STUDIA INFORMATICA 2009
Volume 30 Number 2A (83)

Michał WIDERA
Politechnika Śląska, Instytut Informatyki

ZASTOSOWANIE STRUMIENIOWEGO SYSTEMU ZARZĄDZANIA DANYMI W REALIZACJI ZADANIA PRZETWARZANIA SYGNAŁÓW

Streszczenie. W ramach prowadzonych prac badawczych stworzono system zarządzania danymi przeznaczony do zastosowań w medycznych systemach monitorujących. W trakcie dalszych prac podjęto dodatkowy wysiłek mający na celu poszerzenie dziedziny zastosowań. Zaimplementowano rozszerzenie projektowanego języka, umożliwiające przygotowanie zapytania wyznaczającego transformatę Fouriera. Przygotowane rozszerzenie przystosowano do dziedziny przetwarzania sygnałów w czasie rzeczywistym. Dzięki wprowadzonym zmianom zapis szeregu algorytmów występujących w zadaniach przetwarzania sygnałów stał się prostszy. W pracy przedstawiono przykład zastosowania nowych właściwości języka.

Słowa kluczowe: strumieniowe bazy danych, przetwarzanie sygnałów, systemy monitorujące

APPLICATION OF DATA STREAM MANAGEMENT SYSTEM IN SIGNAL PROCESSING TASK

Summary. On medical monitoring systems research a data stream management system was created. Additional effort was taken to spread application domain. The aim was to adapt query language for real-time signal processing domain. The notation of typical algorithms was simplified. This work presents a use case of stream query based on Discrete Fourier Transform.

Keywords: data stream management systems, signal processing, monitoring systems

1. Wstęp

Potrzeba zastosowania systemu zarządzania danymi w przypadku systemów monitorujących dała początek koncepcji strumieniowych baz danych [1]. W wyniku prac badawczych prowadzonych na świecie przedstawiono szereg implementacji dedykowanych rozwiązaniu wybranych klas problemów. W ramach prowadzonych prac badawczych [2] zaprojektowano język oraz zaimplementowano system zarządzania danymi przeznaczony do przetwarzania sygnałów biomedycznych. Omawiany system nadal jest rozwijany na podstawie pojawiających się na bieżąco potrzeb. W międzyczasie opublikowano prace naukowe opisujące podobne rozwiązania na świecie [3, 4].

W początkowej fazie badań projektowano system zarządzania danymi pod kątem zastosowania go w systemach medycznych [2]. Z upływem czasu, prowadząc prace badawcze, rozszerzono zakres jego zastosowań, umożliwiając zapis bardziej ogólnych algorytmów. Główny wysiłek skupiono nad rozwojem deklaratywnego języka zapytań oraz implementacją systemu zarządzania danymi.

Jednym z problemów, jakie napotykamy w trakcie analizy typowych zadań przetwarzania sygnałów, jest zadanie wyznaczenia transformaty Fouriera. Istnieje szereg rozwiązań dedykowanych efektywnemu rozwiązaniu tego typu problemów. Jako przykład można przytoczyć rozwiązania oferowane przez pakiety Octave [5] oraz Matlab. Dysponują one zbiorem funkcji ukrywających szczegóły implementacji. W wyniku zastosowania operacji FFT na wprowadzonej tablicy danych otrzymujemy wynik w postaci zbioru liczb zespolonych. W przypadku zastosowania Octave wystarczy wydać następujący ciąg poleceń:

```
>a=[1,2,3,4]
>b=fft(a)
ans =   10 +  0i   -2 +  2i   -2 +  0i   -2 -  2i
>iffb(b)
ans =   1   2   3   4
```

Jak widać powyżej, rozwiązanie tego problemu okazuje się trywialne w przypadku zastosowania dedykowanego oprogramowania. Podejmując jednak próbę zastosowania systemu zarządzania danymi w celu rozwiązania tego zadania, napotkamy kilka podstawowych problemów. Jednym z nich jest brak typu danych w postaci liczb zespolonych. Nie odnaleziono również w literaturze rozszerzeń abstrakcji systemów relacyjnych umożliwiających bezproblemowe wyznaczanie transformat sygnałów. Dane tego typu muszą być przechowywane w postaci zbioru krotek, zawierających liczby rzeczywiste. Najprostszym sposobem rozwiązania problemu na chwilę obecną jest stworzenie dodatkowych funkcji zdefiniowanych przez użytkownika i połączenia ich z systemem zarządzania danymi [6]. Rozwiązaniom tego typu brakuje elegancji, naturalności oraz efektywności, z jaką mamy do czynienia w przypadku narzędzi dedykowanych.

