ROZDZIAŁ 99

STRUMIENIOWE HURTOWNIE DANYCH I STRUMIENIOWY JĘZYK ZAPYTAŃ

Marcin GORAWSKI[1]

System przetwarzania strumieniowego SPS bazuje na modelach danych obejmujących strumienie danych w postaci powiązanych ze sobą tematycznie elementów (rekordów) należących do nieograniczonych zbiorów krotek oraz znaczników czasowych. Większość SPS to systemy dedykowane, przez co ich język zapytań ciągłych CQL jest silnie związany z konkretną dziedziną. Chcąc uczynić taki język uniwersalnym dla strumieniowych hurtowni danych (ang. *Stream Data Warehouse*, StrDW) proponuje się strumieniowy język zapytań (ang. *Stream Query Language*). W rozdziale przedstawiona zostanie autorska architektura strumieniowej hurtowni danych wraz z językiem StreamAPAS.

1. WSTĘP

Systemy przetwarzania strumieniowego SPS (ang. *Stream Processing System*) (potocznie nazywany także strumieniową bazą danych) są wykorzystywane m.in. jako systemy nadzoru ruchu ulicznego, czy monitorowania notowań giełdowych. SPS wyróżniają cechy: ciągłe przetwarzanie zapytań, dynamiczne „wypychanie" danych, krótki czas atencji, potokowość, skalowalność oraz wysoka przepustowość. Ostatnio, ukazało się kilka znaczących modeli SPS, gdzie wszystkie strumienie źródłowe są łączone w centralnej jednostce, i tam są przetwarzane. Znane scentralizowane SPS to Aurora, STREAM, TelegraphCQ, NiagaraCQ [4, 6]. Ostatnie propozycje dotyczą rozproszonych SPS, którego najnowszym reprezentantem jest silnik Borealis [1]. SPS służą do przetwarzania strumieni danych, a nie ich magazynowania i archiwizowania [2, 3,18, 20, 21, 22]. Ta koncepcyjna ograniczoność SPS uniemożliwia ich zastosowanie do budowy systemów wspomagania decyzji DSS. Z drugiej strony, koncepcja przetwarzania strumieniowego wzbogacona o archiwizację strumieni danych przestrzennych wiąże się z większą złożonością procesu uruchamiania zapytania, które wymaga zasilania danymi historycznymi, słownikami oraz mapami. Ponadto w trakcie realizacji zapytania należy monitorować

[1] Politechnika Śląska, Instytut Informatyki; ul. Akademicka 16, 44-101 Gliwice; marcin.gorawski@polsl.pl.

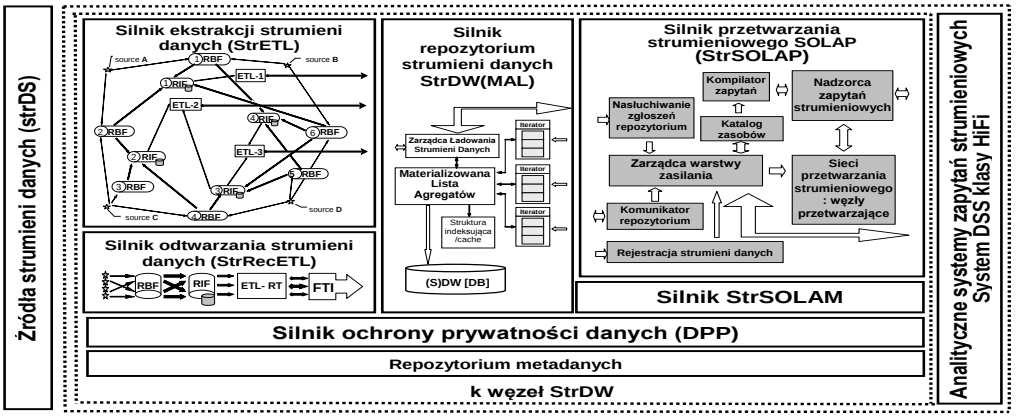
węzły przetwarzające, aby w przypadku awarii poprawnie wyłączyć sieć przetwarzania [5, 23]. Nową koncepcją przetwarzania strumieni danych wraz z ich archiwizacją jest autorski model strumieniowej hurtowni danych (ang. *Stream Data Warehous*, StrDW).

2. ARCHITEKTURA STRUMIENIOWEJ HURTOWNI DANYCH (STRDW)

Strumieniową hurtownię danych StrDW zalicza się do klasy zaawansowanych hurtowni danych (ang. *Advanced Data Warehouse*, AdvDW) [9].

Definicja 1

Strumieniową hurtownię danych StrDW tworzy platforma przetwarzania różnorodnych zbiorów danych strumieniowych (pojedynczych danych przestrzennych i nieprzestrzennych oraz strumieni danych) StrDS zmagazynowanych w StrDW [MAL] w informację analityczną przy użyciu zintegrowanych ze sobą silników tj.: StrETL, StrRecETL, StrDW(MAL), StrSOLAP, StrSOLAM, PDA. Architekturę StrDW przedstawia rys. 1.



Rys. 1. Architektura strumieniowej hurtowni danych StrDW

Do opisania architektury StrDW wprowadzono wiele nowych koncepcji i pojęć, m.in. ***strumieniowe przetwarzanie analityczne w trybie on-line StrSOLAP*** **(**ang. *Stream Spatial On-Line Analytical Processing)* (w skrócie: strumieniowy SOLAP). StrSOLAP ma na celu wydajne obliczanie wielowymiarowych agregatów z predykatami zakresów przestrzennych i czasowych na zbiorze danych bieżących lub zmagazynowanych fragmentach strumieni danych. StrSOLAP tworzy nową klasę w AdvOLAP. Silniki StrSOLAP oraz pozostałe silniki strumieniowe przestawione na rys. 1 pozostają w fazie koncepcji, modeli, metod projektowania oraz prototypowania.

