دانشکده مهندسی کامپیوتر

پردازش داده‌های جریانی با ابزار WSO2

گزارش پروژه درس سیستم‌های توزیع‌شده

در رشته مهندسی کامپیوتر گرایش نرم‌افزار

نام اعضای گروه:

آرین ابراهیم‌پور

احمد فاضلی

حمیدرضا محمدی

زهرا اخگری

استاد راهنما:

دکتر محسن شریفی

دکتر علی جعفری

دی ماه ۱۳۹۸


بسم الله الرحمن الرحيم

چکیده

DEBS Grand Challenge یک چالش سالیانه برای پردازش رویدادهای داده‌های واقعی است که توسط کنفرانس Distributed Event Based Systems ارسال می‌شود. چالش سال ۲۰۱۵ از مجموعه داده‌های سفرهای تاکسی‌های شهر نیویورک استفاده می‌کند که شامل ۱۷۳ میلیون رویداد جمع‌آوری شده در طول سال ۲۰۱۳ است. در این گزارش، چگونگی استفاده از WSO2 CEP، یک موتور پردازش رویدادهای پیچیده بیان شده‌است که به‌صورت متن‌باز در دسترس است. همچنین در این گزارش، راه‌حل و نتایج به تفصیل بیان شده‌است و در جهت بهینه‌سازی راه‌حل با هدف افزایش کارآیی، بحث شده‌است.

واژه‌های کلیدی: DEBS Grand Challenge، پردازش رویدادهای پیچیده، WSO2 CEP، بهینه‌سازی.

فهرست مطالب

فهرست اشکال

فهرست جداول

فصل ۱:

مقدمه

۱-۱- مقدمه

هر سال، چالش بزرگ DEBS به منظور ارائه‌ی یک زمینه‌ی مشترک برای رقابت میان محققان برگزار می‌شود. این چالش، با هدف کمک به سیستم‌های مبتنی بر رویداد[1] صنعتی و تحقیقاتی برپا می‌شود.
هدف چالش DEBS2015، ارزیابی سیستم‌های مبتنی بر رویداد در زمینه‌ی تحلیل جریان داده‌های حجیم جغرافیایی[2] به‌صورت بلادرنگ[3] است. داده‌های انتخابی برای تحلیل، شامل گزارشات سفرهای تعدادی از تاکسی‌ها در شهر نیویورک است. اهداف کلی این رقابت، شامل موارد زیر است:

- شناسایی ۱۰ مسیر پر تردد اخیر در پنجره‌ی زمانی[4] ۳۰ دقیقه.
- شناسایی ۱۰ منطقه‌ی سودآور در پنجره‌ی زمانی ۳۰ دقیقه.

برای یافتن این دو سناریو، باید به تجزیه و تحلیل اطلاعات مکانی تاکسی‌ها و دیگر اطلاعات داده‌شده در مسئله بپردازیم. هدف این پژوهش[1]، یافتن نتایج با حداکثر بازدهی[5] و حداقل تاخیر[6] است.
برای سناریوهای ذکر شده در بالا، از سیستم‌های متعددی می‌توان استفاده کرد. برای نمونه، می‌توان به سیستم‌های Apache Storm، Spark Streaming، Esper و StreamBase اشاره نمود[3][2]. برخی از این سیستم‌ها در مقالات دیگر استفاده شده‌است. همچنین برخی از آن‌ها، برای پردازش رویدادهای پیچیده مناسب نیستند. برای مثال سیستم Apache Storm، برای پردازش رویدادها کند عمل می‌کند و تنها می‌تواند تعداد ۱۰ هزار رویداد بر ثانیه را پردازش کند.
در مقاله‌ی [1]، از سیستم WSO2 CEP برای پردازش رویدادها استفاده شده‌است. این سیستم، قابلیت پردازش ۴۰۰ هزار رویداد در ثانیه را دارد و نویسندگان مقاله‌ی [1]، توانسته‌اند با ایجاد چندین افزونه[7]، حدود ۰,۳ میلیون رویداد در ثانیه را پردازش کنند. در بخش (۲-۱)، به معرفی این ابزار می‌پردازیم.

1 Event-Based System
2 Geo-spatial
3 Real-time
4 Sliding window
5 Throughput
6 Latency
7 Extension

۲-۱- معرفی محصول WSO2

موتور[1] WSO2 CEP، یک سرویس‌دهنده برای پردازش رویدادهای پیچیده است که توسط شرکت WSO2 ارائه شده‌است. از ویژگی‌های این ابزار، می‌توان به متن‌باز بودن،[2] کاربری آسان، مقیاس‌پذیر بودن[3] و سبک‌بودن آن اشاره کرد. این ابزار، می‌تواند رویدادهای معنادار را شناسایی کند، اثر آن‌ها را تحلیل کند و به‌صورت بلادرنگ بر روی آن‌ها عملیاتی را انجام دهد. WSO2 CEP امکاناتی را برای کاربران فراهم کرده که بتوانند با کمک زبان پرس‌وجویی[4] مشابه SQL به جریان داده‌های ورودی گوش کرده و در صورتی که این داده‌ها شرایط موجود در پرس‌وجو را داشته‌باشند، رویداد[5] جدید تولید کند. در جدول (۱-۱) برخی از ویژگی‌های این ابزار بیان شده‌است.