2. Wyznaczanie transformaty Fouriera

Wyznaczenie transformaty Fouriera jest typowym zadaniem z zakresu przetwarzania sygnałów. Rozwiązanie tego zadania jest bardzo dobrze udokumentowane, aczkolwiek większość najbardziej efektywnych algorytmów jest chroniona prawnie [7]. Dlatego, między innymi, ze względów formalnoprawnych zasadne wydaje się wprowadzenie zapisu deklaratywnego.

W trakcie prac projektowych rozważano możliwość zapisu transformat, podobnie jak to rozwiązano w przypadku pakietów Octave czy Matlab. Projektując język, poszukiwano rozwiązania umożliwiającego prezentację przepływu strumienia danych przez algorytm przetwarzania sygnału. Dlatego odrzucono jako przykład użycia najprostsze rozwiązanie polegające na implementacji funkcji FFT z parametrami.

Prosty algorytm wyznaczania transformaty Fouriera zapisany w formie imperatywnej przy użyciu języka C++ został przedstawiony poniżej. Język C++ dysponuje wsparciem dla liczb zespolonych umożliwiając efektywny zapis.

```
#include <complex>
using namespace std ;
typedef long double LDT ;
const LDT pi = 3.14159265358979323846264338327950288419 68;

template < typename T >
complex<LDT> dft( int x , T xTab[] , int argc , int rev )
{
    complex<LDT> sum = 0;
    for ( int k = 0 ; k < argc ; k ++ )
        sum += xTab[k]*exp(complex<LDT>(0, 2*pi*x*k*rev/argc ));
    return sum ;
};

int main(int argc, char* argv[])
{
    LDT xTab [] = { 1, 2, 3, 4 };
    complex<LDT> xDft [] = { 0, 0, 0, 0 };
    complex<LDT> IDft [] = { 0, 0, 0, 0 };
    int k ;
    int NN = sizeof(xTab)/sizeof(LDT) ;
    for ( k = 0 ; k < NN ; k ++ ) xDft[k] = dft(k, xTab, NN, 1) ;
    for ( k = 0 ; k < NN ; k ++ )
        IDft[k] = dft(k, xDft, NN, -1) / complex<LDT>( NN ) ;
    return 0;
}
```

Pomimo faktu, że język C++ dysponuje wsparciem dla liczb zespolonych, natrafimy na trudne do wychwycenia na pierwszy rzut oka błędy i zagrożenia. W programie pojawią się błędy wynikające z zaokrągleń typu liczb rzeczywistych. Po uruchomieniu programu w tabelach xDft oraz IDft zostaną umieszczone następujące wartości:

```
xDft[0] = 10+0j
xDft[1] = -2-2j
xDft[2] = -2+9.79717e-016j
```

```
xDft[3] = -2+2j

IDft[0] = 1+5.55112e-016j
IDft[1] = 2+2.77556e-016j
IDft[2] = 3+1.11022e-016j
IDft[3] = 4-2.77556e-016j
```

Korzystając z języków tego typu, możemy w większości wypadków efektywnie zapisać algorytm, jednak elegancja i prostota pozostawia pole do dalszych poszukiwań. Dodatkowo tworząc podobne rozwiązania pozostaje podjąć trud szczegółowego i bardzo czasochłonnego przeanalizowania każdego kroku algorytmu. Możemy też zastosować również gotowy pakiet oprogramowania przeznaczony do tego celu.

3. Wprowadzone rozszerzenia języka zapytań

Rozwijane w ramach prac badawczych strumieniowe środowisko przetwarzania i zarządzania danymi w założeniach ma za zadanie umożliwić eleganckie połączenie metod zarządzania wraz z algorytmami przetwarzania sygnałów. W tym celu opracowano dodatkowe cechy funkcjonalne języka zapytań, umożliwiające zapis algorytmów i danych stosowanych w tego typu zadaniach. Na pierwszy plan wysuwa się konieczność implementacji typu danych, która umożliwi przetwarzanie liczb zespolonych. W kolejnym kroku konieczna jest implementacja dodatkowych operacji umożliwiających zmianę postaci strumienia.

