STUDIA INFORMATICA 2009
Volume 30 Number 2B (84)

Artur WILCZEK, Karol WOŹNIAK
Politechnika Wrocławska, Instytut Informatyki

IMPLEMENTACJA STRUMIENIOWEGO SYSTEMU MONITORUJĄCEGO RUCH NA SERWERZE WWW

Streszczenie. Pojawienie się komercyjnych systemów strumieniowych baz danych ogólnego przeznaczenia otwiera możliwości badań nad ich zastosowaniami w różnorodnych dziedzinach. W niniejszej pracy omówiono implementację systemu monitorującego serwery WWW w czasie rzeczywistym przy wykorzystaniu strumieniowej bazy danych.

Słowa kluczowe: strumieniowe bazy danych, monitorowanie ruchu na serwerze WWW

WEB TRAFFIC MONITORING SYSTEM IMPLEMENTATION BASED ON STREAM MANAGEMENT SYSTEM

Summary. Emerging general purpose data stream management systems made it possible to conduct research focused on various applications. This chapter presents an application of data stream management systems in real-time www server traffic monitoring..

Keywords: data stream management systems, web server traffic monitoring

1. Wprowadzenie

W odrębnej pracy autorzy dokonali przeglądu istniejących narzędzi monitorowania ruchu na serwerach WWW, przedstawili także wymagania, które powinien spełniać strumieniowy system monitorujący. Autorzy doszli do wniosku, że krótki czas reakcji i zdolność do przetwarzania znacznych ilości danych, charakterystyczny dla systemów strumieniowych, może wyeliminować wiele ograniczeń tradycyjnych systemów monitorujących. System monitorujący oparty na strumieniowej bazie danych może spełniać jednocześnie funkcję analizatora

dzienników serwera i wbudowanego systemu monitorującego aktywnie reagującego na zdarzenia, łącząc zalety obu tych rozwiązań.

Wyłoniono podstawowe wymagania funkcjonalne, których implementacja pozwoli ocenić zalety i ewentualne wady stosowanych systemów strumieniowych w monitorowaniu ruchu na serwerach WWW. Należą do nich: badanie rozkładu obciążenia, wykrywanie błędów w przetwarzaniu żądań, wykrywanie „martwych linków", monitorowanie adresów IP zachowujących się podejrzanie oraz identyfikacja sesji. Szersze omówienie tych wymagań znaleźć można w odrębnej publikacji autorów.

2. Założenia dotyczące architektury systemu

Implementowany system monitorujący będzie się składał się z dwóch podstawowych komponentów. Pierwszym z nich będzie moduł akwizycji danych, którego zadaniem będzie pobranie potrzebnych danych z serwera WWW i przesłanie ich do dalszego przetwarzania. Przetwarzanie to będzie wykonywał drugi komponent systemu, będący aplikacją strumieniową działającą pod kontrolą systemu zarządzania strumieniami danych.

Do implementacji modułu akwizycji danych wybrano serwer WWW Apache w wersji 2.2.4 [2]. Apache jest obecnie najbardziej popularnym serwerem WWW. Dodatkowo, jest on dostępny na licencji Open Source, co ułatwia i zachęca do modyfikacji jego działania.

Moduł przetwarzania danych wykorzystuje komercyjny system StreamBase firmy StreamBase Inc.[5]. System ten został stworzony przez osoby, które wcześniej brały udział w projektach badawczych Aurora [3] oraz STREAM [1].

W dalszej części pracy omówiono krótko najważniejsze cechy systemu StreamBase, a następnie przedstawiono implementację poszczególnych komponentów systemu monitorującego.

3. System StreamBase

Podstawowym typem danych w systemie StreamBase jest strumień, czyli ciąg krotek posiadających wspólnych schemat i pojawiających się w systemie w określonym czasie. Strumienie krotek nie tylko przenoszą dane, ale także inicjują wszelkie akcje w systemie. Aplikacje strumieniowe komunikują się ze środowiskiem zewnętrznym za pomocą strumieni danych. Aplikacja strumieniowa definiuje pewną liczbę nazwanych strumieni wejściowych i wyjściowych wraz z ich schematami. Zewnętrzne (klienckie) aplikacje mogą łączyć się z tymi strumieniami, aby wysyłać lub pobierać krotki.

Krotki wprowadzone do strumienia są następnie przetwarzane przez aplikację strumieniową. Aplikacje strumieniowe w systemie StreamBase można tworzyć na dwa sposoby. Podobnie jak w systemie Aurora, możliwe jest wykorzystanie proceduralnego języka zapytań – tworzenie grafu operatorów. Taki graf operatorów nazywany jest diagramem aplikacji EventFlow (ang. *EventFlow application diagram*). Innym sposobem na tworzenie aplikacji jest stosowanie deklaratywnego języka StreamSQL, opartego na języku SQL. Oba te podejścia są sobie równoważne, to znaczy, poza pewnymi szczególnymi przypadkami, dowolną aplikację zapisaną w postaci diagramu EventFlow można zapisać w postaci zapytania języka StreamSQL. W implementacji opisywanego w tej pracy systemu wykorzystano wyłącznie diagramy EventFlow.

