MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

TWITRENDS – SISTEM DE PROCESARE A STREAM-URILOR ÎN TIMP REAL ÎN ERA BIG DATA

LUCRARE DE LICENȚĂ

Absolvent: **Andrei MOLDOVAN**

Coordonator științific: **asis. ing. Cosmina IVAN**

2016

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Andrei MOLDOVAN**

TWITRENDS – SISTEM DE PROCESARE A STREAM-URILOR ÎN TIMP REAL ÎN ERA BIG DATA

1. **Enunţul temei:** Proiectul își propune realizarea unui sistem de analiză big-data scalabil, pentru determinarea în timp real a celor mai discutate subiecte din rețeaua socială Twitter. Această clasificare se va realiza pe baza unor date de intrare reale, iar rezultatele obținute vor reflecta teme din lumea actuală, caracterizate de evenimente care au un impact la nivel global. De asemenea, proiectul va oferi și o clasificare la nivel de geolocație a temelor menționate.

2. **Conţinutul lucrării:** Cuprins, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare, Validare și Evaluare, Concluzii, Bibliografie, Anexe.

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**: asis. ing. Cosmina Ivan

5. **Data emiterii temei:** 1 noiembrie 2015

6. **Data predării:** _____

Absolvent: _____

Coordonator ştiinţific: _____

Cuprins

Capitolul 1. Introducere

În decursul ultimilor ani, atât volumul de date care necesită stocare și prelucrare, cât și sursele care furnizează aceste date au crescut în mod exponential. Datele provin din nenumărate surse: senzorii stochează informații referitoare la mediul lor, sisteme de calcul complexe generează fișiere de logare, tranzacțiile online reprezintă 98% din plățiile făcute la nivel global, iar oamenii distribuie informații prin intermediul rețelelor sociale din toate colțurile lumii.

Aceasta fenomen se datoreaza evolutiei tehnologice, companiilor precum Amazon, Google, dar in special retelele sociale precum Facebook și Twitter au contribuit la aceasta crestere. Datele devin tot mai complexe, ele circulă la o viteză tot mai mare, iar volumul lor crește exponențial. Fenomenul este cunoscut sub numele de “Big Data”, care astăzi reprezintă un subiect foarte popular în domeniul Științei Calculatoarelor.

1.1. Contextul proiectului și motivația

Datorită mărimii si a complexității acestor seturi de date, ele au devenit foarte greu, chiar imposibil de gestionat cu instrumentele clasice de procesare a datelor. Printre tool-urile actuale din domeniul “Big Data Analytics” se numară și Hadoop, fiind unul din cele mai folosite la momentul actual. În tot mai multe cazuri este nevoie de procesarea datelor în timp real, care presupune analiza acestora în momentul în care sunt interceptate, pentru a obține rezultate de actualitate. Obținerea rezultatelor aproape instantaneu este necesară pentru a putea reacționa și a lua decizii imediat după ce un eveniment a avut loc.

Viteza cu care datele sunt create și trebuie procesate poate deveni extrem de mare. Ca urmare, a fost necesară dezvoltarea a unor tehnologii noi de procesare și stocare a datelor pentru a face posibilă analiza unei asemenea informații complexe. Noi arhitecturi au fost dezvoltate și framework-uri noi au fost implementate pentru a depăși limitările atinse de către sistemelor tradiționale. Arhitecturile de procesare big-data și cele mai populare tool-uri pentru procesarea datelor în timp real vor fi descrise în cadrul lucrării.

Cele mai multe companii folosesc rețelele sociale pentru a-și promova afacerea și pentru a fi mai aproape de actualii și posibilii viitori clienți. Folosind rețele sociale precum Facebook, Twitter sau Google+, întreprindere postează și distribuie infomații, însă metricile puse la dispoziție de acestea eșuează în a reflecta detalii de o relevanță semnificativă în campaniile de marketing, obținerea a noi clienți sau a generării de venituri.

Din fericite, inseparabilitatea rețelelor sociale și big-data facilitează aplicarea unor noi strategii de marketing. Relevanța informației și domeniul de aplicabilitate al datelor permit crearea mai multor abordări predictive de analiză.

1.2. Conținutul lucrării

Lucrarea este alcătuită din 7 capitole, iar în cele ce urmează vor fi prezentate structura lucrării și conținutul acestor capitole:

Capitolul 1 – Introducere – realizează o plasare inițială în tema lucrării. În acest capitol este prezentat contextul proiectului prin descrierea evoluției complexității datelor care necesită procesare. De asemenea sunt descrie și conținutul și motivația lucrării.

Capitolul 2 – Obiectivele Proiectului – enumără pornind de la contextul proiectului și motivația prezentată în capitolul de introducere temele tratate de această lucrare. Obiectivul lucrării este de a prezenta o serie de noțiuni teoretice din domeniul big-data, a metodelor de procesare și a diferitelor framework-uri disponibile pe piață, urmată de implementarea unei aplicații concrete de analiză de date în timp real folosind framework-urile Apache Strom și Heron.

Capitolul 3 – Studiu Bibliografic – introduce conceptul de "Big Data" și îl devinește din mai multe puncte de vedere. De asemenea, acest capitol prezintă metodele de procesare batch și streaming folosite în analiza big-data și a arhitecturii lambda, care reprezintă o abordare hibridă propusă pentru depășirea limitărilor celor doua metode de analiză.

Capitolul 4 – Analiză și Fundamentare Teoretică – este centrat în jurul diferitelor framework-uri de procesare big-data în timp real. În acest capitol se realizează o descriere în detaliu a celor mai folosite tool-uri de analiză și prezintă modelul de procesare a acestora. Studiul acestor unelte/framework-uri a fost necesar în alegerea tehnologiei folosite la implementarea aplicației descrise în cadrul lucrării și pune în evidentă atât avantajele cât și limitările framework-ului Apache Strom, care sunt depășite o dată cu apariția sistemului de procesare Heron.

Capitolul 5 - Proiectare de Detaliu și Implementare – descrie în detaliu arhitectura și modelul de procesare al unei aplicații realizate în Apache Storm și migrată apoi pe framework-ul Heron de clasificare a mesajelor din rețeaua socială Twitter, intitulată TwiTrends. Clasificare acestor mesaje, numite tweets, presupune raportarea acestora la nivel de geolocație și determinarea celor mai discutate subiecte.

Capitolul 6 – Testare și Validare – descrie criteriile pe baza cărora a fost testată aplicația implementată. De asemenea, acest capitol validează rezultatele obținute prin raportarea datelor de ieșire a aplicației TwiTrends la cele mai importante evenimente care au avut loc la nivel global în momentul testării aceteia.

Capitolul 7 – Concluzii – prezintă un rezumat al contribuțiilor aduse și al rezultatelor obținute prin clasificarea celor mai discutate subiecte din rețeaua socială Twitter, obținute prin rularea sistemului TwiTrends. Capitolul propune și o serie de îmbunătățiri și extensii ale aplicației, enumerate în secțiunea de dezvoltări ulterioare.

Capitolul 2. Obiectivele Proiectului

În acest capitol vor fi prezentate obiectivele propuse spre a fi realizate. Lucrarea va realiza o descriere detaliată a conceptului de "Big Data", prin definirea acestuia din diferite perspective. De asemenea, lucrarea va prezenta diferitele modele de procesare și arhitecturi, urmată de o analiză a framework-urilor folosite la ora actuală. Pe baza acestor concepte și a relevanței datelor aflate în rețele sociale, atât pentru întreprinderi care se promovează prin intermediul acestora, cât și pentru pentru companii și personane individuale, lucrarea va descrie implementarea și funcționalitatea unui sistem de procesare în timp real intitulat TwiTrends.

2.1 Obiectivul principal

Scopul lucrării este de a descrie noțiunile de bază în ceea ce privește diferitele metode de analiză big-data. De asemenea, aceasta propune o soluție pentru determinarea celor mai discutate subiecte din rețeaua socială Twitter și clasificarea acestora la nivel de geolocație folosind framework-urile Apache Strom și Heron.

2.2 Obiective secundare

Pentru a obține rezultate relevante în urma analizei realizate, procesarea trebuie să fie una de timp real. Latența permisă într-un astfel de model de procesare este cel mult de ordinul secundelor.

De asemenea, sistemul implementat va folosi ca și date de intrare date reale din cadrul rețelei sociale Twitter. Accesul la acestea poate fi realizat pe bază de stream-uri, care facilitează execuția în timp real. Motivul pentru care datele de intrare sunt alese astfel este pentru a putea obține rezultate care reflectă realitatea. Validarea rezultatelor se va face considerând cele mai importante evenimente la nivel global care generează discuții în cadrul rețelelor sociale. Astfel, un obiectiv propus devine și obținerea unei clasificări a celor mai discutate subiecte care să reflecte realitatea, obținute pe baza tweet-urilor postate de utilizatori reali.

Un alt obiectiv este folosirea unei metode de clasificare prin care să se obțină rezultate cât mai precise, prin selectarea informației relevante din imensul volum de date produs de rețeaua socială Twitter. Metoda folosită pentru determinarea celor mai discutate subiecte, de asemenea cunoscute sub numele de trending topics, este una bazată pe asocierea unui hashtag acestui subiect, astfel încât procesarea întregului mesaj nu mai este necesară la nivelul de clasificare.

Implementarea trebuie să facă față creșterii exponențiale a volumului de date care necesită procesare. Așadar, scalabilitatea sistemului devine un obiectiv major în ceea ce privește implementarea sistemului descris în lucrare.

Pe baza acestor caracteristici propuse pentru implementarea aplicației, modelul de procesare va presupune următorii pași, prezentați și în figura 2.1:

- Citirea unor date reale
- Procesarea acestora in timp real
- Furnizarea datelor de ieșire în timp real

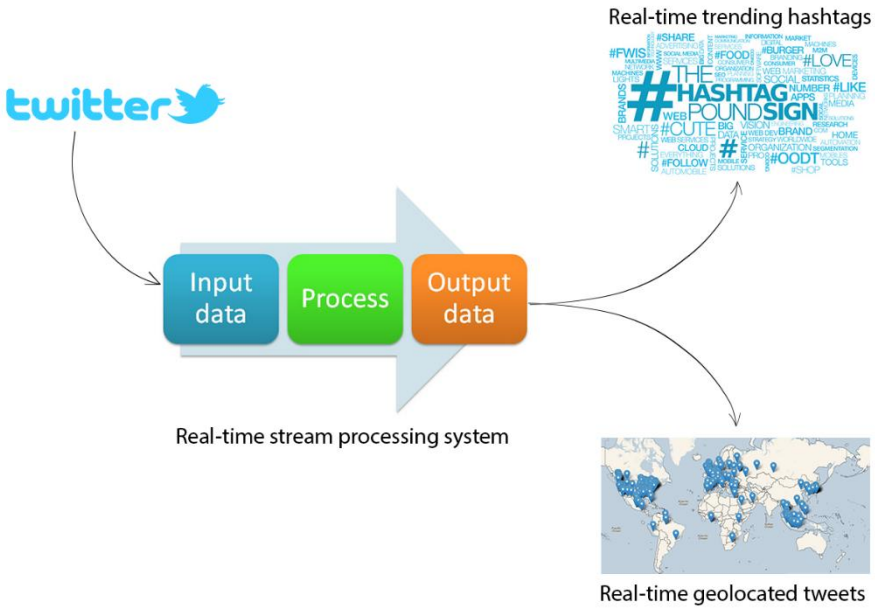


Figura 2.1: Obiectivele sistemului de analiză în timp real a tweet-urilor

Apache Storm și Heron reprezintă sisteme distribuite, open-source utilizate în analiză big-data în timp real, care sunt tolerante la eșec si garantează procesarea datelor, aceste propietăți ale framework-urilor devenind obiective în cadrul implementării aplicației. De asemenea, ele reprezintă sisteme scalabile, fiind concepute pentru rularea în cluster. Un cluster reprezintă o grupare independentă de servere care colaborează ca un sistem unitar în scopul de a oferi servicii de mare disponibilitate. Scalarea orizontală este obținută prin alocarea a mai multor noduri de execuție în cluster și implicit a mai multor resurse hardware.

Capitolul 3. Studiu Bibliografic

Documentarea bibliografică are ca obiectiv prezentarea stadiului actual al domeniului/sub-domeniului în care se situează tema. Conceptul de "Big Data" este unul relativ nou, devenit popular în ultimul deceniu, iar definirea acestuia este relativ complexă datorită proprietăților care caracterizează volumele imense de date aflate într-o continuă creștere. De asemenea, metodele tradiționale de procesare nu au putut fi scalate la cerințele impuse de către analiza big-data. Capitolul 3 va defini acest concept și va prezenta diferitele arhitecturi și modele de procesare folosite pentru a depăși limitele atinse de sisteme tradiționale.

3.1. Big Data

3.1.1. Definiția conceptului de Big Data

Ultimii 5 ani au văzut o creștere exponențială în ceea ce privește interesul pentru domeniul cunoscut sub numele de "Big Data". Folosind Google Trends, un tool bazat pe motorul de căutare Google, care analizează cât de des apare un cuvânt-cheie raportat la numărul total de căutări, se poate observa creșterea frecvenței de căutări ale acestui termen, reprezentate in Figura 3.1.

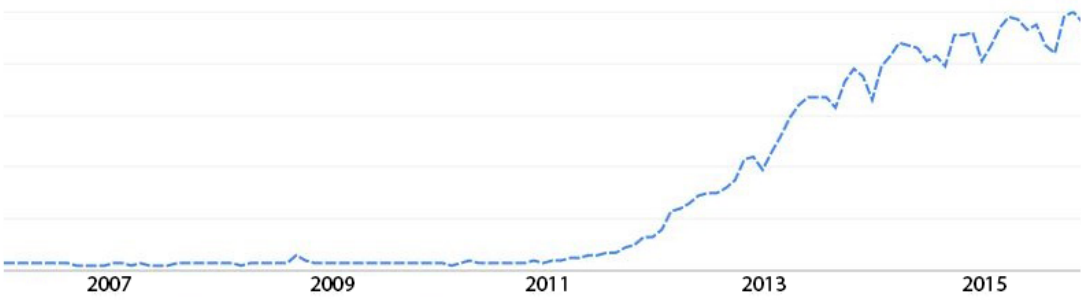


Figura 3.1: Căutări pe Google a termenului "Big Data" [1]

În ciuda acestei creșteri de interes nu s-a impus încă o definiție unanim acceptată pentru acest termen. Există divergențe în special asupra primei jumătăți a termenului, "big". Conform MIT Technology Review (2013)[2]: "un set de date definit ca mare astăzi va fi, cu o mare certitudine, considerat mic in viitorul apropiat". Mărimea seturilor de date este deci adeseori raportată la tehnologia existentă în prezent pentru procesarea acestora, termenul de "big" fiind

[1] https://www.google.com/trends/explore#q=Big%20Data
[2] https://www.technologyreview.com/s/519851/the-big-data-conundrum-how-to-define-it/

definit ca mai mult decât poate fi procesat folosind tehnici convenționale. Din punct de vedere al volumului de date, creșterea din ultimul deceniu este una exponențiala, iar prognozele pentru următorii ani arata ca acest trend va continua, precum sugerează și Figura 3.2.

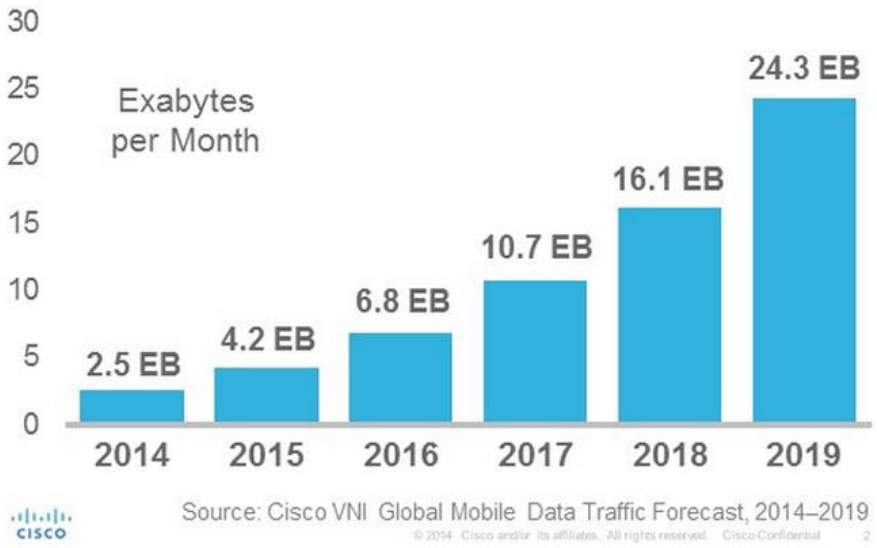


Figura 3.2 Prognoza creșterii traficului de date 2014-2019[3]

În lipsa unei definiții unice, marile companii care au contribuit la fenomenul Big Data folosesc o serie de definiții proprii [1]:

Oracle: Big Data este o derivare semnificativă de la business-ul tradițional bazat pe baze de date relaționale, augmentat cu surse noi de date nestructurate.

Intel: Oportunități Big Data apar în organizații care generează un volum de date mediu de 300 terabytes pe săptămână.

Microsoft: Big Data este un termen folosit tot mai des pentru a descrie aplicarea puterii mari de procesare – adesea întâlnite în domenii precum machine learning și inteligență artificială – pe seturi de informații masive și adesea foarte complexe.

[3] http://www.convergedigest.com/2015/02/cisco-2019-mobile-data-traffic-forecast.html

Cu toate aceste definiții diferite, cel mai popular mod de a caracteriza "Big Data" este pe baza celor 3 V-uri ale big-data, conform sugestiei publicate de Meta [13] (în prezent cunoscuți sub numele de Gartner) în anul 2001: Volum, Varietate, Viteza.

3.1.2. Cele 3 V-uri din Big Data

Când vine vorba de Big Data, termenul de "big" nu se refera doar la mărimea volumului de informație. Pe lângă dimensiunea datelor, viteza și varietatea acestora sunt de asemenea caracteristici ce trebuie luate în considerare. Astfel, volumul, viteza și varietatea sunt numite "The 3-Vs of Big Data".

Volumul: se referă la ordinul de mărime al datelor. Acesta a crescut exponențial pe parcursul anilor, iar acest trend continuă, de la terabytes la petabytes, de la peta la exa și așa mai departe.

Viteza: descrie frecvența cu care datele sunt generate și recepționate. Datorită evoluției tehnologiilor, companiile și consumatorii generează un volum mai mare de date într-un interval de timp mai scurt [2]. Diferite tipuri de Big Data ar putea necesita o procesare la viteze diferite, conform următoarei clasificări [3]:

- **Batch**: Datele sunt stocate (adunate) și procesate la anumite intervale de timp. Mai multe detalii vor fi prezentate în capitolul 3.2.

- **Near-Time**: Intervalul de timp dintre doua procesări consecutive este unul foarte mic, apropiat de procesarea în timp real. În procesarea near-time, datele sunt colectate instantaneu, însă procesarea poate avea loc periodic, la anumite intervale de timp: de la o dată pe zi, la o data la câteva ore sau chiar câteva minute.

- **Real-Time/Streaming**: procesarea este una continuă. Aceasta metodă va fi de asemenea prezentată în detaliu în capitolul 3.2.

Varietatea: este una din cele mai importante caracteristici. Diferite surse de big-data generează diferite forme de date. Pe măsură ce sunt dezvoltate aplicații noi, apar și formate de date noi. O dată cu creșterea numărului acestor formate, proiectarea algoritmilor pentru minare și analizare devine o provocare tot mai mare [3].

În anul 2012, un al 4-lea V a fost adăugat acestui mod de caracterizare, și anume veridicitatea. Veridicitatea se referă la fiabilitatea, precizia, acuratețea, inteligibilitatea și gradul de încredere al datelor [3]. Ea se aplică atât la datele de intrare, cât și în contextul datelor de ieșire obținute în urma procesării acestora [1].

