

# **CARATULA**

**UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA**

*La Universidad Católica de Loja*

**ÁREA TÉCNICA**

TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

**Implementación de un Ambiente de Ciencia de Datos a través de la librería Open Source RAPIDS.**

TRABAJO DE TITULACIÓN.

**AUTOR**: Guarnizo Romero, José Alberto

**DIRECTOR**: Mgtr. Elizalde Solano, René Rolando

LOJA-ECUADOR

2019 – 2020

# **APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN**

Magister.

René Rolando Elizalde Solano

**DOCENTE DE LA TITULACIÓN**

De mi consideración:

El presente trabajo de fin de titulación: Implementación de un Ambiente de Ciencia de Datos a través de la librería Open Source RAPIDS, realizado por Guarnizo Romero José Alberto, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación de este.

Loja, mes de 2019

f) ……………………….…..

# **DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS**

“Yo Guarnizo Romero José Alberto declaro ser autor del presente trabajo de titulación: Implementación de un ambiente de Ciencia de Datos a través de la Librería Open Source RAPIDS, de la Titulación de Sistemas Informáticos y Computación, siendo René Rolando Elizalde Solano director del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además, certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad”

f)…………………….

Guarnizo Romero José Alberto

C.I: 1105774200

# **DEDICATORIA**

# **AGRADECIMIENTO**

**ÍNDICE**

CARATULA…………………………………………………………………………i

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN…………...ii

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS……………………iii

DEDICATORIA……………………………………………………………………..iv

AGRADECIMIENTO……………………………………………………………….v

INDICE DE CONTENIDOS………………………………………………………...vi

INDICE DE FIGURAS………………………………………………………………vii

INDICE DE TABLAS……………………………………………………………….viii

1. **CAPÍTULO 1 INTRODUCCIÓN** 
   1. Introducción
   2. Problemática
   3. Justificación
   4. Objetivos
      1. General
      2. Específicos
   5. Estrategia o Metodología de Desarrollo
   6. Estructura del Documento
2. **CAPÍTULO 2 MARCO TEÓRICO**
   1. Ciencia de Datos
      1. Big Data
         1. Tipos de Datos
            1. Datos Estructurados
            2. Datos Semiestructurados
            3. Datos no Estructurados
         2. Características
         3. Ventajas y Desventajas
      2. Procesamiento de Datos
         1. Evolución del Procesamiento de Datos para llegar a RAPIDS
            1. Distributed Storage
            2. Spark In-Memory Processing
            3. GPU-Accelerated Computer
            4. RAPIDS
   2. GPU
      1. GPU vs CPU
      2. Arquitecturas GPU
         1. Arquitectura NVDIA GeForce
         2. Arquitectura Unificada
         3. Arquitecturas orientadas a computación de propósito general sobre GPU
         4. Arquitectura NVDIA
         5. Arquitectura CUDA
      3. Funcionamiento GPU
         1. CUDA
            1. Ventajas y Desventajas CUDA
   3. Open Source RAPIDS
      1. Ambiente de Trabajo RAPIDS
         1. Preparación de la Data
         2. Entrenando el Modelo
            1. Machine Learning

Algoritmos Machine Learning

Tipos de Algoritmos de Machine Learning

* + - 1. Visualización con RAPIDS
    1. GPU Memory Apache Arrow
       1. Funcionamiento de Apache Arrow
       2. Arquitectura de Apache Arrow
       3. Ventajas de Apache Arrow
    2. Biblioteca cuDF
       1. Características de cuDF
          1. libGDF
          2. PyGDF
       2. Funcionamiento cuDF
    3. Biblioteca cuML
       1. Algoritmos soportados por cuML
    4. Biblioteca cuGraph
       1. Algoritmos soportados por cuGraph
    5. Biblioteca cuXfilter
       1. Arquitectura cuXfilter
    6. DASK
       1. DASK a Computadoras Portátiles
       2. DASK-RAPIDS
    7. Integración con Bibliotecas de Deep Learning
       1. Chainer
          1. Características Chainer
       2. MXNet
          1. Características MXNet
       3. PyTorch
          1. Características PyTorch
       4. Numba
          1. Python-Numba
          2. Características Numba
    8. Análisis de la Arquitectura y Características RAPIDS

1. **CAPÍTULO 3 TRABAJOS RELACIONADOS**
   1. Fuentes de Información
   2. Cadenas de Búsqueda
   3. Criterios de inclusión y exclusión
   4. Análisis de Trabajos Relacionados
2. **CAPITULO 4 IMPLEMENTACIÓN RAPIDS**
   1. Instalación RAPIDS
      1. Instalación en Máquina Personal
         1. Instalación con Docker
         2. Instalación con Conda
      2. Google Colabority
         1. Entorno de Ejecución
         2. Instalación
      3. BlanzingSQL
   2. Proceso de *ETL*
      1. Extracción concuDF
      2. Transformación con cuDF
      3. Carga con cuDF
   3. Proceso de Implementación de Machine Learning
      1. Dividir Data en Train y Test
      2. Preprocesamiento
      3. Algoritmos Machine Learning
   4. Presentación de Resultados

ÍNDICE DE FIGURAS

Figura 1. Ciencia de Datos.