جدول (۱-۱) برخی از ویژگی‌های ابزار WSO2 CEP

ویژگی‌ها	توضیحات
موتور پردازش با عملکرد بسیار بالا	- پردازش بیش از 100K رویداد در ثانیه به‌صورت تک ماشین - طراحی‌شده توسط WSO2 Siddhi
زبان پرس‌وجوی قدرتمند و گسترده	- زبان پرس‌وجو مشابه SQL - فیلتر کردن رویدادها با شرایط ذکرشده - امکان پارتیشن‌بندی داده‌ها برای پشتیبانی از پردازش موازی - اجرای پرس‌وجوها با کمک پنجره‌های زمانی مختلف
امکانات اجرای کاربرپسند	- امکان استفاده کاربران از فرم‌هایی به جای استفاده

1 WSO2 Complex Event Processor

2 Open source

3 Scalable

4 Query language

5 Event

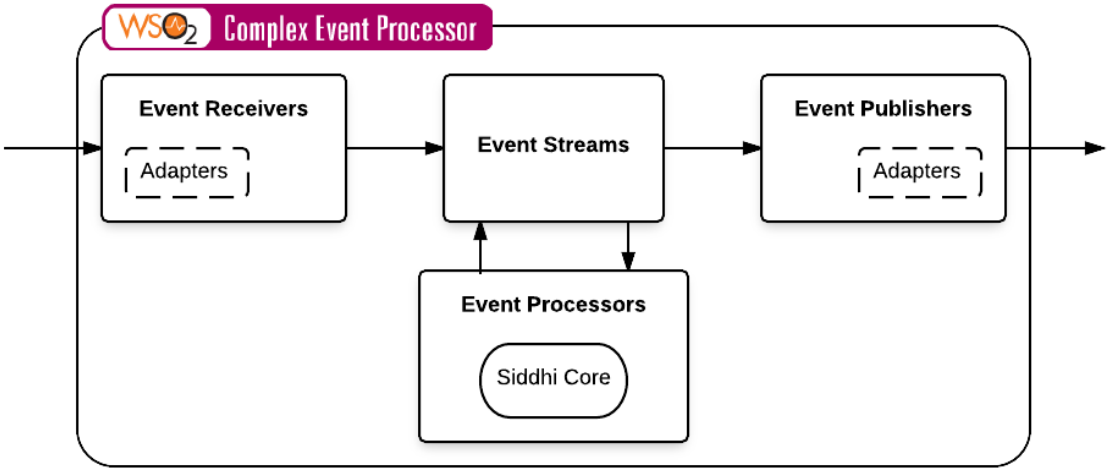
	از پرس‌و جوهای SQL. - طراحی پرس‌وجوها به‌گونه‌ای که پیچیدگی‌های مسئله پنهان شده‌است.
ادغام آسان سیستم با دیگر سیستم‌ها برای ثبت و ضبط رویدادها	- Restful HTTP protocol - SOAP protocol - Kafka, MQTT, File, Socket, Email protocol
پشتیبانی از چندین مکانیسم برای هشدار دادن	- XML, json, Map, Text events via JMS protocol - Email, SMS notification - RESTful, Web services - Kafka, MQTT, File, Web socket protocols
توسعه‌ی آسان	
مقیاس‌پذیر	- پردازش بلادرنگ توزیع‌شده - بخش‌بندی و تقسیم کوئری‌ها در میان چندین سرویس‌دهنده
پشتیبانی از مدل‌های رویداد پیچیده	- پشتیبانی از ویژگی‌های مختلف همچون Float، String و Boolean.

در طی سالیان گذشته، شرکت WSO2، به‌روزرسانی‌هایی بر روی محصول خود انجام داده‌است. در حال حاضر، محصول WSO2 Streaming Integrator، جدیدترین محصول شرکت WSO2 است. کاربرد این محصول مشابه محصول قبلی است با این تفاوت که بر روی محصول جدید، بهینه‌سازی‌هایی انجام شده‌است. ما در این پروژه، از محصول WSO2 Streaming Integrator استفاده می‌کنیم.

در بخش (۱-۲-۱) توضیحی درباره معماری WSO2 CEP بیان شده‌است. در بخش (۲-۲-۱) توضیح مختصری از WSO2 Streaming Integrator داده شده‌است. در بخش (۳-۱-۲) نیز درباره زبان این محصول توضیح داده شده‌است.

۱-۲-۱- معماری WSO2 CEP

در شکل (۲-۱) ، معماری این ابزار و بخش‌های آن نمایش داده شده‌است.



شکل (۲-۱) معماری WSO2 CEP

معماری WSO2 CEP شامل بخش‌های زیر است:

- گیرنده رویداد:[1] گیرندگان رویداد، وظیفه‌ی دریافت رویدادهایی را دارند که به سمت WSO2 CEP می‌آیند. این گیرندگان، می‌توانند انواع مختلفی داشته باشند و رویدادها را با پروتکل‌های مختلفی دریافت کنند. در زیر، تعدادی از انواع مختلف گیرندگان رویداد بیان شده‌است:

  - Email Event Receiver
  - File-tail Event Receiver
  - HTTP Event Receiver
  - JMS Event Receiver
  - Kafka Event Receiver
  - MQTT Event Receiver
  - SOAP Event Receiver
  - WebSocket Event Receiver
  - WebSocket Local Event Receiver
  - WSO2Event Event Receiver

[1] Event Receiver

ما در این پروژه، از گیرنده‌ی رویداد با نوع File-tail استفاده می‌کنیم. این گیرنده، رویدادها را از یک فایل ورودی می‌خواند و آن را ذخیره می‌کند. این نوع، تنها از ورودی متن پشتیبانی می‌کند.

