**UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA**

**FACULTATEA DE MATEMATICǍ ŞI INFORMATICǍ**

**SPECIALIZAREA INFORMATICĂ**

**LUCRARE DE LICENŢĂ**

**Flux continuu de date pe baze de date**

**Conducător științific**

Dr. SURDU Sabina, Lector Universitar

**Absolvent**

**Crișan Camelia Daniela**

**2018**

Cuprins

[Abstract 4](#_Toc517517641)

[Capitolul 1 5](#_Toc517517642)

[Introducere 5](#_Toc517517643)

[Capitolul 2 6](#_Toc517517644)

[Descrierea domeniului aplicației 6](#_Toc517517645)

[2.1 Evenimente 6](#_Toc517517646)

[2.2CEP(Complex Event Processing) 6](#_Toc517517647)

[2.3 Ferestrele de evenimente(Window events) 7](#_Toc517517648)

[2.6 Aggregations 8](#_Toc517517649)

[2.7 Stream 8](#_Toc517517650)

[2.8 Query 8](#_Toc517517651)

[2.8.1 CQL 8](#_Toc517517652)

[Capitolul 3 10](#_Toc517517653)

[Tehnologii utilizate 10](#_Toc517517654)

[3.1 StreamInsight 10](#_Toc517517655)

[3.1.1 Despre 10](#_Toc517517656)

[3.1.2 Monitorizare 12](#_Toc517517657)

[3.1.3 Publicarea serverului StreamInsight 13](#_Toc517517658)

[3.1.4 Hopping Windows 14](#_Toc517517659)

[3.1.5 Snapshot Windows 15](#_Toc517517660)

[3.2 Event flow debugger 15](#_Toc517517661)

[3.3 LINQ 19](#_Toc517517662)

[3.4 Dynamic SQL 19](#_Toc517517663)

[Capitolul 4 21](#_Toc517517664)

[Studiu experimental 21](#_Toc517517665)

[4.1 Despre 21](#_Toc517517666)

[4.2 Implementare 24](#_Toc517517667)

[4.2.1 Server 24](#_Toc517517668)

[4.2.2 Sursa de evenimente 24](#_Toc517517669)

[4.2.3Interogările 26](#_Toc517517670)

[4.2.4 Structura 30](#_Toc517517671)

[4.2.5 Monitorizare 34](#_Toc517517672)

[4.3Rezultatele studiului experimental 36](#_Toc517517673)

[4.Comparatie cu studiul experimental Aurora 42](#_Toc517517674)

[4.Oportunității viitoare 42](#_Toc517517675)

[Capitolul 5 43](#_Toc517517676)

[Concluzii 43](#_Toc517517677)

[6.Bibliografie 44](#_Toc517517678)

# Abstract

Această lucrare are scopul de a descrie procesul și arhitectura sistemului creat pe baza unei arhitecturi StreamInsight, pentru administrarea fluxurilor de date continue. Pentru început vom prezentate date generale care vor consolida baza teoretică și cunoștințele necesare pentru a putea înțelege domeniul aplicației, dar și cum aceasta acționează asupra datelor. Ne vom concentra atenția asupra interogărilor agregate și asupra performanțelor acestora. Vom prezenta impactul distribuirii datelor, dar și impactul dimensionări ferestrelor de interogare. În final se vor prezenta rezultatele obținute în urma studiului, se vor trage concluzii și se vor lua decizii asupra unor direcții viitoare de dezvoltare ale aplicației.

# Capitolul 1

## Introducere

Bazele de date au evoluat și au început să crească din ce în ce mai mult datorită fluxului imens de informație primit. Ținând cont că în ultimul deceniul tehnologia și-a făcut loc în viețile noastre ca o parte esențială a acesteia, am devenit dependenți de utilizarea acestora și a informațiilor pe care acestea le dețin, astfel acesta este unul din fluxurile importante de date.

Accesul omniprezent la date ușurează utilizatorilor munca zilnică și viața personală, astfel eliminarea barierelor tehnologice de scară și economie sunt realizate pe diferite platforme prin calul utilitar. Aceste tendințe aduc un impact major asupra aplicațiilor IT, un grad sporit de conștientizare a timpului, deoarece care fiecare punct de date care sosește reprezintă un punct de contact al clientului astfel se poate observa cum fiecare trend se diminuează odată cu timpul. Datorită concurenței în cea ce privește aplicațiile și a modului în care evoluează, acestea necesită o trecere la o procesare bazată pe evenimente bine poziționate într-o perioadă de timp. Astfel am ales să studiez impactul fluxurilor de evenimente, în diferite condiții de evaluare, pentru a putea observa viitoarele oportunități de optimizare a aplicațiilor și a datelor existente.

Astfel după studiul mai multor lucrări științifice în care am studiat sisteme de gestiune a bazelor de date precum Aurora sau STREAM, am vrut să studiez și eu impactul pe care anumiți operatori îi au asupra fluxului de date, dar și clasele de interogări care manipulează fluxul și se folosesc de multiple resurse pentru a putea calcula cât mai corect și eficient rezultatul. Așadar am pornit de la ideea prezentată în articolul Linear Road, dar am materializat această idee pe un singur flux de date pentru orașul Cluj-Napoca.

În această lucrare vom folosi tipuri de ferestre asupra datelor, pentru care va fi calculat un rezultat aproximat, datorită continuității datelor procesate. Vom încerca să comparăm rezultatele obținute de ferestrele de date și vom încerca să găsim tipul de fereastră optimal, pentru fiecare tip de fereastră care sunt limitele de eroare peste care putem trece pentru a avea o performanță sporită.

# Capitolul 2

## Descrierea domeniului aplicației

### 2.1 Evenimente

Pentru a putea vorbi despre evenimente și procesarea acestora trebuie mai întâi să înțelegem ce sunt evenimentele. Evenimentul este o acțiune trecută care poate fi reprezentată ca obiect ( în programare) și care poate fi analizată în diferite moduri. De obicei fiecare flux de evenimente are anumite caracteristici, de exemplu un flux de evenimente provenite de la mașini din trafic. Toate acestea vor conține același tip de informație, dar fiecare va avea un timp de apariție diferit, o viteză a mașinii diferită și o marcă diferită. Astfel vom putea defini, un eveniment generic care să fie o specificare pentru un set de astfel de evenimente și să le încapsuleze într-o sferă semantică, grupându-le ca tipuri de evenimente.

Evenimentul poate fi descris ca și o colecție de atribute împărțite pe trei categorii : antet, încărcătura utilă și conținut deschis (opțional). Antetul este constituit din meta-informația evenimentului spre exemplu timpul de apariție al acestuia. Încărcătura conține o colecție de atribute specifice fiecărui tip de eveniment în parte, fiecare din aceste atribuite poate fi un tip de dată simplu precum un întreg (int), un sir de caractere (string) sau poate fi o structură complexă de date (obiecte).

Printre atribute se regăsesc timpul de apariție, este un tip de dată care se definește ca și un punct in timp, precizia acestuia fiind dată de granularitatea temporală care se aplică tipului de eveniment. Granularitatea temporală, denotă un atom temporal dintr-un punct de vedere al aplicației precum : minutul, ora sau ziua. Din acestea rezultă timpul de apariție fiind definit ca si atribut in care precizia timpului este dată de granularitatea temporală a evenimentului. [3]

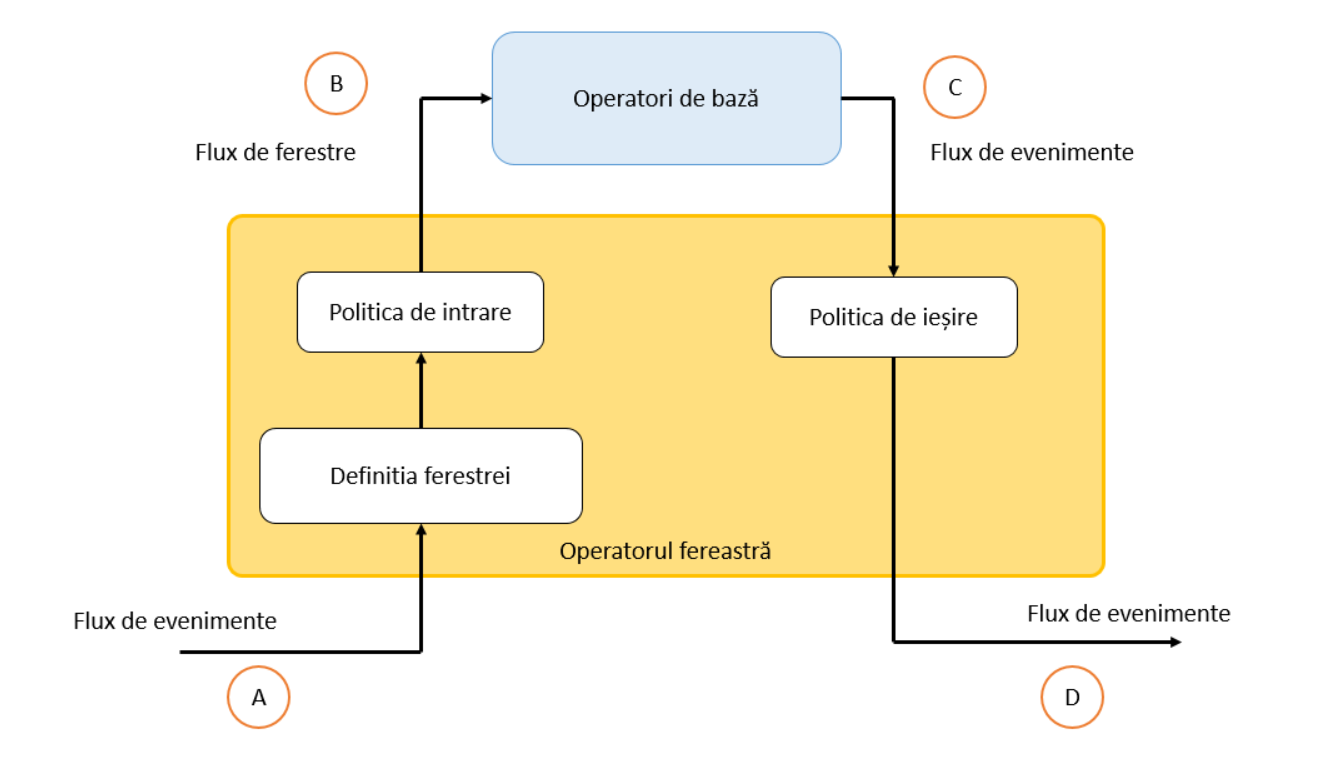
### 2.2CEP(Complex Event Processing)

Procesarea de evenimente se bazează pe analiza a ceea ce se întâmplă in fluxul de date și pe rezultate concrete pe baza acestora. Procesarea complexă de evenimente ne permite analiza și corelarea cantităților mari de date sub formă de evenimente cu scopul detectării unor șabloane sau situații de interes în timp real. Structura este formată din mai multe tipuri de adaptori care pot face legătura între sursele de date și sistemul de analiză.

### 2.3 Ferestrele de evenimente(Window events)

În general, aplicațiile care procesează evenimente in timp real, necesită o performanță ridicată in calculul diferitelor operații pe subseturi de date. In StreamInsight aceste subseturi sunt definite ca si ferestre, acestea sunt de 3 feluri : fereastră hopping (Hopping Window), fereastră de numărare(Count Window), fereastră instant(Snapshot Window). Fluxul de evenimente pentru a putea fi procesat este transformat într-un flux fereastră **IQWindowedStreamable<T> , astfel operațiile de bază vor putea fi executate. Aceste evenimente sunt grupate după tipul de fereastră folosit : timp sau număr, ferestrele fiind caracterizate după operatorul acesteia.**

**Operațiile de bază readuc fluxul de ferestre la un flux de evenimente, aceste operații se împart în două categorii : agregările și operațiile de generare de evenimente de ieșire pentru un set de evenimente de intrare. În cadrul agregărilor avem operații precum sum, avg, count sau operații definite de utilizator. Într-un flux de ferestre pot fi aplicate una sau mai multe agregări, atât asupra ferestrei cât și a datelor din acesta, fiind posibilitatea de a crea un alt flux de date care să conțină date agregate. Operațiile de generare pot fi TopK sau operații definite de utilizator, acestea sunt utilizate peste un flux de ferestre și generează zero sau mai multe evenimente în cadrul unei ferestre ca și rezultat al unul calcul.**

 **Fiecare fereastră este alcătuită din trei părți: definiția specifică, transformarea temporală a datelor de input și transformarea temporală a datelor de output. Configurația unei ferestre:**

Configurația unei ferestre (adaptată după documentația oficială, în limba română)

Transformările temporale sunt făcute după anumite politici, politica de input este singulară și se referă la timpul de intrare și ieșire al evenimentelor, în care fiecare operație poate vedea timpul evenimentului doar in interiorul ferestrei, iar ca politică de ieșire, fiecare tip de fereastră are politica ei specifică. Ferestrele instantanee adaptează timpul final al evenimentului după timpul final al ferestrei. Rezultatele date de ferestrele hopping sunt așezate în ordine după timpul final al ferestrei, iar rezultatul dat de ferestrele de numărare

### 2.6 Aggregations

Tipurile de agregări pe care le putem găsi în interiorul interogărilor sunt media unei încărcături întregi (avg), suma unei încărcături de date(sum), minimul(min) si maximul(max) unei încărcături și numărarea evenimentelor(count). Aceste operații sunt efectuate pe subseturi de date specificate de tipul ferestrelor. Calculele agregate sunt realizate în mod incremental, adică se reactualizează la momentul declanșării unui eveniment, de aceea precizia poate fi afectată. Fiecare din funcțiile de agregare au un singur parametru, care reprezintă valoarea care trebuie agregată.

