STUDIA INFORMATICA 2012
Volume 33 Number 2A (105)

Marcin GORAWSKI
Politechnika Śląska, Instytut Informatyki, Politechnika Wrocławska, Instytut Informatyki
Anna GORAWSKA
Politechnika Śląska, Instytut Informatyki

AGKPSTREAM A OPERATORY STRUMIENIOWE

Streszczenie. Przedstawiono ramowo prototypowy system strumieniowego przetwarzania danych o nazwie *AGKPStream*. Obiektem przeprowadzonych badań są jego pojedyncze komponenty przetwarzające – operatory strumieniowe. W artykule szczegółowo przedstawiono modele i zasady działania tych operatorów oraz wyniki testów operatora selekcji.

Słowa kluczowe: operatory strumieniowe, strumieniowe przetwarzanie danych

STREAM SCHEDULERS

Summary. The following paper introduces a prototype Data Stream Management System called *AGKPStream*. The objects of the study were the operators constituting a single stream processing components created in the system. We present basic concepts, principles of stream operators (including selection, projection, and union) and experimental results of the select operator.

Keywords: stream operators, stream data processing

1. Wstęp

W dziedzinach, które wymagają szybkiego przetwarzania dużej ilości danych napływających w trybie on-line z wielu źródeł, zapis danych przed ich analizą znacznie spowalnia pracę klasycznych systemów baz danych i czyni je nieefektywnymi [5, 7, 10, 11]. Istotne jest wymaganie zapewnienia ciągłej i dynamicznej analizy danych połączonych z prezentacją bieżących wyników. Przykładami motywującymi są systemy dynamicznej eksploracji danych giełdowych, monitorujące ruch drogowy (lub lotniczy) bądź analizujące logi serwerowe. W przypadku systemów giełdowych istotą ich efektywności jest minimalizacja opóźnień

przetwarzania danych (czasowa optymalizacja); dopuszczalne jest pominięcie pewnych danych, jednak opóźnienie rzędu sekund ma zwykle konsekwencje w postaci zwrócenia nieaktualnych już (potocznie „nieświeżych") danych. Sytuacja taka jest niedopuszczalna w przypadku systemów monitorujących bezpieczeństwo lokalizacji publicznych takich, jak lotniska bądź ambasady; pominięcie jakichkolwiek danych może mieć tragiczny skutek, opóźnienie natomiast nie wpłynie zasadniczo na przydatność i efektywność systemu IT. Pożądanym rozwiązaniem wymienionych problemów są systemy przetwarzania strumieniowego.

Zaprojektowany i zaimplementowany autorski system przetwarzania strumieniowego *AGKPStream* został zrealizowany w Zespole Teorii Przestrzeni Danych i Algorytmów w Zakładzie Teorii Informatyki Instytutu Informatyki Wydziału AEiI Politechniki Śląskiej w ramach projektów inżynierskich [1, 2]. *AGKPStream* jest kontynuacją badań nad strumieniowym modelem przetwarzania danych z prac wcześniejszych [3, 4].

W niniejszym artykule przestawiono modele operatorów strumieniowych zaimplementowane w *AGKPStream*. Przedstawiony system jest oparty na modelu strumieniowym przetwarzania danych [3-11]. Model obiektowy, tj. pojęcie klas, odnosi się do strony implementacyjnej systemu. Przeprowadzono wiele testów funkcjonalności i wydajności operatorów pod kątem wybranych metryk.

2. Podstawowe pojęcia

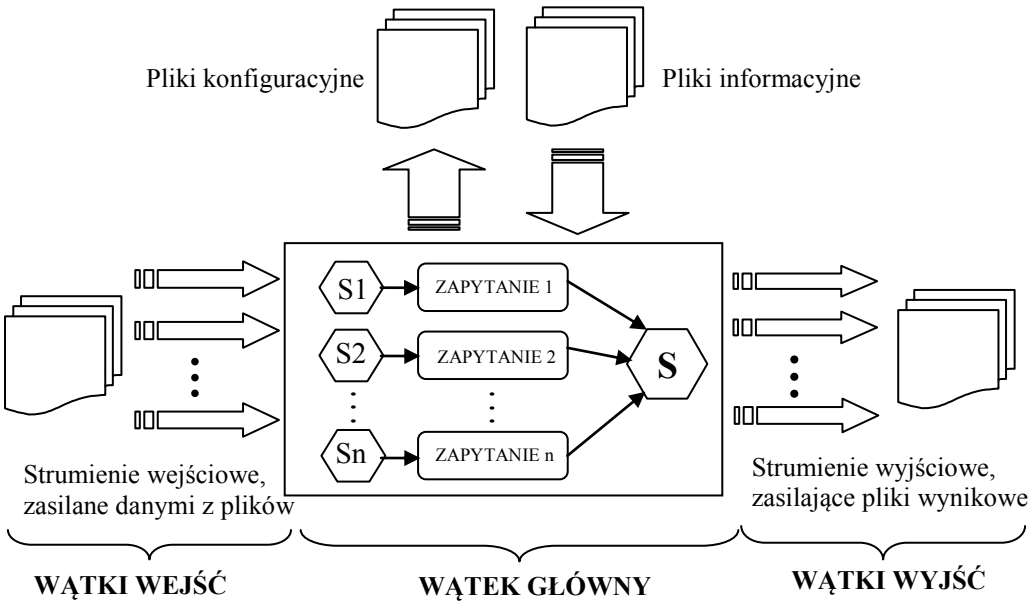
System zarządzania strumieniową bazą danych SDMS (*Stream Database Management System*) ma za zadanie przetwarzanie danych przychodzących w postaci strumieni, które cechują ciągłość, nieskończoność i zmienność w czasie. Dane napływają do systemu na bieżąco i nie można przyjąć założeń dotyczących ograniczonej ich liczby. Ponadto kolejność oraz zawartość napływających danych jest niezależna od SDMS. Po wykonaniu operacji odczytu dane są usuwane ze strumienia bądź archiwizowane. Często aplikacje łączą modele przetwarzania strumieniowego z przetwarzaniem transakcyjnym.