- جریان‌های رویداد:[1] جریان رویداد، شامل مجموعه‌ی منحصربفرد از ویژگی‌ها با انواع خاص است. این جریان‌ها، کمک می‌کنند تا ساختاری فراهم شود که بر اساس آن، رویدادهای پردازش‌شده انتخاب شود.

- پردازش‌کننده‌ی رویداد:[2] پردازشگر رویداد، وظیفه‌ی پردازش اصلی رویداد را برعهده دارد و به عبارتی دیگر، واحد اصلی پردازش رویداد CEP است. این واحد، پردازش رویداد را با کمک کوئری‌های Siddhi انجام می‌دهد. فرآیند پردازش رویداد در این واحد به شرح زیر است:

۱. این بخش، مجموعه‌ای از جریان‌های رویداد (Event Stream) را از واحد مدیریت جریان رویداد (Event Stream Manager) دریافت می‌کند.
۲. با استفاده از موتور Siddhi، پردازشی بر روی آن‌ها انجام می‌دهد.
۳. رویدادهای جدید را به بخش Event Stream Manager باز می‌گرداند.

- منتشرکننده‌ی رویداد:[3]این بخش، رویدادها را به سیستم‌های خارجی ارسال کرده و داده‌ها را در پایگاه‌داده برای تحلیل‌های بیشتر ذخیره می‌کند. همانند بخش دریافت‌کننده‌ی رویداد، این بخش هم آداپتورهای مختلف برای پیاده‌سازی دارد. از جمله:
  - Cassandra Event Publisher
  - Email Event Publisher
  - HTTP Event Publisher
  - JMS Event Publisher
  - Kafka Event Publisher
  - Logger Event Publisher
  - MQTT Event Publisher
  - RDMS Event Publisher
  - SMS Event Publisher

1 Event Streams

2 Event Processor

3 Event Publisher

- SOAP Event Publisher
- UI Event Publisher
- WebSocket Event Publisher
- WebSocket Local Event Publisher
- WSO2Event Event Publisher

۲-۲-۱- محصول WSO2 Streaming Integrator

WSO2 SI یک سرویس‌دهنده پردازش داده‌های جریانی است که توسط شرکت WSO2 ارائه شده‌است و به کاربران اجازه می‌دهد داده‌های جریانی را جمع‌آوری کرده و بر روی آن‌ها عملیاتی را انجام دهند. WSO2 SI یک محصول متن‌باز برای پردازش داده‌های جریانی است که با استفاده از زبانی مشابه زبان SQL قابل استفاده است. این زبان، با نام SiddhiQL شناخته می‌شود.

با استفاده از ساختار و کوئری‌های Siddhi کارهای زیر قابل انجام است:

- تبدیل داده:[۲] تبدیل داده از یک نوع به دیگر نوع.
- غنی‌سازی داده:[۳] دریافت داده از یک منبع مشخص و ترکیب آن با پایگاه‌داده‌ها، سرویس‌ها برای محاسبات.
- همبستگی:[۴] اتصال داده‌های جریانی با چندین جریان برای ایجاد یک جریان داده واحد.
- پاکسازی:[۵] فیلتر کردن و اصلاح محتویات پیام‌ها.
- خلاصه‌سازی:[۶] خلاصه‌سازی داده‌ها در پنجره‌های زمانی مشخص.

1 WSO2 Streaming Integrator

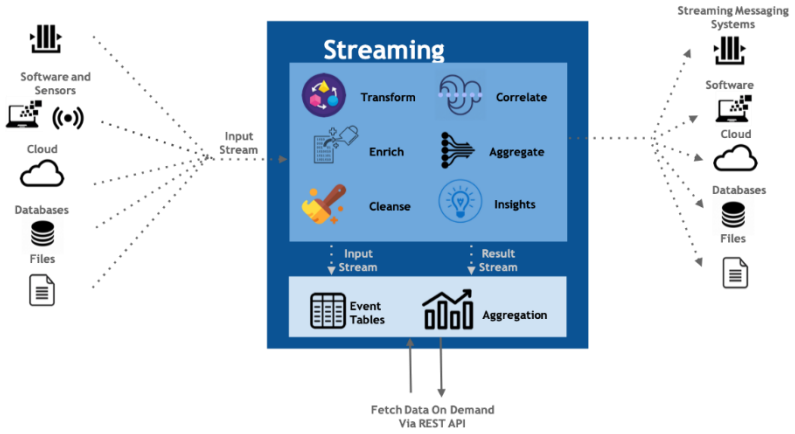
2 Transforming Data

3 Enriching Data

4 Correlating

5 Cleaning

6 Summarizing



شکل (۳-۱) نمای کلی از WSO2 SI

WSO2 SI با کمک REST API، داده‌های جریانی را جمع‌آوری می‌کند و بر روی آن‌ها، کوئری‌ها را اجرا می‌کند و اطلاعاتی را تولید می‌کند.