Przykład 1
Modelowanie źródła strumieni danych (StrDS). Przyjmuje się, że źródło danych jest mechanizmem zdolnym jedynie do transmisji ciągłego strumienia danych, bez możliwości ich buforowania. Zakłada się także, że każde źródło jest w stanie zaznaczyć każdą produkowaną krotkę, za pomocą znacznika o rosnącej wartości - globalne oznaczanie nie jest konieczne. Zatem jedynymi operacjami, jakie należy wykonać, to podłączenie do źródła i odbieranie bieżących danych. Załóżmy, że mamy N liczników. Każdy licznik transmituje swój pomiar co T sekund. Jeżeli licznik połączył się ze źródłem w chwili t_i, to następna transmisja wykona się w chwili $\{t_i + T\}$. W przedziale czasu T, nie wiadomo, kiedy każdy licznik połączy się z węzłem zbiorczym. Niech okres czasu T ma n_s szczelin czasowych, każda o rozmiarze t. Liczba szczelin czasowych $n_s = T / t$. Niech Z oznacza transmisję danych przez licznik w jakiejkolwiek szczelinie czasowej. Prawdopodobieństwo, że licznik transmituje dane w konkretnej szczelinie czasowej wynosi: $P(Z) = 1/n_s = t / T$. Każde zdarzenie Z_i możemy opisać za pomocą wzoru: $P(Z_i) = 1 / n_s$. W [12] wykazano, że:

- Oczekiwana liczba pomiarów w k-tej szczelnie czasowej wynosi:

$$E_k(X) = E(X) = \frac{Nt}{T} \quad (1)$$

- Prawdopodobieństwo, że w sekwencji m szczelin czasowych, w każdej szczelinie uzyskamy więcej niż E(X) + k * D(X) krotek, wynosi k^{-2m}.

$$P(|E(X) - \mu| \geq k \cdot D(X)) \leq \frac{1}{k^2}; \quad D(X) \leq \sqrt{\frac{Nt}{T}} \quad (2)$$

3. MODEL PRZETWARZANIA STRUMIENIOWEGO (STRSOLAP)

Podstawowym pojęciem SPS jest strumień krotek. Niech $I := \{[t_s, t_e) \in T \times T \mid t_s \leq t_e\}$ oznacza zbiór przedziałów czasowych z punktami czasowymi, T - dziedzina czasu.

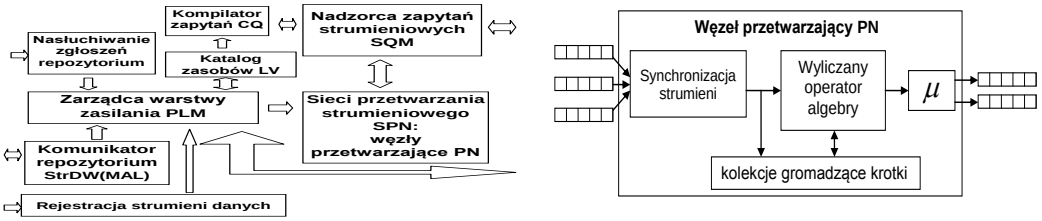
Definicja 2
Strumień krotek opisuje para symboli $S = (M, \leq_{t_s,t_e})$ gdzie: M - nieskończony strumień krotek $(type, e, [t_s, t_e))$ gdzie: type - typ krotki, e – dane transportowane przez krotkę $[t_s, t_e) \in I$, $\leq_{t_s,t_e}$ - leksykograficzne uporządkowanie elementów strumienia M.

W [10,] zdefiniowano wiele typów krotek, przez co możliwe jest zarządzanie siecią przetwarzania przy użyciu strumieni źródłowych.

Model strumieniowego przetwarzania analitycznego w trybie on-line (ang. *Stream-Spatial On-Line Analytical Processing* _Stream SOLAP) (w skrócie: **strumieniowy**

SOLAP lub zamiennie **StrSOLAP**) ma na celu wydajne obliczanie wielowymiarowych agregatów z predykatami zakresów przestrzenno-czasowych na zbiorze strumieni danych zmagazynowanych w repozytorium strumieni danych StrDW(MAL).

Model przetwarzania StrSOLAP przedstawia rys. 2.



a) b)

Rys.2. a) Model przetwarzania strumieniowego SOLAP (StrSOLAP), b) Przepływ krotek w węźle przetwarzającym

Nadzorca zapytań strumieniowych *(SQM)* kontroluje przebieg procesu kompilacji zapytań strumieniowych oraz sprawuje nadzór nad uruchomioną siecią przetwarzania strumieniowego. SQM obsługuje komunikaty specjalne generowane przez sieć przetwarzania, które sygnalizują: a) wykrycie zmiany meta-schematu jednego z strumieni źródłowych, b) usunięcie strumienia źródłowego z katalogu strumieni oraz c) odbiór wyjątku w trakcie pracy węzła przetwarzającego PN.

Sieć przetwarzania strumieniowego *(SPN)* tworzy węzły przetwarzające PN połączone strumieniami danych zgodnie z definicją zapytania. SPN przy użyciu operatorów fizycznych uruchamianych w PN implementuje operacje logiczne zdefiniowane poprzez zapytanie w postaci tekstowej. Logika tych operatorów bazuje na algebrze operatorowej z [19]. Aktualnie SPN realizuje operatory: filtracji(σ), mapowania(μ), łączenia($\bowtie$), okien czasowych(ω), unii($\bigcup$), agregacji(α). W architekturze SPN rozdzielono funkcjonalność użytą przez większość operatów fizycznych (taką jak porządkowanie krotek strumieni wejściowych, obsługa komunikatów błędów) od konkretnej ich realizacji. Ponieważ przetwarzanie strumieniowe nie jest blokowalane, każdy z PN to odrębny wątek w ramach okna czasowego.

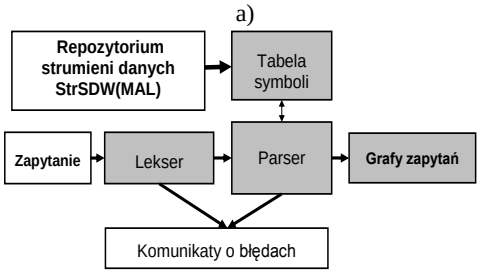
Węzeł przetwarzający *(PN)* tworzy środowisko SPN, w którym uruchamiane są operatory fizyczne. Na rys. 2b przedstawiono przepływ krotek w PN, w trakcie obliczania zapytania wg poniższego scenariusza działania:

- Uporządkowanie leksykograficzne krotek dla kilku strumieni wejściowych.
- Reorganizacja kolekcji krotek w oparciu o znacznik czasu krotki przekazanej.
- Wstawienie krotki do kolekcji (jeżeli kolekcja krotek zasila dodatkowo indeks, zamiany zgłaszane są przy użyciu wydarzeń: *insert, update, delete*).
- Realizacja operacji algebry i wyznaczenie krotki wynikowej (wstawienie do strumienia wynikowego krotki z zredukowaną liczbą atrybutów).

Katalog zasobów *(LV)* to repozytorium metaschematów strumieni danych.

