Implementación de un Ambiente de Ciencia de Datos a través de la Librería Open Source RAPIDS

Guarnizo Romero José Alberto
Ánalisis de Información
Universidad Técnica Particular de Loja
Loja, Ecuador
jaguarnizo4@utpl.edu.ec

***Abstract*—This document focuses on an analysis of the RAPIDS library created by NVIDIA, applying its main functions, features and libraries, in addition, it handles large volumes of information in real time and works with accelerated computing, RAPIDS has the focus to analyze, process and implement Machine Learning algorithms to end-to-end dataset on GPU.**

***Index Terms*—RAPIDS, NVIDIA, GPU, dataset, Machine Learning, accelerated computing.**

I. Introducción

La cantidad de datos producidos en la actualidad son de gran volumen, es creciente en instituciones públicas y privadas, producen millones de datos al día, empresas como: bancos, entidades del estado, negocios independientes, entre otros, aportan considerables cantidades de información que se encuentran almacenadas en BD (bases de datos), cloud y otros medios con la finalidad de almacenarlas y respaldarlas, lo que causa que no tenga un adecuado tratamiento en cuanto a su procesamiento, implementación y utilización para generar valor y rápidez.

La implementación de un ambiente de ciencia de datos a través de la Librería Open Source RAPIDS, se enfoca en analizar un conjunto de datos abiertos utilizando las principales características de la librería.

Mediante un análisis completo de la librería RAPIDS permitirá manejar grandes volúmenes de información en tiempo real, además, poder determinar y hacer uso de las características más importantes de la librería que permiten: ocupar al máximo los recursos disponibles, tiempos mínimos de latencia, tolerancia a fallos y capacidad para el análisis, procesamiento e implementación de algoritmos de Machine Learning a los datos.

II. Marco Teórico

A. Ciencia de Datos

La ciencia de datos es parte fundamental de todas las ciencias de aprendizaje automático y profundo, la inteligencia artificial y de negocios, además, involucra métodos científicos, procesos y sistemas para extraer conocimiento para un mejor entendimiento de los datos y proporcionar que los mismos sean útiles, entiende y permite analizar los datos desde un punto de vista comercial, el principal objetivo es proporcionar la predicción más precisa.

Cabe destacar que la misma a tenido un sin número de definiciones y se lo relaciona muy a menudo con la Big Data, he visto conveniente incorporar su concepto y principales enfoques.

B. Big Data

Los grandes volúmenes de datos han ido creciendo de modo exponencial. La Big Data está brotando por todas partes utilizándolos adecuadamente, una gran ventaja competitiva a las organizaciones tanto internacionales como locales. En general existen diferentes aspectos donde casi todas las decisiones están de acuerdo y con conceptos consistentes para capturar la esencia de lo que es Big Data: Se interpreta como crecimiento exponencial de grandes volúmenes de datos, origen o fuentes de datos y la necesidad de su captura, almacenamiento y análisis para conseguir el mayor beneficio para organizaciones y empresas junto con las oportunidades que ofrecen y los riesgos de su no adopción[1]. La Big Data puede variar de distintas formas, según las características de cada organización, es decir, para algunas lo primero que importa es capturar la información la cual va hacer procesada, guardada y actualizada para después incorporarla en procesos de negocios que puedan ser beneficiosos, para otros, interesa que la información sea guardada en tiempo real; como enfoque cada organización busca el beneficio de la misma.

C. Tipos de Datos

a) Datos Estructurados: Se consideran como datos tradicionales, datos con formato y esquema fijo. En estos campos son datos de BD relacionales, hojas de cálculo y archivos. Los datos estructurales se componen por distintas piezas de información, también viene en un formato ya especificado, además, estos formatos son conocidos como: fecha, nacionalidad, cédula, entre otras[1].

b) Datos Semiestructurados: Tienen un flujo que pueden ser definidos, pero no es fácil de entender para el usuario, es decir, no tienen formatos fijos, así como los datos estructurados. La lectura de los datos requiere el uso de reglas complejas que determinan como proceder después de la lectura de cada pieza de información, ejemplo de datos semiestructurados son los archivos de logs[1].

c) Datos no Estructurados: Son datos sin tipos predefinidos, se almacenan como documentos u objetos sin estructura, se tiene un poco o ningún control sobre ellos, ejemplo de estos datos tenemos: texto, video, audio y fotografía. El 80% de la información que tienen las organizaciones corresponde a este tipo de datos.

D. Procesamiento de Datos

El procesamiento de Datos es utilizado por organizaciones, permiten tener sistemas de grandes transacciones de un sin número de datos. Además, el procesamiento es en tiempo real, se debe tener en cuenta que el tipo de procesamiento es instantáneo, esto quiere decir que el dato llega, se procesa y se da una salida automática. Los datos pueden ser como números, caracteres y son representados por distintos valores, pueden ser manipulados en distintas formas para que sean útiles y comprensibles, convirtiendo los datos en información. Los datos realizan un proceso que se denomina fase de procesamiento de datos:

a) Entrada de Datos: Los datos sin procesar comienzan a tomar forma de información utilizable, la mayoría de las tareas de entrada de datos consumen mucho tiempo, sin embargo, se considera una tarea básica y necesaria.

b) Preparación de Datos: La preparación de los datos, a menudo se lo denomina como "preprocesamiento", los datos sin procesar se limpian y organizan.

c) Proceso: Se ejecutan operaciones precisas para cambiar los datos en información importante.