Figura 2. Representación de Datos Estructurados.

Figura 3. Representación de Datos Semiestructurados.

Figura 4. Representación de Datos no Estructurados.

Figura 5. Procesamiento Hadoop, leyendo desde el disco.

Figura 6. Spark procesamiento en memoria.

Figura 7. Procesamiento tradicional GPU.

Figura 8. RAPIDS.

Figura 9. Modelo de computación heterogéneo CPU + GPU.

Figura 10. Comparativa de memoria lógica de control para CPY y GPU.

Figura 11. Interconexión entre CPU y GPU mediante PCLe.

Figura 12. Arquitectura básica de GPU.

Figura 13. Esquema de la Arquitectura de la GPU GeForce de NVIDIA.

Figura 14. Comparativa de asignación de procesadores de una GPU en el procesamiento de vértices y de fragmentos en arquitecturas unificadas y no unificadas.

Figura 15. Esquema de arquitecturas G80 de NVIDIA.

Figura 16. Arquitectura CUDA.

Figura 17. Ciencia de Datos acelerados de GPU de extremo a extremo con RAPIDS.

Figura 18. Fases de Preparación de Datos.

Figura 19. Sin Apache Arrow.

Figura 20. Con Apache Arrow.

Figura 21. Arquitectura Apache Arrow.

Figura 22. Estructura de biblioteca cuDF.

Figura 23. Arquitectura cuXfilter.py.

Figura 24. Arquitectura general cuXfilter.

Figura 25. Proceso de Chainer Define-by-Run.

Figura 26. Proceso de Trabajo Python-Numpy.

Figura 27. Arquitectura RAPIDS.

Figura 28. Evaluación comparativa de XGBoost con CPU, GPU y XGBoost cuDF + GPU.

Figura 29. PySpark + RAPIDS y PySpark nativo.

Figura 30. Análisis comparativo de alertas con una sola GPU.

Figura 31. Evaluación comparativa de múltiples nodos, múltiples GPUs cuML y Dask-CUML.

Figura 32. Nvidia Activa.

Figura 33. Modelo tarjeta gráfica.

Figura 34. Drivers disponibles.

Figura 35. Instalar controlador.

Figura 36. GPU Nvidia activa.

Figura 37. Reposirotio sobre HTTPS.

Figura 38. Agregar clave.

Figura 39. Clave huella digital.

Figura 40. Repositorio estable.

Figura 41. Instalar Docker.

Figura 42. Prueba Docker.

Figura 43. Contendores Docker con GPU.

Figura 44. Prueba GPU en Docker.

Figura 45. Instalación RAPIDS.

Figura 46. Ejecutar Docker y RAPIDS.

Figura 47. Plataforma Jupyter con Docker y RAPIDS.

Figura 48. Crear Entorno.

Figura 49. Activar Entorno.

Figura 50. Instalación Jupyter.

Figura 51. Prueba GPU en Conda.

Figura 52. Instalación RAPIDS al entorno de trabajo datascience creado por Conda.

Figura 53. Entrar a Jupyter.

Figura 54. Plataforma Jupyter con Conda y RAPIDS.

Figura 55. Entorno de ejecución GPU.

Figura 56. Tipos de GPU Tesla.

Figura 57. Instalación RAPIDS.

ÍNDICE DE TABLAS

Tabla 1. Núcleos de procesamiento de un dispositivo CUDA.

Tabla 2. Algoritmos cuML.

Tabla 3. Algoritmos soportados por cuGraph.

Tabla 4. Fast Chainer.

Tabla 5. Flexible Chainer.

Tabla 6. Intuitivo Chainer.

Tabla 7. Componentes Pytorch.

Tabla 8. Prerrequisitos de Instalación RAPIDS.

Tabla 9. Especificaciones computador portátil.

**CAPÍTULO 1 INTRODUCCIÓN**

* 1. 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 encuentra almacenadas en *BD* (Base de Datos), *cloud* (información en la nube) 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 general valor y rapidez.

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 datos de información en tiempo real, además, poder determinar y hacer uso de las características más importantes de la librería que permiten: ocupar el máximo de recursos disponibles, tiempos mínimos de latencia, tolerancia a fallas y capacidad para el análisis, procesamiento e implementación de algoritmos de *ML* (*Machine Learning*) a los datos.

