PROCESAMIENTO DE DATOS DINÁMICOS EN FORMATO RDF CON EL API GENÉRICO DE APACHE FLINK

DIAZ RIASCOS ESTEBAN DANILO

PROGRAMA DE INGENIERÍA DE SISTEMAS

DEPARTAMENTO DE SISTEMAS

FACULTAD DE INGENIERÍA

UNIVERSIDAD DE NARIÑO

FEBRERO, 2022

PROCESAMIENTO DE DATOS DINÁMICOS EN FORMATO RDF CON EL API GENÉRICO DE APACHE FLINK

Autor:

DIAZ RIASCOS ESTEBAN DANILO, stban94diaz@gmail.com

Informe final de trabajo de grado presentado como requisito parcial para optar al título de Ingeniero de Sistemas, en modalidad investigación

Director

Mg. GONZALO JOSÉ HERNÁNDEZ GARZÓN

Co-Director

Dr.(c). OSCAR ORLANDO CEBALLOS ARGOTE

PROGRAMA DE INGENIERÍA DE SISTEMAS

DEPARTAMENTO DE SISTEMAS

FACULTAD DE INGENIERÍA

UNIVERSIDAD DE NARIÑO

FEBRERO, 2022

Nota exclusión de responsabilidad intelectual

“Las ideas y conclusiones aportadas en este Trabajo de Grado son responsabilidad exclusiva de los autores”.

Artículo 1° del Acuerdo No. 324 de octubre 11 de 1966, emanado del Honorable Consejo Directivo de la Universidad de Nariño.

Nota de Aceptación

_____

_____

_____

_____

_____

_____

_____

Firma del director

_____

Firma del jurado evaluador

_____

Firma del jurado evaluador

San Juan de Pasto, febrero de 2022

DEDICATORIA

A mis padres, Cristina del Pilar Riascos Salazar y Gilberto Enrique Díaz Torres, a mis hermanos, Karol Valeria Díaz Riascos y Jonathan Alejandro Díaz Riascos, quienes han creído en mí siempre, que son el motor que me motiva para lograr mis metas, fomentando en mí el deseo de superación y triunfo en la vida.

A mi amigo Jimmy Chávez quien fue para mí como un hermano mayor, que me ayudo, cuido y protegió, siendo incondicional y un ejemplo de vida.

Esteban Danilo Díaz Riascos

AGRADECIMIENTOS

Agradezco a mi familia, quienes incondicionalmente me han brindado todo su amor y motivación para cumplir nuestros sueños.

Mg. Gonzalo José Hernández Garzón, director y Dr.(c) Oscar Orlando Ceballos Argote, codirector del proyecto, quienes me brindaron todo el apoyo y la orientación posible para la realización y culminación de este proyecto.

RESUMEN

En el procesamiento de datos dinámicos representados en RDF, varias propuestas abordan retos computacionales que van desde su modelado, la definición de consultas en este, el procesamiento y la optimización de consultas continuas hasta la implementación de sistemas que facilitan la integración de datos dinámicos en RDF con otras fuentes de datos disponibles como datos enlazados (Linked Data). Sin embargo, la mayoría de los sistemas propuestos (Streaming SPARQL, C-SPARQL, EP-SPARQL, SPARQL stream, CQELS y CQELS Cloud) aún adolecen de problemas con la escalabilidad.

Por lo tanto, se presenta una propuesta de trabajo de grado para procesar datos dinámicos en formato RDF sobre Apache Flink, un framework de procesamiento general enmarcado en el contexto de Big Data.

Palabras clave: LSD, Formato, RDF, Datos Dinámicos, Procesamiento, Big Data, Apache Flink, Swapping, Framework, Escalabilidad, Triples.

ABSTRACT

In the processing of data streams represented in RDF, several proposals address computational challenges ranging from data stream modeling, query modeling, continuous query processing and optimization to the implementation of systems that facilitate the integration of data streams in RDF with other available data sources such as Linked Data. However, most of the proposed systems (Streaming SPARQL, C-SPARQL, EP-SPARQL, SPARQL stream, CQELS, and CQELS Cloud) still suffer from scalability issues.

Therefore, a proposal for a degree work is presented to process data streams in RDF on Apache Flink, a general processing framework framed in the context of Big Data.

Keywords: LSD, Format, RDF, Stream, Processing, Big Data, Apache Flink, Swapping, Framework, Scalability, Triples.

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

GLOSARIO

APACHE FLINK: framework de procesamiento general enmarcado en el contexto de Big Data.

BIG DATA: macrodatos, datos a escala masiva.

CACHING: método de optimización que usa la memoria de la cache.

CORRECTITUD: característica de un sistema de procesamiento que evalúa la efectividad de este, por lo general esto se logra comparándolo frente a otro ya probado.

CUAD: nodo RDF que transforma la estructura de una tripleta agregándole una marca de tiempo, su nombre se debe a que en este caso cuenta con cuatro componentes, sujeto (S), predicado (P), objeto (O) y tiempo (T).

DATA ENCODING: método de optimización que realiza la codificación de sujeto, predicado y objeto, con el fin de reducir el tamaño de los datos en procesamiento, esto hace necesaria la creación de un diccionario durante el procesamiento y su respectiva decodificación al final de este.

EVENT PROCESSING: tipo de procesamiento que permite realizar control por eventos y define de esta forma la apertura o finalización de las ventanas gracias a la semántica de las consultas.

FRAMEWORK: marco de trabajo, que define técnicas, métodos y tecnologías, enmarcadas alrededor de un tema en específico.

INDEXING: método de optimización bastante usado en sistemas SQL, que usa índices para identificar rápidamente datos con respecto a su posición.

LINKED OPEN DATA: estándar de datos abiertos enlazados definido por la W3C,

LSD: Datos Dinámicos Enlazados o Linked Stream Data.

MAPPER: software encargado de analizar la sintaxis de una consulta en RDF y transformarla en un Programa expresado en la sintaxis del lenguaje Java usando la librería Data Stream API de Apache Flink.

MULTIWAY JOIN: tipo de operador que optimiza la unión de múltiples streams, ventanas o resultados a la vez.

RDF: framework de descripción o representación de recursos.

RUNNER: software encargado de tomar el programa resultante del mapper, definir el ambiente basado en la librería de Apache Flink y la lógica para los operadores que se definan en este en Java y de esta forma sea posible su ejecución.

SLIDING WINDOWS: tipos de ventanas que son capaces de solaparse unas con otras, pueden ser de tiempo fijo o variable.

SOLUTION MAPPING: son los objetos encargados de guardar en formato string la lógica establecida para cada operador definido en el OP.

STREAM: flujo, que es dinámico y se transmite en tiempo real.

STREAM REASONING: modelo de ejecución basado en programación lógica, con definiciones en uso de eventos, con capacidad de inferencia sobre conocimiento temporal y estático.

SWAPPING: técnica de intercambio de memoria entre RAM y disco duro.

TRIPLES RDF: o tripletas, son los nodos del grafo que define la interconexión de los recursos, estableciendo así la semántica y toman su nombre ya que están conformados por los elementos sujeto (S), predicado (P) y objeto (O) .

VISITORS: métodos encargados de definir la lógica específica para cada operador de acuerdo a los argumentos que llegan, de modo que "se visita" a uno u otro y se define si es una ventana, un join u otro en particular.

INTRODUCCIÓN

En los últimos años, un creciente número de fuentes producen de manera continua streams de datos. Ejemplos de tales fuentes incluyen teléfonos móviles (acelerómetro, brújula, GPS, cámara, etc.), sistemas de monitoreo de pacientes (ritmo cardíaco, presión sanguínea, etc.), estaciones para la observación del clima (temperatura, humedad, etc.),sistemas de localización (GPS, RFID, etc.), sistemas de gestión de edificios (consumo de energía, condiciones ambientales, etc.) y de vehículos (monitoreo del motor y conducción, etc.), archivos logs y redes sociales (Twitter, Facebook, etc.), entre otras.

A diferencia del procesamiento de consultas en sistemas de bases de datos (ej., relacionales) donde los datos se almacenan para luego procesarse, los streams de datos se procesan directamente en memoria produciendo resultados de forma continua [1]. Este modelo de procesamiento trajo consigo sus propios retos de investigación en áreas como el modelado de datos para la representación de streams, el procesamiento de consultas continuas y la optimización [2], entre otras. Estos retos se han abordado desde diferentes enfoques y se ha desarrollado un importante número de sistemas para el procesamiento de streams de datos bajo el modelo Relacional [1, 3-11].

En particular, a nivel semántico, el W3C Semantic Sensor Network Incubator Group [12] tienen por objetivo representar sensores y stream de datos siguiendo los principios de Linked Data [13], un concepto que se conoce como Linked Stream Data (LSD) [14]. Actualmente, existen varias propuestas con enfoques diferentes que abordan el procesamiento de stream de datos en RDF [15], como por ejemplo Streaming SPARQL [16], C-SPARQL [17], EP-SPARQL [18], SPARQL stream [19], Instans [20], CQELS [21] y CQELS Cloud [22].

Por otra parte, la cantidad de información generada por la sociedad crece en volumen (ej., de terabytes a petabytes o aún mayor), en variedad (ej., datos estructurados y no estructurados) y velocidad [23]. Esta enorme generación de datos junto con la adopción de nuevas estrategias, han provocado la aparición de una nueva era de la gestión de datos, comúnmente conocida como Big

Data. En términos generales, Big Data se puede definir como la aparición de nuevos conjuntos de datos de volumen masivo que cambian rápidamente, son complejos y exceden el alcance de las capacidades analíticas de los entornos de hardware y de software de uso común para el procesamiento de datos [24, 25]. En el contexto de Big Data se han propuesto varios sistemas con enfoques diferentes que abordan el procesamiento de datos a escala masiva. Ejemplos de estos sistemas son Map Reduce [26] y su implementación Hadoop [27], Flume [28], Storm [29], Spark [30, 31], Flink [32] y Cloud Dataflow [33], entre otros.

En la presente investigación se aborda el problema que las propuestas hechas para el procesamiento de stream de datos en RDF aún presentan inconvenientes con la escalabilidad. Por lo tanto, se propone adaptar modelos de procesamiento y tecnologías relacionadas a Big Data con el fin de procesar streams de datos en RDF y así superar los problemas de escalabilidad de los sistemas actuales

.

I. .PROBLEMA DE INVESTIGACIÓN

A. Modalidad

El presente trabajo de grado se enmarca dentro de la modalidad de **Trabajo de Investigación**.

B. Línea de investigación

El presente trabajo de grado se enmarca dentro la línea de investigación **Línea de Software y Manejo de Información** abarcando las temáticas relacionadas con Fundamentos de Programación, Programación, Estructuras de Información y Bases de Datos.

C. Planteamiento del problema

RDF Stream Processing Community Group [34] es un grupo que hace parte de la W3C [35] y tiene por objetivo definir un modelo común para producir, transmitir y consultar continuamente RDF Stream, incluyendo extensiones a RDF y SPARQL [36] tanto para representar streams de datos como su semántica. En este grupo, al igual que en [37, 38], se identifican algunos problemas que se presentan en el procesamiento de streams de datos en RDF, principalmente, aquellos relacionados con la heterogeneidad entre los modelos para representar streams de datos en RDF, los modelos de consulta, la semántica operacional, el procesamiento y la optimización de consultas continuas, entre otros.

En particular, en el procesamiento de stream de datos en RDF, un aspecto importante de los sistemas propuestos [16-20, 37] es la escalabilidad. A nivel general, la escalabilidad se relaciona con la capacidad de un sistema para procesar consultas sobre grandes volúmenes de datos, siempre crecientes [38]. La escalabilidad en los sistemas propuestos se ve afectada por varios aspectos como el número de consultas continuas concurrentes, el tamaño de los datos, el número de streams de datos, las técnicas de optimización de consultas, la tasa de llegada de los streams de datos y la arquitectura, entre otros [37].

En [38]se reportan resultados de escalabilidad de algunos sistemas y concluye que presentan problemas en su desempeño cuando alcanzan ciertos umbrales en el número de consultas

concurrentes y el tamaño de los datos. En [37] se reporta que algunos sistemas (ej., ETALIS y C-SPARQL) solo pueden procesar conjuntos de datos en RDF de tamaño pequeño, de hasta 1 millón de tripletas; con tamaños de 2 millones fallan o no responden. Esto se debe en parte a que algunos sistemas como C-SPARQL, SPARQLstream, EP-SPARQL e Instans implementan una arquitectura con un enfoque de caja negra; es decir, delegan el procesamiento a otros sistemas (ej., procesadores de streams/eventos o procesadores SPARQL), lo cual impacta seriamente la escalabilidad debido a que se requieren procesos adicionales para transformación de datos y traslación de consultas según los requerimientos del sistema subyacente [22].

Por su parte, CQELS implementa una arquitectura de caja blanca, proporcionando operadores y técnicas para el procesamiento de streams de datos en RDF nativamente, presentando un mejor desempeño y logrando escalar hasta 10 millones de tripletas para 1000 consultas concurrentes por segundo [21]. A pesar de esto, CQELS no escala bien cuando se incrementa el número de consultas y el tamaño de los datos [38]. Para mejorar estas limitaciones, se desarrolló el sistema CQELS Cloud, el cual adapta operadores (ej., sliding Windows y multiway join) y técnicas de optimización (data encoding y caching and indexing) existentes a un ambiente de ejecución en la Nube. En [22] se reporta que CQELS Cloud puede escalar hasta 100.000 tripletas por segundo para 10.000 consultas concurrentes sobre un cluster de 32 nodos tipo medium en la plataforma AWS EC2. Sin embargo, estos umbrales no son el límite final si se consideran conjuntos de datos disponibles como RDF data streams y static RDF datasets de gran volumen, además que actualmente no se cuenta con documentación detallada de dicha implementación ya que según la información disponible se realizaron ajustes a nivel de maquina en AWS EC2, que impide su replicación, aún más a nivel de pregrado debido a su elevado costo, por lo cual es importante hacer uso de tecnologías como Apache Flink, que permitan realizar procesamiento de stream de datos.

A partir de lo anterior, el problema objeto de estudio se centra en que las propuestas formuladas para el procesamiento de stream de datos en RDF aún presentan inconvenientes con la escalabilidad, dado que los enfoques existentes fallan cuando alcanzan determinados umbrales, así como también el tiempo y costos que generan implementaciones como CQELS Cloud debido a su complejidad.

D. Formulación del problema

Para este problema, se plantea la siguiente pregunta de investigación: ¿Cómo adaptar tecnologías para el procesamiento de datos a escala masiva (Big Data) de tal manera que sea posible el procesamiento de datos dinámicos en formato RDF?

E. Objetivos

1) Objetivo general

Desarrollar una librería en Java usando Data Stream API de Apache Flink que permita consultar datos dinámicos representados en formato RDF.

2) Objetivos específicos

Desarrollar un mapper para transformar una consulta expresada en un lenguaje para streams de datos en RDF a un Programa expresado en la sintaxis del Data Stream API de Apache Flink.

Desarrollar un runner para ejecutar un Programa expresado en la sintaxis del Data Stream API de Apache Flink en un cluster local.

Configurar un ambiente de pruebas basado en un benchmark existente para evaluar, de manera empírica, los resultados de las consultas.

F. Justificación

Este trabajo de grado está enmarcado dentro de uno de los objetivos específicos de la tesis doctoral titulada “PROCESAMIENTO DE CONSULTAS HÍBRIDAS SOBRE STATIC RDF DATASET Y STREAM DE DATOS EN RDF” propuesta por Dr(C). Oscar Orlando Ceballos Argote, dirigido por los doctores María Constanza Pabón de la Pontificia Universidad Javeriana Cali, Andrés Mauricio Castillo Robles de la Universidad el Valle y Oscar Corcho de la Universidad Politécnica de Madrid. La tesis doctoral fue presentada ante el Comité de Investigaciones de la Universidad del Valle y aprobada por los doctores Jean Paul Calbimonte Pérez del Institut Informatique de Gestion y Juan Francisco Sequeda de University of Texas at Austin, Texas.

La tesis doctoral plantea como hipótesis que es posible evaluar en tiempo real consultas híbridas sobre combinaciones de grandes volúmenes de datos disponibles como RDF data stream y static

RDF datasets, mediante el uso de operadores lógicos especializados para este tipo y formato de datos y su implementación con técnicas de procesamiento a gran escala, superando de este modo las barreras de escalabilidad de los sistemas para el procesamiento de stream de datos en RDF actuales, manteniendo propiedades de correctitud y completitud. El desarrollo del trabajo de grado presenta un proceso con una curva de aprendizaje alta, pues se hace necesario estudiar modelos de procesamiento y tecnologías enmarcadas en el contexto de dos áreas de conocimiento diferentes como lo son, por una parte, el procesamiento de consultas SPARQL sobre static datasets y la extensión de SPARQL para procesar stream de datos en RDF; por otra, las tecnologías asociadas al procesamiento de grandes volúmenes de datos o Big Data.

G. Viabilidad

Por lo dicho en la justificación anteriormente planteada se puede asegurar que la presente investigación es viable, ya que está enmarcada dentro de una tesis de doctoral aprobada, por otra parte la gran importancia y atención que ha venido adquiriendo el desarrollo de tecnologías para Big Data tanto en el sector público como privado hace que esta investigación a pesar de ser innovadora cuente con antecedentes que serán importantes referentes de partida y de comparación como se evidenciara en este documento.

H. Alcance y delimitación

El trabajo que se propone estará delimitado por las siguientes consideraciones:

Las consultas en CQELS y los stream de datos en RDF estarán determinados por el benchmark seleccionado. Este proyecto se considera una aproximación al procesamiento de stream de datos en RDF sobre tecnologías de Big Data, por tanto, se desarrollará la librería con base a un subconjunto de operadores del Álgebra de CQELS, como, por ejemplo: Window, Join y Optional.

Las pruebas de correctitud de los resultados se realizarán de forma empírica, no se realizarán pruebas formales.

II. MARCO TEÓRICO

A. Antecedentes

1) Tecnologías

A partir del desarrollo y uso generalizado de tecnologías para la computación paralelo distribuida, se ha propuesto un número importante de sistemas con enfoques diferentes que abordan el procesamiento de datos a escala masiva. Ejemplos de estos sistemas son MapReduce [26] y su implementación Hadoop, Flume [28], Storm [29], Spark [30, 31], Flink [32] y Cloud [33], entre otros.

- **Hadoop** es un sistema eficiente, escalable y tolerante a fallos para el almacenamiento distribuido de grandes cantidades de datos. Plataformas como Hortonworks [39] y Claudera [40] incluyen a MapReduce, Hadoop Distributed File System - HDFS, HBase [41], Hive [42], Zookeeper [43], Avro [44] y Pig [45], los cuales hacen parte del ecosistema de Hadoop. MapReduce, en particular, es un modelo de programación diseñado para el procesamiento en paralelo de grandes volúmenes de datos y consta de dos procesos principales: map, que particiona un trabajo en varias tareas y reduce que combina el resultado de las tareas.
- **Flume** [46] es un sistema que hace parte del ecosistema de Hadoop y permite capturar, agregar y mover en tiempo real grandes volúmenes de datos provenientes de distintas fuentes (ej., eventos ligados al tráfico de red, redes sociales, mensajes de correo electrónico) hacia el sistema de almacenamiento HDFS. La arquitectura de Flume basada en agentes es simple y flexible, compuesta de tres elementos: las fuentes (sources), el canal (chanel) y los sumideros (skins). Flume puede escalar horizontalmente debido a que su arquitectura distribuida no requiere un punto de coordinación central, cada agente se ejecuta de forma independiente.
- **Storm** [47, 48] es un sistema distribuido para procesar stream de datos en tiempo real generados desde múltiples fuentes (ej., sensores, redes sociales). Storm es escalable y tolerante a fallos. Storm basa su flujo de trabajo en Directed Acyclic Graphs - DAG, normalmente, llamados topologías. Las topologías son una red de spouts y bolts, dos tipos de transformaciones que provee Storm. Un spout funciona como una fuente de stream. Por ejemplo, un spout podría conectarse a la API de Twitter y emitir un stream de tweets. Un bolt consume los stream generados por un spout,

los procesa y luego emite nuevos stream. Las topologías se caracterizan porque siempre están ejecutándose y solo se interrumpen cuando se finalizan o se genere un fallo irrecuperable. Los stream de datos en Storm se modelan como tuplas. Una tupla es una lista de valores y cada campo en la tupla puede ser un objeto de cualquier tipo. Storm puede operar en dos modos: local y distribuido. En cada modo de operación, Storm cuenta con dos tipos de nodos: el master node y los worker nodes. El master node ejecuta el demonio Nimbus, el cuales responsable de la distribución de código, la asignación de las tareas a las distintas máquinas y del seguimiento de fallos. Los worker nodes ejecutan el demonio Supervisor, el cual es encargado de recibir y ejecutar las tareas asignadas a la máquina donde está alojado. Storm también implementa Zookeeper, un sistema encargado de la coordinación entre los demonios Nimbus y Supervisors.