Kompilator zapytań *(CQ)* potrzebne informacje o strukturach strumieni pobiera z LV, a wykryte błędy w zapytaniu kieruje do SQM. Na rys.3 przedstawiono przepływ sterowania podczas kompilacji zapytania.

a)



b)

```
    NodeDecl:   QualifiedPath   {wstaw
ścieżkę} '=' Exp {zdejmij ścieżkę}
            …
    Level:  QualifiedPath  '['  {wstaw
ścieżkę} NodesDecl ']'
         {zdejmij ścieżkę}
         |   '['   {wstaw   tymczasową
zmienną}     NodesDecl     ']'{zdejmij
ścieżkę}
```

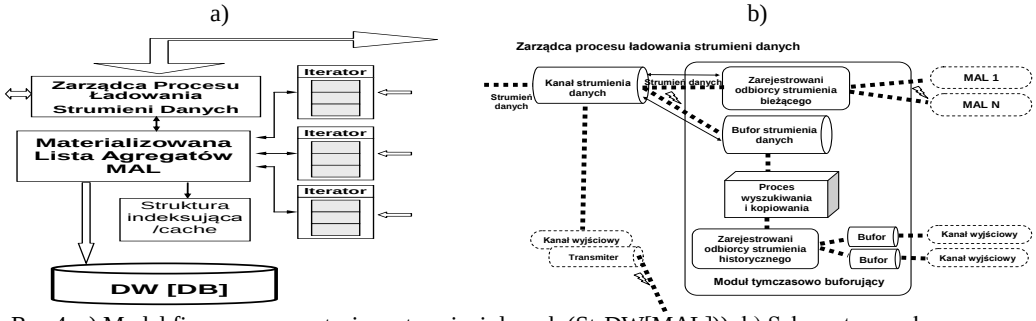
Rys.3. Kompilator zapytań: a) przebieg kompilacji zapytania, b) Produkcje gramatyki

Języki zapytań odwołują się do atrybutów nie zdefiniowanych wewnątrz ich treści. Są nimi zmienne przechowywane w repozytorium StrDW(MAL) (np. lista dostępnych strumieni, metaschemat krotek strumieni, aktywność strumieni). W trakcie kompilacji zapytania, informacje przechowywane w StrDW(MAL) mogą ulegać zmianie, dla zachowania spójności tabeli symboli kompilatora przyjęto, że pod nadzorem parsera kopiowane są kolejne struktury opisujące strumienie. Podejście takie wymaga podczas uruchamiania SPN dodatkowego zbadania, czy użyte meta-schematy nie uległy zmianie od chwili przystąpienia do kompilacji. Zbudowany język korzysta z atrybutów ułożonych w hierarchie, co wiąże się z rozbudową parsera o dodatkowy stos, na który odkładana jest bieżąca pozycja kontekstu zmiennych. Każdy atrybut musi istnieć w tablicy symboli zanim zostanie zdefiniowany operator realizujący na nim obliczenia. Stąd akcje semantyczne sterujące działaniem stosu kontekstu zostały umieszczone wewnątrz produkcji gramatyki (rys. 3b). Składnia taka sprawia, że tabela symboli jest zasilana tylko przez parser. Przedstawione produkcje gramatyki sprawiają, że znana jest pełna nazwa atrybutu hierarchicznego i na bieżąco kontrolowane są typy wyliczeń.

Zarządca warstwy zasilania *(PLM)* aktualizuje informacje o meta-schemacie strumieni w ramach LV oraz zasila danymi uruchomione SPN. Aby podłączyć do silnika StrSOLAP nowe źródło danych należy wpierw zarejestrować nazwę, pod którą dany zasób będzie dostępny. W trakcie przepływu strumienia danych możliwa jest zmiana metaschematu. W tym celu źródło przesyła krotkę systemową z nową definicją. Zarządca warstwy zasilania aktualizuje wtedy zawartość katalogu zasobów oraz przekazuje tę krotkę do strumieni wewnątrz silnika. Dzięki temu uruchomione zapytania zasilane tym źródłem informowane są o aktualizacji.

3.1. MODEL FIZYCZNY REPOZYTORIUM STRUMIENI DANYCH (STRDW[MAL])

Materializowana Lista Agregatów (MAL) stanowi strumieniową bazę danych modelu fizycznego **repozytorium strumieni danych StrDW[MAL])** (rys. 4). Schemat *Zarządcy Procesu Ładowania Strumieni Danych* dzięki modułowi buforującemu bezpośrednio wpływa na wydajność obsługi żądań strumieni danych zarówno bieżących, jak i historycznych (rys. 4b) [11]^{w}.



Rys.4. a) Model fizyczny repozytorium strumieni danych (StrDW[MAL])), b) Schemat zarządcy procesu ładowania strumieni danych z modułem tymczasowo buforującym

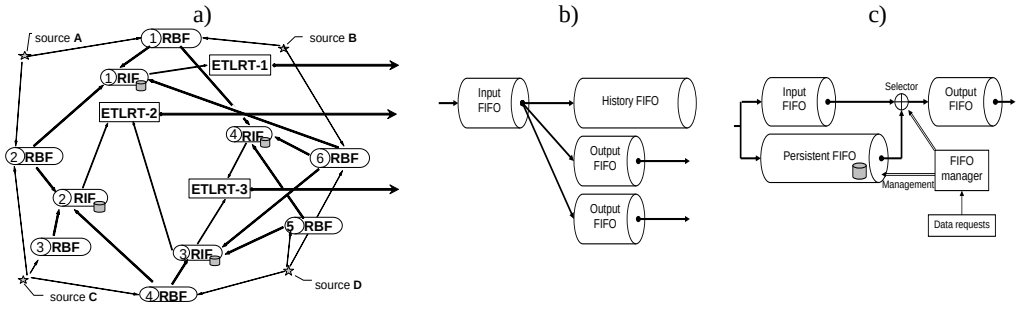
Moduł buforujący występuje w dwóch podstawowych wersjach: (a) trwale buforującej i (b) tymczasowo buforującej. Podstawowe funkcje modułu buforującego są następujące: trwałe lub tymczasowe buforowanie strumienia danych, równoczesna obsługa żądań strumieni danych (bieżących/historycznych), zamiana typu obsługi żądań.

3.2. MODELOWANIE SILNIKA EKSTRAKCJI STRUMIENI DANYCH (STRETL)

Silnik StrETL jest modelowany jako sieć połączonych komponentów o różnej funkcjonalności i jest w pełni autorskim rozwiązaniem (rys. 5a) [12,13]. Architekturę silnika StrETL tworzy sieć przetwarzania zbudowana z następujących komponentów:

- Zdalne Bufory Strumieni **RBF** (ang *Remote Buffer Framework*),
- Zdalne Trwałe Integratory Strumieni **RIF** (ang. *Remote Integrator Framework),*
- Środowisko ekstrakcji strumieni danych **ETL-RT**.

