STUDIA INFORMATICA 2008
Volume 29 Number 2A (76)

Marcin GORAWSKI, Aleksander CHRÓSZCZ
Politechnika Śląska, Instytut Informatyki

PROTOTYPOWY JĘZYK ZAPYTAŃ STRUMIENIOWYCH – STREAMAPAS v2.0

Streszczenie. Przedstawiony zostanie prototypowy język zapytań strumieniowych StreamAPAS v2.0 oraz system przetwarzania strumieniowego. Składnia języka StreamAPAS wspiera struktury hierarchiczne, które w czytelny sposób grupują atrybuty oraz reprezentują dane przestrzenne. Jednym z celów utworzonego systemu jest udostępnienie indeksów w przetwarzaniu strumieniowym. Wiąże się to z dodaniem nowych typów kolekcji krotek reprezentujących okna czasowe oraz rozbudową interfejsu funkcji. Rozwiązanie takie pozwala w prosty sposób zmieniać zbiór dostępnych funkcji, dzięki czemu łatwiej dostosować system przetwarzania strumieniowego do zmieniających się potrzeb aplikacji.

Słowa kluczowe: drzewa atrybutów, podejście funkcjonalne, przetwarzanie strumieniowe, CQL, synchronizacja strumieni, okna czasowe, algebra temporalnych operatorów

STREAM QUERY LANGUAGE – STREAMAPAS v2.0

Summary. The following paper introduces a new stream query language StreamAPAS v2.0 and the continuous processing system. The language syntax supports hierarchical data structures which offer grouping attributes and a better representation of spatial data. The paper describes also the extension of the data collections which enables the stream processing nodes to use indexes. The language syntax bases on the functional approach in order to simplify embedding new indexes and new functionality into the system. The above features make continuous processing applications easier to develop and maintain.

Keywords: attributes tree, functional approach, stream processing, CQL, streams synchronization, time windows, temporal logical operator algebra

1. Wprowadzenie

Systemy przetwarzania strumieni danych (w skrócie: systemy strumieniowe) znacząco różnią się od tradycyjnych systemów bazodanowych, w których dane są najpierw zapisywane, następnie wyliczane indeksy, na koniec wyznaczane odpowiedzi na zgromadzonych zasobach. W motywującym nasze badania systemie monitorowania i sterowania zużyciem mediów [1] źródła generują strumienie danych nieograniczonych rozmiarów, które są na bieżąco przetwarzane przy użyciu ciągłych (długookresowych) zapytań. Przykładowym celem użycia takiego systemu jest wyznaczanie statystyk.

Systemy przetwarzania strumieni danych wyróżniają następujące cechy [2]:

- Ciągłe przetwarzania – w systemach DBMS obowiązkiem użytkownika jest zadawanie zapytań na zgromadzonych danych (np. „czy stopa zwrotu osiągnęła zadany pułap„). W systemie strumieniowym użytkownik uruchamia długookresowe zapytanie (np. „powiadom mnie, gdy stopa zwrotu osiągnie zadany pułap”).
- „Wypychanie” danych – źródła danych oraz węzły przetwarzające strumienie przekazują krotki kolejnym obiektom w systemie. W DBMS klienci aktywnie wywołują zapytania w celu aktualizacji wyników, podczas gdy w systemie strumieniowym klient oczekuje na nie pasywnie.
- Krótki czas latencji – wiele systemów strumieniowych monitoruje zachodzące zjawiska przy rygorze krótkiego czasu odpowiedzi. Przykładowo, dla systemu nadzorującego ruch uliczny ważniejsza jest bieżąca informacja o stanie drożności ulic niż umiejscowienie wystąpienia „korków” kilka godzin wcześniej.
- Potokowość – przetwarzanie danych bazuje na nieblokowalności, własność ta pozwala na wydajną współbieżną pracę węzłów sieci przetwarzania.
- Skalowalność – każdy węzeł stanowi samodzielną jednostkę, co ułatwia umieszczenie sieci przetwarzającej w środowisku rozproszonym.
- Wysoka przepustowość – system strumieniowy nie gromadzi danych, tylko je przetwarza.

Warunki oraz cel działania systemu strumieniowego odbiegają od tradycyjnych systemów zarządzania bazami danych, np. RDBMS, przez co proste wzbogacenie języka SQL o okna czasowe jest rozwiązaniem niezalecanym z punktu widzenia zastosowań praktycznych.

Większość systemów strumieniowych to systemy dedykowane, przez co ich język zapytań jest silnie związany z konkretną dziedziną. Chcąc uczynić takie systemy uniwersalnymi (przydatnymi do wielu innych zastosowań), należy zbudować platformę z uniwersalnym językiem zapytań strumieniowych.

Głównym zastosowaniem systemów strumieniowych to obliczenia analityczne, które korzystają z indeksów. Ponieważ stale powstają nowe rozwiązania indeksowania, zrezygnowa-

no z ich obsługi zaszytej w składni języka, w zamian rozszerzono interfejs dołączania nowych funkcji. Możliwe jest wtedy rozszerzenie funkcjonalności poprzez załadowanie modułu definiującego nowe operacje aniżeli konstruowanie nowej wersji języka.