- **Spark** [49] es un sistema de propósito general para el procesamiento de datos a gran escala por lotes o en tiempo real. Spark se compone de los módulos Spark Core, Spark SQL, Spark Streaming, MLlib y GraphX. Spark cuenta con APIs de alto nivel disponibles en los lenguajes Python, Java, Scala, R y SQL. Spark Core contiene las funcionalidades básicas de Spark, como por ejemplo planificación de tareas, gestión de la memoria, recuperación de fallos, interacción con sistemas de almacenamiento, entre otras. Spark Core implementa Resilient Distributed Datasets - RDDs, una abstracción para el procesamiento de datos en paralelo directamente en memoria. Un RDD es una colección de elementos distribuida a través de los nodos de un cluster que pueden gestionarse en paralelo. A un RDD se puede aplicar dos tipos de operaciones, transformaciones (ej., map, filter, join) y acciones (ej., count, collect, save). Las transformaciones construyen nuevos RDD y las acciones procesan resultados sobre un RDD. Spark SQL permite procesar datos estructurados y semiestructurados a través de un tipo especial de RDD llamado Schema RDD. Cada Schema RDD es un RDD de Row objects, cada uno representa un registro. Un Schema RDD puede crearse desde una fuente externa, resultados de consultas o de RDD existentes. Spark Streaming permite el procesamiento de stream de datos a través de una abstracción llamada Discretized Streams o DStreams la cual es una secuencia de datos que llega a lo largo del tiempo. Internamente, cada DStreams se representa como una secuencia de RDDs que llega en cada momento de tiempo. Un DStreams puede crearse a partir de varias fuentes, tales como Flume, Kafka o HDFS. Spark Streaming usa una arquitectura tipo micro-batch, en la cual los stream se procesan como una serie de pequeños lotes de datos. El tamaño del intervalo de tiempo se determina por un parámetro

llamado batch interval, el cual, típicamente está entre 500 milisegundos y varios segundos. Spark también implementa las librerías MLlib y Graph. MLlib proporciona algoritmos de Machine Learning, entre ellos clasificación, regresión, agrupamiento y filtrado colaborativo. GraphX permite la gestión de grafos. Al igual que Spark Streaming y Spark SQL, GraphX extiende la abstracción RDD para permitir crear grafos dirigidos con propiedades arbitrarias unidas a cada vértice y arco. Spark está diseñado para escalar eficientemente de uno a miles de nodos. Para lograrlo, requiere de un gestor de cluster y de un sistema de almacenamiento distribuido. Para la gestión del cluster, Spark soporta Standalone Scheduler, YARN o Mesos. Para el almacenamiento distribuido, Spark interactúa con diferentes sistemas tales como HDFS, Casandra, HBase y Amazon S3.

- **Flink** [50], al igual que Hadoop y Spark, es un sistema distribuido y tolerante a fallos para el procesamiento de stream de datos a gran escala. Flink soporta principalmente el procesamiento de stream, sin embargo, también permite el procesamiento por lotes como un caso especial de stream. Flink soporta ventanas de diferentes tipos, por ejemplo, ventanas basadas en tiempo, en cuenta, en sesiones o dirigidas por datos. Flink implementa su propio gestor de memoria dentro de Máquina Virtual de Java. La arquitectura de Flink consta de tres capas: Deploy, Core y APIs and Libraries. La capa Deploy permite la gestión del sistema paralelo distribuido; para ello ofrece distintas alternativas como por ejemplo Local (ej., Single JVM, Embedded), Cluster (ej., Standalone, YARN) o Cloud (ej., GCE, EC2). La capa Core implementa un modelo de programación completamente centrado en el procesamiento de stream de datos. La capa APIs and Libraries proporciona, por una parte, un conjunto de APIs, como por ejemplo Dataset API la cual es un entorno de ejecución en el que las transformaciones se ejecutan sobre conjuntos de datos tomados de fuentes estáticas (ej., archivos o bases de datos locales) y Data Stream API, similar a la anterior, pero con la gran diferencia de que los datos son tomados desde fuentes dinámicas (ej., sockets). Por otra parte, proporciona un conjunto de librerías tales como Flink ML y Gelly las cuales contienen algoritmos iterativos para Machine Learning y Graph, respectivamente.
- **Cloud Dataflow** [51] es un modelo de programación y un servicio en la nube desarrollado por Google para el procesamiento en modo batch y en modo streaming de grandes cantidades de datos. Cloud Dataflow implementa un modelo de programación unificado por cuanto se basa parcialmente en tecnologías anteriores como Flume Java [28] y MillWheel [52]. Este modelo

ofrece diversas primitivas para la gestión de ventana y control de correctitud que pueden ser aplicadas, tanto a fuentes de datos basadas en batch como a stream. A diferencia de los anteriores sistemas, Cloud Dataflow se encarga de las tareas operacionales como gestión de recursos y optimización del desempeño, dejando a usuarios finales el desarrollo de aplicaciones en Big Data a través de una API de alto nivel en Java o Python. A nivel general, el flujo de trabajo en Cloud Dataflow se compone de cuatro fases: captura, almacenamiento, procesamiento y análisis. A lo largo de estas fases, Cloud Dataflow ofrece integración directa con aplicaciones de Google Cloud Platform (ej., Cloud Storage, Cloud Pub/Sub, Cloud Bigtable, BigQuery, Developers Console) para facilitar el procesamiento de datos y su seguimiento.

- **Benchmark,** Con el fin de determinar el grado de confiabilidad de herramientas y sistemas informáticos, en este caso los motores de procesamiento de stream datos en RDF, se suele definir estructuras de pruebas, manuales o automatizadas por software, con las cuales se define suites de evaluación con datos estandarizados, aplicadas en diversos entornos y condiciones, este concepto es conocido como benchmark y de acuerdo a esto, en este proyecto de investigación se hace uso de la herramientas más actual para el contexto planteado, la cual es Citybench, ya que esta realiza pruebas con los motores CQELS y C-SPARQL utilizando CityBench Benchmarking Suite con la cual se evaluamos los dos motores RSP con respecto a (i) latencia, (ii) consumo de memoria y (iii) integridad [38, 53, 54].

B. De la investigación

1) Procesamiento de flujos de datos

Un stream de datos es una secuencia de elementos continua, de tiempo real y ordenada [2]. A diferencia del procesamiento de consultas en sistemas de bases de datos relacionales (Fig. 1), donde los datos entrantes primero se almacenan, luego se procesan de acuerdo a los criterios de una consulta y finalmente se entrega un resultado, los stream de datos (Fig. 2) se procesan directamente en memoria produciendo resultados de forma continua. La Fig. 1 muestra una comparación de alto nivel entre el procesamiento de datos en el modelo relacional y la Fig. 2 los flujos de datos.

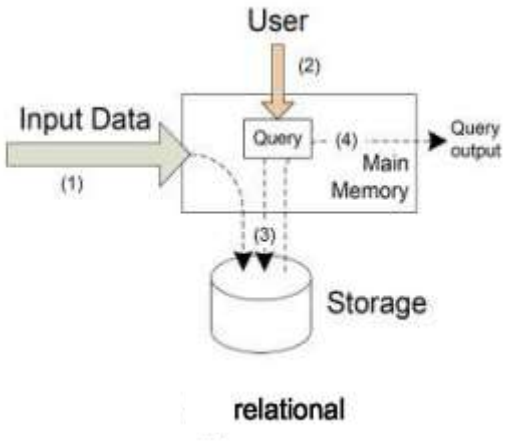

Fig. 1. Modelos de procesamiento de datos: relacional.

Fuente: Adaptada de «StreamCloud: An Elastic and Scalable Data Streaming System» IEEE Transactions on Parallel and Distributed Systems, 2012, enlace https://bit.ly/3DJQEQq

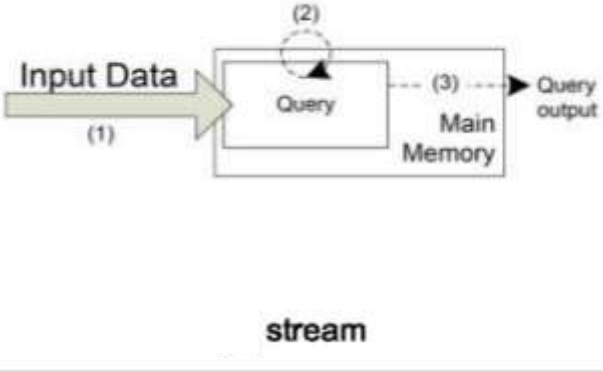

Fig. 2. Modelos de procesamiento de datos: stream.

Fuente: Adaptada de «StreamCloud: An Elastic and Scalable Data Streaming System» IEEE Transactions on Parallel and Distributed Systems, 2012, enlace https://bit.ly/3DJQEQq

El procesamiento de stream de datos trajo consigo sus propios retos de investigación en áreas como por ejemplo el modelado de datos para la representación de streams, el modelado de consultas, el procesamiento de consultas continuas y la optimización de consultas, entre otras. Estos retos se han abordado desde diferentes enfoques y se ha propuesto un importante número de sistemas para su procesamiento.

- **Procesamiento de stream de datos relacionales y eventos**

En [1, 19] se presenta un estado del arte sobre los modelos para el procesamiento para streams de datos y eventos, el procesamiento y los lenguajes de consultas continuas. De igual manera, se presenta una comparación entre los principales sistemas, clasificados en cuatro grandes grupos: (a) Data Stream Management Systems – DSMS [1, 3-11], (b) Complex Event Processor – CEP [55-64], (c) Sensor Network Query Processing Systems [65-67], y (d) Sensor Data Middleware [68-74]. Estos sistemas procesan streams de datos representados con base en modelos de datos tradicionales como el Relacional, Orientado a Objetos y XML, siendo el modelo relacional el más usado. A continuación, se muestra un ejemplo de stream de datos como una secuencia de tuplas relacionales más una marca de tiempo (Fig. 3) y un modelo para la generación de eventos complejos (Fig. 4).

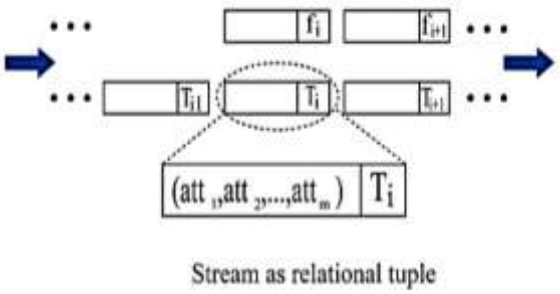

Fig. 3. Flujos de datos relacionales.

Fuente: Adaptada de «Lightweight Asynchronous Snapshots for Distributed Dataflows,»

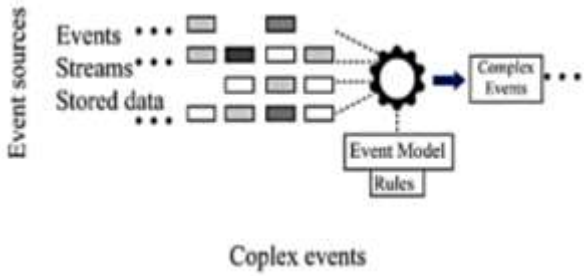

Fig. 4. Modelo de eventos complejos.

Fuente: Adaptada de «Lightweight Asynchronous Snapshots for Distributed Dataflows,»

- **Procesamiento de stream de datos en RDF**

En el procesamiento de stream de datos, también se han llevado a cabo investigaciones que tienen por objetivo llevar a éstos a un nivel semántico; es decir, los datos generados por los sensores (o sensor data) se anotan con metadatos semánticos que incluyen información espacial, temporal y temática extraída de diferentes fuentes, como, por ejemplo, ontologías de dominio. Ejemplos de estas investigaciones son Semantic Stream [75], Semantic System S [76] y Semantic Sensor Web [77]. Paralelo a estas investigaciones, otras como las llevadas a cabo por el W3C Semantic Sensor Network Incubator Group representan los sensores y los stream de datos siguiendo los principios de Linked Data [13], un concepto que se conoce como Linked Stream Data – LSD [14]. LSD facilita, no solamente la integración entre datos generados por sensores heterogéneos, sino también, entre sensores y conjuntos de datos disponibles como Linked Data.

RDF Stream Processing Community Group [15] es un grupo que hace parte de la W3C [35] tiene por objetivo definir un modelo común para producir, transmitir y consultar continuamente stream de datos en RDF (o flujos RDF) incluyendo extensiones a RDF [15] y SPARQL [36] tanto para representar stream data como su semántica. En este grupo, al igual que en [21, 28] se identifican retos que se presentan en el procesamiento de Linked Stream Data - LSD, como por ejemplo en el modelado de stream de datos en RDF, los modelos de consulta, la semántica operacional, el procesamiento y optimización de consultas continuas y las implementaciones, entre otros.

- **Representación**

Desde el punto de vista de la representación de stream de datos en RDF, se han propuesto varios modelos con enfoques diferentes. Por ejemplo, Streaming SPARQL y C-SPARQL adicionan etiquetas temporales a triples RDF. CQELS sugiere una representación general que aplica notación RDF temporal. Además, cada enfoque define su propia noción de stream (ej., secuencia ordenada), elemento de dato (ej., triple o graph), tiempo (ej., implícito, un punto en el tiempo o un intervalo de tiempo) y número de marcas de tiempo (ej., una o dos) [37]. La tabla I, muestra una comparación entre los diferentes enfoques para representar stream de datos en RDF.

- **Implementaciones**

Existen varias propuestas de sistemas que permiten el procesamiento de stream de datos en RDF, como por ejemplo Streaming SPARQL [16], C-SPARQL [17], EP-SPARQL [18], SPARQL stream [19], Instans [20], CQELS [21] y CQELS Cloud [22]. Streaming SPARQL por su parte, extiende la gramática y el álgebra de SPARQL para definir conceptos como estado del stream, marca de tiempo y ventanas; propone nuevos operadores lógicos y físicos y un algoritmo de traslación entre ellos.

TABLA I

ENFOQUES PARA REPRESENTAR FLUJOS RDF

Sistema	Ítem	Tiempo	Marcas de tiempo
INSTANS	Triple	Implícito	0
C-SPARQL	Triple	Punto en el tiempo	1
SPARQL stream	Triple	Punto en el tiempo	1
CQELS	Triple	Punto en el tiempo	1
Sparkware	Triple	Punto en el tiempo	1
Streaming Linked Data	Grafo RDF	Punto en el tiempo	1
ETALIS	Triple	Intervalo	2

Fuente: Tomada de RDF Stream Processing Community Group

C-SPARQL propone un lenguaje para expresar consultas continuas sobre stream de datos y datasets estáticos; define la sintaxis y la semántica de un lenguaje y un modelo de transformación basado en reglas para trasladar consultas a sistemas subyacentes como STREAM [10] y SESAME [78], dos de los motores más representativos en DSMS y SPARQL, respectivamente. SPARQL stream propone un enfoque basado en ontologías para el acceso y consulta de streaming data independiente de la implementación y del lenguaje; define la sintaxis y semántica de un nuevo lenguaje y propone el diseño de un motor que reutiliza internamente tanto motores de stream de datos existentes como procesadores de eventos complejos y sensor middleware. SPARQL stream usa R2RML [79] para definir relaciones entre el modelo streaming data y conceptos ontológicos. EP-SPARQL implementado como prototipo sobre Prolog, extiende el lenguaje de consultas SPARQL con nuevas capacidades para facilitar Event Processing y Stream Reasoning; define la sintaxis y la semántica de un nuevo lenguaje de alto nivel denominado Event Processing SPARQL y propone un modelo de ejecución basado en programación lógica y algunas características propias del procesamiento de eventos con capacidades de inferencia sobre conocimiento temporal y estático. INSTANS presenta una nueva propuesta para el procesamiento de eventos complejos la cual se apoya en el uso de tecnologías de la Web Semántica para la representación de eventos en RDF y la especificación de patrones de eventos en SPARQL. CQELS propone un nuevo motor para el procesamiento nativo y adaptativo de consultas continuas sobre stream de datos en RDF integrado con Linked Open Data; incluye definiciones formales para el modelo de datos, la semántica matemática y operacional de consultas y un modelo de ejecución. CQELS implementa técnicas para el almacenamiento eficiente de datos, el pre procesamiento de consultas y la optimización de consultas basada en costos para fuentes de datos dinámicas [22]. CQELS-Cloud es una extensión de CQELS; adapta operadores existentes como sliding windows y multiway join para que funcionen en un ambiente paralelo distribuidas como la Nube.

- **Escalabilidad**

Un aspecto importante de los sistemas para el procesamiento de stream de datos en RDF es la escalabilidad. Por lo general, la escalabilidad se relaciona con la capacidad de un sistema para procesar consultas concurrentes sobre grandes cantidades de datos, siempre crecientes [38]. La escalabilidad en estos sistemas se ve afectada por varios aspectos, como por ejemplo el número de

consultas continuas concurrentes, el tamaño de los datos, el número de stream de datos, las técnicas de optimización de consultas, la tasa de llegada de los stream de datos y la arquitectura, entre otros [21]. En [38] se reportan resultados de escalabilidad de algunos sistemas y concluyen que ellos presentan problemas en su desempeño cuando alcanzan ciertos umbrales en el número de consultas concurrentes y el tamaño de los datos. En [21] se reporta que algunos sistemas (ej., ETALIS y C-SPARQL) solo pueden procesar conjuntos de datos en RDF de tamaño pequeño, de hasta 1 millón de triples; con tamaños de 2 millones fallan o no responden. Esto se debe en parte a que algunos sistemas como C-SPARQL, SPARQL stream, EP-SPARQL e Instans implementan una arquitectura con un enfoque de caja negra, como se muestra en la Fig.5.

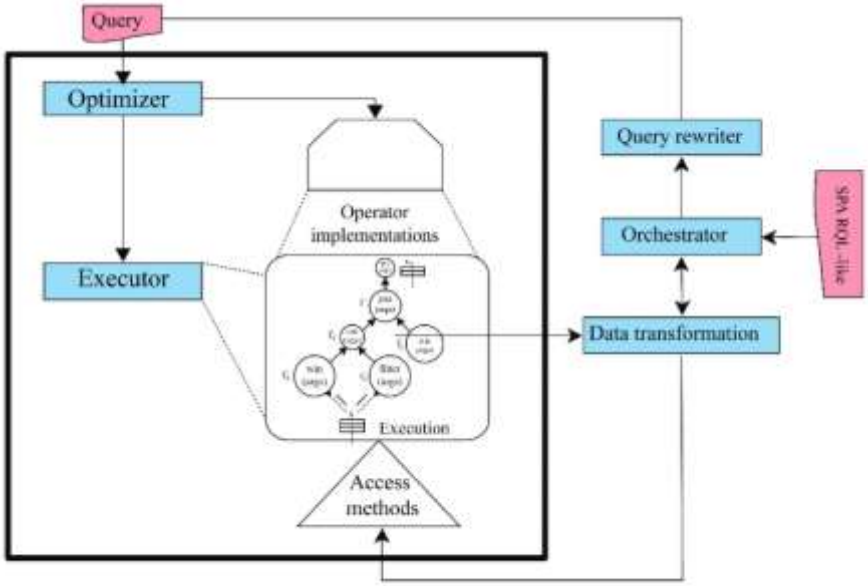

Fig. 5. Arquitectura de caja negra.

Fuente: Adaptada de Challenges in Linked Stream Data Processing: A

Bajo el enfoque de arquitectura de caja negra, los sistemas delegan el procesamiento a otros sistemas (ej., procesadores de flujos/eventos o procesadores SPARQL) lo cual impacta seriamente la escalabilidad, debido a que se requieren procesos adicionales para transformación de datos y traslación de consultas según los requerimientos del sistema subyacente [22].