E. Procesamiento de Datos para llegar a RAPIDS

La invención de las computadoras es una clara necesidad de información y procesamiento, los informáticos tuvieron que escribir programas personalizados para procesar datos. El proceso de evolución de datos determinara de manera gratificante como ha ido evolucionando, hasta llegar al proceso de construcción de la librería RAPIDS.

a) Distributed Storage: Proporcionan acceso confiable a los datos a través de la redundancia distribuida en nodos que son confiables individualmente. Se centra en Hadoop y MapReduce que permiten el almacenamiento distribuido y el procesamiento de múltiples máquinas. Hadoop (ver figura 1) está diseñado para trabajar y ejecutarse en una gran cantidad de máquinas que no comparten memoria ni disco, rompe los datos en pedazos que se extiendan a través de los diferentes servidores, realizan un respectivo seguimiento donde residen los datos, pueden ejecutar su trabajo de indexación enviando su código a cada uno de los servidores y cada servidor opera en su propia pequeña porción de datos[2].

La figura 1 representa el proceso de trabajo Hadoop, HDFS Read (archivos de lectura) se lee el archivo, pasa a Query interpretado como una cadena de consulta en las respectivos BD y se escriben en los HDFS Write (archivos de escritura), nuevamente pasa por HDFS Read, el archivo leído pasa por ETL (extracción, transformación y carga) es un proceso de reformateo, limpieza; nuevamente terminado el proceso de ETL se reescribe en los HDFS Write; se lee los HDFS Read nuevamente, para enfocarse en el proceso de ML (Machine Learning) que implica utilizar algoritmos de aprendizaje con el fin de obtener predicciones futuras.

Fig. 1. Procesamiento Hadoop, leyendo desde el disco[3].

MapReduce se ha convertido en una de las plataformas de computación paralela para procesar datos en escala de terabytes y petabytes. Utilizado diariamente en empresas como Yahoo, Google, Amazon y Facebook, adaptado por varias universidades, permite una paralelización fácil de cálculos intensivos de datos en muchas máquinas, una principal característica que tiene MapReduce es que intercala computación secuencial y paralela[4].

b) Spark In-Memory Processing: Es el primer paradigma de computación distribuido y de propósito general, debido a su velocidad y adaptabilidad. Spark logra velocidad a través de un modelo de datos llamado conjunto de RDD (datos distribuidos), se almacenan en la memoria mientras se calculan eliminando así costosas escrituras intermedias en el disco. Además, consta con un motor de ejecución de DAG (gráfico acíclico dirigido), puede optimizar la computación, particularmente la computación iterativa, es esencial para las tareas teóricas de datos como la optimización y el aprendizaje automático, está conformado por dos principales cálculos en memoria los cuales son: Almacenamiento en RAM y Procesamiento distribuido Paralelo (ver figura 2).

La figura 2 representa el proceso de trabajo de Spark, HDFS Read (archivos de lectura) se lee el archivo, pasa a Query interpretado como una cadena de consulta en las respectivos BD, pasa por ETL (extracción, transformación y carga) es un proceso de reformateo, limpieza y carga, para después analizarlos y guardarlos en memoria cache, terminado el proceso de ETL se enfoca en ML (Machine Learning), implica utilizar algoritmos de aprendizaje con el fin de obtener predicciones futuras, mediante Spark procesamiento en memoria, se obtiene menos código, lenguaje flexible y se encuentra principalmente en memoria.

Fig. 2. Spark Procesamiento en Memoria[3]

c) GPU-Accelerated Computer: Es el empleo de una unidad de procesamiento gráfico, conjuntamente con una unidad de procesamiento de computadora (ver figura 3), para facilitar las respectivas operaciones de procesamiento intensivo como: aprendizaje profundo, análisis y aplicaciones de ingeniería, la GPU hace que los programas o aplicaciones sean más rápidas. La aceleración a través de hardware especializado por procesadores no es nueva: varias áreas de aplicación como gráficos, multimedia, redes, criptografía, generalmente se acelera en PC y dispositivos integrados[5].

La figura 3 representa el proceso de trabajo de Procesamiento tradicional de GPU, HDFS Read (archivos de lectura) se lee el archivo pasan a GPU Read, la GPU realiza múltiples tareas en paralelo, pasan por Query que son interpretadas como una cadena de consulta en las respectivas BD, además, CPU Write se encarga de realizar los cálculos secuenciales, es aquí cuando trabajan conjuntamente la GPU y CPU; nuevamente se aplica GPU Read para realizar los cálculos complicados y pasa al proceso de ETL (extracción, transformación y carga) realiza el proceso de reformateo, CPU Write realiza el cálculo secuencial, lee la GPU Read y pasan al proceso de ML que implica utilizar algoritmos de aprendizaje a los datos. En la GPU existe más código, lenguaje rígido y sustancialmente deben realizarse las tareas en GPU.

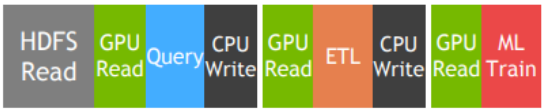

Fig. 3. Procesamiento tradicional GPU[3].

d) RAPIDS: Escribe en memoria del sistema RAPIDS (ver figura 4), donde se mantiene cosas en la GPU siempre como sea posible, se tiene formatos comunes que permiten obtener un ecosistema acelerado y permite reducir el proceso de ETL y ML cuando existen grandes volúmenes de datos.