System StreamBase umożliwia, niezależnie od wybranego sposobu tworzenia aplikacji, korzystanie z wielu operacji na strumieniach. Do podstawowych operacji należą: filtrowanie strumieni, które usuwa ze strumienia wszystkie krotki niespełniające określonego predykatu, a także operacja *Map* przekształcająca krotki strumienia. Do bardziej złożonych mechanizmów systemu należą ruchome okna (ang. *sliding windows*). Ruchome okno w każdym momencie czasu zawiera pewien zbiór ostatnich krotek strumienia. To, ile krotek znajdzie się w oknie, zależy od jego rodzaju: w przypadku okien czasowych (ang. *time-based*) elementy są dodawane do okna na podstawie znaczników czasowych (np. krotki, które pojawiły się w ciągu ostatniej minuty). Okno krotkowe (ang *tuple-based*) zawiera określoną liczbę ostatnich krotek w strumieniu. Okna mogą podlegać partycjonowaniu, co odpowiada wynikom polecenia GROUP BY w języku SQL.

Możliwe jest również wykonywanie złączeń strumieni danych. Definiować można złączenia na krotkach pochodzących ze strumieni o różnych schematach i opierających się na łączeniu krotek spełniających określone warunki. Ponadto, operatory złączenia mogą ingerować w kolejność krotek w strumieniach, na przykład wprowadzając uporządkowanie krotek.

Oprócz strumieni, w systemie StreamBase istnieją także inne typy danych nazywane *data constructs*, przeznaczone do przechowywania danych w sposób trwały. W dokumentacji systemu dane przechowywane w takich strukturach nazywa się też danymi dzielonymi (ang. *shared data*), gdyż każda taka struktura danych może być dzielona pomiędzy różnymi strumieniami.

Jedną z takich struktur są tabele zapytań (ang. *query tables*). Tabele te w dalszej części pracy będziemy nazywać tabelami natywnymi, gdyż są one zarządzane wewnętrznie przez system StreamBase. Tabela natywna jest w istocie reprezentacją relacji z relacyjnego modelu danych. Posiadają one schemat i klucz główny (ang. *primary index*) oraz pewną liczbę indeksów dodatkowych. Tabele zapytań mogą być przechowywane w pamięci operacyjnej bądź zewnętrznej.

Innym rodzajem tabel w systemie StreamBase są tabele JDBC (ang. *JDBC tables*). Umożliwiają one współpracę z tradycyjnymi serwerami baz danych obsługującymi standard JDBC. Dane są więc przechowywane na zewnątrz systemu strumieniowego. Komunikacja odbywa się za pomocą dialektu języka SQL właściwego dla danego zewnętrznego SZBD.

Zmaterializowane okno (ang. *materialized windows*) umożliwia zdefiniowanie okna na strumieniu i wykonywanie na nim zapytań. Obsługiwane są okna czasowe i krotkowe oraz dodatkowo okna oparte na polach (ang. *field based*). Okno oparte na polach zakłada, że w strumieniu istnieje pole typu numerycznego (*int* lub *double*), którego wartości rosną w kolejnych krotkach. Okno definiowane jest przez rozmiar (nazwijmy go *r*). Do okna należą krotki, dla których wartości wybranego pola są większe od $n - r$, gdzie n jest ostatnią (i najwyższą) wartością w strumieniu. Okno oparte na polach można traktować jak okno czasowe, w którym rolę znacznika czasowego pełni pole typu numerycznego.

Do definiowania operacji na danych w systemie StreamBase wykorzystuje się zestaw wbudowanych funkcji, który jest taki sam niezależnie od tego, czy tworzymy diagram EventFlow czy zapytanie języka StreamSQL. Funkcje te dzielą się na proste – operujące na pojedynczych polach i agregacyjne, które obliczają wartości na podstawie wielu krotek. Zestaw funkcji prostych jest bardzo podobny do występujących w typowych systemach zarządzania bazami danych i obejmuje funkcje operujące na wartościach czasowych, ciągach znaków, wykrywające wartości NULL i tym podobne. Funkcje agregujące (SUM, COUNT, AVG itp.) znane z systemów relacyjnych można w StreamBase stosować zarówno do tabel, jak i do okien. Dodatkowo, wprowadzono specjalne funkcje agregujące, działające wyłącznie na oknach: FIRSTVAL – zwracającą pierwszą wartość określonego pola w oknie i LASTVAL – zwracającą ostatnią wartość.