A diferencia de los anteriores sistemas, CQELS implementa una arquitectura de caja blanca, como se muestra en la Fig. 6. Bajo este enfoque, el sistema implementa operadores y técnicas para el

procesamiento de stream de datos en RDF nativamente, presentando un mejor desempeño y logrando escalar hasta 10 millones de triples para 1000 consultas concurrentes por segundo [37]. A pesar de esto, CQELS no escala bien cuando se incrementa el número de consultas y el tamaño de los datos [38].

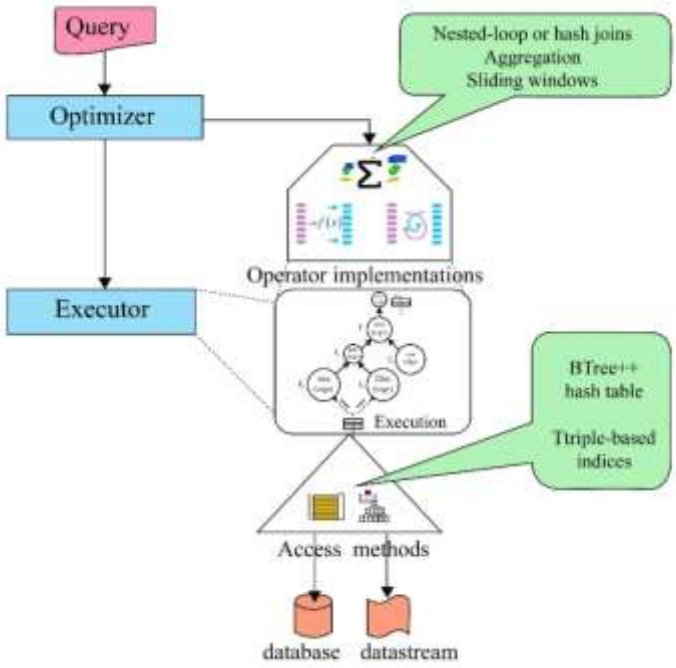

Fig. 6. Arquitectura de caja blanca.

Fuente: Adaptada de Challenges in Linked Stream Data Processing: A

Para mejorar estas limitaciones, se desarrolló el sistema CQELS Cloud, el cual adapta operadores (ej., sliding windows y multiway join) y técnicas de optimización (data encoding y caching and indexing) existentes a un ambiente de ejecución en la Nube. En [22] se reporta que CQELS Cloud puede escalar hasta 100.000 triples por segundo para 10.000 consultas concurrentes sobre un cluster de 32 nodos tipo medium en la plataforma AWS EC2 [80]. Sin embargo, estos umbrales no son el límite final si se consideran conjuntos de datos disponibles como RDF stream y static RDF datasets de gran volumen, como por ejemplo Linked Open Data Cloud [12] que, al año de 2014, según la W3C, contaba con 1014 conjuntos de datos con más de 32.000 millones de tripletas en RDF con 570 millones de enlaces entre ellas. Así como también, el grafo de conocimiento para combatir el tráfico de personas implementado en DIG (Domain-Insight Graphs). En julio de 2015,

la aplicación contenía 60 millones de anuncios, con 162 mil nuevos anuncios por día. El número sujetos RDF era de 1.4 mil millones y el número de características de 222 millones [81].

- **Operadores**

Los operadores son la unidad básica para procesar datos de entrada y producir datos de salida. En el procesamiento de stream de datos, los operadores se definen al menos por un stream de entrada y un stream de salida. Los operadores se clasifican en función de si mantienen o no el estado en la evaluación mientras se procesa stream de datos entrantes [82]. Por una parte, están los operadores stateless que procesan datos uno a uno; es decir, que cada dato se procesa individualmente y la salida correspondiente se genera sin mantener ningún estado. Ejemplos de estos operadores son map, filter, union, select y project entre otros. Por otra, están los operadores statefull que mantienen el estado de la evaluación, puesto que procesan múltiples datos de entrada para producir un dato de salida. Ejemplos de estos operadores son join y aggregates.

El Lenguaje de Consultas Continuas o Continuous Query Langague - CQL [82] es una extensión de SQL para la evaluación de consultas continuas sobre stream de datos relacionales. La semántica de CQL define un modelo formal con tres operadores de mapeo: stream-to-relational que produce relaciones a partir de un flujo, relational-to-relational que produce una relación desde uno o más relaciones y relational-to-stream que produce flujos desde una relación. A su vez, CQL define tres tipos de operadores relationa-to-stream: Istream (Insert stream), Dstream (Delete stream) y Rstream (Relation stream).

Debido a la naturaleza infinita de los stream de datos y a la limitada capacidad de memoria RAM, solamente es posible procesar una porción del stream, normalmente, la más reciente. Esta técnica se conoce como windowing [83]. Las ventanas se usan para mantener únicamente la parte más reciente del flujo. En CQL, por lo general, los operadores stream-to-relation se basan en el concepto sliding window, a partir del cual se definen tres tipos de ventanas: time-based window, tuple-based window y partitioned window [82]. Sintácticamente, este tipo de ventanas se especifican usando un lenguaje derivado de SQL. También existe otro tipo de sliding windows, como fixed windows,

tumbling windows y value-based windows [83]. La Fig.7. muestra algunos ejemplos de combinaciones de tipos de ventanas.

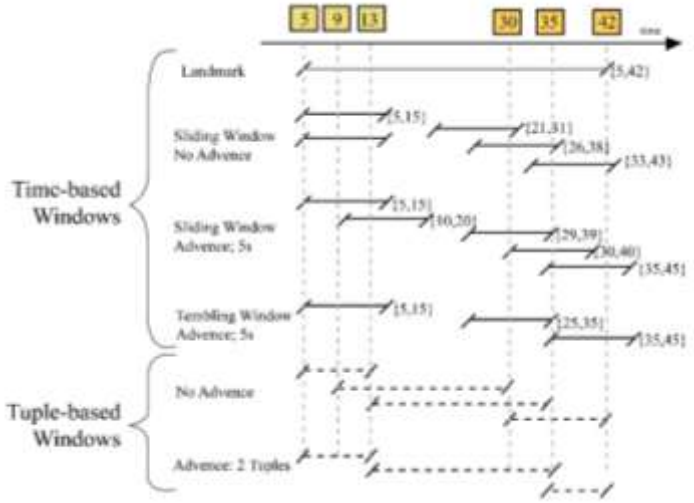

Fig. 7. Ejemplos de operadores de ventanas.

Fuente: Adaptada de StreamCloud: An Elastic and Scalable Data Streaming System.

Los operadores propuestos en CQL son el referente para los operadores en el contexto del procesamiento de stream de datos en RDF. Sin embargo, las extensiones se realizan sobre SPARQL, el lenguaje de consulta para datos en RDF. Por ejemplo, Streaming SPARQL, C-SPARQL, SPARQL stream, EP-SPARQL, CQELS implementan operadores sliding window. Por su parte, Streaming SPARQL implementa operadores triple pattern matching, filter, union, join y otros operadores como tolist, order by, duplicate elimination, duplicate reduction, slice, projection, construct, describe y ask. C-SPARQL implementa operadores unión y filter. Como EP-SPARQL transforma la consulta en una expresión algebraica, implementa operadores a manera de funciones Join, Union, Left Join, SeqJoin, EqJoin, SeqRightJoin y EqLeftJoin. CQELS procesa stactics RDF datasets y RDF streams, por lo tanto, implementa tres tipos de operadores: window operator, para tomar snapshots de la entrada y validarlos con respecto a alguna condición; relational operators, para operar sobre datos finitos y resultados intermedios y streaming operators, para convertir el resultado final nuevamente a stream. En CQELS los operadores se asignan en un Data Flow. Un Data Flow es un árbol de operadores cuyo nodo raíz puede ser un relational operator o un streaming operator, mientras que las hojas y los nodos intermedios son window operator o relational operator,

respectivamente. CQELS Cloud adapta operadores existentes como sliding Windows y multiway join para que funcionen en un ambiente paralelo distribuido como la Nube.

- **Procesamiento de datos a escala masiva**

La cantidad de información generada permanentemente por la sociedad crece en volumen (ej., de terabytes a petabytes o aún mayor), en variedad (ej., datos estructurados y no estructurados) y a una velocidad alta [23]. Esta enorme generación de datos junto con la adopción de nuevas estrategias para hacerles frente, han provocado la aparición de una nueva era de la gestión de datos, comúnmente conocida como Big Data. En términos generales, Big Data se puede definir como la aparición de nuevos conjuntos de datos de volumen masivo que cambian rápidamente, son complejos y exceden el alcance de las capacidades analíticas de los entornos de hardware y de software de uso común para el procesamiento de datos [24, 25].

II. METODOLOGÍA

A. Preguntas de investigación

¿Qué lenguaje de consulta para stream de datos en RDF se puede usar para transformar una consulta a un Programa en Apache Flink?

¿Qué operadores del lenguaje de consulta para stream de datos en RDF se pueden representar con los operadores del Data Stream API de Apache Flink?

¿Cuál de los benchmark propuestos en el contexto del procesamiento de stream de datos en RDF se puede usar para realizar las pruebas sobre un cluster local de Apache Flink?

B. Formulación de hipótesis

Apache Flink es un motor de procesamiento de datos que es capaz de procesar datos dinámicos en formato RDF.

C. Enfoque

El enfoque de esta investigación es un enfoque cuantitativo, porque representa un conjunto de procesos secuencial y probatorio. En donde se plantea un problema de estudio delimitado y concreto.

D. Tipo de investigación

Investigación cuantitativa de tipo investigación experimental, ya que se desea comprobar la viabilidad y el mejor desempeño de la tecnología propuesta para el procesamiento de stream de datos en RDF, a través de la captura de métricas del rendimiento que esta tiene en memoria, disco y CPU usadas al momento de ejecución de las consultas propuestas bajo los operadores definidos con respecto a otras existentes.

E. Muestra

Para esta investigación se realizar pruebas empíricas basadas en el benchmark Citybench, el cual cuenta con alrededor de 16 tipos de diferentes sensores simulados, que a su vez son capaces de generar entre 50 y 1000 cuads (Sujeto, predicado, Objeto y Tiempo) de datos generados a través de ontologías que interpretan la información substraída de archivos csv o txt.

F. Técnicas de recolección de información

Se ajustó las consultas disponibles en Citybench, adaptándolas a los operadores soportados por la librería de procesamiento de datos en RDF desarrollada para esta investigación, igualmente por CQELS y con ello se hizo pruebas empíricas que permitieron realizar comparaciones entre ambas herramientas, siguiendo lo planteado en este benchmark.

G. Instrumentos de recolección de información

En esta investigación se tomó como referencia el benchmark seleccionado Citybench, ya que este contempla comparaciones entre las herramientas de procesamiento de datos mencionadas anteriormente en esta investigación. Se realizó ajustes a la librería disponible de la última versión del Citybench, en donde se agregó un módulo que permite lanzar los sensores disponibles específicamente diseñados para la librería que se desarrolló para el procesamiento de datos en RDF con Apache Flink y se usó herramientas de monitoreo del sistema operativo para registrar el rendimiento de memoria, disco y CPU usadas en el momento de ejecución de cada consulta.

H. Actividades generales

Para cumplir con los objetivos propuestos, a continuación, se describen las siguientes actividades generales:

- **A1:** Estudio del estado del arte de los sistemas para el procesamiento de datos a escala masiva. A partir del estudio del estado del arte en esta área será posible identificar las principales limitaciones de los modelos y sistemas que se han propuesto hasta la fecha, especialmente en lo que respecta a su escalabilidad, así como una comparación entre los sistemas más representativos que ayude a determinar el sistema sobre el cual se implementará el mapper.
- **A2:** Selección de un lenguaje de consulta para procesar stream de datos en RDF Stream. Este lenguaje de consulta se seleccionará con base en los resultados de las discusiones del W3C RDF Stream Processing Community Group. En especial, se prestará atención a aquellos modelos que tengan en cuenta el procesamiento de stream de datos.
- **A3:** Desarrollo de mapper para transformar una consulta a un programa en el framework de procesamiento de escala masiva. El principal resultado que se buscará con esta fase será el desarrollo de una librería para la transformación de una consulta expresada en el lenguaje de

consulta seleccionado a un framework de procesamiento de escala masiva. En esta fase, es importante definir qué operadores se transformarán teniendo en cuenta su equivalencia semántica.

- **A4:** Desarrollo de runner para ejecutar el programa en el framework de procesamiento de escala masiva. El runner permitirá ejecutar el programa definido con base en el framework seleccionado, así como la carga de los stream de datos.
- **A5:** Configuración de un ambiente de pruebas sobre un cluster de datos.
- **A6:** Selección de stream de datos y consultas basadas en un benchmark existente.
- **A7:** Escritura del trabajo de grado y de artículos científicos.

III. METODOLOGÍA DE DESARROLLO DE SOFTWARE

La Ingeniería del Software es una disciplina dentro de la rama informática en constante evolución. En la actualidad la realización de proyectos de software está fuertemente marcada por los procesos involucrados. Dichos procesos identifican el conjunto de tareas y actividades a realizar para la correcta ejecución de un proyecto de software.

Para poder llevar a cabo los objetivos de este proyecto, es necesario desarrollar tres herramientas de software, por lo cual se seguirá cada uno de los pasos en la elaboración de un proyecto software de alta calidad según la metodología de desarrollo en cascada, que son:

- Análisis de requisitos
- Diseño
- Desarrollo
- Prueba
- Mantenimiento
- Entrega

A. Planeación

1) Levantamiento de Requerimientos

TABLA II

REQUISITOS FUNCIONALES MAPPER

Identificador	Descripción
RQF-01	El mapper le solicita al usuario pasar como argumento la ruta del archivo en el cual se encuentra la consulta expresada en la sintaxis del lenguaje de consulta seleccionado.
RQF-02	El mapper le solicita al usuario pasar como argumento la ruta de escritura del archivo con el código resultante del mapeo de la consulta.

Fuente: esta investigación.

TABLA III

REQUISITOS NO FUNCIONALES MAPPER

Identificador	Descripción
RQNF-01	El mapper debe generar un plan operacional (OP) generado a partir de la interpretación sintáctica de la consulta leída.
RQNF-02	El mapper debe ser capaz de definir el flujo de ejecución del programa resultado de la interpretación del árbol generado del OP relacionado los operadores soportados con los métodos disponibles del API Stream de Apache Flink.
RQNF-03	El mapper debe definir en el programa generado la forma de salida de los resultados obtenidos al realizar el proceso de ejecución a través de este.
RQNF-04	El mapper debe escribir el código resultante en un archivo .java y generar el jar del runner que contiene los módulos necesarios para realizar el procesamiento.

Fuente: esta investigación.

TABLA IV

REQUISITOS FUNCIONALES RUNNER

Identificador	Descripción
RQF-01	El runner le permite al usuario pasar como argumento la ruta de escritura del archivo con los resultados generados después del procesamiento, en caso de que no sea así se imprime por consola.
RQF-02	La interfaz web dispuesta por Apache Flink permite ejecutar el jar del runner en standalone o cluster, de esta forma obtener los datos relacionados al desempeño del procesamiento realizado a parte de los resultados que este genera.

Fuente: esta investigación.

TABLA V

REQUISITOS NO FUNCIONALES RUNNER

Identificador	Descripción
RQNF-01	El Runner debe ser capaz de conectarse a los sensores disponibles a través de sockets.
RQNF-02	El Runner debe ser capaz de procesar hasta petabytes de datos esto según el tamaño en disco del cual disponga las maquinas en las que se ejecute en caso de cluster o standalone.

Fuente: esta investigación.

TABLA VI

REQUISITOS FUNCIONALES CITYBENCH EXTENDIDO

Identificador	Descripción
RQF-01	El programa le solicita el tipo de simulación: Generación de los datos del sensor y escribirlos por archivo para lo cual debe pasar la ruta de salida como argumento también. Correr los sensores por socket.
RQF-02	El programa le solicita el nombre del archivo de la consulta de donde se realizara la extracción de los sensores a lanzar.
RQF-03	El programa le permite ingresar como argumento un puerto inicial para lanzar los sockets.
RQF-04	El programa imprime por consola los sockets generados o lanzados.

Fuente: esta investigación.

TABLA VII

REQUISITOS NO FUNCIONALES CITYBENCH EXTENDIDO

Identificador	Descripción
RQNF-01	El programa debe ser capaz de lanzar los sensores necesarios a través de sockets en Java.

Fuente: esta investigación.

B. Diseño

1) Definición de diagramas de clases

- **Diagramas de clases del Mapper**

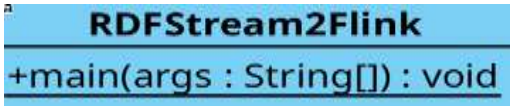

Fig. 8. Diagrama de clase RDFStream2Flink.

Fuente: esta investigación.

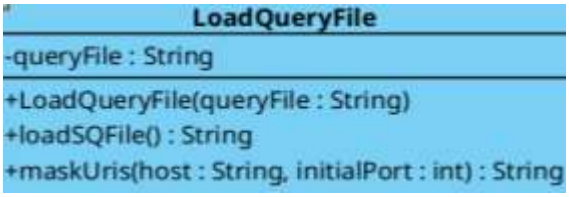

Fig. 9. Diagrama de clase LoadQueryFile.

Fuente: esta investigación.

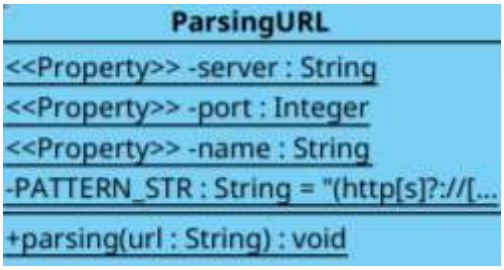

Fig. 10. Diagrama de clase ParsingURL.

Fuente: esta investigación.

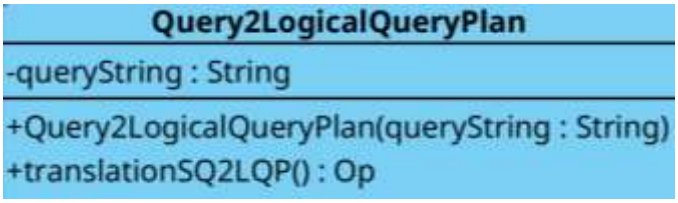

Fig. 11. Diagrama de clase Query2LogicalQueryPlan.

Fuente: esta investigación.

Fig. 12. Diagrama de clase LogicalQueryPlan2FlinkProgram.

Fuente: esta investigación.

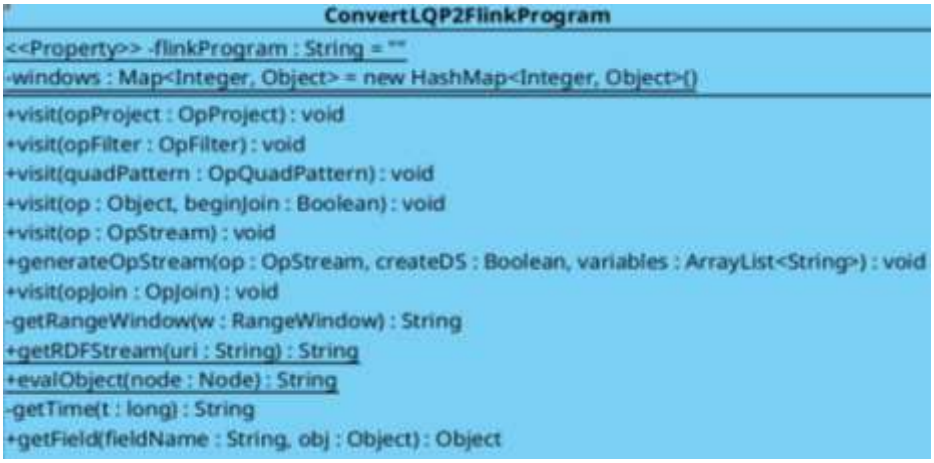

Fig. 13. Diagrama de clase ConvertLQP2FlinkProgram.

Fuente: esta investigación.

Fig. 14. Diagrama de clase ConvertTriplePattern.

Fuente: esta investigación.

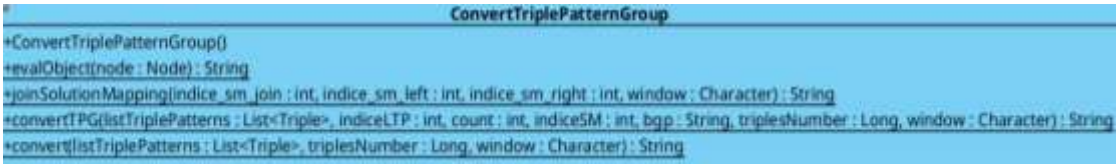

Fig. 15. Diagrama de clase ConvertTriplePatternGroup.