El trabajo de titulación se realizó con el interés de conocer, analizar, ampliar y utilizar la librería *RAPIDS*, para determinar un resultado de este análisis e implementación, tanto como investigadores, docentes y estudiantes puedan ser uso de esta librería que permitirá ayudar al crecimiento de una empresa, negocio o una institución.

Del desarrollo e investigación teórica del tema en curso, se consultó en fuentes académicas confiables sobre la librería *RAPIDS*, que den solución al tema planteado, donde se pondrá a prueba el funcionamiento, características y ventajas que brinda esta librería por medio de procesamiento y análisis de datos, dando como resultado que cumple con las especificaciones de la misma.

* 1. Problemática

La generación de datos crece minuto a minuto, grupos de investigación, instituciones públicas capturan datos de sus procesos internos a gran escala. Dicha información necesita ser proceda e interpretada para un posterior uso específico de acuerdo a las necesidades de la organización que lo necesite.

Para esto empresas como *NVIDIA*, a través de su equipo de investigación han creado la librería *RAPIDS*, que permiten realizar un tratamiento eficaz, rápido y preciso de la información y obtener resultados que sirven a las empresas en la toma acertadas de decisiones. Permitiendo además que utilicen sus recursos de forma optimizada para cubrir las verdaderas necesidades de sus usuarios y su adaptación a diversos escenarios.

* 1. Justificación

En el desarrollo de la presente investigación se pretende realizar un análisis de la librería *RAPIDS* que permitan analizar un conjunto de datos abiertos, procesamiento de datos en tiempo real que servirá como base conceptual para la implementación y ejecución de *RAPIDS*. Cuyo principal objetivo es la implementación de un ambiente de ciencia de datos a través de la Librería *Open* *Source* *RAPIDS*, debido a que actualmente las empresas e instituciones están realizando el análisis de información para generar valor tanto monetario como estratégico que ayudan a su crecimiento lo que resulta de gran interés y curiosidad.

En al ámbito educativo el trabajo de titulación de investigación servirá como referencia y base para estudiantes que realicen posteriores trabajos sobre el análisis de datos en tiempo real utilizando la librería. Así que los resultados obtenidos de las pruebas de funcionalidad y rendimiento de esta librería puedan servir para determinar la eficacia en el procesamiento y análisis de grandes cantidades de datos, tolerancia a fallos y mostrar la integridad y flexibilidad para relacionarse con diferentes herramientas, dando a conocer con estos resultados lo importante del análisis de la información en la actualidad.

* 1. Objetivos
     1. General

Implementar un ambiente de ciencia de datos a través de la Librería *Open* *Source* *RAPIDS*.

* + 1. Específicos
* Elaborar una investigación documentada sobre la *Librería Open Source RAPIDS.*
* Realizar analítica de grandes volúmenes de datos a través del uso de las características de la Librería *Open Source RAPIDS.*
* Implementar algoritmos de Machine Learning haciendo uso de Librería *Open* *Source* *RAPIDS*.
  1. Estrategia o Metodología de Desarrollo

Para el desarrollo del presente trabajo de titulación se ha considerado los siguientes puntos a tratar:

* Recolección de información de fuentes académicas confiables.
* Revisión de casos de éxito relacionados a la librería O*pen Source Rapids*.
* Experimentación y puesta en funcionamiento de la librería *Rapids*.
* Plan de pruebas para la solución planteada.
  1. Estructura del Documento

El siguiente trabajo de titulación se ha organizado en los siguientes capítulos:

* En el Capítulo 1 se hace la presentación formal de la introducción, problemática que da raíz al tema, la relación entre el objetivo general, los objetivos específicos y la metodología utilizada en el Trabajo de Titulación.
* En el Capítulo 2 constituye el encuadre del problema que se va a tratar dentro de limitantes teóricas y constituyen el punto de partida para orientar el desarrollo de la investigación. Se incluyen estudios relacionados, problemas relacionados y situación actual de la Librería *Open Source RAPIDS*.
* En el Capítulo 3 se enfoca en la representación de todos los trabajos relacionados, con respecto a la Librería *RAPIDS*.
* En el Capítulo 4 se enfoca al desarrollo o implementación de la librería *RAPIDS* a los respectivos datos a tratar y presentación de resultados.

**CAPÍTULO 2 MARCO TEÓRICO**

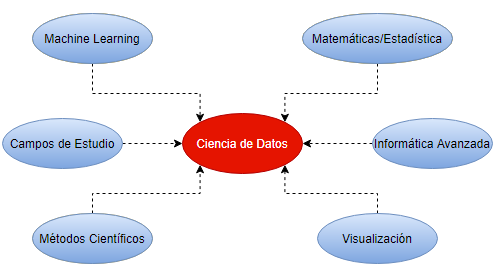
1. Marco Teórico

En el presente trabajo de titulación tiene la finalidad de Implementar un ambiente de ciencia de datos utilizando la *Librería Open Source RAPIDS* para el análisis, procesamiento de datos e implementación de algoritmos *Machine Learning*, el marco teórico permite conocer los fundamentos conceptuales necesarios para comprender el tema, además de clarificar lo que se pretende desarrollar.