3.2. Metode de procesare Big Data

3.2.1. Introducere

O dată cu începerea erei "Big Data" și cu creșterea exponențială atât a volumului de date creat, care necesită procesare și analiză, cât și a varietății datelor și a surselor din care acestea provin, metodele tradiționale de procesare nu au mai făcut față. Pentru tot ce înseamnă procesare, de la captarea datelor, prelucrarea și analiza acestora, la generarea - respectiv stocarea - datelor de ieșire, a fost necesar un progres din punct de vedere tehnologic. Acest progres a dus la nașterea unor tehnologii, tool-uri și platforme noi, cu ajutorul cărora procesarea big-data a devenit posibila. Aceste tehnologii și tehnici de procesare trebuie să ofere aplicabilitate într-o varietate de domenii, de la informatică, matematică, fizică, la probleme economice, de statistică, de rețelistică și aplicații sociale, iar lista ar putea continua pe măsură ce noi experți descoperă utilitatea acestui tip de procesare [4].

Una din cele mai importante caracteristici ale instrumentelor folosite pentru analiza acestor date este metoda de procesare pe care se bazează. Aceasta lucrare va descrie în continuare cele mai populare două metode aplicate în momentul de fată, și anume procesarea în manieră batch, prezentată în capitolul 3.2.2 și cea în timp real, cunoscută și sub numele de procesare de tip streaming, descrisă în capitolul 3.2.3. De asemenea trebuie menționat și un al treilea tip de metodă, și anume cea interactivă. Analiza interactivă aplică un set de tehnici de combinare a puterii computaționale cu capabilitățiile percepvite și cognitive ale omului, permițând utilizatorului să efectueze propria analiză a informațiilor [4]. Utilizatorul este deci conectat în mod direct la sistemul de procesare, cu care poate interacționa în timp real. Datele pot fi vizualizate, comparate și analizate în format tabelar sau grafic [4].

3.2.2. Procesarea batch folosind modelul MapReduce

Majoritatea sistemelor de procesare în manieră batch se bazează paradigma de programare cunoscută sub numele de MapReduce. Aceasta paradigmă a fost introdusă de Google pentru a procesa volume imense de date în clustere de dimensiuni mari [3] și implementată în infrastructura Apache Hadoop [4], pe care se bazează majoritatea tehnologiilor de procesare batch.

MapReduce reprezintă un model de programare destinat procesării de date pe un număr foarte mare de noduri, reprezentate de mașini disponibile în comerț, fără performanțe deosebite. MapReduce se bazează pe împărțirea procesării în două etape: Map și Reduce, fiecare primind ca date de intrare o pereche cheie-valoare, al cărei tip poate fi stabilit de programator și întorcând ca rezultat tot o pereche cheie-valoare[4]. Framework-ul rulează pe un cluster, cu scopul de a mina

[4] https://static.dzone.com/dz1/dz-files/refcardz/rc117-010d-hadoop_0.pdf

seturi de date de dimensiuni foarte mari. După cum poate fi observat în figura 3.3, întâi se aplica funcția Map tuturor membrilor unui set de date care eturnează o listă cu rezultate, iar apoi funcția Reduce colectează și rezolvă rezultatele din una sau mai multe operații de mapare executate in paralel. În urma executării repetate a acestor doua funcții, setul de date de intrare este redus semnificativ, astfel încât la ieșire se va produce un subset al acestuia, reprezentând informația relevantă extrasă în urma analizei.

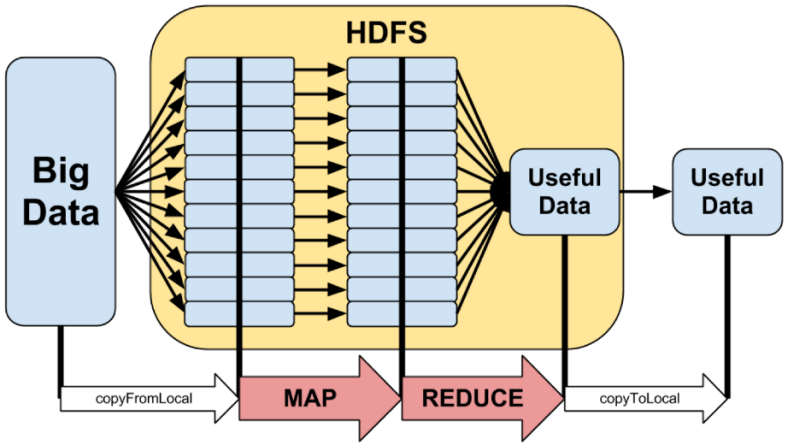


Figura 3.3: Modelul Map-Reduce[5]

Procesarea batch, la baza căruia stă modelul Map-Reduce prezentat anterior, presupune trei etape: colectarea datelor într-un batch, procesarea batch-ului și stocarea datelor de ieșire. Colectarea datelor presupune citirea unui set de date de dimensiunea batch-ului. În momentul în care batch-ul este plin (sau atunci când execuția este forțată de către un planificator), datele sunt submise pentru procesare. Astfel se obține un model de execuție “framed”, fie din punct de vedere al timpului (la anumite intervale impuse), fie din punct de vedere al volumului de date (impus de dimensiunea batch-ului).

Hadoop este un framework foarte bun pentru ceea ce a fost conceput: procesarea unor seturi imense de date. Cu toate acestea, modelul are câteva limitări. În primul rând, modelul de date este oarecum restrictiv, nu este mereu ușor de a transpune o problema într-un set de perechi cheie-valoare. Mai mult, Hadoop poate procesa doar seturi de date gata recepționate, ceea ce înseamnă că MapReduce poate procesa date doar în maniera batch. În funcție de dimensiunea blocurilor de date care trebuie procesate și de puterea de calcul a sistemului, datele de ieșire pot fi produse cu întârzieri relativ mari, precum evidențiat în figura 3.4 [2]. Aceste limitări au dus la dezvoltarea unor noi metode de procesare precum cea în timp real sau de tip streaming.

[5] http://www.glennklockwood.com/data-intensive/hadoop/mapreduce-workflow.png

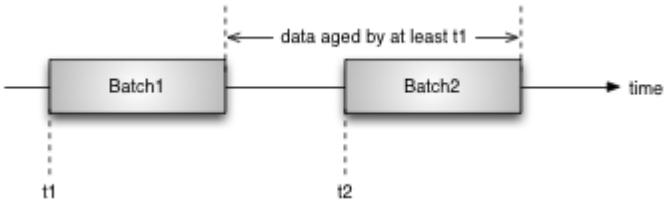


Figura 3.4: Limitarea modelului de procesare batch [2]

3.2.3. Procesarea de tip streaming

Dezavantajul major al modelului Map-Reduce și al framework-ului Apache Hadoop, cel mai răspândite în ceea ce privește procesarea de tip batch – și anume faptul că datele care sunt obținute la ieșire ca și rezultate ale analizei big data apar cu o întârziere relativ mare, raportat la momentul de timp când acestea au fost captate – a inspirat apariția unor modele noi. Ca alternativă, procesarea de tip streaming, de asemenea cunoscută sub numele de procesare în timp real, presupune o procesare continuă a datelor de intrare. Spre deosebire de modelul batch (3.2.2) bazat pe paradigma Map-Reduce, care presupune un timp relativ mare de colectare a datelor urmat apoi de procesarea propriu-zisă, procesarea în timp real are loc imediat după interceptarea datelor de intrare. Drept urmare, rezultatele sunt produse aproape instantaneu și pot fi vizualizate în timp real. Astfel, în contrast cu modelul batch, în care un volum de date mai mari este procesat într-un timp mult mai îndelungat, modelul de tip streaming se bazează pe o procesare la viteză mai mare a unui volum redus de date (imediat ce sunt submise pentru procesare), după cum ilustrează figura 3.5.

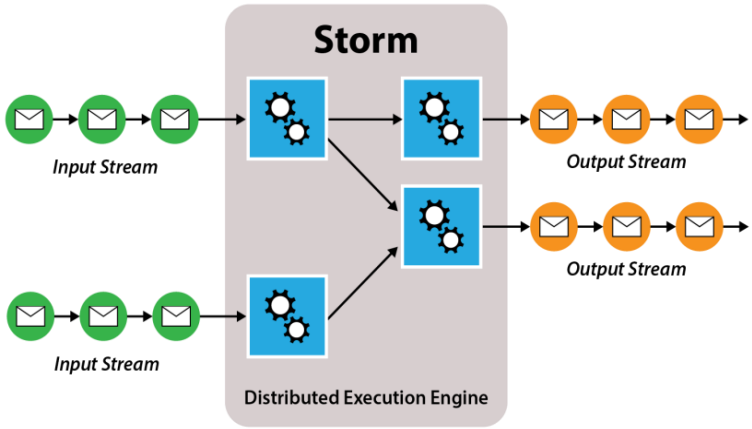


Figura 3.5: Procesare de tip streaming[6]

[6] http://www.business-software.com/wp-content/uploads/2014/09/Storm.png

Chiar dacă procesarea de tip streaming rezolvă problema latenței mari, apare în schimb problema definirii termenului „real time" în contextul Big Data. Apare deci întrebarea ce se înțelege prin real-time (timp real)? În acest context, real-time poate fi analizat din două puncte de vedere: din punct de vedere al datelor sau din punct de vedere al utilizatorului final [2].

Din perspectiva datelor, termenul de timp real se referă la procesarea datelor imediat ce sunt recepționate. Așadar rezultatele obținute în urma analizei vor fi mereu actuale, latența procesării fiind foarte mica. Presupunând ca o tuplă este recepționată la momentul de timp t1, la momentul de timp t2 ea va fi gata procesată, iar rezultatul în urma analizei tuplei este disponibil utilizatorului după o durată de timp t2 – t1. Din acest punct de vedere, întrebarea se transformă deci în cât de mic trebuie să fie acest interval de timp t2 – t1 pentru a putea considera că procesarea are într-adevăr loc in timp real. În funcție de domeniul în care acest termen este folosit, ordinul de mărime este diferit. În cazul unui sistem hardware, noțiunea de timp real este definită la nivel de nanosecunde – milisecunde, pe când în procesarea unor stream-uri de tweet-uri, ordinul este de nivelul secundelor, acceptându-se o întârziere de câteva secunde [2].

Din perspectiva utilizatorului final, definirea noțiunii de big data se face în funcție de timpul necesar sistemului pentru a răspunde unei interogări, astfel încât utilizatorul poate vizualiza răspunsul la request-ul trimis. Din nou, intervalul t2 – t1 trebuie să fie cât mai mic, astfel încât latența să fie minimă. Din acest punct de vedere noțiunea de timp real poate fi comparată cu un apel de serviciu REST sau orice alt apel de tip RPC, spre deosebire de procesarea batch, când rezultatele sunt disponibile unei interogări după câteva minute sau chiar câteva ore [5].

3.2.4. Arhitectura Lambda

"Arhitectura Lambda... oferă o abordare generică pentru implementarea unei funcții arbitrare, aplicată pe un set de date arbitrar, astfel încât funcția va returna rezultatul ei cu o latență minimă" [6]

Modelul arhitectural Lambda prezentat în figura 3.5 presupune integrarea a doua nivele de procesare: un nivel de procesare rapidă, bazată pe stream-uri (3.2.3) și un nivel de procesare masivă, folosind modelul batch (3.2.2). Astfel, se păstrează avantajul procesării batch de a procesa un volum mare de date deodată, iar dezavantajul latenței introduse de acesta este rezolvat de layer-ul de viteză care prelucrează datele în timp real.

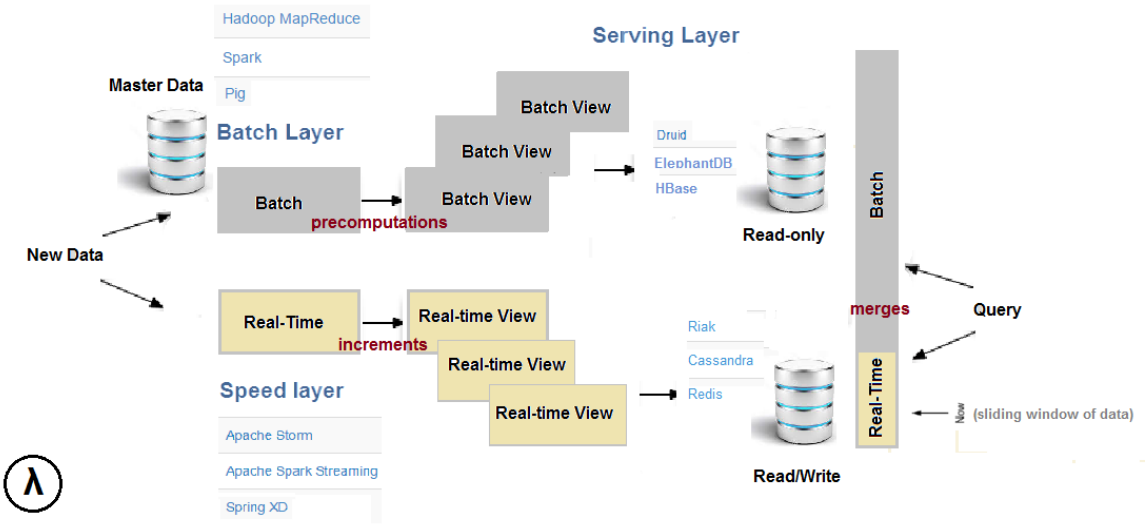


Figura 3.6: Arhitectura Lambda[7]

Arhitectura Lambda cuprinde trei nivele: nivelul batch, nivelul de viteza (timp real/streaming) și nivelul de servire. Primele două nivele realizează procesarea datelor folosind cele două metode menționate, care dau și numele layer-elor. Datele de intrare sunt submise atât pentru o procesare batch cât și pentru una de tip streaming. Spre deosebire de layer-ul de viteză, în care datele sunt păstrate doar pentru o perioadă de timp, nivelul de procesare batch salvează datele în manieră istorică, sau, altfel spus, toate datele procesate sunt salvate persistent. În continuare vor fi descrise aceste trei nivele mai în detaliu:

1) **Nivelul de procesare batch**[8]: administrează datele de tip master și calculează in prealabil vederile batch. El poate fi văzut ca o arhivă istorică a tuturor datelor colectate. O implementare posibilă este una folosind Apache Hadoop, bazat pe modelul Map-Reduce, însă acest nivel poate fi implementat și folosind un sistem sau framework de tip OLAP (online analytical processing) precum Vertica sau Netezza. După ce un batch de date este procesat, iar rezultatele analizei de big-data sunt disponibile, vederile batch sunt actualizate cu aceste date, astfel încât ele devin disponibile pentru interogare.

[7] https://tsicilian.files.wordpress.com/2015/01/lambda1.png
[8] https://dzone.com/articles/lambda-architecture-big-data

2) **Nivelul de viteză**: realizează o analiză în timp real a unui subset din totalul de date colectate. Acest subset de date reprezintă datele cele mai actuale, de obicei specificate printr-o fereastră de timp (de exemplu 10 minute). Acest nivel este folosit pentru a oferi un acces imediat la rezultatele cele mai actuale. La fel ca și layer-ul de batch, cel de viteză își actualizează propriile vederi de timp real. Actualizarea este una continuă, iar rezultatele pot fi obținute prin interogărarea acestor vederi.

3) **Nivelul de servire**: reprezintă un mecanism de caching a rezultatelor obținute în urma analizei efectuate de către layer-ul de batch. El aduce o îmbunătățire în ceea ce privește performanța interogărilor la nivelul întregului volumul de date. Periodic, datele din layer-ul de servire sunt recalculate, iar rezultatele salvate în memoria cache sunt reactualizate în acest layer. Pentru nivelul de viteză nu este necesar un asemenea mecanism de caching, deoarece acesta îl include. Având in vedere că analiza de date în timp real are loc pe date actuale, datele stocate în acest nivel sunt salvate temporar, precum într-o memorie cache.

Prin urmare, Lambda definește o arhitectură big-data care permite interogări predefinite și arbitrare, precum și computații atât pe date care circulă la o viteză mare, cât și pe un model de date istorice[9]. Acest model computațional reprezintă o abordare hibridă, fiind o combinație între modelul batch (3.2.2) și cel de timp real (3.2.3). El cuprinde avantajele modelului Map-Reduce pentru procesarea simultană a unui volum mare de date, iar latența introdusă de acesta este rezolvată prin nivelul de procesare în timp real. Cu toate acestea, există și un compromis semnificativ care are loc în această arhitectură legat de complexitatea acesteia. Procesările de tip batch și streaming sunt în sine modele complexe, iar aplicarea lor în paralel și introducerea adițională a unui al treilea nivel sporesc complexitatea arhitecturii. Pasarea mesajelor introduce un timp de execuție suplimentar, iar din punct de vedere al infrastructurii numărul minim de noduri într-un cluster necesar rulării crește subsanțial.

3.2.5. Analiză comparativă

În urma prezentării celor doua metode alternative de procesare big data, cât și a modelului hibrid, putem concluziona că fiecare aduce atât avantaje cât și dezavantaje. Modelul batch permite analiza unui volum mare de date simultan, însă introduce latență mare în ceea ce privește obținerea rezultatelor. Procesarea de tip streaming rezolvă această problemă, însă datele sunt disponibile doar pe o anumită fereastră de timp, cele istorice pierzându-se. Arhitectura Lambda oferă o soluție pentru a trece peste aceste dezavantaje, însă complexitatea semnificativă

[9] http://radar.oreilly.com/2015/02/improving-on-the-lambda-architecture-for-streaming-analysis.html

a modelului poate deveni o problemă, deoarece presupune implementarea simultană a două metode de procesare diferite care necesită deja un număr mare de resurse hardware.

Alegerea modelului corespunzător se realizează în funcție de contextul în care se plasează problema. Pentru aplicații în care latența nu reprezintă o problemă, iar de exemplu rezultatele analizei pot fi obținute zilnic, la sfârșitul zilei, procesarea batch este cea mai adecvată. Dacă utilizatorii au însă nevoie de rezultate imediate – o situație foarte des întâlnită în aplicații sociale pentru sugerarea unor teme de actualitate – procesarea în timp real este opțiunea corectă. Pentru sisteme complexe, în care combinarea celor două metode este una necesară, arhitectura Lambda oferă soluția corespunzătoare.

Capitolul 4. Analiză şi Fundamentare Teoretică

Lucrarea prezentă are ca focus analiza de streaming, deocarece modelul de procesare batch nu este adevat pentru implementarea sistemului propus, un obiectiv major reprezentând procesarea datelor și obținerea rezultatelor în timp real. De asemenea, complexitatea arhitecturii lamba nu este necesară in această abordare, deoarece layer-ul de procesare batch nu aduce beneficii semnificative. Prin urmare, în acest capitol vor fi prezentate în detaliu o serie de tehnologii și framework-uri de procesare big-data sub formă de stream-uri, dezvoltate de diferite companii pentru a obține o analiză real-time.

4.1. Framework-uri de Stream Processing

Motivul care a dus la dezvoltarea unui număr relativ mare de framework-uri pentru prelucrare de tip streaming, a fost nevoia de a trece peste limitările impuse de către analiza batch, și anume latența mare pe care aceasta o introduce. Procesarea în timp real presupune procesarea unui flow continuu de date, infinit, astfel încât rezultatele obținute să fie disponibile cu o latență minimă, de ordinul a câtorva secunde, și să fie direct accesibile de către utilizatorul final. Prin urmare, volumul de date acumulat pentru analiză la un moment de timp nu este foarte mare, însă viteza pentru Big Data (3.1.2) este considerabilă.

Printre sistemele care folosesc procesarea de tip streaming se numără Apache Storm, Heron, Yahoo S4, Splunk, Samza, Spark Streaming și alte platforme precum SQLStream, Kafka sau Apache SAMOA. Apache Storm și Heron vor fi prezentate în detaliu în capitolul 4.2 și 4.3 deoarece au fost alese ca și framework-uri pentru implementare.

În 2010 cei de la Yahoo! au introdus **S4**, o platformă care permite computații distribuite, printr-un sistem tolerant la eșec și scalabil, scris folosind limbajul de programare Java. S4 a fost proiectat în scopul procesării stream-urilor de date la o scală largă. Sistemul a fost folosit în producție de cei de la Yahoo! pentru a procesa mii de interogări de căutare, dar a dovedit performantă bună și în alte aplicații. [4]

Splunk este o platforma folosită pentru analizarea în timp real a stream-urilor de date produse de alte mașini de calcul, precum servere. Companii precum Amazon, Heroku sau Sentihun au folosit Splunk ca o platforma inteligentă big-data pentru a monitoriza și analiza fișiere log și alte date produse de servere, folosind o interfață web de asemenea pusă la dispoziție de către Splunk [3]. Splunk poate fi folosit atât pentru procesarea fișierelor structurate cât și pentru cele nestructurate.