SDMS zarządza i organizuje pracę systemu strumieniowego w kontekście źródeł danych, operatorów i ich strumieni, schedulerów oraz strumieni wyjściowych. Dane ze źródła przechodzą przez sieć operatorów zdefiniowanych w zapytaniach i trafiają na wyjścia w postaci już przetworzonych danych. SDMS pozwala użytkownikowi na rejestrację wielu ciągłych zapytań strumieniowych, których wykonanie następuje w chwili nadejścia nowych danych. Sposób wykonania zapytań jest określany przez wybrany scheduler.

W klasycznych rozwiązaniach baz danych procesor zapytań przetwarza dane na zasadzie żądanie-zapytanie, co oznacza, że użytkownik konstruuje zapytanie logiczne bądź kwerendę w języku zapytań, obsługiwanym przez system zarządzania bazą danych. Silnik zapytań, po-

wiązany z bazą danych, przetwarza otrzymane zapytanie i zwraca zbiór wynikowy. W modelu strumieniowym realizacja zapytań ma charakter nieustający – silnik zapytań ma za zadanie ciągłe i nieprzerwane filtrowanie danych pochodzących ze źródeł strumieni danych.

SDMS systemu *AGKPStream* pozwala na rejestrację wielu zapytań strumieniowych, pracujących równocześnie. Scheduler przydziela aktywnemu zapytaniu pewien interwał czasu i zarządza operatorami tworzącymi zapytanie. Po upływie wyznaczonego czasu następuje zatrzymanie pracy zapytania i uaktywniane jest kolejne. Dzięki takiemu podejściu zapytania nie są od siebie zależne, co – ze względu na ciągłe wykonywanie zapytania – uniemożliwia przetwarzanie całych zapytań jedno po drugim. Dane napływają strumieniowo i mogą zasilać więcej niż jedno zapytanie. Schemat budowy wielowątkowego systemu *AGKPStream* przedstawia rysunek 1, gdzie: S_i dla i = 1,2…n są schedulerami pracującymi na poziomie konkretnych zapytań, a S jest schedulerem globalnym, zarządzającym pracą całego systemu.



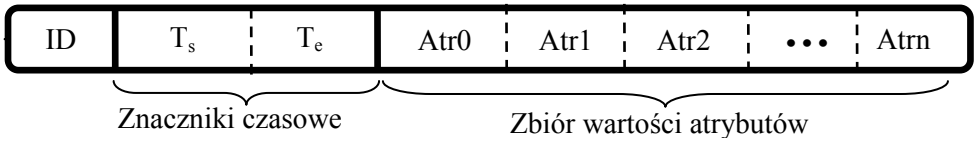
Rys. 1. Schemat działania systemu *AGKPStream* z uwzględnieniem organizacji wątków
Fig. 1. *AGKPStream* system principle of operations with thread organization

Praca w systemie *AGKPStream* została podzielona na wątki. W ramach głównego wątku działają operatory i schedulery. Ponadto każde wejście i wyjście pracuje w osobnym wątku. Dzięki takiemu podziałowi praca *AGKPStream* nie jest opóźniana przez operacje zapisu i odczytu z pliku. W efekcie działania przypisane operatorom są wykonywane w krótszym czasie, co usprawnia pracę całego systemu *AGKPStream*.

2.1. Krotka i jej schemat

W systemie *AGKPStream* przyjęty został model temporalny krotki jak na rysunku 2. Pole *ID* jest unikalnym polem każdej krotki, znaczniki T_s i T_e oznaczają początek oraz koniec

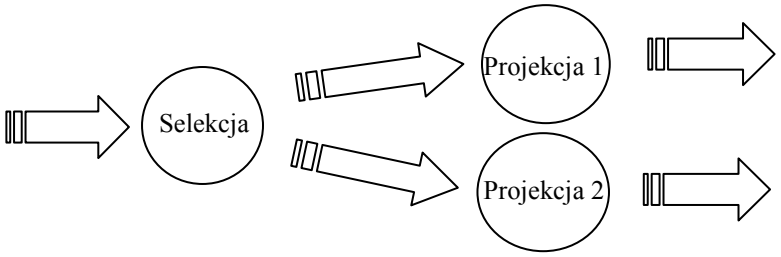
przedziału czasowego życia krotki. Klasa *Tuple* reprezentuje krotkę. Przechowuje informacje o unikalnym *ID* krotki, czas życia oraz mapę nazw atrybutów na ich wartości.



Rys. 2. Schemat krotki
Fig. 2. Tuple schema

Klasa reprezentująca czas życia, jaki charakteryzuje krotkę, udostępnia początek oraz koniec przedziału czasowego, w którym krotka – z punktu widzenia systemu – może zostać przetworzona. Krotka może być wykorzystywana przez wiele zapytań jednocześnie, dlatego nie ma możliwości modyfikacji jej danych. Ochronę danych krotki zapewnia zbudowany interfejs *ITuple.* W takim rozwiązaniu klasa krotki rozszerza interfejs tak, że obiekty utworzone jako *ITuple* nie mogą zostać zmodyfikowane równocześnie, zachowując cechy charakterystyczne krotki.