Ambiente de ciencia de datos es el principal tópico de este trabajo por lo que se lo detalla en la sección 2.1, teniendo en cuenta que la información es la base principal por la que se desarrolla el trabajo; sección 2.2, se habla de cómo funciona la *GPU* y las arquitecturas de la misma, una comparativa entre *GPU* y *CPU* y funcionamiento; sección 2.3, es la parte final de este capítulo tenemos la descripción, análisis, ambiente de trabajo y características de la Librería *Open* *Source* *RAPIDS*, este punto es primordial para dar a conocer el flujo de la librería, dar a conocer la compatibilidad con otras librerías que se detallarán en esta sección.

* 1. Ciencia de Datos

La ciencia de datos es parte muy fundamental de todas las ciencias de aprendizaje automático y profundo, la inteligencia artificial y de negocios, involucra métodos científicos, procesos y sistemas para extraer conocimiento para un mejor entendimiento de los datos, con el fin de que esos datos sean útiles a nivel económico, social, educativo y empresarial.



*Figura 1.* Ciencia de Datos.

Fuente*:* Autor

Elaboración*:* Autor

En la figura 1 se demuestra de manera global lo que involucra y comprende la ciencia de datos, como: conocimientos en el campo de estudio (geología, finanzas, medicina, etc.), se toman en cuenta aspectos computacionales, además, involucra matemáticas y estadística, punto fuerte es el aprendizaje automático o *machine Learning,* visualización de prototipos de software.

Además, la ciencia de datos entiende y permite analizar los datos desde un punto de vista comercial, el principal objetivo es proporcionar la predicción más precisa, que permite impedir que una empresa o un empresario pierda en el futuro.

Cabe destacar que la misma ha 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.

* + 1. *Big Data*

*Big* *Data* (grandes datos o volúmenes de datos) supone la confluencia de una multitud de tendencias tecnológicas que se han ido consolidando durante los años 2011 y 2013, además han explosionado e irrumpido con gran fuerza en distintas organizaciones, instituciones y empresas.

Los grandes datos o volúmenes de datos han ido creciendo de modo magnifico y espectacular. Los *Big* *Data* están 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 la creación 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. (Joyones Aguilar, 2016).

Cabe recalcar que la *Big* *Data* tiene muchas definiciones que han sido implementadas por grandes científicos tecnológicos, por medio de ello se abarcara hablando de tres definiciones que se puedan entender de manera conceptual.

La primera definición que daremos a conocer es la de *Mickinsey Global* *Institute* que da a conocer que la:

“*Big* *Data* se refiere a los conjuntos de datos cuyo tamaño estas más allá de las capacidades de las herramientas típicas de software de bases de datos para capturar, gestionar y analizar”. *Mickinsey* destaca que la definición puede variar por cada sector, dependiendo de cuales sean los tipos de herramientas de los conjuntos de datos en ese sector o industria (Joyones Aguilar, 2016).

La segunda definición que aporta la empresa multinacional de auditoría *Deloitte* es que:

“*Big* *Data* aplica un conjunto de datos cuyo volumen supera la capacidad de las herramientas informáticas de su uso común, para capturar, gestionar y procesar datos en un lapso de tiempo razonable” (Joyones Aguilar, 2016).

La tercera y última definición la cual es aportada por la empresa *Gartner* es que:

“*Big* *Data* son los grandes conjuntos de datos que tiene tres características principales: volumen, velocidad y variedad” (Joyones Aguilar, 2016).

En definitiva la *Big* *Data* puede variar de distintas formas, según las características que tiene cada empresa o una organización en general, es decir, para algunas organizaciones lo primero que importa es el volumen, capturar la información la cual va a hacer procesada, guardada y actualizada para después incorporarla en procesos de negocio que pueden ser beneficiosos para las organizaciones, para otros les interesa que la información sea guardada con una velocidad impresionante pero en tiempo real, como enfoque cada organización busca el beneficio de la misma enfocado en la *Big* *Data*.