Abordarea în analiza de streamuri introdusă de către **Apache Samza** se bazează pe procesarea mesajelor în momentul în care sunt recepționate, unul câte unul. Primitiva pe care se bazează Samza nu este tupla (concept prezentat în capitolul 4.2.2), precum în cazul Framework-ului Apache Storm sau Heron, ci mesajul. Streamurile sunt împărțite în partiții, iar fiecare partiție

este organizată ca o secvență ordonată de mesaje read-only, fiecare mesaj având un identificator (offset) unic.

Spark Streaming, o extensie a Framework-ului de bază Spark care oferă un API de nivel înalt în Java, Scala, Python și R pentru procesarea de Big Data[10], este un sistem rapid și general-purpose de procesare Big Data în cluster, care oferă suport pentru procesarea stream-urilor continue in Spark [8]. Procesarea de tip streaming reprezintă un model "one-at-a-time": data este procesată imediat ce este citită. Cu toate acestea, modelul de procesare aplicat de Spark Streaming este un model micro-batch, un caz particular de procesare batch care presupune o dimensiune foarte redusă a batch-ului, astfel încât computațiile să fie executate în timp near-time sau foarte aproape de real-time.

Primitiva de bază folosită de Spark se numește RDD (Resilient Distributed Datasets). Un RDD este un set de elemente read-only distribuit, asupra căruia pot fi aplicate o serie de transformări în paralel de către un set de funcții arbitrare[11]. Spark Streaming folosește ca și primitivă de bază tot RRD-uri, pe baza cărora realizează abstractizarea unui stream continuu de date, transformându-l într-un stream discretizat, de asemenea cunoscut sub termenul de DStream. Acest DStream reprezintă un micro-batch de RDD-uri și este creat dintr-o sursă de date de intrare, cum ar fi Apache Kafka sau poate fi rezultatul transformării unui alt DStream în urma aplicării unei serii de operație pe cel inițial. Modelul este prezentat în figura 4.1.

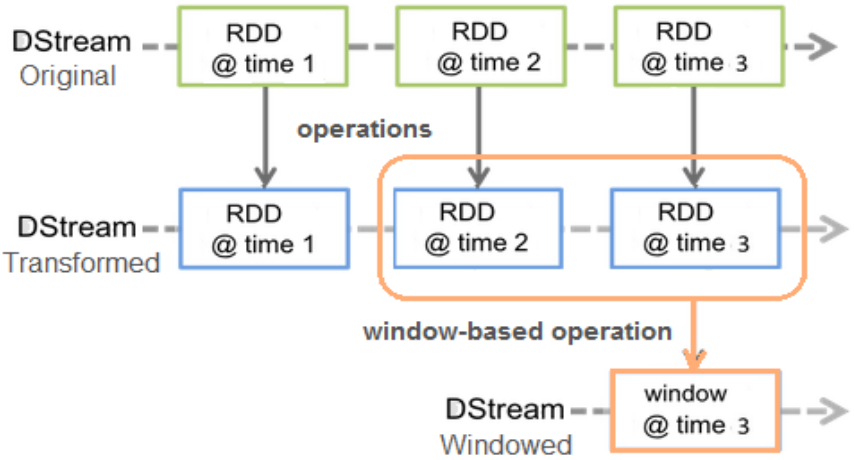


Figura 4.1: Aplicarea conceptului de DStream în Spark Streaming[12]

[10] http://spark.apache.org/docs/latest/
[11] https://dzone.com/articles/streaming-big-data-storm-spark
[12] https://tsicilian.files.wordpress.com/2015/02/spark-architecture4.png

4.2. Apache Storm

O dată cu apariția framework-urilor Spark și Storm a apărut și o concurență mare între acestea în ceea ce înseamnă aplicarea lor în industrie. Pentru a cunoaște avantajele fiecăreia, trebuie înțelese diferențele dintre aceste două tehnologii, atât în ceea ce privește modelul de procesare (Spark utilizează abstractizarea unui stream ca și DStream, iar Storm procesarea directă a streamului în timp real), cât și al garantării procesării (cel puțin, cel mult sau exact o dată). Alegerea framework-ului adecvat depinde cel mai des de domeniul de aplicare ale acestuia: pe când Spark prezintă un model excelent pentru realizarea unor analize interactive și machine learning iterativ, Apache Storm excelează în ceea ce înseamnă analiza datelor real-time, procesarea limbajului natural și normalizarea datelor[13].

4.2.1. Descriere generala

Apache Storm este un sistem distribuit, open-source de analiză big-data în timp real, care este tolerant la eșec și garantează procesarea datelor. Dezvoltat inițial de către BackType, proiectul a fost introdus sub licență Apache în data de 17 septembrie 2011 ca un sistem care "realizează pentru procesarea în timp real ceea ce Hadoop realizează pentru procesarea batch"[14].

Storm include o serie de feature-uri precum scalabilitate orizontală, toleranță la eșec, garantarea procesării datelor și suport pentru diferite limbaje de programare. **Scalabilitatea** framework-ului include posibilitatea de a rebalansa clusterul atunci când un nod de procesare nou este adăugat. Fiind un framework scris în limbajele Java și Clojure, pe lângă acestea Storm suportă toate limbajele de programare care pot citi și scrie în standard I/O. Pentru a oferi un sistem de pasare de mesaje **tolerant la eșec**, Storm ține evidența fiecărei înregistrări. [9]

De asemenea, **garantarea procesării** este asigurată prin trei semantici diferite: "cel puțin o dată", "cel mult o dată" și "exact o dată". Semantica implicită de procesare este cea care asigură că un mesaj va fi prelucrat "cel puțin o dată", însă framework-ul permite configurarea acestor semantici pentru a include si celelalte două strategii alternative, semnificația celor trei fiind:

- "Cel mult o dată" (At most once): există posibilitatea pierderi unui mesaj, iar recuperarea acestuia nu poate avea loc.

- "Cel puțin o dată" (At least once): prelucrarea oricărui mesaj este garantată, însă există posibilitatea de retransmitere a acestuia.

[13] http://zdatainc.com/2014/09/apache-storm-apache-spark/
[14] http://storm.apache.org/index.html

- "Exact o dată" (Exactly once): livrarea perfectă a mesajelor, un mesaj nu va fi pierdut sau retransmis niciodată.

Cu toate aceste caracteristici, una din cele mai importante este posibilitatea execuției în timp real [9].

4.2.2. Concepte de bază în Apache Storm

Programarea în Storm presupune proiectarea unui graf de calcul în timp real, numit topologie, și deploy-ul acestuia într-un cluster, unde un nod te tip master va distribui codul pe nodurile acestuia pentru execuție. Elementele unei topologii cuprind noduri de tip spout, care citesc și emit stream-uri și noduri de tip bolts, unități de procesarea a datelor. Arcele care leagă aceste două tipuri de noduri reprezintă o serie de tuple, care sunt emise sub forma unui stream continuu de date.

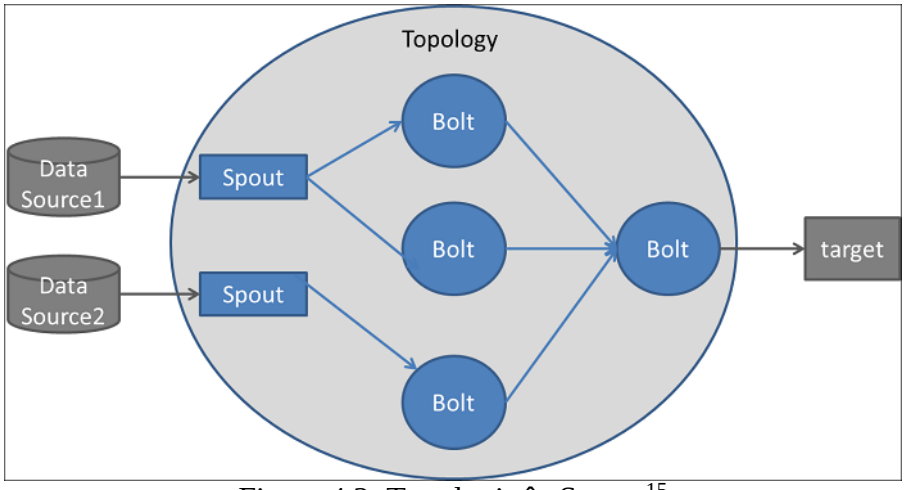


Figura 4.2: Topologia în Storm [15]

Fiecare nod dintr-o topologie (spout sau bolt) poate fi executat pe mai multe instanțe, în paralel. Executarea în paralel a unui bolt, de exemplu, are loc prin replicarea acestui nod în cluster și distribuirea datelor citite dintr-un stream sau dintr-un set de stream-uri la aceste instanțe. Regula de distribuire a datelor către instanțele multiple ale aceluiași nod poate fi specificată explicit. Detalii referitoare la diferitele metode de grupare ale stream-urilor și de

[15] https://www.accenture.com/t20150521T024358__w__/us-en/_acnmedia/Accenture/Conversion-Assets/Blogs/Images/1/Accenture-Storm-Topology.png?la=en

distribuire a valorilor către multiple instanțe ale aceluiași nod vor fi prezentate în capitolul 5, pe baza implementării realizate.

Pentru o înțelegere mai bună a terminologiei folosite în Apache Storm, se va realiza o descriere a elementelor de bază, adesea numite și componente, care sunt folosite în crearea unei aplicații, cunoscute sub numele de topologie:

- **Topologie**: Reprezintă abstractizarea top-level care este folosită pentru a descrie workflow-ul unei aplicații în Strom. O topologie descrie un graf orientat aciclic (DAG), care reprezintă structura și logica execuției a aplicației de procesare real-time. Fiecare nod din acest DAG procesează și înaintează tuple în paralel. O topologie cuprinde ca și elemente spouts, bolts și streamurile de date. Muchiile din graf reprezintă streamurile de tuple, iar nodurile reprezintă spouts și bolts. De asemenea, muchiile indică care bolt subscrie cărui stream de date.

- **Tuplă**: o structură de date identificabile care conține o listă ordonată de valori. Un element (valoare) dintr-o tuplă poate fi un obiect de orice tip.

- **Stream**: este abstractizarea de bază din Storm. Un stream reprezintă o secvență infinită de tuple, care sunt procesate și create in paralel. Storm oferă funcții primitive pentru transformarea unui stream de intrare într-un alt stream de ieșire într-o manieră distribuită și fiabilă. Fiecărui stream îi este atribuit un identificator unic în momentul declarării, fie cel implicit, fie unul specificat în momentul creării sale. Astfel devine posibilă diferențierea proveniență tuplelor în cazul unui bolt care citește mai multe stream-uri.

- **Bolt**: În Strom, procesarea propriu zisă a unui task este executată de către un bolt. Un bolt citește tuplele a unui sau a mai multor stream-uri, le procesează și poate emite unul sau mai multe stream-uri noi tuple, reprezentând rezultatul procesării. Este de menționat faptul că un astfel de element poate, dar nu trebuie neapărat să emită un stream nou. De exemplu, un bolt care realizează clasificarea tuplelor de intrare, ar putea emite mai multe stream-uri pe baza tipului acestora, pe când un bolt care reprezintă un end-point al topologiei va reține tuplele citite, dar nu va emite nici un alt stream. Din punct de vedere al operațiilor executate asupra unei tuple, pe lângă aplicarea uneia sau a mai multor funcții asupra acestora, un bolt poate transforma și agrega tuple din diferite surse.

- **Spout**: termenul de Spout definește o sursă de tuple în Storm. Un spout citește date de la o sursă externă și le emite într-o topologie Storm. Tipul sursei externe nu este unul fixat, aceasta poate fi de exemplu un sistem bazat pe cozi precum Apache Kafka), un API sau

chiar un fișier text. Framework-ul suportă spout-uri fiabile sau nu. Daca în timpul execuției unei tuple aceasta eșuează, un spout fiabil va re-emite tupla pentru o nouă procesare, pe când unul de tip "unreliable" pierde starea unei tuple odată cu emiterea acesteia.

4.2.3. Arhitectura și paralelismul unui cluster Storm

Secțiunea 4.2.2 a realizat o prezentare a terminologiei folosite în Storm. Cu toate că framework-ul permite rularea aplicației într-un mod de simulare, topologiile sunt dedicate executării intr-un sistem de calcul distribuit, folosind mai multe noduri. O asemenea topologie este rulată într-un cluster Storm care permite scalabilitatea și executarea acesteia cu o viteză adecvată procesării big-data.

Un cluster Storm este dedicat execuției unei topologii și poate fi comparat cu un cluster Hadoop. În timp ce Hadoop rulează joburi de tip MapReduce, Storm rulează topologii. Diferența majoră dintr-un job MapReduce și o topologie Storm este aceea că execuția unui job este finită în ceea ce privește dimensiunea setului de date și a timpului de procesare, pe când o topologie este rulată la infinit sau până aceasta este eliminată, pe un stream infinit de date.

Arhitectura Storm se bazează pe un pattern de tip master-slave, de asemenea similar cu cel aplicat în Hadoop. Coordonarea proceselor master și a celor slave are loc printr-o componentă third-party numită **ZooKeeper** [9]. Nodul de tip master se numește **Nimbus**. Nimbus este responsabil pentru distribuirea codului în cadrul clusterului, asignând task-uri nodurilor de procesare, similar unui scheduler, și pentru monitorizarea erorilor la execuție. Nodurile care efectuează procesarea conțin un așa-numit **Supervisor**, iar fiecare Supervisor deține controlul asupra unuia sau a mai multor **Workeri** pe acel nod. Coordonarea dintre Nimbus și un Supervisor este asigurată de către ZooKeeper.

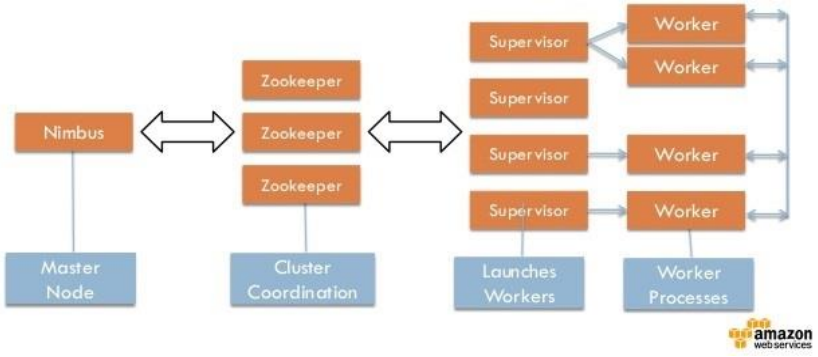


Figura 4.3: Arhitectura unui cluster Storm[16]

[16] http://image.slidesharecdn.com/clickstreamanalytics-amazonkinesisandapachestorm-150219125547-conversion-gate01/95/aws-webcast-amazon-kinesis-and-apache-storm-18-638.jpg?cb=1424350645

Pe lângă coordonarea realizată de ZooKeeper, acestă componentă este responsabilă și pentru reținerea stării fiecărui nod din cluster, având în vedere că Nimbus și nodurile Supervisor sunt componente stateless. Clusterul ZooKeeper constă in mod optim din 3, 5 sau (2n+1) noduri ZooKeeper [9], care coordonează și interschimbă informații între Nimbus și componentele Supervisor. Toate stările task-urilor aflate în execuție, dar și stările Nibusului și a Supervisorilor sunt păstrate în clusterul ZooKeeper. Prin urmare, acestea pot fi restartate sau realocate în cluster fără a întrerupe topologia care rulează. Scopul principal al Apache ZooKeeper-ului este de a fi rapid, simplu și de a putea fi folosit ca o bază pentru alte servicii mai complexe. De asemenea, acesta oferă garanția unor proprietăți semnificative precum consistență secvențială, atomicitate, imagine unică a sistemului (clienții văd aceleași date, indiferent din ce nod ZooKeeper le accesează) [10] și faptul că imaginea sistemului va fi up-to-date, cu o latență de maxim câteva secunde.

În ceea ce privește paralelismul, acesta se realizează la nivelul proceselor de tip worker. Un asemenea proces execută un task specific unei topologii. Un proces de tip worker nu rulează de sine stătător, ci creează un **executor** căruia îi atribuie sarcina de executa acest task. Un proces worker va fi părintele a mai multor asemenea executori. Un executor nu reprezintă nimic altceva decât un simplu thread creat de procesul worker părinte, iar task-ul executat de acest proces reprezintă într-o topologie Storm fie un bolt fie un spout. În mod implicit, Storm rulează un singur task per thread executor, dar există si posibilitatea rulării mai multor taskuri/executor în mod serial. Acest model de execuție paralelă este prezentat în figura 4.4.

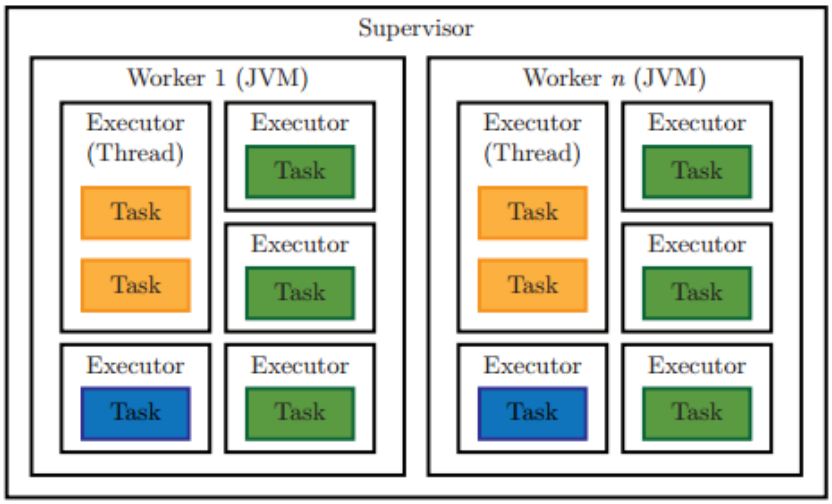


Figura 4.4: Modelul de executare în paralel într-un cluster Storm[17]

[17] https://hadoopabcd.files.wordpress.com/2015/04/storm-parallelism1.png

4.3. Heron

Storm a fost pentru multă vreme platforma principală pentru analize în timp real utilizată de Twitter. Cu toate acestea, pe măsură ce dimensiunea datelor care necesită procesare a crescut, s-a mărit și diversitatea și numărul de cazuri de utilizare care definesc contextul de aplicare a tehnologiei. Iar astfel multe din limitările framework-ului Storm au devenit vizibile. În consecință, pe data de 2 iunie 2015, Twitter a anunțat printr-o postare intitulată "Flying faster with Heron"[18] lansarea inițială a unui nou framework de analiză real-time numit Heron:

"*Procesăm miliarde de evenimente din Twitter zilnic. După cum probabil puteți ghici, analiză în timp real prezintă o provocare masivă. Sistemul nostru principal pentru o asemenea analiză a fost Storm, un sistem distribuit de procesare a stream-urilor pe care l-am lansat sub licență open-source. Însă pe măsură ce scala și diversitatea datelor Twitter au crescut, cerințele noastre au evoluat. Așadar am dezvoltat un sistem nou, Heron*"

4.3.1. Limitările Arhitecturii Storm

După cum a fost descris în capitolul 4.2.3, arhitectura unui cluster Storm se bazează pe execuția unui spout sau unui bolt sub forma unui task. Acest task este rulat de către un thread executor, găzduit de un proces de tip worker. Astfel, un nod de execuție dintr-un cluster este format dintr-un supervizor, care analizează o serie de instanțe JVM, fiecare dintre acestea rulând un număr de thread-uri executoare, ceea ce duce la un **design complex**. De asemenea, un singur host este capabil de a rula multiple procese worker, fiecare putând aparține unor topologii diferite.

Din punct de vedere al **sistemului de planificare**, fiecare thread executor este planificat de către JVM pe baza unui algoritm preemptiv și bazat pe priorități. Din moment ce fiecare thread rulează mai multe task-uri, executorul implementează la rândul său un al doilea algoritm de planificare pentru executarea task-ului corespunzător. Un asemenea mecanism de planificare pe mai multe nivele și interacțiunea complexă dintre acestea a introdus problema incertitudinii referitoare la determinarea momentului de timp când un task este planificat [11].