نصب و پیاده‌سازی این محصول به‌سادگی امکان‌پذیر است. برای نصب این ابزار، دو نسخه برای دانلود موجود است:

- WSO2 Streaming Integrator
- WSO2 Tooling (نسخه‌ی گرافیکی ابزار)

۳-۲-۱- زبان پرس‌وجوی Siddhi

WSO2 SI، انواع مختلفی از کوئری‌ها را پشتیبانی می‌کند. برای پیاده‌سازی این کوئری‌ها، از زبان SiddhiQL استفاده می‌شود. این زبان شامل انواع مختلفی از عملیات است:

- Filter: این عملیات برای فیلتر کردن ورودی‌ها استفاده می‌شود و معمولاً برای فیلتر، از یک شرط استفاده می‌شود. در صورتی که شرط درست باشد، ورودی فرستاده می‌شود (برای مثال total_amount < 10).
- Windows: این عملیات برای جمع‌آوری و نگهداری رویدادها در یک پنجره‌ی زمانی مشخص استفاده می‌شود و به کاربران این اجازه را می‌دهد که توابعی بر روی رویدادهای جمع‌آوری شده، انجام دهند.

- Join: این عملیات، برای پیوند میان دو جریان داده و تولید یک جریان‌داده‌ی واحد استفاده می‌شود.
- Pattern: این عملیات، با کمک عبارات باقاعده برای نوشتن شرط‌ها پیاده‌سازی می‌شود و هنگامی که شرط درست باشد، رویداد فرستاده می‌شود. برای مثال، عبارت:
"Trip[total amount <10] -> Trip[total amount >50]" به معنای این است که دو رویداد اتفاق بیفتد که اولی total_amount کمتر از ۱۰ داشته باشد و دومی بیشتر از ۵۰ داشته باشد.

- Event Table: این عملیات، برای نگاشت یک پایگاه‌داده یا جدول به یک مجموعه‌ای از رویدادها استفاده می‌شود.

در شکل (۴-۱) ، یک نمونه کوئری با زبان SiddhiQL نوشته شده‌است.

```
@app:name('Temperature-Analytics')

define stream TempStream (deviceID long, roomNo int, temp double);

@info(name = '5minAvgQuery')
from TempStream#window.time(5 min)
select roomNo, avg(temp) as avgTemp
  group by roomNo
insert into OutputStream;
```

شکل (۴-۱) یک Siddhi Application با نام Temperature-Analytics

برای پردازش رویدادها، معمولاً یک فایل ایجاد کرده و درون آن از زبان SiddhiQL استفاده می‌شود. این فایل با نام SiddhiApp شناخته می‌شود. هر کوئری در زبان SiddhiQL شامل سه بخش اصلی است:

۱. اطلاعات برنامه: اولین بخش برنامه که با تگ @app شروع می‌شود و اطلاعاتی را درباره برنامه در اختیار می‌گذارد. برای مثال، @app:name() برای نمایش نام برنامه استفاده می‌شود.
۲. تعریف Stream: کوئری نوشته‌شده بر روی یک Stream اعمال می‌شود که این Stream قبل از کوئری تعریف می‌شود. هر Stream، مجموعه‌ای از رویدادها است که به ترتیب زمانی مرتب شده‌است و شامل یک نام و تعدادی ویژگی با نوع مشخص است. برای تعریف Stream از عبارت define stream استفاده می‌شود.

۳. نوشتن Query: در پایان، کوئری مربوطه و نام آن ذکر می‌شود. در مثال بالا، مقدار میانگین در فواصل زمانی ۵ دقیقه محاسبه می‌شود.

۳-۱- نتیجه

در این فصل، ابزارهای استفاده شده در پروژه و زبان پرس‌وجوی مخصوص این ابزار را توضیح دادیم. در فصل بعد، درباره‌ی شیوه‌ی پیاده‌سازی پروژه در مقاله‌ی انتخابی صحبت می‌کنیم.

فصل ۲:

روش پیاده‌سازی مقاله

۲-۱- مقدمه

در این فصل، قصد داریم جزئیات پیاده‌سازی پروژه را بیان کنیم. برای پیاده‌سازی مسئله‌ی ذکرشده در DEBS Grand Challenge 2015، ما از مقاله‌ی [1] استفاده کرده‌ایم.

در بخش (۲-۲)، به معرفی مجموعه‌داده می‌پردازیم. در بخش (۲-۳) مسئله‌ی مورد نظر و راه‌حل آن را بیان می‌کنیم. در بخش (۲-۴)، افزونه‌هایی را بیان می‌کنیم که در مقاله‌ی اصلی با هدف بهینه‌سازی نتایج ارائه شده‌است.

۲-۲- معرفی مجموعه‌داده

مجموعه‌داده‌ی مسئله، شامل اطلاعات سفرهای تاکسی‌های شهر نیویورک در سال ۲۰۱۳ است. در این مجموعه‌داده، اطلاعاتی همچون مختصات مبدأ سفر، مختصات مقصد سفر، زمان سفر و اطلاعات مربوط به پرداخت سفرها ذکر شده‌است. این مجموعه‌داده شامل ۱۷۳ میلیون رکورد و حجم آن ۳۳,۳ گیگابایت است. همچنین گزارشات سفر مربوط به ۱۴۱۴۴ تاکسی است. در جدول (۲-۱) اطلاعات مربوط به این مجموعه‌داده بیان شده‌است.