Fuente: esta investigación.

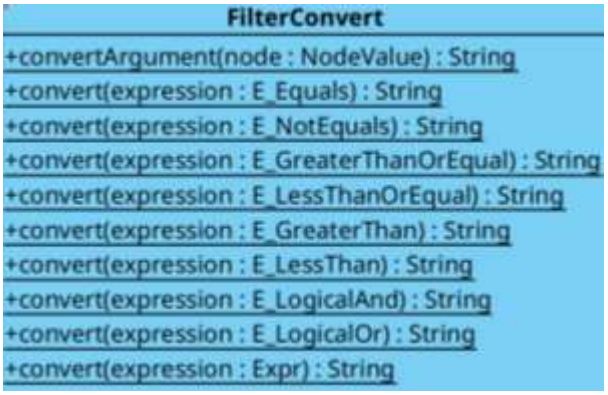

Fig. 16. Diagrama de clase FilterConvert.

Fuente: esta investigación.

Fig. 17. Diagrama de clase JoinKeys.

Fuente: esta investigación.

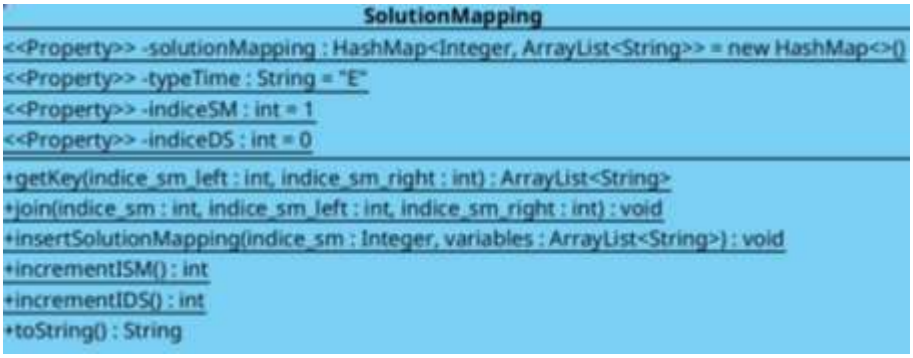

Fig. 18. Diagrama de clase SolutionMapping.

Fuente: esta investigación.

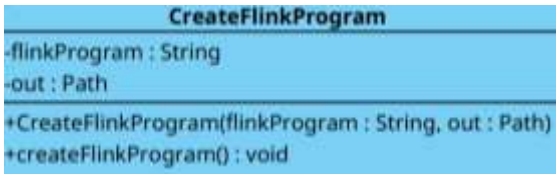

Fig. 19. Diagrama de clase CreateFlinkProgram.

Fuente: esta investigación.

- **Diagramas de clases del Runner**

Fig. 20. Diagrama de clase Runner.

Fuente: esta investigación.

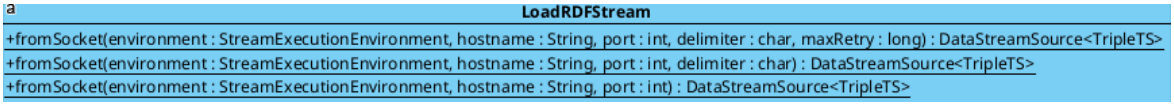

Fig. 21. Diagrama de clase LoadRDFStream.

Fuente: esta investigación.

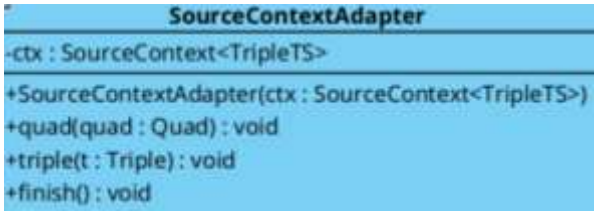

Fig. 22 Diagrama de clase SourceContextAdapter.

Fuente: esta investigación.

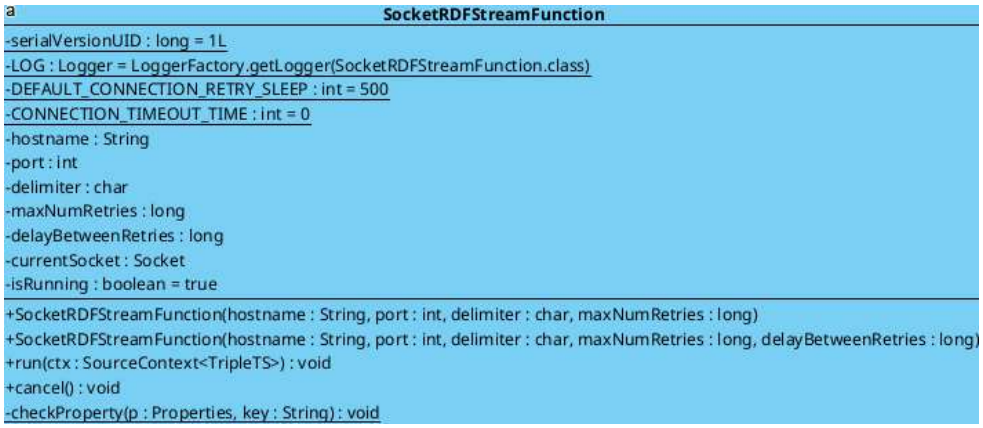

Fig. 23. Diagrama de clase SocketRDFStreamFunction.

Fuente: esta investigación.

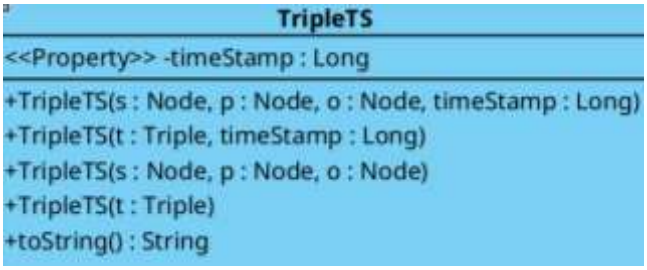

Fig. 24. Diagrama de clase TripleTS.

Fuente: esta investigación.

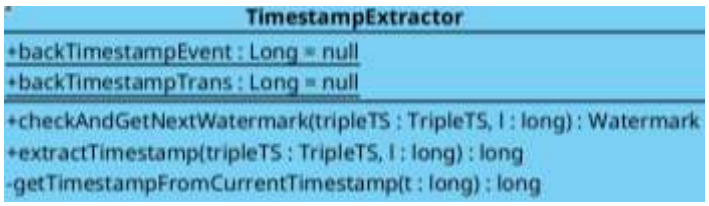

Fig. 25. Diagrama de clase TimestampExtractor.

Fuente: esta investigación.

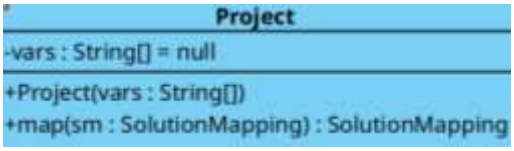

Fig. 26. Diagrama de clase Project.

Fuente: esta investigación.

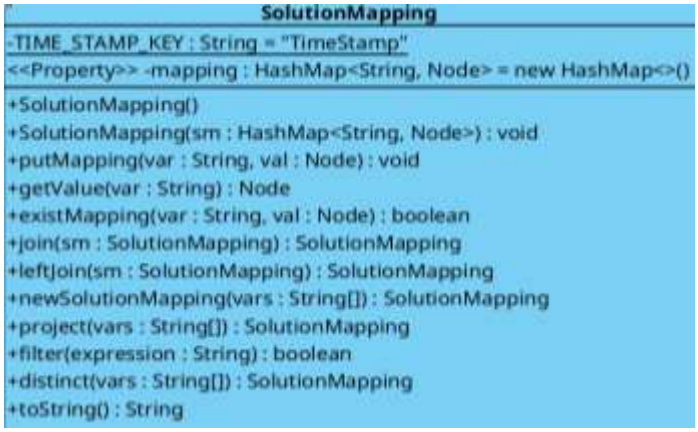

Fig. 27. Diagrama de clase SolutionMapping.

Fuente: esta investigación.

Fig. 28 Diagrama de clase WindowKeySelector.

Fuente: esta investigación.

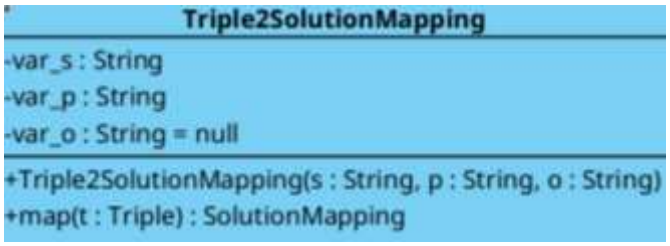

Fig. 29 Diagrama de clase Triple2SolucionMapping.

Fuente: esta investigación.

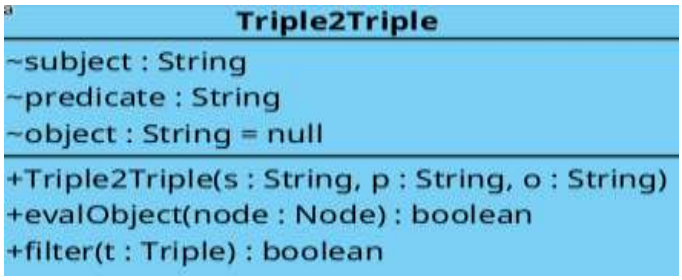

Fig. 30. Diagrama de clase Triple2Triple.

Fuente: esta investigación.

Fig. 31. Diagrama de clase Join.

Fuente: esta investigación.

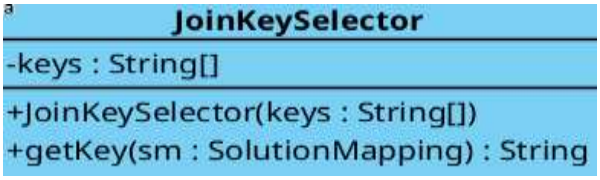

Fig. 32. Diagrama de clase JoinKeySelector.

Fuente: esta investigación.

- **Diagramas de clases del Citybench extendido**

Fig. 33. Diagrama de clase Main.

Fuente: esta investigación.

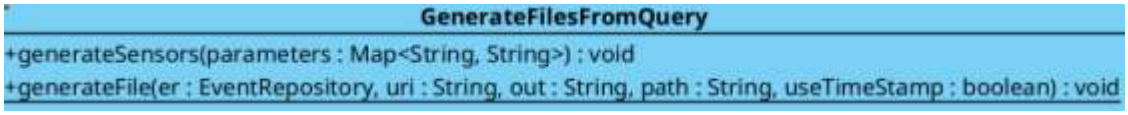

Fig. 34. Diagrama de clase GenerateFilesFromQuery.

Fuente: esta investigación.

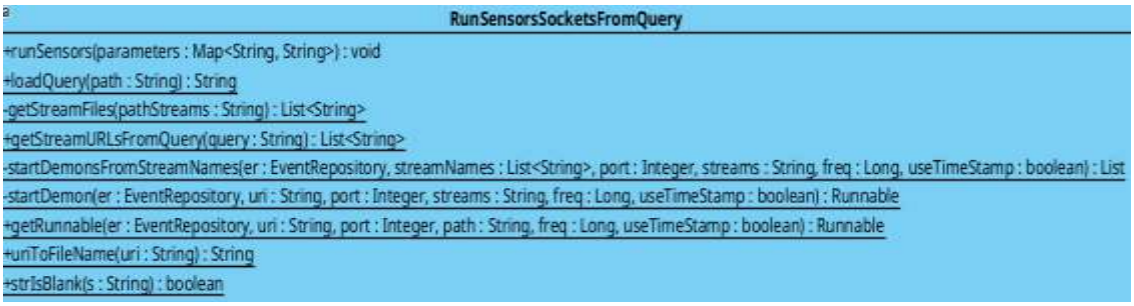

Fig. 35. Diagrama de clase RunSensorSocketsFromQuery.

Fuente: esta investigación.

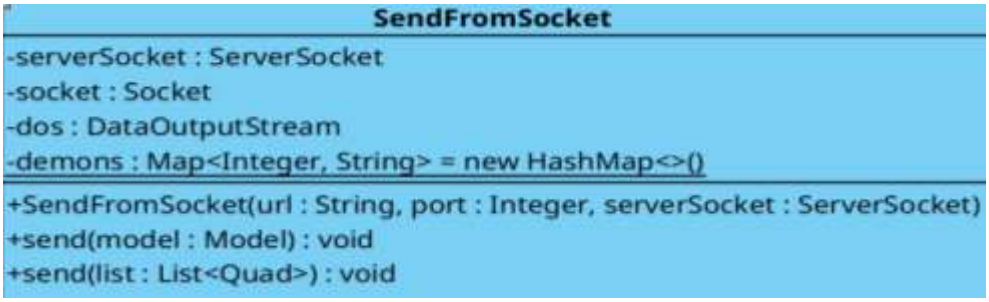

Fig. 36 Diagrama de clase SendFromSocket.

Fuente: esta investigación.

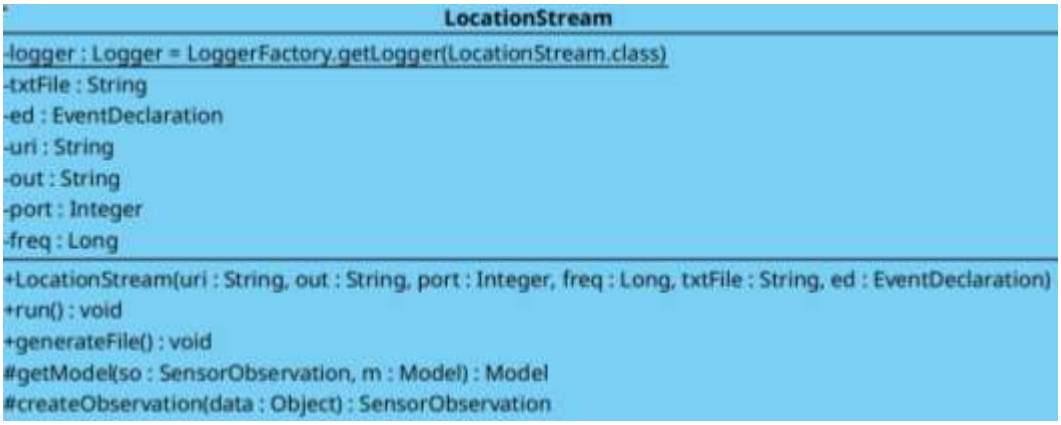

Fig. 37. Diagrama de clase LocationStream.

Fuente: esta investigación.

Fig. 38. Diagramas de clase streams.

Fuente: esta investigación.

C. Definición de diagramas de flujo

1) Diagrama de flujo del Mapper

En la Fig. 39 se define el diagrama de flujo para el mapper, que establece principalmente el flujo funcional el cual se genera en el momento en que el usuario ejecuta el jar suministrado, establece los argumentos y se realiza la ejecución de los respectivos procesos necesarios según las validaciones estipuladas debido a estos, finalmente el usuario obtiene el programa final, de acuerdo a las rutas correspondientes a la consulta y a la salida ingresadas.

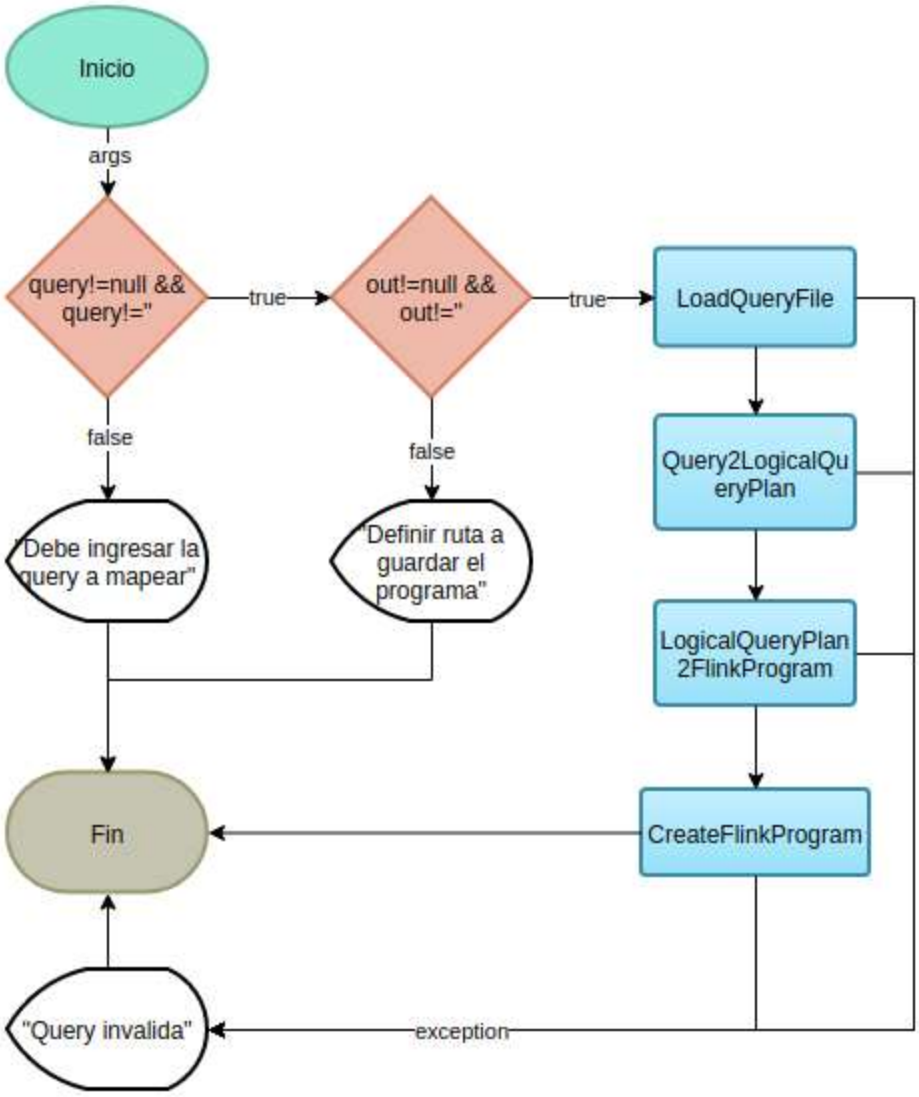

Fig. 39. Diagrama de flujo Mapper.

Fuente: esta investigación.

- **Diagrama de flujo del Runner**

En la Fig. 40 se define el diagrama de flujo para el runner, que establece principalmente el flujo funcional el cual se genera en el momento en que el usuario ejecuta el jar suministrado, establece los argumentos y se realiza la ejecución de los respectivos procesos necesarios para capturar la información de los sockets definidos para los streams, según las validaciones estipuladas debido a

estos, finalmente el usuario en caso de haber definido la ruta de salida, obtiene los resultados generados del procesamiento almacenados en esta, por lo contrario simplemente se realiza la impresión por consola.