Zaproponowany język zapytań strumieniowych StreamAPAS v2.0 ogranicza listę dostępnych funkcji poprzez słowa kluczowe zaszyte w składni języka (z czym spotykamy się w SQL). Alternatywnie użyto podejścia wzorowanego na językach programowania, gdzie zbiór dostępnych funkcji zależy od dołączonych bibliotek. Rozwiązanie takie pozwala uniknąć sytuacji, w której na potrzebę dodania nowych funkcji należy zmieniać składnię języka.

Kolejnym założeniem języka jest opis krotki jako drzewa atrybutów (podobnie jak w XML). Uzasadnione jest to tym, że dane hierarchiczne łatwiej obrazują struktury analityczne. Rozwiązanie takie również upraszcza zapis argumentów funkcji.

Utworzony system jest kontynuacją prac badawczych nad językiem CQL [1], prowadzonych przez Zespół Algorytmów, Programowania i Systemów Autonomicznych w Zakładzie Teorii Informatyki Instytutu Informatyki Politechniki Śląskiej.

2. Podstawowe pojęcia

Niech $I := \{[t_s, t_e) \in T \times T \mid t_s \le t_e\}$ oznacza zbiór przedziałów czasowych włącznie z punktami czasowymi, gdzie T jest dziedziną czasu.

Definicja 1.

Strumień opisuje para symboli $S = (M, \le_{t_s, t_e})$, gdzie:

M – nieskończony strumień krotek $(type, e, [t_s, t_e))$, gdzie: *type* - typ krotki, e – dane transportowane przez krotkę $[t_s, t_e) \in I$,

$\le_{t_s, t_e}$ – uporządkowanie leksykograficzne elementów strumienia M (najpierw po t_s, potem po t_e).

W StreamAPAS v2.0 rozróżniamy kilka typów krotek (tab. 1), przez co możliwe jest zarządzanie siecią przetwarzania przy ograniczonej liczbie dodatkowych strumieni.

Logika StreamAPAS v2.0 pozwala modelować okna czasowe poprzez przypisanie każdej krotce czasu życia [3]. Niech $w \in T$, operator okna czasowego ω definiujemy: $\omega_w : S \times T \to S$. Pozwala to utworzyć między innymi okna:

infinity – czas życia krotki ustawiany jest na nieskończoność; krotka $(e, [t_s, t_e))$ jest przetwarzana do postaci $(e, [t_s, \infty))$.

range – tworzy okno przesuwające się; krotka $(e,[t_s,t_e))$ jest przetwarzana do postaci $(e,[t_s,t_s+w))$, gdzie w – czas życia krotki.

fixed – dziedzina czasu zostaje podzielona w sekcje o stałym rozmiarze w. Sekcja rozpoczyna się w chwili $i*w$, gdzie $i \in N_0$, a kończy $(i+1)*w$. Dla każdej krotki $(e,[t_s,t_e))$ wyznaczany jest nowy znacznik $t_e = i*w$, będący liczbą najbliższą t_s spełniającą warunek $t_s < t_e$.

Tabela 1

Typy krotek w systemie

Type	Opis
HEADER	definiuje metaschemat krotek INSERTION
REMOVE	informuje o usunięciu strumienia z systemu
STOP	wstrzymuje działanie operatorów
INSERTION	służy transmisji wartości atrybutów
BOUNDARY	znacznik upływu czasu

Jeżeli nie znamy czasu życia krotki źródłowej, należy go ustawić na zero, w przeciwieństwie do zaproponowanej w pracy [3] nieskończoności. Założenie to nie zmienia funkcjonalności, ponieważ dysponujemy operatorami okien czasowych, ale zabezpiecza przed nieumyślną budową okien o nieskończonym rozmiarze.

3. Podstawowe operatory

Logika operatorów algebry bazuje na algebrze zaproponowanej w [3]. Aktualnie system wspiera operatory: filtr (σ), mapa (μ), łączenie ($\triangleright\triangleleft$), okno czasowe ($\omega$), unia ($\bigcup$) i agregacja ($\alpha$). Ponieważ w dalszej części przedstawione zostaną operatory algebry, które są zdefiniowane tylko na krotkach typu INSERTION, pominięty zostanie element *type* w definicji krotki.

Filtr

Niech P jest zbiorem predykatów filtracji takich, że $p \in P \quad p:(\Omega \times T) \rightarrow \{true, false\}$. Filtr $\sigma : S \times P \rightarrow S$ zwraca krotki, dla których predykat ma wartość pozytywną. W systemie strumieniowym operator filtracji definiujemy:

$$\sigma_p(S) := \{(e,t) \in S \mid p(e,t)\} \tag{1}$$

Mapa

Niech F_{map} jest zbiorem wszystkich funkcji mapujących takich, że $f \in F_{map}$ $f : \Omega \to \Omega$; warto zauważyć, że f może być funkcją n-argumentową, ponieważ Ω jest uniwersum atrybutów krotki. Operator mapy $\mu_f : S \times F_{map} \to S$ dla systemu strumieniowego definiujemy:

$$\mu_f(S) := \{(e,t) \mid (\hat{e},t) \in S \mid e = f(\hat{e})\} \quad (2)$$

Należy zauważyć, że ogólność definicji operatora mapy pozwala na realizowanie przy jego użyciu również operacji projekcji atrybutów.

