejno zgłaszane zadania są uruchamiane na najmniej obciążonym węźle, przy założeniu braku możliwości migracji zadań oraz przyjmując, że zadana są niepodzielne.



Rysunek 31: Przykład przetwarzania równoległego usług i zadań na platformie KASKADA.

Usługa *s1* jest uruchomiona jako pierwsza. W momencie τ_1 wszystkie trzy węzły klastra są puste

($\omega^{\tau 1}(c_i)$ zwraca zbiór zadań wykonywanych na węźle c_i) – nieobciążone żadnymi zadaniami ($\gamma(c_i, T)$ określa obciążenie węzła c_i przez zadania ze zbioru T):

$\omega^{\tau 1}(c_1) = \emptyset$, $\gamma(c_1, \omega^{\tau 1}(c_1)) = 0$,
$\omega^{\tau 1}(c_2) = \emptyset$, $\gamma(c_2, \omega^{\tau 1}(c_2)) = 0$,
$\omega^{\tau 1}(c_3) = \emptyset$, $\gamma(c_3, \omega^{\tau 1}(c_3)) = 0$.

Po uruchomieniu zadań ze zbioru T w momencie τ_2 działają już na nich 4 zadania związane z usługą *s1*: t_1, t_2, t_3, t_4, tak więc:

$\omega^{\tau 2}(c_1) = \{t_1\}$, $\gamma(c_1, \omega^{\tau 2}(c_1)) = 0{,}0375$,
$\omega^{\tau 2}(c_2) = \{t_2\}$, $\gamma(c_2, \omega^{\tau 2}(c_2)) = 0{,}3000$,
$\omega^{\tau 2}(c_3) = \{t_3, t_4\}$, $\gamma(c_3, \omega^{\tau 2}(c_3)) = 0{,}2040$.

Jak łatwo zauważyć, zgodnie z założeniami algorytm alokacji rozdziela wszystkie zadania na wszystkie węzły, równoważąc ich obciążenie.

Następnie uruchamiane są zadania związane z usługą *s2* i tym razem algorytm równoważy obciążenie, powodując zwiększenie obciążenia na węzłach c_1 i c_3:

$\omega^{\tau 3}(c_1) = \{t_1, t_5, t_6\}$, $\gamma(c_1, \omega^{\tau 3}(c_1)) = 0{,}3567$,
$\omega^{\tau 3}(c_2) = \{t_2\}$, $\gamma(c_2, \omega^{\tau 3}(c_2)) = 0{,}3000$,
$\omega^{\tau 3}(c_3) = \{t_3, t_4, t_7, t_8\}$, $\gamma(c_3, \omega^{\tau 3}(c_3)) = 0{,}4149$.

Kolejna uruchomiona usługa: *s3* powoduje zwiększenie obciążenia węzłów c_1 i c_2 o następne 4 zadania:

$\omega^{\tau 4}(c_1) = \{t_1, t_5, t_6, t_{11}, t_{12}\}$, $\gamma(c_1, \omega^{\tau 4}(c_1)) = 0{,}5825$,
$\omega^{\tau 4}(c_2) = \{t_2, t_9, t_{10}\}$, $\gamma(c_2, \omega^{\tau 4}(c_2)) = 0{,}6063$,
$\omega^{\tau 4}(c_3) = \{t_3, t_4, t_7, t_8\}$, $\gamma(c_3, \omega^{\tau 4}(c_3)) = 0{,}4149$.

Ostatnia usługa, dla której udało się alokować zasoby: *s4* powoduje, że algorytm alokacji, umieszcza po 2 zadania na węźle c_1 i c_3, co implikuje następujący stan klastra w momencie τ_5:

$\omega^{\tau 5}(c_1) = \{t_1, t_5, t_6, t_{11}, t_{12}, t_{15}, t_{16}\}$, $\gamma(c_1, \omega^{\tau 5}(c_1)) = 0{,}8351$,
$\omega^{\tau 5}(c_2) = \{t_2, t_9, t_{10}\}$, $\gamma(c_2, \omega^{\tau 5}(c_2)) = 0{,}6063$,
$\omega^{\tau 5}(c_3) = \{t_3, t_4, t_7, t_8, t_{13}, t_{14}\}$, $\gamma(c_3, \omega^{\tau 5}(c_3)) = 0{,}7683$.

W tym momencie (τ_5) na węźle c_1 zauważa się nieliniowy przyrost obciążenia, tzn.:

$\gamma(c_1, \{t_1, t_5, t_6, t_{11}, t_{12}, t_{15}, t_{16}\}) > \gamma(c_1, \{t_1\})+\gamma(c_1, \{t_5\})+\gamma(c_1, \{t_6\})+\gamma(c_1, \{t_{11}\})+\gamma(c_1, \{t_{12}\})+\gamma(c_1, \{t_{15}\})+\gamma(c_1, \{t_{16}\})$

$0{,}8351 > 0{,}0375+0{,}0375+0{,}3+0{,}0125+0{,}2+0{,}0125+0{,}2$

$0{,}8351 > 0.8000$

Dotychczasowe obciążenie klastra jest już na tyle duże, że przy założonym algorytmie alokacji próba uruchomienia kolejnej usług: *s5* i *s6* kończy się porażką – ich wywołanie zwraca błąd z informacją o braku zasobów.

W momencie τ_7 użytkownik zdecydował się na wyłączenie usługi *s1*, co spowodowało zatrzymanie związanych z nią zadań: t_1, t_2, t_3, t_4 na węzłach c_1, c_2 i c_3, co przyczyniło się do tego, że w momencie τ_8 stan klastra był następujący:

$\omega^{\tau 7}(c_1) = \{t_5, t_6, t_{11}, t_{12}, t_{15}, t_{16}\}$, $\gamma(c_1, \omega^{\tau 7}(c_1)) = 0{,}7683$,
$\omega^{\tau 7}(c_2) = \{t_9, t_{10}\}$, $\gamma(c_2, \omega^{\tau 7}(c_2)) = 0{,}3241$,
$\omega^{\tau 7}(c_3) = \{t_7, t_8, t_{13}, t_{14}\}$, $\gamma(c_3, \omega^{\tau 7}(c_3)) = 0{,}5370$.

Rozważmy obecnie próbę ponownego uruchomienia usługi *s6*. Byłaby ona udana, jeśli jej

wykonanie mogłoby być przełożone do momentu czasu τ_8.

Powyższy algorytm alokacji może być wykorzystany również do planowania uruchomienia zadań w przyszłości. Załóżmy, że zadania oprócz czasu realizacji a priori wskazuje również moment jego startu i zakończenia. To wymaga rozszerzenia definicji zadania do następującej:

$t_i=(a_i, SI_i, SO_i, \gamma_i, \tau_{si}, \tau_{fi})$

gdzie: τ_{si} to moment startu zadania t_i, a τ_{fi} to moment jego zakończenia, $\tau_E = \tau_{fi}-\tau_{si},$. W takim przypadku można przewidzieć jakie zadnia mają być wykonywane w poszczególnych przedziałach czasu i uruchamiać je dopiero wtedy kiedy będzie to wymagane. Wówczas można wykorzystać istniejące dotychczas algorytmy szeregowania [el-rewini04], uwzględniając jednak nieliniowy przyrost obciążenia na węzły obliczeniowych.

4 Charakterystyka węzła obliczeniowego

Dokonano oceny możliwości przetwarzania strumieni multimedialnych na pojedynczym węźle obliczeniowym platformy KASKADA w kontekście wykorzystania modeli przetwarzania oraz odpowiadających im algorytmów analizy. Opisano architekturę węzła, strukturę strumieni danych oraz kategorie zadań obliczeniowych wykonywanych na węźle. Zaprezentowano metodę oceny charakterystyk wydajnościowych i wiarygodnościowych węzła, ze względu na rozmiar strumieni multimedialnych oraz złożoność zadań wykonujących analizę. Przedstawiono wyniki eksperymentów oraz podano zasady efektywnego wykorzystywania węzła w przypadku równoległego przetwarzania strumieni multimedialnych.

4.1 Badane charakterystyki węzła obliczeniowego

Głównym zadaniem platformy KASKADA jest przetwarzanie strumieni multimedialnych w trybach on-line i off-line, patrz rozdz. 3.1 . Usługi on-line działają bezpośrednio na strumieniach pochodzących z urządzeń rejestrujących obraz i dźwięk. Ze względu na wymagania klienta zadania obliczeniowe je realizujące możemy kwalifikować jako zadania czasu rzeczywistego (ang. real-time), a całą platformę jako system czasu rzeczywistego.

Przez **wydajność** węzła rozumieć będziemy zdolność do przetwarzania wymaganej liczby zadań, bądź inaczej wykonywanych na nim zadań. W niniejszym rozdziale rozważamy następującą metrykę związane z wydajnością: **obciążenie obliczeniowe** (definicja 5). Przez wiarygodność funkcjonowania węzła będziemy rozumieć poziom zaufania odnośnie wyników otrzymywanych przez działające algorytmy lub odpowiadające im zadania. W niniejszym rozdziale rozważamy następującą metrykę związane z wiarygodnością: **strata danych** (definicja 7). Zgodnie z przyjętymi założeniami zintegrowanego modelu zarządzania (por. rozdz. 3.1), rozważamy kompleksowo wpływ przetwarzanych danych, wymaganej komunikacji oraz strategi alokacji zasobów na wydajność obliczeń, przy dopuszczalnym poziomie wiarygodności wyników obliczeń.

4.1.1 Standardowe benchmarki węzła obliczeniowego

Istnieje wiele różnych testów wydajności – benchmarków dla systemów komputerowych umożliwiających ocenę i porównanie badanych systemów między sobą, a nawet tworzenie list najszybszych komputerów. Najbardziej popularną listą tego typu jest lista TOP500 [top500], przedstawiająca ranking superkomputerów umieszczonych w centrach obliczeniowych. Do oceny mocy obliczeniowej wykorzystywany jest benchmark Linpack [linpack].

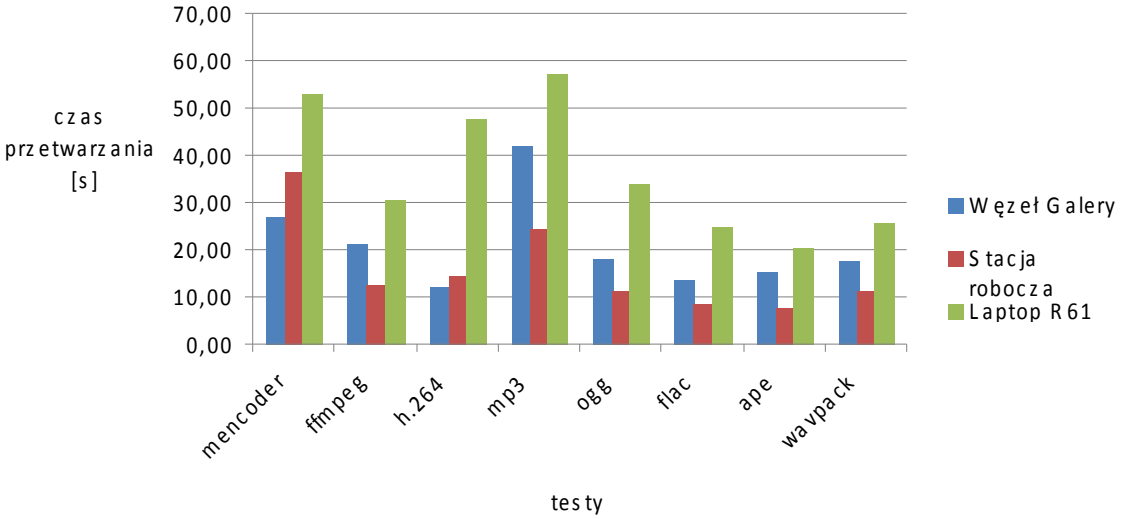
Inne, często używane benchmarki to Dhrystone [weiss02] i Fhourstones [fhourstones], umożliwiający ocena wydajności obliczeń dla liczb całkowitych, Iometer [iometer] określający szybkość działania operacji wejścia/wyjścia, zbiór testów NAS parallel benchmarks [bailey94] określający wydajność systemu dla przetwarzania równoległego, POV-Ray [povray] oceniający możliwości systemu dla generowania grafiki 3D, czy Whetstone [whetstone] dla obliczeń zmiennoprzecinkowych. Tabela 4 przedstawia wyniki porównawcze wybranych benchmarków badanego węzła.

Ze względu na specyfikę przetwarzania platformy KASKADA, wykorzystany został benchmark Phoronix Test Suite [pts] oceniający wydajność kodowania strumieni audio i wideo. Jest to jedyny popularny test mierzący wydajność przetwarzania strumieni multimedialnych, reprezentowaną przez czas przetwarzania – kodowania nieskompresowanego filmu wideo oraz nagrania dźwiękowego za pomocą kolejnych algorytmów koderów: mpeg(mencoder), mpeg(ffmpeg), h.264, mp3, ogg, flac, ape i wave(wavpack).

Tabela 4: Wyniki standardowych benchmarków wydajności węzła.

benchmark	wynik
Linpack	58,69 GFlops
Whetstone	35714.3 MIPS
POV-Ray	17,52 s
NAS:CG	975,67 MFlops
NAS:EP	226,90 MFlops
NAS:FT	2507,75 MFlops
NAS:IS	146,91 indeksów
NAS:LU	1903,83 MFlops
NAS:MG	2039,71 MFlops

Przykładowe wyniki porównania wykorzystywanego węzła obliczeniowego (procesor: 2 x Intel Xeon CPU E5345 @2.33GHz (Total Cores: 8), pamięć 8GB) dla przykładowych algorytmów kodujących w porównaniu z dwoma innymi systemami – typowym komputerem stacjonarnym (procesor: Intel Core i5 CPU 650 @3.19GHz (Total Cores: 4), pamięć: 4GB) i laptopem R61 (procesor: Intel Core 2 Duo CPU T5550 @1.83GHz (Total Cores: 2), pamięć: 2GB) zostały przedstawione na wykresie z rys. 32. Ze względu na słabe zrównoleglenie badanych algorytmów w wykorzystywanych bibliotekach benchmarkowych, niektóre wyniki były lepsze dla stacji roboczej posiadającej tylko pojedynczy procesor, ale działający z wykorzystaniem szybszego zegara (stacja: 3.19GHz – węzeł: 2.33GHz).



Rysunek 32: Wyniki testów wydajnościowych – algorytmy kodowania strumieni multimedialnych, dla pojedynczego węzła klastra, typowej stacji roboczej i laptopa R61. Benchmarki przeprowadzono na tych samych danych, a wyniki podano w sekundach.

Brak popularnych benchmarków umożliwiających badanie wydajności mierzonej jako obciążenie procesorów węzła dla przetwarzania strumieni multimedialnych odbieranych przez sieć (on-line), spowodował iż konieczne było utworzenia własnej procedury oceny charakterystyk wydajnościowych. Powinna ona brać pod uwagę różne rodzaje strumieni multimedialnych (audio i wideo o różnych rozdzielczościach), jak i różne typy przetwarzania – nie tylko kodowanie, ale również inne często wykorzystywane algorytmy analizy, np. detekcja twarzy, maska tła. Należy przy tym określić precyzyjnie co rozumiemy przez strumień danych, węzeł obliczeniowy, obciążenie obliczeniowe, sposób doboru benchmarków oraz metodę realizacji pomiarów. Propozycja takiego testu została przedstawiona w kolejnych podrozdziałach.

4.1.2 Struktura danych

Platforma KASKADA zorientowana jest na optymalizację wykorzystania **danych** multimedialnych. Zadania przetwarzające strumień audio lub wideo otrzymują odpowiednio zestawy próbek dźwięku lub klatki obrazu i analizują je zgodnie ze wskazanym algorytmem. Przy czym odpowiedni algorytm może być wykorzystywany do przetwarzania **on-line**, kiedy strumień przekazywany jest bezpośrednio z urządzeń (np. kamer lub sprzętu medycznego), jak również **off-line**, gdy wykorzystywane jest nagranie zapamiętane w archiwum.

Definicja 1: S**trumień danych** to skończony ciąg niezależnych elementów danych przybywających kolejno na wejście $si=(si^1, si^2, \ldots si^{|si|})$ lub wychodzących $so=(so^1, so^2, \ldots so^{|so|})$ stanowiący wynik przetwarzania ciągu wejściowego przez konkretne zadanie obliczeniowe. Gdzie $|si|$ i $|so|$ są liczbami elementów ciągu wejściowego i wyjściowego.

Typowym przykładem strumienia wejściowego są dane wideo odbierane z serwera zarządzania strumieniami, za pomocą protokołu RTSP [schulzrinne98], dekodowane przez zadanie obliczeniowe i przesyłane w formacie rozpakowanym jako strumień wynikowy. Zgodnie z założeniami dot. opisu danych z rozdz. 2.2 , wyróżniamy następujące charakterystyki strumieni multimedialnych:

- rozmiar: liczba MB (megabajtów) jakie zajmują wszystkie elementy strumienia, może być wyrażony jako:
    - iloczyn liczby elementów i ich wielkości (MB);
    - iloczyn **zajmowanego pasma sieciowego** wykorzystywanego do transmisji danych (MB/s) i **czasu trwania** strumienia (sekundy);
    - iloczyn **częstotliwości przesyłania elementów** strumienia (np. fps dla strumieni wideo, ang. frames per second), czasu trwania strumienia (s) i wielkości elementów (MB);
- struktura: strumień jest ciągiem niezależnych **elementów** jednego **typu**: klatek obrazu albo próbek dźwięku,
- ziarnistość: jest określana przez **rozmiar** (MB) **pojedynczego elementu** strumienia, np. im mniejsza klatka obrazu tym mniejsza ziarnistość – czyli strumień PAL ma mniejszą ziarnistość niż strumień HD.

Na tej podstawie określamy typy strumieni multimedialnych.

Definicja 2: Zbiór typów strumieni multimedialnych $H=\{hd, pal, aud, nil\}$ gdzie typy strumieni są następujące: wideo HD (*hd*), PAL (*pal*), audio (*aud*) oraz komunikaty o zdarzeniach (*nil*). Typ strumienia *sx* oznaczmy jako $\theta(sx), \theta(sx) \in H,$ gdzie *sx* może być strumieniem wejściowym *si* lub wyjściowym *so*.

Tabela 5: Charakterystyki strumieni testowych wykorzystanych w badaniach wydajności węzła.

charakterystyka/typ strumienia	wideo PAL	wideo HD	audio
zajmowane pasmo sieciowe	29,0MB/s	183,5MB/s	358kB/s
czas trwania	600s	600s	600s
typ elementu	obraz 704x576	obraz 1920x1080	ok. 1000 próbek audio
częstotliwości przesyłania elementów (fps)	21,7 na sek.	30,9 na sek.	38,3 na sek.
rozmiar elementu	1,16MB	5,93MB	9kB

W tabeli 5 zostały przedstawione podstawowe charakterystyki przykładowych strumieni multimedialnych.

Charakterystyki strumienia danych zawierającego ciąg komunikatów o zdarzeniach (*nil*) są wysoko

zależne od algorytmu wykrywania zdarzeń – generujących komunikaty, jednakże zakładamy, że jego rozmiar jest pomijalny w stosunku do strumieni multimedialnych wejściowych i wyjściowych.

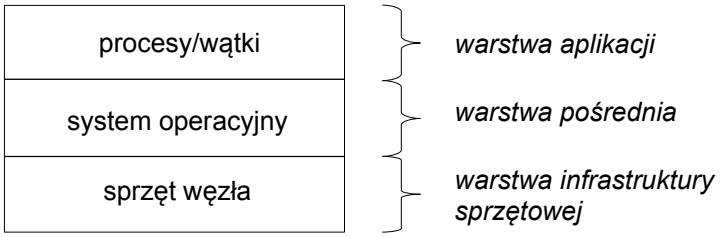
Zbiór typów strumieni multimedialnych *H* jest uporządkowany malejąco według następującego porządku: *hd* > *pal* > *aud* > *nil*. Ten porządek jest odzwierciedleniem intuicyjnego klasyfikowania typów strumieni pod względem zajmowanego pasma sieciowego.

4.1.3 Model węzła obliczeniowego

Klaster obliczeniowy jest to równoległy system komputerowy składający się z połączonych ze sobą **węzłów obliczeniowych** wyposażonych w procesory, pamięć operacyjną i urządzenia umożliwiające komunikację wewnątrz klastra [czech10]. W dalszych rozważaniach będziemy brali pod uwagę klastry obliczeniowe, dla których wszystkie węzły są reprezentowane przez komputery o jednakowej architekturze, o jednakowych zasobach sprzętowych i posiadają ten sam system operacyjny. Innymi słowy rozpatrujemy homogeniczne klastry obliczeniowe, które są najczęściej oferowane na rynku przez producentów komputerów dużej mocy.

Typowym przykładem takiego klastra obliczeniowego jest superkomputer Galera zainstalowany na Politechnice Gdańskiej w Centrum Informatycznym Trójmiejskiej Akademickiej Sieci Komputerowej (CI TASK) [task]. Posiada on 672 węzły obliczeniowe połączone szybką siecią lokalną: Infiniband w topologii *fat tree* o przepływności nominalnej 20Gb/s. Wspólna pamięć masowa, dostarczana jest przez sieciowy system plików LUSTRE [lustre] z serwera plików o pojemności 500TB.

Definicja 3: Klaster obliczeniowy definiujemy jako zbiór homogenicznych węzłów obliczeniowych $C=\{c_1, c_2, \ldots c_{|C|}\}$, gdzie $|C|$ jest liczbą węzłów klastra.



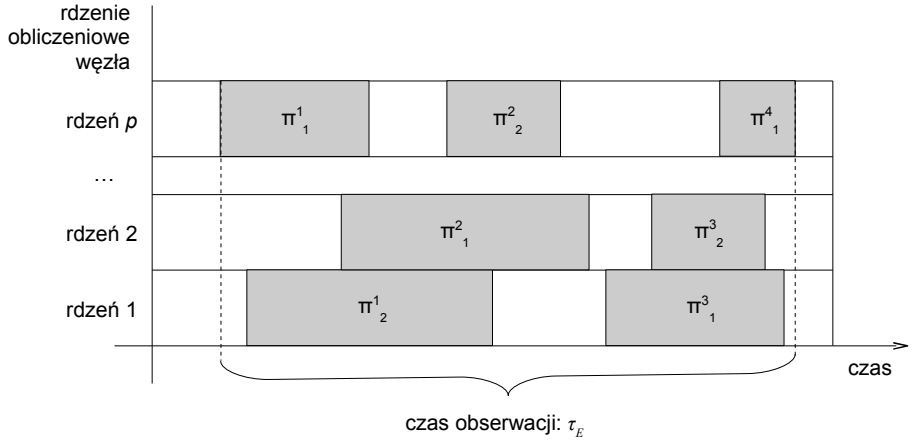
Rysunek 33: Model warstwowy węzła obliczeniowego.

Sam węzeł obliczeniowy może być autonomicznie analizowany jako system komputerowy (rozdz. 2.1), który posiada strukturę warstwową, patrz rys. 33. **Warstwa sprzętu węzła** składa się z dwóch 64-bitowych procesorów Intel Xeon, każdy z procesorów z czterema rdzeniami obliczeniowymi (Quad Core) – 8 rdzeni na węzeł, taktowane zegarem o częstotliwości 2,33GHz, z pamięcią podręczną L2 o pojemności 12MB w architekturze EM64T. Ponadto wyposażona jest w pamięć operacyjną o wielkości 8GB współdzieloną przez rdzenie obu procesorów, kartę sieciową Infiniband 20Gb/s oraz lokalny dysk twardy o pojemności 160GB. **Warstwa procesów** jest odpowiedzialna za realizację zadań obliczeniowych platformy KASKADA. Każdy z procesów posiadają swój własny wydzielony obszar pamięci oraz **wątki** działające równolegle w celu wykonania zadanego algorytmu. Możliwa jest także współpraca międzyprocesowa z wykorzystaniem typowych mechanizmów systemu Unix, takich jak kolejki, semafory itp. [posix]. **System operacyjny** jest odpowiedzialny za bezpośrednią współpracę ze sprzętem, do jego zadań należy zarządzanie procesami w tym ich szeregowanie i współpraca z innymi węzłami. Należy podkreślić, że na poziomie procesów i wątków modelu warstwowego platformy KASKADA (patrz rys. 11), to właśnie system operacyjny jest odpowiedzialny za szeregowanie procesów na rdzenie obliczeniowe procesorów. Każdy z rdzeni ma swoją kolejkę procesów oczekujących na kwant czasu. Długość przydzielonego czasu jest bezpośrednio zależna od priorytetu procesu i domyślnie

dla komputerów klasy PC, a także rozpatrywanych węzłów klastra obliczeniowego wynosi ok. 210ms. W przypadku jeśli żaden z procesów nie wymaga czasu procesora, to znajduje się on w stanie bezczynności (ang. idle).

Definicja 4: Zadanie obliczeniowe $t_i=(a_i, SI_i, SO_i, \gamma_i)$ definiujemy jako pojedynczy proces wykonujący algorytm a_i analizujący wejściowe strumienie danych: $SI_i=\{si_{i1}, si_{i2}, \ldots si_{i|SIi|}\}$, oraz generujący strumienie wynikowe $SO_i=\{so_{i1}, so_{i2}, \ldots so_{i|SOi|}\}$. Poza tym zadanie t_i wymaga odpowiednich zasobów obliczeniowych: γ_i potrzebnych do wykonywania algorytmu a_i, γ_i jest określone jako obciążenie obliczeniowe węzła klastra C przy wykonaniu na nim tylko zadania t_i.

Działające zadania obciążają węzeł w różny sposób powodując wykorzystanie części lub całości mocy obliczeniowej procesorów węzła (ang. processor utilization) [liu73]. Liczba i długość strumieni wyjściowych SO_i zależy od algorytmu zadania i danych wejściowych tzn. zbioru przetwarzanych strumieni SI_i. Przy założeniu poprawnego wykonania algorytmu zadania – bez żadnych strat związanych z brakiem zasobów obliczeniowych, zbiór SO_i będziemy również nazywać **pożądanym zbiorem strumieni wyjściowych** odpowiadającym danemu zbiorowi strumieni wejściowych SI_i i wykorzystywanemu algorytmowi a_i.



Rysunek 34: Przykład przydziału rdzeni obliczeniowych w czasie obserwacji τ_E, dla dwóch zadań, gdzie kolorem szarym oznaczono przedziały aktywności rdzeni gdzie dla zadania 1: $\Pi_1=\{\pi^1_1, \pi^2_1, \pi^3_1, \pi^4_1\}$ oraz dla zadania 2: $\Pi_2=\{\pi^1_2, \pi^2_2, \pi^3_2\}$. Pozostałe przedziały oznaczone kolorem białym odpowiadają czasom bezczynności rdzeni.

Rozpatrzmy działanie pojedynczego węzła obliczeniowego c, posiada on określoną liczbę p rdzeni obliczeniowych (p=8 w naszym przypadku) oraz wspólną dla nich pamięć operacyjną (8GB). Zakładamy również, że węzeł wyposażony jest w dwukierunkowe łącze sieciowe o określonej maksymalnej przepływności, jak również posiada system operacyjny umożliwiający uruchamianie wielowątkowych zadań obliczeniowych, przełączający procesy w sposób równomierny – mają one ten sam priorytet. W wyniku działania algorytmu szeregowania procesów (realizujących zadania obliczeniowe) na rdzeniach procesorów następuje przydział czasu rdzeni procesorów. Na każdym z tych rdzeni obserwujemy aktywność obliczeniową, zbiór takich okresów aktywności konkretnego zadania t_i oznaczamy przez $\Pi_i=\{\pi^1_i, \pi^2_i, \ldots \pi^{|\Pi i|}_i\}$. Aktywności obliczeniowe nawet w ramach jednego zadania mogą zachodzić równolegle dzięki wykorzystaniu wątków działających na kilku rdzeniach obliczeniowych węzła. Rys. 34 przedstawia przykładowe szeregowania 2 procesów w czasie obserwacji τ_E.

Definicja 5: Obciążenie obliczeniowe węzła c przez zbiór zadań T oznaczamy przez $\gamma(c, T)$ i definiujemy jako:

$$\gamma(c,T)=\frac{\tau r(c,T)}{p\cdot\tau_E} \quad (4)$$

gdzie τ_E ($\tau_E>0$) jest całkowitym czasem przetwarzania zadań, p liczbą rdzeni obliczeniowych, a $\tau r(c, T)$ jest sumą wszystkich przedziałów czasu aktywności, w których rdzenie węzła c wykonywały obliczenia dla zadań ze zbioru T:

$$\tau r(c,T)=\sum_{i=1}^{|T|}\sum_{j=1}^{|\Pi_i|}\pi_i^j \quad (5)$$

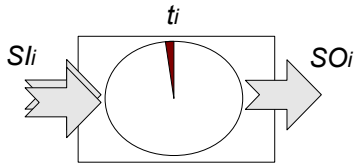
Łatwo zauważyć, że w przypadku gdy wszystkie rdzenie obliczeniowe węzła przez cały czas przeznaczony na obliczenia to $\gamma(c, T)=1$, natomiast jeśli w całości przeznaczony są na oczekiwanie to $\gamma(c, T)=0$.

Zgodnie z definicją 4, γ_i jest wyznaczane gdy na węźle c realizowane jest tylko zadanie t_i, inaczej kiedy $T=\{t_i\}$. Wówczas:

$$\gamma_i=\gamma(c,\{t_i\})=\frac{\tau r(c,\{t_i\})}{p\cdot\tau_E} \quad (6)$$

Obciążenie obliczeniowe jest często używane do oceny algorytmów szeregowania dla twardych systemów czasu rzeczywistego. W zależności od liczby i częstotliwości żądań obliczeń w zadaniach, można określić maksymalne obciążenie (ang. utilization bound), generowane przez zadany zbiór zadań czasu rzeczywistego, dla którego dany system może być wykorzystany (ang. schedulability), np. [liu73, oh95, chen03, qi11].

Przyjmując założenie, że strumienie wejściowe są przetwarzane na bieżąco przez zadania obliczeniowe, rys. 35 przedstawia przykładową reprezentacje graficzną pojedynczego zadania t_i analizującego dwa strumienie danych i generującego jeden strumień wyjściowy, ciemnym kolorem zaznaczone zostało obciążenie γ_i węzła c jakie generowane jest przez to zadanie, przy założeniu, że jest ono jedynym zadaniem wykonywanym na tym węźle. Z rysunku wynika, że $\gamma(c, \{t_i\})$=0,05.



Rysunek 35: Przykładowa reprezentacja graficzna zadania t_i przetwarzającego dwa strumienie wejściowe $SI_i=\{si_{i1}, si_{i2}\}$ i zwracjącego jeden wyjściowym $SO_i=\{so_{i1}\}$.

Definicja 6: Strata danych wyjściowych w zadaniu $t_i \in T$ na węźle c przy obciążeniu zbiorem zadań T oznaczamy jako $\varphi(c, t_i, T)$, i definiujemy jako stosunek liczby straconych elementów strumieni wyjściowych zadania t_i podczas wykonania zadań ze zbioru T na węźle c do liczby wszystkich elementów wyjściowych strumieni, które powinny być generowane przez zadanie t_i:

$$\varphi(c,t_i,T)=1-\frac{\sum_{j=1}^{|SO_i^R|}|so_{ij}^R|}{\sum_{k=1}^{|SO_i|}|so_{ik}|} \quad (7)$$

gdzie $SO_i^R=\{so^R_{i1}, so^R_{i2}, \ldots so^R_{i|SO^R_i|}\}$ jest uzyskanym zbiorem strumieni wyjściowych dla rzeczywistego wykonania zadania t_i na węźle c, w którym mogła nastąpić strata danych wyjściowych, zaś $SO_i=\{so_{i1}, so_{i2}, \ldots so_{i|SOi|}\}$ jest pożądanym zbiorem strumieni wyjściowych.