Existen algunos conceptos relacionados como *Data Warehouse*, *Data Mining* y *Cloud Computing.*

***Data Warehouse***

El *Data* *Warehouse* es una evolución de los sistemas de bases de datos relacionales, es un proceso, no un producto. En 1988 los investigadores de *IBM Barry Devlin* y *Paul Murphy* inventaron el término *Warehouse* de información, aunque el considerado padre de los Data Warehouse es *William Harvey Inmon*. Los *Data Warehouse* fueron creados en la década de los 90, utilizan de apoyo para la toma de decisiones y que pueden consultar mediante las tecnologías de los *Data Mining.* La definición de *William Harvey Inmon* dice:

“Una colección de datos que sirve de apoyo a la toma de decisiones, organizados por temas, integrados, no volátiles y en los que el concepto de tiempo varía respecto a los sistemas tradicionales” (López García, 2012).

***Data Mining***

La minería de datos es una herramienta que permite extraer conocimiento de los datos que tenemos almacenados para tratarlos y convertirlos en información útil y objetiva que ayudará al empresario o toda una organización a tomar las decisiones más adecuadas. La definición de la minería de datos dada por (Fayad y otros 1996) es:

“Es un proceso no trivial de identificación válida, novedosa, potencialmente útil y entendible de patrones comprensibles que se encuentran ocultos en los datos”

Según el portal *Daedalus* la Minería de Datos se define como la extracción no ligera de información implícita, previamente desconocida y potencialmente útil. En la actual sociedad de la información, la minería de datos es una herramienta fundamental para analizar y explotar de forma eficaz para los objetivos de cualquier organización (López García, 2012).

***Cloud Computing***

Es una tecnología joven al igual que *Big Data* que brinda la posibilidad de ofrecer servicios a través de internet. Esta tecnología busca tener nuestros archivos e información en internet sin preocuparnos de tener la capacidad suficiente para almacenar dicha información. *Cloud Computing* coge fuerza cuando la provisión de hardware se convierte en un problema, además de costes monetarios los tiene de espacio, escalabilidad es aquí donde *Cloud Computing* es una gran alternativa (López García, 2012).

* + - 1. Tipos de Datos

*Big Data* contiene distintos tipos de datos que pueden ser almacenados en las BD, es frecuente dividir las categorías, nuevas herramientas han surgido para la manipulación y han originado nuevas categorías.

* + - * 1. Datos Estructurados

Los Datos Estructurados se los considera como datos tradicionales, los cuales son datos con un formato y un esquema fijo, el cual posee campos fijos. En estos casos son los datos de las bases de datos relacionales, las hojas de cálculo y los archivos. Los datos estructurales se componen por distintas piezas de información, también vienen en un formato ya especificado, además estos formatos son conocidos como: fecha, nacionalidad, cedula, etc. (Joyones Aguilar, 2016).



*Figura 2.* Representación de Datos Estructurados.

Fuente*:* Autor

Elaboración*:* Autor

La figura 2 representa de manera gráfica los tipos de datos estructurados que pueden ser almacenados dentro de una base de datos relacional, estos datos pueden ser: estadísticos, información de usuarios (nombres, apellidos, cédula), cadenas de texto, contenido de registros, por lo que es una cuestión fácil de usar.

* + - * 1. Datos Semiestructurados

Los Datos Semiestructurados tienen un flujo que pueden ser definidos, pero no es fácil de entender para el usuario. Es decir que 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, como ejemplo para dar a conocer los datos semiestructurados son los web logs. Un web log este compuesto por diferentes piezas de información, cada una sirven para un propósito en específico, ejemplo son las etiquetas de *HTML* y *XML* (Joyones Aguilar, 2016).



*Figura 3.* Representación de Datos Semiestructurados.

Fuente*:* Autor

Elaboración*:* Autor

La figura 3 representa de manera gráfica los tipos de datos semiestructurados, se entiende que este tipo de datos es difícil de entender, pero en si estos datos son una mezcla de estructurados y no estructurados, además, estos tipos de datos representan una organización definida en sus metadatos donde se describen los objetos y sus relaciones.

* + - * 1. Datos no Estructurados

Los Datos no estructurados son datos sin tipos predefinidos. Estos se almacenan como documentos u objetos sin estructura uniforme, y se tiene poco o ningún control sobre ellos, como ejemplo de estos datos son: texto, video, audio, fotografía, etc. Al menos el 80% de la información de las organizaciones contiene este tipo de datos.



*Figura 4.* Representación de Datos no Estructurados.

*Fuente:* Autor.

*Elaboración:* Autor.

La figura 4 representa de manera gráfica los tipos de datos no estructurados, estos datos no tienen un formato en específico, pueden estar almacenados en múltiples formatos como texto, audio, PDF, Word, correos electrónicos, etc.

* + - 1. Características

Los datos pueden proceder de redes sociales, logs, registros de servidores Web, sensores, imágenes de satélites, audios, radio, etc.

