Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki
Katedra Informatyki Stosowanej

Rozprawa Doktorska

Strumieniowe hurtownie danych zorientowane na przetwarzanie wielkich zbiorów danych kontekstowych

mgr inż. Krzysztof Pasterak

Promotor:
prof. dr hab. inż. Marcin Gorawski

Gliwice, lipiec 2022

Niniejszą rozprawę dedykuję Alicji, której wsparcie było i jest nieocenione

Pragnę podziękować panu profesorowi Marcinowi Gorawskiemu,
promotorowi tej rozprawy doktorskiej

Spis treści

Rozdział 1

Wprowadzenie

1.1 Geneza rozprawy

Koncepcja hurtowni danych (ang. *data warehouse*) narodziła się w drugiej połowie XX wieku jako odpowiedź na rosnące zapotrzebowanie w zakresie systemów wspomagania decyzji w przedsiębiorstwach. Idea hurtowni danych polegała na zastosowaniu odrębnej bazy danych o wyspecjalizowanym schemacie, przeznaczonej wyłącznie do celów analityczne. Głównym założeniem hurtowni danych było wielowymiarowe spojrzenie na procesy zachodzące w przedsiębiorstwach na przestrzeni wielu lat. Doprowadziło to do wyodrębnienia się specyficznego nazewnictwa i zarazem podziału danych na fakty (analizowane zjawiska) i wymiary (okoliczności powstawania tych pierwszych). Schemat hurtowni danych również nawiązywał do rzeczonego podziału, odzwierciedlając niejako umiejscowienie poszczególnych faktów w przestrzeni wielowymiarowej.

Rozwój modeli hurtowni danych ukształtował z czasem dwa nurty, wywodzące się od pionierów: Ralpha Kimballa i Billa Inmona. Pierwszy z tych nurtów zakładał wielowymiarowy model danych na poziomie całej hurtowni, która projektowana jest poprzez integrację wielu minihurtowni tematycznych (ang. *data marts*) [92]. Drugie podejście wykorzystywało centralny model hurtowni danych (niekoniecznie wielowymiarowy), z którego następnie dane przenoszone były do minihurtowni tematycznych (o modelu wielowymiarowym) [83].

Wykorzystanie metod wywodzących się z hurtowni danych do analizy zjawisk niepowiązanych z działalnością przedsiębiorstwa, znajduje swoje odzwierciedlenie w literaturze [16]. Rozwój Internetu Rzeczy (IoT, od ang. *Internet of Things*) [128, 73, 74], systemów informacji geograficznej (GIS, od ang. *Geographical Information Systems*) [19] oraz paradygmatu strumieniowego przetwarzania danych [127, 130] stał się okazją do rozwoju systemów bazujących na klasycznych hurtowniach danych. Rozwiązania te również są wykorzystywane w procesach podejmowania decyzji, lecz ich budowa i zało-

żenia nawiązują do charakteru i natury przetwarzanych danych, co zazwyczaj wiąże się z ich przetwarzaniem w sposób nieprzerwany [90].

Dominującym typem operacji w tego rodzaju systemach, niezależnie od ich przeznaczenia i specjalizacji, są wielowymiarowe zapytania agregujące. Odpowiadają one na pytania o średnie lub ekstremalne wartości zarejestrowanych pomiarów w określonych obszarach geograficznych i wybranych przedziałach czasu. Dodatkowo uwzględniają przy tym zazwyczaj hierarchiczną naturę wymiarów, a więc operują na wielu poziomach szczegółowości jednocześnie [13, 103, 100]. W tym zakresie pokrywa się to z operacjami wykonywanymi w klasycznych hurtowniach danych, co pozwala na wyjście tej koncepcji poza ramy analizy strategii biznesowej.

Rozwój i dalsza specjalizacja opisanych wyżej rozwiązań doprowadziła do powstania osobnej klasy systemów, którą można nazwać mianem zaawansowanych hurtowni danych (ADW, od ang. *Advanced Data Warehouses*) [29]. Składają się na nią systemy, które wyewoluowały z modelu klasycznych hurtowni danych. Wyróżnić można kilka przedstawicieli tej klasy: przestrzenno-temporalne hurtownie danych [31, 42], chronione hurtownie danych [65, 34], gridowe hurtownie danych [43, 33], rozproszone hurtownie danych [63, 55], trajektoryjne hurtownie danych [46, 45], bezopóźnieniowe hurtownie danych [48, 49] oraz strumieniowe hurtownie danych [35, 59, 62].

W literaturze można spotkać również inne rodzaje systemów, które spełniają założenia zaawansowanych hurtowni danych [17]: hurtownie pamięciowe (ang. *in-memory*), rozproszone hurtownie danych korzystające z paradygmatu *map-reduce*, hurtownie danych działające w czasie zbliżonym do rzeczywistego (ang. *near real-time*), hurtownie danych działające w chmurach obliczeniowych (ang. *cloud data warehouses*). Co więcej, oprócz tworzenia nowych modeli zaawansowanych hurtowni danych, wiele prac badawczych koncentruje się na rozwoju metod, które usprawniają działanie istniejących systemów, czyniąc je *de facto* systemami z pogranicza zaawansowanych hurtowni danych. Wśród wspomnianych technik należy wymienić przede wszystkim metody sztucznej inteligencji i eksploracji danych oraz metody szyfrowania i anonimizacji danych [125].

Jednym z kierunków rozwoju hurtowni danych, a zarazem cechą charakterystyczną klasy zaawansowanych hurtowni danych, jest umiejętność pracy ze strumieniowymi źródłami danych. Rzeczona zdolność rzutuje na konieczność natychmiastowego przetwarzania nowych danych i reagowania na zmieniającą się dynamicznie rzeczywistość. Można zatem stwierdzić, iż rozwój koncepcji strumieniowych hurtowni danych jest jednym z ważniejszych obszarów badawczych w dziedzinie zaawansowanych hurtowni danych. Ich szczególną cechą jest umiejętność łączenia analizy danych bieżących z analizą danych historycznych, co stwarza możliwości badania różnic pomiędzy teraźniejszością a przeszłością oraz prognozowania przyszłości. Przykładem takiego zastosowania jest wykrywanie anomalii w strumieniach danych bieżących wsparte analizą danych historycznych, przeprowadzaną w celu uwia-

rygodnienia wyników tego wykrywania. Dzięki wglądowi w dane historyczne, możliwe staje się poznanie charakterystyki zmienności w czasie badanego zjawiska. Niejednokrotnie wpływa to na decyzję o klasyfikacji wykrytego zdarzenia jako anomalnego bądź nie. W przytoczonym przykładzie, dane historyczne stanowią *kontekst* dla danych bieżących i pozwalają na jednoznaczną interpretację wyników ich analizy.

1.2 Tezy i zakres rozprawy

Tematyka niniejszej rozprawy obejmuje ogół zagadnień powiązanych ze strumieniowymi hurtowniami danych, a w szczególności – tymi zorientowanymi na przetwarzanie wielkich zbiorów danych kontekstowych. Celem rozprawy jest obszerne omówienie zagadnień teoretycznych, przedstawienie propozycji nowych metod i modeli, przeprowadzenie stosownych badań eksperymentalnych oraz analiza ich wyników prowadząca do sformułowania wniosków.

Kolejność prezentowanych treści w rozprawie odzwierciedla przyjęty tok rozumowania. Prowadzi on od dostrzeżenia i identyfikacji problemu, poprzez sformułowanie podstaw teoretycznych wraz z podaniem propozycji algorytmów i struktur danych, aż do przeprowadzenia eksperymentów weryfikacyjnych i porównawczych, umożliwiających opracowanie wniosków końcowych. Poszczególne treści stanowią niejednokrotnie podsumowanie oraz rozwinięcie zagadnień opisanych w pracach własnych autora niniejszej rozprawy[1].

Pierwszym zagadnieniem omawianym szerzej w rozprawie, będącym zarazem punktem wyjścia do rozpoczęcia dalszych prac badawczych, jest system dystrybucji i składowania paliw płynnych, przy którym prowadzono badania nad problemem detekcji anomalii krytycznych, a w szczególności wycieków paliwa z podziemnych zbiorników. Skutkiem tych badań było opracowanie metody wykrywania wycieków wykorzystującej detekcję i interpretację trendów w szeregach czasowych. W tym zakresie tematycznym powstały trzy artykuły naukowe [38, 28, 39]$^{\mathrm{W}}$ oraz trzy patenty [70, 40, 71]$^{\mathrm{W}}$.

Metoda ta, pomimo swojej dowiedzionej skuteczności, nie dawała w pełni satysfakcjonujących wyników. Jak później stwierdzono, problem nie leżał w samej metodzie detekcji, lecz w braku całościowego podejścia do problemu i analizie tylko wybranego wycinka rzeczywistości – danych bezpośrednio związanych z wyciekami paliwa [39]$^{\mathrm{W}}$. W konkluzjach stwierdzono, iż bez wzięcia pod uwagę współistniejących czynników, nie jest możliwe uzyskanie w pełni wiarygodnych wyników, ze względu na wzajemną zależność badanych zjawisk. W tym zakresie stawiana jest pierwsza teza niniejszej rozprawy:

[1] Dla odróżnienia od pozostałych publikacji, cytowania prac własnych oznaczane będą dużą literą W w indeksie górnym numeru cytowanej pozycji, np. [0]$^{\mathrm{W}}$.

Teza 1 *Uzyskanie w pełni jednoznacznych wyników analizy danych ukierunkowanej na wykrywanie zdarzeń anomalnych jest możliwe dopiero po uwzględnieniu kontekstu występowania poszczególnych anomalii, na który składają się dane współistniejące w czasie i przestrzeni oraz powiązane semantycznie z analizowanym zjawiskiem.*

Wnioski z omówionych badań posłużyły za podstawę do sformułowania teorii danych kontekstowych, wraz z podaniem ich definicji, klasyfikacji oraz zarysu metod ich przetwarzania. Ten obszar tematyczny stanowi drugie istotne zagadnienie niniejszej rozprawy. Jest ono rozwinięciem wcześniejszych prac badawczych związanych z przetwarzaniem i magazynowaniem strumieni danych, w wyniku których powstały cztery publikacje [66, 36, 37, 67]^{W} i dwa patenty [44, 41]^{W}.

Z owego zagadnienia wynika bezpośrednio kolejne, stanowiące próbę praktycznego ujęcia tematu przetwarzania danych kontekstowych: model strumieniowej hurtowni danych kontekstowych. Został on przedstawiony w niniejszej rozprawie jako kompletny system składowania i przetwarzania danych kontekstowych, ukierunkowany na wykrywanie oraz weryfikację anomalii krytycznych. W tym zakresie stawiana jest druga teza niniejszej rozprawy:

Teza 2 *Możliwe jest zaprojektowanie strumieniowej hurtowni danych zorientowanej na przetwarzanie wielkich zbiorów danych kontekstowych, wykorzystującej wielotorowy model przetwarzania danych, w którym analiza danych krytycznych jest wsparta przez przeprowadzaną niezależnie wieloaspektową analizę danych kontekstowych, w celu uwiarygodnienia wyników tej pierwszej.*

W ramach modelu strumieniowej hurtowni danych kontekstowych zaproponowano i opisano szereg metod i modeli, przeznaczonych do wspierania przetwarzania danych kontekstowych. Są to w szczególności: silnik strumieniowej kostki CUBIT oraz indeks przestrzenny BRI. Ta pierwsza jest odpowiednikiem kostki OLAP dla wielowymiarowych danych strumieniowych. Drugie rozwiązanie to wielowymiarowy bitowy indeks zakresowy, wspierający wykonywanie zapytań o agregaty zakresowe w wielowymiarowej przestrzeni cech.

Ostatnim zagadnieniem omawianym w niniejszej rozprawie jest problem efektywnego dostarczania agregatów wielowymiarowych. W ramach tego problemu zaprojektowano trzy nowe adaptacyjne algorytmy stronicowania, przeznaczone dla omówionego silnika CUBIT. Algorytmy te wykorzystują metody optymalizacji wielokryterialnej do zapewnienia należytej jakości usług, zarówno klienta (użytkownika), jak i źródła (bazy) danych. Rzeczone algorytmy zostały poddane analizie weryfikacyjnej oraz porównawczej – zarówno pomiędzy sobą, jak i z poprzednią generacją algorytmów, opisaną

w literaturze [32]. Wyniki badań zostały przedstawione w publikacji będącej obecnie w recenzji [68]^{W}. Badania przeprowadzono przy użyciu dwóch zaproponowanych metryk jakości usług dla strumieniowych hurtowni danych. W tym zakresie stawiana jest trzecia i ostatnia teza niniejszej rozprawy:

Teza 3 *Zastosowanie metod optymalizacji wielokryterialnej w procesie stronicowania w strumieniowych hurtowniach danych oraz uwzględnienie bieżących parametrów pracy i istniejących ograniczeń, pozwala na zwiększenie jakości usług, rozumianej zarówno jako poprawę efektywności i ciągłości dostarczania danych użytkownikowi, jak również zmniejszenie obciążenia źródła danych.*

1.3 Struktura rozprawy

Drugi rozdział rozprawy opisuje ewolucję modeli hurtowni danych, skupiając się przede wszystkim na strumieniowych hurtowniach danych, jako tle dla dalszych zagadnień poruszanych w rozprawie. W rozdziale tym dokonano obszernego przeglądu literatury oraz przedstawiono historię prac badawczych nad Materializowaną Listą Agregatów (MAL), która jako jeden z głównych komponentów modelu strumieniowej hurtowni danych, stała się inspiracją do opracowania modelu silnika strumieniowej kostki CUBIT.

Rozdział trzeci rozprawy przedstawia dokładny model sieci stacji paliw, jako źródła strumieni danych. W rozdziale tym dokonano klasyfikacji danych paliwowych oraz omówiono podstawowe zjawiska fizyczne zachodzące na stacjach paliw. Dodatkowo rozdział ten zawiera model konceptualny przykładowej hurtowni danych paliwowych – jego celem jest zwrócenie uwagi na istotne aspekty analizy takich danych oraz ich wielowymiarowość.

Rozdział czwarty rozprawy opisuje anomalie występujące w procesie dystrybucji i składowania paliw płynnych. Przedstawiona została klasyfikacja anomalii oraz opis metody wykrywania wycieków paliwa. Metoda ta stanowi jedną z kontrybucji pobocznych rozprawy. W rozdziale streszczono w formie graficznej wyniki ewaluacji tejże metody i sformułowano wnioski, stanowiące przyczynek do dalszych badań w zakresie analizy danych kontekstowych.

Rozdział piąty rozprawy jest pierwszym z rozdziałów zawierających kontrybucje główne – omawia model teoretyczny danych kontekstowych. W rozdziale podano definicje formalne danych kontekstowych, dokonano ich klasyfikacji oraz podano modele przetwarzania poszczególnych ich rodzajów. Zdefiniowano również siedem poziomów kontekstowości, które określają stopień uwzględniania danych kontekstowych w obliczeniach.

Rozdział szósty rozprawy przedstawia propozycję modelu strumieniowej hurtowni danych kontekstowych, będącej wielotorowym systemem składowania i przetwarzania danych na różnych poziomach kontekstowości. W rozdziale omówiono ogólny zarys architektury, a także zaprezentowano modele

logiczne trzech rodzajów baz danych dla trzech rodzajów danych kontekstowych – bazy te wchodzą w skład przytoczonego modelu strumieniowej hurtowni danych kontekstowych, stanowiąc dla niego warstwę składowania danych.

Rozdział siódmy rozprawy omawia wybrane modele procesów, związanych z funkcjonowaniem strumieniowego serwera OLAP, istotnych dla zaproponowanej w rozprawie strumieniowej hurtowni danych kontekstowych. Są to w szczególności: silnik strumieniowej kostki CUBIT, będący rozwinięciem koncepcji silnika Materializowanej Listy Agregatów oraz wielowymiarowy bitowy indeks zakresowy BRI.

Rozdział ósmy rozprawy zawiera propozycje trzech nowych algorytmów stronicowania dla strumieniowych hurtowni danych, które cechują się adaptacyjnością, elastycznością i przewidywalnością w stosunku do swoich poprzedników. Algorytmy te wykorzystują metody optymalizacji wielokryterialnej. W rozdziale przedstawiono obszerny opis procesu projektowania tych algorytmów, uwzględniający między innymi definicje przestrzeni rozwiązań, funkcji celu oraz strategii wyboru rozwiązań niezdominowanych.

Rozdział dziewiąty rozprawy wprowadza dwie nowe metryki jakości usług dla strumieniowych hurtowni danych. Sa to: jakość usług konsumenta, rozumiana jako efektywność dostarczania nowych danych użytkownikowi oraz jakość usług producenta, rozumiana jako stabilność funkcjonowania źródła danych. Ponadto, rozdział ten zawiera obszerne sprawozdanie z wykonanych badań eksperymentalnych.

Dziesiąty i zarazem ostatni rozdział rozprawy, stanowi jej podsumowanie. Zawiera spostrzeżenia i wnioski z ogółu wykonanych prac badawczych, a także wskazuje potencjalne plany rozwoju i nowe perspektywy badawcze.

Rozdział 2

Rozwój modeli strumieniowych hurtowni danych

Niniejszy rozdział przedstawia ewolucję modeli strumieniowych hurtowni danych. W rozdziale omówiono różne definicje i podejścia do tego zagadnienia, ze wskazaniem kluczowych cech strumieniowych hurtowni danych. Ponadto, przedstawiono również historię prac badawczych nad silnikiem Materializowanej Listy Agregatów, który jest istotnym komponentem modelu strumieniowej hurtowni danych i zarazem punktem wyjścia do dalszych badań.

2.1 Model strumieniowej hurtowni danych

Strumieniowa hurtownia danych (StrDW, od ang. *Stream Data Warehouse*), choć pod tym pojęciem występuje w literaturze, nie doczekała się pojedynczej definicji formalnej. Nie istnieje też żaden standard, który by określał w szczegółach jej założenia oraz budowę. Co więcej, termin ten jest często używany wymiennie z hurtownią *czasu rzeczywistego* lub hurtownią *aktywną*, do określenia różnych adaptacji i modyfikacji dobrze znanego, klasycznego modelu hurtowni danych.

Rzeczony brak definicji i standardu wskazywany jest w literaturze. Agrawal [4] podaje: „Niestety, brak jest jasności w sprawie krytycznych składowych technologii, które wyróżniałyby systemy analityki biznesowej (BI) czasu rzeczywistego od tradycyjnych hurtowni danych i rozwiązań BI".[1] Niemniej, ten sam autor stwierdza dalej, iż analiza strumieni danych, przetwarzanie złożonych zdarzeń i integracja danych w czasie rzeczywistym są

[1] W oryginale: „Unfortunately, there is no clarity about the critical technology components that distinguish a real-time business intelligence system from traditional data warehousing and business intelligence solutions" [4].

kluczowymi cechami charakterystycznymi systemów, które można określić mianem strumieniowych hurtowni danych.

Pojęcie czasu rzeczywistego (ang. *real-time*) w kontekście baz danych ma długą historię. Już w 1995 roku odwoływano się do systemów bazodanowych typu *soft real-time*, aktualizowanych przez strumienie danych [2]. Późniejsze prace, datowane na pierwszą dekadę XXI wieku, skupiają się już na adaptacji klasycznych hurtowni danych do obsługi strumieni danych i pracy w czasie rzeczywistym. Systemy te w sposób aktywny reagują na zmieniającą się rzeczywistość, oferując między innymi realizację złączeń dokonywanych pomiędzy utrwalonymi danymi historycznymi a bieżącymi danymi napływającymi w formie strumienia [114, 113].

W kolejnych pracach badawczych, strumieniowa hurtownia danych jest postrzegana jako system zdolny do wykonywania zapytań na danych bieżących w czasie rzeczywistym, jak i głębokich analiz na danych historycznych, co podkreśla jej dwoistą naturę. Naturę tę można rozumieć jako pomost między teraźniejszością a przeszłością [27]. W konsekwencji, jedną z kluczowych metryk określających sprawność strumieniowej hurtowni danych staje się *świeżość* danych, interpretowana jako różnica czasu pomiędzy ostatnio zapisanymi a najnowszymi danymi [8, 23]. Golab i Johnson [26] podają następującą definicję:

Definicja 2.1 *Strumieniowa hurtowania danych jest systemem zarządzania strumieniami danych, który przechowuje bardzo długą historię, np. lat lub dekad i jednocześnie jest hurtownią danych, która jest nieustannie uaktualniana.*[2]

W literaturze, oprócz przytoczonej definicji 2.1 można znaleźć jeszcze inne, podobne: „Strumieniowa hurtownia danych jest nową technologią zarządzania danymi, która pozwala na niemalże ciągłe odświeżanie widoków, w miarę pojawiania się nowych danych, co umożliwia płynną integrację monitorowania w czasie rzeczywistym z eksploracją dużych zbiorów danych”[3] [7]; „Strumieniowa hurtownia danych łączy cechy tradycyjnej hurtowni danych z systemami przetwarzania strumieni danych”[4] [24]; „Strumieniowa hurtownia danych nieustannie przyjmuje strumienie danych, wylicza złożone wartości pochodzne i przechowuje długoterminową historię”[5] [87].

[2]W oryginale: „A stream warehouse is a data stream management system (DSMS) that stores a very long history, e.g. years or decades; or equivalently a data warehouse that is continuously loaded” [26].

[3]W oryginale: „Stream data warehousing is a new data management technology that allows nearly-continuous view refresh as new data arrive, which enables seamless integration of real-time monitoring and business intelligence with long-term data mining” [7].

[4]W oryginale: „Streaming data warehouses combines the features of traditional data warehouses and data stream systems” [24].

[5]W oryginale: „A data stream warehouse continually ingests data streams, computes complex derived data products, and stores long term histories” [87].

Wyczerpujący raport na temat rozwiązań analityki biznesowej (BI, od ang. *Business Intelligence*), włączając w to rozwiązania BI czasu rzeczywistego, został podany przez Chaudhuri et al. [17]. Znaleźć tam można stwierdzenie, iż „celem analityki biznesowej czasu (prawie) rzeczywistego jest zmniejszenie czasu pomiędzy akwizycją danych operacyjnych a ich przetwarzaniem".[6] W tym raporcie podkreślone zostaje znaczenie minimalizacji opóźnienia, a więc zwiększenie wspomnianej już świeżości danych.

Podsumowując, większość przytoczonych tutaj prac badawczych, rozważając pojęcie strumieniowej hurtowni danych, skupia się głównie na aspekcie aktualizacji hurtowni za pomocą strumienia danych. Część badań bierze pod uwagę również inne zagadnienia, takie jak podwójna natura strumieniowych hurtowni danych [27], format przechowywanych w niej danych [26], rodzaj przeprowadzanych analiz [87, 4] oraz zapewnienie niskiego opóźnienia podczas przesyłu danych [17].

2.1.1 Kluczowe cechy strumieniowej hurtowni danych

Na potrzeby niniejszej rozprawy przyjmuje się, że cechami charakterystycznymi strumieniowej hurtowni, które odróżniają ją zarówno od klasycznych hurtowni danych, jak i od systemów przetwarzania strumieni danych, są:

a) zasilanie strumieniami danych i aktualizacja natychmiastowo po pojawieniu się nowych danych;

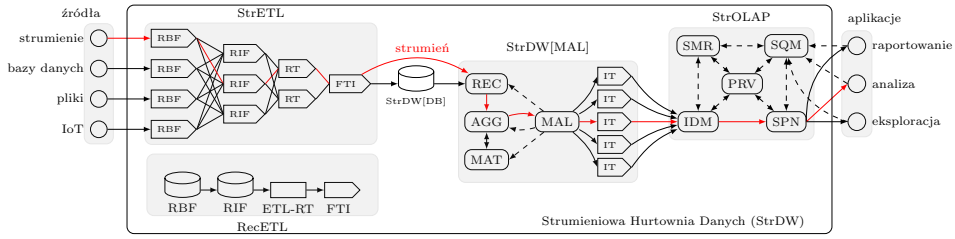
b) wykorzystywanie schematu bazy danych przystosowanego do przechowywania strumieni danych oraz efektywnego wykonywania zapytań o dłuższe sekwencje rekordów;

c) stosowanie wewnętrznego przesyłu i przetwarzania danych w postaci strumieni, przy wsparciu struktur danych znajdujących się w całości w pamięci operacyjnej;

d) wykonywanie analiz o naturze strumieniowej lub zbliżonej oraz dostarczanie wyników w formie strumieni odpowiedzi.

Pierwsza i ostatnia cecha mogą zostać określone mianem cech *zewnętrznych*, gdyż charakteryzują wejście i wyjście strumieniowej hurtowni danych. Z drugiej strony, dwie środkowe cechy sklasyfikować można jako *wewnętrzne*, ponieważ określają procesy zachodzące wewnątrz hurtowni. Podczas gdy większość przytoczonych uprzednio prac badawczych skupia się na cechach zewnętrznych, analiza tych wewnętrznych jest równie istotna i jest omawiana szeroko w niniejszej rozprawie.

[6] W oryginale: „The goal of near real-time BI (also called operational BI or just in-time BI) is to reduce the latency between when operational data is acquired and when analysis over that data is possible" [17].



Rysunek 2.1: Architektura strumieniowej hurtowni danych [29]

Pełny model strumieniowej hurtowni danych, obejmujący zarówno cechy zewnętrzne, jak i wewnętrzne, przedstawiony został w monografii *Zaawansowane hurtownie danych* [29]. Ilustrację tego modelu przedstawia rysunek 2.1. Wprowadzono na nim następującą konwencję: strzałki ciągłe oznaczają przepływ danych (w formie strumieni), a strzałkami przerywanymi oznaczono przepływ sterowania pomiędzy poszczególnymi komponentami.

2.1.2 Architektura strumieniowej hurtowni danych

Omówiony model strumieniowej hurtowni danych (StrDW) składa się z kilku kluczowych komponentów. Ponadto, model zawiera również specyfikację źródeł i aplikacji pracujących na wyjściu. Dane wejściowe są podawane w formie strumieni na wejście pierwszego głównego modułu: silnika strumieniowej ekstrakcji danych StrETL (ang. *Stream Extract, Transform and Load*). Jest on odpowiedzialny za ich wstępne przetwarzanie oraz wprowadzanie do docelowego formatu przyjętego w hurtowni. Silnik ten współpracuje z silnikiem odtwarzania RecETL (ang. *Recovery ETL*), który zapewnia mechanizm tworzenia kopii zapasowych i odtwarzania procesów ekstrakcji na wypadek awarii. Wykorzystuje w tym celu komponenty RBF (ang. *Remote Buffer Framework*) i RIF (ang. *Remote Integrator Framework*) [64].

Po wstępnym przetworzeniu w silniku StrETL, dane są ładowane do hurtowni (wewnętrznej bazy danych). Oprócz surowych danych, przechowuje ona również agregaty i wszelkiego rodzaju metadane, opisujące przetwarzane strumienie danych. W tym zakresie strumieniowa hurtownia danych nie odbiega znacząco od funkcjonowania klasycznej hurtowni danych. Różnica jednak zaczyna się rysować na dalszych etapach przetwarzania danych.

W odróżnieniu od klasycznej hurtowni, w modelu StrDW występuje dodatkowy komponent stanowiący pomost między bazą danych a silnikiem analitycznym StrOLAP (ang. *Stream On-Line Analytical Processing*). Komponent ten nazywany jest silnikiem Materializowanej Listy Agregatów (MAL, od ang. *Materialized Aggregate List*) [53]. Jest on swego rodzaju widokiem dla strumieni danych zapisanych w hurtowni. Jego zadaniem jest zapewnienie efektywnego transferu danych, na różnych poziomach agregacji. Silnik MAL dodatkowo umożliwia przekazywanie źródłowego strumienia danych

bezpośrednio do odbiorcy, z pominięciem procesu zapisu w bazie danych. Silnik MAL razem z bazą danych pełni funkcję repozytorium strumieni danych.

Silnik MAL składa się z kilku modułów, z których najważniejsze to: moduł rekonstrukcji (REC) odpowiedzialny za pozyskiwanie surowych danych z bazy danych; moduł agregacji (AGG) zajmujący się agregacją danych w wymiarze czasu; moduł materializacji (MAT), którego zadaniem jest odtwarzanie zmaterializowanych agregatów. Ponadto, w silniku MAL wyróżnić można itertory (IT), które są odpowiedzialne za dostęp do zrekonstruowanych, zagregowanych i potencjalnie zmaterializowanych strumieni danych.

Modułem nadrzędnym i zarządzającym całą strumieniową hurtownią danych jest silnik StrOLAP, który jest adaptacją silnika OLAP znanego z modelu klasycznej hurtowni danych. Zapewnia on usługi analiz i raportowania dla danych strumieniowych, pobranych z silnika MAL. Silnik StrOLAP składa się z kilku modułów, spośród których najważniejsze to: zarządca zapytań strumieniowych (SQM), repozytorium metadanych strumieni (SMR), sieć przetwarzania strumieni danych (SPN), zarządca danych wejściowych (IDM) oraz silnik ochrony prywatności (PRIV).

Strumieniowa hurtownia danych zdolna jest to przetwarzania zapytań analitycznych na danych zarówno historycznych, odczytanych z bazy danych przez silnik MAL, jak również na danych ulotnych (strumieniowych), uzyskanych wprost z silnika StrETL. Płynne przełączanie się pomiędzy danymi historycznymi a bieżącymi jest zadaniem silnika Materializowanej Listy Agregatów. Na rysunku 2.1 za pomocą czerwonych strzałek naniesiono przepływ strumieni danych, który omija proces archiwizacji w bazie danych.

2.2 Materializowana Lista Agregatów

Materializowana Lista Agregatów stanowi istotny komponent modelu strumieniowej hurtowni danych. Co więcej, ze względu na swoją rolę, stała się punktem wyjścia do dalszych badań, będących przedmiotem niniejszej rozprawy. Zasadne jest zatem przybliżenie historii jej rozwoju i dokładne omówienie podstawowych założeń.

Materializowana Lista Agregatów (MAL) w literaturze ukazana została jako struktura współpracująca z indeksem przestrzennym i wspomagająca wykonywanie zapytań zakresowych [53]. Ponadto została ona zdefiniowana jako „połączenie pamięciowej struktury danych i zbioru algorytmów”[7] [60] oraz przyrównana do „iteratora bazy danych” [56]. Wskazano także trzy najważniejsze cechy charakterystyczne MAL: materializację agregatów, uniwersalność pod względem typu danych i źródeł oraz kompatybilność ze

[7] W oryginale: „A combination of a memory structure and a set of dedicated algorithms” [60].

znanym interfejsem listy i iteratora [32]. Na potrzeby niniejszej rozprawy przyjmuje się następującą definicję MAL:

Definicja 2.2 *Materializowana Lista Agregatów (MAL) jest deskryptorem strumienia danych, będącego wynikiem wykonania pewnego zapytania lub powstałego na skutek odtworzenia utrwalonych wyników wcześniejszych zapytań.*

Zgodnie z definicją 2.2, MAL stanowi jednoznaczny opis strumienia danych spełniającego określone w zapytaniu parametry i może służyć jako *uchwyt* (ang. *handle*), za pośrednictwem którego możliwy jest dostęp do znajdujących się w nim danych. Stanowi to analogię do deskryptora (uchwytu) pliku w systemie operacyjnym. Ponadto z powyższego wynika, iż MAL nie jest strukturą przechowującą dane, a raczej przepisem na ich uzyskanie. Dostęp do samych danych realizowany jest za pośrednictwem iteratorów (kursorów), czyli obiektów uogólniających sekwencyjny dostęp do dowolnych struktur danych. W przypadku Materializowanej Listy Agregatów, iterator jest wyposażony w kolejkę (pamięć), w której przechowywany jest aktualnie przeglądany fragment strumienia danych oraz mechanizm zarządzający tą kolejką. W sposób formalny określa to definicja:

Definicja 2.3 *Iterator MAL jest obiektem zapewniającym dostęp do wskazanego znacznikiem czasowym podzbioru danych opisywanych przez Materializowaną Listę Agregatów, poprzez ich tymczasowe przechowywanie w skojarzonej z iteratorem pamięci oraz dostarczenie metod umożliwiających ich sekwencyjny odczyt.*

Materializowana Lista Agregatów oraz operujące na niej iteratory tworzą razem mechanizm, zwany silnikiem MAL. Może on zostać określony jako warstwa pośrednicząca pomiędzy dynamicznymi i statycznymi źródłami danych a modułami analitycznymi i raportującymi. Silnik MAL można również postrzegać jako tymczasową pamięciową bazę danych, wspomagającą realizację zapytań wykonywanych w strumieniowej hurtowni danych. W konsekwencji, silnik MAL definiuje się w sposób następujący:

Definicja 2.4 *Silnik MAL jest obiektem łączącym w sobie pamięciowe struktury danych oraz operujące na nich algorytmy, którego przeznaczeniem jest pośredniczenie w wymianie danych pomiędzy dwoma procesami: producenta i konsumenta, o odmiennej i zmiennej w czasie charakterystyce pracy, przy jednoczesnym zapewnieniu należytej jakości usług obu tych procesów, rozumianej jako minimalizacja zarówno czasu oczekiwania konsumenta, jak i obciążenia producenta.*

Definicja 2.4 wprowadza pojęcie jakości usług i przypisuje silnikowi MAL rolę jej zapewniania. Jest to na tyle istotne, iż podkreśla znaczenie silnika

MAL jako modułu pośredniczącego w wymianie danych i aktywnie biorącego udział w kształtowaniu charakterystyki tego procesu, zgodnie z założonymi standardami jakości. Funkcjonowanie silnika MAL jest zdeterminowane przez algorytm wypełniania stron, który zarządza powiązaną z iteratorem kolejką. Ma on bezpośredni wpływ na jakość usług, zarówno z punktu widzenia producenta, jak i konsumenta, gdyż modyfikuje intensywność strumienia danych przepływającego przez silnik MAL. Poniżej przedstawiono definicję algorytmu wypełniania stron:

Definicja 2.5 *Algorytm wypełniania stron jest algorytmem operującym wewnątrz iteratora MAL, który bazując na aktualnym rozmiarze kolejki, jak również biorąc pod uwagę charakterystykę czasową procesu produkcji i konsumpcji danych, określa właściwą porcję danych, która będzie dostarczona do kolejki podczas następnej procedury ładowania, jak również moment rozpoczęcia tej procedury.*

2.2.1 Geneza Materializowanej Listy Agregatów

Koncepcja Materializowanej Listy Agregatów narodziła się w pierwszej dekadzie XXI wieku. Jej geneza wiązała się z problemem przetwarzania danych telemetrycznych generowanych przez liczniki mediów (woda, prąd, gaz) [51]. Dane te gromadzone były w uaktualnianej periodycznie, rozproszonej i przestrzennej hurtowni danych nastawionej na analizę zużycia poszczególnych typów mediów (ang. *Distributed Spatial Data Warehouse* – DSDW). Stosowanie rozwiązań znanych z hurtowni danych do analizy dynamicznie zmieniających się danych telemetrycznych jest ciągle rozwijającym się trendem [16]. Dominującymi zapytaniami w tego typu systemach są kwerendy z funkcjami agregującymi i predykatami zakresowymi zdefiniowanymi na przestrzeni dwuwymiarowej. Odpowiadają one na pytanie o średnie lub maksymalne wartości pomiarów w zadanych obszarach w różnych okresach czasu [13, 103, 100].

Analiza danych w hurtowni DSDW wymagała częstego pobierania dłuższych sekwencji pomiarów pochodzących z konkretnych źródeł danych. Sekwencje te były agregowane w stałych oknach czasowych, tak aby tworzyły ciąg agregatów reprezentujących uogólnione zużycie konkretnego medium w dłuższym przedziale czasu. Problem wyznaczania agregatów w oknach czasowych może zostać rozwiązany przy użyciu wyspecjalizowanych zapytań SQL [14] lub też po stronie aplikacji, po uprzednim pobraniu niezbędnych danych surowych. Niemniej, to pierwsze rozwiązanie nie zawsze jest możliwe do zastosowania, ze względu na specyfikę danych. Dlatego też, w przypadku hurtowni DSDW, zdecydowano się na drugą opcję.

W celu przyspieszenia wykonywania zapytań, hurtownia danych DSDW wykorzystywała przestrzenny indeks agregujący – drzewiastą strukturę przechowującą w swoich węzłach wyznaczone uprzednio agregaty. Budowa takie-

go indeksu bazowała na sprawdzonej idei R-drzewa (ang. *R-tree*) [76], a rozwiniętej później jako aR-drzewo (ang. *aR-tree*) – agregacyjne R-drzewo [111]. Modyfikacja polegała na przechowywaniu zagregowanych wartości w poszczególnych węzłach indeksu. Ze względu na fakt, iż zastosowana w hurtowni DSDW struktura drzewiasta indeksowała źródła danych generujące nieustannie nowe pomiary, poszczególne węzły drzewa musiały przechowywać całe ciągi agregatów w formie list. Jest to sytuacja odmienna w stosunku do klasycznego indeksu agregującego, przechowującego jedynie pojedyncze wartości. W konsekwencji, rozmiar danych gromadzonych w węzłach indeksu rósł nieprzerwanie, co powodowało problemy z brakiem dostępnej pamięci operacyjnej.

Rozwiązanie problemu przetwarzania danych telemetrycznych w hurtowni DSDW, polegało na przeprowadzeniu materializacji wybranych list agregatów [50] – były one serializowane i zapisywane jako ciąg bitów do bazy danych, w kolumnach typu BLOB (ang. *Binary Large Object*). Przechowywanie danych w mocno zdenormalizowanej formie binarnej jest praktyką, która może znacznie przyspieszyć operacje wyszukiwania i odczytywania. Ma to miejsce w przypadku takich zbiorów danych, które zazwyczaj czytane i zapisywane są w całości, podobnie jak listy agregatów. Podobne podejście opisano w literaturze do zapisu danych z dziedziny architektury budynków [96].

Operacji materializacji w hurtowni DSDW poddawano te listy agregatów, które znajdowały się w najrzadziej odczytywanych węzłach (strategia LRU – ang. *Last Recently Used*). Ponadto materializowano wszystkie węzły w momencie usuwania z pamięci całej struktury indeksującej. Rozwiązanie to nazwano VMAT (ang. *Virtual Memory Aggregation Tree*), co było nawiązaniem do mechanizmów stronicowania i pamięci wirtualnej stosowanych w systemach komputerowych [11] i używanych przez systemy operacyjne [109]. Próba odczytu zmaterializowanej i usuniętej z pamięci listy agregatów skutkowała jej odtworzeniem z bazy danych i ponownym umieszczeniem w pamięci.

Zastosowany mechanizm materializacji list agregatów, a więc wyników wykonanych uprzednio obliczeń, jest często wykorzystywany w hurtowniach danych pod nazwą widoków materializowanych. Ich idea polega na składowaniu w osobnych tabelach wyników zapytań wykonanych na bazie danych, tak aby w przyszłości możliwe było ich wykorzystanie, bez potrzeby ponownego wykonywania zapytań. Ze względu na ograniczoną pamięć, najistotniejszym problemem przy zarządzaniu widokami w hurtowniach danych jest wybór tych, które mają być materializowane [119, 75, 81].

Kolejnym problemem występującym w hurtowni DSDW była aktualizacja zmaterializowanych list agregatów, która wykonywana była w momencie pojawienia się nowych danych [55, 57]. Miało to szczególne znaczenie w przypadku agregatów wyliczonych na podstawie niepełnego okna czasowego. Po zaktualizowaniu list agregatów znajdujących się na niższych po-

ziomach drzewa indeksu, wyższe węzły były aktualizowane wyłącznie przy użyciu agregatów niższego rzędu, bez konieczności przetwarzania surowych danych. Problem aktualizacji widoków materializowanych, obok opisanego wcześniej problemu ich selekcji, jest równie często analizowany w literaturze [25].

Pomimo zastosowania wyżej wymienionych mechanizmów, zarządzanie długimi listami agregatów okazało się dość złożonym problemem. Po pierwsze ich rozmiar był znaczący, a po drugie istniała konieczność aktualizacji ich zmaterializowanych postaci, co wymagało wczytania do pamięci i ponownej materializacji. Powyższe stało stało się motywacją do wszczęcia prac nad nowym rozwiązaniem, wolnym od wspomnianych wad. W ten sposób powstała koncepcja Materializowanej Listy Agregatów [53]. Główna jej idea wyrażała się w uniwersalności, rozumianej poprzez zarówno wsteczną kompatybilność, jak i funkcjonowanie niezależnie od rozmiaru danych.

2.2.2 Rozwój Materializowanej Listy Agregatów

Rozwiązanie VMAT wykorzystywało interfejs listy, czyli sekwencyjnej kolekcji danych. W przypadku Materializowanej Listy Agregatów, zasadne było zatem zachowanie tego interfejsu, ze względu na wspomnianą już wsteczną kompatybilność. Jednakże MAL nie miała być rodzajem kontenera dla danych, a raczej swego rodzaju obiektem pośredniczącym (ang. *proxy*). Dostęp do samych danych odbywał się za pomocą iteratora umożliwiającego sekwencyjne pobieranie kolejnych agregatów – stanowiło to analogię do obiektu o tej samej nazwie występującego w wielu językach programowania oraz do kursora w systemach bazodanowych. W literaturze można znaleźć podobne wykorzystanie idei iteratora (kursora), aczkolwiek przeznaczonego do zupełnie celu – nawigacji po dokumentach XML [80].

Dotychczasowa lista agregatów, po zastosowaniu koncepcji MAL, została podzielona na fragmenty (nazwane stronami) o stałym rozmiarze, które stanowiły podstawową jednostkę wymiany danych. Partycjonowanie danych stanowi skuteczną technikę reorganizacji wewnętrznej reprezentacji danych i może być przeprowadzane zarówno pionowo [107], gdy dane są dzielone jakościowo, jak i poziomo [5], gdzie dane są dzielone ilościowo. W przypadku MAL obrana, została ta druga strategia, gdyż celem było usprawnienie operacji wejścia/wyjścia poprzez zmniejszenie liczby transakcji i jednoczesne zwiększenie ich zakresu.

Odmiennie niż w przypadku poprzedniego rozwiązania (VMAT), materializacji były poddawane pojedyncze strony (nie całe listy), a sam proces odbywał się zaraz po wytworzeniu danej strony (nie w przypadku wykrycia braku wolnej pamięci). Porównanie obu rozwiązań dało wynik zdecydowanie korzystniejszy dla Materializowanej Listy Agregatów [54, 56]. Proces utrwalania wyników czasochłonnych obliczeń zaraz po ich wykonaniu w celu umożliwienia ich późniejszego wykorzystania nazywany jest również *memo-*

izacją i jest techniką stosowaną między innymi przy optymalizacji procesu kompilacji [9] oraz w funkcyjnych językach programowania [108, 1].

Iteratory odgrywały w MAL kluczową rolę, gdyż odpowiadały za tymczasowe buforowanie agregatów. Każdy z iteratorów zawierał tablicę o stałym rozmiarze, będącym wielokrotnością rozmiaru pojedynczej strony. Tablica ta wypełniona była aktualnie przeglądanym fragmentem listy agregatów, która w całości przechowywana była w bazie danych (dane zmaterializowane) lub też tworzona była na bieżąco (surowe dane z bazy były agregowane). W miarę pobierania kolejnych agregatów za pomocą iteratora, doczytywane były kolejne strony. Przypominało to nieco mechanizm pobierania wstępnego (ang. *prefetch*), stosowany między innymi w przyspieszaniu ładowania stron internetowych [97] oraz treści multimedialnych [91].

Bardzo istotną kwestią w funkcjonowaniu Materializowanej Listy Agregatów okazał się problem efektywnego doczytywania nowych danych do tablic iteratorów. Zdefiniowano trzy podstawowe strategie rozwiązujące ten problem (nazwane algorytmami wypełniania stron): TRIGG, SPARE i RENEW [53]. Każda z tych strategii określała warunki, po spełnieniu których rozpoczynany był proces ładowania nowych danych. Algorytmy te różniły się między sobą stopniem nadmiarowości, rozumianej jako ilość pobranych zawczasu danych do bufora. W swojej istocie problem wypełniania stron jest zbliżony do zagadnień zarządzania kolejkami. Analiza kolejek jest bardzo często podejmowaną tematyką w przypadku systemów przetwarzania strumieni danych [85, 129], baz danych [110] oraz różnego typu zastosowań sieciowych [106, 3, 77].

W toku dalszych badań dokonano obszernej analizy problemu wypełniania stron, wraz z podaniem definicji poszczególnych algorytmów, wyznaczeniem ich złożoności czasowej oraz przeprowadzeniem dokładnych badań eksperymentalnych [32]. Scharakteryzowano ponadto optymalne warunki pracy dla każdego z trzech algorytmów. Algorytm TRIGG (o najmniejszej nadmiarowości) okazał się najlepszy dla przypadków, w których czas wypełniania pojedynczej strony był mniejszy od czasu jej konsumpcji. Algorytm SPARE (o średniej nadmiarowości) okazał się odpowiedni dla sytuacji, gdy oba te czasy były porównywalne. Algorytm RENEW (o najwyższej możliwej nadmiarowości) znalazł swoje zastosowanie w przypadkach najtrudniejszych, dla których czas ładowania pojedynczej strony był wyższy od czasu jej konsumpcji.

Każdy ze wspomnianych trzech algorytmów cechował się stałymi parametrami pracy, które czyniły go odpowiednim do z góry określonej sytuacji. Jednakże, w przypadku zmiennych warunków pracy, wybór konkretnego algorytmu nie był już prostą decyzją. Co więcej, wybór ten spoczywał na użytkowniku, który musiał być świadomy charakterystyki czasowej odpowiednich modułów: zarówno Materializowanej Listy Agregatów, jak i obejmującej jej hurtowni danych.

Przy projektowaniu poszczególnych algorytmów, położono duży nacisk

na zastosowanie wielowątkowości w procesie wypełniania stron [52]. Każdy z algorytmów zlecał załadowanie kolejnej strony wątkom rezydującym w puli, co umożliwiało współbieżne ładowanie kilku stron jednocześnie. Motywacją do takiego działania była chęć zapewnienia płynnej pracy MAL, nawet w przypadku szybszego tempa konsumpcji danych w stosunku do ich produkcji. Niemniej, zrównoleglanie operacji wejścia/wyjścia jest wykonalne jedynie do pewnego stopnia, ze względu na ograniczoną przepustowość pamięci masowych. Zostało to wzięte pod uwagę poprzez przyjęcie założenia, że czas ładowania pojedynczej strony jest wprost proporcjonalnie zależny od liczby jednocześnie pracujących wątków. Zaproponowano w tym celu istnienie współczynnika proporcjonalności, związanego z udziałem narzutu realizacji wielowątkowej w całkowitym czasie pracy. Współczynnik ten określał, jak bardzo zwiększenie liczby wątków wpływa na wydłużenie czasu pracy każdego z nich. Należy tutaj zwrócić uwagę na fakt, iż w skrajnych przypadkach wartość takiego współczynnika może wynosić 1, co przekłada się na faktyczny brak wielowątkowości i sekwencyjne wykonywania operacji ładowania stron.

Początkowo Materializowana Lista Agregatów przedstawiana była wyłącznie jako narzędzie współpracujące ze statyczną bazą danych. Wraz z rozwojem koncepcji Internetu Rzeczy [128, 73, 74] i systemów informacji geograficznej [19], przetwarzanie strumieni danych stało się bardzo popularnym obszarem badań naukowych. Badano przy tym wiele zagadnień, takich jak równoważenie obciążenia [124] oraz analizę zmiany trendu (ang. *concept drift*) [130]. W konsekwencji MAL została umiejscowiona w modelu strumieniowej hurtowni danych telemetrycznych [58]. Następnym krokiem ku adaptacji MAL do pracy w środowisku strumieniowym, było wskazanie strumienia danych jako jednego z czterech głównych źródeł [59, 62] obok bazy danych surowych i zmaterializowanych oraz innych instancji MAL.

Kolejnym etapem w badaniach nad Materializowaną Listą Agregatów było wprowadzenie mechanizmu kontroli zasobów, a konkretniej przydzielonej pamięci do tablic iteratorów MAL [58, 62, 61]. Zdecydowano się na zastosowanie sprawdzonego rozwiązania puli zasobów [122], z której korzystały poszczególne iteratory MAL. Na podstawie analizy wybranych parametrów (takich jak stopnień materializacji danych, zdolność do ich materializacji i liczba obiektów w zapytaniu), część iteratorów MAL otrzymywała swoje własne tablice, a część współdzieliła je z innymi iteratorami. Dzięki takiemu rozwiązaniu, ilość alokowanej pamięci była pod ścisłą kontrolą, przy jednoczesnej minimalizacji czasu realizacji zapytań. Należy mieć jednak na uwadze fakt, iż w obecnych czasach coraz bardziej popularne stają się rozwiązania działające w pełni w pamięci operacyjnej (ang. *in-memory*) [88, 120], na które składają się zarówno platformy sprzętowe wyposażone w terabajty pamięci operacyjnej, jak i dedykowane pamięciowe bazy danych [99], a więc aspekt optymalizacji pamięciowej, pomimo że wciąż istotny, schodzi na drugi plan.

W pracy własnej [67]^{W} podjęto tematykę zasilania MAL danymi historycznymi i strumieniowymi. Rezultatem tych prac była częściowa realizacja kluczowych wymagań dla systemów przetwarzających strumienie danych znanych z literatury [123] – a konkretniej wymagania dotyczącego jednakowego przetwarzania danych historycznych i bieżących. Materializowana Lista Agregatów umożliwiała płynne przełączanie się pomiędzy statycznymi danymi zarchiwizowanymi w bazie danych a dynamicznymi danymi pobranymi wprost ze strumienia danych w sposób niewidocznych dla jej użytkownika. Dzięki temu MAL stała się pewnego rodzaju warstwą pośrednią (ang. *middleware*) pomiędzy źródłami danych: pamięcią trwałą i strumieniami danych a modułami analitycznymi i raportującymi hurtowni. W literaturze znane są przykłady stosowania modułów pośredniczących w hurtowniach danych w celu optymalizacji dostępu do danych [116].

Rozdział 3

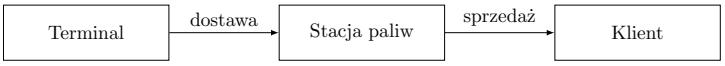
Motywujące podstawy badań: dystrybucja paliw płynnych

Niniejszy rozdział opisuje motywujące podstawy badań będących przedmiotem rozprawy doktorskiej. W pierwszej części przybliżony został przykład sieci stacji paliw jako źródła strumieni danych, które poddawane były analizie ukierunkowanej na wykrycie wycieków paliwa. Wnioski z tej analizy posłużyły następnie za podstawę do sformułowania głównego problemu badawczego poruszanego w rozprawie. W rozdziale dokonano również klasyfikacji danych paliwowych, z podziałem na trzy kategorie: dane pierwotne, wtórne i zagregowane. Rozdział kończy się przedstawieniem model konceptualnych dla hurtowni danych paliwowych.

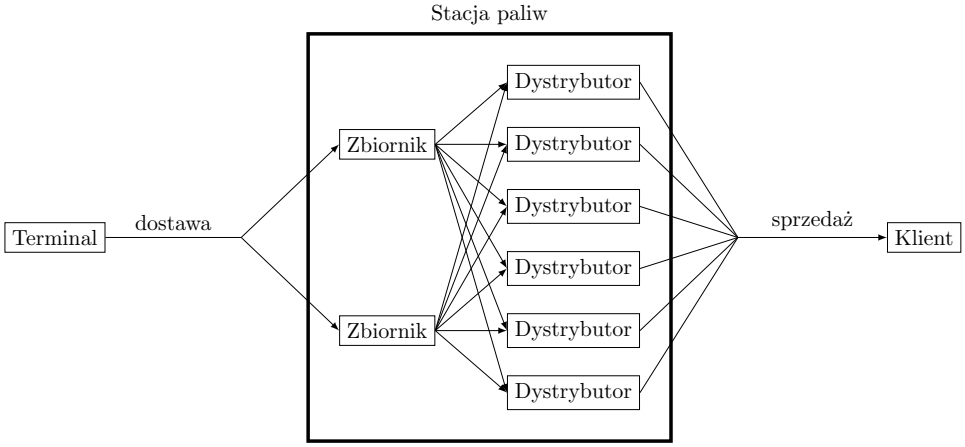
3.1 Sieć stacji paliw jako źródło danych

Dystrybucję paliw płynnych scharakteryzować można jako proces polegający na składowaniu i transportowaniu paliwa, począwszy od stacji źródłowych (terminali), poprzez sieć stacji paliw, a skończywszy na klientach końcowych. Składowanie paliwa odbywa się na każdym ze wskazanych trzech obiektów, zaś pomiędzy nimi następuje jego transport. Całość może zostać określona mianem *toru przepływu paliwa*. Należy zwrócić uwagę na fakt, iż warunkiem początkowym tego procesu jest istnienie gotowego produktu (paliwa płynnego). Wszelkie procesy chemiczne związane z obróbką substancji ropopochodnych nie są tutaj brane pod uwagę, gdyż odbywają się jeszcze przed właściwą dystrybucją. Rysunek 3.1 przedstawia schemat toru przepływu paliwa, będącego modelem procesu dystrybucji paliw płynnych.

Punktem startowym toru przepływu paliwa jest stacja źródłowa, zwana także terminalem. Zadaniem tego typu obiektów jest przechowywanie znacznych ilości paliw płynnych. Pełnią one w tym sensie rolę buforów pomiędzy



Rysunek 3.1: Schemat toru przepływu paliwa



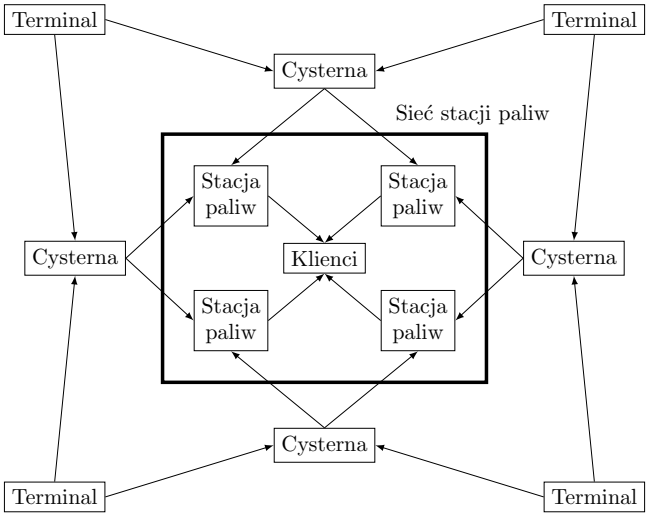
Rysunek 3.2: Model pojedynczej stacji paliw

zakładami zajmującymi się produkcją lub importem paliwa a jednostkami prowadzącymi jego sprzedaż. Z terminali paliwo dostarczane jest cysternami do właściwych stacji paliw, gdzie jest składowane w zbiornikach do czasu, aż zostanie sprzedane. Odbiorcami paliwa są klienci końcowi, którzy zużywają go w swoich pojazdach.

Każda stacja paliw wyposażona jest w podobną infrastrukturę. Oprócz zbiorników paliwa wyróżnić można dystrybutory, czyli urządzenia wyposażone w pompy, przepływomierze oraz nalewaki paliwa, zwane też pistoletami. Zadaniem dystrybutorów jest dostarczanie paliwa do pojazdów klientów końcowych oraz prowadzenie ewidencji jego zużycia. Rysunek 3.2 przedstawia model stacji paliw. Na rysunku uwzględniono również proces dostawy paliwa, który z punktu widzenia stacji może być traktowany w sposób uproszczony – jako pojedyncze zdarzenie.

Stacje paliw funkcjonują jako autonomiczne obiekty i są rozproszone na dużym obszarze geograficznym, tworząc sieć. Użycie terminu *sieć* zostało podyktowane istnieniem mechanizmu dostaw paliwa, który łączy ze sobą pojedyncze stacje. Zależność pomiędzy terminalem (jako początkiem dostawy), a stacją paliw (jako jej końcem) wyraża się relacją N:M. Podobna zależność występuje pomiędzy terminalem a cysterną biorącą udział w dostawie oraz pomiędzy cysterną a stacją paliw, na którą dana dostawa jest kierowana. Powyższe fakty wpływają na konieczność rozpatrywania sieci stacji paliw jako powiązanej ze sobą struktury, a nie zbioru niezależnych obiektów. Na rysunku 3.3 został przedstawiony model sieci stacji paliw.

Monitorowanie sieci stacji paliw wymaga rejestrowania różnych para-



Rysunek 3.3: Model sieci stacji paliw

metrów świata rzeczywistego. Dotyczy to przede wszystkim infrastruktury wchodzącej w skład poszczególnych stacji paliw, jak również procesu dostarczania paliwa. Historycznie proces ten przeprowadzany był ręcznie, przy pomocy prostych narzędzi pomiarowych i prowadzonej dokumentacji. Obecnie możliwe jest ciągłe pozyskiwanie dokładnych pomiarów dzięki zastosowaniu sieci sensorów, a w dalszej kolejności gromadzenie i analiza danych w wyspecjalizowanych systemach, między innymi w hurtowniach danych.

Na potrzeby niniejszej rozprawy przyjmuje się, ze dane pochodzące z sieci stacji paliw nazywane będą zwyczajowo *danymi paliwowymi*. Podzielić je można na kilka kategorii, przy czym podział ten podyktowany jest zarówno stopniem przetworzenia danych, jak i aspektem, z którym są związane. Można zatem wyróżnić:

1) dane pierwotne – dane pozyskane bezpośrednio z czujników;
2) dane wtórne – dane powstałe z danych pierwotnych, na skutek zastosowania pewnych przekształceń:
    a) dane związane z objętością paliwa,
    b) dane związane ze rozszerzalnością cieplną,
3) dane zagregowane – dane powstałe z danych pierwotnych bądź wtórnych na skutek zastosowania operacji agregacji:
    a) dane związane z rekoncyliacją zbiornika,
    b) dane związane z rekoncyliacją dostawy.

Na dane paliwowe składają się pomiary zarejestrowane w określonych momentach czasu oraz wartości zagregowane, reprezentujące przedział czasu

Tabela 3.1: Objaśnienie symboli używanych w notacji matematycznej

Symbol	Oznaczenie
X	pomiar
$X(t)$	pomiar zarejestrowany w chwili t
$\bar{X}$	pomiar skumulowany w określonym przedziale czasu
$\hat{X}$	szacowany pomiar
$\tilde{X}$	pomiar w temperaturze referencyjnej
$\hat{\tilde{X}}$	szacowany pomiar w temperaturze referencyjnej
$\dot{X}$	pomiar „surowy", wymagający dalszego przetworzenia

Tabela 3.2: Dane paliwowe pierwotne

Źródło	Nazwa	Symbol
Zbiornik	Czas pomiaru	t_p
	Wysokość paliwa	H_p
	Temperatura paliwa	T_p
	Wysokość wody	H_w
	Temperatura wody	T_w
Dystrybutor	Czas pomiaru	t_s
	Całkowita objętość paliwa	$\bar{V}_s$
	Chwilowa temperatura paliwa	T_s
Dostawa	Czas początku dostawy	t_0
	Czas końca dostawy	t_1
	Deklarowana objętość paliwa	V_d
	Deklarowana objętość paliwa w temp. ref.	$\tilde{V}_d$

o niezerowej długości. W celu uproszczenia notacji proponuje się następującą konwencję: jawne wskazanie czasu zarejestrowania pomiaru występuje wyłącznie w przypadkach, w których w danym równaniu bądź wyrażeniu istnieje konieczność użycia tych samych typów pomiarów, ale pochodzących z różnych momentów lub przedziałów czasu; w pozostałych przypadkach zapis wskazujący na czas pomiaru pomija się. W dalszym toku rozprawy używane są w notacji matematycznej różne symbole, które zostały przedstawione w tabeli 3.1.

3.2 Dane pierwotne

Dane pierwotne w obrębie pojedynczej stacji paliw pozyskiwane są w zbiornikach paliwa oraz dystrybutorach. Ponadto zbiór ten uzupełniany jest o wprowadzane ręcznie dane o dostawach paliwa. Tabela 3.2 zawiera wykaz danych pierwotnych, wraz ze wskazaniem ich źródeł oraz podaniem symboli używanych w dalszych częściach rozprawy.

Podstawową wielkością rejestrowaną w zbiorniku paliwa jest wysokość paliwa. Zdarza się czasem, że w zbiorniku gromadzi się również woda, co jest zjawiskiem niepożądanym. Przez wzgląd na powyższe, oprócz pomiaru wysokości paliwa, przeprowadzany jest również pomiar wysokości wody. Wielkości te mierzone są przy wykorzystaniu czujnika pływakowego wyposażonego w dwa pływaki, po jednym dla każdej z substancji (woda ma wyższą gęstość niż paliwo). Razem z pomiarami wysokości mierzona jest także temperatura obu cieczy. Wykonywane jest to za pośrednictwem kilku czujników temperatury, przy czym zwracana przez urządzenie pomiarowe wartość jest średnią ze wskazań czujników zanurzonych w danej cieczy [20, 21].

Urządzenia pomiarowe zainstalowane w dystrybutorach mierzy całkowitą objętość paliwa, które zostało przepompowane przez dany pistolet, przez cały okres czasu pracy dystrybutora. W każdym dystrybutorze znajduje się kilka pistoletów – ich liczba przeważnie odpowiada liczbie typów paliwa sprzedawanego na stacji. W związku z tym, pojedynczy dystrybutor jest źródłem kilku niezależnych pomiarów objętości, po jednym dla każdego pistoletu. Ponadto, nowsze typy dystrybutorów dysponują modułem mierzącym temperaturę sprzedawanego paliwa. W tabeli 3.2 wielkość ta została oznaczona jako chwilowa temperatura paliwa T_s.

Ze względu na swoją odrębność od infrastruktury stacji paliw, proces dostawy nie podlega automatycznej rejestracji. Objętość dostarczanego paliwa jest wielkością deklarowaną przez dostawcę i jest wprowadzana ręcznie przez pracowników stacji. W odróżnieniu też od poprzednio opisanych pomiarów, deklarowana objętość nie jest mierzona w sposób ciągły, a wyłącznie podawana podczas dostawy paliwa.

W zależności od regulacji prawnych, deklarowana przez dostawcę wartość objętości może być mierzona bez, bądź z uwzględnieniem zjawiska rozszerzalności cieplnej. W tym drugim przypadku podawana jest wartość objętości paliwa w temperaturze referencyjnej. Dokładniejszy opis tego zagadnienia znajduje się w dalszej części rozdziału, poświęconej zjawisku rozszerzalności cieplnej.

3.3 Dane wtórne

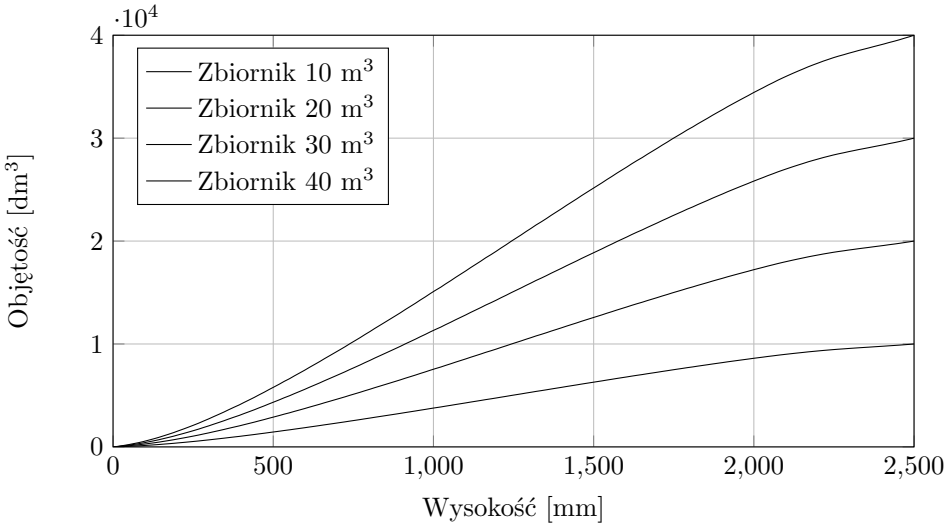
Zbiór danych pierwotnych jest stosunkowo niewielki. Na jego podstawie wyznaczany jest zbiór danych wtórnych, które dostarczają więcej użytecznych informacji. Podobnie jak dane pierwotne, z których powstają, związane są one z tymi samymi elementami infrastruktury stacji paliw oraz procesem dostawy.

3.3.1 Dane wtórne związane z objętością

Pomiary objętości paliwa i wody nie znajdują się w zbiorze danych pierwotnych, gdyż są wyliczane na podstawie tych pierwszych oraz dodatkowych

Tabela 3.3: Dane paliwowe wtórne związane z objętością

Źródło	Nazwa	Symbol
Zbiornik	Objętość paliwa	V_p
	Objętość wody	V_w
Dystrybutor	Nieskalibrowana obj. paliwa	$\dot{V}_s$
	Skalibrowana obj. paliwa	V_s



Rysunek 3.4: Przykładowe krzywe kalibracyjne czterech zbiorników paliwa

danych. Tabela 3.3 przedstawia podzbiór danych wtórnych związanych z objętością.

Wartości objętości cieczy znajdujących się w zbiorniku paliwa są wyznaczane na podstawie pomiarów wysokości tych cieczy oraz przekształcenia wysokość-objętość. Przekształcenie to zwane jest również zwyczajowo *krzywą kalibracyjną* i jest charakterystyczne dla danego zbiornika, zależy bowiem od jego kształtu oraz wymiarów. Na rysunku 3.4 przedstawiono przykładowe krzywe kalibracyjne (wygenerowane w sposób syntetyczny, podobnie jak w pracy własnej [28]$^{\mathrm{W}}$) dla czterech zbiorników paliwa, z których każdy ma wysokość 2500 mm, a ich objętości to odpowiednio 10, 20, 30 i 40 m^3.

Wartości objętości sprzedaży wyznaczane są na podstawie pomiarów całkowitej objętości sprzedanego paliwa dokonanych w dwóch chwilach czasu. Wybór tych momentów może być podyktowany czasem trwania pojedynczej operacji sprzedaży paliwa, może również wynikać z przyjętego okna czasowego wykorzystywanego w dalszej analizie tych danych. Poniższe równanie przedstawia sposób wyznaczania objętości sprzedanego paliwa w przedziale czasu $\langle t_a, t_b \rangle$:

$$\dot{V}_s = \bar{V}_s(t_a) - \bar{V}_s(t_b). \tag{3.1}$$

Urządzenia pomiarowe zainstalowane w dystrybutorach paliwa są obarczone pewnym błędem pomiarowych i w związku z tym są okresowo sprawdzane i kalibrowane. Proces ten polega na wyznaczeniu tak zwanego współczynnika kalibracji α, który określa iloraz rzeczywistej objętości paliwa do tej zmierzonej i wskazywanej przez dany czujnik. Skalibrowana objętość sprzedanego paliwa może zostać wyrażona wzorem:

$$V_s = \alpha \dot{V}_s, \tag{3.2}$$

przy czym wielkość ta jest używana w dalszych obliczeniach zamiast wartości nieskalibrowanej.

3.3.2 Dane wtórne związane z rozszerzalnością cieplną

Paliwo, jak każda inna ciecz, podlega prawom fizyki, w tym i zjawisku rozszerzalności cieplnej. Ma to szczególne znaczenie ze względu na wiarygodność pomiarów jego objętości. Gdy temperatura paliwa rośnie, zwiększa się również jego objętość i analogicznie – objętość paliwa maleje wraz ze spadkiem jego temperatury. Niezmienna za to w obu przypadkach pozostaje masa paliwa. Chociaż nie jest możliwe ważenie paliwa znajdującego się w zbiorniku, to wielkość tę można wyznaczyć na podstawie bieżącej objętości i gęstości paliwa, zgodnie z zależnością: $m = \rho V$. Należy tutaj zwrócić uwagę na fakt, iż gęstość cieczy nie jest stała, lecz zależna od jej temperatury. Powyższą zależność opisuje równanie:

$$\rho(T) = \frac{\rho_0}{1 + \beta(T - T_0)}, \tag{3.3}$$

w którym ρ_0 oznacza gęstość paliwa w temperaturze referencyjnej T_0, a β – współczynnik rozszerzalności cieplnej.

Operowanie na masach cieczy, choć w pełni niezależne od jego temperatury, nie jest popularną praktyką w przemyśle petrochemicznym. Zamiast tego wykorzystuje się obliczenia bazujące na objętościach. W tym celu wprowadza się syntetyczną wartość objętości cieczy w temperaturze referencyjnej $\tilde{V}$. Jest ona równa objętości identycznej masy tej samej cieczy, lecz znajdującej się w temperaturze referencyjnej T_0. W literaturze znaleźć można również terminy *objętość brutto* dla określenia zwykłej objętości cieczy oraz *objętość netto* dla objętości cieczy w temperaturze referencyjnej [38]$^{\mathrm{W}}$. Poniższe równanie przedstawia sposób wyznaczania tej drugiej wartości, przy czym β oznacza współczynnik rozszerzalności temperaturowej cieczy:

$$\tilde{V} = V(1 + \beta(T - T_0)). \tag{3.4}$$

Podzbiór danych wtórnych związanych z rozszerzalnością cieplną przedstawia tabela 3.4. W tabeli tej występuje również szacowana temperatura

Tabela 3.4: Dane paliwowe wtórne związane z rozszerzalnością cieplną

Źródło	Nazwa	Symbol
Zbiornik	Objętość paliwa w temp. ref.	$\tilde{V}_p$
	Objętość wody w temp. ref	$\tilde{V}_w$
Dystrybutor	Szacowana temp. paliwa	$\hat{T}_s$
	Objętość paliwa w temp. ref.	$\tilde{V}_s$

paliwa $\hat{T}_s$, która jest obliczana dla konkretnego przedziału czasu na podstawie pomiarów chwilowej temperatury sprzedanego paliwa T_s, co przedstawia następujące równanie – w tym przypadku dla przedziału $\langle t_a, t_b \rangle$:

$$\hat{T}_s = \frac{1}{t_b - t_a} \int_{t_a}^{t_b} T_s(t)dt \approx \frac{T_s(t_a) + T_s(t_b)}{2}. \tag{3.5}$$

Powyższy sposób wykorzystuje wzór na wartość średnią funkcji w zadanym przedziale. W celu uproszczenia obliczeń, można założyć, że zmiana chwilowej temperatury sprzedanego paliwa w rzeczonym przedziale czasu jest liniowa – wówczas prawa strona równania 3.5 redukuje się do średniej arytmetycznej.

Wartość chwilowej temperatury sprzedanego paliwa może zostać uzyskana bezpośrednio z dystrybutorów wyposażonych w pomiar temperatury. Gdy nie ma takowej możliwości, korzysta się z pomiarów temperatury paliwa w zbiorniku, zarejestrowanych w tym samym przedziale czasu, co wyznaczana temperatura sprzedanego paliwa. Wówczas przyjmuje się, że $T_s(t) = T_p(t)$. Działanie takie nie uwzględnia zmiany temperatury paliwa zachodzącego w orurowaniu znajdującym się pomiędzy zbiornikiem a dystrybutorem. Wyznaczenie tej zmiany jest zadaniem bardzo trudnym, ze względu na mnogość parametrów i czynników fizycznych oraz brak znajomości wartości sporej części z nich. Pewne próby zbudowania modelu matematycznego zachowania się paliwa w orurowaniu zostały podjęte w literaturze [22]. Co więcej, temperatura paliwa w zbiorniku nie zawsze jest jednakowa w całej jego objętości. Jest tak ze względu na niezerowy czas mieszania się w nim cieczy podczas dostawy oraz fakt, iż paliwo pobierane jest ze zbiornika w pobliżu jego dna, podczas gdy nowe paliwo dolewane jest od góry.

3.4 Dane zagregowane

Przedstawione w poprzednich częściach dane w większości opisują zjawiska zarejestrowane w wybranych momentach czasu. Wyjątkiem są tutaj pomiary związane z dostawą paliwa oraz jego sprzedażą. W ich przypadku opisywane zjawiska trwają przez określony czas – jest to odpowiednio proces dostawy bądź sprzedaży paliwa.