Produkt kartezjański

Produkt kartezjański $\times : S \times S \to S$ dla systemu strumieniowego definiujemy jako:

$$\times(S_1, S_2) := \left\{ \begin{array}{l} (\circ(e_1, e_2), t) \mid (\exists(e_1, t_1) \in S_1 \wedge \exists(e_2, t_2) \in S_2 \wedge \\ \qquad \wedge\, intersect(t_1, t_2)), t = overlap(t_1, t_2) \end{array} \right\} \quad (3)$$

Operator łączy ze sobą tylko te krotki, których czas życia nachodzi na siebie. Wynikowa krotka otrzymuje czas równy części wspólnej czasów życia łączonych krotek.

Operacja reorganizacji

Mechanizm ten jest wyzwalany przez kolejne krotki przekazywane węzłowi na wejście. Przetworzenie każdej z nich poprzedza wywołanie operacji reorganizacji, która usuwa krotki, których czas życia upłynął wewnątrz kolekcji węzła [3].

Niech $S_1,...,S_n \in S$, gdzie $n \in N$ są strumieniami wejściowymi operatora. Na potrzebę operacji reorganizacji przechowujemy najstarsze znaczniki czasu t_{s_j}, gdzie $j \in \{1,...,n\}$ krotek kolejnych strumieni wejściowych oczekujących na przetworzenie. Każda krotka $(e,[t_s,t_e))$ może zostać bezpiecznie usunięta ze struktur wewnętrznych operatora, jeżeli jej t_e jest mniejsze lub równe $\min\{t_{s_j} \mid j \in \{1,...,n\}\}$.

Wywołanie reorganizacji przed wykonywaniem właściwej operacji węzła pozwala przyspieszyć wyliczania wyników, ponieważ operator nie musi wtedy sprawdzać krotek, co do których mamy pewność, że nie będą tworzyły wyniku.

3.1. Uporządkowanie krotek strumieni wejściowych

Definicja 2.

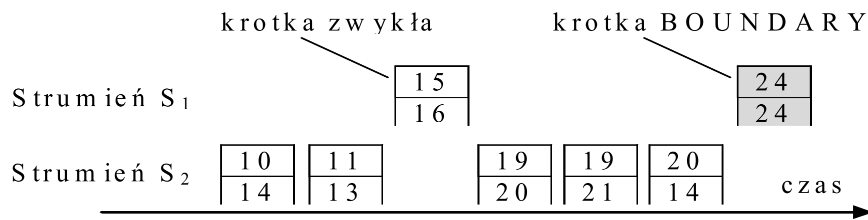
Prefiks sekwencji krotek jest podsekwencją krotek rozpoczynająca się od najstarszej kończąc na krotce ze znacznikiem $[t_s,t_e)$ (korzystamy z porządku leksykograficznego).

Operatory zasilane kilkoma strumieniami wejściowymi wymagają wstępnego uporządkowania krotek, zgodnie z porządkiem leksykograficznym. Dzięki powyższej operacji gwaran-

tujemy jednoznaczność generowanych wyników, ponieważ nie modyfikujemy prefiksu krotek dla $[t_s,t_e)$, który wskazuje bieżący czas węzła.

Aby uniknąć konieczności przesyłania replik krotek w strumieniach reprezentujących zjawiska o małej zmienności w celu utrzymania płynności przetwarzania, wprowadzono krotkę typu BOUNDARY [2]. Pełni ona rolę mechanizmu „bicia serca”. Znacznik czasowy krotki tego typu $(e,[t_s,t_s))$ informuje, że kolejne elementy w strumieniu nie będą posiadały dat wcześniejszych. Podobnie jak dla operacji reorganizacji, przechowujemy najstarsze znaczniki $[t_{s_j},t_{e_j})$, gdzie $j \in \{1,...,n\}$ krotek kolejnych strumieni wejściowych oczekujących na przetworzenie. Kolejna krotka pobrana ze strumienia wejściowego i przekazana do dalszego przetwarzania spełnia warunek:

$$[t_s,t_e) \leq_{t_s,t_e} \min\{[t_{s_j},t_{e_j}) \mid j \in \{1,...,n\}\} \quad (4)$$



Rys. 1. Porządkowanie krotek strumieni wejściowych. Liczby w wierszach oznaczają odpowiednio t_s, t_e
Fig. 1. Sorting tuples from input streams. The values in a tuple-box represent t_s, t_e

Po przetworzeniu kolejek wejściowych z rys.1 pozostaną w kolejkach ostanie obiekty. Krotka BOUNDARY pozwoliła na przekazanie do dalszego przetworzenia obiekty o $t_s = 19, 19, 20$. Mechanizm ten rozwiązuje problem zachowania płynności przetwarzania, w przypadku gdy strumienie rzadko nadsyłają krotki z danymi.

Ponieważ liczba strumieni wejściowych węzłów nie ulega zmianie w trakcie działania zapytania oraz krotki są uporządkowane w ramach pojedynczego strumienia, implementację synchronizacji strumieni oparto na sortowaniu poprzez kopcowanie.