O altă problemă care apare în cadrul arhitecturii Storm este legată de Nimbus, nodul master din cluster. Printre funcțiile efectuate de către acesta se numără planificarea, monitorizarea și distribuirea fișierelor executabile JAR. De asemenea, Nimbus realizează și funcția de raportare a metricilor și gestionează contorizarea topologiilor care rulează. Așadar, Nimbus devine o componentă suprasolicitată din punct de vedere funcțional, fiindu-i atribuită prea multă responsabilitate. Cu atât mai mult, nodul master reprezintă un punct de cădere unic. În momentul în care Nimbus eșuează, utilizatorii nu mai sunt capabili să submită topologii pentru

[18] https://blog.twitter.com/2015/flying-faster-with-twitter-heron

execuție sau de a elimina topologii care deja rulează în cluster, iar o topologie care eșuează în momentul căderii nodului master nu poate fi detectată și recuperată în mod automat.

Nu în ultimul rând, au fost observate o serie de limitări în ceea ce privește eficiența. O serie de scenarii au fost identificate în sistemul de producție a celor de la Twitter, în care performanța impredictibilă din timpul execuției unei topologii au dus la reprocesarea sau pierderea unor tuple și au introdus o latență semnificativă. Cele mai comune cauze care au dus la reducerea performanței au fost următoarele [11]:

- **Replay-uri suboptime**: eșecul unei tuple într-un arbore de tuple conducea la eșecul întregului arbore. Acest efect devine o problemă mai ales în topologii cu frecvență mare de emitere de tuple, care se opreau din execuția propriu zisă pentru a reprocesa tuple care au fost pierdute.

- **Cicluri lungi de garbage collection**: Topologiile consumau multă memorie RAM pentru un worker care trebuia să realizeze cicluri de garbage collection, un astfel de ciclu ajungând la o durată de execuție mai mare de un minut. Acest lucru introducea ca și consecință latențe mari în procesarea tuplelor și rate mari de eșec.

Conform unui exemplu de topologie implementat de cei de la Twitter [11] special pentru a pune în evidență limitările framework-ului Storm, s-a observat o performanță cu opt ori mai scăzută în comparație cu un program clasic scris în Java, care realiza același set de operații.

4.3.2. Arhitectura Heron

Analizând o serie de desing-uri alternative pentru Apache Storm, Twitter a ajuns la concluzia că rescrierea acestui framework reprezintă o muncă prea laborioasă și a decis să implementeze o platformă nouă pentru procesarea stream-urilor în timp real, Heron. Obiectivul principal de design a fost menținerea compatibilității cu API-ul Storm pentru a facilita o migrare cu efort minimal. Astfel, modelul de date și API-ul pentru Heron sunt identice cu cel folosit de Storm. La fel ca și Storm, Heron execută topologii în care componente de tip Spout emit tuple sub formă de stream-uri, iar Bolt-urile sunt folosite pentru procesarea propriu-zisă a acestora.

Îmbunătățirile aduse de Heron pentru a depăși limitările evidențiate ale framework-ului Apache Storm se regăsesc în noul model arhitectural, prezentat în fig. 4.3.2. Utilizatorul submite topologiile planificatorului, care le execută sub forma unui job care constă în mai multe containere. Unul dintre containere rulează așa-numitul **Topology Master**, echivalentul nodului Nimbus din Storm, care este responsabil pentru managementul topologei. Fiecare dintre celelalte cointainere rulează un **Stream Manager** care realizează rutarea datelor, un manager pentru metrici care colectează și raportează diferite metrici și un număr de procese numite **instanțe**

Heron, care rulează spouts si bolts definite de către utilizator. Metadatele unei topologii care cuprind planul fizic și detaliile de execuție al acesteia sunt păstrate în **ZooKeeper**, prescurtat folosind acronimul ZK.

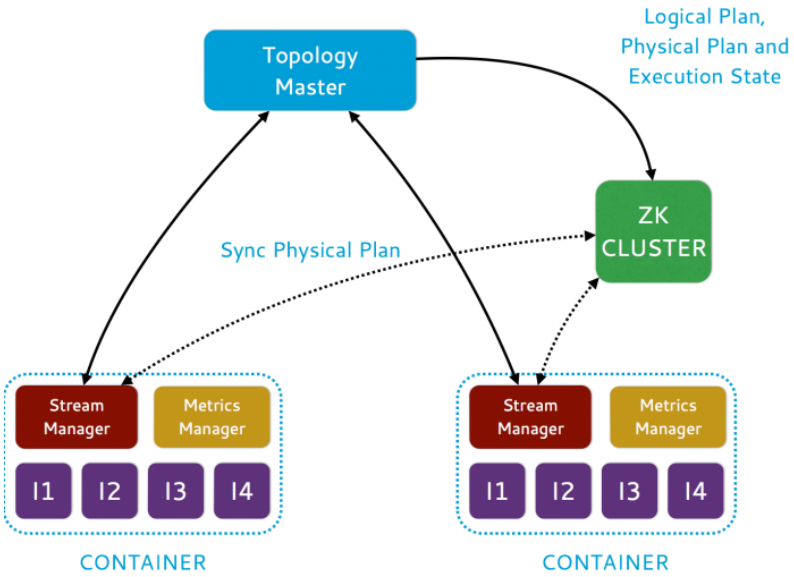


Figura 4.5: Arhitectura Heron [19]

Nimbus, nodul master care coordona toate topologiile dintr-un cluster Storm este înlocuit cu un *Topology Master* per topologie, astfel încât fiecare topologie are propriul ei coordonator. De asemenea, complexitatea unui nod de tip worker este redusă semnificativ prin abordarea folosind instanțe heron. Dimensiunea unei instanțe este mult mai mică decât a unui worker, iar ele pot fi distribuite pe mai multe containere, astfel încât obținerea unui cluster balansat este mult mai ușoară, iar relocarea unei instanțe în caz de întrerupere a execuției devine mult mai rapidă.

În urma unor benchmark-uri realizate de cei de la Twitter [11], Heron dovedește o performanță superioară fată de Storm, atât în ceea ce privește viteza de execuție, cât și ușurința de depanare a unei topologii și detectarea componentei din topologie care eșuează sau introduce latență mare. Arhitectura Heron permite utilizatorului un control mult mai bun al resurselor, Topology Master-ul permite gestiunea la nivel de topologie, iar migrarea unei topologii de pe un set de container pe altul devine mult mai ușoară. De asemenea, în ceea ce privește nodul master (Nimbus) din topologia Storm, problema de punct unic de eșec este eliminată prin reproiectarea arhitecturii inițiale.

[19] https://g.twimg.com/blog/blog/image/blog-figure-2.png

Capitolul 5. Proiectare de Detaliu și Implementare

În acest capitol va fi prezentată o soluție pentru un sistem de analiză de tip big-data pe baza studiului bibliografic descris în capitolul 3 și al analizei și a fundamentării teoretice realizate și prezentate în capitolul 4. Sistemul permite procesarea datelor în timp real și realizează o analiză continuă pentru identificarea celor mai discutate subiecte din rețeaua socială Twitter, cunoscute sub numele de "trending topics".

Solutia reprezinta o aplicatie implementata folosind framework-ul Apache Storm, scrisă folosind limbajul de programare Java. Aplicatia va realiza o analiza in timp real a tuturol tweet-urilor trimise pe reteaua sociala numita Twitter având ca scopul analizei acestor tweet-urilor pentru a determina asa numitele trending topics, un termen asociat celor mai discutate subiecte dintr-o retea sociala.

Sistemul a fost implementat initial în Apache Storm prezentat in capitolul 4.2 și migrat ulterior pe noul framework de analiza în timp real propus de cei de la Twitter, Heron descris în secțiunea 4.3. Capitolul 5 prezinta metoda de clasificare a celor mai discutate subiecte, cunoscute sub numele de trending topics, având ca și criteriu de clasificare hashtag-ul. Capitolul 5.2 prezintă metoda aleasă pentru realizarea acestei clasificări, dar și o serie de metode alternative.

De asemenea, vor fi prezentate tehnologiile folosite, formatul datelor de intrare, arhitectura topologiei, modelul de procesare și flow-ul execuției în cadrul sistemului implementat în Apache Storm, modelul de grupare a stream-urilor și diferite optimizări care cresc performanța modulului de calcul a geolocației. Rularea aplicației necesită o serie de configurări pentru a permite rularea într-un cluster Strom, pașii necesari fiind prezentați in capitolul 5.6.

Capitolul 5 se încheie cu prezentarea procesului de migrare a aplicației pe framework-ul Heron, a rulării ei în acest cluster și a modului de vizualizare și evaluarea topologiei prin interfața grafică oferită de acesta. În continuare mă voi referi la sistemul implementat în Apache Storm ca și **TwiTrends**, iar versiunea migrată pe Heron se va numi **TwiTrends-Heron**.

5.1. Framework-uri și Tool-uri

Pentru implementarea, rularea, testarea și integrarea aplicației TwiTrends au fost folosite o serie de tool-uri și framework-uri. Ca și mediu de dezvoltare am folosit Eclipse IDE, versiunea Mars, iar pentru managementul integrării și al compilării Apache Maven 3.3. Aplicația este scrisă în întregime în limbajul Java și comunică cu o serie de librării externe pentru realizarea analizei de tweet-uri în timp real și clasificarea acestora, precum Twitter4J și Bing Maps API. Pentru testarea, depanarea și validarea aplicației au fost folosite framework-ul JUnit și librărila log4j pentru generarea fișierelor log. Pentru rularea aplicației, atât într-un cluster simulat cât și unul real au fost folosite Apache Storm și Heron.

- **Eclipse Mars (June 2015)**[20]: mediu de dezvoltare (IDE) pentru dezvoltarea aplicațiilor în Java. Oferă suport pentru integrarea proiectelor folosind Maven.

- **Apache Maven 3.3**[21]: tool pentru managementul proiectelor software, bazat pe conceptul de POM (project object model). Maven permite o integrare simplă a diferitelor componente externe într-un proiect și realizează compilarea și integrarea aplicației în mod automat.

- **Twitter4J**[22]: o librărie open-source pentru accesarea API-ului Twitter din Java. Twitter4J permite integrarea serviciilor Twitter în aplicații Java și reprezintă o metodă ușoară pentru accesarea funcționalității acestuia. Twitter4J implementează o serie de funcționalități precum autentificarea utilizatorului, accesul la stream-uri de tweet-uri și diferite metode de acces la câmpurile din care acesta este compus.

- **Twitter Streaming API**[23]: permite programatorului acces cu latență mică la stream-ul global de tweet-uri din Twitter. O implementare corespunzătoare a acestui API va trimite un mesaj indicând un nou tweet sau un alt tip de eveniment din rețeaua socială Twitter, evitând overhead-ul asociat accesării unui serviciu de tip REST.

- **Apache Strom 1.0.1**[24]: un sistem open-source gratuit dedicat implementării unui sistem de computații distribuite în timp real. Acest framework a fost prezentat în detaliu în capitolul 4.2.

- **Heron 0.14.0**[25]: un sistem de procesare real-time, distribuit și tolerant la eșec de la Twitter, care oferă un API compatibil cu cel al framework-ului Apache Storm. Conceptele de bază și arhitectura Heron au fost descrise în capitolul 4.3.

- **Bing Maps API**[26]: dezvoltat de Microsoft, Bing Maps API permite accesul la Bing Maps pentru dezvoltarea aplicațiilor care necesită servicii de geolocație.

[20] https://eclipse.org/mars/
[21] https://maven.apache.org/
[22] http://twitter4j.org/en/
[23] https://dev.twitter.com/streaming/overview
[24] http://storm.apache.org/releases/1.0.1/index.html
[25] http://twitter.github.io/heron/
[26] https://www.microsoft.com/maps/Default.aspx

Acest API a fost folosit în dezvoltarea TwiTrends pentru calculul locației (oraș și țară) la nivel de tweet pe baza longitudinii și a latiduninii conținute în acesta, folosite în cadrul clasificării la nivel de geolocație.

- **JUnit 4**[27]: un framework simplu pentru scrierea de teste repetabile. Reprezintă o instanță a arhitecturii xUnit pentru frameworkuri dedicate testării componentelor.

- **Log4j**[28]: librărie open-source de logging pentru Java, aflată sub licență Apache. Log4j permite o serie de nivele de logging (info, debug, warn, error) și specificarea formatului mesajului logat, atât la nivel de consolă cât și la nivel de fișier text.

5.1.1. Formatul datelor de intrare

Datele de intrare folosite de TwiTrends și TwiTrends-Heron pentru a determina cele mai discutate subiecte din rețeaua socială Twitter reprezintă tweet-uri reale citite accesând API-ul Twitter Stream API. Acest API pune la dispoziție mai multe metode de acces și oferă posibilitatea filtrării Tweet-urilor accesate, pe baza unui criteriu specificat în URL-ul requestului.

Primul pas necesar pentru accesul la aceste date este autentificarea folosind următoarele date specifice unui programator de aplicații Twitter, obținute în urma creării unui cont pentru dezvoltatori: cheia consumatorului (Consumer Key), secretul consumatorului (Consumer Secret), token de acces (Access Token) și parola acestui token, numită secretul tokenului de acces (Access Token Secret).

O dată autentificat, dezvoltatorul are acces la stream-urile publice, pe care le poate accesa pentru a obține în timp real o mostră de tweet-uri reale, numite statusuri. Stream-urile publice sunt înpărțite în trei categori:

- POST statuses / filter
- GET statuses / sample
- GET statuses / firehose

Primul endpoint returnează toate tweet-urile care corespund cel puțin unui filtru specificat ca și parametu. Este posibilă specificarea unui număr multiplu de filtre, însă cel puțin unul trebuie să fie prezent. Posibilitatea de filtrare este o caracteristică demnă de considerat la

[27] http://junit.org/junit4/
[28] https://logging.apache.org/log4j/1.2/

implementarea diferitelor aplicații, însă în cadrul aplicațiilor TwiTrends acest mecanism nu reprezintă un avantaj.

Endpoint-ul "GET statuses / sample" returnează un număr de statusuri publice alese aleator și este cel ales pentru implementarea clasificării tweet-urilor. TwiTrends consideră toate tweet-urile citite și nu necesită o filtrare inițială a datelor de intrare. La rândul ei însă, aplicația le elimină pe cele cu conținut irelevant pe măsură ce acestea sunt înaintate prin topologie, acest aspect va fi prezentat în detaliu în capitolul 5.4. La un nivel intern, cel de-al doilea endpoint este accesat pe baza următoarului URL:

https://stream.twitter.com/1.1/statuses/sample.json

Pentru accesul aplicației TwiTrends la acest stream de status-uri este folosită librăria Twitter4J care permite atât autentificarea la API-ul Twitter cât și un acces simplificat la această resursă prin metoda sample(). Conform descrierii metodei, aceasta returnează un subset de tweet-uri aleatore a tuturor statusurilor publice. Nivelul de acces implicit oferă un subset a stream-ului Firehose. Endpoint-ul Firehose permite accesul la totalitatea tweet-urile publice din rețeaua socială Twitter, însă necesită drepturi de acces suplimentare, care au dovedit a fi foarte dificile de obținut.

Metoda sample() returnează Tweet-uri sub forma unui obiect de tip Status. Acest obiect este construit pe baza json-ului returnat de API-ul Twitter Stream API și încapsulează câmpurile și valorile din interiorul structurii de date de tip json. O formă simplificată a unui status este prezentată in figura 5.1. Câmpurile relevante pentru TwiTrends sunt următoarele:

- text: conține textului unui status (tweet) postat de către un utilizator al rețelei sociale.

- geoLocation: reprezintă coordonatele de longitudine și latitudine ale utilizatorului care a emis tweet-ul. Coordonatele sunt reprezentate sub forma unui obiect de tip GeoLocation, care conține cele două câmpuri.

- place: reprezintă locații specifice și coorrdonatele geo corespunzătoare. Ele pot fi atașate unui Tweet, însă câmpul este unul opțional. Obiectul de tip Place conține o serie de atribute folosite pentru o descriere mai detaliată a locației. În ceea ce privește TwiTrends, atributele relevante sunt numele, de exemplu: name= "Paris", țara (country="France") și tipul locației.

- hashtagEntities: reprezintă o listă cu toate hashtag-urile menționate în tweet-ul curent. Valorile de tip hashtag sunt extrase din textul statusului și sunt reprezentate sub forma unui șir de caractere, fără a fi precedate de simbolul de hashtag "#", precum în mesajul inițial.

```
StatusJSONImpl{
    createdAt=Tue Jun 21 16:30:10 EEST 2016,
    id=745247441159610368,
    text='Just posted a photo @ Folkestone Harbour https://t.co/hYeArT43Vk #summer',
    source='<a href="http://instagram.com" rel="nofollow">Instagram</a>',
    isTruncated=false,
    inReplyToStatusId=-1,
    inReplyToUserId=-1,
    isFavorited=false,
    isRetweeted=false,
    favoriteCount=0,
    inReplyToScreenName='null',
    geoLocation=GeoLocation{latitude=51.07954851, longitude=1.18627838},
    place=PlaceJSONImpl{
        name='Folkestone',
        streetAddress='null',
        countryCode='GB',
        id='4569cb6dc10540ba',
        country='United Kingdom',
        placeType='city',
        url='https://api.twitter.com/1.1/geo/id/4569cb6dc10540ba.json',
        fullName='Folkestone, England',
        boundingBoxType='Polygon',
        boundingBoxCoordinates=[[Ltwitter4j.GeoLocation;@4d9b5cd],
        geometryType='null',
        geometryCoordinates=null,
        containedWithIn=null
    },
    retweetCount=0,
    isPossiblySensitive=false,
    lang='en',
    contributorsIDs=[],
    retweetedStatus=null,
    userMentionEntities=[],
    urlEntities=[URLEntityJSONImpl{url='https://t.co/hYeArT43Vk', ...]
    hashtagEntities=[summer],
    mediaEntities=[],
    symbolEntities=[],
    currentUserRetweetId=-1,
    user=UserJSONImpl{...}
}
```

Figura 5.1: Structura unui status emis de Twitter4J

5.2. Metoda de determinare a "Trending Topics"

"Un cuvânt, o frază sau un subiect care este menționat la o rată mai mare decât altele se definește ca fiind un trending topic. Acestea devin populare fie printr-un efort concentrat al utilizatorilor, fie printr-un evenimet care determină discuții pe o anumită tematică." [29]

Determinarea celor mai discutate subiecte, de asemenea cunoscute sub numele de "trending topics" a devenit o provocare o dată cu explozia numărului de utilizatori al rețelelor sociale. Diferite metode au fost implementate, ulterior îmbunătățite și apoi înlocuite de alte strategii de clasificare ale acestora.

O metodă prezentată intr-un jurnal publicat în anul 2013 [12] presupune analiza la nivelul întregului mesaj conținut într-un tweet. Pentru a determina cele mai discutate subiecte, metoda presupune contorizarea celor mai menționați termeni în cadrul tweet-urilor postate în rețeaua socială Twitter, i.e. cel mai frecvent folosit vocabular într-un trending topic. În primă fază se elimină cuvintele considerate irelevante, după care se calculează frecvența de apariție al fiecărui cuvânt. Frecvența cuvintelor este raportată la o serie de categorii de subiecte: comemorări, evenimente aflate în desfășurare, știri sau poze care reprezintă un anumit topic. Problema majoră în aplicarea acestei metode este faptul că multe tweet-uri conțin cuvinte relevante pentru mai mult decât doar o singură categorie menționată. Mai mult, clasificarea se face la nivel de cuvânt și nu la nivel de semantică al mesajului. De multe ori o persoană este sarcastică, astfel încât cuvântul "fericit" ar putea avea o semnificație cu totul opusă.

În contextul TwiTrends, metoda de clasificare aleasă pentru determinarea de trending topics pe un interval de timp este o metodă bazată pe extragerea și contorizarea hashtag-urilor dintr-un tweet. Un hashtag este reprezentat printr-un cuvânt sau mai multe cuvinte concatenate care încep cu simbolul "#". Simbolul de hashtag este folosit în cadrul unei rețele sociale precum Twitter pentru a identifica un mesaj ca aparținând unui anumit topic. Astfel, un topic este reprezentat printr-o mulțime de hashtag-uri asociate acestuia. Metoda este cunoscută în domeniul analizei de date provenite din rețele sociale și sub numele de "Trending Hashtags".

