



دانشکده مهندسی کامپیوتر

## پردازش داده‌های جریانی با ابزار WSO2

گزارش پروژه درس سیستم‌های توزیع شده  
در رشته مهندسی کامپیوتر گرایش نرم افزار

نام اعضای گروه:

آرین ابراهیم پور

احمد فاضلی

حمیدرضا محمدی

زهرا اخگری

استاد راهنما:

دکتر محسن شریفی

دکتر علی جعفری

دی ماه 1398

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

## چکیده

DEBS Grand Challenge یک چالش سالیانه برای پردازش رویدادهای داده‌های واقعی است که توسط کنفرانس Distributed Event Based Systems ارسال می‌شود. چالش سال 2015 از مجموعه داده‌های سفرهای تاکسی‌های شهر نیویورک استفاده می‌کند که شامل 173 میلیون رویداد جمع‌آوری شده در طول سال 2013 است. در این گزارش، چگونگی استفاده از WSO2 CEP، یک موتور پردازش رویدادهای پیچیده بیان شده است که به صورت متن‌باز در دسترس است. همچنین در این گزارش، راه حل و نتایج به تفصیل بیان شده است و در جهت بهینه‌سازی راه حل با هدف افزایش کارایی، بحث شده است.

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

## فهرست مطالب

1	فصل 1: مقدمه
2	1-1- مقدمه
3	2-1- معرفی محصول WSO2
5	1-2-1 معماری WSO2 CEP
7	2-2-1 محصول WSO2 Streaming Integrator
8	3-2-1 زبان پرس و جوی Siddhi
10	3-1- نتیجه
11	فصل 2: روش پیاده سازی مقاله
12	1-2- مقدمه
12	2-2- معرفی مجموعه داده
13	3-2- تعریف مسأله
13	1-3-2 شناسایی مسیرهای پرتدد
14	2-3-2 شناسایی مسیرهای پرسود
17	4-2- بهینه سازی
17	2-4-1 بهینه سازی در محاسبه frequentK
18	2-4-2 محاسبه میانه
18	2-4-3 جلوگیری از عمل Join
19	2-4-4 بهینه سازی در الگوی شمارش تاکسی ها
19	2-4-5 موازی سازی
20	2-5- نتیجه
21	فصل 3: نحوه اجرا
22	1-3- مقدمه
22	3-2- پیاده سازی
25	3-3- نحوه ی اجرای برنامه
29	فصل 4: نتایج اجرا
30	4-1- مقدمه
30	4-2- نتایج پروژه
31	فصل 5: کارهای انجام شده
32	5-1- مقدمه
33	مراجع



## فهرست اشکال

- شکل (2-1) معماری WSO2 CEP ..... 5
- شکل (3-1) نمای کلی از WSO2 SI ..... 8
- شکل (4-1) یک Siddhi Application با نام Temperature-Analytics ..... 9
- شکل (2-2) نمودار کوئری اول ..... 13
- شکل (3-2) نمودار کوئری دوم ..... 15
- شکل (4-2) ماشین حالت برای شمارش تاکسی‌های خالی ..... 19
- شکل (5-2) موازی‌سازی در کوئری دوم ..... 20
- شکل (1-3) ایجاد پروژه در محیط Visual Studio ..... 22
- شکل (2-3) نحوه‌ی اجرای WSO2 SI Tooling ..... 25
- شکل (3-3) واسط گرافیکی WSO2 SI Streaming ..... 26
- شکل (3-4) اجرای دو پروژه ..... 27
- شکل (5-3) اجرای برنامه و انتخاب کوئری توسط کاربر ..... 27
- شکل (6-3) ارسال داده‌ها به سرور توسط برنامه ..... 28

## فهرست جداول

جدول (1-1) برخی از ویژگی‌های ابزار WSO2 CEP.....	3
جدول (1-2) ویژگی‌های مجموعه داده سفر تاکسی‌ها.....	12
جدول (1-4) نتایج اجرا.....	30
جدول (1-5) شرح وظایف اعضای گروه.....	32

# فصل 1:

## مقدمه



## 1-1- 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<sup>1</sup> صنعتی و تحقیقاتی برپا می‌شود. هدف چالش DEBS2015، ارزیابی سیستم‌های مبتنی بر رویداد در زمینه‌ی تحلیل جریان داده‌های حجیم جغرافیایی<sup>2</sup> به صورت بلادرنگ<sup>3</sup> است. داده‌های انتخابی برای تحلیل، شامل گزارشات سفرهای تعدادی از تاکسی‌ها در شهر نیویورک است. اهداف کلی این رقابت، شامل موارد زیر است:

- شناسایی 10 مسیر پر تردد اخیر در پنجره‌ی زمانی<sup>4</sup> 30 دقیقه.
- شناسایی 10 منطقه‌ی سودآور در پنجره‌ی زمانی 30 دقیقه.

برای یافتن این دو سناریو، باید به تجزیه و تحلیل اطلاعات مکانی تاکسی‌ها و دیگر اطلاعات داده‌شده در مسئله پردازیم. هدف این پژوهش [1]، یافتن نتایج با حداکثر بازدهی<sup>5</sup> و حداقل تاخیر<sup>6</sup> 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 10 هزار رویداد بر ثانیه را پردازش کند.

در مقاله‌ی [1]، از سیستم WSO2 CEP برای پردازش رویدادها استفاده شده‌است. این سیستم، قابلیت پردازش 400 هزار رویداد در ثانیه را دارد و نویسندگان مقاله‌ی [1]، توانسته‌اند با ایجاد چندین افزونه<sup>7</sup>، حدود 0.3 میلیون رویداد در ثانیه را پردازش کنند. در بخش (2-1)، به معرفی این ابزار می‌پردازیم.

