STUDIA INFORMATICA 2012
Volume 33 Number 2A (105)

Michał WIDERA
Streams Lab

IMPLEMENTACJA ZAPYTAŃ CIĄGŁYCH W STRUMIENIOWYM SYSTEMIE ZARZĄDZANIA DANYMI NA POTRZEBY PRZETWARZANIA SYGNAŁÓW

Streszczenie. Implementacja ciągłych zapytań w systemie strumieniowym wymaga rozwiązania kilku problemów technicznych i podstawowych. W artykule przedstawiono przykład procesu tłumaczenia zapytania oraz algorytm jego realizacji. Opisano szczegóły techniczne wybranej operacji złączenia wraz z przykładem implementacji. W artykule przedstawiono również opracowaną regułę złączenia strumieni nadchodzących ze zmienną szybkością.

Słowa kluczowe: język zapytań ciągłych, przetwarzanie sygnałów, strumieniowe przetwarzanie danych

CONTINOUS QUERY IN DATA STREAM MANAGEMENT SYSTEM FOR SIGNAL PROCESSING PURPOSES

Summary. Continuous query in data stream management system require a few technical details to be presented. This paper present an example of query translation process and realization algorithm. Technical details of streams join operation with example was presented. This paper present also join operation rules for variable stream intensity.

Keywords: continuous query, signal processing, data stream processing

1. Wstęp

Rozwój technologii związanych z wytwarzaniem systemów komputerowych skutkuje wzrostem wymagań dla systemów zarządzania danymi. W szczególności w przypadku systemów wbudowanych, pracujących w rygorze czasu rzeczywistego i przeznaczonych do

przetwarzania sygnałów, poszukiwane jest rozwiązanie dedykowane – strumieniowy system zarządzania danymi na potrzeby przetwarzania sygnałów czasu rzeczywistego [1].

Sposób implementacji ciągłych zapytań jest istotnym elementem konstrukcji strumieniowego systemu zarządzania danymi. Potrzeba istnienia [2] tego typu zapytań została umotywowana dużą liczbą przypadków użycia (np. monitorowanie sieci, analiza ruchu pojazdów itp.). W celu zapewnienia efektywnego przetwarzania tego typu danych przedstawiono kilka systemów wraz z językami zapytań.

Jednym z podstawowych problemów, przed jakim stają twórcy systemu strumieniowego, jest dobór właściwej metody szeregowania zadań oraz sposobu implementacji ciągłych zapytań. Jednak w przypadku systemu zarządzania danymi, przeznaczonego do ciągłego przetwarzania sygnałów w czasie rzeczywistym, przedstawione dotychczas wyniki prac [3] mają ograniczone zastosowanie.

W pierwszej części artykułu przedstawiono istniejący stan wiedzy z zakresu metod implementacji ciągłych zapytań oraz (po krótkim wstępie) – przegląd podstawowych i dynamicznie rozwijających się systemów strumieniowych. W ramach przeglądu ukazano rys historyczny rozwoju sposobów zapisu ciągłych zapytań. Na koniec wskazano na konieczność rewizji strumieniowego modelu danych w aspekcie problemu porównywania systemów strumieniowych.

W punkcie 3 opisano wyniki prac nad własną implementacją ciągłych zapytań. Po krótkim wprowadzeniu, w którym przedstawiono techniczne aspekty implementacji, można odnaleźć opis procesu tłumaczenia zapytania na plan realizacji zapytania. W dalszej części przedstawiono formalny sposób złączenia strumieni danych. Na podstawie wzoru (1), opisującego sposób złączenia strumienia danych, przedstawiono dotychczas niepublikowany i jeszcze nieudowodniony formalnie sposób (wzór (2)), umożliwiający złączenie strumieni danych o zmiennej szybkości napływu.