### 2.7 Stream

Stream-ul, adică fluxul de date este o secvență de evenimente continue disponibile o perioadă de timp. Acesta se poate defini ca un set de două valori in care prima valoare este un tuplu, iar cea de-a doua reprezintă timpul apariției tuplului.

### 2.8 Query

### 2.8.1 CQL

Asupra fluxurilor de date, se pot crea interogări simple pentru care este suficient un limbaj relațional precum SQL, dar cum aceste iterogări au început să devină din ce în ce mai complexe introducând în structura lor ferestre(ex. Thumblig window), agregări sau join-uri între fluxuri, aceste limbaje simple au fost depășite, astfel s-a creat un nou tip abstract pentru interogări continue numit Continoious Query Language(CQL).

Acesta se poate împărți în trei categorii: relation-to-relation care are ca și input una sau mai multe relații si ca output are tot o relație, stream-to-relation care are ca input un flux si produce ca output o relație și reation-to-stream care are ca input o relație si generează un flux ca și date de output.

Un plan de interogare este compilat pentru o interogare continuă cand acesta este înregistrat, acesta procesează cozi de operatori. Acești operatori pot fi de inserare sau de ștergere. Elementele cozilor dintr-o interogare sunt reprezentate de fluxuri sau relații și are doi operatori operatorul care produce și cel care consumă. Fiecare plan are alocat câte un rezumat care de cele mai mule ori este folosit pentru a afla starea curentă a relației. Execuția se realizează pe baza unui plan care depinde de timpul apariției elementului procesat.

Pentru a crește performanța, avem nevoie de câteva îmbunătățiri la interogările acestea brute. Deoarece într-un singur plan de interogare pot apărea mai multe rezumate, acestea uneori pot fi redundante astfel ele trebuie înlocuite de niște stub-uri care implementează aceeași interfață. Constrângerile sunt un mod bun de a crește performanța, deoarece dacă acestea sunt specificate dinainte, iar datele care vin au un anumit șablon se poate reduce dimensiunea rezumatului.

# Capitolul 3

## Tehnologii utilizate

### 3.1 StreamInsight

### 3.1.1 Despre

Tehnologia principală utilizată este StreamInsight, aceasta este o platformă care poate fi folosită pentru a dezvolta si implementa aplicații care procesează evenimente complexe(CEP), aceasta are o arhitectură de nivel înalt în privința procesării fluxurilor de date și datorită implementării bazate pe Microsoft .NET Framework, aceasta permite dezvoltarea rapidă si eficientă a unor aplicații de procesare a evenimentelor. Fluxurile de evenimente conțin în general date care trebuie analizate sau procesate, precum datele din aplicațiile de tranzacționare financiară, analiză web sau analiză operațională. Astfel a venit soluția, StreamInsight poate dezvolta CEP care oferă direct rezultatul fără a fi nevoie de foarte multe resurse astfel reducându-se costurile extragerii, analizei si corelărilor de date și îți permite să monitorizezi, să gestionezi și să furnizeze datele pentru anumite condiții. Această tehnologie este folosită în special pentru a monitoriza date din mai multe surse pentru modele, tendințe, excepții și oportunități semnificative.

Are o bună optimizare în ceea ce privește performanța si transferul de date, deoarece acesta implementează o arhitectură care suportă paralelizarea la nivel înalt a interogărilor continue asupra datelor. Transferul de date este facilitat de utilizarea memoriei cache și de calculul incremental al rezultatelor. Datele nefiind stocate, procesându-se la momentul apariției acestuia, se obține un timp de latență scăzut.

Aplicațiile care folosesc această tehnologie sunt scrise de obicei sub platforma Microsoft.NET folosind limbajul C#, în care se regăsește limbajul interogărilor integrat (LINQ) care, deoarece este foarte cunoscut și foarte folosit poate reduce costurile de dezvoltare ale aplicației.

Tipurile de șcenarii pe care StreamInsight le suportă sunt : aplicațiile DLL hostate în care se poate interoga complet, aplicații de tip server care este folosit de mai mulți clienți sau în aplicații autonome sau hostate, care sunt alcătuite din mai multe servere.

Monitorizarea și administrarea funcțiilor sunt incluse in StreamInsight, asigură un cost scăzut, iar managementul interfeței și vizualizarea diagnosticului date de acesta asigură monitorizarea și administrarea unei aplicații CEP. Framework-ul de administrare suportă și aplicațiile hostate. Pentru analiză și diagnosticare a aplicațiilor autonome se folosește event flow debugger.

Această tehnologie, poate suporta anumite cerințe precum procesarea a peste 10.000 de evenimente pe secundă, un timp de latență scăzut în cazul agregărilor, avertizări în situațiile în care ceva s-a defectat, nu merge conform standardelor sau mentenanță susținută pe condiții.

Pentru un client cu o aplicație care necesită o procesare a unui flux de date impresionant, la o latență scăzută, precum aplicațiile web, acestea includ anumite cerințe precum abilitatea de a procesa peste 100.000 date/secundă, abilitatea de a naviga și prezenta o pagină pe baza fluxului de analiză și descoperirea șabloanelor în fluxurile de date cu un timp de răspuns imediat.

Pentru a putea administra rețele imense, precum utilitățile, internetul, aceste aplicații au anumite specificații. Aplicație trebuie să poată procesa peste 100.000 date/secundă care provin din milioane de fluxuri de date, acestea trebuie să poată fi împărțite pe mai multe nivele de agregare, eficientizarea acestora prin mutarea în rețele inteligente și un răspuns imediat la diferitele variații de date.

Ca și arhitectură partea de StreamInsight e văzută ca parte de server în care datele venite de la generatorii de fluxuri sunt procesate cu ajutorul interogărilor și generează rezultate pentru consumatori, precum aplicații web. Fluxurile de intrare sunt continue, ele fiind procesate în ordinea apariției acestora în concordanță cu fiecare interogare, iar ca și date de output acestea pot fi distribuite pentru diferite acțiuni.

Procesarea datelor este efectuată cu ajutorul a trei tipuri diferite de date. O listă de evenimente bine ordonată, care implementează interfața Ienumerable<>, poate fi accesată în mod iterativ folosind framework-ul LINQ-to-Objects, acesta se numește secvență de date de tragere (pull). O secvență de date de același tip, ordonate, care implementează interfața Iobservable<> și care oferă secvența pentru unul sau mai mulți consumatori, acest acces realizându-se cu ajutorul framework-ului Reactive LINQ, acestea se numesc secvențe de date de împingere (push). Stream-urile temporale sunt secvențe de elemente cu caracteristici unice, ele implementează interfața IStreamable<>.

Fluxurile de date temporale sunt specifice StreamInsight-ului, fiecare dată componentă a acestui flux infinit de date este constituită din două părți încărcătura utilă și timpul evenimentului, atât timpul de început cât și de sfârșit. Aceste fluxuri au anumite caracteristici precum fiecare eveniment din cadrul fluxului are un timp de apariție, printre evenimentele fluxului se regăsesc o serie de evenimente speciale care definesc gradul de completare al fluxului odată cu trecerea timpului, acestea fiind Current Time Increment(CTI) events, aceste date pot fi privite și ca niște bariere temporale deoarece, după ce un astfel de eveniment apare în flux cu un anumit timp, nici o altă dată care îl urmează nu poate avea un timp de apariție mai mic.

În secțiunea 2.1 am descris în mod general un eveniment, acum vom putea vedea care din caracteristicile lui se aplică in StreamInsight. Datele din fluxurile temporale sunt încapsulate în evenimente, acestea fiind unitățile de bază ale unei aplicații dezvoltate în această tehnologie. În antet sunt incluse tipul evenimentului precum și unul sau mai multe momente de timp, iar încărcătura utilă este construită ca o structură .NET, câmpurile definite în acesta fiind bazate pe tipurile de date existente in sistemul .NET. Inserarea și CTI(incrementarea timpului curent) sunt cele două tipuri de evenimente pe care StreamInsight le poate susține. Inserarea adaugă evenimentele în fluxul de date, împreună cu timpul se start si de final al acestuia. În schimb evenimentele de tipul CTI au un singur câmp acesta fiind timpul actual, astfel indică faptul că in fluxul de date nu mai există alte evenimente de procesat a căror timp de apariție nu mai corespunde.

Ca și entități ale unei aplicații, există cinci tipuri de bază. Sursa, care generează datele, suportă interogările de tipul LINQ în fluxurile continue sau în secvențe de evenimente, acestea pot implementa interfețele specializate pentru asta precum IQueryable, IQbservable și IQStreamable. Ca și consumator avem sink-ul, acesta este implementat după tipul sursei, dacă sursa implementează interfețele IObservable/IStreamable, atunci sink-ul va implementa un observer sau dacă sursa implementează IStreamable sink-ul va mai putea implementa interfața IStreamableSink. Subiectul este atât generator cât și consumator de date, fiind de folos și în cazul partajării calculelor între consumatori și producători, față de sursă acesta nu poate implementa în mod direct interfața IStreamable. Conexiunea între surse și sink-uri este realizată de o legătură, care este un executabil, iar execuția acestuia se numește proces.

### 3.1.2 Monitorizare

Monitorizarea unui server StreamInsight necesită observarea efectului interogărilor asupra sistemului, dar și a performanței acestora. Starea unui server este dată de monitorizarea interogărilor care rulează și a consumului de date pe care îl creează fluxul de evenimente generat.

Această informație poate fi obținută apelând o interfață de diagnosticare, dar pentru a avea acces la această interfață aplicația trebuie să suporte servicii web, deoarece clientul trebuie să se conecteze la aceasta. Altfel se poate folosi event flow debugger-ul pentru analiza datelor care se folosește de o interfață grafică pentru a arăta rezultatele.

Putem obține rezultate pentru fiecare obiect din server, care poate fi accesat utilizând un URL de bază cep:/Server [6] din care derivă mai apoi toate celelalte nivele ale aplicației, astfel putem urmări interogări specifice sau entități, mai putem urmări din fiecare interogare, operatori specifici sau fluxuri de date care trec prin acea interogare. În ceea ce privește entitățile putem urmări ce interogări sau semnătura entității corespunzătoare.

Monitorizarea performanțelor interogărilor poate fi făcută pe nivelele interogării. În server se creează patru puncte cheie pentru înregistrarea sau monitorizarea performanțelor. Un prim punct este intrarea, datele care încep să vină în fluxul de date, aici se măsoară rata de apariție a datelor de input neprelucrate, în cadrul adaptorului de intrare. Al doi-lea punct se referă la datele consumate de server, acest punct se află în imediata apropiere a adaptorului de intrare și măsoară dimensiunea evenimentelor care așteaptă să fie procesate. Al trei-lea punct se referă la datele produse de către server, adică datele care au fost procesate de către interogări și urmează sa fie trimise spre consumator. Un ultim punct este ieșirea, din care evenimentele părăsesc adaptorul de ieșire, aceasta măsurând timpul de ieșire al evenimentelor spre consumator.

Timpul de latență al interogărilor poate fi monitorizat, acest timp fiind calculat pentru fiecare flux care trece pin toate cele patru puncte ale monitorizării performanței. Se pot calcula și medii între diferitele performanțe obținute la fiecare punct pentru date diferite. Putem vedea totalitatea memoriei consumată pe tot parcursul aplicației sau eficiența adaptoarelor pentru transferul datelor de la intrarea în server până la ieșirea din acesta.

Diagnosticarea ne oferă date despre fiecare componentă a serverului trecând prin mai multe nivele de granularitate. Construcția acestuia oferă posibilitatea de a avea mai multe tipuri de informație din fiecare nivel de granularitate. Informație statică care nu se modifică odată cu parcurgerea interogărilor, deoarece aceasta returnează proprietățile unui obiect. Informația neagregată care aduce informații despre obiectele copil care nu au fost agregare la obiectele lor părinte și informația agregată care se află in completarea celei neagregate.

### 3.1.3 Publicarea serverului StreamInsight

Aplicațiile StreamInsight pot fi servere locale sau servere accesate de la distanță, la care clientul trebuie să se poată conecta într-un anumit mod. Pentru cele două tipuri de servere, modelul de bază este identic, dar în cazul aplicației cu acces la distanță aceasta trebuie să fie atribuită unui URL care să se folosească de servicii web.