Na ogół liczba strumieni jest zachowana $|SO_i^R|=|SO_i|$, zaś straty dotyczą tylko ich elementów.

Definicja 7: Strata danych na węźle c przy obciążeniu zbiorem zadań T oznaczamy jako $\varphi(c, T)$,

i definiujemy jako średnią stratę danych wszystkich zrealizowanych zadań ze zbioru T:

$$\varphi(c,T)=\frac{\sum_{i=1}^{|T|}\varphi(c,t_i,T)}{|T|} \quad (8)$$

Tak więc stratę danych wyjściowych określa się po wykonaniu zadań alokowanych na węźle w całym przedziale ich realizacji τ_E.

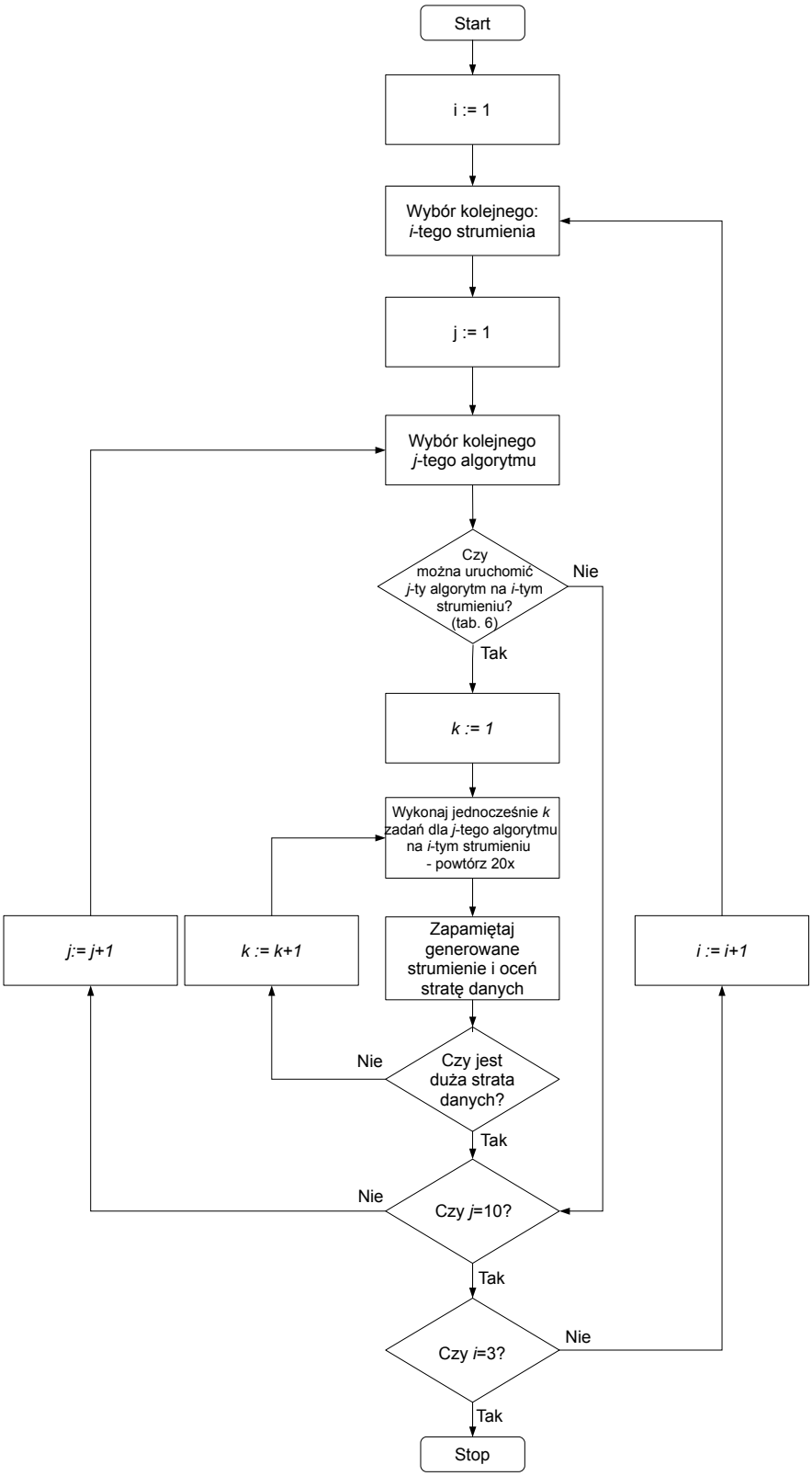
4.1.4 Procedura oceny charakterystyk wydajnościowych i wiarygodnościowych węzła obliczeniowego

Na rys. 36 została przedstawiona procedura wykonywania badań oceny charakterystyk wydajnościowych. Jest ona tak skonstruowana, aby określić parametry obciążenia węzła i ewentualnej straty danych przy zwiększającym się obciążeniu węzła dla badanych zadań testowych.

Pierwszym krokiem procedury jest wybór pierwszego strumienia testowego. Platforma KASKADA jest używana do przetwarzania różnych danych, zaczynając od danych tekstowych, poprzez statyczne obrazy, audio, a na wideo PAL i HD kończąc. Na platformie KASKADA spośród wymienianych typów do przetwarzania w czasie rzeczywistym najczęściej wykorzystuje się następujące strumienie multimedialne: wideo HD, PAL oraz audio. W tabeli 5 zostały przedstawione ich podstawowe charakterystyki (por. rozdz. 2.2). Są one również często wykorzystywane przez aplikacje użytkowe i właśnie dla nich zostały wykonane kolejne badania.

Jak wynika z tabeli 6 wybór pierwszego algorytmu testującego jest zależny od rodzaju badanego strumienia. Przyjęto następujące kryteria wyboru algorytmów:

1. Kryterium funkcjonalności: nie każdy algorytm może być wykonany na dowolnym strumieniu multimedialnym ze względów funkcjonalnych, np. detekcja twarzy jest możliwa jedynie na strumieniu wideo, natomiast np. zadanie zliczające liczbę elementów strumienia może obsługiwać wykorzystywać dowolny rodzaj strumienia.
2. Kryterium jakości: jeśli format strumienia danych jest akceptowalny przez implementowany algorytm (kryterium funkcjonalności), nie zawsze pojedynczy węzeł jest w stanie wykonać zaplanowaną analizę i przetwarzanie, np. dotychczasowe doświadczenia wykazały, że przy założonym sprzęcie niemożliwe jest wykonanie detekcji twarzy, na co 4-tej klatce strumienia wideo HD.
3. Kryterium klasy: wybrane algorytmy muszą należeć do założonej klasy algorytmów, tzn. są zaprojektowane i zaimplementowane według schematu z rys. 9. Ich główny rdzeń, pętla odbioru elementów strumieni multimedialnych jest w każdym przypadku taka sama, natomiast różnią się ona procedurą która jest wykonywana w kroku *Przetwarzanie elementu strumienia*. Zarówno cechy funkcjonalne jak i jakościowe tej procedury zależą bezpośrednio od projektanta algorytmu. Dlatego wybrane algorytmy testujące powinny różnić się między sobą zarówno rodzajem przetwarzania, np. detekcja twarzy na obrazie czy maska tła, jak i jego jakością, np. częstością co którą klatkę przetwarzany jest strumień, czy jak dokładnie rozpoznawane są kształty.



Rysunek 36: Procedura realizacji badań dla oceny charakterystyk wydajnościowych i wiarygodnościowcyh węzła obliczeniowego.

W wyniku wykorzystania powyższych kryteriów wybrano algorytmy reprezentatywne dla platformy KASKADA, które tworzą rozpatrywaną **klasę algorytmów** (patrz rozdz. 2.7) o następujących cechach:

1. Mają różne wymagania co do niezbędnej mocy obliczeniowej.
2. Są charakterystyczne dla przetwarzania strumienie multimedialnych.
3. Wykorzystują typowe procedury i funkcje biblioteki OpenCV – otwartej (opensource) i często wykorzystywanej do implementacji algorytmów analizy strumieni multimedialnych.
4. Zachowują różnorodność ze względu na zaprojektowaną funkcjonalność dostarczając różnych rodzajów analizy i modyfikacji strumienia jak i na uwzględnienie poziomu jakości umożliwiając sterowanie częstością wykonywania analizy.

Do badań wykorzystano następujące algorytmy:

1. Przekaźnik: odbiera strumień wideo i przekazuje dalej bez zmian – nie wykonując żadnych obliczeń – algorytm pusty, wykorzystywany ze względów referencyjnych.
2. Zegar: odbiera strumień wideo, dodaje etykietę z aktualnym czasem i przekazuje dalej, obliczenia, wykonywane przez pojedynczy wątek, dla pojedynczej klatki obrazu PAL zajmują: 1,5ms oraz dla obrazu HD: 13ms.
3. Maska tła: odbiera strumień wideo, analizuje poszczególne klatki obrazu rozpoznając jakie jego elementy są ruchome, a co jest tłem, po czym tworzy nowy obraz z zaznaczoną maską ruchomych elementów, obliczenia dla pojedynczej klatki obrazu PAL zajmują: 25ms oraz dla obrazu HD: 160ms.
4. Detekcja twarzy: odbiera strumień wideo, odnajduje twarze na co *k*-tej klatce obrazu, gdzie *k* jest parametrem wejściowym regulującym jakość analizy zgodnie z 3-cim kryterium wyboru algorytmów, następnie zaznacza miejsce wystąpienia wykrytego obiektu i przekazuje zaznaczony obraz dalej, obliczenia są wykonywane wielowątkowo, obliczenia dla pojedynczej klatki obrazu PAL zajmują: 400ms oraz dla obrazu HD: 980ms.
5. Scalenie obrazów: odbiera 2 strumienie wejściowe, składa je w jeden jako dwie połowy obrazu, ze skalowaniem do rozdzielczości strumieni wejściowych, obliczenia wykonywane przez pojedynczy wątek, dla pojedynczej klatki obrazu PAL zajmują: 4,4ms oraz dla obrazu HD: 32ms.
6. Wykrywanie krawędzi: odbiera strumień wideo, wykonuje detekcję konturów obrazu na każdej klatce z wykorzystaniem transformaty Laplace'a, następnie umieszcza te kontury na nowych obrazach, które są wysyłane jako strumień wyjściowy, wykonywane przez pojedynczy wątek, obliczenia dla pojedynczej klatki obrazu PAL zajmują: 13ms oraz dla obrazu HD: 53ms.
7. Filtr środkowoprzepustowy Butterworth'a [smith97]: odbiera strumień audio i „obcina" częstotliwości zarówno od góry jak i z dołu, a wynik wysyła jako strumień wyjściowy, wykonywane przez pojedynczy wątek, dla pojedynczego elementu strumienia: 1152 próbki obliczenia zajmują: 0,15ms.
8. Statyczna detekcja twarzy: detekcja twarzy jako program jednowątkowy na danych z dysku lokalnego węzła – bez wykorzystania strumienia danych, złożoność obliczeniowa oraz obciążenie procesora jest takie samo jak dla wersji strumieniowej (punkt 4).

Przytoczone czasy obliczeń zostały wyznaczone dla pojedynczych elementów strumienia (klatki obrazu lub zbioru próbek dźwięku), pobranych bezpośrednio z pamięci operacyjnej bez operacji wejścia-wyjścia z dysku lub sieci.

Tabela 6 przedstawia badane algorytmy w zależności od wykorzystanego strumienia testowego. Algorytm pusty – przekaźnik może być wykonywany dla wszystkich rodzajów strumieni, zegar dla dowolnego strumienia wideo, natomiast bardziej obciążające algorytmy takie jak detekcja twarzy na obrazie, czy wykrywanie konturów mogą być wykonywane tylko na strumieniu wideo PAL. Algorytm filtra częstotliwości może być jedynie wykonywany na strumieniu audio.

W następnym kroku są wykonywane pomiary dla *k* jednocześnie działających zadań na pojedynczym węźle obliczeniowym i wykonujących ten sam kod wybranego algorytmu dla tego samego strumienia testowego. Po dokonaniu pomiarów węzeł jest zerowany – wszystkie działające na nim zadania są kończone. Pomiary w ramach tego kroku są wykonywane 20-krotnie, a następnie uśredniane, tak aby uniknąć wpływu przypadkowych czynników – takich jak np. natężenie ruchu sieciowego w klastrze oraz określić zakres błędu pomiaru. Zmienna iteracyjna *k* początkowo jest równa 1, czyli pierwsze 20 pomiarów jest wykonywane dla jednego zadania, następnie jeśli zostanie podjęta decyzja o kontynuacji to wartości zmiennej *k* zwiększana jest o 1 i wykonane jest kolejne 20 pomiarów, itd.

Tabela 6: Testowane zadania i odpowiadające im strumienie multimedialne.

algorytm/strumień	1. wideo PAL	2. wideo HD	3. audio
1. Przekaźnik	+	+	+
2. Zegar	+	+	-
3. Detekcja twarzy na co 32-giej klatce	+	-	-
4. Detekcja twarzy na co 16-tej klatce	+	-	-
5. Detekcja twarzy na co 4-tej klatce	+	-	-
6. Statyczna detekcja twarzy na co 32-giej klatce	+	-	-
7. Scalenie obrazów	+	-	-
8. Wykrywanie krawędzi	+	-	-
9. Maska tła	+	-	-
10. Filtr częstotliwości	-	-	+

Podczas testów wydajnościowych pojedynczego węzła obliczeniowego wykonywano pomiary następujących charakterystyk:

#1: obciążenie obliczeniowe węzła – wykorzystując program narzędziowy top [top], rejestrowano co sekundę czas bezczynności procesorów (idle), samo obciążenie węzła zostało określone jako procentowy stosunek czasu pracy całego węzła (okres próbkowania (1s) – czas bezczynności) do okresu próbkowania (1s), przy powyższych założeniach obciążenie generowane przez narzędzie jest poniżej 1% pojedynczego rdzenia procesora;

#2: wielkość wykorzystanej pamięci operacyjnej – podobnie jak w przypadku obciążenia procesorów, wykorzystano program top [top], rejestrowano z częstotliwością 1s, procent użytej pamięci przez cały węzeł;

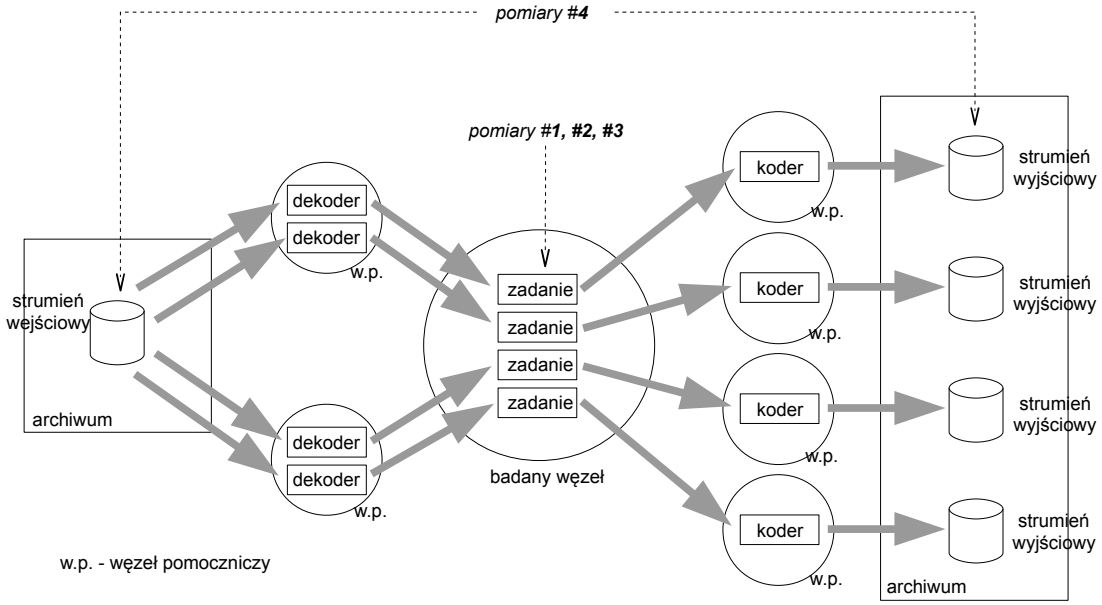
#3: wykorzystywane pasmo sieciowe wejściowe i wyjściowe – wielkość ruchu sieciowego węzła była rejestrowana z częstotliwością 1s, za pomocą programu narzędziowego perfquery, umożliwiającego odczyt statystyk i parametrów bezpośrednio z karty sieciowej Infiniband [inifniband];

#4: strata danych wyjściowych – stosunek procentowy liczby straconych elementów strumienia wyjściowego w czasie eksperymentu do całkowitej liczby elementów strumienia wyjściowego wygenerowanych przez algorytm (patrz definicja 7), wartości charakterystyki były ustalane po wykonaniu pojedynczego testu, na podstawie danych strumieni wejściowych i wynikowych zapisanych w archiwum.

W celu minimalizacji czynników zewnętrznych, podczas badań węzła uruchamiano na nim tylko

zadania testowe, bez udziału innych procesów użytkowych. Rys. 37 przedstawia infrastrukturę systemowo-sprzętową wykorzystywaną podczas przeprowadzanych testów dla przykładowych 4 zadań. Badany strumień wejściowy znajduje się w archiwum strumieni multimedialnych, do którego dostęp mają węzły pomocnicze. Dane strumienia odczytywane są przez dekodery i przesyłane w formie rozpakowanej do zadań testowych. Podczas jednego pomiaru wykorzystywany jest ten sam strumień powielany dla każdego zadania, także inne dane wejściowe, tj. parametry wywołania i zmienne środowiskowe są te same.

Badany węzeł obliczeniowy jest obciążany jedynie zadaniami testującymi, każde z nich wykonuje swój algorytm: analizuje i przetwarza napływające elementy strumienia wejściowego i przesyła wyniki w strumieniu wyjściowym do procesów kodujących. Procesy koderów umieszczone są na osobnych węzłach pomocniczych, wykonują one kompresję strumienia do formatu akceptowanego w archiwum strumieni, gdzie dane są umieszczane. Ważnym elementem jest dobranie właściwej liczby węzłów pomocniczych, tak aby procesy kodujące i dekodujące, nie ograniczały przepływu danych i nie powodowały dodatkowej straty danych.



Rysunek 37: Infrastruktura systemowo-sprzętowa dla pomiarów pojedynczego węzła obliczeniowego. Pomiary: #1 – obciążenie obliczeniowe węzła, #2 – zajętość pamięci operacyjnej, #3 – wykorzystanie pasma sieciowe, #4 – strata danych.

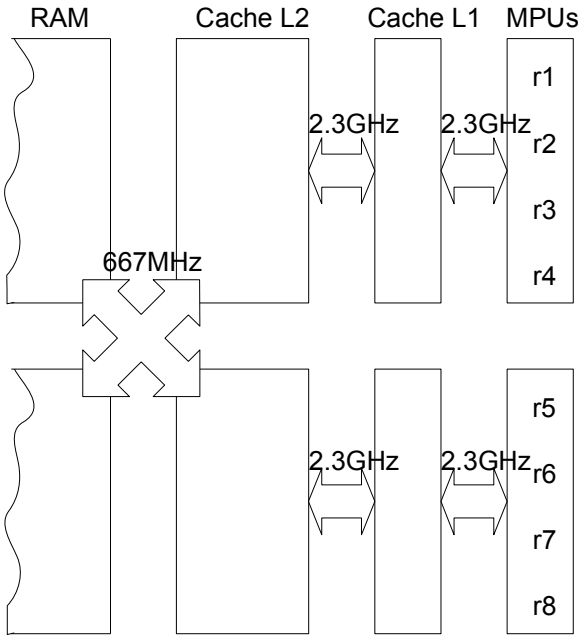
W kolejnym kroku procedury następuje sprawdzenie warunku: czy jest duża strata danych**.** Jeśli wszystkie dane zostały przetworzone przez zadanie, czyli liczba rzeczywista elementów strumienia wyjściowego jest równa liczbie elementów kompletnego i poprawnego strumienia wyjściowego, lub jest nieznacznie mniejsza – przyjęliśmy granicę tolerancji do 1% strat, wówczas następuje decyzja o kontynuacji lub zaprzestaniu pomiarów, dla większej liczby zadań. Jeśli strata jest powyżej przyjętego limitu, to następuje przejście do następnego kroku procedury (patrz rys. 36), w którym sprawdzane jest czy wszystkie dziesięć algorytmów testujących dla obecnego strumienia zostało już zbadanych i jeśli nie, badany jest kolejny algorytm. Jeśli natomiast wszystkie algorytmy testujące dla obecnego strumienia zostały zbadane, to następuje przejście do kroku, w którym sprawdza się czy zbadano wszystkie 3 strumienie testowe, po którym następuje decyzja, czy należy badać kolejny strumień, czy też zakończyć procedurę.

4.1.5 Oczekiwane obciążenia węzła

Na skutek specyficznej budowy węzłów (patrz rozdz. 4.1.3) i wynikającej z niej hierarchii

pamięci podręcznej (ang. cache), patrz rys. 38, możemy się spodziewać, że obciążenie procesorów wielordzeniowych będzie się zwiększać nie tylko na skutek bardziej złożonych obliczeń wykonywanych na danych, ale także poprzez wzmożoną komunikację między poszczególnymi warstwami cache'a (L1 i L2), jak również pamięcią RAM. Podobnie jak w modelu Roofline [williams09], zakładamy, że większa wymiana danych z pamięcią RAM wymusi zmniejszenie maksymalnej mocy obliczeniowej danego procesora. Jest to spowodowane mniejszą przepustowością połączenia z pamięcią RAM (zegar 667MHz) w porównaniu z wewnętrznymi połączeniami w procesorach (zegar 2.3GHz).

Z drugiej strony należy rozważyć typowe zadanie działające na platformie KASKADA, tzw. reprezentatywną klasę algorytmów (por. rozdz. 2.9). Taki algorytm wymaga danych ze strumieni wejściowych, odkodowania ich, wykonania obliczeń, następnie modyfikuje dane – bez dodatkowej alokacji pamięci, koduje je i wysyła. Wymaga to kilkukrotnego przekopiowania pamięci (w czasie odbioru, odkodowywania, kodowania i wysłania), dla każdego przetwarzanego elementu strumienia. Oczekujemy, że kiedy na węźle przetwarzanych jest niewiele strumieni, to wszystkie te operacje mogą być wykonane w pamięci podręcznej (2.3GHz), natomiast kiedy liczba strumieni będzie wzrastać to może to spowodować znaczne zmniejszenie wydajności procesorów węzła, ze względu na konieczność oczekiwania na załadowanie danych z wolnej pamięci RAM (667MHz). Dlatego też możemy się spodziewać, że zależność między liczbą obsługiwanych strumieni oraz zaobserwowanym obciążeniem węzła będzie nieliniowa.



Rysunek 38: Sposób połaczenia wielordzeniowych jednostek obliczeniowych (MPUs) węzła z pamięcią podręczną (ang. cache) L1 i L2 oraz pamięcią RAM.

Rozważmy obciążenie węzła dla zadanego zbioru zadań uruchomionym na węźle, zakładając, że znamy obciążenie obliczeniowe dla każdego z tych zadań w przypadku gdy zadanie posiada pojedynczy strumień wejściowy i generuje pojedynczy strumień wyjściowy oraz tylko to zadanie jest przetwarzane na węźle.

Stawiamy tezę, że po zwiększeniu obciążenia obliczeniowego poprzez zwiększenie złożoności obliczeń wykonywanych na każdym elemencie strumienia multimedialnego (np. klatce obrazu), przyrost wymagań obliczeniowych jest proporcjonalny dla wszystkich zadań przetwarzających tego typu strumień, czyli:

$$\gamma(c,T) \approx \gamma'(c,T) \quad \text{gdzie} \quad \gamma'(c,T) = \eta_h\left(\sum_{i=1}^{|T|} |SI_i|\right) \sum_{i=1}^{|T|} \gamma(c,\{t_i\}) \quad (9)$$

gdzie $\gamma'(c, T)$ jest aproksymowanym obciążeniem, T zbiorem zadań strumieniowych uruchomionych na węźle c, $\eta_h(n)$ jest **funkcją korekty** określaną eksperymentalnie, przyjmującą jako argument liczbę obsługiwanych strumieni oraz jest zależna od ich typu h, gdzie $h \in H$ (definicja 2).

4.2 Wyznaczenie funkcji korekty dla strumieni PAL

Wykonano badania umożliwiające wyznaczenie **funkcji korekty**: $\eta_{pal}(n)$ *na podstawie* wzoru (9), rozpatrując 4 różne konfiguracje zadań dla strumieni PAL. Następnie w podrozdziale 4.3 dokonano weryfikacji tej funkcji przy użyciu innych konfiguracji i algorytmów, również dla strumieni PAL. W podrozdziale 4.4 zbadano zachowanie zadanych algorytmów na strumieniach audio i HD. Badania wykonano zgodnie z procedurą opisaną w rozdz. 4.1.4 . Przetestowano cztery algorytmy przetwarzania strumieni zaczynając od zadań przekaźnika, poprzez zegar generujące niskie obciążenie, aż po złożone rozpoznawanie twarzy realizowane na co 32 i co 16 klatce strumienia wejściowego. Poniżej w kolejnych podrozdziałach zostały przedstawione wyniki z kolejnych eksperymentów. Do każdego z nich przedstawiliśmy krótki opis, cel jego wykonania oraz ważniejsze spostrzeżenia. Dla łatwiejszego odczytu wyników eksperymentów: obciążenie obliczeniowe węzła (definicja 5) oraz strata danych wyjściowych (definicja 8) na wykresach i w tabelach zostały one przedstawione jako wartości procentowe.

Podczas badań, zgodnie z procedurą z rys. 36 pomiary dla każdego zestawu zadań były powtarzane 20-krotnie. Pomiary dotyczące wykorzystania pamięci operacyjnej i przepustowości sieci wykazały, że te parametry nie mają praktycznie wpływu na obciążenie obliczeniowe i stratę danych. Natomiast obciążenie obliczeniowe i strata danych zwiększają się wraz ze wzrostem liczby przetwarzanych strumieni, gdzie odchyłka standardowa dla pomiarów obciążenia obliczeniowego waha się w przedziale 0,0005-0,0040. Przyjmując wartość średnią 0,0017, natomiast odchylenie dla straty danych wyjściowych wynosi ok. 0.0001 dla eksperymentów z małą stratą oraz 0,0030-0,0040 dla eksperymentów z dużą stratą danych. Co oznacza, że istotny wpływ na obciążenie obliczeniowe węzła przy wykonywaniu zadań z algorytmami o niskiej złożoności (przekaźnik i zegar) ma system operacyjny. Dlatego do wyznaczenia funkcji korekty wzięto pod uwagę pomiary obciążenia obliczeniowego dla małej liczby strumieni przy wykonaniu złożonych algorytmów zadań oraz dla dużej liczby strumieni przy wykorzystaniu mniej złożonych algorytmów zadań.

4.2.1 Strumień PAL: Przekaźnik

Eksperyment umożliwia badanie zachowania się węzła obliczeniowego bez obciążenia obliczeniowego, a jedynie z ruchem sieciowym dla strumieni wideo PAL (25MB/s). Program nie wykonuje żadnej analizy ani nie modyfikuje otrzymywanych klatek obrazu, a jedynie przekazuje je na wyjście.

Tabela 7 zawiera wyniki eksperymentu. Jak można zauważyć dla powyższej konfiguracji znaczące straty danych (znacznie powyżej 1% przesyłanych klatek obrazu) występują dla jednocześnie realizowanych 21 zadań, co więcej, dla tego samego obciążenia następuje również całkowite (99,36%) obciążenie węzła.

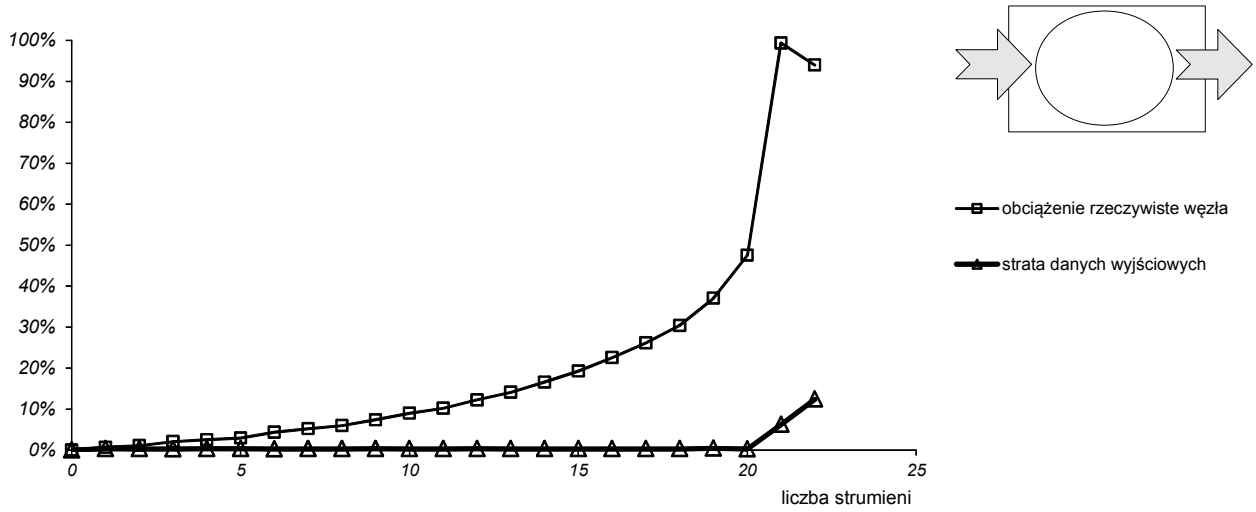
Tabela 7: Wyniki badań obciążenia obliczeniowego węzła dla przekaźnika – strumień PAL.

liczba strumieni	0	1	2	3	4	5	6	7	8	9	10	11	12
obciążenie węzła	0,00%	0,68%	1,11%	2,04%	2,51%	2,93%	4,35%	5,22%	5,95%	7,38%	8,97%	10,17%	12,22%
strata danych	0,00%	0,31%	0,30%	0,26%	0,31%	0,31%	0,25%	0,29%	0,29%	0,31%	0,28%	0,30%	0,30%
liczba strumieni	13	14	15	16	17	18	19	20	21	22			
obciążenie węzła	14,11%	16,59%	19,26%	22,56%	26,15%	30,39%	37,07%	47,55%	99,36%	93,96%			
strata danych	0,29%	0,30%	0,28%	0,28%	0,28%	0,27%	0,41%	0,26%	6,21%	12,40%			

Na rys. 39 został przedstawiony wykres obrazujący wyniki z tabeli 7. Należy zauważyć, że zgodnie z sugestiami przedstawionymi w rozdz. 4.1.5 obciążenie obliczeniowe dla każdego następnego zadania wrasta nieliniowo, inaczej:

$$\gamma(c,T) \neq \sum_{i=1}^{|T|} \gamma(c,\{t_i\}) \quad (10)$$

zauważa się to szczególnie dla większej liczby uruchomionych zadań, np. obciążenie obliczeniowe węzła przypadające na jedno zadanie, przy 20 tych samych zadaniach wynosi: 2,38%, zaś dla pojedynczego tego zadania wynosi 0,68%.



Rysunek 39: Wyniki badań obciążenia obliczeniowego węzła dla przekaźnika – strumień PAL.