2. Ciągłe zapytania w systemach strumieniowych

W ostatniej dekadzie przedstawiono kilka rozwiązań strumieniowych systemów zarządzania danymi [4]. Komercyjne produkty tego typu oznaczane są akronimem CEP, oznaczającym system przeznaczony do złożonego przetwarzania zdarzeń (ang. *Complex Event Processing*). Kilka systemów coraz bardziej stabilizuje swoją pozycję na rynku: Oracle CEP, Sybase CEP oraz StreamBase. Rozwiązanie Oracle CEP oparto na wynikach badań zespołu badawczego skupionego wokół projektu STREAM powstałego na uczelni Stanford, system StreamBase stanowi komercjalizację wyników badań zespołu związanego z uczelnią MIT, system Sybase CEP stanowi natomiast efekt syntezy obu poprzednich rozwiązań, przedstawiony przez firmę Sybase na podstawie systemu Coral8.

Dobór definicji strumienia danych ma istotny wpływ na działanie systemu zarządzania danymi. Pomimo przyjętej wspólnej definicji przedstawiono różne zbiory możliwych operacji. W efekcie wystąpiła rozbieżność w wynikach zwracanych przez różne systemy SPE (ang. *Stream Processing Engine*) [4]. Przez SPE rozumiemy typ mechanizmu przetwarzania strumieni danych, zastosowany w danym CEP. Problem pośrednio rozwiązuje tzw. liniowy test drogowy [5] (ang. *Linear Road Benchmark*). Test powstał w celu wykazania większej użyteczności i efektywności systemów strumieniowych w zadaniu monitorowania zdarzeń w stosunku do systemów relacyjnych.

Poszukiwania właściwej formy języka zapytań na strumieniach danych zaowocowały przedstawieniem prostego modelu strumienia danych i stosunkowo skomplikowanego zbioru operatorów zdefiniowanych na tym modelu. Przedstawiony model strumienia danych (s,t), gdzie s oznacza krotkę, a t − czas jej wystąpienia, jest stosowany przez społeczność skupioną nad rozwojem strumieniowych systemów zarządzania danymi w literaturze światowej.

2.1. Graficzna forma zapisu ciągłych zapytań

Zespół badawczy związany z projektem StreamBase, prowadząc prace badawcze, skupił się nad istotą przetwarzania strumieni danych. Pomijając syntaktyczne niuanse, wynikające z wprowadzenia modyfikacji języka zapytań, skupiono się na graficznej formie zapisu planów zapytań na podstawie komponentów graficznych. Środowisko rozwojowe programisty dla systemu StreamBase stworzono w formie rozszerzenia dla platformy Eclipse. Graficzna metoda zapisu planów zapytań jest obecnie jedną z metod współpracy z systemami typu CEP. Podobne środowiska powstały również dla systemów Oracle CEP oraz Sybase CEP.

Zespoły i firmy skupione nad rozwojem systemów typu CEP prowadzą prace mające na celu przedstawienie podobnej funkcjonalności w swoich produktach. Dlatego w trakcie dalszych prac nad systemem StreamBase stworzono język StreamSQL, będący rozszerzeniem istniejącego graficznego rozwiązania w formie alfanumerycznej [6].

2.2. Rozszerzenia istniejących języków zapytań

Pierwsze efekty prac nad postacią alfanumeryczną języka można odnaleźć w pracach związanych z projektem STREAM. Język CQL (ang. *Continious Query Language*) stanowi syntaktyczne i semantyczne rozszerzenie języka SQL [7]. Aby przyswoić semantykę operacji realizowanych w języku CQL, konieczne jest odwołanie do opracowanego modelu. Istniejące operatory algebry relacji określono jako operatory typu relacja-w-relację, a ich implementacja pozostała zgodna ze standardem SQL:1999. Jako rozszerzenie zdefiniowano dodatkowy zbiór operatorów. Określono je jako operatory typu relacja-w-strumień i strumień-w-relację.

Na podstawie tego podziału przedstawiono wiele modyfikacji języka zapytań, umożliwiających operacje na strumieniach danych.