W celu przeprowadzenia jakiejkolwiek analizy polegającej na zestawieniu ze sobą pomiarów pochodzących z różnych źródeł o odmiennej charakterystyce czasowej, konieczne jest unormowanie tych wartości w czasie. Takie działanie obywa się poprzez agregację poszczególnych pomiarów w wymiarze czasu, w oknach czasowych o określonej szerokości. Okno czasowe może zostać zdefiniowane jako przedział czasu o szerokości λ. Celem agregacji wykorzystującej okna czasowe jest wyznaczenie pojedynczych wartości na podstawie wszystkich tych pomiarów, które zostały zarejestrowane w czasie należącym do danego przedziału. Szerokość okna λ nie musi być stała dla każdego okna, może ona być uzależniona od różnych zjawisk, przykładowo od trwającej dostawy paliwa.

Podstawową operacją wykonywaną na danych paliwowych jest *rekoncyliacja*, która oznacza uzgadnianie, bilansowanie. Wyróżnić można rekoncyliację zbiornika paliwa oraz rekoncyliację dostawy. W obu przypadkach porównywane są ze sobą wartości pomiarów z przeciwstawnych źródeł, a wynik takiej różnicy nosi nazwę *błędu rekoncyliacji*. W idealnym przypadku, wartość tego błędu jest równa zeru. Wszelkie odchylenia od tej wartości świadczą o niedoborach lub naddatkach paliwa.

3.4.1 Dane zagregowane związane z rekoncyliacją zbiornika

Rekoncyliacja zbiornika jest operacją polegającą na przeciwstawieniu sobie czynników modyfikujących objętość paliwa w zbiorniku. Do tych czynników można zaliczyć sprzedaż paliwa oraz jego dostawę. Przeprowadzanie rekoncyliacji zbiornika ma na celu wykrycie wszelkich niezgodności związanych z utratą paliwa, takich jak choćby wyciek lub kradzież.

Historycznie rekoncyliacja zbiornika wykonywana była w sposób manualny przy pomocy dokumentacji prowadzonej na stacji [126]. Polegało to na codziennym pomiarze objętości paliwa w zbiorniku przed rozpoczęciem oraz po zakończeniu pracy stacji. Oprócz tego zapisywano objętość sprzedanego paliwa odczytaną z dystrybutorów oraz deklarowaną przez dostawcę objętość dostarczonego paliwa. Następnie na podstawie tych wartości obliczano szacowaną objętość paliwa w zbiorniku po zakończeniu pracy stacji. Ilustruje to równanie:

$$\hat{V}_p(t_b) = V_p(t_a) - V_s + V_d, \tag{3.6}$$

w którym szacowana objętość paliwa $\hat{V}_p$ wyznaczana jest w przedziale czasu pracy stacji: $\langle t_a, t_b \rangle$. Następnie, uzyskana w ten sposób wielkość porównywana była z rzeczywistym pomiarem. Wynik tego porównania stanowił błąd rekoncyliacji zbiornika E_z:

$$E_z = V_p(t_b) - \hat{V}_p(t_b). \tag{3.7}$$

Obecnie proces rekoncyliacji zbiornika przeprowadzany jest w sposób automatyczny i ciągły, w krótkich odstępach czasowych. Sytuacja taka spowodowana jest dwudziestoczterogodzinnym trybem pracy stacji paliwa oraz

Tabela 3.5: Zagregowane dane paliwowe związane z rekoncyliacją zbiornika

Źródło	Nazwa	Symbol
Zbiornik	Zmiana objętości paliwa	ΔV_p
	Zmiana obj. paliwa w temp. ref.	$\Delta \tilde{V}_p$
Dystrybutor	Sumaryczna objętość paliwa	ΣV_s
	Sumaryczna obj. paliwa w temp. ref.	$\Sigma \tilde{V}_s$
Dostawa	Cząstkowa objętość paliwa	ΠV_d
	Cząstkowa obj. paliwa w temp. ref.	$\Pi \tilde{V}_d$
Rekoncyliacja	Czas początku przedziału	t_i
	Czas początku przedziału	$t_{i+\lambda}$
	Błąd rekoncyliacji zbiornika	E_z
	Błąd rekoncyliacji zb. w temp. ref.	$\tilde{E}_z$

elektronicznym pozyskiwaniem pomiarów. Szerokość przedziałów czasowych jest w zasadzie dowolna, ograniczona jedynie czasem trwania mierzonych zjawisk fizycznych.

Wykonywanie wszelkich obliczeń w trybie on-line, czyli podczas pracującej stacji paliw, wymaga operowania na danych zagregowanych w wymiarze czasu. W związku z tym, przed wyznaczeniem błędu rekoncyliacji konieczne jest przeprowadzenie agregację rzeczonych pomiarów. Zagregowane dane paliwowe związane z rekoncyliacją zbiornika przedstawia tabela 3.5.

W przypadku zbiornika paliwa, istotna jest zmiana objętości paliwa w nim przechowywanego, w danym przedziale czasu. Obliczenie takiej różnicy może odbywać się zarówno dla zwykłych pomiarów objętości, jak i dla ich wersji w temperaturze referencyjnej. Poniższe równanie przedstawia sposób wyznaczania zmiany objętości paliwa w zbiorniku w oknie czasowym $\langle t_i, t_{i+\lambda} \rangle$ dla pomiarów objętości brutto:

$$\Delta V_p = V_p(t_{i+\lambda}) - V_p(t_i). \tag{3.8}$$

Ze względu na fakt, iż pojedynczy zbiornik zasila w paliwo wiele dystrybutorów, konieczne jest zsumowanie odczytanych z nich wartości objętości sprzedanego paliwa. Na oznaczenie pomiaru uzyskanego z i-tego dystrybutora wprowadzone zostaje oznaczenie: V_s^i. Równanie:

$$\Sigma V_s = \sum_{i=1}^{n} V_s^i \tag{3.9}$$

ilustruje sposób obliczania sumarycznej objętości sprzedanego paliwa dla n podłączonych do danego zbiornika dystrybutorów. W analogiczny sposób przeprowadzane jest sumowanie objętości w temperaturze referencyjnej. Zakłada się przy tym, że przedział czasowy potrzebny do obliczenia wartości V_s^i (zgodnie ze wzorami 3.1 i 3.2) jest równy oknu czasowemu rekoncyliacji $\langle t_i, t_{i+\lambda} \rangle$.

Zasadniczo przedziały czasowe, w których przeprowadzana jest rekoncyliacja mają stałą szerokość. Dopuszcza się jedna odejście od tej zasady w przypadku, gdy występuje dostawa paliwa. Wówczas dany przedział czasowy jest tak dobierany, aby w całości ją obejmował. Dzięki temu wprowadzona w tabeli 3.5 wielkość cząstkowej objętości dostarczonego paliwa jest równa deklarowanej objętości dostawy: $\Pi V_d = V_d$. Identyczna sytuacja ma miejsce w przypadku operowania na objętościach w temperaturze referencyjnej.

W sytuacji, w której przedziały czasowe są zawsze stałe, konieczne jest obliczenie cząstkowej objętości dostarczonego paliwa. Wyznaczona może być ona na podstawie deklarowanej objętości dostawy, proporcjonalnie do czasu części wspólnej danego przedziału czasu a czasu trwania dostawy paliwa. Przy założeniu, że dostawa trwa przez okres czasu $\langle t_0, t_1 \rangle$, a przedział czasu wykorzystywany w operacji rekoncyliacji jest równy: $\langle t_i, t_{i+\lambda} \rangle$, wyznaczenie rzeczonej proporcji η może zostać przeprowadzone w następujący sposób:

$$\eta = \begin{cases} 0 & \text{jeśli} \quad t_0 \notin \langle t_i, t_{i+\lambda} \rangle \wedge t_1 \notin \langle t_i, t_{i+\lambda} \rangle \\ \frac{t_1 - t_i}{t_1 - t_0} & \text{jeśli} \quad t_0 \notin \langle t_i, t_{i+\lambda} \rangle \wedge t_1 \in \langle t_i, t_{i+\lambda} \rangle \\ 1 & \text{jeśli} \quad t_0 \in \langle t_i, t_{i+\lambda} \rangle \wedge t_1 \in \langle t_i, t_{i+\lambda} \rangle \\ \frac{t_{i+\lambda} - t_0}{t_1 - t_0} & \text{jeśli} \quad t_0 \in \langle t_i, t_{i+\lambda} \rangle \wedge t_1 \notin \langle t_i, t_{i+\lambda} \rangle \end{cases}. \tag{3.10}$$

W rezultacie cząstkowa objętość dostarczonego paliwa ΠV_d może zostać wyrażona jako:

$$\Pi V_d = \eta V_d. \tag{3.11}$$

Dysponując wielkościami wyznaczonymi w równaniach 3.8, 3.9 i 3.11, możliwe jest wyznaczenie błędu rekoncyliacji zbiornika w dowolnym przedziale czasu. Ilustruje to poniższe równanie:

$$E_z = \Delta V_p + \Sigma V_s - \Pi V_d. \tag{3.12}$$

Kolejne wartości błędu rekoncyliacji zbiornika określają jego stan. W idealnym przypadku są one zawsze równe zero. Oznacz to, że zmiana objętości w zbiorniku odpowiada dokładnie objętości sprzedanego paliwa, z uwzględnieniem ewentualnej dostawy. Ujemne wartości błędu świadczyć mogą między innymi o wystąpieniu wycieku paliwa bądź jego kradzieży. Dodatnie wartości błędu sugerują dolanie pewnej ilości paliwa, aczkolwiek tego typu sytuacje są wielce nieprawdopodobne. Najczęściej jednak, niezerowe wartości błędu rekoncyliacji zbiornika wynikają z niepoprawnie skalibrowanych urządzeń pomiarowych oraz innych czynników wpływających na wartości mierzonych parametrów.

3.4.2 Dane zagregowane związane z rekoncyliacją dostawy

Rekoncyliacja dostawy jest operacją polegającą na porównaniu ze sobą dwóch wartości: deklarowanej przez producenta objętości dostarczonego pa-

liwa oraz wykrytego na stacji przyrostu jego objętości, zwanego szacowaną objętością dostawy $\hat{V}_d$. Celem takiej operacji jest wykrycie wszelkich niezgodności mających miejsce podczas dostawy, a przede wszystkim kradzieży paliwa. Poniższe równanie przedstawia metodę obliczania wartości błędu rekoncyliacji dostawy:

$$E_d = \hat{V}_d - V_d. \tag{3.13}$$

W idealnym przypadku, wartości błędu rekoncyliacji dostawy są równe zero, gdyż objętość paliwa deklarowana przez dostawcę dokładnie odpowiada przyrostowi Ujemne wartości błędu świadczyć mogą o utracie pewnej ilości paliwa podczas dostawy. Może to być jeden z objawów jego kradzieży odbywającej się w trasie. Dodatnie wartości, podobnie jak w przypadku rekoncyliacji zbiornika, najczęściej spowodowane są niepoprawną kalibracją urządzeń pomiarowych.

Podczas dostawy paliwa następuje mieszanie się dużych objętości cieczy o różnych temperaturach. W konsekwencji zmiany objętości paliwa wynikłe ze zjawiska rozszerzalności cieplnej są zauważalne. W przypadku, w którym dostawca deklaruje objętość dostawy netto, czyli w temperaturze referencyjnej, problem ten przestaje być istotny. Wyznaczenie szacowanej objętości dostawy w temperaturze referencyjnej polega na obliczeniu zmiany objętości paliwa w zbiorniku w okresie przed i po dostawie (obie wielkości w temperaturze referencyjnej) oraz uwzględnieniu sprzedanego paliwa podczas trwania dostawy (również w temperaturze referencyjnej). Ilustruje to równanie:

$$\hat{\tilde{V}}_d = \Delta\tilde{V}_p + \Sigma\tilde{V}_s. \tag{3.14}$$

W sytuacji, w której dostawca podaje jedynie deklarowaną objętość dostawy brutto, konieczne jest oszacowanie objętości dostarczonego paliwa, z uwzględnieniem wpływu zmiany temperatury. W tym celu konieczne jest obliczenie dodatkowych danych zagregowanych, które zostały przedstawione w tabeli 3.6. Tabela ta zawiera również błąd rekoncyliacji zbiornika w temperaturze referencyjnej $\tilde{E}_d$ – odnosi się to do opisanej powyżej sytuacji, w której dostępny jest pomiar deklarowanej objętości dostarczonego paliwa netto.

Rozwiązaniem problemu braku wartości deklarowanej objętości dostarczonego paliwa w temperaturze referencyjnej może być wykorzystanie prawa zachowania masy. Na jego podstawie można stwierdzić, że masa dostarczonego paliwa jest równa różnicy masy paliwa w zbiorniku przed i po dostawie powiększonej o masę sprzedanego paliwa podczas trwania dostawy. Wyraża się to następującą zależnością:

$$\hat{m}_d = \Delta m_p + \Sigma m_s. \tag{3.15}$$

W celu znalezienia szacowanej masy dostarczonego paliwa $\hat{m}_d$, należy zatem wyznaczyć oba składniki sumy przedstawionej w równaniu 3.15. Masa

Tabela 3.6: Dane paliwowe wtórne związane z rozszerzalnością cieplną

Źródło	Nazwa	Symbol
Zbiornik	Zmiana masy paliwa	Δm_p
Dystrybutor	Sumaryczna masa paliwa	Σm_s
Dostawa	Szacowana masa paliwa	$\hat{m}_d$
	Szacowana temperatura paliwa	$\hat{T}_d$
	Szacowana objętość paliwa	$\hat{V}_d$
	Szacowana objętość paliwa w tem. ref.	$\hat{\tilde{V}}_d$
Rekoncyliacja	Błąd rekoncyliacji dostawy	E_d
	Błąd rekoncyliacji zb. w temp. ref.	$\tilde{E}_d$

cieczy może zostać wyznaczona na podstawie jej objętości, gęstości i temperatury, zgodnie z tym co zostało przedstawione w równaniu 3.3.

Zmiana masy paliwa w zbiorniku, może zostać wyznaczona na podstawie zmiany objętości paliwa w temperaturze referencyjnej $\tilde{V}_p$ oraz gęstości paliwa w tejże temperaturze ρ_0:

$$\Delta m_p = \rho_0 \cdot \Delta \tilde{V}_p. \tag{3.16}$$

Znalezienie sumarycznej masy sprzedanego paliwa polega na zsumowaniu mas sprzedanego paliwa przez wszystkie dystrybutory podłączone do danego zbiornika, w całym okresie dostawy. Podobnie jak powyżej, wielkość ta może zostać wyznaczona na podstawie sumarycznej objętości sprzedanego paliwa w temperaturze referencyjnej $\Sigma\tilde{V}_s$ oraz gęstości paliwa w tejże temperaturze ρ_0:

$$\Sigma m_s = \rho_0 \cdot \Sigma \tilde{V}_s. \tag{3.17}$$

Znając oba składniki sumy z równania 3.15 możliwe jest obliczenie szacowanej masy dostawy $\hat{m}_d$. Celem obliczeń jest jednak wyznaczenie szacowanej objętości dostawy $\hat{V}_d$. Dysponując wartością masy dostarczonego paliwa możliwe jest, przy wykorzystaniu szacowanej jego temperatury $\hat{T}_d$ oraz gęstości ρ_0 w temperaturze referencyjnej T_0, obliczenie poszukiwanej objętości:

$$\hat{V}_d = \frac{\hat{m}_d(1 + \beta(\hat{T}_d - T_0))}{\rho_0}. \tag{3.18}$$

Problemem jest znalezienie szacowanej temperatury dostarczonego paliwa $\hat{T}_d$. Wielkość ta nie jest w żaden sposób mierzona w cysternie przewożącej paliwo. Wpływa jednak na temperaturę paliwa w zbiorniku po dostawie (trwającej w przedziale czasu $\langle t_0, t_1 \rangle$), która zmienia się zgodnie z równaniem:

$$T_p(t_1) = \frac{m_p(t_0) \cdot T_p(t_0) + \hat{m}_d \cdot \hat{T}_d}{m_p(t_0) + \hat{m}_d}. \tag{3.19}$$

Należy tutaj zwrócić uwagę na przyjęte uproszczenie, polegające na tym, że zaniedbywana jest zmiana objętości paliwa w zbiorniku spowodowana sprzedażą paliwa podczas trwania dostawy. Działanie takie jest podyktowane małą objętością sprzedawanego paliwa w stosunku do objętości dostarczanego paliwa, jak również fakt, iż sprzedaż paliwa może nastąpić w dowolnym momencie dostawy i wpływ pomniejszania przez nią objętości częściowo zmieszanych płynów jest trudny do oszacowania. Z równania 3.19 można wyznaczyć szacowaną temperaturę dostarczonego paliwa $\hat{T}_d$:

$$\hat{T}_d = \frac{(m_p(t_1) + \Sigma m_s) \cdot T_p(t_1) - m_p(t_0) \cdot T_p(t_0)}{\hat{m}_d}. \tag{3.20}$$

W równaniu 3.20 wykorzystano równość $m_p(t_0) + \hat{m}_d = m_p(t_1) + \Sigma m_s$, polegającą na tym, że masa paliwa w zbiorniku przed dostawą powiększona o szacowaną masę dostarczonego paliwa jest równa masie paliwa w zbiorniku po dostawie, powiększoną o sumaryczną masę sprzedanego paliwa podczas dostawy. W konsekwencji, dysponując szacowaną temperaturą dostarczonego paliwa $\hat{T}_d$ (wyznaczoną w równaniu 3.20) możliwe jest wyznaczenie szacowanej objętości dostawy $\hat{V}_d$, zgodnie z równaniem 3.18. Ta wartość jest wystarczająca do obliczenia wartości błędu rekoncyliacji dostawy E_d, zgodnie z równaniem 3.13. Należy jednakże pamiętać, że bez znajomości temperatury dostarczanego paliwa podczas jego nalewania do cysterny w terminalu, uzyskana w ten sposób wartość błędu będzie zawsze zawierała składową wynikłą ze zmiany temperatury paliwa podczas jego podróży. W związku z tym, optymalną strategią dostawcy jest deklarowanie objętości dostarczonego paliwa w temperaturze referencyjnej.

3.5 Wielowymiarowy model danych paliwowych

Dane paliwowe mają charakter pomiarów, które są rejestrowane w określonych momentach czasu. W konsekwencji czas jest podstawowym aspektem ich analizy. Ponadto, można wyróżnić jeszcze kilka pomocniczych aspektów, związanych przede wszystkim z pochodzeniem tychże danych, takich jak identyfikatory poszczególnych czujników. W przypadku przetwarzania danych pochodzących z całej sieci stacji paliw, a więc z wielu obiektów rozproszonych na sporym obszarze geograficznym, wyróżnić można również aspekt lokalizacji danej stacji oraz aspekt charakteryzujący dostawcę paliwa. Ze względu na przedstawione powyżej argumenty, dane paliwowe można traktować jako dane wielowymiarowe i poddać takiej też analizie. W tym celu należy wyszczególnić fakty związane ze zjawiskami będącymi przedmiotem rzeczonej analizy oraz wymiary, określające jej aspekty – tworzą one razem model konceptualny.

Pierwszym krokiem do zbudowania modelu konceptualnego danych paliwowych jest identyfikacja procesów zachodzących w sieci stacji paliw. Zwią-

zane są one z poszczególnymi etapami toru przepływu paliwa. Można zatem wyróżnić:

a) składowanie paliwa na stacji paliw – proces związany z przechowywaniem paliwa w zbiornikach;

b) sprzedaż paliwa na stacji paliw – proces związany ze sprzedażą paliwa za pośrednictwem dystrybutorów;

c) dostawę paliwa na stację paliw – proces związany z dostarczaniem paliwa na stację;

d) rekoncyliację zbiornika paliwa – proces związany z bilansem ubytków i naddatków paliwa podczas jego przechowywania;

e) rekoncyliację dostawy paliwa – proces związany z bilansem ubytków i naddatków paliwa podczas jego dostawy.

W powyższej liście w sposób odrębny przedstawiono rekoncyliację zbiornika oraz dostawy paliwa w stosunku do jego składowania oraz dostarczania na stację. Zostało to podyktowane chęcią wyraźnego oddzielenia od siebie pomiarów rejestrowanych na stacji paliw a wyników operacji bilansowania. Jako uzasadnienie należy wskazać odmienny cel analizy tychże danych. W przypadku zwykłych pomiarów może on wiązać się z raportowaniem oraz wizualizacją stanu danej stacji bądź całej sieci, zaś w przypadku obu typów rekoncyliacji – ukierunkowany jest na wykrywanie zjawisk niepożądanych.

Kolejnym krokiem na drodze stworzenia modelu konceptualnego danych paliwowych jest zdefiniowanie aspektów, pod którymi analizowane będą rzeczone procesy. Wyróżnić można:

a) czas – podstawowy aspekt, pod którym analizowane są wszelkiego rodzaju dane pomiarowe;

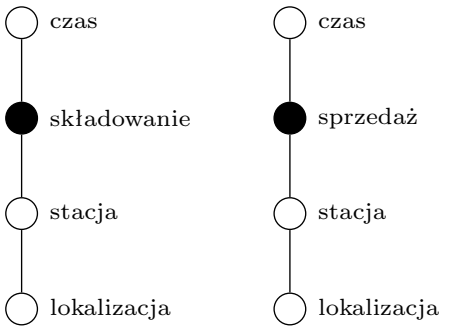
b) lokalizacja – aspekt związany z rozproszeniem geograficznym źródeł danych;

c) stacja – aspekt rozumiany jako całość infrastruktury zainstalowanej na stacji paliw, określonej przez identyfikatory źródeł danych oraz ich parametry techniczne;

d) dostawca – aspekt opisujący dostawcę paliw płynnych.

Podsumowując, można wyodrębnić pięć podstawowych procesów odbywających się w sieci stacji paliw, będących zarazem przedmiotem analizy, oraz cztery główne aspekty, pod którymi może być ona przeprowadzana. W konsekwencji tworzony model konceptualny danych paliwowych składa się z pięciu faktów oraz czterech wymiarów.

Pierwszym z wymienionych faktów jest składowanie paliwa. Proces ten odbywa się w zbiorniku i wiąże z rejestracją pomiarów wysokości, temperatury i objętości znajdujących się w nim cieczy: paliwa i ewentualnie wody.



(a) składowanie (b) sprzedaż

Rysunek 3.5: Modele konceptualne faktów składowania oraz sprzedaży

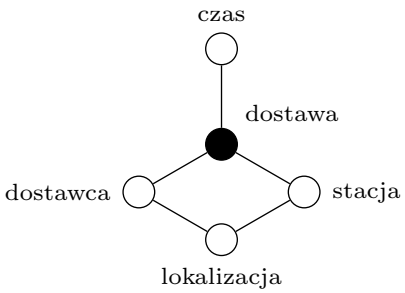
Analiza tych wielkości obywać się może pod kątem zarówno czasu ich zarejestrowania, jak również z uwzględnieniem podziału na konkretne elementy infrastruktury stacji paliw (zbiorniki). W ten sposób nakreślona analiza ograniczona jest do pojedynczej stacji. Rozszerzając ją na całą sieć, konieczne jest wzięcie pod uwagę aspektu lokalizacji. Należy tutaj zauważyć, iż tym samym rozszerza ona aspekt stacji, stojąc niejako wyżej w hierarchii.

Podobnie rzecz ma się w przypadku sprzedaży paliwa. Fakt ten może również być analizowany zarówno pod kątem czasu oraz konkretnych elementów infrastruktury stacji (pistolety, dystrybutory, zbiorniki). Ponadto, w ujęciu całej sieci stacji paliw uwzględnić należy wymiar lokalizacji.

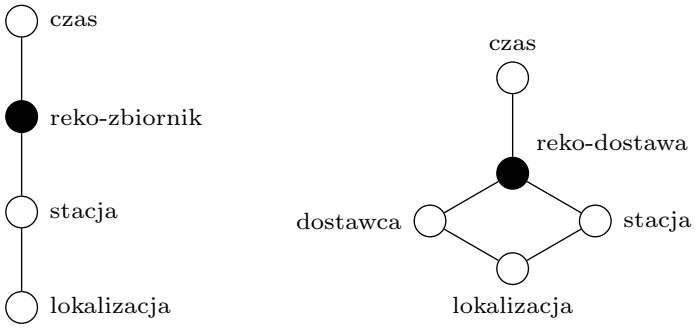
Ze względu na podobieństwo do siebie tych faktów pod kątem wymiarów ich analizy, przedstawiono je na wspólnym rysunku 3.5. Jego lewa część (rysunek 3.5a) przedstawia model konceptualny faktu składowania paliwa, a jego prawa część (rysunek 3.5b) – model konceptualny faktu jego sprzedaży. Czarnymi węzłami oznaczono fakty, a białymi wymiary.

Nieco odmienna sytuacja kształtuje się w przypadku faktu dostawy. W proces dostarczania paliwa jest zaangażowany jego dostawca, a więc podmiot odrębny od stacji paliw, na której odbywa się dostawa. W konsekwencji, oprócz wymiarów czasu i stacji, do zbioru wymiarów faktu dostawy należy jeszcze zaliczyć wymiar dostawcy. Co więcej, zarówno dana stacja, jak i dostawca paliwa mogą zostać określeni pod kątem ich położenia geograficznego. W konsekwencji, wymiar lokalizacji rozszerza zarówno wymiar stacji, jak i dostawcy. Model konceptualny faktu dostawy został przedstawiony na rysunku 3.6.

Pozostałe dwa fakty: rekoncyliacji zbiornika oraz dostawy, pod względem swojej wymiarowości zbliżone są do opisanych już faktów składowania paliwa oraz dostawy. Jest tak, gdyż wiążą się one z tym samym elementem bądź zjawiskiem, ale analizowanym w odmienny sposób. Rysunek 3.7 przedstawia modele konceptualne dla tych wymiarów. Jego lewa część (rysunek 3.7a) odnosi się do faktu rekoncyliacji zbiornika, a jego prawa część (rysunek 3.7b)



Rysunek 3.6: Model konceptualny faktu dostawy

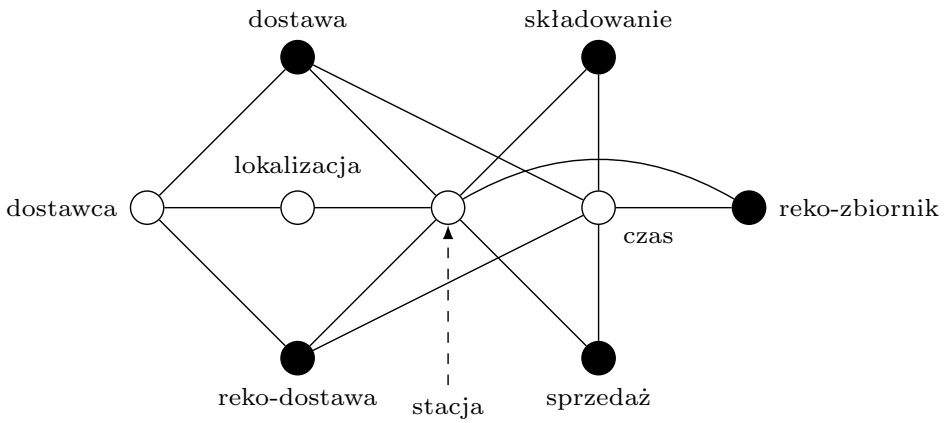


(a) rek. zbiornika (b) rek. dostawy

Rysunek 3.7: Modele konceptualne faktów rekoncyliacji: zbiornika i dostawy

– do faktu rekoncyliacji dostawy.

W celu zbudowania pełnego modelu konceptualnego danych paliwowych należy zestawić ze sobą poszczególne model zaprezentowane uprzednio. Przedstawia to rysunek 3.8. Wymiary czasu i stacji są współdzielone przez wszystkie fakty. Dodatkowo, fakty związane z dostawą charakteryzowane są przez wymiar dostawcy. Wymiar lokalizacji jest rozszerzeniem wymiaru stacji i dostawcy i nie jest połączony bezpośrednio z żadnym faktem.



Rysunek 3.8: Pełny model konceptualny danych paliwowych

Rozdział 4

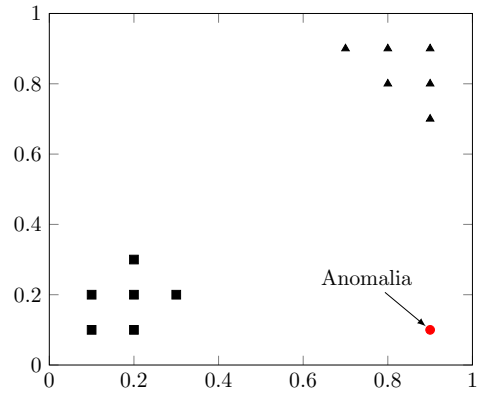
Anomalie w sieci stacji paliw i metody ich wykrywania

Niniejszy rozdział traktuje o anomaliach, czyli zjawiskach niepożądanych, występujących w sieci stacji paliw. Rozdział rozpoczyna się od przybliżenia definicji anomalii oraz przeglądu metod ich detekcji. Następnie, analizie poddane zostały anomalie występujące na stacjach paliw – przedstawiono ich klasyfikację oraz charakterystykę. Rozdział zamyka opis metody wykrywania wycieków paliwa – algorytmu TUBE, wraz z omówieniem rezultatów badań eksperymentalnych i wynikłych z nich konkluzji.

4.1 Anomalie i metody ich wykrywania

Anomalie można przedstawić jako odchylenia od przyjętych reguł i zasad, wynikłych z powtarzających się wzorców i zachowań, które są uznawane za normalne i oczekiwane [15, 89, 79, 84, 10]. Anomalie są zjawiskami nieuniknionymi, których wystąpienia nie da się w pełni przewidzieć, ze względu na ich nieregularność. Niemniej, ich obecność powinna być zawsze brana pod uwagę podczas analizy danych. Anomalie stanowią cenne źródło wiedzy, gdyż wprowadzają nowe i nieznane informacje do powtarzalnych i sprawdzonych wzorców. Nie wszystkie anomalie należy traktować jako zagrożenie, aczkolwiek zwyczajowo poszukuje się tych powiązanych z niebezpieczeństwem.

Anomalią krytyczną określić można anomalię, która powiązana jest z niebezpiecznym zjawiskiem, a skutki jej niewykrycia mogą być katastrofalne. Przykłady anomalii krytycznych występujących na stacjach paliw to wyciek paliwa ze zbiornika podziemnego [39]. Innymi przykładami anomalii krytycznych, występujących w różnych dziedzinach życia, mogą być: wypadek drogowy [93], niebezpieczne zachowanie się ludzi na ulicy, bagaż pozostawiony bez opieki na lotnisku, podejrzane transakcje w systemie bankowym [95, 132]. Analiza danych ukierunkowana na wykrywanie anomalii krytycznych jest przedmiotem wielu badań naukowych [132, 102, 10, 69].



Rysunek 4.1: Przykład anomalii w przestrzeni dwuwymiarowej

4.1.1 Klasyfikacja anomalii

Anomalie występujące w świecie rzeczywistym mają swoje odzwierciedlenie w danych opisujących różne aspekty tego świata. Można postawić tutaj tezę, iż każdej anomalii rzeczywistej, rozumianej jako zjawisko niepasujące do przyjętej normy, odpowiada anomalia w danych, rozumiana jako wartość lub ciąg wartości odstających. Wartości te mogą zostać wykryte metodami statystycznymi, a w konsekwencji – również i anomalia rzeczywista. Rysunek 4.1 przedstawia w sposób poglądowy anomalię w danych, która może być śladem po wystąpieniu anomalii rzeczywistej. Na przytoczonym rysunku, w dwuwymiarowym zbiorze danych wartość znajdująca się w prawym dolnym narożniku wykazuje zdecydowaną odrębność od pozostałych elementów, tworzących dwie grupy.

Anomalie występujące w danych można podzielić na kilka kategorii. W literaturze można spotkać się z podziałem na trzy podstawowe grupy [15]. Jako pierwsze, wyróżnione zostały anomalie punktowe, które są tożsame z wartościami odstającymi, podobnie jak na wspomnianym już rysunku 4.1. Drugim rodzajem są anomalie kontekstowe, dla które można zinterpretować w różny sposób, w zależności od kontekstu – w szczególności mogą one nie zostać sklasyfikowane jako anomalie w ogóle. Kontekstem jest w takim przypadku zazwyczaj otoczenie danej wartości, rozumiane jako ciąg poprzedzających lub następujących wartości, lub też inne okoliczności, współwystępujące z badaną wartością. Wiele anomalii paliwowych (anomalii rzeczywistych wraz z ich odzwierciedleniem w danych) ma charakter kontekstowy. Ostatnią grupą anomalii są anomalie kolektywne, które nie są pojedynczymi wartościami, a raczej ciągami tychże. Ten typ anomalii może być jednocześnie kontekstowy i również występuje na stacjach paliw – różnego rodzaju zjawiska niepożądane zazwyczaj trwają przez dłuższy czas, co objawia się całym szeregiem wartości, który odstaje od przyjętej normy.

4.1.2 Metody wykrywania anomalii

Przez pojęcie wykrywania anomalii rozumie się zazwyczaj ogół czynności podejmowanych w celu stwierdzenia faktu wystąpienia anomalii (informacja jakościowa), a czasem również określenia jej rozmiaru lub lokalizacji (informacja ilościowa). Co więcej, odnosząc się do zaproponowanego rozróżnienia na anomalie rzeczywiste (zjawiska nietypowe) oraz anomalie w danych (wartości odstające lub ich ciągi), wśród metod wykrywania anomalii można także rozróżnić dwa podstawowe podejścia. Pierwsze z nich ukierunkowane jest na anomalie w danych i wykorzystuje różnego rodzaju metody analizy statystycznej. Drugie skupia się na anomaliach rzeczywistych i polega na poszukiwaniu znanych wzorców, które są charakterystyczne dla znanej anomalii rzeczywistej. To drugie podejście wymaga wiedzy eksperckiej *a priori*. Istnieje też możliwość zastosowania obu podejść – wówczas poszukuje się pewnych znanych wzorców, a dodatkowo zwraca uwagę na te nieznane dotąd, acz nietypowe.

Istnieje wiele metod wykrywania anomalii w danych. W większości przypadków, analizie poddane są rekordy (wektory) złożone z wartości różnych atrybutów, które interpretuje się jako punkty pewnej przestrzeni (atrybutów). Wyróżnić można metody bazujące na miarach odległości [94, 98], których główną zasadą działania jest wyznaczanie rzeczonej *odległości* pomiędzy poszczególnymi wektorami. Jako anomalie klasyfikowane są te wektory, które ulokowane są w znaczącej odległości od pozostałych. Drugim typem metod są metody gęstościowe [112], w których wyznacza się *gęstość* w różnych rejonach przestrzeni atrybutów. Za anomalię uznaje się takie wektory, które znajdują się w obszarach o niskiej gęstości. Kolejną rodziną metod są metody statystyczne [78], analizujące rozkład prawdopodobieństwa w badanych zbiorach danych. Wykrycie anomalii następuje w momencie pojawienia się danych, których prawdopodobieństwo wystąpienia w podanych okolicznościach (oszacowane na podstawie analizy wcześniejszych przypadków) jest bardzo niskie. Do tej rodziny zaliczyć można również metody badające odchylenie standardowe [6], w których zakłada się, że anomalne wektory danych mają wysoki wpływ na uśrednione wartości całego zbioru danych, przez swoją unikalność. W metodach tych analizuje się zmianę odchylenia standardowego na skutek usunięcia ze zbioru kolejnych wartości – znacząca zmiana oznacza, że usunięty wektor można potraktować jako anomalię. Przedstawione powyżej metody wykrywania anomalii w danych, razem z innymi, takimi jak metody głębokościowe [86, 117] lub badające izolację [102], wpisują się w pierwszy z omówionych podejść do problemu wykrywania anomalii.

W przypadku drugiego podejścia, poszukuje się śladów w danych wystąpienia konkretnego i rzeczywistego zjawiska niepożądanego, wykorzystując do tego celu między innymi wiedzę ekspercką na temat istoty samego zjawiska, jak również możliwych jego efektów na „zachowanie się" danych [12]. W literaturze można często spotkać się z terminem eksploracji wzorców

sekwencyjnych (ang. *sequential pattern mining*). Metody takiej eksploracji stosowane są zazwyczaj w analizie szeregów czasowych i strumienie danych. Do popularnych zastosowań należy wyszukiwanie zjawisk pożądanych, jak i niepożądanych: w sieci [82, 131], w chmurach obliczeniowych [121], w hurtowniach danych [47], w danych finansowych [104], w zachowaniu człowieka [115].

4.1.3 Weryfikacja anomalii

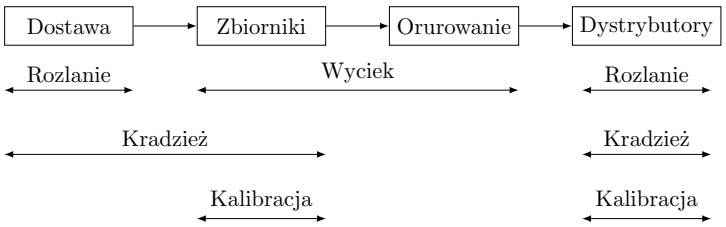
Weryfikacja anomalii polega na oszacowaniu prawdziwości jej anormalności, co ma miejsce już po wykryciu samej anomalii. W szczególności weryfikacja taka ma na celu uwiarygodnienie wyniku detekcji poprzez analizę kontekstu wystąpienia wykrytej anomalii – dotyczy to przede wszystkim anomalii kontekstowych. Zgodnie z podziałem anomalii zaproponowanym w literaturze [15], anomalie kontekstowe charakteryzują się ścisłą zależnością od powiązanych okoliczności występowania – w pewnych przypadkach ten sam wektor danych może zostać potraktowany jako anomalia, zaś w innych nie. Popularnym przykładem anomalii kontekstowej występującej w naturze jest temperatura powietrza, której ta sama wartość, w zależności od kombinacji pory roku i strefy klimatycznej, może być nietypowa lub całkiem zwyczajna. Weryfikacja anomalii w przytoczonym przykładzie polegałaby na zbadaniu pory roku oraz historycznych zakresów temperatury w określonych miesiącach i lokalizacjach.

W rzeczywistości anomalie kontekstowe często nie są traktowane jako takie właśnie, ze względu na ograniczoną wiedzę o ich naturze i pochodzeniu. Taki stan rzeczy prowadzić może do zbyt częstego (lub zbyt rzadkiego) zgłaszania alarmów oraz do niemożności odróżnienia faktycznego zagrożenia od zdarzenia nieistotnego. Można zatem stwierdzić, iż weryfikacja anomalii ma istotne znaczenie z punktu widzenia bezpieczeństwa, gdyż pozwala na obniżenie liczby fałszywych alarmów. Weryfikacja anomalii jest zazwyczaj znacznie bardziej kosztowna obliczeniowo niż ich detekcja, ze względu na zdecydowanie obszerniejszy rozmiar przetwarzanego wolumenu danych.

4.2 Anomalie w sieci stacji paliw

Pojedyncza stacja paliw oraz sieć takich stacji stanowią zespół połączonych ze sobą obiektów, w których mogą występować anomalie (rzeczywiste). Każdy z obiektów wchodzących w skład stacji, a w konsekwencji i całej sieci, jest zarazem źródłem danych. Zbiory danych powstałe na skutek sukcesywnego pozyskiwania i gromadzenia pomiarów mogą zostać poddane analizie ukierunkowanej na wykrywanie anomalii.

Niemniej, udane przeprowadzenie procesu detekcji anomalii w danych paliwowych wymaga dokładnego poznania istoty i charakterystyki poszczególnych anomalii rzeczywistych, występujących na badanych obiektach. W tym



Rysunek 4.2: Model toru przepływu paliwa na pojedynczej stacji paliw, uwzględniający problemy występujące na poszczególnych jego etapach

celu niezbędne jest stworzenie modelu przepływu paliwa, na poziomie pojedynczej stacji, jak również i całej sieci stacji paliw. Model ów należy rozumieć jako wizualizacja kolejnych obiektów i połączonych z nimi procesów, w których przebywa paliwo – począwszy od źródła (terminal), a skończywszy na ujściu (klienci końcowi).

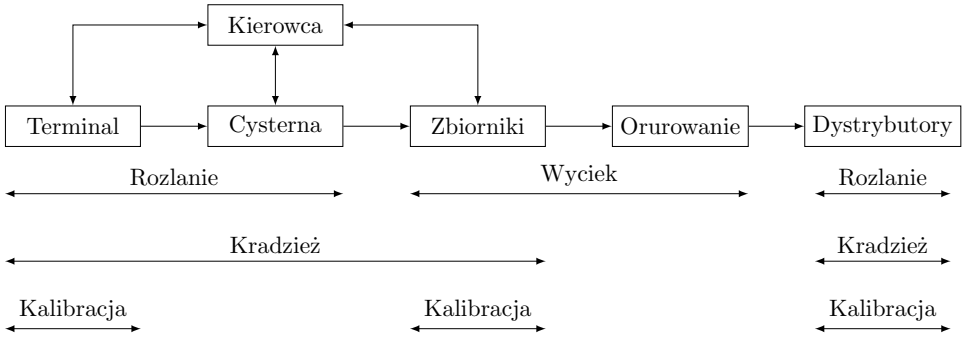
4.2.1 Modele torów przepływu paliwa

Na pojedynczej stacji, tor przepływu paliwa rozpoczyna się od procesu dostawy, kiedy to nowe paliwo jest dolewane do zbiornika. Wszystko to, co dzieje się przed dostawą, związane jest już z siecią stacji paliw i nie jest rozpatrywane na poziomie pojedynczej stacji. W następnym kroku paliwo jest przechowywane w zbiornikach, aż do momentu jego zakupu. Podczas transakcji sprzedaży paliwo jest pompowane przez zespół rur, aż do dystrybutorów, skąd trafia do odbiorców. Rysunek 4.2 przedstawia tor przepływu paliwa na pojedynczej stacji paliw.

Na każdym etapie toru przepływu paliwa może dojść do różnego rodzaju zjawisk niepożądanych. Na rysunku 4.2 anomalie te zaznaczono za pomocą strzałek, symbolizujących zakres ich występowania. Rozróżnienie pomiędzy pojęciami *wycieku* a *rozlania* zostało wprowadzone, w celu oddzielenia od siebie często niezauważonego zjawiska wycieku paliwa z instalacji, które powinny być szczelne, od przypadkowego rozlania paliwa, na skutek błędu człowieka. Występujący na wspomnianym rysunku termin *kalibracja* należy w tym kontekście traktować jako niepoprawną kalibrację, będącą przyczyną wielu przekłamań w danych pomiarowych. *Kradzież* paliwa jest zdarzeniem spowodowanym umyślnym działaniem człowieka i może zostać przeprowadzona zarówno ze zbiornika, jak i podczas dostawy lub sprzedaży.

Model toru przepływu paliwa w całej sieci stacji paliw jest nieco bardziej złożony (rysunek 4.3). Paliwo rozpoczyna swoją podróż od terminala, czyli obiektu zaopatrującego cysterny w paliwo. Następnie paliwo jest rozwożone przez kierowców na poszczególne stacje paliw i trafia do odpowiednich zbiorników. Dalsze etapy tego toru są już zbliżone do analogicznego toru dla pojedynczej stacji paliw.

Na rysunku 4.3) wprowadzono blok kierowcy, ze względu na jego znaczą-



Rysunek 4.3: Model toru przepływu paliwa w całej sieci stacji paliw, uwzględniający problemy występujące na poszczególnych jego etapach

cą rolę w procesie dostarczania paliw, a także potencjalną rolę w procesie jego kradzieży. Podobnie jak w przypadku pojedynczej stacji, również i tutaj zaznaczono strzałkami główne zjawiska niepożądane za pomocą strzałek.

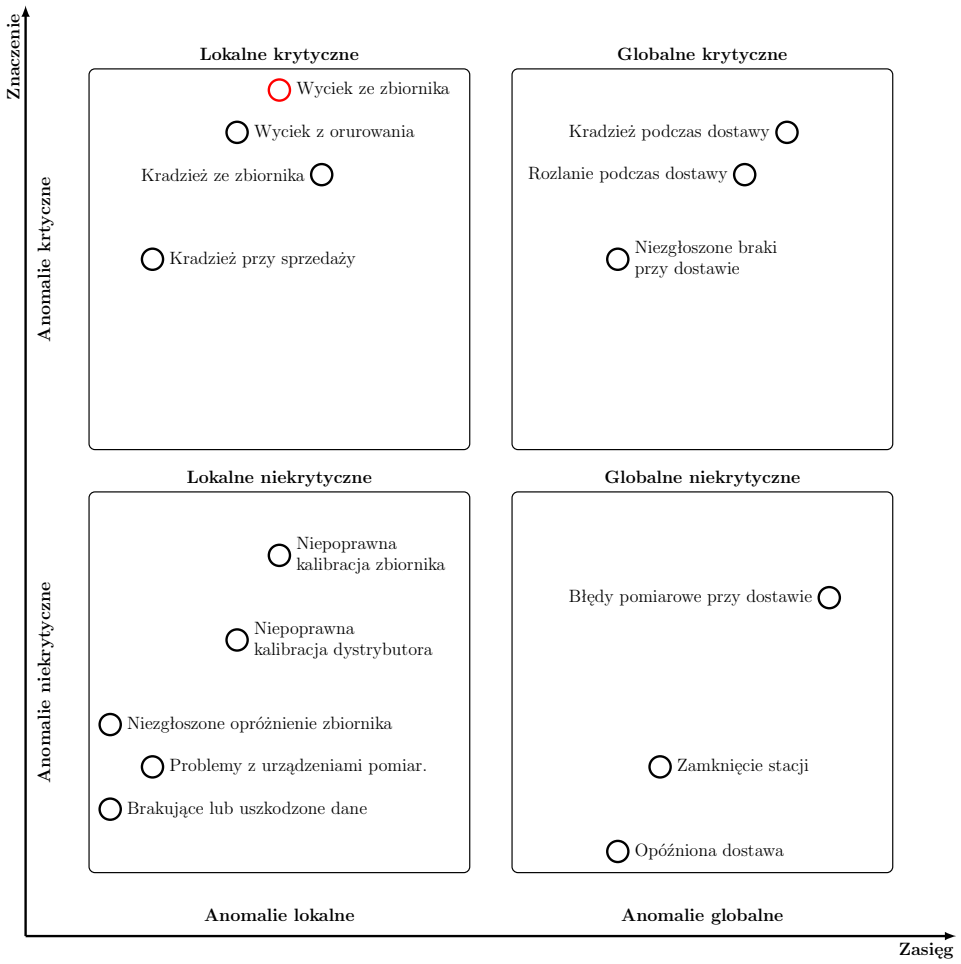
4.2.2 Podział anomalii występujących w sieci stacji paliw

W celu klasyfikacji wszystkich anomalii występujących na stacjach paliw, proponuje się wprowadzenie dwuosiowego podziału: *zasięg-znaczenie*. Kryterium zasięgu wiąże się z lokalnością danego zdarzenia, podczas gdy kryterium znaczenia – z istotnością skutków anomalii. Poniższa klasyfikacja stanowi rozwinięcie podziału anomalii zaproponowanego w pracy własnej [38]$^{\mathrm{W}}$ i przedstawia się następująco:

a) anomalie lokalne – zjawiska występujące w obrębie pojedynczej stacji paliw, których wykrycie jest możliwe wyłącznie na podstawie danych zebranych z tej stacji;

b) anomalie globalne – zjawiska obejmujące swoim zasięgiem więcej niż jedną stację lub występujące na pojedynczej stacji, lecz których wykrycie wymaga analizy danych z więcej niż pojedynczej stacji.

Drugie kryterium (znaczenie) określa istotność zagrożenia będącego wynikiem zajścia danego zjawiska. Najbardziej niebezpieczne zjawiska (anomalie krytyczne) związane są z wyciekiem paliwa, gdyż powodują skażenie środowiska. Nieco mniejsze znaczenie, aczkolwiek wciąż wysokie, ma kradzież paliwa, gdyż pociąga za sobą straty finansowe. Istnieją również pomniejsze anomalie (niekrytyczne), takie jak choćby niepoprawne funkcjonowanie urządzeń pomiarowych, które choć same w sobie nie stanowią zagrożenia, to jednak zmniejszają dokładność pomiarów, utrudniając tym samym skuteczność detekcji anomalii krytycznych. Podział anomalii paliwowych zgodnie z kryterium znaczenia przedstawia się następująco:



Rysunek 4.4: Dwuosiowy podział anomalii występujących w sieci stacji paliw

a) anomalie krytyczne – zjawiska niepożądane, których skutki są niebezpieczne dla środowiska lub powodują poważne straty finansowe;

b) anomalie niekrytyczne – zjawiska niepożądane lub inne nietypowe, które nie niosą ze sobą bezpośredniego zagrożenia, lecz są w stanie wpłynąć na jakość danych lub dostarczyć dodatkowych informacji.

Graficzne przedstawienie omawianego dwuosiowego podziału anomalii paliwowych zostało przedstawione na rysunku 4.4. Wyciek ze zbiornika paliwa został wyróżniony kolorem czerwonym, jako szczególnie niebezpieczny dla środowiska.

4.3 Metody wykrywania wycieków paliwa

Metody wykrywania anomalii występujących w przemyśle paliwowym skupiają się przeważnie na wykrywaniu wycieków. Jest tak ze względu na opisane wcześniej poważne skutki tego zjawiska dla środowiska naturalnego. Tym też zagadnieniem zajmowano się przede wszystkim w ramach badań własnych [39]^{W}. Pozostałe anomalie paliwowe, takie jak kradzież paliwa podczas dostaw oraz niepoprawna kalibracja urządzeń pomiarowych, również były analizowane w toku pozostałych badań własnych i opublikowane w formie artykułów [28, 38]^{W} oraz patentów [70, 40, 71]^{W}.

W literaturze najczęściej można spotkać się z metodami wykrywania wycieków przeznaczonymi dla rurociągów transportujących paliwo [105, 18]. Jednakże, nie jest to zastosowanie związane bezpośrednio ze stacjami paliw, a jedynie pokrewne tematycznie. Natomiast w przypadku wykrywania wycieków ze zbiorników paliw płynnych, istnieje kilka metod, które zostały sklasyfikowane w europejskiej normie EN 13160 [20]. Są to najczęściej rozwiązania wykorzystujące wyspecjalizowany osprzęt, taki jak czujniki wykrywające obecność paliwa poza zbiornikiem. Przykładowo, monitorowane może być ciśnienie pomiędzy wewnętrzną a zewnętrzną ścianką zbiornika lub rury, które ulega zmianie w przypadku wystąpienia wycieku. Metody te są bardzo dokładne i charakteryzują się wysoką czułością oraz wczesnym wykryciem wycieku.

Pomimo zalet opisanych wyżej metod, istnieją i są wykorzystywane techniki bazujące wyłącznie na analizie danych pomiarowych. Jest tak ze względu na fakt, iż dane potrzebne do analizy są łatwo dostępne, gdyż są gromadzone podczas normalnej pracy stacji. W konsekwencji, możliwe jest wykorzystywanie równolegle metod obu typów w celu zwiększenia bezpieczeństwa na stacji. Ponadto, możliwe staje się również wykonywanie analizy na danych historycznych, co jest niemożliwe w przypadku metod wykorzystujących urządzenia wykrywające wycieki. Kolejną zaletą metod analitycznych jest ich koszt wdrożenia, który jest zdecydowanie niższy niż instalacja wyspecjalizowanych urządzeń. Na chwilę obecną istnieje wciąż wiele stacji paliw i innych podobnych obiektów, które wykorzystują stosunkowo starą infrastrukturę, niewyposażoną we wspomniane urządzenia wykrywające wycieki. Metodom opierającym swoje działanie wyłącznie na zasadzie analizy danych poświęcona została inna część tej samej normy europejskiej [21] oraz norma amerykańska [126].

Niewiele jest prac naukowych opisujących zagadnienie wykrywania wycieków ze zbiorników paliw płynnych wyłącznie na podstawie analizy danych pomiarowych. Najczęściej stosowana jest tutaj opisana wcześniej operacja rekoncyliacji: w postaci rekoncyliacji masy, uwzględniająca dodatkowo efekt parowania ze zbiornika [101], lub też w przy użyciu metod uczenia maszynowego, wykrywających te przedziały czasu, w których wykryto wyciek [118].

4.3.1 Algorytm TUBE

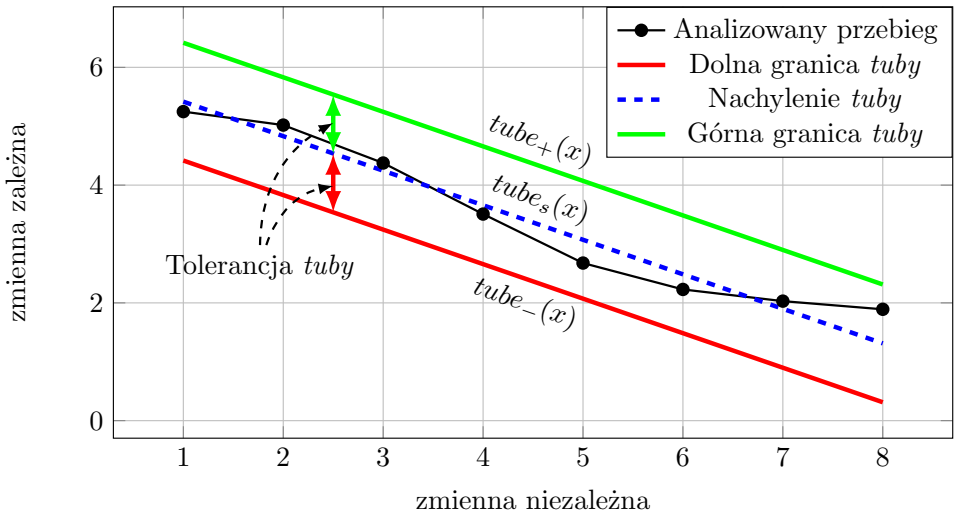
Przykładem metody wykrywania wycieków ze zbiorników paliw płynnych jest algorytm TUBE opisany w pracy własnej [39]^{W}. Algorytm ten został zaprojektowany jako nowa metoda detekcji trendów w szeregach czasowych, a jego zastosowanie do problemu wykrywania wycieków ze zbiorników paliw płynnych nie wyczerpuje jego potencjalnych aplikacji. Idea algorytmu TUBE, jako metody detekcji wycieków, polega na wykrywaniu dłuższych trendów w wartościach skumulowanego błędu rekoncyliacji zbiornika oraz ich interpretacji.

W algorytmie TUBE poczynione zostało założenie, że wyciek paliwa objawia się utrzymującym się w czasie spadkiem wartości skumulowanego błędu rekoncyliacji zbiornika. Etap wykrywania trendów poprzedzony jest filtracją danych, mającą na celu eliminację jak największej liczby krótkotrwałych zmian, nie świadczących o potencjalnym wycieku paliwa, a mających swą przyczynę w zjawiskach innej natury. Interpretacja wykrytych trendów przeprowadzana jest w celu określenia, które z nich mogą świadczyć o utracie paliwa (która nie musie jeszcze oznaczać wycieku). Jako metoda wykrywania anomalii, algorytm TUBE można zaklasyfikować do metod poszukujących wzorców charakterystycznych dla wykrywanych anomalii – w tym przypadku wykorzystuję on wiedzę *a priori* odnośnie wpływu wycieku paliwa na przebieg skumulowanych wartości błędu rekoncyliacji zbiornika paliwa.

Wejściem dla algorytmu TUBE jest zbiór pomiarów z określonego okresu czasu. Wyjściem algorytmu jest decyzja uznająca, że w danym okresie czasu zbiornik był szczelny, nastąpił wyciek paliwa lub też, że zbiór wymaga dokładniejszej analizy ze względu na niemożność wydania jednoznacznego werdyktu co do stanu zbiornika. Taka forma działania algorytmu została podyktowana wymaganiami ze strony amerykańskiej normy dotyczącej certyfikacji metod wykrywania wycieków EPA SIR [126]. W konsekwencji, w celu adaptacji algorytmu TUBE do rozwiązań czasu rzeczywistego należy zastosować sukcesywną analizę w nachodzących na siebie przedziałach czasu. Algorytm TUBE został poddany ewaluacji przedstawionej we wspomnianej normie EPA SIR i uzyskał prawdopodobieństwo wykrycia wycieku równe 93,11% oraz prawdopodobieństwo wystąpienia fałszywego alarmu równe 0,73%.

Algorytm TUBE nazwę swoją wziął od pojęcia *tuby*, czyli swoistej obwiedni podciągu rekordów, które wykazują stosunkowo spójny (mieszczący się w zadanej tolerancji) trend. Główną ideą algorytmu TUBE jest aproksymowanie szeregu czasowego za pomocą ciągu *tub*, z których każda charakteryzuje się określonym nachyleniem oraz długością. Dzięki temu, początkowo zanieczyszczony różnymi szumami przebieg staje się czytelny w interpretacji.

Algorytm TUBE wykazuje przyrostowe działanie: początkowo zakłada, że pierwszych kilka rekordów tworzy pierwszą *tubę*, a dla kolejnych sprawdzane jest, czy można je do niej zakwalifikować (wówczas długość *tuby* wzrasta), czy też konieczne jest utworzenie nowej, o innym nachyleniu. Sytuacja taka



Rysunek 4.5: Graficzne przedstawienie idei *tuby*, jako metody reprezentacji trendu [39]

wynika ze zmiany trendu w danych.

O tym, czy dany rekord zostanie „dopisany" do bieżącej *tuby*, czy też rozpocznie kolejną, decydują dwa parametry: tolerancji *pionowej* oraz *poziomej*. Ten pierwszy określa, jak bardzo od środka *tuby* oddalić się może dany rekord, zaś ten drugi określa maksymalną liczbę rekordów następujących po sobie, które mogą przekroczyć wartość tolerancji pionowej. Rysunek 4.5 ilustruje ideę *tuby* jako przybliżenia trendu występującego w danych. Zaznaczone na rysunku dwie proste (zielona i czerwona) wyznaczają zakresy tolerancji pionowej, a niebieska przerywana linia określa nachylenie *tuby*.

Interpretacja uzyskanych we wspomniany sposób trendów polega na sklasyfikowaniu ich jako potencjalnie wyciekowych (mogących powstać na skutek wycieku) lub niewyciekowych (nie noszących znamion wycieku). Jeśli w danym zbiorze danych przeważa liczba *tub* wyciekowych, algorytm TUBE zgłosi fakt wystąpienia wycieku oraz oszacuje jego natężenie (wyrażone jako iloraz objętości do czasu), na podstawie nachylenia *tub* wyciekowych.

Algorytm TUBE został zaimplementowany na potrzeby badań eksperymentalnych [39] związanych z jego ewaluacją zgodnie z kryteriami podanymi w amerykańskiej normie EPA SIR [126]. W kwestiach architektonicznych, zdecydowano się na podejście strumieniowe, w którym kolejne etapy przetwarzania danych zostały zaimplementowane w formie operatorów tworzących razem potok (ang. *pipeline*). Każdy z operatorów na wejściu otrzymuje jeden rekord, a na wyjściu zwraca strumień, który w szczególnym przypadku może być pusty, lub też składać się z jednego bądź wielu rekordów. Dzięki takiemu podejściu udało się uwzględnić przypadki brzegowe, w których, ze względu na przejściowy brak wymaganej liczby rekordów, jedynym zadaniem

operatorów było gromadzenie danych, a także przypadków, w których należało wysłać na wyjście wszystkie zaległe rekordy. Na potok przetwarzania danych składają się następujące operatory:

a) operator filtracji pierwszego poziomu,
b) operator sumowania,
c) operator filtracji drugiego poziomu,
d) operator detekcji trendów,
e) operator interpretacji trendów.

Oprócz wyżej wymienionych operatorów, w potok wpleciono kilka pobocznych operacji, takich jak choćby formatowanie tekstu oraz odczyt i zapis danych pomiarowych. Cały potok podzielić można na trzy zasadnicze etapy: filtracji (przetwarzanie wstępne, ang. *preprocessing*) oraz detekcję (przetwarzanie zasadnicze) i interpretację (przetwarzanie końcowe, ang. *postprocessing*).

4.3.2 Etap filtracji danych

Zadaniem operatorów filtracji jest wygładzanie analizowanego szeregu czasowego – w przypadku problemu wykrywania wycieków jest to przebieg skumulowanego błędu rekoncyliacji zbiornika. Etap filtracji jest niezbędny z punktu widzenia docelowej metody wykrywania oraz interpretacji trendów, gdyż nieprzetworzone wartości błędów rekoncyliacji są obarczone sporym zaszumieniem, które negatywnie wpływa na jakość dalszych analiz. Działanie filtracji w algorytmie TUBE sprowadza się do sukcesywnego zastępowania rekordów wartością wybranej funkcji, której argumenty stanowią rekordy będące otoczeniem filtrowanego rekordu (otoczenie to zwane jest też oknem filtracji).

Ze względów praktycznych, proces filtracji realizowany w ramach algorytmu TUBE przeprowadzany jest w dwóch etapach, przy czym pierwszy odbywa się jeszcze przed operacją sumowania, w wyniku której powstają wartości skumulowanego błędu rekoncyliacji zbiornika – filtracji pierwszego poziomu poddawane są nieskumulowane wartości tego błędu. Jest tak ze względu na konieczność usunięcia licznych jednostkowych skoków wartości, które zdecydowanie łatwiej jest usunąć jeszcze przed sumowaniem.

Drugi poziom filtracji ma na celu końcowe wygładzenie skumulowanych wartości błędu rekoncyliacji. Niezależnie od poziomu, filtracja wykonywana jest w tym samym operatorze (innej jego instancji), którego charakterystykę przedstawia algorytm 4.1. Jego działanie jest niezależne do przyjętej metody filtrującej dane – jest ona wywoływana jako zewnętrzna funkcja o nazwie FILTER (przykładowo w linii 4). Zadaniem operatora filtracji jest przede wszystkim obsługa samych danych, na którą składa się grupowanie rekordów w przesuwnym oknie liczebnościowym o zadanym rozmiarze oraz uwzględnianie przypadków brzegowych (początek i koniec zbioru danych).

Algorytm 4.1 Operator filtracji

Wejście: *record* : przetwarzany rekord strumienia
Wyjście: strumień przetworzonych rekordów
Stałe:
k : rozmiar okna filtrującego
Q: pusta kolejka
FILTER : funkcja filtrująca

```
1:  function FILTEROPERATOR(record)
2:      let filtered = ∅                                  ▷ pusty rekord
3:      if record = EOS then                              ▷ koniec strumienia
4:          filtered ← FILTER(Q)
5:          return ⟨filtered × ⌊k/2⌋⟩
6:      end if
7:      record ⇒ Q                                        ▷ dopisanie na koniec kolejki
8:      if ‖Q‖ < k then                                   ▷ niepełna kolejka
9:          return ⟨ ⟩
10:     end if
11:     if ‖Q‖ = k then                                   ▷ pierwsze pełne okno
12:         filtered ← FILTER(Q)
13:         return ⟨filtered × ⌈k/2⌉⟩
14:     end if
15:     φ ⇐ Q                                             ▷ usunięcie pierwszego elementu
16:     filtered ← FILTER(Q)
17:     return ⟨filtered⟩
18: end function
```

Na uwagę zasługują linie zawierające słowo kluczowe *return* (przykładowo linia 5), w których zwracana jest różna liczba rekordów – dla uproszczenia przyjęto zapis, w którym w nawiasach trójkątnych znajduje się konstruktor strumienia: puste nawiasy oznaczają pusty strumień, a zapis z iloczynem wskazuje na powielenie danego rekordu. W algorytmie TUBE zastosowano kilka metod filtrujących – na podstawie badań wybrano rozszerzony filtr medianowy oraz regresyjny, których działanie pokazuje algorytm 4.2.

Idea klasycznego filtru medianowego dla danych jednowymiarowych zakłada, że dany rekord zostaje zastąpiony medianą ze swojego otoczenia, co powoduje usunięcie wartości odstających. Wykorzystywana w algorytmie TUBE metoda rozszerzonego filtra medianowego polega na uwzględnieniu w operacji mediany również wartości średniej z otoczenia rekordu oraz wartości zerowej (linia 3). Ta ostatnia wartość ma swoje uzasadnienie charakterem przetwarzanych danych – są to wartości błędu, które w idealnych warunkach przyjmują wartość zerową. Działanie filtru regresyjnego polega na wyznaczeniu współczynników prostej na podstawie rekordów wchodzących w skład okna (regresja liniowa), a następnie na zastąpieniu filtrowanego

Algorytm 4.2 Metody filtrowania w algorytmie TUBE

Wejście: Q : kolejka rekordów do przefiltrowania
Wyjście: przefiltrowany rekord

```
1: function augMedFilter(Q)
2:     let avg = average(Q)
3:     let arg = Q ∪ {avg, 0}                    ▷ rozszerzenie zbioru
4:     let med = median(arg)
5:     return med
6: end function

7: function regFilter(Q)
8:     let a = regLinA(Q)
9:     let b = regLinB(Q, a)
10:    let t = time(Q)                           ▷ czas środka okna
11:    let result = a · t + b
12:    return result
13: end function
```

rekordu wartością należącą do wyznaczonej prostej (linia 11).

Jak uprzednio wspomniano, pomiędzy dwoma etapami filtracji odbywa się sumowanie wartości błędów rekoncyliacji zbiornika, w wyniku którego powstaje ciąg wartości skumulowanego błędu rekoncyliacji. Operator sumowania (algorytm 4.3) sukcesywnie dodaje kolejne wartości błędów do siebie, zwracając każdorazowo zaktualizowaną wartość sumy.

Algorytm 4.3 Operator sumujący

Wejście: *record* : przetwarzany rekord strumienia
Wyjście: strumień przetworzonych rekordów
Stałe: *acc* : akumulator, początkowo równy 0

```
1: function sumOperator(record)
2:     acc ← acc + record                        ▷ aktualizacja akumulatora
3:     return ⟨acc⟩
4: end function
```

4.3.3 Etap detekcji trendów

Po zakończeniu wstępnego przetwarzania (filtracja i sumowanie), dane w postaci skumulowanych wartości błędu rekoncyliacji zbiornika poddawane są operacji wykrywania trendów. W wyniku przeprowadzenia tejże, wygenerowany zostaje ciąg *tub*, czyli struktur opisujących fragment analizowanego przebiegu, których cechuje się stabilnym trendem (z zadaną tolerancją). Algorytm 4.4 przedstawia postać struktury *tuby*.

Algorytm 4.4 Struktura opisująca *tubę*

structure *tube*
a : współczynnik kierunkowy równania prostej tworzącej tubę
b : wyraz wolny równania prostej tworzącej tubę
α : współczynnik nachylenia tuby
λ : długość tuby wyrażona liczbą rekordów
end structure

Każda *tuba* zawiera w sobie informacje o nachyleniu oraz o długości – połączenie tych dwóch informacji pozwala na zinterpretowanie *tuby* jako potencjalnie niebezpiecznej (dłuższa, o dużym spadku), normalnej (dłuższa, bez spadku) lub nietypowej (krótka, o różnym nachyleniu). Tworzenie *tub* jest procesem dość złożonym – główny proces przedstawiony został jako algorytm 4.5, który wykorzystuje kilka funkcji zewnętrznych, omówionych w dalszej części.

Proces tworzenia *tub* rozpoczyna się od oczekiwania na wymaganą liczbę rekordów (linia 7). Po jej osiągnięciu tworzona jest pierwsza *tuba* (linia 10) oraz wyznaczane są współczynniki prostej nachylenia oraz testowane jest odstępstwo od tej prostej. Jeśli kolejne rekordy mieszczą się w założonej tolerancji, to są dodawane do aktualnej tuby (linia 15) – w przeciwnym wypadku, po przekroczeniu określonego limitu niepasujących rekordów (linia 20) tworzona jest nowa *tuba*, a aktualna jest kończona (linia 23).

Procedury i funkcje pomocnicze w procesie wykrywania trendów przedstawia algorytm 4.6. Tworzenie i kończenie *tuby* polega na wyznaczeniu jej parametrów, przy czym współczynniki nachylenia prostej ustawiane są na początku, podczas gdy nachylenie jest wartością wyznaczaną na końcu, na podstawie rzeczywistych danych objętych zakresem *tuby*. Wyznaczanie tolerancji polega na obliczeniu odległości danego rekordu od prostej będącej wynikiem regresji liniowej rekordów objętych *tubą*. Testowanie przynależności rekordu do danej *tuby* sprowadza się do sprawdzenia, czy dany rekord mieści się w obliczonej uprzednio tolerancji.

4.3.4 Etap interpretacji trendów

Wynikiem procesu detekcji trendów jest zbiór *tub*, z których każda opisuje utrzymujący się dłużej stabilny trend. Dane te stanowią wejście do procesu interpretacji, w którego wyniku powstaje pojedyncza decyzja, charakteryzująca dany zbiornik w całym objętym analizą przedziale czasu. Decyzja ta opisana jest strukturą (algorytm 4.7), na którą składa się etykieta (*wyciek*, *brak wycieku*, *decyzja niejednoznaczna*) oraz wartość rzeczywista, opisująca natężenie wycieku.

Proces interpretacji trendów przedstawia algorytm 4.8. Wydanie decyzji następuje dopiero po przetworzeniu ostatniej *tuby* (linia 13). Zanim to

Algorytm 4.5 Operator wykrywania trendów

Wejście: *record* : przetwarzany rekord strumienia
Wyjście: strumień przetworzonych rekordów
Stałe:
ξ : tolerancja pozioma
υ : tolerancja pionowa
υ_{min} : dolne ograniczenie tolerancji pionowej
υ_{max} : górne ograniczenie tolerancji pionowej
υ_{fac} : mnożnik tolerancji pionowej
Q : pusta kolejka (dla rekordów należących do *tuby*)
O : pusta kolejka (dla rekordów leżących poza *tubą*)
T : aktualna *tuba*

```
1:  function tubeDetectionOperator(record)
2:      if record = EOS then                                  ▷ koniec strumienia
3:          finishTube(T, Q)
4:          return ⟨T⟩
5:      end if
6:      record ⇒ Q
7:      if ‖Q‖ < ξ then          ▷ oczekiwanie na minimalną liczbę rekordów
8:          return ⟨ ⟩
9:      end if
10:     if T = NULL and ‖Q‖ = ξ then                ▷ początek pierwszej tuby
11:         createTube(T, Q)
12:         υ ← tolerance(Q)
13:         return ⟨ ⟩
14:     end if
15:     if inTube(record, Q, υ) then                ▷ rekord należy do tuby
16:         clear(O)
17:         return ⟨ ⟩
18:     else                                        ▷ rekord nie należy do tuby
19:         record ⇒ O                                  ▷ dodanie do kolejki O
20:         if ‖O‖ ⩽ ξ then                             ▷ nie osiągnięto limitu
21:             return ⟨ ⟩
22:         end if
23:         finishTube(T, Q \ O)                        ▷ tworzenie nowej tuby
24:         let T′ = T
25:         Q ← O
26:         clear(O)
27:         createTube(T, Q)
28:         υ ← tolerance(Q)
29:         return ⟨T′⟩
30:     end if
31: end function
```

Algorytm 4.6 Procedury i funkcje pomocnicze operatora detekcji trendów

procedure createTube(T, Q)
    $T.a \leftarrow$ regLinA(Q) ▷ współczynnik kierunkowy równania prostej
    $T.b \leftarrow$ regLinB(Q, a) ▷ wyraz wolny równania prostej
end procedure

procedure finishTube(T, Q)
    $T.\alpha \leftarrow$ regLinA(Q) ▷ współczynnik kierunkowy równania prostej
    $T.\lambda \leftarrow \|Q\|$ ▷ rozmiar kolejki
end procedure

function tolerance(Q)
    let $sum = 0$
    for $i \leftarrow 0 \ldots (\|Q\| - 1)$ **do**
        let $r = Q[i]$ ▷ wartość rekordu
        let $\hat{r} = T.a \cdot i + T.b$ ▷ szacowana wartość rekordu
        let $\delta = |r - \hat{r}|$ ▷ różnica bezwzględna
        $sum \leftarrow sum + \delta$
    end for
    let $dist = \frac{sum}{\|Q\|} \cdot \upsilon_{fac}$ ▷ mnożenie przez mnożnik tolerancji
    return median($\{\upsilon_{min}, dist, \upsilon_{max}\}$) ▷ uwzględnienie ograniczenia
end function

function inTube($record, Q, \upsilon$)
    let $min = T.a \cdot$ index($Q, record$) $+ T.b - \upsilon$ ▷ dolne ograniczenie
    let $max = T.a \cdot$ index($Q, record$) $+ T.b + \upsilon$ ▷ górne ograniczenie
    return $min \leqslant record$ **and** $record \leqslant max$
end function

Algorytm 4.7 Struktura opisująca status zbiornika

structure *status*
    d : decyzja {`LEAK`, `TIGHT`, `INCONCLUSIVE`}
    ϕ : natężenie wycieku
end structure

jednak nastąpi, poszczególne tuby są dzielone na dwie kategorie: wyciekowe (gdy natężenie przekracza zadany próg – linia 6) oraz niewyciekowe (wszystkie pozostałe). Następnie, dla obu kategorii następuje zliczanie liczby rekordów objętych tymi *tubami*. Powstałe w wyniku sumy są podstawą do wyznaczenia odsetka rekordów wyciekowych oraz zestawienia go z założonym limitem (linia 16), po przekroczeniu którego cały zbiór danych uznany jest za wyciekowy.