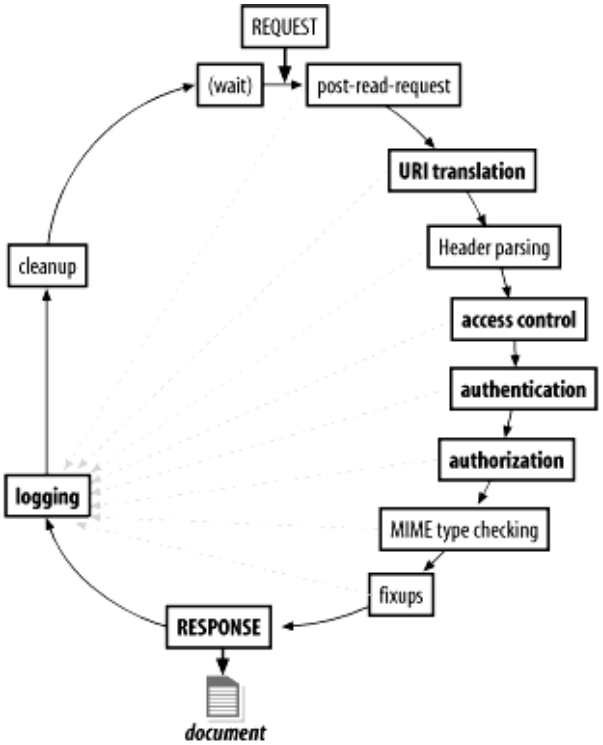
4. Implementacja podsystemu akwizycji danych

Serwer Apache składa się z rdzenia (ang. *core*) oraz zestawu modułów. Rdzeń serwera Apache oczekuje na żądania oraz zarządza ich przetwarzaniem i generowaniem odpowiedzi. Udostępnia też interfejs dla modułów, które rozszerzają jego funkcje. Interfejs modułów, zwany Apache API (od ang. *Application Program Interface*), składa się z zestawu bibliotek APR (ang. *Apache Portable Runtime*) oraz z tak zwanych haków (ang. *Hooks*). Biblioteki APR obudowują funkcje systemu operacyjnego i zapewniają przenośność kodu źródłowego modułów pomiędzy systemami. Zawierają też definicje struktur danych przenoszących informacje o obiektach w systemie, takich jak żądania HTTP, piki konfiguracyjne i tym podobne. Haki natomiast powiązane są z określonymi zdarzeniami podczas działania serwera (wczytanie pliku konfiguracyjnego, uruchomienie wątku potomnego itp.) lub poszczególnymi fazami

przetwarzania żądania (przyjęcie żądania, określenie rodzaju danych, generowanie odpowiedzi, zapis pliku dziennika itp.).

Typowy moduł serwera WWW Apache posiada zestaw funkcji zarejestrowanych dla określonych haków. Funkcje te są następnie w odpowiednich momentach wywoływane przez rdzeń serwera WWW. Moduły tej klasy nie mogą bezpośrednio wywoływać funkcji systemu operacyjnego, a wszystkie potrzebne operacje wykonują za pomocą funkcji biblioteki APR. Dzięki temu moduły te są całkowicie niezależne od systemu operacyjnego. Cykl obsługi żądania jest przedstawiony na rys. 1.

Podsystem akwizycji danych zaimplementowano w postaci modułu serwera Apache działającego w fazie zapisu plików dziennika – we wcześniejszych fazach nie wszystkie istotne dane na temat żądania są określone. Taka architektura potencjalnie umożliwia zastąpienie domyślnego mechanizmu zapisu dzienników przekazywaniem danych do systemu strumieniowego. Zdecydowano się jednak na pozostawienie bez zmian istniejących mechanizmów dla celów testowych i diagnostycznych.



Rys. 1. Cykl działania wątku lub procesu roboczego Apache [4]
Fig. 1. Apache process or thread cycle [4]

Istotnym problemem, jaki należało rozwiązać, było zarządzanie połączeniami z systemem StreamBase. Tworzenie nowego połączenia dla każdego żądania prowadziłoby do straty zasobów, dlatego konieczne było zaimplementowanie mechanizmu współdzielenia połączeń z systemem strumieniowym pomiędzy żądaniami.

Zdecydowano się zatem na wykorzystanie rozwiązania nazywanego pulą połączeń (ang. *connection pool*). Polega ono na tym, że w momencie uruchomienia serwera WWW tworzo-

na jest określona liczba połączeń z bazą danych. Kiedy zachodzi potrzeba wysłania danych, wątki lub procesy robocze pobierają z puli połączenie, wykorzystują je, a następnie zwracają do puli. Wadą tego rozwiązania jest fakt, że jeśli liczba połączeń w puli jest niższa niż liczba wątków, to może pojawić się sytuacja, w której wątek oczekuje na wolne połączenie. Jednocześnie, może okazać się, że wystarczająca jest mniejsza liczba połączeń, które są dzielone pomiędzy wątki robocze, co prowadzi do mniejszego zużycia pamięci operacyjnej.