<sup>1</sup> Event-Based System

<sup>2</sup> Geo-spatial

<sup>3</sup> Real-time

<sup>4</sup> Sliding window

<sup>5</sup> Throughput

<sup>6</sup> Latency

<sup>7</sup> Extension

## 2-1- معرفی محصول WSO2

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

جدول (1-1) برخی از ویژگی‌های ابزار WSO2 CEP

ویژگی‌ها	توضیحات
موتور پردازش با عملکرد بسیار بالا	<ul style="list-style-type: none"> <li>- پردازش بیش از 100K رویداد در ثانیه به صورت تک ماشین</li> <li>- طراحی شده توسط WSO2 Siddhi</li> </ul>
زبان پرس‌وجوی قدرتمند و گسترده	<ul style="list-style-type: none"> <li>- زبان پرس‌وجو مشابه SQL</li> <li>- فیلتر کردن رویدادها با شرایط ذکر شده</li> <li>- امکان پارتیشن‌بندی داده‌ها برای پشتیبانی از پردازش موازی</li> <li>- اجرای پرس‌وجوها با کمک پنجره‌های زمانی مختلف</li> </ul>
امکانات اجرای کاربرپسند	<ul style="list-style-type: none"> <li>- امکان استفاده کاربران از فرم‌هایی به جای استفاده</li> </ul>

<sup>1</sup> WSO2 Complex Event Processor

<sup>2</sup> Open source

<sup>3</sup> Scalable

<sup>4</sup> Query language

<sup>5</sup> Event

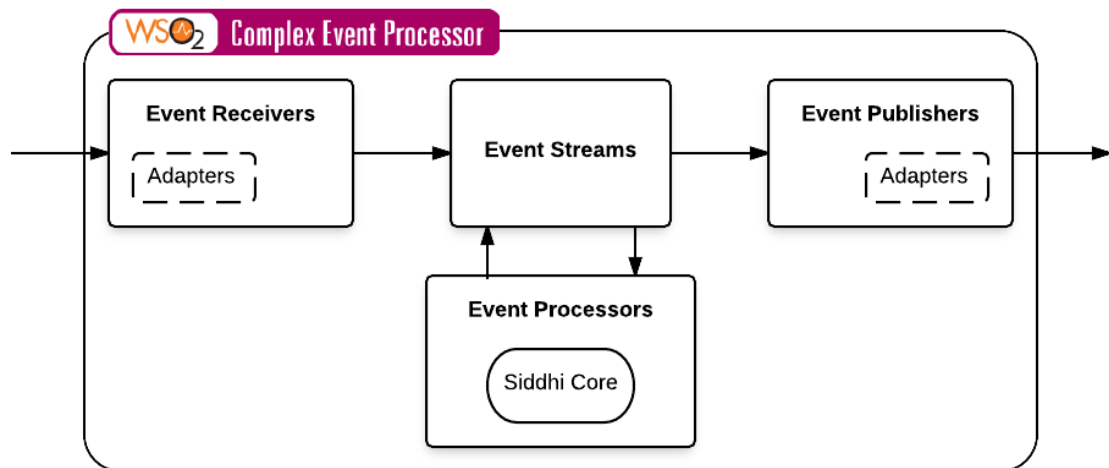
از پرس و جوهای SQL.	
- طراحی پرس و جوها به گونه‌ای که پیچیدگی‌های مسئله پنهان شده‌است.	
<ul style="list-style-type: none"> <li>- Restful HTTP protocol</li> <li>- SOAP protocol</li> <li>- Kafka, MQTT, File, Socket, Email protocol</li> </ul>	ادغام آسان سیستم با دیگر سیستم‌ها برای ثبت و ضبط رویدادها
<ul style="list-style-type: none"> <li>- XML, json, Map, Text events via JMS protocol</li> <li>- Email, SMS notification</li> <li>- RESTful, Web services</li> <li>- Kafka, MQTT, File, Web socket protocols</li> </ul>	پشتیبانی از چندین مکانیسم برای هشدار دادن
	توسعه‌ی آسان
<ul style="list-style-type: none"> <li>- پردازش بلادرنگ توزیع شده</li> <li>- بخش‌بندی و تقسیم کوئری‌ها در میان چندین سرویس‌دهنده</li> </ul>	مقیاس‌پذیر
<ul style="list-style-type: none"> <li>- پشتیبانی از ویژگی‌های مختلف همچون Float, Boolean و String.</li> </ul>	پشتیبانی از مدل‌های رویداد پیچیده

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 (1-2-1) توضیحی درباره معماری WSO2 CEP بیان شده‌است. در بخش (1-2-2) توضیح مختصری از WSO2 Streaming Integrator داده شده‌است. در بخش (1-2-3) نیز درباره زبان این محصول توضیح داده شده‌است.

## 1-2-1 معماری WSO2 CEP

در شکل (1-2)، معماری این ابزار و بخش‌های آن نمایش داده شده‌است.



شکل (1-2) معماری WSO2 CEP

معماری WSO2 CEP شامل بخش‌های زیر است:

- گیرنده رویداد<sup>1</sup>: 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

<sup>1</sup> 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<sup>1</sup>: 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<sup>2</sup>: 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:

1. این بخش، مجموعه‌ای از جریان‌های رویداد (Event Stream) را از واحد مدیریت جریان رویداد (Event Stream Manager) دریافت می‌کند.

2. با استفاده از موتور Siddhi، پردازشی بر روی آن‌ها انجام می‌دهد.

3. رویدادهای جدید را به بخش Event Stream Manager باز می‌گرداند.

- منتشرکننده‌ی رویداد<sup>3</sup>: این بخش، رویدادها را به سیستم‌های خارجی ارسال کرده و داده‌ها را در پایگاه داده 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
- SOAP Event Publisher

<sup>1</sup> Event Streams

<sup>2</sup> Event Processor

<sup>3</sup> Event Publisher

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

## 2-2-1- محصول WSO2 Streaming Integrator

WSO2 SI<sup>1</sup> 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<sup>2</sup>: تبدیل داده از یک نوع به دیگر نوع.
- غنی‌سازی داده<sup>3</sup>: دریافت داده از یک منبع مشخص و ترکیب آن با پایگاه داده‌ها، سرویس‌ها برای محاسبات.
- همبستگی<sup>4</sup>: اتصال داده‌های جریانی با چندین جریان برای ایجاد یک جریان داده واحد.
- پاکسازی<sup>5</sup>: فیلتر کردن و اصلاح محتویات پیام‌ها.
- خلاصه‌سازی<sup>6</sup>: خلاصه‌سازی داده‌ها در پنجره‌های زمانی مشخص.