2.3. Metodyka porównania SPE

W ramach prac badawczych zidentyfikowano różnice w odpowiedziach na te same zapytania realizowane w różnych SPE. W ramach badań [4] poddano analizie systemy: STREAM, Coral8 oraz StreamBase. W celu przedstawienia metodyki porównywania zdefiniowano dodatkowy, rozszerzony model strumienia danych [4]. Podobną modyfikację definicji strumienia danych przedstawiono w trakcie prac nad systemem StreamAPAS [8]. Przedstawiony model strumienia danych zawiera dodatkowe informacje o typie oraz o monotoniczności występujących w strumieniu krotek. Po analizie przytoczonych prac można przyjąć, że o ile przyjęty model strumienia danych (s,t) umożliwia zdefiniowanie zbioru operatorów oraz stworzenie systemu zarządzania strumieniami danych, o tyle porównanie lub uszczegółowienie pewnych operacji wymaga rozszerzania tego modelu. Prace badawcze mające na celu przedstawienie ogólnej metodyki porównywania systemów strumieniowych są nadal w toku. Problem jeszcze bardziej komplikuje istnienie deterministycznych metod przetwarzania strumieni danych, na podstawie których prowadzone są prace w ramach systemu na potrzeby przetwarzania sygnałów.

3. Implementacja ciągłych zapytań na potrzeby przetwarzania sygnałów

Przedstawiając sposób implementacji ciągłych zapytań w tworzonym systemie strumieniowym, wymaga się przedstawienia kilku szczegółów technicznych. Kod źródłowy systemu zarządzania danymi tworzony jest w języku C++ za pomocą zbioru bibliotek Boost. Modele implementacji są testowane za pomocą języka Python.

Zapytania przekazywane są do systemu w postaci sformalizowanego, alfanumerycznego języka zapytań. Tłumaczenie realizowane jest przez parser opracowany za pomocą biblioteki Boost/Spirit [9]. W wyniku tego procesu powstaje struktura o charakterze skierowanego grafu acyklicznego − tzw. skierowanego drzewa. Opracowana i przedstawiona algebra [10] umożliwia zastosowanie metod redukcji planów zapytań. W wyniku tej operacji powstaje kilka prostych, acyklicznych skierowanych grafów − tzw. skierowany las. Każde z drzew opisuje plan realizacji zapytania, realizacja każdego planu wymaga dostępu do danych z określonego w planie źródła. W wyniku sekwencyjnej realizacji szeregu operacji zapisanych w planie tworzone są elementy kolejnego strumienia danych.

Jedną z podstawowych różnic pomiędzy istniejącymi systemami strumieniowymi a opracowanym rozwiązaniem jest celowe zaniechanie implementacji operatorów typu relacja-w-strumień. W rozwijanym systemie wprowadzono zbiór operacji strumień-w-strumień, wnioskując, że operacje typu realacja-w-strumień, realizowane w SPE, w przypadku zadania przetwarzania sygnałów i systemu czasu rzeczywistego są zbędne [12].

3.1. Przykład analizy leksykalnej i procesu tłumaczenia zapytania

W celu przedstawienia pierwszej fazy procesu realizacji ciągłego zapytania konieczne jest przedstawienie sposobu tworzenia planu realizacji zapytania. Parser zapytań został stworzony w języku C++ za pomocą biblioteki Boost Spirit. Gramatyka języka zapytań została zapisana w postaci podobnej do gramatyk stosowanych w narzędziach typu YACC lub Bison [9].

Jako przykład procesu tłumaczenia można przedstawić plik tekstowy, zawierający zbiór poleceń:

```
DECLARE pole_A STREAM strumien_A,1
DECLARE pole_B STREAM strumien_B,0.5
SELECT ( pole_A+1 ) * pole_B[1] as Pole_C1, 1 as pole_C2
STREAM strumien_C
FROM strumien_A # strumien_B
```

Pierwsze dwa polecenia mają na celu deklarację dwóch strumieni danych. Jeśli w plikach konfiguracyjnych wymienione strumienie nie zostaną przyporządkowane do fizycznych źródeł danych – system, po zgłoszeniu ostrzeżenia, stworzy strumienie z danymi przypadkowymi.