La figura 4 representa el proceso de trabajo de Procesamiento tradicional de GPU, HDFS Read (archivos de lectura) se lee el archivo pasan a GPU Read, la GPU realiza múltiples tareas en paralelo, pasan por Query que son interpretadas como una cadena de consulta en las respectivas BD, además, CPU Write se encarga de realizar los cálculos secuenciales, es aquí cuando trabajan conjuntamente la GPU y CPU; nuevamente se aplica GPU Read para realizar los cálculos complicados y pasa al proceso de ETL (extracción, transformación y carga) realiza el proceso de reformateo, CPU Write realiza el cálculo secuencial, lee la GPU Read y pasan al proceso de ML que implica utilizar algoritmos de aprendizaje a los datos. En la GPU existe más código, lenguaje rígido y sustancialmente deben realizarse las tareas en GPU.

Fig. 4. RAPIDS[3].

III. GPU

A mediados de la década de 1990, la demanda de gráficos 3D por parte de los usuarios aumento vertiginosamente a partir de la aparición de juegos inmersivos en primera persona. Al mismo tiempo, empresas como NVIDIA, ATI Technologies y 3dfx Interactive empezaron a comercializar aceleradores gráficos que era suficientemente económicos para los mercados de gran consumo[6]. La GPU (unidad de procesamiento gráfico) moderna no es solo un potente motor de gráficos, es un procesador programable altamente paralelo con un ancho de banda de memoria y aritmética que supera a su contraparte de CPU (ver figura 3). Además, es un procesador de propósito específico, permite optimizar grandes cantidades de datos y realizan las mismas operaciones una y otra vez[7].

A. GPU vs CPU

La CPU está diseñada para el procesamiento en serie, es decir, se componen de pocos núcleos muy complejos que pueden ejecutar unos pocos programas al mismo tiempo, en cambio la GPU tiene cientos o miles de núcleos sencillos que pueden ejecutar bastantes programas a la vez. La GPU requiere un alto grado de paralelismo tradicionalmente una instrucción y múltiples datos.

Las diferencias son grandes entre el rendimiento de CPU y GPU (ver figura 5) se deben principalmente a una cuestión de filosofía de diseño. Mientras que la GPU están pensadas para explotar el paralelismo a nivel de datos y una lógica bastante simple. La CPU utiliza lógica de control que permite un paralelismo a nivel de instrucción y fuera de orden, utilizan memorias caches bastantes grandes para reducir el tiempo de acceso a los datos en memoria. Las GPU actuales tienen anchos de banda a memoria en torno a diez veces superiores a los de la CPU[6].

La figura 5 demuestra la comparativa de CPU y GPU, se requiere de menos lógica de control para cada flujo de ejecución, al mismo tiempo se dispone de una pequeña memoria cache que permite que los flujos que comparten memoria tengan el respectivo ancho de banda. Es por esto que mucha área del chip se dedica al procesamiento de datos.

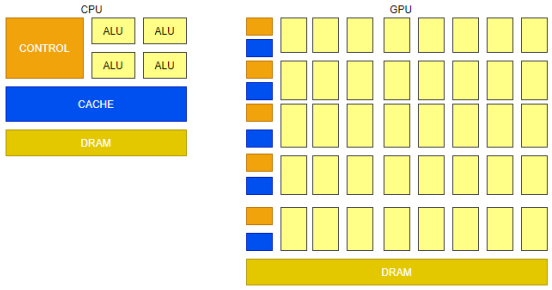

Fig. 5. Comparativa de memoria y lógica de control para CPU y GPU[6].

B. Arquitecturas GPU

Las arquitecturas GPU es una de las tendencias en computación paralela con más crecimiento en los últimos años y goza de gran popularidad, debido a la excelente relación entre las prestaciones que ofrece y su coste.

a) Arquitectura NVIDIA GeForce: La arquitectura NVIDIA GeForce brinda diferentes productos:

- GeForce: Enfocado en un mercado de consumo multimedia como videojuegos, edición de videos, fotografía digital[6].

- Cuadro: Enfocado para soluciones profesionales las cuales requiere modelos 3D, ejemplo en el ambiente de ingeniería y arquitectura[6].
- Tesla: Para la computación de altas prestaciones, como el procesamiento de información sísmica, simulaciones bioquímicas, modelos meteorológicos y cambio climático, computación financiera y análisis de datos[6].

b) Arquitectura Unificada: No existe la división a nivel de hardware entre procesadores de vértices y procesadores de fragmentos, estas son capaces de trabajar a nivel de vértice como a nivel de fragmento, sin estar especializada en un tipo concreto[6].

c) Arquitecturas Orientadas a Computación de Propósito General sobre GPU: La computación de propósito general se conoce como GPGPU, proporciona una mejora en términos de speedup de la ejecución de aplicaciones asociadas, estas aplicaciones pueden ser adaptadas a la GPU, se pueden destacar algunas como: algebra lineal, procesamiento de imágenes, procesamiento de consulta de BD, utilizan el procesador de fragmentos como unidad de cómputo[6].

d) Arquitectura NVIDIA: Permite analizar enormes cantidades de datos y realiza predicciones comerciales precisas a una velocidad sin precedentes, en el año 2006 NVIDIA presento un hardware orientado a computación general de prestaciones llamada Tesla[6].

e) Arquitectura CUDA: Se enfoca en computación paralela, ofrece el rendimiento del procesamiento gráfico de NVIDIA para aplicaciones de uso general de cálculo en la GPU, aprovechan una base instalada de cien millones de GPUs habilitadas para CUDA en ordenadores de sobremesa y portátiles[8].