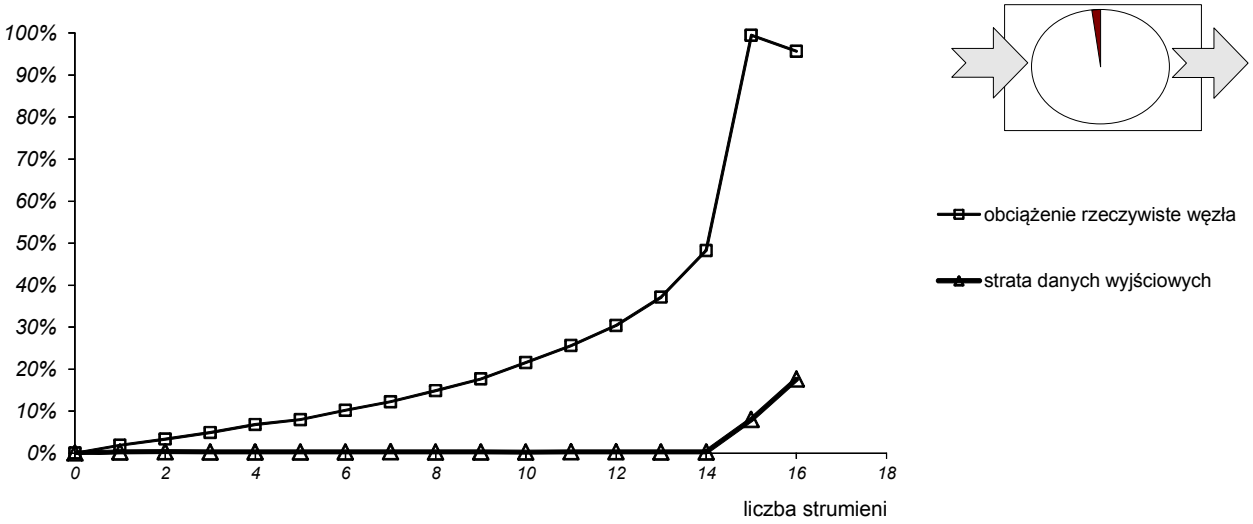
4.2.2 Strumień PAL: Zegar

Eksperyment ten umożliwia badanie zachowania się węzła obliczeniowego przy bardzo niskim obciążeniu obliczeniowym w stosunku do ruchu sieciowego – **jeden wątek** modyfikujący klatki obrazu i wysyłający je jako strumień wyjściowy.

Tabela 8: Wyniki badań obciążenia obliczeniowego węzła dla zegara – strumień PAL.

liczba strumieni	0	1	2	3	4	5	6	7	8	9	10	11	12
obciążenie węzła	0,00%	1,87%	3,32%	4,89%	6,78%	7,97%	10,19%	12,20%	14,85%	17,66%	21,55%	25,58%	30,37%
strata danych	0,00%	0,26%	0,33%	0,27%	0,26%	0,28%	0,28%	0,29%	0,27%	0,27%	0,25%	0,31%	0,32%
liczba strumieni	13	14	15	16									
obciążenie węzła	37,13%	48,19%	99,44%	95,63%									
strata danych	0,26%	0,26%	8,00%	17,58%									

Tabela 8 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych



Rysunek 40: Wyniki badań obciążenia obliczeniowego węzła dla zegara – strumień PAL.

wystąpiła już przy 15 jednocześnie analizowanych strumieniach. Otrzymane wyniki (tabela 8) zostały zilustrowane graficznie na wykresie na rys. 40. Podobnie jak dla eksperymentu przekaźnika przyrost wykorzystywanej mocy obliczeniowej jest wyraźnie nieliniowy wraz ze wzrostem liczby przetwarzanych tych samych zadań.

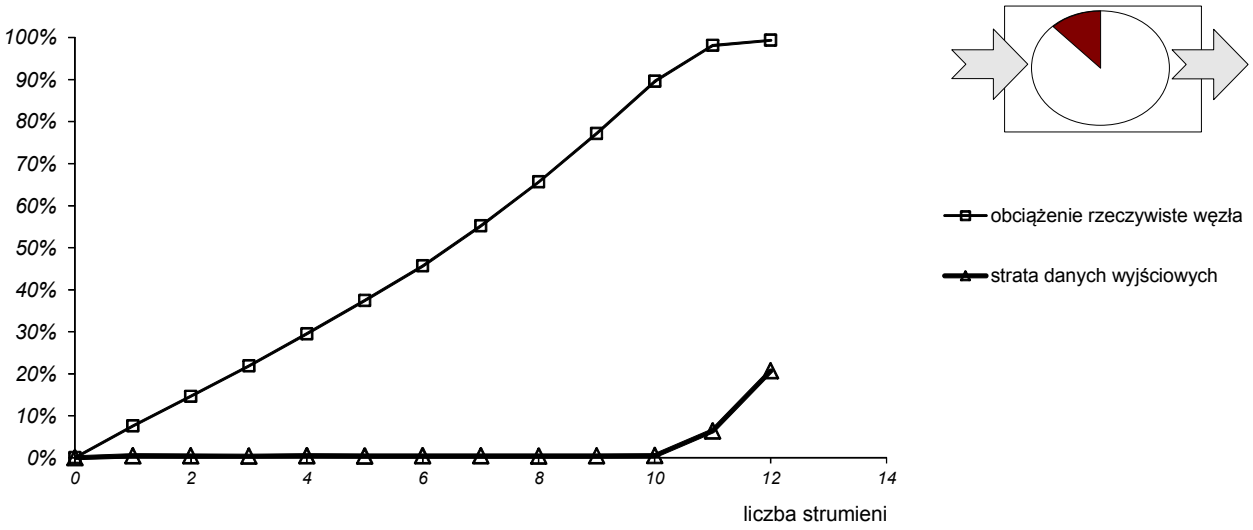
4.2.3 Strumień PAL: Detekcja twarzy, na co 32-giej klatce

Eksperyment umożliwia badanie zachowania się węzła obliczeniowego przy niskim obciążeniu obliczeniowym w stosunku do ruchu sieciowego – **jeden wątek** wykonujący analizę klatek pojedynczego strumienia wideo, wraz z ich modyfikacją i wysyłaniem jako strumień wyjściowy, przy założeniu, że analiza twarzy jest wykonywana na co 32-giej klatce.

Tabela 9: Wyniki badań obciążenia obliczeniowego węzła dla rozpozawania twarzy na co 32-giej klatce – strumień PAL.

liczba strumieni	0	1	2	3	4	5	6	7	8	9	10	11	12
obciążenie węzła	0,00%	7,55%	14,62%	21,83%	29,48%	37,43%	45,65%	55,18%	65,66%	77,18%	89,59%	98,15%	99,36%
strata danych	0,00%	0,46%	0,43%	0,37%	0,44%	0,39%	0,42%	0,44%	0,41%	0,42%	0,49%	6,35%	20,68%

Tabela 9 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła już przy 11 jednocześnie analizowanych strumieniach. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 41. Co potwierdza, że przyrost wykorzystywanej mocy obliczeniowej jest nieliniowy.



Rysunek 41: Wyniki badań obciążenia obliczeniowego węzła dla detekcji twarzy na co 32-giej klatce – strumień PAL.

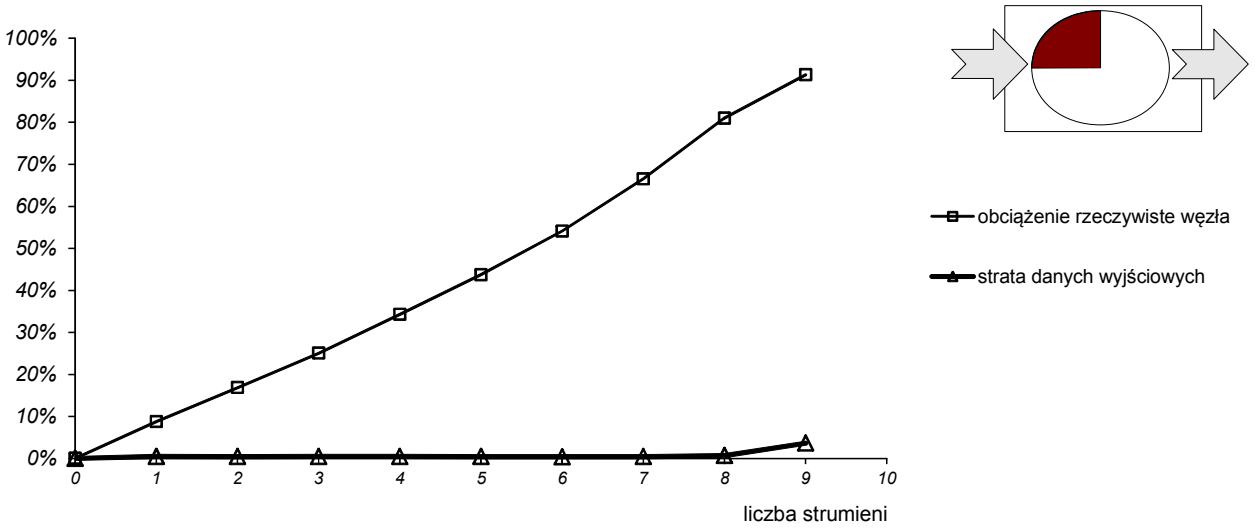
4.2.4 Strumień PAL: Detekcja twarzy, na co 16-tej klatce

Eksperyment umożliwia badanie zachowania się węzła obliczeniowego przy średnim obciążeniu obliczeniowym w stosunku do ruchu sieciowego – **dwa wątki** wykonujące analizę klatek wideo, wraz z ich modyfikacją i wysyłaniem jako strumień wyjściowy, przy założeniu, że analiza twarzy jest wykonywana na co 16-tej klatce.

Tabela 10: Wyniki badań obciążenia obliczeniowego węzła dla rozpozawania twarzy na co 16 klatce – strumień PAL.

liczba strumieni	0	1	2	3	4	5	6	7	8	9			
obciążenie węzła	0,00%	8,77%	16,85%	25,03%	34,26%	43,70%	54,12%	66,53%	81,00%	91,29%			
strata danych	0,00%	0,46%	0,41%	0,44%	0,46%	0,42%	0,40%	0,41%	0,67%	3,63%			

Tabela 10 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła już przy 9 strumieniach. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 42. Liczba uruchomionych zadań jest na tyle mała, że nie można już zauważyć nieliniowości w przyroście wymaganych zasobów obliczeniowych.



Rysunek 42: Wyniki badań obciążenia obliczeniowego węzła dla detekcji twarzy na co 16-tej klatce.

4.2.5 Wyznaczenie funkcji korekty dla strumieni PAL

Zgodnie z założeniami z rozdz. 4.1.5 znając zadany zbiór zadań uruchomionym na węźle, można oszacować jakie będzie obciążenie na tym węźle. O ile znane jest obciążenie obliczeniowe jakie każde z tych zadań generuje samodzielnie na węźle oraz, gdy każde zadanie posiada pojedynczy strumień wejściowy i pojedynczy strumień wyjściowy. Dla łatwiejszego odczytu wyników eksperymentów: obciążenie obliczeniowe węzła (definicja 5) oraz strata danych wyjściowych (definicja 8) na wykresach i w tabelach zostały one przedstawione jako wartości procentowe.

Stawiamy tezę, że po zwiększeniu obciążenia obliczeniowego poprzez dodanie operacji wykonywanych na każdym elemencie strumienia multimedialnego (np. klatce obrazu), przyrost wymagań obliczeniowych jest porównywalny z przyrostem zmierzonym dla algorytmów przekaźnika (15-20 zadań), zegara (11-15 zadań) i rozpoznawania twarzy (2-10 zadań) zgodnie ze wzorem (9), w którym $\eta_h(n)$ jest funkcją korekty określaną eksperymentalnie, przyjmującą jako argument liczbę obsługiwanych strumieni i jest zależna od ich rodzaju. Tabela 11 przestawia wartości funkcji $\eta_h(n)$ dla strumieni PAL ($h=pal$): $\eta_{pal}(n)$. W dalszych eksperymentach, w rozdz. 4.3 dokonano weryfikacji uzyskanej funkcji, określonej przez tabelę 11.

Tabela 11: Wartości funkcji korekty $\eta_{pal}(n)$ dla strumieni PAL.

liczba strumieni	0	1	2	3	4	5	6	7	8	9	10	11	12
η_{PAL}	0,00	1,00	0,96	0,95	0,98	0,99	1,01	1,04	1,09	1,14	1,19	1,25	1,35
liczba strumieni	13	14	15	16	17	18	19	20					
η_{PAL}	1,53	1,84	1,89	2,08	2,27	2,49	2,88	3,51					

4.3 Weryfikacja funkcji korekty dla strumieni PAL

Kolejne eksperymenty służą weryfikacji wyznaczonej funkcji korekty $\eta_{pal}(n)$, dla każdego eksperymentu wyznaczono wykres funkcji aproksymującej obciążenie $\gamma'(c, T)$, której wartości zależą bezpośrednio on weryfikowanej funkcji korekty $\eta_{pal}(n)$ (patrz wzór (9)). Do ilościowego określenia podobieństwa aproksymacji ($\gamma'(c, T)$) do rzeczywistego obciążenia ($\gamma(c, T)$) wykorzystano kwadrat współczynnika korelacji Pearsona: R^2; podobieństwo badano dla zakresów, w których strata danych wyjściowych była na akceptowalnym poziomie: $\varphi(c, T)<L$, gdzie L=0,01.

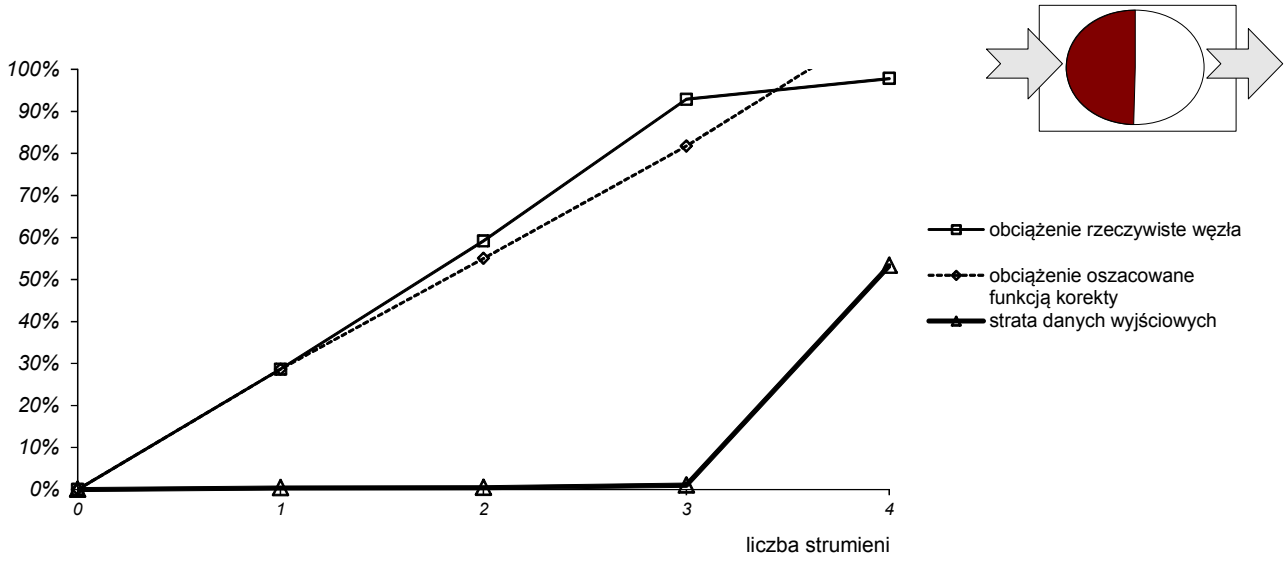
4.3.1 Strumień PAL: Detekcja twarzy, na co 4-tej klatce

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego przy dużym obciążeniu obliczeniowym w stosunku do ruchu sieciowego – **cztery wątki** wykonujące analizę klatek wideo, wraz z ich modyfikacją i wysyłaniem jako strumień wyjściowy.

Tabela 12: Wyniki badań obciążenia obliczeniowego węzła dla rozpozawania twarzy na co 4 klatce – strumień PAL.

liczba strumieni	0	1	2	3	4
obciążenie węzła	0,00%	28,65%	59,16%	92,86%	97,84%
strata danych	0,00%	0,36%	0,41%	1,00%	53,29%

Tabela 12 zawiera wyniki eksperymentu, w tym przypadku znacząca strata danych wyjściowych wystąpiła już przy 4 strumieniach. Uzyskane wyniki zostały zilustrowane graficznie na wykresie z rys. 43. Liczba uruchomionych zadań jest na tyle mała, że nie można już zauważyć nieliniowości w przyroście wymaganych zasobów obliczeniowych. Ponadto na wykresie zaznaczono linią przerywaną obciążenie $\gamma'(c, T)$ wyznaczone przy użyciu funkcji $\eta_{pal}(n)$ na podstawie równania (9), dla sprawdzenia zgodności z rzeczywistymi wynikami, określono dla tych krzywych kwadrat współczynnika korelacji Pearsona, którego wartość wynosi, R^2=0,9565.



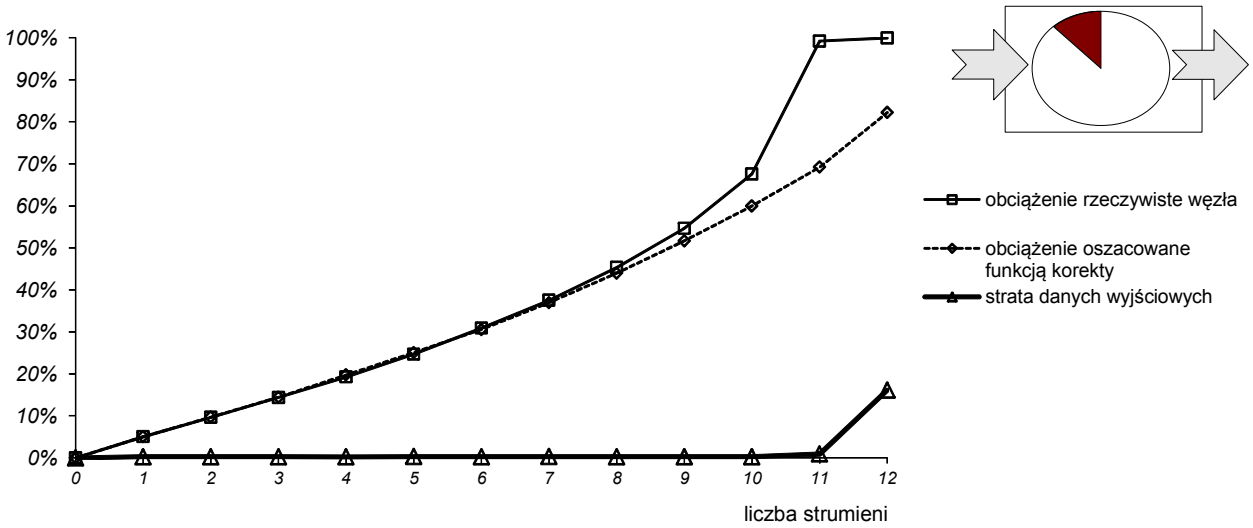
Rysunek 43: Wyniki badań obciążenia obliczeniowego węzła dla detekcji twarzy na co 4-tej klatce.

4.3.2 Strumień PAL: Wykrywanie krawędzi

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego przy niskim obciążeniu obliczeniowym w stosunku do ruchu sieciowego – **jeden wątek** wykonujący analizę klatek pojedynczego strumienia wideo, wraz z ich modyfikacją i wysyłaniem jako strumień wyjściowy.

Tabela 13: Wyniki badań obciążenia obliczeniowego węzła dla wykrywania krawędzi – strumień PAL.

liczba strumieni	0	1	2	3	4	5	6	7	8	9	10	11	12
obciążenie węzła	0,00%	5,05%	9,66%	14,35%	19,24%	24,71%	30,88%	37,52%	45,33%	54,64%	67,60%	99,22%	99,96%
strata danych	0,00%	0,29%	0,27%	0,28%	0,24%	0,31%	0,26%	0,30%	0,26%	0,26%	0,26%	0,92%	16,11%



Rysunek 44: Wyniki badań obciążenia obliczeniowego węzła dla wykrywania krawędzi – strumień PAL.

Tabela 13 zawiera wyniki eksperymentu, w tym przypadku znacząca strata danych wyjściowych wystąpiła już przy 11 strumieniach. Uzyskane wyniki zostały zilustrowane graficznie na wykresie z rys. 41. Podobnie jak dla wcześniejszych eksperymentów przyrost wykorzystywanej mocy obliczeniowej jest nieliniowy. Ponadto na wykresie zaznaczono linią przerywaną obciążenie $\gamma'(c, T)$ wyznaczone przy użyciu funkcji $\eta_{pal}(n)$ na podstawie równania (9), dla sprawdzenia zgodności z rzeczywistymi wynikami, określono dla tych krzywych kwadrat współczynnika korelacji Pearsona, którego wartość wynosi, R^2=0,9682.

4.3.3 Strumień PAL: Maska tła

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego przy dużym obciążeniu obliczeniowym w stosunku do ruchu sieciowego – **cztery wątki** wykonujące analizę klatek wideo, wraz z ich modyfikacją i wysyłaniem jako strumień wyjściowy.

Tabela 14: Wyniki badań obciążenia obliczeniowego węzła dla maski tła – strumień PAL.

liczba strumieni	0	1	2	3	4	5
obciążenie węzła	0,00%	26,71%	65,03%	80,52%	92,53%	96,84%
strata danych	0,00%	0,30%	27,88%	51,16%	63,88%	71,72%

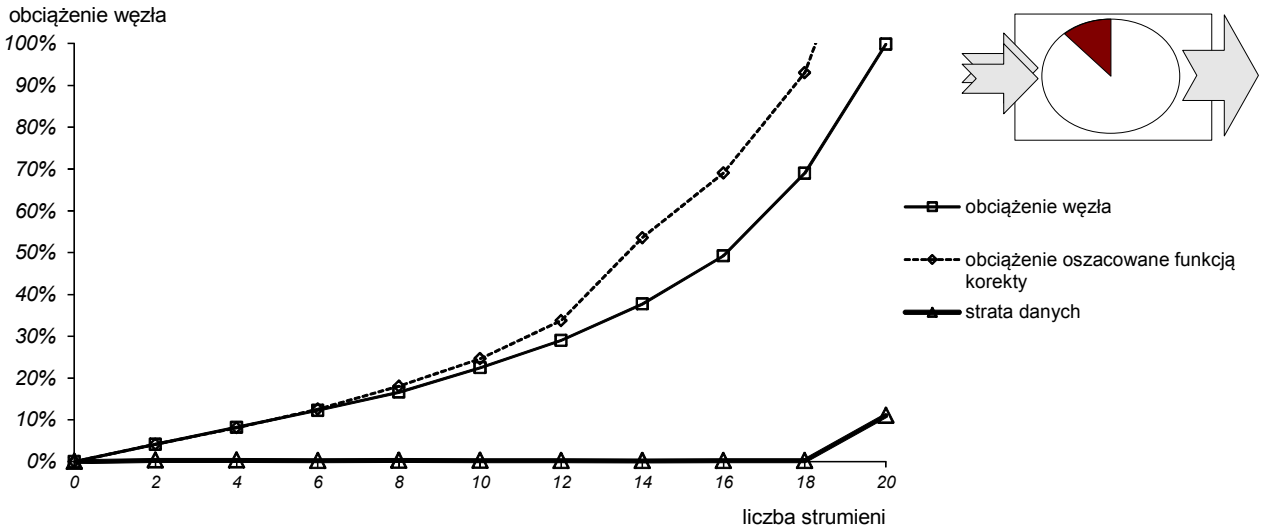
Tabela 14 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła już przy 2 strumieniach.

4.3.4 Strumień PAL: Scalenie obrazów

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego dla zadań z więcej niż jednym (dwoma w tym przypadku) wejściowymi strumieniami danych, jednym strumieniem wyjściowym i stosunkowo niskim obciążeniem obliczeniowym – jeden wątek przetwarzający dane.

Tabela 15: Wyniki badań obciążenia obliczeniowego węzła dla scalania strumieni PAL.

liczba strumieni	0	2	4	6	8	10	12	14	16	18	20
obciążenie węzła	0,00%	4,15%	8,21%	12,27%	16,62%	22,45%	29,01%	37,71%	49,26%	68,99%	99,84%
strata danych	0,00%	0,31%	0,31%	0,20%	0,29%	0,23%	0,22%	0,19%	0,20%	0,21%	11,02%



Rysunek 45: Wyniki badań obciążenia obliczeniowego węzła dla scalania strumieni PAL.

Tabela 15 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła przy 5 strumieniach. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 45. Ponadto na wykresie zaznaczono linią przerywaną obciążenie $\gamma'(c, T)$ wyznaczone przy użyciu funkcji $\eta_{pal}(n)$ na podstawie równania (9), dla sprawdzenia zgodności z rzeczywistymi wynikami, określono dla tych krzywych kwadrat współczynnika korelacji Pearsona, którego wartość wynosi, R^2=0,9951.

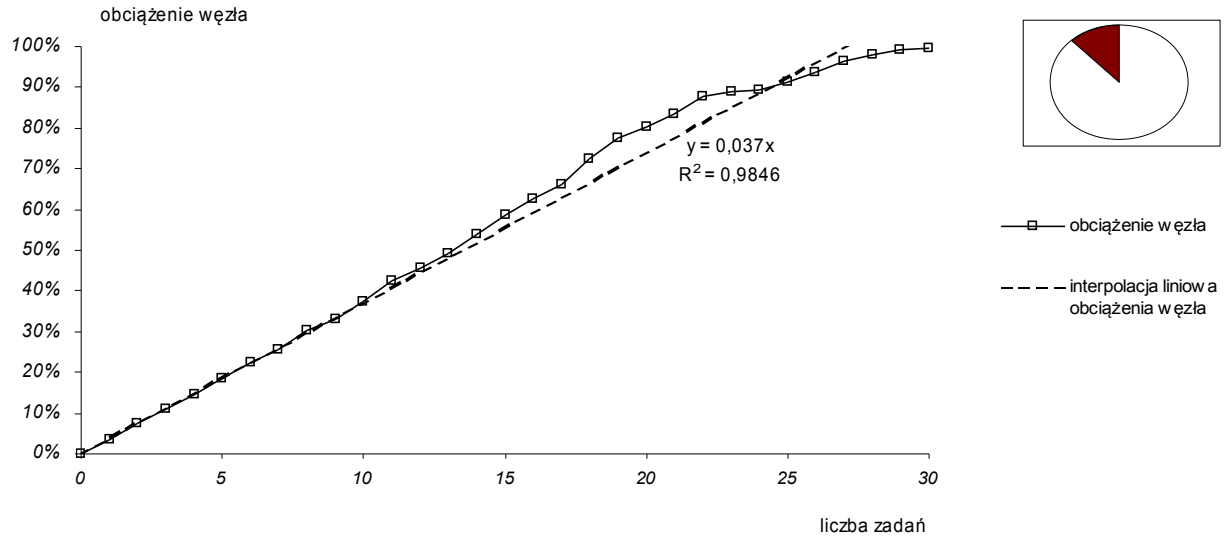
4.3.5 Obraz PAL: Statyczna detekcja twarzy, na co 32-giej klatce

Eksperyment umożliwia zbadanie zachowania się węzła obciążonego jedynie obliczeniami, bez transmisji strumieni poprzez sieć, z wykorzystaniem zadania zawierającego **jeden wątek** analizujący dane: klatki obrazu o wymiarach PAL.

Tabela 16: Wyniki badań obciążenia obliczeniowego węzła rozpoznawiania twarzy na 32 klatce z wykorzystaniem obrazu statycznego.

liczba strumieni	0	1	2	3	4	5	6	7	8	9	10	11	12
obciążenie węzła	0,00%	3,68%	7,42%	11,15%	14,38%	18,32%	22,46%	25,62%	30,22%	32,92%	37,33%	42,39%	45,57%
liczba strumieni	13	14	15	16	17	18	19	20	21	22	23	24	25
obciążenie węzła	49,30%	54,13%	58,80%	62,64%	66,20%	72,28%	77,66%	80,39%	83,29%	87,65%	88,81%	89,20%	91,36%
liczba strumieni	26	27	28	29	30								
obciążenie węzła	93,79%	96,35%	97,93%	99,31%	99,70%								

Tabela 16 zawiera wyniki eksperymentu, obserwujemy, że zanim rdzenie obliczeniowe się nasyciły, można było uruchomić aż 29 zadań. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 46. W przeciwieństwie do zadań przetwarzających strumienie multimedialne, rozpoznawanie na statycznym obrazie skaluje się liniowo, aż do momentu całkowitego obciążenia węzła.



Rysunek 46: Wyniki badań obciążenia obliczeniowego węzła dla lokalnej detekcji twarzy na co 32-giej klatce.

4.4 Wyniki badań wydajnościowych i wiarygodnościowych – strumienie HD i audio

Kolejne eksperymenty zostały wykonane dla innych niż wideo PAL strumieni. Wykonano zadania dla strumieni wideo HD oraz audio. Dla łatwiejszego odczytu wyników eksperymentów: obciążenie obliczeniowe węzła (definicja 5) oraz strata danych wyjściowych (definicja 8) na wykresach i w tabelach zostały one przedstawione jako wartości procentowe.

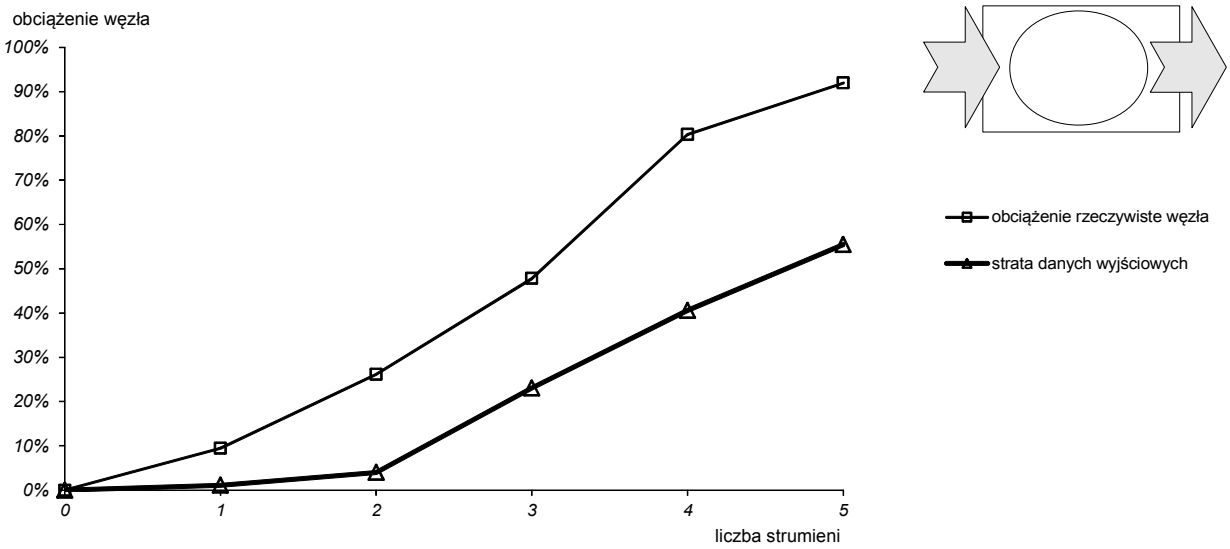
4.4.1 Strumień HD: Przekaźnik

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego bez obciążenia obliczeniowego, a jedynie z ruchem sieciowym: strumień wideo HD. Program nie wykonuje żadnej analizy ani nie modyfikuje otrzymywanych klatek obrazu, a jedynie przekazuje je dalej.

Tabela 17: Wyniki badań obciążenia obliczeniowego węzła dla przekaźnika – strumień HD.

liczba strumieni	0	1	2	3	4	5							
obciążenie węzła	0,00%	9,48%	26,19%	47,81%	80,34%	91,96%							
strata danych	0,00%	1,12%	4,01%	23,08%	40,60%	55,47%							

Tabela 17 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła już przy 2 strumieniach, co oznacza, że dla zadań pracujących na strumieniach HD, na pojedynczym węźle nie powinno się znajdować więcej niż jedno zadanie. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 47.



Rysunek 47: Wyniki badań obciążenia obliczeniowego węzła dla przekaźnika – strumień HD.