3.2. Kolekcje krotek

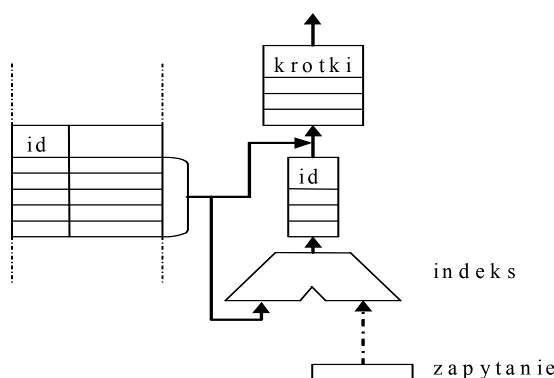
Kolekcje służą przechowywaniu krotek pochodzących z jednego strumienia. Występują one tylko w węzłach przetwarzających, zawierających operatory stanowe. Po uwzględnieniu faktu, że każda krotka ma ustalony czas życia oraz można definiować indeksy na kolekcji, zdefiniowano rozszerzony interfejs kolekcji, w skład którego wchodzi:

1) przeglądanie zawartości, zgodnie z porządkiem leksykograficznym,
2) wstawianie krotek,
3) realizacja operacji reorganizacji,
4) dostęp do krotek poprzez identyfikator numeryczny,
5) zgłaszanie zmian indeksom zbudowanym na danej kolekcji.

Dodanie indeksów do węzłów przetwarzających wiąże się z potrzebą zdefiniowania atrybutu identyfikującego jednoznacznie krotkę. Indeks przechowuje strumień w postaci wstępnie przetworzonej, chcąc odwołać się do atrybutów krotki źródłowej podczas dalszego procesu przetwarzania strumienia, należy dokonać mapowania kluczy wynikowych na odpowiadające im dane źródłowe. Rysunek 3 przedstawia przepływ krotek wewnątrz węzła przetwarzania opartego na indeksie.

Identyfikator jest wartością numeryczną, jeżeli krotka nie posiada atrybutu-identyfikatora, wówczas kolekcja automatycznie nadaje unikalne identyfikatory krotką.

Aby zapewnić spójność danych, aktualizacja indeksu jest realizowana natychmiast, gdy następuje aktualizacja kolekcji, na którym dany indeks jest zbudowany. Wymaga to zarejestrowania indeksu jako obiektu nasłuchującego wydarzeń: update, insert, remove generowanych przez kolekcję.



Rys. 2. Przepływ krotek, gdy kolekcja zasila strukturę indeksującą
Fig. 2. The control flow for an indexing structure

Zaproponowana w [3] algebra przetwarzania strumieniowego przyjmuje, że tylko czas życia krotki decyduje o jej aktywności. Podejście takie stwarza problem, jak obsługiwać krotki, których czas życia nie jest znany w chwili ich utworzenia. Przykładem są rekordy tabel z DBMS, które nie definiują czasu ich usuwania. Stąd powstał pomysł rozszerzenia liczby typów kolekcji. Prezentowany system wspiera kolekcje:

Kolekcja czasowa:

- wstawianym krotką przypisywane są unikalne wartości identyfikatorów,
- czas życia obiektów zależy od wartości atrybutów t_s, t_e krotek,
- elementy kolekcji są usuwane w wyniku wywołania operacji reorganizacji.

Kolekcja czasowo-tabelaryczna:

- obsługuje czas życia krotek analogicznie do kolekcji czasowej,
- elementy kolekcji są usuwane w wyniku wywołania operacji reorganizacji.

Wyróżniamy atrybut krotki, pełniący rolę unikalnego identyfikatora id. W zależności od jego wartości przekazanie krotki do kolekcji skutkuje operacjami:

id krotki nie istnieje w zbiorze – krotka jest dodana do zbioru,

id krotki istnieje w zbiorze – krotka poprzednia jest zastąpiona nową,

id krotki jest liczbą ujemną – atrybut t_e krotki będącej w kolekcji jest zastępowany atrybutem t_e nowej krotki.

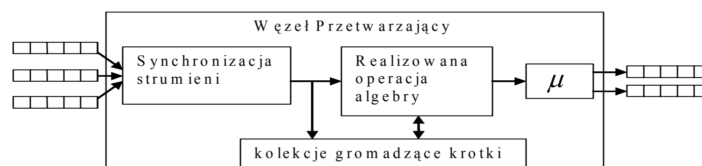
Importowanie rekordów z tabel można zrealizować poprzez skopiowanie wartości atrybutów do krotki, ustawienie znacznika t_s, zgodnie z bieżącym czasem oraz $t_e = \infty$. Operacja usuwania rekordu odpowiada wysłaniu krotki z ujemnym identyfikatorem oraz przekazaniu informacji o czasie usunięcia w atrybucie t_e. Usuwanie elementów z kolekcji jest wykonywane dopiero podczas reorganizacji. Jeżeli zostanie wysłana krotka o identyfikatorze występującym w kolekcji, zastąpiony zostanie poprzedni wpis nowym włącznie z aktualizacją znaczników czasu. Uaktualnianie znacznika t_s podczas każdej operacji na krotce o wskazanym identyfikatorze gwarantuje niezmienniczość prefiksu krotek przetworzonych przez operator (zmiany nie będą obejmować historii).

Ustalony na początku jednoznacznie czas życia krotek prowadzi do własności redukowalności-migawkowej (ang. *snapshot-reducability*). Korzystając z niej, możemy zadania z dziedziny temporalnej przenieść do algebry relacyjnej dla chwili *t*.