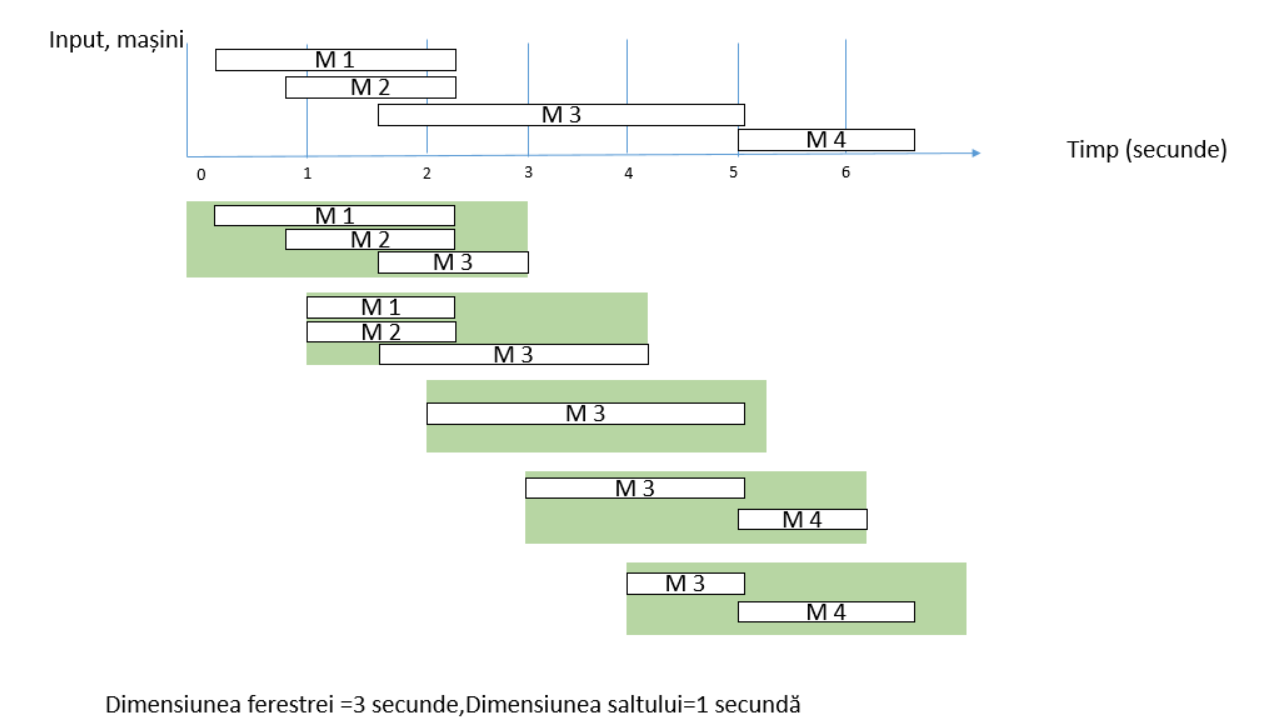
Serverul poate fi atât creat cât se poate și conecta la un server accesat la distanță, dar care trebuie să existe la o anumită adresă publică. Pentru realizarea conexiunii cu serverul creat la distanță pe lângă serviciul web este nevoie de o resursă uniformă de identificare.

### 3.1.4 Hopping Windows

Acest tip de fereastră definește un subset de evenimente care se încadrează într-un interval de timp și care acceptă să se efectueze diferite calcule sau agregări asupra datelor. Timpul acestor evenimente este împărțit în intervale de timp regulare.

Numele de hopping vine de la saltul temporal executat de o fereastră în momentul începerii unei ferestre, adică timpul de întârziere față de fereastra anterioară. Aceste ferestre sunt compuse din două intervale temporale: saltul temporal și dimensiunea ferestrei, care este creată odată la fiecare salt temporal. Atunci când saltul temporal și dimensiunea ferestrei sunt egale, ne aflăm într-un caz particular, deoarece ferestrele nu se suprapun, acest model de fereastră se numește fereastră thumbling.

Evenimente sunt împărțite de această fereastră în bucăți temporale pentru a putea fi procesate, aceasta se datorează politicii de input a datelor. Operațiile se efectuează chiar dacă față de fereastra anterioară nu s-a modificat structura evenimentelor, deoarece acestea sunt împărțite în bucăți temporale. De exemplu:



### 3.1.5 Snapshot Windows

Acest tip de fereastră definește de asemenea evenimente dintr-o perioadă de timp asupra cărora se pot efectua diferite operații. Totuși diferența față de hopping windows este împărțirea timpului, acesta fiind delimitat de începutul și de sfârșitul evenimentelor, astfel dimensiunile temporale vor fi inegale. Dimensiunile ferestrelor sunt egale cu cele temporale, dar doar când in acel moment temporal există cel puțin un eveniment.

Definirea acestui tip de fereastră nu are nevoie de nici o componentă deoarece dimensiunile ferestrelor vor fi alocate dinamic după apariția evenimentelor, acestea fiind adaptate de politica de intrare a datelor. Pot fi folosite pentru implementarea ferestrelor de alunecare, acestea fiind foarte utile în momentul în care vrem să aplicăm agregări asupra unui operator de tipul grupare, deoarece fluxul de evenimente este comprimat.

### 3.2 Event flow debugger

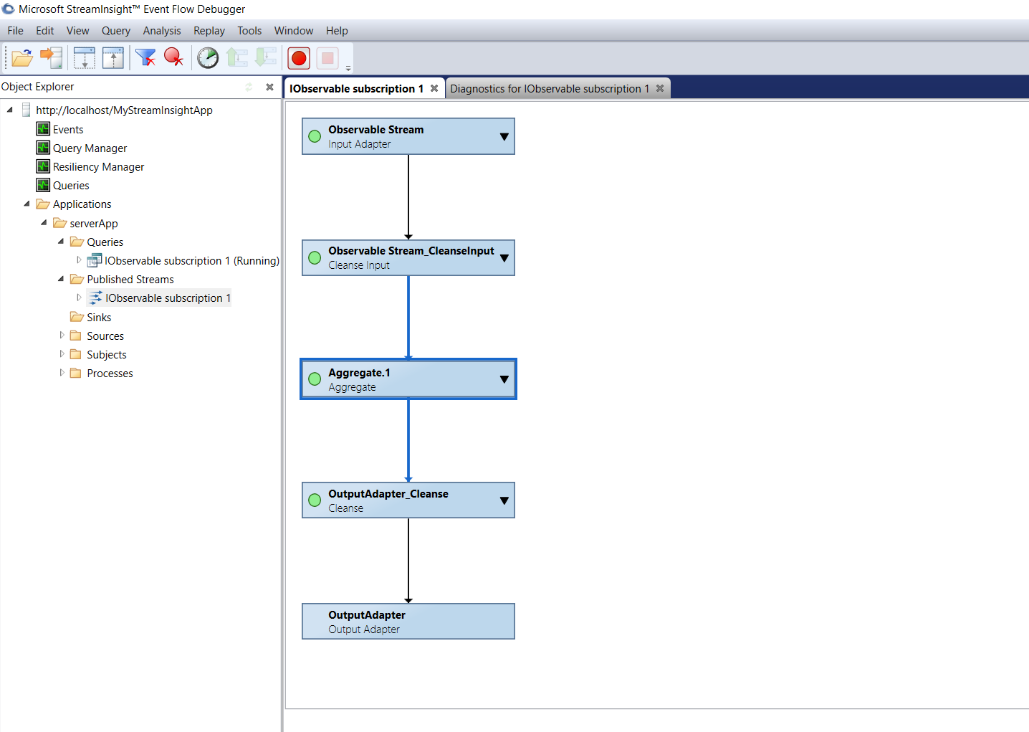
Majoritatea aplicațiilor produc fluxuri mari de date, iar aplicațiile de administrare ale acestora trebuie sa aibă o latență scăzută și relații complexe între fluxuri. Datorită elementelor temporale din fluxurile de date, determinarea unui răspuns al operațiilor cât mai corect este esențială, dar și determinarea cauzelor problemelor interogărilor într-o eventuală eșuare.

Astfel s-au creat nevoi pentru un nou instrument care să poată administra foarte multe date, dar care să reducă problema spațiului de căutare. Cerințele de riguroase de consistență trebuie de asemenea administrate, iar soluția sau diagnosticul trebuie să fie ușor de intuit.

Event flow debugger este o interfață grafică autonomă dată de Microsoft StreamInsight, care te ajută ca developer sau ca administrator al unei aplicații de procesare complexă a evenimentelor, să faci debugging asupra unei interogări create peste un flux de date

Trebuie înțeles faptul că acest instrument nu este un debugger de controlare al fluxului. Controlul fluxului construiește programele in diferitele limbaje de programare, dar pentru ca acestora să li se facă debugging, acceptând puncte de întrerupere și rulând codul în mod normal până la întâlnirea unuia, se pot vedea stările actuale ale sistemului, variabilele existente în cod, se poate intra sau ieși din anumite funcții totul până la finalul execuției, terminate în mod normal sau de utilizator. Pe altă parte event flow debugger analizează evenimentele raportându-se la trecerea timpului și se mută de la o interogare CEP la o alta, de la un nivel diferit. Aici debuggingul arată cum un eveniment are efect asupra fluxului de date și cum se generează evenimente noi după ce au fost manipulate evenimentele de intrare. Astfel față de un debugging al fluxului de control acesta arată modul în care operatorii afectează fluxul de date și nu cum funcționează ei mai exact. Astfel vom putea înțelege impactul evenimentelor asupra celui analizat și impactul fiecărui eveniment în parte asupra celorlalte evenimente.

Acest instrument autonom își propune să facă debugging pe o urmă a unui flux de evenimente bazându-se pe înregistrarea în timp real a unei anumite interogări, fiind conectat la un server în timp real sau această urmă mai poate fi generată în afara degubber-ului, într-un fișier și apoi încărcată pentru a putea fi analizată. Debugger-ul își mai propune să monitorizeze serverul, astfel să se poată obține identificarea stării despre obiectele aplicației și despre listele Object Explorer. Cu ajutorul interfeței putem vedea modul de funcționarea al interogărilor, le putem porni sau opri, le putem activa sau dezactiva în timpul execuției.

Se poate conecta la o aplicație StreamInsight ca si un client sau local pentru a înregistra evenimentele rezultate după efectuarea a unei sau mai multor interogări, dar debugger-ul se mai poate folosi fără a fi conectat la un server, pentru a analiza interogările pe urmele de date obținute offline. Astfel obținem o flexibilitate în modul de lucru, deoarece putem face deubgging pe o anumită interogare cât timp fluxul este în desfășurare sau putem să o testăm pe urmele de date obținute offline.

Înregistrarea interogărilor evenimentelor în timp real se poate face dacă la event flow debugger este conectat un server live de StreamInsight. Ca evenimentele să fie stocate de debugger, serverul trebuie să fie unul autonom și să ruleze pe același computer. Totuși pentru ca o aplicație să fie conectată la degubber aceasta trebuie să își activeze serviciile web, clientul trebuie să aibă permisiuni asupra serverului pentru a se putea conecta la server. Apoi din interfață se va conecta la o adresă care a fost inițializată în server. După ce s-a realizat conexiunea în secțiunea de Object Explorer, vom putea vedea o ierarhizare a obiectelor unde pot fi găsite interogările pentru a se face deubgging. Apoi procesul de înregistrare poate începe. Putem observa partea de ierarhizare în imaginea de mai sus.

StreamInsight poate fi atât o aplicație autonomă cât și una hostată, dar dacă aceasta nu are disponibil un serviciu web nici un fel de client nu se poate conecta, astfel nici debuggerul nu va putea face legătura cu acesta. Astfel s-a creat necesitatea unui mecanism de diagnosticare și rezolvare a problemelor în cazul acestor situații. Pentru rezolvarea acestei probleme se stochează istoricul evenimentelor venite de la un server cu ajutorul utilitarului trace.cmd care este inclus în StreamInsight. Pașii de manipulare ale acestui istoric pentru ca mai apoi datele să fie procesate de event flow debugger sunt : crearea unui fișier cu extensia .etl care să înregistreze datele venite de la server, acestea sunt înregistrate cu ajutorul comenzii trace.cmd start, în momentul în care s-a înregistrat suficientă informație se poate opri înregistrarea cu ajutorul comenzii stop, după aceea datele obținute vor putea fi încărcate manual in event flow debugger pentru a putea analiza datele. În acest caz secțiunea de diagnosticare a obiectelor pe nivele nu poate fi afișată. Utilitarul folosit în acest caz este un script bazat pe comanda Windows Logman, care se folosește de Event Trace for Windows (ETW) pentru colectarea datelor. În timpul procesului de încărcare unele date se pot pierde, dar debugger-ul va anunța acest eveniment, acest lucru se poate evita dacă zona tampon a fișierului își mărește dimensiunea sau se alocă mai multe zone tampon pentru un fișier.

Pentru analizarea interogărilor event flow debugger are anumite funcționalități cheie. Pentru înțelegerea limbajului integrat al interogărilor (LINQ) și a modului în care interogările din acest limbaj acționează asupra fluxului de date, debugger-ul pune la dispoziție vizualizarea planurilor de interogare, a operatorilor si a stream-urilor de evenimente. Datele din fluxul de date sunt puse la dispoziție pentru a putea fi analizate, atât cele de intrare cât și de ieșire, împreună cu calculele intermediare aferente. Pentru fiecare dată se pune la dispoziție timpul de start și de sfârșit si încărcătura utilă. Operatorul Group partiționată fluxul de date, iar operatorul Apply îl modifică, cu toate acestea fluxul poate fi urmărit în debugger. Problema spațiului de căutare și corelarea între stagiile fluxului de evenimente este redusă de abilitatea de analiză globală a acestuia. Ajută la înțelegerea propagării evenimentelor printr-o interogare a fluxului, deoarece poate trece prin pașii de execuție ai interogării raportându-se la trecerea timpului. Asupra timpului si încărcăturii unui eveniment se răsfrânge impactul interogărilor, acesta putând fi analizat și înțeles prin vizualizarea schimbărilor stărilor evenimentelor. Putem observa viitoare date care trebuie procesate și care vor afecta rezultatul curent.