Moduł zaimplementowano tak, aby dla każdego żądania obsługiwanego przez serwer przekazywał do systemu zarządzania strumieniami danych jedną krotkę o schemacie, który jest zgodny ze schematem opisanym w tabeli 1.

Pewne wątpliwości pojawiły się przy tych polach strumienia, które reprezentują nagłówki HTTP żądania, to jest pola referer i user_agent. Specyfikacja protokołu HTTP określa maksymalną długość nagłówka na 4 KB. W poniższym schemacie występują dwa takie pola, więc powinno się na nie zarezerwować co najmniej 8 KB pamięci. Dokumentacja techniczna systemu StreamBase ostrzega jednak, że system ten w obecnej wersji niezbyt dobrze radzi sobie z efektywnym przetwarzaniem strumieni, których krotki są tej wielkości. Ponadto, praktyka wykazuje, że zawartość tych nagłówków z reguły jest dużo krótsza. Zdecydowano się zatem ograniczyć długość ciągów znaków reprezentujących te pola strumienia tak, aby mieściły się w nich typowe wartości odpowiednich nagłówków. Podobne względy zadecydowały o ograniczeniu długości pola URI.

Ostateczny schemat strumienia przyjęty w implementacji jest opisany w poniższej tabeli. Pola referer i user_agent przyjmują wartość NULL, jeśli żądanie klienta nie zawiera odpowiadających im nagłówków HTTP.

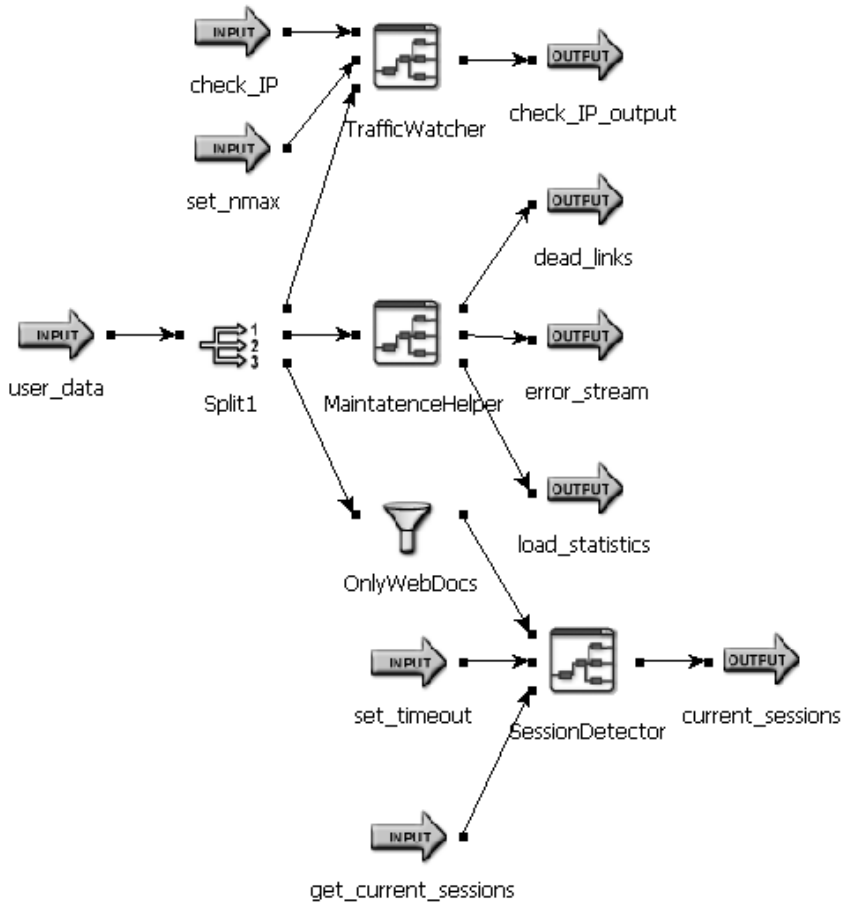
Tabela 1

Schemat strumienia user_data

Nazwa pola	Znaczenie	Typ	Wartość NULL
remote_ip	Numer IP klienta	String(15)	nie
time	Czas otrzymana żądania przez serwer WWW	Timestamp	nie
uri	URI żądanego dokumentu	String(128)	nie
method	Metoda żądania (patrz [RFC2616], rozdział 9)	String(7)	nie
status	Kod stanu	String(3)	nie
bytes_sent	Liczba bajtów odpowiedzi (nie licząc nagłówków HTTP)	Int	nie
referer	Zawartość pola Referer żądania	String(128)	tak
user_agent	Informacja o przeglądarce i systemie operacyjnym klienta	String(64)	tak