<sup>1</sup> WSO2 Streaming Integrator

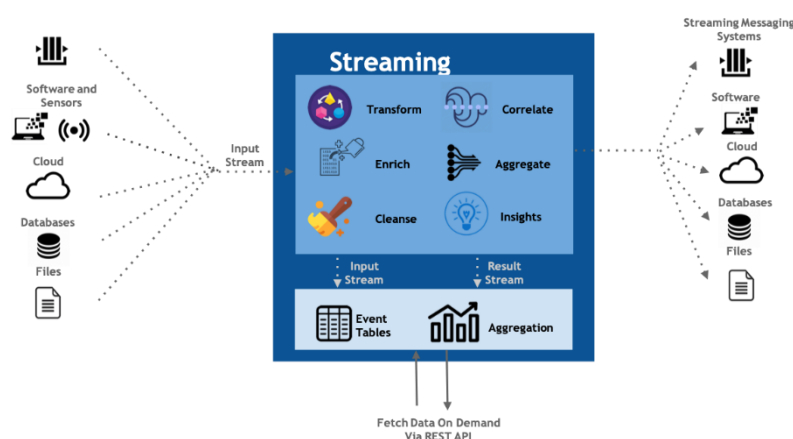
<sup>2</sup> Transforming Data

<sup>3</sup> Enriching Data

<sup>4</sup> Correlating

<sup>5</sup> Cleaning

<sup>6</sup> Summarizing



شکل (1-3) 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)

## 3-2-1- 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 جریان داده‌ی 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 10 داشته باشد و دومی بیشتر از 50 داشته باشد.
- Event Table: این عملیات، برای نگاشت یک پایگاه داده یا جدول به یک مجموعه‌ای از رویدادها استفاده می‌شود.

در شکل (4-1)، یک نمونه کوئری با زبان SiddhiQL نوشته شده است.

```
@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 (4-1) 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:

1. 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.
2. تعریف Stream: کوئری نوشته شده 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 شده است و شامل یک نام و تعدادی ویژگی با نوع مشخص است. برای تعریف Stream از عبارت define stream استفاده می‌شود.



3. نوشتن Query: در پایان، کوئری مربوطه و نام آن ذکر می‌شود. در مثال بالا، مقدار میانگین در فواصل زمانی 5 دقیقه محاسبه می‌شود.

### 3-1- 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 2:

### روش پیاده‌سازی مقاله

## 2-1- 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 (2-2)، به معرفی مجموعه‌داده می‌پردازیم. در بخش (2-3) مسئله‌ی مورد نظر و راه‌حل آن را بیان می‌کنیم. در بخش (2-4)، افزونه‌هایی را بیان می‌کنیم که در مقاله‌ی اصلی با هدف بهینه‌سازی نتایج ارائه شده‌است.

## 2-2- معرفی مجموعه‌داده

مجموعه‌داده‌ی مسئله، شامل اطلاعات سفرهای تاکسی‌های شهر نیویورک در سال 2013 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 173 میلیون رکورد و حجم آن 33.3 گیگابایت است. همچنین گزارشات سفر مربوط به 14144 تاکسی است. در جدول (2-1) اطلاعات مربوط به این مجموعه‌داده بیان شده‌است.

جدول (2-1) 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

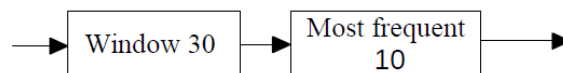
## 3-2- تعریف مسأله

هدف مسأله‌ی موردنظر این است که با تحلیل مجموعه داده‌ی ذکر شده در بخش (2-2)، پاسخ دو کوئری زیر را در پنجره‌های زمانی 30 دقیقه بیابیم:

- یافتن 10 مسیر پرتردد در پنجره‌ی زمانی اخیر.
- یافتن 10 مسیر پرسود در پنجره‌ی زمانی اخیر.

### 1-3-2- شناسایی مسیرهای پرتردد

هدف کوئری اول، شمارش پیوسته‌ی مسیرها در 30 دقیقه‌ی اخیر و نمایش 10 مسیر پرتردد در این بازه است. به همین خاطر، این کوئری شامل دو مرحله‌ی پنجره‌بندی و شناسایی مسیرهای پرتردد و آمد است (مطابق شکل (2-2)).



شکل (2-2) 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;

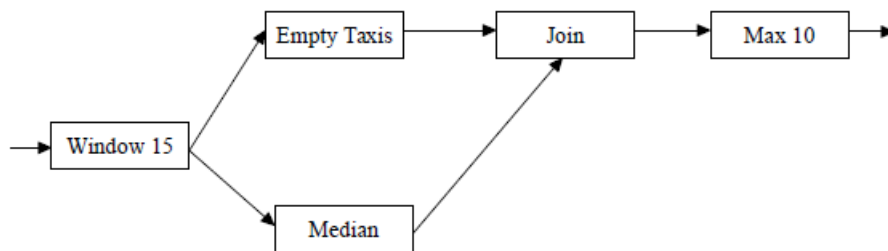
```

## 2-3-2- شناسایی مسیرهای پرسود

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

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

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 10 منطقه‌ی پرسود شناسایی شود (مطابق شکل (2-3)).