Una din caracteristicile debugger-ului este reluarea, aceasta te ajută să treci de la un operator la altul și să poți observa progresul de-a lungul fluxului sau poți să aplici puncte de întrerupere asupra grafului de interogare care activează acțiunile până la întâlnirea acestuia. Pentru a putea găsi ceea ce a provocat starea actuală a evenimentului, trebuie să putem să ne întoarcem la sursa problemei și să găsim secvența de operații dorită. O altă caracteristică este analiza propagării evenimentelor, astfel se va putea observa efectul evenimentelor asupra fluxului de date, odată cu schimbarea stării acestuia, dar și efectul asupra altor evenimente sau cauza pentru care s-au generat evenimente noi.

Interogările din StreamInsight sun descompuse în noduri de execuție, acestea fiind reprezentate de operatori. Operatori suportați de query-uri sunt Select (Filter), Project, Import, Export, Group-and-Apply, Join, Multicast, Union, Top-K, AlterLifetime, Advancetime și Cleanse [4]. Operatorii din graful de interogare s-ar putea să nu corespundă in totalitate cu cei din interogarea LINQ, deoarece unii operatori sunt introduși în mod automat precum cel introdus de optimizatorul de interogare Clenease, care sortează datele și care nu se regăsește in interogarea LINQ. Totuși acestea nu îngreunează înțelegerea grafului interogării.

Fiecare grilă de evenimente are un tabel operator, fiecare tabel având ca denumire numele operatorului pe care îl reprezintă și tipul acestuia. Filtrul, fiind un text box, acceptă cod C#, care să ofere condiționarea dorită pentru a obține anumite rezultate. Câmpurile evenimentului este o altă componentă a grilei de evenimente, acestea conțin tipurile evenimentelor definite de utilizator pe tot parcursul evenimentelor interne. Aceste câmpuri sunt atribuite în mod automat, dar pot fi modificate prin adăugarea sau eliminarea unor câmpuri. Tabela creată cu aceste câmpuri va fi afișată în ordinea stabilită de debugger, evenimentele putând fi exportate intr-un fișier pentru o analiză viitoare executată de alte programe.

Când vine vorba de operatorul de grupare și aplicare îl includem într-o categorie specială, gruparea după cum ii spune și numele grupează evenimentele la momentul intrării asupra cărora se aplică mai mulți operatori, pentru datele de ieșire se folosește operatorul de grupare unificat. În momentul extinderii unui astfel de nod se vor afișa toate fluxurile de date grupate după o valoare numită cheie pentru care s-a efectuat operatorul de aplicare.

Alte utilități ale debuggerului sunt posibilitatea de a vedea graful de interogare în ansamblu, trecând sau nu prin operatori. Este foarte ușor de folosit, deoarece acțiunile posibile sunt reprezentate atât prin butoane cât și în opțiunile meniului. În momentul conectării debuggerului la server , orice eroare survenită pe parcurs va fi returnată, în limba engleză, toate celelalte erori definite de utilizator vor fi folosite pentru diagnosticarea diferitelor probleme. Operatorii pot fi vizualizați deodată sau pot fi închiși cu ajutorul funcționalităților de extindere sau restrângere. Pentru o mai ușoară folosire a debugger-ului, meniul acestuia pune la dispoziție o varietate de instrumente prin care ne putem personaliza anumite specificații precum data și ora pentru zona de timp pentru toate operațiile din graful de interogare sau putem alege segmentele de date care să fie încărcate din fișiere sau dimensiunile de încadrare ale înregistrării unui flux de evenimente în momentul în care acesta este conectat la un server.

Acesta nu este numai un instrument de depanare, el mai deservește și ca un instrument de monitorizare pentru serverul de StreamInsight. În interfața acestuia vom putea regăsi pe lângă operatori niște statistici ale obiectelor cu privire la stările acestora în momentul rulării. De asemenea se pot vizualiza date despre anumite interogări care au fost înregistrate de către server, precum date despre întârzierea, consumul si producerea de date, dar și memoria utilizată pentru efectuare unei interogări.

### 3.3 LINQ

Este o tehnologie bazată pe integrarea interogărilor în limbajul de programare C#. Aceste interogări se pot face pe diferite surse prezum bazele de date, documente XML sau diferite servicii web.

### 3.4 Dynamic SQL

Acest tip de interogări este foarte util atunci când nu este cunoscut de la început momentul apariției datelor, astfel pentru evitarea supraîncărcări memoriei cu date se va folosi SQL-ul încorporat denumit și SQL dinamic. Această dinamicitate este dată de momentul construirii interogării, față de cel clasic acesta se construiește doar în momentul rulări programului. Cu toate că, sunt create în mod dinamic de cele mai multe ori au un timp de răspuns mult mai mare, pentru că sistemul de gestiune al bazei de date trebuie să creeze planul de acces la momentul rulării. Totuși este mai eficient în mare față de cel static deoarece programul construiește într-o memorie tampon o pre execuție a acestuia, iar variabilele vor fi înlocuite cu constante care se vor înlocui ulterior cu noile valori. Codul va fi executat utilizând comanda EXECUTE, acest cod poate fi rulat de mai multe ori, cu valori ale parametrilor difetiți.

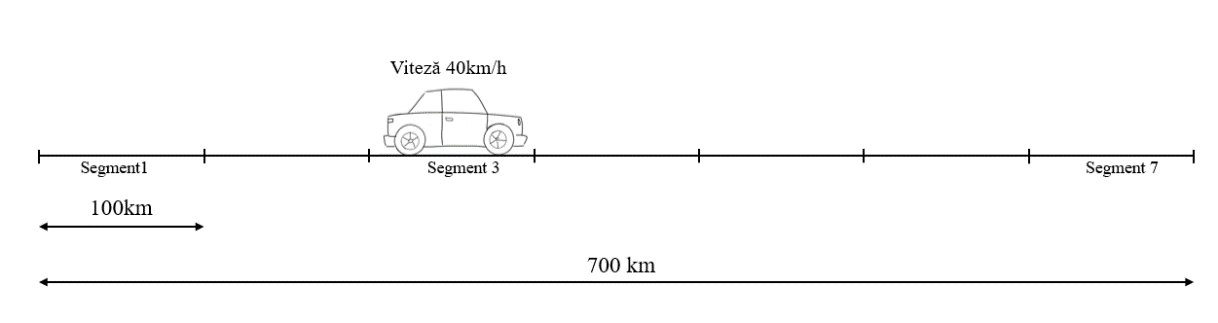
# Capitolul 4

## Studiu experimental

### 4.1 Despre

Ca prim experiment, am început prin a studia un flux de evenimente, în care fiecare eveniment este reprezentat de o mașină care are ca și date viteza cu care merge și numele mașinii. Pentru a putea vedea impactul asupra diferitelor ferestre de timp, trebuie calculat mai întâi care ar fi dimensiunea optimă a ferestrelor. Astfel la terminarea unui flux de date, după ce acestea au fost salvate într-o bază de date uniform distribuită, se vor prelucra date, de la 1...n până cand marja de eroare dintre rezultatul obținut si rezultatul mediu obținut după terminarea interogărilor este minim.

Al doilea experiment pornește de la studiul eficiențelor tipurilor de ferestre existente descrise în capitolul 2. Astfel pentru a pune în valoare modul în care acestea au efect asupra datelor avem nevoie de un scenariu plauzibil în lumea reală. Pentru asta vom introduce un scenariu din trafic, aglomerația este o problemă contemporană, de aceea am încercat să studiem un segment de drum pentru a observa vitezele medii de pe totalitatea acestuia. Pe acest drum fiecare mașină va transmite date despre ea și despre poziția ei în trafic, adică segmentul porționat de drum din care face parte. Acest scenariu a pornit de la studiul intitulat Linear Road [6] care mi s-a părut fezabil de a fi studiat pe comunitatea locală, la un nivel mult mai mic. De exemplu totalitatea rețelei stradale din orașul Cluj-Napoca are o dimensiune de aproximativ 700km, în care viteza medie este de 46km/h, aceasta putând fi împărțită în segmente de 10 km, astfel vom avea 70 de segmente egale. La orele de vârf se va putea observa o scădere a vitezei medii pe anumite segmente de drum datorită aglomerației și astfel ne vom putea da seama de momentul în care segmentul începe să se suprasatureze. Astfel datele vor fi primite la un interval de t secunde, de aceea ele trebuie să fie procesate în mod continuu, deci aplicația va trebui să suporte interogări continue asupra acestor date. După ce datele vor fi prelucrate acestea vor fi transmise mai departe pentru prelucrări statistice și pentru analizarea eficienței.

Figura 1

După cum se poate observa în Figura 1, aici totalitatea drumurilor din orașul Cluj-Napoca au fost concatenate într-un singur drum, care mai apoi a fost împărțit în 7 segmente egale fiecare având câte 100km. Pentru o precizie mai bună a porțiunilor aglomerate, acesta va trebui împărțit în segmente de dimensiuni mai mici, de exemplu de câte 5-10km, așa cum am exemplificat mai sus.

Această aplicație se bazează pe o arhitectură StreamInsight, astfel ea trebuie să susțină un flux de evenimente, care au o încărcătură și care să poată fi depanată la nevoie. Astfel am început prin crearea unui server local, utilizând instanța implicită care pentru care se creează un punct de acces putând fi găsit la momentul depanări de către event flow debugger la o adresă specificată.

Pentru cele două experimente am folosit o distribuție uniformă a datelor astfel datele continue generate vor veni în mod aleatoriu, cu ajutorul funcției de Random din limbajul C#. Presupunem faptul că datele vin în ordinea apariției lor, astfel timpul de apariție de care avem nevoie este timpul curent pentru a le putea înregistra în flux.

Între drum și mașini se creează o anumită relație de interdependență, deoarece drumul pe care îl studiem este împărțit în segmente și pe fiecare segment există un anumit număr de mașini, care au o viteză care va determina viteza medie a fiecărui segment.

Pentru început avem nevoie de sursa de evenimente, care mai apoi să fie transformată într-un flux de date care să poată fi procesat de diferitele tipuri de interogări. Aceste interogări vor fi de trei feluri, cele mai simple dintre ele vor selecta datele din flux care îndeplinesc o anumită condiție. Cel de-al doilea tip de interogări sunt acelea asupra cărora pe lângă faptul că fac parte din fluxul de evenimente și îndeplinesc o anumită condiție(nu neapărat), aplică asupra datelor operatori de agregare precum suma, media, maximul sau minimul. Cel mai complex tip de interogare pe care îl voi folosi este cel care adaugă tipului dinainte gruparea, adică grupează elementele din secvența de intrare după anumite atribute ale evenimentelor procesate, în cazul nostru mașinile vor fi grupate pe segmente de drum, dar ar mai putut fi grupate și după viteza pe care o au. Această grupare o facem pentru ca să putem analiza pe fiecare segment de drum media aferentă care ne oferă anumite informații.

Asupra primului tip de interogări vom lucra direct asupra fluxului de evenimente și ne dorim să aflăm care sunt mașinile care în oraș nu depășesc viteza legală, adică 50km/h. Deci interogarea va trebui să ne ofere date despre toate mașinile care au intrat în flux, care respectă condiția impusă mai sus, fără să țină cont de segmentul de drum din care fac parte sau momentul l-a care s-au înregistrat. Ca parte a studiului vrem să vedem impactul dimensiuni fluxului de evenimente asupra interogării cu privire la întârzierea de răspuns a interogării sau utilizarea memoriei de procesare, care sunt diferențele, dacă sunt, față de rezultatul adevărat.

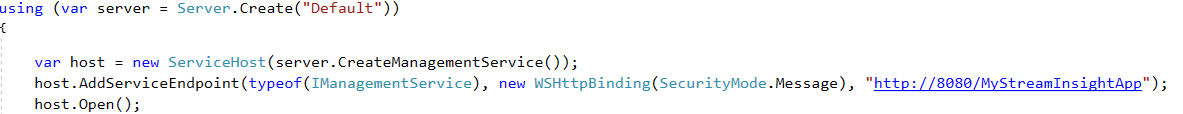
Celelalte două tipuri de interogări, nu vor mai fi aplicate direct pe flux, ci pe segmente continue din fluxul de date, aceste segmente se numesc ferestre, care prelucrează bucăți de flux în diferite moduri. Acestor ferestre li se vor aplica mai întâi agregări, adică fiecare fereastră va calcula media mașinilor din flux, rezultatele vor veni în mod continu pe datele prelucrate până când datele se vor termina. Apoi la aceste interogări se va adăuga și o grupare a datelor pe segmentele de drum din care fac parte.