Presupunem două subiecte A și B. Faptul că subiectul A este mai popular decât B este echivalent cu a spune că numărul de menționări ale subiectului A este mai mare decât numărul de menționări ale subiectului B. Această relație este descrisă prin Formula 5.1.

$$popularite(A) \geq popularite(B) <=> nr\_mentionari(A) \geq nr\_mentionari(B)$$

Formula 5.1: Relația dintr-e popularitatea a unui subiect și numărul de menționări

[29] https://en.wikipedia.org/wiki/Twitter#Trending_topics

Aplicând semantica unui hashtag pe care se bazează metoda de "Trending Hashtags", putem deduce că numărul de menționări ale unui subiect reprezintă de fapt numărul de hashtag-uri asociat acestuia, hashtag-ul identificând topicul căruia îi aparține. Astfel, identificarea unui subiect devine echivalentă cu identificarea numărul de hashtag-uri care reprezintă acel subiect.

De asemenea, în analiză trebuie considerată și dimensiunea de timp. Popularitatea unui subiect este raportată la o durată de timp, astfel aceasta este definită pe un interval mărginit t1 – t0, notat cu Δt. Timpul t1 este timpul curent, iar t0 reprezintă un moment din trecutul apropiat. Ca și exemplu, considerăm termenul "big data" ca find un trending topic la momentul de timp t1, daca numărul de hashtag-uri care sunt asociate acest subiect identificate pe un interval de timp prestabilit de oridinul orelor este de o anumită mărime. Prin urmare, folosind notația anteriară, în care A și B reprezintă două subiecte diferite și în plus, hashtag_a reprezintă un hashtag care determină subiectul A, iar hashtag_b este asociat topicului B, Formula 5.1 poate fi rescrisă sub forma:

$$popularity(A) \geq popularity(B)$$

$$<=>$$

$$\frac{\sum_{t0}^{t1} Count(hashtag_a)}{\Delta t} \geq \frac{\sum_{t0}^{t1} Count(hashtag_b)}{\Delta t}$$

Formula 5.2: Popularitatea unui subiect în funție de
numărul de hashtag-uri care îl determină într-un interval de timp.

5.3. Topologia TwiTrends

Implementarea unui sistem de procesare în timp real în Apache Storm sau Heron presupune proiectarea la nivel de detaliu a arhitecturii unei topologii, care va fi transpusă ulterior în cod și rulată pe un cluster. Astfel, implementarea aplicației TwiTrends începe cu implementarea arhitecturii topologiei la nivel de componentă. De asemenea, proiectarea presupune și definirea relațiilor între spouts și bolts prin intermediul stream-urilor, a specificării paralelismului fiecărui bolt și al descrierii modelului de mapare a tuplelor fiecărui stream la aceste instanțe.

În continuare va fi descrisă topologia TwiTrends care reprezintă arhitectura de bază a sistemului de determinare ale celor mai discutate subiecte din rețeaua socială Twitter. Topologia reprezintă un graf orientat aciclic, alcătuit din dintr-un singur nod spout, cel care emite tweet-uri accesând librăria Twitter4J, o serie de noduri de tip bolts, folosite pentru procesarea, filtrarea și înaintarea datelor și stream-urile de tuple care leagă aceste componente. Figura 5.2 descrie arhitectura topologiei TwiTrends.

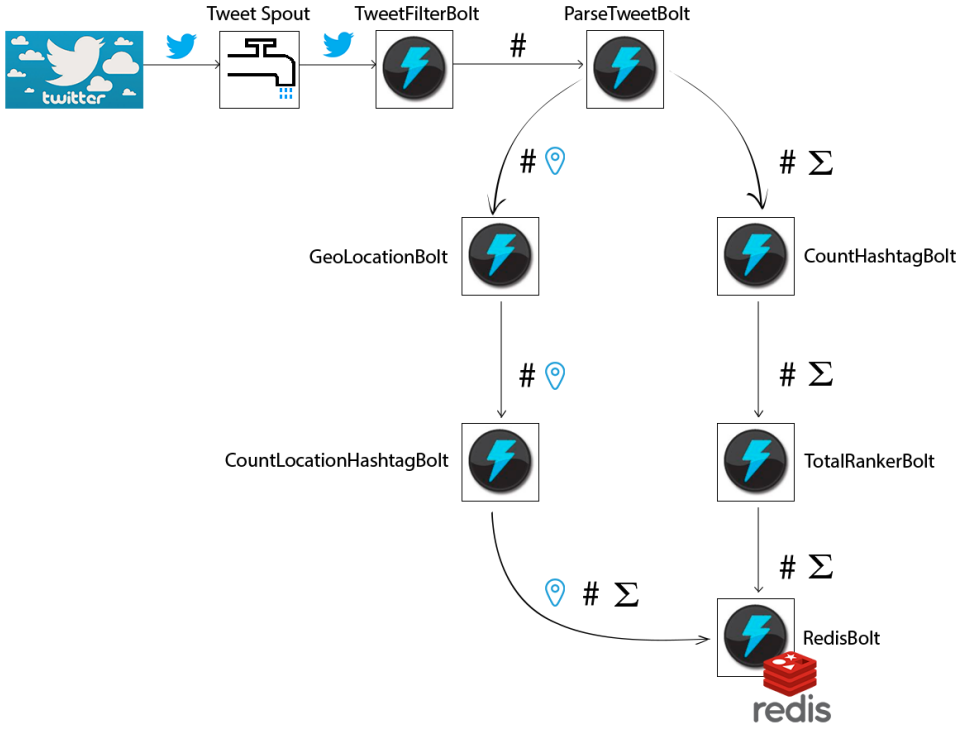


Figura 5.2: Arhitectura topologiei TwiTrends

5.4.1. Componentele topologiei TwiTrends

Topologia TwiTrends este alcătuită dintr-o componentă top-level, care conține o serie de elemente și conexiuni între acestea. Prin elemente înțelegem un spout, folosit pentru emiterea tweet-urilor în topologie și un număr de opt bolts, folosite pentru procesarea acestora. De asemenea, arhitectura descrie și interacțiunea dintre aceste componente, definită pe baza stream-urilor de tuple prin care acestea comunică. Componentele arhitecturii TwiTrends prezentate în Figura 5.2 sunt următoarele:

- **TwiTrendTolology**: reprezintă componenta top-level a topologiei. Cuprinde spout-ul folosit pentru emiterea tweet-urilor și cele opt bolt-uri folosite pentru procesarea acestora. De asemenea, topologia definește și modelul de comunicare între aceste componente pe baza stream-urilor de date, nivelul de paralelism a

fiecărei dintre acestea și modul de grupare a stream-urilor. Gruparea stream-urilor și modelul de paralelism vor fi discutate în detaliu în secțiunea 5.4.2.

- **TweetSpout**: reprezintă componenta folosită pentru emiterea tweet-urilor în topologia TwiTrends. Aceasta este singurul spout folosit, având în vedere ca este necesară doar citirea și înaintarea a unui singur stream. TweetSpout folosește API-ul twitter4j pentru a accesa stream-ul public de tip sample pus la dispoziție de către Twitter Streaming API prezentat în secțiunea 5.1.1. Spout-ul realizează autentificarea care permite accesul la acest stream, iar apoi citește în mod continuu fiecare tweet emis de către stream. Fiecare tweet citit este plasat într-o coadă, folosită pe post de buffer și emite apoi fiecare element al cozii urmând regula FIFO(first-in-first-out). Tuplele emise de către TweetSpout conțin un singur câmp, acesta reprezentând întregul tweet.

- **TweetFilterBolt**: citește tweet-urile emise de TweetSpout și realizează filtrarea acestora. Având în vedere faptul că TwiTrends realizează o clasificare bazată pe hashtag-urilie prezente într-un tweet, componenta de filtrare evită emiterea tweet-urilor irelevante prin selectarea unui subset a acestora care conțin cel puțin un hashtag. De asemenea, filtrarea are loc și la nivelul formatului mesajului. Vor fi emise doar tweet-uri care conțin mesaje codificate conform standardului Unicode care cuprinde cifrele ASCII, alfabetul latin și extensia acestuia pentru caractere speciale folosite în diverse alfabete europene precum cel croat, român sau slovac. După filtrarea unui tweet citit, TweetFilterBolt emite un stream a cărui tuple conțin doar acele tweet-uri care satisfac condițiile de filtrare enumerate.

- **ParseTweetBolt**: prelucrează tweet-urile filtrare emise ca și tuple de către componenta TweetFilterBolt. Având în vedere că fiecare tuplă este filtrată, la acest nivel avem gatanția că fiecare tweet conține cel puțin un hashtag. ParseTweetBolt parsează textul unui tweet și extrage toate hashtag-urile alături de locația din care a fost postat. Pentru un acces mai rapid, hashtag-urile și geolocația sunt accesate direct din obiectul de tip Status, care reprezintă un tweet, pe baza entităților prezentate în secțiunea 5.1.1. Odată extrase aceste valori, bolt-ul emite un nou stream în care o tuplă este alcătuită dintr-o pereche de câmpuri care reprezintă textul hashtag-ului și locația acestuia.

- **CountHashtagBolt**: preia tweet-urile parsate prin intermediul componentei ParseTweetBolt și contorizează fiecare hashtag. CountHashtagBolt folosește o tabelă de dispersie pe post de buffer pentru a mapa fiecare hastag la contorul său. Tabela este actualizată la fiecare tuplă citită, pe baza valorilor acesteia. Componenta emite un stream de ieșire care conține o perechea corespunzătoare intrării în tabela de disperie care conține hashtag-ul și contorul asociat acestuia.

- **IntermediateRankerBolt**: reprezintă o componentă intermediară generică pentru determinarea celor mai menționate N hashtag-uri. Numărul de hashtag-uri considerate poate fi menționat ca și parametru. IntermediateRankerBolt poate fi folosit pentru clasificarea oricărui tip de obiecte, însă in contextul TwiTrends obiectele care necesită clasificare sunt hashtag-urile. Acestea sunt citite din stream-ul emis de către CountHashtagBolt, iar rezultatele intermediare sunt pasate către TotalRankerBolt, care realizează clasificarea finală.

- **TotalRankerBolt**: realizează un ranking total al tuturor hashtag-urilor contorizate. Folosirea unui model de clasificare intermediar, urmat de agregarea rezultatelor facilitează execuția în paralel. IntermediateRankerBolt poate fi executat în paralel, pe mai multe thread-uri în cluster, iar rezultatele finale sunt grupate într-o instanți finală, cea de TotalRankerBolt. Tuplele clasificate provin din mai multe stream-uri intermediare emise de boltul pentru clasificarea intermediară. Componenta permite de asemenea specificarea frecvenței de emitere a tuplelor clasificate prin intermediul unui parametru. Stream-ul de ieșire, emis o dată la 5 secunde, conține primele N hashtag-uri, împreună cu numărul de apariții acestora în toate tweet-urile procesate.

- **GeoLocationBolt**: preia hashtag-ul emis de către ParseTweetBolt, alături de locația tweet-ului. Având în vedere faptul că locația este codificată prin coordonate de longitudine și latitudine, este necesară transformarea acestora într-o locație concretă. Pentru determinarea orașului și a tării corespunzătoare locației citite, GeoLocationBolt accesează un modul de calcul bazat pe chaching, descris în detaliu în secțiunea 5.4.4. În ceea ce privește emiterea stream-ului de ieșire trebuie considerate două cazuri: în cazul în care tupla de intrare conținea o valoarea pentru locație, GeoLocationBolt va emite orașul și țara acestei locații, alături de hashtag-ul ei corespunzător. În caz contrar, se folosește o locație santinelă care reprezintă absența locației in tweet-ul corespunzător hashtag-ului.

- **CountLocationHashtagBolt**: prezintă o funcționalitate similară cu cea a componentei CountHashtagBolt însă introduce o a doua dimensiune în ceea ce privește contorizarea. Contorizarea nu mai este raportată doar la textul hashtag-ului ci și la geolocația din care acesta a fost emis. În cazul în care nu este posibilă extragerea locației a tweet-ului reprezentat de tupla de intrare, bolt-ul folosește o variabilă de tip santinelă numită "UNKNOWN LOCATION" ca și cheie în tabela de dispersie, aceasta fiind emisă de către bolt-ul responsabil pentru calcularea locației. Structura buffer-ului este reprezentată printr-o relație de tip cheie -> valoare, în care cheia este compusă din două câmpuri: locație și textul hashtag-ului. CountLocationHashtagBolt emite un stream de ieșire, în care fiecare tuplă conține trei câmpuri, primele două reprezentând cheia tabelei folosite pe post de structură de date intermediară, urmate de contorul corespunzător cheii.

- **RedisBolt**: este bolt-ul final al topologiei TwiTrends. RedisBolt reprezintă o componentă care grupează rezultatele la nivel global. Prin nivel global înțelegem toate rezultatele finale obținute în urma procesării tweet-urilor. Componenta salvează în mod generic cele mai discutate N topicuri identificate pe baza hashtag-urilor procesate în topologie. De asemenea, acest bolt conține și contorizarea hashtag-urilor la nivel de geolocație, obținute din stream-ul emis de către CountLocationHashtagBolt. Fiind componenta finală, RedisBolt trebuie să permită accesul la datele stocate din exteriorul aplicației. Pentru a facilita această funționalitate, bolt-ul publică toate datele conținute către un broker de mesaje Redis[30], care reprezintă un spațiu de stocare local, folosit ca și o bază de date pentru salvarea în manieră cache a datelor și care poate fi accesat conform paradigmei de pasare de mesaje publish/subscribe[31]. RedisBolt realizează agregarea stream-urilor emise de componentele TotalRankerBolt și CountLocationHashtagBolt. Nu este emis nici un stream de ieșire, având în vedere că acesta este ultimul bolt din topologie, iar datele sunt gata procesate.

Atât Apache Strom cât și Heron necesită identificarea unui bolt sau a unui spout printr-un ID unic. Tabelul 5.1 prezintă fiecare componentă a arhitecturii TwiTrends alături de identificatorul ei corespunzător:

[30] http://redis.io/
[31] http://redis.io/topics/pubsub

Numele componentei	Identificatorul componentei
TwiTrendTolologi	twi-trends-topology
TweetSpout	tweet-spout
TweetFilterBolt	tweet-filter-bolt
ParseTweetBolt	parse-tweet-bolt
CountHashtagBolt	count-hashtag-bolt
GeoLocationBolt	geolocation-bolt
CountLocationHashtagBolt	count-location-hashtag-bolt
IntermediateRankerBolt	intermediate-ranker
TotalRankerBolt	total-ranker
RedisBolt	redis-bolt

Tabelul 5.1: Identificatorii componentelor arhitecturii TwiTrends

5.4.2. Nivelul de paralelism şi gruparea stream-urilor

Fiecare componentă a topologiei (spout sau bolt) poate fi executată într-un cluster Storm sau Heron pe mai multe instanțe care rulează în paralel. Numărul de instanțe este specificat în elementul top-level. Ca și regulă de bază, un bolt care necesită un timp mai mare de procesare a unei tuple trebuie rulat cu un nivel de paralelizare mai mare pentru a menține un flux de date continuu. În cadrul Topologiei TwiTrends, astfel de componente de procesare sunt:

- GeoLocationBolt: necesită un timp de procesare mai lung datorită latenței introduse la accesarea API-ului Bing pentru calculul geolocației

- IntermediateRankingBolt: procesarea în paralel a ranking-urilor, a cărei rezultate sunt agregate.

Există si noduri de procesare care nu pot fi rulate în paralel. De exemplu, TotalRankerBolt este reprezentat printr-o singură instanță, având în vedere că emite un singur stream care reprezintă rezultatele agregării tuplelor obținute de la IntermediateRankerBolt. La fel este și în cazul RedisBolt, acesta fiind de asemenea un bolt de colectare a datelor la nivel global pentru a le pune la dispoziție mediului exterior. Tabelul 5.2 prezintă fiecare componentă și numărul de instanțe care sunt executate în paralel:

Numele componentei	Nivelul de paralelism
TweetSpout	1
TweetFilterBolt	2
ParseTweetBolt	4
CountHashtagBolt	2
GeoLocationBolt	4
CountLocationHashtagBolt	2
IntermediateRankerBolt	4
TotalRankerBolt	1
RedisBolt	1

Tabelul 5.2: Nivelul de paralelism al componentelor topologiei TwiTrends

Framework-urile Apache Strom și Heron oferă o serie de implementări pentru diferite semantici de gruparea stream-urilor. Această grupare este necesară pentru a putea specifica la nivel programatic modul de distribuire a tuplelor către mai multe instanțe ale aceluiași bolt. Metodele de grupare folosite în cadrul topologiei TwiTrends sunt următoarele:

- **Shuffle grouping**: tuplele sunt distribuiete aleator către task-urile elementelor de tip bolt. Gruparea asigură faptul distribuirea este una uniformă, astfel încât fiecare task va primi un număr egal de tuple.

- **Fields grouping**: stream-ul de tuple este partiționat în funcție de câmpurile specificate în grupare. De exemplu, dacă stream-ul este grupat după câmpul hashtag, tuplele conținând același hashtag vor fi procesate de aceeași instanță a unui bolt, dar tuple care conțin un hashtag diferit pot fi procesate de către alte instanțe ale acestuia.

- **Global grouping**: intregul stream de date este transmis unui singur task corespunzător instanței unui bolt. În cazul în care există mai multe instanțe, este aleasă cea cu cel mai mic identificator.

Componentele care doar procesează și înaintează date în topologie folosesc gruparea de tip shuffle, sau aleator. Ele nu salvază date intermediare, deci nu depind la nivel de valoare conținută în tuplă de stream-ul pe care în procesează. Bolt-uri precum *CountHashtagBolt* sau *CountLocationHashtagBolt* salvează valoarea hashtag-ului respeciv al geolocației pentru un tweet procesat. Pentru a reduce timpul de regrupare al stream-urilor emise, folosirea unei grupări la nivel de hashtag sau locație evită prezența aceleiași chei în map-uri din instanțe diferite.

Astfel, două hashtag-uri sau geolocații identice conținute în tuple diferite vor fi procesate de aceeași instanță a unui bolt.

Pentru componentele *TotalRankerBolt* și *RedisBolt* s-a ales o grupare globală. Ele reprezintă noduri care nu vor fi paralelizate, deoarece stream-ul emis de acestea reprezintă un rezultat al analizei realizate de către topologia TwiTrends. Mai mult, *RedisBolt* este un bolt terminal care trebuie să conțină toate rezultatele obținute în urma întregii procesări a tuturor tweet-urilor citite. Gruparea stream-urilor la nivel de componentă în cadrul topologiei TwiTrends este enumerată în tabelul 5.3.

Numele boltului	**Gruparea stream-urilor de intrare**
TweetFilterBolt	Shuffle grouping
ParseTweetBolt	Shuffle grouping
CountHashtagBolt	Fields grouping
GeoLocationBolt	Shuffle grouping
CountLocationHashtagBolt	Fields grouping
IntermediateRankerBolt	Fields grouping
TotalRankerBolt	Global grouping
RedisBolt	Global grouping

Tabelul 5.3: Gruparea stream-urilor pentru fiecare bolt

5.4. Implementarea topologiei TwiTrends

Implementarea topologiei presupune transpunerea acesteia în cod Java. Fiecare componentă (spout sau bolt) corespunde unei clase, iar nivelul de paralelism, stream-urile prin care comunică componentele și tipul de grupare a acestora sunt specificate la nivelul clasei care implementează topologia. .