IV. Estado del Arte

La librería de código abierto RAPIDS brinda la capacidad de ejecutar canalizaciones de Ciencia de Datos y análisis de extremo a extremo en GPU, es incubado por NVIDIA basado en una amplia experiencia en hardware, permite un procesamiento y capacitación enormemente acelerados en tamaños de conjuntos de datos mucho más grandes, brinda desafíos comerciales más complejos como predecir fraudes con tarjetas de créditos, pronosticar inventarios, comprender el comportamiento de los clientes, entre otras.

RAPIDS comenzó a partir de los proyectos de Apache Arrow, plataforma de desarrollo en varios idiomas para datos en memoria y basados en una estructura de datos columnar en memoria, ofrece un intercambio de datos eficiente y rápido con flexibilidad para soportar modelos de datos complejos. El objetivo principal de RAPIDS no es solo acelerar las partes individuales del flujo de trabajo típico de la ciencia de datos, sino acelerar el flujo de trabajo de extremo a extremo.

A. Ambiente de Trabajo RAPIDS

El ambiente de trabajo de la librería RAPIDS (ver figura 6), expone de manera determinada como es el proceso de trabajo aplicando los diferentes puntos como “Data Preparation”, “Model Training”, “Visualization” y el proceso interno que hay detrás.

La figura 6 representa el ambiente de trabajo de RAPIDS, cabe destacar que se menciona principales bibliotecas que permiten realizar la preparación, entrenamiento y visualización.

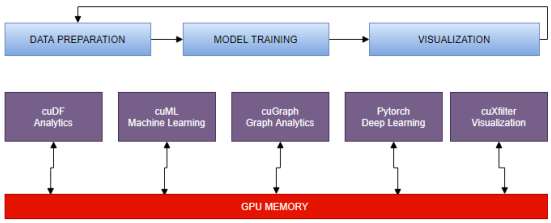

Fig. 6. Ciencia de Datos acelerado de GPU de extremo a extremo con RAPIDS[3].

a) Preparación de la Data: Principal etapa de análisis de datos, cuando existe información de baja calidad en varias fuentes de datos, las organizaciones están sumamente enfocadas en como transformar los datos en formularios limpios que puedan usarse con fines de alto beneficio, aplicando cuDF; la preparación de la data es identificar datos de calidad y formatearlos adecuadamente, lo que puede conducir a la generación de patrones de calidad, generan un conjunto de datos original pero con mejor calidad y datos relevantes que pueden mejorar significativamente[9].

b) Entrenando el Modelo: Obtener precisión predictiva con los distintitos datos, entrenando y evaluando la data, utilizando algoritmos programados (ver tabla I) que reciben y analizan para predecir valores de salida encontrando patrones en los datos de entrenamiento.

c) Visualización: Representación gráfica de datos, presenta los datos como una imagen; gráfico para facilitar la identificación de patrones y comprender conceptos difíciles, utilizando algoritmos (ver tabla II) para obtener valores de salida gráficamente, la tecnología permite que los usuarios interactúen con los datos cambiando los parámetros para ver mas detalles y crear nuevas ideas[10].

B. GPU Memory Apache Arrow

Apache Arrow es una plataforma en varios idiomas para datos en memoria, específica un formato de memoria columnar estandarizado independiente del lenguaje para datos planos y jerárquicos, organizado para operaciones analíticas eficientes en hardware moderno. Además, proporciona bibliotecas computacionales y mensajes de transmisión de copia cero y comunicación entre procesos[11].

Arrow evita la necesidad de serialización entre diferentes tiempos de ejecución del lenguaje y proporciona comunicación entre conjuntos de datos entre procesos sin copia[12].

a) Características Apache Arrow: Es rápido porque permite a los principales motores de ejecución aprovechar las últimas operaciones de SIMD (instrucción única datos múltiples) incluidas en los procesadores modernos, para la optimización vectorizadas por la localidad de los distintos datos

para un respectivo rendimiento en hardware moderno tanto para CPU y GPU. Es flexible porque Apache Arrow actúa como una interfaz de alto rendimiento entre varios sistemas (ver figura 7), es decir, que dentro de cada sistema Apache Arrow visualiza como una interfaz nueva para cada sistema, adquiere una amplia variedad de lenguajes de programación estándar como C, C++, C, Java, JavaScript, MatLab, Python, R, Ruby y Rust [11].

La figura 7 representa a Apache Arrow, demuestra que no existe gasto excesivo de comunicación entre los sistemas, se encarga de realizar un acceso común para los mismos, utilizan el mismo formato de memoria y se destaca el procesamiento paralelo para poder compartir funcionalidad.

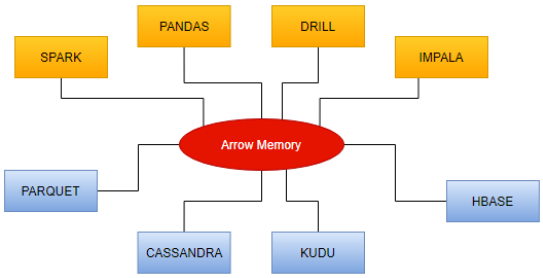

Fig. 7. Proceso Apache Arrow[3].