4.4.2 Strumień HD: Zegar

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego przy bardzo niskim obciążeniu obliczeniowym w stosunku do ruchu sieciowego dla strumienia HD. Program modyfikuje klatki obrazu i wysyłający je jako strumień wyjściowy.

Tabela 18: Wyniki badań obciążenia obliczeniowego węzła dla zegara – strumień HD.

liczba strumieni	0	1	2	3	4								
obciążenie węzła	0,00%	14,87%	46,25%	87,89%	97,96%								
strata danych	0,00%	1,28%	18,26%	46,25%	59,54%								

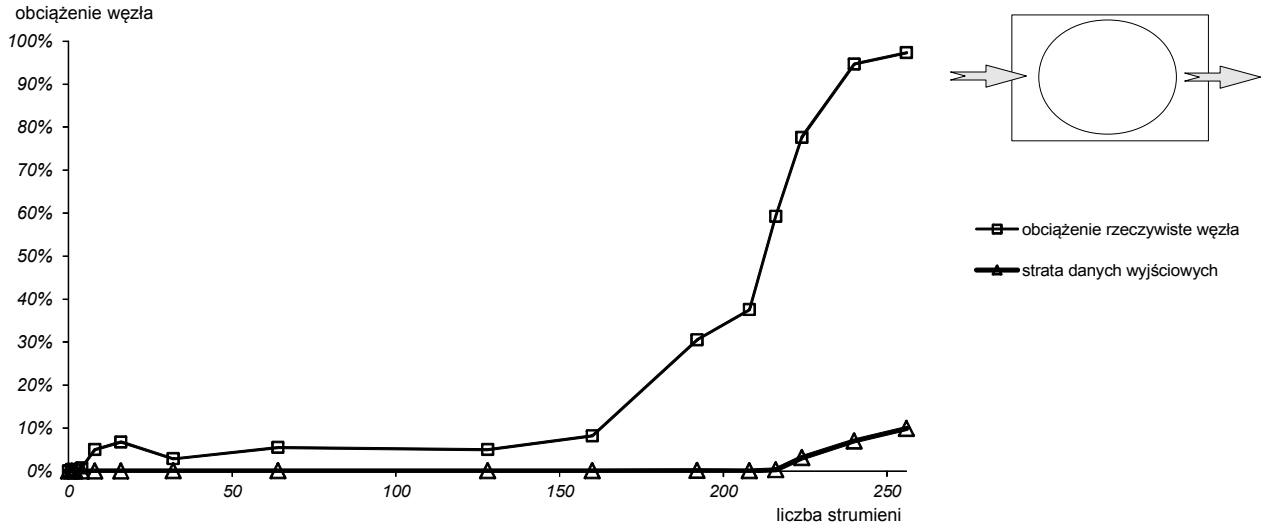
Tabela 17 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła już przy 2 strumieniach, co potwierdza, że dla przetwarzania strumieni HD na pojedynczym węźle nie powinno się znajdować więcej niż jedno zadanie.

4.4.3 Strumień audio: Przekaźnik

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego bez obciążenia obliczeniowego, a jedynie z ruchem sieciowym: strumień audio (0,35MB/s). Program nie wykonuje żadnej analizy ani nie modyfikuje otrzymywanych klatek obrazu, a jedynie przekazuje je dalej.

Tabela 19: Wyniki badań obciążenia obliczeniowego węzła dla przekaźnika – strumień audio.

liczba strumieni	0	1	2	4	8	16	32	64	128	160	192	208	216
obciążenie węzła	0,00%	0,40%	0,13%	0,75%	5,00%	6,76%	2,86%	5,48%	5,01%	8,20%	30,53%	37,55%	59,26%
strata danych	0,00%	0,00%	0,02%	0,00%	0,03%	0,03%	0,08%	0,07%	0,06%	0,05%	0,12%	0,07%	0,29%
liczba strumieni	224	240	256										
obciążenie węzła	77,63%	94,66%	97,32%										
strata danych	3,12%	7,01%	9,93%										



Rysunek 48: Wyniki badań obciążenia obliczeniowego węzła dla przekaźnika – strumień audio.

Tabela 19 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła dopiero przy 224 strumieniach. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 48. Podobnie jak dla innych strumieni multimedialnych widać nieliniowy przyrost wymaganych zasobów obliczeniowych dla większej liczby zadań, jednakże przy odpowiednio niskiej liczbie uruchomionych na węźle zadań (<150), efekt nieliniowości może zostać pominięty – czyli możemy założyć, że funkcja korekty jest neutralna: $\eta_{aud}(n)=1.0$.

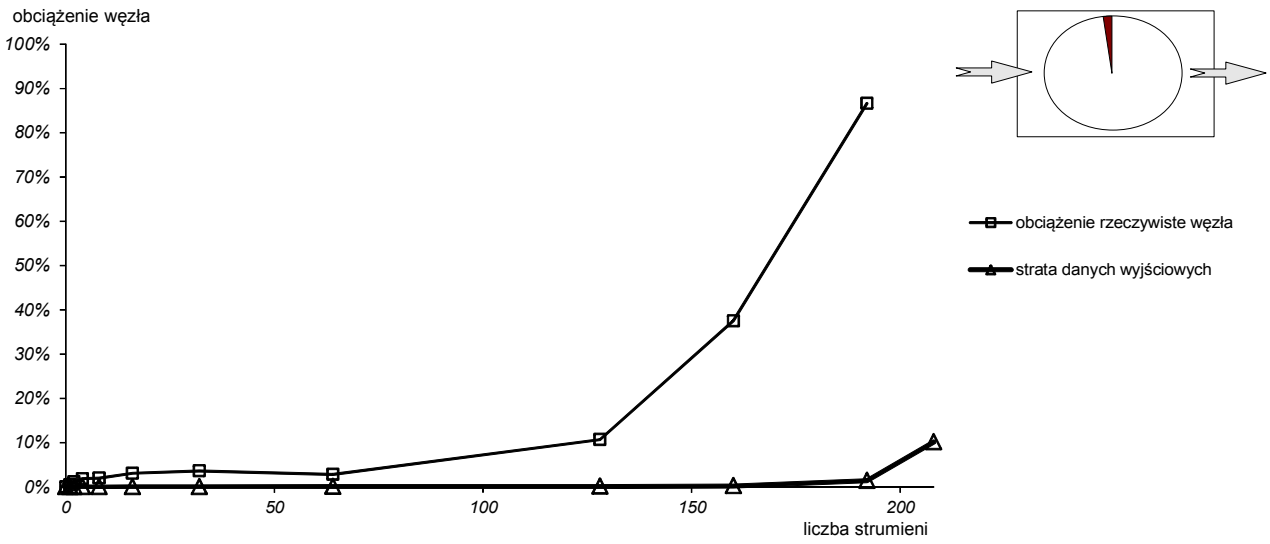
4.4.4 Strumień audio: Filtr częstotliwości

Eksperyment umożliwia zbadanie zachowania się węzła obliczeniowego przy obciążeniu generowanym przez algorytm przetwarzający strumień audio. Program analizuje poszczególne próbki dźwięku, przekształca na spektrum częstotliwości, a następnie ogranicza pasmo do przedziału 1-3kHz – środkowoprzepustowy filtr Butterwortha.

Tabela 20: Wyniki badań obciążenia obliczeniowego węzła dla filtra środkowoprzepustowego – strumień audio.

liczba strumieni	0	1	2	4	8	16	32	64	128	160	192	208
obciążenie węzła	0,00%	0,52%	1,17%	1,89%	2,04%	3,09%	3,66%	2,84%	10,72%	37,52%	86,64%	34,70%
strata danych	0,00%	0,02%	0,01%	0,03%	0,02%	0,03%	0,02%	0,08%	0,13%	0,24%	1,41%	10,17%

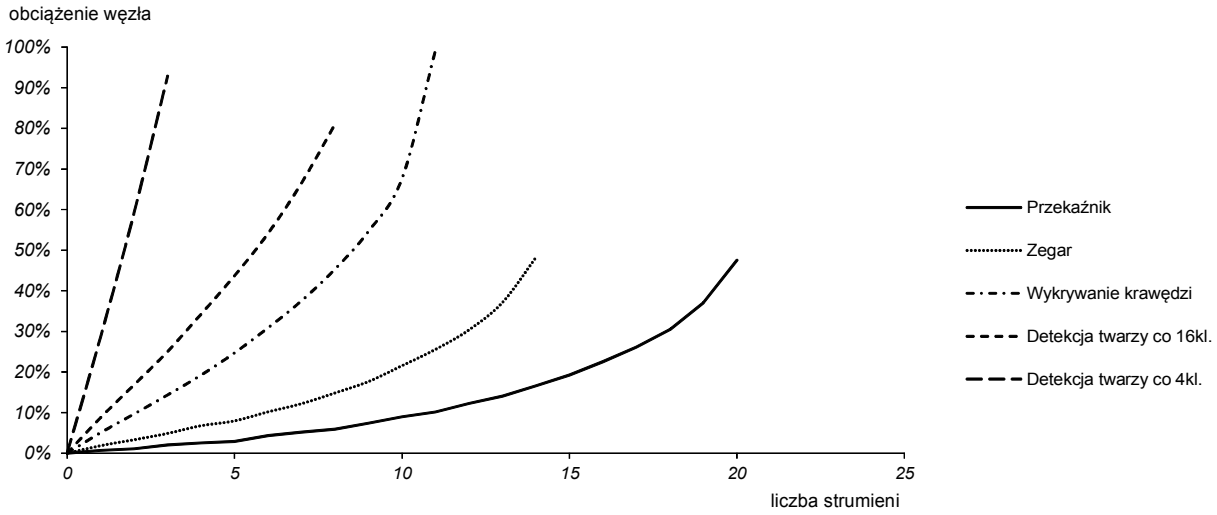
Tabela 20 zawiera wyniki eksperymentu, w tym przypadku znacząca strata przesyłanych danych wystąpiła przy 208 strumieniach. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 49.



Rysunek 49: Wyniki badań obciążenia obliczeniowego węzła dla filra środkowoprzepustowego – strumień audio.

4.5 Podsumowanie badań pojedynczego węzła

Z badań wynika, że obciążenie węzła dla pełnej obróbki kolejnych klatek w strumieniu związanych z ich odbiorem, analizą i przekazaniem klatek przekształconych na wyście jest dwukrotnie wyższe niż sama analiza tych klatek dla algorytmu detekcji twarzy na co 32 klatce (por. rozdz. 4.2.3 i 4.3.5). To oznacza, że transport klatek strumieni konsumuje taką samą ilość mocy obliczeniowej co ich analiza. Z tego powodu istotna jest dalsza optymalizacja mechanizmów komunikacji na platformie KASKADA.

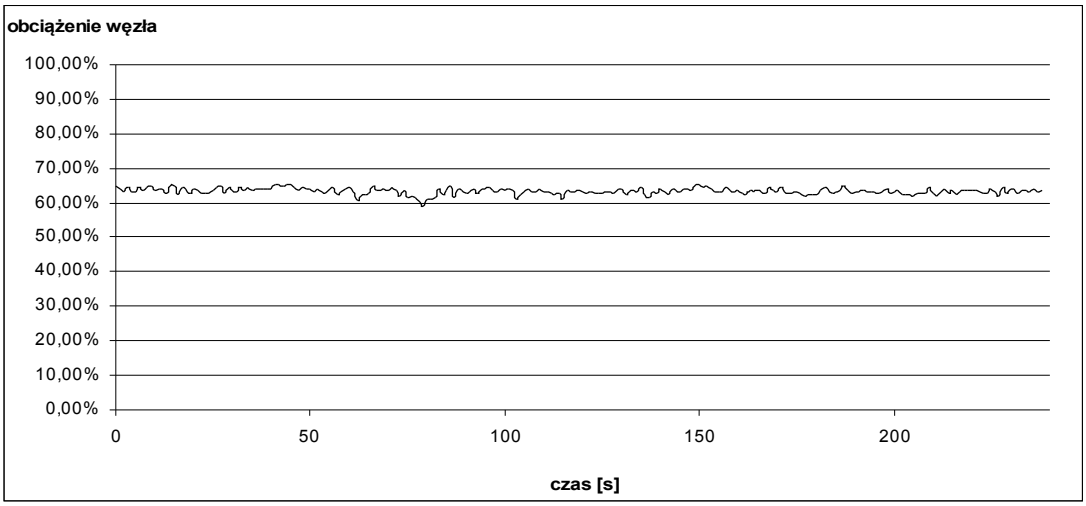


Rysunek 50: Charakterystyki nieliniowości obciążenia węzła dla poszczególnych zadań testowych wykonywanych na strumieniach PAL.

Na rys. 50 przedstawiono charakterystyki obciążenia węzła gdzie obserwowana nieliniowość wzrasta wraz ze zrostem złożoności obliczeniowej algorytmów wykonywanych przez badane zadania obliczeniowe (z większą intensywnością obliczeń [williams09]). Jak można zauważyć dla kolejnych algorytmów wraz ze zwiększeniem ilości obliczeń w stosunku do ilości przesyłanych danych, nieliniowość w przyroście obciążenia jest coraz mniejsza, aż do momentu kiedy wykres

jest praktycznie liniowy, co jest to zgodne z teoretycznymi przewidywaniami z rozdz. 4.1.5 .

Ponadto sam **rozmiar** przetwarzanego **strumienia** multimedialnego ma duży wpływ na wykorzystywaną moc obliczeniową węzła. Zgodnie z oczekiwaniami, większy strumień powoduje większe obciążenie węzła; należy jednak zauważyć, że przyrost ten nie jest liniowy. W przytoczonym przykładnie wykorzystanie algorytmu przekaźnika strumienia HD (patrz rozdz. 4.4.1), którego rozmiar był większy 6.3 razy od strumienia PAL (por. tabela 5), obciążenie węzła było aż 8.22 raza większe od obciążenia dla strumienia PAL (por. tabele 7 i 17).



Rysunek 51: Wyniki badań obciążenia obliczeniowego węzła w czasie dla 14 zadań rozpoznawania twarzy na co 32 klatce – strumień PAL (dla doświadczenia z rozdz. 4.2.3).

Podczas badań zaobserwowano (np. dla doświadczenia z rozdz. 4.2.3), że wartość obciążenia dla badanych zadań obliczeniowych jest zmienna w czasie, ale zmiany tej wartości mieszczą się w wąskim, kiluprocentowym przedziale, (patrz wykres na rys. 51). Z powyższej zależności można wyciągnąć wniosek, że obciążenie węzła dla badanych zadań nie zależy od **długości strumienia**.

Eksperymenty z dużym obciążeniem komunikacyjnym (4.2.1 , 4.2.2 , 4.2.3 i 4.4.1) pokazały, że dla kolejnej większej liczby uruchamianych zadań, korzystających z tych samych strumieni danych, również utrzymywał się nieliniowy przyrost obciążenia obliczeniowego – im więcej uruchomionych zadań tym większe obciążenie na pojedyncze zadanie/strumień. Eksperyment 4.3.5 wykazał, że dla samodzielnych obliczeń na statycznych danych, np. na pojedynczym raz załadowanym obrazie (eksperyment z rozdz. 4.3.5) powyższa zależność jest liniowa – bez względu na liczbę uruchomionych zadań, pojedyncze zadanie tak samo obciąża węzeł obliczeniowy.

Na podstawie przeprowadzonych eksperymentów zaproponowano funkcję $\gamma'(c, T)$ umożliwiającą oszacowanie obciążenia węzła dla zadanego zbioru zadań (równanie (10)), jak również wyznaczono (rozdz. 4.2.5) funkcję korekty dla strumieni PAL $\eta_{pal}(n)$ oraz zweryfikowano jej zastosowanie. Przeprowadzono także badania dla strumieni HD, dla których można uruchomić co najwyżej jedno zadanie na węźle (por. rozdz. 4.3) oraz audio, gdzie można założyć, że efekt nieliniowości obciążenia występuje dopiero przy dużej liczbie przetwarzanych strumieni (por. rozdz. 4.4).

Do wykonania pojedynczego zadania przetwarzającego strumień HD potrzebny jest cały węzeł obliczeniowy, mogą mu towarzyszyć jedynie inne zadania bezstrumieniowe. Dla zadań przetwarzających strumienie PAL należy wykorzystywać aproksymacje $\gamma'(c, T)$ opartą o funkcję korekty $\eta_{pal}(n)$. Natomiast zadania przetwarzające strumienie audio można traktować na równi z zadaniami bezstrumieniowymi i umieszczać je zarówno na węzłach z zadaniami przetwarzającymi strumienie PAL, HD jak i osobno.

5 Zarządzanie zadaniami na platformie KASKADA

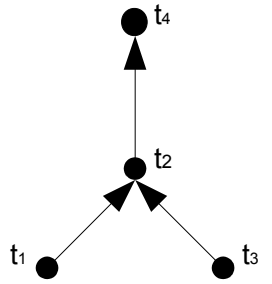
Przedstawiono problematykę alokacji zadań benchmarkowych na węzły obliczeniowe na platformie KASKADA. Wychodząc z definicji klastra obliczeniowego oraz zbioru zadań opisano przyjętą metodę alokacji zadań na węzły klastra obliczeniowego oraz oszacowano jej złożoność obliczeniową. Rozpatrzono heurystyczne algorytmy alokacji klasycznych zadań obliczeniowych, które nie analizują strumieni multimedialnych, takie jak FF (ang. First Fit), FFD (ang. First Fit Descending), BF (ang. Best Fit), BFD (ang. Best Fit Descending) i SSP (ang. Subset Sum Problem). Następnie zaproponowano ulepszoną wersję algorytmu BFD: MBFD (ang. modified best fit descending) umożliwiającą alokację zadań przetwarzających strumienie multimedialne. Dla tego algorytmu alokacji przedstawiono wyniki pomiarów wydajności i wiarygodności obliczeń realizowanych na wielu węzłach platformy KASKADA.

5.1 Model klastra obliczeniowego dla platformy KASKADA

Proponowany opis klastra uwzględnia już wprowadzone definicje z rozdz. 4.1 : definicję 3 klastra obliczeniowego, definicję 4 zadania obliczeniowego, definicję 5 obciążenia węzła zbiorem zadań oraz definicję 6 straty danych i rozszerza go aktualny stan realizowanych obliczeń (zbiór aktualnie wykonywanych zadań na węzłach oraz średnią stratę danych wyjściowych dla całego klastra).

Definicja 8: Zbiór zadań obliczeniowych wykonywanych jednocześnie oznaczamy jako $T=\{t_1, t_2, \ldots t_{|T|}\}$, gdzie $|T|$ jest liczbą zadań.

Tak zdefiniowany zbiór zadań wykonywany na klastrze obliczeniowym jest zwykle skojarzony z usługą złożoną *s*, dla której zadania ze zbioru wykonać obliczenia.



$\gamma(c, \{t_1\}) = 0{,}1$ $\quad \gamma(c, \{t_1, t_2\}) = 0{,}2$
$\gamma(c, \{t_2\}) = 0{,}1$ $\quad \gamma(c, \{t_1, t_3\}) = \mathbf{0{,}4}$
$\gamma(c, \{t_3\}) = 0{,}2$ $\quad \gamma(c, \{t_1, t_4\}) = 0{,}2$
$\gamma(c, \{t_4\}) = 0{,}1$ $\quad \gamma(c, \{t_2, t_4\}) = 0{,}2$
$\gamma(c, \{t_1, t_2, t_4\}) = 0{,}3$
...

Rysunek 52: Przykład grafu zadań oraz wartości funkcji obciążenia obliczeniowego dla wybranych podzbiorów zadań.

Zbiór uruchamianych zadań można przedstawić jako graf skierowany, którego krawędzie reprezentują przepływ strumieni danych, zaś wierzchołki opisują zadania. Na rys. 52 został przedstawiony przykład takiego grafu dla zbioru $T=\{t_1, t_2, t_3\}$ wraz z przykładowymi obciążeniami jakie zadania powodują na węźle *c*. Należy podkreślić, jak to wynika już z rozdz. 4 , przyjmujemy że obciążenia obliczeniowe poszczególnych węzłów nie muszą zachować reguły addytywność ze względu na zbiory zadań tam realizowanych, i tak dla przykładu z rys. 52: $\gamma(c, \{t_1, t_3\}) \neq \gamma(c, \{t_1\}) + \gamma(c, \{t_3\})$.

Definicja 9: Funkcja $\omega^{\tau}(c_i)$, $\omega: C \rightarrow \Theta$ określa jakie **zadania** są **wykonywane na węźle** c_i **w momencie** τ. Θ jest zbiorem wszystkich zadań jakie mogą być uruchomione na platformie. Funkcja $\omega^{<\tau s,\ \tau>}(c_i)$ określa jakie zadania były i są wykonywane na węźle c_i w przedziale czasu: $<\tau s, \tau>$. Między tymi funkcjami występuje następująca zależność: $\omega^{\tau}(c_i)=\omega^{<\tau,\ \tau>}(c_i)$.

W dalszej części rozprawy przyjmujemy założenie, że węzły są obciążone jedynie zadaniami ze

zbioru Θ, a przetwarzane przez nie strumienie, należą do jednego z czterech typów: *hd, pal, aud, nil*, których zbiór został określony w definicji 2, w rozdz. 4.1.2 . Co więcej każde zadanie jest bezpośrednio powiązane z określonym zestawem strumieni danych i od typu przetwarzanego strumienia zależy zachowanie bądź niezachowanie reguły addytywności.

Definicja 10: Stopień strumieniowy zadania jest to funkcja $\varrho_S(t_i)$, $\varrho_S: \Theta \rightarrow \boldsymbol{N}$, której wartością jest liczba strumieni wejściowych o typie *pal* lub *hd przypisanych* do danego zadania t_i.

Definicja 11: Typ strumieniowy zadania jest to funkcja $v(t_i)$, $v: \Theta \rightarrow H$, której wartością jest najwyższy typ strumienia wejściowego zadania t_i. Najwyższy typ określa się zgodnie z następującym porządkiem: *hd* > *pal* > *aud* > *nil,* por. rozdz. 4.1.2 .

Na przykład zadanie t_1 z rys. 52 przetwarzające dwa strumienie wejściowe *pal* i dwa *aud*, posiada stopień strumieniowy $\varrho_S(t)=2$, natomiast typ strumieniowy $v(t)=pal$. Ponieważ zadanie t_3 również przetwarza strumień *pal*, to po umieszczeniu t_1 i t_3 na jednym węźle c obciążenie obliczeniowe jakie będą razem generować jest większe od sumy ich obciążeń gdyby pracowały na min pojedynczo, tzn. $\gamma(c, \{t_1, t_3\}) > \gamma(c, \{t_1\}) + \gamma(c, \{t_3\})$.

Definicja 12: Średnia strata danych dla zadań działających na klastrze C, w przedziale czasu obserwacji $<\tau s, \tau>$:

$$\varphi^{\langle \tau s, \tau \rangle}(C)=\frac{\sum_{i=1}^{|C|}\sum_{j=1}^{|\omega^{\langle \tau s, \tau \rangle}(c_i)|}\varphi\left(c_i, t_j, \omega^{\langle \tau s, \tau \rangle}(c_i)\right)}{\sum_{i=1}^{|C|}|\omega^{\langle \tau s, \tau \rangle}(c_i)|} \quad (11)$$

gdzie funkcja $\varphi(c, t, T)$ została zdefiniowana w rodz. 4.1.3 , def. 7.

Definicja 13: Funkcja fragmentacji klastra C: $\psi^{\tau}(C)$ określa liczbę węzłów w klastrze C, na których są uruchomione zadania, a które nie są w pełni obciążone, tzn. ich obciążenie jest mniejsze od 1, w momencie τ, inaczej:

$$\psi^{\tau}(C)=\left|\{c: c\in C \wedge 0<\gamma\left(c, \omega^{\tau}(c)\right)<1\}\right| \quad (12)$$

5.1.1 Alokacja węzłów dla zadań obliczeniowych

Uruchamiając usługę złożoną, po przypisaniu jej odpowiednich usług prostych, otrzymujemy zbiór zadań, które są rozlokowane na poszczególnych węzłach klastra obliczeniowego (patrz rozdz. 2.5). W przypadku przetwarzania on-line, z jednej strony wymagane jest zagwarantowanie odpowiednich zasobów w czasie działania zadania, z drugiej strony zaś zadanie nie może przekraczać zadeklarowanych wartości ich wykorzystania.

Definicja 14: Alokacja węzłów jest definiowana jako funkcja $\mu^{\tau}(t_i)$, $\mu^{\tau}: T \rightarrow C$ przypisująca każdemu z zadań danego zbioru jeden z węzłów klastra, w taki sposób aby żadne z przypisywanych zadań nie zostało zagłodzone, czyli w danym momencie τ, średnia strata danych podczas przetwarzania strumieni nie była większa niż stała L:

$$\varphi^{\tau}(C)\leqslant L \quad (13)$$

w dalszej części rozprawy przyjmujemy że $L=0,01$.

Wybór funkcji alokacji w momencie $\tau 0$ ma wpływ na stan klastra w momencie $\tau 1$ po jej wykonaniu (uruchomieniu zadań ze zbioru T), tzn.

$$\omega^{\tau 1}(c)=\omega^{\tau 0}(c)\cup\{t: \mu^{\tau 0}(t)=c\} \quad (14)$$

Można zauważyć, że dla pojedynczego klastra obliczeniowego C w momencie $\tau 0$ może istnieć wiele alokacji zadanego zbioru zadań T spełniających definicję 14, oznaczmy zbiór takich funkcji

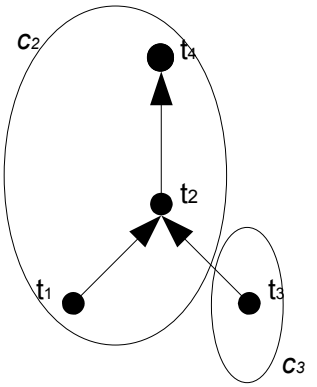
przez $M^{\tau 0}(C, T)$. Dlatego dodatkowo można wybrać taką alokację, która w momencie $\tau 1$ (po jej wyznaczeniu) będzie spełniała dodatkowe kryteria. W przykładzie z rozdz. 3.8 wykorzystano heurystyczny algorytm alokacji równoważenia obciążenia LB (ang. Load Balancing), tzn. wyszukujący funkcję alokacji według następującego, dodatkowego kryterium:

$$\min_{\mu \in M^{\tau 0}(C,T)} \left[\left(\max_{c \in C} \gamma(c, \omega^{\tau 1}(c)) \right) - \left(\min_{c \in C} \gamma(c, \omega^{\tau 1}(c)) \right) \right] \quad (15)$$

W dalszych rozważaniach rozprawy zaproponowano kryterium minimalizacji fragmentacji klastra (definicja 13) tak aby jak najmniejsza liczba węzłów była tylko częściowo wykorzystana, inaczej:

$$\min_{\mu \in M^{\tau 0}(C,T)} \left(\psi^{\tau 1}(C) \right) \quad (16)$$

Analiza porównawcza obu algorytmów alokacji wraz z uzasadnieniem wyboru heurystycznego algorytmu alokacji minimalizującego fragmentację została przedstawiona w rozdz. 5.4.1 .



$\mu(t_1) = c_2$
$\mu(t_2) = c_2$
$\mu(t_3) = c_3$
$\mu(t_4) = c_2$

Rysunek 53: Przykład alokacji węzłów dla grafu zadań z rys. 52.

Na rys. 53 został przedstawiony przykład alokacji węzłów: c_2 i c_3 dla zadań z grafu z rys. 52.

Rozważmy przypadek szczególny dotyczący zadań nie obsługujących strumieni multimedialnych. Zgodnie z wnioskami z rozdz. 4.3.5 możemy założyć, że obciążenie takich zadań wzrasta równomiernie, tzn. że funkcja $\gamma(c, T)$ jest addytywna i spełnia następujący warunek:

$$\gamma(c, T) = \sum_{t \in T} \gamma(c, \{t\}) \quad (17).$$

Uwzględniając powyższe założenie możemy uprościć definicję alokacji (definicja 14) dla zadań obliczeniowych nieprzetwarzających strumieni multimedialnych:

Definicja 15: Alokacja węzłów dla zadań bezstrumieniowych jest definiowana jako funkcja $\mu^{\tau}_C(t_i)$, μ^{τ}_C: $T \rightarrow C$, która w danym momencie τ przypisuje każdemu z zadań danego zbioru T jeden z węzłów klastra, w taki sposób aby suma obciążeń obliczeniowych każdego z zadań z osobna, przypisanych aktualnie do danego węzła nie przekraczała 1, tzn. że zachowany jest poniższy warunek:

$$\forall_{c \in C} \gamma(c, \omega^{\tau}(c)) + \sum_{t \in \{t : \mu^{\tau}_C(t) = c\}} \gamma(c, \{t\}) \leqslant 1 \quad (18)$$

Warto zauważyć, że tak zdefiniowana alokacja wraz z założeniem optymalizacji fragmentacji odpowiada problemowi pakowania koszy (ang. Bin Packing Problem, BPP) [garey79]. Zakłada się, że posiadamy zbiór przedmiotów (zadania) o różnej masie (obciążenie), które należy umieszczać w jednakowych koszach (węzły obliczeniowe), o określonym maksymalnym udźwigu (moc

obliczeniowa), tak aby wykorzystać jak najmniejszą liczbę koszy. Jeśli weźmiemy pod uwagę założenie, że początkowe obciążenie węzłów może być niezerowe ($\gamma(c)>0$), to należy rozważyć wykorzystanie wyników dotychczasowych badań dla wersji BPP ze zmienną wielkością koszy (ang. Variable size BPP, VBPP) [friesen86].

Twierdzenie 1: Pytanie, czy istnieje funkcja alokacji $\mu^{\tau}(t_i)$, $\mu: T \rightarrow C$, taka, że zachodzi warunek ze wzoru (13) w danym momencie τ, dla danego zbioru zadań T i zbioru węzłów C jest problemem NP-trudnym.

Dowód: Weźmy pod uwagę uproszczony, szczególny przypadek kiedy obciążenie obliczeniowe węzła $\gamma(c, T)$ jest addytywne (zgodnie z definicją 15), odpowiada on NP-trudnemu problemowi pakowania koszy – kiedy zbiór zadań jest zbiorem przedmiotów, generowane przez nich obciążenie odpowiada ich masie, a węzły klastra są pakowanymi koszami. W tym przypadku węzły obliczeniowe są koszami, natomiast zadania produktami, które należy do nich włożyć. Załóżmy, że używamy algorytmu potrafiącego określić, czy do k węzłów obliczeniowych (koszy), możliwe jest upakowanie zbioru zadań T (produktów), wtedy zadając to pytanie dla kolejnych wartości naturalnych k (maksymalnie $|T|$ albo $log_2|T|$ razy z użyciem wyszukiwania binarnego) otrzymalibyśmy rozwiązanie problemu pakowania koszy: wynik (minimalna liczba koszy potrzebnych do zapakowania przedmiotów) jest najmniejszą liczbą dla której odpowiedź byłaby pozytywna. ❒

Należy zauważyć, że skoro znalezienie jakiegokolwiek funkcji $\mu^{\tau}(t_i)$, a nawet określenie czy taka istnieje, jest problemem NP-trudnym, to jej optymalizacja ze względu na fragmentację również posiada taką właściwość.

5.1.2 Algorytmy alokacji dla zadań bezstrumieniowych

Ze względu na to, że problem alokacji jest problemem NP-trudnym proponujemy algorytmy heurystyczne oparte na rozwiązaniach używanych dla problemu pakowania koszy o zmiennej wielkości [haouari10].

Uwaga: w przedstawianych poniżej listingach przyjmuje się, że funkcje zależne od czasu czasu (np. $\mu^{\tau}(t_i)$) z reguły dotyczą aktualnego momentu dlatego dla uproszczenia pominięto w nich oznaczenie τ.