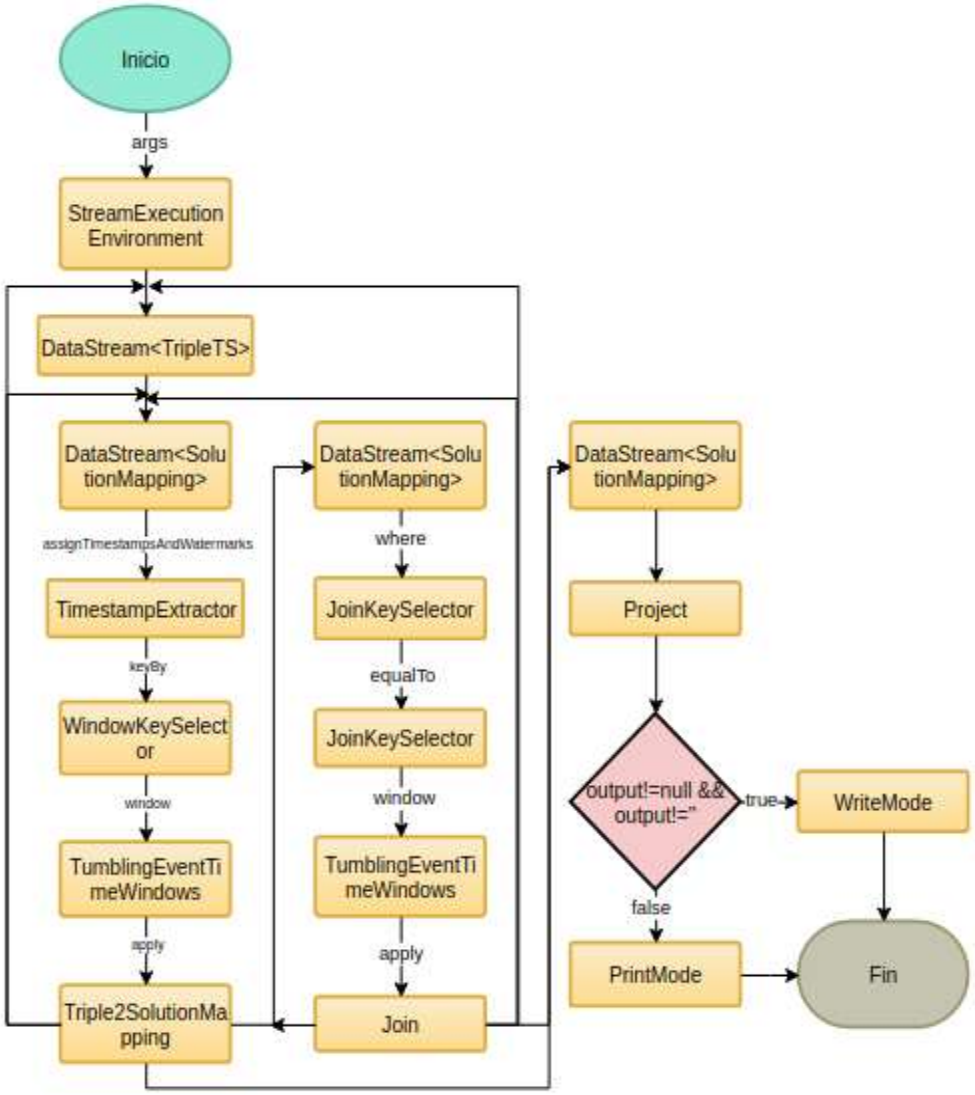

Fig. 40. Diagrama de flujo Runner.

Fuente: esta investigación.

- **Diagrama de flujo del Citybench Extendido**

En la Fig. 41 se define el diagrama de flujo para el Citybench Extendido, que establece principalmente el flujo funcional el cual se genera en el momento en que el usuario ejecuta el jar

suministrado, establece los argumentos y se realiza la ejecución de los respectivos procesos necesarios según las validaciones estipuladas debido a estos, finalmente el usuario en caso de definir el tipo “G” obtiene los archivos relacionados a los streams con los datos en formato RDF y en caso de que el tipo sea “R” se levanta los sockets necesarios para cada stream y se notifica por consola el estado de cada uno.

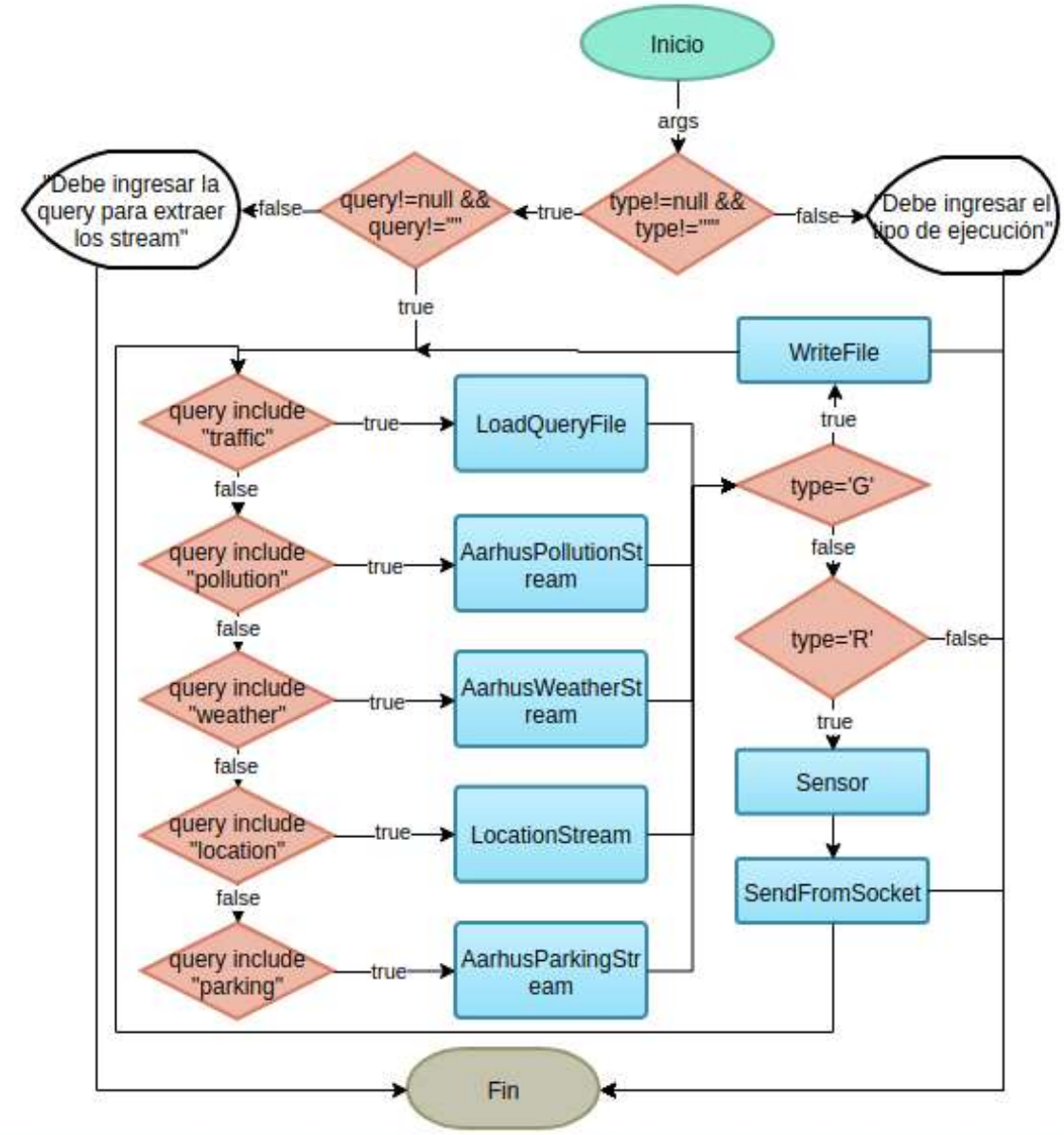

Fig. 41. Diagrama de flujo Citybench Extendido.

Fuente: esta investigación.

D. Definición de diagramas de uso

1) Diagrama de uso del Mapper

- **Especificación de caso de uso**

<Generar Flink Program> Fig. 42

- **Descripción**

Permite al usuario generar un programa basado en el API Stream de Apache Flink a partir de una consulta realizada sobre la sintaxis del lenguaje de consulta seleccionado.

- **Actores**

Usuario S.O.

- **Pre-condiciones**

Sistema Operativo (S.O.) Linux, base Debian.

Usuario S.O., debe tener instalado jdk 8.

Usuario S.O., debe tener instalado CLI de Maven.

Query realizada sobre la sintaxis del lenguaje de consulta seleccionado.

- **Post-condiciones**

Archivo con el programa basado en el API stream de Apache Flink.

- **Flujo de eventos**

✓ *Flujo Principal*

P1. El flujo de eventos principal se inicia cuando el usuario realiza la ejecución por consola del programa mapper, como argumentos define la ruta a la consulta y la ruta de salida del programa resultante.

P2. El programa valida los datos y lee la consulta desde el archivo ubicado en la ruta especificada.

P3. El programa realiza el mapeo de la consulta y genera el plan operacional (OP) relacionada a esta.

P4. El programa realiza el recorrido del OP y genera un string del programa basado en la sintaxis del API Stream de Apache Flink.

P5. El programa realiza la escritura del string del programa en la ruta de salida definida como argumento.

- ✓ *Flujos de Excepción*

E1. Usuario no define la ruta de la consulta o la salida.

El programa imprime por consola "You must specify the query and output paths in the specified arguments".

El programa finaliza.

E2. Consulta invalida.

El programa imprime por consola la traza generada al momento de realizar la validación sintáctica de la consulta.

El programa finaliza.

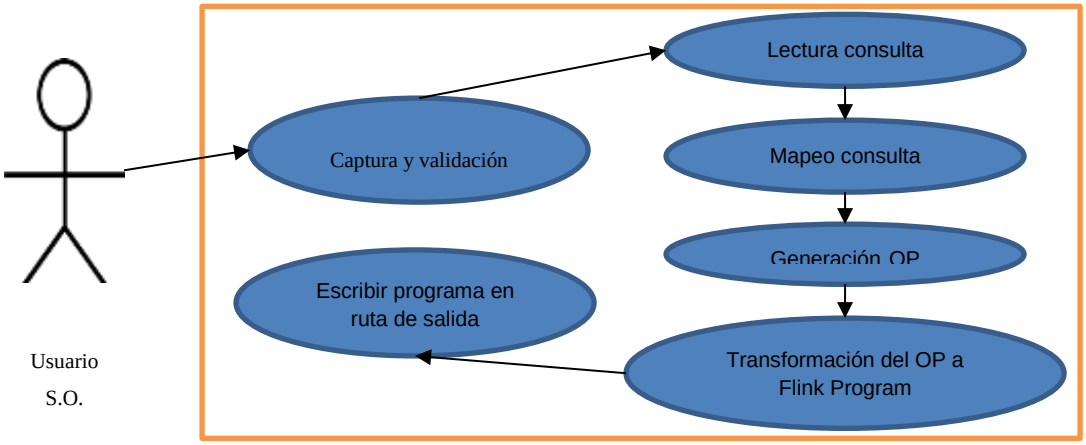

Fig. 42. Diagrama de uso <Generar Flink Program>.

Fuente: esta investigación.

2) Diagramas de uso del Runner

- **Especificación de caso de uso**

<Procesamiento de Stream de Datos en RDF> Fig. 43

- ✓ *Descripción*

Permite el procesamiento de streams de datos en formato RDF a través del API Stream de Apache Flink.

- ✓ *Actores*

Usuario S.O.

- ✓ *Pre-condiciones*

Sistema Operativo (S.O.) Linux, base Debian.

Usuario S.O., debe tener instalado jdk 8.

Usuario S.O., debe tener instalado CLI de Maven.

Streams de datos en formato RDF transmitidos a través de sockets.

- ✓ *Post-condiciones*

Archivo con el resultado del procesamiento.

- ✓ *Flujo de eventos*

Flujo Principal

P1. El flujo de eventos principal se inicia cuando el usuario realiza la ejecución por consola del programa runner, el usuario define la ruta de salida como argumento.

P2. El programa conecta a los streams a través de clientes Sockets.

P3. El programa toma los triples RDF en string y los trata en los Solutions Mapping.

P4. El programa usa las ventanas para tratar los datos disponibles en Solutions Mapping durante los intervalos de tiempo definidos para estas.

P5. El programa usa el operador Join para combinar los datos disponibles de dos streams a través de las ventanas y secuencialmente unen múltiples de estos.

P6. El programa genera objetos por cada combinación de Joins como en P2.

P7. El programa finalmente escribe los resultados en tiempo real en el output especificado como parámetro.

Flujos de Alternativo

A1. Usuario no define output de salida.

El programa continúa normalmente hasta P6.

El programa imprime los resultados por consola en tiempo real.

Flujos de Excepción

E1. Streams no existe.

El programa encuentra que uno de los streams no está disponible para conexión.

El programa finaliza.

E2. Stream no transmite los datos en formato RDF.

El programa encuentra que no puede convertir el string entrante por el socket al Solution Mapping.

El programa finaliza.

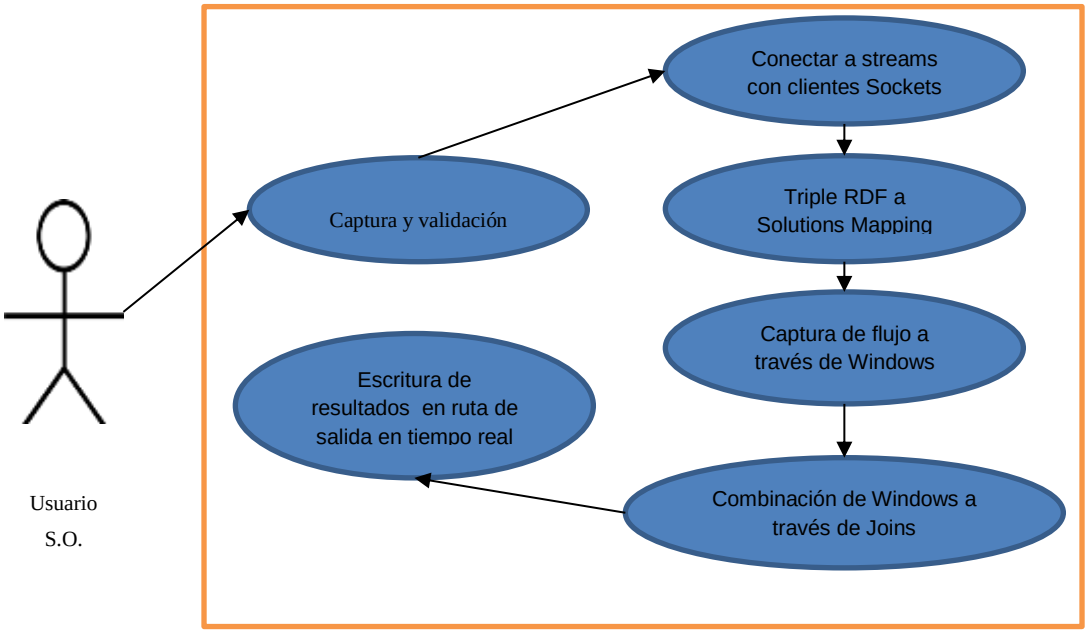

Fig. 43. Diagrama de uso <Procesamiento de Stream de Datos en RDF>.

Fuente: esta investigación.

3) Diagramas de uso del Citybench Extendido

- **Especificación de caso de uso**

<Levantar streams sensors> Fig. 44

✓ *Descripción*

Permite al usuario levantar los streams de los sensores definidos en una consulta de acuerdo a los datasets disponibles.

✓ *Actores*

Usuario S.O.

✓ *Pre-condiciones*

Sistema Operativo (S.O.) Linux, base Debian.

Usuario S.O., debe tener instalado jdk 8.

Query realizada sobre la sintaxis del lenguaje de consulta seleccionado.

- ✓ *Post-condiciones*

Sockets que simulan la transición de los datos en formato RDF.

- ✓ *Flujo de eventos*

Flujo Principal

P1. El flujo de eventos principal se inicia cuando el usuario realiza la ejecución por consola del programa Citybench Extendido, como argumentos define la ruta de la consulta, el tipo "R" y el puerto origen para los streams.

P2. El programa valida los datos y lee la consulta desde el archivo ubicado en la ruta especificada.

P3. El programa realiza el mapeo de la consulta y obtiene los streams definidos en esta.

P4. El programa realiza el recorrido de la lista de streams encontrados y lee los datasets correspondientes para cada uno.

P5. El programa a través de las ontologías da formato RDF a los datos obtenidos.

P6. El programa pasa los datos formateados a los sockets encargados de realizar la transmisión de datos.

Flujos de Alternativo

A1. Usuario define si desea generar archivos con los datos en RDF de los streams a simular.

El usuario ingresa el tipo "G" y adicionalmente define la ruta del folder a escribir los archivos.

El programa continúa desde P2 hasta P5.

El programa escribe los datos formateados en el archivo correspondiente al stream.

Flujos de Excepción

E1. Usuario no define el argumento type.

El programa imprime por consola "Error, not found parameter 'type'".

El programa finaliza.

E2. Usuario no define un tipo soportado.

El programa imprime por consola "Error, type specified is not known".

El programa finaliza.

E3. Usuario no define el argumento query.

El programa imprime por consola "Error, not found parameter 'query'".

El programa finaliza.

E4. Usuario no define el argumento out al definir el tipo "G".

El programa imprime por consola "Error, not found parameter 'out'".

El programa finaliza.

E5. Consulta invalida.

El programa imprime por consola la traza generada al momento de realizar la validación sintáctica de la consulta.

El programa finaliza.

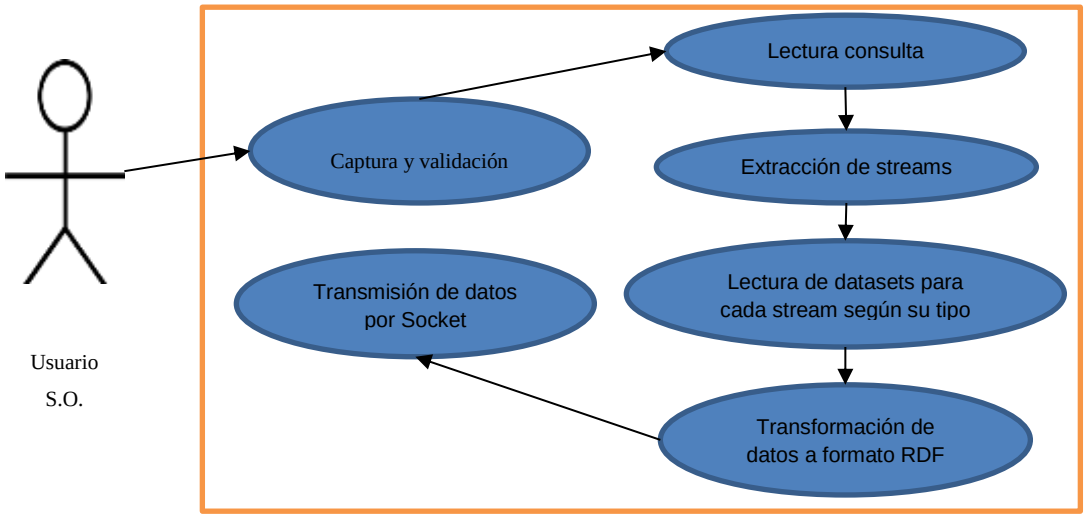

Fig. 44. Diagrama de uso <Levantar streams sensors>.

Fuente: esta investigación.

E. Producción

1) Software a realizar

- **Mapper**, software encargado de tomar una consulta en la sintaxis de RDF según el lenguaje de consulta elegido para streams de datos, generar un plan operacional (OP) a través de esta, recórrelo haciendo uso visitors para definir un programa en Java usando el API Stream de Apache Flink que estructure el flujo de procesamiento de stream de datos y finalmente escribir el resultado en la ruta de salida definida.
- **Runner**, software generado a través del mapper el cual usa el API Stream de Apache Flink, que se encarga de conectarse a los stream transmitidos por sockets, hacer la manipulación de los mismos a través de ventanas, definir los operadores encargados de procesar los datos disponibles en las mismas y escribir los resultados en tiempo real en la ruta de salida especificada.
- **Citybench Extendido**, software basado en Citybench, que es un software que cuenta con una suite para evaluar motores RSP dentro de aplicaciones de ciudad inteligente e incluye flujos de datos de IoT en tiempo real generados a partir de varios sensores implementados en la ciudad de Aarhus, Dinamarca. Al cual se le realizo una extensión para transmitir los datos disponibles en sus datasets por sockets con el fin de simular los sensores relacionados a una consulta.

2) Desarrollo

Herramientas de software utilizadas:

Linux

Java JDK 8

CQELS

Citybench

Apache Flink

Jena

Opencsv

Javacsv

3) Construcción de MAPPER

Para realizar la construcción del mapper, se partió por determinar el lenguaje de consulta de RDF Stream a utilizar, según la documentación y la bibliografía encontrada, el más reciente es el usado por CQELS [21] y CQELS Cloud [22] en donde se tomó SPARQL (SPARQL Protocol and RDF Query Language) y se lo expandió para dar estructura a consultas que usen streams, agregando nuevos operadores que permitan la manipulación de estos y definir los intervalos de apertura de las ventanas, mediante las cuales se trata la información de los nodos del grafo RDF que se transmite. De acuerdo a esto, se tomó el código fuente de CQELS realizado en Java, del cual se extrajo la lógica encargada de generar el árbol que define el plan operacional (OP) de la consulta dada, esto fue posible ya que es código abierto.