Powyższe założenie ogranicza zakres użycia kolekcji czasowo-tabelarycznej: na strumieniu, na którym zdefiniowano taką kolekcję można zbudować tylko pojedynczą operację relacyjną. Gwarantujemy wtedy, że w ramach pojedynczego zapytania strumieniowego tylko na początku drzewa przetwarzania wylicza się czas życia krotek (identyczne kryterium spełniają operatory okien czasowych).

3.3. Architektura węzła przetwarzającego

Aby strumienie wewnątrz pojedynczego zapytania działały wydajnie, każdy z nich transportuje minimalny zbiór atrybutów koniecznych do ukończenia przetwarzania zapytania.



Rys. 3. Przepływ krotek w węźle przetwarzającym
Fig. 3. The control flow in a processing node

Cel ten jest osiągnięty poprzez wprowadzenie operacji mapującej na wyjściu każdego węzła przetwarzającego.

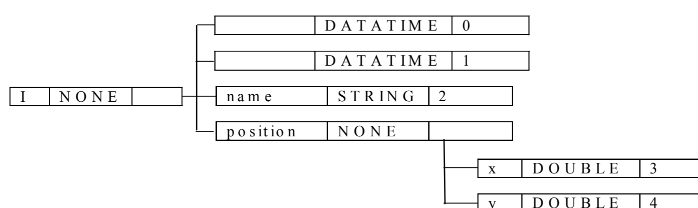
Na rys. 3. przedstawiono przepływ krotek w trakcie wyliczania zapytania. Scenariusz działania jest następujący:

1) uporządkowanie leksykograficzne krotek, jeżeli operator posiada kilka strumieni wejściowych,
2) reorganizacja kolekcji na podstawie znacznika czasu krotki przekazanej do przetworzenia,
3) wstawienie krotki do kolekcji,
4) realizacja operacji algebry,
5) wyznaczenie krotki wynikowej (wstawienie do strumienia wynikowego krotki ze zredukowaną liczbą atrybutów).

Jeżeli dana kolekcja zasila dodatkowo indeks, zamiany zachodzące w wyniku wstawienia krotki i reorganizacji struktury zgłaszane są przy użyciu wydarzeń: insert, update oraz delete.

4. Drzewa informacji

Dane w języku zapytań StreamAPAS v2.0 reprezentowane są jako drzewa atrybutów. Etykieta korzenia wskazuje strumień systemu. Każdy z węzłów drzewa posiada nazwę, typ oraz identyfikator atrybutu w krotce. Nie każdy węzeł struktury musi posiadać zdefiniowany typ. Takie puste węzły tworzą strukturę danych czytelniejszą poprzez wprowadzenie zgrupowań. W poniższym przykładzie rolę tę pełni węzeł *position*.



Rys. 4. Przykład struktury drzewa etykiet
Fig. 4. Example of attribute tree

Ponieważ każda krotka posiada znacznik początku i końca życia, ich identyfikatory mają wartości stałe kolejno *0*, *1*. Chcąc odczytać atrybut *I.position.x* (rys. 4), należy najpierw odczytać wartość identyfikatora atrybutu, następnie pobrać atrybut o wskazanym identyfikatorze z krotki strumienia.

Drzewo informacji jest reprezentowane przez zagnieżdżone listy atrybutów. Skorzystanie z definicji, w której rozpatrywalibyśmy drzewo jako zagnieżdżone wielozbiory [6], nie pozwoliłoby odwoływać się do atrybutów przy użyciu numeru porządkowego.

W konsekwencji, funkcje operujące na drzewach informacji mogłyby odwoływać się do atrybutów tylko poprzez etykietę atrybutu. Przyjęcie koncepcji zagnieżdżonych list atrybutów pozwala rozpoznawać atrybuty funkcji, zgodnie z numerem porządkowym (tak jak ma to miejsce w językach C/C++) lub na podstawie etykiety.

Dla danej zagnieżdżonej listy etykiet A definiujemy listę LT atrybutów indeksowaną przez I tak, że:

- pusta lista etykiet *{}* należy do LT,
- jeśli m występuje w A oraz I w LT, wtedy istnieje para $\{<m,I>\}$ w LT,
- LT jest unią zagnieżdżonych list $\bigcup_{j \in J} M(j)$, gdzie J jest zbiorem indeksów i $M \in J \rightarrow LT$.

Przyjmując powyższą definicję drzewa etykiet, utworzono funkcję mapującą wyrażenie A na atrybuty krotki strumienia.

η ::=	etykieta węzła drzewa,
\$nazwa	etykieta korzenia,
nazwa	etykieta węzła w bieżącym kontekście,
A, B ::=	wyrażenie,
T	drzewo atrybutów począwszy od bieżącego kontekstu,
$\eta[A]$	lokacja bieżącego kontekstu na węźle η,
A , B	kompozycja drzew atrybutów.

Aby uprościć odwoływanie się do pojedynczych węzłów, np. *O*[*z*[*x* = 1.1]], następujące po sobie klamry można zastąpić „.”. Otrzymamy wtedy postać *O.z.x* = 1.1.

Na zdefiniowanej powyżej strukturze danych zostały zdefiniowane podstawowe operacje matematyczne, logiczne oraz operacja kopiowania drzewa atrybutów.