Pierwszym rozważanym algorytmem jest algorytm FF (ang. First Fit), przypisuje on kolejne zadanie do pierwszego węzła, w którym zapewniona będzie wystarczająca moc obliczeniową. Istnieje także jego wersja zmodyfikowana, w której lista zadań jest najpierw sortowana malejąco: FFD (ang. First Fit Descending). Listing 1 przedstawia pseudokod dla obu wersji algorytmów FF/FFD. Złożoność obliczeniowa FF: $O(|C|\ log|C|+|T||C|)$, FFD: $O(|T|log|T|+|C|log|C|+|T||C|)$

```
1. utwórz listę T* z elementów zbioru T
2. opcja dla FFD: sortuj T* malejąco wg wartości γ_i
3. utwórz listę C* ze zbioru C, posortowaną rosnąco wg aktualnego obciążenia γ(c, ω(c))
4. for i := 1 to |T|
5.     for j := 0 to |C|
6.         if γ(C*[j],{T*[i]})+γ(C*[j], ω(C*[j])) ≤ 1 then
7.             μ_C(T*[i]) := C*[i]
9. warunek powodzenia: dla każdego węzła ze zbioru C warunek z linii 6 był spełniony
przynajmniej raz
```

Listing 1: Algorytm alokacji FF/FFD.

Podobnie jak FF, algorytm BF (ang. Best Fit) iteruje po kolejnych zadaniach, jednakże węzły nie są

przypisywane przypadkowo, a wybierany jest najlepszy, czyli taki, w którym po przypisaniu zadania zostanie najmniej niewykorzystanej mocy obliczeniowej. Również dla tego algorytmu istnieje wersja z posortowanymi malejąco zadaniami: BFD (ang. Best Fit Descending). Listing 2 przedstawia pseudokod dla algorytmu BF wraz z dodatkowym przepływem realizującym BFD. Złożoność obliczeniowa BF: $O(|T||C|)$, BFD: $O(|T|log|T|+|T||C|)$.

```
1. utwórz listę T* z elementów zbioru T
2. opcja dla BFD: sortuj T* malejąco wg wartości χ_i
3. for i := 1 to |T|
4.     znajdź c∈C gdzie (χ(c,{T*[i]})+χ(c,ω(c))) jest maksymalne i mniejsze równe 1
5.     if c<>null then
6.             μ_C(T*[i]) := c
7.     else
8.             FAIL(„brak alokacji")
```

Listing 2: Algorytm alokacji BF/BFD.

Kolejnym algorytmem heurystycznym alokacji węzłów obliczeniowych dla zadań, jest algorytm przedstawiony na listingu 3 wykorzystujący sumę podzbioru. Wspomniana w pseudokodzie funkcja SSP jest algorytmem rozwiązującym problem sumy podzbioru (ang. Subset Sum Problem, SSP) [garey79]. Ponieważ jest to problem NP-zupełny, w przypadku ogólnym należy użyć algorytmu przybliżonego np.: [ibarra75], złożoność obliczeniowa: $O(|C|log|C|+|C|SSP(|T|))$.

```
1. T_R := T
2. utwórz listę C* ze zbioru C, posortowaną rosnąco wg aktualnego obciążenia χ(c, ω(c))
3. for i := 1 to |C|
4.     c := C*[i]
5.     T':= SSP(T_R, χ(c,ω(c)))
6.       ∀_{t∈T'}  μ_C(t) := c
7.     T_R := T_R\T'
8. if T_R <> Ø then
9.     FAIL(„brak alokacji")
```

Listing 3: Algorytm alokacji bazujący na SSP.

Powyższe algorytmy zostały przebadane na drodze symulacji pod kątem ich wykorzystania w docelowym środowisku platformy KASKADA. W tym celu przyjęliśmy następujące założenia:

1. losowe obciążenie zadań: $\gamma i \in (0, 1>$ (rozkład równomierny),
2. początkowa liczba węzłów niezajętych ($\gamma^{\tau 0}(c, \omega^{\tau}(c))=0$): 20,
3. losowe obciążenie na węzłach częściowo zajętych: (0,1) (rozkład równomierny),
4. liczba węzłów częściowo zajętych: 8,
5. wykorzystanie podprogramu rozwiązującego w sposób dokładny problem sumy podzbiorów (SSP() w linii 5 na listingu 3) – dla wymienionych warunków jest możliwa wydajna implementacja np.: przy użyciu programowania dynamicznego [cormen90].

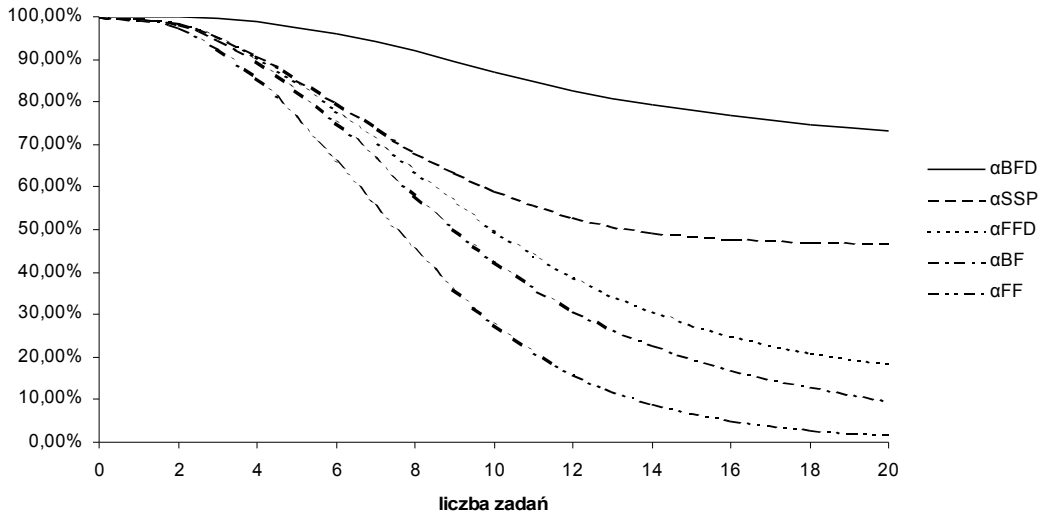
Wyniki symulacji zostały przedstawione za pomocą α_X współczynnika, gdzie X jest badanym algorytmem alokacji:

$$\alpha_X = \frac{\textit{liczba eksperymentów gdzie algorytm alokacji X zwrócił najlepszy wynik (minimalną fragmentację)}}{\textit{liczba wszystkich eksperymentów dla algorytmu alokacji X}}$$

Tabela 21: Wyniki symulacji dla algorytmów alokacji.

liczba zadań	α_{FF}	α_{FFD}	α_{BF}	α_{BFD}	α_{SSP}
2	97,49%	98,43%	98,28%	100,00%	98,43%
4	85,33%	90,74%	89,51%	99,03%	91,00%
6	65,93%	78,04%	75,03%	96,17%	79,60%
8	45,13%	63,64%	58,11%	91,85%	68,12%
10	27,57%	49,77%	42,53%	86,78%	58,89%
12	15,97%	38,85%	30,85%	82,52%	53,03%
14	9,10%	30,75%	22,83%	79,41%	49,24%
16	5,14%	25,06%	16,87%	76,79%	47,69%
18	3,01%	21,18%	13,02%	74,72%	46,94%
20	1,71%	18,53%	9,77%	73,03%	46,72%

Tabela 21 przedstawia wyniki symulacji dla poszczególnych algorytmów, natomiast wykres na rys. 54 podaje ich reprezentację graficzną. Należy zauważyć, że najlepsze rezultaty osiągnął algorytm BFD, następnie w kolejności malejącej algorytmy alokacji: SSP, FFD, BF i FF. Stąd w dalszych rozważaniach uwzględniono algorytm BFD.



Rysunek 54: Wyniki symulacji dla rozpatrywanych algorytmów alokacji.

5.1.3 Algorytm alokacji dla zadań strumieniowych

Jak wynika z badań przeprowadzonych na pojedynczym węźle (rozdz. 4.2), dla zadań przetwarzających strumienie multimedialne, obciążenie wzrasta nieliniowo wraz z wzrostem liczby zadań, czyli że funkcja $\gamma(c, T)$ nie posiada cechy addytywności dla sumy zbiorów realizowanych zadań – zatem równanie (17) nie jest prawdziwe. Z tego powodu w rzeczywistym systemie bezpośrednie użycie algorytmów opisanych w rodz. 5.1.2 mogłoby spowodować przypisanie pojedynczemu węzłowi zbyt wielu zadań, tak że mogłaby nastąpić większa strata danych podczas przetwarzania strumieni multimedialnych tzn. $\varphi^{<\tau s, \tau>}(C)>L$.

Funkcja $\gamma(c, T)$, $\gamma: (C, 2^{\Theta}) \rightarrow \boldsymbol{R}$, w swojej ogólnej postaci jest trudna do wyznaczenia, zbiór wszystkich potencjalnych zadań 2^{Θ} jest skończony, ale bardzo duży: $2^{|\Theta|}$. Z kolei stosunkowo łatwo jest określić powyższą funkcję dla każdego z podzbiorów składających się z pojedynczych zadań, w takim przypadku należy jedynie przebadać działanie każdego zadania na pustym węźle obliczeniowym, czyli $|\Theta|$ razy.

W badaniach pojedynczego węzła określiliśmy doświadczalnie w jaki sposób zmienia się funkcja $\varphi^{<\tau s,\ \tau>}(C)$ dla konkretnych typów strumieniowych zadań. Dla każdego z nich wyznaczano też funkcję korekty $\eta_h(n)$, która określa dla konkretnego typu strumieniowego zadań h, gdzie $h \in H$, o ile należy zwiększyć przewidywane obciążenie procesora w stosunku do obciążenia addytywnego:

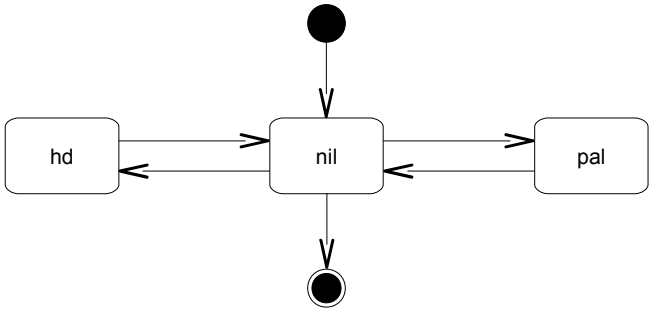
$$\forall_{c \in C} \gamma(c, \omega^{\tau}(c)) \approx \eta_h\left(\sum_{t \in \omega^{\tau}(c)} \varrho_S(t)\right) \sum_{t \in \omega^{\tau}(c)} \gamma(c, \{t\}) \quad (19)$$

Należy pamiętać, że powyższa estymacja jest jedynie oszacowaniem wyznaczonym doświadczalnie, dlatego dla pewnych nietypowych sytuacji wyznaczone przez nią wartości mogą różnić się od rzeczywistych. Była ona sprawdzona podczas testów dla pojedynczego węzła (rozdz. 4.2) i wykazywała poprawne wyniki dla jednego rodzaju strumienia. Nie można przy jej pomocy estymować zadań działających na tym samym węźle, ale przetwarzających różne typy strumieni.

Uwzględniając nieaddytywność obciążeń węzłów konieczna jest modyfikacja algorytmu aloakcji BFD. Uwzględniający funkcję korekty. Zmodyfikowany algorytm BFD (MBFD) został przedstawiony na listingu 4. Podobnie jak w wersji klasycznej, następuje wyszukiwanie najlepiej pasującego (ang. best fit) węzła, tzn. takiego, w którym po alokacji dla zadania, obciążenie całkowite będzie jak najbliższe wartości 1, ponieważ zamiast przyjmowanej dotychczas sumy obciążeń, wykorzystana jest funkcja EST (linia 38), której zadaniem jest oszacowanie obciążenia przy użyciu funkcji korekty $\eta_{pal}(n)$.

Węzły są alokowane jedynie dla zadań tego samego typu strumieniowego, tzn. dany węzeł może być wykorzystywany przez zadania typu *hd* i *nil* albo przez zadania *pal* i *nil*. Zgodnie z definicją 11 funkcja $v(t_i)$ w przypadku zadania mieszanego bierze pod uwagę (linia 3 na listingu 4) „najcięższy" typ strumienia jaki obsługuje (zgodnie z kolejnością określoną w definicji 11). Ze względu na niskie obciążenie procesora oraz sieci (rozdz. 4.4.3) zadania audio zostały potraktowane tak samo jak bezstrumieniowe (typu *nil*) zadania obliczeniowe.

Zakłada się pewną kolejność realizowania zadań. Przyjmuje się, że węzeł analizujący strumienie danego typu jest w stanie określonym przez ten typ. Wówczas dopuszczalny diagram stanów węzła podaje rys. 55. W stanie *nil* węzeł może zostać obciążony zadaniami przetwarzającymi strumienie *pal* lub *hd*. Kiedy jednak realizuje zadania analizujące strumień *pal*, nie może przyjąć zadań *hd* i na odwrót, węzły realizujące zadania analizujące strumień *hd* nie mogą być obciążane zadaniami związanymi ze strumieniami *pal*.



Rysunek 55: Diagram stanów węzła obliczeniowego, ze względu na typ obsługiwanych zadań.

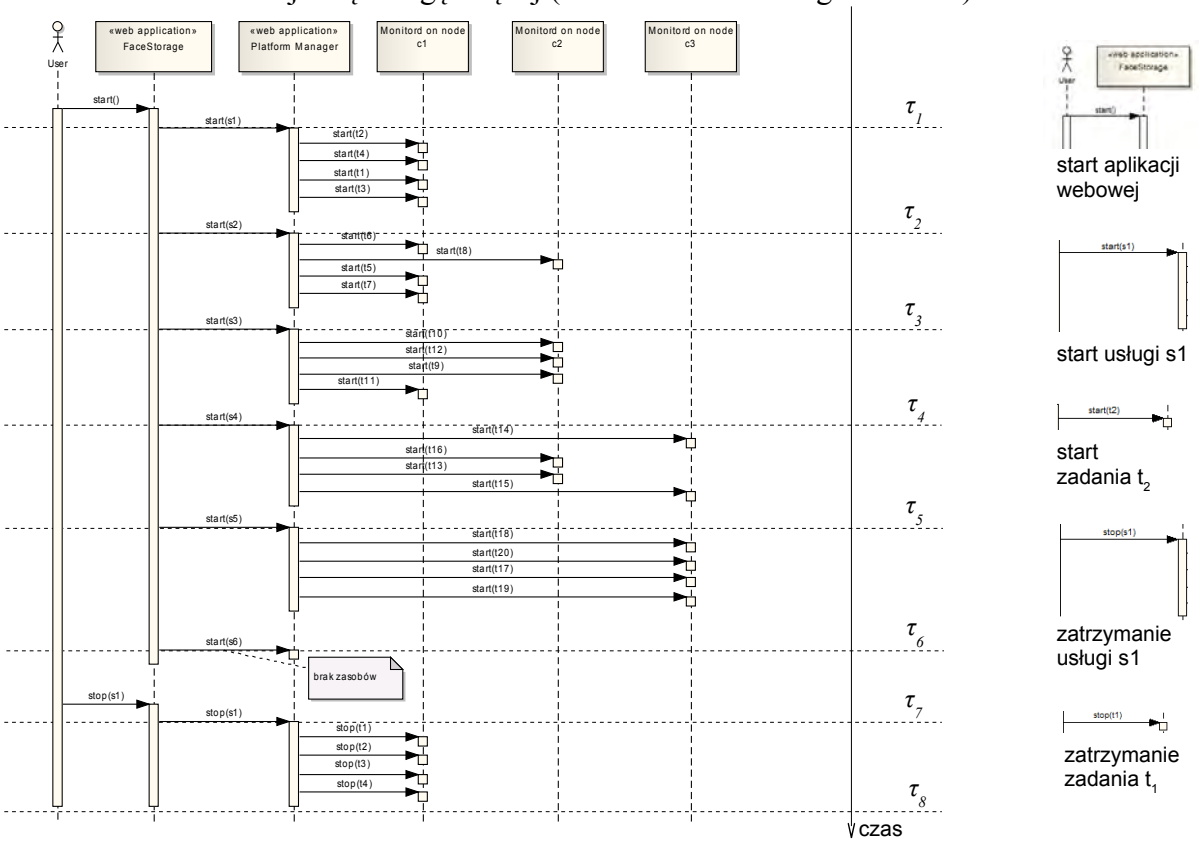
```
1. utwórz listę T* z elementów zbioru T, posortowaną malejąco wg wartości χ_i
2. for i := 1 to |T|
3.     switch ν(T*[i])
4.     case PAL:
5.             znajdź c∈C_PAL gdzie EST(c, T*[i]) jest maksymalne i mniejsze równe 1
6.             if c<>null then
7.                     μ(T*[i]) := c
8.             else
9.                     znajdź c∈C_NIL gdzie EST(c, T*[i]) jest maksymalne i mniejsze równe 1
10.                    if c<>null then
11.                            μ(T*[i]) := c
12.                            C_NIL := C_NIL \ {c}
13.                            C_PAL := C_PAL ∪ {c}
14.                    else
15.                            FAIL(„brak alokacji”)
16.    case HD:
17.            znajdź c∈C_NIL gdzie EST(c, T*[i]) jest maksymalne i mniejsze równe 1
18.            if c<>null then
19.                    μ(T*[i]) := c
20.                    C_NIL := C_NIL \ {c}
21.                    C_HD := C_HD ∪ {c}
22.            else
23.                    FAIL(„brak alokacji”)
24.    case AUD/NIL:
25.            znajdź c∈C_HD gdzie EST(c, T*[i]) jest maksymalne i mniejsze równe 1
26.            if c<>null then
27.                    μ(T*[i]) := c
28.            else
29.                    znajdź c∈C_PAL gdzie EST(c, T*[i]) jest maksymalne i mniejsze równe 1
30.                    if c<>null then
31.                            μ(T*[i]) := c
32.                    else
33.                            znajdź c∈C_NIL gdzie EST(c, T*[i]) jest maks. i mniejsze równe 1
34.                            if c <> null then
35.                                    μ(T*[i]) := c
36.                            else
37.                                    FAIL(„brak alokacji”)

38. function EST(c, t)
38.    Tc := ω(c)
40.    obc := χ(c, {t})
41.    str := ϱ_S(t)
42.    for j := 1 to |Tc|
43.            obc := obc + χ(c, {Tc[j]})
44.            str := str + ϱ_S(Tc[j])
45.    return η_pal(str)·obc
```

Listing 4: Algorytm alokacji MBFD.

5.1.4 Alokacja MBFD dla aplikacji *Faces*

Rozpatrzmy problem alokacji dla aplikacji *Faces* z rodz. 3.8 po uruchomieniu jej na platformie wykorzystującej zaproponowany algorytm MBFD. Na rys. 56 został przedstawiony możliwy przebieg działania aplikacji. W porównaniu z alokacją równoważenia obciążenia (patrz rys. 31) jest możliwe uruchomienie o jedną usługę więcej (uruchomiono 5 usług zamiast 4).



Rysunek 56: Przykład przetwarzania równoległego usług i zadań na platformie KASKADA z wykorzystaniem algorytmu alokacji MBFD.

W tabeli 22 porównano alokacje węzłów obliczeniowych dla kolejno uruchamianych usług dla algorytmów alokacji LB i MBFD. Algorytm MBFD „stara się" używać jak najmniejszej liczby węzłów, dzięki czemu możliwe było w momencie τ_5 uruchomienie kolejnej (piątej) usługi, kiedy LB odmówił jej aktywacji. Różnica w działaniu tych algorytmów jest szczególnie widoczna przy uruchamianiu pierwszych trzech usług, kiedy to LB rozkłada obciążenie na wszystkie węzły natomiast MBFD pozostawia węzły c_2 i c_3 bez zadań, tak długo jak to możliwe.

Tabela 22: Alokacje LB i MBFD dla aplikacji Faces – rozłożenie zadań na wezłach.

Czas	Usługa	Węzeł	LB zadania/obciążenie obliczeniowe węzła		MBFD zadania/obciążenie obliczeniowe węzła	
τ_1	-	c_1	ø	0,00	ø	0,00
		c_2	ø	0,00	ø	0,00
		c_3	ø	0,00	ø	0,00
τ_2	s1	c_1	$\{t_1\}$	0,04	$\{t_1, t_2, t_3, t_4\}$	0,53
		c_2	$\{t_2\}$	0,30	ø	0,00
		c_3	$\{t_3, t_4\}$	0,20	ø	0,00
τ_3	s2	c_1	$\{t_1, t_5, t_6\}$	0,36	$\{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$	0,89
		c_2	$\{t_2\}$	0,30	$\{t_8\}$	0,20
		c_3	$\{t_3, t_4, t_7, t_8\}$	0,41	ø	0,00
τ_4	s3	c_1	$\{t_1, t_5, t_6, t_{11}, t_{12}\}$	0,58	$\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_{11}\}$	0,99
		c_2	$\{t_2, t_9, t_{10}\}$	0,61	$\{t_8, t_9, t_{10}, t_{12}\}$	0,72
		c_3	$\{t_3, t_4, t_7, t_8\}$	0,41	ø	0,00
τ_5	s4	c_1	$\{t_1, t_5, t_6, t_{11}, t_{12}, t_{15}, t_{16}\}$	0,84	$\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_{11}\}$	0,99
		c_2	$\{t_2, t_9, t_{10}\}$	0,61	$\{t_8, t_9, t_{10}, t_{12}, t_{13}, t_{16}\}$	0,98
		c_3	$\{t_3, t_4, t_7, t_8, t_{13}, t_{14}\}$	0,77	$\{t_{14}, t_{15}\}$	0,30
τ_6	s5	c_1	odmowa wykonania usługi s5 – brak zasobów	0,84	$\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_{11}\}$	0,99
		c_2		0,61	$\{t_8, t_9, t_{10}, t_{12}, t_{13}, t_{16}\}$	0,98
		c_3		0,77	$\{t_{14}, t_{15}, t_{17}, t_{18}, t_{19}, t_{20}\}$	0,66
τ_7	s6	c_1	odmowa wykonania usługi s6 – brak zasobów		odmowa wykonania usługi s6 – brak zasobów	0,99
		c_2				0,98
		c_3				0,66
τ_8	-	c_1	$\{t_5, t_6, t_{11}, t_{12}, t_{15}, t_{16}\}$	0,77	$\{t_5, t_6, t_7, t_{11}\}$	0,39
		c_2	$\{t_9, t_{10}\}$	0,32	$\{t_8, t_9, t_{10}, t_{12}, t_{13}, t_{16}\}$	0,98
		c_3	$\{t_7, t_8, t_{13}, t_{14}\}$	0,54	$\{t_{14}, t_{15}, t_{17}, t_{18}, t_{19}, t_{20}\}$	0,66

5.1.5 Skalowalność klastra obliczeniowego

Jak zostało to przedstawione w rozdz. 2.8 istnieje wiele definicji skalowalności. Ze względu na swoją uniwersalność oraz możliwość przedstawienia w formie ilościowej zdecydowno się na wykorzystanie metryki skalowalności zaproponowanej przez Jogalekera w [jogaleker00]. Jego podejście bazuje na pojęciu produktywności, które z kolei jest powiązane z przepływnością badanego systemu, jakością dostarczanych przez niego usług (ang. Quality of Service, QoS) oraz kosztem ich uzyskania. Sama metryka skalowalności (równanie (3)) jest ilorazem produktywności otrzymanej dla skali systemu p do skali referencyjnej – pojedynczego węzła, jednostki obliczeniowej itp.

Sposób oceny skalowalności dla przetwarzania multimedialnego został dokonany poprzez modyfikację analizy zaproponowanej przez Jogalekera. W [jogaleker00] przedstawiono dwa systemy rozproszone, dla których analizowano metryki skalowalności. W pierwszym przypadku był to system zarządzania połączeniami (CMS, ang. Connection Management System) do prywatnych sieci wirtualnych (VPN, ang. Virtual Private Network). Natomiast w drugim rozpatrzono prototyp systemu przetwarzania telefonii sieciowej (CPM, ang. Call Processing System).

W systemie CMS skalą systemu p była liczba replikacji bazy danych, w których znajdowały się dane sieci VPN oraz użytkowników z nich korzystających. Przepustowość systemu $\lambda(p)$ została przedstawiona jako liczba operacji wykonywanych przez system na sekundę, przykładem takiej operacji jest zestawienie sieci VPN. Koszt $\kappa(p)$ jest zależny od liczby replikacji bazy danych, wliczając w to cenę sprzętu i licencji i po znormalizowaniu wynosił $1+0{,}005k$.

Dla systemu CPS skalą systemu p była liczba wykorzystywanych procesorów, przepustowością $\lambda(p)$ liczba połączeń telefonicznych na godzinę, a koszt był bezpośrednio zależny od skali systemu i wynosił w przybliżeniu p. Dla obydwu powyższych systemów funkcja jakości $f(p)$ jest zależna od czasu oczekiwania na wykonanie operacji/połaczeń i po znormalizowaniu wynosiła:

$$f(p)=\frac{1}{1+\frac{\tau(p)}{\tau'}} \quad (20)$$

gdzie $\tau(p)$ jest średnim czasem wykonania danej operacji/wołania dla zadanej skali p, zaś τ' jest czasem oczekiwanym na wynik przez użytkownika.

Dla platformy KASKADA proponujemy wykorzystanie powyższej metryki skalowalności $\varepsilon(p)$ (wzór (3)), dla której skala (p) jest wyznaczona przez liczbę wykorzystywanych węzłów obliczeniowych, koszt jest wprost proporcjonalny tej skali $\kappa(p)=p$, przepustowość $\lambda(p)$ jest mierzona liczbą jednocześnie przetwarzanych strumieni, a funkcja jakości $f(p)$ jest zależna od straty danych w momencie pomiaru τ i jej wartości oczekiwanej $L=0{,}01$. Zatem:

$$f(p)=\frac{1}{1+\frac{\varphi^{\tau}(C_p)}{L}}=\frac{1}{1+100\cdot\varphi^{\tau}(C_p)} \quad (21)$$

gdzie C_p jest klastrem obliczeniowym składającym się z p węzłów ($|C|=p$), realizujących konkretne zadania i wykorzystujących proponowane algorytmy alokacji.

Przyjęta duża ziarnistość przepustowości $(\lambda(p))$, określona jako liczba strumieni, może oznaczać zerową przepustowość dla klastra z pojedynczym węzłem, tzn. $\lambda(1)=0$, występuje wówczas gdy pojedyncza usługa złożona (rozumiana jako zbiór zadań T) nie może zostać uruchomiona na pojedynczym węźle. Z tego powodu zdecydowano na określenie $\lambda(1)$ w zależności od sumy obciążenia wszystkich zadań w usłudze – bez względu na funkcję korekty, według wzoru:

$$\lambda(1)=\frac{1}{\sum_{t\in T}\gamma(c,\{t\})} \quad (22)$$

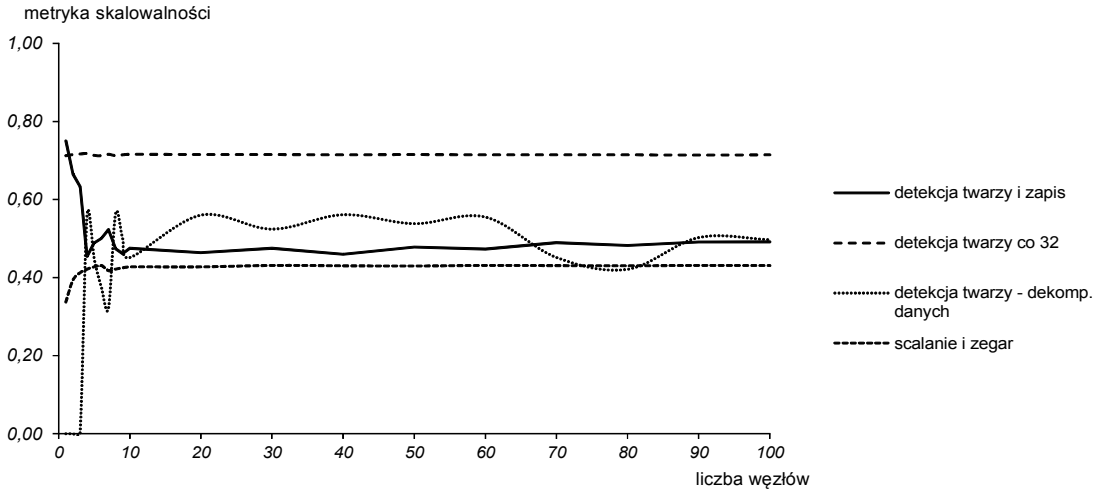
gdzie T jest zbiorem zadań realizujących badaną usługę złożoną.

Tabela 23 przedstawia porównanie poszczególnych wskaźników dla systemów CMS, CPM i KASKADA.

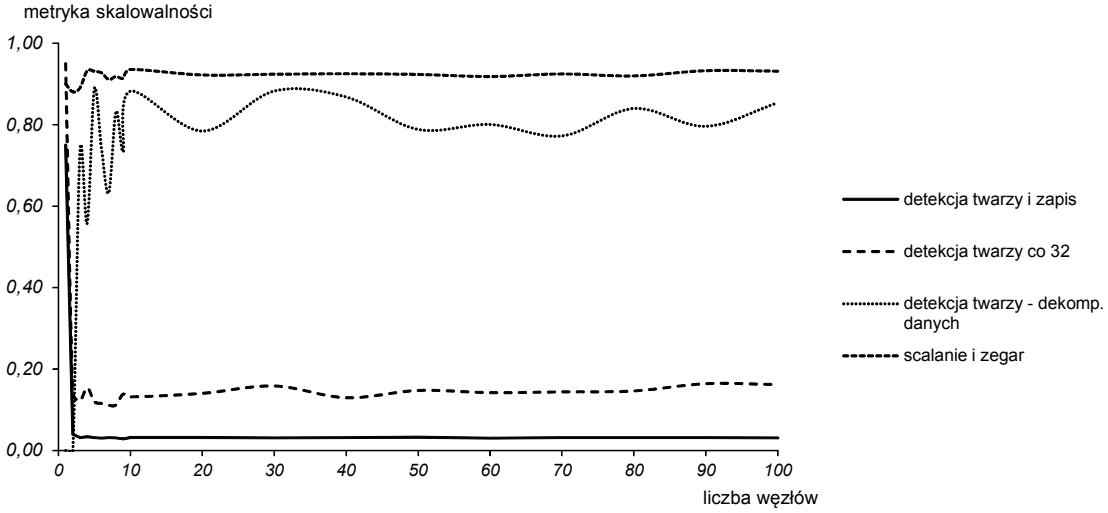
Tabela 23: Porównanie wskaźników metryki skalowalności dla systemów CMS, CPM i KASKADA.

Wskaźnik	CMS	CPM	KASKADA
p	liczba replikowanych baz danych	liczba procesorów w systemie	liczba węzłów klastra
$\lambda(p)$	liczba operacji na sek dla skali p	liczba wołań na godz. dla skali p	liczba przetwarzanych strumieni dla skali p
$\kappa(p)$	$1+0{,}005p$	$\sim p$	p
$f(p)$	znormalizowany czas tworzenia VPN	znormalizowany czas nawiązania połączenia	znormalizowana strata danych

Przeprowadzone badania umożliwiły przedstawienie wykresów metryki skalowalności dla usług wykonywanych na platformie KASKADA i przy alokacji przeprowadzanej za pomocą algorytmów MBFD i BFD, odpowiednio na rys. 57 (a) i (b).