Timpul este și el segmentat, am ales ca segmentele temporale să fie de câte o secundă, adică toate evenimentele care sunt introduse în acea secundă in fluxul de date vor face parte din acel segment de timp. Am creat aceste segmente temporale pentru a putea simula mult mai bine aproximarea datelor care se face în ultimul moment temporal. Pe aceste momente temporale se va alege un punct aleator pentru care se va aproxima începând de la acest punct până la punctul inițial, în intervale de timp tot mai mari, de la o secundă(ultimul moment temporal), până la punctul fixat. De exemplu pe un flux de date care este împărțit în 10 segmente temporale, punctul se va fixa la 7 astfel se vor calcula aproximările începând de la fereastra temporală 7-6, apoi 7-5 până la fereastra de dimensiune 7, 7-1.

Pentru prelucrări ulterioare, statistici și studiu experimental, am ales ca anumite date să fie scrise într-o bază de date. Datele din fluxul de evenimente au fost scrise în baza de date pentru a se putea calcula rezultatul exact cu ajutorul unor interogări directe pe baza de date și nu pe flux. Pentru fiecare mașină se vastoca astfel un număr de identificare unic, segmentul de timp din care face parte, segmentul de drum și viteza pe care o are. Rezultatele provenite din interogări vor fi stocate de asemenea într-o tabelă în baza de date, pentru primul tip de interogări simple se vor stoca direct mașinile, dar pentru celelalte două tipuri de interogări se vor stoca timpul de apariție, media și grupul din care face parte. Deoarece pentru cel de-al doilea tip de interogări nu avem o grupare pe segmente de drum acestea vor si salvate ca un segment 1, adică tot drumul este un singur segment.

### 4.2 Implementare

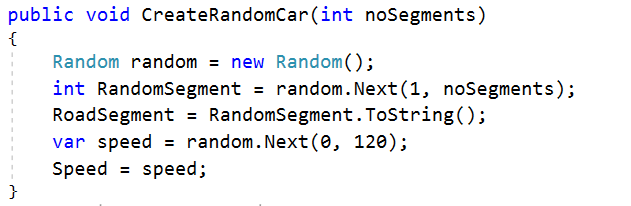
### 4.2.1 Server

 Pentru implementarea serverului avem nevoie de o adresă care să îl găzduiască, întâi avem nevoie de un server creat local care mai apoi să fie găzduit la o anumită adresă. După cum se poate observa serverul a fost creat și pus în găzduirea site-ului: *http://8080/MyStreamInsightApp* (1).

După cum putem observa, instanța de creare a serverului este cea pe care am dat-o ca și instanță la instalarea utilitarului. Datorită serviciului de găzduire se activează un punct de serviciu web cu un identificator uniform de resursă specificat mai sus la (1). După ce variabila de găzduire a fost creată cu un serviciu aferent , trebuie să se creeze un punct de acces la care să fie găsită aplicația, cu ajutorul serviciului securizat de http, acesta fiind un serviciu de administrare. Acum că punctul a fost setat, acesta poate fi deschis pentru a putea fi folosit cu ajutorul comenzii de Open().

### 4.2.2 Sursa de evenimente

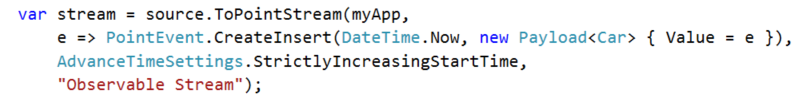
Fiecare eveniment este construit din antet și încărcătura utilă, după cum am discutat mai sus in capitolul 2.1. În cadrul aplicației mele încărcătura utilă conține o referință către un obiect generic, adică acesta va putea fi apelat pentru orice entitate. Am ales ca scenariu care are la bază o mașină, aceasta fiind unitatea pe care se bazează aplicația. În entitatea din încărcătura utilă a evenimentului(mașina) se regăsesc următoarele atribute: viteza mașinii (0...Vm\*) care poate fi definită ca un întreg și segmentul de drum din care face parte, denumirea acestuia, definită ca un șir de caractere. În exemplul din Figura 1, mașina se află pe segmentul 3 de drum și are o viteză de 40km/h.

Pentru cele două experimente am folosit o distribuție uniformă a datelor astfel datele continue generate vor veni în mod aleatoriu, cu ajutorul funcției de Random din limbajul C#. Presupunem faptul că datele vin în ordinea apariției lor, astfel timpul de apariție de care avem nevoie este timpul curent pentru a le putea înregistra în flux. După cum putem observa, următoarea secvență generează datele din flux, aceasta primește ca parametrii numărul de segmente în care este împărțit drumul și generează date pentru segmentul de drum si viteza, care poate fi între 0, adică mașinile sunt oprite pe loc și 120 viteza maximă admisă pe segmentul de drum de autostradă, această funcție nu returnează nimic deoarece ea este inclusă în clasa obiectului astfel atributele obiectului sunt vizibile și editabile :

Ca și relații în acest flux avem relația între drum și mașinile care trec pe acest drum, deoarece odată la o perioadă de timp specificat în milisecunde, numărul de segmente în care este împărțită strada pe care se află mașinile, fiecare mașină trebuie să se încadreze în aceste segmente pentru a putea fi relevantă pentru studiu, dar și numărul total de mașini existente pe tot segmentul de drum pe tot parcursul experimentului.

(2)

După cum putem observa în această secvență se creează o sursă de obiecte aleatorii, care are ca parametrii : NoSeconds, ce reprezintă numărul de milisecunde la care să se adauge o nouă valoare în sursa de date, acesta este calculat ca 1000/numărul de valori. NoValues reprezintă numărul de valori care trebuie generate pe fiecare unitate temporală, CountMax este o variabilă de tip întreg care reprezintă numărul total de mașini pe care eu doresc să le studiez, deoarece studiile sunt bazate pe performanță trebuie să vedem care este efectul diferitelor dimensiuni, în general în creștere, asupra interogărilor.



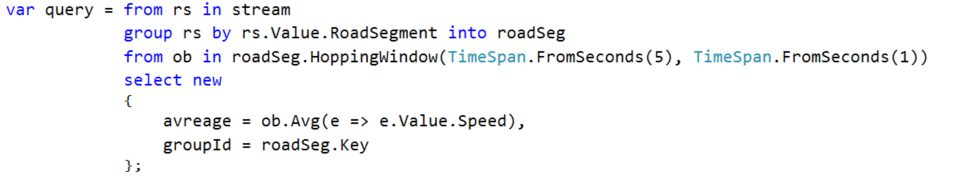
(3)

Sursa de date trebuie convertită la un flux de date, astfel se creează pentru fiecare data un punct de flux. Știind din capitolul 2.8.1, că cei doi operatori disponibili asupra fluxul de date sunt inserarea și ștergerea, am ales pentru această sursă să inserez fiecare dată la momentul actual, modelat desigur după încărcătura utilă (Payload). Deoarece sursa de date creează un nou eveniment la câteva milisecunde, după fiecare astfel de eveniment se incrementează timpul curent (CTI event), aceasta fiind asigurată de condiția din apelarea metodei de creare a unei inserări de evenimente: AdvanceTimeSettings.StrictlyIncreasingStartTime. Aceste puncte sunt acum pregătite pentru a putea fi „consumate” de server.

### 4.2.3Interogările

Deoarece asupra datelor care vin în mod continu avem nevoie de agregări, medii ale vitezelor mașinilor grupate pe segmentele de drum, care să prelucreze datele în timp real, astfel simplele interogări în SQL sunt depășite și avem nevoie de un limbaj care să vină în compensația acestuia. De aceea asupra fluxului de date am aplicat interogări continue, care au la bază anumiți pași de creare și executare pentru o analiză eficientă a datelor.

Pentru a putea interoga date avem nevoie de surse care să poată fi interogate, de aceea în (2) și (3) s-a definit fluxul de intrare în server. De acum datele vor fi prelucrate în diferite moduri pentru a obține rezultatul dorit.

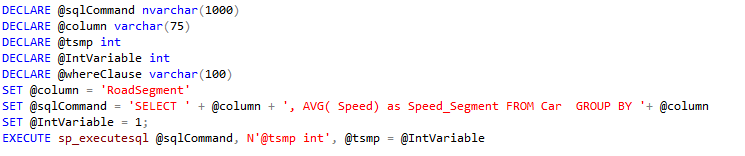


(4)

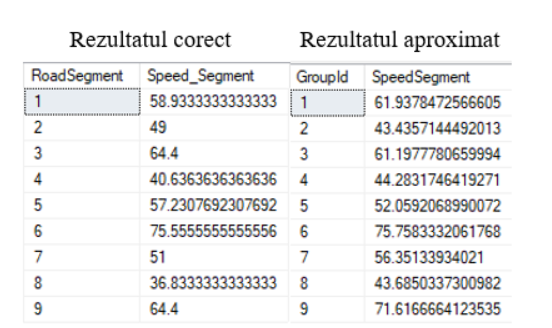
După cum putem observa, interogarea scrisă la (4), e o combinație între limbajul LINQ și C#, această funcție declarativă descrie procesarea evenimentelor venite în flux. Această interogare produce ca și rezultat un alt flux de date care are ca și informație utilă media mașinilor, adică câmpul avreage si segmentul de drum din care mașina face parte, adică groupId. Pentru a găsi un rezultat aproximat cât mai aproape de cel real am început prin a grupa mașinile după segmentul din care provin și le vom calcula media la fiecare secundă pe segmente de fereastră de câte cinci secunde.

În interogarea de la (4) am folosit o fereastră de tipul hopping care are ca și dimensiune cinci secunde, iar dimensiunea saltului este de o secundă. Adică ferestrele se suprapun câte patru secunde cu următoarea fereastră de calcul. Cu toate acestea există diferențe de calcul între cel exact, calculat în SQL Server cu ajutorul următoarei secvențe de cod și cel aproximat de acest tip de fereastră. În interiorul ferestrei avem operatorul de agregare care calculează media(Avg). Ca și date de ieșire interogările au tipul devenimentului (inserat/șters), data la care a fost introdus eveniemntul și la care acesta nu a mai fost activ și încărcătura utilă, creată aici ca un obiect nou.

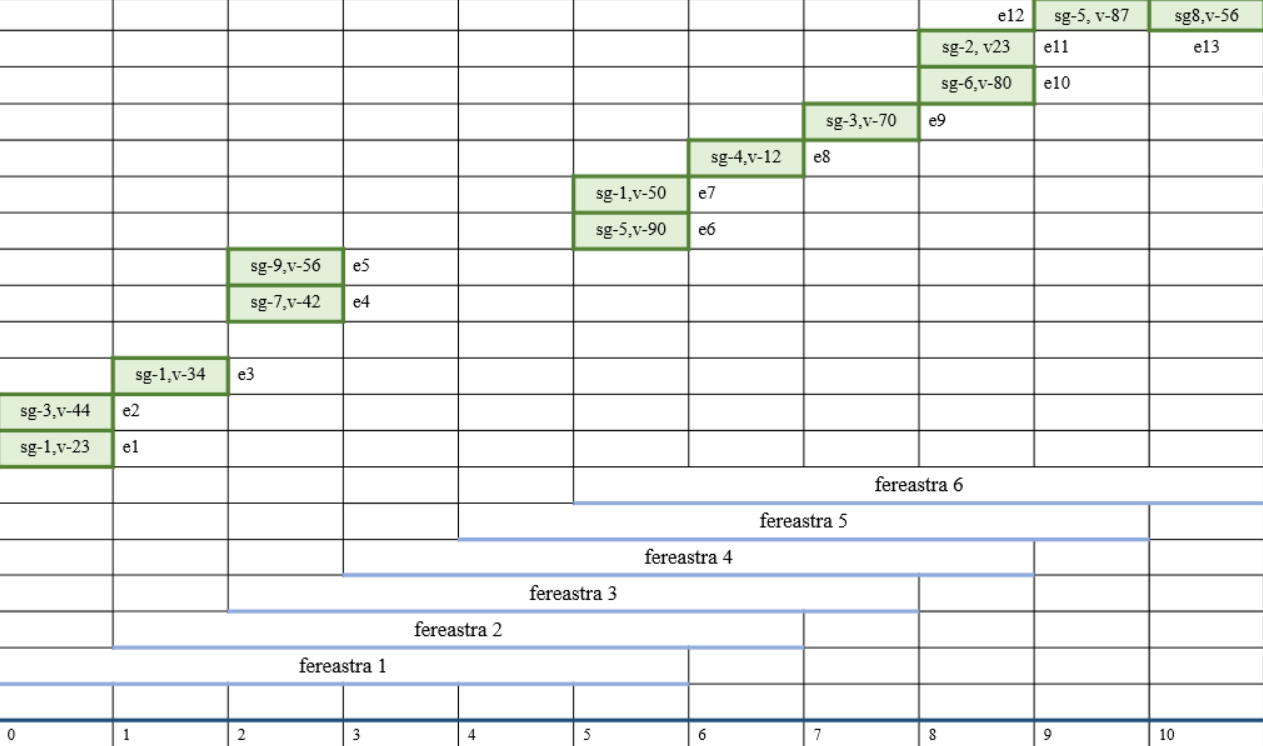
(5)

 După cum se poate observa, această bucată de SQL, este un SQL dinamic, care transformă șirul de caractere @sqlCommand într-o interogare. Am declarat @column ca un șir de caractere pentru coloanele de care avem nevoie din cadrul interogării. Pentru o dezvoltare ulterioară în care datele vor fi computate pentru fiecare timp de apariție al evenimentului. Adică vom considera că într-un interval de t secunde vor apărea un anumit număr x de evenimente, deci odată la x evenimente timpul de apariție se va mări cu unu.