4.3.5 Wyniki badań eksperymentalnych algorytmu TUBE

W celu dobrania optymalnego zestawu parametrów algorytmu TUBE, jak również jego ewaluacji pod kątem spełniania norm podanych przez amerykańską agencję ochrony środowiska EPA SIR [126], dokonano szeregu eksperymentów. Eksperymenty przeprowadzane były na syntetycznych zbiorach danych zawierających około 30 pomiarów dobowych – jest to standard narzucony przez wspomnianą normę EPA SIR. Przeprowadzone badania można podzielić na dwie podstawowe grupy: związane z wyborem najlepszej metody filtrowania danych oraz z wyznaczeniem współczynników jakości detekcji wycieków.

W obrębie grupy eksperymentów odnoszących się do metod filtrowania danych, przeprowadzono badania porównawcze dla trzech strategii (opisanych dokładnie w rozdziale 4.3.2):

a) filtra medianowego – zastępującego dany rekord medianą spośród jego otoczenia;

b) rozszerzonego filtra medianowego (zwanego dalej filtrem *rozszerzonym*) – zastępującego dany rekord medianą spośród jego otoczenia, średniej z tego otoczenia oraz wartości zero;

c) filtra regresyjnego – zastępującego dany rekord wartością należącą do prostej powstałej na skutek obliczenia regresji liniowej dla jego otoczenia.

Dodatkowo, każdy z filtrów testowany był dla trzech różnych *strategii brzegowych* – pod tym terminem należy rozumieć metodę radzenia sobie z brakiem pełnego okna dla rekordów leżących na obu końcach przetwarzanego zbioru danych. W ramach badań przeprowadzonych dla algorytmu TUBE zaimplementowano następujące strategie brzegowe:

a) strategia *none* – polegająca na zaniechaniu filtracji brzegowych rekordów i odrzuceniu ich z wynikowego zbioru danych (w konsekwencji jest on mniejszy od zbioru wejściowego);

b) strategia *raw* – polegająca na zaniechaniu filtracji brzegowych rekordów i przeniesieniu ich w niezmienionej postaci do zbioru wynikowego;

Algorytm 4.8 Operator interpretacji trendów

Wejście: *record* : przetwarzany rekord strumienia
Wyjście: strumień przetworzonych rekordów
Stałe:
θ : próg natężenia wycieku
leaklimit : górny próg liczby wyciekowych rekordów
tightlimit : dolny próg liczby wyciekowych rekordów

1: **function** TUBEINTERPRETATIONOPERATOR(*record*)
2: **let** $leakcounter = 0$ ▷ licznik rekordów z wyciekiem
3: **let** $tightcounter = 0$ ▷ licznik rekordów bez wycieku
4: **let** $leaksum = 0$ ▷ sumaryczny wyciek
5: **if** $record \neq$ `EOS` **then** ▷ kolejna tuba do przetworzenia
6: **if** $record.\alpha < -\theta$ **then** ▷ wyciek znaczący
7: $leakcounter \leftarrow leakcounter + record.\lambda$
8: $leaksum \leftarrow leaksum - record.\alpha \cdot record.\lambda$
9: **else**
10: $tightcounter \leftarrow tightcounter + record.\lambda$
11: **end if**
12: **return** $\langle\,\rangle$
13: **else** ▷ koniec strumienia
14: **let** $S = \emptyset$ ▷ status zbiornika
15: **let** $totalcounter = leakcounter + tightcounter$
16: **let** $leakratio = \frac{leakcounter}{totalcounter}$ ▷ odsetek rekordów wyciekowych
17: **if** $leakratio \geqslant leaklimit$ **then** ▷ wyciek
18: **let** $leakrate = \frac{leaksum}{leakcounter}$
19: $S.d \leftarrow$ `LEAK`
20: $S.\phi \leftarrow leakrate$
21: **return** $\langle S \rangle$
22: **else if** $leakration \leqslant tightlimit$ **then** ▷ brak wycieku
23: $S.d \leftarrow$ `TIGHT`
24: $S.\phi \leftarrow 0$
25: **return** $\langle S \rangle$
26: **else**
27: $S.d \leftarrow$ `INCONCLUSIVE` ▷ dane niewystarczające
28: $S.\phi \leftarrow 0$
29: **return** $\langle S \rangle$
30: **end if**
31: **end if**
32: **end function**

c) strategia *copy* – polegająca na zaniechaniu filtracji brzegowych rekordów i zastąpieniu ich kopią przefiltrowanych brzegowych rekordów (strategia ta została użyta w algorytmie 4.1).

Badania porównawcze poszczególnych metod filtrowania przeprowadzono na zasadzie porównania natężenia sztucznie nałożonego wycieku (wartości znanej) ze średnią arytmetyczną z wartości błędu rekoncyliacji zbiornika. Ta ostatnia wartość, w idealnych warunkach (brak zaszumień oraz innych anomalii poza wyciekiem) jest równa natężeniu wycieku – każda wartość błędu rekoncyliacji oznacza różnicę pomiędzy paliwem dolanym a ubyłym ze zbiornika. W rzeczywistości średnia ta jest dość naiwnym przybliżeniem rzeczywistego natężenia wycieku, niemniej stanowi zarazem podstawę do oszacowania skuteczności filtra – im lepiej dany filtr usuwa zniekształcenia, tym mniejsza jest rozbieżność pomiędzy oryginalnym natężeniem wycieku, a uzyskaną w sposób naiwny jego estymacją. W konsekwencji, zdecydowano się na wykorzystanie błędu średniokwadratowego pomiędzy rzeczonymi wartościami jako miary jakości filtra.

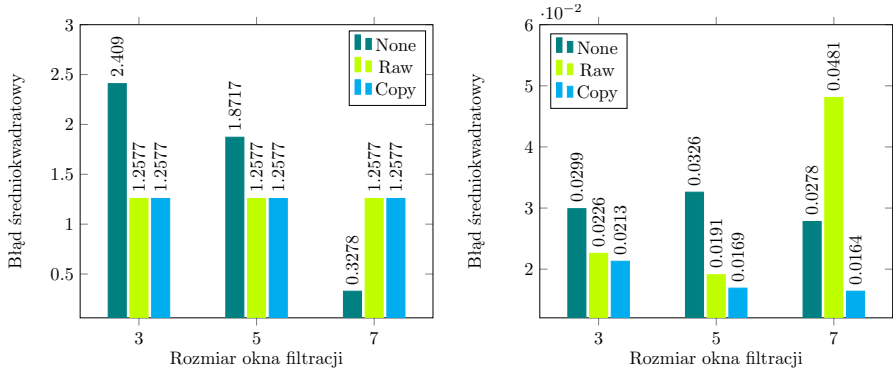
Na rysunku 4.6 przedstawiono wyniki eksperymentów porównawczych dla trzech metod filtrowania, trzech strategii brzegowych oraz trzech, arbitralnie wybranych, rozmiarów okna: 3, 5, 7 (wybór był częściowo podyktowany rozmiarem danych używanych w ewaluacji: 30 rekordów).

Jako wartość referencyjna, w badaniu uwzględniono również brak filtra (rysunek 4.6a) – w tym przypadku rozmiar okna miał wpływ jedynie na wyniki dla strategii brzegowej *none*, w której usuwano część danych. Przytoczone wyniki badania jasno pokazują, że najlepszą metodą filtrowania (charakteryzującą się najniższymi wartościami błędu śrerdniokwadratowego) jest filtr rozszerzony, a w drugiej kolejności (i z niewielką różnicą) filtr medianowy. Filtr regresyjny nie poprawił, a nawet nieznacznie pogorszył wyniki.

W celu alternatywnego ukazania wyników tego badania, wykonano również wykres pudełkowy, ukazany na rysunku 4.7. Każde *pudełko* obejmuje całość pomiarów z pojedynczego wykresu rysunku 4.6 (filtr rozszerzony oznaczono jako *MED**). Na wykresie tym czytelnie widać zalety filtra rozszerzonego – nie tylko uzyskał on najniższą wartość błędu średniokwadratowego, lecz również cechuje się najmniejszym rozrzutem wartości, przy różnych strategiach brzegowych oraz różnych rozmiarach okna filtracji.

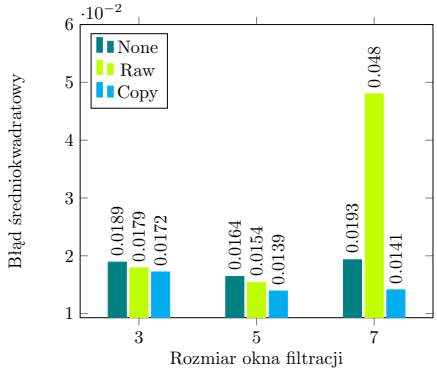
Kolejnym badaniem wykonanym w ramach wybrania optymalnej strategii filtrowania było porównanie filtrów drugiego poziomu, operujących na skumulowanych wartościach błędu rekoncyliacji zbiornika. W tym celu zawężono zakres parametrów, bazując na wynikach poprzedniego badania. W konsekwencji zdecydowano się na wzięcie pod uwagę wyłącznie okna filtracji o rozmiarze 5, strategii brzegowych: *copy* oraz *raw* oraz filtrów: medianowego oraz rozszerzonego jako metody filtracji pierwszego poziomu.

Rysunek 4.8 przedstawia wyniki omówionego badania. Strategia brzego-

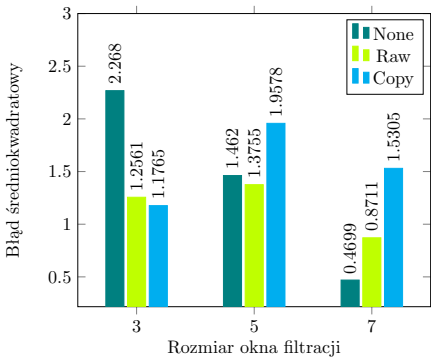


(a) brak filtra

(b) filtr medianowy

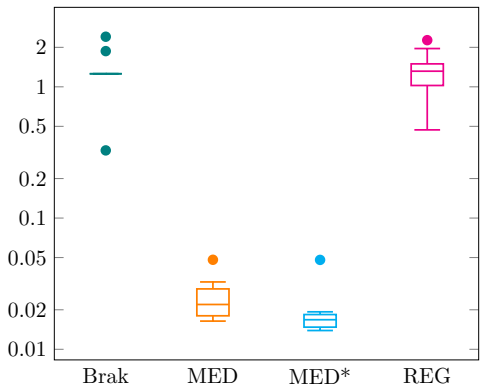


(c) filtr rozszerzony

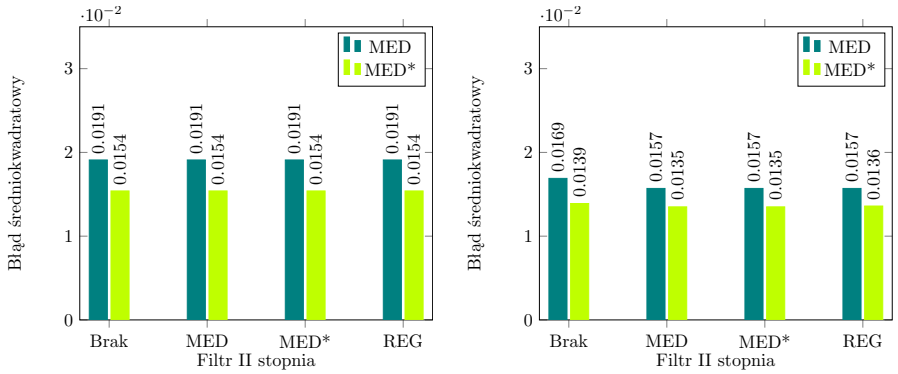


(d) filtr regresyjny

Rysunek 4.6: Wyniki testów filtrów pierwszego stopnia



Rysunek 4.7: Porównanie filtrów pierwszego stopnia



(a) strategia brzegowa *raw* (b) strategia brzegowa *copy*

Rysunek 4.8: Wyniki testów filtrów drugiego stopnia

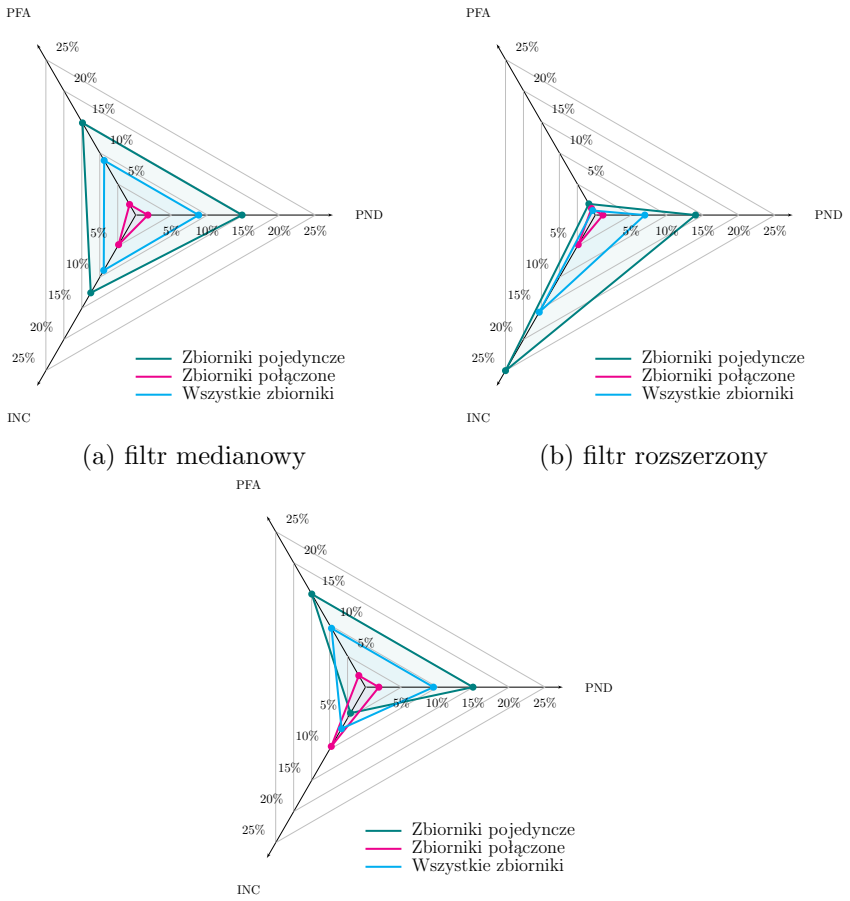
wa polegająca na powielaniu przefiltrowanych wartości (*copy*) okazała się nieznacznie lepsza, bez względu na wartości pozostałych parametrów, podobnie jak filtr rozszerzony jako metoda filtracji pierwszego poziomu. Natomiast wyniki porównania różnych metod filtrowania dla drugiego poziomu filtracji okazały się niezbyt jednoznaczne – wszystkie trzy analizowane metody uzyskały zbliżone wyniki, które są niewiele lepsze (lecz lepsze) od braku filtracji drugiego poziomu.

Całość eksperymentów poświęconych metodom filtracji danych pokazała, że jest to ważny element w potoku przetwarzania danych, który znacząco polepsza jakość danych. Filtr rozszerzony okazał się najlepszą strategią filtrowania niezsumowanych wartości błędu rekoncyliacji, a strategia *copy* okazała się najtrafniejszą przy rozwiązywaniu problemu danych brzegowych.

Kolejna grupa badań związana była już z mechanizmem wykrywania i interpretacji trendów za pomocą *tub*. Zgodnie z przytoczoną normą, najbardziej istotnymi metrykami opisującymi metodę wykrywania wycieków paliwa jest prawdopodobieństwo wykrycia wycieku oraz prawdopodobieństwo fałszywego alarmu. Dodatkowo, ze względu na możliwość niejednoznacznej interpretacji danego zbioru danych i oznaczenia go jako *inconclusive* (niejednoznaczny), trzecią mierzoną metryką był procent zbiorów w ten sposób oznaczonych.

Rysunek 4.9 przedstawia podsumowanie badania mechanizmu wykrywania i interpretacji trendów, w formie wykresów radarowych o trzech osiach. Każda oś odpowiada jednej z podanych wyżej metryk: PND (prawdopodobieństwo *niewykrycia* wycieku – odwrócone prawdopodobieństwo wykrycia wycieku, wprowadzone ze względu na charakter wykresu), PFA (prawdopodobieństwo wystąpienia fałszywego alarmu) oraz INC (procent zbiorów oznaczonych jako niejednoznaczne). Im mniejsze jest pole zakreślone na poszczególnych wykresach, tym uzyskany wynik jest lepszy.

Rzeczone badanie przeprowadzono dla trzech różnych strategii filtro-



(a) filtr medianowy

(b) filtr rozszerzony

(c) filtr regresyjny

Rysunek 4.9: Wyniki testów mechanizmu detekcji i interpretacji trendów

wania drugiego poziomu, gdyż poprzednie badanie w sposób jednoznaczny nie wskazało żadnej z nich jako optymalnej. Dodatkowo rozróżniono zbiory danych na te pochodzące ze zbiorników pojedynczych oraz połączonych. Uzyskane wyniki, niezależnie od przyjętej strategii filtrowania, pokazują, iż zbiorniki połączone uzyskały zupełnie lepsze parametry, niż zbiorniki pojedyncze, co jest kwestią niezwykle interesującą (zbiorniki te uważane są za trudniejsze w analizie). Prawdopodobnym wyjaśnieniem takiego zjawiska jest, spowodowana większą sumaryczną objętością, mniejsza podatność na zmiany temperatury związane z rozszerzalnością cieplną cieczy, powstałe w wyniku dostaw paliwa.

Wybór metody filtracji drugiego stopnia w dalszym ciągu pozostał kwestią nierozstrzygniętą – o ile z jednej strony najlepsze wyniki uzyskano (ponownie) dla filtra rozszerzonego, to z drugiej strony filtr regresyjny dał zdecydowanie mniej wyników niejednoznacznych. Najlepszy wynik uzyskano dla

filtra rozszerzonego, użytego na obu etapach filtracji – prawdopodobieństwo wykrycia wyniosło wówczas 93,11%, a prawdopodobieństwo wystąpienia fałszywego alarmu – 0,73%. Warto zwrócić uwagę na fakt, iż najmniejszy procent wyników niejednoznacznych uzyskano dla filtra regresyjnego użytego na drugim etapie filtracji.

4.3.6 Dyskusja na temat uzyskanych wyników

Uzyskane wyniki, zwłaszcza w zestawieniu z minimalnymi wytycznymi podanymi w normie EPA SIR, wydawać się mogą nie w pełni satysfakcjonujące. Niemniej, należy zauważyć, iż znaczący wpływ na jakość wyników miała postać danych wejściowych. Rzeczona norma zakłada przetwarzanie zbiorów danych zawierających dobowe wartości błędu rekoncyliacji zbiornika w liczbie równej jednemu pełnemu miesiącowi. Co więcej, norma ta zakłada również, że wyciek, jeśli jest obecny, musi trwać przez cały okres objęty danym zbiorem. Powyższe wymagania przekładają się na kilka istotnych spostrzeżeń:

a) zakres czasu wynoszący jeden miesiąc jest zdecydowanie zbyt krótki – zjawiska zachodzące na stacjach paliw wymagają dłuższych okresów analizy;

b) dobowa rozdzielczość danych jest niewystarczająca, zwłaszcza w kontekście poprzedniego spostrzeżenia;

c) występowanie wycieku przez cały okres objęty zbiorem danych powoduje niemożność odróżnienia sytuacji normalnej od anomalnej;

d) analiza wyłącznie wartości błędu rekoncyliacji zbiornika jest niemiarodajna, gdyż nie uwzględnia innych zjawisk, znacząco wpływających na wartości błędu rekoncyliacji.

Szczególną uwagę należy poświęcić ostatniemu spostrzeżeniu, związanemu z nieuwzględnianiem innych miar niż błąd rekoncyliacji zbiornika. Z obserwacji dokonanych podczas eksperymentów jasno wynika, iż wpływ zmian temperatury po dostawach paliwa na wiarygodność rekoncyliacji zbiornika był istotny – stan nieustalony trwał przeważnie od 2 do 3 dni. W przypadku zbiorów danych, w którym dostawy występowały kilka razy w ciągu miesiąca, nawet jedna trzecia pomiarów przypadała na okres, w którym rozszerzalność temperaturowa cieczy dominowała nad innymi zjawiskami (w tym nad ewentualnym wyciekiem), co przekładało się na zupełne ukrycie tychże. W konsekwencji zbiory takie były zazwyczaj klasyfikowane przez algorytm TUBE jako niejednoznaczne.

Owa niejednoznaczność jest częścią szerszego problemu. Kolejnym zjawiskiem, które ma znaczący wpływ na jakość wykrywania wycieków paliwa jest niepoprawna kalibracja urządzeń pomiarowych, a w szczególności sond

pływakowych mierzących objętość paliwa w zbiornikach [38, 72, 70]. Skutkiem rozbieżności w tablicach kalibracyjnych tych urządzeń jest okresowe przekłamywanie (w pewnych zakresach wysokości sondy) pomiarów objętości paliwa, co przekłada się na pozorne straty lub zyski przy rekoncyliacji zbiornika. Te pierwsze potrafią do złudzenia przypominać początek wycieku paliwa, szczególnie gdy zakres wysokości, w którym przekłamanie występuje, jest niezbyt często osiągany przez paliwo.

Podążając dalej tym tokiem rozumowania, można dojść do wniosku, iż wszelkie zjawiska anomalne, które występują w sieci stacji paliw, mogą nakładać się na siebie w czasie i przez to wpływać na wartości rejestrowanych pomiarów. Wniosek ten z kolei prowadzi do konkluzji, iż nie jest możliwe analizowanie pojedynczego zjawiska anomalnego w oderwaniu od innych, bez uwzględnienia ich współwystępowania i jednoczesne uzyskanie wiarygodnych i jednoznacznych wyników analizy. W przypadku wycieków paliwa, uzyskanie satysfakcjonujących wyników analizy wartości błędu rekoncyliacji zbiornika jest możliwe dopiero po wzięciu pod uwagę kontekstu występowania spadków tej metryki. Kontekst ten stanowią wymienione uprzednio zjawiska, a przede wszystkim: rozszerzalność temperaturowa oraz niepoprawna kalibracja urządzeń pomiarowych, a także różne inne, pomniejsze anomalie rzeczywiste.

Rozdział 5

Teoretyczne podstawy analizy danych kontekstowych

Niniejszy rozdział omawia teoretyczne podstawy analizy danych kontekstowych. Rozdział rozpoczyna się od sformułowania definicji kontekstu, a następnie przedstawia klasyfikację danych kontekstowych, z podziałem na trzy zasadnicze grupy. W dalszej części opisane zostały modele danych kontekstowych, z uwzględnieniem różnej liczby wymiarów analizy. Rozdział kończy się wprowadzeniem hierarchii poziomów kontekstowości – dla każdego z nich zostaje przedstawiona krótka charakterystyka wykonywanych tam operacji i powiązanych z nimi typów danych.

5.1 Pojęcie kontekstu w analizie danych

Przedstawione w rozdziale 4 wyniki badań pozwalają skłonić się ku stwierdzeniu, iż analiza danych wyłącznie bieżących cechować się będzie pewną dozą niedokładności, implikowaną przez ich wieloznaczność. Taki stan rzeczy wynika z braku uwzględnienia wielu czynników znajdujących się poza obszarem analizy – ze względu na ich historyczność, odległość geograficzną, względnie brak powiązania tematycznego.

Podobnie jak w przypadku przetwarzania języka naturalnego, w którym wszechobecna jest wieloznaczność, tak i w przypadku analizy danych bieżących, konieczna jest interpretacja uzyskanych wyników w świetle przyjętego *kontekstu*. Kontekst ów stanowi zbiór dodatkowych informacji, które nadają jednoznacznego wydźwięku wypowiedzianym słowom (posługując się ponownie analogią do języka naturalnego) lub wynikom analizy danych bieżących. Powyższe stwierdzenia, powstałe na kanwie wniosków z badań przedstawionych w rozdziale 4, stanowią zarazem dowód tezy 1.

Kontekst w sposób intuicyjny może zostać zdefiniowany jako zbiór informacji stanowiących tło dla innych informacji. Słownik języka polskiego PWN (w swoim internetowym wydaniu[1]) podaje cztery definicje kontekstu:

a) „fragment tekstu potrzebny do dokładnego rozumienia danych wyrazów lub wyrażeń";
b) „zespół czynników współistniejących, powiązanych z czymś";
c) „zespół jednostek językowych, które stanowią otoczenie danej jednostki";
d) „zespół odniesień niezbędnych do zrozumienia utworu literackiego, dzieła naukowego".

Powyższe definicje w przeważającej większości (poza jedną), odnoszą się do rozumienia i interpretacji języka naturalnego lub wręcz dzieła literackiego. Mają one natomiast jedną cechę wspólną – określają kontekst jako zbiór powiązanych informacji, niezbędnych do jednoznacznej interpretacji danego zjawiska i współistniejących z nim. Na potrzeby niniejszej rozprawy przyjmuje się zatem takie właśnie znaczenie tego słowa.

W odniesieniu do danych przetwarzanych i gromadzonych w systemach informatycznych, posługiwać się należy pojęciem *danych kontekstowych* – danych, w których ów kontekst się zawiera. Proponuje się zatem następującą definicję:

Definicja 5.1 *Dane kontekstowe to wszystkie dane, które współistnieją w czasie lub przestrzeni, są powiązane bezpośrednio lub pośrednio oraz stanowią otoczenie w ujęciu geograficznym lub semantycznym dla innych danych, będących przedmiotem zasadniczej analizy i tym samym umożliwiają jednoznaczną interpretację jej wyników.*

W sposób formalny, wykorzystanie danych kontekstowych w analizie można przedstawić następującym modelem:

$$\text{interpretacja}\begin{pmatrix}\text{dane wieloznaczne}\\ \text{dane kontekstowe}\end{pmatrix} = \text{informacja jednoznaczna.}$$

Implementacja tego modelu dla systemów zorientowanych na wykrywanie anomalii przedstawia się następująco:

$$\text{interpretacja}\begin{pmatrix}\text{dane krytyczne}\\ \text{dane kontekstowe}\end{pmatrix} = \text{informacja wiarygodna.}$$

Powyższy zapis wprowadza pojęcie *danych krytycznych* w miejsce danych wieloznacznych. Jest tak ze względu na ich znaczenie przy analizie ukierunkowanej na wykrywanie sytuacji niepożądanych. W dalszym ciągu, są one danymi wieloznacznymi, a interpretacja wyników ich analizy wymaga uwiarygodnienia. Wprowadza się następującą definicję danych krytycznych:

[1] https://sjp.pwn.pl/

Definicja 5.2 *Dane krytyczne to najświeższe dane, pochodzące wprost z najbliższego otoczenia badanego zjawiska i bezpośrednio z nim powiązane tematycznie, których przetwarzanie jest szybkie, lecz dostarcza wyniki obarczone pewną niedokładnością, wynikającą z ich wieloznaczności.*

Analiza danych krytycznych, wsparta analizą danych kontekstowych jest głównym zadaniem strumieniowej hurtowni danych kontekstowych, która w szczegółach zostanie omówiona w rozdziale 6, stanowiąc zarazem dowód tezy 2.

5.2 Klasyfikacja danych kontekstowych

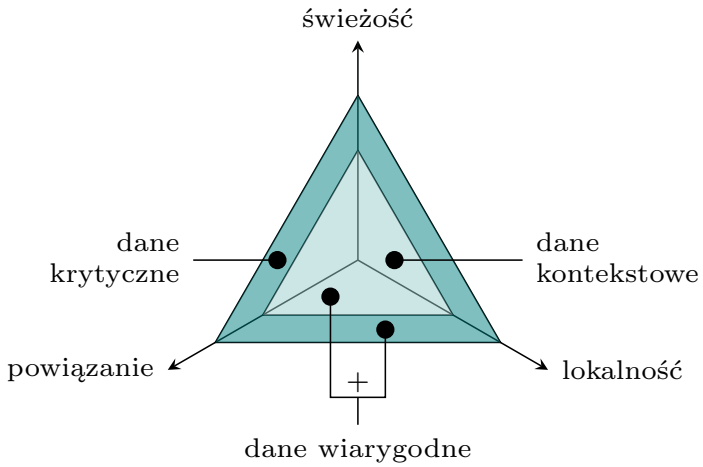
Analiza danych kontekstowych może być przeprowadzana pod różnymi aspektami, które wynikają ze stopnia zaawansowania samej analizy, jak i istotności jej wyników. Wyróżnić można trzy najważniejsze typy kontekstu, które stanowią zarazem trzy podstawowe osie analizy, związane z występowaniem i ze znaczeniem badanego zjawiska na przestrzeni czasu, na danym obszarze oraz we współistnieniu z innymi zjawiskami:

a) kontekst czasowy,
b) kontekst przestrzenny,
c) kontekst środowiskowy.

Analiza kontekstu czasowego ukierunkowana jest na badanie kompletnej historii występowania jakiegoś zjawiska. Przykładem analizy kontekstu czasowego jest sprawdzanie powtarzalności występowania nietypowego wzorca na przestrzeni czasu. W założeniu analiza taka ogranicza się ona do danych lokalnych, pochodzących z pojedynczego źródła oraz bezpośrednio związanych z badanym zjawiskiem.

Analiza kontekstu przestrzennego wiąże się z uwzględnieniem lokalizacji źródeł danych określonego typu. Przedmiotem analizy jest tutaj badanie zasięgu danego zjawiska lub powtarzalności występowania danego wzorca w bliższym bądź dalszym sąsiedztwie geograficznym. Założeniem takiej analizy jest zgodność czasowa (w pewnym przedziale wartości) przetwarzanych danych pochodzących z różnych źródeł.

Analiza kontekstu środowiskowego przeprowadzana jest w celu zbadania wzajemnego wpływu na siebie różnych, potencjalnie niepowiązanych ze sobą zjawisk. Zjawiska takie współistnieją w czasie i przestrzeni i przenikają się wzajemnie, co może się objawiać zarówno wzmocnieniem, jak i wygaszeniem pewnych cech. Analiza takiego wpływu może dostarczyć istotnych informacji przekładających się na wiarygodność wyników. Założyć należy, iż dane opisujące różne zjawiska są zgodne zarówno w czasie, jak i przestrzeni (w pewnych przedziałach wartości).



Rysunek 5.1: Wizualizacja danych kontekstowych w układzie 3 osi: świeżości, lokalności i powiązania

Rysunek 5.1 przedstawia obrazowe umieszczenie danych kontekstowych w układzie trzech osi, związanych z wymienionymi wcześniej typami kontekstu. Pierwsza z nich związana jest ze świeżością danych i wyznacza podział na dane historyczne (bliżej środka) i bieżące (bliżej grotu strzałki). Druga oś wyraża lokalność danych – w środku układu współrzędnych znajdują się dane globalne, a wraz z oddalaniem się od tego punktu zwiększa się ich regionalizacja. Ostatnia oś przekłada się na powiązanie z tematyką analizy – począwszy od danych pośrednio powiązanych (w środku), a skończywszy na danych bezpośrednio powiązanych (na zewnątrz).

Zewnętrzny obszar przedstawiony na rysunku 5.1 odpowiada danym wieloznacznym. Ich analiza daje wyniki obarczone niedokładnościami, gdyż jedynie powierzchownie dotyka ona tematu. Z drugiej strony, przetwarzanie danych najbardziej świeżych, pochodzących jedynie z lokalnego otoczenia badanego zjawiska oraz bezpośrednio z nim tematycznie powiązanych jest najbardziej efektywna czasowo i nie wymaga zaawansowanych metod analizy. Odwrotnie sytuacja przedstawia się dla drugiego, wewnętrznego obszaru ukazanego na tym samym rysunku. Obszar ten odpowiada danym kontekstowym, które nie zawsze są najnowsze (dane historyczne), pochodzą z różnych źródeł (dane globalne) oraz są mniej lub bardziej pośrednio powiązane z przedmiotem analizy (dane poboczne). Ich przetwarzanie jest czasochłonne, lecz umożliwia uwiarygodnienie wyników uzyskanych na drodze analizy powierzchownej.

5.3 Modele danych kontekstowych

Przez pojęcie modelu danych kontekstowych należy rozumieć zbiór formalnych definicji określających istotę oraz charakterystykę przetwarzania tych danych. W zależności od typu kontekstu (czasowy, przestrzenny, środowiskowy), można wyróżnić trzy modele danych kontekstowych, które różnią się między sobą wymiarowością.

5.3.1 Jednowymiarowy model danych kontekstowych czasowych

Dane kontekstowe czasowe mają postać strumieni danych, czyli długich sekwencji rekordów uporządkowanych w czasie. Strumienie danych są z natury jednowymiarowe – czas jest jedynym atrybutem, w kontekście którego przeprowadzana jest ich analiza w większości zastosowań. Definicja formalna strumienia danych ilustruje powyższe rozważania:

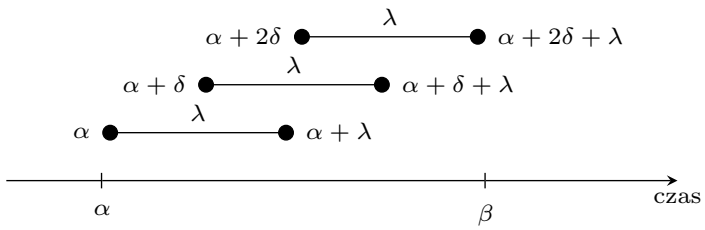
Definicja 5.3 *Strumień danych Σ jest uporządkowaną dwójką (S, A), gdzie $S = (r_1, r_2, \ldots, r_n)$ jest nieskończoną sekwencją rekordów r_i o stałym schemacie $A = (t_0, t_1, a_1, a_2, \ldots, a_m)$ będącym ciągiem atrybutów, w którym można wyróżnić dwa znaczniki czasowe definiujące przedział czasu, w którym powstał dany rekord lub w którym występowało opisywane przezeń zjawisko.*

W celu uzyskania wartości na nieco wyższym poziomie ogólności w stosunku do danych znajdujących się w strumieniu, przeprowadza się proces agregacji czasowej. Działanie takie odbywa się w ustalonych oknach (przedziałach) czasowych – w wyniku powstaje strumień agregatów czasowych. W sposób formalny, okno czasowe można zdefiniować następująco:

Definicja 5.4 *Okno czasowe $\omega = (\alpha, \lambda)$ to uporządkowana para wartości, która w sposób jednoznaczny definiuje przedział czasu o początku w α i długości równej λ.*

Zazwyczaj zapytania agregujące definiują wiele okien wraz ze schematem ich przemieszczania się wzdłuż wymiaru czasu. Możliwe są *okna stałe*, wyznaczone pomiędzy punktami w czasie oddalonymi od siebie o stałą wartość, jak również *okna przesuwne*, w których poszczególne przedziały zachodzą na siebie. Pojęciem łączącym oba te aspekty, czyli wielkość przedziału oraz sposób jego przemieszczania się, jest *wielokrotne okno czasowe*. W sposób formalny zostało ono określone w definicji:

Definicja 5.5 *Wielokrotne okno czasowe $\Omega = (\omega, \delta, \beta)$ to trójka wartości, która łączy okno czasowe ω ze sposobem jego propagacji wzdłuż wymiaru czasu, o kroku δ, przy czym liczba przedziałów n jest określona przez wartość końcową β, taką że $\beta = \alpha + (n-1) \cdot \delta + \lambda$.*



Rysunek 5.2: Wizualizacja wielokrotnego okna czasowego

Z definicji 5.5 wynika, że wielokrotne okno czasowe to zbiór przedziałów czasu, które mogą na siebie nachodzić, przylegać do siebie lub być rozłączne względem siebie. Brak jawnego podania liczby przedziałów w tej definicji wynika z faktu, iż zazwyczaj zapytania określają wartość początkową i końcową (klauzula `WHERE` w języku SQL) oraz rozmiar przedziału (klauzula `GROUP BY` w języku SQL).

Rysunek 5.2 przedstawia wizualizację wielokrotnego okna czasowego o początku w α i końcu w β oraz o kroku równym δ i długości λ. W przypadku, w którym $\lambda = \delta$ okno jest oknem stałym, a dla $\delta < \lambda$ oknem przesuwnym.

5.3.2 Dwuwymiarowy model danych kontekstowych przestrzennych

Dane kontekstowe przestrzenne mają postać zbiorów agregatów czasowych, które reprezentują określony przedział czasu oraz powstały w wybranym obszarze przestrzeni. Zbiory te nazywane będą dalej *klastrami*. Rekordy wchodzące w skład klastra mają swoje fizyczne położenie (związane z lokalizacją ich źródeł), które może być jednoznacznie określone przez współrzędne geograficzne lub inne, w zależności od przyjętego układu odniesienia. Najczęściej współrzędne te odnoszą się do przestrzeni dwuwymiarowych (długość i szerokość geograficzna) lub trójwymiarowych (długość, szerokość i wysokość). Na potrzeby dalszych rozważań przyjmuje się dwuwymiarowy scenariusz, wychodząc z założenia, iż przypadek trójwymiarowy jest rozbudowaniem dwuwymiarowego o jeden dodatkowy wymiar. Definicja klastra przedstawia się w sposób następujący:

Definicja 5.6 *Klaster danych C jest uporządkowaną czwórką (R, A, T, L), gdzie $R = \{r_1, r_2, \ldots, r_n\}$ jest zbiorem agregatów czasowych o stałym schemacie $A = (t_0, t_1, x_0, x_1, y_0, y_1, a_1, a_2, \ldots, a_m)$ będącym ciągiem atrybutów, w którym wyróżnia się dwa znaczniki czasowe definiujące zakres danego agregatu oraz dwie pary współrzędnych określających położenie źródła; klaster dodatkowo określany jest przez przedział czasu $T = \langle T_0, T_1 \rangle$ tożsamy z przedziałami czasu poszczególnych rekordów oraz obszar przestrzeni, na którym umiejscowione są źródła: $L = (\langle X_0, X_1 \rangle, \langle Y_0, Y_1 \rangle)$.*

Ciąg kolejnych klastrów, powstałych w następujących po sobie przedziałach czasu, można potraktować jako *unię strumieni*, która, w przeciwieństwie do zwykłego strumienia, składa się z klastrów, a nie rekordów. Ilustruje to następująca definicja:

Definicja 5.7 *Unia strumieni* Υ *jest uporządkowaną trójką* (S, A, L)*, gdzie* S *jest nieskończoną sekwencją klastrów danych:* $S = (C_1, C_2, \ldots, C_n)$ *o stałym schemacie* $A = (t_0, t_1, x, y, a_1, a_2, \ldots, a_m)$ *będącym ciągiem atrybutów (tożsamym ze schematem poszczególnych klastrów), a* $L = (\langle X_0, X_1 \rangle, \langle Y_0, Y_1 \rangle)$ *jest obszarem przestrzeni, z którego dane znajdują się w unii.*

W celu uzyskania wartości zbiorczych, reprezentujących większe obszary, w których rejestrowane są poszczególne rekordy, przeprowadza się agregację przestrzenną – w wyniku powstaje zagregowana unia strumieni. Poprzez analogię do definicji 5.4, wprowadzić można pojęcie okna przestrzennego, będącego dwuwymiarowym odpowiednikiem okna czasowego:

Definicja 5.8 *Okno przestrzenne:* $\omega^2 = (\vec{\alpha}, \vec{\lambda})$ *to uporządkowana para wektorów dwuelementowych, która w jednoznaczny sposób definiuje podzbiór przestrzeni dwuwymiarowej o początku określonym za pomocą wektora* $\vec{\alpha} = [\alpha_1, \alpha_2]$ *oraz długościach określonych przez wektor* $\vec{\lambda} = [\lambda_1, \lambda_2]$*.*

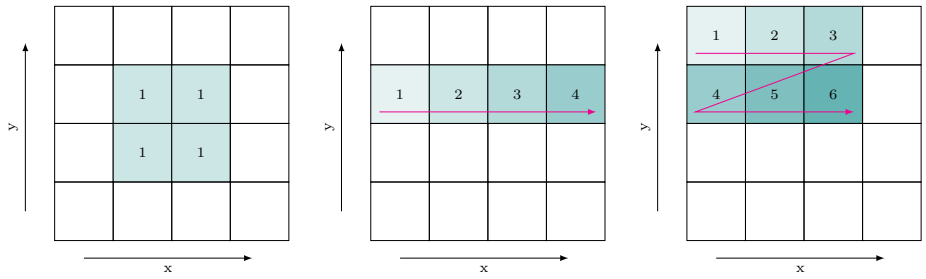
Podobnie do okna czasowego, okno przestrzenne może być zdefiniowane w sposób wielokrotny, a więc przesuwać się wzdłuż obu wymiarów przestrzeni. W rezultacie tworzona jest siatka okien przestrzennych, zwana (przez analogię do definicji 5.5) wielokrotnym oknem przestrzennym:

Definicja 5.9 *Wielokrotne okno przestrzenne* $\Omega^2 = (\omega^2, \vec{\delta}, \vec{\beta})$ *to trójka wartości, która łączy okno przestrzenne* ω^2*, określone na pewnej dwuwymiarowej przestrzeni, ze sposobem jego propagacji wzdłuż obu jej wymiarów, przy czym krok dla każdego wymiaru dany jest jako dwuelementowy wektor* $\vec{\delta} = [\delta_1, \delta_2]$*, a liczba przedziałów dla każdego wymiaru wymiaru jest określona przez dwuelementowy wektor wartości końcowych* $\vec{\beta} = [\beta_1, \beta_2]$*, takich że dla wymiaru* i *o* n_i *przedziałach:* $\beta_i = \alpha_i + (n_i - 1) \cdot \delta_i + \lambda_i$*.*

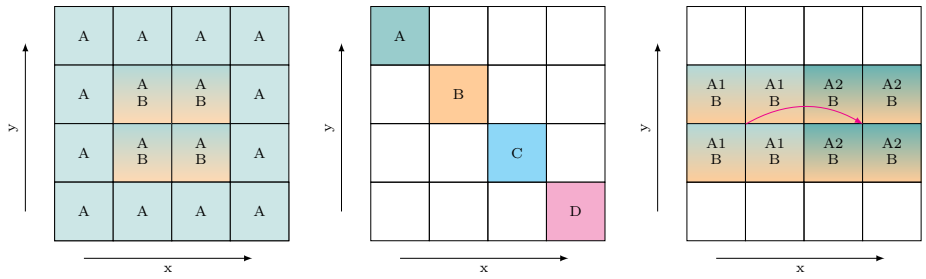
Rysunek 5.3 przedstawia sześć przykładów zapytań przestrzennych w pewnej dwuwymiarowej przestrzeni (x, y). Pierwsza połowa przykładów (rysunki 5.3a, 5.3b, 5.3c) to zapytania proste, składające się tylko z jednego wielokrotnego okna przestrzennego, podczas gdy druga połowa (rysunki 5.3d, 5.3e, 5.3f) to zapytania złożone, w skład których wchodzi kilka wielokrotnych okien przestrzennych.

Zastosowana na rysunku 5.3 siatka kwadratów dzieli każdy z wymiarów na elementarne jednostki (u, od ang. *unit*). Kolorowymi kwadratami oznaczono kolejne położenia okien przestrzennych – odpowiada im numeracja

przyjęta w obrębie każdego rysunku. W przypadku zapytań prostych numerami oraz gradacją koloru oznaczono kolejne położenia wielokrotnego okna przestrzennego, a w przypadku zapytań złożonych – literami alfabetu oraz różnymi kolorami oznaczono kolejne wielokrotne okna przestrzenne.



(a) studium przypadku (b) zmiana w wymiarze (c) rozkład cechy



(d) zawężanie obszaru (e) ścieżka (f) operacja *roll-up*

Rysunek 5.3: Przykłady zapytań przestrzennych: prostych (a-c) i złożonych (d-f)

Pierwszy przypadek (rysunek 5.3a) przedstawia zapytanie definiujące tylko jedno okno przestrzenne (obejmuje ona po dwie jednostki dla każdego wymiaru). Tego typu zapytanie może być wykorzystane w celu studium konkretnego przypadku – jego wynik niesie ze sobą informację na temat pewnego zjawiska w konkretnym obszarze agregowanej przestrzeni.

Drugie przypadek (rysunek 5.3b) prezentuje zapytanie definiujące wielokrotne okno przestrzenne o rozmiarze równym pojedynczej jednostce kwadratowej ($1\,u^2$) i przesuwające się wzdłuż wymiaru x o jedną jednostkę ($1\,u$). Intencją użytkownika może być zbadanie zmiany pewnej cechy w miarę przemieszczania się wzdłuż wymiaru x.

Trzeci przypadek (rysunek 5.3c) prezentuje przykład zapytania, w którym wielokrotne okno przestrzenne przesuwa się w dwóch wymiarach jednocześnie. W rezultacie powstaje sześć okien przestrzennych o rozmiarze $1\,u^2$, w których przeprowadzana jest agregacja. Zapytanie tego typu dostarcza informacji o dwuwymiarowym rozkładzie badanej cechy, a jego wynik ma postać „mapy ciepła” (ang. *heat map*).

Czwarty przypadek (rysunek 5.3d) przedstawia przykład zapytania powstałego na skutek unii (sumy zbiorów) dwóch wielokrotnych okien wielowymiarowych. Pierwsze z nich obejmuje całą przestrzeń ($16\,u^2$), podczas gdy drugie zawęża ją do centralnego obszaru ($4\,u^2$). Wynik takiego zapytania informuje o globalności badanej cechy i jej zmianie wraz ze zmniejszaniem się obszaru poszukiwania.

Piąty przypadek (rysunek 5.3e) pokazuje podobną sytuację do tej przedstawionej na rysunku 5.3e, aczkolwiek kierunek przesuwania się okna nie jest tożsamy z przemieszczaniem się wzdłuż tylko jednego wymiaru, a raczej z przemierzaniem pewnej ścieżki w przestrzeni. Zapytanie takie jest unią czterech okien przestrzennych. Przykładem zastosowania tego typu zapytania jest chęć zdobycia wiedzy o zmieniających się warunkach w różnych obszarach przestrzeni.

Szósty przypadek (rysunek 5.3f) przedstawia kolejne zapytanie powstałe w wyniku unii dwóch wielokrotnych okien przestrzennych. Pierwsze z nich ma rozmiar równy $4\,u^2$ i przemieszcza wzdłuż wymiaru x o $2\,u$, podczas gdy drugie obejmuje cały zakres wartości tegoż wymiaru. Tego typu zapytanie informuje użytkownika o wartościach agregatów na kilku różnych poziomach granulacji danych i odpowiada operacji *roll-up*.

5.3.3 Wielowymiarowy model danych kontekstowych środowiskowych

Dane środowiskowe mają postać zbiorów agregatów czasowo-przestrzennych, współistniejących w czasie i przestrzeni. Zbiory takie będą nazywane *blokami* i mogą być utożsamiane z wycinkami czasoprzestrzeni. W przeciwieństwie do klastra, w którym znajdują się agregaty powstałe w tym samym przedziale czasu, blok obejmuje agregaty opisujące wiele przedziałów zarówno czasu, jak i przestrzeni. W sposób formalny blok danych można zdefiniować jako:

Definicja 5.10 *Blok danych B jest uporządkowaną czwórką (R, A, T, L), gdzie $R = \{r_1, r_2, \ldots, r_n\}$ jest zbiorem agregatów czasowo-przestrzennych o stałym schemacie $A = (t_0, t_1, x_0, x_1, y_0, y_1, a_1, a_2, \ldots, a_m)$ będącym ciągiem atrybutów, w którym wyróżnia się dwa znaczniki czasowe definiujące zakres czasowy danego agregatu oraz dwie pary współrzędnych określających obszar jego przestrzeni; dodatkowo blok określany jest przez przedział czasu $T = \langle T_0, T_1\rangle$ oraz obszar przestrzeni $L = (\langle X_0, X_1\rangle, \langle Y_0, Y_1\rangle)$, w których poszczególne agregaty się zawierają.*

Agregaty czasowo-przestrzenne wchodzące w skład bloku mogą pochodzić bezpośrednio z unii strumieni, bądź też powstać na skutek złączenia więcej niż jednej z nich. Definicja tego działania (złączenia unii) przedstawia się następująco:

Definicja 5.11 *Złączenie dwóch unii strumieni o różnych schematach jest operacją polegającą na sukcesywnym dopasowywaniu do siebie rekordów pochodzących z obu unii i tworzeniu rekordów o schemacie będącym sumą (lub podzbiorem tejże) schematów łączonych unii; dopasowywanie owo związane jest ze współistnieniem w czasie i przestrzeni, rozumiane jako przynależność do wspólnego przedziału czasu oraz obszaru przestrzeni.*

Z definicji 5.11 wynika, iż złączenie dwóch rekordów jest możliwe tylko wtedy, gdy oba rekordy są zgodne w czasie i przestrzeni, czyli opisują różne aspekty tego samego zjawiska. Zgodność ta jest wymuszona przez agregację w wymiarze czasu i przestrzeni. Ciąg bloków danych charakteryzowanych przez występujące po sobie przedziały czasu tworzy *strumień syntetyczny*:

Definicja 5.12 *Strumień syntetyczny jest uporządkowaną trójką* (S, A, L)*, gdzie* S *jest nieskończoną sekwencją bloków danych:* $S = (B_1, B_2, \ldots, B_n)$ *o stałym schemacie* $A = (t_0, t_1, x_0, x_1, y_0, y_1, a_1, a_2, \ldots, a_m)$ *będącym ciągiem atrybutów, w którym wyróżnia się dwa znaczniki czasowe definiujące zakresy czasowe poszczególnych bloków oraz dwie pary współrzędnych określających obszar ich przestrzeni; dodatkowo strumień syntetyczny określany jest przez obszar przestrzeni* $L = (\langle X_0, X_1\rangle, \langle Y_0, Y_1\rangle)$*, który jest tożsamy z obszarem każdego z bloków.*

Zasadniczą rolą bloków danych, jako elementów składowych strumieni syntetycznych, jest ułatwienie analizy wielowymiarowej, wymagającej agregacji wzdłuż atrybutów innych niż czas i przestrzeń. W takim rozumieniu, każdy rekord można potraktować jako punkt lub hiperprostopadłościan (w przypadku przedziałów wartości) pewnej wielowymiarowej przestrzeni o liczbie wymiarów równej liczbie atrybutów schematu strumienia. Biorąc na wzgląd powyższy tok rozumowania, można wprowadzić definicję wielowymiarowej przestrzeni atrybutów:

Definicja 5.13 *Wielowymiarowa przestrzeń atrybutów* $\aleph_d$ *o* d *wymiarach jest iloczynem kartezjańskim zbiorów wartości atrybutów należących do pewnego schematu:* $\aleph_d = A_1 \times A_2 \times \ldots A_d$*.*

Z definicji 5.13 wprost wynika, że każdemu atrybutowi odpowiada jeden wymiar przestrzeni. Równocześnie jednak, każdy z atrybutów może być potraktowany jako miara i być przez to przedmiotem analizy. W wielu przypadkach dany atrybut jest traktowany raz jak miara, a raz jak wymiar. Powyższe założenie stanowi odejście od sztywnego podziału na wymiary i miary, stosowanego w klasycznych modelach hurtowni danych.

W celu uzyskania zbiorczych wartości reprezentujących określone obszary (podzbiory) wielowymiarowej przestrzeni atrybutów, przeprowadza się agregację wielowymiarową. Przedziały określone w zapytaniu wielowymiarowym przypominają okna czasowe, lecz mogą być zdefiniowane na dowolnych

wymiarach przestrzeni atrybutów. Możliwe jest zatem uogólnienie definicji 5.4 na dowolny wymiar:

Definicja 5.14 *Okno uogólnione dla wymiaru i:* $\omega_i = (\alpha_i, \lambda_i)$ *to uporządkowana para wartości, która w sposób jednoznaczny definiuje przedział wartości wymiaru i o początku w* α_i *i długości równej* λ_i*.*

Iloczyn kartezjański okien uogólnionych dla poszczególnych wymiarów przestrzeni atrybutów stanowi zarazem okno wielowymiarowe. Jest ono odpowiednikiem okna czasowego w przestrzeni jednowymiarowej oraz okna przestrzennego w przestrzeni dwuwymiarowej. Z formalnego punktu widzenia, definicja okna wielowymiarowego przedstawia się w sposób następujący:

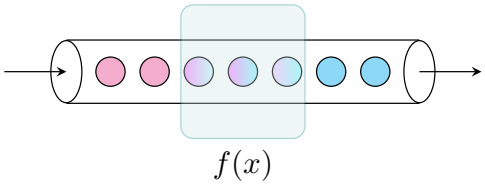
Definicja 5.15 *Okno wielowymiarowe* $\omega^d = \omega_1 \times \omega_2 \times \ldots \times \omega_d$ *o d wymiarach jest iloczynem kartezjańskim okien uogólnionych dla d wymiarów wielowymiarowej przestrzeni atrybutów* $\aleph_d$*.*

Zastosowanie wielokrotnych okien uogólnionych na wybranym podzbiorze wielowymiarowej przestrzeni atrybutów skutkuje jej podziałem na zbiór podprzestrzeni, których liczba jest równa iloczynowi liczby okien uogólnionych dla każdego z wymiarów. Można zatem wprowadzić definicję *wielokrotnego okna wielowymiarowego*, które łączy każde ze wspomnianych wielokrotnych okien uogólnionych:

Definicja 5.16 *Wielokrotne okno wielowymiarowe* $\Omega^d = (\omega^d, \vec{\delta}, \vec{\beta})$ *to trójka wartości, która łączy okno wielowymiarowe* ω^d*, określone na pewnej wielowymiarowej przestrzeni atrybutów* $\aleph_d$*, ze sposobem jego propagacji wzdłuż wszystkich jej d wymiarów, przy czym krok dla każdego wymiaru jest dany jako wektor* $\vec{\delta} = [\delta_1, \delta_2, \ldots, \delta_d]$*, a liczba przedziałów dla każdego wymiaru jest określona przez wektor wartości końcowych* $\vec{\beta} = [\beta_1, \beta_2, \ldots, \beta_d]$*, takich że dla wymiaru i o* n_i *przedziałach:* $\beta_i = \alpha_i + (n_i - 1) \cdot \delta_i + \lambda_i$*.*

Agregacja wielowymiarowa jest przeprowadzana w celu uzyskania odpowiedzi na zapytania o wzajemny wpływ wartości poszczególnych atrybutów na siebie. Kontynuując przyjęty tok rozumowania, agregaty powstałe w wyniku realizacji takiego zapytania tworzą hiperprostopadłościany w nowej przestrzeni o wymiarowości równej bądź mniejszej od wymiarowości agregowanej przestrzeni. W związku z tym, można podać formalną definicję zapytania wielowymiarowego:

Definicja 5.17 *Zapytanie wielowymiarowe* $Q_{d'} : \aleph_d \mapsto \aleph_{d'}$ *jest przekształceniem d wymiarowej przestrzeni atrybutów* $\aleph_d$ *w* d' *wymiarową przestrzeń atrybutów* $\aleph_{d'}$ *taką, że* $d' \leqslant d$*.*



Rysunek 5.4: Symboliczne przedstawienie zerowego poziomu kontekstowości

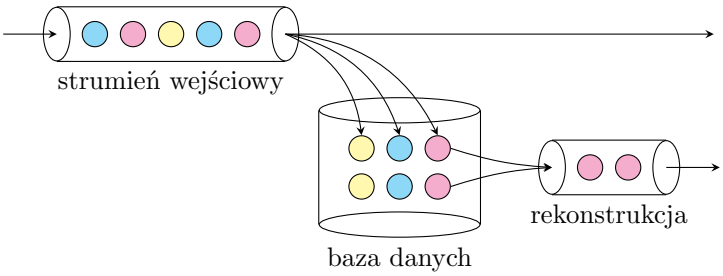
5.4 Hierarchia poziomów kontekstowości

Zaproponowany podział kontekstu na: czasowy, przestrzenny i środowiskowy został wprowadzony w celu ukazania gradacji metod przetwarzania danych kontekstowych. Możliwe jest uwzględnianie przykładowo tylko jednego lub dwóch typów kontekstu w analizie, jeśli jest to wystarczające z punktu widzenia wiarygodności uzyskiwanych wyników. W zależności od stopnia uwzględnienia danych kontekstowych oraz od złożoności przeprowadzanych operacji, można mówić o konkretnym *poziomie kontekstowości* analizy. Poziomy te tworzą model kaskadowy – oznacza to, że każdy kolejny poziom wykorzystuje dane i metody obecne na niższych poziomach. W zależności od uwzględnienia (bądź nie) operacji agregacji, trzy wyżej opisane typy kontekstu przekładają się na sześć poziomów kontekstowości. Do tego zbioru należy doliczyć jeszcze poziom zerowy, który związany jest z analizą bezkontekstową – przykładem takiej analizy jest przetwarzanie danych krytycznych.

5.4.1 Poziom 0: brak kontekstu

Zerowy poziom kontekstowości zakłada przetwarzanie danych w sposób strumieniowy, a więc z pominięciem procesu ich utrwalania. Na tym poziomie dominują zapytania ciągłe (ang. *Continuous Queries*) przetwarzające dane w oknach czasowych. Uzyskiwane wyniki dostarczają informacji o bieżącym stanie badanych zjawisk. Taka forma jest charakterystyczna dla systemów zarządzania strumieniami danych (DSMS – ang. *Data Stream Management System*). Poszczególne strumienie danych analizowane są osobno, bez uwzględnienia ewentualnych powiązań i wzajemnego wpływu na siebie zawartych w nich danych. W konsekwencji, wyniki uzyskiwane są szybko, lecz swoim zakresem obejmują wyłącznie najnowszą historię i lokalne otoczenie badanego zjawiska.

Rysunek 5.4 w sposób symboliczny przedstawia procesy zachodzące na zerowym poziomie konktekstowości. Ulotne dane ze strumienia wejściowego przechodzą przez sieć operatorów, które dokonują różnego rodzaju przekształceń – oznaczonych w umowny sposób jako $f(x)$. W wyniku tych operacji powstaje strumień wyjściowy, który zawiera dane w postaci interesującej dla użytkownika zlecającego przetwarzanie. Zagadnienie przetwarzania stru-



Rysunek 5.5: Symboliczne przedstawienie pierwszego poziomu kontekstowości

mieni danych bez uwzględniania kontekstu było przedmiotem wcześniejszych badań, stanowiących podstawę do opracowania teorii przetwarzania danych kontekstowych. Wyniki rzeczonych prac badawczych opisano w artykułach naukowych [66, 36, 37]^{W}.

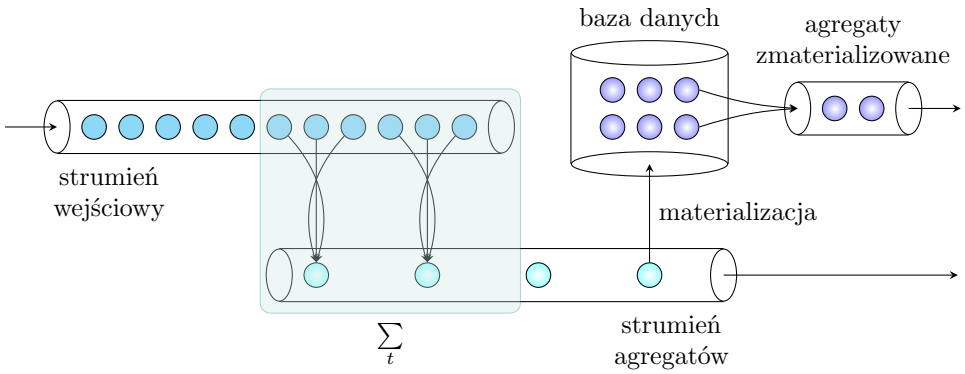
5.4.2 Poziom 1: kontekst czasowy

Pierwszy poziom kontekstowości charakteryzuje się składowaniem strumieni danych w pamięci trwałej. W wyniku tego procesu dostępna jest cała historia analizowanego zjawiska. Zarchiwizowane strumienie danych mogą zostać w każdej chwili odtworzone (zrekonstruowane), począwszy od dowolnego momentu w czasie. Operacje wykonywane na tym poziomie kontekstowości pokrywają się z działaniami dostępnymi w bazach szeregów czasowych (ang. *Time Series Databases*). Należy tutaj zaznaczyć, że rekonstrukcja dotyczy wyłącznie pojedynczych strumieni, a więc danych pochodzących z wybranego źródła, bez tworzenia złączeń oraz agregatów – celem jest wyłącznie odtworzenie historycznego strumienia danych. Wzbogaca to analizę danych o możliwość wglądu w historię badanego zjawiska, a co za tym idzie – uwzględnienie kontekstu czasowego w analizie.

Na rysunku 5.5 symbolicznie przedstawiono procesy zachodzące na pierwszym poziomie kontekstowości. Strumień wejściowy, który pochodzi wprost ze źródeł, podlega archiwizacji (utrwaleniu) w bazie danych. Kolorystycznie przedstawiono jakościowy podział danych w pamięci trwałej – jest on przeprowadzany na podstawie identyfikatorów źródeł oraz schematów poszczególnych strumieni. Zarchiwizowane dane mogą zostać wykorzystane do rekonstrukcji historycznego strumienia.

5.4.3 Poziom 2: kontekst czasowy z agregacją

Drugi poziom kontekstowości uzupełnia poprzedni o agregację w czasie (w oknach czasowych) strumieni danych. Dzięki temu, istnieje możliwość przeglądania danych historycznych z dowolnym stopniem szczegółowości.



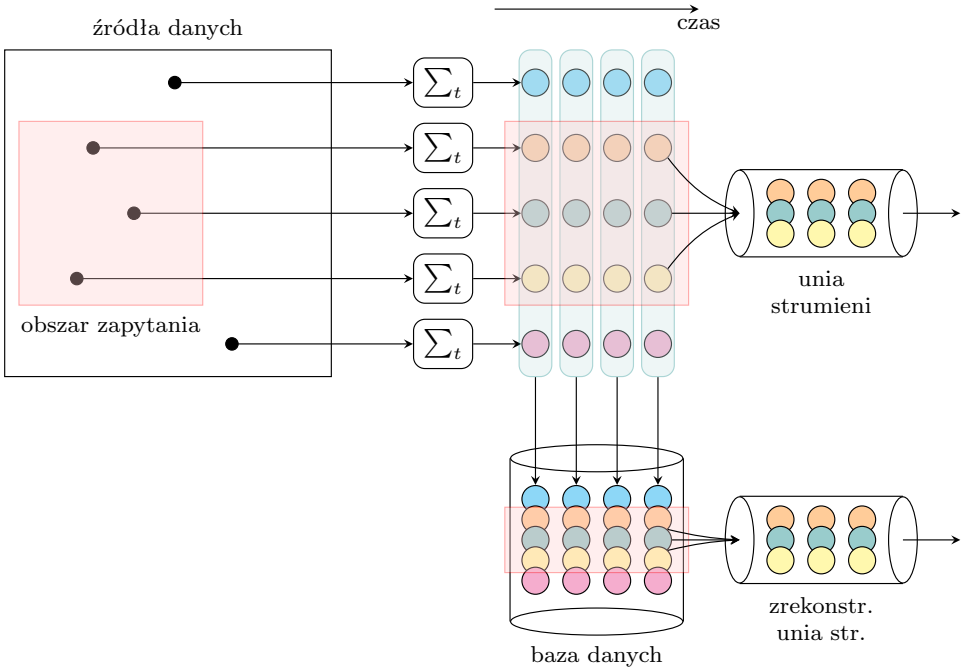
Rysunek 5.6: Symboliczne przedstawienie drugiego poziomu kontekstowości

Działanie takie rozszerza możliwości analizy kontekstu czasowego i wzbogaca ją o umiejętność generalizowania i przeglądania danych z pominięciem nieistotnych szczegółów. W dalszym ciągu zrekonstruowane i zagregowane dane pochodzą z pojedynczego źródła. W tym zakresie poziom ten jest zgodny z funkcjonalnościami baz danych szeregów czasowych. Dodatkowo istnieje możliwość materializacji obliczonych agregatów, w celu ich ponownego wykorzystania.

Rysunek 5.6 ilustruje w sposób symboliczny procesy zachodzące na drugim poziomie kontekstowości. Strumień wejściowy, który może pochodzić zarówno bezpośrednio ze źródeł, jak i powstać na drodze rekonstrukcji, podawany jest na wejście bloku agregacji w wymiarze czasu (oznaczonego w umowny sposób znakiem sumy). Poszczególne agregaty tworzone są na podstawie tych rekordów strumienia wejściowego, które należą do kolejnych okien czasowych (zgodnie z parametrami zapytania). Gotowe agregaty podlegają utrwaleniu w bazie danych (materializacji) i mogą w przyszłości zostać wykorzystane – czyni to zbędnym konieczność ich ponownego obliczania.

5.4.4 Poziom 3: kontekst przestrzenny

Trzeci poziom kontekstowości związany jest z zagadnieniem fizycznego rozmieszczenia źródeł danych (ich lokalizacji). Ma ono szczególne znaczenie przy analizie zasięgu występowania badanego zjawiska lub jego wpływu na inne, znajdujące się w pobliżu. Podstawową operacją wykonywaną na tym poziomie jest przeszukiwanie obszarów i znajdowanie objętych ich zakresem źródeł oraz ekstrakcja strumieni z nich pochodzących. Uzyskiwane dane stanowią unię kilku strumieni (definicja 5.7), które pod względem czasu są zagregowane w przyjętych minimalnych przedziałach czasu – działanie takie gwarantuje niejako „wyrównanie w czasie" poszczególnych rekordów. Na tym poziomie kontekstowości wykorzystywane są różnego rodzaju metody



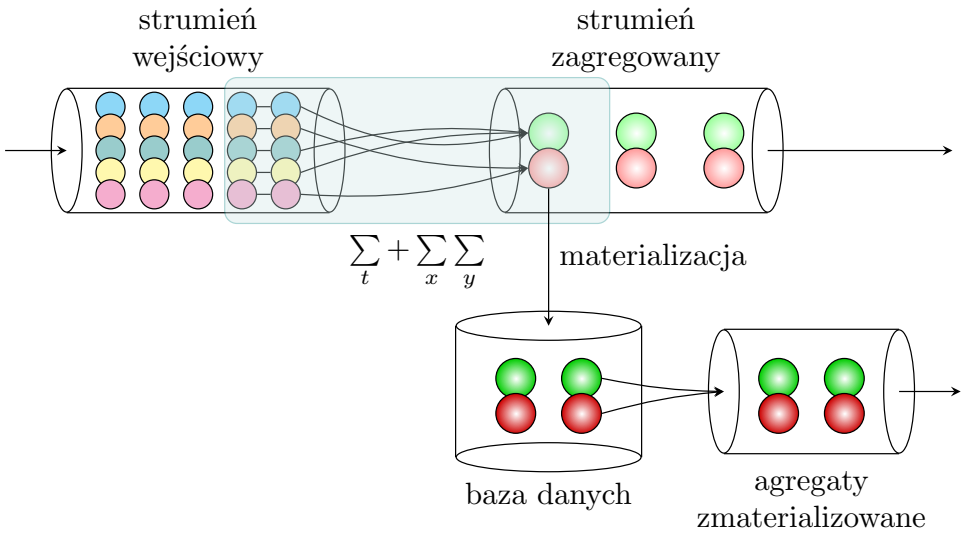
Rysunek 5.7: Symboliczne przedstawienie trzeciego poziomu kontekstowości

znane z baz przestrzennych i geograficznych.

Na rysunku 5.7 w sposób symboliczny przedstawiono procesy zachodzące na trzecim poziomie kontekstowości. Źródła danych zostały przedstawione po lewej stronie rysunku, rozmieszczone arbitralnie w przestrzeni dwuwymiarowej. Strumienie danych pochodzące z tych źródeł podlegają wstępnej agregacji w czasie w celu ich wyrównania, a następnie są zapisywane w bazie danych. Zapis ten charakteryzuje się uporządkowaniem poszczególnych rekordów względem ich znaczników czasowych (wyrównanych w procesie agregacji) – w ten sposób rekordy powstałe w tym samym czasie są zapisywane razem, w formie strony, przy czym strona taka zawiera dane pochodzące z wielu źródeł. Dodatkowo na rysunku uwzględniono obszar zapytania, zaznaczony czerwonym prostokątem. Obejmuje on pewien podzbiór źródeł i służy do wyekstrahowania danych wygenerowanych z tego podzbioru, co ma miejsce zarówno dla danych strumieniowych, jak i danych historycznych.

5.4.5 Poziom 4: kontekst przestrzenny z agregacją

Czwarty poziom kontekstowości rozszerza poprzedni o agregację w przestrzeni, która jest przeprowadzana w zadanych obszarach (oknach przestrzennych). Wzbogaca to analizę kontekstu przestrzennego o możliwość wyciągnięcia zbiorczych wartości, reprezentujących większe obszary przestrzeni. Warunkiem koniecznym przy agregacji przestrzennej jest wcześniej-



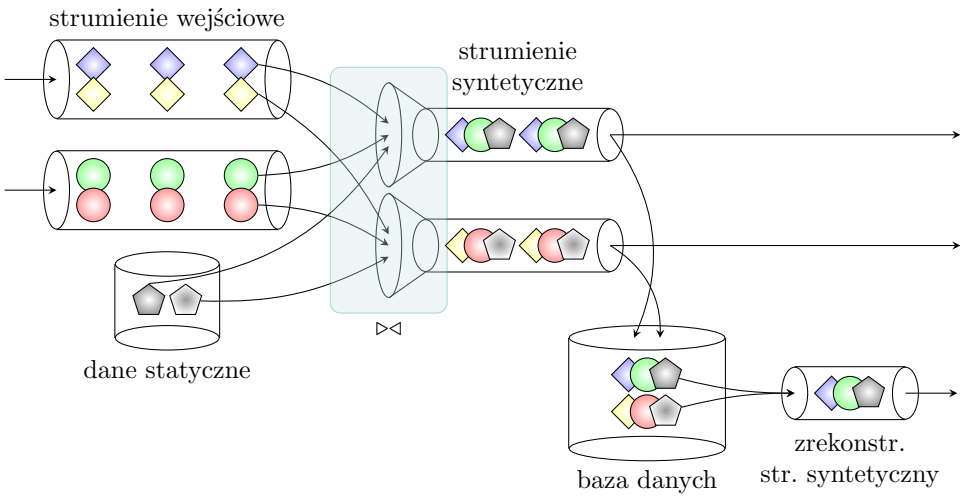
Rysunek 5.8: Symboliczne przedstawienie czwartego poziomu kontekstowości

sza agregacja w wymiarze czasu, w celu wyrównania przedziałów czasowych agregowanych przestrzennie danych. W wyniku powstają agregaty czasoprzestrzenne, które opisują badane zjawisko na danym obszarze i w pewnym przedziale czasu. Również w tym przypadku istnieje możliwość materializacji agregatów.