Komponenty RBF oraz RIF mogą przetwarzać wiele strumieni w niezależnych kanałach. *Komponent RBF* (*Zdalne Bufory Strumieni*) przyjmuje natychmiastowo dane ze źródeł strumieni danych. RBF jest zbudowany w oparciu o niezależne kanały (rys. 5b). Każdy kanał przetwarza pojedynczy strumień danych. W każdym kanale znajdują się przynajmniej 2 kolejki FIFO. Jedna kolejka pracuje jako bufor wejściowy dla otrzymanych i jeszcze niewysłanych danych. Druga kolejka (historii) buforuje krotki, które zostały już wysłane. Wszystkie pozostałe kolejki są buforami wyjść RBF (po jednym buforze wyjściowym dla każdego podłączonego następnego komponentu silnika StrETL).



Rys.5. Przykładowa architektura silnika StrETL: a) komponenty silnika StrETL, b) Zdalne Bufory Strumieni (RBF), c) Zdalne Trwałe Integratory Strumieni (RIF)

Rozmiar kolejki historii jest ograniczony i powinien być ustawiony na taką wartość, by w przypadku krótkiej awarii w następnej warstwie systemu - RIF - miała ona szansę na powtórne połączenie się z komponentem RBF, pobranie krotek i kontynuowanie swojej pracy. W przypadku dłuższej awarii warstwy RIF, historyczne dane w kolejce FIFO komponentu RBF będą nadpisane przez dane aktualne i kontynuacja pracy warstwy RIF nie będzie możliwa. Źródło może transmitować dane równocześnie do wielu komponentów RBF.

Komponent RIF (*Zdalny Trwały Integrator Strumieni*) pozwala uzyskać w każdym kanale spójny strumień danych (rys. 5c). Krotki przechowywane w RIF mogą być z niego pobrane w każdej chwili. RIF utrzymuje listę RBF oraz identyfikatory kanałów (strumieni). RIF utrzymuje połączenia z wszystkimi swoimi źródłowymi RBF. Umożliwia mu to natychmiastowe przełączenie się pomiędzy RBF i ma miejsce natychmiast po wykryciu awarii (komponentu lub kanału komunikacyjnego). Maksymalny czas, jaki potrzebny jest do wykrycia awarii, zależy od częstotliwości zapytań o stan źródła (podobnie do funkcji *ping* lub protokołu *heartbeat*). W [6] wyznaczono maksymalny czas przerwy, która może wystąpić w trakcie przyjmowania przez RIF danych z komponentu. Czas ten jest sumą czasów wykrywania awarii i przełączania źródeł

$$T_{break} = (T_{bp} + T_{pingA}) \times (1 + n_{err}) + 4 \times T_{pingB}$$

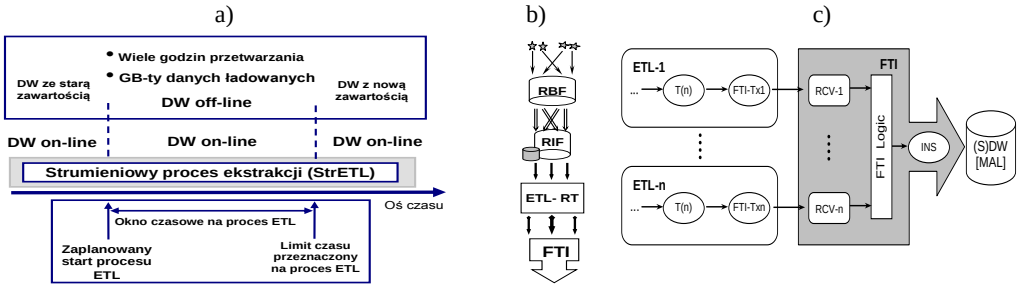
gdzie: T_{bp} - czas pomiędzy pakietami *ping* (konfigurowalne), $T_{pingA(B)}$ – czas zwrócenia odpowiedzi przez pakiet *ping* (test *A* - przerwana komunikacja RBF-RIF, a test *B*- przerwana komunikacja z nowym RBF), n_{err} – liczba błędów pakietów *ping*.

Środowisko ETL-RT jest następcą autorskiego środowiska ETL-DR. ulepszone o mechanizmy przetwarzania strumieni danych i wznawiania przerwanego procesu ekstrakcji. W ETL-RT, tak samo jak w ETL-DR, proces ekstrakcji opisany jest przez bezpośrednie, acykliczne grafy DAG. Najważniejszą zmianą, wprowadzoną w nowym środowisku ETL-RT, jest nowy algorytm wznawiania przerwanego procesu ekstrakcji. Hybrydowe odtwarzania SDR(m), nie jest odpowiednie dla przetwarzania strumieni da-

nych. Stąd nowe rozwiązanie autorskie StrRecETL, które jest kolejnym hybrydowym algorytmem, łączącym pewne cechy algorytmu DR i punktów kontrolnych.

3.3. METODA ODTWARZANIA STRUMIENI DANYCH (STRRECETL)

W trakcie działania zapytania strumieniowego należy monitorować proces ETL, aby w przypadku jego upadku poprawnie wyłączyć silnik StrETL (rys. 6a). Jeżeli dostępny jest zbiór punktów kontrolnych procesu ekstrakcji, to można go wznowić po jego przerwaniu i odzyskać utracone dane nawet na innym węźle obliczeniowym a upadek taki procesu nazywa się upadkiem bezpiecznym. Dużo trudniejszym przypadkiem jest ten, gdy proces zostaje zakłócony produkując błędne wyniki (dane mogą być przetworzone niepoprawnie bez naszej wiedzy o tym) – wtedy występuje niebezpieczny upadek procesu.



Rys.6. (a) Schemat odtwarzania on-line, b) Silnik StrRecETL, c) Schemat połączenia procesów ETL z komponentem FTI

Wówczas jedyną metodą wykrycia takich przypadków jest równoległe przetwarzanie kilku replik całego procesu ETL i porównywanie uzyskanych wyników [13, 14, 15].

W silniku StrRecETL uwzględniono wystąpienie upadku niebezpiecznego (rys. 6b). Awaria jednego RBF nie zakłóca procesu przetwarzania strumieni danych, pod warunkiem, że pozostaje przynajmniej jeden sprawny RBF, który przetwarza każdy strumień danych uszkodzonego komponentu [14]. Ponieważ RIF zbiera statystyki jakości połączeń (szybkość połączeń, dostępność źródła), w przypadku awarii RBF dokonuje efektywnego wyboru nowego RBF. W metodzie StrRecETL występuje *Integrator Tolerujący Błędy* (ang. *Fault Tolerant Integrator* _ **FTI**), który analizuje redundantne strumienie danych, wykrywa błędy, scala strumienie i ostatecznie ładowuje dane do StrDW[MAL]) (rys. 6c). Użycie FTI wymaga modyfikacji komponentu ETL.Węzeł ładujący INS musi zostać zamieniony przez węzeł FTI-Tx, który przesyła dane do komponentu detekcji. Węzeł ten jest traktowany przez komponent ETL jako inserter, ponieważ implementuje wszystkie wymagane interfejsy. Jednak zamiast ładowania danych do miejsc docelowych, strumienie są przesyłane do odbiorników RCV FTI. Otrzymane w ten sposób krotki, przetwarzane są przez logikę FTI.