Schemat krotki tworzy zbiór nazw atrybutów, które po stronie krotki są mapowane na konkretne wartości. W systemie *AGKPStream* schemat jest obiektem klasy *LinkedHashSet.* W związku z tym kolejność wstawiania elementów jest równoznaczna z kolejnością ich ekstrakcji [1, 2]. Pola *ID* oraz znaczniki czasowe nie należą do schematu krotki. Struktura krotki jest spójna ze schematem, jednak krotka jako obiekt w systemie *AGKPStream* jest uniezależniona od schematu, tzn. referencja na schemat znajduje się po stronie strumienia [1, 2]. Takie rozwiązanie ma ogromne zalety, gdy krotka jest przetwarzana przez kilka operatorów, np. najpierw przez operator selekcji, a następnie przez dwie projekcje równolegle (rys. 3).



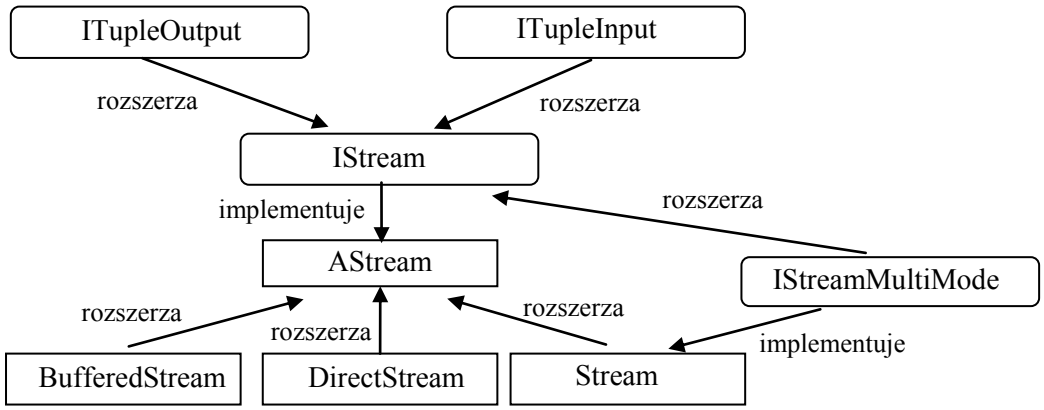
Rys. 3. Przykładowe zapytanie
Fig. 3. Exemplary query

Uzależnienie krotki od schematu spowodowałoby powstanie dwóch nowych krotek – po jednej na wyjściu każdej z projekcji. Zastosowanie referencji na schemat w strumieniu rozwiązuje ten problem, gdyż ciągle bazujemy na krotce wejściowej, a projekcje jedynie przycinają widok na krotkę po stronie strumieni wyjściowych operatorów. Ponadto ta sama krotka może zasilać wiele strumieni bez konieczności modyfikacji jej struktury. Klasa schematu oferuje metody umożliwiające pobranie, modyfikację schematu oraz sprawdzenie poprawności, tzn. czy schemat zawiera wybrany atrybut bądź grupę atrybutów.

2.2. Strumień danych

Strumienie *AGKPStream* przechowują informacje o schemacie obowiązującym krotki do niego wpływające oraz o operatorze, który pobiera z niego krotki do przetworzenia. Ponadto każdy ze strumieni w systemie nakazuje podłączonemu operatorowi pracę aż do momentu opróżnienia strumienia. Dzięki temu działanie operatorów nie jest przeznaczone na konkretną ilość przetworzonych danych. Również po stronie strumienia następuje wymuszenie obliczenia schematu krotki na wyjściu podłączonego operatora.

Ogólny schemat organizacji klas i interfejsów stworzonych w ramach *AGKPStream* na potrzeby realizacji strumieni przedstawia rysunek 4.

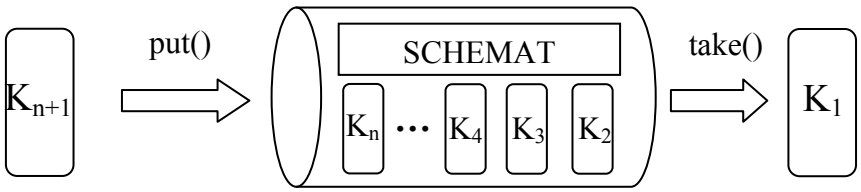


Rys. 4. Schemat klas strumieni
Fig. 4. Stream class diagram

Klasa abstrakcyjna *AStream* stanowi podstawę dla każdego ze strumieni stworzonych w ramach systemu. Każdy strumień musi przechowywać informacje o schemacie oraz operatorze, z którego zostały pobrane krotki.

System przewiduje dwa typy strumieni reprezentowane przez klasy *DirectStream* oraz *BufferedStream*, które różnią się jedynie posiadaniem bufora danych. W pierwszym umiejscowienie krotki w strumieniu powoduje wymuszenie pracy operatora, którego zdaniem jest przetworzenie krotki. Po pobraniu krotki przez operator jest ona usuwana ze strumienia. W przypadku *BufferdStream* krotka ze strumienia jest dodawana do bufora, a jej pobranie przez operator powoduje usunięcie tej samej krotki z bufora. Klasa *Stream* uogólnia spojrzenie na strumienie w systemie, umożliwiając ustawienie typu strumienia, w związku z czym przechowuje referencję na strumienie typów *BufferedStream* oraz *DirectStream*. Ponadto przechowuje wskaźnik na aktualnie używany typ strumienia, tj. *currentStream.* Pole *currentStream* jest domyślnie ustawione na *BufferedStream*, jednak istnieje możliwość zmiany typu w trakcie działania systemu. Zmiana wymusza przetworzenie wszystkich krotek, które znajdują się w przekształcanym strumieniu. Klasa *Stream* ma te same metody, co klasy strumieni *DirectStream* oraz *BufferedStream*, i stosuje mechanizm delegacji na typ strumienia.