(a)



(b)

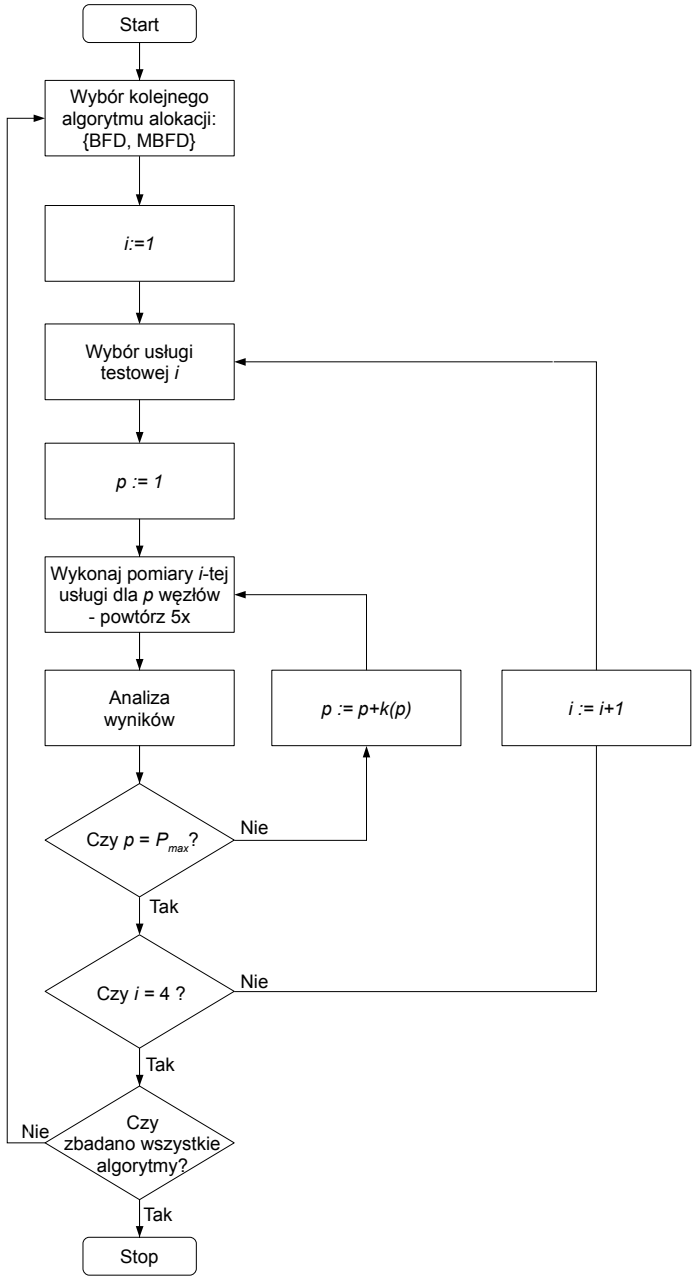
Rysunek 57: Metryki skalowalności dla algorytmów (a) MBFD i (b) BFD.

Wyniki potwierdzają słuszność drugiej tezy tezy rozprawy, że dla algorytmy alokacji MBFD „skalowalność klastra określona wzorem utrzymuje się na stałym poziomie przy wzroście liczby jednocześnie przetwarzanych zadań i strumieni”.

5.1.6 Usługi benchmarkowe dla oceny algorytmów alokacji

Standardowe benchmarki opisane w rozdz. 4.1.1 nie umożliwiają oceny charakterystyk wydajnościowych, w tym skalowalności zadań przetwarzania strumieni multimedialnych. Tym bardziej, należy zastosować własną procedurę ich oceny dla całego klastra. Powinna ona mierzyć obciążenie poszczególnych węzłów, dla zadanej alokacji, jak również brać pod uwagę możliwy spadek wiarygodności: utratę części przetwarzanych danych wyjściowych.

Na rys. 58 przedstawiono proponowaną procedurę pomiaru charakterystyk wydajnościowych. Zakłada się iteracyjne zwiększanie obciążenia klastra poprzez zgłaszanie do wykonania tych samych usług przetwarzających te same strumienie multimedialne.



Rysunek 58: Procedura pomiaru obciążenie obliczeniowego i straty danych dla algorytmów alokacji zadań na węzły klastra obliczeniowego.

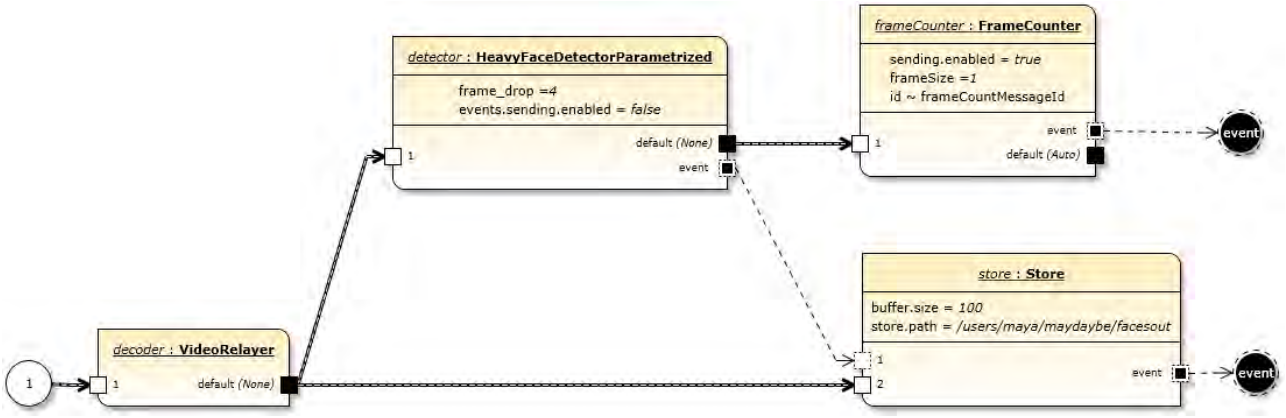
W pierwszym kroku procedury zostaje wybrany badany algorytm alokacji ze zbioru: {BFD, MBFD}. Najpierw badany był algorytm MBFD, a następnie BFD. Kolejnym krokiem jest wybór usług testowych. Zostały one tak dobrane, aby reprezentować typowe algorytmy przetwarzania na platformie KASKADA: rozpoznawanie obrazów zawartych w analizowanych strumieniach, dodawanie informacji tekstowych i graficznych do strumienia czy łączenie dwóch różnych strumieni tego samego typu. Lista usług wraz z informacją nt. ich strumieni wejściowych i wyjściowych została przedstawiona w tabeli 24. Wspólną cechą badanych usług jest konieczność przetwarzania wstępnego, podczas którego następuje dekodowanie strumienia (usługa VideoRelayer na rys. 60) oraz końcow ego gdzie dla strumienia wyjściowego jest wyznaczana strata danych (usługa FrameCounter).

Tabela 24: Testowane usługi złożone i odpowiadające im strumienie multimedialne.

usługa	liczba strumieni wejściowych	liczba strumieni wyjściowych
1. Detekcja twarzy na co 4-tej klatce z zapisem do plików	1	1
2. Detekcja twarzy na każdej klatce z dekompozycją danych	1	1
3. Detekcja twarzy na co 32-giej klatce z generacją zdarzeń	1	1
4. Scalanie strumieni z wpisywaniem czasu	2	1

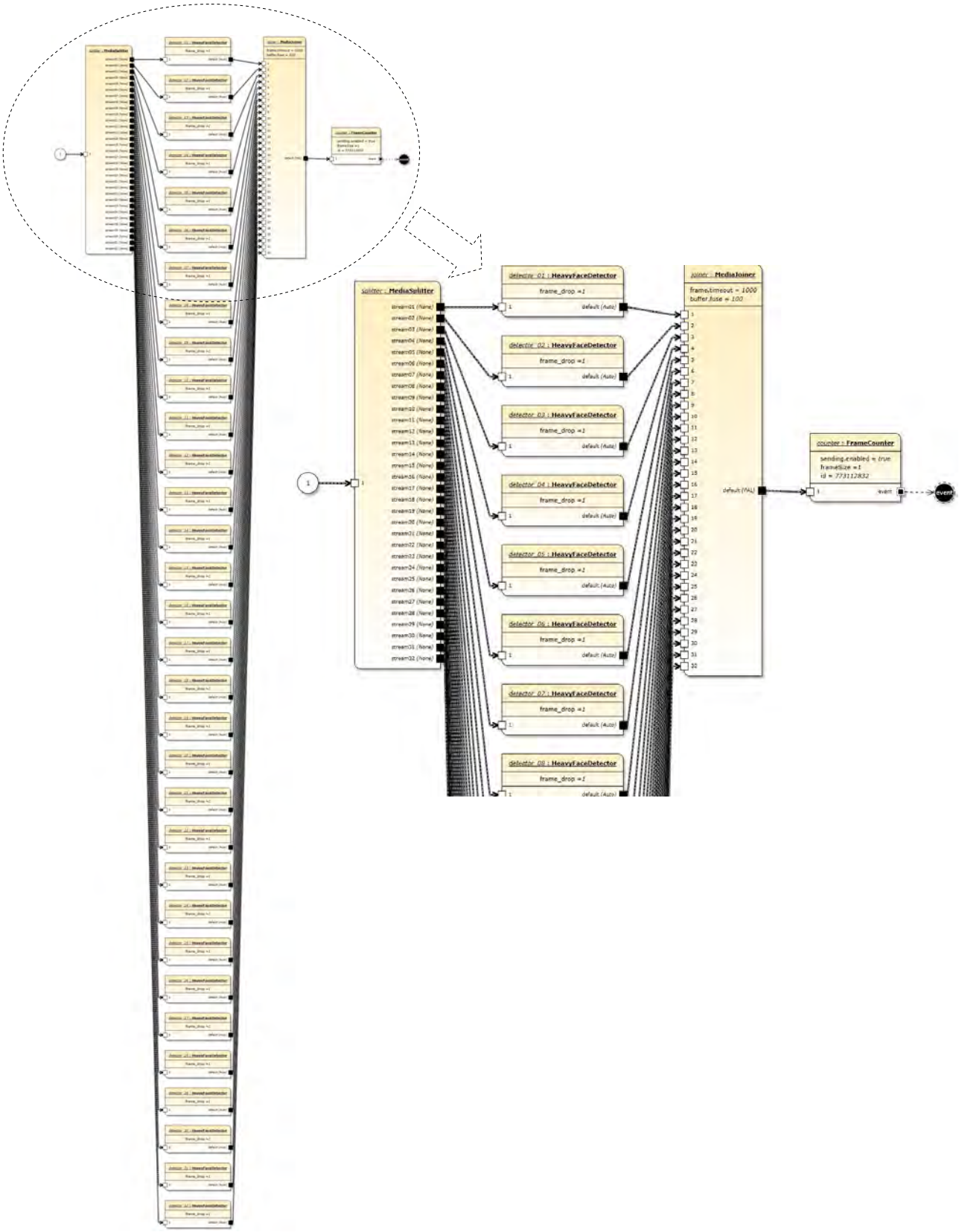
Scenariusze usług testowych zostały przedstawione w języku MSP-ML [bobkowska11], w większości wykorzystują one zadania opisane w tabeli 6, z rozdz 4.1.4 . Pierwszy scenariusz, przedstawiony na rys. 59 umożliwia detekcję twarzy, a następnie jej zapis na dysku – jest on odpowiednikiem usługi złożonej wykorzystywanej w przykładzie z rozdz. 3.4 . Drugi scenariusz z rys. 60 również wykorzystuje detekcję twarzy. W tym przypadku dzięki zrównolegleniu zadań wykonywany jest na każdej klatce (dekompozycja danych). Najpierw strumień dzielony jest na 32 strumienie o odpowiednio mniejszej częstotliwości przepływu klatek, a następnie każdy z tych strumieni jest przetwarzany przez osobne zadanie. Trzecia usługa (rys. 62) wykonuje detekcję twarzy na co 32-giej klatce obrazu, wzbogaca strumień wynikowy o informację o aktualnym czasie przetwarzania (usługa VideoTagger) i dodatkowo generuje strumień komunikatów o zdarzeniach (usługa EventStepper). Ostatnia proponowana usługa (rys. 61) umożliwia scalenie dwóch obrazów, a następnie dodanie informacji o aktualnym czasie ich przetwarzania.

W wybranych usługach wykorzystana jest zarówno dekompozycja danych (usługa 2 z tabeli 24) jak i dekompozycja funkcjonalna (pozostałe usługi). Używane są pojedyncze strumienie (usługi 1, 2 i 3) jak i przetwarzanie wielostrumieniowe (usługa 4). Wykorzystano przesyłanie strumieni danych (we wszystkich usługach) jak i intensywną komunikację do przesyłania informacji o specjalnie w tym celu sztucznie generowanych zdarzeniach (usługa 4). Co więcej, usługa 1 wykorzystuje zdalny system plików do składowania danych (zdjęć cyfrowych) co dodatkowo wpływa na komunikację międzywęzłową. Tak dobrany zestaw usług testowych jest reprezentatywny dla proponowanych badań, ponieważ zawiera różnorodne algorytmy wspierające operacje najczęściej wykonywane na strumieniach multimedialnych.



Rysunek 59: Usługa detekcji twarzy na co 4-tej klatce obrazu z zapisem wyników w postaci zdjęć cyfrowych.

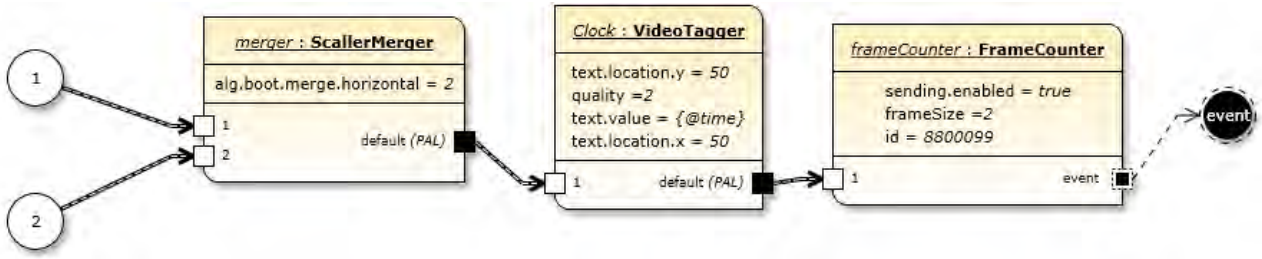
W następnym kroku procedury z rys. 58 są wykonywane pomiary, które dotyczą obciążenia obliczeniowego i straty danych dla p węzłów. Pomiary w ramach tego kroku są wykonywane 5-krotnie, a następnie uśredniane, tak aby uniknąć wpływu przypadkowych czynników – takich jak np. chwilowe natężenie ruchu sieciowego w klastrze oraz określić zakres błędu pomiaru. Zmienna iteracyjna p początkowo jest równa 1, czyli pierwsze 5 pomiarów jest wykonywane dla jednego węzła, następnie jeśli zostanie podjęta decyzja o kontynuacji to wartość zmiennej i zwiększana jest o krok $k(p)$, *tzn.* co 1 dla wartości $i<10$ oraz co 10 dla wartości $i\geq10$ i w ten sposób wykonane są kolejne pomiary.



Rysunek 60: Usługa rozpoznawania twarzy na każdej klatce z wykorzystaniem dekompozycji danych.

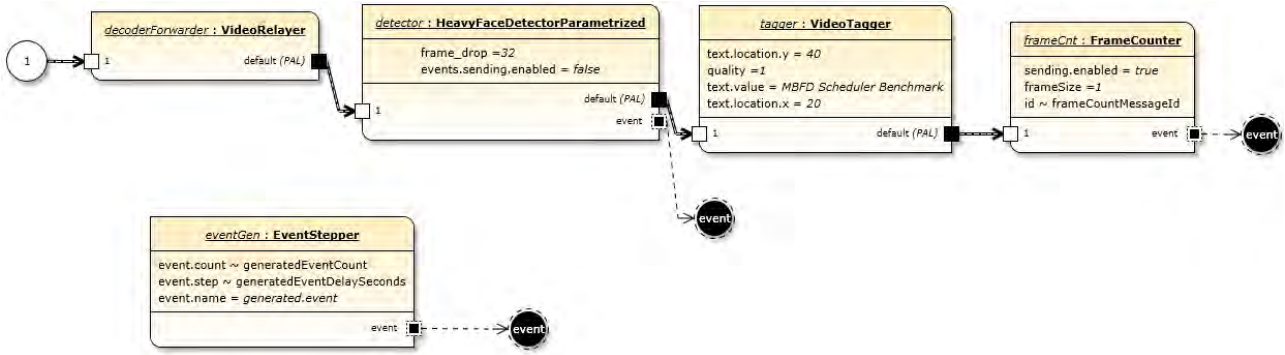
Podczas testów wydajnościowych klastra obliczeniowego wykonywano pomiary następujących charakterystyk:

- maksymalna liczba uruchomionych usług – algorytm alokujący dostaje zlecenia przydziału węzłów obliczeniowych kolejnym usługom, aż do momentu kiedy nie będzie miał dostępnych zasobów obliczeniowych – wszystkie *p* węzłów zostanie wykorzystane,
- strata danych – określona przez stosunek liczby straconych elementów strumienia wyjściowego w czasie eksperymentu do całkowitej liczby elementów strumienia wyjściowego, które powinny być wygenerowane przez algorytm (patrz definicja 13). Wartości charakterystyki były ustalane po wykonaniu każdego pomiaru, na podstawie danych strumieni wejściowych i wynikowych zapisanych w archiwum.



Rysunek 61: Usługa scalająca dwa strumienie wideo i wzbogacająca wynik o informacje o czasie przetwarzania.

Poza tym sprawdzane jest czy $p = P_{max}$, (patrz rys. 59) gdzie P_{max} jest maksymalną liczbą węzłów klastra dostępną do badań, w naszym przypadku $P_{max}=100$. Jeśli badania zostały wykonane dla wszystkich dostępnych węzłów, następuje sprawdzenie czy zbadano wszystkie usługi, jeśli nie to badana jest kolejna usługa, jeśli tak to następuje przejście do kroku, w którym sprawdza się czy zbadano wszystkie strumienie, po którym następuje decyzja, czy należy badać kolejny strumień, czy też zakończyć procedurę.



Rysunek 62: Usługa detekcji twarzy na co 32-giej klatce obrazu, oznaczania czasu przetwarzania na strumieniu wynikowym i generacji strumienia komunikatów zdarzeń.

5.2 Wyniki badań algorytmu alokacji: MBFD

Badania wykonano zgodnie z procedurą opisaną w rozdziale 5.1.6 . Poniżej zostały przedstawione wyniki z kolejnych eksperymentów dla algorytmu alokacji MBFD. Do każdego z nich przedstawiliśmy krótki opis oraz ważniejsze spostrzeżenia. Dla łatwiejszego odczytu wyniki straty danych wyjściowych (definicja 8) na wykresach i w tabelach zostały przedstawione jako wartości procentowe.

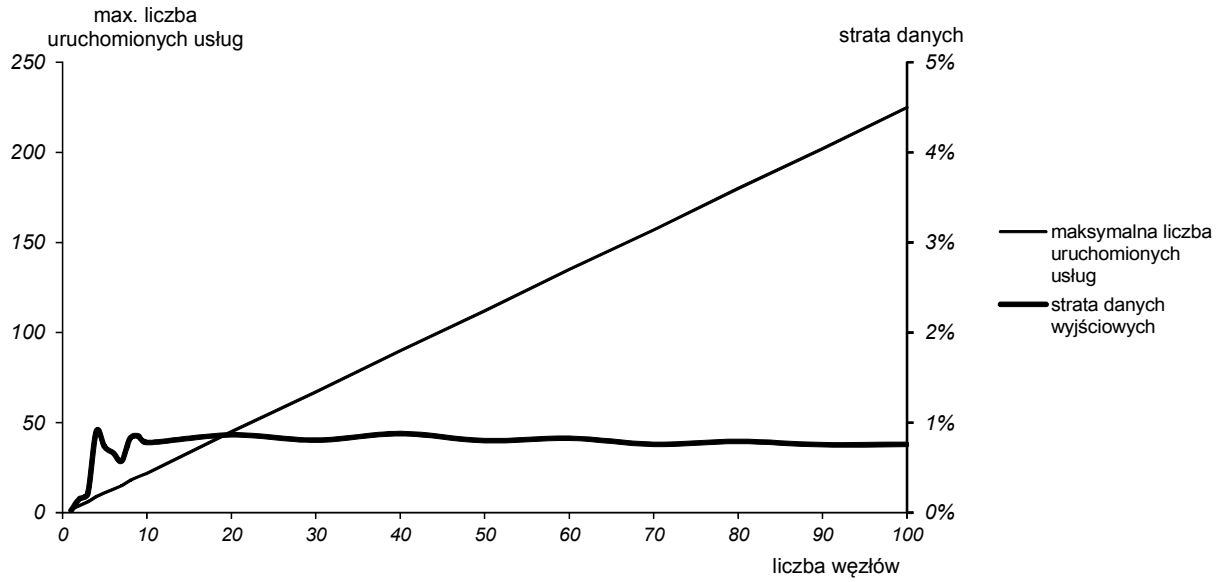
5.2.1 MBFD: Detekcja twarzy na co 4-tej klatce z zapisem do pliku

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji MBFD węzłów na klastrze obliczeniowym dla usługi detekcji twarzy na co 4-tej klatce z zapisem zdjęć do plików.

Tabela 25: Wyniki badań wydajnościowych algorytmu MBFD dla usłgi detekcji twarzy na co 4-tej klatce z zapisem do pliku.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	2	4	6	9	11	13	15	18	20
strata danych	0,00%	0,0249%	0,1554%	0,2140%	0,9029%	0,7297%	0,6644%	0,5745%	0,8254%	0,8521%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	22	45	67	90	112	135	157	180	202	225
strata danych	0,78%	0,8647%	0,8059%	0,8790%	0,8002%	0,8277%	0,7602%	0,7925%	0,7551%	0,7598%

Tabela 25 zawiera wyniki eksperymentu, należy zwrócić uwagę na utrzymujący się niski poziom strat danych – poniżej 1% w całym zakresie badań. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 63.



Rysunek 63: Wyniki badań wydajnościowych algorytmu MBFD dla usługi detekcji twarzy na co 4-tej klatce z zapisem do pliku.

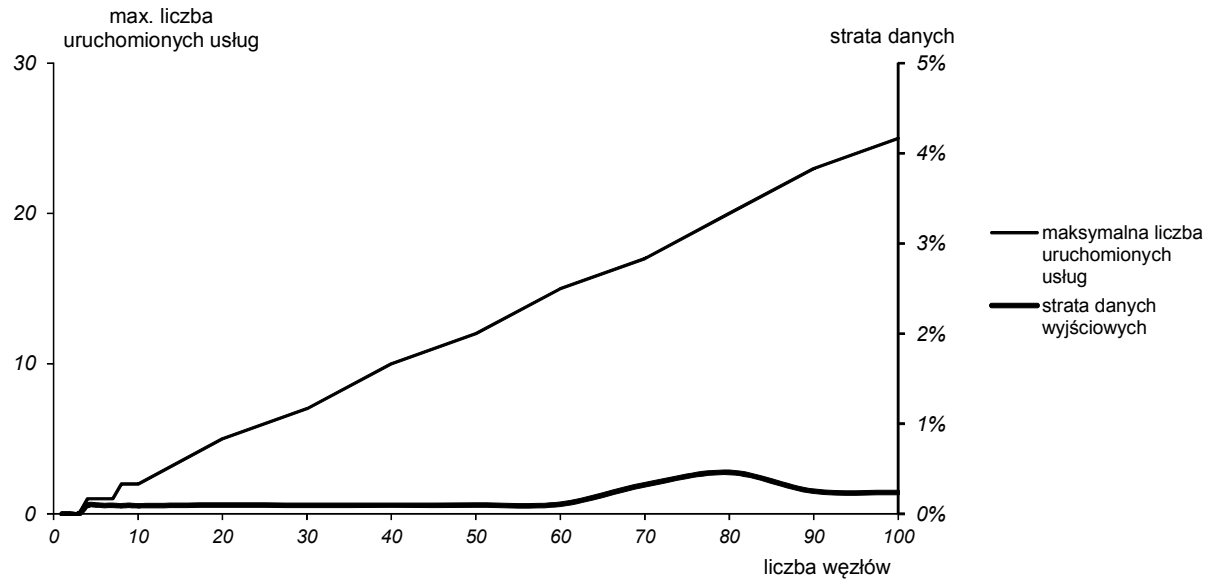
5.2.2 MBFD: Detekcja twarzy na każdej klatce z dekompozycją danych

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji MBFD węzłów na klastrze obliczeniowym dla usługi detekcji twarzy na każdej klatce z dekompozycją danych.

Tabela 26: Wyniki badań wydajnościowych algorytmu MBFD dla usłgi detekcji twarzy na każdej klatce z dekompozycją danych.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	0	0	0	1	1	1	1	2	2
strata danych	0,00%	0,0000%	0,0000%	0,0000%	0,0970%	0,1000%	0,0940%	0,0950%	0,0910%	0,0950%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	2	5	7	10	12	15	17	20	23	25
strata danych	0,0910%	0,0990%	0,0960%	0,0970%	0,0980%	0,1090%	0,3240%	0,4610%	0,2520%	0,2380%

Tabela 26 zawiera wyniki eksperymentu, należy zwrócić uwagę na utrzymujący się niski poziom strat danych – poniżej 1% w całym zakresie badań. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 64.



Rysunek 64: Wyniki badań wydajnościowych algorytmu MBFD dla usługi detekcji twarzy na każdej klatce z dekompozycją danych.

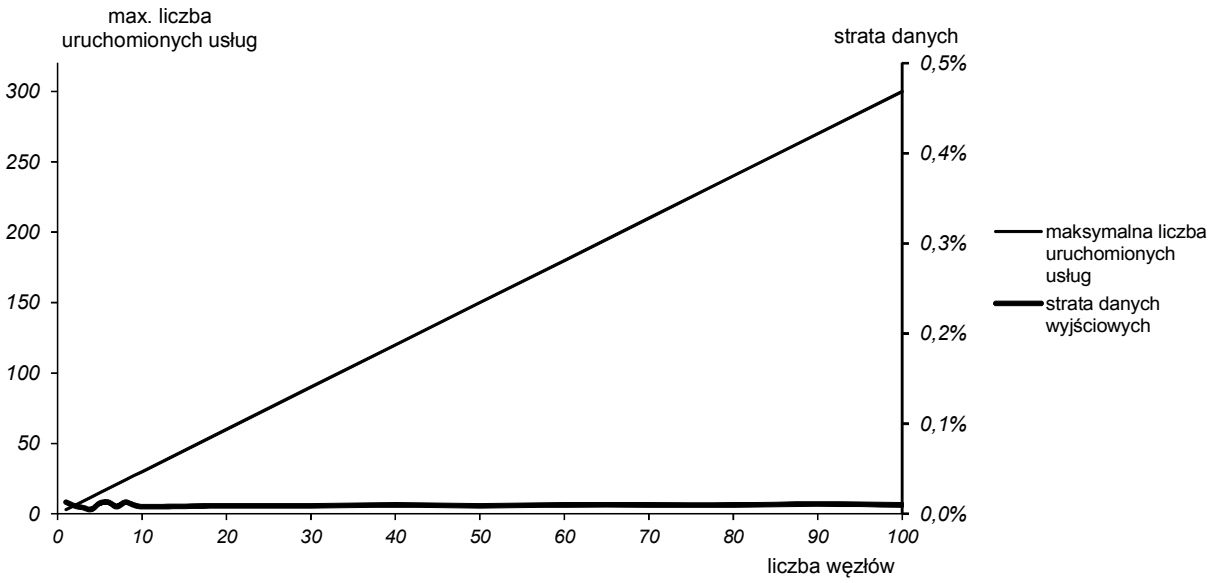
5.2.3 MBFD: Detekcja twarzy na co 32-giej klatce z generacją zdarzeń

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji MBFD węzłów na klastrze obliczeniowym dla usługi detekcji twarzy na co 32-giej klatce z generacją zdarzeń.

Tabela 27: Wyniki badań wydajnościowych algorytmu MBFD dla usłgi detekcji twarzy na co 32-giej klatce z generacją zdarzeń.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	3	6	9	12	15	18	21	24	27
strata danych	0,00%	0,0130%	0,0090%	0,0070%	0,0050%	0,0120%	0,0130%	0,0080%	0,0130%	0,0100%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	30	60	90	120	150	180	210	240	270	300
strata danych	0,0080%	0,0090%	0,0090%	0,0100%	0,0090%	0,0100%	0,0100%	0,0100%	0,0110%	0,0100%

Tabela 27 zawiera wyniki eksperymentu, należy zwrócić uwagę na utrzymujący się niski poziom strat danych – poniżej 1% w całym zakresie badań. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 65.



Rysunek 65: Wyniki badań wydajnościowych algorytmu MBFD dla usługi detekcji twarzy na co 32-giej klatce z generacją zdarzeń.

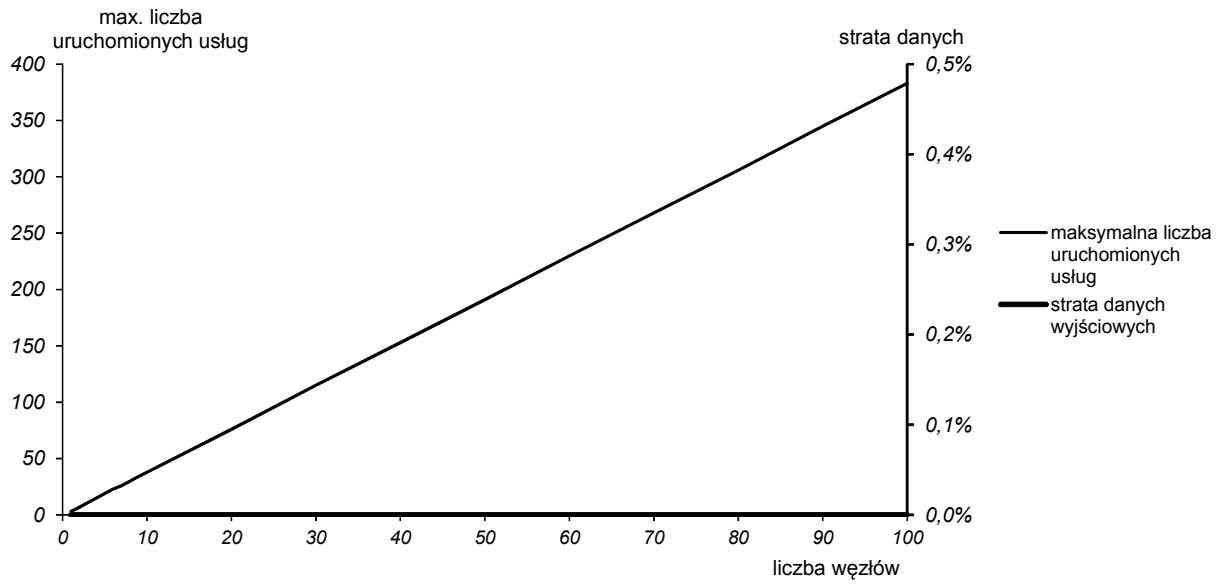
5.2.4 MBFD: Scalanie strumieni z wpisywaniem czasu

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji MBFD węzłów na klastrze obliczeniowym dla usługi scalania strumieni z wpisywaniem czasu.

Tabela 28: Wyniki badań wydajnościowych algorytmu MBFD dla usługi scalania strumieni z wpisywaniem czasu.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	3	7	11	15	19	23	26	30	34
strata danych	0,00%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	38	76	115	153	191	230	268	306	345	383
strata danych	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%	0,0000%

Tabela 28 zawiera wyniki eksperymentu, należy zwrócić uwagę na utrzymujący się bardzo niski poziom strat danych: 0% w całym zakresie badań. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 66.



Rysunek 66: Wyniki badań wydajnościowych algorytmu MBFD dla usługi scalania strumieni z wpisywaniem czasu.

5.3 Wyniki badań algorytmu alokacji: BFD

Badania wykonano zgodnie z procedurą opisaną w rozdziale 5.1.6 . Poniżej zostały przedstawione wyniki z kolejnych eksperymentów dla algorytmu alokacji BFD. Do każdego z nich przedstawiliśmy krótki opis oraz ważniejsze spostrzeżenia. Dla łatwiejszego odczytu wyniki straty danych wyjściowych (definicja 8) na wykresach i w tabelach zostały przedstawione jako wartości procentowe.