Przykładowe wyrażenie:

O[*z = $I.position.x + $I.position.y, $I. position*[*T*]]

Domyślnym kontekstem etykiet jest lista nazw strumieni, dlatego nie ma potrzeby zapisu *$O*. W podanym przykładzie występuje operacja przypisania *z = ...,* która tworzy nową etykietę o typie zgodnym z wyliczeniem po prawej stronie. Jeżeli etykieta zostałaby wcześniej zdefiniowana, wynik konwertowany byłby do zdefiniowanego typu. Jeżeli brak odpowiedniego rzutowania, nastąpi zgłoszenie błędu. W wyniku kompozycji *..., $I.position* do struktury wynikowej zostaną skopiowane atrybuty ze wskazanego węzła. Należy zwrócić uwagę na prostotę przenoszenia ze struktury źródłowej grupy zmiennych związanych ze sobą znaczeniowo.

Obliczenia etykiet wykonywane są zgodnie z kolejnością wystąpienia, np.:

O[*z = $I.place.x + $I.place.y, z = 1.0*]

zostanie wyliczone

$O.z = $I.place.x + $I.place.y
$O.z = 1.0

5. Składnia zapytania

Pojedyncze zapytanie przetwarzania strumieniowego to sieć powiązanych ze sobą podzapytań. Po słowie kluczowym *from* następuje definicja kolejnych strumieni wewnątrz tej sieci. Na koniec we frazie *select* podajemy strumień wynikowy zdefiniowany na jednym ze strumieni wewnętrznych. Lista strumieni źródłowych potrzebna do budowy podzapytań wyznaczana jest na podstawie analizy atrybutów w frazie *select* i *where*.

Składnia języka:

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

Prezentowany język posiada następującą składnię okien czasowych (patrz sekcja 2):

window.range(source, time) – realizuje funkcję okna zakresowego.
window.fixed(source, time) – realizuje funkcję okna ze 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*

Domyślną kolekcją krotek dla operatorów stanowych jest kolekcja czasowa. Jeżeli chcemy skorzystać z innej, podajemy tę informację w frazie *where*. Alternatywą dla domyślnej kolekcji czasowej jest kolekcja czasowo-tablearyczna o składni:

window.table(source)

Atrybuty:

source – nazwa atrybutu będącego kluczem głównym krotki; atrybut musi być typu numerycznego. Dzięki zastosowaniu drzewa atrybutów nazwa strumienia jest elementem atrybutu source, np. $I.id.

Operatory stanowe mogą być zdefiniowane na kilku strumieniach, zatem występuje wiele konfiguracji okien czasowych. Przeanalizujmy działanie operacji łączenia zdefiniowanej: *$I.a == $II.a* ze względu na konfigurację okien czasowych (tab. 2). Przyjęto, że dane źródłowe mają wstępnie ustawiony czas życia na zero (patrz sekcja 2).

Tabela 2

Konfiguracje okien dla operacji łączenia

Sytuacja	Okno dla I	Okno dla II	Wynik relacji
1.	brak	brak	kolekcje pozostają puste, brak krotek wynikowych
2.	istnieje	brak	krotka ze zbioru $II jest dopasowywana do kolekcji krotek z okna $I, przy czym krotki $I nie są dopasowywane do $II
3.	istnieje	istnieje	zarówno pojawienie się krotki na wyjściu strumienia $I, jak i $II skutkuje rozpoczęciem operacji łączenia

Sytuacja 1 to błąd użytkownika, który nie zdefiniował w pełni zapytania, przy czym nie ma ryzyka utworzenia kolekcji nieskończonej.

Sytuację 2 ilustruje przykład: „Jesteśmy zainteresowani zbiorem (strumień $I) pojazdów z ostatnich 5 min, których szybkość przekroczyła zadaną wartość” (strumień $II).

Rozbudowując powyższy przykład, ilustrujemy sytuację 3: „Jesteśmy zainteresowani zbiorem (strumień $I) pojazdów z ostatnich 5 min, których szybkość będzie przekraczała zadaną wartość (strumień $II) przez najbliższą godzinę”.

6. Przykładowe zapytanie

Zaprezentowane poniżej zapytanie może służyć wskazaniu trzech najbliższych stacji serwisowych dla każdego z nadzorowanych samochodów. Przyjmijmy, że strumień *I* zawiera informacje o bieżącym położeniu przemieszczających się obiektów, a strumień *II* przekazuje informacje o położeniu stacji serwisowych.

```
from
   where window.table($service.id) AND
   util.knn([$car.point[T], k = 3], $service[T])
   select sol[
              name = $car.id,
```

```
                query = $car.point[T],
                service = $service.point[T]];
select out[$sol[T]]
```

Widzimy tutaj zaletę stosowania drzewiastej struktury atrybutów. Jeżeli dobrze zaprojektuje się schematy strumieni w systemie StreamAPAS v2.0, to korzystanie z funkcji sprowadza się do wskazania węzłów struktury atrybutów strumienia, z których będą pobierane dane. Unikamy w ten sposób konieczności wielokrotnego podawania długiej listy atrybutów. Powyższy przykład korzysta również z tymczasowego drzewa atrybutów, które służy rozszerzeniu zbioru atrybutów strumienia zasilającego o atrybut zawierający liczbę poszukiwanych najbliższych sąsiadów.

7. Kompilacja zapytania strumieniowego

Definicja 3.