Wejście oraz wyjście są z punktu widzenia operatora traktowane jako strumień. Dzięki zastosowaniu interfejsów *ITupleInput* oraz *ITupleOutput* tworzenie nowych typów strumieni jest zbędne. Pierwszy z nich reprezentuje obiekt, z którego można pobrać krotki, a drugi obiekt jest miejscem docelowym przesyłu krotki. Ogólna koncepcja strumienia reprezentującego obiekt klasy *Stream* została przedstawiona na rysunku 5:



Rys. 5. Strumień buforowany *BufferedStream*
Fig. 5. Buffered stream *BufferedStream*

Rysunek 5 przedstawia obiekt klasy *Stream*, którego pole *currentStream* wskazuje na strumień buforowany. Elementy K_i, gdzie $i = 2,3,\ldots,n$, są krotkami, które znajdują się wewnątrz strumienia; krotka K_1 jest przesyłana metodą *take()* do następnego elementu zapytania bądź na wyjście. Krotka K_{n+1} przychodzi natomiast do strumienia i jest wstawiana na koniec jego kolejki metodą *put()*.

3. Operatory strumieniowe

Operator jest pojedynczym komponentem, którego zadaniem jest przetworzenie dostarczonej krotki bądź grupy krotek. Każdy z operatorów stanowi odrębną i niepodzielną jednostkę, której działanie nie jest bezpośrednio zależne od funkcjonowania operatora innego typu. W modelu strumieniowym stosowane są między innymi operatory będące odpowiednikami operatorów stosowanych w systemach zarządzania relacyjnymi bazami danych. Bezpośrednie zastosowanie relacyjnych operatorów nie jest możliwe ze względu na zjawisko blokowania pracy operatora. U źródła zjawiska leży fakt, iż działania w modelu relacyjnym opierają się na skończonym zbiorze danych. W przypadku strumieni nie można określić maksymalnej liczby przetwarzanych elementów (nieskończona kolekcja danych). W związku z tym operator oczekiwałby na zamknięcie strumienia, bo dopiero wówczas można byłoby uznać, że otrzymał wszystkie dane. Ponadto część operatorów wymaga przechowywania wszystkich otrzymanych wcześniej krotek, co doprowadziłoby do przerwania pracy z powodu braku wolnych zasobów. Stąd w przetwarzaniu strumieniowym nie występuje operator sortowania, który pracuje na całej kolekcji danych, i przyjście nowego elementu wymaga przejrzenia poprzednich krotek.

Ze względu na wymagania pamięciowe operatory można podzielić na bezstanowe i stanowe. Pierwsza grupa wylicza wynik tylko na podstawie krotki wejściowej i są to: selekcja,

projekcja, unia oraz operatory okien czasowych. Operatory stanowe wymagają natomiast podania dodatkowych danych, takich jak historia przetworzonych krotek. Reprezentantami drugiego typu są operatory złączenia i agregacji.

Dla każdego utworzonego operatora klasą bazową jest *AOperator*, którą musi rozszerzać każdy operator zaimplementowany w systemie. Ponadto system został wyposażony w trzy interfejsy, których zadaniem jest umożliwienie pracy operatora na trzech różnych poziomach. Interfejs *IOperator* udostępnia wszystkie metody operatora. Interfejs *IWorker* jest skupiony wokół wykonywania pracy w postaci przetwarzania danych. Natomiast *IOwner* umożliwia pracę operatora z punktu widzenia elementu podłączonego na wyjście. Każdy z działających w systemie operatorów bazuje na abstrakcyjnej klasie *AOperator*, implementującej interfejs *IOperator*, która wyznacza podstawowe funkcjonalności oraz budowę operatorów w systemie. W klasie przechowywana jest informacja o wejściu operatora oraz lista jego wyjść. Wejście operatora jest obiektem typu *ITupleInput*, a każde wyjście – *ITupleOut*.

Do ważniejszych operatorów, które zostały stworzone na potrzeby nowego modelu przetwarzania danych, należą okna, których głównym zadaniem jest ograniczenie okresu aktywności krotki w systemie. Są wykorzystywane jako operatory pomocnicze w celu ograniczenia liczby krotek pobranych przez system.

Operatory okien można podzielić na czasowe i fizyczne, zwane inaczej liczebnościowymi. Okna czasowe bazują na wartościach znaczników czasu, a fizyczne operują na innych, dostępnych znacznikach. Wśród okien czasowych można wyróżnić **okna przesuwno-czasowe** oraz **okna stało-czasowe.**

W ramach stworzonego systemu *AGKPStream* zostały zaimplementowane i przebadane następujące operatory: ***selekcja***, ***projekcja***, ***unia***, ***złączenie kartezjańskie***, ***złączenie z warunkiem równości***, ***złączenie z warunkiem*** $>, <, \geq, \leq, \neq$, ***okno przesuwno-czasowe*** oraz ***okno stało-czasowe***. Każdy operator musi umożliwiać obliczenie schematu wyjściowego krotki na podstawie schematu wejściowego oraz przetworzenie krotek, które przyszły ze strumienia (strumieni) wejściowych.

Praca operatora została podzielona na trzy etapy:

- Przygotowanie do przetworzenia krotki, tj. jeśli na wejściu znajduje się krotka, do przetworzenia wyznaczany jest czas rozpoczęcia pracy i zwiększany licznik krotek wejściowych.
- Przetworzenie danych następujące w *processTuple()*. Ze względu na różnorodność zadań operatorów jest to metoda abstrakcyjna, której implementacja jest wymuszona przy tworzeniu wszystkich typów operatorów.
- Zakończenie przetwarzania, tj. wyznaczenie czasu pracy operatora.