En la Fig. 8 se define la clase principal del Mapper, encargada de capturar como argumentos (args), con los datos relacionados a la ruta de la consulta, en la sintaxis del lenguaje seleccionado y la ruta de salida para almacenar el archivo con el programa resultante después del proceso.

Después de capturar los argumentos se realiza la lectura de la información de la query en el archivo, con la clase de la Fig. 9 con esto a través de la clase expresada en la Fig. 10 se realiza el proceso transformación de la query en texto plano al plan operacional (OP), el cual se define como una estructura de datos conocida como árbol y en este se define el plan de ejecución a seguir para realizar la consulta.

Vale la pena mencionar que a nivel de pruebas en standalone se manejan sensores simulados localmente, para lo cual fue necesario enmascarar las urls de conexión a los streams definidas en las consultas a través de la clase de la Fig. 11, para esto la clase definida en la Fig. 12 también se captura los argumentos host y port que no son obligatorias ya que se usan únicamente en caso de que se desee leer los datos desde otro anfitrión diferente al definido en la query.

Una vez definido el OP se procede a transformarlo al programa objetivo en lenguaje Java usando el API Stream de Apache Flink, para esto se usan las clases definidas en las Fig. 13 y 14 las cuales

a través de visitors realizan el recorrido del árbol en donde, dependiendo de las entradas de las hojas visitadas, se define los diferentes elementos operacionales del runner, que deberían resolver específicamente esa parte de la consulta.

Para definir los triples en las entradas de los operadores y ventanas del API Stream de Apache Flink encargadas de capturar, filtrar y procesar la información obtenida de los streams (en el caso de las pruebas empíricas propuestas en ese proyecto de grado son sensores simulados) a los cuales se está escuchando, se usan las clases definidas en las Fig. 15 a la 17 que validan y transforman la estructura de los triples (S, P, O) que es sujeto, predicado y objeto, expresadas en formato RDF, los cuales en sí son las que definen los nodos del grafo generado en los recursos disponibles de la información capturada a través de los clientes sockets que los escuchan, principalmente en el caso de los objetos los cuales a la vez pueden ser nodos o datos primitivos finales como texto, números o fechas.

Naturalmente ya que el producto final de este programa es generar un archivo (.java) con un programa expresado en la sintaxis de Java, es necesario definir variables y para ese propósito se usan las clases de las Fig. 18 y 19 que guardan el mapeo de las variables generadas a través de los visitors, llevan un histórico basado en los nombres de variables asignados principalmente para joins y solutions mapping, las cuales son las variables encargadas de capturar y cruzar los datos obtenidos a través de las ventanas.

Finalmente, una vez generado todo el programa en base al API Stream de Apache Flink se usa la clase de la Fig. 20 que lo escribe en la ruta de salida definida como argumento.

4) Construcción de RUNNER

En la Fig. 21 se define la clase principal del Runner (este es el programa generado por el Mapper) encargada de capturar los argumentos (args) relacionados a la ruta de escritura de los resultados obtenidos al final del procesamiento, eso en caso de que el usuario así lo quiera ya que por lo contrario estos simplemente se imprimen en consola.

Se realiza la conexión a streams de datos a través de sockets para lo cual el Runner hace uso de las clases definidas en las Fig. 22 a la 24, con las cuales se generan los clientes que escuchan los datos emitidos.

Una vez capturados los datos, se realiza el proceso de transformación al formato RDF (sujeto, predicado y objeto) usando las clases de las Fig. 25 y 26, con las cuales adicionalmente se obtiene las marcas de tiempo en caso de que los triples sean datos históricos, por lo contrario, simplemente se generan marcas de tiempo basadas en el tiempo de procesamiento, en esta investigación se trabajara con este tipo de marcas de tiempo.

Después de realizar la captura y transformación de los datos se define la proyección del procesamiento a través de los objetos Solution Mapping con las clases definidas en las Fig. 27 y 28. Los Solution Mapping son objetos que a través de las ventanas pueden capturar información durante una cantidad de tiempo expresada por segundos, minutos y horas, para ser almacenados temporalmente y de esta forma ser procesados.

Las ventanas usan la clase definida en la Fig. 29 para ser identificadas en la manipulación de la data, que mediante el uso de las clases de las Fig. 30 y 31 se le define a la ventana como manejar la información que capturó, ya que el propósito de esta investigación es realizar procesamiento de datos en formato RDF se usa el triple para estructurar la misma.

Una vez se define los agrupamientos de datos y su almacenamiento temporal se procede a determinar los operadores definidos en las Fig. 32 y 33 que son encargados de procesar la información y por último se realiza la salida de los resultados obtenidos, esto se genera a medida que se va liberando información de las ventanas.

5) Construcción de CITYBENCH extendido

Como se encontró en [38, 53, 54] el benchmark más completo debido a la cantidad de pruebas y tipos de motores evaluados es Citybench, por lo cual en esta investigación se lo tomo referencia para el desarrollo del producto del tercer objetivo, que es realizar un ambiente de pruebas estandarizado para evaluar el motor de procesamiento que se realizó.

En la Fig. 33 se define la clase principal de la extensión realizada en Citybench encargada de capturar como argumentos (args) el tipo de ejecución, la query de la cual se va a extraer los streams a simular, el puerto inicial para asignar a los sockets generados (solo es necesario para el tipo “R”) y por último el path del directorio en el cual se va a almacenar los archivos generados con la información de los streams (solo es necesario para el tipo “G”).

Cuando el usuario ingresa por argumento el tipo “G”, se usa la clase definida en la Fig. 34, para tomar la información de los sensores en formato RDF y escribir los archivos respectivos en el directorio de salida definido como argumento.

Cuando el usuario ingresa como argumento el tipo “R” se usa la clase definida en la Fig. 35, para tomar la información de los sensores en formato RDF y transmitirla por socket, usando la clase de la Fig. 36, se usa el número de puerto ingresado como argumento para establecer el puerto usado por el socket, en caso de que exista más de un stream definido en la query ingresada y se autoincrementa el puerto hasta tener los sockets requeridos para simular todos los streams necesarios en la prueba.

Con las clases de las Fig. 37 y 38 se definen las ontologías encargadas de tomar las líneas de los archivos que tienen la información en bruto, transformarlas al formato RDF y enviarlas a los sockets respectivos a medida que estos lo soliciten.

F. Pruebas

De acuerdo a lo planteado en esta investigación, el principal objetivo es realizar el procesamiento de streams de datos, en RDF con el API Stream de Apache Flink, para lo cual, como ya se ha mencionado anteriormente, fue necesario establecer un ambiente de pruebas, definiendo a Citybench como el principal eje de estas, al cual se le realizo una extensión especialmente pensada para desplegar sockets, que simulan los sensores. Finalmente, el Runner generado por el Mapper captura los nodos RDF transmitidos por los streams y resuelve el plan operacional según lo especifica la consulta involucrada, en la tabla VIII, se definen las características de los equipos usados en las pruebas.

TABLA VIII

CARACTERÍSTICAS EQUIPOS USADOS EN LAS PRUEBAS

Tipo	Cantidad	Descripción	Uso
Computador	1	Portátil Asus - Intel Core I7 - 14" - Disco sólido 1TB – RAM 16 GB	Pruebas en standalone y cluster
Computador	3	Portatil HP - Ryzen 3 - 14" - Disco duro 1TB - RAM 8 GB	Pruebas en cluster
Router	1	Router Repetidor Tenda Inalámbrico N301	Pruebas en cluster

Fuente: esta investigación.

G. Evaluación

Como producto de esta investigación se definió el uso de un benchmark, que en este caso se tomó la decisión de usar Citybench, ya que como se puede ver [38], [53], [54] este realiza pruebas sobre los principales, motores de procesamiento de stream de datos en RDF, conocidos hasta el momento. Y para realizar pruebas sobre el motor propuesto, se estableció un ambiente, en el cual se realizaran pruebas en standalone y cluster, en dicho ambiente se establecerá un host, en donde se realizará el despliegue de los streams a través del Citybench y se ejecutara el Runner por cada consulta en standalone y cluster, el cual realizara el procesamiento de la información, de los streams según esta lo indique, en el caso de cluster se definirá un maestro y hasta tres esclavos, en los cuales se repartirá el procesamiento, haciendo pruebas consecutivas por cada consulta, empezando por el master como standalone, incrementando sucesivamente los esclavos hasta llegar al cluster de tres. Como resultado de estas pruebas se obtendrá datos relacionados a la cantidad de datos obtenidos, uso de memoria y velocidad de procesamiento, comparándolos con CQELS, además de confrontar las mejoras obtenidas al empezar a incrementar uno a uno los nodos del cluster.

IV. RESULTADOS

A continuación, se expondrá los resultados obtenidos por cada query, especialmente diseñadas para el procesamiento de RDF streams, su plan operacional generado a través del Mapper (Runner) y la ponderación de los resultados obtenidos en el ambiente de ejecución empezando por standalone hasta llegar al cluster de un master y tres esclavos como máximo.

En las tablas VIII, XIX, X, XI y XII se definen los valores ponderados para cada prueba realizada, describiendo cada una de estas según el motor y en caso de ser posible la ejecución por cluster, el número de nodos utilizados. El tiempo de ejecución total, el cual se ve afectado por, la velocidad de consumo del stream, lo cual depende de la latencia de transferencia que se presente por red, en este caso la cual no tuvo efecto, ya que en standalone es el mismo equipo el que provee los datos y realiza el procesamiento y en cluster, se instaló una intranet cableada sin conexión a internet, lo cual, en sí, no debe afectar la transferencia. En el caso del consumo de memoria es un parámetro importante en los datos obtenidos, ya que, uno de los principales motivos por el cual se usó Apache Flink, es por la implementación de técnicas de manejo de memoria como swapping. Y finalmente se establece la cantidad de datos obtenidos como resultado para verificar la correcta respuesta por parte del motor propuesto, con respecto a uno ya probado y validado por la comunidad de procesamiento de stream datos en RDF, el cual es CQELS, reiteramos estas son pruebas empíricas teniendo en cuenta lo planteado en esta investigación.

En la consulta que se puede visualizar en la Fig. 45, se define la conexión a un stream de tráfico y se establece la combinación del valor a proyectar por cualquier tipo sin especificar y en la Fig. 46 se establece el plan operacional obtenido a partir de esta consulta, el cual define la captura de los nodos a través del stream y para realizar posteriormente, la devolución de los resultados del valor proyectado, que en este caso en particular es “?obId1”, como se define en la misma, en intervalos de tres (3) segundos.

```
cqels_query > Q1-A.txt
1  SELECT ?obId1
2
3  WHERE {
4      STREAM <http://www.udenar.edu.co/dataset#AarhusTrafficData182955> [range 3s]
5      {
6          ?obId1 a ?ob
7      }
8  }
9
```

Fig. 45. Consulta 1-A.

Fuente: esta investigación.

Fig. 46. Plan lógico operacional consulta 1-A.

Fuente: esta investigación.

TABLA IX

PONDERACIÓN RESULTADOS CONSULTA 1-A

MOTOR	No. NODOS	TIEMPO	MEM. RAM	MEM. EXT	No. RES
Propuesta	-	1m 28s	14,797 MB	567,203 MB	296.875
Propuesta	1	1m 28s	14,797 MB	567,203 MB	296.875
Propuesta	2	1m 5s	26,861 MB	555,139 MB	296.875
Propuesta	3	55s	31,746 MB	550,746 MB	296.875
CQELS	-	1m 53s	582 MB	0	296.875

Fuente: esta investigación.

La consulta que se puede visualizar en la Fig. 45, es una extensión a la de la Fig. 43 con el fin de agregar complejidad sucesivamente adicionándole, que sea cualquier valor, que contenga cualquier propiedad de observación, perteneciente al tráfico y finalmente, realizar la devolución de los resultados de los valores proyectados, que en este caso en particular son “?obId1” y “?v1”, como se define en la misma.

```
cqels_query > Q1-B.txt
1  SELECT ?obId1 ?v1
2
3  WHERE {
4      STREAM <http://www.udenar.edu.co/dataset#AarhusTrafficData182955> [range 3s]
5      {
6          ?obId1 a ?ob.
7          ?obId1 <http://purl.oclc.org/NET/ssnx/ssn#observedProperty> ?p1.
8          ?obId1 <http://purl.oclc.org/NET/sao/hasValue> ?v1.
9          ?obId1 <http://purl.oclc.org/NET/ssnx/ssn#observedBy> <http://www.udenar.edu.co/dataset#AarhusTrafficData182955>
10     }
11 }
12
```

Fig. 47. Consulta 1-B.

Fuente: esta investigación.

En la Fig. 48 se define el plan operacional de la continuación de la consulta “Q1” mostrada en la Fig. 47.

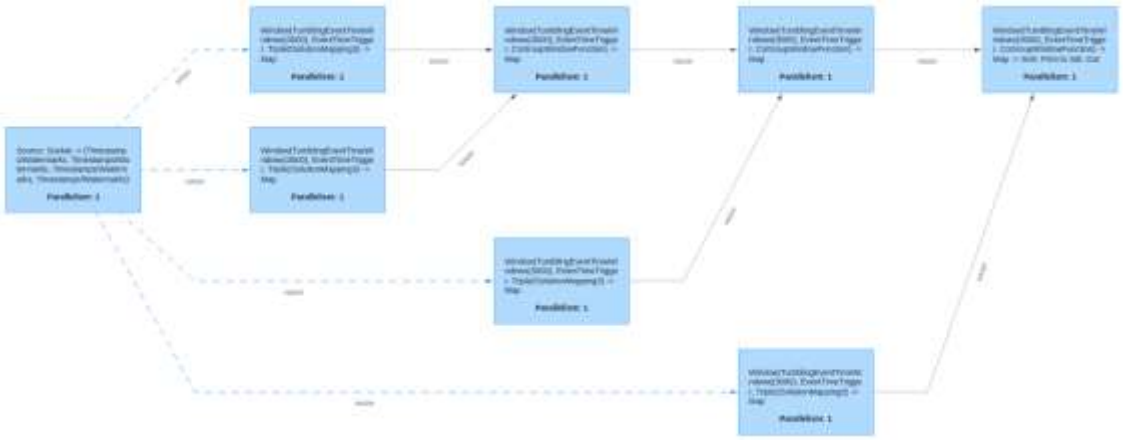

Fig. 48. Plan lógico consulta 1-B.

Fuente: esta investigación.

TABLA X

PONDERACIÓN RESULTADOS CONSULTA 1-B

MOTOR	No. NODOS	TIEMPO	MEM. RAM	MEM. EXT	No. RES
Propuesta	-	1m 44s	67,052 MB	2,205 GB	197.802
Propuesta	1	1m 44s	67,052 MB	2,205 GB	197.802
Propuesta	2	1m 22s	85,043 MB	2,187 GB	197.802
Propuesta	3	1m 3s	110,690 MB	2,161GB	197.803
CQELS	-	5m 24s	2,270 GB	0	197.802

Fuente: esta investigación.

En la Fig. 49 se muestra la redacción completa para la consulta 1, en la cual se adiciona un segundo stream, que se conecta a otro dataset de tráfico disponible y se establece, que se filtre, por cualquier tipo, para cualquier valor, que contenga cualquier propiedad de observación, perteneciente al tráfico y al final realiza la proyección de los valores definidos como “?obId1”, “?obId2”, “?v1” y “?v2”, en intervalos de tres (3) segundos para cada stream.

```
cqels_query > Q1-C.txt
1  SELECT ?obId1 ?obId2 ?v1 ?v2
2
3  WHERE {
4      STREAM <http://www.udenar.edu.co/dataset#AarhusTrafficData182955> [range 3s]
5      {
6          ?obId1 a ?ob.
7          ?obId1 <http://purl.oclc.org/NET/ssnx/ssn#observedPropesty> ?p1.
8          ?obId1 <http://purl.oclc.org/NET/sao/hasValue> ?v1.
9          ?obId1 <http://purl.oclc.org/NET/ssnx/ssn#observedBy> <http://www.udenar.edu.co/dataset#AarhusTrafficData182955>.
10     }
11     STREAM <http://www.udenar.edu.co/dataset#AarhusTrafficData158585> [range 3s]
12     {
13         ?obId2 a ?ob.
14         ?obId2 <http://purl.oclc.org/NET/ssnx/ssn#observedPropesty> ?p2.
15         ?obId2 <http://purl.oclc.org/NET/sao/hasValue> ?v2.
16         ?obId2 <http://purl.oclc.org/NET/ssnx/ssn#observedBy> <http://www.udenar.edu.co/dataset#AarhusTrafficData158585>
17     }
18 }
19
```

Fig. 49. consulta 1-C.

Fuente: esta investigación.

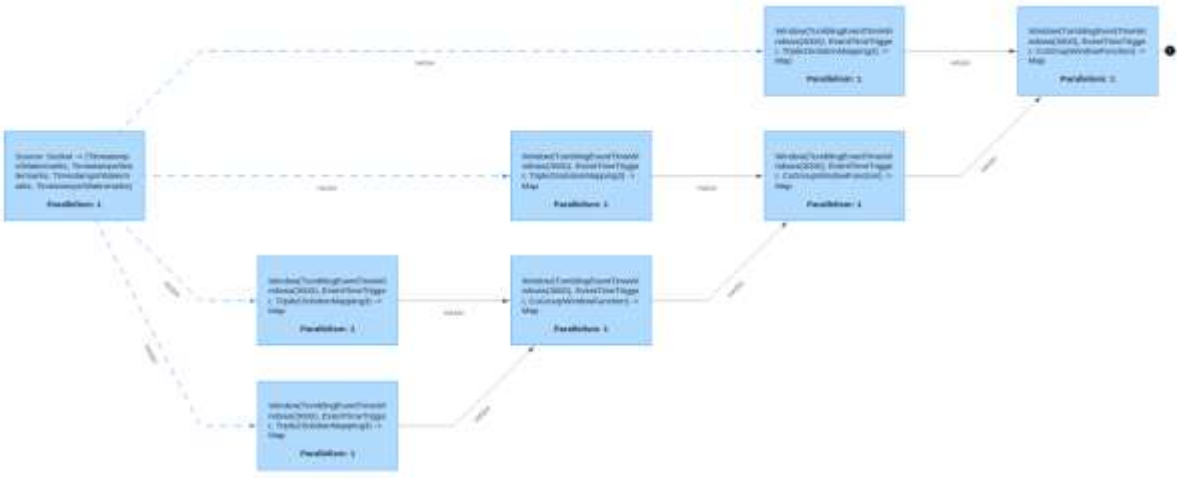

Fig. 50. Plan lógico consulta 1-C, 1ra parte.

Fuente: esta investigación.

En las Fig. 50 y 51, divididas en dos partes, se define el plan operacional de la continuación ya completa de la consulta “Q1” mostrada en la Fig. 49, se ilustra con líneas punteadas las

transferencias de datos, que realizan los streams en tiempo de ejecución, en este caso para los dos sockets.

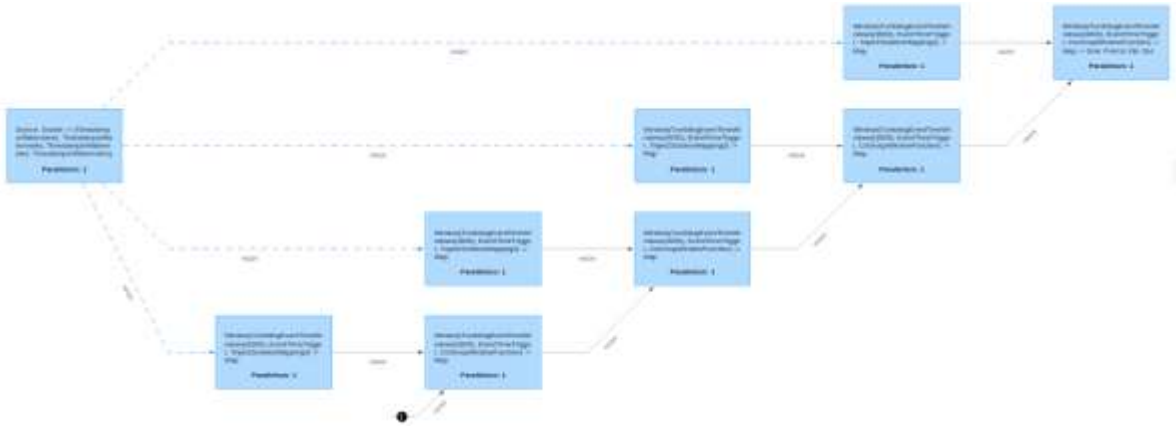

Fig. 51. Plan lógico consulta 1-C, 2da parte.