Trzecie polecenie o składni SELECT-STREAM-FROM jest ciągłym zapytaniem tworzącym nowy strumień danych. Plan realizacji tego zapytania po przetłumaczeniu będzie się składał z następujących elementów:

1. Plan złączenia strumieni:

```
strumien_c:
1: ARGUMENT strumien_A
2: ARGUMENT strumien_B
3: Operacja połączenia strumieni operatorem #
```

2. Plany konstrukcji kolejnych krotek strumienia:

```
strumien_c.pole_C1:
1: PUSH strumien_A.pole_A[0]
2: PUSH 1
3: Operacja arytmetyczna na stosie +
4: PUSH strumien_B.pole_B[1]
5: Operacja arytmetyczna na stosie *

strumien_c.pole_C2:
1: PUSH 1
```

Plan złączenia strumieni danych w projektowanym systemie jest dwuargumentowy. W poprzednich publikacjach przedstawiono metody rozkładu planów realizacji zapytań umożliwiające doprowadzenie dowolnego wyrażenia opracowanej algebry do postaci dwuargumentowej [10].

Wyjaśnienia wymaga zastosowany operator tablicowy, widoczny w odwołaniu do pól strumienia. Odpowiada on przesunięciu w czasie, a uszczegóławiając − przesunięciu w sekwencji nadchodzących krotek. Jego wartość ma bezpośredni wpływ na liczbę krotek koniecznych do buforowania, a będących elementami strumieni składowych wyrażenia strumieniowego.

Plan złączenia strumieni podlega zasadom algebry opartej na teorii liczb [10], plan tworzenia kolejnych pól strumienia danych jest natomiast oparty na zasadach przetwarzania wyrażeń zapisanych w odwrotnej notacji polskiej.

3.2. Implementacja operacji złączenia strumieni danych

Przedstawione złączenie strumieni danych za pomocą operatora # wymaga wyjaśnienia. W literaturze przedstawiono pełną definicję wraz z dowodem poprawności tej operacji [10], jednak dotychczas nie przedstawiono przykładowego sposobu implementacji.

Operację złączenia strumieni operatorem # zdefiniowano w sposób przedstawiony równaniem (1). Kolejne krotki strumienia wynikowego oznaczono jako c_n, krotki strumieni wejściowych przedstawiono we wzorze jako b_n oraz a_n.

$$c_n = \begin{cases} b_{n-\lfloor nz \rfloor} & \lfloor nz \rfloor = \lfloor (n+1)z \rfloor \\ a_{\lfloor nz \rfloor} & \lfloor nz \rfloor \neq \lfloor (n+1)z \rfloor \end{cases}, z = \frac{\Delta_b}{\Delta_b + \Delta_a}, \Delta_c = \frac{\Delta_a \Delta_b}{\Delta_a + \Delta_b} \tag{1}$$

Każdy ze strumieni ma określony schemat oraz stałą szybkość napływu danych: Δ_a oraz Δ_b. Wielkość Δ_c stanowi nową, wyliczoną szybkość napływu danych. Przedstawione równanie (1) stanowi stosunkowo skomplikowany sposób zapisu operacji na napływających danych. Dla celów implementacyjnych o wiele prostszy do przyswojenia jest zapis w postaci algorytmicznej, zapisanej np. w języku Python:

```
from rational import *
R = Rational
from math import floor
A=range(1,40)
B=map(chr,range(ord('a'),ord('z')+1))+map(chr,range(ord('A'),ord('Z')+1))

deltaA = R(2)
deltaB = R(1)

z = deltaB / ( deltaA + deltaB )
for i in range(0,10):
     if floor(i*z)==floor((i+1)*z):
         print B[i-int(floor((i+1)*z))],
     else:
         print A[int(floor(i*z))],
deltaC = (deltaA*deltaB)/(deltaA+deltaB)
```

W wyniku uruchomienia tego programu na ekranie otrzymamy następujący wynik:

```
a b 1 c d 2 e f 3 g
```

W przypadku strumieni danych napływających ze stałą szybkością przedstawiono i udowodniono zbiór operacji złączeń, rozłączeń oraz reguł nimi rządzących. Jednak nadal poszukiwano równań opisujących operacje na danych napływających ze zmienną szybkością. W trakcie prowadzonych prac badawczych odkryto kolejną regułę rządzącą przetwarzaniem strumieni danych.