4. JĘZYK ZAPYTAŃ STRUMIENIOWYCH STREAMAPAS DLA STRDW

Praca [19, 24] zainspirowała autora do badań i zaimplementowania języka CQL (ang. *Continues Query Language)* [16]. Kontynuacją prac nad językiem CQL jest autorski język zapytań strumieniowych StreamAPAS.

Język StreamAPAS obsługujący silnik StrSOLAP ogranicza listę dostępnych funkcji poprzez słowa kluczowe zaszyte w składni języka (podobnie jak w SQL) na korzyść użycia zbioru funkcji zależnych od dołączonych bibliotek [10]. Rozwiązanie to pozwala zachować składnię języka w przypadku dodania nowych funkcji. Kolejnym cechą języka StreamAPAS jest wprowadzenie opisu krotki jako drzewa atrybutów, co pozwala dobrze obrazować struktury analityczne i upraszcza zapis argumentów funkcji. Pojedyncze zapytanie strumieniowe to sieć powiązanych ze sobą podzapytań o składni językowej:
from: {<*deklaracja strumienia*>;}
select <*wyliczenie wartości atrybutów*>
Deklaracja strumienia:
uproszczona <*wyliczenie wartości atrybutów*>,
pełna:
where *wyrażenie operatorowe*
select <*wyliczenie wartości atrybutów*>

operatory – operacja: łączenia, filtracji, okno czasowe, funkcja relacyjna (np. implementacja indeksu), unii

wyliczenie wartości atrybutów –definicja drzewa atrybutów dla strumienia wynikowego, można skorzystać z funkcji agregacyjnej

Okna czasowe o składni:

- *window.range(source, time)* – realizuje funkcję okna zakresowego.
- *window.fixed(source,time)*–realizuje funkcję okna z stałymi interwałami czasu.
- *window.infinite(source)*–realizuje funkcję okna gromadzącego wszystkie krotki.

Atrybuty:

- *source* – nazwa strumienia zasilającego np. *$I*
- *time* – stała wartość definiująca interwał czasu np. *10s*

Kolekcja czasowo-tabelaryczna o składni:
window.table(source),source–nazwa atrybutu będącego kluczem głównym krotki;

Operatory stanowe mogą być zdefiniowane na kilku strumieniach, zatem występuje wiele konfiguracji okien czasowych.

Pojedyncze zapytanie strumieniowe będące wyrażeniem algebry języka StreamAPAS v5.0 przedstawiamy jako graf o konstrukcji zdefiniowanej jak niżej.

Definicja 3

Graf pojedynczego zapytania strumieniowego. Dla danego zapytania strumieniowego p zbudowanego na zbiorze strumieni $S_1,\ldots,S_m$, pojedynczy graf SGp jest oznaczony etykietą oraz węzły SGp są zbiorem strumieni $\{S_1,\ldots,S_m\}$, na którym definiujemy p. Etykieta każdego węzła S_i to $p: X_i, i \in <1,m>$, gdzie X jest zbiorem atrybutów S_i występujących w

wyliczeniach zapytania. Dla każdego wyrażenia atomowego q w p zachodzi warunek, jeśli:

- q jest operatorem jednoargumentowym zdefiniowanym na S_i, istnieje krawędź z S_i do samego siebie (pętla operacyjna) w SGp z etykietą q.
- q jest operatorem wieloargumentowym zdefiniowanym na S_i,..,S_j to istnieje sieć powiązań pomiędzy S_i,..,S_j w SGp z etykietą q.
- q jest definicją kolekcji krotek etykieta, umieszczana przy węźle, na którym jest zdefiniowana.
- SGp nie posiada kolekcji krotek to przyjmuje się, za domyślną, kolekcję czasową.
- Deklaracje strumieni kompilowane są do postaci grafu pojedynczego zapytania. Reprezentacja taka abstrahuje od konkretnego planu produkcji, na której można uruchomić optymalizator zapytania tworzący plan drzewa przetwarzania [25].

Algorytm 1

Budowa drzewa przetwarzania na podstawie grafu pojedynczego zapytania strumieniowego (zapisanego jako lista operatorów):

```
lista_produkcji := {operatory strumieni źródłowych}
for element in lista_operatorów
{
  for strumień_źródłowy in strumienie_źródłowe_operatora(element)
  {
    produkcja := li-
sta_produkcji.operator_zawierający(strumień_źródłowy)
    lista_produkcji.usuń(produkcja)
    element.połącz_strumień_wejściowy_z(produkcja)
  }
  lista_produkcji.wstaw(element)
}
```

Rola listy operatorów jest podobna do stosu operacji i argumentów w odwrotnej notacji polskiej. Informację o położeniu strumienia zasilającego operator *element* otrzymujemy po sprawdzeniu, który operator w bieżącej liście produkcji udostępnia atrybuty wskazanego strumienia źródłowego. Zbiór udostępnianych przez operator atrybutów jest sumą zbiorów udostępnianych atrybutów operatorów dzieci. Operator strumienia źródłowego zamyka cykl wywołań rekurencyjnych, operator ten definiuje zbiór atrybutów danego strumienia źródłowego oraz typ kolekcji krotek. Kolejne permutacje ułożeń operatorów w liście rozbioru, odpowiadają kolejnym planom produkcji. Mechanizm sterowania optymalizacją dla przyjętej struktury sprowadza się do wyznaczenia właściwej permutacji obiektów w liście. Na końcu procesu tworzenia drzewa węzłów przetwarzających definiowane są operatory mapowania, minimalizujące liczbę transmitowanych danych w strumieniach wewnętrznych.