5. Implementacja podsystemu przetwarzania danych

Podsystem przetwarzania danych przyjmuje strumień user_data generowany przez moduł akwizycji danych i na tej podstawie wykonuje wcześniej określone akcje. Podsystem ten został podzielony na trzy moduły realizujące poszczególne funkcje. Moduł Maintanence-Helper wykonuje zadania wspierające administracje systemu: wykrywanie błędów w przetwarzaniu żądań HTTP, wykrywanie martwych linków, a także oblicza rozkład ruchu na serwerze w czasie. Moduł TrafficWatcher prowadzi monitorowanie adresów IP klientów zachowujących się podejrzanie. Moduł SessionDetector natomiast identyfikuje sesje użytkowników. Ogólny widok systemu przedstawia rys. 2.



Rys. 2. Ogólny widok modułu przetwarzania danych
Fig. 2. Main view of data processing module

5.1. Moduł SessionDetector

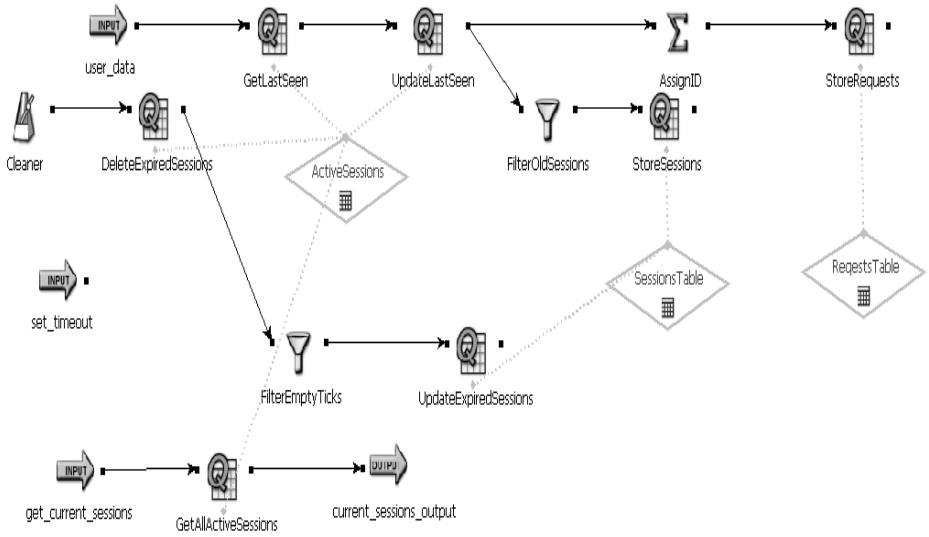
Działanie modułu SessionDetector opiera się na tabelach. Utrzymuje on przechowywaną w pamięci RAM tabelę natywną ActiveSessions, która zawiera wszystkie aktualnie aktywne sesje oraz utrzymywane w pamięci trwałej tabele SessionsTable i RequestsTable, które przechowują dane o wszystkich sesjach. Należy zwrócić uwagę na to, że w tym przypadku znacz-

nie lepiej spisywałyby się tabele JDBC składowane w relacyjnej bazie danych, gdyż przed skorzystaniem z tych tabel należy wykonać na nich złączenie, a system StreamBase nie potrafi wykonywać złączeń. Niestety, dostępna wersja systemu StreamBase nie obsługuje tabel JDBC.

Kiedy krotka strumienia user_data trafia do modułu SessionDetector, w pierwszej kolejności kierowana jest do operatora *Query* GetLastSeen, który wyszukuje w tabeli ActiveSessions aktywną sesję, do której można przyporządkować daną krotkę. Jeśli operator nie znajdzie właściwego wiersza, to emituje krotkę, która zawiera dane odpowiadające nowej sesji. Operator UpdateLastSeen aktualizuje pole last_seen wiersza tabeli reprezentującego określoną sesję lub dodaje do tabeli nowy wiersz.

Następnie, strumień rozwidla się. Jedna z gałęzi trafia do operatora FilterOldSessions, który odrzuca wszystkie żądania będące kontynuacją aktywnych sesji. Pozostałe krotki trafiają do operatora StoreSessions, który składuje je w tablicy SessionsTable. Czas zakończenia sesji nie jest jeszcze znany, więc pole session_end otrzymuje tymczasowo wartość NULL.