3.1. Agregacja i serializacja oraz dodatkowe operatory

Jednym z rozszerzeń projektowanego strumieniowego języka zapytań jest wspomniane pobieżnie w poprzednich pracach polecenie AGSE [8]. Nazwa polecenia powstała z połączenia początkowych sylab dwóch słów – agregacja oraz serializacja. Polecenie umożliwia zmianę formy oraz szybkości napływu strumienia danych. Opisywane w literaturze algorytmy oparte na ruchomym oknie danych [9] mogą zostać zaimplementowane przy użyciu tego polecenia. Jako przykład użycia polecenia AGSE można przedstawić następujący ciąg przetwarzania strumienia danych. Zakładając istnienie strumienia danych „Zrodlo” o schemacie danych w postaci jednego pola zawierającego liczby nadchodzące z szybkością jednej wartości na sekundę, wydanie następującego polecenia:

```
SELECT AGSE( Zrodlo(0) , number<4> , 4 )
FROM Zrodlo
AS CoCztery;
```

utworzy strumień danych „CoCztery”, którego krotki będą pojawiać się raz na cztery sekundy i zawierać cztery kolejne liczby ze strumienia „Zrodlo”.

Wydanie kolejnego polecenia:

```
SELECT AGSE( (CoCztery(0),CoCztery(1),CoCztery(2),CoCztery(3)), number<1>, 1)
FROM  CoCztery
AS PostacPierwotna;
```

Umożliwia utworzenie strumienia, pod nazwą „PostacPierwotna”, równoważnego ze strumieniem „Zrodlo”. Przez określenie równoważny rozumiemy, że kolejne krotki obu strumieni przechowują te same wartości danych oraz szybkość napływu obu strumieni danych jest identyczna.

W odróżnieniu od poprzednio przedstawionej formy polecenia [8] – wprowadzono konieczność odwoływania się do pola przez jego indeks. Z uwagi na fakt, że operator w postaci nawiasów kwadratowych „[ ]” został zarezerwowany dla potrzeb oznaczania pozycji krotki względem początku strumienia, do oznaczenia pozycji elementu wewnątrz krotki zastosowano operator nawiasowy „( )”.

3.2. Wymierne liczby zespolone

W przypadku języków rodziny SQL mamy do czynienia ze ścisłym typowaniem. Deklaracja typu pola ściśle wiąże pole z danym typem. Z początku rozważano dodanie dodatkowego typu danych do istniejącego zbioru. Jednak w trakcie prac projektowych postanowiono zmienić zasady rządzące tą częścią semantyki języka. Jako podstawowy i jedyny sposób przechowywania i przetwarzania liczb przyjęto typ danych w postaci wymiernych liczb zespolonych. Usunięto wszystkie typy dla liczb rzeczywistych oraz całkowitoliczbowych. Funkcjonalność usuniętych typów zapewniono przez algorytmy domyślnej prezentacji i konwersji danych.

Wprowadzony wymierny i zespolony typ danych stanowi agregat czterech liczb całkowitych. Pierwsze dwie liczby opisują ułamek wymierny części rzeczywistej, drugie dwie liczby – części urojonej. Do obsługi tak zdefiniowanego typu danych przygotowano szereg procedur odpowiedzialnych za podstawowe operacje arytmetyczne oraz mechanizmy składowania. Dodatkowym problemem okazała się nierozwiązana kwestia konwersji danych przedstawionych w postaci rzeczywistej do postaci wymiernej. Intuicyjnie jest to problem bardzo prosty, jednak nie odnaleziono efektywnego rozwiązania wewnątrz ogólnodostępnych bibliotek numerycznych [10]. W celu rozwiązania tego problemu zastosowano własną implementację funkcji konwersji, która wprowadziła możliwie najmniejszy błąd zaokrągleń w obliczeniach.

4. Przykład zapytania o transformatę Fouriera

Zapytanie o wyznaczenie transformaty Fouriera nie należy do zbioru najprostszych przykładów zapytań realizowanych w projektowanym systemie strumieniowym. Wynika to z faktu, że do wyznaczenia wyniku konieczne są odwołania do wartości historycznych. W przypadku klasycznego systemu strumieniowego taka właściwość stanowi komplikację z uwagi na fakt, że dane w takim systemie przetwarzane powinny być tylko jednokrotnie [1].