Deoarece nu este o interogare foarte complexă și nu are nevoie de un plan de execuție foarte mare, timpul de răspuns este mult mai mic, așadar ne oferă o eficiență sporită. Acest cod va putea fi pus într-o procedură stocată pentru a putea fi parametrizat și apelat pentru fiecare valoare temporală.

 Următoarele rezultate fac o comparație între (4) și (5), având ca date de intrare 100 de mașini de pe 9 segmente de drum. După cum se poate observa că la aceste dimensiuni ale ferestrei hopping, diferența între rezultatul corect și cel aproximat, în medie este de 5.769. Așadar pentru a obține un rezultat mai bine aproximat va trebui să mărim dimensiunea ferestrei sau să mărim dimensiunea hop-ului? Despre asta vom discuta în capitolul următor la rezultatele studiului experimental.

Pentru a putea înțelege mai bine cum funcționează acest tip de ferestre pe care le-am aplicat am construit următorul grafic în care am 13 evenimente care vin în decurs de 10 secunde și asupra cărora am aplicat interogarea de la (4):

 În imaginea de mai sus avem reprezentat în chenarele verzi mașinile care se află în flux în acel moment, se poate observa că se vede încărcătura utilă a fiecărui eveniment, acestea au ca timp de intrare timpul local al mașinii si rămân în flux timp de o secundă. Ferestrele sunt de tipul hopping și sunt marcate cu linii albastre pe toată durata lor de viață, fiecare fereastră are un timp de start și un timp de final între care trebuie să se încadreze evenimentele, care au trecut de politica de intrare, adică au fost tăiate la dimensiunea ferestrei. Timpul este reprezentat de linia albastră, de-a lungul acesteia putem urmări în intervalul de timp (0 ... 10), ce se schimbă în fiecare secundă.

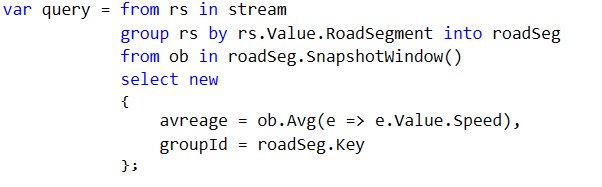
După cum se poate observa avem 6 ferestre care grupează cele 13 evenimente primite și pentru fiecare le face media vitezelor, în acest caz granularitatea timpului este foarte fină deoarece șansele ca evenimentele să se piardă în calcul sunt destul de mici. În secundele 3 și 4 nu sunt date care să vină in flux, ceea ce poate fi generat de pierderi de memorie sau de întârzieri ale sistemului. Fiecare eveniment rămâne în acest caz în flux timp de o secundă în timp ce este prelucrat de una sau mai multe ferestre. Evenimentele e1(sg-1,v-23) sau e2(sg-3,v-44) vor fi prelucrate doar de fereastra 1 în schimb evenimentele e6(sg-1, v-50) sau e7(sg-1, v-50) vor fi prelucrate de toate ce 6 ferestre.

Rezultatele oferite de acest tip de ferestre sunt recalculate odată cu apariția fiecărui nou eveniment, fiind grupate după segmentul de drum, în acest caz avem 9 grupări. Astfel pentru fiecare astfel de grup se definește o nouă fereastră în care se calculează media în acord cu valoarea vitezei mașinii. Operatorul de medie Avg care este folosit în interogarea (4), este un operator de agregare care este un operator incremental.

 În momentul în care nu ne dorim să ne construim o fereastră cu dimensiuni bine stabilite ci una care să fie influențată de apariția evenimentelor și să de construiască in mod dinamic vorbim despre ferestrele de tipul shnapshot. După cum le spune și numele aceste ferestre instantanee sunt ferestre dinamice care se redimensionează după timpul de început și de sfârșit al evenimentelor.

Pentru a putea observa diferențele dintre ferestrele hopping și acestea, am luat ca și date de intrare același flux de date. În figura de mai sus avem prezentate mașinile în căsuțele verzi, cu portocaliu sunt ferestrele de tipul snapshot care se creează, iar linia mov reprezintă timpul.

Marea diferență observabilă este modul în care ferestrele gestionează evenimentele, reduce din redundanța evenimentelor procesate și de asemenea din datele de ieșire. Putem observa asta știind faptul că ferestrele hopping recalculează toate datele din fereastră odată cu apariția unui nou eveniment producând astfel cumva date redundante.

 Cea mai complexă dintre interogările scrise pentru acest tip de fereastră este cea în care am grupat datele pe segmente de drum și am calculat media pe aceasta, așa cum am făcut mai sus la ferestrele de tipul hopping. Aceasta se poate vedea în următorul cod:

O altă interogare cu acest tip de ferestre, este cea în care asupra fluxului de date s-au aplicat doar agregări. Am calculat media tuturor mașinilor din flux, fără să ținem cont de segmentul de drum din care fac parte.

### 4.2.4 Structura

Ca și structură aplicația a avut un server și o bază de date. Pentru început se poate observa în Figura 2, diagrama de clase. Principala clasă care creează o instanță a serverului este clasa StreamInsight. În această clasă sursa de evenimente este transformată într-un flux de date, sunt definite interogările, care mai apoi sunt procesate creându-se sursa de output care va fi scrisă în baza de date.

Legătura cu baza de date, dar și operațiile de I/O se regăsesc în clasa BD, care va scrie toate evenimentele din fluxul de date în tabela aferentă, rezultatele interogărilor, dar va citi din fișier datele pentru primul experiment în care aveam nevoie de un anumit număr de mașini pentru dimensionarea optimală a ferestrei.

Evenimentele din sursa de evenimente sunt create în clasa generică RandomObject<T>, această clasă nu poate fi moștenită și se folosește ca atare. În cadrul acestei clase este implementat un observer care notifică reînnoirea subscipției și astfel a timpului de creare pentru fiecare eveniment în parte. Aceasta primește ca și parametrii timpul la care să fie generat un eveniment, de acea avem nevoie de un obsever, numărul de evenimente totale din flux, care ne asigură condiția de oprire a generării datelor. După ce un eveniment este generat acesta trebuie să fie scris în baza de date, aceasta fiind administrată de funcția DBWrite care pentru obiectul T, în cazul nostru mașina va apela scrierea din baza de date din clasa BD pentru datele mașinii curente și segmentul de timp în care acesta a fost înregistrat.

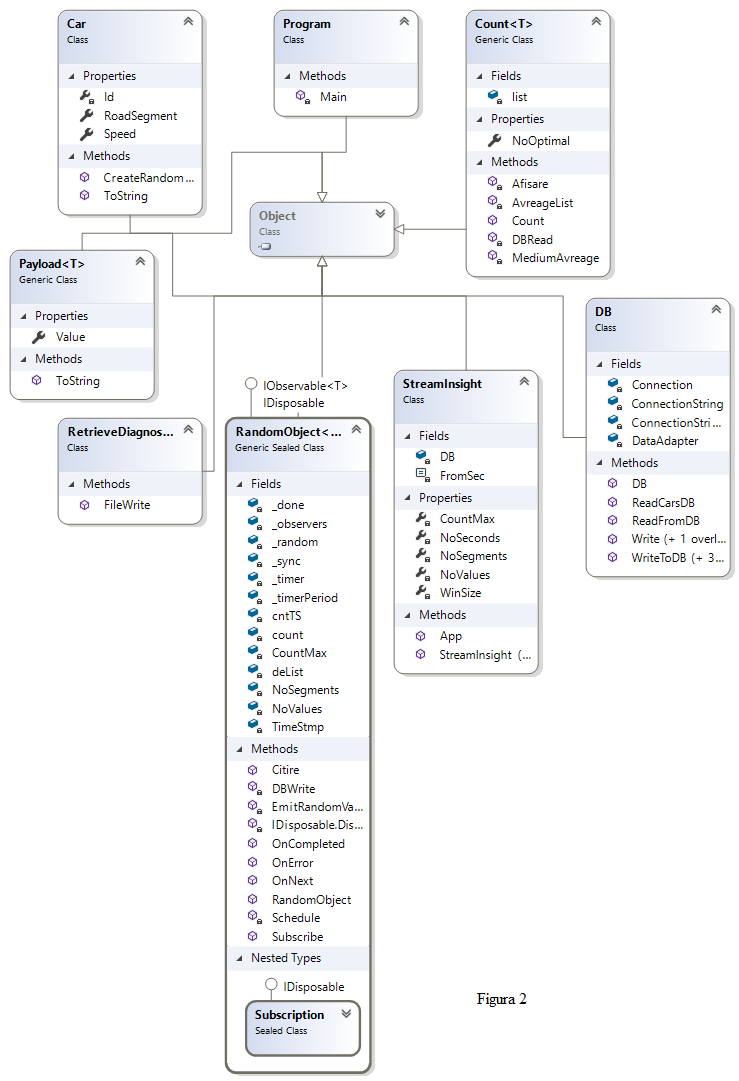
Clasa care reprezintă unitatea de bază a aplicației este Car, ea reprezintă pentru aceste experimente încărcătura utilă din clasa Payload. Aceasta are două metode, una de creare a unui eveniment de tip mașină aleator, deoarece am ales ca datele să fie distribuite uniform avem nevoie de ceva care să ne garanteze acest lucru, astfel am folosit funcția de Random. Metoda de convertire la un șir de caractere este folosită pentru afișarea datelor care intră in flux în consolă.

Payload, adică încărcătura utilă, este o clasă generică care poate fi apelată pentru orice tip de obiect. Aceasta va fi folosită de către interogări ca dată pentru prelucrare. În cazul aplicației mele datele furnizate sunt segmentul de drum după care datele sunt grupate și viteza fiecărei mașini pentru care se calculează viteza medie.

RetriveDiagnostics este o clasă care a fost creată inițial cu scopul de a scrie datele care țin de monitorizarea interogărilor într-un fișier, dar cum am avut nevoie de aceste date pentru prelucrări ulterioare de medii de CPU, întârziere și memorie, am decis că datele ar fi mai bine dacă ar fi scrise și ele într-o bază de date. Aceasta face legătura între clasa destinată serverului și baza de date, pentru monitorizare. După ce primește o listă de tupluri de tipul <cheie, valoare>, va apela pentru fiecare scrierea în baza de date.

Clasa Count<T> este o clasă generică, care ajută la calcularea optimală a dimensiunii ferestrei după o medie de date din flux. Aceasta va cere date din baza de date, începând de la 1 până când se găsește numărul optimal, astfel încât eroarea de calcul să fie minimă.

Toate acestea sunt încorporate și apelate din clasa de bază . NET Program, în care se află funcția Main. Aici se va apela serverul din clasa StreamInsight, bazat pe tehnologia cu același nume, pentru fiecare segment de timp.



Pentru salvarea datelor am ales o bază de date creată in SQL Server, deoarece acesta conferă sprijin pentru gestionarea bazelor de date de dimensiuni medii. Mai jos în Figura 3 se găsesc tabelele din baza de date pe care le-am utilizat pentru post-analiza datelor.

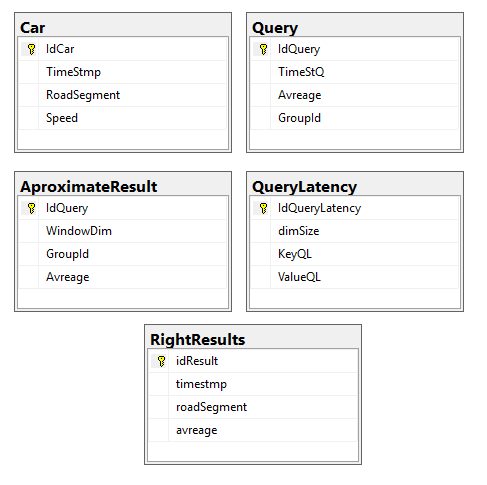
 Tabela Car, este cea în care se salvează datele din clasa cu același nume, are ca și cheie primară câmpul IdCar, TimeStmp reprezintă segmentul de timp al mașinii care va fi de tip întreg, segmentul de drum se regăsește în câmpul RoadSegment care este definit ca un șir de caractere pentru extinderi viitoare, iar viteza este regăsită în câmpul Speed care îl definim ca un număr rațional pentru a putea calcula cu exactitate media corectă a vitezelor.

Figura 3 (Tabelele bazei de date)

În tabela Query se salvează datele despre interogările primite, din cele 3 tipuri de interogări pe care le-am menționat în descrierea aplicației. Pentru aceste date cu ajutorul unei proceduri stocate, care calculează rezultatul aproximat, care mai apoi va fi scris în tabela ApproximateResult.

Pentru monitorizare, spuneam mai sus că avem nevoie să ne stocăm într-o tabelă din baza de date, informații cu privire la memorie, CPU și întârzierea de răspuns. Aceste date sunt salvate în tabela QueryLatency, care pentru a afla rezultatele de interes se va calcula media grupată pe cele 3 informații.