3.1. Selekcja

Operator selekcji ma charakter filtru, który przepuszcza do dalszego przetworzenia krotki, które spełniają określony warunek bądź listę warunków. Warunek selekcji może dotyczyć jedynie wartości atrybutów krotki.

$$\sigma_f(S) = \{(t.T_s, t.T_e, f(t.Data)) \mid t \in S, gdzie\ f(t.Data)\ jest\ prawdą\}, \quad (1)$$

gdzie: S jest strumieniem wejściowym, $t.T_s$ i $t.T_e$ są znacznikami czasowymi krotki t, a $t.Data$ jest zbiorem wartości atrybutów krotki t.

Na pojedynczy warunek selekcji składają się: nazwa atrybutu należącego do schematu krotki, którego dotyczy warunek, operator prosty ($>, <, \geq, \leq$ lub $\neq$) oraz wartość, z którą jest porównywany atrybut. W stworzonym systemie dopuszczalne jest stworzenie operatora selekcji sprawdzającego wiele warunków jednocześnie. Selekcja nie zmienia schematu przetwarzanych krotek, w związku tym pobierany jest schemat ze strumienia wejściowego i następuje sprawdzenie, czy ma wszystkie nazwy atrybutów, wykorzystane w warunkach. Jeśli warunki dotyczące istniejących w ramach krotki atrybutów są spełnione, schemat wejściowy zostaje przesłany na wszystkie wyjścia operatora selekcji.

3.2. Projekcja

Projekcja przycina krotkę do zadanych w konstruktorze atrybutów. W związku z tym predykat stanowi podzbiór zbioru atrybutów wejściowych krotki. *ID* oraz znaczniki czasowe krotki nie należą do schematu, stanowią integralną i nienaruszalną część krotki. Krotka wyjściowa zatem jest zaopatrzona w *ID*, znaczniki czasowe oraz przycięty zbiór atrybutów. Projekcja jest rozumiana jako funkcja zawartości wejściowej krotki i ma następującą postać:

$$\pi_f(S) = \{(t.T_s, t.T_e, f(t.Data)) \mid t \in S, gdzie\ f(t.Data)\ jest\ prawdą\}, \quad (2)$$

gdzie: S jest strumieniem wejściowym, $t.T_s$ i $t.T_e$ są znacznikami czasowymi krotki t, a $t.Data$ jest zbiorem wartości atrybutów krotki t.

Z względu na specyfikę *AGKPStream* selekcja wybranych atrybutów nie następuje po stronie krotki. Krotka może zasilać wiele zapytań strumieniowych, w związku z czym jej dane nie mogą ulec modyfikacji. Operacja projekcji jest zatem realizowana na schemacie. Przed przesłaniem pierwszej krotki do zapytania następuje wyznaczenie wszystkich schematów. W konsekwencji pobranie schematu wejściowego krotki oraz wyznaczenie na jego podstawie nowego schematu, składającego się tylko z wyznaczonych w predykacie atrybutów, następuje przed rozpoczęciem przetwarzania krotek przez zapytanie. Utworzony schemat wyjściowy jest następnie przesyłany na strumienie wyjściowe operatora projekcji. Krotki nie ulegają modyfikacji, w związku z czym są bezpośrednio przesyłane na strumienie wyjściowe. Kolejne operatory podłączone do strumieni wyjściowych projekcji pobierają zatem schemat

wyjściowy projekcji, tj. przycięty widok krotki, i nie mają możliwości odczytu atrybutów nieuwzględnionych w schemacie.

W konsekwencji kluczową operacją z punktu widzenia operatora jest wyznaczenie schematu wyjściowego. Po pobraniu schematu wejściowego krotki następuje sprawdzenie, czy atrybuty podane w konstruktorze operatora stanowią podzbiór zbioru atrybutów wejściowych. Jeśli predykat jest poprawny, następuje utworzenie nowego schematu krotki na podstawie podanej listy nazw atrybutów.

3.3. Unia

Operator unii łączy dowolną liczbę strumieni wejściowych, tworząc z nich jeden strumień wyjściowy. Unia nie przetwarza krotek, jedynie przesyła je na wyjście.

W *AGKPStream* zaimplementowano dwa operatory unii: *UnionSingleStream*, gdzie strumienie wejściowe wywodzą się z tego samego źródła, oraz *UnionMultiStream* – strumienie wejściowe mają różne źródła. Przyjęto następujące nazewnictwo atrybutów: *źródło#atrybut,* gdzie *źródło* jest unikalną nazwą źródła danych, z którego pochodzi atrybut o nazwie *atrybut.* Strumienie wywodzące się z jednego źródła danych mają taką samą nazwę atrybutu *źródło.* W wielu źródłach nazwa atrybutu *atrybut* może się powtarzać, tj. w systemie mogą występować następujące atrybuty: *źródło_1#Napis* oraz *źródło_2#Napis*. W zaprezentowanym przypadku nazwy atrybutów są identyczne, jednak pochodzą z różnych źródeł danych. Zatem wszystkie nazwy atrybutów w systemie są w rzeczywistości unikalne dzięki unikalności kombinacji nazwy źródła i nazwy atrybutu.

UnionSingleStream. W operatorze *UnionSingleStream* schemat wyjściowy unii jest ustalany na podstawie pierwszego, podłączonego do operatora strumienia wejściowego, kolejne schematy są z nim porównywane. Pierwszy otrzymany schemat jest przesyłany do wszystkich strumieni wyjściowych. W ramach *UnionSingleStream* strumienie charakteryzują się takimi samymi wartościami *źródło* oraz *atrybut*. W efekcie wystarczy przypisać pierwszy schemat na wyjście, jeśli dla każdej pary atrybutów zachodzi taka zależność.