Dentro de ellos se destacarán las características principales que contiene la *Big Data:*

**Volumen:** Cada organización contiene y guarda grandes volúmenes de datos, desde el año 2000 se han almacenado en el mundo 800.000 petabytes, se espera que en el año 2020 se alcances los 35 *zettabytes* (ZB) (Joyones Aguilar, 2016). Las organizaciones que se enfrentan a volúmenes masivos de datos desconocen cómo pueden gestionar esta información, es por eso que se han creado librerías que puedan analizar y satisfacer las necesidades del mismo, es por eso que más adelante se hablara de Rapids, permite gestionar la información en grandes volúmenes de datos. IBM plantea que el volumen de datos disponible en las organizaciones hoy en día está en ascenso mientras que el porcentaje de los datos que se analiza está en disminución.

**Velocidad:** La velocidad de los datos es de sumamente importancia para el creciente flujo de información en tiempo real que almacena cada organización. Requiere que el procesamiento y posterior análisis debe hacerse en tiempo real para la mejora de toma de decisiones. La importancia de la velocidad de los datos se une a las características de volumen y variedad, de modo que la idea de velocidad no sea asociada a la tarea de crecimiento de los depósitos o almacenes de datos, sino que se aplica a la definición al concepto de los datos en movimiento, es decir, la velocidad a la cual fluyen los datos (Joyones Aguilar, 2016).

**Variedad:** En cuanto a variedad, una organización puede almacenar distintos tipos de datos, pueden ser datos estructurados o no estructurados y cuando estos dos se analizan juntos se requieren de nuevas técnicas.

Variedad representa todos los tipos de datos, y supone un desplazamiento fundamental en el análisis de requisitos desde los datos estructurados tradicionales hasta la inclusión de los datos en bruto, semiestructurados y no estructurados como parte del proceso fundamental. Sin embargo, el éxito de una organización dependerá de su capacidad para resaltar el conocimiento de los diferentes tipos de datos disponibles en ella, que incluirá tanto los datos tradicionales como los no tradicionales (Joyones Aguilar, 2016).

**Veracidad:** Según IBM, en su definición de *Big Data,* al comentar la característica de veracidad proporciona un dato estremecedor: “Uno de cada tres líderes de negocio no se fija de las informaciones que utilizan para tomar decisiones”. El establecimiento de la veracidad o fiabilidad de *Big Data* supone un gran reto a las fuentes de datos que crecen (Joyones Aguilar, 2016).

**Valor:** Cada organización se enfoca en obtener información de los grandes datos de manera rentable y factible, el cual les ayudara a crecer exponencialmente, aquí es donde las tecnologías de código abierto permiten realizar el análisis de los datos obtienen una fuente de valor muy determinante que les permita a las organizaciones saber que datos son más factibles o que datos no lo son.

* + - 1. Ventajas y Desventajas

*Big Data* produce un gran avance, permite automatizar el proceso en gran medida, permitiendo obtener una respuesta de los datos muy rápido, existen ventajas y desventajas que serán detalladas de la siguiente manera:

Ventajas:

* Implementación de mejoras tecnologías o librerías que permitan la posibilidad de analizar, procesar datos en tiempo real y descubrir necesidades de puntos de mejora para una organización.
* El análisis de los datos puede mejorar sustancialmente dentro de la organización, permitirá reducir numerosos riesgos que puedan existir.
* Facilitar que las empresas puedan evaluar productos mediante el análisis de datos, obteniendo información muy valiosa, permitirá crear nuevos productos y ofrecer mejores servicios.
* Mejorar la accesibilidad y fluidez de la información dentro de la empresa y organización.

Desventajas:

* Para las empresas que han evolucionado y han decidido que su empresa siga a delante para conseguir metas más ambiciosas, suelen minimizar el riesgo, es decir lo más importante son los resultados que la inversión genera mas no se escatima ni los recursos necesarios para afrontar un proyecto de Big Data, así como también los costos asociados. (Estrada et al., 2018).
* Las acciones lentas permiten obtener un duro fracaso para una organización, esperan que otros lo usen para ver si funciona o no. A demás de esto, aunque las empresas sean del mismo sector, siempre la información va a hacer diferente y el análisis no va a hacer el mismo.
  + 1. Procesamiento de Datos

Los datos pueden ser cualquiera dato como número o carácter que son representados por distintos valores, pueden ser manipulados en otras formas para que sean útiles y comprensibles, convirtiendo los datos en información.

El procesamiento de datos es utilizando principalmente 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 o información llega se procesa y se da una salida automática, permite tener una respuesta oportuna y confiable para el usuario o sistema que necesita de esta solución.