Tabela RightResults este rezultatul calcului dat de asemenea de o procedură stocată care mai apoi va fi comparat cu rezultatul aproximat. Această tabelă stochează aproximativ aceleași date pe care tabela de Query le deține, dar aceste date sunt calculate pe baza de date, unde datele nu se pot pierde, în schimb rezultatele procesate pe fluxul de date nu sunt tocmai exacte.

### 4.2.5 Monitorizare

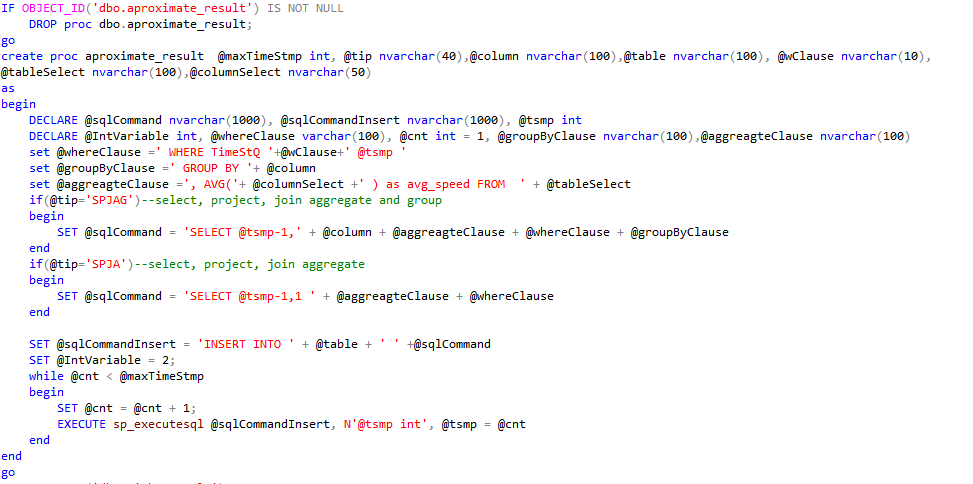
 Întotdeauna problema performanței este prioritară când vine vorba de aplicații soft, așadar am ales să monitorizez interogările cu ajutorul event flow debugger-ului. Pentru a putea accesa datele care ne interesează din debugger trebuie să știm de ce date avem nevoie, apoi să ne creăm o resursă de identificare uniformă cu care să îl accesăm, iar după ce am primit aceste date să le stocăm pentru a le putea prelucra.

În bucata de cod de mai sus este ultima parte a serverului care după terminarea interogărilor și monitorizarea acestora serverul găzduit se închide. După cum știm din 4.2.3, interogările definite se numesc query, ele stochează rezultatul acelei interogări, astfel trebuie puse într-un enumerator pentru ca fluxul rezultat să poată fi parcurs și „consumat”.

Cât timp în rezultatul dat de interogare există date de output se caută evenimentele care au ca tip de eveniment inserarea, adică vom căuta doar ultimele rezultate introduse în fluxul rezultat. Deoarece rezultatul generat de interogare este un flux, acesta conține evenimente, care au o încărcătură utilă numită Payload, astfel pentru fiecare eveniment rezultat vom aduce încărcătura utilă la o formă afișabilă. Aceste rezultate fiind afișate în consolă și memorate în baza de date.

După ce am scris rezultatul în baza de date vrem să vedem câte resurse s-au consumat pentru producerea acelui eveniment. Așadar ne creăm anumite setări pentru diagnosticare, în care ca resursă de identificare uniformă de bază avem “cep:/Server”. Mai departe ca date transmise vom avea diagnosticele de monitorizare și dimensiunea ferestrei pentru care acestea sunt calculate.

Ne dorim să monitorizăm și acuratețea rezultatelor, deoarece datele rezultate trebuie să fie cât mai aproape de cele reale. După ce datele au fost trecute în baza de date ele urmează să fie procesate de două proceduri stocate.

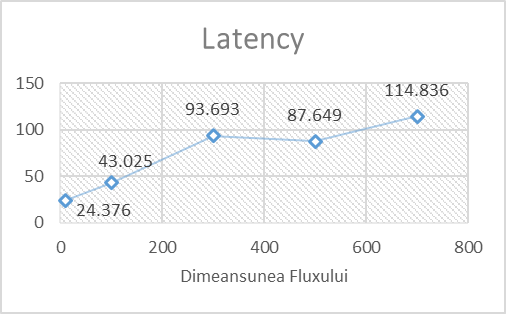
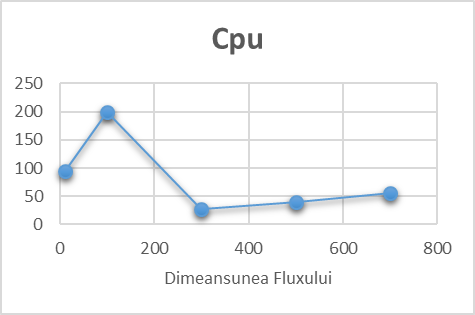
 Prima procedură este cea care calculează rezultatul corect sau cel aproximat, totul depinde de parametrii primiți. Această procedură creează SQL dinamic pentru fiecare interogare făcută asupra tabelelor. Putem observa în codul de mai jos definirea procedurii.

După cum se poate observa această procedură a fost dezvoltată pe baza procedurii din capitolul 4.2.3. Pentru procesarea și modificarea mai rapidă a acestei proceduri, am verificat dacă există ca și obiect în baza de date, dacă există va fi șters apoi recreat după cum urmează.

Am creat pentru ultimele două tipuri de interogări acele selecții de date deoarece am avut de calculat media reală și cea aproximativă, iar fiecare dintre aceste rezultate a fost nevoie să fie stocate într-o altă tabelă.

### 4.3Rezultatele studiului experimental

Pentru studiul experimental am trecut prin cele 3 clase de interogări și le-am studiat comportamentul asupra fluxului de date. Pentru primul tip de interogare selecție-proiecție-join, am dorit să văd care este impactul datelor asupra latenței interogării, dar și asupra folosirii CPU. Studiul pentru această interogare se bazează pe căutarea tuturor mașinilor din flux care au viteza legala în oraș, adică mai mică de 50. Acest tip de interogare are o acuratețe de 99% în ceea ce privește acest experiment. Pentru acuratețe am comparat numărul de rezultate, diferența fiind 0 pentru date cu un volum mai mic de 500 de evenimente în flux. Pentru fiecare dimensiune a fluxului experimentul a fost rulat și calculat de 10 ori, iar aceste rezultate reprezintă media lor.

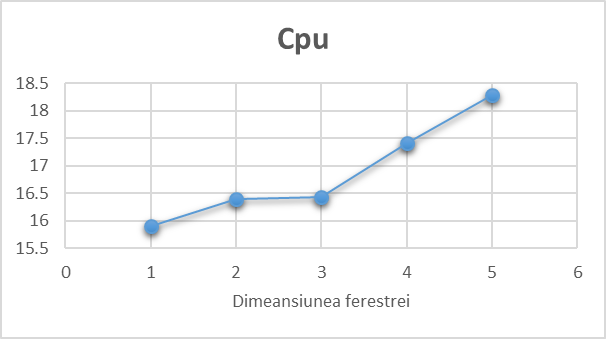
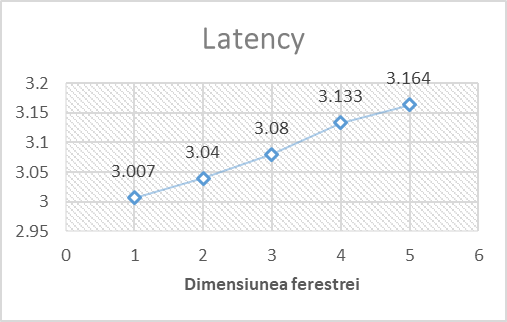
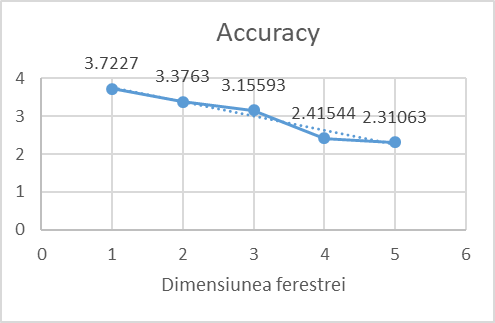


După cum se poate observa în graficele de mai sus, aproximativ odată cu creșterea fluxului crește și latența interogării, dar lucrurile nu stau la fel și când vorbim despre CPU, deoarece există o creștere semnificativă a folosirii acestuia atunci când în fluxul de date sunt 100 de valori, care vin cu o viteză de 10 evenimente/secundă.

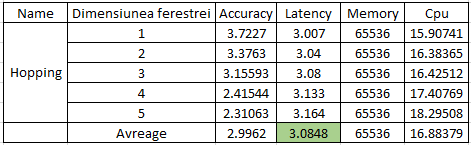
Pentru cel de-al doilea tip de interogare selecție-proiecție-join-agregare, am luat ca scenariu calculul mediei vitezelor fără a ține cont de segmentul de drum din care mașina face parte. Acest tip de interogări pot fi aplicate asupra ferestrelor, în acest caz am studiat acuratețea, latența și CPU pentru ferestre de tipul snapshot și hopping. Dimensiunea fluxului a fost influențată și de rezultatul anterior, observând că fără să aplicăm asupra fluxului nici o fereastră, rezultatele cele mai bune ținând cont de latența și utilizarea CPU, au fost la 500 de date.

De aceea am luat un flux de date care să conțină 500 evenimente, care să vină cu o frecvență de 10 date/secundă și care să fie împărțit în 50 de segmente temporale. Pentru calcularea acurateței am avut nevoie de un rezultat aproximat, dar și de cel real pentru a putea fi comparate. Pentru calcularea celor două rezultate am scris o procedură stocată care să ne proceseze cele două rezultate descrisă în subcapitolul anterior. Pentru fiecare flux ne-am luat un punct de aproximare, la punctul de timp 30, pentru care am calculat rezultatul aproximat. După cele 10 rulări pentru fiecare test avem următoarele rezultate:

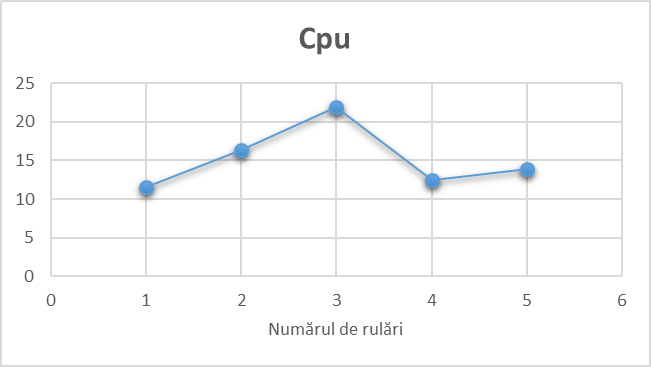
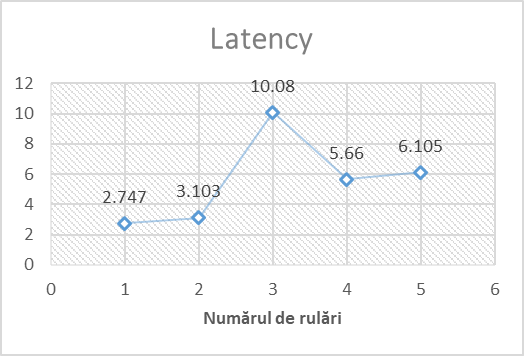
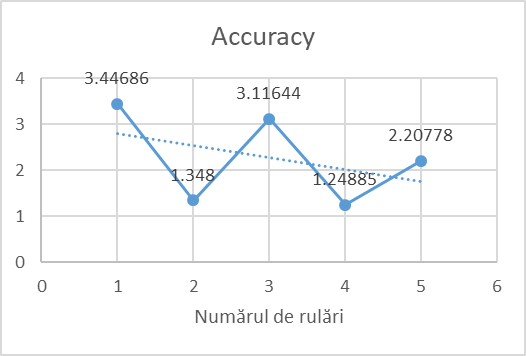
Ferestre Hopping

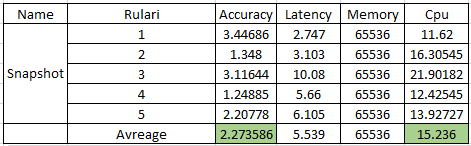


Pentru ferestrele de acest tip a fost important să vedem impactul pe care îl are dimensiunea ferestrei asupra rezultatelor. Putem observa cum odată cu creșterea dimensiunii ferestrei acuratețea devine din ce în ce mai precisă, dar consumul de resurse crește, la fel și latența. Putem observa că pentru aceste date dimensiunea optimală a ferestrei este de 3 cu un salt de o secundă. În tabelul următor vom putea vizualiza datele cu ajutorul cărora am creat graficele și am marcat cu verde valoarea minimă în comparație cu fereastra snapshot.