Druga z gałęzi trafia do operatora AssignID. Jest to operator *Aggregate,* który jest skonfigurowany, tak aby nadawał trafiającym do niego krotkom kolejne numery. Krotka wraz z numerem trafia następnie do operatora StoreRequests, który dodaje informacje o żądaniu do tabeli RequestsTable.



Rys. 3. Moduł SessionDetector
Fig. 3. SessionDetector module

Operator Cleaner służy do usuwania nieaktywnych sesji z tabeli ActiveSessions. Generuje on w regularnych odstępach czasu jedną krotkę. Krotka ta trafia do operatora DeleteExpiredSessions, który uruchamia operację kasowania wierszy z tabeli ActiveSessions. Dla każdego usuniętego wiersza operator ten generuje jedną krotkę zawierającą dane sesji. Jeśli żaden wiersz tabeli nie został usunięty, to emitowana jest jedna krotka, której wszystkie pola mają wartość NULL. Taka krotka jest następnie usuwana ze strumienia przez operator

FilterEmpytTicks. Pozostałe krotki trafiają do operatora UpdateExpiredSessions i używane są do uzupełnienia wartości pola session_end.

Warto zwrócić uwagę na to, że taki sposób implementacji może powodować niewielkie błędy, wynikające z faktu, że aktywne sesje usuwane są tylko wtedy, gdy operator Cleaner wyemituje krotkę. Nowe żądanie może zostać dodane do sesji, która zgodnie ze specyfikacją jest już nieaktywna, ale reprezentujący ją wiersz nie został jeszcze usunięty z tabeli ActiveSessions. Aby ograniczyć ten błąd, krotki inicjujące czyszczenie są generowane często (domyślnie co 10 sekund). Ponieważ sesje dezaktualizują się po czasie rzędu kilku minut, błąd ten jest pomijalny.

Strumień wejściowy GetCurrentSessions ma za zadanie uruchomienie odczytu wszystkich wierszy z tabeli ActiveSessions, która ma taki sam schemat jak strumień wyjściowy current_sessions_output. Jeśli pojawi się w nim krotka, to spowoduje to wysłanie całej zawartości tej tabeli na strumień wyjściowy CurrentSessionsOutput.

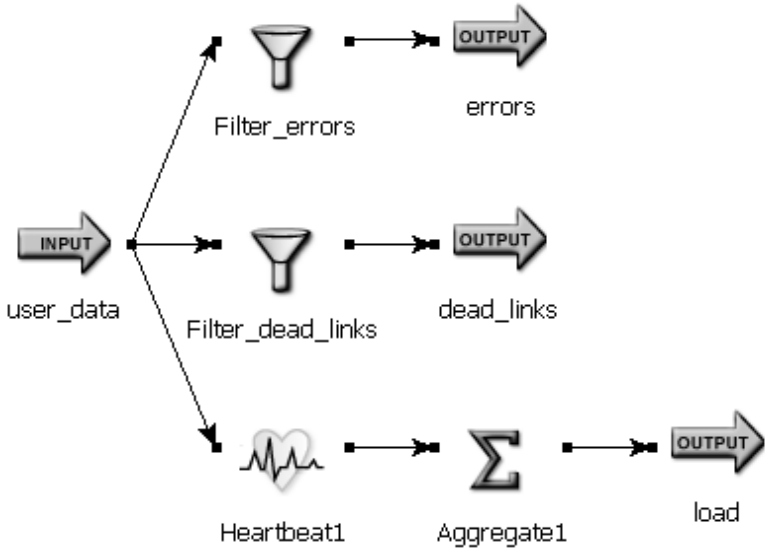
Strumień SetTimeout jest powiązany ze zmiennymi dynamicznymi w tym module i służy do zmiany wartości dozwolonego czasu pomiędzy żądaniami w jednej sesji. Zauważyć można, że wykorzystanie strumienia do zmiany wartości zmiennych dynamicznych nie jest uwidocznione na diagramie – widać to wyłącznie w zakładce Dynamic Variables odpowiednich operatorów (rys. 3).

5.2. Moduł MaintananceHelper

Moduł MaintenanceHelper generuje strumienie wyjściowe informujące o różnych rodzajach błędów oraz o obciążeniu systemu. Generowanie strumieni wyjściowych errors oraz dead_links okazało się trywialne – wystarczyło zastosowanie pojedynczego operatora *Filter* dla każdego z nich.