Rysunek 5.8 przedstawia w sposób symboliczny procesy zachodzące na czwartym poziomie kontekstowości. Wejściowy strumień (unia strumieni), zawierający rekordy złożone i wyrównane w czasie, podlega agregacji czasowej i przestrzennej w dwóch wymiarach. Na rysunku zaznaczono to za pomocą sumy po czasie i podwójnej sumy po arbitralnie nazwanych wymiarach: x i y. Otrzymywany w wyniku strumień zagregowany jest poddawany materializacji, w celu ponownego wykorzystania gotowych wyników w przyszłości. Każdy agregat składa się z kilku agregatów elementarnych, opisujących wybrany obszar agregowanej przestrzeni.

5.4.6 Poziom 5: kontekst środowiskowy

Piąty poziom kontekstowości zakłada łączenie różnych typów strumieni danych oraz danych statycznych w jeden, syntetyczny strumień. Oznacza to, że dane zarejestrowane przez odrębne typy urządzeń pomiarowych, o odmiennych charakterystykach i parametrach oraz dane statyczne, łączone są ze sobą w jeden strumień, jak gdyby pochodził on z wirtualnego (syntetycznego) źródła. Warunkiem koniecznym takiego złączenia jest zgodność na poziomie zarówno czasu, jak i przestrzeni łączonych danych – ideą jest dopasowanie do siebie tych rekordów, które opisują różne aspekty tego samego



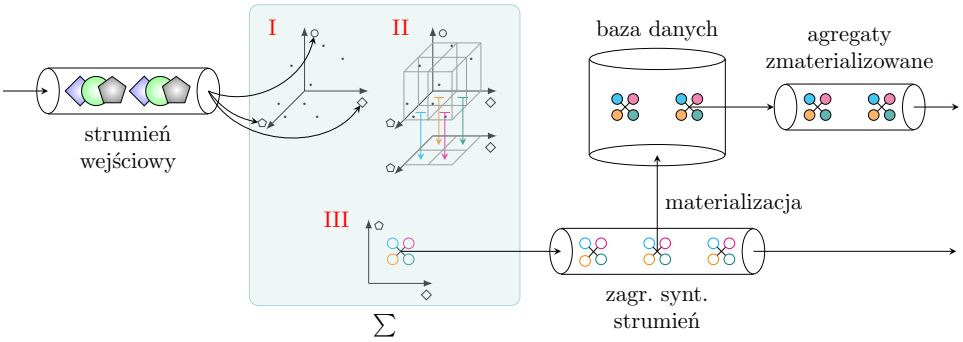
Rysunek 5.9: Symboliczne przedstawienie piątego poziomu kontekstowości

zjawiska, zarejestrowanego na danym obszarze i w danym przedziale czasu przez różne źródła. W konsekwencji niezbędna jest wcześniejsza agregacja czasowa i przestrzenna przynajmniej jednego z łączonych strumieni – drugi będzie wówczas dołączony na zasadzie przynależności do okna czasoprzestrzennego, w którym agregowany był pierwszy strumień. Dane powstałe w wyniku złączenia (syntezy) strumieni danych wykorzystywane mogą być w celu badania wzajemnego wpływu na siebie różnych zjawisk, współwystępujących na danym obszarze i dziejących się w zbliżonym czasie.

Na rysunku 5.9 przedstawiono w sposób symboliczny procesy zachodzące na piątym poziomie kontekstowości. Wejściowe strumienie zawierają agregaty czasoprzestrzenne pochodzące z różnych typów źródeł (o odmiennych schematach i charakterystykach czasowych). Dane w nich zawarte, wraz z danymi statycznymi (zapisanymi w pamięci trwałej) podlegają operacji złączenia, oznaczonej na rysunku dwoma skierowanymi do siebie trójkątami. W wyniku tego działania, tworzone są strumienie syntetyczne, zawierające rekordy opisujące złożone zjawiska zarejestrowane wieloaspektowo przez różne niezależnie źródła. Strumienie syntetyczne podlegają zapisowi w bazie danych, z której mogą zostać później odtworzone.

5.4.7 Poziom 6: kontekst środowiskowy z agregacją

Szósty i jednocześnie najwyższy poziom kontekstowości wiąże się z przedstawieniem syntetycznych strumieni danych w wielowymiarowej przestrzeni atrybutów. Każdy atrybut syntetycznego strumienia stanowi odrębny jej wymiar, a poszczególne rekordy reprezentują punkty bądź zbiory punktów należących do tej przestrzeni. Taka forma reprezentacji danych koresponduje z modelem wielowymiarowym stosowanym w klasycznych hurtowniach



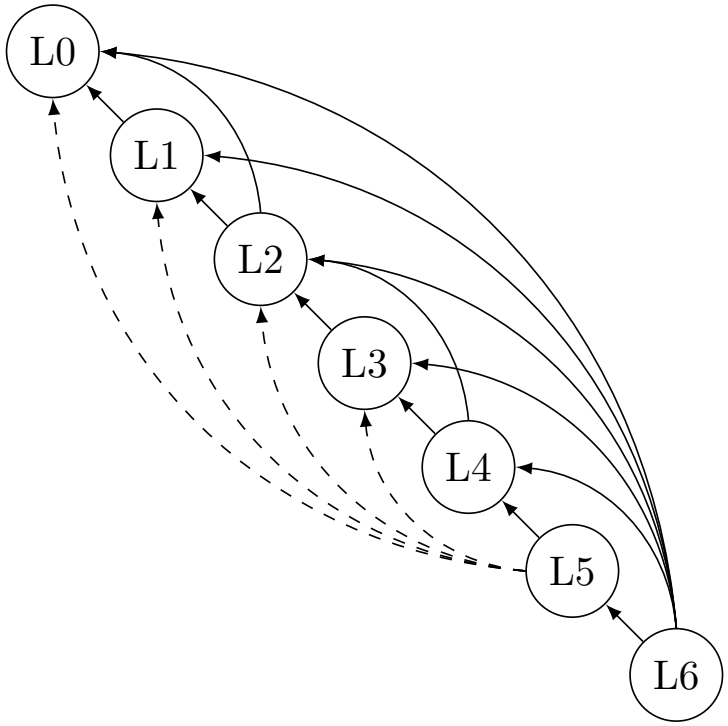
Rysunek 5.10: Symboliczne przedstawienie szóstego poziomu kontekstowości

danych – umożliwia wieloaspektową analizę wpływu poszczególnych zjawisk na siebie, co stanowi istotę badania kontekstu środowiskowego. Na tym poziomie występują między innymi zapytania agregujące w różnych zakresach wartości wybranych atrybutów, w szczególności innych niż czas i przestrzeń. Działanie takie wzbogaca analizę przeprowadzaną na niższym poziomie o możliwość wyznaczenia pewnych uśrednionych cech, z pominięciem nieistotnych szczegółów, a także obliczenie różnego rodzaju miar statystycznych wspomagających badanie wzajemnego wpływu współistniejących zjawisk. Podobnie jak we wcześniejszych przypadkach, również na tym poziomie kontekstowości możliwa jest materializacja uzyskanych agregatów.

Rysunek 5.10 przedstawia w sposób symboliczny procesy zachodzące na szóstym poziomie kontekstowości. Wejściowy strumień syntetyczny przekształcany jest na wielowymiarową przestrzeń atrybutów, a poszczególne rekordy wchodzące w jego skład tworzą punkty w tej przestrzeni. Zostało to przedstawione w części oznaczonej rzymską cyfrą I. W następnym kroku przestrzeń ta dzielona jest zgodnie z kryteriami zapytania wielowymiarowego na podprzestrzenie, w których przeprowadzana jest agregacja (rzymska cyfra II). W wyniku powstają agregaty wielowymiarowe (rzymska cyfra III), które tworzą zagregowany strumień syntetyczny. Strumień ów podlega materializacji i jest zapisywany w bazie danych, w celu ponownego wykorzystania wybranych agregatów.

5.4.8 Wzajemna zależność poziomów kontekstowości

Opisane wyżej poziomy kontekstowości zależą od siebie, tworząc wspomniany model kaskadowy. Jednakże zależność ta nie jest trywialna, czyli taka, w której każdy poziom wykorzystuje dane i metody objęte poziomem bezpośrednio niższym. Niektóre poziomy mogą wykorzystywać dane i metody z kilku niższych poziomów, a niektóre nie – zależy to przede wszystkim od samych operacji wykonywanych na poszczególnych poziomach kontekstowości.



Rysunek 5.11: Diagram zależności pomiędzy poziomami kontekstowości

Przykładowo, zbudowanie wielowymiarowej przestrzeni atrybutów (poziom szósty) niekoniecznie odbywać się musi na syntetycznych strumieniach powstałych na skutek złączenia (poziom piąty), lecz równie dobrze przeprowadzone może być na dowolnych danych, wliczywszy w to nawet surowe strumienie danych (poziom zerowy). Z kolei archiwizacja danych przestrzennych (poziom trzeci) musi koniecznie operować na zagregowanych czasowo danych (poziom drugi), gdyż w przeciwnym przypadku skutkowałoby to niejednoznacznością w zapisie (dane z różnych okresów mieszałyby się ze sobą).

Rysunek 5.11 przedstawia diagram zależności pomiędzy poszczególnymi poziomami kontekstowości. Strzałki oznaczają zależność – ich groty wskazują na poziomy, z których dane mogą być wykorzystane na poziomie, z którego wychodzi dana strzałka. Strzałkami przerywanymi oznaczono częściową zależność, charakterystyczną dla poziomu piątego.

Wspomniana wyżej częściowa zależność dla poziomu piątego wynika z faktu, iż złączenie dwóch lub więcej strumieni wymaga, aby przynajmniej jeden z nich był zagregowany w czasie i przestrzeni. W przypadku, w którym wszystkie łączone strumienie są zagregowane w tych samych oknach czasoprzestrzennych, złączenie ma charakter trywialny i skutkuje połączeniem odpowiadających sobie agregatów. W przypadku niezagregowanych danych w jednym strumieniu, konieczne jest dopasowanie surowych rekordów do

okien zagregowanego strumienia. Z kolei w przypadku dwóch strumieni zagregowanych w nieprzystających oknach, konieczne jest złączenie ze sobą tych okien, które mają niezerową część wspólną.

Rozdział 6

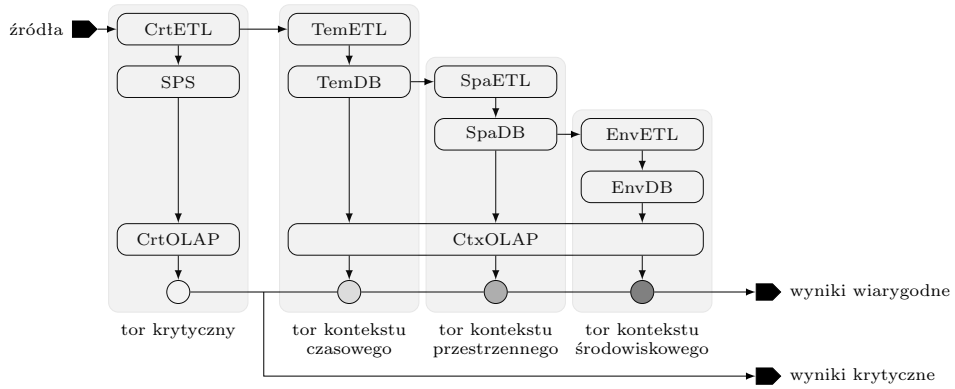
Model strumieniowej hurtowni danych kontekstowych

Niniejszy rozdział przedstawia model strumieniowej hurtowni danych zorientowanej na przetwarzanie wielkich zbiorów danych kontekstowych, zwanej dalej strumieniową hurtownią danych kontekstowych. Na początku rozdziału znajduje się ogólny zarys architektury, wykorzystujący wielotorowe podejście. Następnie, w szczegółach, scharakteryzowane zostały trzy tory kontekstowe – dla każdego podane zostały schematy baz danych w trzech wersjach: dla danych surowych, dla metadanych i dla danych zagregowanych.

6.1 Wielotorowa budowa strumieniowej hurtowni danych kontekstowych

W rozdziale 5 przedstawiono definicje dwóch rodzajów danych: krytycznych (definicja 5.2) oraz kontekstowych (definicja 5.1). Z ich istnienia wynikają dwa podstawowe typy analizy: analiza danych krytycznych oraz analiza stanowiących dla nich kontekst – danych kontekstowych. Chociaż ta druga dostarcza najpełniejszej wiedzy, wymaga odpowiedniego czasu i zasobów, gdyż wolumen danych kontekstowych znacznie przekracza wolumen danych krytycznych. Z drugiej strony, analiza danych krytycznych może być przeprowadzona szybko, lecz dostarcza wyników przybliżonych.

Optymalnym rozwiązaniem wydaje się wypracowanie w krótkim czasie niedokładnych wyników wstępnych, powstałych na podstawie analizy danych krytycznych, a następnie przeprowadzenie ich weryfikacji na podstawie analizy danych kontekstowych. Weryfikacja pozwala uzyskać wyniki wiarygodne, które są dostępne po pewnym czasie od otrzymania wyników krytycznych. Zdefiniowane w ten sposób zachowania daje możliwość szybkiej reak-



Rysunek 6.1: Architektura strumieniowej hurtowni danych kontekstowych

cji na potencjalne zdarzenia interesujące z punktu widzenia użytkowników, a w kolejnym kroku dostarczenie wiarygodnej informacji, potwierdzającej bądź dementującej uprzednio uzyskane wstępne informacje.

Środowiskiem, w którym opisane wyżej operacje mogą zostać przeprowadzone jest strumieniowa hurtownia danych. Z jednej strony umożliwia ona przetwarzanie danych bieżących, z drugiej – udostępnia pełną historię danych, co umożliwia analizę kontekstu. Hurtownia taka, ze względu na duży nacisk położony na rzeczony kontekst, zwana będzie dalej strumieniową hurtownią zorientowaną na przetwarzanie danych kontekstowych, lub krócej – *strumieniową hurtownią danych kontekstowych*, a w skrócie: CtxDW (od ang. *Context Data Warehouse*).

Najważniejszym założeniem architektonicznym strumieniowej hurtowni danych kontekstowych jest jej dwudzielność, rozumiana jako istnienie dwóch głównych torów przetwarzania danych: krytycznego oraz kontekstowego. Ten pierwszy związany jest z analizą danych krytycznych i działa w czasie rzeczywistym. W ramach toru kontekstowego odbywa się analiza danych kontekstowych w celu ujednoznacznienia wyników działania toru krytycznego. Dane te w przeważającej większości stanowią dane historyczne, zapisane w pamięci trwałej, lecz budowa toru kontekstowego umożliwia również przetwarzanie najświeższych danych w formie zbliżonej do strumieniowej – w postaci strumienia aktualizacji pochodzącego z procesu ekstrakcji kontekstowej. Powyższe rozważania, razem z przedstawionymi w dalszej części rozdziału treściami, stanowią zarazem dowód tezy 2.

Tor kontekstowy może zostać podzielony na trzy kolejne tory, zgodnie z przedstawionym w rozdziale 5 podziałem danych kontekstowych i metod ich przetwarzania. W konsekwencji, model strumieniowej hurtowni danych kontekstowych składa się z czterech torów: jednego krytycznego i trzech kontekstowych: czasowego, przestrzennego i środowiskowego. Koncepcję tę ilustruje rysunek 6.1.

W skład toru krytycznego wchodzi silnik ekstrakcji krytycznej (CrtETL,

od ang. *critical*), silnik przetwarzania strumieniowego (SPS, od ang. *Stream Processing System*) oraz serwer OLAP danych krytycznych (CrtOLAP). Zadaniem tego pierwszego jest wstępna obróbka danych napływających nieustannym strumieniem prosto ze źródeł. Zasadnicze przetwarzanie odbywa się w silniku SPS, a uzyskane wyniki wykorzystywane są przez serwer CrtOLAP w celu przeprowadzenia analiz, zgodnie z wytycznymi użytkowników. Na wyjściu otrzymywane są wyniki krytyczne w postaci alarmów i alertów.

Budowa poszczególnych torów kontekstowych jest zbliżona do siebie. W każdym z nich wyróżnić można silnik ekstrakcji: TemETL (czasowy, od ang. *temporal*), SpaETL (przestrzenny, od ang. *spatial*) oraz EnvETL (środowiskowy, od ang. *environmental*). Silniki te są odpowiedzialne za wstępne przetwarzanie danych i wprowadzanie ich do docelowego formatu obowiązującego w bazie danych. W każdym torze kontekstowym znajduje się oddzielna baza danych (TemDB, SpaDB, EnvDB), która przechowuje zarówno surowe, jak i zagregowane dane kontekstowe, a ponadto również metadane je opisujące. W następnej kolejności znajduje się serwer OLAP danych kontekstowych (CtxOLAP), który jest współdzielony przez wszystkie trzy tory kontekstowe. Jego zadaniem jest dostarczenie danych kontekstowych w formie wielowymiarowych struktur (kostek), uwzględniając przy tym wielopoziomową agregację wzdłuż wybranych wymiarów.

6.2 Model bazy danych czasowych

Baza danych czasowych TemDB jest jednym z podstawowych magazynów danych wchodzących w skład strumieniowej hurtowni danych kontekstowych. Baza ta jest również pierwszym miejscem, w którym przechowywane są rekordy pochodzące z wejściowych strumieni danych, po przejściu procesu ekstrakcji oraz transformacji. Baza TemDB stanowi zarazem główną pamięć trwałą toru kontekstu czasowego hurtowni CtxDW.

Ze względu na swoje umiejscowienie w torze kontekstu czasowego, baza danych czasowych ma za zadanie przechowywać pełną historię zarejestrowanych w hurtowni strumieni danych w formie zbliżonej do ich pierwotnej natury. Ma to na celu łatwą rekonstrukcję wybranych strumieni od dowolnego momentu w czasie. Wymaganie to narzuca konieczność opisania poszczególnych strumieni danych za pomocą okoliczności ich powstania i pochodzenia. Okoliczności te można określić mianem metadanych.

Wieloaspektowe spojrzenie na pochodzenie strumieni danych wpływa bezpośrednio na schemat danych oraz metadanych zastosowany w TemDB. Proponuje się wykorzystanie klasycznych dla hurtowni danych schematów gwiazdy i konstelacji faktów, lecz z jedną zasadniczą modyfikacją, polegającą na przechowywaniu całych fragmentów poszczególnych strumieni danych w formie binarnej typu BLOB (ang. *Binary Large Object*). Działanie to ma na celu zapobieżenie zbytniej granulacji danych, co mogłoby mieć nieko-

rzystne skutki ze względu na wysokie tempo aktualizacji bazy danych, jak również charakter zapytań, w których dominuje pobieranie dłuższych sekwencji rekordów. Stosowanie zmodyfikowanych schematów znanych z klasycznych hurtowni danych na potrzeby przechowywania dynamicznie zmieniających się danych strumieniowych zostało opisane w literaturze, czego przykładem jest choćby rozszerzona gwiazda kaskadowa [30]. Wykorzystanie kolumn typu BLOB do zapisu całych fragmentów strumieni danych stanowi rozszerzenie koncepcji przechowywania zmaterializowanych agregatów w formie ciągów bitów, wprowadzonej przy projektowaniu Materializowanej Listy Agregatów [50].

Model bazy danych czasowych składa się z trzech powiązanych ze sobą modeli cząstkowych: bazy surowych danych czasowych, bazy metadanych czasowych oraz bazy agregatów czasowych. Modele te zawierają pewne współwystępujące encje, które w modelu fizycznym mogą być zaimplementowane w postaci wspólnych tabel. Niemniej, ich rozdzielenie na trzy niezależne modele logiczne pozwala na oddzielenie dynamicznej części przechowującej strumienie danych od statycznej części przeznaczonej dla metadanych.

6.2.1 Model bazy surowych danych czasowych

Model bazy surowych danych czasowych zbudowany jest na podstawie schematu gwiazdy, w której tabela faktu przechowuje wszystkie pomiary (w formie binarnej), a wymiarami są różne aspekty pochodzenia strumieni danych. Wyróżnić można cztery podstawowe wymiary, które są wspólne dla wielu systemów przechowujących i przetwarzających strumienie danych:

a) wymiar źródła – opisujący konkretne źródło generujące strumień danych;

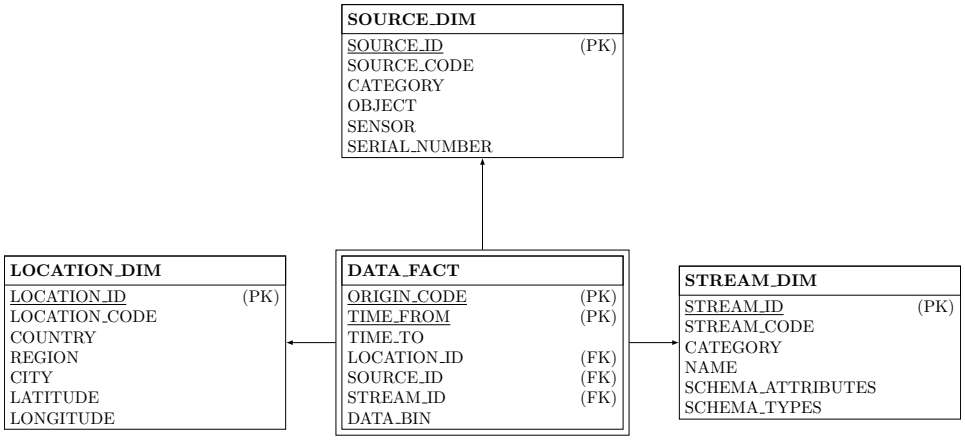
b) wymiar strumienia – opisujący dane znajdujące się w strumieniu;

c) wymiar lokalizacji – opisujący geograficzne położenie zjawiska opisywanego przez strumień;

d) wymiar czasu – opisujący moment zarejestrowania poszczególnych pomiarów.

Trzy pierwsze wymiary charakteryzują źródła strumieni danych, a ostatni – generowane przez nie dane. Ze względu na możliwość zmian zachodzących w strukturze samych źródeł, takich jak na przykład: podmiana urządzenia, zmiana lokalizacji, aktualizacja schematu generowanego strumienia, zdecydowano się na wyodrębnienie aż trzech wymiarów związanych ze źródłem – możliwe wówczas staje się wystosowanie zapytania dotyczącego danych pochodzących z wybranej lokalizacji, niezależnie od konkretnych urządzeń działających w tej lokalizacji na przestrzeni czasu.

Na rysunku 6.2 przedstawiono schemat bazy surowych danych czasowych, zaznaczając podwójnym obramowaniem tabelę faktu. Jest ona oto-



Rysunek 6.2: Model bazy surowych danych czasowych

czona przez trzy z opisanych czterech wymiarów – wymiar czasu został zdegenerowany do pary znaczników czasowych, ze względu na bardzo wysoki poziom szczegółowości.

Kluczem głównym tabeli faktu jest para wartości: kod pochodzenia oraz znacznik czasowy początku przechowywanego fragmentu strumienia danych, będący zarazem wymiarem zdegenerowanym. Ponadto, tabela faktu zawiera klucze obce do tabel wymiarów oraz jedną miarę: strumień danych zapisany w formie binarnej (DATA_BIN). Każdy wiersz tabeli faktu odpowiada sekwencji rekordów z pojedynczego strumienia, które zostały zarejestrowane w pewnym przedziale czasu. Przedział ten jest zdefiniowany za pomocą dwóch znaczników czasowych, z których pierwszy jest wspomnianym już znacznikiem początkowym, a drugi – odpowiadającym mu znacznikiem końcowym.

Kod pochodzenia (oznaczenie: ORIGIN_CODE) łączy w sobie trzy wymiary: źródła, strumieni i lokalizacji w pojedynczą liczbową wartość. W ten sposób niejako dubluje dołączone przez klucze obce wymiary, lecz został wprowadzony w celu uniezależnienia operacji wyszukania pochodzenia danego strumienia od przeszukiwania tabeli faktu – jest to możliwe dzięki dodatkowym tabelom wchodzącym w skład modelu bazy metadanych. W dalszym ciągu jednak jest możliwe jest klasyczne wyszukiwanie przy użyciu kluczy obcych, bowiem umożliwia ono pominięcie któregoś z wymiarów lub dopuszczenie pewnego podzbioru atrybutów w kryteriach zapytania. Tego typu działania mogą przykładowo służyć pobraniu większego fragmentu danego strumienia bądź też kilku fragmentów z różnych strumieni, które zostały zarejestrowane w tym samym czasie.

Poszczególne tabele wymiarów zbudowane są według tego samego wzorca: klucz główny (sztuczny), kod wymiaru (klucz kandydujący, który jest zgłaszany przez poszczególne źródła) oraz atrybuty. Dla każdego wymiaru

zdefiniowano hierarchię. Należy tutaj zaznaczyć, iż jest to jedynie propozycja, gdyż liczba poziomów hierarchii, jak i ich nazwy są zawsze zależne od konkretnego zastosowania, a niniejszy przykład jest jedynie prezentacją modelu teoretycznego. Proces przeszukiwania bazy strumieni właściwych polega na realizacji zapytań uwzględniających kod pochodzenia oraz zakres dat. W wyniku uzyskiwany jest ciąg bitów będący zakodowanym binarnie podzbiorem rekordów należących do przeszukiwanego strumienia danych.

6.2.2 Model bazy metadanych czasowych

Metadane to wszystkie dane opisujące źródła danych oraz generowane przez nie strumienie. Część z nich została już omówiona w poprzedniej części, przy prezentacji modelu bazy surowych danych czasowych – są to wymiary: źródła, strumieni i lokalizacji.

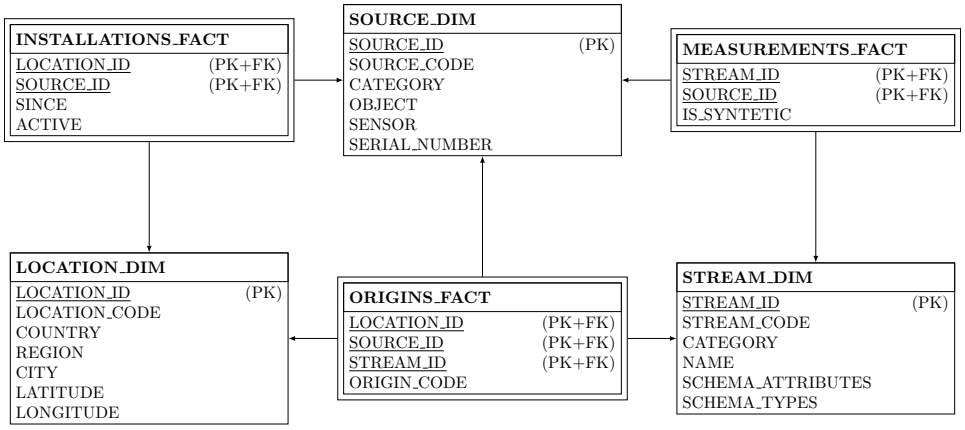
Ideą bazy metadanych czasowych jest uzupełnienie tych trzech wymiarów o dodatkowe informacje oraz powiązanie ich ze sobą nawzajem. Wymiary: źródła i strumieni wiążą się ze sobą faktem dokonywania pomiarów – konkretne źródło generuje określony strumień danych. Z kolei wymiary: źródła i lokalizacji wiążą się ze sobą faktem instalacji – konkretne źródło zostało zainstalowane w danej lokalizacji.

Trzy opisane wymiary są ze sobą połączone faktem współistnienia w procesie wytwarzania strumieni danych – konkretne źródło, zainstalowane w konkretnej lokalizacji i generujące konkretny typ danych stanowi pochodzenie wszystkich danych należących do związanego z nim strumienia. Zmiana choć jednego z rzeczonych elementów skutkuje odmiennych pochodzeniem. Sytuacja taka wystąpić może na przykład na skutek wymiany urządzenia bądź jego przemieszczenia lub też aktualizacji zakresu zbieranych danych (zmiany schematu strumienia). Z każdym pochodzeniem można powiązać unikalną wartość numeryczną, którą jest wspomniany już kod ORIGIN_CODE.

Schemat bazy danych dla metadanych czasowych bazuje na schemacie konstelacji faktów. Wyróżnić w nim można tabele wymiarów (te same, co w schemacie bazy surowych danych czasowych), które są współdzielone przez kilka tabel faktów, opisujących różne aspekty powstawania strumieni danych. Ilustruje to rysunek 6.3.

Należy tutaj zwrócić uwagę na fakt, iż tabele wymiarów w obu modelach mogą być fizycznie tymi samymi tabelami lub istnieć niezależnie. Pierwsze rozwiązanie zapewnia spójność danych oraz jest mniej kosztowne pamięciowo, ale wymusza przechowywanie strumieni danych oraz metadanych w tej samej bazie. W drugim rozwiązaniu konieczne jest zadbanie o poprawną replikację danych, ale możliwe jest zastosowanie odrębnego magazynu danych dla metadanych. Pozostałe tabele charakteryzują się w sposób następujący:

a) fakt pomiarów – łączy wymiar źródła z wymiarem strumienia i zawiera dodatkowe informacje o strumieniu generowanym przez dane źródło



Rysunek 6.3: Model bazy metadanych czasowych

– przykładowo może to być informacja, czy dany strumień pochodzi wprost ze źródła, czy został wstępnie przetworzony;

b) fakt instalacji – łączy wymiar źródła z wymiarem lokalizacji i zawiera informacje na temat czasu instalacji danego źródła w danej lokalizacji, jak również inne informacje, jak przykładowo status źródła;

c) fakt pochodzenia – łączy w sobie wszystkie trzy wymiary i zawiera dodatkowo unikalny identyfikator, będący zarazem wymiarem zdegenerowanym w tabeli faktu w modelu bazy surowych danych czasowych.

Proces przeszukiwania bazy metadanych czasowych polega na realizacji zapytań odnoszących się do konkretnych poziomów w hierarchii wymiarów, w celu uzyskania zbiorów wartości numerycznych reprezentujących pochodzenie strumieni wygenerowanych przez źródła spełniające warunki zapytania. Ponadto, zastosowany model bazy metadanych czasowych umożliwia zadawanie różnorakich zapytań odnoszących się do infrastruktury pomiarowej. Przykładowo możliwe jest wyszukanie tych źródeł, które były zainstalowane od wybranego roku, ale już nie są aktywne, albo zbadanie ile poziomów agregacji występuje w strumieniach generowanych przez wszystkie aktywne źródła zainstalowane na danym obszarze geograficznym. Kolejnym zastosowaniem bazy metadanych czasowych, oprócz wymienionych zapytań o infrastrukturę, jest dostarczanie wszystkich informacji niezbędnych do jednoznacznej identyfikacji tych rekordów, które spełniają określone warunki, a które nie zostały jeszcze utrwalone w bazie strumieni właściwych. Rekordy te zaliczają się wciąż do danych ulotnych i są tymczasowo przechowywane w różnych strukturach pamięci nietrwałej.

6.2.3 Model bazy agregatów czasowych

Idea materializacji agregatów wynika z kosztowności ich wytwarzania. W wyniku agregacji strumienia danych w wymiarze czasu, powstaje strumień agregatów, które nie ulegają zmianie, gdyż raz zapisane strumienie danych nie są modyfikowane. Utrwalenie wyników agregacji może posłużyć przyspieszeniu innych zapytań zadawanych w przyszłości, które przynajmniej w części pokrywają się ze zmaterializowanymi danymi [53].

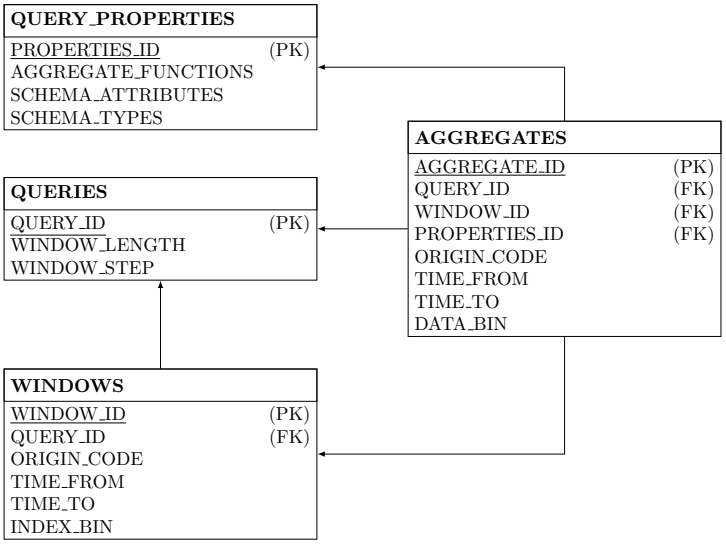
Powyższe założenie jest o tyle prawdziwe, o ile parametry zapytania korzystającego ze zmaterializowanych danych są tożsame z oryginalnym zapytaniem. Parametry te, to przede wszystkim rozmiar i krok okna czasowego oraz zastosowana funkcja agregująca, a także zakres czasowy zapytania i przetwarzany strumień danych. W przypadku zastosowania innej funkcji agregującej dla tego samego okna czasowego lub agregacji wykonywanej na innych atrybutach, konieczne jest ponowne wykonanie zapytania. Rozwiązaniem tego problemu może być przechowywanie razem z danymi indeksów charakterystycznych, kodujących liczby rekordów wchodzących w skład kolejnych okien czasowych i przekształcających te okna w okna liczebnościowe (o z góry określonej liczbie rekordów).

Model bazy agregatów czasowych zakłada istnienie osobnej tabeli dla indeksów charakterystycznych oraz dla agregatów, a także dwóch tabel przechowujących informacje o zapytaniach – tych związanych z oknami czasowymi oraz tych niezależnych od sposobu agregacji czasu. Zarówno indeksy charakterystyczne, jak i same agregaty zapisywane są w postaci ciągów bitów w kolumnach typu BLOB. Schemat taki umożliwia wyszukiwanie zarówno gotowych agregatów, jak i tylko indeksów charakterystycznych i został przedstawiony na rysunku 6.4.

Zastosowana kolumna ORIGIN_CODE w tabelach WINDOWS i AGGREGATES jest kodem pochodzenia, który występuje również w poprzednio omówionych modelach cząstkowych bazy strumieni danych i w sposób jednoznaczny identyfikuje każdy strumień danych. Tabela WINDOWS przechowująca indeksy charakterystyczne jest niezależna zarówno od podzbioru atrybutów, jak i od zastosowanych funkcji agregujących.

6.3 Model bazy danych przestrzennych

Baza danych przestrzennych SpaDB jest drugą w kolejności bazą, z punktu widzenia przepływu danych przez strumieniową hurtownię danych kontekstowych CtxDW. Umiejscowiona w torze kontekstu przestrzennego, baza SpaDB przechowuje dane w odmiennej formie, niż omówiona uprzednio baza danych czasowych – ukierunkowanej bowiem na zapytania odnoszące się do aspektu lokalizacji geograficznej źródeł danych. Część danych może się pokrywać w obu bazach, lecz sposób ich reprezentacji jest inny. Ponadto w bazie SpaDB znajdują się nowe w stosunku do bazy TemDB dane oraz



Rysunek 6.4: Model bazy agregatów czasowych

agregaty.

Podobnie jak baza danych czasowych, również baza SpaDB zbudowana jest w sposób nawiązujący do schematów spotykanych w tradycyjnych hurtowniach danych. Co więcej, w przypadku bazy danych przestrzennych także zdecydowano się na zastosowanie binarnej reprezentacji danych, lecz w nieco innej formie. Każdy ciąg bitów, będący zarazem pojedynczą wartością typu BLOB, reprezentuje pomiary zarejestrowane w danym momencie czasu, przez wszystkie źródła leżące w wybranym regionie. Taką „paczkę" danych przestrzennych można określić mianem *klastra* (definicja 5.6).

Z powyższego wynikają dwa wnioski: cały obszar, który objęty jest analizą przez hurtownię CtxDW musi być podzielony na regiony oraz poszczególne strumienie danych powinny być zagregowane w czasie, w celu wyrównania charakterystyk poszczególnych źródeł. Podział na regiony wynika z potencjalnie dużej liczby źródeł oraz ich prawdopodobnego nierównomiernego rozmieszczenia, a konieczność agregacji w czasie jest założeniem trzeciego poziomu kontekstowości (opisanego w rozdziale 5.4.4). Dzięki temu znacznie ułatwiona jest operacja agregacji w przestrzeni – dane pochodzące z blisko siebie położonych źródeł znajdują się w jednym bloku pamięci oraz mają jednakowe wartości znaczników czasowych.

Model bazy danych przestrzennych również składa się z trzech modeli cząstkowych: bazy surowych danych przestrzennych, bazy metadanych przestrzennych oraz bazy agregatów przestrzennych. Co więcej, część tabel jest tożsama lub podobna do tych, występujących w bazie danych czasowych – dotyczy to przede wszystkim metadanych opisujących okoliczności powstania strumieni danych.

6.3.1 Model bazy surowych danych przestrzennych

Model bazy surowych danych przestrzennych wykorzystuje schemat gwiazdy, podobnie jak analogiczny model dla danych czasowych. Tabela faktu zawiera dane przestrzenne w postaci binarnej i jest połączona z trzema tabelami wymiarów. Wyróżnić można następujące wymiary:

a) wymiar regionu – opisujący wymiary (fizyczne) danego regionu;

b) wymiar strumienia – opisujący dane znajdujące się w strumieniu;

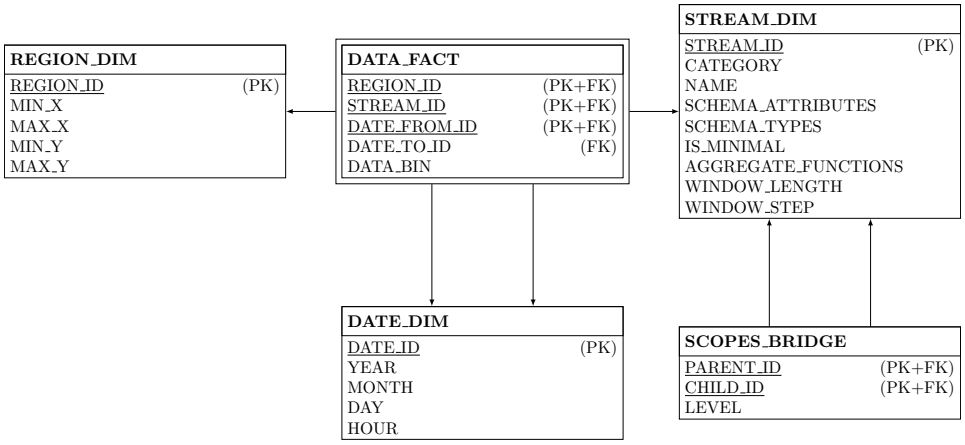
c) wymiar czasu – opisujący moment zarejestrowania poszczególnych pomiarów.

Należy nadmienić, iż nie wymieniono tutaj wymiaru lokalizacji – jest tak z powodu przechowywania całej grupy pomiarów zarejestrowanych przez różne źródła. Implikuje to relację N:M pomiędzy tabelą faktu a wymiarem lokalizacji. Rozwiązaniem jest w tym przypadku wymiar regionu, który jest połączony za pomocą mostu z rzeczonym wymiarem lokalizacji (zostanie to dokładniej opisane w części przeznaczonej dla bazy metadanych przestrzennych). Pozostałe dwa wymiary (strumienia i czasu) są zbliżone do tych, które zostały przedstawione przy okazji omawiania modelu bazy surowych danych czasowych.

Na dokładniejszą uwagę zasługuje wymiar strumienia. Ze względu na możliwość współistnienia wielu strumieni pochodzących z tego samego źródła i różniących się jedynie stopniem agregacji (zawierających bardziej i mniej szczegółowe dane), zdecydowano się na zastosowanie mostu hierarchii do modelowania hierarchii poszczególnych strumieni w pionie. Most hierarchii jest dodatkową tabelą, która zawiera dwa klucze obce odnoszące się do tabeli wymiaru strumienia i przechowuje w sobie informację o relacji rodzic-dziecko zachodzącej pomiędzy poszczególnymi strumieniami.

Opisane trzy wymiary w sposób jednoznaczny opisują fakt zgrupowania pomiarów pochodzących ze wszystkich źródeł w danym regionie, które to pomiary zostały zarejestrowane w danymi momencie (przedziale) czasu oraz są danego typu – to ostatnie określa wymiar strumienia. Ilustracją do powyższych rozważań jest rysunek 6.5, który przedstawia kompletny model bazy surowych danych przestrzennych.

Tabela faktu zawiera klucze obce do tabel wymiarów, wymiary zdegenerowane oraz dane w postaci binarnej. Kluczem głównym jest kombinacja identyfikatora regionu, identyfikatora daty początkowej oraz identyfikatora strumienia. Wymiar czasu został przedstawiony, odmiennie niż w przypadku bazy danych czasowych, za pomocą tabeli wymiaru daty, co zostało podyktowany niższym poziomem szczegółowości czasu, niż w przypadku surowych danych czasowych. Przeszukiwanie bazy surowych danych przestrzennych polega na odnalezieniu odpowiednich klastrów danych, pochodzących z wybranych regionów i zarejestrowanych we wskazanym przedziale czasu.



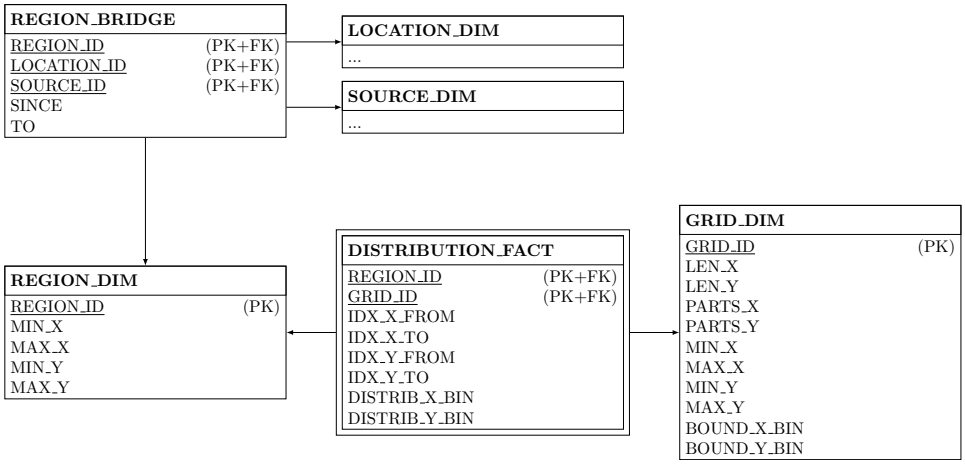
Rysunek 6.5: Model bazy surowych danych przestrzennych

6.3.2 Model bazy metadanych przestrzennych

Metadane przestrzenne związane są z podziałem przestrzeni na minimalne jednostki agregacji (MAU, od ang. *Minimal Aggregation Unit*), tworzące siatkę (ang. *grid*). Minimalna jednostka agregacji wyznacza zarazem najdrobniejszy poziom granulacji danych, poniżej którego nie analizuje się już danych. Każdy region obejmuje swym obszarem wiele MAU, co przekłada się na potrzebę przechowywania informacji o fakcie przyporządkowania regionów do odpowiednich MAU, a także – o fakcie przyporządkowania określonych pomiarów wchodzących w skład klastrów (przyporządkowanych regionom) do określonych MAU.

W konsekwencji, centralną tabelą bazy metadanych przestrzennych jest tabela faktu podziału, która łączy się z wymiarami regionu i siatki minimalnych jednostek agregacji, co zostało przedstawione na rysunku 6.6. Tabela wymiaru regionu jest tożsama z tą, używaną w bazie surowych danych przestrzennych, lecz dodatkowo łączy się za pomocą mostu z wymiarami lokalizacji i źródeł, znanymi z bazy danych czasowych. Tabela siatki (GRID_DIM) zawiera informacje na temat podziału przestrzeni na minimalne jednostki agregacji: liczbę i długość tych jednostek, obszar objęty siatką, a także tablice liczb rzeczywistych, wyznaczających początki kolejnych MAU dla poszczególnych wymiarów.

Tabela faktu DISTRIBUTION_FACT składa się z klucza głównego, będącego złożeniem kluczy obcych do tabel wymiarów oraz z atrybutów będących indeksami w tablicach wartości granicznych (przechowywanych w tabeli wymiaru siatki), wyznaczających zakres tych wartości przyporządkowanych do danego regionu. Ponadto, w tabeli faktu znajdują się dwie tablice liczb całkowitych, kodujące binarnie przyporządkowanie określonych pomiarów do minimalnych jednostek agregacji. Szczegółowy opis metody kodowania binarnego oraz jej wykorzystania do obliczania agregatów przestrzennych



Rysunek 6.6: Model bazy metadanych przestrzennych

zostanie opisany w rozdziale 7.2.

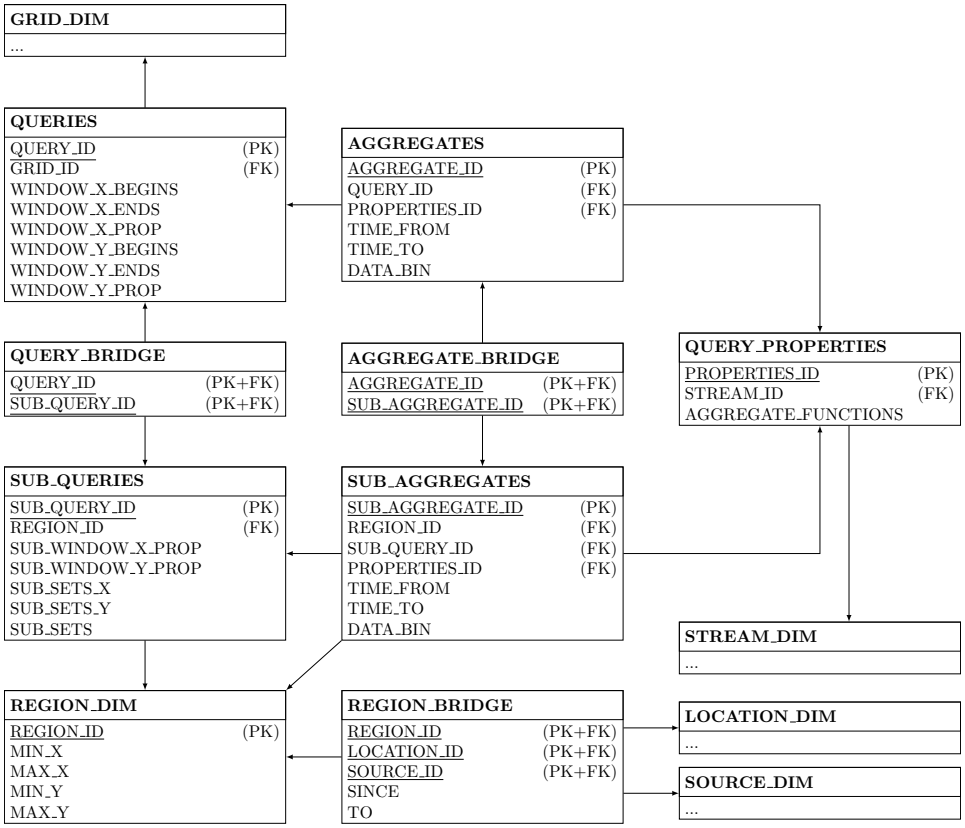
Zapytania kierowane do bazy metadanych przestrzennych zwracają informacje o wspomnianym przyporządkowaniu, które jest funkcją zarówno regionu, jak i sposobu podziału przestrzeni na MAU (czyli siatki). Dzięki temu możliwe jest wyłonienie podzbioru tych pomiarów (w obrębie klastra), które mają wziąć udział w realizacji pojedynczego kroku operacji agregacji przestrzennej, czyli pomiarów należących do jednego okna przestrzennego.

6.3.3 Model bazy agregatów przestrzennych

W przypadku bazy agregatów przestrzennych konieczne jest uwzględnienie zapytań obejmujących swym zakresem obszar kilku regionów. Zasadne zatem jest podzielenie zapytania przestrzennego na podzapytania, a jego wyników (agregatów) na wyniki cząstkowe. Dzięki temu możliwe jest przyspieszenie realizacji innych zapytań, które mają część wspólną ze zmaterializowanym zapytaniem.

Rysunek 6.7 przedstawia schemat bazy agregatów przestrzennych. Każde zapytanie przestrzenne może składać się z kilku okien (z wielokrotnego okna przestrzennego, zgodnie z definicją 5.9). W tabeli QUERIES znajdują się początki i końce okien przestrzennych w obu wymiarach, jak również zakodowana binarnie informacja o propagacji okna przestrzennego wzdłuż obu wymiarów.

Z tabelą QUERIES połączona jest tabela SUB_QUERIES, przy czym związek ten ma charakter N:M i realizowany jest przez osobną tabelę. Reprezentuje ona podzapytanie związane z konkretnym regionem i łączy się z wymiarem regionu. Każde podzapytanie zawiera podzbiór informacji o propagacji okna dla danego regionu oraz zakodowaną binarnie informację o tych pomiarach, które należą do kolejnych okien podzapytania – zarówno w po-



Rysunek 6.7: Model bazy agregatów przestrzennych

szczególnych wymiarach, jak i globalnie. Szczegółowy opis metod kodowania zapytania przestrzennego zostanie podany w rozdziale 7.2.

Z tabelą zapytań i podzapytań powiązane są odpowiednio tabele: agregatów i agregatów cząstkowych. Zawierają one, oprócz informacji o typie agregacji i agregowanych atrybutach, binarną reprezentację agregatów. Zapytanie do bazy agregatów przestrzennych kierowane jest na początku do tabel zapytań i agregatów – jeśli nie zwróci żadnego wyniku, oznacza to konieczność przeszukania częściowych informacji w tabelach podzapytań i wyników cząstkowych. W sytuacji, w której również to zapytanie nie zwróci wyników, konieczna jest agregacja surowych danych, a więc wystosowanie zapytania do bazy surowych danych przestrzennych. Dany wynik cząstkowy może wchodzić w skład więcej niż jednego wyniku – związek między tabelami AGGREGATE oraz SUB_AGGREGATES ma charakter N:M. Obie tabele łączą się z tabelą reprezentującą dodatkowe parametry zapytania (QUERY_PROPERTIES), które są niezależne zarówno od okna przestrzennego, jak i podziału na agregaty cząstkowe i główne.

6.4 Model bazy danych środowiskowych

Baza danych środowiskowych jest ostatnią i najbardziej złożoną bazą danych w modelu strumieniowej hurtowni danych kontekstowych. Założeniem tej bazy jest przechowywanie złączonych (syntetycznych) strumieni danych (definicja 5.12), powstałych na skutek dopasowania do siebie kilku strumieni źródłowych, po ich wcześniejszym wyrównaniu (agregacji) w czasie i przestrzeni.

Baza danych środowiskowych w swoich założeniach projektowych jest zbliżona do dwóch wcześniejszych baz – wykorzystuje pewne elementy schematów klasycznych hurtowni danych oraz binarną reprezentację danych. Odmiennie niż w przypadku bazy danych czasowych i przestrzennych, poszczególne ciągi bitów reprezentują zbiór rekordów zarejestrowanych w pewnym przedziale czasu i na pewnym obszarze przez syntetyczne (powstałe w wyniku złączenia) źródła.

Taka forma organizacji danych wynika bezpośrednio z głównego przeznaczenia bazy danych środowiskowych – jej zadaniem jest optymalizacja zapytań agregujących wzdłuż innych wymiarów niż czas i przestrzeń, przy czym agregacji poddawane są rekordy związane podobnymi okolicznościami czasu oraz miejsca. W konsekwencji czasoprzestrzeń dzielona jest na *bloki* (definicja 5.10), a wszystkie rekordy z danego bloku zapisywane są w jednym obszarze pamięci, jako pojedyncza wartość typu BLOB.

Dodatkowo, w odróżnieniu od poprzednich baz danych, dane środowiskowe zapisywane są w formie kolumnowej – każdy atrybut osobno. Ułatwia to agregację wybranych atrybutów, jak również agregowanie wzdłuż ich wartości. Taki układ danych umożliwia ponadto dowolną interpretację wymiarowości – dany atrybut w pewnych zapytaniach może występować w formie wymiaru, a w innych w formie miary.

Model bazy danych środowiskowych składa się, podobnie jak poprzednio opisane modele, z trzech modeli cząstkowych: bazy surowych danych środowiskowych, bazy metadanych środowiskowych oraz bazy agregatów środowiskowych. Część tabel pokrywa się z tymi z poprzednich modeli lub z nimi łączy, przede wszystkim w przypadku metadanych opisujących podział czasoprzestrzeni na bloki oraz partycjonowanie poszczególnych atrybutów.

6.4.1 Model bazy surowych danych środowiskowych

Model bazy surowych danych środowiskowych wykorzystuje, podobnie jak poprzednie modele (czasowy i przestrzenny), model gwiazdy jako sposób wewnętrznej organizacji danych. W tabeli faktu zapisane są poszczególne bloki rekordów (podzbiory czasoprzestrzeni), zakodowane w formie ciągów bitów. W omawianym modelu wyróżnić można następujące wymiary:

a) wymiar bloku – opisuje zakres danego bloku wyrażony jako para granicznych wartości w poszczególnych wymiarach (przestrzeni i czasu);

b) wymiar syntezy – opisuje strumień syntetyczny powstały na skutek złączenia strumieni pierwotnych;

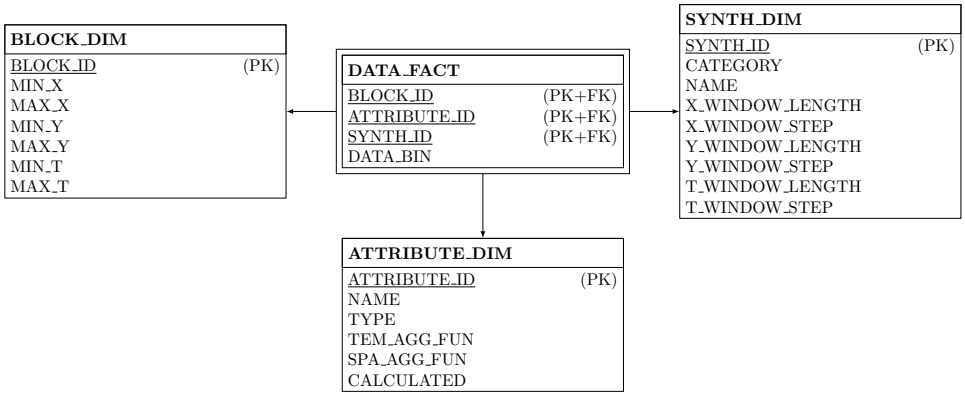
c) wymiar atrybutu – opisuje nazwę, typ oraz funkcje agregujące charakterystyczne dla danego atrybutu.

Opisane powyżej wymiary są odmienne od tych, które były do tej pory wykorzystywane w poprzednich modelach. Wymiar bloku jest w pewnym sensie rozszerzeniem wymiaru regionu o czas. Wymiar syntezy został wprowadzony ze względu na możliwość powstawania nowych strumieni (syntetycznych), które składają się z atrybutów pochodzących z kilku różnych strumieni pierwotnych, przy czym nie zawsze muszą to być wszystkie atrybuty z łączonych strumieni. Na szczególną uwagę zasługuje wymiar atrybutu, którego obecność wynika bezpośrednio z założenia o kolumnowej organizacji danych środowiskowych. W rezultacie, każdy wiersz tabeli faktu odnosi się do jednego konkretnego atrybutu syntetycznego strumienia.

Opisane trzy wymiary w sposób jednoznaczny identyfikują zbiór rekordów, na który składają się pojedyncze wartości danego atrybutu, który należy do wybranego schematu strumienia syntetycznego i został zarejestrowany w określonych okolicznościach czasowo-przestrzennych. Należy tutaj zwrócić uwagę na fakt, iż pomiędzy wymiarem atrybutu a syntezy występuje związek N:M, gdyż dany atrybut może wchodzić w skład więcej niż jednego strumienia syntetycznego – w przeciwnym wypadku, nie byłoby uzasadnione wydzielanie atrybutu jako osobnego wymiaru.

Rysunek 6.8 przedstawia diagram bazy surowych danych środowiskowych. Kluczem głównym tabeli faktu jest złożenie kluczy obcych do tabel wymiarów. Oprócz tych atrybutów, posiada ona jedynie zbiór rekordów dany jako ciąg bitów. Przeszukiwanie bazy surowych danych środowiskowych polega na znalezieniu takiego zbioru rekordów tabeli faktu, które zawierają zakodowane binarnie rekordu opisujące wybrane atrybuty syntetycznego strumienia powstałego w określonych obszarach i przedziałach czasu.

Wymiar syntezy łączy się z wymiarem strumienia związkiem N:M, który jednak nie został naniesiony na wspomniany diagram ze względu na czytelność – zostanie to dokładniej opisane przy okazji metadanych. Wymiar syntezy zawiera parametry (długość i krok) okna, zarówno czasowego, jak i przestrzennego, które zostały użyte do wstępnej agregacji danych wykonywanej przy okazji operacji złączenia, jeszcze przed zapisem w bazie danych środowiskowych. Użyte funkcje agregujące (osobno dla czasu i przestrzeni) zostały umieszczone w tabeli reprezentującej wymiar atrybutu, gdyż istnieje możliwość zastosowania różnych funkcji agregujących dla różnych atrybutów danego strumienia syntetycznego.



Rysunek 6.8: Model bazy surowych danych środowiskowych

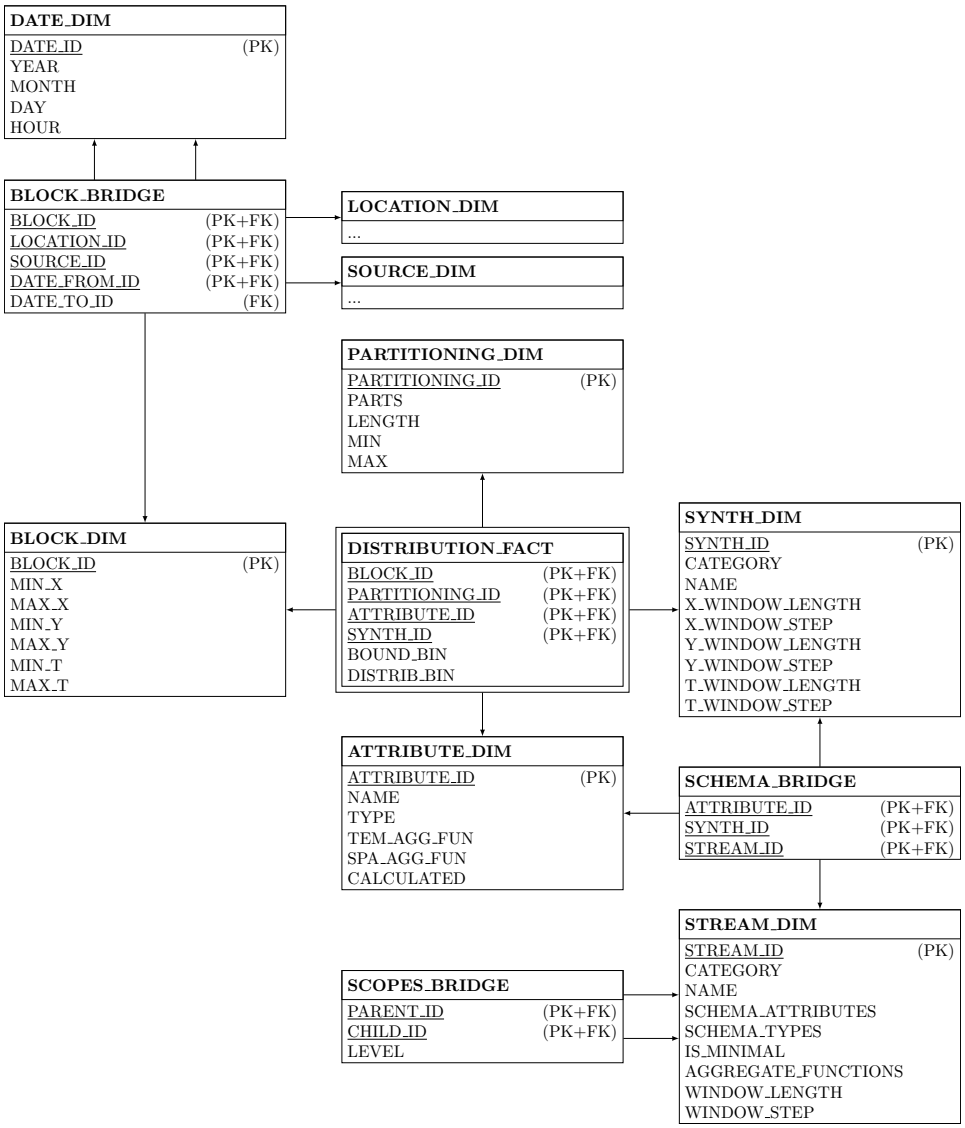
6.4.2 Model bazy metadanych środowiskowych

Metadane środowiskowe opisują rozkład wartości wszystkich atrybutów strumienia syntetycznego, w odniesieniu do minimalnych jednostek agregacji, podobnie jak w przypadku bazy danych przestrzennych. W przypadku danych środowiskowych jednak, MAU zdefiniowane są dla każdego atrybutu osobno. Metadane środowiskowe uwzględniają podział na bloki – to w danym bloku wyznaczane są zakresy występowania poszczególnych wartości atrybutów.

Rysunek 6.9 przedstawia diagram bazy metadanych środowiskowych. Rolę tabeli faktu pełni tabela rozkładu (DISTRIBUTION_FACT), która połączona jest z czterema tabelami wymiarów. Trzy spośród nich są tożsame z tymi, występującymi w modelu bazy surowych danych środowiskowych. Czwartym wymiarem jest w tym przypadku wymiar podziału (partycjonowania), który określa wspomniane minimalne jednostki agregacyjne.

Każdy wiersz tabeli faktu zawiera informację o tym, w jaki sposób w obrębie danego bloku rozłożone są wartości danego atrybutu pochodzącego z danego strumienia syntetycznego, przy zastosowaniu danego schematu partycjonowania (który definiuje rozmiar oraz liczbę MAU). Tabela ta, oprócz kluczy obcych do tabel wymiaru (stanowiących zarazem jej złożony klucz główny) zawiera dwie tablice liczb. Pierwsza z nich koduje wartości graniczne kolejnych MAU, a druga zawiera informację o tym, które pomiary danego bloku przynależą do której minimalnej jednostki agregacji. Szczegółowy opis metody kodowania binarnego oraz jej wykorzystania do obliczania agregatów środowiskowych zostanie opisany w rozdziale 7.2 – podobnie jak w przypadku danych przestrzennych.

Ponadto, diagram bazy metadanych środowiskowych zawiera również wspomniane wcześniej połączenie pomiędzy wymiarem syntezy a wymiarem strumienia, które dodatkowo łączy się z wymiarem atrybutu (jest to związek ternarny). Tabela SCHEMA_BRIDGE jest mostem łączącym te trzy



Rysunek 6.9: Model bazy metadanych środowiskowych

wymiary – każdy jej wiersz wskazuje na fakt występowania danego atrybutu w danym strumieniu oraz danym strumieniu syntetycznym oraz jednocześnie na strumienie, które brały udział w operacji złączenia dającej w rezultacie strumień syntetyczny.

Wymiar bloku połączony jest mostem z wymiarami lokalizacji i źródeł oraz (podwójnie) z wymiarem daty. Daje to dodatkowe możliwości wyszukiwania, gdyż każdy blok wyznacza zarazem podzbiór czasoprzestrzeni, a więc obejmuje swoim zakresem pewien zbiór lokalizacji i źródeł danych oraz przedział czasu.

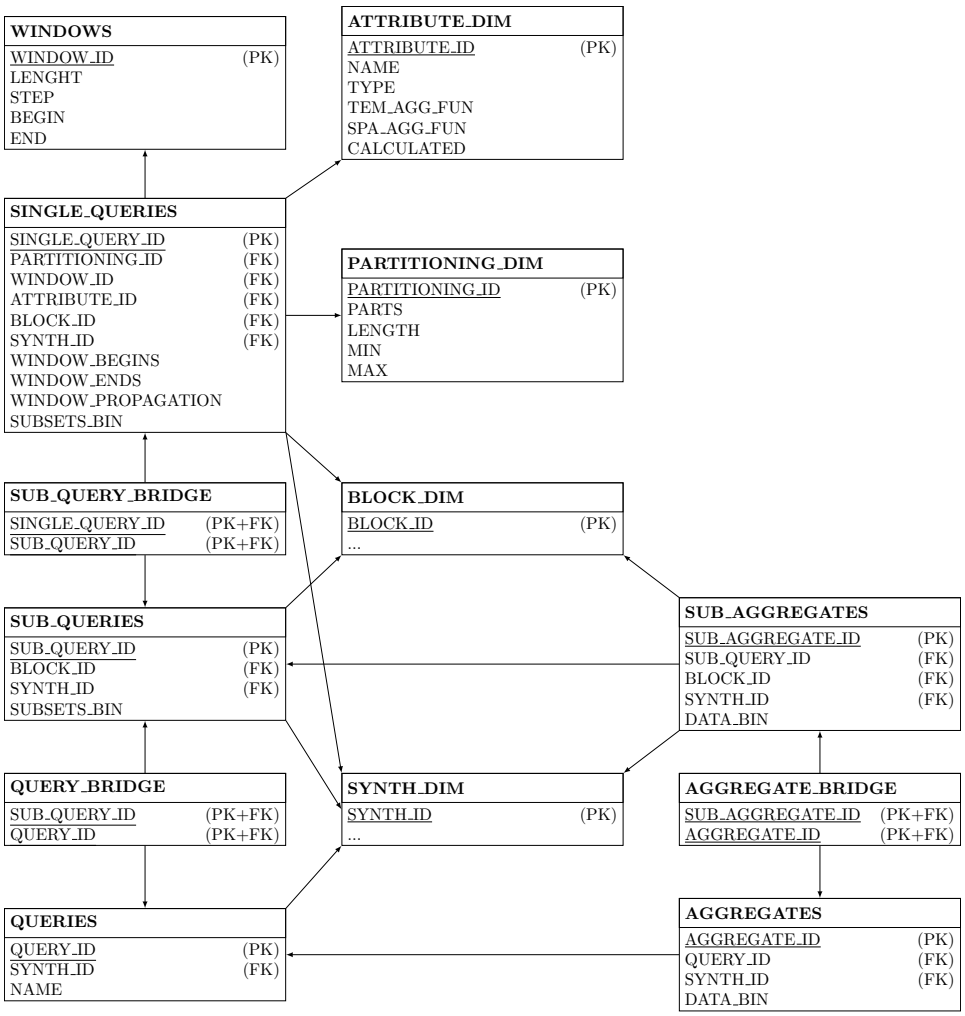
Zapytania kierowane do bazy metadanych środowiskowych odpowiadają na pytania o podzbiory tych rekordów danego bloku, które spełniają określone predykaty zakresowe zdefiniowane na wybranym podzbiorze swoich atrybutów. Jest to osiągalne przez wykorzystanie informacji o rozkładzie wartości tychże. Dzięki temu ułatwiona jest operacja agregacji wzdłuż wymiarów innych niż czas i przestrzeń.

6.4.3 Model bazy agregatów środowiskowych

Baza agregatów środowiskowych przechowuje zmaterializowane wyniki zapytań agregujących wzdłuż dowolnych atrybutów. Zapytania takie, podobnie jak to miało miejsce w przypadku bazy agregatów przestrzennych, mogą obejmować swoim zakresem kilka bloków. W konsekwencji konieczne jest podzielenie zapytania na podzapytania, które są wykonywane w obrębie pojedynczych bloków. Co więcej, każde takie zapytanie może być rozbite na podzapytania jednowymiarowe, odnoszące się tylko do pojedynczego atrybutu. Podobnie jak w przypadku zapytań, również agregaty mogą zostać podzielone na agregaty częściowe, będące wynikami operacji agregacji przeprowadzonej w obrębie pojedynczego bloku.

Na rysunku 6.10 przedstawiono diagram bazy agregatów środowiskowych. Jego lewa strona związana jest z zapytaniami, a prawa – z agregatami. Każde zapytanie (tabela QUERIES) składa się z kilku podzapytań, które z kolei składają się z kilku podzapytań jednowymiarowych (tabela SINGLE_QUERIES). Ta ostatnia tabela łączy się z trzema tabelami: wymiarem podziału, wymiarem atrybutu oraz tabelą reprezentującą pojedyncze okno uogólnione (definicja 5.14). Oprócz tego na rzeczonym diagramie występują również wymiary: bloku i syntezy.

Agregaty zapisane są w formie binarnej w tabeli AGGREGATES, która łączy się z tabelą agregatów cząstkowych, która również przechowuje dane w formie binarnej. Obie tabele agregatów (całościowych i cząstkowych) łączą się z odpowiadającym sobie tabelom zapytań. Wyjątkiem jest tutaj tabela podzapytań jednowymiarowych, która nie ma swojego odpowiednika po stronie tabel agregatów – jest tak, ponieważ rozdzielenie na pojedyncze wymiary ma sens w przypadku okien, lecz nie w przypadku agregatów – te tworzone są zawsze dla pełnego zestawu atrybutów wymiarowych (zgodnie



Rysunek 6.10: Model bazy agregatów środowiskowych

z zapytaniem).

Tabele podzapytań (wielowymiarowych i jednowymiarowych) zawierają zapisane w binarnej formie informacje o podzbiorach rekordów (w obrębie powiązanego z nimi bloku), które wchodzą w skład kolejnych okien wielowymiarowych, które składają się na zapytanie.

Rozdział 7

Wybrane modele procesów w serwerze CtxOLAP

Niniejszy rozdział opisuje wybrane modele procesów realizowanych w serwerze danych kontekstowych CtxOLAP, stanowiącym jedną z głównych składowych modelu strumieniowej hurtowni danych kontekstowych. Rozdział składa się z dwóch głównych części. Pierwsza z nich omawia silnik strumieniowej kostki danych CUBIT, stanowiący rozwinięcie silnika Materializowanej Listy Agregatów i zarazem wykorzystujący adaptację idei kostki danych na potrzeby przetwarzania danych kontekstowych. Drugą część rozdziału poświęcono propozycji wielowymiarowego bitowego indeksu zakresowego BRI, który wykorzystując kodowanie binarne oraz model obliczeń równoległych, wspomaga wykonywanie wielowymiarowych zapytań zakresowych.

7.1 Silnik strumieniowej kostki CUBIT

Model strumieniowej hurtowni danych kontekstowych stanowi rozwinięcie modelu strumieniowej hurtowni danych o istnienie kilku torów przetwarzania danych kontekstowych. W oryginalnym modelu, istotnymi komponentami był silnik MAL (Materializowanej Listy Agregatów) oraz serwer StrOLAP (strumieniowy OLAP). Te dwa elementy odpowiadały za efektywne wyodrębnianie poszukiwanych danych oraz ich przetwarzanie analityczne, skupiające się przede wszystkim na agregacji w czasie.

W przypadku modelu strumieniowej hurtowni danych kontekstowych, ze względu na występowanie kilku torów przetwarzania danych, Materializowaną Listę Agregatów zastąpiono bardziej rozbudowanym rozwiązaniem, przystosowanym do przetwarzania danych o różnym stopniu wymiarowości. Rozwiązanie to nazwane zostało silnikiem strumieniowej kostki danych (w skrócie: silnikiem CUBIT, od ang. *Cube Iterator*). Nazwa ta nawiązuje do kostki danych, będącej podstawową formą reprezentacji danych w klasycznych hurtowniach danych oraz do iteratora, stanowiącego podstawę działa-

nia silnika MAL.