C. Biblioteca cuDF

cuDF es la biblioteca de RAPIDS, de manipulación de DataFrame basado en Apache Arrow, está enfocado en la preparación de datos, acelera la carga, el filtrado y la manipulación de datos, contiene la definición del modelo de memoria GDF, específicamente la columna gdf_, las funciones de procesamiento de datos pueden operar en esas respectivas columnas[13].

a) Características cuDF: Esta conformado por dos partes principales que son libGDF y PyGDF (ver figura 8), estos hacen que la biblioteca trabaje de manera eficiente en el tratamiento de la data.

La figura 8 representa la estructura de la biblioteca cuDF, la biblioteca se enfoca en realizar un trabajo de manera eficaz, reduciendo el tiempo de proceso de ETL (extracción, transformación y carga), es un repositorio único que contiene tanto la implementación de bajo nivel como envolturas y API de alto nivel.

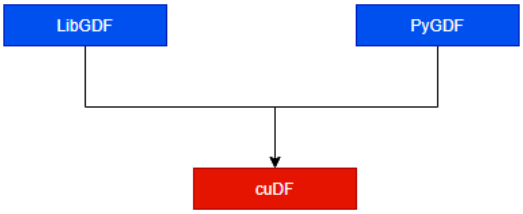

Fig. 8. Estructura de la biblioteca cuDF

b) libGDF: Biblioteca de bajo nivel que contiene implementaciones de funciones y API C y C++. Importa y exporta un GDF utilizando el mecanismo de CUDA IPC, contiene núcleos CUDA para realizar operaciones matemáticas basadas en elementos en columnas DataFrame GPU y operaciones de clasificación, unión, agrupación y reducción CUDA en marcos de datos GPU.

c) PyGDF: Biblioteca de Python para manipular marcos de datos en GPU, interfaz de Python para la biblioteca libGDF con funcionalidad adicional, crea GDF a partir de matrices Numpy, Pandas Dtaframe y PyArrow Tables.

D. Biblioteca cuML

cuML es un conjunto de bibliotecas que implementan algoritmos de aprendizaje automático acelerados por GPU y funciones primitivas matemáticas que comparten API compatibles con otros proyectos de RAPIDS. cuML permite a los científicos de datos, desarrolladores ejecutar tareas tradicionales de ML tabular en GPU, presenta operaciones con múltiples GPU y múltiples nodos, contiene una lista creciente de algoritmos.

La tabla I representa los principales conjuntos de algoritmos que tiene cuML [10], están agrupados por categoría y dentro de categoría los algoritmos que corresponden a cada uno de ellos. Todos estos algoritmos son soportados por la librería cuML, facilita el proceso de aprendizaje automático y la misma trabaja conjuntamente con RAPIDS.

TABLE I
ALGORITMOS CUML

Clasificación	Algoritmos	Notas
Clustering	*Density-Bases Spatial Clustering of Applicaction with Noise*	
	K-means	*Multi-nodo multi GPU a tráves de Dask*
	Principal Components Analysis (PCA).	*Multi-nodo multi-GPU a través de Dask*
Dimesionality	*Truncated Singular Value Decompositttion (tSDV)*	*Multi-nodo multi-GPU a través de Dask*
	Uniform Manifold Approximation and Projection (UMAP).	
	Random Projection	
Linear Models for Regression or Classification	*Linear Regression (OLS)*	*Multi-GPU disponible en el paquete con CUDA 10.*
	Linear Regression with Lasso or Ridge Regularization	
	ElasticNet Regression	
	Logistic Regression	
	Stochastic Gradient Descent (SGD), Coordinate Descent (CD), and Quasi-Newton (QN) (includin L-BFGS and OWL-QN) solvers for linear models	
Nonlinear Models for Regression or Classification	*Random Forest (RF) Classifaction*	*GPU multi-nodo-experimental a través de Dask*
	Random Forest (RF) Regression	*GPU multi-nodo-experimental a través de Dask*
	K-Nearest Neighbors (KNN) Classification	*Multi-nodo multi-GPU a través de Dask, disponible en la version 0.12 y paquetes conda nocturnos. Utiliza Faiss para la consulta de vecinos más cercanos*
	K-Nearest Neighbors (KNN) Regression	*Multi-nodo multi-GPU a través de Dask, disponible en la version 0.12 y paquetes conda nocturnos. Utiliza Faiss para la consulta de vecinos más cercanos*
	Support Vector Machine Classifier	
Time Series	*Support Vector Machine Classifier (SVC).*	
	Linear Kalman Filter	
	Holt-Winters Exponential Smoothing	
	Auto-regressive Integrated Moving Average (ARIMA).	

[a]Algoritmos que soporta la biblioteca cuML.

E. Biblioteca cuGraph

cuGraph es una biblioteca de algoritmos de gráficos que se integra a la perfección con el ecosistema de ciencia de datos de RAPIDS, permite al analista de datos y científico de datos llamar fácilmente algoritmos de gráficos utilizando los datos almacenados en un marco de datos GPU.

cuGraph es hacer que el análisis del gráfico sea omnipresente hasta el punto de que los usuarios solo piensen en términos de análisis y no en tecnologías o marcos. La velocidad de memoria interna de una GPU permite que cuGraph cambie rápidamente la estructura de datos para adaptarse mejor a las necesidades de la analítica en lugar de limitarse a una sola estructura de datos[10].