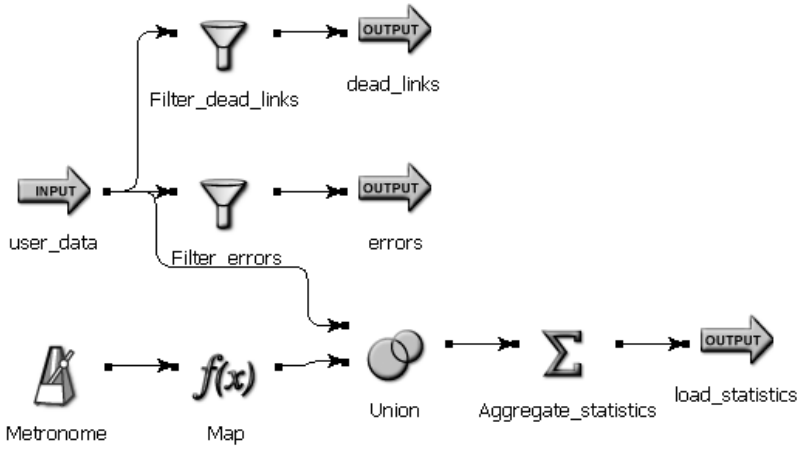
Nieco bardziej złożone było wygenerowanie strumienia load, zawierającego liczbę obsłużonych żądań i sumę wielkości wszystkich odpowiedzi serwera w ciągu każdej minuty. Do obliczenia tych sum wykorzystano operator *Aggregate*, który oblicza agregacje na podstawie okna czasowego o długości 60 sekund i emituje krotkę wynikową kiedy okno zamyka się.

Należy zwrócić uwagę na to, że jeśli w strumieniu wejściowym operatora *Aggregate* przez ponad 60 sekund nie pojawi się żadna krotka, to operator ten nie wygeneruje w tym czasie krotek wynikowych. W związku z tym zastosowano operator *Heartbeat*, który co 60 sekund doda do strumienia wejściowego krotkę, której wszystkie pola mają wartość NULL. Ta krotka nie wpłynie na wynik agregacji, ale w razie konieczności spowoduje jej obliczenie i przekazanie wyniku. Rozwiązanie takie jest przedstawione na rys. 4.



Rys. 4. Moduł MaintananceHelper z operatorem Heartbeat
Fig. 4. MaintananceHelper module with Heartbeat operator

W czasie testowania modułu okazało się, że operator *Heartbeat* nie emituje żadnych krotek wyjściowych do czasu, kiedy otrzyma pierwszą krotkę wejściową – można odnieść wrażenie, że dopiero otrzymanie krotki wejściowej „uruchamia" ten operator. Nie udało się ustalić, czy takie zachowanie jest zamierzone, czy też wynika ze sposobu implementacji obecnej wersji systemu. Takie zachowanie jest niezgodne z dokumentacją techniczną systemu StreamBase.

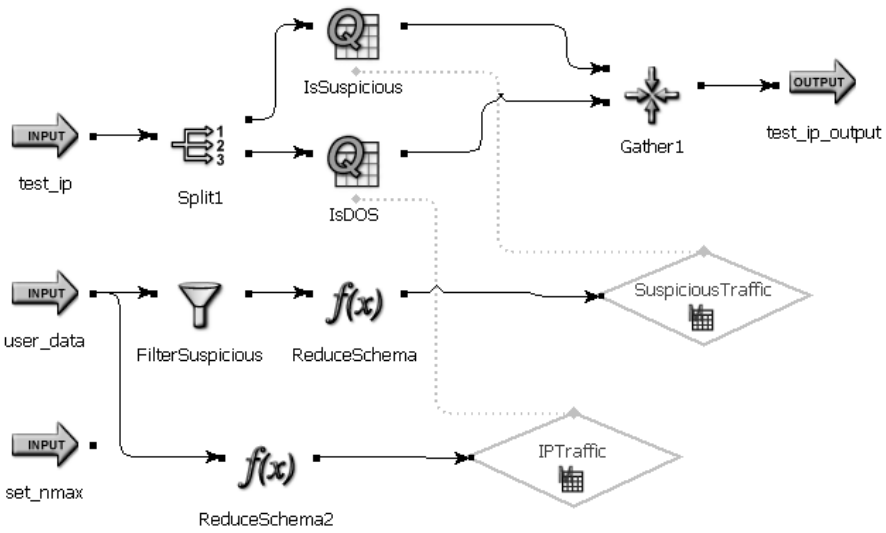


Rys. 5. Ostateczna wersja modułu MaintananceHelper
Fig. 5. MaintananceHelper module final version

W związku z powyższym, aby uzyskać zakładaną funkcjonalność, zdecydowano się zastosować rozwiązanie przedstawione na rys. 5. W przedstawionym rozwiązaniu wykorzystano operator *Metronome* generujący krotkę co 60 sekund. Następujący po nim operator *Map* przekształca krotkę tak, aby miała taki sam schemat jak krotki strumienia user_data, a wartości wszystkich pól były równe NULL. Powstały strumień krotek jest następnie łączony ze strumieniem user_data za pomocą operatora *Union*, co imituje działanie operatora *Heartbeat*.