***UnionMultiStream*.** Operator *UnionMultiStream* odnosi się do strumieni, które mają różne źródła, ale charakteryzują się taką samą wartością *atrybut*, np. *Źródło_1#Napis* oraz *Źródło_2#Napis.*

Zgodnie ze specyfikacją strumieni atrybuty należące do schematu krotki muszą mieć unikalne nazwy. W pierwszym przypadku, rozpatrzonym w ramach *UnionSingleStream*, strumienie wejściowe operatora wywodzą się z jednego źródła, dzięki czemu obowiązujący schemat wyjściowy na pewno nie zawiera zduplikowanych nazw atrybutów. Operator *UnionMultiStream* odnosi się do przypadku, gdy istnieje prawdopodobieństwo powtarzania się nazw atrybutów. Nazwa *Napis* jest unikalna w odniesieniu do konkretnego źródła, jednak istnieje problem w nazewnictwie, gdy tworzony jest schemat wyjściowy. Przy bezpośrednim

przepisaniu nazw atrybutów w oderwaniu od nazw ich źródeł nastąpiłby konflikt nazewnictwa. W metodzie wyznaczającej schemat wyjściowy operatora zostało zbudowane zabezpieczenie zapewniające unikalność nazw atrybutów w schemacie wyjściowym. Dla wymienionego przypadku w schemacie wyjściowym atrybut będzie mieć następującą nazwę: *Źródło_1_u_Źródło_2#Napis.*

Algorytm wyznaczania schematu wyjściowego *UnionMultiStream* prezentuje się następująco:

- Pobranie schematu aktualnego wejścia.
- Sprawdzenie, czy lista schematów wejściowych jest pusta. Jeśli lista zawiera wpisy wykonywane jest sprawdzenie, czy któryś z już zapisanych schematów ma atrybuty o identycznych nazwach jak schemat aktywnego wejścia.
- Po pomyślnym przejściu sekcji sprawdzającej bądź gdy lista schematów wejściowych była pusta, pobrany schemat jest dopisywany do listy.
- Porównanie liczby wejść operatora, określonych w konstruktorze, z licznikiem wejść. Jeśli są sobie równe oznacza to, że wszystkie wejścia zostały sprawdzone i schemat wyjściowy jest wyznaczany jako suma wszystkich schematów wejściowych.

3.4. Okna czasowe

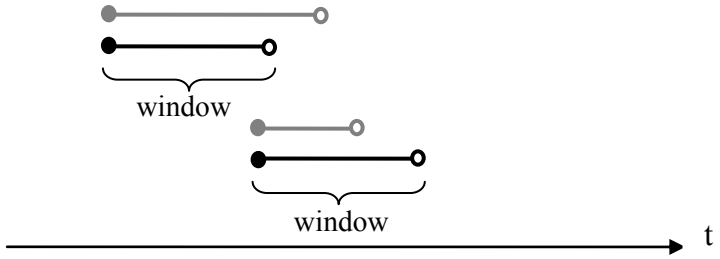
Zgodnie z teoretycznym modelem przetwarzania strumieniowego strumienie nie są ograniczone w czasie. W konsekwencji operatory stanowe musiałyby mieć możliwość przechowywania nieskończonej liczby krotek, co jest mało realne. Operatory okien czasowych mają funkcję pomocniczą i operują na znacznikach czasowych krotek. Ich celem jest ograniczenie czasu życia krotek, tak by operatory stanowe nie stawały przed problemem przechowywania ogromnej ilości danych. Zmniejszenie wymagań pamięciowych operatorów nie stanowi ograniczenia dla liczby aktywnych krotek, które mogą znajdować się w systemie. Ich liczba jest determinowana przez intensywność strumieni.

Okno przesuwno-czasowe. Okno przesuwno-czasowe zmienia długość czasu życia krotek wejściowych na stałą wartość względem znacznika początkowego czasu. Oznacza to, że dla wszystkich krotek wchodzących na wejście operatora do czasu początkowego zostanie dodana stała wartość, która wyznaczy nowy znacznik końcowy życia krotki. W związku z tym po przetworzeniu przez operator okna przesuwno-czasowego krotki będą się charakteryzowały identyczną długością życia. Operator można zdefiniować jako funkcję, gdzie: *t* jest krotką wejściową, *S* strumieniem, a ω wartością okna czasowego:

$$SlidingWindow_{\omega}(S) = \{(t.t_s, t.t_s + \omega, t.data) \mid t \in S\}. \quad (3)$$

Jak zauważono w [4], kolejność krotek zawsze będzie taka sama, niezależnie od wyboru znacznika czasowego względem którego nastąpi porządkowanie. W przyjętym modelu tem-

poralnym krotka ma jeden wymiar czasowy, w związku z czym działanie operatora można przedstawić jak na rysunku 6.

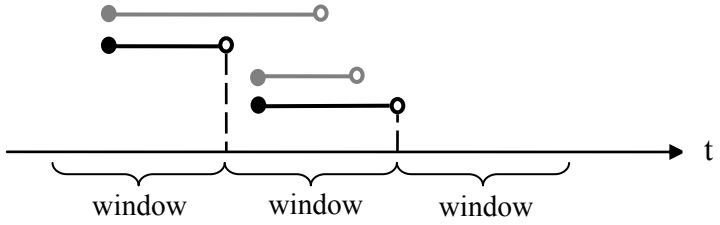


Rys. 6. Działanie operatora SlidingWindow
Fig. 6. SlidingWindow principle of operation

Kolorem szarym zaznaczono krotki, które zostały przesłane na wejście operatora, a czarnym – te same krotki w postaci wyjściowej. Jak widać, czas życia krotek jest identyczny, równy parametrowi *window*.