جدول (۲-۱) ویژگی‌های مجموعه‌داده سفر تاکسی‌ها

Attribute	Description
medallion	an md5sum of the identifier of the taxi - vehicle bound
hack_license	an md5sum of the identifier for the taxi license
pickup_datetime	time when the passenger(s) were picked up
dropoff_datetime	time when the passenger(s) were dropped off
trip_time_in_secs	duration of the trip
trip_distance	trip distance in miles
pickup_longitude	longitude coordinate of the pickup location
pickup_latitude	latitude coordinate of the pickup location
dropoff_longitude	longitude coordinate of the drop-off location
dropoff_latitude	latitude coordinate of the drop-off location
payment_type	the payment method - credit card or cash
fare_amount	fare amount in dollars
surcharge	surcharge in dollars
mta_tax	tax in dollars
tip_amount	tip in dollars

tolls_amount	bridge and tunnel tolls in dollars
total_amount	total paid amount in dollars

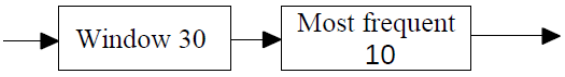
۳-۲- تعریف مسأله

هدف مساله‌ی موردنظر این است که با تحلیل مجموعه‌داده‌ی ذکر شده در بخش (۲-۲)، پاسخ دو کوئری زیر را در پنجره‌های زمانی ۳۰ دقیقه بیابیم:

- یافتن ۱۰ مسیر پرتردد در پنجره‌ی زمانی اخیر.
- یافتن ۱۰ مسیر پرسود در پنجره‌ی زمانی اخیر.

۱-۳-۲- شناسایی مسیرهای پرتردد

هدف کوئری اول، شمارش پیوسته‌ی مسیرها در ۳۰ دقیقه‌ی اخیر و نمایش ۱۰ مسیر پرتردد در این بازه است. به همین خاطر، این کوئری شامل دو مرحله‌ی پنجره‌بندی و شناسایی مسیرهای پر رفت و آمد است(مطابق شکل (۲-۲)).



شکل (۲-۲) نمودار کوئری اول

کوئری مربوط به این پرسش، به شرح زیر است:

```
@App:name('DebsQuery1')
@App:description('Queries for DEBS Query 1')

@source(type = 'http', source.id="get-TripEvent", receiver.url =
"http://0.0.0.0:8006/q1", basic.auth.enabled = "false",
       @map(type = 'json'))
define stream TripEvent (
    PickupTime long,
    DropoffTime long,
    PickupTimeOrig string,
    DropoffTimeOrig string,
    PickCell string,
    DropCell string,
    EventTimestamp long
);
```

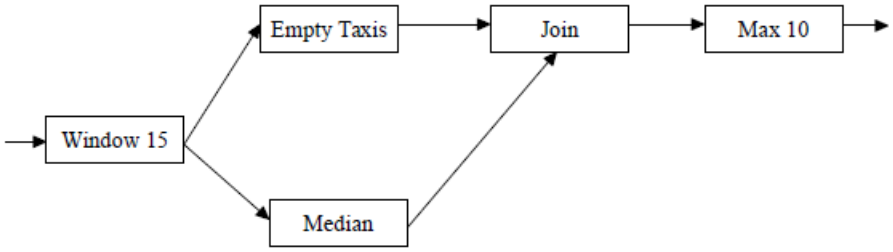
```
@sink(type='file', @map(type='json'), file.uri='C:/foo/resp.log')
@sink(type='http', methd='POST', publisher.url='http://localhost:5000/Query1Frequent',
@map(type='json'))
define stream OutputStream (AggregatedCells string, Timestamp long, PickupTime long,
DropoffTime long);


define function aggregateCells[JavaScript] return string {
    var d0 = data[0];
    var del = '-';
    var d1 = data[1];
    var str = d0 + del + d1;
    return str;
};

@info(name = 'windows over events')
from TripEvent#window.externalTime(DropoffTime, 30 min)
select *
insert all events into windowedStream;

@info(name='submit q')
from windowedStream
select
    aggregateCells(PickCell, DropCell) as AggregatedCells,
    EventTimestamp as Timestamp,
    PickupTime,
    DropoffTime

insert into OutputStream;
```

۲-۳-۲- شناسایی مسیرهای پرسود

هدف کوئری دوم، شناسایی مناطق با سوددهی بالا برای رانندگان تاکسی‌ها است. برای محاسبه سوددهی از فرمول زیر استفاده می‌شود:

$$profitability = \frac{median(fare + tip)\ for\ last\ 15\ minutes}{\#EmptyTaxies\ in\ last\ 30\ minutes}$$

با توجه به فرمول سوددهی، برای پاسخ به این کوئری نیاز است تا میانه‌ی سود در هر سلول و تعداد تاکسی‌های خالی محاسبه شود. سپس ۱۰ منطقه‌ی پرسود شناسایی شود (مطابق شکل (۳-۲)).