شکل (2-3) 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;

```

## 2-4- بهینه‌سازی

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

### 2-4-1- بهینه‌سازی در محاسبه frequentK

محاسبه‌ی  $k$  عنصر پرتکرار<sup>1</sup>، 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

<sup>1</sup> FrequentK



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

## 2-4-2- محاسبه میانه

برای محاسبه‌ی میانه، چندین روش وجود دارد:

1. 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.

2. با روش Min-Max heap: برای محاسبه‌ی میانه، می‌توان مقادیر را در یک ساختمان داده 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.

3. 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.

4. 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.

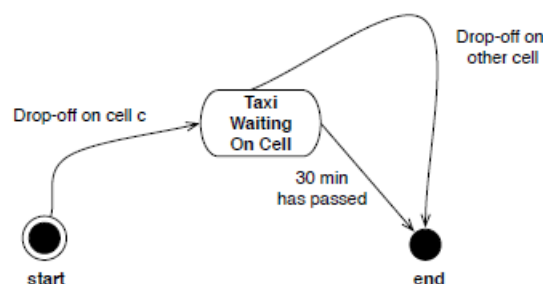
در مقاله‌ی انتخابی، از روش سوم برای پیاده‌سازی میانه استفاده شده‌است زیرا این روش پیچیدگی کمتری دارد و می‌تواند به عنوان روش بهینه استفاده شود.

## 2-4-3- 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.

## 4-2-4- بهینه‌سازی در الگوی شمارش تاکسی‌ها

برای بهبود سرعت در شمارش تاکسی‌های خالی در 30 دقیقه‌ی اخیر، از یک ماشین حالت<sup>1</sup> برای پیاده‌سازی استفاده شده‌است.



شکل (2-4) ماشین حالت برای شمارش تاکسی‌های خالی

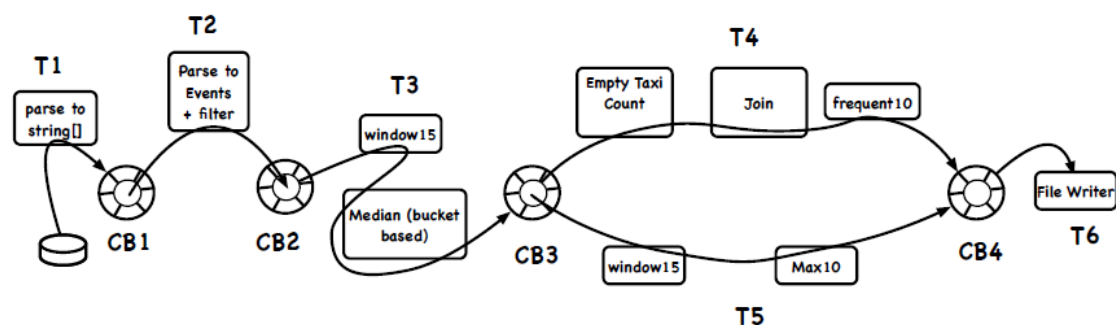
ماشین حالت برای شمارش تاکسی‌های خالی مشابه شکل (2-4) است. هنگامی که پیاده‌شدن در ناحیه‌ی مشخصی از نقشه رخ دهد، یک حالت برای آن ایجاد می‌شود. در صورتی که 30 دقیقه از پیاده‌شدن از آن تاکسی بگذرد یا پیاده‌شدن دیگری توسط آن تاکسی شناسایی شود، ماشین به حالت پایانی می‌رود. این روش، باعث می‌شود شمارش تاکسی‌های خالی سریع‌تر انجام شود.

## 5-4-2- 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<sup>2</sup> به حداقل مقدار خود رسیده است (مشابه شکل (2-5)).

<sup>1</sup> State machine

<sup>2</sup> Latency



شکل (2-5) موازی‌سازی در کوئری دوم

## 2-5- نتیجه

در این فصل، به معرفی مجموعه داده 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 شده بود و برای بهینه‌سازی در نتایج، چندین افزونه اضافه شده بود. این افزونه‌ها را نیز در بخش (2-4) توضیح دادیم. در فصل بعد، قصد داریم فرآیند پیاده‌سازی مقاله توسط اعضای گروه را شرح دهیم.

## فصل 3:

### نحوه اجرا

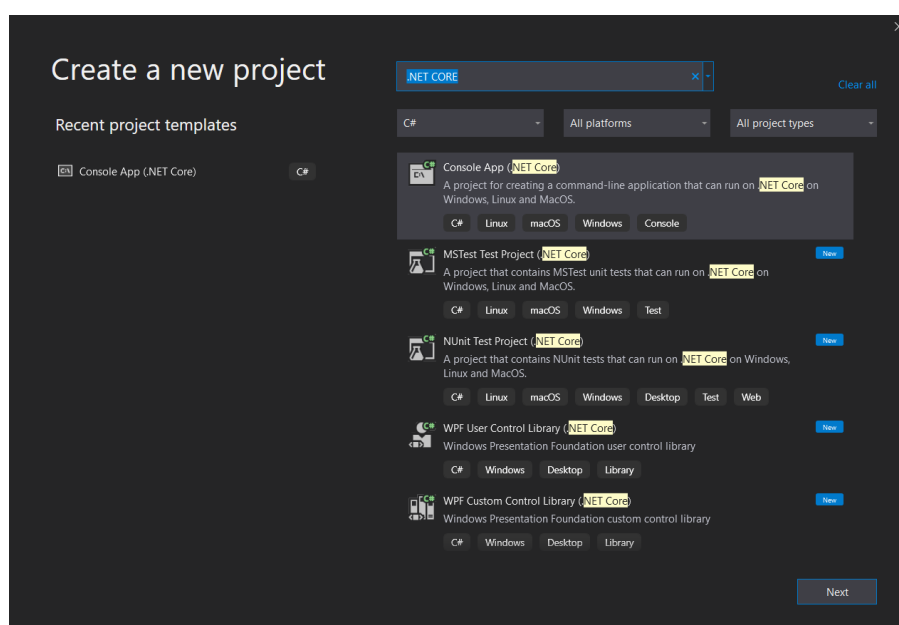
## 1-3- مقدمه

در این فصل، قصد داریم درباره پیاده‌سازی مقاله صحبت کنیم. در بخش (2-3) به جزئیات پیاده‌سازی می‌پردازیم. در بخش (3-3) درباره‌ی نحوه‌ی اجرای برنامه صحبت می‌کنیم.

## 2-3- پیاده‌سازی

فرآیند پیاده‌سازی کوثری‌ها به شرح زیر است:

1. ایجاد پروژه در C#: برای پیاده‌سازی، ما از زبان C# در محیط برنامه‌نویسی Visual studio استفاده کردیم. در این محیط، برای ایجاد پروژه‌ی جدید، مسیر File → New Project را طی کرده و Console App(.NET Core) را انتخاب نمودیم.



شکل (1-3) ایجاد پروژه در محیط Visual Studio

2. ایجاد مدل ورودی و خروجی: پردازش داده‌های اصلی به دلیل حجم بالا، زمان‌بر است. به همین خاطر، ما یک نمونه از داده‌ها را برای پردازش انتخاب کردیم. این نمونه داده در یک فایل با پسوند CSV ذخیره شده‌است. برای خواندن این فایل، از کلاس CsvReader استفاده کرده‌ایم. هر یک از سطرهای این فایل، به فرم ورودی کوثری‌ها Parse می‌شود (فایل Program.cs در پروژه‌ی GrandChallenge).

برای مثال، مدل ورودی در کوئری اول به شرح زیر است:

- PickupTime: این متغیر، زمان جهانی سوار شدن مسافر را نشان می‌دهد. برای تبدیل زمان سوار شدن مسافر به زمان جهانی، از تابع GetUnixTime استفاده شده است.

- DropoffTime: این متغیر، زمان جهانی پیاده شدن مسافر را نشان می‌دهد. برای تبدیل زمان پیاده شدن مسافر به زمان جهانی، از تابع GetUnixTime استفاده شده است.

تابع GetUnixTime، در پوشه‌ی Extensions و فایل TimeStampExtensions.cs موجود است و وظیفه‌ی آن تبدیل زمان به ساعت جهانی است.

- PickCell: این متغیر، سلول<sup>1</sup> مربوط به سوار شدن مسافر را نشان می‌دهد که یک متغیر دوتایی (x,y) است.

- DropCell: این متغیر، سلول مربوط به پیاده شدن مسافر را نشان می‌دهد که یک متغیر دوتایی (x,y) است.

برای محاسبه‌ی سلول جغرافیایی، از مختصات جغرافیایی ارائه شده در مجموعه داده استفاده شده است. به این معنا که نقشه‌ی جغرافیایی را به یک محدوده‌ی 300\*600 تبدیل کرده و سلول هر مختصات جغرافیایی را در این محدوده (با کمک کمترین طول و عرض جغرافیایی) محاسبه می‌کنیم. روش تبدیل مختصات جغرافیایی به سلول، در پوشه‌ی Geography و در فایل TaxiLocation.cs قابل مشاهده است.

- EventTimeStamp: این متغیر، زمان ارسال داده را نشان می‌دهد.

- PickupTimeOrig: این متغیر، زمان ثبت شده سوار شدن مسافر در مجموعه داده‌ی اصلی را نشان می‌دهد.

- DropoffTimeOrig: این متغیر، زمان ثبت شده پیاده شدن مسافر در مجموعه داده‌ی اصلی را نشان می‌دهد.

مدل ورودی در کوئری دوم نیز به شرح زیر است:

- Medallion: این متغیر، شناسه‌ی<sup>2</sup> مربوط به هر تاکسی را نشان می‌دهد.

- PickupTime: مطابق کوئری قبلی.

- DropoffTime: مطابق کوئری قبلی.

- PickCell: مطابق کوئری قبلی.

- DropCell: مطابق کوئری قبلی.

<sup>1</sup> Cell

<sup>2</sup> Identifier

- EventTimeStamp: مطابق کوئری قبلی.
- PickupTimeOrig: مطابق کوئری قبلی.
- DropoffTimeOrig: مطابق کوئری قبلی.
- FareAmount: این متغیر، بیانگر مقدار کرایه‌ی مسافر است که برای محاسبه‌ی نتایج در کوئری دوم لازم است.
- TipAmount: این متغیر، بیانگر مقدار انعام پرداخت‌شده توسط مسافر است که برای محاسبه‌ی نتایج در کوئری دوم لازم است..

فرم ورودی کوئری‌ها در پوشه‌ی Models از پروژه، یافت می‌شود:  
distributed-system-final\src\GrandChallenge\Models

فرم ورودی کوئری اول، فایل FirstQueryInputModel.cs و فرم ورودی کوئری دوم، فایل SecondQueryInputModel.cs در پوشه‌ی ذکر شده‌است.

مدل خروجی نیز مطابق صورت سوال تعریف می‌شود. کد مربوط به مدل خروجی دو کوئری، در پوشه‌ی Models قابل مشاهده است (فایل‌های Query1Result.cs و Query2Result.cs).

3. نوشتن کوئری‌ها: برای نوشتن کوئری‌ها، از زبان پرس‌وجوی Siddhi استفاده شده‌است. این کوئری‌ها، با کمک توضیحات مقاله‌ی اصلی و تکمیل شبه‌کوئری‌های آن مقاله نوشته شده‌است. فایل مربوط به هر دو کوئری در پوشه‌ی SiddhiApps از پروژه‌ی GrandChallenge قابل دسترسی است. برای نوشتن این کوئری‌ها، به محیط WSO2 SI وارد شده و از مسیر File → New، دو فایل ایجاد کرده و محتوای مربوط به کوئری‌ها را در آن نوشتیم. طریقه‌ی اجرای WSO2 در بخش بعد ذکر شده‌است.

4. نوشتن توابع بهینه‌سازی<sup>1</sup>: برای پیاده‌سازی توابع موجود در کوئری‌ها (تابع FrequentK و Median)، از وب‌سرویس استفاده کردیم. روش پیاده‌سازی این توابع مشابه مطالب بیان شده در فصل دوم است. پیاده‌سازی این توابع، در پروژه‌ی GrandChallenge.EventWebService موجود است.

به‌طور کلی، ما برای پیاده‌سازی از زبان‌های زیر استفاده کردیم:

- C# Language
- Siddhi Query Language
- JavaScript Language

<sup>1</sup> Optimization