Indeks agregatów to lista agregatów, aktualizowana inkrementacyjnie kolejnymi krotkami strumienia wejściowego. Każdy z agregatów ma ze sobą skojarzony interwał czasu życia. Lista agregatów jest systematycznie czyszczona z elementów wygasłych, które są następnie przekazywane do strumienia wynikowego. Niech $S = (M, \leq_{t_s,t_e})$ jest strumieniem wejściowym operatora oraz funkcję $f : \Omega \cup \{\perp\} \times M \to \Omega$, której argumentami są: bieżąca wartość agregatu $\breve{s} \in \Omega \cup \{\perp\}$ oraz wartość, o jaką następuje aktualizacja $s = (e, [t_s, t_e)) \in M$, gdzie $\{\perp\}$ służy inicjalizacji pustego agregatu. Na początku następuje identyfikacja, które agregaty w indeksie nachodzą na interwał $[t_s, t_e)$. W przypadku częściowego pokrycia wykonany jest podział elementu na dwa interwały: część pokrytą i wolną, której wartość pozostaje bez zmian. Następnie aktualizowane zostają wartości elementów nachodzących na interwał $[t_s, t_e)$. W trakcie czyszczenia listy agregatów usuwane są wszystkie elementy, których znaczniki końca życia są mniejsze lub równe t_s aktualnie przetwarzanej krotki wejściowej, po czym usunięte elementy przekazywane są do listy wynikowej. Elementy indeksu agregatów są uporządkowane wg $\leq_{t_s,t_e}$, stąd otrzymana lista agregatów w wyniku operacji reorganizacji może zasilić strumień wynikowy bez potrzeby dodatkowego sortowania (wymóg porządku leksykograficznego). Prezentowany indeks nie przechowując krotek źródłowych, wyklucza zdefiniowanie operacji usuwania wartości dla agregatów typu *min, max* (użycie okna czasowo-tabelarycznego pozwala wykonać tę operację). W przypadku agregatów *count, sum i avg*, realizacja funkcji usuwania polega na wstawianiu elementu o wartości przeciwnej.

Przykład 2

Niech na danym odcinku drogi jest kilka punktów pomiaru prędkości i należy wskazać sektory, na których jest duże prawdopodobieństwo przekraczania szybkości. Każde urządzenie monitorujące nadaje znacznik początku czasu życia pomiaru oraz znacznik końca, który odpowiada czasu pobytu monitorowanego obiektu we wskazanym sektorze. W zapytaniu wylicza się średnią wartość prędkości samochodów dla kolejnych obszarów monitorowanych. Następnie łączy się wyliczone strumienie, po czym odfiltrowuje te sektory, których wartość średnia przekracza ustalony limit. Dodatkowo wylicza się wartość równą połowie między najmniejszą a największą szybkością odnotowaną w danym sektorze. Wskaźnik ten rozszerza informację o zarys rozkładu prędkości dla monitorowanego sektora. Wynikowy strumień można użyć do podjęcia decyzji, gdzie należy umieścić patrole. Strumienie I i II reprezentują pomiary prędkości samochodów w dwóch sektorach.

```
from
o1[sektor = 1, avg = agg.avg($I.val),
          minmax = (agg.min($I.val) + agg.max($I.val)) / 2];
o2[sektor = 2, avg = agg.avg($II.val),
          minmax = (agg.min($II.val) + agg.max($II.val)) / 2];
where util.union($o1[T], $o2[T])
select facts[$new[T]];
where $facts.avg > 0.4
```

```
select result[$facts[T]];
select [$result[T]]
```

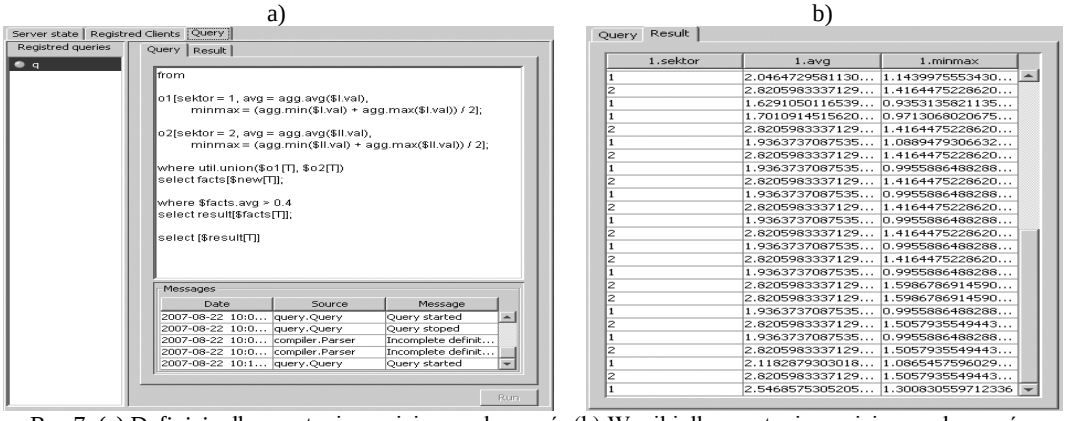
Pomiar składa się z atrybutu val i interwału czasu jego aktywności. Wartości pomiarów generowało losowe źródło wartości z zakresu <0, 2.8). Poniżej przytoczono listing zapytania strumieniowego. W tab.1 zamieszczono przykładową zawartość strumieni dla zdefiniowanego zapytania. Przyjęto, że średnia prędkości 70 km/h kwalifikuje sektor do kontroli. Dla uproszczenia zapisu kolumna *result* informuje, które krotki ze strumienia *facts* zostaną przekazane do strumienia *result*. Opóźnienie obsługi krotek przez operator *util.union* wynika z potrzeby uporządkowania krotek wejściowych. Operator *util.union* łączy strumienie wejściowe w jeden strumień wyjściowy o nazwie *new*. Zapis *facts[$new[T]]* oznacza, że całe poddrzewo atrybutów dla strumienia *$new* jest kopiowane do strumienia *facts*.

Tabela 1. Wyniki pomiarów

I		o1				II		o2				facts				result
(interval)	val	(interval)	slot	avg	minmax	(interval)	val	(interval)	Slot	avg	minmax	(interval)	slot	avg	minmax	
<2,10)	40															
						<3,7)	90									
<4,10)	70	<2,4)	1	40	40											
						<5,9)	70	<3,5)	2	90	90	<2,4)	1	40	40	
						<7,14)	50	<5,7)	2	80	80					
<8,14)	70	<4,8)	1	55	55							<3,5)	2	90	90	<-
												<4,8)	1	55	55	
<9,17)	80	<8,9)	1	60	55							<5,7)	2	80	80	<-
						<9,18)	100	<7,9)	2	60	60	<7,9)	2	60	60	

Na rys. 7a przedstawiono uzyskane wyniki a na rys. 7b postać formatki definiującej zapytanie. Oba z monitorowanych sektorów odnotowują wartości przekraczające limit.

a) b)



Rys.7. (a) Definicja dla zapytania o miejsca wykroczeń, (b) Wyniki dla zapytania o miejsca wykroczeń

Praca StrDW jest płynna, co wskazuje atrybut *sektor*, ponieważ wartości nadchodzą w zasadzie na przemian. Jeżeli chce się wyliczyć średnią prędkość samochodów, które przebywały w danym sektorze w przeciągu ostatniej godziny należy zdefiniować okno zakresowe o rozmiarze 1h na strumieniach zasilających. W odniesieniu do poprzedniego zapytania zmianie ulegnie poniższy fragment.

```
where window.range($I, 1h)
select o1[sektor = 1, avg = agg.avg($I.val),
          minmax = (agg.min($I.val) + agg.max($I.val)) / 2];
where window.range($II, 1h)
select o2[sektor = 2, avg = agg.avg($II.val),
          minmax = (agg.min($II.val) + agg.max($II.val)/ 2];
```

Odnosząc się do wyników z tab.1, modyfikacja skutkuje następującymi wartościami dla strumienia o2 (tab. 2). Przyjęto, że okno zakresowe ma rozmiar 10 jednostek. W przypadku błędnego zdefiniowania zapytania, nie nastąpi przepełnienie kolekcji krotek.