Fuente: esta investigación.

Como se puede ver en las Fig. 50 y 51 se define un punto (1) de conexión hacia la transición que realiza el programa para definir el cruce entre los dos streams definidos en la consulta.

TABLA XI

PONDERACIÓN RESULTADOS CONSULTA 1-C

MOTOR	No. NODOS	TIEMPO	MEM. RAM	MEM. EXT	No. RES
Propuesta	-	3m 29s	69,818 MB	4,681 GB	17.340
Propuesta	1	3m 29s	69,818 MB	4,681 GB	17.340
Propuesta	2	3m 5s	78,875 MB	4.673 GB	17.337
Propuesta	3	2m 40s	91,201 MB	4.661 GB	17.337
CQELS	-	10m 4s	4,750 GB	0	17.337

Fuente: esta investigación.

La consulta de la Fig. 52, es la más compleja, de las pruebas realizadas y su plan operacional está definido por las Fig. 53 a la 56. En este caso se define una consulta que espera proyectar los valores descritos como “?obId1”, “?obId2”, “?obId3”, “?obId4” , “?v1” , “?v2” , “?v3” y “?v4”. Que cruza dos streams, de los cuales, el primero es un stream simulado por datasets con datos climáticos y el segundo es un stream que simula datos de un dataset de tráfico, de la ciudad inteligente Aarhus.

El primer stream de la consulta mostrada en Fig. 52, define la extracción de tres variables por cualquier tipo, en donde "?obId1" debe ser una variable que observe una propiedad de datos de temperatura, la variable "?obId2" debe observar una propiedad de datos de humedad y la variable "?obId3" debe observar una propiedad de datos de velocidad del viento y todas pueden ser cualquier valor que sea del clima, todo esto con valores tomados en intervalos de diez (10) segundos para ambos streams.

```
cqels_query > Q2.txt
    You, an hour ago | 1 author (You)
1   SELECT ?obId1 ?obId2 ?obId3 ?obId4 ?v1 ?v2 ?v3 ?v4
2
3   WHERE {
4       STREAM <http://www.udenar.edu.co/dataset#AarhusWeatherData0> [range 10s ]
5       {
6           ?obId1 a ?ob.
7           ?obId1 <http://purl.oclc.org/NET/ssnx/ssn#observedProperty> <http://www.insight-centre.org/citytraffic#Temperature>.
8           ?obId1 <http://purl.oclc.org/NET/sao/hasValue> ?v1.
9           ?obId1 <http://purl.oclc.org/NET/ssnx/ssn#observedBy> <http://www.udenar.edu.co/dataset#AarhusWeatherData0>.
10
11          ?obId2 a ?ob.
12          ?obId2 <http://purl.oclc.org/NET/ssnx/ssn#observedProperty> <http://www.insight-centre.org/citytraffic#Humidity>.
13          ?obId2 <http://purl.oclc.org/NET/sao/hasValue> ?v2.
14          ?obId2 <http://purl.oclc.org/NET/ssnx/ssn#observedBy> <http://www.udenar.edu.co/dataset#AarhusWeatherData0>.
15
16          ?obId3 a ?ob.
17          ?obId3 <http://purl.oclc.org/NET/ssnx/ssn#observedProperty> <http://www.insight-centre.org/citytraffic#WindSpeed>.
18          ?obId3 <http://purl.oclc.org/NET/sao/hasValue> ?v3.
19          ?obId3 <http://purl.oclc.org/NET/ssnx/ssn#observedBy> <http://www.udenar.edu.co/dataset#AarhusWeatherData0>.
20
21      }
22      STREAM <http://www.udenar.edu.co/dataset#AarhusTrafficData158505> [range 10s]
23      {
24          ?obId4 a ?ob.
25          ?obId4 <http://purl.oclc.org/NET/ssnx/ssn#observedProperty> <http://www.insight-centre.org/citytraffic#CongestionLevel>.
26          ?obId4 <http://purl.oclc.org/NET/sao/hasValue> ?v4.
27          ?obId4 <http://purl.oclc.org/NET/ssnx/ssn#observedBy> <http://www.udenar.edu.co/dataset#AarhusTrafficData158505>.
28      }
29  }
30
```

Fig. 52. Consulta 2.

Fuente: esta investigación.

El segundo stream de la consulta de la Fig. 52, define la extracción de la variable "?obId4", que debe ser una variable que observe una propiedad de datos de congestión vehicular y puede ser cualquier valor que sea de tráfico. La finalidad de esta consulta, como tal es determinar condiciones climáticas y de tráfico, de las cual se verían afectados, los conductores de la ciudad inteligente Aarhus de la cual se tomó los datos.

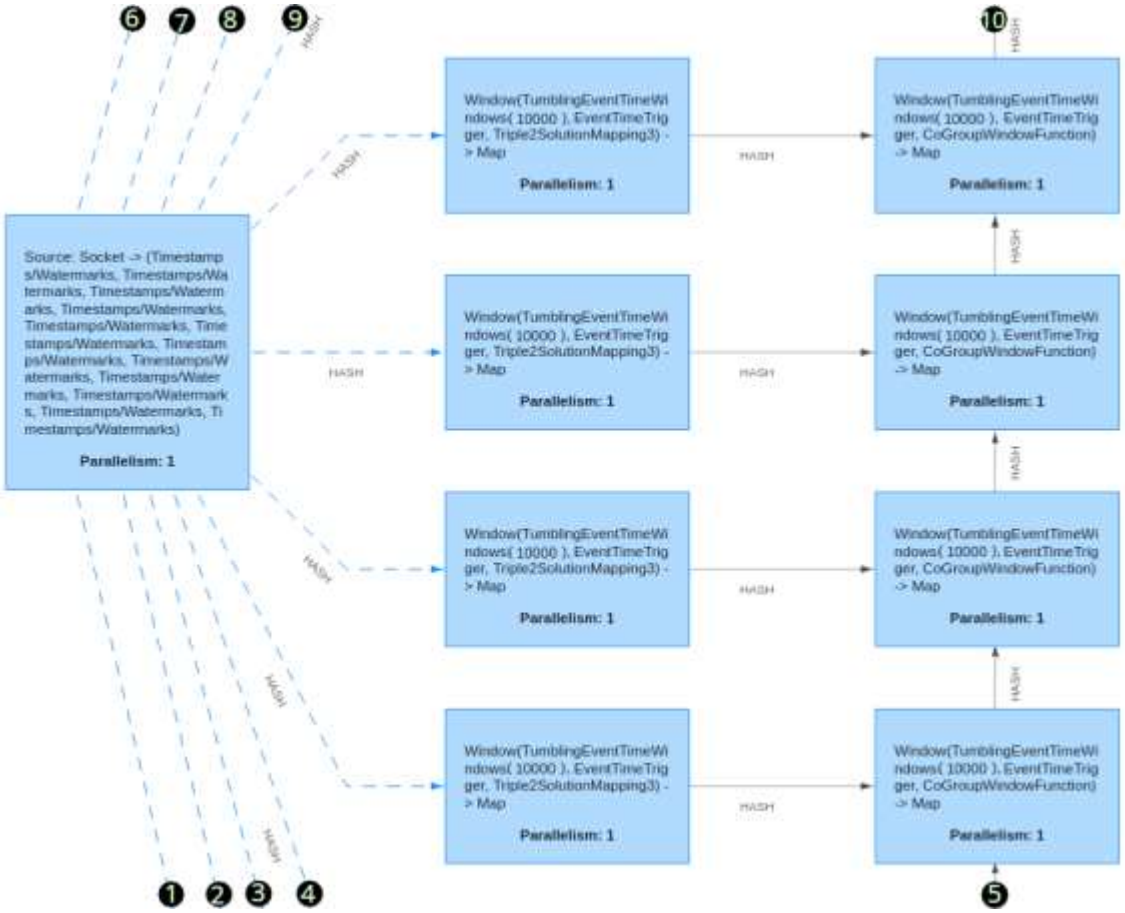

Fig. 53. Plan lógico consulta 2, 1ra parte.

Fuente: esta investigación.

Los puntos que se pueden visualizar en las Fig. 53 a la 55 (1-10), establecen la continuidad del plan operacional que corresponde a la consulta 2 en cuanto respecta al primer stream.

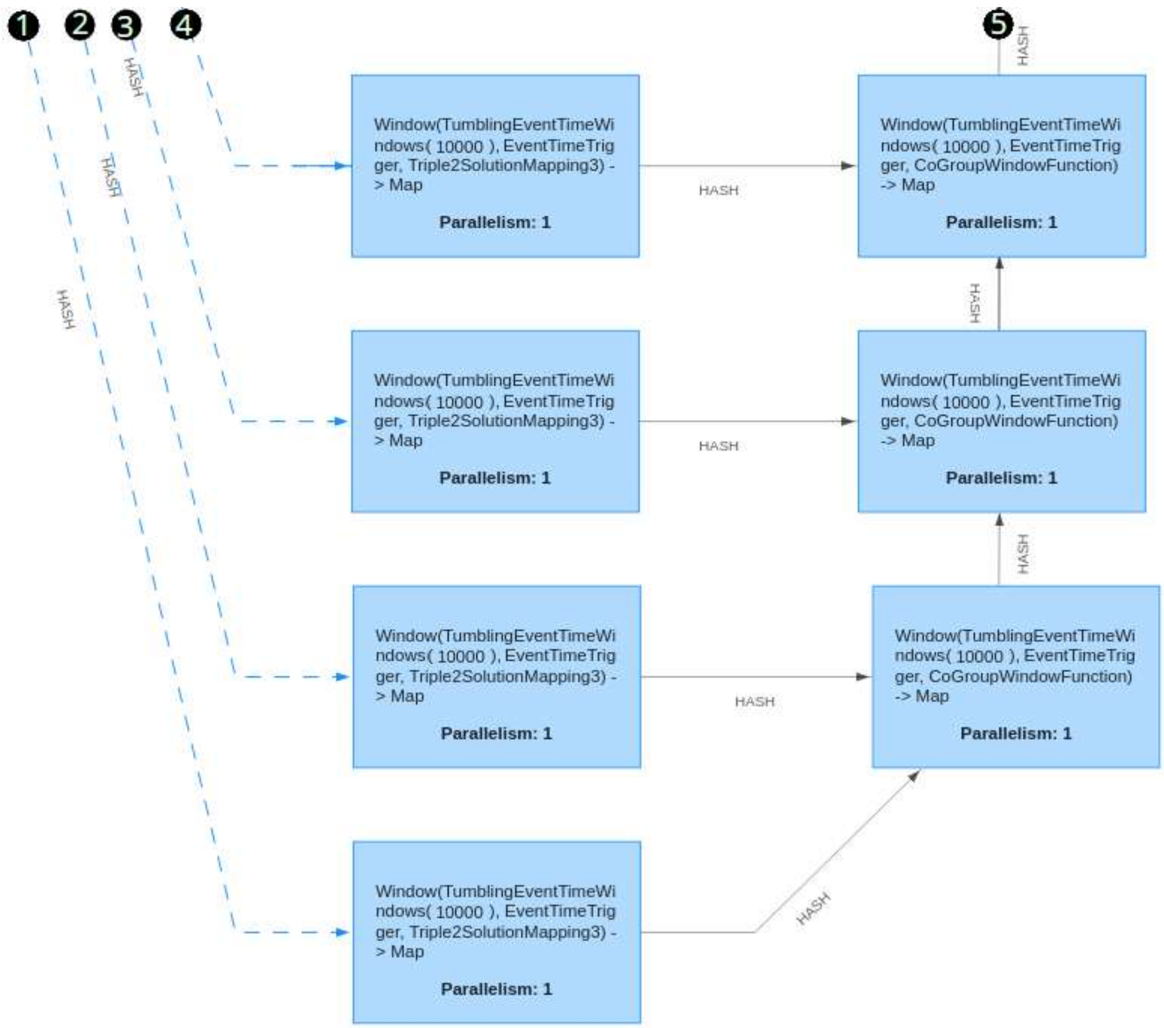

Fig. 54. Plan lógico consulta 2, 2da parte.

Fuente: esta investigación.

El punto (11) que puede visualizar en las Fig. 55 y 56, establece la transición en el cruce del stream con datos climáticos, con el stream de datos de tráfico.

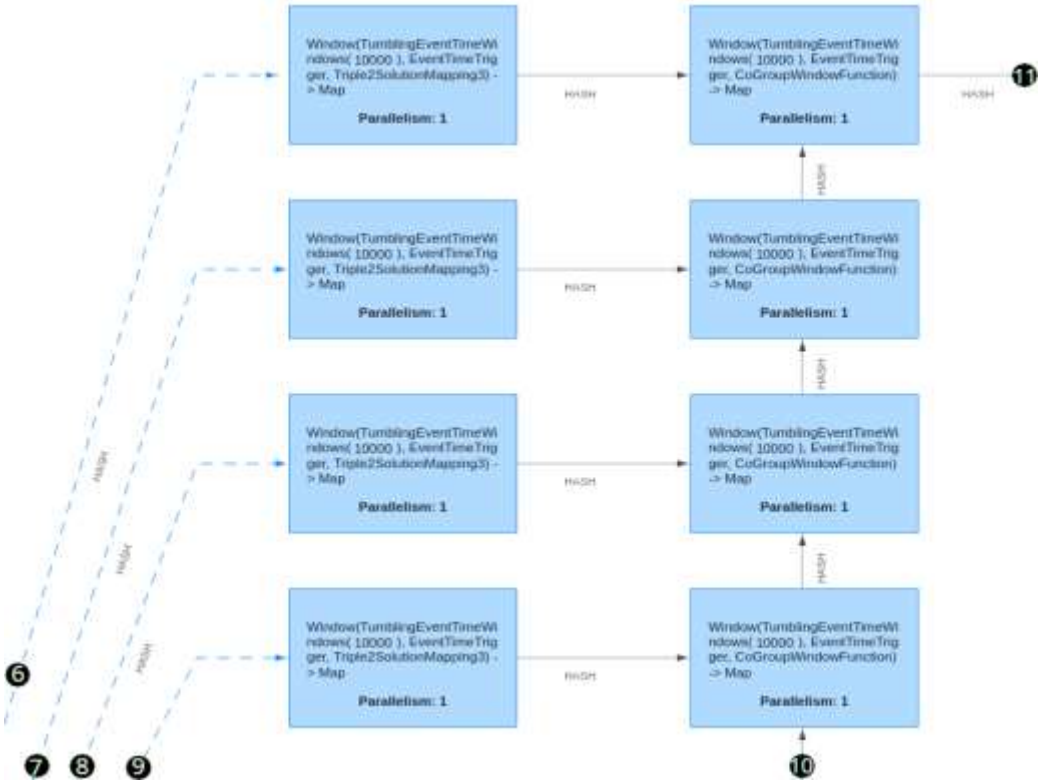

Fig. 55. Plan lógico consulta 2, 3ra parte.

Fuente: esta investigación.

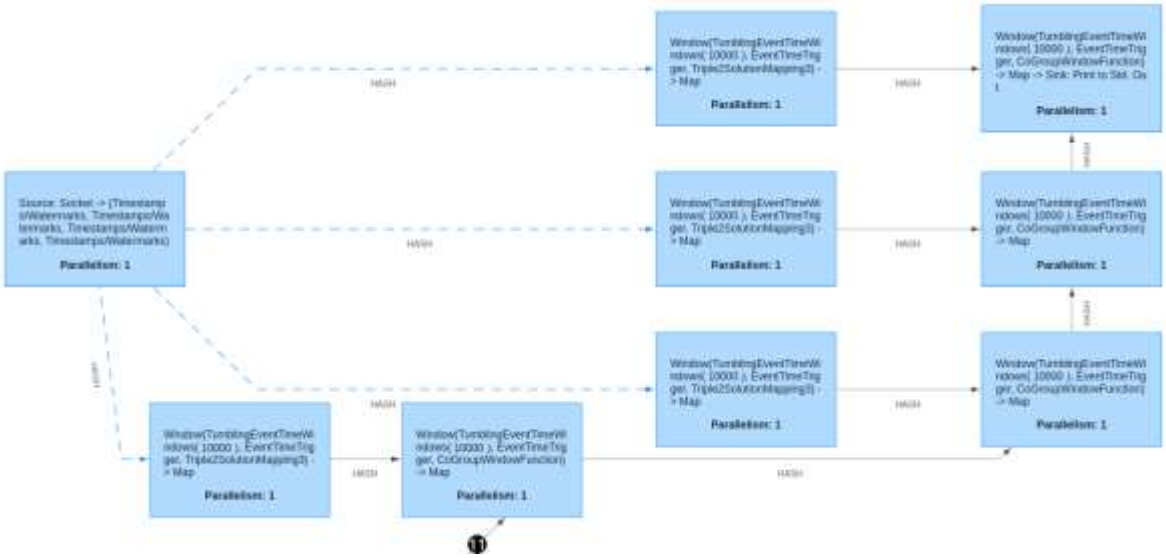

Fig. 56. Plan lógico consulta 2, 4ta parte.

Fuente: esta investigación.

TABLA XII

PONDERACIÓN RESULTADOS CONSULTA 2

MOTOR	No. NODOS	TIEMPO	MEM. RAM	MEM. EXT	No. RES
Propuesta	-	1m 38s	110 MB	2,58 GB	52.282
Propuesta	1	1m 38s	275,053 MB	2,4 GB	52.284
Propuesta	2	1m 23s	280,703 MB	2,39 GB	52.289
Propuesta	3	1m 2s	284,128 MB	2,37 GB	52.289
CQELS	-	3m 23s	2,67 GB	0	52.284

Fuente: esta investigación.

La consulta definida en la Fig. 57, realiza el cruce de información de dos streams con datos de parqueo, de los parqueaderos “KALKVAERKSVEJ” y “SKOLEBAKKEN”, donde se obtiene cualquier valor, que observe la propiedad espacio de parqueadero vacante, para cada parqueadero respectivamente y proyecta los valores de las variables “?obId1”, “?obId2”, “?v1” y “?v2”, todo esto en intervalos de cinco (5) segundos para el primer y segundo stream.

Fig. 57. Consulta 3.

Fuente: esta investigación.

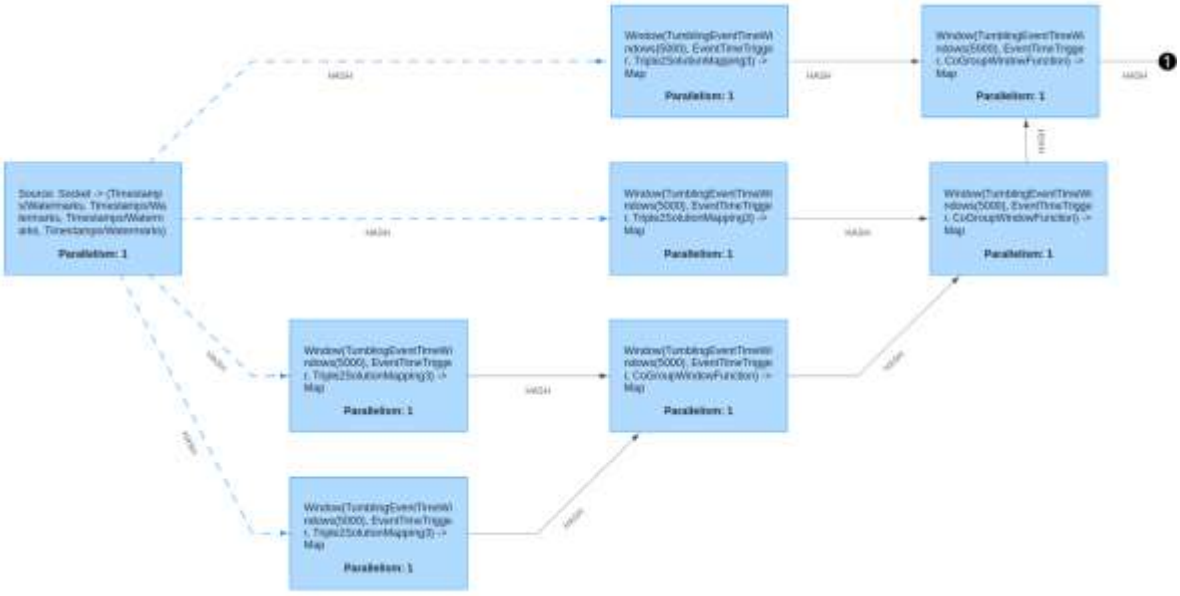

Fig. 58. Plan lógico consulta 3, 1ra parte.