Okno stało-czasowe. Drugi typ okna czasowego dostępny w systemie to okno stało-czasowe, które wyznacza koniec życia krotki jako najbliższą wielokrotność *window*, która jest większa od t_s. Opis ten jest odzwierciedleniem zależności, którą przedstawia wzór:

$$FixedWindow_{\omega}(S) = \left\{ \begin{array}{c} (t.t_s, (n+1)\cdot\omega, t.Data) \mid \\ (t \in S \wedge t_s \geq n\cdot\omega \wedge t_s < (n+1)\cdot\omega \end{array} \right\} \tag{4}$$



Rys. 7. Działanie operatora FixedWindow
Fig. 7. FixedWindow principle of operation

Tak samo jak wcześniej szare krotki mają postać wejściową, a czarne prezentują długość czasu życia, która charakteryzuje zmodyfikowaną krotkę. W tym typie okna długość życia krotki zależy od podanej wartości parametru *window* oraz znacznika początkowego krotki.

4. Proces testowania

Niniejszy rozdział przedstawia przygotowanie do fazy testowania *AGKPStream*. W ramach podjętych testów sprawdzane były wartości trzech metryk, określonych zgodnie z [5], tj. zajętość pamięci, czas odpowiedzi oraz spowolnienie.

Metryki testowe. *Zajętość pamięci* określa liczba krotek, które aktualnie znajdują się w *AGKPStream*. Dane niezbędne do wyznaczenia wartości tej metryki oraz jej zmienności w czasie są pobierane co określony interwał czasu. Czas odpowiedzi krotki R_i jest równo-

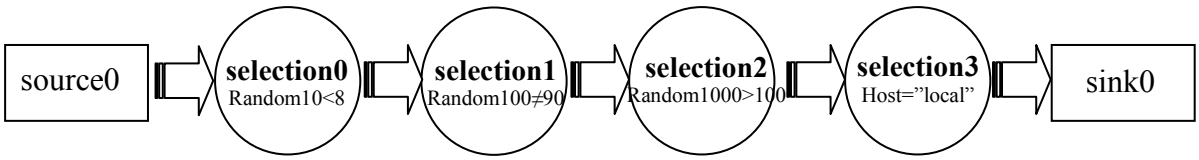
znaczny z czasem, który krotka spędziła w systemie *AGKPStream*, tj. była przetwarzana. Stąd wyznaczamy go jako różnicę czasu wyjściowego krotki D_i i jej czasu wejściowego A_i.

Czas odpowiedzi dla krotki o *id* = *i* można zdefiniować jak następuje: $R_i = D_i - A_i$.

Spowolnienie odnosi się do opóźnienia, jakiego krotka doświadcza podczas przetwarzania przez system. Oblicza się je jako iloraz czasu odpowiedzi krotki i idealnego czasu przetwarzania. Idealne spowolnienie wynosi 1, co oznacza, że krotka została przetworzona bez zbędnego oczekiwania.

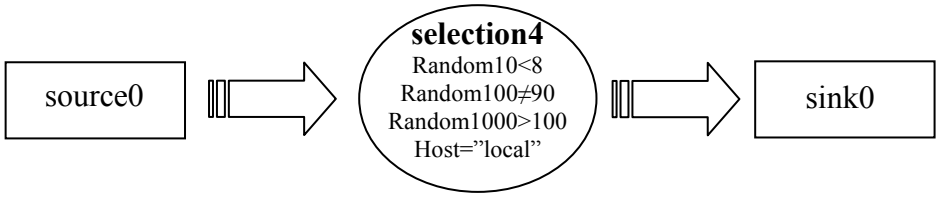
Testy operatora selekcji. Stworzona implementacja operatora selekcji umożliwia zdefiniowanie wielu warunków, które będą filtrować przesłane krotki. W pierwszej fazie testów sprawdzana jest wydajność pracy systemu *AGKPStream* pod kątem analizowanych metryk w zależności od liczby warunków testowych.

Zapytanie 1 składa się z czterech selekcji, z których każda ma pojedynczy warunek (rysunek 8).



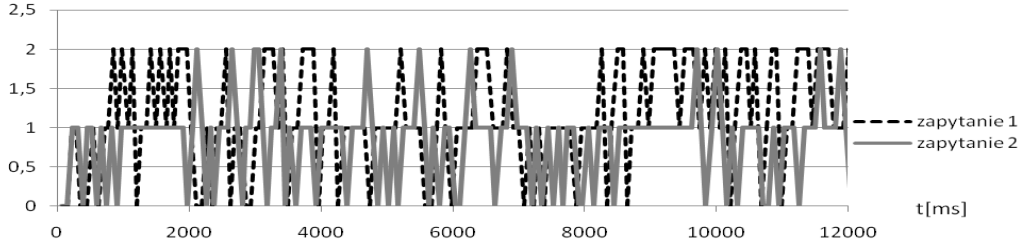
Rys. 8. Zapytanie 1
Fig. 8. Query n^o 1

Zapytanie 2 (rysunek 9) zbudowane jest z jednego operatora selekcji, którego lista predykatów jest zbiorem warunków wszystkich pojedynczych selekcji z poprzedniego zapytania.