همچنین از محیط‌های برنامه‌نویسی زیر برای پیاده‌سازی استفاده کردیم:

- Visual studio
- WSO2 Streaming Integrator

برای هماهنگی کار تیمی از Azure Version Control استفاده نمودیم و پروژه در [اینجا](#) قرار دارد.

### 3-3- نحوه‌ی اجرای برنامه

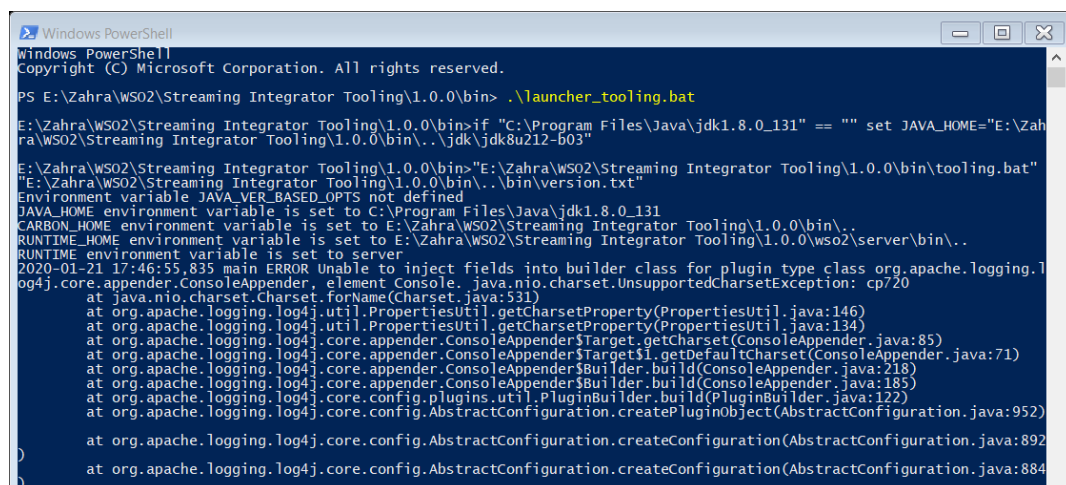
برای اجرای پروژه، نیاز است تا چندین کار صورت گیرد که در ادامه به آن‌ها می‌پردازیم:

#### 1. اجرای WSO2 SI Tooling:

ابتدا باید ابزار WSO2 SI Tooling را نصب و نصب شود. دانلود این ابزار از لینک زیر، امکان پذیر است.

<https://wso2.com/integration/streaming-integrator/>

در این لینک، هم نسخه‌ی کنسول و هم نسخه‌ی گرافیکی موجود است. همچنین مستندات مربوط به این ابزار نیز وجود دارد. ما از نسخه‌ی Tooling برای ادامه‌ی پروژه استفاده کردیم. پس از دانلود، نصب آن را انجام می‌دهیم. نصب این ابزار به‌سادگی امکان‌پذیر است. پس از نصب، برای اجرای برنامه لازم است به پوشه‌ی bin برنامه رفته و در آنجا cmd را باز کرده و فایل launcher\_tooling.bat را اجرا می‌کنیم (مطابق شکل (2-3)).



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS E:\Zahra\WSO2\Streaming Integrator Tooling\1.0.0\bin> .\launcher_tooling.bat

E:\Zahra\WSO2\Streaming Integrator Tooling\1.0.0\bin> if "C:\Program Files\Java\jdk1.8.0_131" == "" set JAVA_HOME="E:\Zahra\WSO2\Streaming Integrator Tooling\1.0.0\bin\..\jdk\jdk8u212-b03"

E:\Zahra\WSO2\Streaming Integrator Tooling\1.0.0\bin> "E:\Zahra\WSO2\Streaming Integrator Tooling\1.0.0\bin\tooling.bat"
"E:\Zahra\WSO2\Streaming Integrator Tooling\1.0.0\bin\..\bin\version.txt"
Environment variable JAVA_VER_BASED_OPTS not defined
JAVA_HOME environment variable is set to C:\Program Files\Java\jdk1.8.0_131
CARBON_HOME environment variable is set to E:\Zahra\WSO2\Streaming Integrator Tooling\1.0.0\bin\..
RUNTIME_HOME environment variable is set to server
2020-01-21 17:46:55.835 main ERROR Unable to inject fields into builder class for plugin type class org.apache.logging.log4j.core.appender.ConsoleAppender, element Console. java.nio.charset.UnsupportedCharsetException: cp720
at java.nio.charset.Charset.forName(Charset.java:531)
at org.apache.logging.log4j.util.PropertiesUtil.getCharsetProperty(PropertiesUtil.java:146)
at org.apache.logging.log4j.util.PropertiesUtil.getCharsetProperty(PropertiesUtil.java:134)
at org.apache.logging.log4j.core.appender.ConsoleAppender$Target.getCharset(ConsoleAppender.java:85)
at org.apache.logging.log4j.core.appender.ConsoleAppender$Target.getDefaultCharset(ConsoleAppender.java:71)
at org.apache.logging.log4j.core.appender.ConsoleAppender$Builder.build(ConsoleAppender.java:218)
at org.apache.logging.log4j.core.appender.ConsoleAppender$Builder.build(ConsoleAppender.java:185)
at org.apache.logging.log4j.core.config.plugins.util.PluginBuilder.build(PluginBuilder.java:122)
at org.apache.logging.log4j.core.config.AbstractConfiguration.createPluginObject(AbstractConfiguration.java:952)
at org.apache.logging.log4j.core.config.AbstractConfiguration.createConfiguration(AbstractConfiguration.java:892)
at org.apache.logging.log4j.core.config.AbstractConfiguration.createConfiguration(AbstractConfiguration.java:884)

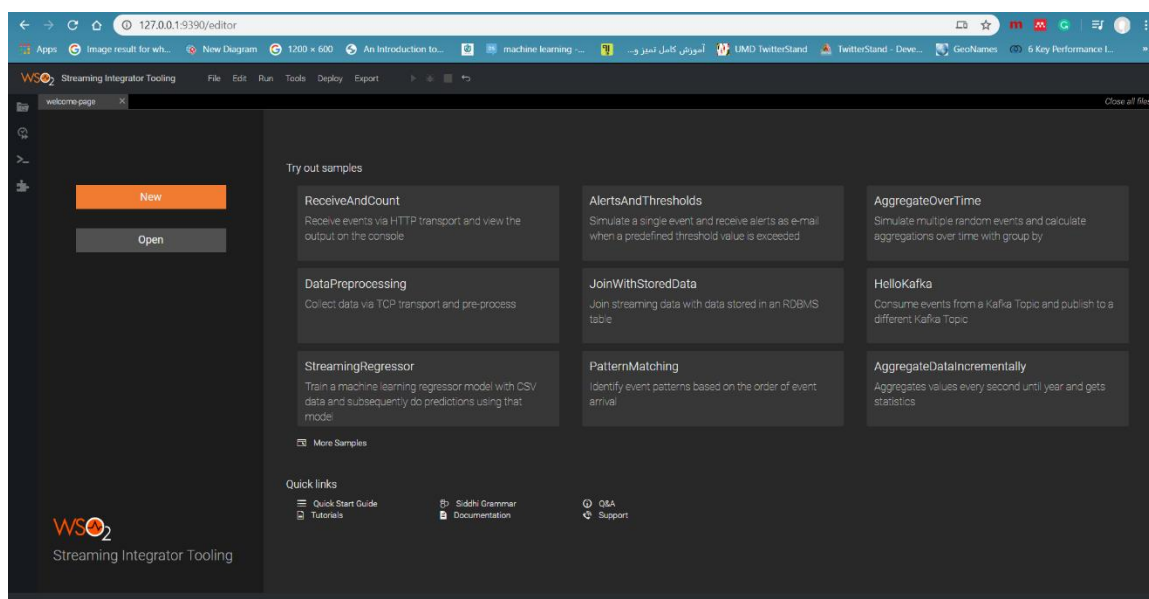
```