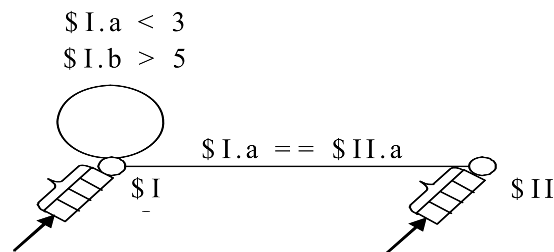
Graf pojedynczego zapytania strumieniowego. Dla danego zapytania strumieniowego p zbudowanego na zbiorze strumieni $S_1,...,S_m$, pojedynczy graf SGp jest oznaczony etykietą oraz węzły SGp są zbiorem strumieni $\{S_1,...,S_m\}$, na którym definiujemy p. Etykieta każdego węzła S_i to $p : X_i, i \in 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

Dla każdego wyrażenia atomowego q w p:

- jeśli q jest operatorem jednoargumentowym zdefiniowanym na S_i, istnieje krawędź z S_i do samego siebie (pętla operacyjna) w SGp z etykietą q,
- jeśli q jest operatorem wieloargumentowym zdefiniowanym na $S_i,..,S_j$, istnieje sieć powiązań pomiędzy $S_i,..,S_j$ (dla dwuargumentowych krawędź operacyjna) w SGp z etykietą q,
- jeśli q jest definicją kolekcji krotek, etykieta umieszczana jest przy węźle, na którym jest zdefiniowana,
- jeżeli węzeł SGp nie posiada kolekcji krotek, przyjmuje się za domyślną kolekcję czasową.

Deklaracje strumieni kompilowane są do postaci grafu pojedynczego zapytania. Reprezentacja taka abstrahuje od konkretnego planu produkcji, co czyni nią dobrą strukturą danych, na której można uruchomić optymalizator zapytania, tworzący plan drzewa przetwarzania [5]. Dla poniższego zapytania przedstawiono graf, składający się z operatorów frazy *where* (rys. 5):

```
where $I.a == $II.a AND $I.a < 3 AND $I.b > 5
select odp[place[x = $I.a, y = $I.b]];
```



Rys. 5. Graf pojedynczego zapytania: dla uproszczenia przy węzłach brak listy atrybutów pobieranych do wyliczenia frazy select

Fig. 5. The single query graph: for simplicity there isn't attributes list used by a phrase select for each node

Definicja grafu pojedynczego zapytania bazuje na analogicznej strukturze używanej w tradycyjnych BDMS [5]. Dzięki temu, że użytą algebrę przetwarzania strumieniowego cechuje redukowalność migawkowa, statyczną optymalizację zapytania strumieniowego można przeprowadzać podobnie jak w DBMS.

W wyniku rozbioru frazy *where* na składowe operatory algebry, dysponujemy grafem pojedynczego zapytania w postaci listy operatorów. Kolejność ich umieszczenia w niej pełni kluczową rolę w trakcie tworzenia planu drzewa przetwarzania. Działanie optymalizatora zostanie przedstawione na przykładzie operatorów frazy *where*. Poniżej podano listę operatorów (tab. 3.) tworzących graf z rys. 5. oraz mechanizm budowy drzewa przetwarzającego (algorytm 1).