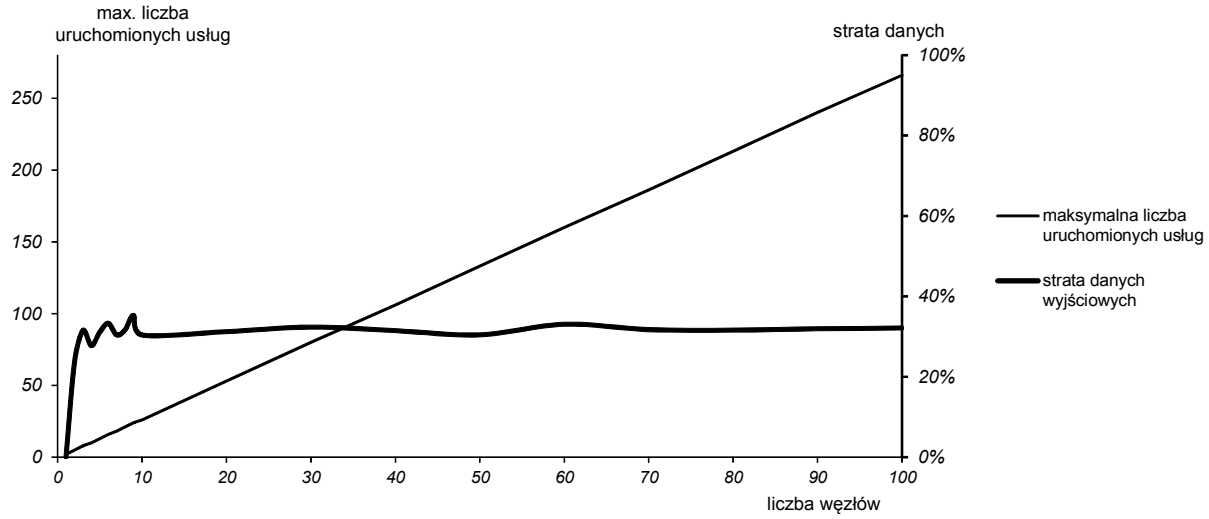
5.3.1 BFD: Detekcja twarzy na co 4-tej klatce z zapisem do plików

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji BFD węzłów na klastrze obliczeniowym dla usługi detekcji twarzy na co 4-tej klatce z zapisem zdjęć do plików.

Tabela 29: Wyniki badań wydajnościowych algorytmu BFD dla usłgi detekcji twarzy na co 4-tej klatce z zapisem do pliku.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	2	5	8	10	13	16	18	21	24
strata danych	0,00%	0,041%	23,597%	31,545%	27,758%	31,150%	33,297%	30,407%	31,733%	35,267%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	26	53	80	106	133	160	186	213	240	266
strata danych	30,388%	31,234%	32,375%	31,468%	30,433%	33,074%	31,728%	31,596%	31,935%	32,117%

Tabela 29 zawiera wyniki eksperymentu, należy zwrócić uwagę na utrzymujący się wysoki poziom strat danych, ponad 30%. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 67. Można zauważyć jest liniowy wzrost liczby uruchomionych usług w zależności od liczby wykorzystywanych węzłów, jednakże poziom strat danych uniemożliwia wykorzystanie badanego algorytmu alokacji do praktycznych zastosowań.



Rysunek 67: Wyniki badań wydajnościowych algorytmu BFD dla usługi detekcji twarzy na co 4-tej klatce z zapisem do plików.

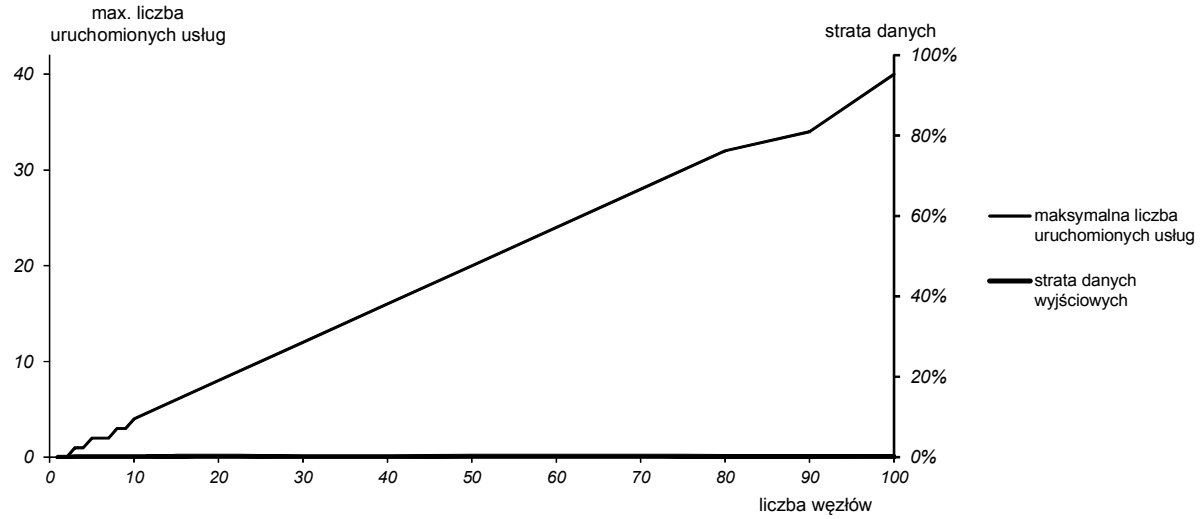
5.3.2 BFD: Detekcja twarzy na każdej klatce z dekompozycją danych

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji BFD węzłów na klastrze obliczeniowym dla usługi detekcji twarzy na każdej klatce z dekompozycją danych.

Tabela 30: Wyniki badań wydajnościowych algorytmu BFD dla usłgi detekcji twarzy na każdej klatce z dekompozycją danych.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	0	0	1	1	2	2	2	3	3
strata danych	0,00%	0,000%	0,000%	0,117%	0,103%	0,110%	0,110%	0,111%	0,110%	0,119%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	4	8	12	16	20	24	28	32	34	40
strata danych	0,116%	0,255%	0,116%	0,135%	0,249%	0,230%	0,275%	0,173%	0,168%	0,153%

Tabela 30 zawiera wyniki eksperymentu, należy zwrócić uwagę, że w przeciwieństwie do pozostałych usług poziom strat danych jest znikomy (poniżej 0.002). Jest to spowodowane dekompozycją danych – strumienie zostały „rozrzedzone” 32-krotnie, dzięki czemu zadania zachowywały się jak typowe zadania obliczeniowe, a nie strumieniowe. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 68. Można również zauważyć liniowy wzrost liczby uruchomionych usług w zależności od liczby wykorzystywanych węzłów.



Rysunek 68: Wyniki badań wydajnościowych algorytmu BFD dla usługi detekcji twarzy na każdej klatce z dekompozycją danych.

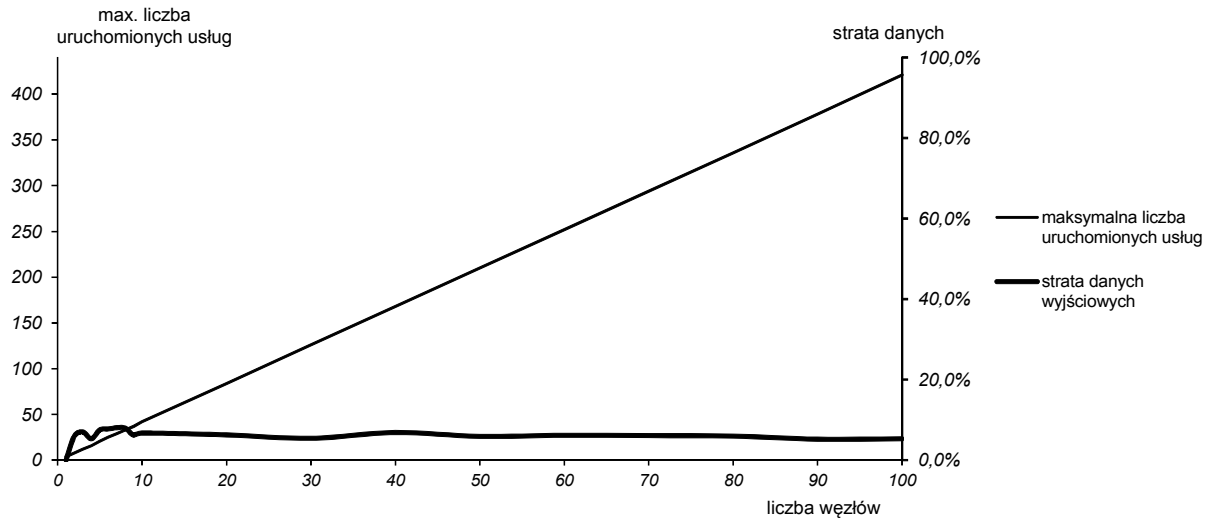
5.3.3 BFD: Detekcja twarzy na co 32-giej klatce z generacją zdarzeń

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji BFD węzłów na klastrze obliczeniowym dla usługi detekcji twarzy na co 32-giej klatce z generacją zdarzeń.

Tabela 31: Wyniki badań wydajnościowych algorytmu BFD dla usługi detekcji twarzy na co 32-giej klatce z generacją zdarzeń.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	4	8	12	16	21	25	29	33	37
strata danych	0,00%	0,025%	6,048%	7,017%	5,314%	7,523%	7,759%	8,069%	7,974%	6,212%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	42	84	126	168	210	252	294	336	378	421
strata danych	6,750%	6,281%	5,445%	6,877%	5,929%	6,186%	6,092%	5,986%	5,225%	5,320%

Tabela 31 zawiera wyniki eksperymentu, należy zwrócić uwagę na utrzymujący się wysoki poziom strat danych, ponad 6%. Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 69. Można zauważyć jest liniowy wzrost liczby uruchomionych usług w zależności od liczby wykorzystywanych węzłów, jednakże poziom strat danych uniemożliwia wykorzystanie badanego algorytmu alokacji do praktycznych zastosowań.



Rysunek 69: Wyniki badań wydajnościowych algorytmu BFD dla usługi detekcji twarzy na co 32-giej klatce z generacją zdarzeń.

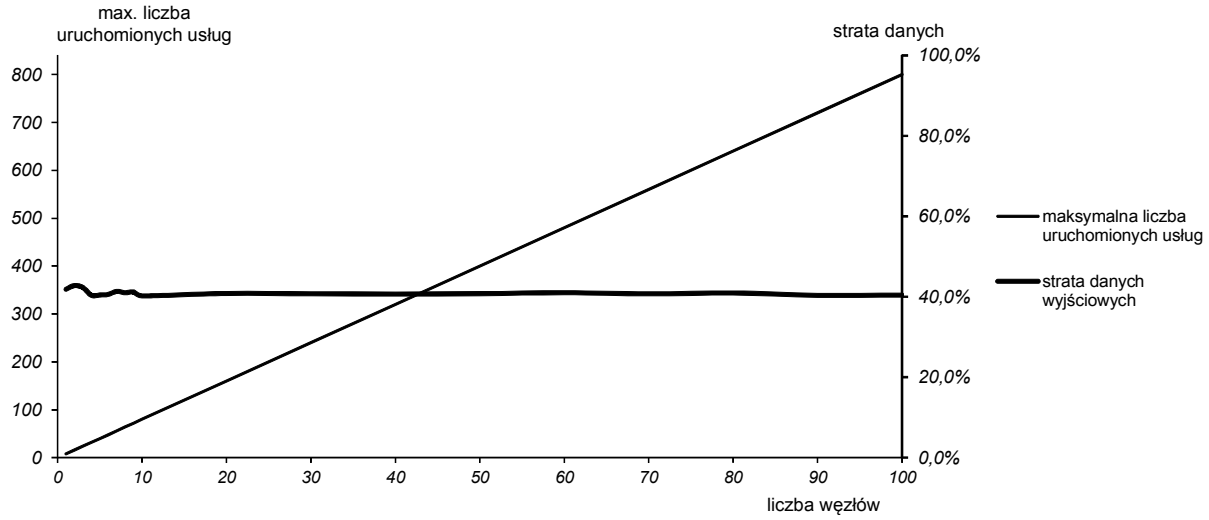
5.3.4 BFD: Scalanie strumieni z wpisywaniem czasu

Eksperyment umożliwia zbadanie zachowania się algorytmu alokacji BFD węzłów na klastrze obliczeniowym dla usługi scalania strumieni z wpisywaniem czasu.

Tabela 32: Wyniki badań wydajnościowych algorytmu BFD dla usłgi scalania strumieni z wpisywania czasu.

liczba węzłów	0	1	2	3	4	5	6	7	8	9
liczba uruchomionych usług	0	8	16	24	32	40	48	56	64	72
strata danych	0,00%	41,866%	42,811%	42,318%	40,372%	40,447%	40,597%	41,346%	41,000%	41,206%
liczba węzłów	10	20	30	40	50	60	70	80	90	100
liczba uruchomionych usług	80	160	240	320	400	480	560	640	720	800
strata danych	40,235%	40,834%	40,752%	40,699%	40,774%	41,009%	40,721%	40,940%	40,366%	40,417%

Tabela 32 zawiera wyniki eksperymentu, należy zwrócić uwagę na utrzymujący się wysoki poziom strat danych (ponad 40%). Powyższe wyniki zostały zilustrowane graficznie na wykresie z rys. 67. Można zauważyć jest liniowy wzrost liczby uruchomionych usług w zależności od liczby wykorzystywanych węzłów, jednakże poziom strat danych uniemożliwia wykorzystanie badanego algorytmu alokacji do praktycznych zastosowań.



Rysunek 70: Wyniki badań wydajnościowych algorytmu BFD dla usługi scalania strumieni z wpisywaniem czasu.

5.4 Analiza porównawcza algorytmów alokacji

Algorytm alokacji MBFD został porównany z dwoma standardowymi podejściami do alokacji zasobów obliczeniowych w systemach rozproszonych. Pierwsze z nich dotyczy równomiernego rozłożenie obciążenia (LB, ang. Load Balancing). Drugie zaś typowe dla systemów czasu rzeczywistego, dotyczy maksymalnego lokowania zadań na każdy z węzłów [oh95] zgodnie z algorytmem pakowania koszy [garey79].

5.4.1 Porównanie algorytmu alokacji MBFD z LB

Typowy algorytm alokacji równomiernie rozkładający obciążenie na węzły (LB) przypisuje kolejne uruchamiane zadanie do najmniej obciążonego węzła, a co za tym idzie równoważy obciążenie wszystkich węzłów. Dla klastra C składającego się z $|C|$ węzłów oraz jego sumarycznego obciążenia, które oznaczymy jako K, obciążenie każdego węzła będzie dążyć do średniej czyli: $K/|C|$ (przypadek idealny, w którym obciążenie będzie rozłożone całkowicie równomiernie). Natomiast dla rozważnego algorytmu minimalizacji liczby wykorzystanych węzłów (MBFD), węzły będą albo zapakowane maksymalnie, albo puste, również dając średnie obciążenie na poziomie $K/|C|$. W takim przypadku żądanie uruchomienia kolejnej usługi z zadaniem t_k, powodującym duże obciążenie $y_k>1-K/|C|$ spowoduje odmowę jej uruchomienia ze względu na brak węzła, który miałby wystarczające zasoby. Natomiast w przypadku algorytmu MBFD, jeśli pozostał jeszcze choć jeden wolny węzeł takie zadanie będzie mogło być uruchomione. Na przykład przy średnim obciążeniu węzła równym 0,75 (w przypadku idealnej alokacji równomiernego obciążenia), usługa z zadaniem obciążającym węzeł więcej niż 0,25 (np. *face_detector* obciąża węzeł na poziomie: 0,3) nie będzie mogła znaleźć żadnej alokacji (por. przykład w rozdz. 5.1.4).

5.4.2 Porównanie algorytmu alokacji MBFD z BFD

Strategia pakowania zadań na kolejne węzły, w kontekście platformy KASKADA została dokładnie przebadana w rozdz. 5.3 niniejszej rozprawy. Nie bierze ona pod uwagę nieliniowego zwiększania się wymaganego obciążenia wraz ze wzrostem liczby analizowanych strumieni multimedialnych – może to spowodować, że zbyt wiele zadań strumieniowych zostanie zaalokowanych na tym samym węźle, co spowoduje stratę części danych przetwarzanych przez algorytmy tych zadań. To zjawisko sygnalizują wyniki badań algorytmu BFD, dla którego wykazują

bardzo wysokie straty danych wyjściowych, sięgające nawet poziomu 0,4.

Przeprowadzone badania umożliwiają również analizę skalowalności z wykorzystaniem metryki skalowalności (wzór (3)). Z badań wynika, że skalowalność platformy wykorzystującej algorytm BFD jest niska w porównaniu do skalowalności tej platformy wykorzystującej algorytm alokacji MBFD (por. rys. 57). Jest to spowodowane dużym znaczeniem funkcji jakości *f(p)* w przyjętej metryce skalowalności (wzór (21)). Wyjątek stanowi usługa detekcji twarzy na każdej klatce, kiedy to „rozrzedzenie" danych spowodowane ich dekompozycją powoduje, że zadania strumieniowe zachowują się podobnie do typowych zadań obliczeniowych (por. rozdz. 5.2.2) oraz usługa scalania i zegara dla której nawet przy jednym węźle występuje bardzo duża strata danych (ok. 0,4) i utrzymuje się na tym samym poziomie, przez co metryka skalowalności jest wysoka.

Podsumowując, ze względu na powyższe spostrzeżenia wykorzystanie algorytmu alokacji **MBFD** jest w pełni uzasadnione i praktycznie akceptowalne dla **platformy KASKADA**.

6 Uwagi końcowe

Skomentowano podstawowe własności platformy KASKADA w odniesieniu do tez rozprawy doktorskiej oraz z nawiązaniem do wyników badań eksperymentalnych. Wykazano, że procedura zintegrowanego zarządzania zaimplementowana na platformie spełnia założone oczekiwania. Zaproponowano kierunki dalszych prac badawczych odnośnie rozwoju platformy jak też jej praktycznego wykorzystania.

6.1 Wykazanie tez rozprawy

W rozprawie doktorskiej rozpatrzono problematykę zarządzania równoległym systemem komputerowym przetwarzającym strumienie multimedialne na bazie homogenicznego klastra obliczeniowego, ze szczególnym uwzględnieniem charakterystyk jakościowych: wydajności (rozumianej jako zachowanie właściwego poziomu przyspieszenia obliczeń) i wiarygodności (rozumiana jako niska strata przetwarzanych danych).

W rozdziale 4 rozważano i wykazano pierwszą tezę rozprawy:

Przy akceptowanej wiarygodności przetwarzania strumieni, obciążenie węzła dla analizowanej klasy algorytmów i dla danego typu przetwarzanego strumienia wzrasta nieliniowo wraz z liczbą przetwarzanych strumieni, przy czym wzrost ten można oszacować tzw. funkcją korekty podającą przyrost obciążenia w stosunku do jego wzrostu liniowego.

W tym celu wykonano eksperymenty na pojedynczym węźle klastra obliczeniowego $c \in C$ z wykorzystaniem zadań obliczeniowych określanych zgodnie z definicją 4: $t_i=(a_i, SI_i, SO_i, \gamma_i)$, gdzie a_i jest określone przez tabelę 6, $SI_i=\{si_{1i}\}$, si_{1i} określa tabela 5, $|SI_i|=1$ z wyjątkiem algorytmu *Scalenia obrazów,* dla którego wynosi 2, $|SI_i|=2$. γ_i określa się dla a_i poprzez pomiary obciążenia węzła z wykorzystaniem programu top [top] przy wykonaniu tylko a_i dla $s_i \in SI_i$ z tabeli 5.

Eksperyment polega na obciążaniu pojedynczego węzła c rosnącą liczbą tych samych zadań t_i wykonywanych jednocześnie. Rozpoczynamy od jednego zadania t_i, następnie zwiększając ciągle o jeden, aż do momentu zaobserwowania strat danych w strumieniu wyjściowym większych od 1% (patrz wzór 8). Podczas eksperymentów mierzono obciążenie obliczeniowe węzła $\gamma(c, T)$ oraz stratę danych $\varphi(c, T)$. Znając obciążenie tylko dla pojedynczego zadania liniowa estymacja obciążenie obliczeniowego dla jednoczesnego wykonania n zadań jest określana w sposób następujący:

$$\gamma\left(c,\{t_1=t,t_2=t,\ldots,t_n=t\}\right)=n\cdot\gamma\left(c,\{t\}\right) \quad (23)$$

Dla zwiększenia wiarygodności pomiarów każdy pomiar jest powtarzany 20 razy, a wyniki są uśredniane. Funkcja korekty, wprowadzająca poprawkę do estymacji liniowej z równania (23) określa się następująco:

$$\eta(c,t,n)=\frac{\gamma\left(c,\{t_1=t,t_2=t,\ldots,t_n=t\}\right)}{n\cdot\gamma\left(c,\{t\}\right)} \quad (24)$$

Dokonano pomiarów dla $\gamma(c,\{t_1=t, t_2=t, \ldots, t_n=t\}$ dla różnych zadań t zgodnie z tabelą 6. Podczas eksperymentów zauważono, że istotny wpływ na obciążenie węzła ma przede wszystkim typ strumienia danych (h) oraz ich liczba (n), a także obciążenie dodatkowo generowane przez system operacyjny węzła (Linux), zwłaszcza przy wykonywaniu algorytmów o niskiej złożoności (przekaźnik i zegar). Dlatego do obliczenia funkcji korekty wzięto pod uwagę pomiary obciążenia γ, uzyskane dla małych n przy wykorzystaniu złożonych algorytmów oraz dla dużych n przy wykorzystaniu mniej złożonych algorytmów. To pozwoliło na uniezależnienie funkcji korekty od algorytmu tzn.:

$$\eta(c,t,n)=\eta_h(c,n) \quad (25)$$

gdzie $h\in H$, $H=\{hd, pal, aud, nil\}$. Otrzymane dla $h=pal$ wartości $\eta_h(c, n)$ przedstawiono w tabeli 11. Zakres został ograniczony do $n<21$, gdyż dalszy wzrost n powoduje nieakceptowalną stratę danych. W przypadku $h=hd$ funkcja korekty nie jest konieczna gdyż węzeł może obsługiwać tylko jedno takie zadanie, natomiast w przypadku $h=nil$ i $h=aud$ obciążenia węzła są liniowe, ale znikome nawet przy $n=150$ strumieni.

Tak wyznaczoną funkcję korekty (w formie tabelarycznej) poddano dalszej weryfikacji w następującym eksperymentach. Wzięto pod uwagę 4 typy nowych algorytmów (detekcja twarzy na co 4-tej klatce, wykrywanie krawędzi, maska tła, scalanie obrazów) i porównano estymowane obciążenie węzłów (z uwzględnieniem funkcji korekty) z rzeczywistym obciążeniem dla tych samych strumieni typu PAL. Z rys. 43-46 wynika, że opracowana funkcja korekty dobrze estymuje obciążenie obliczeniowe węzłów, tzn. kwadrat współczynnika korelacji Pearsona wynosi od 0,95-0.99.

W rozdziale 5 rozważano i wykazano drugą tezę rozprawy:

Dla zadanej klasy algorytmów analizy strumieni oraz dla zaproponowanego algorytmu alokacji zadań na węzły klastra (wykorzystującego funkcję korekty), skalowalność klastra określona wzorem (3) utrzymuje się na stałym poziomie przy wzroście liczby jednocześnie przetwarzanych zadań i strumieni pod warunkiem, proporcjonalnego do niego wzrostu liczby węzłów obliczeniowych zaangażowanych w to przetwarzanie.

Zaprezentowano pięć algorytmów alokacji zadań uruchamianej usługi złożonej na węzły obliczeniowe klastra w momencie τ, pracujące przy założeniu liniowego wzrostu obciążenia od liczby wykonywanych zadań, zgodnie ze wzorem (23). Wszystkie te algorytmy oparte są na heurystyce rozwiązującej problem pakowania koszy zmiennej wielkości. Algorytmy FF (ang. First Fit) i FFD (ang. First Fit Descending) z listing 1 alokują zadania ze zbioru zadań na pierwszy pasujący węzeł, w wersji FFD, zadania te są posortowane malejąco. Algorytmy BF (ang. Best Fit) i BFD (ang. Best Fit Descending) z listingu 2 alokuje zadania do węzłów najbardziej do nich pasujących – takich, które po alokacji będą najbardziej „wypełnione”, tzn. będą miały jak największe obciążenie:

$$\max_{c\in C}\left(\gamma(c,\omega^{\tau}(c))+\sum_{t\in\mu_C^{\tau}(c)}\gamma(c,\{t\})\right) \quad (26)$$

ale zgodnie ze wzorem (18) nie większe od 1. Kolejny algorytm alokacji (patrz listing 3) oparty jest na procedurze rozwiązującej problem SSP (ang. Subset Sum Problem) w ten sposób aby dobrać jak najbardziej pasujący podzbiór zadań dla kolejnego węzła.

Wykonano symulację działania powyższych algorytmów na klastrze obliczeniowym, przy założeniu następującego stanu klastra C w momencie τ alokacji zbioru zadań T: losowe obciążenie alokowanych zadań (rozkład równomierny), stała liczba wolnych i częściowo zajętych węzłów, odpowiednio 20 i 8, losowego obciążenie częściowo zajętych węzłów (rozkład równomierny). Wyniki symulacji dla zbioru zadań o wielkości $|T|=2, 4, 6, ..., 20$ zostały przedstawione w tabeli 21, a ich graficzne reprezentacja na wykresie z rys. 54. Najczęściej najlepsze rezultaty osiągnął algorytm BFD, następnie w kolejności malejącej SSP, FFD, BF i FF. Dlatego do dalszych rozważań wybrano właśnie algorytm BFD.

Następnie zaproponowano zmodyfikowaną wersję najlepiej ocenionego algorytmu: MBFD (ang. Modified Best Fit Descending). W wyniku zmian uwzględniono w nim 3 przypadki dotyczące różnych typów strumieni obsługiwanych przez alokowane zadania *hd, pal* i *aud/nil* oraz wprowadzono funkcję korekty $\eta_{pal}(c, n)$ dla zadań obsługujących strumienie *pal*. Tak skonstruowany algorytm został przedstawiony na listingu 4.

Wykonano eksperymenty na homogenicznym klastrze obliczeniowym przy założeniu zmiennej liczby wykorzystywanych węzłów $|C|$ w przedziale 1-100, dwóch różnych algorytmów alokacji: BFD i MBFD oraz z wykorzystaniem 4 różnych typów usług złożonych: (1) Detekcja twarzy na co 4-tej klatce z zapisem do plików, (2) Detekcja twarzy na każdej klatce z dekompozycją danych, (3) Detekcja twarzy na co 32-giej klatce z generacją zdarzeń oraz (4) Scalanie strumieni z wpisywaniem czasu.

Eksperyment polega na obciążaniu klastra obliczeniowego usługami jednego typu, uruchamianymi wielokrotnie i wykonywanymi jednocześnie, aż do momentu kiedy algorytm alokacji odmówi uruchomienia kolejnej usługi. Rozpoczynamy od jednego węzła, następnie dwóch zwiększając rozmiar klastra o 1, a dla $|C|>9$ o 10 węzłów. Podczas eksperymentu mierzono maksymalną liczbę uruchomionych usług złożonych danego typu oraz średnią stratę danych wyrażoną wzorem (11). Wyniki przedstawione w rozdz. 5.2 i 5.3 wykazały mniejszą liczbę uruchamianych usług oraz mniejszą stratę danych dla algorytmu alokacji MBFD w porównaniu z algorytmem BFD.

Zgodnie z założeniami z rozdz. 2.9 wykorzystana została metryka skalowalności według wzoru (3), co po uwzględnieniu założeń z rozdz. 5.1.5 daje:

$$\varepsilon(p)=\frac{\lambda(p)\left(1+100\varphi^{\langle \tau s,\tau\rangle}(C_1)\right)\sum_{t\in T}\gamma(c,\{t\})}{p\left(1+100\varphi^{\langle \tau s,\tau\rangle}(C_p)\right)} \quad (27)$$

gdzie p jest liczą węzłów w klastrze C_p, $\lambda(p)$ jest liczbą przetwarzanych strumieni, T jest zbiorem zadań badanej usługi złożonej, a $\varphi^{<\tau s,\tau>}(C)$ jest stratą danych w przedziale pomiaru $<\tau s, \tau>$. Wyniki oceny skalowalności zostały na wykresach przedstawione na rys. 57, gdzie proponowany algorytm MBFD zapewnia stałą i stabilną charakterystykę w całym zakresie rozmiarów badanego klastra.

6.2 Możliwości platformy KASKADA

Platforma KASKADA jest systemem czasu rzeczywistego, którego głównym zadaniem jest przetwarzanie strumieni multimedialnych na klastrze obliczeniowym. Posiada ona architekturę warstwową, składa się z szeregu komponentów programowych oraz dostarcza repozytorium gotowych usług multimedialnych, dzięki którym można w ławy sposób budować nowe aplikacje. Szerszy opis platformy oraz mechanizmów ją wspierających został umieszczony w rozdz. 3 .

Oprócz opisanych i zaimplementowanych procedur zarządzania, a w szczególności algorytmu alokacji zadań na węzły klastra, prace nad platformą zaowocowały szeregiem oryginalnych rozwiązań, które zmieniły cechy typowego klastra obliczeniowego:

- sama idea przetwarzania strumieni multimedialnych, gromadzonych z urządzeń rozmieszczonych w różnych lokacjach, w jednym centrum komputerowym jest nowatorska, i jej realizacja wymagała rozwiązania kilku nowych problemów związanych z zarządzaniem strumieniami, zadaniami oraz zasobami,
- dzięki wprowadzonym mechanizmom platforma umożliwia przetwarzanie czasu rzeczywistego na klastrach komputerowych co jest nowością, gdyż przeważnie wykorzystuje się w tym celu dedykowane urządzenia,
- platforma jest nie tylko środowiskiem umożliwiającym przetwarzanie strumieni multimedialnych, ale także dostarcza mechanizmów wspierających projektowanie oraz implementację algorytmów i usług multimedialnych,
- repozytoria danych i usług umożliwiają łatwą budowę nowych aplikacji, jak również ponowne użycie ich komponentów programowych,
- w trakcie prac zaprojektowano nowy język graficzny MSP-ML [bobkowska11], znacznie

ułatwiający projektowanie usług złożonych, jak i ich weryfikację.

Rozwój platformy był przeprowadzony przez zespół 9-10 osób pracujących w ramach projektu MAYDAY EURO 2012*. Zastosowano iteracyjny i inkrementalny proces wytwarzania oprogramowania. Posiłkowano się elementami zaczerpniętymi z metodyki RUP [kruchten03] oraz *agile* [highsmith10]. Po każdej iteracji następowało wydanie nowej wersji platformy zawierającej nową funkcjonalność. Autor rozprawy jest koordynatorem zespołu budowy platformy, twórcą architektury oraz części projektów szczegółowych i kodu platformy.

Zespół tworzący platformę oraz sama platforma wykorzystuje sprzęt oraz inne zasoby Centrum Informatycznego Trójmiejskiej Akademickiej Sieci Komputerowej na Politechnice Gdańskiej [task]. Podczas prac był wykorzystywany klaster obliczeniowy Galera, na którym wykonano eksperymenty opisane w rozdz. 4 i 5 rozprawy. Docelowo platforma KASKADA jest zainstalowana na nowym klastrze Galera+, posiadającym więcej zasobów i bardziej nowoczesną architekturę.