شکل (2-3) نحوه‌ی اجرای WSO2 SI Tooling

پس از اجرای فایل مربوطه، واسط گرافیکی WSO2 SI Tooling از لینک زیر قابل دسترسی است:

<http://127.0.0.1:9390/editor>



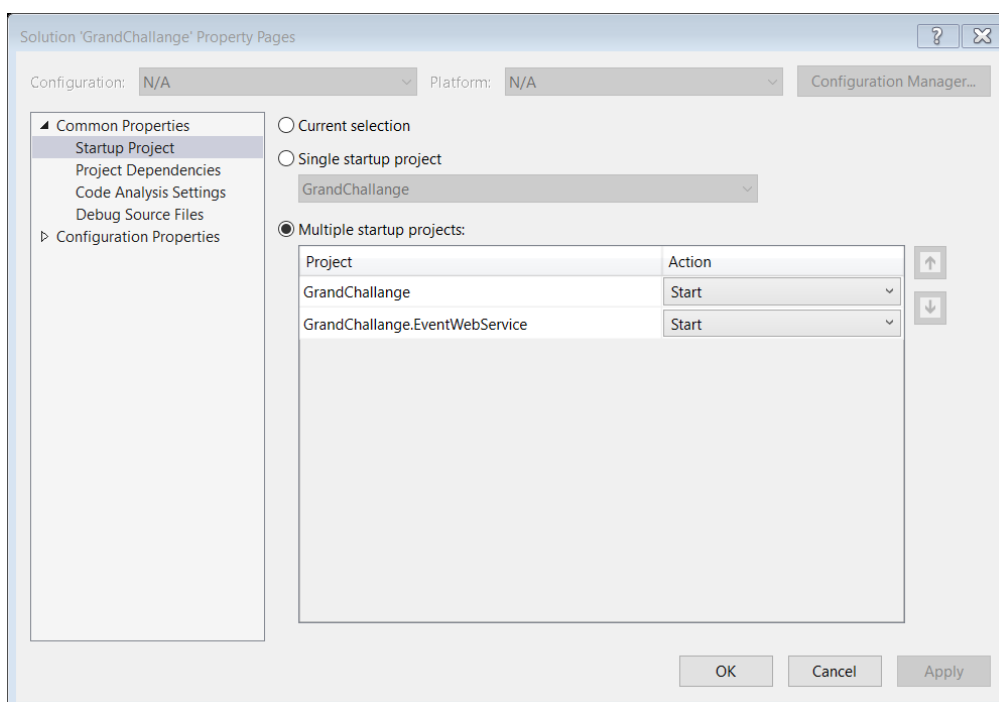


شکل (3-3) واسط گرافیکی WSO2 SI Streaming

برای نوشتن کوئری‌ها، می‌توان از منوی بالای صفحه و از مسیر `File → new`، فایل جدیدی ایجاد کرد و کوئری مربوطه را در آن نوشت.

## 2. اجرای برنامه:

برای اجرای برنامه، باید هر دو پروژه‌ی `GrandChallenge` و `GrandChallenge.EventWebService` اجرا شود. به همین خاطر، به مسیر `Solution → Properties → Startup Project` رفته و گزینه‌ی `multiple startup project` را انتخاب می‌کنیم و سپس برنامه را اجرا می‌کنیم.