5.4.1. TwiTrendsTopology

Clasa TwiTrendsTopology reprezintă întreaga topologie. Instanțierea elementelor de tip spout și bolt este delegată unei clase specializate din framework-ul Apache Storm numită TopologyBuilder. La momentul instanțierii trebuie specificate identificatorul componentei, clasa care o reprezintă, numărul de instanțe care vor executa în paralel și tipul grupării stream-urilor. Pe lângă instanțierea acestor componente, clasa este responsabilă pentru executarea tuturor operațiilor de inițializare. În cazul de față, singura operație care trebuie realizată înaintea de execuția topologiei este inițializarea memoriei cache pentru locații. Următorul fragment de cod descrie operațiile realizate în cadrul clasei care au fost menționate:

```
// Initialize Cache
cache = ConcurrentLocationCache.getInstantce();

// Set Spouts
builder.setSpout(TopologyIdentifiers.TWEET_SPOUT, tweetSpout, 1);

// Set Bolts
builder.setBolt(TopologyIdentifiers.TWEET_FILTER_BOLT, new TweetFilterBolt(), 2)
    .shuffleGrouping(TopologyIdentifiers.TWEET_SPOUT);
builder.setBolt(TopologyIdentifiers.PARSE_TWEET_BOLT, new ParseTweetBolt(), 4)
    .shuffleGrouping(TopologyIdentifiers.TWEET_FILTER_BOLT);
```

În acest fragment de cod este prezentată inițializarea memoriei cache apelând metoda getInstance(). Fiind un obiect de tip singleton, inițializarea are loc în momentul apelării constructorului. De asemenea putem vedea adăugarea spout-ului la TopologyBuilder cu executare pe un singur thread, variabila tweetSpout fiind instanțată in prealabil deoarece necesită autentificarea pentru Twitter Stream API. După crearea spout-ului sunt adăugate componentele de tip bolt la topologie. Fragmentul de cod prezintă apoi adăugarea bolt-ului de filtrare și a celui de parsare a tweet-urilor. Ambele folosesc un "shuffle grouping", deoarece tuplele pot fi alocate oricăror instanțe. În plus, gruparea specifică și faptul că cele două componente comunică: un element TweetParseBolt citește streamul de date emis de un TweetFilterBolt.

Pentru citirea și parsarea unei tuple este important ca fiecare componentă să cunoască formatul tuplei pe care o recepționează. Accesul la elementele unei tuple se face pe baza identificatorului acesteia. Tabelul 5.4 specifică formatul stream-ului emis de fiecare spout și bolt implementat:

Numele componentei	**Formatul tuplei emise**	**Identificatorii câmpului**
TweetSpout	(Status)	tweet
TweetFilterBolt	(Status)	tweet
ParseTweetBolt	(String, TweetLocationData)	hashtag, location-data
CountHashtagBolt	(String, Long)	hashtag, count
GeoLocationBolt	(Location, String)	location, hashtag
CountLocationHashtagBolt	(LocationHashtagCount)	location-hashtag-count
IntermediateRankerBolt	(Rankings)	rankings
TotalRankerBolt	(Rankings)	rankings
RedisBolt	-	-

Tabelul 5.4: Formatul tuplelor emise de fiecare componentă a topologiei TwiTrends

Deoarece execuția unui TweetFilterBolt este mai rapidă decât cea a unui ParseTweetBolt, paralelismul specificat pentru acesta este 2, iar pentru cel de parsare este 4, ceea ce înseamnă că poate fi executat pe patru thread-uri, în paralel.

Toate bolt-urile topologiei se află în pachetele com.twitrends.bolt. Subpachetul com.twitrends.bolt.apache conține clase preluate din proiectul open-source "storm starter"[32]. Analog, spout-ul folosit se află în pachetul com.twitrends.spout. Pe lângă aceste două tipuri de componente, TwiTrendsTopology accesează și o serie de interfețe din pachetul com.twitrends.util, pentru a obține constante folosite în cadrul topologiei, identificatorii componentelor pe care le instanțează, numele câmpurilor fiecărei tuple și valorile necesare autentificării la Twitter Stream API. Figura 5.3 prezintă o diagramă de clase simplificată a topologiei:

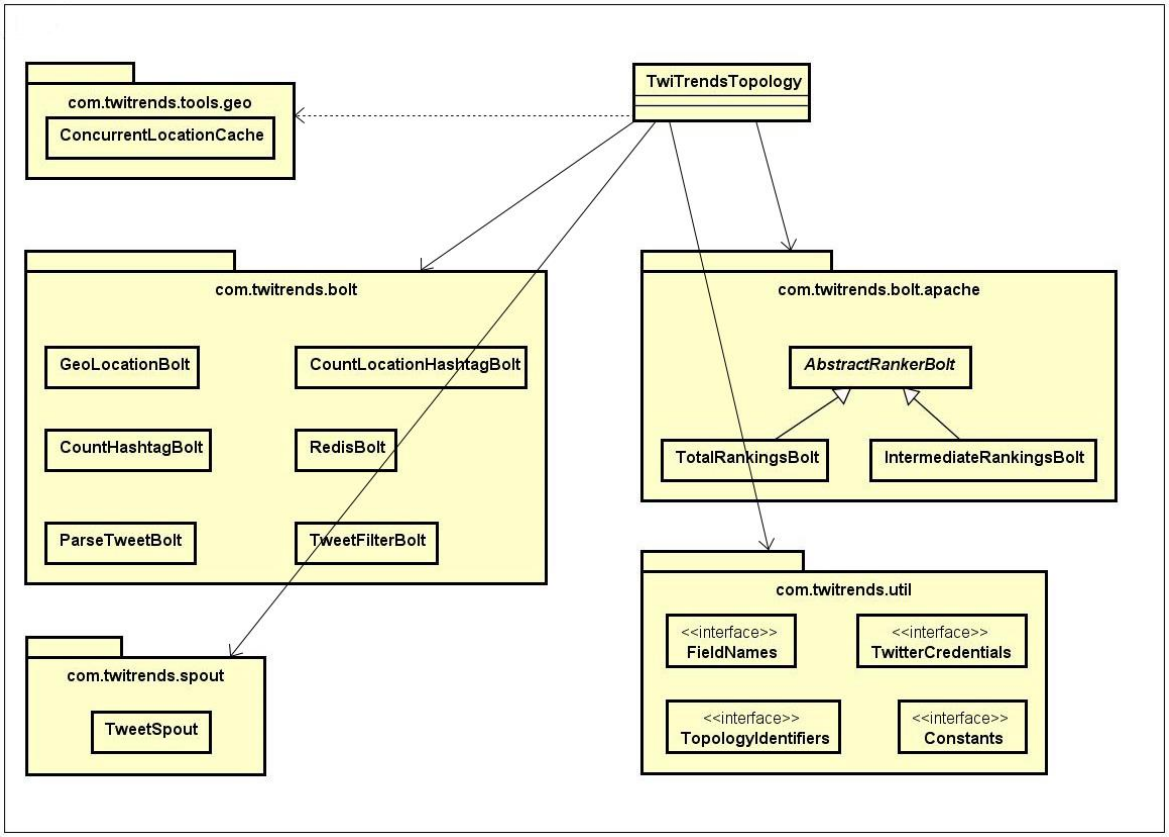


Figura 5.3: Diagrama de clase simplificată a sistemului TwiTrends

[32] https://github.com/apache/storm/tree/master/examples/storm-starter

5.4.2. Citirea, filtrarea și parsarea unui Tweet

Citirea, filtrarea și parsarea unui tweet sunt realizate prin intermediul a trei clase principale: TweetSpout, TweetFilterBolt și ParseTweetBolt. Structura celor trei clase și clasele intermediare folosite pentru îndeplinirea acestor funcționalități sunt prezentate în Figura 5.4:

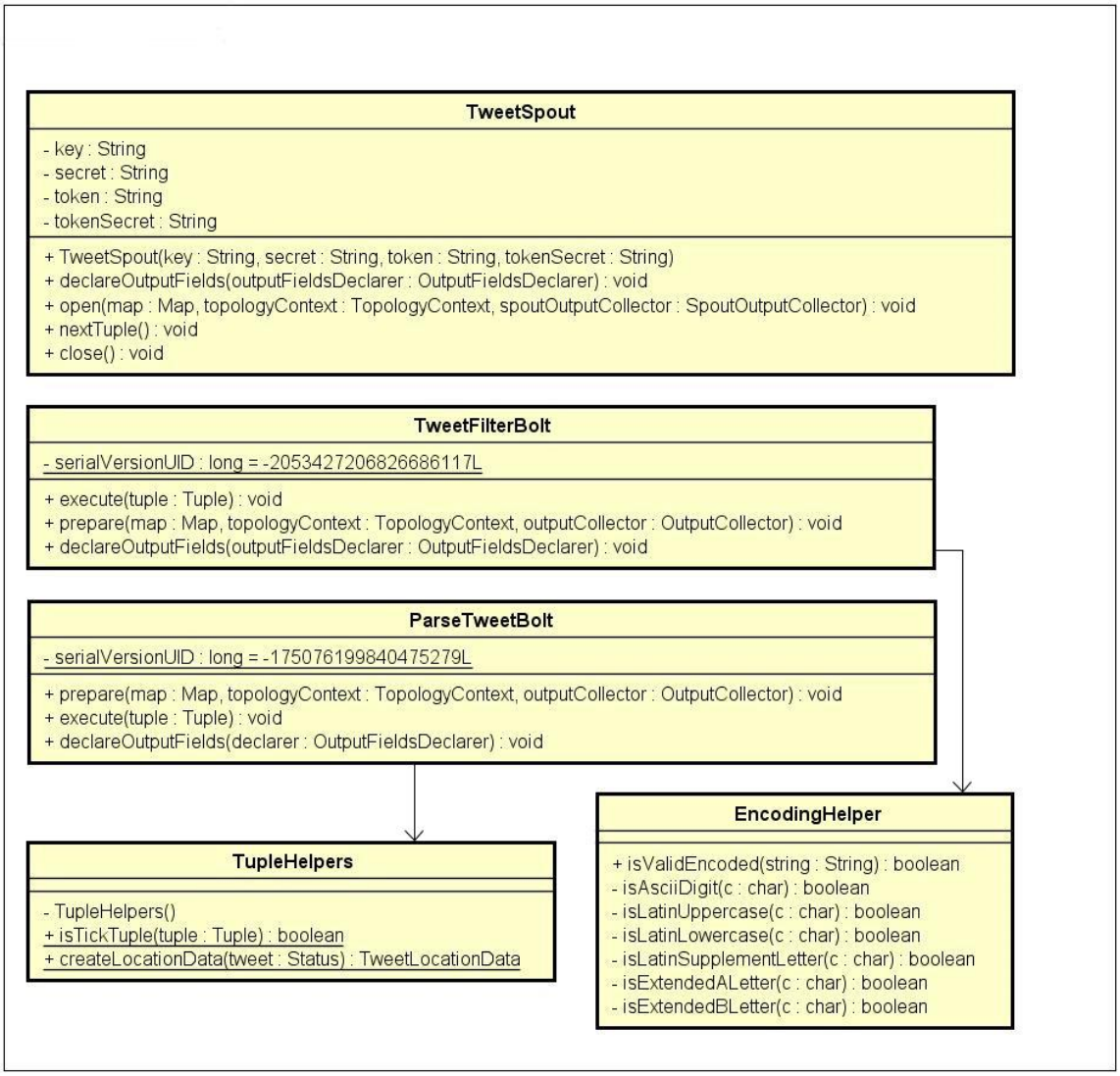


Figura 5.4: Diagrama de clase a modulului de citire, filtrare și parsare

Pentru a implementa o componentă de tip spout, Apache Storm impune extinderea unei clase de tip spout implementate în framework, de exemplu BaseRichSpout, și suprascrierea metodelor acesteia pentru a implementa o funcționalitate personalizată. După cum se poate observa în diagrama de clase, TweetSpout implementează metodele open() și close(), care specifică comportamentul componentei atunci când începe (open) și încetează (close) să emită tuple. De asemenea, trebuie specificat si metoda de procesare a unei tuple, i.e. cum va trata spout-ul o tuplă în momentul recepționării acesteia. Acest comportament este specificat prin suprascrierea metodei nextTuple().

Implementarea unui bolt se realizează analog implementării unui spout, însă clasa care trebuie extinsă este una corespunzătoare din framework-ul Apache Storm, de exemplu BaseRichBolt. Diferă de asemenea și metodele care trebuie suprascrise. În cazul unei implementări personalizate a unui BaseRichBolt, metodele suprascrise sunt prepare(), folosită pentru inițializarea unui bolt și execute(), care specifică metoda de procesare a unei tuple.
Metoda declareOutputFields() este implementată atât de către un spout cât și de către un bolt și specifică formatul tuplei de ieșire prin specificarea identificatorului fiecărui câmp al acesteia. În cazul în care un bolt nu dorește să emită un stream de ieșire, metoda este lăsată goală. De exemplu, formatul tuplei emis de către ParseTweetBolt, care conține un hashtag și un obiect cu date referitoare la locația unui tweet este specificat astfel:

```
@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields(FieldNames.HASHTAG, FieldNames.LOCATION_DATA));
}
```

Componenta topologiei care citește tweet-uri reale accesând Twitter Stream API prin intermediul librăriei twitter4j se numește TweetSpout. Spout-ul se autentifică pentru avea acces la stream-ul de date, după care citește câte un tweet și îl salvează intr-o structură de tip LinkedBlockingQueue. Această structură de date este folosită pe post de buffer, după care fiecare tweet din această coadă este emis în topologie.

TweetFilterBolt elimină tweet-urile care nu sunt relevante pentru TwiTrends. Prin tweet-uri relevante întelegem un tweet care conține cel puțin un hashtag, iar mesajul său este alcătuit doar din simboluri codificate conform standardului Unicode[33]. Validarea codificării mesajul se realizează prin intermediul metodei isValidEncoded() a clasei ajutătoare EncodingHelper(). Un caracter valid este fie o cifră, fie un caracter din alfabetul latin, fie unul din alfabetul latin suplimentat, extins A sau extins B.

ParseTweetBolt citește tuplele valide emise de către TweetFilterBolt și extrage toate hashtag-urile și locația unui tweet. Aceste date sunt impachetate într-o tuplă și înaintate în topologie.

[33] https://en.wikipedia.org/wiki/Unicode

5.4.3. Top N Hashtags

Determinarea de "trending topics" se bazează pe metoda identificării hashtag-urilor care determină acest topic prezentată în capitolul 5.2. Astfel, pentru determinarea celor mai menționate subiecte, TwiTrends validează și parsează un tweet, după care contorizează aparițiile fiecărui hashtag. Pe baza acestor contoare se realizează apoi un ranking al celor mai menționate N hashtag-uri, valoarea N fiind o constantă predefinită în aplicație.

Îm determinarea celor mai menționate N hashtag-uri sunt implicate trei bolturi ale topologiei TwiTrends: CountHashtagBolt, IntermediateRankerBolt și TotalRankerBolt. CountHashtagBolt contorizează numărul de apariții al fiecărui hashtag și emite tuple de forma (String, Long) care reprezintă textul hashtag-ului și contorul asociat. Aceste perechi sunt ordonate descrescător după numărul de apariții folosind patru instanțe de IntermediateRankerBolt. Deoarece acest bolt execută pe mai multe thread-uri, rezultatele obținute trebuie agregate pentru a obține o clasificare finală. Agregarea și emiterea celor mai menționate N hashtag-uri este realizată de TotalRankingsBolt. Figura 5.5 descrie clasele implicate în realizarea acestui clasament:

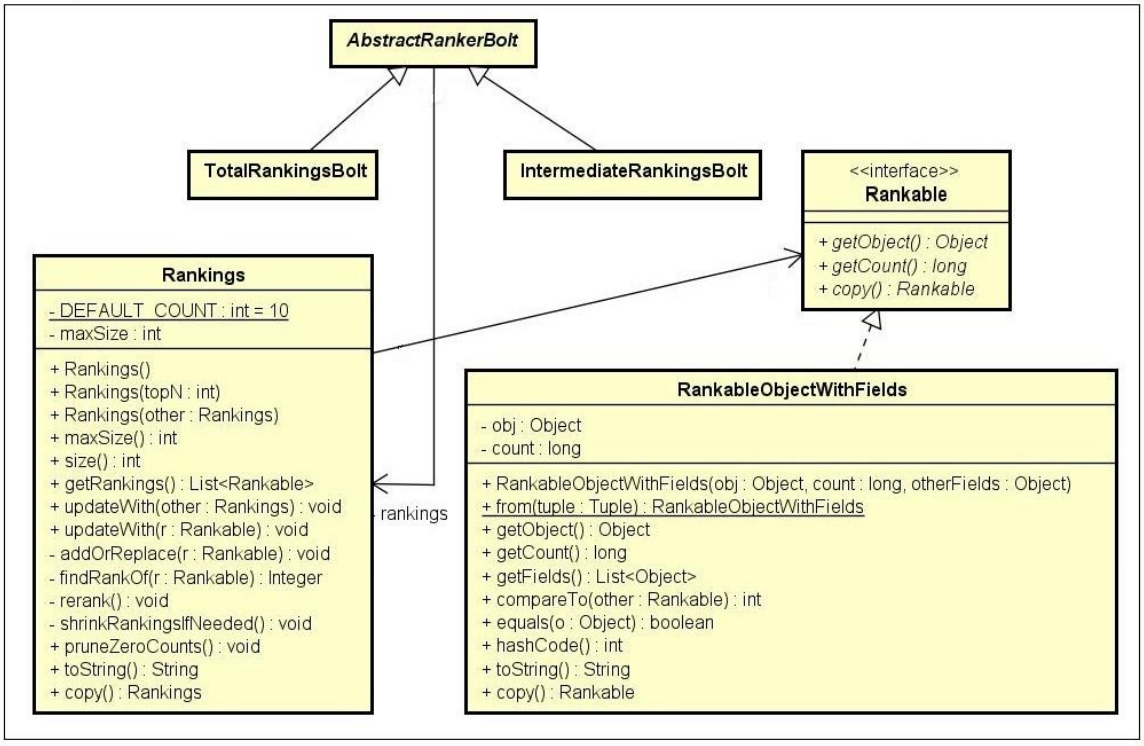


Figura 5.5: Diagrama de clase pentru determinarea celor mai menționate hashtag-uri

5.4.4. Modulul de geolocație

În cadrul analizei stream-urilor de tweet-uri, TwiTrends realizează și o clasificare la nivel de geolocație al acestora. Informația necesară este obținută din obiectele de tip Status, care reprezintă un tweet. Acesta conține coordonatele de longitudine și latitudine ale locației din care a fost postat. Pe baza acestor coordonate, TwiTrends calculează numele orașului și al țării din care hashtag-ul - respectiv hashtag-urile - conținute în mesajul acestuia. Această conversie este realizată accesând Bing Maps API, care returnează detaliile locației pe baza coordonatelor specificate în request-ul trimis. Pentru a obține o performanță mai bună, TwiTrends folosește un mecanism de caching pentru toate locațile deja calculate, evitând accesul la Bing Maps API pentru locații identice și astfel obținând un timp de execuție mult mai rapid.

Clasele modulului de calcul al geolocației sunt prezentate în Fingura 5.6. GeoLocationBolt, componenta care este responsabilă pentru emiterea tuplelor care conțin informația referitoare la locația unui hashtag accesează memoria cache pentru a obține aceste date. În cazul în care locația este prezentă în memorie, ea este citită și emisă alături de hashtag-ul corespunzător acesteia. În caz contrar, numele orașului și al țării sunt calculate folosind clasa ajutătoare BingMapsHelper, noua locație este introdusă în cache și apoi returnată către bolt.