También es una colección de datos para analizar gráficos que procesa los datos que se encuentran en los marcos de datos de GPU. El principal objetivo que tiene cuGraph es proporcionar una API similar a Network: es una biblioteca que se enfoca en estudiar gráficos y redes, con el fin que los científicos de datos puedan construir flujos de trabajo acelerados por GPU más fácilmente. cuGraph utiliza otros proyectos basados en Dask de RAPIDS como dask-cuda ayuda al despliegue y la gestión de los trabajadores de Dask en sistemas con múltiples GPU [10].

Los algoritmos de cuGraph están diseñados para ejecutarse en una sola GPU con conjuntos de datos de alrededor de 500 millones de bordes o menos.

TABLE II
ALGORITMOS CUGRAPH

Algoritmo	Escala	Notas
PageRank	*Multi-GPU*	
Personal PageRank	*Single-GPU*	
Katz Centrality	*Single-GPU*	
Jaccard Similarity	*Single-GPU*	
Weighted Jaccard	*Single-GPU*	
Overlap Similarity	*Single-GPU*	
SSSP	*Single-GPU*	*Actualizado para proporcionar información de ruta*
BFS	*Single-GPU*	*Tambien version BSP*
Traingle Counting	*Single-GPU*	
K-Cores	*Single-GPU*	
Core Number	*Single-GPU*	
Subgraph Extraction	*Single-GPU*	
Spectral Clusterin-Balanced-Cut	*Single-GPU*	
Spectral Clusterin-Modularity Maximization	*Single-GPU*	
Louvain	*Single-GPU*	
Renumberin	*Single-GPU*	
Basic Graph Statistics	*Single-GPU*	
Weakly Connected Components	*Single-GPU*	
Strongly Connected Components	*Single-GPU*	

[a]Algoritmos que soporta la biblioteca cuGraph [10].

F. Biblioteca cuXfilter

Como la visualización es un componente clave para un científico de datos, existe cuXfilter (ver figura 9), es un framework de RAPIDS que conecta visualizaciones web a filtros cruzados por GPU, además, realiza un filtrado multidimensional interactivo y super rápido de más de 100 millones de conjuntos de datos tabulares de filas a través de cuDF.

La figura 9 representa la arquitectura general de cuXfilter, necesita de un backend completo y una API del lado del cliente, utiliza un servidor Sanic para acceder a las funciones cuDF de Python, un servidor Express con node.js para manejar las llamadas de encadenamiento y una conexión socket.io a la API cuXfilter para conectarse a cualquier biblioteca de visualización basada en JavaScript.

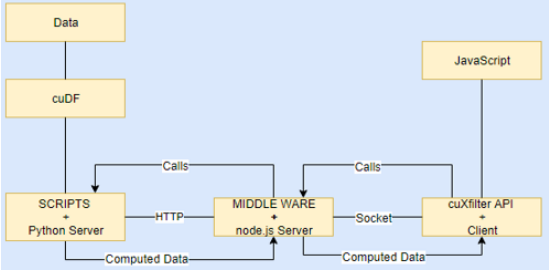

Fig. 9. Arquitectura cuXfilter[14]

cuXfilter.py (ver figura 10) actúa como una biblioteca de conectores, proporciona las conexiones entre diferentes bibliotecas de visualización y un marco de datos de GPU, permite al usuario usar gráficos de diferentes bibliotecas en un solo tablero.

La figura 10 representa la arquitectura de cuXfilter.py, aprovecha el portátil de jupyter y el servidor de bokeh para reducir en gran medida la complejidad del backend.

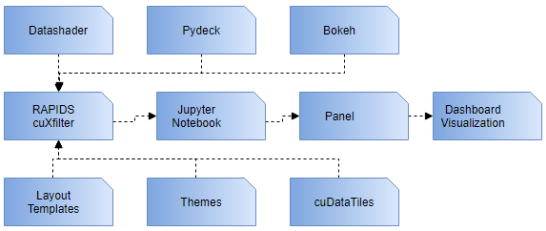

Fig. 10. Arquitectura cuXfilter.py[10]

G. Dask

Proporciona formas de escalar los flujos de trabajo de Pandas, Scikit-Learn, Numpy de formas más nativa, con una reescritura mínima, descubre como dividir grandes cálculos y enrutar partes de ellos de manera más eficiente en hardware distribuido. Dask se ejecuta rutinariamente en clústeres de miles de máquinas para procesar cientos de terabytes de datos de manera eficiente dentro de entornos seguros[15]. Dask programador de computo distribuido, es extremadamente modular con programación, computo, transferencia de datos y manejo fuera del núcleo, todo desunido, lo que permite conectar nuestras propias implementaciones[16].

a) Dask-cuDf: Puede escalar hacia arriba y hacia afuera con cuDF, utiliza primitivas cuDF debajo en operaciones de estilo de reducción de mapas con la misma API de alto nivel, aprovecha el hardware utilizando un marco de comunicaciones llamada OpenUCX[16].

b) Dask-cuMl: Una integración nativa con Dask-cuDF, puede usar fácilmente a los trabajadores de Dask para inicializar avances NCCL para operaciones de recolección, dispersión optimizadas, proporciona primitivas fáciles de usar y de alto nivel para la sincronización de trabajadores, lo cual es necesario para muchos algoritmos de ML.

c) Dask-cuGraph: Contiene algoritmos de análisis de gráficos paralelos que pueden hacer uso de múltiples GPU en un solo host, es capaz de jugar muy bien con otros proyectos en el ecosistema Dask, así como otros proyectos de RAPIDS, como Dask-cuDF y Dask-cuML.