Tabela 2. Przykładowa zawartość strumieni po użyciu okna zakresowego

II			o2			
(interval)	New(interval)	val	(interval)	Slot	avg	minmax
<3,3)	<3,13)	90				
<5,5)	<5,15)	70	<3,5)	2	90	90
<7,7)	<7,17)	50	<5,7)	2	80	80
<9,9)	<9,19)	100	<7,9)	2	70	70

.

Przyjęta składnia języka zapytań strumieniowych StreamAPAS pozwala efektywnie zarządzać złożonymi strukturami danych.

4.1. ANALIZY WYDAJNOŚCIOWE SILNIKÓW STRDW

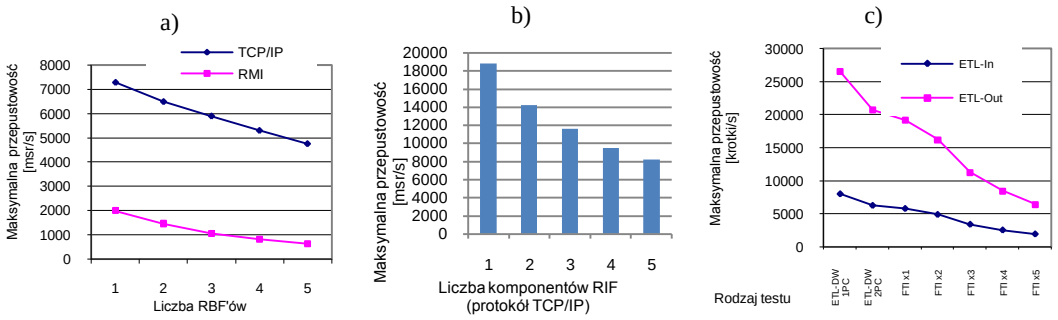
Silnik StrETL badano w zakresie wydajności: a) połączenia źródła strumieni danych StrDS z komponetami RBF, b) komunikacji pomiędzy komponentami RBF-RIF oraz c) ekstrakcji i ładowania z użyciem komponentu ETLRT. Każdy komponent silnika StrETL znajduje się na komputerze: PIV 2.8GHz 512MB RAM WinXP or Win2000, Java 1.5 a silnik StrSDW[MAL] to PIV 3.2GHz HT 1GB RAM WinXP, Oracle 10g.

Badanie wydajności połączenia źródła StrDS z RBF zostało przeprowadzone w dwóch przypadkach: przy komunikacji kanałem TCP/IP oraz standardowym RMI. Z rys. 8a wynika, że maksymalną do osiągnięcia prędkością „wysyłania danych" ze źródła StrDS kanałem TCP/IP jest w naszym przypadku ok. 7300 rekordów na sekundę. Ograniczenie to nie jest krytycznym i w systemie 2MDSS nie występuje. Ponieważ każdemu licznikowi odpowiada oddzielny wątek aplikacji JAVY (testowanych liczników było ok. 2200 tj. synchronizowanych i przełączanych wątków).

Komunikacja standardowym RMI jest znacznie wolniejsza, co spowodowane jest znacznie większym narzutem programowym. Co ciekawe, gdy mamy tylko 1 RBF, ograniczeniem wydajności był nie komputer (PIV 3,2GHz) lecz sieć - połączenie 100Mbit/s okazuje się niewystarczające, aby protokołem RMI wykonać 2000 przesyłów/sek. W dalszych testach nie badano już małowydajnych połączeń RMI.

Badanie wydajności komunikacji komponentów RBF-RIF wykazało, że występują sytuacje, w których z jednego RBF pobiera dane wiele RIF. Z rys. 8b wynika, że

maksymalna szybkość przesyłu danych to prawie 19 tys. pomiarów na sekundę przy jednym komponencie RIF'ie i ok. 8200 przy 5 - RIF'ach.



Rys.8. Maksymalna przepustowość; a) RBF, b)RIF, c) różne konfiguracje

Badanie wydajności ekstrakcji i ładowania z użyciem komponentu ETLRT. *K*omponent ETLRT realizuje strumieniowe przetwarzanie danych (pomiarów) dla gwiazdy kaskadowej. Dla uzyskania obrazu zmian wydajności ciągłego procesu ETL przebadano przy różnych konfiguracjach (rys.8c). ETL-In to szybkość odbierania krotek przez proces ETLRT na każdym komputerze, ETL-Out to szybkość produkcji krotek przez proces ETLRT i jednocześnie przepustowość modułu FTI, gdyż w tym teście to on stanowił „wąskie gardło" silnika StrETL. Wartości **In** oraz **Out** różnią się, gdyż w wyniku przetwarzania liczba krotek zwiększa się mniej więcej 3,3 razy. Na całość wyników w testach FTI xN składają się: przetwarzanie realizowane przez proces ETL, komunikacja ETL-FTI, scalanie wyników z wielu modułów ETL oraz ładowanie danych do bazy Oracle 10g. Testy ETLRT-DW pokazują jak wprowadzenie komponentu FTI wpływa na obniżenie wydajności. Zwiększając nadmiarowość przetwarzania z 1 do 5 równoległych procesów, zmniejsza się wydajność o 67% (wpływ implementacji komponentu FTI) [13].

Przeprowadzono wiele innych testów m.in.: a) *silnika zarządcy dystrybucją strumieni danych* [17], b) *silnika IPETL* w hierarchicznej sieci LAN [11] oraz c) silnika StrRecETL [12].