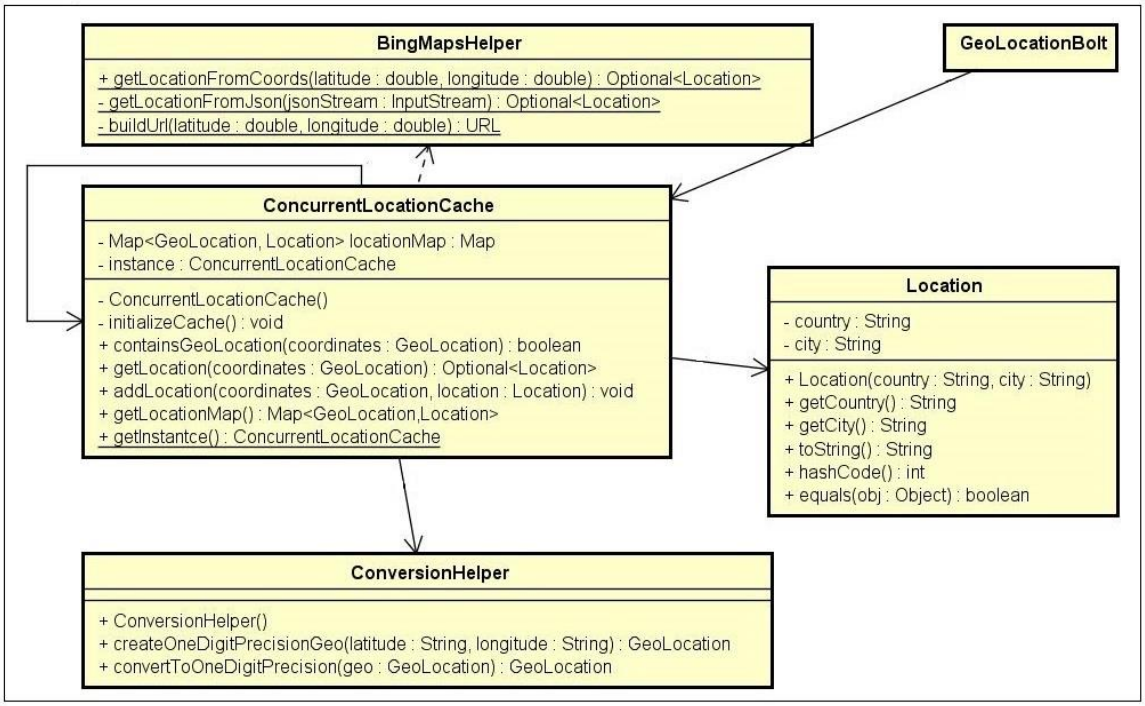


Figura 5.6: Diagrama de clase a modulului de geolocație

Clasificarea unui hashtag la nivel de geolocație reprezintă în cadrul TwiTrends cea mai costisitoare operație din punct de vedere al timpului de execuție. În cazul în care toate locațiile ar fi calculate folosind Bing Maps API, latența ar crește substanțial. Folosind o memorie cache pentru salvarea valorilor deja calculate imbunătățesc timpul de execuție considerabil. Mai mult, inițializarea memoriei înainte de executarea propriu-zisă a topologiei aduc o performanță mult superioară cazurilor menționate anterior. Figura 5.7 prezintă sub forma unui grafic timpii de execuție necesari obținerii locaților pentru un anumit număr de request-uri, folosind cele trei metode de calcul: acces direct la API, folosirea unei memorii cache, folosirea unei memorii cache inițializate:

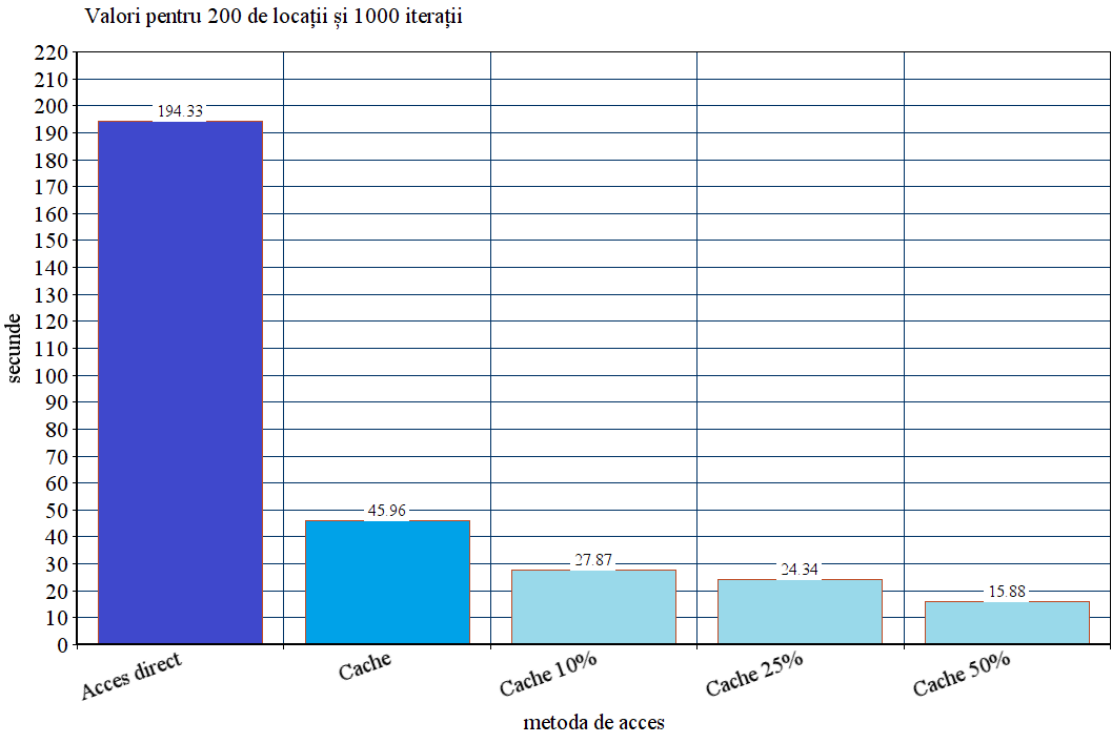


Figura 5.7: Perfomanța calculării geolocației

Conform figurii 5.7, timpul necesar calculării unei locație folosind memorie chache este mult sub cel folosind doar accesul direct la API. Pentru această analiză comparavită de performanțe au fost folosite 200 de perechi (longitudine, latitudine) diferite, iar calcularea locației acestora a fost realizată alegând de 1000 de ori o pereche aleatoare din această mulțime. De asemenea, inițializarea memoriei cache, cu 20, 50 respectiv 100 de locații predefinite a fost

realizată prin alegerea unui număr corespunzător de locații aleatoare din mulțimea de perechi calculate în avans.

Pentru a obține un număr cât mai mic de miss-uri în memoria cache este importantă alegerea datelor de inițializare. TwiTrends folosește un fișier în format .csv pentru a citi coordonatele, numele orașului și al tării inițiale pe care îl folosește în inițializarea memoriei cache[34].

Precizia luată în caclcul la nivelul cifrelor zecimale ale coordonatelor de longitudine și latitudine influențează de asemenea performanța modulului de calcul a geolocației. Numărul cifrelor zecimale luate în considerare este direct proporțional cu rata de miss în memoria cache. Tabelul 5.5 reprezintă relația dintre precizia coordonatelor și distanța în km echivalentă:

decimal places	decimal degrees	DMS	qualitative scale that can be identified	N/S or E/W at equator	E/W at 23N/S	E/W at 45N/S	E/W at 67N/S
0	1.0	1° 00′ 0″	country or large region	111.32 km	102.47 km	78.71 km	43.496 km
1	0.1	0° 06′ 0″	large city or district	11.132 km	10.247 km	7.871 km	4.3496 km
2	0.01	0° 00′ 36″	town or village	1.1132 km	1.0247 km	787.1 m	434.96 m
3	0.001	0° 00′ 3.6″	neighborhood, street	111.32 m	102.47 m	78.71 m	43.496 m

Tabelul 5.5: Precizia coordontelor raportată la distanța în km[35]

Pentru TwiTrends, precizia luată în calcul este la nivel de 0.05 grade decimale. Conform tabelului, locația unui hashtag este calculată cu o eroare maximă de aproximativ 5.55 km la ecuator, care scade până la 2.17 km. Această conversie este realizată de către clasa ConversionHelper prezentată în figura 5.6.

Aplicând această strategie de inițializare a memoriei cache, numărul de locații inițiale este 144982. Rata de miss obținută prin rularea topologiei TwiTrends cu patru instanțe la nivel de GeoLocationBolt pe baza unor tweet-uri aleatoare citite în timp real și calculul a 440 geolocații este prezentată în tabelul 5.6:

Nr. thread	Nr. accesări	Miss	Hit	Hit Ratio	Miss Ratio
15	110	109	1	0.99	0.01
21	140	136	4	0.97	0.03
31	90	87	3	0.96	0.04
35	100	99	1	0.99	0.01

Tabelul 5.6: Rata de miss folosind memoria cache inițializată

[34] http://dev.maxmind.com/geoip/legacy/geolite/
[35] https://en.wikipedia.org/wiki/Decimal_degrees

5.5. Modelul de execuție al sistemului

Modelul de execuție al topologiei TwiTrends poate fi rezumat la pașii necesari procesării unui singur tweet. Pentru fiecare tweet, pași realizați pentru determinarea celor mai discutate subiecte din rețeaua socială Twitter și clasificarea lor la nivel de geolocație sunt aceeași. In final, rezultatele obținute în urma analizei sunt agregate și clasificate conform modelului "Top N hashtags" prezentat în secțiunea 5.4.3. Figura 5.8 prezintă modelul de execuție al sistemului TwiTrends:

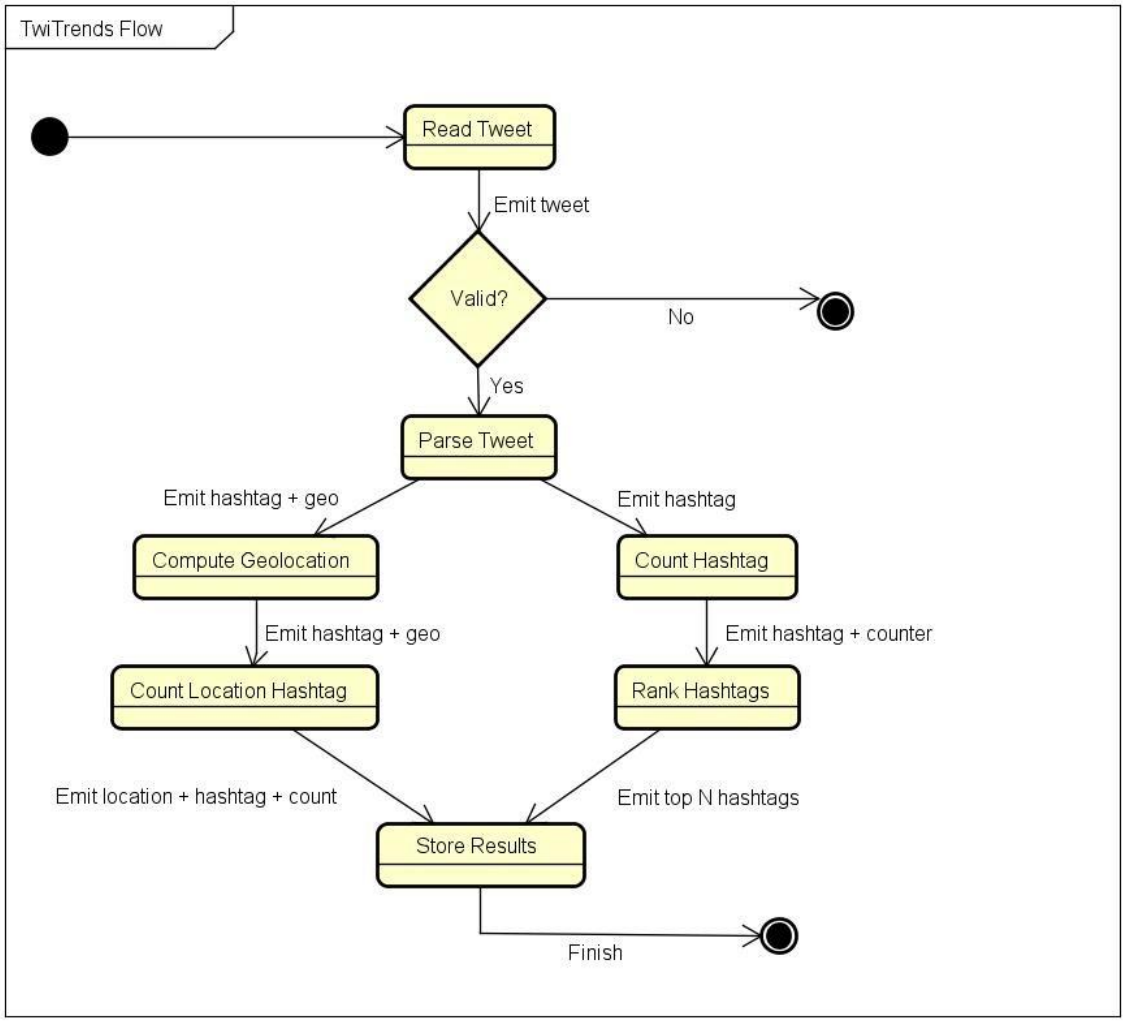


Figura 5.8: Modelul de execuție al sistemului TwiTrends

În prima fază, tweet-ul este citit de către aplicație și apoi emis mai departe în topologie după care are loc validarea acestuia. Un tweet valid reprezintă un tweet a cărui mesaj este codificat conform validării realizate de topologia TwiTrends, prezentată în secțiunea 5.4.2 și conține cel puțin un hashtag. Un tweet al cărui conținut nu corespunde acestor criterii este ignorat de topologie, iar procesarea acestuia se încheie.

Un tweet valid însă este emis mai departe sub forma unei tuple în cadrul unui stream și ajunge în starea de parsare. Parsarea unui tweet presupune extragerea tuturor hashtag-urilor din mesajul acestuia și a informațiilor referitoare la locația tweet-ului emis. Este de menționat faptul că în urma validării, prezența a cel puțin unui hashtag este garantată, însă a informației referitoare la geolocație nu. Fie că aceasta este prezentă, fie că nu, este emisă o tuplă pentru fiecare hashtag, acompaniată de datele locației acestuia.

După parsarea tweet-ului, hashtag-urile acestuia sunt procesate în două stări paralele: una pentru determinarea celor mai discutate subiecte și una pentru clasificarea hashtag-ului la nivel de geolocație. Pentru determinarea de trending topics, hashtag-ului conținut in tweet-ul citit îi este atribuit un contor care inițial are valoarea 1. În cazul în care TwiTrends a mai întâlnit acest hashtag, contorul asociat acestuia este incrementat, iar apoi perechea (hashtag, contor) este emisă pentru clasificarea propriu-zisă. În cazul raportării hashtag-ului la nivel de geolocație, informația referitoare la locația acestuia este extrasă din tuplă, iar orașul și țara din care acesta a fost emis sunt calculate. În cazul în care informația extrasă din tuplă nu este prezentă sau nu este suficientă pentru a determina o locație concretă, TwiTrends consideră că acest hashtag provine dintr-o locație necunoscută.

Clasificarea unui hashtag are loc după asocierea unui contor acestuia. Fiecare dintre acestea este recepționat din starea anterioară, cea de contorizare, iar rezultatele sunt agregate celor obținute prin analiză până în acest moment.

Analog cu contorizarea hashtag-urilor, raportarea acestora la o locație are loc prin asocierea unui contor fiecărui hashtag, însă în funție de locația din care acesta provine. Chiar dacă un hashtag a mai fost menționat, dacă acesta provine dintr-o altă țară, respectiv oraș, îi va fi alocat un contor nou, inițializat cu valoarea 1. În cazul în care a fost analizat deja pentru același oraș si aceeași valoare a hashtag-ului, contorul existent este incrementat.

Datele rezultate în stările “Count Location Hashtag” și “Rank Hashtag” sunt agregate în starea terminală. Dacă un hashtag a fost duplicat în starea de parsare, rezultatele celor două procesări alternative sunt grupate în acest monent.

După ce un tweet a fost citit, validat, parsat, procesat și stocat, execuția sa s-a încheiat, iar acesta ajunge în starea finală. Modelul de execuție este unul continuu, astfel încât mai multe tweet-uri se află simultan în execuție, iar starea în care un tweet se află în cadrul flow-ului prezentat nu determină starea unui alt tweet, atata timp cât nu cauzează întârzierea procesării al acestuia, acesta fiind izolat de restul datelor aflate în procesare.

5.6. TwiTrend într-un cluster Storm

Cu toate că framework-ul Apache Storm oferă posibilitatea execuției unei topologii într-un cluster simulat, acesta este dedicat doar dezvoltării și testării aplicației. Pentru a putea realiza scalarea sistemului, execuția masiv paralelă a fiecărei componente, împreună cu un volum de date de ieșire corespunzător procesării big-data și prin urmare o latență redusă, caracteristici specifice unui sistem de procesare în timp real, Apache Storm folosește execuția topologiei în cluster.

Pentru a putea executa o topologie într-un cluster este necesară instalarea și configurarea componentelor arhitecturii acestora, prezentate în capitolul 4.2.3:

- Server ZooKeeper
- Nimbus
- Supervisor
- Framework Apache Storm.

Condițiile prealabile pentru configurarea unui cluster Storm care trebuie îndeplinite se rezumă la o versiune de Java preinstalată, cea mai veche acceptă fiind 1.7, și Python 2.6.6. Configurarea clusterului a fost realizată folosind sistemul de operare Ubuntu 14.04.4[36], însă conform documentației Storm[37], compatibilitatea nu se rezumă doar la acesta.

Pentru instalarea server-ului ZooKeeper pe mașina locală este necesare descărcarea acestuia de pe pagina de web a celor de la Apache[38], dezarhivarea acestuia, urmată de configurarea sistemului. În momentul de fată, cea mai nouă versiune este 3.4.8, lansată în 20 februarie 2016. Pentru acești pași au fost folosite următoarele comenzi în cadrul terminalului linux:

```
$ cd opt/
$ tar -zxf zookeeper-3.4.6.tar.gz
$ cd zookeeper-3.4.6
$ mkdir data
$ vi conf/zoo.cfg
tickTime=2000
dataDir=/path/to/zookeeper/data
clientPort=2181
initLimit=5
syncLimit=2
```

[36] http://releases.ubuntu.com/14.04/
[37] http://storm.apache.org/releases/1.0.1/Setting-up-a-Storm-cluster.html
[38] http://zookeeper.apache.org/releases.html

Componentele Nimbus și Supervisor nu necesită o instalare adițională, deoarece sunt deja incluse în framework-ul Apache Storm. Acesta trebuie downloadat[39] - analog ZooKeeper - și dezarhivat pe mașina unde va rula. Versiunea folosită pentru rularea topologiei TwiTrends este 1.0.0. Pentru instalarea framework-ul și configurarea nodurilor de tip Nimbus și Supervisor au fost folosite următoarele comenzi:

```
$ cd opt/
$ tar -zxf apache-storm-0.9.5.tar.gz
$ cd apache-storm-0.9.5
$ mkdir data
$ vi conf/storm.yaml
storm.zookeeper.servers:
 - "localhost"
storm.local.dir: “/path/to/storm/data(any path)”
nimbus.host: "localhost"
supervisor.slots.ports:
 - 6700
 - 6701
 - 6702
 - 6703
```

După finalizarea instalării, framework-ul poate fi rulat. Din cinci terminale diferite vor fi pornite componentele acestuia, iar topologia va fi submisă, folosind următoarele comenzi pentru fiecare:

ZooKeeper	bin/zkServer.sh start
Nimbus	bin/storm nimbus
Supervisor	bin/storm supervisor
Storm UI	bin/storm UI
Submitere topologie	storm jar cale_fișier_jar clasă_main nume_topologie

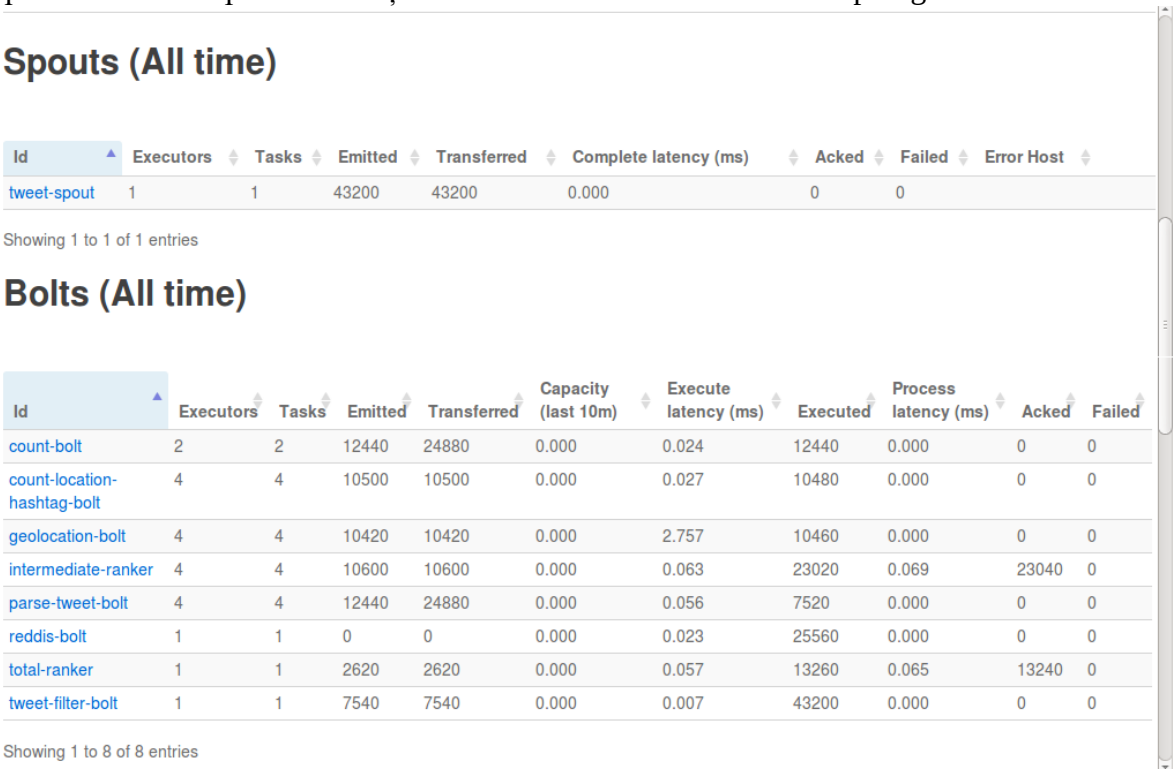
Tabelul 5.7: Comenzile folosite pentru rularea unei aplicații Storm

[39] http://storm.apache.org/downloads.html

Pentru rularea tolopogiei TwiTrends versiunea SNAPSHOT 0.0.1, comanda folosită este următoarea:

```
storm jar twi-trends-topology-0.0.1-SNAPSHOT-jar-with-dependencies
com.twitrends.RunTopology TwiTrends
```

Componenta Storm UI reprezintă o interfață grafică pentru vizualizarea datelor referitoare la execuția topologiei. Aceasta poate fi accesată la adresa http//localhost:8080. Figura 5.9 prezintă un exemplu al interfeței Storm UI cu date referitoare la topologia TwiTrends:



Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host
tweet-spout	1	1	43200	43200	0.000	0	0	

Showing 1 to 1 of 1 entries

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed
count-bolt	2	2	12440	24880	0.000	0.024	12440	0.000	0	0
count-location-hashtag-bolt	4	4	10500	10500	0.000	0.027	10480	0.000	0	0
geolocation-bolt	4	4	10420	10420	0.000	2.757	10460	0.000	0	0
intermediate-ranker	4	4	10600	10600	0.000	0.063	23020	0.069	23040	0
parse-tweet-bolt	4	4	12440	24880	0.000	0.056	7520	0.000	0	0
reddis-bolt	1	1	0	0	0.000	0.023	25560	0.000	0	0
total-ranker	1	1	2620	2620	0.000	0.057	13260	0.065	13240	0
tweet-filter-bolt	1	1	7540	7540	0.000	0.007	43200	0.000	0	0

Showing 1 to 8 of 8 entries

Figura 5.9: Vizualizarea topologiei TwiTrends folosind Storm UI

Interfața grafică prezintă toate elementele topologiei, spouts și bolts și oferă valori referitoare la execuția acestora. De exemplu, se poate observa că *parse-tweet-bolt* este executat pe 4 fire de execuție (tasks), a evmis 12440 tuple cu o latență de executare de 0.056ms. Valorile prezentate sunt raportate la întregul timp de execuție al topologiei.