شکل (۳-۲) نمودار کوئری دوم

کوئری مربوط به این پرسش به شکل زیر است:

```
@App:name("DebsQuery2")
@App:description("")


@source(type = 'http', receiver.url = "http://0.0.0.0:8006/q2", basic.auth.enabled =
"false",
       @map(type = 'json', @attributes(    medallion='$.event.Medallion',
                                           pickup_datetime='$.event.PickupTime',
                                           dropoff_datetime='$.event.DropoffTime',
                                           pickup_cell='$.event.PickCell',
                                           dropoff_cell='$.event.DropCell',
                                           iij_timestamp='$.event.EventTimestamp',
                                           pickup_datetime_orig='$.event.PickupTimeOrig',

dropoff_datetime_orig='$.event.DropoffTimeOrig',
                                           fare_amount='$.event.FareAmount',
                                           tip_amount='$.event.TipAmount'
                                       )))
define stream taxi_trips (
    medallion string,
    pickup_datetime long,
    dropoff_datetime long,
    pickup_datetime_orig string,
    dropoff_datetime_orig string,
    pickup_cell string,
    dropoff_cell string,
    fare_amount float,
    tip_amount float,
    iij_timestamp long);

@sink(type='http-call' , method='POST',
publisher.url='http://localhost:5000/Query2Median', sink.id='median',
@map(type='json'))
define stream reqMedianStream (CurrentProfit float, Now long);

@source(type='http-call-response' , sink.id='median', @map(type='json'))
define stream getMedianStream (profit float);

@primaryKey('cellNo', 'emptyTaxiCount')
```

```
define table emptyTaxiCountTable (cellNo string, emptyTaxiCount long);

@sink(type='http' , publisher.url='http://localhost:5000/Query2Frequent',
method='POST', @map(type='json'))
define stream profitOutputStream (
    CellNumber string,
    MedianProfit float,
    EmptyTaxis long,
    PickupTime long,
    DropoffTime long,
    Profitability float,
    Timestamp long
);


@info(name = 'query1')
from taxi_trips
select
    pickup_cell as startCellNo,
    dropoff_cell as endCellNo,
    pickup_datetime as pickup_datetime,
    dropoff_datetime as dropoff_datetime,
    fare_amount, tip_amount,
    medallion,
    iij_timestamp
 insert into cell_based_taxi_trips;

@info(name = 'query2')
from cell_based_taxi_trips
insert into filtered_cell_based_taxi_trips;

--- Calculate Median using Web Service
from filtered_cell_based_taxi_trips#window.externalTime(dropoff_datetime, 15 min)
select fare_amount+tip_amount as CurrentProfit, iij_timestamp as Now
insert all events into reqMedianStream;
--- Calculate Median using Web Service

@info(name = 'query3')
from filtered_cell_based_taxi_trips#window.externalTime(dropoff_datetime, 15 min) join
getMedianStream
select profit, startCellNo, pickup_datetime, dropoff_datetime, iij_timestamp
group by startCellNo
insert all events into profitStream;

@info(name = 'query4')
from filtered_cell_based_taxi_trips
select endCellNo as cellNo, 1 as emptyTaxiCount insert into taxiCountUpdateStream;

@info(name = 'query5')
from every e1 = filtered_cell_based_taxi_trips -> e2 = filtered_cell_based_taxi_trips
[(e1.medallion == e2.medallion)
or (e2.dropoff_datetime-e1.dropoff_datetime)>=1800000]
select e1.endCellNo as cellNo, -1 as emptyTaxiCount
insert into taxiCountUpdateStream;

@info(name = 'query6')
from taxiCountUpdateStream
select cellNo, sum(emptyTaxiCount) as emptyTaxiCount
group by cellNo
insert into emptyTaxiCountTable;
```

```
@info(name = 'query7')
from profitStream#window.length(0) join emptyTaxiCountTable
on profitStream.startCellNo == emptyTaxiCountTable.cellNo
select
    profitStream.startCellNo as cellNo,
    profitStream.profit as profit,
    emptyTaxiCountTable.emptyTaxiCount as emptyTaxiCount,
    profitStream.pickup_datetime,
    profitStream.dropoff_datetime,
    profitStream.iij_timestamp as iij_timestamp
insert into profitRawData;

@info(name = 'query8')
from profitRawData[emptyTaxiCount != 0]
select
    cellNo,
    profit,
    emptyTaxiCount,
    pickup_datetime,
    dropoff_datetime,
    profit/emptyTaxiCount as profit_per_taxi,
    iij_timestamp
insert into finalProfitStream;

@info(name = 'query9')
from finalProfitStream
select cellNo as CellNumber,
       profit as MedianProfit,
       emptyTaxiCount as EmptyTaxis,
       pickup_datetime as PickupTime,
       dropoff_datetime as DropoffTime,
       profit_per_taxi as Profitability,
       iij_timestamp as Timestamp
insert into profitOutputStream;
```

۴-۲- بهینه‌سازی

برای افزایش سرعت و کاهش تاخیر، تعدادی بهینه‌سازی انجام شده‌است که در ادامه، آن‌ها را توضیح می‌دهیم.

۱-۴-۲- بهینه‌سازی در محاسبه frequentK

محاسبه‌ی k عنصر پرتکرار[1] یک مسئله‌ی چالش برانگیز در هر دو کوئری است. با توجه به وجود تعداد زیادی مسیر، ذخیره تعداد تکرار این مسیرها پرهزینه است و فضای زیادی را اشغال می‌کند. برای مقابله با این چالش، می‌توان از یک نگاشت معکوس استفاده کرد یعنی برای ذخیره‌سازی تعداد تکرار هر مسیر از

[1] FrequentK

ساختمان‌داده‌ای همچون Hash Map استفاده کرد که در آن، کلید تعداد تکرار مسیرها و مقادیر، مسیرها هستند.

۲-۴-۲- محاسبه میانه

برای محاسبه‌ی میانه، چندین روش وجود دارد:
۱. با کمک لیست: برای محاسبه میانه، می‌توان مقادیر را در یک لیست ذخیره کرده و سپس آن لیست را مرتب نموده و مقدار عنصر میانه را به عنوان مقدار میانه در نظر گرفت. این روش، روش بهینه‌ای نیست و کند عمل می‌کند.