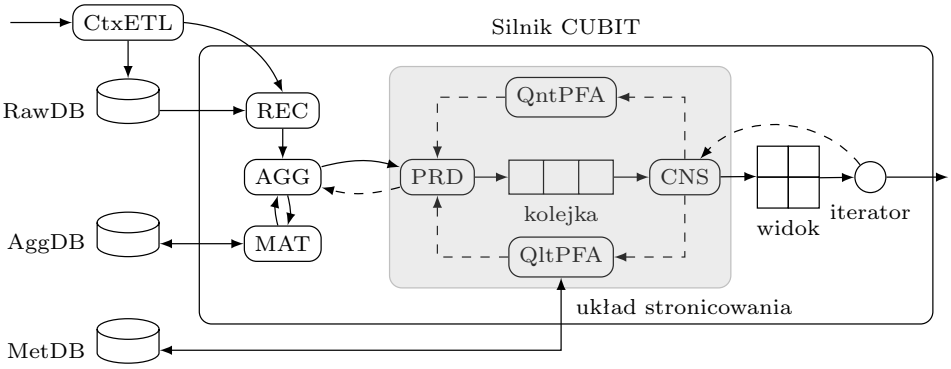
Silnik CUBIT, jako rozwinięcie Materializowanej Listy Agregatów, również wykorzystuje proces stronicowania pamięci wraz z algorytmem wypełniania stron, jako mechanizm kontroli zajętości pamięci, umożliwiający zarazem płynne przechodzenie po długiej sekwencji danych. Podobnie jak silnik MAL, silnik CUBIT udostępnia dane w formie nie tyle kompletnej struktury danych, co dostarczając metod umożliwiających iterowanie po kolejnych elementach. Kolejnym podobieństwem do MAL jest zapewnianie przez silnik CUBIT jednakowego dostępu do danych, niezależnie od ich źródła: mogą to być dane historyczne (zapisane w bazie danych), bieżące (przechowywane w strumieniu), surowe lub zagregowane (wyliczone w locie lub odtworzone ze struktur zmaterializowanych).

W przeciwieństwie jednak do Materializowanej Listy Agregatów, silnik CUBIT bardziej nawiązuje do kostki danych OLAP niż do listy agregatów, adaptując ją do charakterystycznych dla strumieni danych warunków pracy – dane stanowią ciągłą i potencjalnie nieskończoną sekwencję. W konsekwencji, strumieniowa kostka danych niejako „rozciąga" się w wymiarze czasu (potencjalnie w nieskończoność), umożliwiając użytkownikowi na sukcesywne przesuwanie się od przeszłości do teraźniejszości oraz w wymiarze przestrzeni, pozwalając tym samym na odwiedzanie kolejnych obszarów. Silnik CUBIT stanowi zarazem jeden z głównym komponentów kontekstowego serwera OLAP (CtxOLAP). Taki stan rzeczy odróżnia model strumieniowej hurtowni danych kontekstowych od swojego pierwowzoru (strumieniowej hurtowni danych), w którym silnik MAL był swego rodzaju widokiem dla bazy danych.

Zadaniem silnika CUBIT jest przedstawienie danych kontekstowych (czasowych, przestrzennych i środowiskowych) w formie zbliżonej do kostki OLAP, czyli wielowymiarowej struktury ułatwiającej agregację i wizualizację danych. W przypadku Materializowanej Listy Agregatów, dane mogły być przeglądane wyłącznie wzdłuż czasu, w kierunku od najstarszych do najnowszych. Silnik CUBIT umożliwia przechodzenie po danych w różnych kierunkach, wliczając w to zarówno czas, jak i lokalizację, a także dowolne inne atrybuty, jeśli tylko przetwarzane dane mają taką naturę:

a) dla kontekstowych danych czasowych, silnik CUBIT wykazuje duże podobieństwo do MAL (Materializowanej Listy Agregatów) – dane można przeglądać względem czasu oraz identyfikatorów źródeł, a poszczególne elementy mają postać pojedynczych rekordów (podobnie jak w przypadku bazy danych czasowych, opisanej w rozdziale 6.2);

b) dla kontekstowych danych przestrzennych, silnik CUBIT umożliwia przechodzenie w pierwszej kolejności po czasie i regionach, a w dalszej, po wartościach współrzędnych geograficznych – poszczególne elementy (uporządkowane względem czasu) mają postać macierzy (klastrów, zdefiniowanych w rozdziale 6.3) zawierających agregaty czasowe, które



Rysunek 7.1: Model silnika CUBIT

są uporządkowane przestrzennie;

c) dla kontekstowych danych środowiskowych, silnik CUBIT umożliwia przechodzenie w pierwszej kolejności po czasie oraz przestrzeni, a w drugiej po wartościach pozostałych atrybutów, przy czym każdy element ma postać bloku danych (tak jak to opisano w rozdziale 6.4), w którym poszczególne rekordy są uporządkowane według pewnego podzbioru atrybutów.

Funkcjonowanie silnika CUBIT można przybliżyć za pośrednictwem zbioru procesów, z których każdy realizuje wybrane zadanie. Procesy te można podzielić na dwie podstawowe kategorie: procesy ciągłe, działające w sposób nieprzerwany oraz procesy zależne, wywoływane w określonych sytuacjach. Każdy z takich procesów wykonuje się w określonym module silnika CUBIT. Poszczególne moduły stanowią rozbudowaną i usystematyzowaną wersję koncepcji przedstawionej w pracy własnej [67]^{W}. Rysunek 7.1 przedstawia architekturę silnika CUBIT. Strzałkami ciągłymi zilustrowano przepływ danych, podczas gdy strzałki przerywane oznaczają przepływ sterowania. Naniesione na rysunku bazy danych są zgodne z przedstawionym w rozdziale 6 podziałem na bazy: danych surowych, agregatów oraz metadanych, występujących we wszystkich torach kontekstowych.

Moduł rekonstrukcji strumieni danych (REC) zajmuje się pobieraniem danych z różnych źródeł: bieżących (strumień danych pochodzących prosto z silnika ETL) i historycznych (statyczny zbiór danych zapisanych w bazie danych surowych) i ich przekazywaniem do modułu agregacji (AGG), w którym przeprowadzana jest agregacja w zadanych oknach (czasowych, przestrzennych lub uogólnionych). Moduł ów współpracuje z modułem materializacji (MAT), którego przeznaczeniem jest utrwalanie agregatów w celu ich późniejszego wykorzystania. Agregaty wygenerowane przez moduł agregacji trafiają do modułu producenta (PRD), który wraz z modułem konsumenta (CNS) oraz kolejką tworzy układ stronicowania (będący przykładem systemu producent-konsument), stanowiący mechanizm dynamicznego

dostarczania danych klientowi. Nad działaniem tego układu czuwają dwa algorytmy wypełniania stron: *ilościowy* (QntPFA, od ang. *Quantity Page-Filling Algorithm*) oraz *jakościowy* (QltPFA, od ang. *Quality Page-Filling Algorithm*). Ten pierwszy jest odpowiedzialny za wskazanie *ile* danych należy umieścić w kolejce, a ten drugi – *jakie* to mają być dane. Dane odczytane przez konsumenta trafiają do modułu widoku, w którym są transformowane do postaci pożądanej przez klienta, który uzyskuje do nich dostęp za pośrednictwem iteratora.

7.1.1 Proces rekonstrukcji strumieni danych

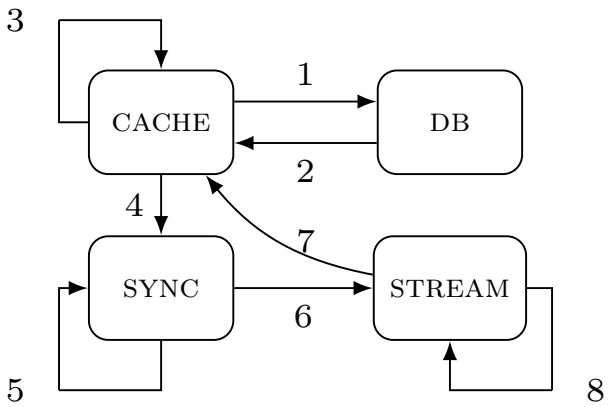
Nazwa modułu rekonstrukcji strumieni danych wzięła się od przeprowadzanego przezeń procesu odtwarzania zarchiwizowanych w bazie danych strumieni danych. W rezultacie generuje on zrekonstruowany strumień danych, który jest tożsamy ze strumieniem pochodzącym z silnika ETL w przeszłości. Moduł rekonstrukcji umożliwia płynne przełączanie się pomiędzy źródłem historycznym a bieżącym i odwrotnie – moduły korzystającego z udostępnianych przez niego danych nie są świadome, z którego źródła one pochodzą.

Kluczowym komponentem modułu rekonstrukcji jest pamięć podręczna (ang. *cache*), w której przechowywane są najświeższe dane, czyli stała liczba ostatnio pobranych z silnika ETL rekordów. W miarę napływania nowych danych, najstarsze są usuwane z pamięci podręcznej – do tego czasu powinny już zostać zarchiwizowane w bazie danych surowych. Moduł rekonstrukcji strumieni danych w swym działaniu wykorzystuje koncepcję maszyny stanów o czterech stanach, związanych z dostępnością danych i miejscem ich zapisu. Interpretacja tych stanów jest następująca:

a) stan CACHE – dane pobierane są z pamięci podręcznej;
b) stan DB – odczyt porcji danych zapisanych w bazie danych surowych;
c) stan SYNC – synchronizacja ze strumieniem;
d) stan STREAM – ciągły odczyt danych ze strumienia.

Rysunek 7.2 przedstawia diagram stanów procesu rekonstrukcji. Początkowym stanem jest stan CACHE, z którego w zależności od dostępności danych możliwe jest przejście albo do stanu DB (dane już zostały usunięte z pamięci podręcznej), albo do stanu SYNC (odczytano wszystkie rekordy z pamięci podręcznej i należy zsynchronizować się ze strumieniem). Proces synchronizacji ze strumieniem jest konieczny ze względu na możliwość istnienia rekordów, które już zostały odczytane ze strumienia, ale nie zostały jeszcze zapisane w pamięci podręcznej (są w trakcie zapisu). W takiej sytuacji bezpośrednie przełączenie się na strumień poskutkowałoby pominięciem części danych.

Po skończeniu synchronizacji następuje przejście do stanu STREAM, w którym dane są bezpośrednio przekazywane ze strumienia do odbiorcy. W pew-



Rysunek 7.2: Diagram stanów procesu rekonstrukcji strumieni danych [67]^{W}

nych okolicznościach możliwa jest desynchronizacja ze strumieniem. Występuje ona w sytuacji, w której odczyt danych przebiega zbyt wolno i następuje przepełnienie bufora odbiorcy. W takim przypadku następuje powrót do początkowego stanu CACHE, z którego ponownie możliwa jest synchronizacja ze strumieniem.

Algorytm 7.1 ilustruje działanie procesu rekonstrukcji – stanowi on implementację maszyny stanów przedstawionej na rysunku 7.2, przy czym w przypadku przejścia do stanu STREAM następuje zakończenie procedury i przekazania sterowania do wywołania zwrotnego (ang. *callback*), uruchamianego każdorazowo po pojawieniu się nowego rekordu w strumieniu. Pozostałe stany realizowane są przez odpowiednie funkcje obsługujące (ang. *handler*).

Realizacja stanu CACHE została przedstawiona jako algorytm 7.2. Na samym początku sprawdza on zakres czasowy rekordów znajdujących się w pamięci podręcznej, a na ich podstawie wyznacza położenie szukanego rekordu. Jeśli rekord ów nie znajduje się w pamięci podręcznej, to następuje przejście do innego stanu, zależnie od wyniku porównania. W szczególności w sytuacji, gdy poszukiwany rekord jeszcze nie został zapisany w pamięci podręcznej, to w zależności od ostatniego znanego znacznika czasu w strumieniu (linia 8) następuje przejście do synchronizacji ze strumieniem lub od razu do stanu STREAM. W przypadku, w którym szukany rekord znajduje się w pamięci podręcznej (linie 14-21), rozpoczyna się ekstrakcja danych

Algorytm 7.1 Proces rekonstrukcji strumienia danych

Wejście:
parameters : parametry zapytania

```
 1: procedure RECONSTRUCTION(parameters)
 2:     let state = CACHE
 3:     loop
 4:         if state = CACHE then
 5:             CACHE-HANDLER(parameters)
 6:         else if state = DB then
 7:             DB-HANDLER(parameters)
 8:         else if state = SYNC then
 9:             SYNC-HANDLER(parameters)
10:         else
11:             stream.register(STREAM-CALLBACK)
12:             break
13:         end if
14:         parameters ← UPDATE()
15:     end loop
16: end procedure
```

z tej pamięci, począwszy od rzeczonego rekordu, aż do końca. Po pobraniu wszystkich możliwych rekordów następuje zmiana na stan SYNC.

Obsługa stanu DB wiąże się z odpytaniem bazy danych o porcję rekordów oraz ich zwróceniem – ilustruje to algorytm 7.3. Po wykonaniu tego zadania następuje powrót do stanu przeszukiwania pamięci podręcznej, w celu sprawdzenia, czy przypadkiem poszukiwane dane się tam nie znajdują.

Stan synchronizacji SYNC (algorytm 7.4) polega na sukcesywnym pobieraniu rekordów z pamięci podręcznej i porównywaniu ich znaczników czasowych z ostatnim znanym znacznikiem ze strumienia. W momencie zrównania się obu znaczników (linia 6), następuje przejście do stanu STREAM – oznacza to, że w strumieniu nie pojawiły się nowe dane w stosunku do pamięci podręcznej.

Ostatni ze stanów jest obsługiwany w nieco odmienny sposób – w strumieniu jest rejestrowana procedura (przedstawia ją algorytm 7.5), która jest wywoływana przez strumień w momencie pojawienia się nowego rekordu – jest on przekazywany do tej funkcji jako argument. Procedura ta zawiera obsługę błędu (linia 5), który występuje na skutek niemożności przesłania odebranego rekordu do odbiorcy. W konsekwencji następuje desynchronizacja ze strumieniem i wznowienie procedury rekonstrukcji strumienia, ze stanem CACHE jako początkowym.

Zgodnie z opisaną zasadą działania, stanem do którego dąży moduł rekonstrukcji strumieni danych jest stan połączenia ze strumieniem, w którym dane z niego są bezpośrednio przekazywane do odbiorców, z pominięciem ich

Algorytm 7.2 Realizacja stanu CACHE

Wejście:
parameters : parametry zapytania
Stałe:
state : bieżący stan
cache : pamięć podręczna
stream: uchwyt do strumienia

```
 1: procedure CACHE-HANDLER(parameters)
 2:     let first = cache.first().time
 3:     let last = cache.last().time
 4:     if parameters.time < first then
 5:         state ← DB
 6:     end if
 7:     if parameters.time > last then
 8:         if parameters.time ⩾ stream.time() then
 9:             state ← STREAM
10:         else
11:             state ← SYNC
12:         end if
13:     end if
14:     let index = cache.find(parameters)
15:     let record = cache[index]
16:     let batch = ∅
17:     while record ≠ NULL do
18:         yield record
19:         index ← index + 1
20:         record ← cache[index]
21:     end while
22:     state ← SYNC
23: end procedure
```

Algorytm 7.3 Realizacja stanu BD

Wejście:
parameters : parametry zapytania
Stałe:
state : bieżący stan
database: połączenie do bazy danych

```
1: procedure BD-HANDLER(parameters)
2:     let batch = database.find(parameters)
3:     state ← CACHE
4:     yield batch
5: end procedure
```

Algorytm 7.4 Realizacja stanu SYNC

Wejście:
parameters : parametry zapytania
Stałe:
state : bieżący stan
cache : pamięć podręczna
stream: uchwyt do strumienia

```
1:  procedure SYNC-HANDLER(parameters)
2:      await parameters.time ⩽ cache.last().time
3:      let index = cache.find(parameters)
4:      let record = cache[index]
5:      while record ≠ NULL do
6:          if record.time = stream.time() then
7:              state ← STREAM
8:              break
9:          end if
10:         yield record
11:         index ← index + 1
12:         record ← cache[index]
13:     end while
14: end procedure
```

Algorytm 7.5 Realizacja stanu STREAM

Wejście:
record : rekord pobrany ze strumienia
Stałe:
state : bieżący stan
stream: uchwyt do strumienia

```
1:  procedure STREAM-CALLBACK(record)
2:      try
3:          yield record
4:      end try
5:      catch(exception)
6:          state ← CACHE
7:          stream.unregister(STREAM-CALLBACK)
8:          RECONSTRUCTION(record.parameters)
9:      end catch
10: end procedure
```

zapisu i odczytu w pamięci trwałej oraz podręcznej. W konsekwencji, odpowiedzią na zapytanie jest strumień danych, który może zostać przerwany jedynie w dwóch przypadkach: wymuszenia zakończenia ze strony jego odbiorcy bądź zbyt powolnego tempa odczytu. W tym drugim przypadku następuje desynchronizacja w celu uniknięcia przepełnienia bufora i konieczne jest ponowne wznowienie procedury rekonstrukcji.

7.1.2 Procesy agregacji i materializacji agregatów

Drugim z omawianych modułów wchodzących w skład silnika CUBIT jest moduł agregacji. Jest on odpowiedzialny za produkcję agregatów w zadanych zapytaniem oknach (czasowych, przestrzennych, uogólnionych). Wynikiem działania tego procesu jest strumień agregatów, który zostaje przekazany dalej (do modułu stronicowania) oraz podlega operacji materializacji. W zależności od typu kontekstu, proces agregacji w różny sposób dokonuje wyboru podzbioru rekordów poddawanych operacji wyznaczania agregatu.

Algorytm 7.6 przedstawia przykład najprostszej wersji tego procesu: agregacji w wymiarze czasu. Działanie to sprowadza się ono do grupowania rekordów uzyskanych z modułu rekonstrukcji strumieni danych względem ich znaczników czasowych, a dokładniej – przynależności tychże do zadanego okna czasowego. Po skompletowaniu wszystkich rekordów, produkowany jest na ich podstawie pojedynczy agregat.

Algorytm 7.6 Proces agregacji strumienia

Wejście:
window : okno agregacji

```
 1: procedure AGGREGATION(window)
 2:     let queue = ∅
 3:     loop
 4:         let record = NULL
 5:         repeat
 6:             receive record
 7:             record ⇒ queue
 8:         until record ∈ window
 9:         let last = queue.remove()
10:         window ← window.next()
11:         let agg = AGG-FUNCTION(queue)
12:         queue.clear()
13:         last ⇒ queue
14:         yield agg
15:     end loop
16: end procedure
```

W linii 5 algorytmu 7.6 znajduje się operacja grupowania, przeprowa-

dzana dopóty, dopóki dany rekord znajduje przynależy do danego okna czasowego. W momencie, gdy pobrany rekord nie należy już do okna, zostaje on również dopisany do kolejki agregowanych danych. Takie działanie wynika z zastosowanej pętli `repeat...until`. W związku z tym w linii 9 jest on usuwany z kolejki, a po jego wyczyszczeniu związanym ze zmianą okna – dodawany z powrotem (linia 13), jako pierwszy rekord nowego okna. Właściwa operacja agregacji przeprowadzana jest w linii 11 poprzez wywołanie konkretnej funkcji agregującej na kolejce rekordów.

Zazwyczaj przy agregacji czasowej zakłada się, że poszczególne okna mają rozmiar większy od odległości w czasie pomiędzy poszczególnymi rekordami, a co za tym idzie – obejmują swoim zasięgiem kilka rekordów. Istnieje jednak możliwość, w której okna są mniejsze od odległości pomiędzy rekordami, a więc ich zakres przypada na okres czasu pomiędzy wybraną parą rekordów Taka sytuacja jest bardziej przypomina proces nadpróbkowania (ang. *oversampling*), aniżeli agregacji, gdyż w rezultacie uzyskuje się zbiór wyników o większej liczności, niż zbiór wejściowy. Konieczne jest wówczas użycie wybranego algorytmu nadpróbkowania zamiast algorytmu agregacji. Zasadność zastosowania tego typu operacji w przypadku silnika CUBIT może wynikać z konieczności dostosowania stosunkowo rzadkiego strumienia danych na potrzeby analizy wymagającej wyższej częstotliwości napływania danych, choćby w celu synchronizacji z innym, gęstszym strumieniem.

Kolejnym z omawianych modułów jest moduł materializacji danych. Jego zadanie polega na realizacji dwóch podstawowych procesów. Pierwszy z nich przeprowadza sukcesywny zapis gotowych agregatów wyprodukowanych w module agregacji. Drugi proces odpowiedzialny jest za odczytywanie zapisanych agregatów, które są uformowane w strony o stałym rozmiarze wyrażonym liczbą agregatów. Agregaty archiwizowane są w bazie zmaterializowanych agregatów, której działanie jest dodatkowo wspomagane przez pamięć podręczną zmaterializowanych agregatów. Jej zadaniem jest przechowywanie najczęściej używanych stron, jak również tych, które nie zostały jeszcze zapełnione i trwa ich proces zapisywania.

Materializacja agregatów przeprowadzana jest w sposób cykliczny, każdorazowo po pojawieniu się nowego agregatu. Proces ten ilustruje algorytm 7.7. W linii 5 przeprowadzana jest próba odczytu strony z pamięci podręcznej. W przypadku niepowodzenia (linia 6) tworzona jest nowa strona. W następnej kolejności przeprowadzane jest sprawdzenie, czy strona jest w pełni zapisana (linia 9). Jeśli tak, może ona zostać utrwalona w bazie danych, a w jej miejsce tworzona jest nowa strona. Na koniec odbywa się dopisanie agregatu do strony i jej zachowanie w pamięci podręcznej.

Zmaterializowane strony agregatów (kompletne i niekompletne) mogą być wykorzystywane na podobnej zasadzie jak widoki materializowane – w celu przyspieszenia realizacji zapytań. Wyszukiwaniem oraz zwracaniem właściwych stron zajmuje się proces odczytu zmaterializowanych stron agregatów, przedstawiony jako algorytm 7.8. Wykorzystuje on parametry zapy-

Algorytm 7.7 Proces materializacji agregatów

Stałe:

cache : pamięć podręczna zmaterializowanych agregatów
database : połączenie z bazą danych zmaterializowanych agregatów

```
1:  procedure MATERIALIZE()
2:      loop
3:          let record = NULL
4:          receive record
5:          let page = cache.find(agg)
6:          if page = NULL then
7:              page ← new PAGE
8:          end if
9:          if page.full() then
10:             database.save(page)
11:             page ← new PAGE
12:         end if
13:         agg ⇒ page
14:         cache.put(page)
15:     end loop
16: end procedure
```

tania w celu odnalezienia żądanej strony, przy czym szukanie w bazie danych poprzedzone jest wykonaniem zapytania do pamięci podręcznej (linia 2). W przypadku braku powodzenia, następuje próba odczytu strony z bazy danych (linia 4). Gdy również w tej lokalizacja strona o podanych parametrach nie istnieje, zwracana jest wartość NULL sugerująca, że dany typ agregatów nie był do tej pory produkowany i w związku z tym konieczne jest przeprowadzenie procesu agregacji danych. Gdy próba pobrania strony z bazy danych powiedzie się, to odczytana strona umieszczana jest w pamięci podręcznej.

7.1.3 Układ stronicowania

Układ stronicowania tworzą współpracujące ze sobą moduły producenta i konsumenta, które współdzielą kolejkę przechowującą aktualnie przeglądany fragment strumienia danych. Składają się nań dane pobrane bezpośrednio ze strumienia bieżącego lub dane zrekonstruowane na podstawie zarchiwizowanych rekordów w bazie danych surowych, jak również zmaterializowane agregaty, które zostały już uprzednio obliczone i utrwalone w bazie agregatów. Ponadto w skład układu stronicowania wchodzą również moduły: algorytmu ilościowego i jakościowego. Ich działanie polega na wyznaczaniu parametrów zapytania zasilającego kolejkę nowymi danymi, na podstawie informacji pochodzących z modułów producenta i konsumenta.

Algorytm 7.8 Proces odczytu zmaterializowanych stron agregatów

Wejście:
parameters : parametry zapytania
Stałe:
cache : pamięć podręczna
database : połączenie z bazą danych

```
 1: procedure RESTORE(parameters)
 2:     let page = cache.find(parameters)
 3:     if page = NULL then
 4:         page ← database.find(parameters)
 5:         if page = NULL then
 6:             return NULL
 7:         end if
 8:         cache.put(page)
 9:     end if
10:     return page
11: end procedure
```

Ze względu na potrzebę synchronizacji dostępu do współdzielonego zasobu, jakim jest rzeczona kolejka, konieczne jest zastosowanie mechanizmu ochrony w postaci semaforów. Uzasadnieniem takiego wyboru jest naturalna zdolność semaforów do śledzenia liczby żądań i zwolnień dostępu. Semafor jest zmienną całkowitą, z którą skojarzono dwie operacje: `P()` podejmującą próbę zajęcia zasobu oraz `V()` zwalniającą przydzielony zasób. Problem synchronizacji dostępu do kolejki układu stronicowania silnika CUBIT jest przykładem klasycznego problemu producenta-konsumenta. Może on zostać rozwiązany przy użyciu trzech semaforów: binarnego semafora chroniącego dostęp do kolejki oraz pary semaforów zliczających pobrane oraz załadowane rekordy. Dzięki temu nie jest możliwe zapisywanie danych do pełnej kolejki, jak również odczyt danych z pustej kolejki, a ponadto tylko jeden proces w danej chwili ma możliwość modyfikacji stanu kolejki.

Kolejka wraz z mechanizmem kontroli dostępu w postaci semaforów jest obsługiwana przez dwa procesy: producenta i konsumenta – każdy z nich rezyduje w powiązanym module układu stronicowania. W pierwszej kolejności zostanie omówiony proces konsumenta, gdyż jest mniej złożony. Algorytm 7.9 prezentuje postać tej procedury. Przed właściwym pobraniem rekordu z kolejki następuje próba zajęcia semafora pilnującego, by kolejka nie była pusta (A). W następnym kroku podejmowana jest próba uzyskania dostępu do samej kolejki, za pośrednictwem semafora binarnego (M). Po pobraniu rekordu z kolejki zwalniany jest binarny semafor (M), jak również semafor pilnujący wolnego miejsca w kolejce (F).

Procedura ładowania (przedstawiona jako algorytm 7.10) stanowi swego rodzaju odwrotność procedury pobierania. Jest przy tym znacznie bardziej

Algorytm 7.9 Proces konsumenta

Stałe:

queue : kolejka iteratora
A : semafor zliczający dostępne rekordy
F : semafor zliczający wolne miejsca
M : semafor chroniący kolejkę

```
1: function CONSUME()
2:     P(A, 1)
3:     P(M, 1)
4:     let record = NULL
5:     record ⇐ queue
6:     V(M, 1)
7:     V(F, 1)
8:     return record
9: end function
```

złożona, gdyż podczas swojej pracy zakłada wywoływanie obu algorytmów (ilościowego i jakościowego) oraz stosowanie się do zwróconych przez nie wyników. Wyniki te to (w przypadku algorytmu ilościowego) para wartości, z których pierwsza (*load*) informuje o liczbie stron do załadowania, a druga (*reminder*) o minimalnej liczbie rekordów w kolejce, po osiągnięciu której powinno się rozpocząć jej uzupełnianie oraz zbiór parametrów zapytania (w przypadku algorytmu jakościowego). Ponadto proces producenta korzysta z dwóch kolejnych zmiennych: *loaded* i *remaining*. Przechowują one informację o wartościach zwróconych w poprzednim wywołaniu algorytmu ilościowego, z zastrzeżeniem, że zmienna pierwsza z nich przyjmuje wartość liczby faktycznie załadowanych stron. W szczególnych przypadkach może być ona inna, aniżeli zalecana przez algorytm wartość *load*. Różnica taka może być skutkiem zmiany lokalizacji danych i wynikłej z tego zmiany rozmiaru strony, co ma miejsce na przykład przy przełączeniu się na strumień bieżący.

Pierwszym etapem procedury ładowania jest wywołanie algorytmu ilościowego oraz jakościowego. W następnej kolejności wyznaczana jest wartość zmiennej *permits* oznaczającej liczbę rekordów, która powinna być skonsumowana, aby rozpocząć proces ładowania. Zmienna ta jest przekazywana do procedury `P()` semafora pilnującego wolnego miejsca w kolejce (linia 8). Wątek producenta jest wstrzymany tak długo, aż liczba skonsumowanych rekordów (wartość zmiennej całkowitej skojarzonej z semaforem F) nie osiągnie przynajmniej wartości równej wartości zmiennej *permits*. Po wznowieniu wątku producenta następuje zgłoszenie żądania do modułów wchodzących w skład warstwy listy. Należy tutaj zauważyć, iż w zależności od tego, czy agregaty będące przedmiotem zapytania były wcześniej wytworzone, czy nie, ich uzyskanie może zostać zlecone modułowi materializacji lub modułom

Algorytm 7.10 Proces producenta

Stałe:

queue : kolejka iteratora
A, F, M : semafory

```
 1: procedure PRODUCE()
 2:     let loaded = queue.size()
 3:     let remaining = 0
 4:     loop
 5:         let [load, remainder] = QNTPFA()
 6:         let parameters = QLTPFA()
 7:         let permits = remaining + loaded − remainder
 8:         P(F, permits)
 9:         let records = ∅
10:         for 1 ... load do
11:             let page = ∅
12:             run MAT(parameters)
13:             receive page
14:             if page = NULL then
15:                 run REC(parameters)
16:                 run AGG(parameters)
17:                 receive page
18:             end if
19:             records ← records ∪ page
20:         end for
21:         loaded ← records.size()
22:         remaining ← reminder
23:         P(M, 1)
24:         records ⇒ queue
25:         V(M, 1)
26:         V(A, loaded)
27:     end loop
28: end procedure
```

rekonstrukcji i agregacji. Po uzyskaniu żądanych rekordów, dokonywana jest aktualizacja zmiennych *loaded* i *remaining*, a następnie otrzymane rekordy są dopisywane do kolejki układu stronicowania silnika CUBIT. Ta ostatnia czynność obwarowana jest, podobnie jak w przypadku procedury pobierania, wywołaniem procedur `P()` i `V()` semafora M strzegącego dostępu do kolejki.

Najważniejszymi składowymi układu stronicowania silnika CUBIT są dwa wspomniane algorytmy wypełniania stron. Dokładny opis algorytmu ilościowego znajduje się w rozdziale 8. Działanie algorytmu jakościowego jest wciąż przedmiotem otwartych badań – jego zasada sprowadza się do ustalenia wartości parametrów kolejnego zapytania. W przypadku kontekstowych danych czasowych parametry te sprowadzają się do znacznika czasowego i potrzeby jego aktualizacji, w przypadku kontekstowych danych przestrzennych, zbiór parametrów powiększa się o współrzędne geograficzne, a w przypadku kontekstowych danych środowiskowych – o inne atrybuty.

7.1.4 Widok i iterator

Moduły widoku i iteratora stanowią najbardziej zewnętrzny komponent silnika CUBIT i odpowiadają za prezentację wyników oraz interakcję z użytkownikiem. Pierwszy z omawianych modułów jest strukturą danych przechowującą aktualnie przeglądany fragment danych (rekord lub grupę rekordów), w formacie naturalnym dla tego fragmentu:

a) w przypadku kontekstowych danych czasowych, widok przechowuje fragment strumienia (podobnie jak kolejka), lecz odczyt poszczególnych rekordów nie wiąże się z ich usunięciem – użytkownik ma możliwość przechodzenia w przód i w tył po widoku oraz przewijania go, zastępując tym samym najstarsze dane nowszymi;

b) w przypadku kontekstowych danych przestrzennych, widok ma postać tablicy dwuwymiarowej, przechowującej pojedynczy klaster danych lub ciągu takich tablic (tablicy trójwymiarowej), w których zapisane są kolejne klastry – taka organizacja umożliwia łatwe przechodzenie po wymiarach przestrzeni oraz dokonywanie operacji *pivot* (zmiany kolejności wymiarów), a także *slice* i *dice*, znanych z klasycznych hurtowni danych;

c) w przypadku kontekstowych danych środowiskowych, widok przyjmuje postać wielowymiarowej tablicy, przechowującej aktualnie przeglądany blok danych – dzięki temu użytkownik może przeglądać agregaty przechodząc po wartościach różnych atrybutów oraz zmieniać zakres czasu i przestrzeni, przenosząc się do sąsiadujących bloków.

Iterator jest obiektem udostępniającym metody pozwalające na przechodzenie po widoku w różnych kierunkach i sukcesywne przeglądanie kolejnych

rekordów. Udostępnia on zbiór metod umożliwiających poruszanie się po aktualnym widoku oraz jego przesuwanie na sąsiadujący zakres (czasu lub przestrzeni, w zależności od typu danych kontekstowych).

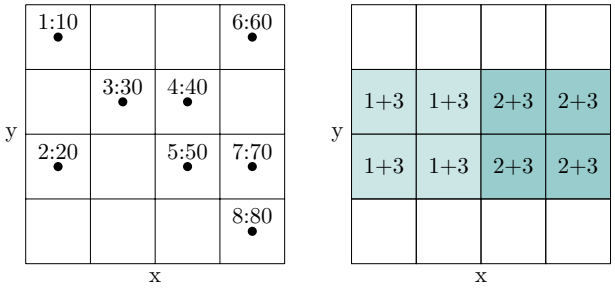
W stosunku do swojego pierwowzoru, iteratora Materializowanej Listy Agregatów (MAL), iterator silnika CUBIT charakteryzuje się większymi możliwościami, jego interfejs bardziej też przypomina kostkę danych OLAP niż listę. Poniżej znajduje się uproszczona propozycja zbioru metod udostępnianych przez iterator silnika CUBIT:

```
cubit.dom(...).next()        // przechodzenie po dziedzinie
              .prev()        // do przodu lub do tyłu,
              .value()       // odczyt wartości,
              .drill()       // uszczegółowienie
              .roll()        // lub uogólnienie
     .dim(...).next()        // iterowanie po wymiarze
              .prev()        // do przodu lub do tyłu,
              .value()       // odczyt wartości,
              .slice(...)    // operacja slice
              .drill()       // uszczegółowienie
              .roll()        // lub uogólnienie
     .measure(...)           // odczyt wartości agregatu
     .pivot(...)             // obrót kostki
     .dice(...)              // operacja dice
```

Dziedziną strumieniowej kostki danych udostępnianej przez silnik CUBIT jest zazwyczaj czas lub przestrzeń. W przypadku kontekstowych danych czasowych, czas jest zarówno dziedziną, jak i wymiarem. W przypadku kontekstowych danych przestrzennych, czas jest dziedziną, a wymiarami współrzędne geograficzne. Poruszanie się po dziedzinie powoduje doczytywanie nowych klastrów, które można przeglądać względem wymiarów przestrzennych. W przypadku kontekstowych danych środowiskowych, zarówno czas, jak i przestrzeń są dziedziną kostki – użytkownik może przesuwać widok na sąsiadujące w czasoprzestrzeni bloki oraz iterować po wymiarach (atrybutach) wewnątrz danego bloku.

7.2 Wielowymiarowy bitowy indeks zakresowy

Realizacja zapytań agregacyjnych dla danych kontekstowych (czasowych, przestrzennych i środowiskowych) może być znacząco przyspieszona przez materializację części z agregatów, czyli ich utrwalenie w bazie danych. Dzięki temu, późniejsze zapytanie o tych samych lub zbliżonych parametrach, może zostać zrealizowane przez odczyt wszystkich lub części zapisanych wcześniej agregatów. Nie zawsze jednak korzystanie z materializacji jest możliwe – jak choćby w przypadku zapytania wykorzystującego inne funkcje agregu-



(a) zbiór danych (b) podział przestrzeni

Rysunek 7.3: Przykładowy zbiór danych oraz zapytanie

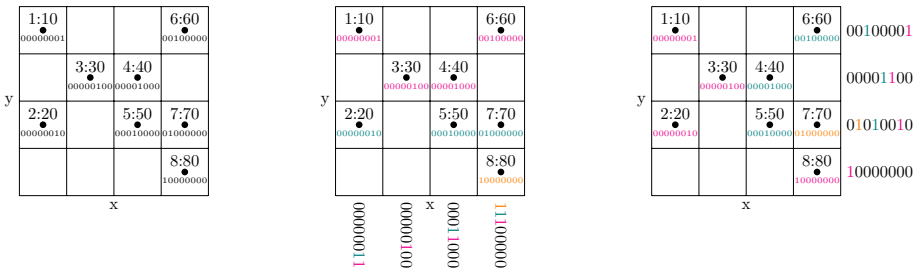
jące lub odnoszącego się do innych danych, pomimo że zostało ono zdefiniowane w tych samych oknach co zapytanie zmaterializowane. W takich przypadkach możliwe jest jednak wykorzystanie pomocniczych struktur indeksujących, które zapamiętują część informacji na temat zakresu rekordów objętych kolejnymi położeniami okien zapytania.

Dla strumieniowej hurtowni danych kontekstowych proponuje się wprowadzenie *bitowego indeksu zakresowego* BRI (ang. *Bitwise Range Index*) na potrzeby wspomagania procesu realizacji zapytań w torze kontekstu przestrzennego oraz środowiskowego – zakłada się, że agregacja tego typu danych kontekstowych jest znaczenie bardziej kosztowna od agregacji danych czasowych. Idea tego indeksu bazuje na bitowej reprezentacji rozkładu danych względem wartości poszczególnych wymiarów (atrybutów wymiarowych) i stanowi rozwinięcie koncepcji przedstawionej w patentach własnych [44, 41]$^{\mathrm{W}}$.

Na potrzeby dalszych rozważań wykorzystany będzie prosty przykład: osiem rekordów w dwuwymiarowej przestrzeni (o współrzędnych x i y) zostanie poddanych zapytaniu typu *roll-up* (podobnym do omówionego w rozdziale 5.3.2). Zapytanie to definiuje okno przestrzenne, które przesuwa się wzdłuż wymiaru x, a następnie obejmuje cały jego zakres, przy czym dla wymiaru y jest stałe. Rysunek 7.3 przedstawia powyższy przykład – zastosowany tam zapis 1:10 oznacza: rekord numer 1 o wartości 10. Kolejne położenia okna przestrzennego zostały ponumerowane począwszy od wartości 1.

7.2.1 Kodowanie bitowe rozkładu danych

Niezależnie od rodzaju wielowymiarowych danych kontekstowych (przestrzenne lub środowiskowego) klastry lub bloki składają się z grupy rekordów, których kolejność nie jest jednoznacznie określona. Wynika to właśnie z faktu ich wielowymiarowości – nie jest możliwe uporządkowanie danych ze względu na wszystkie wymiary na raz. Jest to sytuacja odmienna w stosun-



(a) kodowanie rekordów (b) wektor rozkładu D_x (c) wektor rozkładu D_y

Rysunek 7.4: Etapy wyznaczania wektorów rozkładu w dwuwymiarowej przestrzeni

ku do kontekstowych danych czasowych, które są uporządkowane względem czasu, który jest dla tych danych jedynym wymiarem.

Konsekwencją rzeczonego braku uporządkowania jest bardzo czasochłonne sprawdzanie zakresów kolejnych położeń okien przestrzennych lub uogólnionych – fakt, iż dany rekord należy bądź nie do aktualnego okna w żaden sposób nie warunkuje podobnej przynależności dla sąsiedniego rekordu (jak w przypadku kontekstowych danych czasowych). Proponowany indeks BRI stara się wyjść na przeciw temu problemowi poprzez przyporządkowanie każdego rekordu do określonych przedziałów wartości poszczególnych atrybutów wymiarowych. Dodatkowo, indeks ten, ze względu na zastosowanie kodowania binarnego i zastąpienie operacji porównania działaniami arytmetycznymi na wektorach i macierzach, jest łatwo podatny na zrównoleglenie – zarówno na poziomie bitowym, jak i na poziomie danych.

Wspomniane przyporządkowanie wykorzystuje kodowanie bitowe *1 z n* (ang. *one-hot*) – każdemu rekordowi w klastrze lub bloku (w skrócie: w sekwencji) przypisywany jest kod będący liczbą binarną z tylko jedną jedynką na pozycji odpowiadającej numerowi rekordu w sekwencji. Optymalny rozmiar sekwencji uzależniony jest od długości słowa maszynowego (dla systemu 64 bitowego będzie to 64 lub wielokrotność tej wartości). Dla hipotetycznej czteroelementowej sekwencji, kolejne pomiary będą kodowane w następujący sposób: 0001, 0010, 0100, 1000.

Ponadto zakłada się, że dla każdego z wymiarów zdefiniowana jest (w zależności od preferencji użytkownika) minimalna jednostka agregacji MAU (ang. *Minimal Aggregation Unit*), wyznaczająca zarazem najdrobniejszy poziom granulacji agregacji. Kody binarne generowane dla każdej sekwencji danych mają postać zbioru wektorów zwanych *wektorami rozkładu D*. Dla każdego z wymiarów wyznaczany jest jeden wektor rozkładu o liczbie elementów równej liczbie wszystkich jednostek agregacji dla tego wymiaru. Jego elementy stanowią bitową sumę logiczną kodów pomiarów mających część wspólną z poszczególnymi jednostkami agregacji. Ilustruje to rysunek 7.4 przedstawiający kolejne etapy wyznaczenia wektorów rozkładu dla przykła-

du omówionego na początku tego rozdziału.

Na rysunku 7.4b, na dole siatki znajdują się elementy wektora rozkładu dla wymiaru poziomego (D_x), a na rysunku 7.4c, po prawej, dla wymiaru pionowego (D_y), z kolorystycznym oznaczeniem poszczególnych rekordów oraz odpowiadających im pozycjom w słowie maszynowym. Wektory rozkładu dla tego przykładu, w sposób formalny mogą zostać zapisane jako:

$$D_x = \begin{bmatrix} 00000011 \\ 00000100 \\ 00011000 \\ 11100000 \end{bmatrix}, \qquad D_y = \begin{bmatrix} 00100001 \\ 00001100 \\ 01010010 \\ 10000000 \end{bmatrix}. \tag{7.1}$$

Warto zwrócić uwagę na fakt, iż bitowy sposób przedstawienia rozkładu rekordów w przestrzeni umożliwia łatwe wyznaczenie podzbioru tych rekordów, które należą do dowolnego obszaru, z dokładnością do minimalnych jednostek agregacji agregacji MAU. Proces ten polega na dokonaniu bitowej sumy logicznej wszystkich elementów wektora podziału należących do żądanego obszaru w obrębie odpowiadającego wymiaru, powtórzeniu tej czynności dla wszystkich pozostałych wymiarów, a następnie na dokonaniu bitowego iloczynu logicznego uzyskanych wyników.

Dla rozpatrywanego przykładu, kody rekordów należących do obszaru obejmującego środkowe cztery jednostki agregacji można wyznaczyć w następujący sposób (operacje | oraz & oznaczają odpowiednio sumę i iloczyn logiczny realizowane w sposób bitowy): (00000100 | 00011000) & (00001100 | 01010010) = 00011100 & 01011110 = 00011100. Oznacza to, że rekordy o numerach: 3, 4, 5 należą do wskazanego obszaru, gdyż bity na odpowiadających im pozycjach (licząc od prawej) mają wartość 1.

Algorytm 7.11 przedstawia równoległą operację wyznaczania wektora rozkładu dla wybranego atrybutu wymiarowego. W obliczeniach udział bierze tyle wątków, ile jest MAU dla danego wymiaru. Dla każdego rekordu sprawdzana jest jego przynależność do kolejnych jednostek agregacji, co odbywa się za pomocą sprawdzenia dwóch nierówności (linie 6 i 7). Kod danego rekordu jest obliczany w linii 9, poprzez przesunięcie bitowe w lewo odpowiednio wartości 0 (dla rekordów nie należących do danej jednostki agregacji) lub 1 (dla rekordów należących do danej MAU) o liczbę miejsc równą numerowi bieżącego rekordu. W wyniku uzyskiwany jest kod rekordu należącego do danej MAU lub wartość 0, kodująca brak rekordu. Otrzymane kody jest akumulowane w zmiennej *sum* przy wykorzystaniu sumy bitowej.

Algorytm 7.11 Równoległy algorytm wyznaczania wektorów rozkładu

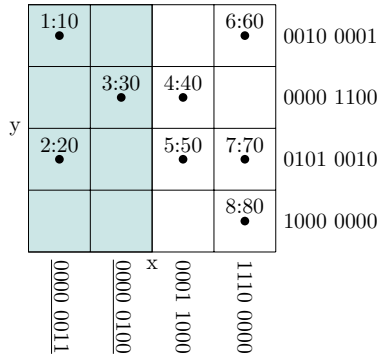
Stałe:
δ : długość MAU dla danego wymiaru
m : liczba MAU dla danego wymiaru
Wejście:
seq : sekwencja rekordów
n : liczba rekordów w sekwencji
$attr$: wybrany atrybut wymiarowy
Wyjście:
$D[]$: wektor rozkładu

```
1:  function DISTRIBUTIONVECTOR(seq, n, attr)
2:      let D[] = ∅                                   ▷ pusty wektor
3:      parfor mau ← 0 ... (m − 1) do                 ▷ równoległa pętla
4:          let sum = 0
5:          for record ← 0 ... (n − 1) do
6:              let c1 = seq[record].attr ⩾ mau · δ
7:              let c2 = seq[record].attr < (mau + 1) · δ
8:              let c = c1 ∧ c2                       ▷ iloczyn logiczny
9:              let code = c ≪ record                 ▷ przesunięcie bitowe
10:             sum ← sum | code                      ▷ suma bitowa
11:         end for
12:         D[mau] ← sum
13:     end parfor
14:     return D[]
15: end function
```

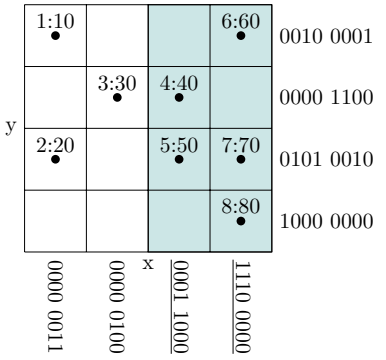
7.2.2 Kodowanie bitowe zapytania

Podobnie jak rozkład danych, również zapytanie (a dokładniej sposób propagacji okna) poddawane jest kodowaniu bitowemu. Dzięki temu możliwe jest zastosowanie operacji bitowych do wyodrębnienia podzbioru rekordów objętych zakresem zapytania. Rysunek 7.5 przedstawia kolejne położenia okna rozpatrywanego przykładowego zapytania, oddzielnie dla wymiaru x oraz y. Podkreśleniem wyróżniono te elementy wektorów rozkładu, które zostały objęte zakresem okna w danym położeniu.

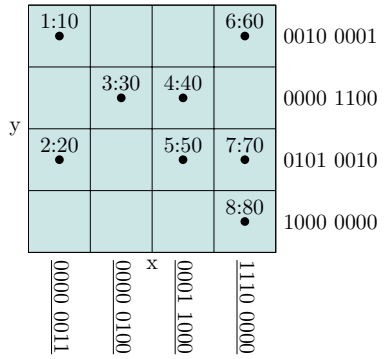
Propagacja okna wzdłuż wybranego wymiaru może zostać wyrażona za pomocą *macierzy wartości granicznych* B, która ma tyle wierszy, ile jest różnych położeń okna dla danego wymiaru oraz dwie kolumny – zawierające wartość początkową i końcową tego atrybutu wymiarowego. Zakłada się, że macierze te powstają na skutek przetworzenia zapytania pochodzącego od użytkowników. W przypadku rozpatrywanego zapytania, przy założeniu, że poszczególne jednostki agregacji mają szerokość 10, macierze wartości



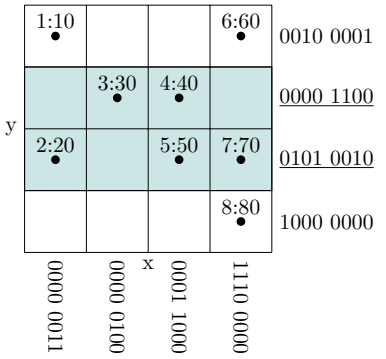
(a) okno nr 1 dla wym. x



(b) okno nr 2 dla wym. x



(c) okno nr 3 dla wym. x



(d) okno dla wym. y

Rysunek 7.5: Zasięg poszczególnych okien dla przykładowego zapytania

granicznych przyjmują postać:

$$B_x = \begin{bmatrix} 0 & 20 \\ 20 & 40 \\ 0 & 40 \end{bmatrix}, \qquad B_y = \begin{bmatrix} 10 & 30 \end{bmatrix}. \tag{7.2}$$

Na podstawie macierzy wartości granicznych oraz liczby i rozmiaru jednostek agregacji, możliwe jest wyznaczenie *macierzy propagacji okna* W. Każda taka macierz zawiera informacje na temat przesuwania się okna po kolejnych jednostkach agregacji. Liczba wierszy macierzy propagacji okna odpowiada liczbie okien zdefiniowanych dla danego wymiaru w zapytaniu, a liczba jej kolumn odpowiada liczbie jednostek agregacji dla danego wymiaru. Elementy tej macierzy to wartości 1 lub 0, przy czym wartości 1 dla każdego okna znajdują się w tych kolumnach, które odpowiadają jednostkom agregacji mającym wspólną część z danym oknem. Pozostałe elementy macierzy propagacji okna mają wartość 0. Dla rozpatrywanego zapytania i zbioru

danych, macierze propagacji okna przyjmują postać:

$$W_x = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \qquad W_y = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}. \tag{7.3}$$

Różnica pomiędzy macierzami B a W polega na tym, że te pierwsze zawierają wyłącznie wartości graniczne kolejnych położeń okna i w żaden sposób nie odnoszą się do objętych tym oknem obszarów agregowanej przestrzeni. Z kolej macierze propagacji okna nie zawierają w ogóle wartości granicznych, a jedynie binarną informację przynależności poszczególnych jednostek agregacji do kolejnych położeń okna.

Algorytm 7.12 przedstawia równoległą operację wyznaczania macierzy propagacji okna dla pojedynczego wymiaru. Jest ona realizowana przez liczbę wątków równą iloczynowi liczby okien zdefiniowanych w zapytaniu dla danego wymiaru oraz liczby jednostek agregacji dla tego wymiaru. W linii 7 następuje sukcesywne wpisywanie do pustej początkowo macierzy W wartości logicznych będących wynikiem koniunkcji dwóch warunków sprawdzanych w poprzednich dwóch liniach.

Algorytm 7.12 Równoległy algorytm wyznaczania macierzy propagacji okna

Stałe:

δ : długość MAU dla danego wymiaru
m : liczba MAU dla danego wymiaru
w : liczba okien dla danego wymiaru

Wejście:

$B[][]$: macierz wartości granicznych

Wyjście:

$W[][]$: macierz propagacji okna

```
1:  function propagationMatrix(B[][])
2:      let W[][] = ∅                                  ▷ pusta macierz
3:      parfor window ← 0 ... (w − 1) do               ▷ równoległa pętla
4:          parfor mau ← 0 ... (m − 1) do              ▷ równoległa pętla
5:              let c1 = B[window][1] ⩾ mau · δ        ▷ koniec okna
6:              let c2 = B[window][0] < (mau + 1) · δ  ▷ początek okna
7:              W[window][mau] ← c1 ∧ c2               ▷ iloczyn logiczny
8:          end parfor
9:      end parfor
10:     return W[][]
11: end function
```

Macierze propagacji okna określają, w jaki sposób okno „przemieszcza" się w danym wymiarze, jednak nie są powiązane w żaden sposób z danymi, które mają być docelowo objęte zasięgiem zapytania. W celu realizacji zapytania, czyli wyznaczenia podzbioru wspomnianych danych, należy zestawić macierze propagacji okna W z wektorami podziału D, oddzielnie dla każdego z wymiarów.

Opisane powyżej działanie sprowadza się do wykonania zmodyfikowanego iloczynu wewnętrznego pomiędzy macierzą propagacji okna W a wektorem podziału D. Iloczyn wewnętrzny dwóch macierzy (wektorów) polega na wyznaczeniu sumy iloczynów odpowiadających sobie elementów z obu macierzy (wektorów). Jego modyfikacja polega na zastosowaniu operacji sumy bitowej, zamiast standardowej sumy arytmetycznej. W wyniku powstaje *wektor zapytania* Q, o liczbie elementów równej liczbie okien zdefiniowanych w zapytaniu dla danego wymiaru. Jego elementy zawierają kody rekordów objętych oknem dla danego wymiaru, o położeniu odpowiadającym pozycji elementu w wektorze. Dla rozpatrywanego zapytania oraz zbioru danych wektory zapytania przyjmują następującą postać:

$$\begin{aligned} Q_x = W_x \cdot D_x &= \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 00000011 \\ 00000100 \\ 00011000 \\ 11100000 \end{bmatrix} = \begin{bmatrix} 00000111 \\ 11111000 \\ 11111111 \end{bmatrix}, \\ Q_y = W_y \cdot D_y &= \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 00100001 \\ 00001100 \\ 01010010 \\ 10000000 \end{bmatrix} = \begin{bmatrix} 01011110 \end{bmatrix}. \end{aligned} \tag{7.4}$$

Wektor zapytania Q_y, pokazany w równaniu 7.4, zawiera pojedynczą wartość binarną 01011110, której interpretacja jest następująca: rekordy o numerach 2, 3, 4, 5, 7 należą do okna zapytania (które ma tylko jedno położenie) dla wymiaru y – jest to zgodne z rysunkiem 7.5d.

Operacja wyznaczania wektorów zapytania może zostać zrealizowana równolegle (algorytm 7.13), w podobny sposób co standardowa operacja mnożenia macierzy. W obliczeniach bierze udział tyle wątków, ile jest okien zdefiniowanych w zapytaniu dla danego wymiaru.

Wektory zapytania zawierają informację o tym, które pomiary zostały objęte oknem zapytania w obrębie danego wymiaru. W celu znalezienia informacji globalnej, tj. dotyczącej wszystkich wymiarów, należy połączyć ze sobą wszystkie wektory zapytania. Można to uzyskać wykonując pomiędzy nimi wszystkimi zmodyfikowaną operację iloczynu zewnętrznego. Iloczyn zewnętrzny macierzy (wektorów) polega na wyznaczeniu iloczynów wszystkich kombinacji par elementów z obu macierzy (wektorów). Jego modyfikacja polega na zastosowaniu iloczynu bitowego, zamiast standardowego iloczynu arytmetycznego.

Algorytm 7.13 Równoległy algorytm wyznaczania wektorów zapytania

Stałe:
m : liczba MAU dla danego wymiaru
w : liczba okien dla danego wymiaru
Wejście:
$W[][]$: macierz propagacji okna
$D[]$: wektor rozkładu
Wyjście:
$Q[]$: wektor zapytania

```
 1: function QUERYVECTOR(W[][], D[])
 2:     let Q[] = ∅                                   ▷ pusty wektor
 3:     parfor window ← 0 ... (w − 1) do              ▷ równoległa pętla
 4:         sum ← 0
 5:         for mau ← 0 ... (m − 1) do
 6:             let c = W[window][mau] · D[mau]
 7:             sum ← sum | c                         ▷ suma bitowa
 8:         end for
 9:         Q[window] ← sum
10:     end parfor
11:     return Q[]
12: end function
```

W wyniku przeprowadzenia powyższej operacji powstaje wielowymiarowa macierz agregacji A, o wymiarowości równej wymiarowości zapytania (a zarazem równej liczbie wektorów zapytania). W macierzy n-wymiarowej każdy element ma n indeksów (klasyczne macierze są dwuwymiarowe). Kolejność mnożenia wektorów zapytania wynika z preferencji użytkownika odnośnie kolejności przedstawiania wyników. Zmiana tej kolejności w pewnym sensie odpowiada operatorowi OLAP *pivot*. Dla rozpatrywanego przykładu, wielowymiarowa macierz agregacji A ma postać klasycznej, dwuwymiarowej macierzy:

$$A = Q_x \otimes Q_y = \begin{bmatrix} 00000111 \\ 11111000 \\ 11111111 \end{bmatrix} \otimes \begin{bmatrix} 01011110 \end{bmatrix} = \begin{bmatrix} 00000110 \\ 01011000 \\ 01011110 \end{bmatrix}. \tag{7.5}$$

Mnożąc wektor zapytania dla wymiaru x Q_x przez wektor zapytania dla wymiaru y Q_x, uzyskiwana jest wielowymiarowa macierz agregacji A, której elementy ułożone są w pierwszej kolejności względem wymiaru x, a następnie wymiaru y. Każdy element takiej macierzy zawiera kody rekordów, które biorą udział w obliczeniach dla wskazanego położenia okna.

Wyznaczenie wielowymiarowej macierzy agregacji może zostać zrealizowane w sposób równoległy, przy użyciu tylu wątków, ile jest okien w całym

zapytaniu (dla wszystkich wymiarów). Liczba ta jest iloczynem liczby okien dla poszczególnych wymiarów. W poprzednich algorytmach liczby te były oznaczane po prostu jako w, gdyż dotyczyły one wyłączenie pojedynczego wymiaru. Niech w_i oznacza liczbę okien dla wymiaru i, a liczba wymiarów jest równa d. Algorytm 7.14 przedstawia równoległy sposób wyznaczania wielowymiarowej macierzy agregacji.

Algorytm 7.14 Równoległy algorytm wyznaczania wielowymiarowej macierzy agregacji

Stałe:
d : liczba wymiarów
w_i : liczba okien dla i-tego wymiaru
Wejście:
$Q_i[]$: wektor zapytania dla i-tego wymiaru
Wyjście:
$Q[]$: macierz agregacji

```
1: function AGGREGATIONMATRIX(Q_x[], Q_y[])
2:     let A[][] = ∅                                  ▷ pusta macierz
3:     parfor x ← 0 ... (w_x − 1) do                  ▷ równoległa pętla
4:         parfor y ← 0 ... (w_y − 1) do              ▷ równoległa pętla
5:             A[x][y] ← Q_x[x] & Q_y[y]              ▷ iloczyn bitowy
6:         end parfor
7:     end parfor
8:     return A[][]
9: end function
```

7.2.3 Proces realizacji zapytania

Wielowymiarowa macierz agregacji zawiera informację o wszystkich danych biorących udział w zapytaniu, uwzględniając przy tym podział na okna. Każdy jej element stanowi informację wejściową dla docelowej funkcji agregującej, informując ją, które rekordy danej sekwencji powinny składać się na dany agregat. W celu ekstrakcji tych informacji, konieczne jest rozkodowanie zapisanych binarnie danych do postaci wektorów maski. Wektor taki ma tyle elementów, ilu bitowy jest każdy element wielowymiarowej macierzy agregacji. Liczba ta odpowiada także liczbie wszystkich rekordów w przetwarzanej sekwencji. Każdemu elementowi wielowymiarowej macierzy agregacji odpowiada jeden wektor maski. Elementy wektora maski to wartości 0 lub 1, na pozycjach odpowiadających kolejnym bitom rozkodowywanego elementu wielowymiarowej macierzy agregacji. Ze względu na obrane kodowanie rekordów, począwszy od najmłodszego bitu, a skończywszy na bicie najstarszym, wektory maski są stanowią niejako „lustrzane odbicie" elementów

wielowymiarowej macierzy agregacji.

Wektory maski, tworzone poprzez rozkodowywanie kolejnych elementów wielowymiarowej macierzy agregacji tworzą macierz maski M, poprzez zapisywanie kolejnych wektorów maski wierszami. Dla pokazanej w równaniu 7.5 wielowymiarowej macierzy agregacji A, uzyskana macierz maski M przedstawia się następująco:

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{7.6}$$

Operacja wyznaczania macierzy maski może zostać zrealizowana w sposób równoległy, przy wykorzystaniu wątków w liczbie równej iloczynowi liczby okien w zapytaniu oraz liczby rekordów w przetwarzanej sekwencji. Ilustruje to algorytm 7.15. Wielowymiarowa macierz agregacji A została przedstawiona jako tablica jednowymiarowa, gdyż w ten właśnie sposób każda macierz o dowolnej liczbie wymiarów jest zapisana w pamięci. Na podstawie numeru rekordu wyznaczany jest jego kod w linii 5 (operacja $1 \ll record$). Następnie, w linii 6 następuje wykonanie operacji iloczynu bitowego pomiędzy danym elementem wielowymiarowej macierzy agregacji a otrzymanym w poprzednim kroku kodem danego rekordu. W wyniku uzyskiwana jest wartość 0 (gdy dany rekord nie bierze udziału w zapytaniu) lub wartość różna od 0, mająca w zapisie binarnym jedną jedynkę na pozycji odpowiadającej przetwarzanemu rekordowi. W linii 7 odbywa się przesunięcie bitowe w prawo, w celu uzyskania wartości 0 lub 1.

Przeznaczeniem macierzy maski jest jednoznaczne wskazanie, które rekordy przetwarzanej sekwencji mają zostać wzięte pod uwagę, a które odrzucone przy wykonywaniu funkcji agregującej dla danego okna. Dla rozpatrywanego od początku przykładu, kolejne wiersze macierzy maski przedstawionej równaniem 7.6 odpowiadają podzbiorom rekordów przedstawionym na rysunku 7.6. Kolorem czerwonym oznaczono rekordy objęte zakresem okna, które z kolei przedstawiono kolorem błękitnym. Przykładowo, na rysunku 7.6b wiersz macierzy maski ma wartość $[0, 0, 0, 1, 1, 0, 1, 0]$, co przekłada się na zakwalifikowanie rekordów o numerach: 4, 5, 7 – dokładnie tych, którym odpowiadają pozycje jedynek w przytoczonym wierszu.

W przypadku zastosowania operacji sumowania (np. w celu wyznaczenia średniej arytmetycznej) jako funkcji agregującej, obliczenie gotowego agregatu sprowadza się do przemnożenia macierzy maski M przez wektor utworzony z agregowanych wartości atrybutów V. Wówczas rekordy, które nie biorą udziału w zapytaniu, mają zerowany atrybut podlegający agregacji, przez co staje się on elementem neutralnym dodawania i nie wpływa na wynik agregacji. Operacja mnożenia macierzy może zostać zrealizowana w sposób równoległy, przy współudziale tylu wątków, ile wierszy ma macierz maski M. Jako, że algorytm ten jest dobrze znany i wielokrotnie opisany w literaturze, nie zostanie tutaj przedstawiony.

Algorytm 7.15 Równoległy algorytm wyznaczania macierzy maski

Stałe:

n : liczba rekordów w sekwencji

w : całkowita liczba okien

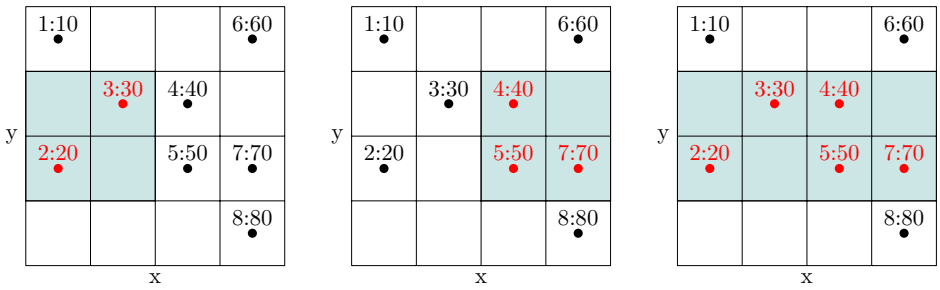
Wejście:

$A[]$: wielowymiarowa macierz agregacji

Wyjście:

$M[][]$: macierz maski

```
1:  function MASKMATRIX(n, w, A[], M[][])
2:      let M[][] = ∅                                  ▷ pusta macierz
3:      parfor window ← 0 ... (w − 1) do               ▷ równoległa pętla
4:          parfor record ← 0 ... (n − 1) do           ▷ równoległa pętla
5:              let code = 1 ≪ record                  ▷ przesunięcie bitowe
6:              let exctracted = A[window] & code      ▷ iloczyn bitowy
7:              let flag = extracted ≫ record          ▷ przesunięcie bitowe
8:              M[window][record] ← flag
9:          end parfor
10:     end parfor
11:     return M[][]
12: end function
```



(a) 0 1 1 0 0 0 0 0 (b) 0 0 0 1 1 0 1 0 (c) 0 1 1 1 1 0 1 0

Rysunek 7.6: Wizualizacja znaczenia kolejnych wierszy macierzy maski

Biorąc za przykład wspomnianą powyżej operację wyznaczania sumy, przemnożenie macierzy maski przez wektor agregowanych wartości skutkuje powstaniem wektora wartości końcowych R, który ma tyle elementów, ile zostało zdefiniowanych okien w zapytaniu. Elementy tego wektora to agregaty dla kolejnych okien. Dla rozpatrywanego przykładu, postać wektora R uzyskanego przez przemnożenie macierzy maski M przez wektor utworzony z wartości atrybutu podlegającego operacji agregacji V przedstawia równanie:

$$R = M \cdot V = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 20 \\ 30 \\ 40 \\ 50 \\ 60 \\ 70 \\ 80 \end{bmatrix} = \begin{bmatrix} 50 \\ 160 \\ 210 \end{bmatrix}. \tag{7.7}$$

W przypadku potrzeby obliczenia średniej arytmetycznej zamiast sumy, pomocna okazuje się wielowymiarowa macierz agregacji. Jej elementy to wartości binarne, o liczbie jedynek równej liczbie rekordów biorących udział w zapytaniu. Wiele procesorów sprzętowo wspiera operację zliczania jedynek w słowie maszynowym – operacja ta nazywa się *population count*. Wówczas, każdy element wektora finalnych wartości R zostaje podzielony przez wynik operacji *population count* i uzyskiwane są wartości średniej arytmetycznej. W przypadku innych funkcji agregujących należy, wykorzystując wartości macierzy maski, odrzucać te rekordy, dla których odpowiadająca wartość w macierzy maski wynosi zero.

Rozdział 8

Metody stronicowania pamięci w strumieniowych hurtowniach danych kontekstowych

Niniejszy rozdział przedstawia propozycję nowych ilościowych algorytmów wypełniania stron dla silnika CUBIT. Algorytmy te stanowią rozwinięcie istniejących w literaturze rozwiązań. Ich innowacja polega na zastosowaniu metod rozwiązywania wielokryterialnych problemów optymalizacyjnych w celu nadania algorytmom cech: adaptacyjności, elastyczności oraz określoności. W rozdziale przedstawiono kolejne kroki tworzenia nowych algorytmów: począwszy od analizy założeń i wymagań, poprzez teoretyczne rozważania nad postacią rozwiązania i funkcjami celu, a skończywszy na zapisie trzech proponowanych algorytmów.

8.1 Algorytmy wypełniania stron

Przedstawiony w rozdziale 7.1 silnik CUBIT, podobnie jak jego pierwowzór w postaci silnika MAL, opiera swoje działanie na mechanizmie stronicowania pamięci, którego zadaniem jest dynamiczne doczytywanie danych z pamięci trwałej lub tworzenie ich na bieżąco, a następnie dostarczanie ich użytkownikowi. Istotę działania tego układu stanowi para algorytmów wypełniania stron: ilościowy i jakościowy, które sterują pracą całego układu, decydując o ilości oraz rodzaju transmitowanych danych.

Działanie ilościowego algorytmu wypełniania stron ma kluczowe znaczenie z punktu widzenia efektywnego zarządzania pamięcią, co z kolei przekłada się na minimalizowanie opóźnienia w dostarczaniu danych, którego doświadcza użytkownik. Jako, że prace nad jakościowym algorytmem wciąż

są na etapie wstępnym, w niniejszym rozdziale omówiony zostanie w szczegółach algorytm ilościowy, zwany dalej po prostu *algorytmem wypełniania stron*.

8.1.1 Oryginalne algorytmy w silniku MAL

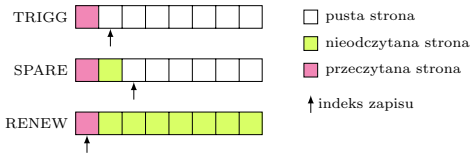
W silniku Materializowanej Listy Agregatów zaimplementowano trzy algorytmy wypełniania stron. Zostały one wyszczególnione ze względu na różnice w nadmiarze agregatów, które znajdowały się w kolejce iteratora. Naczelnym założeniem tych algorytmów było wykorzystanie statycznej tablicy o rozmiarze będącym wielokrotnością rozmiaru strony – tablica ta pełniła funkcję kolejki.

Każdy z trzech wspomnianych algorytmów śledził status odczytania załadowanych do tablicy stron i na podstawie tego zlecał proces wypełniania, przy czym zawsze uzupełniana była tylko pojedyncza strona. W rzeczywistości jednak sprowadzało się to do rozpoczęcia procesu ładowania zawsze po skonsumowaniu całej strony, a jedyna różnica pomiędzy algorytmami znajdowała się jedynie w liczbie pozostałych rekordów w tablicy (kolejce) – każdy algorytm rozpoczynał pracę od załadowania innej ich liczby.

Nazewnictwo trzech oryginalnych algorytmów wypełniania stron zostało zainspirowane ich charakterystyką pracy. Pierwszy algorytm nazwany został TRIGG (ang. *trigger* – wyzwalacz), gdyż uzupełnianie nowych danych było wyzwalane przeczytaniem ostatniego rekordu ze strony znajdującej się w kolejce. Algorytm ten ładował na początku tylko jedną stronę i cechował się przez to najmniejszą nadmiarowością (liczbą rekordów, które musiały być zapisane do tablicy przed ich odczytem), ale też był najbardziej wrażliwy na nierówne tempo konsumpcji danych.

Drugi algorytm (SPARE) swoją nazwę zawdzięczał zapasowi w postaci jednej dodatkowej strony (ang. *spare* – zapasowy). Stanowił on modyfikację poprzedniego algorytmu, poprzez dodanie drugiej strony ładowanej podczas inicjalizacji, która była zapasem rekordów podczas regularnej pracy. Algorytm rozpoczynał proces ładowania nowej strony zawsze, gdy pierwsza z dwóch stron w tablicy została w całości przeczytana.

Ostatni algorytm wypełniał na samym początku całą tablicę danymi, a każdą stronę uzupełniał zaraz po jej odczytaniu. Działanie to przyczyniło się do nadania mu nazwy RENEW (ang. *renew* – odnawiać). Cechował się największą nadmiarowością, ale również najmniejszą wrażliwością na wahania w szybkości konsumpcji danych. Rysunek 8.1 przedstawia podsumowanie działania wszystkich trzech oryginalnych algorytmów wypełniania stron. Dobrze widoczne są różnice w nadmiarze (nieodczytane strony) pomiędzy poszczególnymi algorytmami.



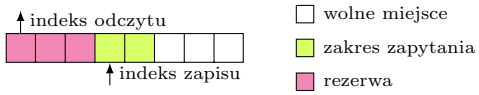
Rysunek 8.1: Porównanie algorytmów wypełniania stron w silniku MAL

8.1.2 Nowe algorytmy w silniku CUBIT

Ze względu na fakt, iż silnik CUBIT swój rodowód wywodzi właśnie z silnika MAL, konieczne było opracowanie nowego algorytmu (ilościowego), który odpowiadałby charakterystyce silnika CUBIT oraz był wolny od wad poprzednich rozwiązań. W przygotowywanej pracy własnej [68]^{W} przedstawiono propozycję nowej klasy algorytmów wypełniania stron, jako odpowiedzi na trzy podstawowe problemy istniejących dotychczas algorytmów.

Po pierwsze, stosowane do tej pory algorytmy wykorzystywały stałą, czyli niezależną od czynników zewnętrznych i niezmienną w czasie, strategię zasilania kolejek iteratorów MAL. Chociaż zdefiniowano trzy różne algorytmy i wskazano ich optymalne warunki pracy, wybór konkretnego z nich spoczywał na użytkowniku i jego wiedzy *a priori* odnośnie charakterystyki czasowej poszczególnych komponentów systemu. Ponadto, założenie o niezmienności tych charakterystyk nie zawsze jest spełnione w rzeczywistych systemach. W konsekwencji można wskazać pierwszą cechę nowej klasy algorytmów wypełniania stron: *adaptacyjność*, a więc zdolność do dopasowania się do aktualnych warunków pracy poprzez modyfikację swojego zachowania.