Jednak w przypadku systemów przetwarzających sygnały tego typu ograniczenie nie może mieć miejsca. Konieczne jest więc zapewnienie wewnątrz systemu efektywnego mechanizmu buforowania oraz usuwania zbędnych danych. Dzięki temu założenie dotyczące jednokrotnego przetwarzania danych można uogólnić dla danych znajdujących się w wybranym oknie czasowym. Poniżej znajduje się przykład zapisu zapytania o transformatę Fouriera w deklaratywnym, strumieniowym języku zapytań:

```
DEFINE STREAM Src( number x ) , 1  ;

DEFINE FUNCTION WK( number x ) :
    RETURN EXP( 0 , PI * 2 * x / 4 )

SELECT
       Src[0].x*WK(0)+Src[1].x*WK(0)+Src[2].x*WK(1)+Src[3].x*WK(0),
       Src[0].x*WK(0)+Src[1].x*WK(1)+Src[2].x*WK(2)+Src[3].x*WK(3),
       Src[0].x*WK(0)+Src[1].x*WK(2)+Src[2].x*WK(4)+Src[3].x*WK(6),
       Src[0].x*WK(0)+Src[1].x*WK(3)+Src[2].x*WK(6)+Src[3].x*WK(9)
FROM STREAM
STEP 4
AS Dft ;

SELECT AGSE( ( Dft(0),Dft(1),Dft(2),Dft(3) ) , number<1> , 1 )
FROM Dft
AS Ftr ;

SELECT
       (Src[0].x*WK(0)+Src[1].x*WK(0) +Src[2].x*WK(1) +Src[3].x*WK(0) )/4,
       (Src[0].x*WK(0)+Src[1].x*WK(-1)+Src[2].x*WK(-2)+Src[3].x*WK(-3))/4,
       (Src[0].x*WK(0)+Src[1].x*WK(-2)+Src[2].x*WK(-4)+Src[3].x*WK(-6))/4,
       (Src[0].x*WK(0)+Src[1].x*WK(-3)+Src[2].x*WK(-6)+Src[3].x*WK(-9))/4
FROM Ftr
STEP 4
AS IDft ;

SELECT AGSE( ( IDft(0),IDft(1),IDft(2),IDft(3) ) , number<1> , 1 )
FROM  IDft
AS SrcEq ;
```

W pierwszym wierszu zdefiniowano strumień danych nadchodzący z przykładowego urządzenia. W przytoczonym przykładzie strumień Src zawiera ciąg krotek nadchodzących z częstotliwością jednej krotki na sekundę.

W kolejnym wierszu zdefiniowano funkcję. Funkcje definiowane stanowią znane i często spotykane rozszerzenie deklaratywnych języków zapytań wewnątrz dostępnych systemów zarządzania danymi.

Pierwsza instrukcja SELECT tworzy strumień z danymi dyskretnej transformaty Fouriera. Źródłem współczynników występujących w obu zapytaniach są macierze przekształceń uzyskane na podstawie algorytmu zapisanego w języku C++. Na uwagę zasługuje wprowadzona dodatkowa klauzula STEP 4 występująca w poleceniu SELECT. Klauzula STEP zapewnia przesunięcie ruchomego okna danych z zadanym krokiem nad strumieniem danych.

Kolejna instrukcja zmienia postać strumienia za pomocą operacji serializacji. Tak przygotowany strumień danych zamieniany jest ponownie do postaci źródłowej, przez zapytanie realizujące odwrotną transformatę Fouriera. W celu uzyskania równoważnej postaci strumienia IDft ze strumieniem Src konieczne jest wprowadzenie kolejnej serializacji.

5. Wnioski i podsumowanie

Dotychczas projektant, rozważając możliwość zastosowania systemu zarządzania danymi w systemie monitorującym, miał stosunkowo niewielki wybór. Rozwijana koncepcja systemów strumieniowych zapowiada znaczącą zmianę stanu faktycznego w tym aspekcie zastosowania systemów baz danych.

Analizując aktualne publikacje naukowe z zakresu baz danych, można dostrzec dyskusję związaną z podstawami teoretycznymi. Podważany jest podstawowy paradygmat modelu relacyjnego, zakładający jego uniwersalność w aspekcie zastosowania [11]. W odpowiedzi pojawiły się prace podejmujące polemikę [12]. W ramach tej dyskusji, niniejsza praca dostarcza dodatkowych argumentów zwolennikom strumieniowego modelu danych. Prowadzona dyskusja dotycząca tego aspektu zastosowania wymaga z pewnością głębszego zastanowienia i analizy. Próba implementacji przedstawionych w pracy zagadnień na podstawie modelu relacyjnego napotka na bariery efektywności, wydajności oraz przejrzystości – stanowi więc argument przemawiający na korzyść zwolenników modelu strumieniowego.