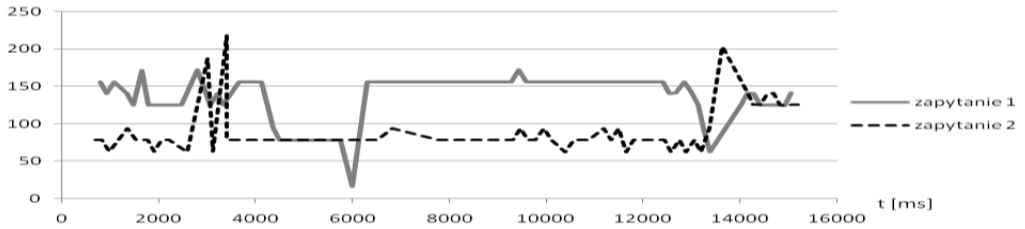
Rys. 9. Zapytanie 2
Fig. 9. Query n^o 2

Celem testów *AGKPStream*, przeprowadzonych na wymienionych zapytaniach, jest sprawdzenie, czy połączenie wielu warunków w jednym operatorze zmienia wartości opisywanych metryk w stosunku do selekcji z pojedynczymi warunkami.



Rys. 10. Zajętość pamięci *AGKPStream* dla zapytań 1 i 2
Fig. 10. *AGKPStream* memory usage for query 1 and 2

Rysunek 10 prezentuje wyniki pomiarów zajętości pamięci, a rysunek 11 – pomiary dla czasu odpowiedzi. Na osi odciętych znajduje się czas pracy systemu *AGKPStream*, a na osi rzędnych pamięci – badana metryka.



Rys. 11. Czasy odpowiedzi na zapytania 1 i 2 *AGKPStream*
Fig. 11. *AGKPStream* response time for query 1 and 2

Przedstawione wyniki nie wskazują jednoznacznie, które zapytanie jest wykonane optymalnie w systemie *AGKPStream*. Jeśli za kryterium oceny przyjąć zajętość pamięci oraz czas odpowiedzi, to należy łączyć warunki selekcji w jednym operatorze, tak jak w *zapytaniu 2*. W przypadku metryki zwanej spowolnieniem dla *zapytania 1* wartości są bardzo małe, zdecydowanie mniejsze od tych dla *zapytania 2*. W efekcie w celu minimalizacji spowolnienia należałoby rozdzielać warunki na pojedyncze selekcje.

5. Podsumowanie

Paradygmat przetwarzania strumieniowego jest stosunkowo nowy i pod wieloma względami stoi w opozycji do klasycznych założeń i rozwiązań z dziedziny baz danych.

W autorskim *AGKPStream* operatory stanowią podstawowe komponenty przetwarzające, których działanie jest kluczowe dla poprawności i efektywności systemu. Szczegóły związane z problemem szeregowania na poziomie operatorów i zapytań w odniesieniu do stworzonego systemu zostały opisane w [1].

AGKPStream może zostać wzbogacony o kolejne operatory strumieniowe, w tym szczególnie interesujące wydają się operatory agregacji oraz okna liczebnościowe. Pierwszy ze wspomnianych operatorów może zostać zaimplementowany na kilka sposobów z wykorzystaniem dostępnych w systemie *AGKPStream* struktur danych i właściwości. Ponadto ważnym elementem, który nie występuje w systemie, jest optymalizator zapytania. Jego zadaniem byłaby modyfikacja struktury wewnętrznej zapytania bez uszczerbku dla danych wyjściowych.

BIBLIOGRAFIA

1. Pasterak K.: Analiza badawczo-implementacyjna wybranych schedulerów strumieniowych. Rozprawa inżynierska, Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Gliwice 2012.
2. Gorawska A.: Analiza i implementacja wybranych operatorów strumieniowych. Rozprawa inżynierska, Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Gliwice 2012.
3. Herud T., Sulski M.: Analiza implementacyjna schedulerów i operatorów dla nowego modelu przetwarzania strumieniowego. Rozprawa inżynierska, Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Gliwice 2011.
4. Chrószcz A.: Model przetwarzania strumieniowego uwzględniający zarówno wpływ synchronizacji jak i język zapytań łączący paradygmaty języka obiektowego i deklaratywnego. Rozprawa doktorska, Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Gliwice 2012.
5. Sharaf M. A., Chrysanthis P. K., Labrinidis A., Pruhs K.: Efficient Scheduling of Heterogeneous Continuous Queries. VLDB Endowment, University of Pittsburgh, 2006.
6. Patroumpas K., Sellis K.: Submisin Multiple Sliding Windows for Shared Stream Computation. ADBIS, 2011.
7. Babcock B., Babu S., Datar M., Motwani R., Thomas D.: Operator scheduling in data stream systems. The VLDB Journal, Stanford University, 2004.
8. Sutherland T. M., Pielech B., Zhu Y., Ding L., Rundensteiner E.A.: An Adaptive Multi-Objective Scheduling Selection Framework for Continuous Query Processing. Worcester Polytechnic Institute, IEEE Computer Society, 2005.
9. Chakraborty A., Singh A.: A Partition-based Approach to Support Streaming Updates over Persistent Data in an Active Data Warehouse. IPDPS, 2009.
10. Stochmiałek M.: Strumieniowe bazy danych STREAM: The Stanford Data Stream Management System. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2005.
11. Barga R. S., Goldstein J., Ali M. H., Hong M.: Consistent Streaming Through Time: A Vision for Event Stream Processing. CIDR, 2007.

Wpłynęło do Redakcji 31 stycznia 2012 r.

Abstract

Data streams will play key role in creating systems responsible for processing multiple data connected with critical events. This data processing paradigm has led to a new generation of systems that can process continuous queries. We show main ideas and foundations of prototype system called *AGKPStream*, that embrace temporal stream model and support stream query features. We present basic concepts, principals of operation of selected stream operators and test results for selection operator.

Adresy

Marcin GORAWSKI: Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, Marcin.Gorawski@polsl.pl; Politechnika Wrocławska, Wydział Informatyki i Zarządzania, Instytut Informatyki, ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Polska, Marcin.Gorawski@pwr.wroc.pl.
Anna GORAWSKA: Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, anna.gorawska@gmail.com.