Cabe recalcar que el tiempo de respuesta para el procesamiento de un flujo de datos, deberíamos hablar de un procesamiento de datos casi en tiempo real Shilpi & Saurabh (2001) afirma:

“Una de las verdades más grandes sobre el análisis en tiempo real es que nada es en realidad en tiempo real; es un mito. En realidad, está cerca del tiempo real. Dependiendo del rendimiento y la capacidad de una solución y la reducción de las latencias operativas, los análisis podrían estar cerca del tiempo real, pero, mientras que día a día estamos cerrando la brecha entre el tiempo real y el tiempo casi real, es prácticamente imposible eliminar la brecha debido a las latencias computacionales, operacionales y de red.”

Para que los datos puedan ser procesados deben pasar por algunas fases, se denominan “Fases de procesamiento de datos”, son las siguientes:

**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, la entrada de datos se considera una tarea básica y necesaria para la mayoría de las organizaciones.

**Preparación de datos:** La preparación de los datos, a menudo se lo denomina como “preprocesamiento”, los datos sin procesar se limpian y organizan.

Los datos sin procesar se verifican diligentemente para detectar cualquier error que pueda surgir, el propósito es eliminar datos que estén incorrectos o que no estén acordes con la información (datos redundantes, incorrectos e incompletos) y comenzar a crear datos de alta calidad.

**Proceso:** Se ejecutan las operaciones precisas para cambiar los datos en información importante, se realiza la operación de salida que emite varios métodos de presentación, se tomará como base para la toma de decisiones de una organización.

**Salida:** En el procesamiento de datos se proyecta como actividad extra, la administración de resultados de salida, es decir se describe como los procesos obligatorios para que la información importante llegue al usuario.

* + - 1. Procesamiento de Datos para llegar a *RAPIDS*.

La invención de las computadoras es una clara necesidad de información y procesamiento de datos, los informáticos tuvieron que escribir programas personalizados para procesar datos y estos probablemente se almacenaban en una tarjeta perforadora.

A través de los últimos años, han existido grandes avances tecnológicos que han llevado a un aumento dramático en el tráfico de información. Por ejemplo, las redes móviles han aumentado cobertura, rendimiento de datos y los teléfonos fijos se están actualizando lentamente de cobre a fibra óptica.

El gran volumen de datos ha introducido nuevos desafíos en la captura, almacenamiento, análisis, búsqueda, uso compartido, transferencia, visualización, consulta, actualización y privacidad de toda la información que es almacenada como: información aérea, imágenes, sonido, datos meteorológicos, etc.

El proceso de evolución de datos determinará de manera gratificante como ha ido evolucionando, hasta llegar al proceso de construcción de la librería *RAPIDS*.

* + - * 1. *Distributed Storage*

Los sistemas de almacenamiento distribuido proporcionan acceso confiable a los datos a través de la redundancia distribuida en nodos que son confiables individualmente.

El almacenamiento distribuido se centra en dos aspectos importantes *Hadoop* y *MapReduce,* permiten el almacenamiento distribuido y el procesamiento en múltiples máquinas.

***Hadoop:*** *Hadoop* fue creado por *Doug Cuttin*, creador de *Apache Lucene*, la biblioteca de búsqueda de texto ampliamente utilizada. Hadoop tiene su principal origen en Apache Nutch, este es un motor de búsqueda web de código abierto, que forma parte del proyecto *Lucene*. El creador del proyecto *Doug*, explica cómo surgió el nombre *Hadoop*, se detalla de la siguiente manera White (2015) afirma:

*“*El nombre que mi hijo le dio a un elefante amarillo de peluche. Corto, relativamente fácil de deletrear y pronuncia, sin sentido, y no se usa en otro lugar: esos son mis criterios de denominación. Los niños son buenos para generar tales nombres.”

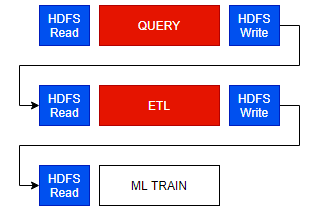
*Hadoop* está siendo utilizada por bastantes compañías como *Yahoo!, Last.fm, Facebook y el New York Times.* En abril del 2008 *Hadoop* rompió un récord mundial al convertirse en el sistema más rápido para clasificar terabyte completo de datos.

Turner (2011) afirma:

“La tecnología subyacente fue inventada por *Google* en sus primeros días para que pudiera indexar últimamente toda la rica información textural y estructural que estaban recopilando y luego presentar resultados provechosos, significativos y procesables a los usuarios. No había nada en el mercado que pudiera dejar hacer eso, así construyeron su propia plataforma, las innovaciones de *Google* se incorporaron a *Nutch*, un proyecto de código abierto y *Hadoop* luego se separó de esto. *Hadoop* fue diseñada para resolver problemas en los que tiene muchos datos, tal vez una mezcla de datos complejos y estructurados y no encaja bien en las tablas, para estas situaciones donde quieres que se ejecute un respectivo análisis, el cual debe ser profundo y computacionalmente extensivos, como la agrupación y la orientación. “