Drugim problemem było zastosowanie wielowątkowości w operacjach wejścia/wyjścia. Dotychczasowe algorytmy zakładały możliwość współbieżnego ładowania nowych danych do kolejek iteratorów MAL w taki sposób, że każda operacja zlecająca pobranie i zapisanie do kolejki pojedynczej strony odbywała się w osobnym wątku i była uruchamiana od razu w momencie spełnienia zdefiniowanych dlań warunków. Oznaczało to, że w przypadku zastosowania algorytmu o największej nadmiarowości i przy jednoczesnym bardzo wysokim tempie pobierania danych (w stosunku do tempa ich ładowania), kolejne żądania nowych danych były wystosowywane jeszcze przed zrealizowaniem poprzednich. Taka asynchroniczność procesu uzyskiwania nowych danych jest akceptowalna, ale pod warunkiem pełnej niezależności wykonywanych operacji wejścia/wyjścia, co nie zawsze ma miejsce w rzeczywistych systemach, a ponadto generuje ona dużą liczbę drobnych transakcji. Rozwiązaniem tego problemu może być synchroniczne zgłaszanie żądań o większą niż jeden liczbę stron do pobrania, co przekłada się na mniejszą liczbę transakcji o grubszej granulacji i umożliwia tym samym lepszą ich optymalizację przez system zarządzania bazą danych. W konsekwencji można wskazać drugą cechę nowej klasy algorytmów wypełniania stron: *elastyczność*, która wyraża się poprzez zgłaszanie zapotrzebowania na dowolną liczbę jednostek



Rysunek 8.2: Koncepcja adaptacyjnego algorytmu wypełniania stron

danych w trybie synchronicznym.

Ostatnim ze wspomnianych problemów był brak wzięcia pod uwagę aspektu zapewnienia krytycznych ograniczeń systemu, innych niż ilość dostępnej pamięci operacyjnej. Dotychczasowe prace badawcze skupiały się wyłącznie na tym ostatnim zagadnieniu, lecz nie zajmowały się w ogóle kwestią ograniczenia obciążenia systemu bazodanowego oraz dotrzymania pożądanego przez użytkownika tempa dostarczania nowych danych. Pierwsze ze wspomnianych ograniczeń nabiera dodatkowo znaczenia w przypadku dużej liczby instancji silnika CUBIT pracujących równolegle, gdyż każda z nich generuje strumień żądań, który następnie przekłada się na ciąg zapytań do wykonania. Drugie ograniczenie związane jest z minimalną częstotliwością dostarczania danych klientowi, a więc z maksymalnym czasem oczekiwania na kolejny agregat. W konsekwencji można wskazać trzecią cechę nowej klasy algorytmów wypełniania stron: *określoność*, rozumiany jako nieustanne dbanie o nieprzekroczenie krytycznych wartości ograniczeń systemu i dążenie do zapewnienia tym samym należytej jakości usług jego komponentów.

Trzy opisanej powyżej problemy stały się motywacją do stworzenia nowej klasy algorytmów wypełniania stron, którą można określić jako algorytmy *adaptacyjne*. Uzupełnieniem takiego nazewnictwa jest termin *stałe* odnoszący się do poprzedniej generacji algorytmów, ze względu na ich prostotę i stosowanie zawsze tych samych wartości parametrów. Algorytmy adaptacyjne charakteryzują się trzema cechami: adaptacyjnością, elastycznością i określonością – zmieniają swoje zachowanie w czasie, korzystają w pełni z możliwości oraz przestrzegają ograniczeń. Rysunek 8.2 przedstawia podsumowanie koncepcji nowej klasy algorytmów wypełniania stron. Zarówno rezerwa (liczba dostępnych rekordów w kolejce), jak i zakres zapytania (planowana liczba rekordów do załadownia) są zmienne i zależne od aktualnych warunków pracy.

8.2 Stronicowanie jako problem optymalizacyjny

Minimalnym zbiorem parametrów potrzebnym do przeprowadzenia procesu wypełniania stron jest uporządkowana para liczb naturalnych (x_l, x_r). Parametr x_l oznacza wymaganą liczbę jednostek danych do załadowania. Drugi z parametrów, x_r, określa progową liczbę rekordów w kolejce, po osiągnięciu której rozpoczyna się proces ładowania. Wspomnianą jednostką danych może być pojedynczy rekord bądź strona używana przy zapisie w pa-

mięciach trwałych. Parę (x_l, x_r) uzupełnia parametr początkowy i określający liczbę jednostek danych znajdujących się w kolejce układu stronicowania silnika CUBIT na początku jego pracy. Nie jest on jednak parametrem poddawanym optymalizacji, a raczej stałą wyznaczającą warunki początkowe.

Para parametrów (x_l, x_r) pozwala na sformułowanie zapytania kierowanego do źródła danych (parametr x_l) oraz jego uruchomienie w odpowiednim momencie czasu (parametr x_r). Znalezienie tej pary jest zadaniem dla algorytmu wypełniania stron i stanowi zarazem wielokryterialny problem optymalizacyjny: szukana jest taka para wartości, która z jednej strony jest dozwolona (spełnia ograniczenia systemu), a z drugiej strony zapewnia największą możliwą jakość usług zarówno dla klienta (konsumenta), jak i źródła danych (producenta). Te dwa ostatnie warunki są z natury przeciwstawne, gdyż zapewnienie wysokiej jakości usług dla klienta wiąże się z wyższym obciążeniem systemu zarządzania bazą danych, a z kolei minimalizacji tego obciążenia pociąga za sobą pogorszenie doświadczeń klienta.

8.2.1 Przestrzeń rozwiązań

Każde rozwiązanie problemu wypełniania stron, czyli para wartości parametrów (x_l, x_r), może zostać przedstawiona jako wektor rozwiązania $\mathbf{x} \in X$, należący do przestrzeni rozwiązań $X \subseteq \mathbb{N}^2$. Dopuszczalne rozwiązania stanowią podzbiór tejże przestrzeni i mogą zostać określone mianem zbioru decyzji i oznaczone jako $Q \subseteq X$. Czynnikiem warunkującym dopuszczalność rozwiązania są różnego rodzaju ograniczenia wyznaczające akceptowalne wartości obu parametrów. Ograniczenia te można podzielić na brzegowe, czyli dotyczące wyłącznie wartości każdego z obu parametrów osobno, jak i zachowawcze – związane z wzajemnymi zależnościami pomiędzy tymi parametrami.

Parametr x_l oznaczający liczbę stron, które powinny zostać załadowane, ogranicza od dołu wartość $l_0 = 1$. Z kolei nieprzekraczalną wartością odgórną jest liczba wynikająca z pojemności kolejki (oznaczona jako c) oraz rozmiaru samej jednostki danych (oznaczonego jako p): $l_1 = \left\lfloor \frac{c}{p} \right\rfloor$. Poniższe nierówności podsumowują ograniczenia parametru x_l:

$$\begin{aligned} l_0 \leqslant x_l \leqslant l_1, \\ 1 \leqslant x_l \leqslant \left\lfloor \frac{c}{p} \right\rfloor. \end{aligned} \tag{8.1}$$

Parametr x_r określający progową liczbę rekordów, której osiągnięcie skutkuje rozpoczęciem procesu ładowania jest od dołu ograniczony przez wartość $r_0 = 0$, gdyż podobnie jak poprzedni parametr, jest on wartością należącą do zbioru liczb naturalnych. Odgórne ograniczenie wynika z kolei z aktualnej liczby rekordów w kolejce (oznaczonej jako s), gdyż liczba ta maleje na skutek pobierania danych przez klienta, i może zostać oznaczone jako $r_1 = s$.

W konsekwencji, parametr x_r podlega poniższym ograniczeniom:

$$\begin{aligned} r_0 \leqslant x_r \leqslant r_1, \\ 0 \leqslant x_r \leqslant s. \end{aligned} \tag{8.2}$$

Podane powyżej ograniczenia są wartościami stałymi i dotyczą każdego z parametrów osobno. Jednakże należy wziąć jeszcze pod uwagę ograniczenia wiążące ze sobą oba parametry. Ich istnienie można uzmysłowić sobie w następujący sposób: pewne pary wartości parametrów x_l i x_r, choć spełniają osobno ograniczenia nałożone na każdy z parametrów, nie mogą ze sobą współistnieć lub ich współistnienie powoduje niepożądane skutki. Rzeczone ograniczenia mają postać nierówności $g(\mathbf{x}) \leqslant 0$, gdzie $g(\mathbf{x})$ jest funkcją wektora rozwiązań $\mathbf{x} = [x_l, x_r]$, a więc funkcją dwóch zmiennych: x_l i x_r.

Funkcja ta wykorzystuje również pewne stałe, takie jak pojemność kolejki c, rozmiar strony p, a także pewne zmienne, które podczas działania algorytmu wypełniania stron mają wartości ustalone. Do tych ostatnich zaliczyć można czas ładowania strony t_p oraz czas konsumpcji rekordu t_c. Wspomniane dwa parametry t_p oraz t_c są zmienne w czasie, a ich uzyskanie polega na cyklicznym dokonywaniu pomiarów oraz uśrednianiu zmierzonych wartości. Wartość parametru t_p jest w przybliżeniu równa przedziałowi czasu, przez który system zarządzania bazą danych jest w stanie odczytać i zwrócić pojedynczą stronę. Z kolei wartość parametru t_c jest w przybliżeniu równa czasowi przetwarzania pojedynczego rekordu przez klienta. Iloraz tych dwóch wartości jest wyznacznikiem niezrównoważenia całego układu producent-konsument:

$$\eta = \frac{t_p}{t_c}. \tag{8.3}$$

Wartość współczynnika niezrównoważenia η oznacza jednocześnie liczbę rekordów, która zostanie pobrana przez klienta podczas ładowania pojedynczej strony. W konsekwencji można sformułować warunek płynnej pracy układu stronicowania, wiążący ze sobą wartość η z wartością p: płynna praca jest możliwa tylko wówczas, gdy liczba konsumowanych rekordów w czasie produkcji pojedynczej strony jest niższa od rozmiaru tejże. Zależność tę ilustruje nierówność:

$$\eta \leqslant p. \tag{8.4}$$

Gdy warunek przedstawiony nierównością 8.4 jest niespełniony, to przy każdym ładowaniu uzupełnianych jest mniej rekordów niż jest konsumowanych, co w rezultacie prowadzi do sytuacji, w której klient jest zmuszony oczekiwać na nowe dane po pobraniu każdej strony.

W dalszych rozważaniach zakłada się, iż warunek płynnej pracy jest spełniony lub wymuszony przez wprowadzenie ograniczenia na minimalny czas konsumpcji. Wielkość ta może zostać obliczona zgodnie z równaniem:

$$\tau_c = \frac{t_p}{p}, \tag{8.5}$$

które powstało z przekształcenia równania maksymalnej dopuszczalnej wartości współczynnika niezrównoważenia: $\eta = p$.

Elementem wiążącym ze sobą wartości obu parametrów x_l i x_r jest niemożność przepełnienia niepustej kolejki. Suma liczby rekordów znajdujących się w kolejce oraz liczby ładowanych rekordów nie może przekroczyć pojemności kolejki. Zależność ta może zostać wyrażona za pomocą nierówności:

$$(x_r - x_l \cdot \eta) + x_l \cdot p \leqslant c. \tag{8.6}$$

Interpretacja pierwszego składnika sumy zawartej w nierówności 8.6 jest następująca: w momencie dopisywania nowych rekordów do kolejki, jej rozmiar jest równy różnicy $r - x_l \cdot \eta$, co wynika z liczby rekordów w kolejce na początku procesu ładowania (równej x_r) oraz liczby rekordów skonsumowanych podczas procesu ładowania (równej $x_l \cdot \eta$).

Należy mieć na uwadze fakt, iż nierówność 8.6 ma sens tylko wówczas, gdy różnica $r - x_l \cdot \eta$ jest nieujemna. Sytuacja, w której odjemnik jest większy występuje wtedy, gdy jeszcze przed załadowaniem nowych rekordów kolejka jest pusta, ale nie oznacza to, że liczba rekordów w niej obecnych (wartość rzeczonej różnicy) jest ujemna. W takim przypadku znaczenia nabiera opisane w nierówności 8.1 ograniczenie $l_1 = \frac{c}{p}$ wynikające z pojemności pustej kolejki, do którego z resztą sprowadza się nierówność 8.6 dla wartości granicznej $x_r = x_l \cdot \eta$. Po przekształceniu nierówności 8.6 otrzymuje się formułę:

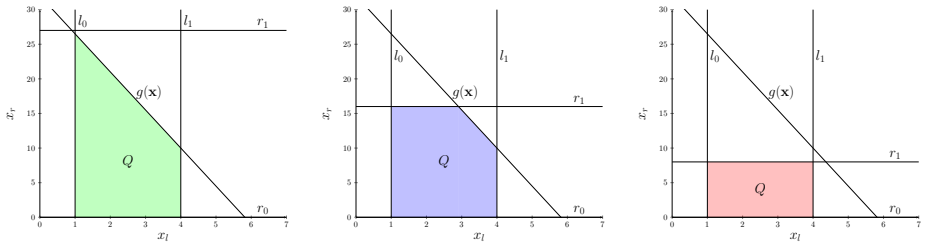
$$g(\mathbf{x}) = x_l \cdot (p - \eta) + x_r - c, \tag{8.7}$$

która definiuje $g(\mathbf{x})$ jako funkcję wektora decyzji $\mathbf{x}$. Powyższa funkcja ma sens wyłącznie dla nieujemnych wartości wyrażenia $p - \eta$, co wynika z warunku płynnej pracy $\eta \leqslant p$. W przypadku wartości współczynnika niezrównoważenia η większych od rozmiaru strony p, liczba skonsumowanych rekordów jest zawsze większa od liczby tych załadowanych, a więc nie występuje problem braku miejsca ze względu na zajętość kolejki.

Na podstawie wyznaczonych ograniczeń brzegowych i zachowawczych, można zdefiniować w sposób formalny podzbiór rozwiązań dopuszczalnych Q przestrzeni rozwiązań X, czyli zbiór decyzji:

$$Q = \{\mathbf{x} \in X : x_l \in \langle l_0, l_1 \rangle \wedge x_r \in \langle r_0, r_1 \rangle \wedge g(\mathbf{x}) \leqslant 0\}. \tag{8.8}$$

Zbiór Q może zostać przedstawiony w formie graficznej w układzie współrzędnych kartezjańskich x_l i x_r, co ilustruje rysunek 8.3. W zależności od wartości parametrów: t_c, t_p, c, s i p, kształt oraz zakres zbioru Q będzie różny. Wyodrębnić można trzy podstawowe przypadki, różniące się między sobą wielkością rezerwy (liczby rekordów w kolejce). Ograniczenie zachowawcze $g(\mathbf{x})$ odcina skrajne wektory $\mathbf{x}$, dla których zajętość kolejki i liczba ładowanych rekordów są wzajemnie zbyt duże, by razem współistnieć.



(a) duża rezerwa (b) średnia rezerwa (c) niewielka rezerwa

Rysunek 8.3: Zbiór decyzji Q jako podzbiór przestrzeni rozwiązań X

8.2.2 Funkcje celu

Problem wypełniania stron jest problemem optymalizacji wielokryterialnej, gdyż istnieją dwie przeciwstawne metryki podlegające optymalizacji: jakość usług konsumenta oraz producenta. Funkcja celu w ten sposób zdefiniowanego problemu optymalizacyjnego ma postać funkcji wektorowej $\mathbf{f}(\mathbf{x})$ określonej wzorem:

$$\mathbf{f}(\mathbf{x}) = [f_c(\mathbf{x}), f_p(\mathbf{x})], \tag{8.9}$$

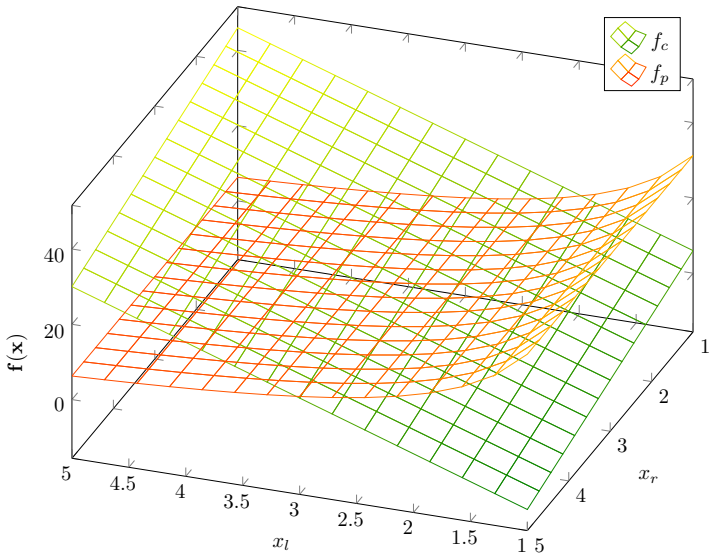
a jej składowe odpowiadają kolejno funkcji celu konsumenta (klienta) oraz funkcji celu producenta (źródła danych).

Pierwszą funkcją celu jest funkcja konsumenta $f_c(\mathbf{x})$. Z punktu widzenia klienta, jako konsumenta rekordów, istotne jest, aby czas oczekiwania na kolejne dane był jak najmniejszy, niezależnie od zmian czasu konsumpcji. Metryka opisująca czas oczekiwania klienta, wyrażona wzorem:

$$f_c(\mathbf{x}) = t_p \cdot x_l - t_c \cdot x_r, \tag{8.10}$$

stanowi różnicę czasu potrzebnego na załadowanie wybranej liczby stron oraz czasu potrzebnego na pobranie pozostałej w kolejce liczby rekordów. W sytuacjach, w których klient nie musi oczekiwać na nowe dane, funkcja $f_c(\mathbf{x})$ przyjmuje ujemne wartości – można je interpretować jako *zapas* czasu klienta.

Druga funkcja celu związana jest ze źródłem danych i określona może zostać jako funkcja producenta $f_p(\mathbf{x})$. Każdy system bazodanowy preferuje mniejszą liczbę transakcji obejmujących większy zakres danych, aniżeli sytuację odwrotną. W przypadku wielu klientów korzystających z jednego repozytorium danych, częstość i rozmiar transakcji wpływa na możliwość lepszego szeregowania wykonywanych operacji. Mniejsza liczba transakcji wiąże się także z ograniczeniem liczby żądań przesyłanych do bazy, co ma znaczenie z punktu widzenia przepustowości zastosowanej infrastruktury sieciowej. Średnia liczba transakcji w jednostce czasu może zostać wyrażona jako odwrotność średniego czasu konsumpcji danych załadowanych w wyniku realizacji transakcji. Czas konsumpcji danych wytworzonych w wyniku



Rysunek 8.4: Kształt funkcji celu: producenta i konsumenta

transakcji obejmującej x_l stron rekordów wynosi: $x_l \cdot p \cdot t_c$. W konsekwencji funkcja celu producenta może zostać opisana wzorem:

$$f_p(\mathbf{x}) = \frac{1}{x_l \cdot p \cdot t_c}. \tag{8.11}$$

Wartości $\mathbf{y} \in Y$ wektorowej funkcji celu $\mathbf{f}(\mathbf{x})$ należą do przestrzeni ocen $Y \subseteq \mathbb{R}^2$. Każdy punkt tej przestrzeni stanowi obraz punktu przestrzeni rozwiązań X. Ze względu na fakt, iż rozwiązania poszukiwane są tylko i wyłącznie w zbiorze decyzji $Q \subseteq X$, odpowiadające im oceny tworzą zbiór celów $A \subseteq Y$, będący podzbiorem przestrzeni ocen Y. W sposób formalny zbiór ten może zostać opisany formułą:

$$A = \{\mathbf{y} \in Y : \mathbf{y} = \mathbf{f}(\mathbf{x}) \wedge \mathbf{x} \in Q\}. \tag{8.12}$$

Kształt obu składowych wektorowej funkcji celu $\mathbf{f}(\mathbf{x})$ zilustrowano na rysunku 8.4. Wartości funkcji celu producenta zostały przeskalowane tysiąckrotnie w celu ich przedstawienia na jednym wykresie razem z wartościami funkcji celu konsumenta. Funkcja konsumenta f_c opada w stronę malejących wartości x_l i rosnących wartości x_r. Z kolei funkcja producenta f_p opada w stronę rosnących wartości x_l i jest przy tym niezależna od wartości x_r. Funkcje te są monotoniczne względem obu swoich argumentów.

W sposób formalny kierunek największego spadku wektorowej funkcji celu $\mathbf{f}(\mathbf{x})$ wyznaczyć można poprzez obliczenie ujemnych gradientów obu jej składowych, czyli funkcji celu konsumenta i funkcji celu producenta. Przed-

stawiają to poniższe równania:

$$\begin{aligned} -\nabla f_c &= -\begin{bmatrix} \frac{\partial f_c}{\partial x_l} \\ \frac{\partial f_c}{\partial x_r} \end{bmatrix} = \begin{bmatrix} -t_p \\ t_c \end{bmatrix}, \\ -\nabla f_p &= -\begin{bmatrix} \frac{\partial f_p}{\partial x_l} \\ \frac{\partial f_p}{\partial x_r} \end{bmatrix} = \begin{bmatrix} \frac{1}{x_l^2 \cdot p \cdot t_c} \\ 0 \end{bmatrix}. \end{aligned} \tag{8.13}$$

8.2.3 Rozwiązania niezdominowane

W przypadku optymalizacji wielokryterialnej, nie zawsze istnieje możliwość znalezienia pojedynczego rozwiązania optymalnego ze względu na wszystkie funkcje celu. W takich przypadkach poszukuje się rozwiązań *niezdominowanych*, dla których polepszenie którejkolwiek z funkcji celu skutkuje pogorszeniem innej. W konsekwencji, nie istnieją rozwiązania od nich jednoznacznie lepsze. Rozwiązania takie nazywane są również rozwiązaniami optymalnymi w sensie Pareto.

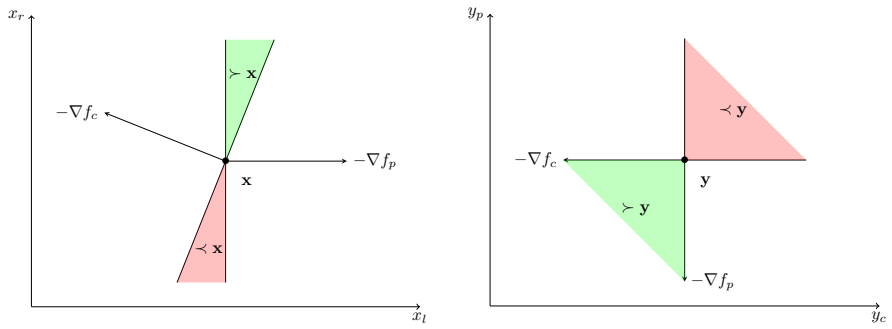
Rozwiązanie $\mathbf{x_1}$ dominuje nad rozwiązaniem $\mathbf{x_2}$ wtedy i tylko wtedy gdy $\mathbf{x_1}$ jest co najmniej tak dobre jak $\mathbf{x_2}$ ze względu na wszystkie kryteria oraz jest ściśle lepsze od $\mathbf{x_2}$ ze względu na przynajmniej jedno kryterium, przy czym *dobroć* ze względu na dane kryterium jest jednoznaczna z minimalizacją powiązanej z nim funkcji celu. Powyższą definicję ilustruje formuła:

$$\mathbf{x_1} \succ \mathbf{x_2} \iff \forall_i\, f_i(\mathbf{x_1}) \leqslant f_i(\mathbf{x_2}) \wedge \exists_j\, f_j(\mathbf{x_1}) < f_j(\mathbf{x_2}). \tag{8.14}$$

Zbiór rozwiązań niezdominowanych złożony jest z tych wszystkich rozwiązań, które nie są zdominowane przez żadne inne rozwiązania. Zbiór ten jest podzbiorem zbioru rozwiązań dopuszczalnych Q, zwanego inaczej zbiorem decyzji. Jego wyznaczenie jest stosunkowo proste w przypadku, gdy wszystkie funkcje celu są monotoniczne ze względu na wszystkie argumenty, co ma miejsce w niniejszych rozważaniach.

W takich sytuacjach, dla każdego rozwiązania $\mathbf{x} \in Q$ możliwe jest zdefiniowanie *stożków* (zwanych stożkami dominacji) wyznaczających obszary, w których leżą: rozwiązania zdominowane przez $\mathbf{x}$, rozwiązania dominujące nad $\mathbf{x}$ oraz rozwiązania indyferentne (niezdominowane, ani nie dominujące) względem $\mathbf{x}$. Stożki te tworzone są przez hiperpłaszczyzny prostopadłe do odwróconych gradientów poszczególnych funkcji celu. Intuicyjną interpretacją powyższego może być stwierdzenie, iż podążając w kierunku prostopadłym do kierunku największego spadku danej funkcji celu (odwróconego gradientu tejże) wartość funkcji celu nie zmienia się. Z drugiej strony podążając w kierunku tworzącym kąt ostry z odwróconym gradientem wartość funkcji celu maleje i analogicznie – podążając w kierunku tworzącym kąt rozwarty z odwróconym gradientem – wartość funkcji celu rośnie.

Podobnie jak w przypadku przestrzeni rozwiązań, wyznaczenie rzeczonych stożków możliwe jest również w przestrzeni ocen. W tym przypadku



(a) przestrzeń rozwiązań (b) przestrzeń ocen

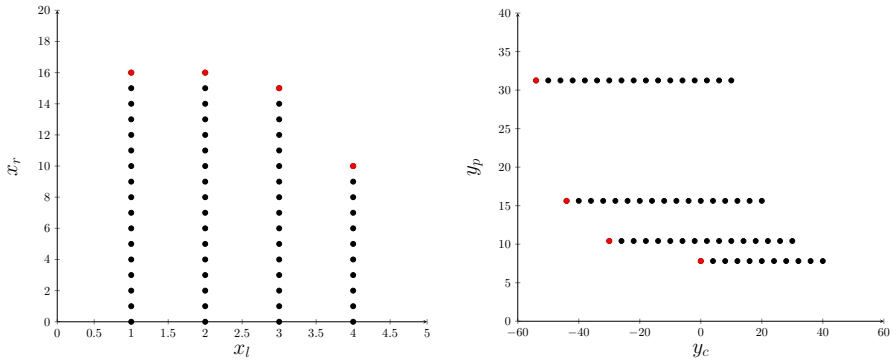
Rysunek 8.5: Stożki dominacji w przestrzeni rozwiązań i ocen

działanie to jest znacznie prostsze, gdyż kierunek spadku każdej funkcji celu jest jasno określony przez osie układu współrzędnych. Na rysunku 8.5 przedstawiono stożki dla przestrzeni rozwiązań (rysunek 8.5a) oraz dla przestrzeni ocen (rysunek 8.5b). Kolorem zielonym i symbolem $\succ$ oznaczono obszar, w którym znajdują się rozwiązania (oceny) dominujące nad bieżącym rozwiązaniem (oceną), a kolorem czerwonym i symbolem $\prec$ obszar zawierający rozwiązania (oceny) zdominowane przez bieżące rozwiązanie (ocenę). Obszary nie zaznaczone żadnym kolorem zawierają rozwiązania (oceny) indyferentne względem bieżącego rozwiązania (oceny).

Ze względu na wspomnianą monotoniczność obu funkcji celu względem obu swoich argumentów, określenie które rozwiązanie należy do poszukiwanego zbioru rozwiązań niezdominowanych polega na „przyłożeniu" odpowiedniego stożka do każdego rozwiązania i wybraniu tych, dla których nie występują żadne rozwiązania dominujące nad rozpatrywanym rozwiązaniem, a więc dla których obszar stożka zaznaczonego na rysunku 8.5a kolorem zielonym byłby pusty. Rysunek 8.6 stanowi ilustrację do powyższych rozważań. Jego lewa część (rysunek 8.6a) przedstawia przykładowy zbiór rozwiązań (należących do zbioru decyzji Q zaznaczonego na rysunku 8.3b). Każde rozwiązanie jest symbolizowane przez kropkę znajdującą się w punkcie o współrzędnych będących liczbami naturalnymi. Kolorem czerwonym oznaczono rozwiązania niezdominowane. Druga część rysunku (rysunek 8.6b) prezentuje analogiczną sytuację występującą w dziedzinie ocen. Kolorem czerwonym zaznaczono oceny rozwiązań niezdominowanych. Tworzą one tak zwany *front Pareto*.

Łatwo jest zauważyć, iż rozwiązania niezdominowane plasują się w górnej i prawej górnej części przestrzeni rozwiązań. Mówiąc ściślej, są to rozwiązania leżące albo na prostej r_1, albo odcięte przez ograniczenie zachowawcze $g(\mathbf{x})$ (równanie 8.7). Graficzną reprezentacją tego ograniczenia jest prosta o równaniu $g(\mathbf{x}) = 0$, z której można otrzymać funkcję:

$$r_g(x_l) = c - x_l \cdot (p - \eta). \tag{8.15}$$



(a) przestrzeń rozwiązań (b) przestrzeń ocen

Rysunek 8.6: Zbiory rozwiązań i ocen niezdominowanych

W sposób formalny, zbiór rozwiązań niezdominowanych (oznaczony jako P) przedstawia formuła:

$$P = \{\mathbf{x} \in Q : x_l \in \langle l_0, l_1 \rangle \cap \mathbb{N} \wedge x_r = \min(r_1, \lfloor r_g(x_l) \rfloor)\}, \tag{8.16}$$

której interpretacja jest następująca: wartość parametru x_l jest dowolna (w zakresie dopuszczalnym), podczas gdy wartość parametru x_r jest mniejszą z dwóch wartości: stałej r_1 oraz wartości funkcji $r_g(x_l)$, zaokrąglonej w dół do najbliższej liczby całkowitej.

8.2.4 Wybór pojedynczego rozwiązania

Z punktu widzenia silnika CUBIT, konieczne jest wyznaczenie pojedynczego rozwiązania w postaci wektora $\mathbf{x} = [x_l, x_r]$. Opisany uprzednio sposób pozwala na wyznaczenie zbioru rozwiązań niezdominowanych, a więc zbioru, spośród którego należy wybrać rzeczone pojedyncze rozwiązanie. Każde z rozwiązań należących do zbioru P jest *dobre*, lecz pod innym względem.

Na potrzeby rozwiązania problemu wypełniania stron wybrano *metodę epsilon-ograniczeń*. W metodzie tej dokonuje się wyboru jednej funkcji celu i poddaje ją optymalizacji, z pozostałych czyniąc dodatkowe ograniczenia. W konsekwencji wybrana funkcja celu reprezentuje główny kierunek optymalizacji, który jest eksploatowany przy zachowaniu minimalnych wymagań ze strony pozostałych funkcji. Rzeczone wymagania oznaczane są zwyczajowo grecką literą ε, skąd opisywana metoda czerpie swoją nazwę. Idea metody epsilon-ograniczeń może zostać wyrażona za pomocą formuły:

$$\mathbf{x}* = \min_{\mathbf{x}} f_1(\mathbf{x}) : \mathbf{x} \in Q \wedge f_2(\mathbf{x}) \leqslant \varepsilon, \tag{8.17}$$

gdzie ε oznacza progową wartość drugiej funkcji celu (ograniczenie), podczas gdy pojedyncze rozwiązanie oznaczono jako $\mathbf{x}*$.

Metoda epsilon-ograniczeń została wybrana ze względu na naturę funkcji celu wykorzystywanych w rozwiązywaniu problemu wypełniania stron. Pierwsza z nich, funkcja celu konsumenta, związana jest pośrednio z czasem oczekiwania (lub zapasem czasu), jakiego doznaje konsument. Osiągnięcie dużych jej wartości wiąże się z niskim tempem dostarczania danych konsumentowi – w pewnych sytuacjach wpływa to wyłącznie na komfort pracy klienta, w innych z kolei może się wiązać z niedotrzymaniem wymaganego reżimu czasowego nałożonego na analizę wykorzystującą dane pobrane z silnika CUBIT. Druga funkcja celu (producenta) związana jest z obciążeniem systemu zarządzania bazą danych. Przekroczenie pewnej wartości granicznej związanej z fizycznymi możliwościami sprzętowymi, poskutkować może niewydolnością całego systemu, a w konsekwencji – dużymi opóźnieniami wszystkich konsumentów. Ponadto, zastosowanie metody epsilon-ograniczeń w naturalny sposób umożliwia osiągnięcie określoności przez algorytm, który ją implementuje, co jest jednym z założeń nowej klasy algorytmów wypełniania stron.

8.3 Warianty adaptacyjnego algorytmu wypełniania stron

Przedstawienie problemu wypełniania stron jako zagadnienia optymalizacji wielokryterialnej oraz propozycja jego rozwiązania z wykorzystaniem metody epsilon-ograniczeń pozwalają na sformułowanie ogólnego zarysu adaptacyjnego algorytmu wypełniania stron. W zależności od uwarunkowań możliwe jest wyróżnienie dwóch strategii: zorientowanej na producenta, w której wprowadza się ograniczenie na funkcję celu producenta oraz zorientowanej na konsumenta, w której ograniczeniu poddawane są wartości funkcji celu konsumenta. Ponadto, wyróżnić można jeszcze strategię hybrydową, powstałą z połączenia obu powyższych.

W celu ułatwienia identyfikacji poszczególnych strategii, zostaną one potraktowane jako samodzielne algorytmy wypełniania stron, o proponowanych nazwach zwyczajowych:

a) algorytm TRAFF (ang. *traffic*) – dla strategii zorientowanej na producenta, ze względu na ograniczenie na maksymalne obciążenie producenta;

b) algorytm LATEN (ang. *latency*) – dla strategii zorientowanej na konsumenta, ze względu na ograniczenie na maksymalne opóźnienie konsumenta;

c) algorytm HYBRI (ang. *hybrid*) – dla strategii hybrydowej, łączącej cechy obu powyższych i nakładającej ograniczenia na obie funkcje celu.

8.3.1 Algorytm TRAFF

Ideą algorytmu TRAFF jest minimalizacja funkcji celu konsumenta (czasu oczekiwania na nowe dane), pod warunkiem nieprzekroczenia maksymalnej wartości funkcji celu producenta (obciążenia źródła danych). Ograniczenie to (oznaczone jako ε_p) jest parametrem systemu i zależy od wielu czynników, w tym również od liczby aktywnie pracujących instancji silnika CUBIT.

Wadą metody epsilon-ograniczeń jest możliwość zastosowania na tyle małej wartości ε_p, dla której nie istnieje żadne rozwiązanie spełniające nierówność $f_p(\mathbf{x}) \leqslant \varepsilon_p$. Przy założeniu, że ograniczenie ε_p jest nieprzekraczalne, konieczne jest wprowadzenie dodatkowych mechanizmów ochronnych, które zapewnią występowanie choć jednego rozwiązania spełniającego powyższą nierówność dla zadanego ε_p.

Wartości funkcji celu producenta $f_p(\mathbf{x})$ są odwrotnie proporcjonalne do trzech wartości: składowej x_l wektora rozwiązania, czasu konsumpcji t_c oraz rozmiaru strony p. W celu zapewnienia istnienia choć jednego rozwiązania, dla którego rzeczona funkcja celu spełni nałożone minimalne ograniczenie ε_p, konieczne jest sztuczne zwiększenie któregoś z trzech wymienionych czynników iloczynu znajdującego się w mianowniku równania 8.11. Pierwsza wielkość nie może przekroczyć wartości l_1 wynikającej z pojemności kolejki, a ostatnia jest zależna wyłącznie od sposobu, w który źródło danych przechowuje dane i nie może być modyfikowana na poziomie innym, niż wewnątrz samego źródła.

Możliwe jest natomiast wprowadzenie kolejnego ograniczenia na minimalny czas konsumpcji pojedynczego rekordu, oprócz już opisanego ograniczenia τ_c (równanie 8.5) wynikającego z warunku płynności układu stronicowania silnika CUBIT. Rzeczone ograniczenie może zostać oznaczone symbolem τ_p (gdyż wynika z uwarunkowań producenta) i wyznaczone zgodnie z formułą:

$$\tau_p = \frac{1}{l_1 \cdot \varepsilon_p \cdot p}, \tag{8.18}$$

w której zastosowano założenie, iż wartość τ_p spełnia równanie $f_p(\mathbf{x}) = \varepsilon_p$, dla wartości składowej x_l wektora rozwiązania równej swojej maksymalnej dopuszczalnej wartości l_1. Współistnienie obu ograniczeń na minimalny czas konsumpcji: τ_c i τ_p prowadzi do konkluzji, iż spełnienie obu z nich jest jednoznaczne ze spełnieniem większego z nich. Ilustruje to równanie:

$$\tau = \max(\tau_c, \tau_p). \tag{8.19}$$

W celu zapewnienia funkcjonowania powyżej zdefiniowanego ograniczenia τ, konieczne jest wymuszenie, aby konsument nie pobierał danych szybciej, niż co τ jednostek czasu. Może to zostać rozwiązane przez wprowadzenie dodatkowego czasu oczekiwania w w procedurze pobierania kolejnych

rekordów. Wartość czasu oczekiwania w może zostać wyznaczona zgodnie z równaniem:

$$w = \begin{cases} 0 & \text{for} \quad t_c \geqslant \tau \\ \tau - t_c & \text{for} \quad t_c < \tau \end{cases}. \tag{8.20}$$

Metoda wyznaczenia pojedynczego rozwiązania przedstawiona w równaniu 8.17 nie oznacza, że konieczne jest liniowe, a więc w czasie $O(n)$, przeglądanie wszystkich rozwiązań ze zbioru decyzji Q, ani nawet ze zbioru rozwiązań niezdominowanych P w celu znalezienia minimum funkcji celu konsumenta dla spełnionego warunku nałożonego na wartości funkcji celu producenta. Istnienie bowiem rozwiązanie analityczne tego problemu, pozwalające na uzyskanie obu składowych wektora rozwiązania $\mathbf{x}$ w czasie stałym $O(1)$. Wykorzystując fakt, iż wartość funkcji celu konsumenta jest odwrotnie proporcjonalna do wartości składowej x_l wektora rozwiązania, a jednocześnie wartość funkcji celu producenta jest do niej wprost proporcjonalna, problem znalezienia minimum funkcji celu konsumenta $f_c(\mathbf{x})$, przy zachowaniu nierówności $f_p(\mathbf{x}) \leqslant \varepsilon_p$ sprowadza się do wyznaczenia takiej wartości l_p, dla której spełnione jest równanie:

$$l_p = \frac{1}{\varepsilon_p \cdot p \cdot t_c}, \tag{8.21}$$

powstałe z podstawienia $x_l = l_p$ do równania funkcji celu producenta (równanie 8.11) dla jej największej dopuszczalnej wartości równej ε_p.

Zdefiniowana powyżej wartość l_p może być interpretowana jako dolne ograniczenie dla wartości składowej x_l wektora rozwiązania, wynikłe z nierówności $f_p(\mathbf{x}) \leqslant \varepsilon_p$. Poszukiwana wartość x_l może zostać obliczona jako zaokrąglenie tejże wartości do najbliższej liczby całkowitej, przy zachowaniu ograniczenia brzegowego $x_l \geqslant l_0$, co ilustruje równanie:

$$x_l = \max(l_0, \lceil l_p \rceil), \tag{8.22}$$

przy czym składowa x_r wektora rozwiązania może zostać wyznaczona zgodnie ze wzorem 8.16.

Podsumowując, rozwiązanie problemu wypełniania stron, zdefiniowanego jako problem optymalizacji wielokryterialnej, po przyjęciu strategii zorientowanej na producenta (algorytm TRAFF) może zostać przedstawione formułą:

$$\begin{gathered} \mathbf{x}* = \min_{\mathbf{x}} f_c(\mathbf{x}) : \mathbf{x} \in Q \wedge f_p(\mathbf{x}) \leqslant \varepsilon_p, \\ \text{przy zał. } t_c \geqslant \tau_p \Rightarrow \exists_{\mathbf{x} \in Q} : f_p(\mathbf{x}) \leqslant \varepsilon_p. \end{gathered} \tag{8.23}$$

Algorytm 8.1 ilustruje działanie strategii TRAFF. W pierwszej kolejności wyznaczane są wartości stałych (mówiąc ściśle: stałych podczas pojedynczego przebiegu algorytmu): η, l_0, r_1. W drugim kroku stosowana jest metoda epsilon-ograniczeń, w wyniku której obliczana jest składowa x_l wektora rozwiązania. Trzecim i ostatnim krokiem jest obliczenie odpowiadającej jej wartości składowej x_r wektora rozwiązania.

Algorytm 8.1 Algorytm TRAFF

Wejście:

s : rozmiar kolejki
t_c : czas konsumpcji agregatu
t_p : czas produkcji strony
ε_p : ograniczenie producenta

Stałe:

c : pojemność kolejki
p : rozmiar strony

1: **function** TRAFF$(s, t_c, t_p, \varepsilon_p)$
2: **let** $\eta = \frac{t_p}{t_c}$
3: **let** $l_0 = 1$
4: **let** $r_1 = s$
5: **let** $l_p = \frac{1}{\varepsilon_p \cdot p \cdot t_c}$
6: **let** $x_l = \max(l_0, \lceil l_p \rceil)$
7: **let** $r_g = c - x_l \cdot (p - \eta)$
8: **let** $x_r = \min(r_1, \lfloor r_g \rfloor)$
9: **return** $[x_l, x_r]$
10: **end function**

8.3.2 Algorytm LATEN

Działanie algorytmu LATEN stanowi w pewnym sensie przeciwieństwo omówionego algorytmu TRAFF. W tym przypadku minimalizacji poddawana jest funkcja celu producenta, przy jednoczesnym ograniczeniu wartości funkcji celu konsumenta do maksymalnej wartości ε_c. Wielkość ta jest zależna od preferencji klienta: wartość zero oznacza przy tym dążenie do braku oczekiwania na nowe rekordy, wartości mniejsze od zera symbolizują *nadmiar* lub *zapas* rekordów w kolejce układu stronicowania, a wartości większe od zera oznaczają dopuszczalne opóźnienie.

Odmiennie niż w przypadku poprzedniego algorytmu, nie jest możliwe zagwarantowanie istnienia zawsze przynajmniej jednego rozwiązania spełniającego ograniczenie $f_c(\mathbf{x}) \leqslant \varepsilon_c$ poprzez wprowadzenie dodatkowych mechanizmów ochronnych, gdyż wymagałoby to zmiany parametrów, na które algorytm nie ma wpływu (jak przykładowo czas produkcji strony). W konsekwencji, zachodzi potrzeba znalezienia rozwiązania *awaryjnego*, które zostanie wybrane w sytuacji, w której nie jest możliwe dotrzymanie ograniczenia na maksymalną wartość funkcji celu konsumenta. Rzeczone rozwiązanie awaryjne to takie, dla którego funkcja celu konsumenta przyjmuje wartości najmniejsze, a więc dla $x_l = l_0$.

Jednocześnie opisany w poprzednim algorytmie minimalny czas konsumpcji pojedynczego rekordu τ przyjmuje postać zgodną z równaniem:

$$\tau_p = 0 \Rightarrow \tau = \tau_c, \tag{8.24}$$

gdyż traci znaczenie ograniczenie: $\tau_p = 0$. Wzór 8.20 pozostaje bez zmian, gdyż konieczne jest zapewnienie warunku płynnej pracy silnika CUBIT.

W przypadku tego algorytmu nie jest możliwe znalezienie pojedynczego rozwiązania w czasie $O(1)$, a więc na drodze obliczeń analitycznych – konieczne jest przeglądnięcie w liniowym czasie zbioru rozwiązań niezdominowanych P i wybraniu tych, które spełniają ograniczenie $f_c(\mathbf{x}) \leqslant \varepsilon_c$. Taki stan rzeczy wynika z faktu, iż funkcja celu konsumenta, w przeciwieństwie do funkcji celu producenta, jest funkcją dwóch zmiennych. Formuła podsumowująca rozwiązanie problemu wypełniania stron za pomocą strategii zorientowanej na konsumenta (algorytm LATEN) przedstawia się następująco:

$$\mathbf{x}* = \begin{cases} \min\limits_{\mathbf{x}} f_p(\mathbf{x}) : \mathbf{x} \in Q \wedge f_c(\mathbf{x}) \leqslant \varepsilon_c, & \text{jeśli} \quad \exists_{\mathbf{x} \in Q} : f_c(\mathbf{x}) \leqslant \varepsilon_c, \\ \min\limits_{\mathbf{x}} f_c(\mathbf{x}) : \mathbf{x} \in Q, & \text{jeśli} \quad \forall_{\mathbf{x} \in Q} : f_c(\mathbf{x}) > \varepsilon_c. \end{cases} \tag{8.25}$$

Algorytm 8.2 przedstawia ideę strategii LATEN. Podobnie jak w przypadku poprzednio opisanej strategii, na początku wyznaczane są wartości stałych. Następnie nadawane są wartości początkowe zmiennym dla rozwiązania awaryjnego (linia 6). Zasadnicza część algorytmu odbywa się w pętli i polega na iteracji po uporządkowany według wartości x_l zbiorze rozwiązań niezdominowanych i poszukiwaniu minimum funkcji celu producenta. W przypadku niespełnienia ograniczenia na maksymalny czas oczekiwania klienta pętla jest przerywana (linia 16), gdyż wartości funkcji celu konsumenta rosną razem z wartościami składowej x_l wektora rozwiązania. W przypadku, gdy nie uda się znaleźć ani jednego rozwiązania w pętli, zwracane jest wyznaczone przy inicjalizacji rozwiązanie awaryjne.

8.3.3 Algorytm HYBRI

Oba zaprezentowane algorytmy wykazują przeciwstawne działanie, optymalizując przypisaną sobie funkcję celu, przy zachowaniu akceptowalnych wartości drugiej funkcji celu. W przypadku strategii zorientowanej na producenta polega to na wybieraniu zawsze najmniejszego możliwego czasu oczekiwania konsumenta, dla którego obciążenie producenta nie przekracza granicznej wartości. Z kolei w przypadku strategii zorientowanej na konsumenta działanie jest odwrotne: wybierane jest najmniejsze możliwe obciążenie producenta, przy którym czas oczekiwania konsumenta nie przekracza dopuszczalnej wartości, o ile takie rozwiązanie istnieje – w przeciwnym wypadku minimalizowany jest czas oczekiwania konsumenta.

Wybór jednej z opisanych strategii oznacza ukierunkowanie się wyłącznie na producenta bądź konsumenta poprzez zastosowanie odpowiedniego ograniczenia. Niemniej istnieją sytuacje, w których istotne są oba ograniczenia na raz, zarówno na maksymalne obciążenie producenta, jak i na maksymalny czas oczekiwania konsumenta. Proponowany algorytm HYBRI wychodzi na

Algorytm 8.2 Algorytm LATEN

Wejście:

s : rozmiar kolejki
t_c : czas konsumpcji agregatu
t_p : czas produkcji strony
ε_c : ograniczenie konsumenta

Stałe:

c : pojemność kolejki
p : rozmiar strony

1: **function** LATEN$(s, t_c, t_p, \varepsilon_c)$
2: **let** $\eta = \frac{t_p}{t_c}$
3: **let** $l_0 = 1$
4: **let** $l_1 = \frac{c}{p}$
5: **let** $r_1 = s$
6: **let** $min = +\infty$
7: **let** $x_l = l_0$
8: **let** $r_g = c - x_l \cdot (p - \eta)$
9: **let** $x_r = \min(r_1, \lfloor r_g \rfloor)$
10: **for** $x'_l \leftarrow l_0 \ldots l_1$ **do**
11: **let** $r'_g = c - x'_l \cdot (p - \eta)$
12: **let** $x'_r = \min(r_1, \lfloor r'_g \rfloor)$
13: **let** $y_p = f_p(x'_l, x'_r)$
14: **let** $y_c = f_c(x'_l, x'_r)$
15: **if** $y_c > \varepsilon_c$ **then**
16: **break**
17: **end if**
18: **if** $y_p < min$ **then**
19: $min \leftarrow y_p$
20: $x_l \leftarrow x'_l$
21: $x_r \leftarrow x'_r$
22: **end if**
23: **end for**
24: **return** $[x_l, x_r]$
25: **end function**

przeciw właśnie takim przypadkom. Łączy on w sobie cechy obu omówionych uprzednio algorytmów wypełniania stron.

Należy mieć na uwadze, iż o ile wymuszenie spełnienia ograniczenia na maksymalne obciążenie producenta jest możliwe poprzez wprowadzenie sztucznego opóźnienia, to ograniczenie maksymalnego czasu oczekiwania konsumenta nie zawsze może być zagwarantowane, podobnie jak ma to miejsce w algorytmu LATEN.

Biorąc pod uwagę oba ograniczenia jednocześnie oraz uwzględniając nieprzekraczalność ograniczenia producenta, można wyróżnić dwa przypadki. W pierwszym z nich istnieje przynajmniej jedno rozwiązanie niezdominowane, które spełnia oba ograniczenia. Wówczas wybór pojedynczego rozwiązania można sprowadzić do minimalizacji dowolnej funkcji celu. Proponuje się jednocześnie, aby była to funkcja celu producenta tak, aby powstrzymać się przed generowaniem niepotrzebnego obciążenia. Drugi z przypadków zachodzi wtedy, gdy spełnienie nieprzekraczalnego ograniczenie producenta jest jednoznaczne z niespełnieniem ograniczenia konsumenta. W takiej sytuacji należy wybrać takie rozwiązanie, które zapewnia najmniejszy czas oczekiwania konsumenta.

W zależności od aktualnego przypadku minimalizowana jest wybrana funkcja celu, przy zachowaniu ograniczenia na dopuszczalne wartości drugiej, przy jednoczesnym wymuszeniu minimalnego czasu konsumpcji pojedynczego rekordu. Ilustruje to formuła:

$$\mathbf{x}* = \begin{cases} \min\limits_{\mathbf{x}} f_p(\mathbf{x}) : \mathbf{x} \in Q \wedge f_c(\mathbf{x}) \leqslant \varepsilon_c, \text{jeśli } \exists_{\mathbf{x}\in Q} : f_p(\mathbf{x}) \leqslant \varepsilon_p \wedge f_c(\mathbf{x}) \leqslant \varepsilon_c, \\ \min\limits_{\mathbf{x}} f_c(\mathbf{x}) : \mathbf{x} \in Q \wedge f_p(\mathbf{x}) \leqslant \varepsilon_p, \text{jeśli } \forall_{\mathbf{x}\in Q} : f_p(\mathbf{x}) \leqslant \varepsilon_p \Rightarrow f_c(\mathbf{x}) > \varepsilon_c, \\ \text{przy zał. } t_c \geqslant \tau_p \Rightarrow \exists_{\mathbf{x}\in Q} : f_p(\mathbf{x}) \leqslant \varepsilon_p. \end{cases} \tag{8.26}$$

Algorytm 8.3 przedstawia ideę strategii HYBRI. W swojej formie jest on zbliżony do algorytmu LATEN, lecz różni się w kilku istotnych szczegółach. Przede wszystkim inna jest postać rozwiązania awaryjnego (linia 6) – jest to rozwiązanie zaczerpnięte z algorytmu TRAFF. Główna pętla algorytmu rozpoczyna iterację nie od najmniejszej możliwej wartości x_l wektora rozwiązania równej l_0, lecz od wartości l_p, wynikłej z zastosowania ograniczenia na maksymalne obciążenie producenta (linia 11).

Algorytm 8.3 Algorytm HYBRI

Wejście:

s : rozmiar kolejki
t_c : czas konsumpcji agregatu
t_p : czas produkcji strony
ε_p : ograniczenie producenta
ε_c : ograniczenie konsumenta

Stałe:

c : pojemność kolejki
p : rozmiar strony

1: **function** HYBRI($s, t_c, t_p, \varepsilon_p, \varepsilon_c$)
2: **let** $\eta = \frac{t_p}{t_c}$
3: **let** $l_0 = 1$
4: **let** $l_1 = \frac{c}{p}$
5: **let** $r_1 = s$
6: **let** $min = +\infty$
7: **let** $l_p = \frac{1}{\varepsilon_p \cdot p \cdot t_c}$
8: **let** $x_l = \max(l_0, \lceil l_p \rceil)$
9: **let** $r_g = c - x_l \cdot (p - \eta)$
10: **let** $x_r = \min(r_1, \lfloor r_g \rfloor)$
11: **for** $x'_l \leftarrow l_p \ldots l_1$ **do**
12: **let** $r'_g = c - x'_l \cdot (p - \eta)$
13: **let** $x'_r = \min(r_1, \lfloor r'_g \rfloor)$
14: **let** $y_p = f_p(x'_l, x'_r)$
15: **let** $y_c = f_c(x'_l, x'_r)$
16: **if** $y_c > \varepsilon_c$ **then**
17: **break for**
18: **end if**
19: **if** $y_p < min$ **then**
20: $min \leftarrow y_p$
21: $x_l \leftarrow x'_l$
22: $x_r \leftarrow x'_r$
23: **end if**
24: **end for**
25: **return** $[x_l, x_r]$
26: **end function**

Rozdział 9

Miary jakości usług w strumieniowych hurtowniach danych kontekstowych

Niniejszy rozdział wprowadza dwie metryki jakości usług: konsumenta oraz producenta, które mogą być stosowane w strumieniowych hurtowniach danych kontekstowych. Obie metryki użyto podczas badań eksperymentalnych: weryfikacyjnych i porównawczych, w ramach których sprawdzono nowe algorytmy wypełniania stron pod kątem spełniania założeń, jak również dokonano porównania z algorytmami poprzedniej generacji. Opis uzyskanych wyników wraz z komentarzem stanowią główną część tego rozdziału. Ponadto, w rozdziale krótko scharakteryzowano środowisko badawcze, za pomocą którego przeprowadzano eksperymenty.

9.1 Metody oceny jakości usług

Pojęcie *jakości usług* (*QoS*, z ang. *Quality of Service*) pojawia się zawsze wtedy, gdy rozpatrywane są metody usprawniające przeprowadzanie różnorakich procesów, uznanych za usługi. Istnieje wówczas konieczność zmierzenia wpływu działania tych metod, czyli analiza zmian w efektywności świadczonych usług. Choć jakość jako taka jest pojęciem abstrakcyjnym i subiektywnym, to miara jakości jest wartością liczbową i może zostać użyta do obiektywnego porównania różnych metod.

W przypadku strumieniowych hurtowni danych kontekstowych, a właściwie silnika CUBIT, stanowiącego istotny komponent kontekstowego serwera CtxOLAP, najważniejszym procesem, który może zostać uznany za usługę, jest proces dostarczania nowych danych, czyli aktualizacji kostki. Proces ten

jest realizowany przez układ stronicowania silnika CUBIT. Ze względu na fakt, iż w procesie dostarczania danych udział biorą dwie strony: producent (źródło danych) oraz konsument (odbiorca danych), zdecydowano się na wprowadzenie dwóch niezależnych miar jakości usług: konsumenta i producenta. Pierwsza z tych miar określa jakość wspomnianego procesu z punktu widzenia odbiorcy i bierze pod uwagę takie czynniki jak opóźnienie w dostarczaniu danych, podczas gdy druga miara skupia się na obciążeniu źródła danych kolejnymi zapytaniami.

9.1.1 Miara jakości usług konsumenta

Za podstawę do obliczenia metryki obrazującej jakość usług konsumenta, wzięto dwa czynniki wpływające na szeroko rozumiany komfort jego pracy: opóźnienie, którego doświadcza konsument w związku z brakiem nowych danych oraz nieciągłość transmisji danych (skokowe zmiany wspomnianego opóźnienia).

Pierwszy czynnik wiąże się bezpośrednio z czasem oczekiwania oraz czasem konsumpcji pojedynczego agregatu przez konsumenta, a precyzyjniej – odwrotnościami tych czasów, reprezentującymi natężenie przepływu danych. W przypadku optymistycznym, konsument nie doświadcza żadnego opóźnienia ze strony układu stronicowania, a więc idealne natężenie przepływu danych wyraża się wzorem:

$$\varphi_0 = \frac{n_c}{\sum\limits_{i=1}^{n_c} t_{c_i}}, \tag{9.1}$$

w którym w mianowniku znajduje się uśredniony czas konsumpcji (t_c), przy założeniu, że zarejestrowano n_c pomiarów na rzecz konsumenta. Rzeczywisty przepływ ma natężenie równe odwrotności średniej sumy czasu konsumpcji (t_c) oraz czasu oczekiwania (t_w), co może zostać zobrazowane równaniem:

$$\varphi = \frac{n_c}{\sum\limits_{i=1}^{n_c} (t_{c_i} + t_{w_i})}. \tag{9.2}$$

Iloraz obu powyższych wartości (wzory 9.1 i 9.2):

$$\Phi = \frac{\varphi}{\varphi_0} \cdot 100\%, \tag{9.3}$$

stanowi wartość informującą, jaką część idealnego natężenia przepływu danych uzyskano w danej sytuacji.

Drugi z czynników wykorzystywanych do obliczenia metryki jakości usług konsumenta związany jest odstającymi pomiarami czasu oczekiwania na nowe dane, które rejestrowane są w przypadku przedłużających się braków danych w kolejce układu stronicowania. Wartości odstające mogą być znalezione na wiele sposobów, niemniej w trakcie badań wstępnych ustalono,

że najlepszą metodą ich wyłonienia jest zastosowanie metody *trzech sigm*, a więc uznania za pomiary odstających tych, które są oddalone od średniej o trzy odchylenia standardowe. Należy tutaj zwrócić uwagę na fakt, iż wspomniane odstawanie rozpatrywane jest jedynie w stronę wartości większych od średniej.

Przeciwieństwem liczby pomiarów odstających jest liczba pomiarów normalnych (nazywanych dalej ciągłymi), która stała się podstawą do wyznaczenia rzeczonego drugiego czynnika. Przyjmując, że zbiór pomiarów to M, idealna liczba pomiarów ciągłych jest równa:

$$\omega_0 = |M| = n_c, \tag{9.4}$$

podczas gdy rzeczywista liczba pomiarów ciągłych wynosi:

$$\omega = |\{t_{w_i} \in M : t_{w_i} \leqslant (\mu_w + 3\sigma_w)\}|, \tag{9.5}$$

gdzie μ_w oraz σ_w to odpowiednio średnia i odchylenie standardowe czasu oczekiwania t_w. Iloraz obu powyższych wartości (wzory 9.4 i 9.5):

$$\Omega = \frac{\omega}{\omega_0} \cdot 100\%, \tag{9.6}$$

informuje o odsetku ciągłych pomiarów i stanowi zarazem miarę stabilności algorytmu.

Na podstawie równań 9.3 oraz 9.6 można wyznaczyć metrykę opisującą jakość usług konsumenta, jako iloczyn obu opisanych przez nie czynników:

$$QoS_c = \Phi \cdot \Omega. \tag{9.7}$$

Zastosowanie obu czynników powoduje, że zarówno przebieg stabilny: cechujący się stałym, lecz niezerowym opóźnieniem, jak i przebieg niestabilny: w większości zerowy, ale z licznymi pikami, spowodują widoczne obniżenie jakości usług konsumenta.

9.1.2 Miara jakości usług producenta

Metryka obrazująca jakość usług producenta związana jest z obciążeniem bazy danych lub innego ich źródła (strumień, plik, pamięć podręczna, operator agregacji, baza zmaterializowanych agregatów). Została ona wyrażona jako kombinacja dwóch czynników, z których pierwszy określa stopień granulacji zapytań, a drugi czas bezczynności pomiędzy zapytaniami.

Granulacja zapytania jest miarą niosącą informację o zakresie danych objętych zapytaniem: zapytania drobnoziarniste są ukierunkowane na niewielkie porcje danych, podczas gdy zapytania gruboziarniste odwrotnie. Ze stopniem granulacji zapytań wiążę się również ich liczba potrzebna do odczytania zadanego wolumenu danych – im grubsza jest granulacja, tym mniej

zapytań jest potrzebnych. Zapytania gruboziarniste są oprócz tego potencjalnie łatwiejsze w optymalizacji i równoległej realizacji po stronie silnika zarządzania bazą danych. W konsekwencji dążenie do zwiększenia granulacji zapytań jest jedną z technik polepszenia jakości usług producenta.

Maksymalny możliwy rozmiar pojedynczego zapytania kierowanego przez algorytm wypełniania stron jest równy:

$$\gamma_0 = \frac{c}{p}, \tag{9.8}$$

gdyż wynika on z rozmiaru kolejki c układu stronicowania silnika CUBIT oraz rozmiaru pojedynczej strony p – rozmiar zapytania jest zdefiniowany liczbą stron. Rzeczywisty rozmiar zapytania może być obliczony jako średnia:

$$\gamma = \frac{1}{n_p} \sum_{i=1}^{n_p} q_{s_i}, \tag{9.9}$$

gdzie n_p to liczba pomiarów zarejestrowanych na rzecz producenta, a q_s oznacza rozmiar zapytania wyrażony w stronach. Iloraz wartości z równań 9.8 oraz 9.9:

$$\Gamma = \frac{\gamma}{\gamma_0} \cdot 100\%, \tag{9.10}$$

informuje o względnej granulacji zapytań, która to wielkość wyrażona jest procentowo.

Drugi czynnik wchodzący w skład metryki opisującej jakość usług producenta związany jest ze średnim interwałem czasu pomiędzy poszczególnymi zapytaniami, a precyzyjniej – jest to różnica czasów pomiędzy każdą parą żądań kierowanych przez algorytm wypełniania stron do bazy danych. Maksymalna wartość tej miary jest trudna do oszacowania, gdyż może ona przyjmować niemalże dowolne wartości. W konsekwencji zdecydowano się na oszacowanie minimalnej możliwej wartości rzeczonego interwału, równego czasowi produkcji pojedynczej strony. Powyższy przypadek jest jednoznaczny z rozpoczęciem wykonywania kolejnego zapytania bezpośrednio zaraz po zakończeniu realizacji poprzedniego. Wyraża się to wzorem:

$$\delta_0 = \frac{1}{n_p} \sum_{i=1}^{n_p} t_{p_i}, \tag{9.11}$$

w którym n_p to liczba pomiarów dokonanych na rzecz producenta, a t_p oznacza czas produkcji pojedynczej strony. Rzeczywisty czas pomiędzy kolejnymi zapytaniami jest średnią z zarejestrowanych pomiarów t_q:

$$\delta = \frac{1}{n_p} \sum_{i=1}^{n_p} t_{q_i}, \tag{9.12}$$

gdzie t_q oznacza interwał czasu pomiędzy kolejnymi zapytaniami.

Iloraz wartości opisanych równaniami 9.11 i 9.12:

$$\Delta = \max\left[0, \left(1 - \frac{\delta_0}{\delta}\right)\right] \cdot 100\%, \tag{9.13}$$

informuje o stopniu podobieństwa do pesymistycznego przypadku największego obciążenia bazy danych. Ze względu na wykorzystanie, odmiennie niż przy pozostałych metrykach, przypadku pesymistycznego jako referencyjnego (δ_0), zastosowana została odwrócona kolejność dzielenia oraz odjęcie wynikowego ilorazu od jedności przy obliczaniu zbiorczej metryki Δ. Ponadto, ze względu na możliwość otrzymania ujemnych wartości w określonych okolicznościach (wielowątkowe wypełnianie stron przez algorytmy stałe), wartość Δ została ograniczona do zera z lewej strony.

Wykorzystując obliczone w formułach 9.10 oraz 9.13 składniki możliwe jest podanie metryki opisującej jakość usług producenta:

$$QoS_p = \Gamma \cdot \Delta, \tag{9.14}$$

która to wielkość jest iloczynem opisanych powyżej czynników.

9.1.3 Pozostałe metryki porównawcze

Oprócz opisanych dwóch miar jakości usług, wyróżnić można pewne pomocnicze wartości zagregowane, charakteryzujące pracę adaptacyjnych algorytmów wypełniania stron. Wśród nich najważniejsze to: średnia względna wartość parametru x_l:

$$\tilde{x}_l = \frac{1}{n_a} \sum_{i=1}^{n_a} \frac{x_{l_i} \cdot p}{c}, \tag{9.15}$$

średnia względna wartość parametru x_r:

$$\tilde{x}_r = \frac{1}{n_a} \sum_{i=1}^{n_a} \frac{x_{r_i}}{c}, \tag{9.16}$$

a także średnia względna długość kolejki:

$$\tilde{s} = \frac{1}{n_c} \sum_{i=1}^{n_c} \frac{s_i}{c}. \tag{9.17}$$

Opisane wyżej trzy wartości zostały wyrażone jako wartość z przedziału $\langle 0, 1\rangle$, w celu ich uniezależnienia od aktualnie przyjętego rozmiaru kolejki. W równaniach 9.15 oraz 9.16 wartość n_a oznacza liczbę zarejestrowanych pomiarów na rzecz algorytmu wypełniania stron, podczas gdy wartość n_c występująca w równaniu 9.17 oznacza analogiczną liczbę pomiarów dla konsumenta.

9.2 Środowisko testowe

Adaptacyjne algorytmy wypełniania stron poddano badaniom eksperymentalnym, które można podzielić na dwie zasadnicze grupy: weryfikacyjne i porównawcze. Celem pierwszej grupy było potwierdzenie zgodności przedstawicieli nowej klasy algorytmów z założeniami, powstałymi na bazie wniosków z krytycznej analizy dotychczasowych algorytmów. W szczególności weryfikacji poddane zostały: adaptacyjność, elastyczność oraz określoność nowych algorytmów. Istotą drugiej grupy badań było porównanie algorytmów adaptacyjnych pod względem wydajnościowym ze swoimi poprzednikami, algorytmami stałymi.

W celu przeprowadzenia założonych eksperymentów, stworzono specjalne środowisko badawcze, którego głównym komponentem był symulator zdarzeń dyskretnych. W tym symulatorze zaimplementowano wszystkie algorytmy wypełniania stron (stałe oraz adaptacyjne) oraz pozostałe, niezbędne do przeprowadzenia testów, komponenty silnika CUBIT. Wybór takiej formy symulacji został podyktowany wnioskami sporządzonymi na podstawie przeprowadzonego rozpoznania wstępnego, w ramach którego zaimplementowano również symulator pracujący w czasie rzeczywistym.

Wyniki porównania obu podejść jednoznacznie wskazały na symulację zdarzeń dyskretnych, która odbywała się, w odrębnym od rzeczywistego, czasie symulacji. Głównym powodem takiego stanu rzeczy okazała się niemożność polegania na wprowadzonych opóźnieniach wątków symulujących rzeczywiste procesy obliczeniowe, które cechowały się wysoką niestabilnością i zależnością od aktualnego obciążenia systemu operacyjnego. Natomiast zastosowanie koncepcji symulacji zdarzeń dyskretnych poskutkowało powtarzalnością eksperymentów oraz możliwością kontroli losowości zastosowanych czasów.

Funkcjonowanie symulatora zdarzeń dyskretnych (algorytm 9.1) opiera się na istnieniu kolejki priorytetowej reprezentującej oś czasu (linia 2), do której dodawane są zdarzenia, przy czym każde zdarzenie charakteryzowane jest przez znacznik czasowy oraz akcję. Podczas symulacji z kolejki pobierane są sukcesywnie kolejne zdarzenia w kolejności ich występowania w czasie (linia 7). Po pobraniu danego zdarzenia następuje aktualizacja czasu symulacji oraz aktywacja powiązanej z nim akcji. W ramach wykonywania akcji aktualizowany jest stan zmiennych oraz planowane są kolejne zdarzenia. Symulacja kończy się w momencie, w którym na osi czasu nie zaplanowano żadnego następnego zdarzenia.

Wszystkie przeprowadzone eksperymenty odbywały się na zasadzie uruchamiania z zadanymi parametrami pojedynczej instancji silnika CUBIT dla ustalonej liczby agregatów pobieranych przez klienta. Podczas pracy symulatora specjalny moduł profilujący mierzył określone wartości w różnych modułach silnika CUBIT i zapisywał je do plików w formie ciągu pomiarów w funkcji czasu symulacji. Po zakończeniu programu, obliczane były zagre-

Algorytm 9.1 Algorytm symulacji zdarzeń dyskretnych

```
1: procedure SIMULATION()
2:     let timeline = ∅
3:     let time = 0
4:     let initial = EVENT(0, init)
5:     timeline.put(initial)
6:     while timeline.size > 0 do
7:         let event = timeline.get()
8:         time ← event.time
9:         event.action()
10:    end while
11: end procedure
```

gowane wielkości zbiorcze, opisujące całość symulacji. Na podstawie tychże wyznaczane były miary jakości usług, stanowiące podstawowy mechanizm weryfikacji i porównania algorytmów wypełniania stron.

Głównym parametrem każdej symulacji było obciążenie systemu – wartość zbiorcza, obejmująca w sobie zarówno czas produkcji i konsumpcji oraz rozmiar strony. Oznaczone jako ϑ, obciążenie systemu stanowiło stosunek współczynnika niezrównoważenia η do rozmiaru strony p, będącego zarazem maksymalną dopuszczalną wartością η, powyżej której nie jest spełniony warunek płynnej pracy (opisany w równaniu 8.4):

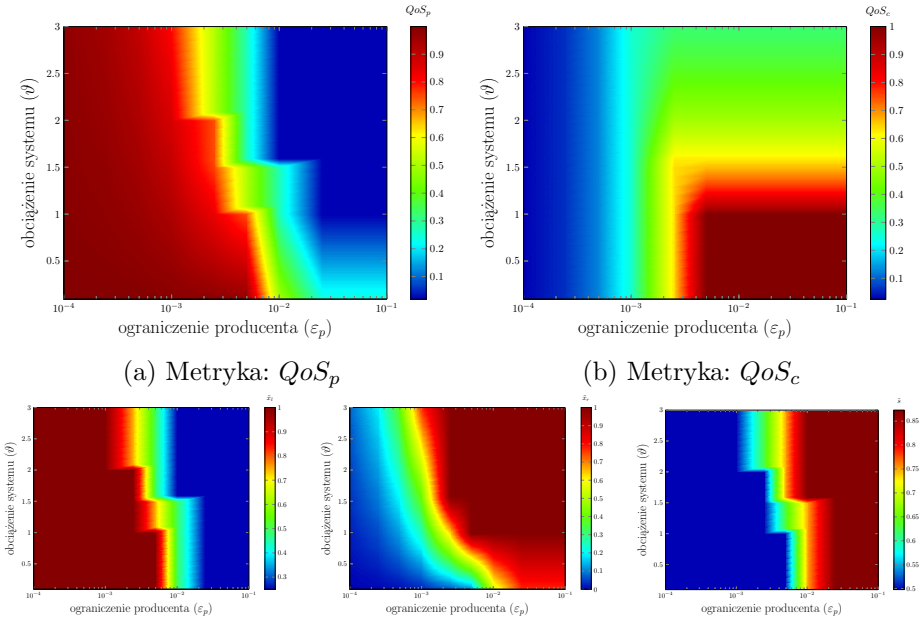
$$\vartheta = \frac{\eta}{p}. \tag{9.18}$$

Przyjmując wartości mniejsze od jedności, parametr ten oznacza, że układ stronicowania pracuje płynnie. Wartość większe wiążą się z niewydolnością systemu producent-konsument spowodowaną zbyt dużym tempem konsumpcji bądź zbyt wolną produkcją nowych danych.

9.3 Weryfikacja w stałym środowisku

Pierwszy rodzaj eksperymentów polegał na przetestowaniu adaptacyjnych algorytmów wypełniania stron w stałych środowisku, czyli przy niezmiennym czasie produkcji oraz konsumpcji. Taki wyidealizowany scenariusz nie występuje co prawda w rzeczywistości, ale dzięki niemu możliwe jest ocenienie algorytmów pod kątem dokonywanych przez nich wyborów oraz ich konsekwencji dla jakości usług producenta i konsumenta.

Badaniom poddano pięć aspektów trzech nowych algorytmów: jakość usług producenta QoS_p, jakość usług konsumenta QoS_c, względny średni rozmiar zapytania $\tilde{x}_l$, względną średnią liczbę rekordów, przy której rozpoczyna się kolejne zapytanie $\tilde{x}_r$ oraz względny średni rozmiar kolejki $\tilde{s}$.