H. Integración con Bibliotecas de Deep Learning

RAPIDS se integra a la perfección con bibliotecas de aprendizaje profundo, para la computación acelerada.

a) Chainer: Framework de Python, permite entrenar y evaluar rápidamente modelos de aprendizaje profundo. Proporciona formas imperativas de declarar redes neuronales para admitir computación numérica basada en GPU [17].

b) MXNET: Librería de aprendizaje profundo, diseñado para la eficiencia y flexibilidad, permite mezclar programación simbólica e imperativa para poder maximizar la eficiencia y la productividad, contiene un programador de dependencia dinámico que paraleliza automáticamente las operaciones simbólicas e imperativas sobre la marcha [18].

c) PyTorch: Esta diseñado para ser intuitivo, lineal en pensamiento y fácil de usar por los programadores, cuando se ejecuta una línea de código, esta se ejecuta, no existe visión asincrónica del mundo [19]. Pytorch es un remplazo de NumPy para poder usar el poder de la GPU y tambien como una plataforma de investigación de aprendizaje profundo que proporciona la máxima flexibilidad y velocidad.

d) Numba: Permite a los respectivos usuarios anotar funciones costosas, luego son compiladas por JIT por LLVM al momento de ser llamadas. Numba hace uso de la sintaxis del decorador de Python, permite que su uso se aplique de forma incremental, sin requerir una reescritura importante [20].

V. Propuesta de Arquitectura RAPIDS

La figura 11 representa el análisis extenso e implementación de la Propuesta de la Arquitectura RAPIDS, se explica de manera dinámica, concisa y sólida, a través de los siguientes puntos:

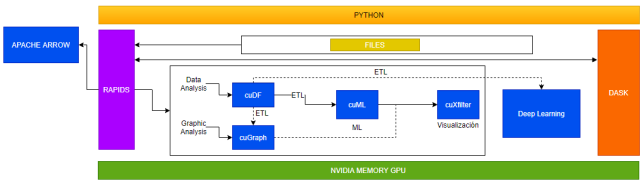

Fig. 11. Implementación Arquitectura RAPIDS

a) RAPIDS: Trabaja de forma correcta, eficiente y eficaz, necesita estar en un ambiente dedicado en base a Ciencia de Datos, para construir un ambiente de tal magnitud se debe realizar un proceso respectivo:

- RAPIDS permite leer y análizar archivos tipo CSV, JSON, LOGS, HTML, entre otros.
- El conjunto de bibliotecas principales de RAPIDS permitirá llevar a cabo un flujo de trabajo de ETL, este proceso lo realizará la biblioteca cuDF para el procesamiento y análisis de los datos. RAPIDS realiza el análisis de gráficos con la biblioteca cuGraph, trabaja conjuntamente con cuDF y cuML.
- ML proceso de entrenamiento y prueba lo realiza la biblioteca cuML a los datos previamente seleccionados.
- Visualización, cuXfilter para la respectiva representación de resultados en forma de gráficos estadísticos y filtrado de información.

b) RAPIDS: Trabaja en ambientes GPU, permite que el procesamiento de los datos sea ágil, reduce tiempos de ETL, ML y para la representación de resultados trabaja de forma dinámica.

c) RAPIDS: Se puede integrar con bibliotecas de Deep Learning, para mejorar la aceleración de extremo a extremo, cabe recalcar que los científicos de datos y analistas se enfocan de manera profunda en este tipo de desarrollo de algoritmos para determinar lo que esta sucediendo, para trabajar con este tipo de librerías debe pasar previamente por el proceso de la biblioteca cuDF para la extracción, transformación y carga de los datos.

d) RAPIDS: Para mejorar su rendimiento y tiempos de esfuerzo, tiene una fusión con Dask, particularmente si estas analizando y procesando bastante información, permiten acelerar las cargas de trabajo de Python desde la computadora portátil y ejecutar fácilmente varios trabajadores.

e) RAPIDS: Está basado en Apache Arrow para minimizar interacciones de datos y las serializaciones de datos cuando una tubería de procesamiento de datos incluye diferentes marcos de cómputo.

f) RAPIDS: Debe trabajar en el lenguaje de programación de Python, algunos puntos precisos del porque trabajar con el lenguaje Python:

- En el 2018, Python fue el lenguaje de programación más popular para la ciencia de datos, año tras año es cada vez más atraído por los científicos de datos [21].
- Python reúne características necesarias para la ciencia de datos, permite simplificar muchas cosas, mejora el código en cuanto a que sea legible a través de la sintaxis.
- Los científicos de datos necesitan lidiar, analizar, procesar problemas complejos, python proporciona todas las herramientas necesarias para llevar el respectivo caso de ETL, ML y visualización trabajando con la librería dedicada que es RAPIDS.
- Python permite equipar a los científicos de datos para implementar soluciones factibles, al mismo tiempo sigue los estándares de los algoritmos requeridos del proceso del anaálisis, desarrollo y solución.

A. Futura Implementación

Dentro del enfoque principal para el desarrollo e implementación se hara uso de la arquitectura propuesta, aplicando las funcionalidades, características, herramientas y plataformas a utilizar, trabajará directamente con las bibliotecas cuDF, cuGraph para el proceso de ETL, cuML para el desarrollo de ML y cuXfilter para la representación de los futuros resultados en cuanto a gráficos estadísticos (barras, pastel) y filtrado de información (ver figura 11), además, poder determinar la agilidad de trabajo y reducción de tiempos de proceso por medio de la GPU y comparar tiempos de respuesta (ETL, ML) con otras librerías como por ejemplo PANDAS.