5.7. TwiTrends-Heron

Limitările framework-ului Apache Storm și îmbunătățirile aduse de Heron prezentate în capitolul 4.3 au dus la migrarea topologiei TwiTrends pe arhitectura Heron, topologie intitulată TwiTrends-Heron.

Procesul de migrare a aplicației este unul relativ simplu, având în vedere că noul framework este compatibil cu Apache Storm. Pentru transpunerea topologiei TwiTrends pe noua arhitectură, schimbările necesare au fost următoarele[40]:

- Îndepărtarea dependențelor pentru Apache Storm.
- Îndepărtarea dependențelor pentru plugin-ul Clojure
- Adăugarea dependențelor pentru API-ul Heron[41]
- Adăugarea dependențelor pentru Heron-Storm[42]

Instalarea cluster-ului Heron este mai simplă decăt cea pentru Apache Storm, deoarece nu necesită configurarea fiecărei componente în parte, ci doar rularea scripturilor oferit de cei de la Twitter[43], corespunzător sistemului de operare folosit. La momentul de față, Heron poate fi instalat pe platforme ubuntu și mac osx. Comenzile folosite necesită adaptarea numelor scripturilor la versiunea folosită și sunt următoarele:

```
$ chmod +x heron-client-install-VERSION-PLATFORM.sh
$ ./heron-client-install-VERSION-PLATFORM.sh --user
Heron client installer
----------------------

Uncompressing......
Heron is now installed!

$ chmod +x heron-tools-install-VERSION-PLATFORM.sh
$ ./heron-tools-install-VERSION-PLATFORM.sh --user
Heron tools installer
---------------------

Uncompressing......
Heron Tools is now installed!
...
```

Figura 5.10: Comenzile folosite pentru instalarea framework-ului Heron[44]

[40] http://twitter.github.io/heron/docs/upgrade-storm-to-heron/
[41] http://mvnrepository.com/artifact/com.twitter.heron/heron-api/0.14.0
[42] http://mvnrepository.com/artifact/com.twitter.heron/heron-storm/0.14.0
[43] https://github.com/twitter/heron/releases
[44] http://twitter.github.io/heron/docs/getting-started/

Similar rulării unei topologi Storm, este necesară pornirea unui Heron Tracker pentru monitorizarea topologiei TwiTrends-Heron și a componentei Heron UI pentru vizualizarea datelor, urmată de submiterea și activarea topologiei. Pentru rularea componentelor au fost folosite următoarele comenzi:

```
$ heron-tracker
... Running on port: 8888
... Using config file: $HOME/.herontools/conf/heron_tracker.yaml

$ heron-ui
... Running on port: 8889
... Using tracker url: http://localhost:8888
```

Heron UI este o interfață grafică similară cu Storm UI și prezintă date referitoare la execuția topologiilor aflate în execuție. În plus, ea permite și vizualizarea grafică a componentelor acestora sub forma unui arbore și prezintă modelul alocării fiecărei instanțe în cluster. Mai mult, Heron UI permite utilizatorului să vizualizele resursele hardware pe care topologia le folosește. Strucura topologiei TwiTrends-Heron folosind Heron UI și resursele folosite sunt prezentate în Figura 5.11.

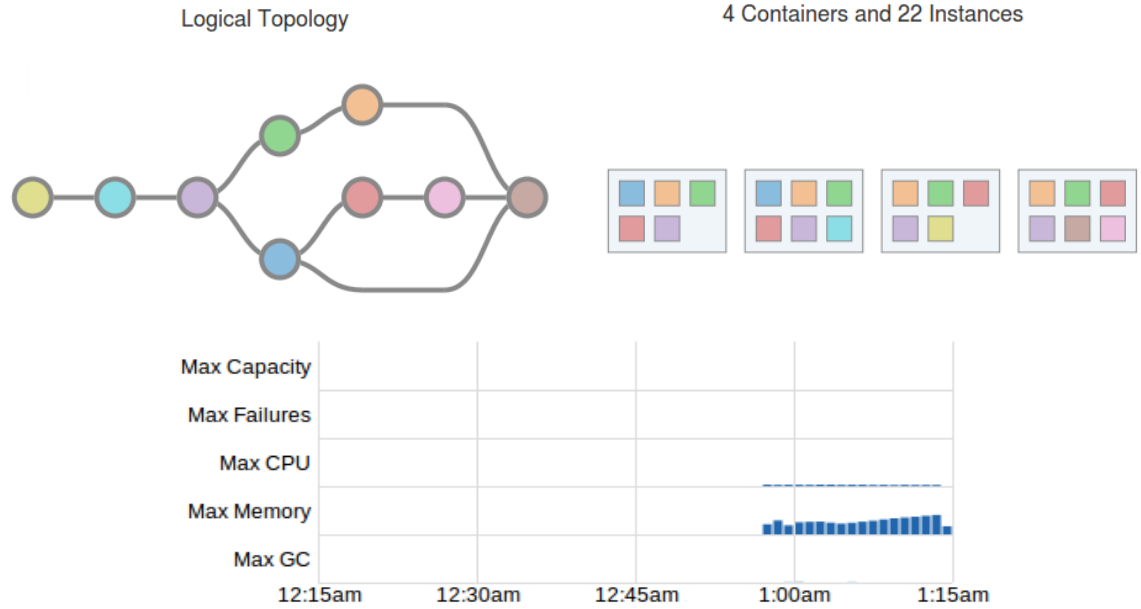


Figura 5.11: Structura tolopogiei TwiTrends-Heron, alocarea instanțelor în cluster și resursele alocate pentru execuția acestora

Capitolul 6. Testare şi Validare

Testarea topologiei TwiTrends a fost realizată din punct de vedere al funcționalității la nivel de componentă și la nivel de topologie. De asemenea, au fost realizate teste de veridicitate a datelor obținute în urma analizei și clasificării tweet-urilor.

Modulul de calcul al geolocației a fost testat cel mai extensiv, deoarece reprezintă cele mai complexe calcule realizate de topologie. Pentru a testa conversia coordonatelor la o precizie de 0.05 grade, a fost ales un set de perechi (latitudine, longitudine) pe care s-a aplicat atât conversia cât și construirea unui obiect de geolocație. Tabelele 6.1 și 6.2 prezintă aceste valori, rezultatele așteptate și cele obținute:

Nr.	Coorodonate initiale	Coordonate convertite	Rezultatul așteptat	Validare
1	12.3, 16.8	12.3, 16.8	12.3, 16.8	✓
2	12.2345, 13.3445	12.2, 13.3	12.2, 13.3	✓
3	12.222, 19.562	12.2, 19.6	12.2, 19.6	✓
4	80.96, -17.55	81.0, -17.6	81.0, -17.6	✓

Tabelul 6.1: Conversia coordonatelor

Nr.	Valori coordonate	Coordonate obținute	Rezultatul așteptat	Validare
1	“12.3”, “16.8”	12.3, 16.8	12.3, 16.8	✓
2	“12.2345”, “13.3445”	12.2, 13.3	12.2, 13.3	✓
3	“12.222”, “19.562”	12.2, 19.6	12.2, 19.6	✓
4	“80.96”, “-17.55”	81.0, -17.6	81.0, -17.6	✓

Tabelul 6.2: Instanțierea obiectelor de locație folosind conversia coordonatelor

Tot din categoria geolocație a fost testată și conversia coordonatelor în date care reprezintă locația acestora. Ca si date de intrare au fost alese 4 orașe, iar coordonatele acestora au fost obținute folosind Google Coordinates. Pe baza acestor coordonate a fost realizată conversia accesând Bing Maps API. De asemenea, a fost introdusă și o pereche (latitudine, longitudine) care nu corespunde nici unui oraș, rezultatul obținut prin accesul la API a fost “locație absentă”, care reprezintă rezultatul așteptat. Tabelul 6.3 reprezintă valorile de intrare ale coordonatelor, rezultatele obținute prin conversia acestora și cele așteptate. Toate conversiile au fost realizate corect:

Nr.	Valori coordonate	Țară obținută	Oraș obținut	Rezultatul așteptat	Validare
1	51.5074, 0.1278	United Kingdom	London	Londra	✓
2	52.5200, 13.4050	Germany	Berlin	Berlin	✓
3	55.7558, 37.6173	Russia	Moscow	Moscova	✓
4	48.8566, 2.3522	France	Paris	Paris	✓
5	100, 100	Absent	Absent	Nu există oraș	✓

Tabelul 6.3: Testarea conversiei coordonatelor folosind Bing Maps API

Metoda de clasificarea a hashtag-urilor și credibilitatea valorilor obținute au fost testate pe baza a 39.000 de tweet-uri aleatoare reale, citite folosind Twitter Streaming API. Rularea testelor a fost realizată în data de 25 iunie 2016 la ora 21:00. Evenimentele majore care caracterizau acest moment de timp au fost retragerea Marii Britanie din Uniunea Europeană (#Brexit) și Campionatul European de Fotbal Euro 2016 (#EURO2016), la ora 22:00 urmând să înceapă confruntarea dintre Irlanda de Nord și Țara Galiilor(#NIR, #WAL #WALNIR). Rezultatele obținute rulând topologia TwiTrends au fost cele așteptate, printre cele mai discutate subiecte din rețeaua socială Twitter numărându-se cele menționate. Tabelul 6.4. prezintă locul în clasament al acestor hashtag-uri, valoarea obținută de TwiTrends și numărul total de menționări ale acestuia, pe baza căruia putem spune că rezultatele obținute prin urma analizei topologiei reflectă realitatea:

Loc Clasament	**Valoare Hashtag**	**Nr. Menționări**
2	#EURO2016	629
4	#WAL	458
6	#Brexit	353
7	#WALNIR	311
8	#NIR	301

Tabelul 6.4: Cele mai menționate hashtag-uri 25 iunie 2016

Capitolul 7. Concluzii

În acest capitol for fi prezentate realizările, obiectivele care au fost atinse prin această lucrare și prin implementarea prezentată. De asemenea, vor fi prezentate și o serie de îmbunătățiri care pot fi aduse sistemului TwiTrends și enumerate posibile dezvoltări ulterioare.

7.1. Obiective atinse și rezultate obținute

Precum a fost descris în capitolul 2, scopul principal al lucrării a fost definirea conceptelor de bază din domeniul "Big Data" și a diferitelor metode de procesare. Pe baza studiului bibliografic realizat, lucrearea prezintă conceptele fundamenale necesare înțelegerii acestui domeniu vast și aflat în plină expansiune și descrie în detaliu atât metodele de procesare folosite în acest moment, cât și cele mai folosite framework-uri folosite pentru analiza big-data. Având subectul centrat în jurul procesării de timp real, lucarea a prezentat diferitele tehnologii de analiză real-time și a descris în detaliu framework-ul Apache Storm, care a dovedid o performanță bună atât în decursul ultimilor ani în industrie, cât și în cadrul aplicației TwiTrends. Mai mult, cu toate că Heron a fost relativ recent, sistemul TwiTrends-Heron a confirmat rezultatele publicate de Twitter care descriu îmbunătățirile aduse vechiului sistem.

În ceea ce privește rezultatele obținute prin testarea și validarea aplicațiilor TwiTrends și TwiTrends-Heron, acestea au fost conform obiectivelor propuse. Sistemele au dovedit a fi scalabile, tolerante la eșec, iar cu un timp de răspuns de ordinul milisecundelor putem afirma că procesarea datelor este una de timp real. Mai mult, conform obiectivelor propuse pentru acestea, rezultatele au arătat veridicitate și au reflectat realitatea lumii în care trăim, prin determinarea celor mai discutate subiecte. Considerând evenimentele principale la nivel global, care au declanșat discuții concentrate în jurul acestora, sistemul implementat a reușit să determine tematicile în care acestea se încadrează.

Conform așteptărilor, atât Apache Storm cât și Heron au dovedit aptitudinile de a procesa stream-uri de date in timp real. Lucrarea a reușit să pună în evidentă principalele caracteristici ale acestora, printe care se numără viteza de procesare, latența redusă, toleranța la eșec, scalabilitatea orizontală, ușurința de dezvoltare a aplicațiilor, reutilizabilitatea codului și nu numai.

7.2. Posibile dezvoltări ulterioare ale sistemului TwiTrends

Cu toate că rezultatele obținute au fost cele așteptate, exită o serie de imbunătățiri care pot fi aduse sistemului TwiTrends.

În ceea ce privește raportarea unui hashtag la o geolocație, TwiTrends consideră o zonă de anumită rază, dimensiunea acesteia fiind dată de precizia aleasă în rotunjirea coordonatelor. Pentru a obține o raportare mai relevantă, modulul de geolocație ar putea fi extins, astfel încât să raporteze aceste valori la anumite zone de interes:

- Toate tweet-urile postate în jurul unui oraș mare să fie raportat la acesta, ținându-se cont de suprafața pe care o acoperă pe hartă.

- De asemnea, în cazul metropolelor s-ar putea realiza o clasificare la nivel de suburbie, populația având alte caracteristici și subiecte de interes.

- În afara acestor orașe, raportarea unui tweet la o locație ar putea fi făcută prin definirea unor zone rurale, care pot fi tratate împreună. Mai mult, dacă un tweet este postat dintr-o zonă nelocuită, acesta ar putea fi un indiciu că persoana respectivă călătorește, ceea ce înseamnă că tweet-ul său nu este neaparat relevant locației în care se află, ci mai degrabă reflectă informații referitoare la destinația sa.

O altă îmbunătățire ar putea fi adusă la nivelul determinării celor mai discutate subiecte din rețeaua socială Twitter. Metoda de determinare a acestora folosind criteriul de hashtag s-a dovedit a fiind una buna, având în vedere că rezultatele obținute rulând aplicația TwiTrends a determinat cele mai discutate tematici la nivel global. Cu toate acestea, TwiTrends rezumă analiza la nivel de hashtag, nu la nivelul unui set de hashtag-uri care reprezintă același topic. Prin realizarea a mai multor rulări am ajuns la concluzia că anumite hashtag-uri precum #Euro2016, #EURO2016, #Euro16 se referă la același subiect discutat, însă ele sunt clasificare independent unul de celălalt.

Chiar dacă există o serie de dezvoltări ulterioare și de îmbunătățiri care pot fi aduse sistemului, consider că scopul acestei lucrări a fost atins. Prin dezvoltarea sistemului TwiTrends am obținut o soluție de încredere în ceea ce privește determinarea așa numitelor “trending topics”, cu toate că mariile companii lucrează la imbunătățirea metodele existente și noi abordări, care să ofere rezultate cu o acuratețe crescută, sunt dezvoltate.

Bibliografie

[1] Jonathan Stuart Ward and Adam Barker, *Undefined By Data: A Survey of Big Data Definitions*, University of St Andrews, School of Computer Science, septembrie 2013.

[2] Thibaud Chardonnens, *Big Data analytics on high velocity streams*, University of Fribourg (Switzerland) , iunie 2013.

[3] Aftabl A. Chandio, Nikos Tziritas, Cheng-Zhong Xu, *Big-Data Processing Techniques and Their Challenges in Transport Domain*, Research Gate, februarie 2015. DOI: 10.3969/j.issn.1673-5188.2015.01.007

[4] C.L. Philip Chen, Chun-Yang Zhang. *Data-intensive applications, challenges, techniques and technologies: A survey on Big Data*, Information Sciences Volume 275, august 2014. [Online]: http://dx.doi.org/10.1016/j.ins.2014.01.015

[5] Benoît Perroud. *A hybrid approach to enabling real-time queries to end-users*. Software Developer's Journal, 2013.

[6] Nathan Marz and James Warren , *Big Data Principles and best practices of scalable realtime data systems*, Manning, aprilie 2015.

[7] Boyang Peng, *Elasticity and Resource Aware Scheduling in Distributed Data Stream Processing Systems*, Master Thesis, University of Illinois at Urbana-Champaign, 2015.

[8] Karan Patel, Yash Sakaria and Chetashri Bhadane, *Real Time Data Processing Frameworks*, International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.5, No.5, septembrie 2015.

[9] Martin Illecker, *Real-time Twitter Sentiment Classification based on Apache Storm*, Master Thesis, Innsbruck, martie 2015.

[10] Bc. Dávid Katuščák, *Dynamic Processing of Event Streams Using Java Tools*, Master's thesis, Brno, 2015.

[11] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, Siddarth Taneja, *Twitter Heron: Stream Processing at Scale*, in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, mai 2015.
[Online]: http://dx.doi.org/10.1145/2723372.2742788

[12] Arkaitz Zubiaga, Damiano Spina, Raquel Martinez, Victor Fresno, *Real-Time Classification of Twitter Trends*, Journal of the American Society for Information Science and Technology, martie 2014.
[Online]: http://arxiv.org/abs/1403.1451v1

[13] Doug Laney, *3D Data Management: Controlling Data Volume, Veocity, and Variety*, META Group, februarie 2001.
[Online]: https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

Anexa 1 – Lista Figurilor, a Tabelelor și a Formulelor din Lucrare

Anexa 2 – Glosar de Abrevieri

OLAP	Online Analytical Processing
I/O	Input/Output
DAG	Directed Acyclic Graph
JVM	Java Virtual Machine
JAR	Java Archive
API	Application programming interface
IDE	Integrated Development Environment
POM	Project Object Model
REST	Representational State Transfer
URL	Uniform Resource Identifier
JSON	JavaScript Object Notation
FIFO	First In, First Out
ID	Identifier