Fuente: esta investigación.

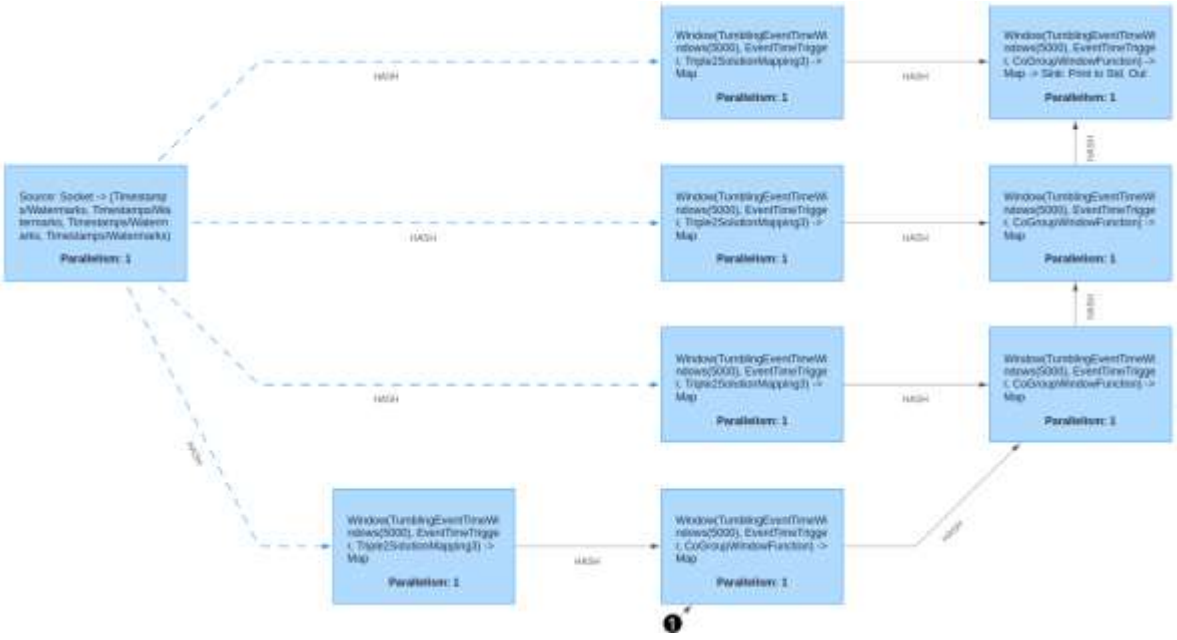

Fig. 59. Plan lógico consulta 3, 2da parte.

Fuente: esta investigación.

TABLA XIII

PONDERACIÓN RESULTADOS CONSULTA 3

MOTOR	No. NODOS	TIEMPO	MEM. RAM	MEM. EXT	No. RES
Propuesta	-	14s	79,286 MB	148,714 MB	27
Propuesta	1	14s	79,286 MB	148,714 MB	27
Propuesta	2	15s	85,385 MB	136,615 MB	27
Propuesta	3	18s	107,453 MB	114,547	27
CQELS	-	1m 22s	222 MB	0	27

Fuente: esta investigación.

V. CONCLUSIONES

Los resultados mostrados en las tablas IX, X, XI, XII y XIII, demuestran que el uso de Apache Flink, para procesar datos en RDF, genera mejoras sustanciales, en tiempo y memoria con respecto a otros motores, como por ejemplo CQELS, debido a las optimizaciones que tiene, en la gestión de memoria, usando la técnica de memoria swapping, en el momento de ejecución del plan de procesamiento de datos, lo cual hace que al dejar de usar únicamente memoria interna, no genere lentitud al momento de procesar, ya que, a pesar de estar leyendo y escribiendo, permanente los datos en disco duro, usa muy poco almacenamiento de memoria interna y por ello, este tipo de actividades no impactan al tiempo de ejecución.

Debido a lo anterior se demuestra efectividad, en la solución de las diferentes consultas planteadas, a través del plan de procesamiento que se genera, ya que al comparar las salidas obtenidas CQELS, los resultados de las pruebas empíricas realizadas fueron similares.

Se puede observar en las tablas IX, X, XI y XII, el uso de cluster, genera mejoras importantes en tiempo, a medida en que se incrementa los nodos, los tiempos de respuesta que se obtienen son menores, ya que el consumo y procesamiento de los datos a través del canal en tiempo real se realiza mucho más rápido, sin embargo, debido a la complejidad computacional que representa el incremento de nodos en un cluster, el uso de memoria RAM incrementa, además, que en casos en donde el volumen de datos es reducido, como por ejemplo los resultados mostrados en la tabla XIII, el tiempo aumenta al incrementar los nodos del cluster, haciendo que en estos casos específicos sea innecesario su uso.

También, debido a problemas de latencia, hay casos excepcionales en los cuales los resultados obtenidos no son iguales para las mismas consultas, como se puede ver en las tablas X, XI y XII, los cuales en si son mínimos y teóricamente posibles, ya que puede haber perdida al momento de capturar los datos por el canal que transmite los datos dinámicos.

Finalmente, se recomienda usar Apache Flink, para el procesamiento de datos dinámicos en formato RDF, ya que realiza gestión de memoria haciendo uso de la técnica swapping, además, permite un escalamiento horizontal, en donde a medida que el flujo de datos aumente, se puede

incrementar los nodos cuando se hace uso de cluster, siendo eficiente y eficaz al obtener los resultados en tiempo real.

BIBLIOGRAFÍA

[1] V. Gulisano, R. Jiménez-Peris, M. Patiño-Martínez, C. Soriente y P. Valduriez, «StreamCloud: An Elastic and Scalable Data Streaming System,» IEEE Transactions on Parallel and Distributed Systems, 2012. [En línea]. Available: https://bit.ly/3DJQEQq.

[2] L. Golab y T. Özsu, «Issues in data stream management,» *ACM SIGMOD Record,* vol. 32, nº 2, pp. 5-14, 2003.

[3] D. Terry, D. Goldberg, D. Nichols y B. Oki, «Continuous queries over append-only databases,» *SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pp. 321-330, 1992.

[4] M. Sullivan y A. Heybey, «Tribeca: a system for managing large databases of network traffic,» *ATEC '98: Proceedings of the annual conference on USENIX Annual Technical Conference*, p. 2, 1998.

[5] L. Liu, C. Pu y W. Tang, «Continual queries for Internet scale event-driven information delivery,» *IEEE Transactions on Knowledge and Data Engineering,* vol. 11, nº 4, pp. 610 - 628, 1999.

[6] J. Chen, D. J. DeWitt, F. Tian y Y. Wang, «NiagaraCQ: a scalable continuous query system for Internet databases,» *ACM SIGMOD Record*, vol. 29, nº 2, p. 379–390, 2000.

[7] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk y O. Spatscheck, «Gigascope: high performance network monitoring with an SQL interface,» *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data,* p. 623, 2002.

[8] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul y S. Zdonik, «Aurora: a new model and architecture for data stream management,» *The VLDB Journal*, vol. 12, p. 120–139, 2003.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss y M. Shah, «TelegraphCQ: continuous dataflow processing,» *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, p. 668, 2003.

[10] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein y J. Widom, «STREAM: the stanford stream data manager (demonstration description),» *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, p. 665, 2003.

[11] Y. Bai, H. Thakkar, H. Wang, C. Luo y C. Zaniolo, «A data stream language and system designed for power and extensibility,» *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, p. 337–346, 2006.

[12] W3C, «W3C Semantic Sensor Network Incubator Group,» Incubator Activity , 2011. [En línea]. Available: https://bit.ly/30lzKZV.

[13] C. Bizer, T. Heath y T. Berners-Lee, «Linked Data: The Story so Far,» *International journal on Semantic Web and information systems*, vol. 5, pp. 1-22, 2009.

[14] J. Sequeda y O. Corcho, «Linked Stream Data: A Position Paper,» *Linked Stream Data: A Position Paper Juan F. Sequeda1, Oscar Corcho2 1 Department of Computer Sciences. University of Texas at Austin 2Ontology Engineering Group. Departamento de Inteligencia Artificial. Universidad Politécnica de Madrid, Spain jsequeda@*, pp. 149-157, 2009.

[15] RDF Working Group, «Resource Description Framework (RDF),» W3C Semantic Web, 2014. [En línea]. Available: https://bit.ly/2YFsrvK.

[16] A. Bolles, M. Grawunder y J. Jacobi, «Streaming SPARQL extending SPARQL to process data streams,» *ESWC'08: Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, p. 448–462, 2008.

[17] D. Barbieri, D. Braga, S. Ceri y M. Grossniklaus, «An execution environment for C-SPARQL queries,» *EDBT '10: Proceedings of the 13th International Conference on Extending Database Technology*, p. 441–452, 2010.

[18] D. Anicic, P. Fodor, S. Rudolph y N. Stojanovic, «EP-SPARQL: a unified language for event processing and stream reasoning,» *WWW '11: Proceedings of the 20th international conference on World wide web*, p. 635–644, 2011.

[19] J.-P. Calbimonte, O. Corcho y A. Gray, «Enabling Ontology-Based Access to Streaming Data Sources,» *The Semantic Web – ISWC*, pp. 96-111, 2010.

[20] M. Rinne, E. Nuutila y S. Törmä, «INSTANS: high-performance event processing with standard RDF and SPARQL,» *ISWC-PD'12: Proceedings of the 2012th International Conference on Posters & Demonstrations Track*, vol. 914, 2012.

[21] D. Le-Phuoc, J. Parreira y M. Hauswirth, «Challenges in Linked Stream Data Processing: A,» *Digital Enterprise Research Institute*, 2011.

[22] D. Le-Phuoc, H.-N. Mau-Quoc, C. L. Van y M. Hauswirth, «Elastic and Scalable Processing of Linked Stream Data in the Cloud,» *The Semantic Web – ISWC*, pp. 280-297, 2013.

[23] A. B. M. Moniruzzaman y S. Hossain, «NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison,» *dblp computer science bibliography,* 2013.

[24] IBM, P. Zikopoulos y C. Eaton, Understanding big data: Analytics for enterprise class hadoop and streaming data, Osborne Media: McGraw-Hill, 2011.

[25] M. Assunçãoa, R. Calheiros, S. Bianchi, M. Netto y R. Buyya, «Big Data computing and clouds: Trends and future directions,» *Journal of Parallel and Distributed Computing,* vol. 79, pp. 3-15, 2015.

[26] J. Dean y S. Ghemawat, «MapReduce: simplified data processing on large clusters,» *Communications of the ACM,* vol. 51, nº 1, p. 107–113, 2008.

[27] The Apache Software Foundation, «The Apache Hadoop software library,» Apache Hadoop, 2008. [En línea]. Available: https://bit.ly/3lxcnED.

[28] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. Henry, R. Bradshaw y N. Weizenbaum, «FlumeJava: easy, efficient data-parallel pipelines,» *PLDI '10: Proceedings of the 31st ACM SIGPLAN,* p. 363–375, 2010.

[29] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. Patel, K. Ramasamy y S. Taneja, «Twitter Heron: Stream Processing at Scale,» *SIGMOD '15: Proceedings of the 2015 ACM SIGMOD,* p. 239–250, 2015.

[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker y I. Stoica, «Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing,» *NSDI'12: Proceedings of the 9th USENIX,* p. 2, 2012.

[31] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker y I. Stoica, «Discretized streams: fault-tolerant streaming computation at scale,» *SOSP '13: Proceedings of the Twenty-Fourth ACM,* p. 423–438, 2013.

[32] P. Carbone, G. Fóra, S. Ewen, S. Haridi y K. Tzoumas, «Lightweight Asynchronous Snapshots for Distributed Dataflows,» *Computer Science,* pp. 8-15, 2015.

[33] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt y S. Whittle, «The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing,» *VLDB Endowment,* vol. 8, pp. 1792-1803, 2015.

[34] W3C Community & Business Group, « RDF Stream Processing Community Group,» 2015. [En línea]. Available: https://bit.ly/2YLYUAc.

[35] W3C, «W3C/SMPTE Workshop on Professional Media Production on the Web,» 2021. [En línea]. Available: https://bit.ly/3oZ8cDM.

[36] E. Prud'hommeaux y A. Seaborne, «SPARQL Query Language for RDF,» 2008. [En línea]. Available: https://bit.ly/3AyW26W.

[37] D. Le-Phuoc, M. Dao-Tran, J. Parreira y M. Hauswirth, «A native and adaptive approach for unified processing of linked streams and linked data,» *ISWC'11: Proceedings of the 10th international conference on The semantic web,* vol. 1, p. 370–388, 2011.

[38] D. Le-Phuo, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter y M. Fink, «Linked Stream Data Processing Engines: Facts and Figures,» *The Semantic Web,* pp. 300-312, 2012.

[39] «Cloudera,» 2021. [En línea]. Available: https://bit.ly/3atNbsz.

[40] Cloudera, «Cloudera Completes Agreement To Become a Private Company,» cloudera.com, 2021. [En línea]. Available: https://bit.ly/3iUS0j9.

[41] Apache HBase, «Welcome to Apache HBase,» 2008. [En línea]. Available: https://bit.ly/2YJkDIZ.

[42] Hive, «The Apache Hive ™,» 2011. [En línea]. Available: https://bit.ly/3iVejVL.

[43] Apache ZooKeeper™, «Welcome to Apache ZooKeeper™,» 2020. [En línea]. Available: https://bit.ly/3lA1Okr.

[44] The Apache Software Foundation., «Welcome to Apache Avro!,» 2012. [En línea]. Available: https://bit.ly/3DETAOb.

[45] The Apache Software Foundation, «Welcome to Apache Pig,» 2018. [En línea]. Available: https://bit.ly/3mO1wFV.

[46] The Apache Software Foundation, «Welcome to Apache Flume,» 2009. [En línea]. Available: https://bit.ly/3Dv1aKU.

[47] Apache Software Foundation, «Apache Storm Version: 2.3.0,» 2021. [En línea]. Available: https://bit.ly/3axHKJ6.

[48] N. Marz, «History of Apache Storm and lessons learned,» 2014. [En línea]. Available: https://bit.ly/3va7Mvl.

[49] The Apache Software Foundation, «Unified engine for large-scale data analytics,» 2018. [En línea]. Available: https://bit.ly/3mWxZdo.

[50] The Apache Software Foundation, «Apache Flink® —Stateful Computations over Data Streams,» 2014. [En línea]. Available: https://bit.ly/3BE61Jb.

[51] Google Cloud, «Acelera tu transformación con Google Cloud,» 2021. [En línea]. Available: https://bit.ly/3n7UcoX.

[52] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom y S. Whittle, «MillWheel: fault-tolerant stream processing at internet scale,» *Proceedings of the VLDB Endowment*, vol. 6, nº 11, p. 1033–1044, 2013.

[53] L. Danh, D. Minh, L. Anh, N. Manh y H. Manfred, «RDF Stream Processing with CQELS Framework for Real-time Analysis,» *DEBS Grand Challenge*, 2015.

[54] I. Muhammad, G. Feng y M. Alessandra, «CityBench: A Configurable Benchmark to Evaluate RSP Engines Using Smart City Datasets,» *Insight Centre for Data Analytics, National University of Ireland*, 2015.

[55] D. Luckham, «Rapide: A language and toolset for simulation of distributed systems by partial orderings of events,» Stanford, California, USA, 1996.

[56] M. Mansouri-Samani, M. Sloman y M. Sloman, «Gem - a generalised event monitoring language for distributed systems,» *IEE/IOP/BCS Distributed Systems Engineering Journal*, vol. 4, 1997.

[57] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte y W. White, «Cayuga: a high-performance event processing engine,» *SIGMOD '07*, p. 1100–1102, 2007.

[58] N. Schultz-Møller, M. Migliavacca y P. Pietzuch, «Distributed complex event processing with query rewriting,» *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems,* p. 1–12, 2009.

[59] G. Cugola y A. Margara, «TESLA: a formally defined event specification language,» *DEBS '10: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems,* p. 50–61, 2010.

[60] Oracle, «Oracle Stream Analytics Documentation,» Oracle Stream Analytics, 2015. [En línea]. Available: https://bit.ly/3j5zOUa.

[61] SAP, «Rise with SAP,» 2015. [En línea]. Available: https://bit.ly/3lJwgbC.

[62] TIBCO, «Scale Data Science across your Enterprise and Act in Real Time,» 2015. [En línea]. Available: https://bit.ly/2YVK7TN.

[63] IBM, «IBM Streams,» 2015. [En línea]. Available: https://ibm.co/3BKNtXR.

[64] SPHERE, «The Sphere community sets standards for humanitarian action and promotes quality and accountability.,» 2015. [En línea]. Available: https://bit.ly/3p4DR74.

[65] TinyDB, «TinyDB A Declarative Database for Sensor Networks,» 2015. [En línea]. Available: https://bit.ly/3BIwGox.

[66] Y. Yao y J. Gehrke, «The cougar approach to in-network query processing in sensor networks,» *ACM SIGMOD Record,* vol. 31, nº 3, p. 9–18, 2002.

[67] I. Galpin, C. Brenninkmeijer, F. Jabeen, A. Fernandes y N. Paton, «Comprehensive Optimization of Declarative Sensor Network Queries,» *Scientific and Statistical Database Management,* pp. 339-360, 2009.

[68] P. Gibbons, B. Karp, Y. Ke, S. Nath y S. Seshan, «IrisNet: an architecture for a worldwide sensor Web,» *IEEE Pervasive Computing,* vol. 2, nº 4, pp. 22 - 33, 2003.

[69] H. Paulino y J. Santos, «A Middleware Framework for the Web Integration of Sensor Networks,» *Sensor Systems and Software,* pp. 75-90, 2010.

[70] K. Aberer, M. Hauswirth y A. Salehi, «A middleware for fast and flexible sensor network deployment,» *VLDB '06: Proceedings of the 32nd international conference on Very large data bases,* p. 1199–1202, 2006.

[71] M. Botts, G. Percivall, C. Reed y J. Davidson, «OGC® Sensor Web Enablement: Overview and High Level Architecture,» *GeoSensor Networks,* pp. 175-190, 2006.

[72] «SStreaMWare: a service oriented middleware for heterogeneous sensor data management».

[73] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro y V. Olive, «SStreaMWare: a service oriented middleware for heterogeneous sensor data management,» *ICPS '08: Proceedings of the 5th international conference on Pervasive services,* p. 121–130, 2008.

[74] Hourglass, «SYRAH Systems Research at Harvard,» 2015. [En línea]. Available: https://bit.ly/3DCk02M.

[75] K. Whitehouse, F. Zhao y J. Liu, «Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data,» *Wireless Sensor Networks,* pp. 5-20, 2006.

[76] E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, A. Riabov y F. Ye, «A semantics-based middleware for utilizing heterogeneous sensor networks,» *DCOSS'07: Proceedings of the 3rd IEEE international conference on Distributed computing in sensor systems,* p. 174–188, 2007.

[77] A. Sheth, C. Henson y S. Sahoo, «Semantic Sensor Web,» *IEEE Internet Computing*, vol. 12, nº 4, pp. 78 - 83, 2008.

[78] J. Broekstra, A. Kampma y F. v. Harmelen, «Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema,» *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, p. 54–68, 2002.

[79] S. Das, S. Sundara y R. Cyganiak, «R2RML: RDB to RDF Mapping Language,» 2012. [En línea]. Available: https://bit.ly/2YNEXbT.

[80] AWS, «Amazon EC2 Capacidad informática segura y de tamaño ajustable que admite prácticamente cualquier carga de trabajo,» 2021. [En línea]. Available: https://go.aws/3FL0aEn.

[81] P. Szekely, C. Knoblock, J. Slepicka, A. Philpot, A. Singh, C. C Yin y LFerreira, «Building and using a knowledge graph to combat human trafficking,» *The semantic web*, pp. 205-221, 2015 .

[82] A. Arasu, S. Babu y J. Widom, «The CQL continuous query language: semantic foundations and query execution,» *The VLDB Journal*, vol. 15, p. 121–142, 2006.

[83] K. Patroumpas y T. Sellis, «Window specification over data streams,» *Proceedings of the 2006 international conference on current trends in database technology*, p. 445–464, 2006.

[84] Oracle Corporation and/or its affiliates, «Tuning the HTTP Service,» 2010. [En línea]. Available: https://bit.ly/3lBKRWM.