(a) Metryka: QoS_p (b) Metryka: QoS_c

(c) Metryka: $\tilde{x}_l$ (d) Metryka: $\tilde{x}_r$ (e) Metryka: $\tilde{s}$

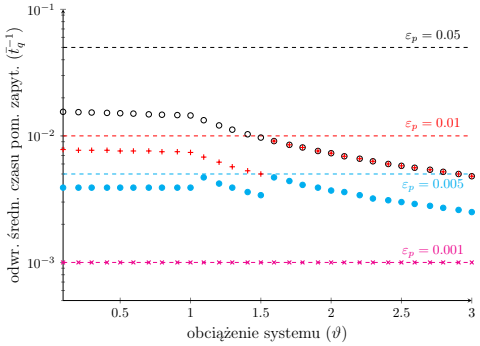
Rysunek 9.1: Charakterystyka pracy algorytmu TRAFF

9.3.1 Algorytm TRAFF

Rysunek 9.1 przedstawia wizualizację badania algorytmu TRAFF. Każdy z wykresów stanowi *mapę ciepła*, na której za pomocą skali kolorów oznaczono wartości danej metryki w zależności od wartości dwóch parametrów: obciążenia systemu oraz wybranego ograniczenia algorytmu.

W przypadku metryki opisującej jakość usług producenta (rysunek 9.1a) widoczna jest wyraźna jej zależność od ograniczenia algorytmu ε_p. Wynika to z faktu, iż im mniejsza jest wartość tego ograniczenie, tym rzadsze staje się odpytywanie producenta, a tym samym poprawia się miara jego jakości usług. Inaczej sytuacja przedstawia się w przypadku drugiej metryki jakości (rysunek 9.1b). Wysokie jej wartości plasują się dla niewielkich wartości ograniczenia algorytmu oraz dla niewielkich (poniżej 1) wartości obciążenia systemu. Dodatkowo wyraźna jest graniczna wartość ograniczenia algorytmu, poniżej której obciążenie systemu przestaje mieć wpływ na metrykę QoS_c, gdyż zaczyna przeważać mechanizm spowalniania konsumenta, w celu spełniania ograniczenia ε_p.

Wartości pozostałych trzech metryk pomocniczych informują o szczegółach funkcjonowania algorytmu TRAFF. Pierwsza z nich, względna średnia wartość parametru x_l (rysunek 9.1c) jest bardzo zbliżona do omówionej już metryki QoS_p, gdyż wraz ze zmniejszaniem się dopuszczalnego natężenia zapytań (parametr ε_p) zwiększa się rozmiar zadawanych zapytań.



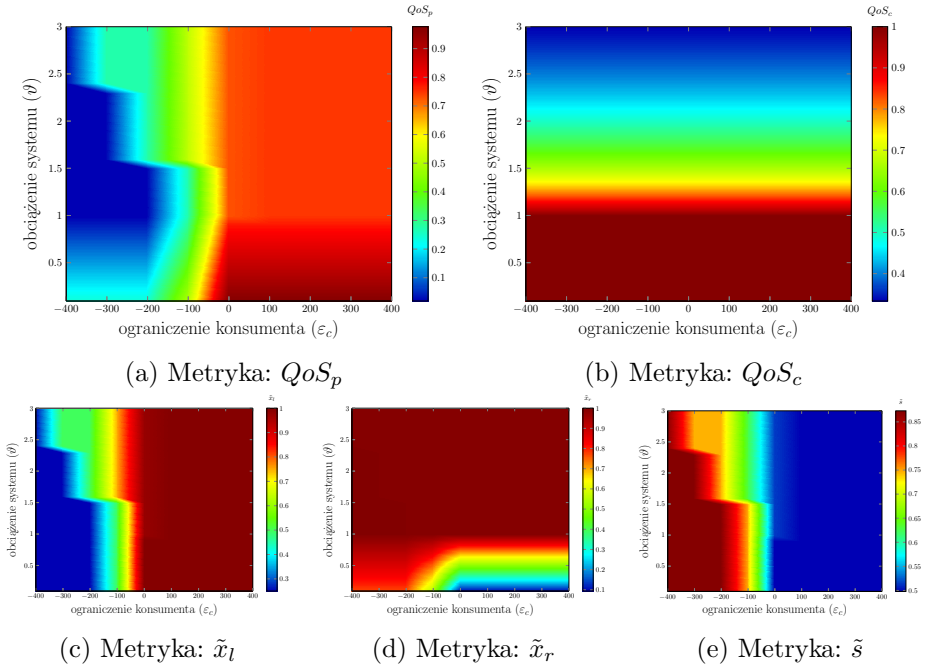
Rysunek 9.2: Spełnianie ograniczeń przez algorytm TRAFF

Druga metryka, względna średnia wartość parametru x_r (rysunek 9.1c), ukazuje ponownie graniczną wartość ograniczenia ε_p, przy której pojawia się realny wpływ tegoż na moment rozpoczęcia każdego kolejnego zapytania. Poniżej tej granicy zapytania zadawane są praktycznie przy pełnym opróżnieniu kolejki, podczas gdy powyżej tej granicy – w środkowym lub górnym jej zakresie, w zależności od aktualnego obciążenia systemu ϑ.

Wartości ostatniej metryki pomocniczej $\tilde{s}$ (rysunek 9.1e) korespondują z metryką $\tilde{x}_l$ oraz QoS_p. Poniżej granicznej wielkości ograniczenia ε_p kolejka pracuje w pełnym swoim zakresie (średni jej rozmiar wynosi około połowę jej pojemności), podczas gdy powyżej tej wartości granicznej kolejka jest utrzymywana w stanie prawie napełnionym.

W celu weryfikacji algorytmu TRAFF pod kątem spełniania przezeń nałożonego ograniczenia ε_p na maksymalne natężenie zapytań kierowanych do producenta, dokonano analizy stanowiącej pewnego rodzaju *przekrój* przez wyniki badania ukazanego na rysunku 9.1. W tym celu wybrano cztery reprezentatywne wartości ograniczenia ε_p: 0.05, 0.01, 0.005 oraz 0.001, a następnie sporządzono dwuwymiarowy wykres (przedstawiony na rysunku 9.2) ukazujący zależność odwrotności średniego czasu pomiędzy zapytaniami (wartość ta jest kluczowa dla działania algorytmu TRAFF, gdyż z nią porównywana jest wartość funkcji celu producenta) od obciążenia systemu.

Dodatkowo, na rysunku 9.2 przedstawiono liniami przerywanymi wybrane cztery wartości ograniczenia ε_p. Dla dużej jego wartości (w tym przypadku 0.05), mierzona wartość znajduje się głęboko poniżej ograniczenia, podczas gdy dla małej – mierzone wartości układają się dokładnie na linii ograniczenia. Dla wartości środkowych ograniczenia (0.01 oraz 0.005) widoczne są skoki, spowodowane próbą minimalizacji drugiej funkcji celu (konsumenta) przez algorytm TRAFF. Jeśli tylko możliwe jest polepszenie tejże, odbywa się to kosztem funkcji celu producenta – objawia się to skokowym, lecz nieprzekraczającym ε_p, wzrostem natężenia zapytań.



(a) Metryka: QoS_p (b) Metryka: QoS_c

(c) Metryka: $\tilde{x}_l$ (d) Metryka: $\tilde{x}_r$ (e) Metryka: $\tilde{s}$

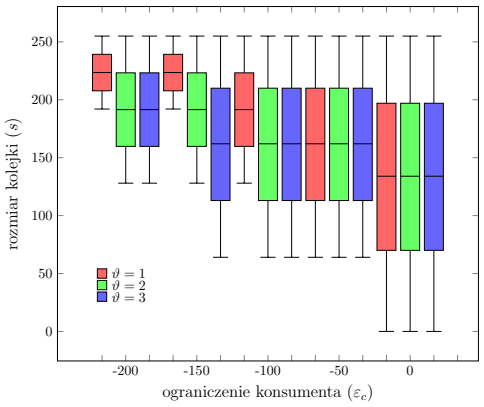
Rysunek 9.3: Charakterystyka pracy algorytmu LATEN

9.3.2 Algorytm LATEN

Równorzędne badanie przeprowadzono dla algorytmu LATEN – jego wyniki zostały przedstawione na rysunku 9.3. Pod pewnymi aspektami zauważalna jest symetryczna zależność względem poprzedniego algorytmu, co niejako wynika z samych założeń obu algorytmów.

Pierwsza metryka jakościowa (QoS_p) w przypadku algorytmu LATEN wykazuje zależność od wartości ograniczenia ε_c (rysunek 9.3a) powiązanego z czasem oczekiwania konsumenta bądź, w przypadku ujemnych wartości tego ograniczenia, zapasem czasu. Im większa jest tolerancja na możliwe opóźnienie, tym większa jest wartość metryki jakości usług producenta. Wyraźna granica zarysowuje się w okolicy zera. Wpływ drugiego parametru, czyli obciążenia systemu, jest również zauważalny.

Druga metryka jakościowa (QoS_c) jest kształtowana wartością wyłącznie jednego parametru: obciążenia systemu ϑ (rysunek 9.3b). Największe wartości osiąga ona dla niewielkich wartości (poniżej 1) obciążenia systemu. Sytuacja taka jest spowodowana dwoma czynnikami: pełnym załadowaniem kolejki na początku (w fazie inicjalizacji) oraz istnieniem ograniczenia τ_c na minimalny czas konsumpcji. Oba te fakty w połączeniu z założeniem stałych warunków pracy podczas symulacji sprawiają, iż nie występuje sytuacja, w której brakuje rekordów w kolejce. Pogarszanie się metryki jakości usług QoS_c wraz ze wzrostem obciążenia systemu następuje liniowo, na skutek



Rysunek 9.4: Wykorzystanie kolejki układu stronicowania przez algorytm LATEN

coraz większego ograniczenia τ_c.

Pierwsza z trzech metryk pomocniczych $\tilde{x}_l$ (rysunek 9.3c) wykazuje odwrotną zależność, w stosunku do odpowiadającej metryki dla algorytmu TRAFF. Jest tak, gdyż wraz ze wzrostem tolerancji na dopuszczalne opóźnienie, zwiększa się rozmiar zapytań kierowanych do producenta.

Wartości metryki $\tilde{x}_r$ (rysunek 9.3d) ukazują wyraźną granicę pomiędzy ujemnymi i dodatnimi wartościami ograniczenia algorytmu ε_c. W przypadku, gdy jego wartości są większe lub równe zero, pojawia się wpływ obciążenia systemu – gdy nie przekracza ono wartości 1, to zapytania zgłaszane są w proporcjonalnym do aktualnego obciążenia zakresie kolejki, podczas gdy po przekroczeniu przez ograniczenie jedności, zapytania zgłaszane są przy prawie pełnej kolejce. Przy ujemnych wartościach ograniczenia ε_c zapytania zgłaszane są w górnej części zakresu rozmiaru kolejki, przy czym im mniejsza jest wartość ograniczenia algorytmu, tym mniejszy jest wpływ obciążenia systemu na wartość parametru $\tilde{x}_r$.

Ostatnia metryka pomocnicza $\tilde{s}$ (rysunek 9.3e) reprezentująca względną średnią zajętość kolejki również wykazuje różną charakterystykę w zależności od znaku ograniczenia algorytmu. Dla nieujemnych wartości tego ograniczenia kolejka operuje w całym zakresie (średnia długość równa połowie pojemności), podczas gdy dla ujemnych wartości ε_c zauważalny jest wpływ obciążenia systemu ϑ oraz wzrost wartości parametru $\tilde{s}$.

Algorytm LATEN nie jest w stanie zawsze zagwarantować spełnienia ograniczenia ε_c, gdyż, jak to zostało wspomniane przy jego opisie, nie jest możliwe przyspieszenie operacji produkcji. Niemniej jednak rzeczone ograniczenie stanowi pewnego rodzaju wytyczną dla zachowania algorytmu, a konkretniej jego zapobiegawczości. Cecha ta wyraża się przez sposób korzystania z kolejki układu stronicowania. Zbiorcze zestawienie tej cechy zostało przedstawione na rysunku 9.4 w formie wykresu pudełkowego.

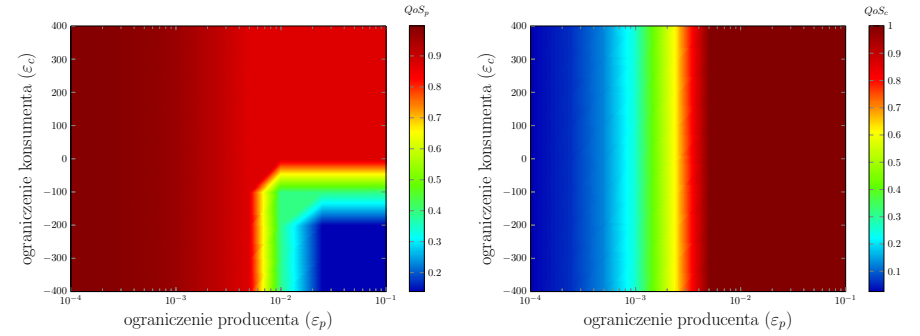
Na rzeczonym wykresie, analizie poddano zakres ograniczenia ε_c od -200 do 0 (wyjątkowo z krokiem równym 50), a badanie przeprowadzono dla trzech różnych wartości obciążenia systemu: 1, 2 oraz 3. Każde z *pudełek* przedstawia takie parametry rozmiaru kolejki, jak: mediana, pierwszy i trzeci kwartyl oraz minimum i maksimum (bez wartości odstających, które w tym przypadku nie występują). Wraz ze zmniejszaniem się ograniczenia algorytmu (poniżej 0), zmienia się sposób użytkowania kolejki: jej zakres roboczy staje się krótszy i przesuwa w stronę górnych wartości. Jest to jednoznaczne ze zwiększeniem poziomu zapobiegawczości przez algorytm, gdyż konsument dysponuje większym zapasem rekordów w kolejce. Zmiana ta jest zależna od obciążenia systemu: im jest ono mniejsze, tym zachodzi szybciej – algorytm jest wówczas bardziej elastyczny. Na wykresie nie przedstawiono wyników dla dodatnich wartości ograniczenia, gdyż, na podstawie wykresów z rysunków 9.3e oraz 9.3c można stwierdzić, iż nie występuje znacząca zmiana parametrów pracy kolejki w stosunku do sytuacji dla $\varepsilon_c = 0$.

9.3.3 Algorytm HYBRI

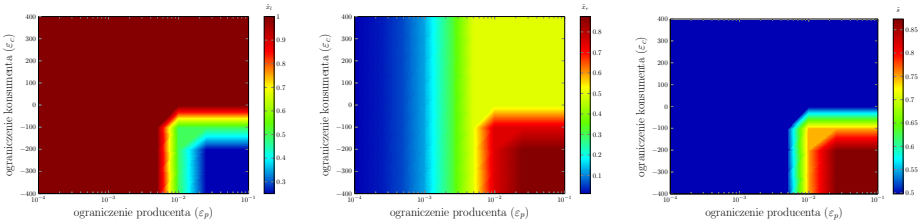
W przypadku ostatniego algorytmu (HYBRI), konieczne było wprowadzenie pewnej zmiany do sposobu przeprowadzania eksperymentów. Algorytm ten bierze pod uwagę oba ograniczenia (ε_p oraz ε_c), a więc zdecydowano się na ustawienie obciążenia systemu (parametr ϑ) na stałą wartość, arbitralnie przyjętą na 0.5. Wybór taki został podyktowany chęcią pozostawienia algorytmowi pewnej swobody działania – celem niniejszego badania była przede wszystkim prezentacja działania algorytmu oraz jego weryfikacja pod kątem zgodności z założeniami. Rysunek 9.5 przedstawia podsumowanie rzeczonego badania.

Charakterystyka algorytmu HYBRI jest pod pewnymi względami połączeniem charakterystyk algorytmów TRAFF i LATEN. Dla niewielkich wartości ograniczenia producenta ε_p oraz ujemnych wartości ograniczenia konsumenta ε_c (prawa dolna ćwiartka wykresów) algorytm charakteryzuje się wysokim obciążeniem producenta (niskie wartości metryki QoS_p – rysunek 9.5a). Z kolej dla dodatnich wartości parametru ε_c następuje sytuacja odwrotna i jakość usług producenta jest wyraźnie wyższa. Sytuacja ta jest analogiczna dla zachowania algorytmu LATEN. Odmiennie sytuacja kształtuje się w momencie, w którym na znaczeniu zyskuje wartość ograniczenia producenta. Dla jego niskich wartości jakość usług producenta jest zawsze wysoka – algorytm stara się wówczas sprostać nałożonemu ograniczeniu poprzez spowolnienie pracy konsumenta, co dodatkowo jest widoczne na rysunku 9.5b. W tym zakresie algorytm HYBRI jest zbliżony do zachowania algorytmu TRAFF.

Opisane zjawisko jest wyraźnie ukazane również na pozostałych wykresach. Przeciętna wielkość zapytania (parametr $\tilde{x}_l$, rysunek 9.5c) jest niewielka jedynie w prawym dolnym obszarze wykresu, a koresponduje z tym



(a) Metryka: QoS_p (b) Metryka: QoS_c



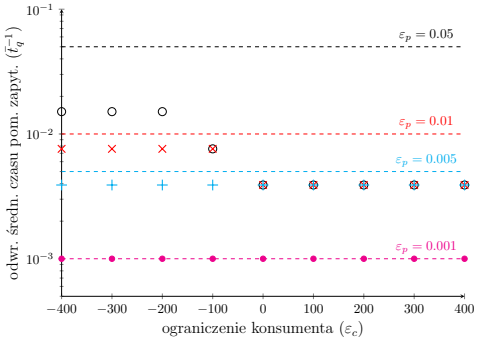
(c) Metryka: $\tilde{x}_l$ (d) Metryka: $\tilde{x}_r$ (e) Metryka: $\tilde{s}$

Rysunek 9.5: Charakterystyka pracy algorytmu HYBRI dla $\vartheta = 0.5$

średnia wielkość kolejki (parametr $\tilde{s}$, rysunek 9.5e), która jest wysoka w tym obszarze. Oznacza to, że dla małych wartości obu ograniczeń algorytm nakazuje producentowi jak najszybciej ładować małe porcje danych tak, aby kolejka była utrzymywana na wysokim poziomie. Również rysunek 9.5d potwierdza ten fakt. Dodatkowo można na nim zaobserwować, że przy niewielkim ograniczeniu producenta oraz dla dodatnich wartości parametru ε_c zapytania są zgłaszane mniej więcej w połowie wielkości kolejki, co wynika bezpośrednio z przyjętego obciążenia systemu równego 0.5 – czas produkcji pojedynczej strony jest równy połowie czasu jej konsumpcji.

Algorytm HYBRI, podobnie jak TRAFF, stara się zawsze spełnić ograniczenie producenta, traktując przy tym ograniczenie konsumenta jako wytyczną do działania, podobnie jak LATEN. Na rysunku 9.6 przedstawiono zależność odwrotności średniego czasu pomiędzy zapytaniami (oznaczenie $\bar{t}_q^{-1}$) od wartości ograniczenia konsumenta ε_c dla czterech wybranych wartości ograniczenia producenta ε_p: 0.05, 0.01, 0.005 oraz 0.001.

Na wykresie zauważalny jest wpływ ograniczenia konsumenta na odwrotność średniego czasu pomiędzy zapytaniami (obciążenie producenta), która rośnie wraz z maleniem tego ograniczenia, jednak nie przekracza założonego ograniczenia producenta. Im to drugie ograniczenie jest bardziej restrykcyjne (osiąga mniejsze wartości), tym algorytm ma mniejszą swobodę w doborze obciążenia producenta.



Rysunek 9.6: Spełnianie ograniczeń przez algorytm HBYRI

9.4 Weryfikacja w zmiennym środowisku

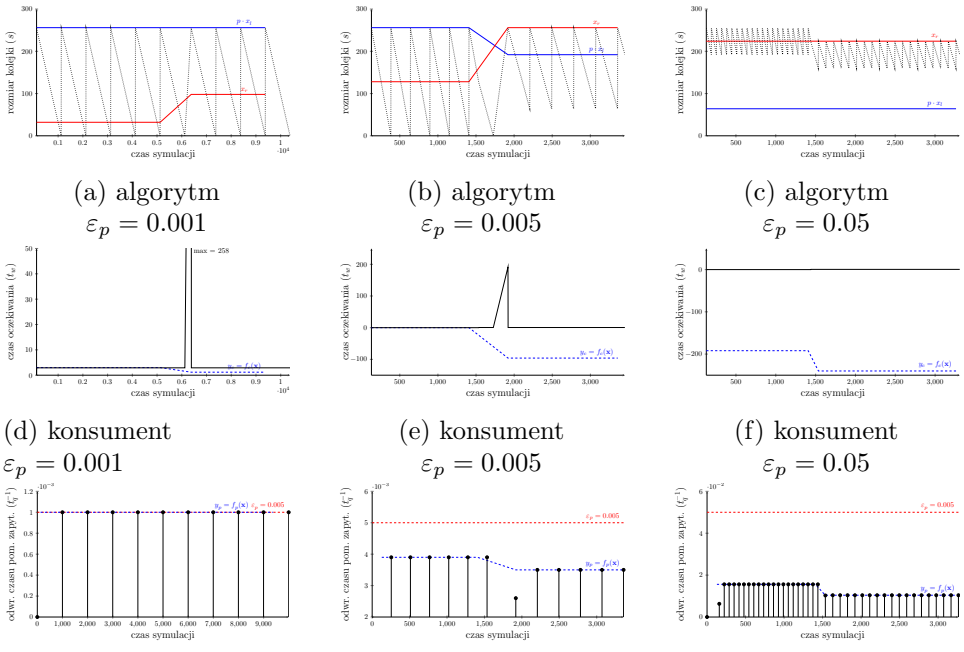
Drugi rodzaj eksperymentów zakładał zmianę warunków pracy w połowie trwania badania na gorsze (zwiększenie obciążenia systemu w połowie eksperymentu). Jego celem była analiza zachowania się każdego z adaptacyjnych algorytmów na skutek nagłego wzrostu czasu produkcji strony.

Badaniom poddano trzy aspekty zachowania się algorytmów: parametry algorytmu (x_l oraz x_r razem z rozmiarem kolejki s), czas oczekiwania t_w wraz z nałożonym ograniczeniem (jeśli występowało) ε_c i parametrem $y_c = f_c(\mathbf{x})$ będącym wartością funkcji celu konsumenta, jak również odwrotność czas pomiędzy zapytaniami t_q^{-1} wraz z nałożonym ograniczeniem (jeśli występowało) ε_p oraz parametrem $y_p = f(\mathbf{x})$ będącym wartością funkcji celu producenta.

9.4.1 Algorytm TRAFF

Jako pierwszy, badaniom poddano algorytm TRAFF, dla trzech różnych wartości ograniczenia producenta ε_p: 0.001 (znaczne), 0.005 (średnie) oraz 0.05 (nieznaczne). Wybór tychże został podyktowany spostrzeżeniami z poprzedniego badania (dla stałych warunków pracy). Wyniki eksperymentu przedstawiono na rysunku 9.7, na którym każdy wiersz oznacza rodzaj analizowanego zachowania, a każda kolumna związana jest z przyjętą wartością ograniczenia algorytmu.

W przypadku parametrów algorytmu widoczny jest wpływ ograniczenia ε_p na sposób uzupełniania kolejki nowymi danymi. Dla najbardziej restrykcyjnego przypadku (rysunek 9.7a) kolejka przez cały czas symulacji pracuje w całym swoim zakresie – zmianie nie ulega wartość parametru x_l (na wykresie przeskalowana o rozmiar strony dla czytelności), zmienia się natomiast moment, w którym rozpoczynają się zapytania (parametr x_r). Wraz ze wzrostem dopuszczalnej wartości obciążenia producenta, zmienia się sposób zarządzania kolejką w momencie zmiany obciążenia systemu (parametr ϑ).



(a) algorytm $\varepsilon_p = 0.001$ (b) algorytm $\varepsilon_p = 0.005$ (c) algorytm $\varepsilon_p = 0.05$

(d) konsument $\varepsilon_p = 0.001$ (e) konsument $\varepsilon_p = 0.005$ (f) konsument $\varepsilon_p = 0.05$

(g) producent $\varepsilon_p = 0.001$ (h) producent $\varepsilon_p = 0.005$ (i) producent $\varepsilon_p = 0.05$

Rysunek 9.7: Charakterystyka pracy algorytmu TRAFF w zmiennym środowisku

Dla średniej wartości ograniczenia algorytmu (rysunek 9.7b), w momencie przełomowym następuje zmniejszenie rozmiaru zapytań (parametr x_l), co skutkuje zmniejszeniem aktywnego zakresu pracy kolejki. W przypadku najmniejszego ograniczenia algorytmu (rysunek 9.7c) kolejka przez cały czas symulacji pracuje w małym zakresie, a zadawane zapytania są drobne (po jednej stronie), gdyż algorytm TRAFF stara się minimalizować czas oczekiwania konsumenta w momencie, gdy spełnione jest jego ograniczenie.

Wpływ ograniczenie producenta ε_p jest również zauważalny w przypadku badania parametrów konsumenta. Dla znacznego ograniczenia (rysunek 9.7d), w momencie zmiany obciążenia systemu następuje wyraźny skok czasu oczekiwania konsumenta – przez moment kolejka jest pusta i brakuje w niej rekordów. Sytuacja taka wynika z pracy kolejki w pełnym zakresie. Zmianie ulega również wartość funkcji celu konsumenta, która jednak w przypadku algorytmu TRAFF nie jest brana pod uwagę i została przedstawiona na wykresie jedynie w celach poglądowych. W przypadku średniego ograniczenia (rysunek 9.7e), również rejestrowany jest skok, ale niższy niż w poprzednio omówionej sytuacji. Dla odmiany, w przypadku nieistotnego ograniczenia (rysunek 9.7f), czas oczekiwania konsumenta wynosi zawsze zero, gdyż kolejka utrzymywana jest przez cały czas symulacji na wysokim poziomie

załadowania.

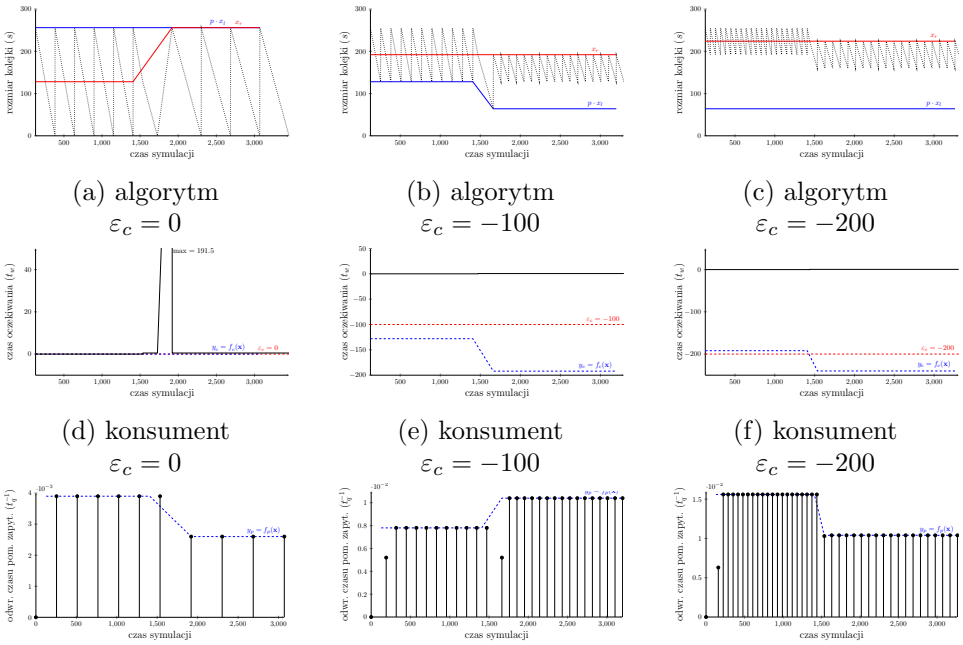
Ostatni mierzony aspekt w niniejszym badaniu to wpływ ograniczenia ε_p na parametry producenta. Dla najbardziej restrykcyjnej wartości tego ograniczenia (rysunek 9.7g), zmierzone wartości bieżącego obciążenia producenta utrzymują się dokładnie na dopuszczalnej granicy wyznaczonej przez ograniczenie algorytmu. Podobny przebieg wykazuje wartość funkcji celu producenta. W przypadku mniej restrykcyjnych ograniczeń (rysunek 9.7h oraz 9.7i), bieżące obciążenie producenta oraz wartości funkcji celu producenta znajdują się przez cały okres analizy poniżej ograniczenia algorytmu i cechują się zauważalnym spadkiem w momencie zmiany obciążenia systemu (parametr ϑ) w połowie czasu symulacji.

9.4.2 Algorytm LATEN

Analogiczne badania przeprowadzono dla algorytmu LATEN, przyjmując trzy arbitralnie wybrane wartości ograniczenia konsumenta ε_c: 0 (nieznaczne), -100 (średnie) oraz -200 (znaczne), również wybrane jako reprezentatywne na podstawie wcześniejszych badań dla stałych warunków pracy. Rysunek 9.8 przedstawia podsumowanie wyników badań. Każdy wiersz związany jest z innym zachowaniem będącym przedmiotem badania, a każda kolumna – z różną wartością ograniczenia.

Wraz ze wzrostem restrykcyjności ograniczenia algorytmu zauważalna jest zmiana aktywnego zakresu pracy kolejki, która jest odwrotna w stosunku do tej zarejestrowanej dla algorytmu TRAFF. Dla nieznacznego ograniczenia (rysunek 9.8a), kolejka jest używana w całym zakresie przez cały czas symulacji, zmianie ulega jedynie moment rozpoczęcia nowego zapytania (parametr x_r). Dla średniego ograniczenia (rysunek 9.8b), kolejka jest wypełniana jedynie w części swojego zakresu, a dodatkowo rozmiar zapytania zmienia się w momencie przełomowym na mniejszy. Zastosowanie najbardziej restrykcyjnego ograniczenia (rysunek 9.8c), wiąże się ze zgłaszaniem przez algorytm LATEN najmniejszego możliwego zapytania bez względu na aktualne warunki pracy, nie zmienia się również moment rozpoczynania zapytań.

W przypadku czasu oczekiwania widoczny jest wpływ wartości ograniczenia konsumenta na stabilność pracy konsumenta. Dla zerowych wartości tego oczekiwania (rysunek 9.8d), w momencie pogorszenia się obciążenia systemu zarejestrowany jest skok czasu oczekiwania spowodowany brakiem rekordów w kolejce. Nie występuje on w przypadku ujemnych wartości tego oczekiwania (rysunek 9.8e oraz 9.8f). Na wspomnianych trzech wykresach przedstawiono również samo ograniczenie, jak i wartość funkcji celu konsumenta. Należy tutaj zwrócić uwagę na fakt, iż ograniczenie konsumenta ε_c dla ujemnych jego wartości nie dotyczy wprost mierzonego czasu oczekiwania t_w, a raczej zapas czasu rozumiany jako niezerową zajętość kolejki w momencie załadowania do niej nowych danych – wartości zmierzonego



(a) algorytm $\varepsilon_c = 0$ (b) algorytm $\varepsilon_c = -100$ (c) algorytm $\varepsilon_c = -200$

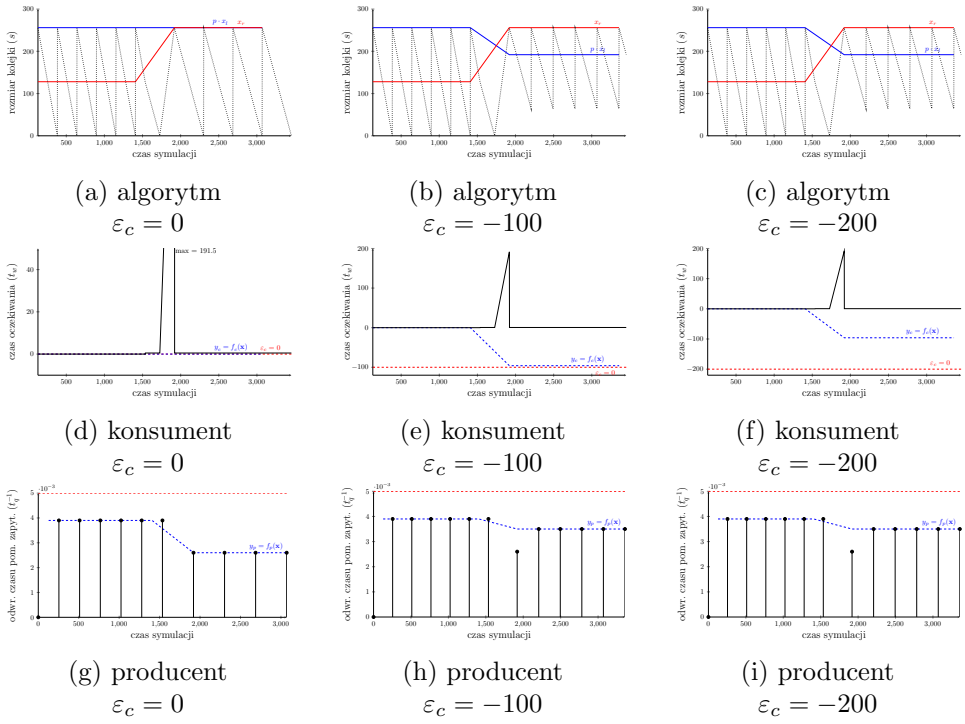
(d) konsument $\varepsilon_c = 0$ (e) konsument $\varepsilon_c = -100$ (f) konsument $\varepsilon_c = -200$

(g) producent $\varepsilon_c = 0$ (h) producent $\varepsilon_c = -100$ (i) producent $\varepsilon_c = -200$

Rysunek 9.8: Charakterystyka pracy algorytmu LATEN w zmiennym środowisku

czasu oczekiwania nie mogą być ujemne. Na ostatnim z omawianych trzech wykresów (rysunek 9.8f), widoczna jest sytuacja, w której wyznaczona przez algorytm wartość funkcji celu konsumenta jest większa od przyjętego ograniczenia – jest to wspomniany przy okazji opisu algorytmu LATEN przypadek awaryjny, w którym dla bieżącego zbioru parametrów algorytm nie jest w stanie zagwarantować przestrzegania ograniczenia.

Ostatnim analizowanym aspektem było bieżące obciążenie producenta – przedstawiono je, razem z wartością funkcji celu producenta (w celach poglądowych), pomimo brak jej analizy przez algorytm. Wpływ przyjętego ograniczenia konsumenta na ten aspekt jest nieoczywisty dla algorytmu LATEN. W przypadku nieznacznego (rysunek 9.8g) oraz znacznego (rysunek 9.8i) ograniczenia, zarejestrowany został spadek obciążenia producenta, podczas gdy dla średniego ograniczenia (rysunek 9.8h) – wzrost. Sytuacja taka wynika wprost ze zmiany parametru x_l (rysunek 9.8b), występującej jedynie dla ograniczenia konsumenta równego -100. Niemniej, algorytm LATEN nie bierze w ogóle pod uwagę obciążenia producenta, a jedynie komfort pracy konsumenta i w ten właśnie sposób dobiera wartości parametrów x_l i x_r.



(a) algorytm $\varepsilon_c = 0$ (b) algorytm $\varepsilon_c = -100$ (c) algorytm $\varepsilon_c = -200$

(d) konsument $\varepsilon_c = 0$ (e) konsument $\varepsilon_c = -100$ (f) konsument $\varepsilon_c = -200$

(g) producent $\varepsilon_c = 0$ (h) producent $\varepsilon_c = -100$ (i) producent $\varepsilon_c = -200$

Rysunek 9.9: Charakterystyka pracy algorytmu HYBRI w zmiennym środowisku dla $\varepsilon_p = 0.005$

9.4.3 Algorytm HYBRI

Badania dla algorytmu HYBRI, ze względu na uwzględnianie przez niego obu ograniczeń, wykonano dla arbitralnie wybranej wartości ograniczenia producenta $\varepsilon_p = 0.005$, zmieniając wartość ograniczenia konsumenta ε_c w sposób analogiczny, jak w przypadku algorytmu LATEN. Wybór wartości ograniczenia producenta podyktowany został spostrzeżeniami z poprzednich badań – odpowiadała ona przypadkowi środkowemu, w którym uwidaczniał się wpływ zastosowanego ograniczenia, ale nie dominował on zachowania algorytmu. Rysunek 9.9 przedstawia podsumowanie badań przeprowadzonych dla algorytmu HYBRI.

Zachowanie się algorytmu HYBRI stanowi połączenie zachowań dwóch poprzednich algorytmów. Jest to wyraźnie widoczne dla wszystkich trzech analizowanych aspektów – parametrów: algorytmu, konsumenta oraz producenta. Dla niewielkiego ograniczenia konsumenta (rysunki 9.9a, 9.9d i 9.9g), algorytm zachowuje się w sposób zbliżony do algorytmu LATEN (rysunki 9.8a, 9.8d i 9.8g). Wynika to z faktu, iż niewielkie ograniczenie konsumenta daje algorytmowi swobodę w optymalizacji funkcji celu producenta, co z kolei powoduje spełnienie ograniczenia producenta bez dodatkowych działań.

Przy bardziej restrykcyjnych ograniczeniach konsumenta (rysunek 9.9

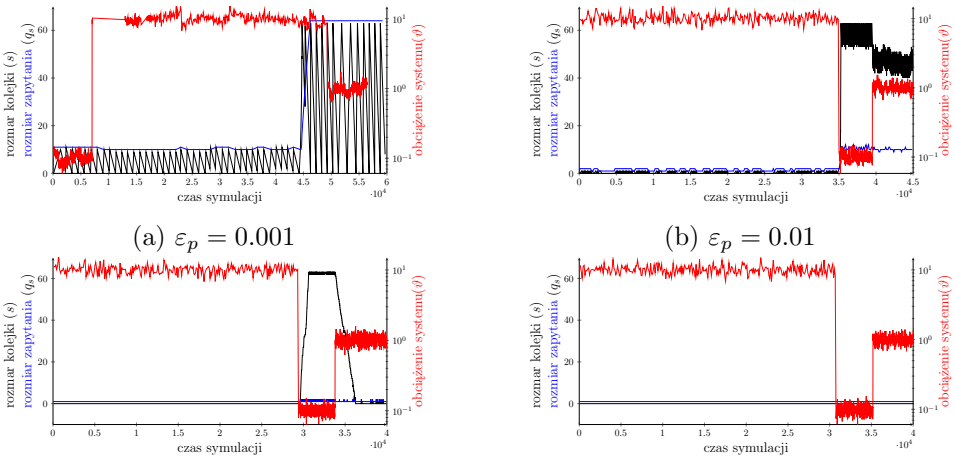
dla $\varepsilon_c < 0$), algorytm HYBRI wykazuje podobieństwo do algorytmu TRAFF (rysunek 9.7 dla $\varepsilon_p = 0.005$). Wyjaśnieniem takiego stanu rzeczy jest ograniczenie producenta, które zaczyna mieć wpływ na zachowanie algorytmu w momencie mniejszej swobody pozostawionej ze strony ograniczenia konsumenta. To pierwsze jest nieprzekraczalne, a więc algorytm HYBRI jest zmuszony wymusić odpowiednią częstotliwość zapytań kosztem pogorszenia jakości usług konsumenta.

9.5 Weryfikacja dla strumieni danych

Ostatni rodzaj eksperymentów weryfikacyjnych zakładał przetestowanie algorytmów adaptacyjnych dla strumienia jako źródła danych. Sytuacja taka ma miejsca w momencie, w którym do silnika CUBIT wystosowane zostaje żądanie o najnowsze dane. Najważniejszą różnicą w stosunku do pracy z historycznym źródłem danych jest brak dostępności wszystkich wymaganych rekordów od razu, co pociąga za sobą konieczność nieustannego oczekiwania na nowe.

Strumieniowy charakter źródła wymusza zastosowanie strony o rozmiarze 1, co w praktyce oznacza brak stosowania strony jako takiej. Co więcej, początkowa liczba stron (rekordów) ładowanych do kolejki układu stronicowania również została ustawiona na 1. Odmiennie niż w przypadków dwóch poprzednich grup eksperymentów, zdecydowano się na wprowadzenie elementu losowości do czasów: produkcji i konsumpcji – w obu przypadkach podane niżej wartości parametrów t_p oraz t_c oznaczają średnią, która razem ze względnym odchyleniem standardowym równym 10% formują parametry rozkładu normalnego. Zdecydowano się na wybór takiego właśnie rozkładu, ze względu na chęć przybliżenia charakterystyki czasowej rzeczywistego źródła, wysyłającego kolejne porcje danych ze stałym interwałem czasowym – w tym przypadku odchylenie od średniej symuluje niedokładność pracy źródła oraz opóźnienia wynikłe z przesyłu danych.

Zdecydowano się również na trójfazowy scenariusz, zakładający dwie zmiany warunków pracy, następujące po przetworzeniu odpowiednio $\frac{1}{3}$ oraz $\frac{2}{3}$ wszystkich rekordów. Rozpatrzono dwa warianty takiego scenariusza: HLM (ang. *High, Low, Medium*), w którym symulacja rozpoczyna się z wysokim obciążeniem systemu ($\vartheta = 10$), przechodzi przez niskie ($\vartheta = 0.1$) i kończy na średnim ($\vartheta = 1$) oraz LHM (ang. *Low, High, Medium*), w którym najpierw występuje faza niskiego obciążenia, następnie wysokiego, a na koniec średniego. Decyzja o wyborze dwóch wariantów została podyktowana chęcią sprawdzenia w jaki sposób poszczególne algorytmy są w stanie wykorzystać okres mniejszego obciążenia do przygotowania się na potencjalnie większe w przyszłości.



(a) $\varepsilon_p = 0.001$ (b) $\varepsilon_p = 0.01$

(c) $\varepsilon_p = 0.1$ (d) $\varepsilon_p = 1.0$

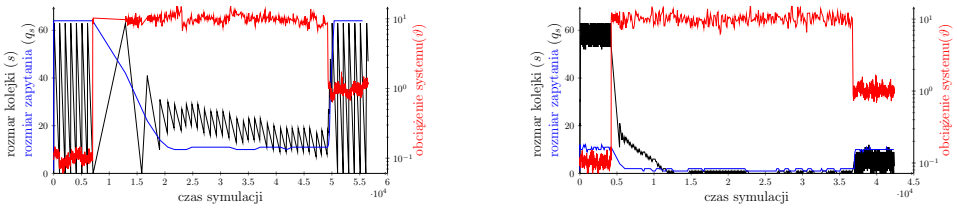
Rysunek 9.10: Charakterystyka pracy algorytmu TRAFF dla zmiennego środowiska (HLM) i strumieniowego źródła danych

9.5.1 Algorytm TRAFF

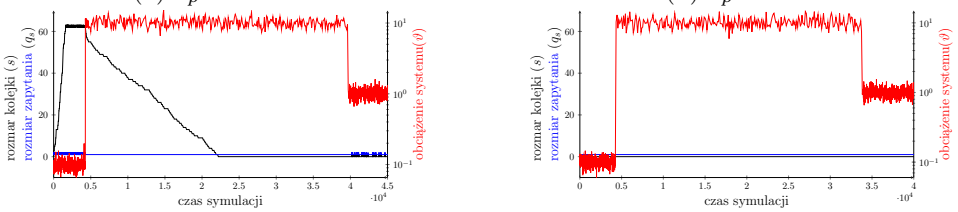
W przypadku algorytmu TRAFF, przyjęto następujące wartości ograniczenia producenta: $\varepsilon_p = 0.001, 0.01, 0.1, 1.0$. W badaniach skupiono się na jednym aspekcie: zarządzaniu kolejką, mierząc przy tym zarówno jej bieżącą objętość, jak i rozmiar zgłaszanych zapytań. Rysunek 9.10 przedstawia podsumowanie wyników rzeczonego badania dla scenariusza HLM. Na każdym z wykresów naniesiono, w oddzielnej skali i w celach poglądowych, bieżący przebieg obciążenia systemu (parametr ϑ).

Na wszystkich czterech wykresach rysunku 9.10 przedstawia się wyraźna zależność od zakresu pracy kolejki od przyjętego ograniczenia producenta. Im ograniczenie jest bardziej restrykcyjne (mniejsze wartości), tym kolejka pracuje w większym zakresie, a zgłaszane zapytania są większe – kosztem komfortu pracy konsumenta zapytania zadawane są rzadziej. Co więcej, na wspomnianych wykresach widoczny jest również wpływ zmieniającego się obciążenia systemu. Wpływ ten jest tym mniejszy, im mniej restrykcyjne (większe wartości) jest ograniczenie producenta. Dla najbardziej restrykcyjnego ograniczenia (rysunek 9.10a) kolejka jest uzupełniana do około $\frac{1}{6}$ swojej pojemności dla wysokiego obciążenia, podczas gdy po jego zmianie na mniejsze pracuje w pełnym swoim zakresie. Z kolei w sytuacji najmniejszego ograniczenia producenta (rysunek 9.10d) bez względu na bieżące obciążenie systemu, kolejka jest uzupełniana z największą częstotliwością, zaraz po pobraniu każdego rekordu, przy czym jej objętość utrzymuje się na stały poziomie równym 0 – zaraz po załadowaniu, rekordy są pobierane przez konsumenta.

Rysunek 9.11 przedstawia wyniki analogicznego badania, ale dla sce-



(a) $\varepsilon_p = 0.001$ (b) $\varepsilon_p = 0.01$



(c) $\varepsilon_p = 0.1$ (d) $\varepsilon_p = 1.0$

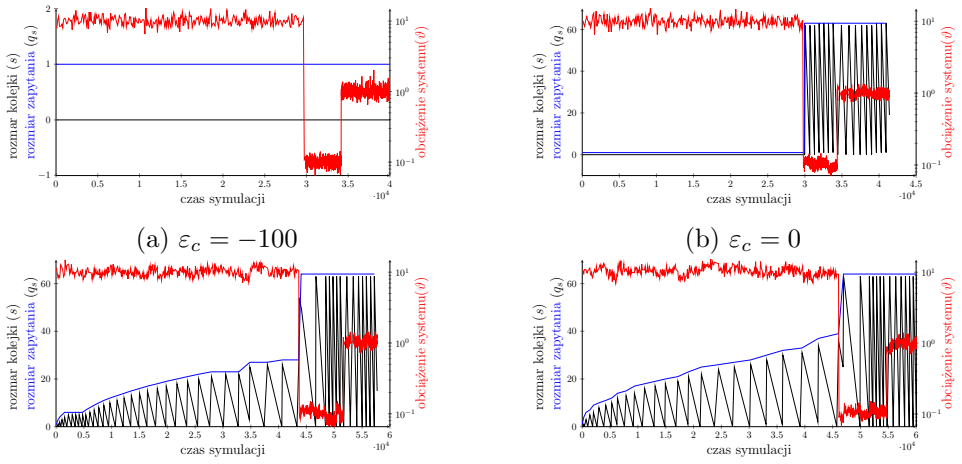
Rysunek 9.11: Charakterystyka pracy algorytmu TRAFF dla zmiennego środowiska (LHM) i strumieniowego źródła danych

nariusza LHM. Zmiana kolejności okresów wysokiego i niskiego obciążenia systemu dość wyraźnie wpłynęła na charakterystykę pracy kolejki. W przypadku najbardziej restrykcyjnego ograniczenia (rysunek 9.11a), kolejka z początku pracuje w pełnym swoim zakresie, a po zmianie obciążenia na wysokie jej roboczy zakres zawęża się przesuwa w stronę mniejszych wartości, by ponownie powrócić do pierwotnego zakresu po drugiej zmianie, na średnie obciążenie. Podobna zależność, choć z malejącym zakresem roboczym kolejki, występuje również na pozostałych wykresach, z wyjątkiem ostatniego, który jest zbliżony w charakterystyce do analogicznego przypadku dla scenariusza HLM.

9.5.2 Algorytm LATEN

Badania dla algorytmu LATEN przeprowadzono dla następujących wartości ograniczenia konsumenta $\varepsilon_c = -100, 0, 100, 200$. Analizowano dokładnie te same parametry, co w przypadku algorytmu TRAFF. Rysunek 9.12 przedstawia wyniki badań dla scenariusza HLM.

W przypadku najbardziej rygorystycznego ograniczenia (rysunek 9.12a), dane są przekazywane niemalże bezpośrednio od producenta do konsumenta – podobnie, jak to miało miejsce w przypadku najmniej restrykcyjnego ograniczenia producenta w algorytmie TRAFF. W sytuacji, w której ograniczenie konsumenta ulega rozluźnieniu (większe wartości) obserwowane jest poszerzenie zakresu roboczego kolejki. Dla wartości $\varepsilon_c = 0$ (rysunek 9.12b) następuje ono dopiero w momencie obniżenia obciążenia systemu, podczas gdy dla $\varepsilon_c > 0$ można zauważyć dążenia algorytmu LATEN do zapełnienia kolejki przy jednoczesnym nieprzekroczeniu ograniczenia konsumenta, de-



(a) $\varepsilon_c = -100$ (b) $\varepsilon_c = 0$

(c) $\varepsilon_c = 100$ (d) $\varepsilon_c = 200$

Rysunek 9.12: Charakterystyka pracy algorytmu LATEN dla zmiennego środowiska (HLM) i strumieniowego źródła danych

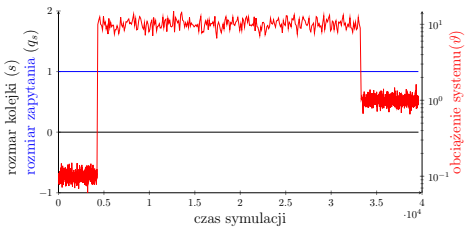
finiującego akceptowalne opóźnienie w pozyskiwaniu przez niego danych – z każdym kolejnym zapytaniem zwiększa się zapełnienie kolejki.

Wyniki badań dla drugiego scenariusza (LHM) zostały przedstawione na rysunku 9.13. Jedynie w przypadku ujemnej wartości ograniczenia konsumenta (rysunek 9.13a), przesył danych pomiędzy producentem a konsumentem następuje niemalże natychmiastowo. W pozostałych przypadkach ($\varepsilon_c \geqslant 0$) algorytm LATEN wykorzystuje fakt niskiego obciążenia systemu na początku symulacji do zapełnienia całej kolejki danymi, co skutkuje możliwością utrzymania jej pełnego zakresu roboczego przez cały okres symulacji.

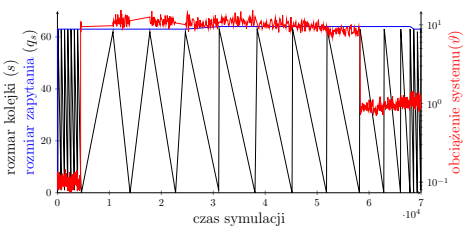
9.5.3 Algorytm HYBRI

W celu przebadania ostatniego algorytmu (HYBRI) zdecydowano się na wybranie czterech par wartości obu ograniczeń: $(\varepsilon_p = 0.001, \varepsilon_c = -100)$, $(\varepsilon_p = 0.001, \varepsilon_c = 100)$, $(\varepsilon_p = 0.1, \varepsilon_c = -100)$, $(\varepsilon_p = 0.1, \varepsilon_c = 100)$. Uzasadnieniem takiego wyboru była chęć sprawdzenia wzajemnego wpływu skrajnych (względem poprzednich badań) wartości obu ograniczeń. Założeniem algorytmu HYBRI jest bezwzględne przestrzeganie ograniczenia producenta, przy dążeniu do nieprzekroczenia również ograniczenia konsumenta. Rysunek 9.14 przedstawia podsumowanie wyników badania dla scenariusza HLM.

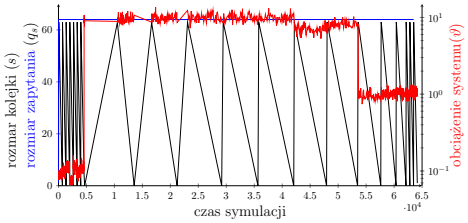
Zachowanie się algorytmu HYBRI dla restrykcyjnych wartości obu ograniczeń (rysunek 9.14a) jest zbliżone do działania algorytmu TRAFF – z jednej strony algorytm HYBRI stara się nie przekroczyć ograniczenia producenta, a z drugiej dąży do minimalizacji funkcji celu konsumenta. W przypadku łagodniejszego ograniczenia konsumenta (rysunek 9.14b) zauważalne staje się podobieństwo do algorytmu LATEN. Algorytm nie próbuje już za wszel-



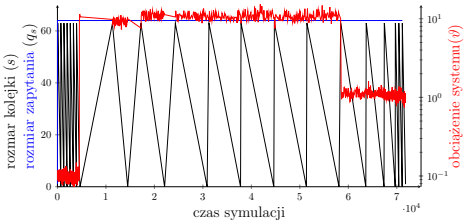
(a) $\varepsilon_c = -100$



(b) $\varepsilon_c = 0$

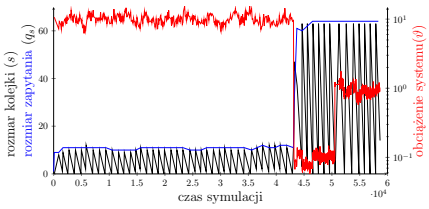


(c) $\varepsilon_c = 100$

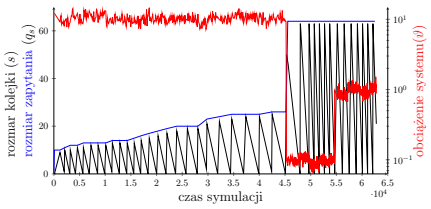


(d) $\varepsilon_c = 200$

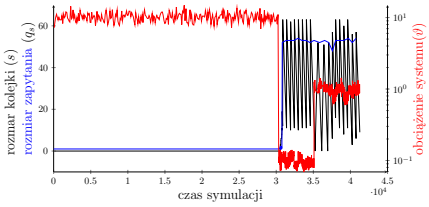
Rysunek 9.13: Charakterystyka pracy algorytmu LATEN dla zmiennego środowiska (LHM) i strumieniowego źródła danych



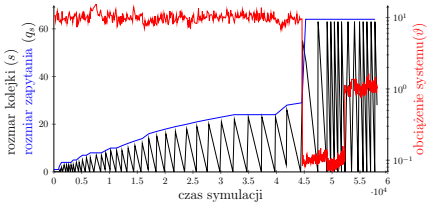
(a) $\varepsilon_p = 0.001$
$\varepsilon_c = -100$



(b) $\varepsilon_p = 0.001$
$\varepsilon_c = 100$

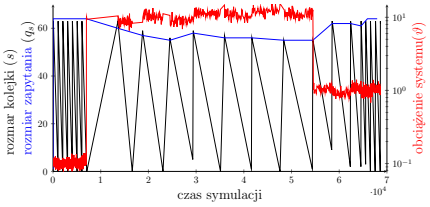


(c) $\varepsilon_p = 0.1$
$\varepsilon_c = -100$

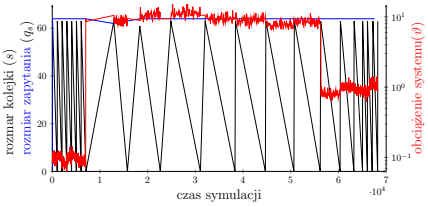


(d) $\varepsilon_p = 0.1$
$\varepsilon_c = 100$

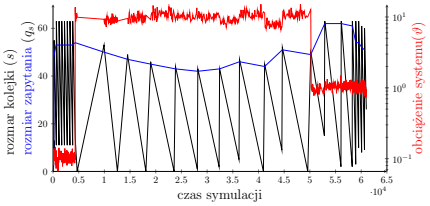
Rysunek 9.14: Charakterystyka pracy algorytmu HYBRI dla zmiennego środowiska (HLM) i strumieniowego źródła danych



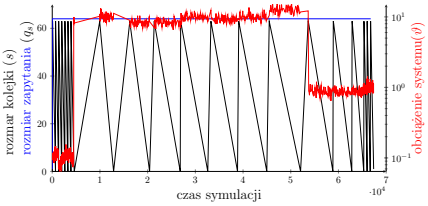
(a) $\varepsilon_p = 0.001$
$\varepsilon_c = -100$



(b) $\varepsilon_p = 0.001$
$\varepsilon_c = 100$



(c) $\varepsilon_p = 0.1$
$\varepsilon_c = -100$



(d) $\varepsilon_p = 0.1$
$\varepsilon_c = 100$

Rysunek 9.15: Charakterystyka pracy algorytmu HYBRI dla zmiennego środowiska (LHM) i strumieniowego źródła danych

ką cenę (przy zachowaniu ograniczenia producenta) minimalizować funkcji celu konsumenta, lecz wykorzystuje dopuszczalne opóźnienie, którego konsument może doświadczyć. W przypadku, w którym ograniczenie producenta staje się niewielkie, a ograniczenie konsumenta restrykcyjne (przedstawia to rysunek 9.14c), algorytm stara się przede wszystkim minimalizować opóźnienie odczuwane przez konsumenta, co uwidacznia się po zmianie obciążenia systemu na niskie. Dopiero wówczas następuje większe wykorzystanie kolejki oraz zgłaszanie rzadszych i większych zapytań. W momencie, w którym oba ograniczenia są niewielkie (rysunek 9.14d), zachowanie algorytmu HYBRI jest zbliżone do omówionego już przypadku przedstawionego na rysunku 9.14b, lecz z tą różnicą, iż algorytm ma większe możliwości obciążenia producenta poprzez stosowanie częstszych zapytań.

Dla scenariusza LHM (rysunek 9.15) zachowanie algorytmu HYBRI jest stosunkowo mało zróżnicowane, niezależnie od zastosowanych ograniczeń. Na każdym z czterech wykresów można zauważyć wpływ obciążenia systemu na częstotliwość opróżniania i napełniania kolejki, która zawsze pracuje w pełnym lub prawie pełnym zakresie. Powyższe jest możliwe do osiągnięcia dzięki początkowej fazie niskiego obciążenia systemu.

9.6 Porównanie ze względu na parametry kolejki

Pierwszy rodzaj eksperymentów porównawczych polegał na sprawdzeniu wpływu rozmiaru kolejki układu stronicowania oraz rozmiaru zapytania za-

danego do silnika CUBIT na jakość usług: producenta i konsumenta oraz czas oczekiwania na nowe dane. W tym celu wybrano dwa rozmiary kolejki: $4p$ i $32p$, gdzie p oznacza rozmiar strony będący wartością stałą równą 64. Zdecydowano się na niezmienność rozmiaru strony z tego względu, iż badania wykonywano dla różnych wartości obciążenia systemu ϑ, który to parametr łączy w sobie zarówno czas produkcji t_p, czas konsumpcji t_c, jak i właśnie rozmiar strony p.

Rozmiar zapytania (parametr λ) również był zmienny – wybrano dwie wartości: $10c$ oraz $100c$, gdzie c jest rozmiarem kolejki. W konsekwencji wykonano badania w czterech wariantach (dwa rozmiary kolejki i dwa rozmiary zapytania), a każde badanie przeprowadzono, podobnie jak w przypadku wcześniejszych eksperymentów, dla szerokiego zakresu obciążeń systemu.

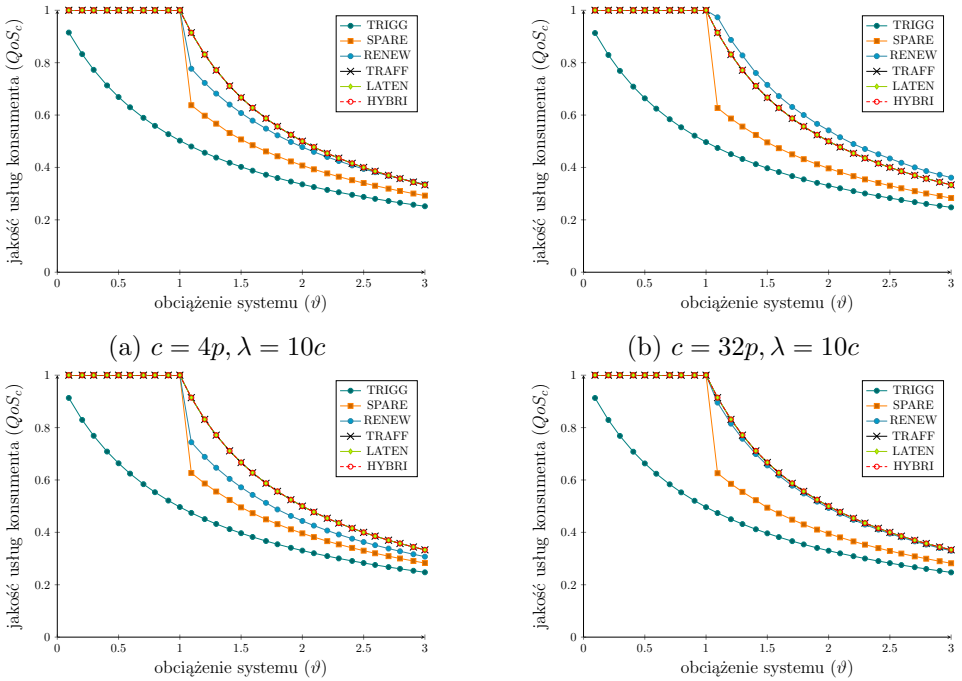
Należy zwrócić uwagę na fakt, iż pomimo nie brania pod uwagę ograniczeń algorytmów, konieczne było nadanie im jakichś wartości. Zdecydowano się na wybór wartości nieznaczących, powodujących praktyczny brak ich uwzględniania przez nowe algorytmy. W przypadku ograniczenia konsumenta wybrano wartość 0, zaś w przypadku ograniczenia producenta – wartość 1.

Każdy z eksperymentów przeprowadzony był dla wszystkich sześciu algorytmów wypełniania stron: trzech stałych oraz trzech adaptacyjnych. Na rysunku 9.16 przedstawiono podsumowanie pomiarów jakości usług konsumenta dla opisanych wcześniej czterech wariantów. Istotnym spostrzeżeniem jest identyczne zachowanie wszystkich trzech nowych algorytmów wypełniania stron – odpowiadające im trzy przebiegi nakładają się na siebie dla wszystkich czterech wariantów. Taka sytuacja wynika z zastosowania nieznaczących wartości ograniczeń algorytmów.

Zestawiając wyniki uzyskane dla nowych algorytmów z tymi uzyskanymi dla algorytmów starych, zarysowuje się widoczna zależność: nowe algorytmy radzą sobie zdecydowanie lepiej niż TRIGG, lepiej niż SPARE i porównywalnie z RENEW pod względem zapewnienia jakości usług konsumenta. W przypadku tego ostatniego algorytmu, uzyskuje on nieco lepsze wyniki niż nowe algorytmy dla mniejszych zapytań, zaś nieznacznie gorsze dla większych zapytań.

Analogiczne badanie dla drugiej miary jakości (usług producenta) przedstawiono na 9.17. W tym przypadku zależności pomiędzy poszczególnymi algorytmami kształtują się nieco odmiennie. Dwa nowe algorytmy: LATEN i HYBRI zdecydowanie dominują nad wszystkimi pozostałymi, przy czym różnica tak jest tym większa, im większy jest rozmiar kolejki.

Algorytm TRAFF uzyskał w tym eksperymencie notowania gorsze od algorytmu TRIGG i porównywalne z algorytmami SPARE i RENEW. Sytuacja taka ma dość trywialne wytłumaczenie – przeznaczeniem algorytmu TRAFF jest przede wszystkim bezwzględne spełnianie nałożonego ograniczenia na maksymalne obciążenie producenta. W przypadku tego eksperymentu zastosowano nieznaczące ograniczenie, gdyż nie istnieje możliwość



(a) $c = 4p, \lambda = 10c$ (b) $c = 32p, \lambda = 10c$

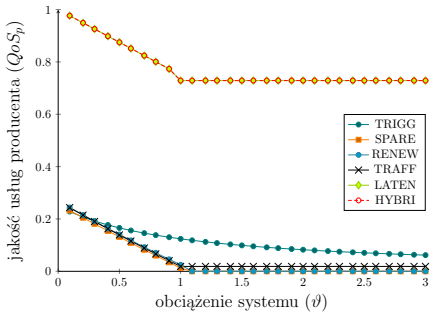
(c) $c = 4p, \lambda = 100c$ (d) $c = 32p, \lambda = 100c$

Rysunek 9.16: Porównanie algorytmów wypełniania stron pod względem jakości usług konsumenta (metryka QoS_c)

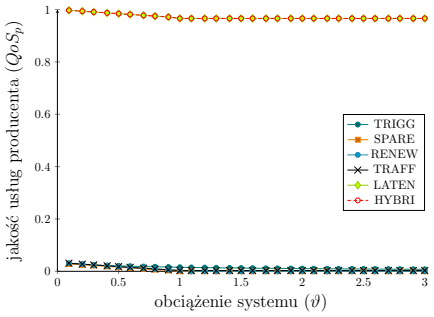
ograniczenia starych algorytmów – w rezultacie algorytm TRIGG działa w sposób jak najbardziej korzystny dla konsumenta, nie przejmując się w ogóle kwestią producenta.

Oprócz mierzenia współczynników jakości usług, dokonano także pomiarów czasu oczekiwania na nowe dane, którego doświadcza konsument. Podsumowanie tego badania przedstawiono na rysunku 9.18, mierząc przy tym średnią oraz maksymalną wartość czasu oczekiwania. Do badań wybrano dwie wartości obciążenia systemu: 1.5 i 2.5 oraz rozmiar zapytania równy dziesięciokrotności rozmiaru kolejki. Wybór pierwszego z omówionych parametrów podyktowany został praktycznym brakiem opóźnienia rejestrowanego dla wartości $\vartheta \leqslant 1$.

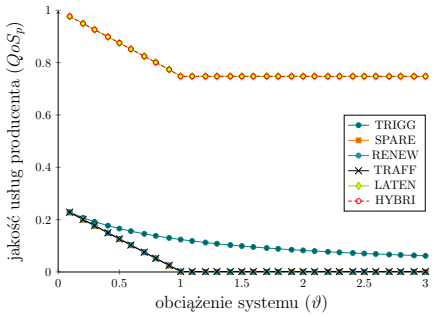
Pierwszy test (rysunek 9.18a) ukazuje zależność widoczną podczas badania miary jakości usług konsumenta – algorytm RENEW dla mniejszych zapytań wykazuje mniejszy średni czas oczekiwania od nowych algorytmów. W pozostałych przypadkach nowe algorytmy charakteryzują się lepszymi notowaniami. Ciekawym uzupełnieniem tego badania jest pomiar maksymalnych wartości czasu oczekiwania (związanych z pikami występującymi w przebiegu tej miary) – przedstawia je rysunek 9.18b. Wyraźnie widoczna jest na nim ważna cecha nowych algorytmów wypełniania stron: stabili-



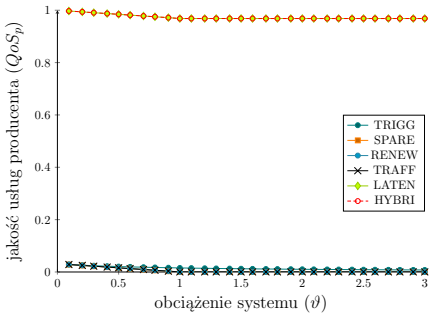
(a) $c = 4p, \lambda = 10c$



(b) $c = 32p, \lambda = 10c$

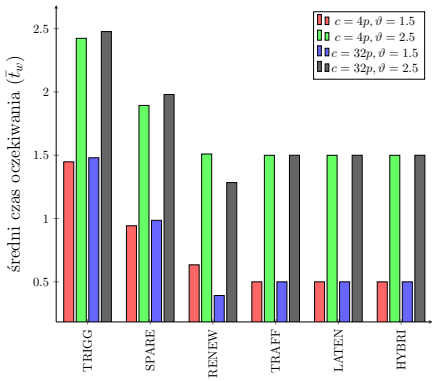


(c) $c = 4p, \lambda = 100c$

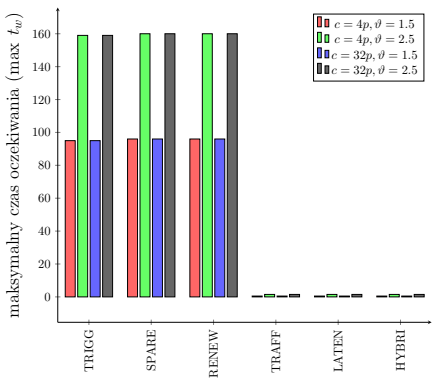


(d) $c = 32p, \lambda = 100c$

Rysunek 9.17: Porównanie algorytmów wypełniania stron pod względem jakości usług producenta (metryka QoS_p)



(a) średnia



(b) maksimum

Rysunek 9.18: Porównanie algorytmów wypełniania stron pod względem czasu oczekiwania (metryka t_w)

zacja przebiegu czasu oczekiwania poprzez wprowadzenie ograniczenia na minimalny czas konsumpcji pojedynczego rekordu. Stare algorytmy osiągają bardzo duże wartości maksymalne czasu oczekiwania – związane jest to z okresowymi brakami danych w kolejce, które nie występują w przypadku nowych algorytmów.

9.7 Porównanie ze względu na wielowątkowość

Drugi rodzaj eksperymentów wykonanych w obrębie grupy badań porównawczych obejmował symulację środowiska wielowątkowego, charakteryzowanego przez pojedynczy parametr wyrażający wzajemny wpływ wątków na siebie. Parametr ten, oznaczony jako k, przyjmował wartości rzeczywiste od 0 do 1, przy czym wartość 0 oznaczała pełną niezależność wątków, podczas gdy wartość 1 – brak wielowątkowości. Wartość parametru k przekładała się na rzeczywisty czas produkcji strony w przypadku algorytmów stałych [32] lub czas realizacji zapytania w przypadku algorytmów adaptacyjnych.

Dla algorytmów stałych przyjęto, że zmodyfikowany czas produkcji pojedynczej strony wynosi:

$$t'_p = (1 + k \cdot w) \cdot t_p, \tag{9.19}$$

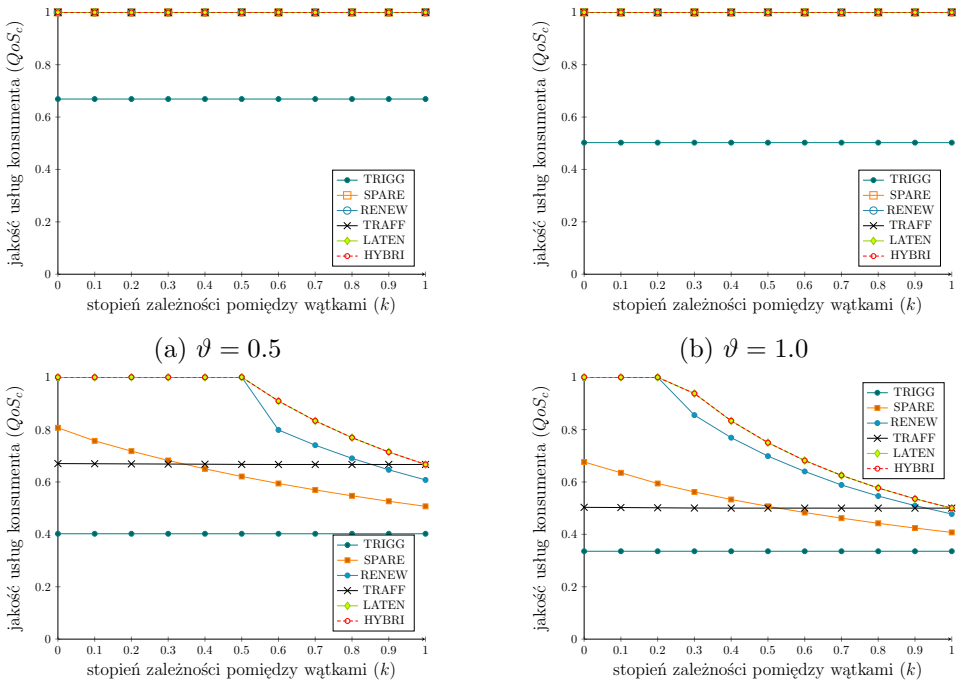
gdzie w oznacza liczbę współbieżnie pracujących wątków odpowiedzialnych za realizację zapytania o pojedynczą stronę. W przypadku, w którym współczynnik $k = 1$, zysk z zastosowania podejścia wielowątkowego jest zerowy, gdyż zmodyfikowany czas produkcji strony jest zwielokrotniony liczbą wątków. W przypadku, gdy $k = 0$, wszystkie wątki, bez względu na ich liczbę, pracują zupełnie niezależnie od siebie i zysk z ich użycia jest maksymalny.