5. PODSUMOWANIE

Przedstawiono autorską architekturę strumieniowej hurtowni danych StrDW. Podstawą StrDW jest strumieniowy język zapytań StreamAPAS obsługujący silnik StrSOLAP. Użyty w tym języku zapytań model danych wymagał rozbudowy analizatora leksykalnego o dodatkowy stos kontekstu drzewa atrybutów; z drugiej strony otrzymano strukturę, która w czytelny sposób ilustruje dane analityczne oraz przestrzenne. Zaproponowany mechanizm kompilacji zapytań korzystających z funkcji agregacyjnych generuje wynik

w postaci drzewa atrybutów, co ułatwia tworzenie funkcji dedykowanych dla StrDW. StreamAPAS bazuje na algebrze CQL z autorskim rozszerzeniem m.in. o selekcję typu kolekcji krotek. Dodana nowa kolekcja czasowo-tabelaryczna upraszcza przenoszenie rekordów z BD do StrDW. Wprowadzono także wydajniejszy algorytm synchronizacji przetwarzania, co skróciło czas oczekiwania krotek. Obecne prace dotyczą: a) rozbudowy StrOLAP o metodę umieszczania węzłów przetwarzających w systemach wielordzeniowych wzbogaconych o większą liczbę operatorów [8], b) metod definiowania źródeł danych, które dotąd są postrzegane jako generatory prostych strumieni danych (co utrudnia uruchamianie zapytań przez ograniczenie definiowania synchronizacji źródeł danych dla warunków początkowych [7]).

Przedstawione metody ekstrakcji strumieni danych StrETL oraz odtwarzania strumieni danych StrRecETL są w fazie intensywnych badań, a prezentowane wstępne wyniki potwierdzają poprawność założeń i właściwy kierunek rozwoju StrDW.

LITERATURA DO ROZDZIAŁU

[1] Abadi D., Ahmad Y., Balazinska M., Cetintemel U., Cherniack M., Hwang J.-H., Lindner W., Maskey A. S., Rasin A., Ryvkina E., Tatbul N., Xing Y., Zdonik S.: The Design of the Borealis Stream Processing Engine. In Second Biennial Conference on Innovative Data Systems Research. CIDR 2005, s. 277-289.

[2] Babcock, B., Babu, S., Datar, M., Motwani, R.: Chain: Operator scheduling for memory minimization in data stream systems. ACM International Conference on Management of Data, SIGMOD, 2003,s 28-39.

[3] Bai Y., Thakkar H., Wang H., Zaniolo C.: Optimizing timestamp management in data stream management systems. ICDE,2007, s. 1334-1338.

[4] Balakrishnan H., Balazinska, M., Carney D., Cetintemel U., Cherniack M., Convey Ch., Galvez Ch., Salz J., Stonebraker M., Tatbul N., Tibbetts R., Zdonik S.: Retrospective on aurora. The VLDB Journal 13(4) , 2004, s. 370-383.

[5] Balazinska, M.: Fault-tolerance and load management in a Distributed Stream Processing System. PhD dissertation, Massachusetts Institute of Technology, 2006.

[6] Chandrasekaran S., Cooper O., Deshpande A., Franklin M., Hellerstein J.,. Hong W, Krishnamurthy S., Madden S., Raman V., Reiss F., Shah M.: TelegraphCQ: Cominuous dataflow processing for an uncertain world. CIDR. 2003.

[7] Fraser K.,Harris T.: Concurrent programming without locks. ACM Trans. Comput. Syst., 25(2):5, 2007.

[8] Ghanem T.M., Hammad M. A., Elmagarmid, A.K.: Query processing using negative tuples in stream query engines. Technical Report 04-040, Purdue University, 2005.

[9] Gorawski M.: Zaawansowane hurtownie danych. Rozprawa habilitacyjna. Politechnika Śląska, 2009 (w druku), s. 1-360.

[10]Gorawski M., Chrószcz A.: Prototypowy język zapytań strumieniowych - StreamAPAS v2.0. Studia Informatica 2008 vol. 29 nr 2A. 2008, s. 5-22.

[11]Gorawski M., Kołodziej W.: Znacznie rozproszona sieć przetwarzania strumieniowego. Studia

Informatica 2008 vol. 29 nr 1. 2008, s. 61-85.

[12]Gorawski M., Marks P.: Distributed stream processing analysis in high availability context. The Second International Conference on Availability, Reliability and Security. ARES 2007, Los Alamitos: IEEE Computer Society. 2007, s.61-68.

[13]Gorawski M., Marks P.: Fault-tolerant distributed stream processing system. Seventeenth International Conference on Database and Expert Systems Applications. DEXA 2006, Los Alamitos: Institute of Electrical and Electronics Engineers, IEEE Computer Society. 2006, s. 395-399.

[14]Gorawski M., Marks P.: Towards automated analysis of connections network in distributed stream processing system. Database systems for advanced applications. DASFAA 2008, Springer, Lecture Notes in Computer Science. vol. 4947. 2008, s. 670-677.

[15]Gorawski M., Marks P., Gorawski M.J.: Collecting data streams from a distributed radio-based measurement system. [red.] R. Kotagiri, V. Pudi J. R. Haritsa. Database systems for advanced applications. DASFAA 2008. Springer,Lecture Notes in Computer Science. vol. 4947. 2008, s. 702-705.

[16] Gorawski M., Skierski A.: Okienkowy język zapytań ciągłych. [red.] T. Morzy, M. Gorawski, R. Wrembel Technologie przetwarzania danych. TPD 2007,, Poznań,, 2007, s.480-490.

[17] Gorawski M., Wolany P.: Integracja systemu rozproszonego przetwarzania strumieni danych z systemem agentów programowych.. [red.] St. Kozielski [i in.]. Bazy danych., WKŁ 2007, s.147-154.

[18]Jiang and S. Chakravarthy.: Scheduling strategies for processing continuous queries over streams. In BNCOD, p. 16-30, 2004.

[19] Krämer J., Seeger B..: A Temporal Foundation for Continuous Queries over Data Stream. 11th International Conference on Management of Data (COMAD) 2005.

[20] Kremer A., Markowetz B., Seeger M., Cammert,Ch., Pipes H.: A multi-threaded publish-subscribe architecture for continuous queries over streaming data sources. Technical Report 32, Department ofMathematics and Computer Science, University of Marburg, July 2003.

[21] Madden S., Shah M., Hellerstein J.M.: Continuously adaptive continuous queries over streams. In In SIGMOD, pages 49{60, 2002.

[22] Motwani R., Widom J.,Varma, R.: Query processing, resource management, and approximation and in a data stream management system. In: Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003). (2003) 245-256.

[23] Ozsoyoglu, G., Snodgrass, R.T.: Temporal and real-time databases: A survey.IEEE Trans. on Knowl. and Data Eng. 7(4) (1995) 513-532.

[24] Slivinskas, G., Jensen, C.S., Snodgrass, R.T.: A foundation for conventional and temporal query optimization addressing duplicates and ordering. IEEE Trans. On Knowl. and Data Eng. 13,1 (2001) 21-4

[25] Yu S., Atluri V., Nabil A.: Optimizing View Materialization Cost in Spatial Data Warehouses. Lecture Notes in Computer Science 4081, 2006, s. 45÷54.