Jedno z tych równań stanowi o istnieniu szerszego obszaru badawczego. W celu przedstawienia tego równania modyfikacji musi ulec przedstawiony model (s, Δ) strumienia danych [10]. Dotychczasowy parametr Δ, opisujący stały odstęp czasu pomiędzy kolejnymi krotkami, musi zostać przedstawiony w postaci funkcji. Strumień danych zdefiniowano w postaci: (s, $\Delta(n)$).

Dla tak zdefiniowanego strumienia danych równanie (2) realizuje złączenie strumieni danych napływających z różną szybkością. Dodatkowo należy przyjąć założenia dotyczące funkcji $\Delta(n)$. Funkcja $\Delta(n)$ musi być zdefiniowana w postaci: $\Delta(n) = A + B/n$ lub $\Delta(n) = A + n/B$, gdzie A i B stanowią dowolne wymierne stałe, większe od 0.

$$c_n = \begin{cases} b_{n-\lfloor nz(n) \rfloor} & \lfloor nz(n) \rfloor = \lfloor (n+1)z(n+1) \rfloor \\ a_{\lfloor nz(n) \rfloor} & \lfloor nz(n) \rfloor \neq \lfloor (n+1)z(n+1) \rfloor \end{cases}, z(n) = \frac{\Delta_b(n)}{\Delta_b(n) + \Delta_a(n)}, \Delta_c(n) = \frac{\Delta_a(n)\Delta_b(n)}{\Delta_a(n) + \Delta_b(n)} \quad (2)$$

Przedstawiając implementację, możemy pominąć pierwszych pięć wierszy, które są takie same jak w poprzednim przykładzie. Implementacja równania (2) przedstawia się następująco:

```
def deltaA(n):    return R(1)+R(n)/20
def deltaB(n):    return R(1)+R(n)/10

def z(i):  return deltaB(i)/(deltaA(i)+deltaB(i))

for i in range(0,20):
     if int(i*z(i))==int((i+1)*z(i+1)):
         print B[i-int((i)*z(i))],
     else:
         print A[int(i*z(i))],
```

W wyniku uruchomienia tego programu na ekranie otrzymamy następujący wynik:

```
a 1 b 2 c 3 d 4 5 e 6 f 7 8 g 9 10 h 11 12
```

Jak widać, pomimo różnej intensywności napływu danych przedstawiona operacja dokonuje pełnego złączenia dwóch zbiorów. Na chwilę obecną przeprowadzono numeryczne testy poprawności tego twierdzenia. Testy wskazują na poprawność równania (2). W przypadku równania (1) przedstawiono matematyczny dowód poprawności [10], doprowadzając równanie do postaci twierdzenia Fraenkela [11]. W przypadku równania (2) formalny dowód poprawności nadal jest poszukiwany.

3.3. Algorytm przetwarzania planu zapytania

Typowe zapytanie typu SELECT w systemie relacyjnym realizowane jest na podstawie algorytmu zbliżonego do następującego:

1. Przetłumacz zapytanie.
2. Stwórz lub znajdź podobny plan realizacji zapytania.
3. Przeszukaj tablice lub sprawdź dostępne indeksy i użyj ich w celu odnalezienia danych.
4. Pobierz dane z bazy i dokonaj wymaganych obliczeń.
5. Przedstaw wyniki.