۲. با روش Min-Max heap: برای محاسبه‌ی میانه، می‌توان مقادیر را در یک ساختمان‌داده heap ذخیره کرده و با شیفت‌دادن مقادیر به چپ و راست، مقدار میانه را تعیین کرد. پیچیدگی این روش، $O(\log(n))$ برای n مقدار است.

۳. با روش Bucket: در این روش، محدوده را به چندین Bucket تقسیم کرده و تعداد مقادیری که در هر Bucket قرار می‌گیرد، را می‌شمارد و در نهایت، میانه محاسبه می‌شود. پیچیدگی این روش $O(1)$ است.

۴. با روش نمونه‌گیری: در این روش، نمونه‌ی تصادفی از داده‌ها نگه‌داری شده و در صورتی که داده‌ی جدیدی وارد شود، با یکی از داده‌های منقضی‌شده جایگزین می‌شود.

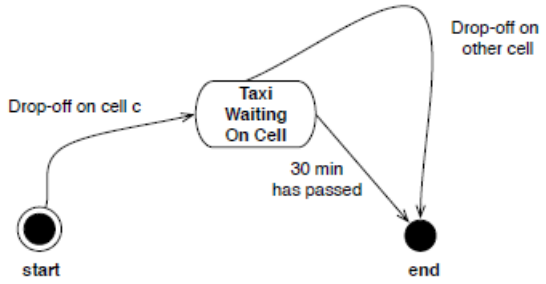
در مقاله‌ی انتخابی، از روش سوم برای پیاده‌سازی میانه استفاده شده‌است زیرا این روش پیچیدگی کمتری دارد و می‌تواند به عنوان روش بهینه استفاده شود.

۲-۴-۳- جلوگیری از عمل Join

در کوئری دوم نیاز است تا مقدار میانه و تاکسی‌های خالی، Join شوند. همان‌گونه که می‌دانیم این عمل، یک عمل پرهزینه است و به همین خاطر نیاز است برای بهینه‌سازی در نتایج، از انجام این عملگر جلوگیری شود. برای این مورد، می‌توان مقدار میانه که از قبل محاسبه شده را در کنار اطلاعات تاکسی‌های خالی کپی کرد، تا از عملگر پرهزینه Join استفاده نکنیم.

۲-۴-۴- بهینه‌سازی در الگوی شمارش تاکسی‌ها

برای بهبود سرعت در شمارش تاکسی‌های خالی در ۳۰ دقیقه‌ی اخیر، از یک ماشین حالت[1] برای پیاده‌سازی استفاده شده‌است.



شکل (۴-۲) ماشین حالت برای شمارش تاکسی‌های خالی

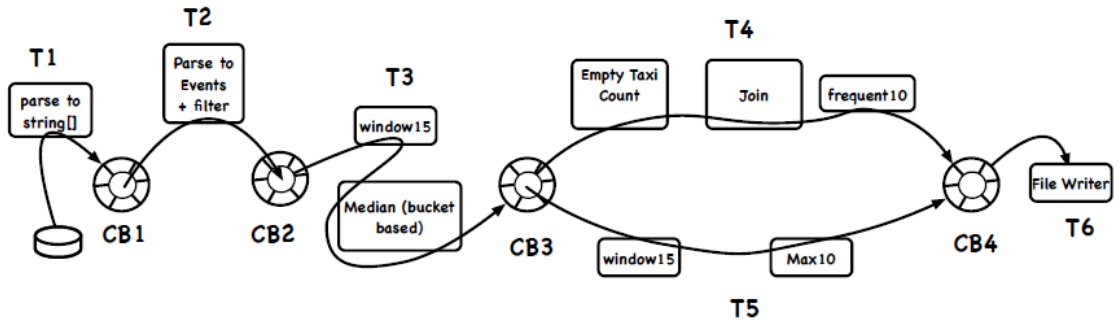
ماشین حالت برای شمارش تاکسی‌های خالی مشابه شکل (۴-۲) است. هنگامی که پیاده‌شدن در ناحیه‌ی مشخصی از نقشه رخ دهد، یک حالت برای آن ایجاد می‌شود. در صورتی که ۳۰ دقیقه از پیاده‌شدن از آن تاکسی بگذرد یا پیاده‌شدن دیگری توسط آن تاکسی شناسایی شود، ماشین به حالت پایانی می‌رود. این روش، باعث می‌شود شمارش تاکسی‌های خالی سریع‌تر انجام شود.

۲-۴-۵- موازی‌سازی

برای استفاده از حداکثر توان کامپیوتر، نیاز است تا موازی‌سازی میان وظایف صورت گیرد. برای این بهینه‌سازی، در مقاله از شبکه‌ای از Threadها استفاده شده که هر یک از این Threadها، وظیفه‌ی مشخصی را بر عهده دارد. به عبارت دیگر، هر وظیفه، Thread مختص خود را دارد و این Threadها درون یک صف حلقوی قرار دارند تا ترتیب اجرای وظایف حفظ شود. با این کار، موازی‌سازی به حداکثر مقدار رسیده و تاخیر[2] به حداقل مقدار خود رسیده است(مشابه شکل (۵-۲)).