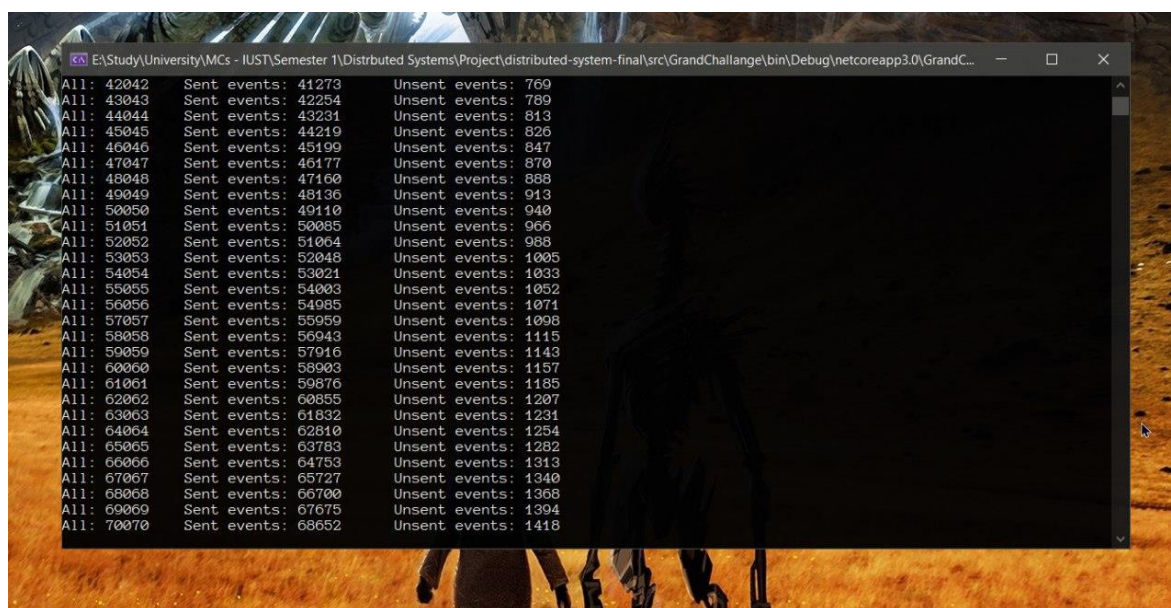
شکل (3-4) اجرای دو پروژه

3. پس از شروع برنامه، با انتخاب کوئری موردنظر، داده‌ها به صورت جریانی به سمت سرور ارسال می‌شود و نتایج به نمایش در می‌آید.

```
C:\Users\asus\Desktop\Final project\distributed-system-final\src\GrandChallenge\bin\Debug\netcoreapp3.0\GrandChallenge.exe
*****
| Welcom to Grand Challenge 2015 solution |
*****
Please enter query number for run
1.first query
2.second query
input your number:
```

شکل (3-5) اجرای برنامه و انتخاب کوئری توسط کاربر

نتایج نمایش داده‌شده تغییر نمی‌کند تا زمانی که نتایج به‌روز شود.



All: 42042	Sent events: 41273	Unsent events: 769
All: 43043	Sent events: 42254	Unsent events: 789
All: 44044	Sent events: 43231	Unsent events: 813
All: 45045	Sent events: 44219	Unsent events: 826
All: 46046	Sent events: 45199	Unsent events: 847
All: 47047	Sent events: 46177	Unsent events: 870
All: 48048	Sent events: 47160	Unsent events: 888
All: 49049	Sent events: 48136	Unsent events: 913
All: 50050	Sent events: 49110	Unsent events: 940
All: 51051	Sent events: 50085	Unsent events: 966
All: 52052	Sent events: 51064	Unsent events: 988
All: 53053	Sent events: 52048	Unsent events: 1005
All: 54054	Sent events: 53021	Unsent events: 1033
All: 55055	Sent events: 54003	Unsent events: 1052
All: 56056	Sent events: 54985	Unsent events: 1071
All: 57057	Sent events: 55959	Unsent events: 1098
All: 58058	Sent events: 56943	Unsent events: 1115
All: 59059	Sent events: 57916	Unsent events: 1143
All: 60060	Sent events: 58903	Unsent events: 1157
All: 61061	Sent events: 59876	Unsent events: 1185
All: 62062	Sent events: 60855	Unsent events: 1207
All: 63063	Sent events: 61832	Unsent events: 1231
All: 64064	Sent events: 62810	Unsent events: 1254
All: 65065	Sent events: 63783	Unsent events: 1282
All: 66066	Sent events: 64753	Unsent events: 1313
All: 67067	Sent events: 65727	Unsent events: 1340
All: 68068	Sent events: 66700	Unsent events: 1368
All: 69069	Sent events: 67675	Unsent events: 1394
All: 70070	Sent events: 68652	Unsent events: 1418

شکل (3-6) ارسال داده‌ها به سرور توسط برنامه

مطابق شکل (3-6)، تعداد کل داده‌ها و تعداد داده‌های ارسالی و داده‌های ارسال نشده مشاهده می‌شود. پس از اجرای برنامه و تحلیل داده‌ها توسط ابزار WSO2 SI، فایل‌های خروجی با پسوند txt. درون پروژه ایجاد می‌شود.

## فصل 4:

### نتائج اجرا

## 1-4- مقدمه

پس از پیاده‌سازی مقاله‌ی مدنظر، نتایجی را به‌دست آوردیم. در این فصل قصد داریم این نتایج را بیان کنیم. در بخش (2-4) نتایج حاصل از دو کوئری شناسایی مسیرهای پرتکرار و شناسایی مناطق پرسود را ذکر می‌کنیم.

## 2-4- نتایج پروژه

ما راه حل خود را بر روی نمونه‌ای از مجموعه‌داده پیاده‌سازی کردیم. نتایج حاصل از دو کوئری در جدول (1-4) مشاهده می‌شود.

نتایج خروجی دو کوئری در دو فایل Query1\_res.txt و Query2\_res.txt در پوشه‌ی پروژه، موجود است.

نکات:

برای محاسبه زمان تاخیر<sup>1</sup>، فاصله زمانی میان خواندن داده و ایجاد خروجی را محاسبه کردیم. برای محاسبه‌ی بازدهی، از فرمول تعداد رویدادهای پردازش‌شده در ثانیه استفاده می‌شود.

جدول (1-4) نتایج اجرا

Scenario	Total time(s)	Throughput(event/sec)	Q1 Delay(ms)	Q2 Delay(ms)
Q1 Q2	5 hours	125	1ms – 3sec	1s – 16s

<sup>1</sup> Delay

## فصل 5:

### کارهای انجام شده

## 1-5- 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 (1-5) شرح وظایف اعضای گروه

اعضای گروه	وظایف
آرین ابراهیم‌پور	نوشتن سرویس تحت وب برای محاسبه میانه و frequentK، و اسکریپت‌های SiddhiQL، ایجاد Location Mapper، مشارکت در CLI
احمد فاضلی	ایجاد برنامه CLI ارتباط با WSO2 و سرور نمایش نتایج
حمیدرضا محمدی	ایجاد فایل‌های اسکریپت مدل اولیه برای کوئری‌ها، بررسی روش‌های محاسبه Location Mapper
زهرا اخگری	نوشتن کامل فایل‌های مستندات، تبدیل مدل‌های ورودی خروجی به کلاس‌های سی شارپ، ایجاد کلاس MedianBucket

## 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.



**IU ST**

**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**