W przypadku zapytania ciągłego ogólny algorytm przedstawia się zazwyczaj inaczej. O ile punkty 1 i 2 mogą pozostać bez zmian, o tyle przeszukanie tablicy lub sprawdzenie dostępności danych przez indeks może okazać się bezcelowe, gdyż dane w systemach strumieniowych nie muszą być przechowywane w nieskończoność. Bardzo często obowiązuje bowiem zasada: liczy się to, co jest tu i teraz, a dostęp do danych archiwalnych jest wymagany tylko w przypadku pewnej kategorii rozpoznanych zdarzeń. Dodatkowo mechanizmy dostępu do danych archiwalnych mogą być implementowane poza SPE. Dlatego punkt 3 wymaga stworzenia odmiennego typu indeksu lub wręcz jego porzucenia, pozostawiając jedynie dostęp do danych bieżących. Dalsze punkty 4 i 5 w przypadku ciągłego zapytania realizowane są w martwej pętli.

W niektórych pracach związanych z przetwarzaniem ciągłych zapytań analizowano dostosowanie planów realizacji zapytań w związku ze zmianą intensywności napływu danych lub zmianą priorytetu, wynikającą ze znaczenia pojawiających się zdarzeń. W takim przypadku punkty 2 i 3 również są brane pod uwagę w trakcie realizacji wspomnianej martwej pętli. Jednak w rozważanym systemie, stworzonym w celu przetwarzania sygnałów, nie występuje ten problem, a strumienie danych przetwarzane są przez zasady opisane równaniami teorii liczb.

3.4. Prezentacja przetworzonych danych

Prezentacja przetworzonych danych jest ostatnim krokiem realizacji ciągłego zapytania. Wszystkie poprzednie operacje realizowane są w rygorze czasu rzeczywistego. Takie podejście ma uzasadnienie w przypadku, kiedy dany strumień danych steruje pewnego rodzaju urządzeniem lub procesem.

Jednak prezentacja wyników nie podlega temu rygorowi. Proces tego typu realizowany jest w większości przypadków na komputerze osobistym. Bardzo często strumienie danych prezentowane są w postaci tabelarycznej. Kolejne elementy dopisywane są na dole pewnej tablicy. W przypadku sygnałów taka prezentacja jest dopuszczalna, jednak prezentacja sygnału w większości przypadków wymaga postaci graficznej.

W przypadku systemów wbudowanych prezentacja strumienia danych realizowana jest poprzez obszar w pamięci oznaczony jako „kanał". Dane w „kanale" dostępne są przez wydzielony obszar pamięci, którego zawartość podlega ciągłym zmianom w czasie, zgodnie z napływającymi danymi. Jeszcze inny przykład realizacji sposobu prezentacji danych na podstawie integracji opracowanego systemu strumieniowego z systemem relacyjnym przedstawiono w literaturze [12].

4. Wnioski i podsumowanie

Jedną z podstawowych cech systemu zarządzania strumieniami danych typu CEP jest wspieranie mechanizmu ciągłych zapytań. Aktualnie przedstawiono kilka implementacji różniących się funkcjonalnością oraz syntaktyką. Na chwilę obecną przedstawienie jednolitego standardu zapytań na wzór rozszerzenia SQL:1999 wydaje się być celem trudnym do osiągnięcia, jednak można stwierdzić, że trwają prace w tym kierunku. Problemem stojącym na drodze jest z pewnością brak spójnego aparatu matematycznego, umożliwiającego opis wszystkich zaimplementowanych rozwiązań. Prawie każdy z producentów lub zespołów badawczych zdefiniował własny zbiór operatorów na strumieniach danych. Co więcej, każdy z producentów przedstawił rozwiązanie poprawne i dokładnie opisujące wybrane zjawisko. Poszukiwania wspólnego modelu i wspólnego języka dały początek kolejnym definicjom pojęć podstawowych − np. strumienia danych. Możliwość jednolitej prezentacji wyników badań, które stanowią podsumowanie całego tematu za pomocą jednej teorii lub zbioru metod i zasad przetwarzania danych, coraz bardziej wydaje się trudne do ociągnięcia.