W pracy przedstawiono przykład zastosowania strumieniowego języka zapytań związanego z dziedziną przetwarzania sygnałów. Niektóre z elementów przedstawionego systemu od kilku lat funkcjonują we wdrożonych medycznych systemach monitorowania [2]. Prace projektowe związane z omawianym systemem i językiem w postaci niezależnego rozwiązania są na ukończeniu. Obecnie prowadzone są intensywne prace wdrożeniowe oraz implementacja. Docelowo przewidziano udostępnienie rozwijanego oprogramowania szerszemu gronu naukowców w celach badawczych przez przygotowywany serwis internetowy.

Grupą systemów, dla których przewidziano zastosowanie projektowanego rozwiązania, są systemy monitorowania. Przewiduje się, że zastosowanie systemu zarządzania danymi w tych systemach będzie pomocne w uproszczeniu ich konstrukcji. Uproszczeniu ulegną

również przechowywane algorytmy. Dzięki temu o wiele prostsza będzie ich konserwacja jak i usuwanie błędów.

Przedstawione zapytanie o transformatę Fouriera jest klasycznym przykładem zastosowania projektowanego języka i systemu zarządzania danymi. Wprowadzony, nowy typ danych oraz właściwości języka z pewnością ułatwią zapis i przetwarzanie danych w porównaniu z istniejącymi systemami zarządzania danymi. Podjęty wysiłek rozszerzenia funkcjonalności dał w efekcie możliwość zapisu szerszej gamy przekształceń.

BIBLIOGRAFIA

1. Golab L., Özsu M.T.: Issues in Data Stream Management. SIGMOD Record, 22, 2003, s. 5÷14.
2. Widera M., Wrobel J., Owczarek A., Matonia A., Jezewski M.: Data Management System for Computer Aided Biophysical Monitoring. IEEE EMBS, 2005, s. 4712÷4715.
3. Kawashima H., KRAFT: A Real-Time Active DBMS for Signal Streams. Int. Conf. on Networked Sensing Systems, 2007, s. 163÷166.
4. Chen C., Agrawal H., Cochinwala M., Rosenbluth D.: Stream Query Processing for Healthcare Bio-sensor Applications. Int. Conf. on Data Engineering, 2004, s. 791÷794.
5. Eaton J.W.: GNU Octave Manual. Network Theory Limited, 2002.
6. Ben-Gan I.: Elegant Solutions to Complex Problems. SQL Server Magazine, 2000.
7. Efficient implementation of large size FFT, United States Patent 7280469.
8. Wrobel J., Widera M., Horoba K., Jezewski J., Winiarczyk R.: Declarative algebra and continuous query language for biomedical stream processing in fetal monitoring system. IEEE EMBS 2004, s. 3175÷3178,
9. Babcock B., Datar M., Motwani R., O'Callaghan L.: Sliding Window Computations over Data Streams. Technical Report, 2002.
10. Zagala M.: Arytmetyka liczb wymiernych, SKI 2006.
11. Stonebraker M., Cetintemel U.: “One Size Fits All”: An Idea Whose Time Has Come and Gone, CIDR 2007, s. 2÷11.
12. Hoppe A., Gryz J., Stream Processing in a Relational Database: a Case Study, IDEAS 2007, s. 216÷224.

Recenzent: Dr hab. Zygmunt Mazur, prof. Pol. Wrocławskiej

Wpłynęło do Redakcji 31 stycznia 2009 r.

Abstract

The need of a data stream management system with the capability of querying continuous data streams is well understood. During worldwide scientific research there was presented a series of dedicated implementation that can handle solve some classes of problems. In this research a medical monitoring system with databases management system was developed. Additional effort was taken to spread an application domain. A notation for general-purpose signal processing queries was implemented. This extension was achieved by introduced changes in query-language and database system. Introduced changes of notation supports algorithm that can lead pattern recognition and filtering. The notation became simplified and more intuitive. Goal was reach by adapting language to domain-specific area of signal processing. Presented results of DFT query and additional examples of use-case causes that this system can be used in signal processing area.

Adres

Michał WIDERA: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, michal@widera.int.pl.