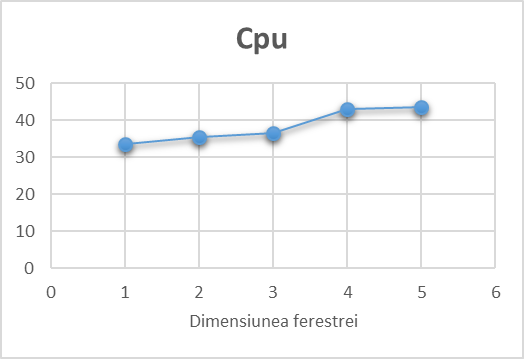
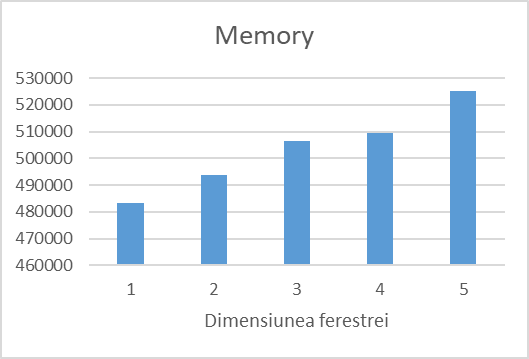
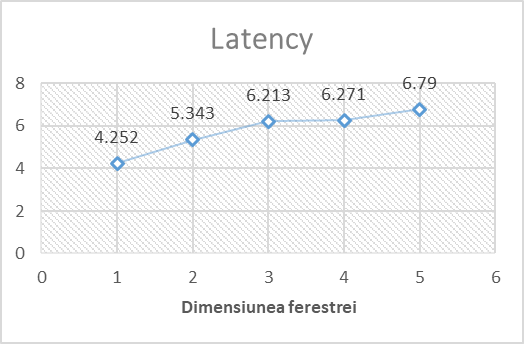
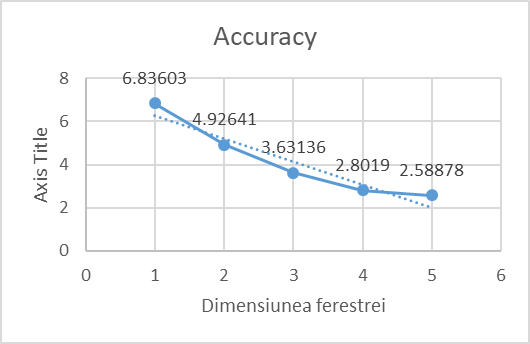
Ferestre Snapshot

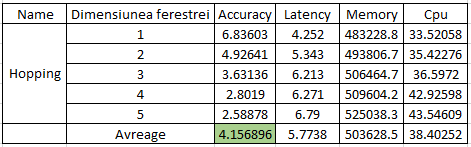


Pentru ferestrele de acest tip am dorit să vedem fluctuația de rezultate după numărul de rulări. Se poate observa că acuratețea maximă s-a atins la rularea a3a. Aici putem observa că consumul de resurse nu este direct proporțional cu acuratețea. Din tabelul următor putem observa că față de ferestrele hopping avem o acuratețe mai mare, dar și o utilizare CPU mai redusă. Totuși există o diferență semnificativă când vine vorba despre latența interogărilor. Am marcat cu verde valorile minime comparativ cu fereastra dinainte.

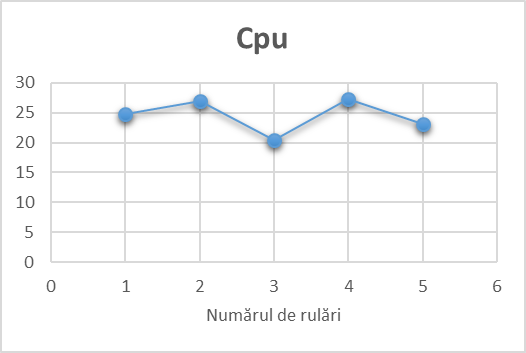
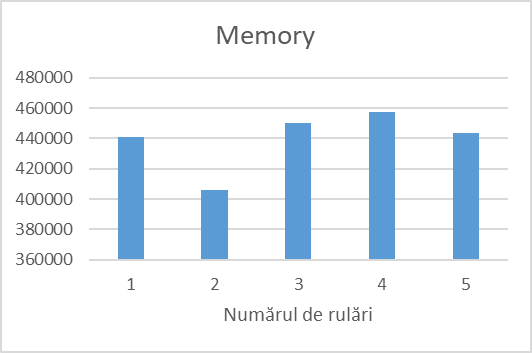
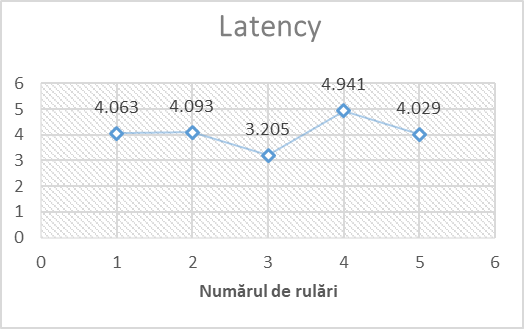
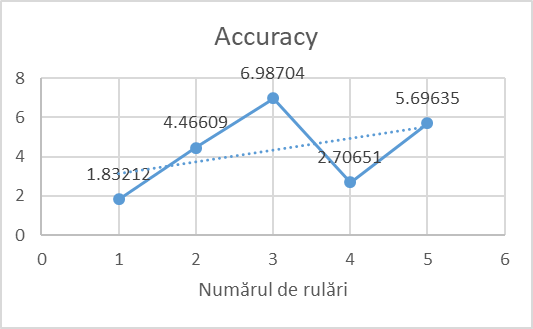
Pentru ultima categorie de interogări selecție-proiecție-join-agregări-grupări ajungem la experimentul descris inițial în care datele sunt grupate pe segmente de drum și apoi pe fiecare segment de drum se calculează media vitezelor mașinilor. Datorită faptului că aceste grupări trebuie salvate temporar pentru procesări continue și memoria va fi afectată.

Ferestre Hopping

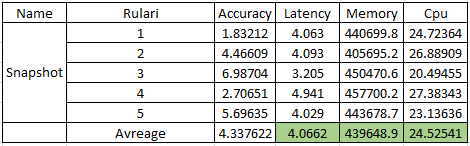


 După cum se poate observa din graficele anterioare, realizate pe baza ferestrelor de dimensiune de la 1-5 cu un hop de o secundă, odată cu creșterea dimensiunii ferestrei acuratețea se îmbunătățește, dar consumul de resurse crește si el. Ca și concluzie a acestui tip de fereastră este că dimensiunea optimală a ferestrei ar fi între 3 și 4 deoarece înainte de 3 acuratețea este mult prea mică, iar peste 4 consumul de resurse crește considerabil. În tabelul următor sunt prezentate date pe baza cărora au fost create graficele, am marcat cu verde valoarea minimă comparativ cu fereastra snapshot prezentată ulterior.

Ferestre snapshot



Acest tip de fereastră este analizat după rulările efectuate, se poate observa că acuratețea maximă obținută s-a regăsit la prima rulare, latența minimă si accesarea CPU a fost la a 3a rulare, iar memoria cea mai puțin accesată s-a realizat în cadrul celei de-a doua rulări. Față de ferestrele hopping folosesc mai puțină memorie și accesează mai puțin CPU, per total, deoarece ferestrele sunt dinamice și pentru fiecare interogare nu este necesar să țină in memoria temporală date pentru a le putea calcula, iar deoarece sunt mai puține grupuri pentru fiecare tip de fereastră, CPU este mai puțin accesat. Latența este mai bună ca cea de la ferestrele hopping, dar acuratețea este puțin mai mare.

 Putem observa în tabelul următor datele cu ajutorul cărora am creat diagramele. Am marcat și aici de asemenea cu verde valorile cele mai bune comparativ cu ferestrele hopping.

Ca ultim experiment ne-a rămas să vedem impactul dimensionării hopului asupra interogării. Am luat o interogare de ultimul tip, pe ferestre hopping de dimensiune 6, cu un hop de la 1-3 și pe un flux optim de 300 de date.

În concluzie, dimensiunea hop-ului este relevantă pentru rezultatul de acuratețe al fiecărei ferestre. Se poate observa că valoarea minimă se găsește atunci când dimensiunea hopului este cu o unitate mai mică decât dimensiunea ferestrei.

### 4.Comparatie cu studiul experimental Aurora

În acest studiu este prezentat sistemul de gestiune pentru fluxuri de date denumit Aurora. Fluxul de evenimente este generat de niște senzori la un anumit interval regulat sau neregulat, acesta fiind administrat de un administrator ceea ce se întâmplă și în aplicația mea, cu mențiunea că datele vin doar la un interval regulat de timp. Asupra acestui flux ambele aplicații suportă diferite tipuri de operatori și interogări continue asupra datelor din flux.

### 4.Oportunității viitoare

În primul rând pentru această aplicație experimentală as dezvolta o interfață grafică care să poată fi folosită la o scară largă. În această interfață se va face prezentarea relației dintre datele continue și datele salvate în baza de date. Apoi în una din ferestrele acestei interfețe să fie grafice care să arate care sunt străzile cele mai aglomerate din Cluj și la ce ore. Aplicația s-ar putea lega la hărțile de la Google pentru a putea avea o imagine vizuală asupra fiecărui tronson de drum.

În ceea ce privește fluxul de evenimente, aplicația va putea suporta mai multe fluxuri de date, care vor fi analizate pe thread-uri paralele pentru o eficiență sporită. Evenimentele vor deveni mai complexe sau vor veni fluxuri de evenimente diferite, unele cu privire la oameni, autobuze, pentru a putea calcula care stații sunt cele mai aglomerate. Fluxurile de date având dimensiuni mult mai mari decât acum, deoarece se vor simula date pentru o zi întreagă de trafic.

Interogările vor fi și din categoria selecție-grupare-join-union, pentru care se vor studia aproximările eficiente și consumul de resurse. Pentru interogările curente se va studia impactul dinamicității si diferențele între operatorii de agregare.

# Capitolul 5

## Concluzii

În această lucrare am prezentat experimentul Linear Road, minimizat pe orașul Cluj-Napoca asupra căruia am aplicat trei clase de interogări. Pentru început am făcut introducerea în domeniul aplicației, pentru a avea niște cunoștințe de bază pentru ca mai apoi să putem relaționa cu experimentele și cu ce se petrece în ele. Apoi au fost prezentate tehnologiile care au fost folosite în partea experimentală.

Este important pentru sistemele care gestionează fluxuri de date continue să fie pregătite la schimbările care pot surveni asupra fluxului, precum rata de apariție a datelor pe secundă sau diferitele caracteristici ale datelor. De aceea avem nevoie să știm care ar fi numărul de date optimale din flux pentru ca memoria CPU să fie utilizată la minim și latența să fie și ea de asemenea minimă. Am putut observa asta din primul tip de interogare în care fluxul de date optimal era de 300 date cu o rată de 10 evenimente pe secundă.

După rezultatele studiului am concluzionat faptul că interogările pe fluxurile continue aduc odată cu ele erori de aproximare a rezultatului corect. Procesarea datelor din fluxurile grupate necesită un acces de memorie mai mare, față de restul interogărilor. În momentul în care avem nevoie de foarte multe date de output este mai indicat să folosim ferestrele de tipul hopping, iar când ne dorim o acuratețe mai bună fără o procesare greoaie a datelor, dar cu puține rezultate de output, vom folosi ferestrele de tipul snapshot.

# 6.Bibliografie

1. [Jason H](https://github.com/JasonWHowell) *: Hopping Window (Azure Stream Analytics),* https://msdn.microsoft.com/en-us/azure/stream-analytics/reference/hopping-window-azure-stream-analytics, Ultima actualizare: 4/27/2018

2. Torsten Grabs, Roman Schindlauer, Ramkumar Krishnan, Jonathan Goldstein, and Rafael Fernández, *Introducing Microsoft StreamInsight*, Published: September 2009, Revised: May 2010

3. O. Etzion and P. Niblett: *Event Processing in Action*, Manning Publications, 2010.

4. N. Leavitt: *Complex-Event Processing Poised for Growth*, in Computer, vol. 42 , pp. 17-20, 2009.

5. Microsoft : *Using the StreamInsight Event Flow Debugger,*

<https://msdn.microsoft.com/en-us/library/ff518532(v=sql.111).aspx>

6.R. Tibbetts, M. Cherniack, and A. Arasu. Linear road: *A stream data management benchmark. Technical Report*, http://www.cs.brown.edu/research/aurora/linear-road.pdf, July 2003.

7.A. Arasu, S. Babu, and J. Widom. *The CQL Continuous Query Language: Semantic Foundations and Query Execution*. Technical report, Stanford University, Oct. 2003. <http://dbpubs.stanford.edu/pub/2003-67>.

8.B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: *Operator scheduling for memory minimization in data stream systems*. In Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of Data, pages 253–264, June 2003.

9.B. Babcock, M. Datar, and R. Motwani*: Load shedding for aggregation queries over data streams*. In Proc. of the 20th Intl. Conf. on Data Engineering, Mar. 2004.

10. Microsoft : *Microsoft StreamInsight*,

<https://msdn.microsoft.com/en-us/library/ee362541(v=sql.111).aspx>

11. Microsoft: *Overview (StreamInsight)* ,

<https://msdn.microsoft.com/en-us/library/ee391416(v=sql.111).aspx>

12.Microsoft : *Event Structure,*

<https://msdn.microsoft.com/en-us/library/ee378905(v=sql.111).aspx>

13. Microsoft: *Snapshot Windows*,

https://msdn.microsoft.com/en-us/library/ff518550(v=sql.111).aspx