5.3. Moduł TrafficWatcher

Moduł TrafficWatcher monitoruje ruch i wyszykuje w nim zdarzenia, mogące potencjalnie zagrażać bezpieczeństwu serwera, wspomagając tym samym zewnętrzny system utrzymujący jego bezpieczeństwo. Moduł ten wykorzystuje dwa okna zmaterializowane: IPTraffic i SuspiciousTraffic. Oba te okna są oknami pływającymi, opartymi na czasie (ang. *Time--based sliding windows*).



Rys. 6. Modułu TrafficWatcher
Fig. 6. TrafficWatcher module

Do okna IPTraffic trafiają wszystkie krotki strumienia user_data. Okno to jest wykorzystywane do sprawdzania, czy któryś z klientów nie generuje nadmiernej liczby żądań. Aby ułatwić wyszukiwanie odpowiednich numerów IP, okno to ma zdefiniowany indeks na polu remote_ip. Do drugiego z okien trafiają tylko krotki, które operator FilterSuspicious uzna za podejrzane, na przykład dlatego, że opisują próby dostępu do chronionych zasobów serwera. Operatory ReduceSchema i ReduceSchema2 usuwają z krotek niepotrzebne pola, co zmniejsza wymaganą do przechowywania okien ilość pamięci.

Strumień wejściowy test_IP służy do wydobywania z systemu informacji na temat podejrzanych numerów IP. Każda pojawiająca się na nim krotka jest przekazywana do operatorów IsDOS i IsSuspicious, które sprawdzają, czy adres, którego dotyczy zapytanie, znajduje się w którymś z okien, a następnie dodają do schematu pole zawierające wartość logiczną informującą o wyniku tej operacji. Operator *Gather* łączy te krotki na podstawie numeru IP oraz na podstawie dodanych pól sprawdza, czy numer ten jest podejrzany. Wynik operacji zwracany jest przez strumień test_ip_output.

Dodatkowy strumień wejściowy set_nmax jest wykorzystywany do zmiany wartości zmiennej dynamicznej n_{max}.

6. Podsumowanie

Omówiony w niniejszej pracy system monitorujący łączy w sobie cechy pochodzące z różnych obszarów zastosowań, od czysto technicznych związanych z administracją serwisem do typowo biznesowych, takich jak identyfikacja użytkowników. Z konieczności, funkcjonalność systemu w każdej z wymienionych dziedzin jest ograniczona. Jednakże, dzięki architekturze systemu dodawanie nowych funkcji może być bardzo proste, gdyż zmiany są ograniczone do modułu przetwarzania danych znajdującego się na serwerze strumieniowym. Fakt, że moduł ten wykorzystuje wysoce wyspecjalizowany język operatorów o dużych możliwościach, dodatkowo ułatwia takie modyfikacje. Elastyczność systemu może być szczególnie przydatna w warunkach intensywnego rozwoju usług wykorzystujących WWW.

Niestety, w ramach niniejszej pracy nie można było opublikować wyników badań efektywnościowych systemu z powodu ograniczeń dotyczących publikacji wyników narzuconych przez firmę StreamBase Inc. Firma ta zgodziła się na udostępnienie swojego systemu tylko pod warunkiem pisemnego zobowiązania się autorów do niepublikowania wyników testów wydajnościowych w żadnej postaci.

BIBLIOGRAFIA

1. Arasu A., Babcock B., Babu S., Datar M., Ito K., Nizhizawa I., Rosenstein J., Widom J.: STREAM: The Stanford Stream Data Manager. In ACM SIGMOD Conference, June 2003.
2. Apache, Apache http Server Documentation, http://httpd.apache.org/docs/2.2/logs.html.
3. Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., Zdonik S.: Monitoring Streams: A New Class of Data Management Applications. In proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02), Hong Kong, China, 2002.
4. Stein L.; MacEachern D.: Writing Apache Modules with Perl and C. O'Reilly & Associates, Sebastopol, 1999.
5. StreamBase Inc., StreamBase Documentation, http://streambase.com/developers/docs-/latest/.

Recenzent: Dr inż. Michał Kawulok

Wpłynęło do Redakcji 1 lutego 2009 r.

Abstract

This paper presents implementation of a web traffic monitoring system based on StreamBase stream managements system and Apache www server. Thanks to system architecture adding new requirements implementation can be easily achieved. Changes that should by applied are limited to data processing module implemented on stream processing system. What makes introducing new functions even more easy is very specialized language utilized by this module. System flexibility can be essential in the face of intensive web systems services development.

Adresy

Artur WILCZEK: Politechnika Wrocławska, Instytut Informatyki,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Polska, artur.wilczek@pwr.wroc.pl.
Karol WOŹNIAK, Politechnika Wrocławska, Instytut Informatyki,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Polska, karol.w.wozniak@gmail.com.