1 State machine

2 Latency



شکل (۵-۲) موازی‌سازی در کوئری دوم

۵-۲- نتیجه

در این فصل، به معرفی مجموعه‌داده و تعریف مساله پرداختیم. سپس روش حل آن توسط مقاله‌ی [1] را توضیح دادیم. برای حل مسئله در این مقاله، از ابزار WSO2 CEP استفاده شده‌بود و برای بهینه‌سازی در نتایج، چندین افزونه اضافه شده‌بود. این افزونه‌ها را نیز در بخش (۲-۴) توضیح دادیم. در فصل بعد، قصد داریم فرآیند پیاده‌سازی مقاله توسط اعضای گروه را شرح دهیم.

فصل ۳:

نحوه اجرا

۳-۱- مقدمه

در این فصل، قصد داریم درباره پیاده‌سازی مقاله صحبت کنیم. در بخش (۳-۲) به جزئیات پیاده‌سازی می‌پردازیم.

۳-۲- پیاده‌سازی

سناریوی پیاده‌سازی کوئری‌ها به شرح زیر است:

۱. اطلاعات سفرهای تاکسی‌ها از فایل csv خوانده می‌شود.

۲. با زبان #C، برای هر یک از سفرها، اطلاعات Parse شده و با کمک Rest API، اطلاعات در قالب json برای سرویس‌دهنده فرستاده می‌شود.

۳. سرویس‌دهنده با کمک کوئری‌ها و افزونه‌های نوشته‌شده در سمت سرویس‌گیرنده، نتایج را اجرا کرده و به سرویس‌گیرنده در قالب json می‌فرستد.

۴. سرویس‌گیرنده، در صورتی که نتایج تغییر کند، آن‌ها را در کنسول نمایش می‌دهد.

ما برای پیاده‌سازی از زبان‌های زیر استفاده کردیم:

- C# Language
- Siddhi Query Language
- JavaScript Language

فصل ۴:

نتایج اجرا

۴-۱- مقدمه

پس از پیاده‌سازی مقاله‌ی مدنظر، نتایجی را به‌دست آوردیم. در این فصل قصد داریم این نتایج را بیان کنیم. در بخش (۴-۲) نتایج حاصل از دو کوئری شناسایی مسیرهای پرتکرار و شناسایی مناطق پرسود را ذکر می‌کنیم.

۴-۲- نتایج پروژه

ما راه حل خود را بر روی کل مجموعه داده پیاده سازی کردیم. نتایج حاصل از دو کوئری در جدول (۴-۱) مشاهده می شود.

برای محاسبه زمان تاخیر[1] فاصله زمانی میان خواندن داده و ایجاد خروجی را محاسبه کردیم.

جدول (۴-۱) نتایج اجرا

Scenario	Total time(s)	Throughput(event/sec)	Q1 Delay(ms)	Q2 Delay(ms)

پاسخ دو کوئری، در فایل‌های پروژه موجود است.

[1] Delay

فصل ۵:

کارهای انجام‌شده

۱-۵- مقدمه

در این بخش، قصد داریم کارهای انجام‌شده در طول پروژه را بیان کنیم. به‌طور کلی، پروژه انجام‌شده شامل چندین بخش است که هر یک از اعضای گروه، بخشی از آن را بر عهده داشته‌اند. جدول زیر، جزئیات پروژه را نمایش می‌دهد.

جدول (۱-۵) شرح وظایف اعضای گروه

اعضای گروه	وظایف
آرین ابراهیم‌پور	نوشتن سرویس تحت وب برای محاسبه میانه و frequentK، و اسکریپت های SiddhiQL، ایجاد Location Mapper
احمد فاضلی	ایجاد برنامه CLI ارتباط با WSO2 و سرور نمایش نتایج
حمیدرضا محمدی	ایجاد فایل های اسکریپت مدل اولیه برای کوئری ها، بررسی روش های محاسبه Location Mapper، نوشتن اسکریپت اولیه Siddhi کوئری دوم
زهرا اخگری	نوشتن کامل فایل های مستندات، تبدیل مدل های ورودی خروجی به کلاس های سی شارپ، ایجاد کلاس MedianBucket

مراجع

مراجع

[1] S. Jayasekara, S. Perera, M. Dayarathna, and S. Suhothayan, “DEBS grand challenge: Continuous analytics on geospatial data streams with WSO2 complex event processor,” in *DEBS 2015 - Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, 2015, pp. 277–284.

[2] M. Zaharia, T. Das, H. Li, T. Hunter, and S. Shenker, “Discretized Streams: Fault-Tolerant Streaming Computation at Scale.”

[3] G. Cugola and A. Margara, “Processing flows of information: From data stream to complex event processing,” *ACM Comput. Surv.*, vol. 44, no. 3, Jun. 2012.

Abstract:

DEBS Grand Challenge is a yearly, real-life data based event processing challenge posted by Distributed Event Based Systems conference. The 2015 challenge uses a taxi trip data set from New York city that includes 173 million events collected over the year 2013. This report presents how we used WSO2 CEP, an open source, commercially available Complex Event Processing Engine, to solve the problem. The report will outline the solution, present results, and discuss how we optimized the solution for maximum performance.

Keywords: DEBS Grand Challenge, Complex Event Processing, WSO2 CEP, Optimization.

Iran University of Science and Technology
Computer Engineering Department

Streaming Data Processing with WSO2 CEP Tools

By:
Aryan Ebrahim Pour
Ahmad Fazeli
Hamid Reza Mohammadi
Zahra Akhgari

Supervisor:
Dr. Mohsen Sharifi
Dr. Ali Jafari

January 2020