VI. Conclusiones

Con el ánalisis expuesto de la presente investigación de ciencia de datos y RAPIDS, se ha destacado la importancia que tiene la GPU, revela detalles del trabajo que adopta en ambientes de proceso exploratorio de datos por medio de algoritmos que contiene la librería, además, apoya en mejorar el futuro proceso en ambientes de ciencias de datos. El objetivo es dar a conocer el mejor funcionamiento de RAPIDS, para poder consechar toda su potencia de trabajo e implementación.

RAPIDS permite trabajar de manera rápida, eficaz, eficiente y de forma acelerada de extremo a extremo utilizando los beneficios de la GPU (aplicando las arquitecturas de NVIDIA), el conjunto de bibliotecas de RAPIDS perfecciona el flujo de analizar, procesar, trabajar y solucionar problemas e inconvenientes futuros; brinda a los desarrolladores y científicos de datos trabajar de manera determinante y progresar en el estudio, para aumentar el ánalisis exploratorio y trabajos con Inteligencia Artificial (Machine Learning y Deep Learning).

References

[1] Joyones Aguilar. L. Big data, análisis de grandes volúmenes de datos en organizaciones. *DATA 2019 - Proceedings of the 8th International Conference on Data Science, Technology and Applications*, 2016. URL: https://books.google.com.ec/books?hl=es&lr=&id=1GywDAAAQBAJ&oi=fnd&pg=PT6&dq=Aguilar,+L.+J.+(2016).+Big+Data,+Anlisis+de+grandes+volmenes+de+datos+en+organizaciones.+Alfaomega+Grupo+Editor.&ots=_WU9L37j_Q&sig=YPC8eRaPaRv56HxrbY01I8IoRmk&redir_esc=y#v=on.

[2] Turner. James. Hadoop: What it is, how it works, and what it can do. page 2, 2011.

[3] Srinath. Ashwin and Kraus. Keith. Hadoop processing, reading from disk. spark in-memory processing. traditional gpu processing. rapids. page 46, 2019.

[4] Howard. Karloff, Siddharth. Suri, and Sergei. Vassilvitskii. Proceedings of the twenty-first annual acm-siam symposium on discrete algorithms. *A Model of Computation for MapReduce*, 2010.

[5] Matsuoka. S, Aoki. T, Endo. T, Nukada. A, Kato. T, and Hasegawa. A. Gpu accelerated computing–from hype to mainstream, the rebirth of vector computing. *Commoditization of HPC and its acceleration, but niche still remains* ., page 11, 2009.

[6] Guim. Frances and Rodero. Ivan. Arquitecturas basadas en computación gráfica (gpu). page 68, 2019.

[7] Owens. John. D, Houston. Mike, Luebke. David, Green. Simon, Stone. John. E., and Phillips. James. C. Gpu computing. 2008.

[8] NVIDIA. Nvidia cuda architecture. *NVIDIA CUDA Architecture Introduction Overview*, page 9, 2009.

[9] Mansingh. Gunjan, Osei-Bryson., Kweku Muata, Rao. Lila, and McNaughton. Maurice. Data preparation: Art or science? 2017. URL: https://doi.org/10.1109/ICDSE.2016.7823936.

[10] RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018. URL: https://rapids.ai.

[11] Arrow. Apache. The apache software foundation. *2018*. URL: https://arrow.apache.org/.

[12] Peltenburg. Johan, van Straten. Jeroen, Wijtemans. Lars, van Leeuwen. Lars, Al-Ars. Zaid, and Hofstee. Peter. Fletcher: A framework to efficiently integrate fpga accelerators with apache arrow. pages 270–277, 2019. URL: https://github.com/abs-tudelft/fletcher.

[13] Aramburo. Rodrigo. Blazingsql parte 1: El gpu dataframe (gdf) y cudf en rapids ai. *cuDF — GPU Data Processing for GDFs*, 2019.

[14] Enemark. Allan. Accelerating cross filtering with cudf. 2018. URL: https://medium.com/rapids-ai/accelerating-cross-filtering-with-cudf-3b4c29c89292.

[15] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. URL: https://dask.org.

[16] Patterson. Joshua. The platform inside and out.

[17] Shohei. Hido. Complex neural networks made easy by chainer. *A define-by-run approach allows for flexibility and simplicity when building deep learning networks.*, 2016.

[18] MXNet. Lightweight, portable, flexible distributed/mobile deep learning with dynamic, mutation-aware dataflow dep scheduler; for python, r, julia, scala, go, javascript and more. 2019. URL: https://github.com/apache/incubator-mxnet.

[19] Pytorch. Tensors and dynamic neural networks in python with strong gpu acceleration. 2019. URL: https://github.com/pytorch/pytorch.

[20] Crist. James. Dask numba: Simple libraries for optimizing scientific python code. *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, pages 2342–2343, 2016.

[21] Aguerzame. Abdallah, Pelletier. Benoit, and Waeselynck. François. Gpu acceleration of pyspark using rapids ai. *DATA 2019 - Proceedings of the 8th International Conference on Data Science, Technology and Applications*, pages 437–442, 2019.