Tabela 3

Lista rozbioru

Operacja	Atrybuty
$I.a == $II.a	I.a, II.a
$I.a < 3	I.a
$I.b > 5	I.b

Algorytm 1. Budowa drzewa przetwarzania na podstawie grafu pojedynczego zapytania (zapisanego jako lista operatorów).

```
lista_produkcji := {operatory strumieni źródłowych}
for element in lista_operatorów
{
   for strumień_źródłowy in strumienie_źródłowe_operatora(element)
   {
      produkcja := lista_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 notacji odwrotnej 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.

Utworzony optymalizator implementuje optymalizację warunkową, której działanie polega na uporządkowaniu operatorów w liście rozbioru w porządku rosnącym zgodnym z przyjętym priorytetem operatora (tab. 4.).

Tabela 4

Tabela priorytetów operatorów

Typ	Priorytet	Operatory
OUTPUT	4	wyliczenie frazy select
STATEFUL	3	Join
WINDOW	2	fixed, infinite, range, table
STATELESS	1	Filter
SOURCE	0	Strumień źródłowy

Skutek przyjętych wartości priorytetów:

- filtry zostają umieszczone na początku drzewa przetwarzania,
- opóźnienie wykonania operatorów okien czasowych zgodnie z regułą

$$\sigma_p(\omega_w(S)) \doteq \omega_w(\sigma_p(S)) \text{ [3]} \qquad (5)$$

Przed uruchomieniem optymalizacji listy operatorów następuje redukcja predykatów zdefiniowanych na tych samych źródłach danych. Wskutek tego następujące po sobie operacje filtracji zastępujemy pojedynczym operatorem. Tabela 5 ilustruje redukcje operatorów z tabeli 3.

Na koniec procesu tworzenia drzewa węzłów przetwarzających definiowane są operatory mapowania, tak aby zminimalizować liczbę transmitowanych danych w strumieniach wewnętrznych.

Tabela 5

Lista rozbioru po optymalizacji

Operacja	Atrybuty
$I.a < 3 AND $I.b > 5	I.a, I.b
$I.a == $II.a	I.a, II.a

Przyjęta składnia języka zapytań strumieniowych StreamAPAS v2.0 pozwala w prosty sposób zarządzać złożonymi strukturami danych. W przeciwieństwie do SQL, gdzie każdy

argument funkcji należy podać oddzielnie, w utworzonym języku zapytań argumentem funkcji może być drzewo atrybutów, co sprawia, że treść zapytania jest czytelniejsza i krótsza.

Drugą istotną zaletą języka jest mocne wsparcie interfejsu funkcji. Wyróżniamy funkcje realizujące: operatory okien, operatory relacji, kolekcji krotek. Systemy przetwarzające strumienie danych służą głównie do przeprowadzania analiz na bieżąco. Ponieważ w tej dziedzinie rozwój jest szybki, więc język zapytań strumieniowych musi udostępniać metody jego łatwej rozbudowy.

Zaproponowana algebra w [3] została rozszerzona o możliwość wyboru typu kolekcji krotek. Dodana nowa kolekcja czasowo-tabelaryczna upraszcza przenoszenie rekordów z tradycyjnych baz danych do systemu strumieniowego.

Rozszerzono i przeniesiono definicję grafu pojedynczego zapytania [5] dla tradycyjnych baz danych do utworzonego systemu przetwarzania strumieniowego. Dzięki temu optymalizacja drzewa operatorów sprowadza się do wyznaczenia właściwej permutacji operatorów w liście produkcji. W dalszej części artykułu przedstawione zostało rozwiązanie optymalizacji regułowej jako sortowanie operatorów zgodnie z przypisanymi im priorytetami.

Wprowadzony wydajniejszy algorytm synchronizacji przetwarzania skrócił czasu oczekiwania krotek na przetworzenie. Dalsza rozbudowa tego mechanizmu [2] pozwala uzyskać informację o czasie odpowiedzi poszczególnych węzłów sieci przetwarzania, co dałoby możliwość monitorowania na bieżąco obciążenie systemu strumieniowego.

8. Podsumowanie

Zaprezentowany prototypowy język zapytań strumieniowych StreamAPAS v2.0 zorientowany jest na dziedziny zastosowań analitycznych. W odróżnieniu do istniejącego CQL składnia omawianego języka definiuje sposób rozbudowy funkcjonalności, co zwiększa zakres potencjalnych aplikacji opartych na takiej platformie przetwarzania strumieniowego.

Zaproponowany mechanizm synchronizacji krotek w strumieniach pozwala na płynne ich przetwarzanie, nawet gdy częstość nadchodzących danych w jednym ze strumieni operatora jest niska (co stanowiło słabą stronę systemu, na którym się wzorowano [3]).

StreamAPAS v2.0 może posłużyć jako baza eksperymentalna do dalszych badań w tej dziedzinie. Ponieważ system StreamAPAS v2.0 wspiera ograniczoną liczbę operatorów, więc jednym z kierunków dalszego rozwoju powinna być ich rozbudowa. Także w celu zwiększenia wydajności przetwarzania systemu StreamAPAS v2.0 należy udostępnić możliwość lokalizacji węzłów operatorów w środowisku rozproszonym.

Istnieje również potrzeba dalszego rozwoju algebry [3] w celu lepszego wsparcia także danych niestrumieniowych, które są typowe dla większości rzeczywistych aplikacji.

Literatura

1. Gorawski M., Skierski A.: Okienkowy język zapytań. II Krajowa Konferencja Naukowa Technologie Przetwarzania Danych. Poznań 24-26 września 2007.
2. Balazinska M.: Fault-tolerance and load management in a Distributed Stream Processing System, PhD dissertation, Massachusetts institute of technology, February 2006.
3. Krämer J., Seeger B.: A Temporal Foundation for Continuous Queries over Data Streams, Department of Mathematics and Computer Science PhilippsUniversity, Marburg, Germany, 11th International Conference on Management of Data (COMAD 2005)
4. Arasu A., Babu S., Widom J.: The CQL Continuous Query Language Semantic Foundations and Query Execution, Stanford University, the International Journal on Very Large Databases (VLDB Journal), June 2006, t. 15 nr 2 , s. 121÷142
5. Yu S., Atluri V., Nabil A.: Optimizing View Materialization Cost in Spatial Data Warehouses, Lecture Notes in Computer Science 4081 Springer 2006, s. 45÷54
6. Ardelli L., Ghelli G.: TQL: A Query Language for Semistructured Data Based on the Ambient logic, MATHEMATICAL STRUCTURES IN COMPUTER SCIENCE 2004, Vol. 14; Part 3, s 285÷328
7. Johannes D., Leijen P.: The λ Abroad – A Functional Approach To Software Components, Daan Leijen. PhD thesis, Department of Computer Science, Utrecht University, November 2003

Recenzent: Dr inż. Tomasz Koszlajda

Wpłynęło do Redakcji 25 listopada 2007 r.

Abstract

The paper describes the query language prototype for stream processing. The proposed language syntax in comparison with CQL define the way of further development system functionality by means of functions interface. This and hierarchical data structures simplify maintaining analytical applications where development is permanent process.

The presented stream processing system combines mainly the temporal logical operator algebra [3] and streams synchronization from [2]. In the result the final system has well defined available optimization rules and can offer low latency for tuples processing even when the query consist of slow data streams.

Adresy

Marcin GORAWSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, Marcin.Gorawski@polsl.pl.

Aleksander CHRÓSZCZ: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, Aleksander.Chroszcz@polsl.pl.