Przedstawione w artykule rozwiązanie, przeznaczone do przetwarzania sygnałów, dodatkowo kompiluje przedstawiony cel badawczy. Od samego początku prace nad strumieniowym systemem zarządzania danymi na potrzeby przetwarzania sygnałów prowadzone są na podstawie różnicowanego modelu strumienia danych. Stosunkowo niewielki niuans, polegający na tym, że wymiar czasu w definicji strumienia zastąpiono różnicą czasu pomiędzy pojawieniem się kolejnych krotek strumienia, umożliwia przedstawienie równań, które opisują operacje na strumieniach danych w sposób deterministyczny, oparty na twardych podstawach teorii liczb. Nie spotykano podobnego rozwiązania w żadnej z dotychczas opublikowanych prac. Co więcej, w wyniku przeprowadzonych i przytoczonych w tym artykule wyników badań można stwierdzić, że istnieją kolejne, jeszcze do końca nieodkryte reguły, pozwalające łączyć strumienie danych napływające ze zmienną częstotliwością. Zadnie polegające na odkryciu dowodów i kolejnych równań dla przedstawionego problemu stanowi dalszy, otwarty obszar badawczy.

BIBLIOGRAFIA

1. Kawashima H.: KRAFT: A Real−Time Active DBMS for Signal Streams. Int. Conf. on Networked Sensing Systems, 2007, s. 163÷166.
2. Stonebraker M., Cetintemel U.: One Size Fits All: An Idea Whose Time Has Come and Gone. CIDR, 2007, s. 2÷11.
3. Golab L., Özsu M. T.: Issues in Data Stream Management. SIGMOD Record, Vol. 22, 2003, s. 5÷14.
4. Botan I., Derakhshan R., Dindar N., Haas L., Miller R. J., Tatbul N.: SECRET: A Model for Analysis of the Execution Semantics of Stream Processing Systems. VLDB Journal, (3)1-2, 2010, s. 232÷243.
5. Arasu A., Cherniack M., Galvez E. F., Maier D., Maskey A., Ryvkina E., Stonebraker M., Tibbetts R.: Linear Road: A Stream Data Management Benchmark. VLDB Journal, 2004, s. 480÷491.
6. StreamSQL Guide. StreamBase White Paper, 2010.
7. Arasu A., Babu S., Widom J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. VLDB Journal, Vol. 15, No. 2, 2006, s. 121÷142.
8. Gorawski M., Chrószcz. A.: StreamAPAS: Query Language and Data Model. Complex Intelligent Systems and Their Applications, Vol. 41, Springer, New York 2010, s. 187÷205.
9. Cybulka J., Jankowska B., Nawrocki J. R.: Automatyczne przetwarzanie tekstów AWK Lex YAC. Wydawnictwo Nakom, Poznań 2002.
10. Widera M.: Deterministic method of data sequence processing. Annales UMCS Sectio AI Informatica, Vol. IV, 2006, s. 314÷331.
11. Fraenkel A. S.: The bracket function and complementary sets of integers. Canad. J.Math, Vol. 21, 1969, s. 6÷27.
12. Widera M.: Integracja strumieniowego i relacyjnego systemu zarządzania danymi. Studia Informatica, Vol. 32, No. 2A(96), Wydawnictwo Politechniki Śląskiej, Gliwice 2011, s. 89÷102.

Wpłynęło do Redakcji 15 stycznia 2012 r.

Abstract

Continuous query in data stream management system is the one of main features and a few technical details need to be presented. The first part of this paper contains a short re-

view of existing continuous query implementations. Methods of continuous query notation was also reviewed. An example of query translation and implementation algorithm is quoted. Query plan internal structure was presented. There is also a short query results presentation methods review. And finally technical details of streams join operations with attached example was presented.

Deterministic methods of stream processing are the essential issue in developed data stream management system for signal processing tasks. Developed methods are based on number theory. Two stream join equations are presented in this paper. First equation (1) was proved by author in previous publications by leading to Fraenkel Theorem. The second one, thesis (2) still require to be proved formally. This (2) join operation rule actually work for specific variable stream intensity and was checked numerically. Presented theorems and results are suggesting that there are unexplored deterministic methods of stream operations.

Adres

Michał WIDERA: Streams Lab, ul. Reymonta 56/1, 41-800 Zabrze, Polska,
michal@streams.pl.