W projekcie MAYDAY EURO 2012 oprócz platformy KASKADA są rozwijane trzy aplikacje pilotażowe wykorzystujące jej możliwości oraz infrastrukturę. Są one jednocześnie oprogramowaniem testującym jej mechanizmy, zarówno w zakresie funkcjonalności jak i poziomu jakości dostarczanych przez nią usług. Każda z aplikacji jest rozwijana przez osobny podzespół co było dodatkowym wyzwaniem podczas jej tworzenia, gdyż wymagania każdego z zespołów w znacznej części się różniły.

Aplikacja wspomagająca detekcję niebezpiecznych zdarzeń i obiektów jest typowym przykładem wykorzystania platformy do przetwarzania strumieni multimedialnych w czasie rzeczywistym. Jej głównym zadaniem jest akwizycja sygnałów przychodzących z kamer monitoringu obiektów użytku publicznego (stadionów, lotniska czy ulic) i ich analiza w celu wykrycia niebezpiecznych zdarzeń, takich jak zagęszczenie tłumu, zasłabnięcie osoby czy obecność dymu, a następnie powiadomienie odpowiednich służb, np. organizatora imprezy, służb medycznych czy straży pożarnej.

Aplikacja wspomagająca badania endoskopowe wykorzystuje platformę do przetwarzania strumieni multimedialnych *off-line*. Jej głównym zadaniem jest gromadzenie strumieni wideo z badań endoskopowych w archiwum platformy, a następnie ich przetwarzanie w celu znalezienia możliwych stanów chorobowych w zgromadzonym materiale. Tak wykonana analiza umożliwia znaczne skrócenie czasu pracy lekarza stawiającego diagnozę, gdyż może przejść bezpośrednio do podejrzanych fragmentów analizowanego strumienia wideo.

Trzecią aplikacją pilotażową implementowaną na platformie KASKADA, jest System Ochrony Własności Intelektualnej (SOWI). Głównym zadaniem tej aplikacji jest zarządzanie odpowiednimi repozytoriami dokumentów cyfrowych w celu weryfikacji ich autorstwa – wykrywanie plagiatów. W tym przypadku platforma KASKADA jest wykorzystywana jako warstwa pośrednia ułatwiająca implementację algorytmów analizy dokumentów tekstowych z wykorzystaniem architektury SOA. Cechą wyróżniającą tę aplikację jest analiza danych w modelu *master-slave* bez wykorzystania wsparcia strumieni multimedialnych, co pokazuje uniwersalność platformy, która wykracza poza jej główne zastosowanie.

Jak podano KASKADA posiada wielkie możliwości przetwarzania nowego typu aplikacji. Dlatego też problematyka zarządzania obliczeniami jest bardzo istotna z punktu widzenia szybkości przetwarzania jak i jej jakości. Dlatego w rozprawie przyjęto dwa uzupełniające się kryteria

* Politechnika Gdańska, „MAYDAY EURO 2012” Superkomputerowa platforma kontekstowej analizy strumieni danych multimedialnych do identyfikacji wyspecyfikowanych obiektów lub niebezpiecznych zdarzeń. Projekt współfinansowany z Europejskiego Funduszu Rozwoju Regionalnego i Budżetu Państwa w ramach Programu Operacyjnego Innowacyjna Gospodarka.

optymalizacji: minimalną fragmentację dla akceptowalnej straty danych.

W celu poprawienia reprezentatywności badań eksperymentalnych przeprowadzonych na platformie KASKADA zdecydowano się na wykorzystanie szerokiej biblioteki algorytmów służących do przeprowadzanych eksperymentów. Cześć z nich jest dostępna jako Open Source (np. algorytm detekcji twarzy), inne były zaprojektowane przez zespoły aplikacji pilotażowych projektu MAYDAY EURO 2012 (np. maska tła), a tylko niektóre z nich były zaimplementowane przez zespół budowy platformy KASKADA, w pracach którego uczestniczył autor rozprawy.

Ponadto przeprowadzona analiza teoretyczna jak i badania eksperymentalne umożliwiły potwierdzenie szeregu przypuszczeń, które choć zgodne z intuicją nie były do końca sprawdzone:

- przewaga algorytmu alokacji MBFD nad algorytmami LB (równoważenia obciążenia) i BFD (bez wykorzystania funkcji korekty),
- brak zależności między obciążeniem węzła a długością przetwarzanego strumienia, należy jednak zaznaczyć, że wykonywanie dłuższych zadań może wpłynąć negatywnie na dostępność platformy przy założeniu napływania nowych zadań ze stała częstotliwością,
- możliwość wykorzystania dekompozycji danych w celu minimalizacji strat danych wyjściowych, kiedy to zadania alokowane na osobnych węzłach analizują tylko fragmenty strumienia przez co, dla każdego z zadań, elementy danych (np. klatki obrazu) „płyną" wolniej,
- możliwość wykorzystania dekompozycji funkcjonalnej w celu zmniejszenia ziarnistości obliczeń na poziomie usług prostych i zadań, przy założeniu wykorzystania potoku przetwarzającego strumienie danych.

W celu ograniczenia zakresu badań z punktu widzenia rozprawy doktorskiej wprowadzono następujące założenia upraszczające:

- wielkość pamięci alokowanej przez każde zadanie realizujące algorytm nie może być zwiększana w czasie jego wykonywania gdyż w takim przypadku szacowane obciążenie węzła nie jest właściwe i powinno być monitorowane na bieżąco poprzez sprzętowe mechanizmy procesora,
- szybkości działania algorytmu alokacji wykonywanego na osobnym serwerze zarządzania przetwarzaniem rozproszonym, który nie powinien dodatkowo opóźniać uruchamiania zadań analizy strumieni na klastrze, stąd algorytmy alokacji muszą mieć proste i efektywne, implementacje na szybkim serwerze.

Powstaje pytanie jak te założenia upraszczające wpływają na ocenę platformy KASKADA, gdzie są rzeczywiste bariery, co należałoby usprawnić w jej architekturze, metodach zarządzania czy w szczególności algorytmach alokacji? Dotychczasowe doświadczenia to ogromna wiedza, która powinna pozwolić na usprawnienie i ulepszenie architektury platformy KASKADA. Czy platforma jako system komputerowy nie jest zbyt skomplikowana? Czy nie ma zbyt wiele warstw? Czy inne dobrze znane rozwiązania, jak np. MPI nie powinny się stać jej stałym elementem? Na te pytania powinny dać odpowiedź dalsze prace badawcze podjęte przez autora oraz cały zespół budowy platformy KASKADA.

6.3 Kierunki dalszych prac badawczych

Tematyka przetwarzania strumieni multimedialnych na klastrach obliczeniowych jest stosunkowo nowa. Niewielka liczba publikacji z nią związanych świadczy o dużych możliwościach dalszych badań. Szczególnie istotne wydają się kierunki rozwoju przetwarzania z wykorzystaniem nowych możliwości sprzętowych: heterogeniczne klastry obliczeniowe, jak również systemowych:

np. przetwarzanie w chmurze, przy zachowaniu odpowiedniej jakości wykonywania usług (QoS). Innym rozważanym kierunkiem jest rozwój i ocena środowiska tworzenia aplikacji multimedialnych na tego typu platformach, w tym jej specjalistycznych narzędzi takich jak edytor usług złożonych w języku MSP-ML.

Pomimo przeprowadzonej analizy i badań procedur zarządzania w systemowych przetwarzających strumienie, autor zdaje sobie sprawę z możliwości dalszego rozwoju prac nad tymi zagadnieniami. Jak to zostało zasygnalizowane w rozdz. 3.8 rozważane jest rozszerzenie algorytmu alokacji o elementy szeregowania zdań ze znanym a priori czasem pracy. Innym polem rozwoju jest rozszerzenie kryteriów optymalizacji alokacji o nowe elementy, takie jak obciążenie komunikacyjne, czy opóźnienie w przetwarzaniu potokowym zadań strumieniowych.

Rozszerzenie działania platformy KASKADA o wykorzystanie nowych rozwiązań sprzętowych jest naturalnym kierunkiem jej rozwoju. Obecnie jest ona dostosowana do wykorzystania homogenicznego klastra obliczeniowego, lecz w przyszłości planowane jest poszerzenie jej możliwości o wykorzystanie wielordzeniowych kart obliczeniowych GPGPU (ang. General Purpose Graphics Porcessing Unit), dla których wstępne badania pokazują duże możliwości wykorzystania w dziedzinie przetwarzania strumieni multimedialnych, a w szczególności ich równoległej analizy. Z drugiej strony rozważane jest zastosowanie platformy do wykorzystania jako warstwy pośredniej dla systemów przetwarzania w chmurze, dzięki czemu ich infrastruktura będzie mogła być łatwo wykorzystywana przez już istniejące usługi z repozytorium platformy, jak i nowe, tworzone aplikacje.

Ekspansja platformy KASKADA na nowe rozwiązania sprzętowe powinna gwarantować stały poziom jakościowy wykonywanych zadań obliczeniowych. Dlatego planuje się rozwój podsystemu monitorowania węzłów obliczeniowych, tak aby umożliwić szybką detekcję niesprzyjających warunków wykonywania analizy strumieni multimedialnych. Prowadzi to do konieczności zapewnienia mechanizmów migracji zadań z nieadekwatnych węzłów, co jest szczególnie trudne jeśli chce się zapewnić ciągłość przetwarzania z zachowaniem wymagań czasu rzeczywistego.

Wykonanie eksperymentów badawczych zaowocowało nie tylko wynikami przedstawionymi w niniejszej rozprawie, ale także spowodowało rozwój biblioteki automatycznych procedur i skryptów umożliwiających ich uzyskanie. Planuje się dalszy rozwój tych procedur w celu uzyskania zestawu standardowych benchmarków umożliwiających ocenę zadanego klastra komputerowego do wykorzystania w celu masowego przetwarzania strumieni multimedialnych.

Platforma KASKADA oprócz typowych funkcji wykonywania aplikacji, usług i zadań przetwarzania strumieni multimedialnych jest środowiskiem wspierającym ich wytwarzanie. Biblioteki ramki KASKADA umożliwiają łatwą implementację algorytmów analizy. Konsola użytkownika wraz z edytorem usług umożliwia ich ekspozycję dla aplikacji użytkowych z wykorzystaniem architektury SOA; ponadto repozytorium gotowych usług jeszcze bardziej skraca czas ich rozwoju. Planowana jest optymalizacji tych narzędzi oraz szerokie badania wykorzystujące eksperymenty kontrolowane w celu weryfikacji i porównania poziomu jakości wytwarzania oprogramowania multimedialnego na innych platformach przetwarzania równoległego.

Autor rozprawy ma także świadomość, że bardzo często jako koszt wielu udogodnień (np. łatwiejsza implementacja algorytmów, szerokie monitorowanie zadań i węzłów, uniwersalny interfejs do przesyłania danych) powstaje strata wydajności. Konkurencyjne rozwiązania, choć bardziej ogólne, wymagają jednak większych nakładów przy implementacji specyficznych algorytmów. Tak więc bardzo ważnym elementem przyszłych badań będzie porównanie nie tylko takich charakterystyk jak obciążenie i strata danych, ale także użyteczności i elastyczności w odniesieniu do takich bibliotek i platform jak MPI [mpi] czy VSMP [vsmp]. Wyniki powyższych badań powinny wpłynąć na dalszy rozwój platformy KASKADA poprzez wprowadzenie odpowiednich ulepszeń do jej budowy czy nawet modyfikacji architektury.

Edytor usług złożonych wykorzystujący język MSP-ML [bokowska11] jest kolejnym polem rozwoju platformy KASKADA. Sam język umożliwia kompozycję nowej funkcjonalności wykorzystując istniejące usługi proste oraz reprezentujące je zadania obliczeniowe. Specyficzne cechy tego języka ułatwiają planowanie odpowiedniego przepływu strumieni i komunikatów między usługami prostymi jak również prawidłowe zaprojektowanie wejść i wyjść samej usługi złożonej. Dodatkowym wyzwaniem jest wprowadzenie mechanizmów automatycznego doboru najbardziej odpowiednich usług przetwarzania strumieni multimedialnych, tak aby zachować odpowiedni poziom jakościowy przeprowadzanej przez nie analizy.

Referencje

122 pozycje

[atint10] AT Internet, *Apple gains more market share in Europe than the giant Microsoft*, http://www.atinternet.com/en/documents/apple-gains-more-market-share-in-europe-than-the-giant-microsoft/, 2010 (dostęp 27.04.2012)

[bader01] Bader D.A., Pennington R. *Cluster Computing: Applications,* The International Journal of High Performance Computing, 15(2): 181-185, May 2001.

[bai08] Bai Y., Thakkar H., Wang H., and Zaniolo C. Time-Stamp Management and Query Execution in Data Stream Management Systems. *IEEE Internet Computing* 12, 6 (November 2008), 13-21

[bailey94] Bailey D. i in. *The NAS Parallel Benchmarks,* NASA, http://www.nas.nasa.gov/assets/pdf/techreports/1994/nas-94-001.pdf, 1994 (dostęp 05.05.2012)

[benoit10] Benoit A., Marchal L., Pineau J. F., Robert, Y., Vivien, F. *Scheduling Concurrent Bag-of-Tasks Applications on Heterogeneous Platforms,* IEEE Transactions on Computers, vol.59, no. 2, pp. 202-217, Feb. 2010

[bertis01] Berstis V. *Fundamentals of Grid Computing,* IBM, http://www.redbooks.ibm.com/abstracts/redp3613.html, 2001 (dostęp 05.05.2012)

[bobkowska11] Bobkowska A., Nykiel M. , Proficz J., *Evaluation of Multimedia Stream Processing Modeling Language from the Perspective of Cognitive Dimensions,* Proceedings of PPIG 2011, York, 2011, 7

[booch93] G. Booch *Object-Oriented Analysis and Design with Applications (2nd Edition),* Addison-Wesley Professional, 1993

[boost] *Boost C++ Libriaries* – strona domowa, http://www.boost.org/, (dostęp 05.05.2012)

[bovik09] Bovik A. *The Essential Guide to Video Processing,* Academic Press, Elsevier, 2009

[bovet00] Bovet D. P., Cesati M. *Understanding the Linux Kernel,* O'Reilly 2000

[bradski08] Bradski G., Kaehler A. *Learning OpenCV: Computer Vision with the OpenCV Library,* O'Reilly Media, 2008

[branden08] Brandenburg B. B., Calandrino J. M., Anderson J. H. *On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study,* Real-Time Systems Symposium, pp. 157-169, 2008

[breuer03] Breuer D. i in. *Scientific Computing with UNICORE,* NIC Series, vol. 20, pp. 429-440, 2003

[brudlo95] Brudło P. *Fully Distributed Task Processing in Local Area Networks,* PhD Thesis, Politechnika Gdańska, 1995

[butazzo05] Buttazzo G., Lipari G., Abeni L., Caccamo M. *Soft Real-Time Systems Predictability vs. Efficiency,* Series in Computer Science, Springer 2005

[buyya11] Buyya R., Broberg J., Goscinski A. *Cloud Computing, Principles and Paradigms,* Willey 2011

[cattel00] Cattell R. i in. *The Object Data Management Standard: ODMG 3.0*, Morgan Kaufmann Publishers, Inc., 2000

[chappell04] Chappell D. *Enterprise Service Bus*, O'Reilly 2004

[chen03] Deji Chen, Mok, A.K.; Tei-Wei Kuo *Utilization bound revisited*, IEEE Transactions on Computers, vol.52(3), str. 351- 361, 2003

[chin05] Chin, J., Harvey, M.J., Jha, S., Coveney, P.V. *Scientific grid computing: the first generation,* IEEE Computing in Science & Engineering, vol.7, no.5, pp. 24- 32, Sept.-Oct. 2005

[corba] CORBA, *Common Object Request Broker Architecture* – strona domowa http://www.corba.org/, (dostęp 05.05.2012)

[cormen90] Cormen T. H., Leiserson C. E., Rivest R. L. *Introduction to Algorithms,* MIT Press, 1990

[cucinotta10] Cucinotta T., Palopoli L. *QoS Control for Pipelines of Tasks Using Multiple Resources,* IEEE Transactions on Computers, vol. 59, no. 3, pp. 416-430, March 2010

[culler99] Culler D. E., Singh J. P. *Parallel Computer Architecture, A Hardware/Software Approach,* Morgan Kaufmann Publishers, Inc. 1999

[czarnul02] Czarnul P. *Dynamic Load Balancing with Data Partitioning for Efficient Parallel Computing,* PhD Thesis, Politechnika Gdańska, 2002

[czech10] Czech Z. *Wprowadzenie do obliczeń równoległych,* PWN 2010

[duboc08] Duboc L., Letier E., Rosenblum D. S., Wicks T. *A Case Study in Eliciting Scalability,* 16th IEEE International Requirements Engineering Conference, pp. 247-252, 2008

[duboc10] Duboc L., Rosenblum D. S., Letier E. *Death, Taxes, & Scalability,* IEEE Software, vol. 27, no. 4, pp. 20-21, July-Aug. 2010

[duboc12] Duboc L., Leiter E., Rosenblum D. *Systematic Elaboration of Scalability Requirements through Goal-Obstacle Analysis,* IEEE Transactions on Software Engineering, w druku

[eide03] Eide V. S. W., Eliassen F., Granmo O. C., Lysne O. *Supporting timeliness and accuracy in distributed real-time content-based video analysis,* Proceedings of the eleventh ACM international conference on Multimedia (MULTIMEDIA '03), pp. 21-32, 2003

[ejb] EJB, *Enterprise Java Beans Technology* – strona domowa http://www.oracle.com/technetwork/java/index-jsp-140203.html, (dostęp 05.05.2012)

[elliot04] Elliot J. *Hibernate: Developer's Notebook,* O'Rielly, 2004

[el-rewini04] El-Rewini H., Abd-El-Barr M. *Advanced Computer Architecture and Parallel Processing,* Wiley-Interscience, 2004

[eql] EJBQL, *EJB Query Language Tutorial* http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html, (dostęp 05.05.2012)

[erl09] Erl T. *SOA Design Patterns,* Prenice Hall, 2009

[fielding99] Fielding R. i in. *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, World Wide

Web Consortium (W3C)
http://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999 (dostęp 05.05.2012)

[fhourstones] *The Fhourstones Benchmark* – strona domowa
http://homepages.cwi.nl/~tromp/c4/fhour.html, (dostęp 05.05.2012)

[flynn72] Flynn M. *Some Computer Organizations and Their Effectiveness,* IEEE Transactions on Computing, C-21, September 1972

[foster96] Foster I., Kesselman C. *Globus: A metacomputing infrastructure toolkit,* International Journal of Supercomputer Applications, vol. 11, pp. 115-128, 1996

[friesen86] Friesen D. K., Langston M. A. *Variable Sized Bin Packing,* SIAM J. Comput. 15, pp. 222-230, 1986

[garey79] Garey M. R., Johnson D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences), W. H. Freeman, 1979

[gorlatch09] Gorlatch S., Glinka F., Ploss A. *Towards a Scalable Real-Time Cyberinfrastructure for Online Computer Games,* Parallel and Distributed Systems 15, pp. 722-727, 2009

[gryffin98] Gryffin R. W. *Podstawy zarządzania organizacjami*, PWM 1998

[gstreamer] *GStreamer, Open Source Multimedia Framework* – strona domowa
http://gstreamer.freedesktop.org/, (dostęp 05.05.2012)

[h264] *H.264 : Advanced video coding for generic audiovisual services,* International Telecommunication Union (ITU), Recommendation H.264
http://www.itu.int/rec/T-REC-H.264, (dostęp 05.05.2012)

[haddad11] Haddad S., Kordon F., Pautet L., Petrucci L. *Models and Analysis for Distributed Systems,* ISTE & Willey, 2011

[haerder83] Haerder T., Reuter A. *Principles of transaction-oriented database recovery,* ACM Comput. Surv. 15, 4 (December 1983), pp. 287-317

[haouari10] Haouari M., Serairi M. *Heuristics for the variable sized bin-packing problem*, Computers & Operational Research 36, pp. 2877-2884, 2009

[hiber] *Hibernate* – strona domowa
http://www.hibernate.org/, (dostęp 05.05.2012)

[highsmith10] Highsmith J. *Agile Project Management (2nd Edition),* Addison-Wesley, 2010

[hyeran10] Hyeran J., Lee W. H., Chung S. W. *Load Unbalancing Strategy for Multicore Embedded Processors,* IEEE Transactions on Computers, vol. 59, no. 10, pp. 1434-1440, Oct. 2010

[ibarra75] Ibarra O. H., Kim C. E. *Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems,* J. ACM 22, pp. 463-468, 1975

[inifniband] *InifniBand Trade Association* – strona domowa
http://www.infinibandta.org/, (dostęp 05.05.2012)

[iometer] *Iometer project* – strona domowa
http://www.iometer.org/, (dostęp 05.05.2012)

[jms] JMS, *Java Message Service Specification,* Oracle Inc.
http://www.oracle.com/technetwork/java/index-jsp-142945.html, (dostęp

05.05.2012)

[jogalekar00] Jogalekar P., Woodside M. *Evaluating the scalability of distributed systems,* IEEE Transactions on Parallel and Distributed Systems, vol.11, no.6, pp.589-603, Jun 2000

[keahey09] Keahey K., Tsugawa M., Matsunaga A., Fortes J. *Sky Computing,* IEEE Internet Computing, vol. 13, no. 5, September/October 2009

[knorr08] Knorr E., Gruman G. *What cloud computing really means,* InfoWorld http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031, 2008 (dostęp 05.05.2012)

[krawczyk09] Krawczyk H., Proficz J. *Podstawowe metody integracji aplikacji trójwarstwowych,* Od modelu do wdrożenia - kierunki badań i zastosowań inżynierii oprogramowania, Wydawnictwo Komunikacji i Łączności, 2009, str. 103-115

[krawczyk10] Krawczyk H., Proficz J., Bańczyk K. *Parallel processing of multimedia streams,* Transactions on Computer Applications in Electrical Engineering, Poznan, April 19-21, 2010, str. 267-268

[krawczyk10a] Krawczyk H., Proficz J. *KASKADA – multimedia processing platform architecture* Proceedings of the International Conference on Signal Processing and Multimedia Applications, SIGMAP 2010

[krawczyk10b] Krawczyk H., Proficz J. *The task graph assignment for KASKADA platform* Proceedings of the International Conference on Software and Data Technologies, ICSOFT 2010

[krawczyk10c] Krawczyk H., Proficz J., Bańczyk K. *Parallel processing of multimedia streams,* Computer Applications in Electrical Engineering, vol. 8 (2010), pp. 9-25

[krawczyk11] Krawczyk H., Konpa R., Proficz J. *Basic management strategies on KASKADA platform,* EUROCON 2011

[krawczyk12] Krawczyk H., Proficz J. *Real-time multimedia stream data processing in a supercomputer environment,* rozdz. w monografii *Interactive Multimedia*, InTech, 2012

[kruchten03] Kruchten P. *The Rational Unified Process: An Introduction (3rd Edition),* Addison-Wesley Professional, 2003

[kuck96] Kuck D. J. *High performance computing, challenges for future systems*, Oxford University Press, 1996

[leontyev10] Leontyev H., Anderson J. *Generalized tardiness bounds for global multiprocessor scheduling,* Real-Time Systems, vol 44(1), str. 26-71, 2011

[lindberg12] Lindberg P., Leingang J., Lysaker D., Khan S., Li J. *Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems,* The Journal of Supercomputing, vol. 59(1), str. 323-360, 2012

[linpack] *Linpack Benchmark* – strona domowa http://www.top500.org/project/linpack, (dostęp 05.05.2012)

[liu73] Liu C. L., Layland J. W. *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment,* Journal of the Association for Computing Machinery, vol. 20(1):46-61, 1973

[lusk97] Lusk E. i in. *MPI-2: Extension to Message Passing Interface*, 1997 http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-2.0/mpi2-report.htm, 1997 (dostęp 05.05.2012)

[lustre] *LUSTRE file system* – strona domowa http://www.lustre.org/, (dostęp 05.05.2012)

[ma10] Ma Wei-feng, Wang Jia-hai *Analysis of the Linux 2.6 kernel scheduler,* International Conference on Computer Design and Applications (ICCDA), vol.1, str.V1-71-V1-74, 2010

[matroska] *Matroska Media Container* – strona domowa http://www.matroska.org/, (dostęp 05.05.2012)

[mayers05] Meyers S. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs,* Addison-Wesley Professional, 2005

[msmq] MSMQ, *Microsoft Message Queuing,* Microsoft, Inc. http://msdn.microsoft.com/en-us/library/ms711472%28v=vs.85%29.aspx, (dostęp 05.05.2012)

[oh95] Oh Y., Son S. H. *Allocating fixed-priority periodic tasks on multiprocessor systems,* Real-Time Systems, vol 9(3), str. 207-239, 1995

[opencv] OpenCV, Open Source Vision Library – strona domowa http://socghop.appspot.com/gsoc/org/google/gsoc2011/opencv, (dostęp 05.05.2012)

[pezoa10] Pezoa J. E., Dhakal S., Hayat M. M. *Maximizing Service Reliability in Distributed Computing Systems with Random Node Failures: Theory and Implementation,* IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 10, pp. 1531-1544, Oct. 2010

[posix] *POSIX.1-2008 Standard* http://pubs.opengroup.org/onlinepubs/9699919799/, 2008 (dostęp 05.05.2012)

[postgres] *PostgreSQL* – strona domowa http://www.postgresql.org/, (dostęp 05.05.2012)

[povray] *Official PoVRay Benchmark* – strona domowa http://new.haveland.com/povbench/index.php, (dostęp 05.05.2012)

[proficz10] Proficz J. *Kaskada - scenariusze analizy strumieni multimedialnych,* Przedsięwzięcia i usługi informacyjne, Praca zbiorowa Katedry Architektury Systemów Komputerowych KASKBOOK, Wydział ETI Politechnika Gdańskiej, Gdańsk, 2010, str. 181-188

[pts] *The Phoronix Test Suite* – strona domowa http://www.phoronix-test-suite.com/, (dostęp 05.05.2012)

[pvm] PVM, *Parallel Virtual Machine* – strona domowa http://www.csm.ornl.gov/pvm/pvm_home.html, (dostęp 05.05.2012)

[qi11] Qi X., Zhu D., Aydin H. *Cluster scheduling for real-time systems: utilization bounds and run-time overhead,* Real-Time Systems, vol. 47(3), str. 253-284, 2011

[qsucc] *Q-Success Web-based Services, Usage of operating systems for websites* http://w3techs.com/technologies/overview/operating_system/all, 2012 (dostęp 27.04.2012)

[refsnes10] *Refsnes Data, W3CSchool Browser statistics*

http://www.w3schools.com/browsers/browsers_stats.asp, (dostęp 05.05.2012)

[rmi] RMI, *Remote Method Invocation* – strona domowa http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, (dostęp 05.05.2012)

[rpc] RPC, *Remote Procedure Call Protocol specification*, RFC 1050, Sun Microsystems, Inc. http://www.faqs.org/rfcs/rfc1050.html, 1988 (dostęp 05.05.2012)

[rogmans08] Rogmans S., Jiangbo Lu, Lafruit G. *A Scalable End-to-End Optimized Real-Time Image-Based Rendering Framework on Graphics Hardware,* 3DTV Conference: The True Vision – Capture, pp. 129-132, 2008

[sandhu94] Sandhu R. S., Samarati P. *Access control: principle and practice,* IEEE Communications Magazine, vol. 32, no. 9, pp. 40-48, 1994

[schulzrinne98] Schulzrinne H., Rao A., Lanphier R. *Real Time Streaming Protocol (RTSP)* http://www.ietf.org/rfc/rfc2326.txt, 1998 (dostęp 05.05.2012)

[smith97] Smith S. W. *The Scientist & Engineer's Guide to Digital Signal Processing,* California Technical Pub., 1997

[soa] SOA, *OASIS Service Oriented Architecture Reference Model RC,* OASIS http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm, 2005 (dostęp 05.05.2012)

[soap] SOAP, *Simple Object Access Protocol standard,* World Wide Web Consortium (W3C) http://www.w3.org/TR/soap/, 2007 (dostęp 05.05.2012)

[sql92] SQL, *Structured Query Language standard* http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt, 1992 (dostęp 05.05.2012)

[stroustrup00] B. Stroustrup *The C++ Programming Language: Special Edition,* Addison-Wesley 2000

[taoyu09] Tao Yu, Baoyao Zhou, Qinghu Li, Rui Liu, Weihong Wang, Cheng Chang *The design of distributed real-time video analytic system,* Proceeding of the first international workshop on Cloud data management (CloudDB '09), pp. 49-52, 2009

[tanenbaum97] Tanenbaum A. S. *Rozproszone systemy operacyjne,* Wydawnictwo Naukowe PWN, 1997

[task] Centrum Informatyczne Trójmiejskiej Akademickiej Sieci Komputerowej – TASK, Politechnika Gdańska – strona domowa http://www.task.gda.pl/, (dostęp 05.05.2012)

[top] *TOP command manual,* http://www.linuxmanpages.com/man1/top.1.php, (dostęp 05.05.2012)

[top500] *TOP500 Supercomputing Sites* http://www.top500.org/, (dostęp 05.05.2012)

[troitzsch09] Troitzsch K. G. *Multi-Agent Systems and Simulation: A Suvey from an Application Perspective*, z Multi-Agent Systems Simulation and Applications, str. 53-76, CRC Press, 2009

[uddi] UDDI, *Universal Description Discovery and Integration* – strona domowa

http://uddi.xml.org/

[uima] UIMA, *Unstructured Information Management Applications* – strona domowa http://uima.apache.org/, (dostęp 05.05.2012)

[vidyarthi10] Vidyarthi D. P., Sarker B. K., Tripathi A. K., Yang L. T. *Scheduling in Distributed Computing Systems: Analysis, Design and Models,* Springer Science+Business Media, 2010

[viola01] Viola P., Jones M. J. *Rapid Object Detection using a Boosted Cascade of Simple Features,* IEEE CVPR, 2001

[vsmp] *Versatile SMP (vSMP) Architecture* http://www.scalemp.com/architecture, (dostęp 05.05.2012)

[wand96] Wand Y., Wang R. Y. *Anchoring Data Quality Dimentions in Ontological Foundations,* Communications of ACM, 1996

[wang96] Wang R. Y., Strong D. *Beyound accuracy: What data quality means to data consumers,* Journal of Management Information Systems, 1996

[watts98] Watts, J., Taylor, S., *A practical approach to dynamic load balancing,* IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 3, str. 235-248, 1998

[weiss02] Weiss A. R. *Dhrystone Benchmark, History, Analysis, "Scores" and Recommendations* – White paper http://www.johnloomis.org/NiosII/dhrystone/ECLDhrystoneWhitePaper.pdf, 2002 (dostęp 05.05.2012)

[welke10] Welke R., Hirschheim R., Schwarz A., *Service-Oriented Architecture Maturity,* IEEE Computer, vol. 44, no. 2, pp. 61-67, Feb. 2011

[whetstone] *The Whetstone Benchmark* – strona domowa http://www.cse.scitech.ac.uk/disco/Benchmarks/whetstone.shtml, (dostęp 05.05.2012)

[williams09] Williams S., Waterman A., Patterson D. *Roofline: an insightful visual performance model for multicore architectures,* ACM Communications vol. 52, no. 4, pp. 65-76, Apr. 2009

[wooldridge09] Wooldridge M. *An Introduction to MultiAgent Systems,* Willey, 2009

[wsdl] Web Services Description Language (WSDL) 1.1, W3C Note, World Wide Web Consortium (W3C) http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001 (dostęp 05.05.2012)

[xiao10] Xiao Qin, Hong Jiang, Manzanares A., Xiaojun Ruan, Shu Yin *Communication-Aware Load Balancing for Parallel Applications on Clusters,* IEEE Transactions on Computers, vol. 59, no. 1, pp. 42-52, Jan. 2010

[yutang10] Yu Tang Y., Changqin Bu, Zhenghua Wu, Sheng Su *A scalable approach to distributed scheduling scheme on cluster architecture of real-time server,* Computer Engineering and Technology 2, pp. V3-123-V3-127, 2010