W celu adaptacji powyższej koncepcji na potrzeby nowych algorytmów wypełniania stron zastosowano podejście, w którym czas realizacji zapytania o rozmiarze x_l skalował się zgodnie z wartością współczynnika k w sposób następujący:

$$t'_p = \frac{t_p + k \cdot (x_l - 1) \cdot t_p}{x_l}. \tag{9.20}$$

W efekcie, dla $k = 1$ czas realizacji zapytania o rozmiarze x_l wynosi $x_l \cdot t_p$, podczas gdy dla zerowej wartości współczynnika k, czas wykonania dowolnie dużego zapytania był zawsze równy t_p. Oba zabiegi (równania 9.19 i 9.20) dawały w rezultacie identyczne działanie, co umożliwiało porównanie dwóch generacji algorytmów wypełniania stron.

Badaniom poddano, podobnie jak w przypadku poprzednich eksperymentów porównawczych, dwie metryki jakości usług oraz czas oczekiwania na nowe dane, wyrażony jako średnia oraz maksymalna zarejestrowana wartość. Eksperymenty przeprowadzono dla czterech wybranych wartości obciążenia systemu: 0.5, 1.0, 1.5 oraz 2.0.



(a) $\vartheta = 0.5$ (b) $\vartheta = 1.0$

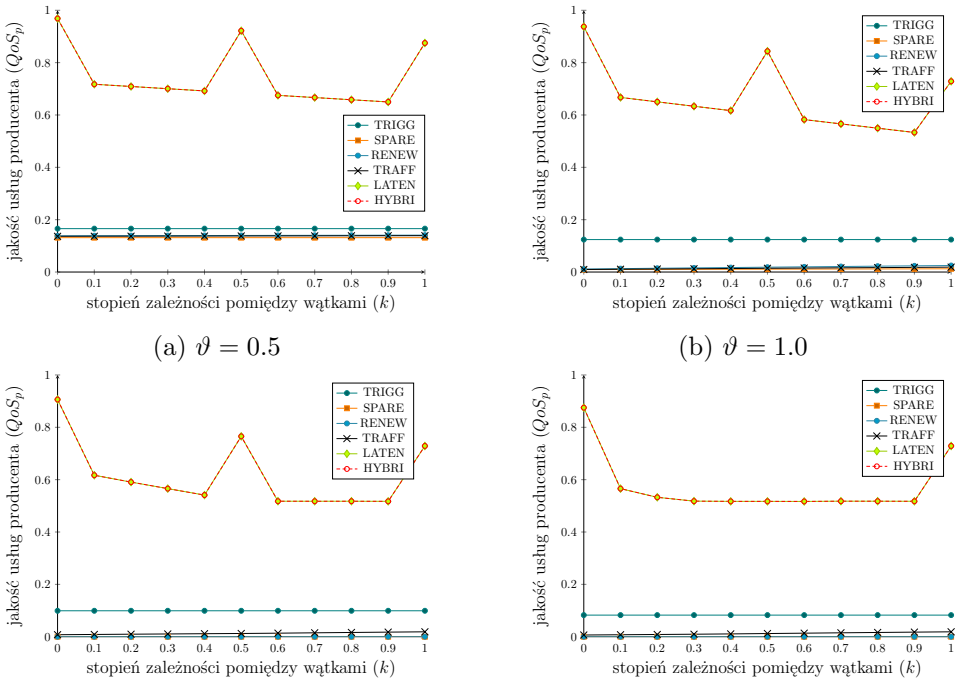
(c) $\vartheta = 1.5$ (d) $\vartheta = 2.0$

Rysunek 9.19: Porównanie algorytmów wypełniania stron pod względem jakości usług konsumenta (metryka QoS_c) – wersja wielowątkowa

Na rysunku 9.19 przedstawiono wyniki pierwszego testu, w którym mierzono jakość usług konsumenta w zależności od stopnia zależności pomiędzy wątkami oraz obciążenia systemu. W przypadku niewielkiego obciążenia ($\vartheta \leqslant 1.0$, rysunki 9.19a i 9.19b), wszystkie algorytmy wykazują brak zależności od wartości parametru k, przy czym poza algorytmem TRIGG, wszystkie cechują się maksymalną jakością usług konsumenta. Wynika to z faktu, iż dla takich obciążeń nie występuje problem braku rekordów w kolejce. W przypadku algorytmu TRIGG problem ten może występować chwilowo, ze względu na charakter jego pracy – zgłaszanie żądania dopiero przy niemalże pustej kolejce.

Przy większych obciążeniach ($\vartheta > 1.0$, rysunki 9.19c i 9.19d), zauważalny jest wpływ wartości parametru k na jakość usług konsumenta, co więcej każdy z algorytmów wykazuje nieco inną charakterystykę. Algorytm TRIGG cechuje się stałą wartością – zgodnie z założeniami nie jest on w stanie zgłosić więcej niż jednego żądania na raz, przez co nie czerpie żadnych korzyści z potencjalnej wielowątkowości. Podobnie w tym zestawieniu wygląda algorytm TRAFF – pracując dla nieistotnej wartości ograniczenia producenta zgłasza on zawsze żądanie o pojedynczą stronę.

Algorytm SPARE, który jest w stanie zgłosić do dwóch stron na raz, wy-



(a) $\vartheta = 0.5$ (b) $\vartheta = 1.0$

(c) $\vartheta = 1.5$ (d) $\vartheta = 2.0$

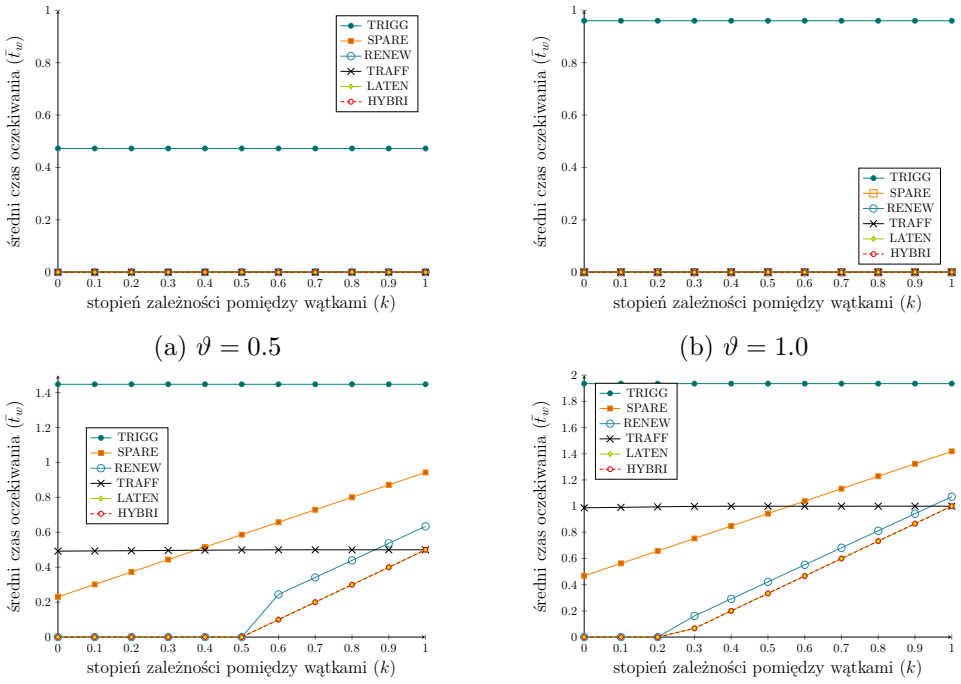
Rysunek 9.20: Porównanie algorytmów wypełniania stron pod względem jakości usług producenta (metryka QoS_p) – wersja wielowątkowa

kazuje się niemalże liniową zależnością. Algorytmy RENEW oraz LATEN i HYBRI charakteryzują się utrzymywaniem maksymalnej jakości usług konsumenta do pewnej wartości parametru k, a po jej przekroczeniu jakość usług zaczyna mniej więcej liniowo spadać. Wszystkie te trzy algorytmy są w stanie formować zapytania o wiele stron lub zadawać wiele zapytań o jedną stronę na raz. Dwa nowe algorytmy: LATEN i HYBRI wypadają w tym zestawieniu najlepiej.

Wyniki analogicznego badania, wykonanego dla drugiej miary jakości (usług producenta) przedstawiono na rysunku 9.20. W tym przypadku wpływ obciążenia systemu jest mniej znaczący, choć zauważalny. Algorytmy LATEN i HYBRI wyraźnie odstają od reszty, oscylując w górnym zakresie mierzonej miary jakości. Wynika to z faktu zadawania przez nich mniejszej liczby zapytań o większą liczbę stron.

Wyniki badania średniego czasu oczekiwania zostały przedstawione na rysunku 9.21. Dla niewielkich obciążeń systemu ($\vartheta \leqslant 1.0$, rysunki 9.21a i 9.21b), jedynie algorytm TRIGG charakteryzuje się niezerowymi wartościami. Pozostałe algorytmy nie generują opóźnień. Sytuacja ta jest zgodna z przedstawionym na rysunku 9.19 badaniem dla jakości miary konsumenta.

Dla większych obciążeń systemu ($\vartheta > 1.0$, rysunki 9.21c i 9.21d), dla



(a) $\vartheta = 0.5$ (b) $\vartheta = 1.0$

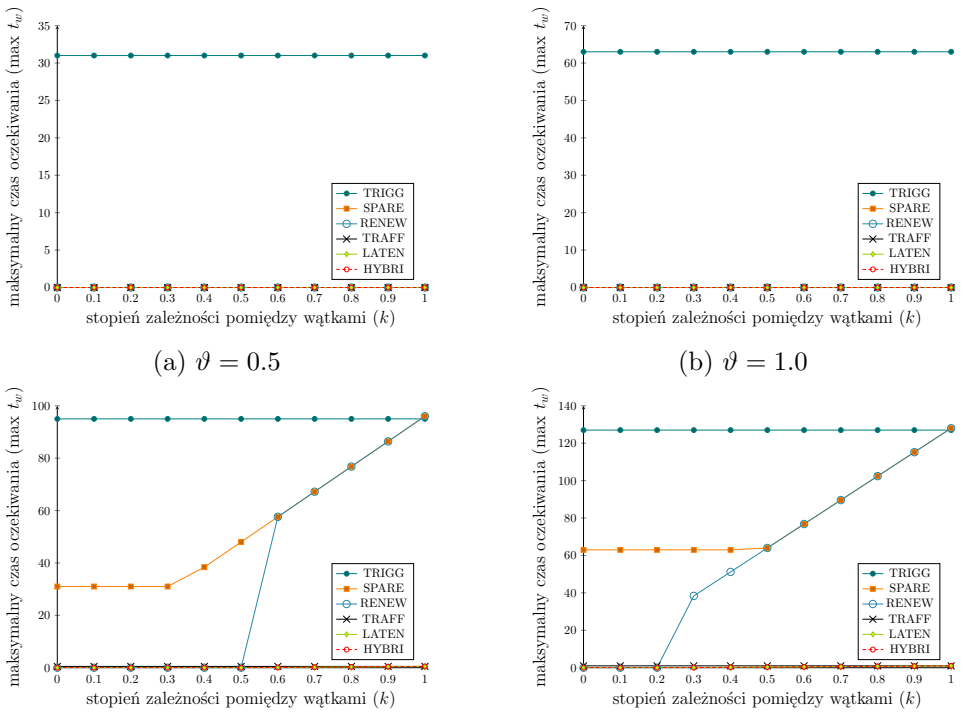
(c) $\vartheta = 1.5$ (d) $\vartheta = 2.0$

Rysunek 9.21: Porównanie algorytmów wypełniania stron pod względem średniego czasu oczekiwania (metryka $\bar{t}_w$) – wersja wielowątkowa

algorytmów: SPARE, RENEW, LATEN i HYBRI można określić graniczną wartość parametru k, dla której czas średni oczekiwania zaczyna przyjmować wartości większe od zera i liniowo rosnąć wraz ze wzrostem wartości parametru k. Granica ta jest tym mniejsza, im większe staje się obciążenie systemu. Dwa nowe algorytmy: LATEN i HYBRI radzą sobie w tym zestawieniu najlepiej.

Wyniki ostatniego badania, w którym zmierzono maksymalną zarejestrowaną wartość opóźnienia doświadczanego przez konsumenta, przedstawiono na rysunku 9.22. Dla niewielkich obciążeń systemu ($\vartheta \leqslant 1.0$, rysunki 9.22a i 9.22b), wszystkie algorytmy z wyjątkiem TRIGG nie wykazują żadnego czasu oczekiwania. Dla większych wartości obciążenia ($\vartheta > 1.0$, rysunki 9.22c i 9.22d), pozostałe stare algorytmy (SPARE i RENEW) zaczynają generować wysokie chwilowe wartości czasu oczekiwania, spowodowane okresowymi przestojami w płynności dostarczania danych. Algorytm RENEW dłużej jest w stanie utrzymać brak opóźnień, ze względu na większe wykorzystanie kolejki, niż algorytm SPARE.

Nowe algorytmy wypełniania stron nie wykazują występowania wysokich skoków czasu oczekiwania, bez względu na zastosowane obciążenie systemu, jak i na wartość parametru k. Spowodowane jest to faktem respektowania



(a) $\vartheta = 0.5$ (b) $\vartheta = 1.0$

(c) $\vartheta = 1.5$ (d) $\vartheta = 2.0$

Rysunek 9.22: Porównanie algorytmów wypełniania stron pod względem maksymalnego czasu oczekiwania (metryka max t_w) – wersja wielowątkowa

przez nie ograniczenia na minimalny czas konsumpcji agregatu i wynikającym z niego wprowadzaniem sztucznego opóźnienia w dostarczaniu danych konsumentowi, co zapobiega skokowym jego zmianom.

9.8 Wnioski i perspektywy rozwoju

Wszystkie trzy adaptacyjne algorytmy wypełniania stron charakteryzują się zdolnością adaptacji do bieżących warunków pracy poprzez zmianę swoich parametrów: rozmiaru zadawanych zapytań oraz momentu ich zadawania. Efekt ten był widoczny zarówno w przypadku badań przeprowadzonych w stałym środowisku, ale dla różnych wartości obciążenia systemu, jak i dla scenariuszy zakładających skokową zmianę tego obciążenia. Stopień reakcji na zmianę obciążenia był uzależniony od zadanych ograniczeń algorytmów.

Ponadto, adaptacyjne cechują się zdolnością do przewidywania optymalnego rozmiaru zapytania i momentu jego zadania w zależności od aktualnego obciążenia systemu oraz ograniczeń, wykazując się tym samym elastycznością w działaniu. Algorytmy TRAFF i HYBRI zachowywały się w sposób ściśle określony pod względem nieprzekraczania nałożonego ograniczenia

producenta, podczas gdy algorytm LATEN starał się dążyć do zadanego ograniczenia konsumenta, o ile było to możliwe.

Uzyskane wyniki badań porównawczych potwierdziły skuteczność adaptacyjnych algorytmów wypełniania stron oraz ich przewagę nad starymi, stałymi. Ze względu na konieczność niestosowania ograniczeń w celu zachowania kompatybilności wstecznej z poprzednim algorytmami, algorytm TRAFF uzyskał znacznie gorsze wyniki, niż LATEN i HYBRI w przypadku niektórych badań. Niemniej należy zauważyć, iż algorytm ten dowiódł swojej skuteczności w badaniach weryfikacyjnych. Przeznaczeniem algorytmu TRAFF jest przede wszystkim dbanie o nieprzekraczanie nałożonego ograniczenia na maksymalne obciążenie producenta.

Podsumowując całość badań porównawczych, nowa klasa algorytmów wypełniania stron potwierdziła swoją skuteczność w różnych warunkach pracy, przewyższając poprzednią pod wieloma względami. Co więcej, adaptacyjne algorytmy dowiodły, że spełniają stawiane im wymagania w zakresie: adaptacyjności (praca w całym zakresie obciążenia systemu), elastyczności (uzyskując lepsze wyniki niż asynchroniczne zgłaszania żądań przez stare algorytmy) oraz wykazaną w badaniach weryfikacyjnych określoność (przestrzeganie ograniczeń). W konsekwencji można uznać, iż prawdziwość tezy 3 została wykazana.

Dla każdego z trzech adaptacyjnych algorytmów wypełniania stron, możliwe jest wskazanie potencjalnego zastosowania. W odróżnieniu od poprzednich, stałych algorytmów, których wybór był podyktowany uwarunkowaniami czasowymi poszczególnych komponentów systemu, wybór konkretnego algorytmu adaptacyjnego wynika jedynie z potrzeb użytkownika, gdyż algorytmy te zdolne są do pracy w całym spektrum obciążenia systemu. Algorytm TRAFF, ze względu na rygorystyczne przestrzeganie ograniczenia producenta, doskonale nadaje się do zastosowania w różnego typu schedulerach pracujących na poziomie globalnym, po stronie producenta. Powoduje to możliwość przydzielania różnym instancjom silnika CUBIT różnych priorytetów wyrażonych maksymalną częstotliwością zadawanych zapytań.

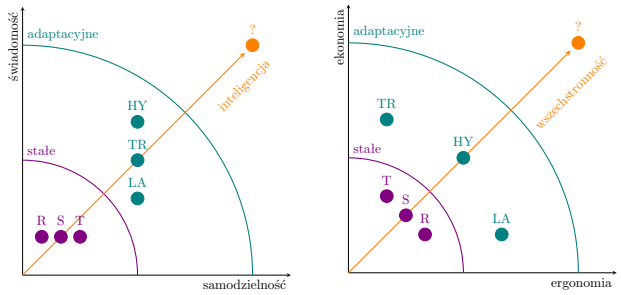
Dla odmiany, algorytm LATEN swoje zastosowanie może znaleźć przy różnego rodzaju schedulerach lokalnych, działających po stronie klienta. W zależności od zadanego ograniczenia konsumenta możliwe jest manipulowanie stopniem przychylności wobec klienta algorytmu wyrażającym się poprzez utrzymywanie zawsze pewnego zapasu rekordów w kolejce (dla źródeł historycznych) lub bezpośrednie przekazywanie dopiero co załadowanych rekordów wprost do klienta (w przypadku źródeł strumieniowych), a przez to nadawanie określonym instancjom silnika CUBIT różnych priorytetów. W przypadku źródeł historycznych ograniczenie konsumenta równe zero reprezentuje najniższy priorytet, a wartości ujemne – odpowiednio wyższe priorytety. Nieco inaczej sytuacja wygląda w przypadku źródła strumieniowego: wartość ograniczenia równa zero oznacza najwyższy priorytet, podczas gdy wartości dodanie – odpowiednio niższe priorytety.

Zastosowaniem algorytmu HYBRI może być dwupoziomowy scheduler lub scheduler lokalny z globalnym ograniczeniem na maksymalne obciążenie producenta. Algorytm umożliwia, podobnie jak LATEN, nadawanie lokalnych priorytetów poszczególnym instancjom silnika CUBIT, przy jednoczesnym zachowaniu ograniczenia producenta. Ponadto, algorytm ten stara się w miarę możliwości minimalizować obciążenie producenta, podobnie jak algorytm LATEN, a inaczej niż algorytm TRAFF, który wybiera zawsze największe akceptowalne obciążenia producenta. Jednocześnie algorytm HYBRI przestrzega ograniczenia producenta – podobnie jak TRAFF, a inaczej niż LATEN.

Pomimo, iż adaptacyjne algorytmy wypełniania stron mogą zostać wykorzystane w scenariuszu zakładającym współpracę wielu instancji silnika CUBIT działających jednocześnie, poszczególne algorytmy nie są świadome tego faktu. Każdy układ stronicowania wraz ze skojarzonym z nim algorytmem ilościowym pracuje w sposób indywidualny, analizując swój lokalny zestaw parametrów. W konsekwencji, wymagane jest istnienie dodatkowego, zewnętrznego algorytmu zarządzającego pracą wszystkich układów stronicowania poszczególnych instancji silnika CUBIT, którego zadaniem jest przydzielanie indywidualnych priorytetów poszczególnym instancjom, poprzez dostosowywanie wartości ograniczeń algorytmów. Opisana tutaj koncepcja nie jest jednak wolna od wad – niski poziom świadomości algorytmów adaptacyjnych czyni niemożliwym globalną optymalizację transferu danych na poziomie całego kontekstowego serwera OLAP, a więc wszystkich instancji silnika CUBIT.

Rzeczony problem może stać się motywacją do wskazania nowej generacji algorytmów wypełniania stron, które można określić mianem *synergicznych*. Algorytmy takie cechować się powinny szeroką świadomością i, w konsekwencji, wyższym poziomem samodzielności. Owa samodzielność wynika wprost z braku dodatkowego mechanizmu zarządzającego. Co więcej, algorytmy synergiczne charakteryzować się lepszą wszechstronnością, rozumianą jako zapewnienie odpowiedniego poziomu jakości usług jednocześnie dla konsumenta (ergonomiczność), jak i producenta (ekonomia). Adaptacyjne algorytmy wypełniania stron, poprzez wykorzystanie metody epsilon-ograniczeń, skupiały się na optymalizacji jednej z tych dwóch metryk – co prawda, algorytm HYBRI starał się uwzględnić obie metryki, lecz bez globalnego spojrzenia nie był w stanie dokonać znaczącego postępu w tej dziedzinie.

Rysunek 9.23 przedstawia podsumowanie przyszłych kierunków rozwoju algorytmów wypełniania stron. Składa się on z dwóch części: pierwsza (rysunek 9.23a) przedstawia zestawienia dotychczasowych algorytmów w układzie osi samodzielność-świadomość, podczas gdy druga część (rysunek 9.23b) – w układzie ergonomia-ekonomia. Jednoczesne polepszanie samodzielności i świadomości przyczynia się do zwiększenia poziomu inteligencji, a zwiększenie ergonomii i ekonomii powoduje przyrost wszechstronności. Zarówno inteligencja, jak i wszechstronność są zarazem dwoma głównymi filarami



(a) samodzielność kontra świadomość

(b) ergonomia kontra ekonomia

Rysunek 9.23: Potencjalne drogi rozwoju algorytmów wypełniania stron

nowej proponowanej klasy algorytmów wypełniania stron – algorytmów synergicznych.

Rozdział 10

Konkluzje

10.1 Podsumowanie rozprawy

Celem niniejszej rozprawy doktorskiej była analiza zagadnień teoretycznych oraz praktycznych związanych z modelem strumieniowej hurtowni danych zorientowanej na przetwarzanie wielkich zbiorów danych kontekstowych. W toku realizacji tego celu dokonano weryfikacji tez postawionych w rozdziale 1. Tezy owe odnosiły się do różnych aspektów przetwarzania i składowania danych kontekstowych. W szczególności dotyczyły trzech zagadnień: uznania istotności danych kontekstowych w analizie ukierunkowanej na wykrywanie zjawisk niepożądanych; weryfikacji możliwości stworzenia kompletnego systemu przeznaczonego do składowania i przetwarzania danych kontekstowych; potwierdzenia zasadności stosowania wybranych technik optymalizacyjnych w procesach transmisji danych, jako metod zwiększania jakości usług.

Weryfikację tez rozprawy przeprowadzono w sposób teoretyczny oraz empiryczny. Ten pierwszy polegał na budowie modeli i analizie powiązanych zagadnień teoretycznych. Sposób empiryczny opierał się na wynikach eksperymentów oraz wnioskach sformułowanych przy pracy z rzeczywistymi obiektami przemysłowymi. Punktem wyjścia do obu podejść był przykład motywujący, na który składał się system dystrybucji i składowania paliw płynnych. Głównym problemem badawczym, a zarazem bezpośrednią motywacją do podjęcia tematyki rozprawy, było wykrywanie wycieków z podziemnych zbiorników paliw.

Pierwsza teza rozprawy (teza 1) zakładała, że uzyskanie w pełni jednoznacznych wyników detekcji anomalii jest możliwe dopiero po uwzględnieniu kontekstu występowania poszczególnych zjawisk. Kontekst ten opisany powinien być przez dane współistniejące w czasie i przestrzeni oraz powiązane semantycznie z analizowanym zjawiskiem. W tym zakresie, teza ta odnosiła się do teoretycznych aspektów danych kontekstowych, podkreślając ich istotność przy analizie danych zasadniczych.

Weryfikacja tezy 1 została rozpoczęta od dokładnego scharakteryzowania sieci stacji paliw i zachodzących w niej procesów, co zostało opisane w rozdziale 3. Dokładne poznanie charakterystyki zjawisk zachodzących na stacjach paliw pozwoliło na identyfikację podstawowych anomalii rzeczywistych (zjawisk niepożądanych) oraz na dokonanie ich klasyfikacji pod kątem lokalności i istotności, co zostało opisane w rozdziale 4. W tym samym rozdziale przedstawiono również opracowaną metodę wykrywania wycieków paliwa (algorytm TUBE) oraz uzyskane przez nią wyniki. Dyskusja na temat jakości rzeczonych wyników, a także wnioski wynikłe z pracy z rzeczywistymi zbiorami danych, pozwoliły na sformułowanie podstaw analizy kontekstowej, udowadniając zarazem tezę 1.

Druga teza rozprawy (teza 2) zakładała, że możliwe jest zaprojektowanie strumieniowej hurtowni danych zorientowanej na przetwarzanie wielkich zbiorów danych kontekstowych. Hurtownia ta powinna wykorzystywać wielotorowy model przetwarzania danych, w którym analiza danych krytycznych byłaby wsparta przez przeprowadzaną niezależnie wieloaspektową analizę danych kontekstowych. W ten sposób sformułowana, teza ta odnosiła się do praktycznego wykorzystania danych kontekstowych. Zasadność postawienia tej tezy wynikała bezpośrednio ze sformułowania i potwierdzenia tezy 1.

Weryfikacja tezy 2 została rozpoczęta od przybliżenia klasy zaawansowanych hurtowni danych, ze szczególnym uwzględnieniem strumieniowych hurtowni danych, co poczyniono w rozdziale 2. Kolejnym etapem prac badawczych było sformułowanie podstaw teoretycznych danych kontekstowych, włączają w to ich model, klasyfikację oraz metody przetwarzania – zostało to opisane w rozdziale 5. Na bazie tych rozważań, zaprojektowano oraz w szczegółach opisano w rozdziale 6 model strumieniowej hurtowni danych kontekstowych (CtxDW). Hurtownia ta jest rozwinięciem koncepcji strumieniowej hurtowni danych poprzez jej adaptację do wielotorowego przetwarzania danych kontekstowych. Ostatnią składową weryfikacji omawianej tezy było opracowanie modelu silnika CUBIT oraz koncepcji wielowymiarowego bitowego indeksu zakresowego BRI, jako kluczowych elementów strumieniowego serwera OLAP – zagadnienia te opisano w rozdziale 7. Pierwsze rozwiązanie stanowi strumieniową adaptację kostki OLAP i zostało opracowane jako rozwinięcie silnika Materializowanej Listy Agregatów (MAL). Drugie rozwiązanie jest propozycją wysuniętą w stronę efektywnego obliczania wielowymiarowych agregatów na danych ciągłych, wykorzystując do tego równoległość na poziomie bitów i danych. Całość przedstawionych w wymienionych rozdziałach metod i narzędzi składa się zarazem na dowód poprawności tezy 2.

Ostatnia, trzecia teza rozprawy (teza 3) zakładała, że zastosowanie metod optymalizacji wielokryterialnej oraz uwzględnienie bieżących parametrów pracy i ograniczeń wpłynie pozytywnie na jakość usług. Jakość ta powinna być rozumiana nie tylko jako efektywność dostarczania danych użyt-

kownikowi, ale również jako płynność pracy źródła danych. W tym zakresie, teza ta odnosiła się do funkcjonowania strumieniowej hurtowni danych kontekstowych – jej postawienie implikowane było sformułowaniem i weryfikacją tezy 2.

Weryfikacja tezy 3 została rozpoczęta już od omówionego uprzednio silnika CUBIT, a dokładniej od będącego integralna jego częścią algorytmu wypełniania stron. Algorytm ten rozwiązuje problem optymalizacyjny o dwóch przeciwstawnych celach: minimalizacji opóźnienia w dostarczaniu wyników zapytań użytkownikom oraz minimalizacji obciążenia bazy danych. W rozdziale 8 dokonano analizy postawionego w ten sposób problemu optymalizacyjnego oraz sformułowano ogólny zarys algorytmu oraz jego trzy warianty: TRAFF, LATEN i HYBRI, zwane dalej algorytmami wypełniania stron. Algorytmy te zostały zaprojektowane w celu zastąpienia poprzedniej generacji algorytmów wypełniania stron (stosowanych w silniku MAL). Wszystkie omówione algorytmy zostały zaimplementowane w postaci symulatora zdarzeń dyskretnych i poddane badaniom eksperymentalnym, których wyniki podano w rozdziale 9. Na potrzeby tychże zaproponowane dwie nowe metryki jakości usług: konsumenta i producenta. Uzyskane wyniki potwierdziły słuszność postawionych założeń i dowiodły prawdziwości tezy 3.

Większość treści przedstawionych w niniejszej rozprawie doktorskiej została opublikowana w formie artykułów naukowych i patentów. Wkład rzeczonej rozprawy w rozwój nauki może zostać przybliżony przez sformułowanie ośmiu kontrybucji, pośród których wyodrębnić można kontrybucje główne oraz poboczne:

1. **Model sieci stacji paliw**, jako złożonego źródła danych krytycznych i kontekstowych oraz klasyfikacja anomalii paliwowych (kontrybucja poboczna) – opisany w rozdziałach 3 i 4 model sieci stacji oraz klasyfikacja anomalii stanowią podstawę do analizy procesów zachodzących na stacjach paliw, a dodatkowo stanowią motywację do rozwoju metod wykrywania anomalii paliwowych – w wyniku prac badawczych powstały dwa artykuły naukowe [38, 28]$^{\mathrm{W}}$;
2. **Algorytm TUBE** jako metoda analizy danych bieżących, ukierunkowana na wykrywanie anomalii krytycznych (kontrybucja poboczna) – opisany w rozdziale 4 algorytm TUBE został opublikowany w czasopiśmie naukowym z listy filadelfijskiej [39]$^{\mathrm{W}}$, a jego wyniki posłużyły za motywację do podjęcia badań nad analizą danych kontekstowych – w wyniku prac badawczych powstały trzy patenty [70, 40, 71]$^{\mathrm{W}}$;
3. **Teoretyczny model danych kontekstowych**, uwzględniający ich klasyfikację oraz hierarchię poziomów przetwarzania (kontrybucja główna) – opisany w rozdziale 5 model wprowadza pojęcie danych kontekstowych oraz krytycznych, podział tych pierwszych na dane kontekstowe historyczne, przestrzenne i środowiskowe, a następnie definiuje siedem poziomów kontekstowości tworzących hierarchię – model powstał

w oparciu o wyniki wcześniejszych prac badawczych, przedstawionych w artykułach naukowych [66, 36, 37]^{W};

4. **Model strumieniowej hurtowni danych kontekstowych**, będącej rozwinięciem koncepcji strumieniowej hurtowni danych poprzez jej ukierunkowanie na przechowywanie i przetwarzanie danych kontekstowych (kontrybucja główna) – opisany w rozdziale 6 model wykorzystuje wielotorową architekturę, w której każdy tor odpowiada za przetwarzanie innego typu danych kontekstowych, wliczając w to również tor krytyczny, dedykowany danym bieżącym – model powstał w oparciu o wyniki wcześniejszych prac badawczych [66, 36, 37]^{W};
5. **Model silnika CUBIT**, stanowiącego adaptację koncepcji kostki danych OLAP na potrzeby wielowymiarowego przetwarzania danych strumieniowych (kontrybucja główna) – opisany w rozdziale 7 model stanowi rozwinięcie silnika Materializowanej Listy Agregatów (MAL) poprzez uwzględnienie wielowymiarowości danych, przez co staje się strumieniową adaptacją kostki OLAP – model powstał jako bezpośrednia kontynuacja i rozwinięcie wcześniejszych prac badawczych [67], a wyniki badań opisane zostały w artykule naukowym (obecnie w recenzji) [68]^{W};
6. **Model indeksu BRI**, czyli wielowymiarowego bitowego indeksu zakresowego (kontrybucja główna) – opisany w rozdziale 7 indeks BRI przeznaczony jest do przyspieszania zapytań obliczających wielowymiarowe agregaty zakresowe zdefiniowane w ciągłej przestrzeni cech, wykorzystując do tego celu kodowanie binarne oraz operacje na wektorach i macierzach – wyniki prac zostały opublikowane w formie dwóch patentów [41, 44]^{W};
7. **Algorytm stronicowania pamięci** dla strumieniowych hurtowni danych, wykorzystujący podejście adaptacyjne, bazujące na optymalizacji wielokryterialnej (kontrybucja główna) – opisany w rozdziale 8 algorytm (w trzech wariantach) został przedstawiony jako główny składnik silnika CUBIT, odpowiedzialny za efektywny transfer danych pomiędzy bazą danych a odbiorcą; celem algorytmu jest minimalizacja dwóch przeciwstawnych metryk: obciążenia bazy danych oraz czasu oczekiwania na nowe dane – algorytm wraz z wynikami badań weryfikacyjnych i eksperymentalnych został opisany w artykule naukowym (obecnie w recenzji) [68]^{W};
8. **Metryki jakości usług** dla strumieniowych hurtowni danych (kontrybucja główna) – opisane w rozdziale 9 metryki jakości usług opisują w formie wartości procentowych obciążenie bazy danych (połączenie granulacji zapytań i czasu pomiędzy żądaniami) oraz czas oczekiwania na nowe dane przez użytkownika (połączenie opóźnienia z płynnością w dostawie) – metryki zostały użyte podczas badań eksperymental-

nych na algorytmach stronicowania pamięci i opisane w artykule naukowym (obecnie w recenzji) [68]^{W}.

10.2 Przyszłe kierunki rozwoju

Tematyka niniejszej rozprawy doktorskiej wpisuje się w szerszą dziedzinę, będącą przedmiotem badań zespołu Teorii Przestrzeni Danych i Algorytmów (TPDiA)[1]. Prowadzone w przeszłości i obecnie prace badawcze związane są z różnymi aspektami zaawansowanych baz i hurtowni danych. Wśród nich wyróżnić można: efektywne metody ekstrakcji, magazynowania i przetwarzania wielkich zbiorów danych w czasie rzeczywistym; algorytmy wykrywania anomalii krytycznych w szeregach czasowych i strumieniach danych; modelowanie systemów wielkiej skali, integrujących wiele różnorodnych metod i narzędzi przeznaczonych do wieloaspektowej analizy danych.

W rozprawie nakreślono model architektury i opisano podstawowe założenia dla nowej klasy zaawansowanych hurtowni danych: strumieniowej hurtowni danych kontekstowych. Otwiera to wiele potencjalnych dróg rozwoju oraz zastosowań praktycznych. W konsekwencji zasadne jest wskazanie przyszłych planów badawczych, uwzględniając zarówno rozwój rozwiązań przedstawionych w rozprawie, jak i poszukiwanie nowych, wpisujących się w tematykę badawczą realizowaną w zespole TPDiA.

Pierwszym kierunkiem rozwoju, który wynika wprost z proponowanego modelu strumieniowej hurtowni danych kontekstowych, jest kontynuacja badań nad strumieniową adaptacją kostki danych (silnik CUBIT). W szczególności, zagadnieniem wartym głębszej analizy jest mechanizm agregacji w wielowymiarowych oknach zdefiniowanych na wartościach ciągłych. W klasycznych hurtowniach danych agregaty budowane są dla wartości kategorycznych, po których następuje grupowanie. W przypadku danych pomiarowych często istnieje konieczność wydzielenia grup *ad hoc*, na podstawie zakresów wartości atrybutów ciągłych. Wskazany tutaj kierunek rozwoju implikuje dwa kolejne, które związane są z efektywnym wyznaczaniem oraz dostarczaniem agregatów wielowymiarowych.

Drugi istotny kierunek rozwoju wiąże się z rozbudową koncepcji indeksu BRI (wielowymiarowego bitowego indeksu zakresowego). W rozprawie nakreślono jedynie jego teoretyczne podstawy, które dodatkowo zostały zastrzeżone w postaci patentów. Niemniej, w tym zakresie istnieje wciąż wiele obszarów, które wymagają przeprowadzenia dogłębnych badań. Przykładem może być ogólna struktura indeksu – w rozprawie podano jedynie charakterystykę głównych algorytmów i założeń. Kwestią otwartą pozostaje również implementacja w środowisku równoległym (procesor graficzny) i związane z nią badania wydajnościowe.

[1] Zespół TPDiA jest podstrukturą w ramach Katedry Informatyki Stosowanej na Wydziale Automatyki, Elektroniki i Informatyki w Politechnice Śląskiej w Gliwicach.

Trzecim kierunkiem rozwoju, również powiązanym z silnikiem CUBIT, jest kontynuacja badań nad algorytmami stronicowania pamięci oraz samym mechanizmem stronicowania. Jego zadaniem jest zapewnienie efektywnego dostarczania wyników zapytań w formie kostki danych. W rozdziale 9 zdefiniowano ograniczenia zaproponowanych algorytmów i wskazano na konieczność spojrzenia na całość pracujących jednocześnie instancji silnika CUBIT. Powyższe zagadnienia stanowią zarazem nowy problem optymalizacyjny, mający pewne cechy wspólne z teorią systemów operacyjnych.

Czwarty kierunek rozwoju związany jest z zastosowaniem koncepcji strumieniowej hurtowni danych kontekstowych do innego problemu rzeczywistego, niż omówiony w rozprawie problem wykrywania wycieków paliwa. Potencjalnym zastosowaniem takiej hurtowni danych może być system notowań giełdowych, który cechuje się dużą dynamiką zmian oraz silną zależnością od kontekstu. Innym zastosowaniem nasuwającym się na myśl jest system monitorowania zmian klimatu, analizujący dane meteorologiczne. W tym przypadku również występuje duża zależność badanych zjawisk i anomalii od kontekstu.

Piątym kierunkiem rozwoju jest integracja zaproponowanego modelu strumieniowej hurtowni danych kontekstowych z modelem Spichlerza Agregatów przedstawionego w monografii *Zaawansowane Hurtownie Danych* [29]. Koncepcja Spichlerza Agregatów polega na integracji wielu heterogenicznych baz i hurtowni danych w jeden spójny system. Poprzez dopasowanie odpowiednich metod analizy do poszczególnych typów danych, Spichlerz Agregatów integruje dane z wielu różnorodnych hurtowni danych, podobnie jak hurtownia danych integruje dane z wielu różnorodnych baz danych.

Podsumowując, nakreślone w rozprawie metody i modele stanowią istotny wkład w dziedzinę zaawansowanych hurtowni danych, otwierając przy tym szeroki wachlarz perspektyw i możliwości rozwoju. Zaproponowany model danych kontekstowych, zakładający uwiarygadnianie wyników obliczeń przez analizę powiązanych danych, może znaleźć zastosowanie w wielu dziedzinach życia, w których istotne jest dostarczanie odpowiedzi nie tylko w możliwie krótkim czasie, ale i z potwierdzoną wiarygodnością.

Bibliografia

[1] Umut A. Acar, Guy E. Blelloch, and Robert Harper. Selective memoization. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '03, page 14–25, New York, NY, USA, 2003. Association for Computing Machinery.

[2] B. Adelberg, H. Garcia-Molina, and B. Kao. Applying update streams in a soft real-time database system. *SIGMOD Rec.*, 24(2):245–256, May 1995.

[3] N.D. Adesh and A. Renuka. Avoiding queue overflow and reducing queuing delay at enodeb in lte networks using congestion feedback mechanism. *Computer Communications*, 146:131 – 143, 2019.

[4] Divyakant Agrawal. The reality of real-time business intelligence. In Malu Castellanos, Umesh Dayal, and Timos Sellis, editors, *Business Intelligence for the Real-Time Enterprise*, pages 75–88, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] Ali A. Amer, Adel A. Sewisy, and Taha M.A. Elgendy. An optimized approach for simultaneous horizontal data fragmentation and allocation in distributed database systems (ddbss). *Heliyon*, 3(12):e00487, 2017.

[6] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. A linear method for deviation detection in large databases. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 164–169. AAAI Press, 1996.

[7] Arian Bär and Lukasz Golab. Towards benchmarking stream data warehouses. In *Proceedings of the Fifteenth International Workshop on Data Warehousing and OLAP*, DOLAP '12, page 105–112, New York, NY, USA, 2012. Association for Computing Machinery.

[8] Mohammad Hossein Bateni, Lukasz Golab, Mohammad Taghi Hajiaghayi, and Howard Karloff. Scheduling to minimize staleness and

stretch in real-time data warehouses. In *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '09, page 29–38, New York, NY, USA, 2009. Association for Computing Machinery.

[9] Loïc Besnard, Pedro Pinto, Imane Lasri, João Bispo, Erven Rohou, and João M.P. Cardoso. A framework for automatic and parameterizable memoization. *SoftwareX*, 10:100322, 2019.

[10] Fábio Bezerra and Jacques Wainer. Algorithms for anomaly detection of traces in logs of process aware information systems. *Information Systems*, 38(1):33 – 44, 2013.

[11] Miao Cai, Diming Zhang, and Hao Huang. A scalable virtual memory system based on decentralization for many-cores. *Journal of Systems Architecture*, 107:101803, 2020.

[12] Saihua Cai, Jinfu Chen, Haibo Chen, Chi Zhang, Qian Li, Rexford Nii Ayitey Sosu, and Shang Yin. An efficient anomaly detection method for uncertain data based on minimal rare patterns with the consideration of anti-monotonic constraints. *Information Sciences*, 580:620–642, 2021.

[13] Emanuele Carlini, Alessandro Lulli, and Laura Ricci. dragon: Multidimensional range queries on distributed aggregation trees. *Future Generation Computer Systems*, 55:101 – 115, 2016.

[14] Joe Celko. 30 - advanced grouping, windowed aggregation, and olap in sql. In Joe Celko, editor, *Joe Celko's SQL for Smarties (Fourth Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 539 – 554. Morgan Kaufmann, Boston, fourth edition edition, 2011.

[15] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[16] Chih-Hung Chang, Fuu-Cheng Jiang, Chao-Tung Yang, and Sheng-Cang Chou. On construction of a big data warehouse accessing platform for campus power usages. *Journal of Parallel and Distributed Computing*, 133:40 – 50, 2019.

[17] Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. An overview of business intelligence technology. *Commun. ACM*, 54(8):88–98, August 2011.

[18] Henrique V Da Silva, Celso K Morooka, Ivan R Guilherme, Tiago C da Fonseca, and José RP Mendes. Leak detection in petroleum pipe-

lines using a fuzzy system. *Journal of Petroleum Science and Engineering*, 49(3):223–238, 2005.

[19] Sungkwang Eom, Xiongnan Jin, and Kyong-Ho Lee. Efficient generation of spatiotemporal relationships from spatial data streams and static data. *Information Processing & Management*, 57(3):102205, 2020.

[20] European Norm 13160-1. Leak Detection Systems - Part 1: General Principles, 2003.

[21] European Norm 13160-5. Leak Detection Systems - Part 5: Tank Gauge Leak Detection Systems, 2005.

[22] Pawel Foszner, Aleksandra Gruca, and Jakub Bularz. Fuel pipeline thermal conductivity in automatic wet stock reconciliation systems. In Petra Perner, editor, *Advances in Data Mining. Applications and Theoretical Aspects - 16th Industrial Conference, ICDM 2016, New York, NY, USA, July 13-17, 2016. Proceedings*, volume 9728 of *Lecture Notes in Computer Science*, pages 297–310. Springer, 2016.

[23] L. Golab, T. Johnson, and V. Shkapenyuk. Scheduling updates in a real-time stream warehouse. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1207–1210, March 2009.

[24] L. Golab, T. Johnson, and V. Shkapenyuk. Scalable scheduling of updates in streaming data warehouses. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):1092–1105, June 2012.

[25] L. Golab, T. Johnson, and V. Shkapenyuk. Scalable scheduling of updates in streaming data warehouses. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):1092–1105, June 2012.

[26] Lukasz Golab and Theodore Johnson. Consistency in a stream warehouse. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 114–122. www.cidrdb.org, 2011.

[27] Lukasz Golab, Theodore Johnson, J. Spencer Seidel, and Vladislav Shkapenyuk. Stream warehousing with datadepot. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, page 847–854, New York, NY, USA, 2009. Association for Computing Machinery.

[28] Anna Gorawska and Krzysztof Pasterak. Anomaly Detection in Data Streams: The Petrol Station Simulator. In *Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery - 12th International Conference, BDAS 2016,*

Ustroń, Poland, May 31 - June 3, 2016, Proceedings, volume 613 of *Communications in Computer and Information Science*, pages 727–736. Springer, 2016.

[29] M. Gorawski. *Zaawansowane hurtownie danych [Advanced Data Warehouses]*. Studia Informatica – Politechnika Slaska. Wydawnictwo Naukowe Politechniki Slaskiej, 2009.

[30] Marcin Gorawski. Extended cascaded star schema for distributed spatial data warehouse. In Roman Wyrzykowski, Jack J. Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics, 8th International Conference, PPAM 2009, Wroclaw, Poland, September 13-16, 2009. Revised Selected Papers, Part I*, volume 6067 of *Lecture Notes in Computer Science*, pages 166–175. Springer, 2009.

[31] Marcin Gorawski. Multiversion spatio-temporal telemetric data warehouse. In Janis Grundspenkis, Marite Kirikova, Yannis Manolopoulos, and Leonids Novickis, editors, *Advances in Databases and Information Systems, Associated Workshops and Doctoral Consortium of the 13th East European Conference, ADBIS 2009, Riga, Latvia, September 7-10, 2009. Revised Selected Papers*, volume 5968 of *Lecture Notes in Computer Science*, pages 63–70. Springer, 2009.

[32] Marcin Gorawski. Time complexity of page filling algorithms in materialized aggregate list (MAL) and MAL/TRIGG materialization cost. *Control. Cybern.*, 38(1):153–172, 2009.

[33] Marcin Gorawski, Slawomir Bankowski, and Michal Gorawski. Selection of structures with grid optimization, in multiagent data warehouse. In Colin Fyfe, Peter Tiňo, Darryl Charles, César Ignacio García-Osorio, and Hujun Yin, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2010, 11th International Conference, Paisley, UK, September 1-3, 2010. Proceedings*, volume 6283 of *Lecture Notes in Computer Science*, pages 292–299. Springer, 2010.

[34] Marcin Gorawski and Jakub Bularz. Distribution-based methods of preserving data privacy in distributed spatial data warehouse. *Int. J. Bus. Intell. Data Min.*, 2(4):383–400, 2007.

[35] Marcin Gorawski and Aleksander Chrószcz. Optimization of operator partitions in stream data warehouse. In Il-Yeol Song, Alfredo Cuzzocrea, and Karen C. Davis, editors, *DOLAP 2011, ACM 14th International Workshop on Data Warehousing and OLAP, Glasgow, United Kingdom, October 28, 2011, Proceedings*, pages 61–66. ACM, 2011.

[36] Marcin Gorawski, Anna Gorawska, and Krzysztof Pasterak. Evaluation and development perspectives of stream data processing systems. In Andrzej Kwiecien, Piotr Gaj, and Piotr Stera, editors, *Computer Networks, 20th International Conference, CN 2013, Lwówek Śląski, Poland, June 17-21, 2013. Proceedings*, volume 370 of *Communications in Computer and Information Science*, pages 300–311. Springer, 2013.

[37] Marcin Gorawski, Anna Gorawska, and Krzysztof Pasterak. A survey of data stream processing tools. In *Information Sciences and Systems 2014 - Proceedings of the 29th International Symposium on Computer and Information Sciences, ISCIS 2014, Krakow, Poland, October 27-28, 2014*, pages 295–303, 2014.

[38] Marcin Gorawski, Anna Gorawska, and Krzysztof Pasterak. Liquefied petroleum storage and distribution problems and research thesis. In *Beyond Databases, Architectures and Structures - 11th International Conference, BDAS 2015, Ustroń, Poland, May 26-29, 2015, Proceedings*, pages 540–550, 2015.

[39] Marcin Gorawski, Anna Gorawska, and Krzysztof Pasterak. The TUBE algorithm: Discovering trends in time series for the early detection of fuel leaks from underground storage tanks. *Expert Syst. Appl.*, 90:356–373, 2017.

[40] Marcin Gorawski, Anna Gorawska, and Krzysztof Pasterak. Sposób i system do inteligentnego wykrywania anomalii w torze przepływu paliwa na stacjach paliw, 2020-05-18. Urząd Patentowy Rzeczpospolitej Polskiej: Patent nr 235055.

[41] Marcin Gorawski, Anna Gorawska, Krzysztof Pasterak, and Michał Gorawski. Sposób ekstrakcji i transformacji strumieniowych danych pomiarowych wykorzystujący obliczenia równoległe, 2019-09-30. Urząd Patentowy Rzeczpospolitej Polskiej: Patent nr 233157.

[42] Marcin Gorawski and Michal Gorawski. Balanced spatio-temporal data warehouse with r-mvb, STCAT and BITMAP indexes. In *Fifth International Conference on Parallel Computing in Electrical Engineering (PARELEC 2006), 13-17 September 2006, Bialystok, Poland*, pages 43–48. IEEE Computer Society, 2006.

[43] Marcin Gorawski, Michal Gorawski, and Slawomir Bankowski. Selection of indexing structures in grid data warehouses with software agents. *Int. J. Comput. Sci. Appl.*, 4(1):39–52, 2007.

[44] Marcin Gorawski, Michał Gorawski, Anna Gorawska, and Krzysztof Pasterak. Sposób magazynowania i agregowania wielowymiarowych

strumieniowych danych pomiarowych z uwzględnieniem aspektu lokalności, 2019-05-31. Urząd Patentowy Rzeczpospolitej Polskiej: Patent nr 232115.

[45] Marcin Gorawski and Pawel Jureczek. Continuous pattern mining using the fcpgrowth algorithm in trajectory data warehouses. In Manuel Graña Romay, Emilio Corchado, and M. Teresa García-Sebastián, editors, *Hybrid Artificial Intelligence Systems, 5th International Conference, HAIS 2010, San Sebastián, Spain, June 23-25, 2010. Proceedings, Part I*, volume 6076 of *Lecture Notes in Computer Science*, pages 187–195. Springer, 2010.

[46] Marcin Gorawski and Pawel Jureczek. Regions of interest in trajectory data warehouse. In Ngoc Thanh Nguyen, Manh Thanh Le, and Jerzy Swiatek, editors, *Intelligent Information and Database Systems, Second International Conference, ACIIDS, Hue City, Vietnam, March 24-26, 2010. Proceedings, Part I*, volume 5990 of *Lecture Notes in Computer Science*, pages 74–81. Springer, 2010.

[47] Marcin Gorawski, Pawel Jureczek, and Michal Gorawski. Exploration of continuous sequential patterns using the cpgrowth algorithm. In Ngoc Thanh Nguyen, Aleksander Zgrzywa, and Andrzej Czyzewski, editors, *Advances in Multimedia and Network Information System Technologies, MISSI 2010*, volume 80 of *Advances in Intelligent and Soft Computing*, pages 165–172. Springer, 2010.

[48] Marcin Gorawski, Damian Lis, and Anna Gorawska. Zero-latency data warehouse system based on parallel processing and cache module. In Emilio Corchado, José Antonio Lozano, Héctor Quintián, and Hujun Yin, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2014 - 15th International Conference, Salamanca, Spain, September 10-12, 2014. Proceedings*, volume 8669 of *Lecture Notes in Computer Science*, pages 465–474. Springer, 2014.

[49] Marcin Gorawski, Damian Lis, and Anna Gorawska. Cuda-powered CTBE algorithm for zero-latency data warehouse. In Tadeusz Morzy, Patrick Valduriez, and Ladjel Bellatreche, editors, *New Trends in Databases and Information Systems - ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8-11, 2015. Proceedings*, volume 539 of *Communications in Computer and Information Science*, pages 358–367. Springer, 2015.

[50] Marcin Gorawski and Rafal Malczok. Aggregation and analysis of spatial data by means of materialized aggregation tree. In Tatyana M. Yakhno, editor, *Advances in Information Systems, Third International*

Conference, ADVIS 2004, Izmir, Turkey, October 20-22, 2004, Proceedings, volume 3261 of *Lecture Notes in Computer Science*, pages 24–33. Springer, 2004.

[51] Marcin Gorawski and Rafal Malczok. Distributed spatial data warehouse indexed with virtual memory aggregation tree. In Jörg Sander and Mario A. Nascimento, editors, *Spatio-Temporal Database Management, 2nd International Workshop STDBM'04, Toronto, Canada, August 30, 2004*, pages 25–32, 2004.

[52] Marcin Gorawski and Rafal Malczok. Multi-thread processing of long aggregates lists. In Roman Wyrzykowski, Jack J. Dongarra, Norbert Meyer, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics, 6th International Conference, PPAM 2005, Poznan, Poland, September 11-14, 2005, Revised Selected Papers*, volume 3911 of *Lecture Notes in Computer Science*, pages 59–66. Springer, 2005.

[53] Marcin Gorawski and Rafal Malczok. On efficient storing and processing of long aggregate lists. In A Min Tjoa and Juan Trujillo, editors, *Data Warehousing and Knowledge Discovery, 7th International Conference, DaWaK 2005, Copenhagen, Denmark, August 22-26, 2005, Proceedings*, volume 3589 of *Lecture Notes in Computer Science*, pages 190–199. Springer, 2005.

[54] Marcin Gorawski and Rafal Malczok. Towards storing and processing of long aggregates lists in spatial data warehouses. In *Conference Proceedings pp*, volume 95, page 103. Citeseer, 2005.

[55] Marcin Gorawski and Rafal Malczok. Updating aggregation tree in distributed spatial telemetric data warehouse. In *13th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP 2005), 6-11 February 2005, Lugano, Switzerland*, pages 329–336. IEEE Computer Society, 2005.

[56] Marcin Gorawski and Rafal Malczok. Comparison of two approaches to processing long aggregates lists in spatial data warehouses. *Annales UMCS, Informatica*, 4(1):149–160, 2006.

[57] Marcin Gorawski and Rafal Malczok. Materialized ar-tree in distributed spatial data warehouse. *Intell. Data Anal.*, 10(4):361–377, 2006.

[58] Marcin Gorawski and Rafal Malczok. Towards stream data parallel processing in spatial aggregating index. In Roman Wyrzykowski, Jack J. Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics, 7th International Conference, PPAM 2007, Gdansk, Poland, September 9-12, 2007, Revised*

Selected Papers, volume 4967 of *Lecture Notes in Computer Science*, pages 209–218. Springer, 2007.

[59] Marcin Gorawski and Rafal Malczok. Performing range aggregate queries in stream data warehouse. In Krzysztof A. Cyran, Stanislaw Kozielski, James F. Peters, Urszula Stanczyk, and Alicja Wakulicz-Deja, editors, *Man-Machine Interactions, Proceedings of the first International Conference on Man-Machine Interactions, ICMMI 2009, The Beskids, Poland, September 25-27, 2009*, volume 59 of *Advances in Intelligent and Soft Computing*, pages 615–622, 2009.

[60] Marcin Gorawski and Rafal Malczok. Answering range-aggregate queries over objects generating data streams. In Hiroyuki Kitagawa, Yoshiharu Ishikawa, Qing Li, and Chiemi Watanabe, editors, *Database Systems for Advanced Applications, 15th International Conference, DASFAA 2010, Tsukuba, Japan, April 1-4, 2010, Proceedings, Part II*, volume 5982 of *Lecture Notes in Computer Science*, pages 436–439. Springer, 2010.

[61] Marcin Gorawski and Rafal Malczok. CAM2S: an integrated indexing structure for spatial objects generating data streams. In Leonard Barolli, Fatos Xhafa, Salvatore Vitabile, and Hui-Huang Hsu, editors, *CISIS 2010, The Fourth International Conference on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, 15-18 February 2010*, pages 33–40. IEEE Computer Society, 2010.

[62] Marcin Gorawski and Rafal Malczok. Indexing spatial objects in stream data warehouse. In Ngoc Thanh Nguyen, Radoslaw P. Katarzyniak, and Shyi-Ming Chen, editors, *Advances in Intelligent Information and Database Systems*, volume 283 of *Studies in Computational Intelligence*, pages 53–65. Springer, 2010.

[63] Marcin Gorawski and Pawel Marks. High efficiency of hybrid resumption in distributed data warehouses. In *16th International Workshop on Database and Expert Systems Applications (DEXA 2005), 22-26 August 2005, Copenhagen, Denmark*, pages 323–327. IEEE Computer Society, 2005.

[64] Marcin Gorawski and Pawel Marks. Fault-tolerant distributed stream processing system. In *17th International Workshop on Database and Expert Systems Applications (DEXA 2006), 4-8 September 2006, Krakow, Poland*, pages 395–399. IEEE Computer Society, 2006.

[65] Marcin Gorawski and Szymon Panfil. A system of privacy preserving distributed spatial data warehouse using relation decomposition. In *Proceedings of the The Forth International Conference on Availability,*

Reliability and Security, ARES 2009, March 16-19, 2009, Fukuoka, Japan, pages 522–527. IEEE Computer Society, 2009.

[66] Marcin Gorawski and Krzysztof Pasterak. Schedulery strumieniowe w AGKPStream. *Studia Informatica*, 33(2A):197–210, 2012.

[67] Marcin Gorawski and Krzysztof Pasterak. Research and analysis of the stream materialized aggregate list. In *Computer Science and Its Applications - 5th IFIP TC 5 International Conference, CIIA 2015, Saida, Algeria, May 20-21, 2015, Proceedings*, pages 269–278, 2015.

[68] Marcin Gorawski, Krzysztof Pasterak, Anna Gorawska, and Michał Gorawski. The stream data warehouse: page replacement algorithms and quality of service metrics. *Future Generation Computer Systems (w recenzji)*, 2022.

[69] Marcin Gorawski, Zacheusz Siedlecki, and Anna Gorawska. Collaborative multiparty association rules mining with threshold homomorphic encryption. In *Algorithms and Architectures for Parallel Processing - ICA3PP International Workshops and Symposiums, Zhangjiajie, China, November 18-20, 2015, Proceedings*, pages 251–263, 2015.

[70] Marcin Gorawski, Mirosław Skrzewski, Anna Gorawska, and Krzysztof Pasterak. Układ do kalibracji stanu zbiornika paliw płynnych, 2021-04-19. Urząd Patentowy Rzeczpospolitej Polskiej: Patent nr 237471.

[71] Marcin Gorawski, Mirosław Skrzewski, Anna Gorawska, Krzysztof Pasterak, and Michał Gorawski. Sposób i system do wykrywania niezgodności w procesie dostaw paliw płynnych na stacjach paliw, 2020-06-01. Urząd Patentowy Rzeczpospolitej Polskiej: Patent nr 235154.

[72] Marcin Gorawski, Miroslaw Skrzewski, Michal Gorawski, and Anna Gorawska. Neural networks in petrol station objects calibration. In *Algorithms and Architectures for Parallel Processing - ICA3PP International Workshops and Symposiums, Zhangjiajie, China, November 18-20, 2015, Proceedings*, pages 714–723, 2015.

[73] Michal Gorawski and Krzysztof Grochla. Graph representation of linear infrastructure in smart city iot systems. In *11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT 2019, Dublin, Ireland, October 28-30, 2019*, pages 1–4. IEEE, 2019.

[74] Michal Gorawski and Krzysztof Grochla. Performance tests of smart city iot data repositories for universal linear infrastructure data and graph databases. *SN Comput. Sci.*, 1(1):31:1–31:7, 2020.

[75] Anjana Gosain, Sangeeta Sabharwal, and Rolly Gupta. Architecture based materialized view evolution: A review. *Procedia Computer Science*, 48:256 – 262, 2015. International Conference on Computer, Communication and Convergence (ICCC 2015).

[76] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, page 47–57, New York, NY, USA, 1984. Association for Computing Machinery.

[77] Sławomir Hanczewski, Maciej Stasiak, and Joanna Weissenberg. Queueing model of a multi-service system with elastic and adaptive traffic. *Computer Networks*, 147:146 – 161, 2018.

[78] Douglas M Hawkins. *Identification of outliers*. London ; New York : Chapman and Hall, 1980. Includes index.

[79] M. A. Hayes and M. A. M. Capretz. Contextual anomaly detection in big sensor data. In *2014 IEEE International Congress on Big Data*, pages 64–71, June 2014.

[80] Joshua Wai-ho Hui, Ning Li, Hui-I Hsiao, and Parag V Tijare. Providing xml cursor support on an xml repository built on top of a relational database system, March 14 2006. US Patent 7,013,311.

[81] Ming-Chuan Hung, Man-Lin Huang, Don-Lin Yang, and Nien-Lin Hsueh. Efficient approaches for materialized views selection in a data warehouse. *Information Sciences*, 177(6):1333 – 1348, 2007.

[82] Huy M. Huynh, Loan T.T. Nguyen, Bay Vo, Zuzana Komínková Oplatková, Philippe Fournier-Viger, and Unil Yun. An efficient parallel algorithm for mining weighted clickstream patterns. *Information Sciences*, 582:349–368, 2022.

[83] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., USA, 1992.

[84] Mikel Iturbe, Inaki Garitano, Urko Zurutuza, and Roberto Uribeetxeberria. Towards large-scale, heterogeneous anomaly detection systems in industrial networks: A survey of current trends. *Security and Communication Networks*, 2017, 2017.

[85] Qingchun Jiang and Sharma Chakravarthy. Queueing analysis of relational operators for continuous data streams. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, page 271–278, New York, NY, USA, 2003. Association for Computing Machinery.

[86] Theodore Johnson, Ivy Kwok, and Raymond T. Ng. Fast computation of 2-dimensional depth contours. In *KDD*, 1998.

[87] Theodore Johnson and Vladislav Shkapenyuk. Data stream warehousing in tidalrace. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org, 2015.

[88] Abdullah Talha Kabakus and Resul Kara. A performance evaluation of in-memory databases. *Journal of King Saud University - Computer and Information Sciences*, 29(4):520 – 525, 2017.

[89] Leonid Kalinichenko, I Shanin, and I Taraban. Methods for anomaly detection: A survey. 1297:20–25, 01 2014.

[90] N Karthikeyan, Saravana Balaji B, and Revathy S. From data warehouses to streaming warehouses: A survey on the challenges for real-time data warehousing and available solutions. *International Journal of Computer Applications*, 81:975–8887, 12 2013.

[91] S. Khemmarat, R. Zhou, D.K. Krishnappa, L. Gao, and M. Zink. Watching user generated videos with prefetching. *Signal Processing: Image Communication*, 27(4):343 – 359, 2012. Modern Media Transport – Dynamic Adaptive Streaming over HTTP (DASH).

[92] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996.

[93] Akira Kinoshita, Atsuhiro Takasu, and Jun Adachi. Real-time traffic incident detection using a probabilistic topic model. *Information Systems*, 54(Supplement C):169 – 188, 2015.

[94] Edwin M. Knorr and Raymond T. Ng. A unified approach for mining outliers. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97, pages 11–. IBM Press, 1997.

[95] Yufeng Kou, Chang-Tien Lu, S. Sirwongwattana, and Yo-Ping Huang. Survey of fraud detection techniques. In *IEEE International Conference on Networking, Sensing and Control, 2004*, volume 2, pages 749–754 Vol.2, 2004.

[96] Thomas Krijnen and Jakob Beetz. An efficient binary storage format for ifc building models using hdf5 hierarchical data format. *Automation in Construction*, 113:103134, 2020.

[97] Kam-Yiu Lam and Chris C.H. Ngan. Temporal pre-fetching of dynamic web pages. *Information Systems*, 31(3):149 – 169, 2006.

[98] Jacek M. Leski. Fuzzy c-ordered-means clustering. *Fuzzy Sets Syst.*, 286(C):114–133, March 2016.

[99] Chen Li, Bo Li, Md Zakirul Alam Bhuiyan, Lihong Wang, Jinghui Si, Guanyu Wei, and Jianxin Li. Flutedb: An efficient and scalable in-memory time series database for sensor-cloud. *Journal of Parallel and Distributed Computing*, 122:95 – 108, 2018.

[100] Xiaocui Li, Zhangbing Zhou, Junqi Guo, Shangguang Wang, and Junsheng Zhang. Aggregated multi-attribute query processing in edge computing for industrial iot applications. *Computer Networks*, 151:114 – 123, 2019.

[101] Zhiyu Li, Aishe Shui, Kaiwen Luo, Jiachuan Chen, and Ming Li. SIR-based oil tanks leak detection method. In *2011 Chinese Control and Decision Conference (CCDC)*, pages 1946–1950. IEEE, 2011.

[102] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 6(1):3:1–3:39, March 2012.

[103] Liang Liu, Xiao-Lin Qin, and Gui-Neng Zheng. Reliable spatial window aggregation query processing algorithm in wireless sensor networks. *Journal of Network and Computer Applications*, 35(5):1537 – 1547, 2012. Service Delivery Management in Broadband Networks.

[104] Fang Lv, Wei Wang, LinXuan Han, Di Wang, Junheng Huang, and Bailing Wang. Mining trading patterns of pyramid schemes from financial time series data. *Future Generation Computer Systems*, 2022.

[105] Santosh Kumar Mandal, Felix T.S. Chan, and M.K. Tiwari. Leak detection of pipeline: An integrated approach of rough set theory and artificial bee colony trained {SVM}. *Expert Systems with Applications*, 39(3):3071 – 3080, 2012.

[106] Andrea Marin, Simonetta Balsamo, and Jean-Michel Fourneau. Lb-networks: A model for dynamic load balancing in queueing networks. *Performance Evaluation*, 115:38 – 53, 2017.

[107] Shikha Mehta, Parul Agarwal, Prakhar Shrivastava, and Jharna Barlawala. Differential bond energy algorithm for optimal vertical fragmentation of distributed databases. *Journal of King Saud University - Computer and Information Sciences*, 2018.

[108] Favio Ezequiel Miranda-Perea and Lourdes Del Carmen González-Huesca. Selective memoization with box types. *Electronic Notes in Theoretical Computer Science*, 256:67 – 85, 2009. Proceedings of the

Fourth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2009).

[109] Debadatta Mishra and Purushottam Kulkarni. A survey of memory management techniques in virtualized systems. *Computer Science Review*, 29:56 – 73, 2018.

[110] Rasha Osman, Irfan Awan, and Michael E. Woodward. Application of queueing network models in the performance evaluation of database designs. *Electronic Notes in Theoretical Computer Science*, 232:101 – 124, 2009. Proceedings of the Third International Workshop on the Practical Application of Stochastic Modelling (PASM 2008).

[111] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In Christian S. Jensen, Markus Schneider, Bernhard Seeger, and Vassilis J. Tsotras, editors, *Advances in Spatial and Temporal Databases*, pages 443–459, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[112] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: fast outlier detection using the local correlation integral. In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, pages 315–326, 2003.

[113] Neoklis Polyzotis, Spiros Skiadopoulos, Panos Vassiliadis, Alkis Simitsis, and N.-E Frantzell. Meshing streaming updates with persistent data in an active data warehouse. *Knowledge and Data Engineering, IEEE Transactions on*, 20:976–991, 08 2008.

[114] Neoklis Polyzotis, Spiros Skiadopoulos, Panos Vassiliadis, Alkis Simitsis, and N.E. Frantzell. Supporting streaming updates in an active data warehouse. pages 476–485, 05 2007.

[115] Parisa Rashidi. Chapter 5 - stream sequence mining for human activity discovery. In Gita Sukthankar, Christopher Geib, Hung Hai Bui, David V. Pynadath, and Robert P. Goldman, editors, *Plan, Activity, and Intent Recognition*, pages 123–148. Morgan Kaufmann, Boston, 2014.

[116] Uwe Röhm, Klemens Böhm, Hans-Jörg Schek, and Heiko Schuldt. Chapter 65 – FAS – a freshness-sensitive coordination middleware for a cluster of olap components. In Philip A. Bernstein, Yannis E. Ioannidis, Raghu Ramakrishnan, and Dimitris Papadias, editors, *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pages 754 – 765. Morgan Kaufmann, San Francisco, 2002.

[117] Ida Ruts and Peter J. Rousseeuw. Computing depth contours of bivariate point clouds. *Comput. Stat. Data Anal.*, 23(1):153–168, November 1996.

[118] Marta Sigut, Silvia Alayón, and Eladio Hernández. Applying Pattern Classification Techniques to the Early Detection of Fuel Leaks in Petrol Stations. *Journal of Cleaner Production*, 80:262–270, 2014.

[119] Mohammad Karim Sohrabi and Hossein Azgomi. Evolutionary game theory approach to materialized view selection in data warehouses. *Knowledge-Based Systems*, 163:558 – 571, 2019.

[120] Krishnarajanagar G. Srinivasa and Srinidhi Hiriyannaiah. Chapter five - comparative study of different in-memory (no/new) sql databases. In Pethuru Raj and Ganesh Chandra Deka, editors, *A Deep Dive into NoSQL Databases: The Use Cases and Applications*, volume 109 of *Advances in Computers*, pages 133 – 156. Elsevier, 2018.

[121] Cédric St-Onge, Nadjia Kara, Omar Abdel Wahab, Claes Edstrom, and Yves Lemieux. Detection of time series patterns and periodicity of cloud computing workloads. *Future Generation Computer Systems*, 109:249–261, 2020.

[122] Thomas Sterling, Matthew Anderson, and Maciej Brodowicz. Chapter 5 - the essential resource management. In Thomas Sterling, Matthew Anderson, and Maciej Brodowicz, editors, *High Performance Computing*, pages 141 – 190. Morgan Kaufmann, Boston, 2018.

[123] Michael Stonebraker, Uundefinedur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Rec.*, 34(4):42–47, December 2005.

[124] Dawei Sun, Shang Gao, Xunyun Liu, Xindong You, and Rajkumar Buyya. Dynamic redirection of real-time data streams for elastic stream computing. *Future Generation Computer Systems*, 112:193 – 208, 2020.

[125] Tatsawan Timakum, Soobin Lee, and Min Song. Exploring the research landscape of data warehousing and mining based on dawak conference full-text articles. *Data & Knowledge Engineering*, 135:101926, 2021.

[126] United States Environmental Protection Agency. Standard Test Procedures For Evaluating Leak Detection Methods: Statistical Inventory Reconciliation Methods. Final Report, 1990.

[127] Vikash, Lalita Mishra, and Shirshu Varma. Performance evaluation of real-time stream processing systems for internet of things applications. *Future Generation Computer Systems*, 113:207 – 217, 2020.

[128] Fatos Xhafa, Burak Kilic, and Paul Krause. Evaluation of iot stream processing at edge computing layer for semantic data enrichment. *Future Generation Computer Systems*, 105:730 – 736, 2020.

[129] Guoqing Xiao, Kenli Li, Xu Zhou, and Keqin Li. Queueing analysis of continuous queries for uncertain data streams over sliding windows. *International Journal of Pattern Recognition and Artificial Intelligence*, 30(09):1660001, 2016.

[130] Myuu Myuu Wai Yan. Accurate detecting concept drift in evolving data streams. *ICT Express*, 2020.

[131] Tao Yang, Bingnan Hou, Zhiping Cai, Kui Wu, Tongqing Zhou, and Chengyu Wang. 6graph: A graph-theoretic approach to address pattern mining for internet-wide ipv6 scanning. *Computer Networks*, 203:108666, 2022.

[132] Tim Zimmermann, Timo Djürken, Arne Mayer, Michael Janke, Martin Boissier, Christian Schwarz, Rainer Schlosser, and Matthias Uflacker. Detecting fraudulent advertisements on a large e-commerce platform. In *EDBT/ICDT Workshops*, 2017.

Spis rysunków

Spis tabel