*Hadoop* (ver figura 5) está diseñado para trabajar y ejecutarse en una gran cantidad de máquinas que no comparten memoria ni discos. Significa que pueden comprar un montón de servidores básicos, colocarlos en un rack (Los armarios rack permiten, en un espacio reducido, contar con todo conectado: computador, *UPS*, teléfono o cualquier otra herramienta que se necesita. Estos se han convertido en una parte imprescindible en empresas ya que son lugares donde se requiere tener unas instalaciones seguras y de prestaciones adecuadas). Y ejecutar *Hadoop* en cada uno, lo que realiza es romper los datos en pedazos para que se extiendan a través de los diferentes servidores, también realiza un respectivo seguimiento de donde residen los datos, además, los datos almacenados en un servidor que se desconecta o mueren pueden replicarse automáticamente desde una copia valida conocida. (Turner, 2011).

*Hadoop* puede 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. Dentro de los resultados esperados, lo que hace *Hadoop* es que devuelve todos los resultados en un todo unificado.



*Figura 5.* Procesamiento *Hadoop*, leyendo desde el disco.

*Fuente:*(Srinath & Kraus, 2019)*.*

*Elaboración:* Autor.

La figura 5 representa la interpretación de *Hadoop*, consiste en *HDFS Read* que son archivos de lectura, pasan a *Query* que son interpretadas como una cadena de consulta en las respectivas bases de datos, siguiendo con el esquema va a *HDFS Write* que son archivos de escritura*; HDFS Read* nuevamente archivos que necesitan ser leídos, esos archivos de lectura pasan a *ETL* (Extracción, transformación y carga) que realiza un proceso de reformateo, limpiarlos y cargarlos en otra base de datos para después analizarlos y nuevamente tener archivos de escritura; *HDFS Read* pasan a *ML Train* el cual consiste en un proceso de entrenamiento de un modelo, implica utilizar algoritmos de aprendizaje con los respectivos datos, para obtener predicciones sobre nuevos datos.

***MapReduce:*** Se ha convertido en una de las plataformas de computación paralela para procesar datos en escala de *terabytes* y *petabytes*. Utilizando diariamente en empresas como *Yahoo!, Google, Amazon y Facebook* y adaptado por varias universidades, permite una paralelización fácil de cálculos intensivos de datos en muchas maquinas. Una principal característica que tiene *MapReduce* es que intercala computación secuencial y paralela. Está diseñada para cálculos sobre conjuntos de datos masivos (Howard et al., 2010).

*MapReduce* es un modelo de programación para el procesamiento de datos. Los programas *MapReduce* son inherentemente paralelos, por lo que ponen análisis de datos a gran escala en manos de cualquier persona con suficientes maquinas a su disposición.

El funcionamiento de *MapReduce* es que divide el procesamiento en dos fases: la fase de mapa y la fase de reducción. Cada fase tiene pares clave-valor como entrada y salida, cuyos tipos puede ser elegido por el programador.

* + - * 1. *Spark In-Memory Processing*

*Spark* 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 *RDD* (conjunto de 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* (grafico a cíclico dirigido), puede optimizar la computación, particularmente la computación iterativa, es esencial para las tareas teóricas de datos como la optimización y el aprendizaje automático (ver figura 6).

En *Apache Spark* el cálculo en memoria define un lugar de almacenar datos en algunas unidades de disco lento, los datos se guardan en la memoria de acceso aleatorio (*RAM*). Los datos que han sido almacenados se procesan en paralelo, al utilizar el procesamiento en memoria, se destaca un patrón, analizar datos de gran tamaño y también reduce el coste de la memoria, está conformado por dos principales cálculos en memoria los cuales son: Almacenamiento en *RAM* y Procesamiento distribuido en Paralelo.

*Apache Spark* es una plataforma de computación en *clúster* que, proporcionada una *API* para programación distribuida similar a *MapReduce*, pero está diseñado para ser veloz para realizar consultas y algoritmos interactivos. Cabe recalcar que *Spark* fue diseñado desde cero para admitir aplicaciones de *Big Data* y ciencia de datos en particular, mantiene los datos en memoria tanto como sea posible durante el transcurso de las aplicaciones, evitando la recarga de datos entre iteraciones.



*Figura 6.* Spark procesamiento en memoria.

*Fuente:*(Srinath & Kraus, 2019)*.*

*Elaboración:* Autor.

La figura 6 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 respectivas BD, pasan por ETL es un proceso de reformateo, limpieza y carga, para después analizarlos y guardarlos en memoria cache, terminado el proceso de EL se enfoca en ML, implica utilizar algoritmos de aprendizaje con el fin de obtener predicciones futuras, mediante Spark procesamiento en memoria se obtiene menos código, lenguaje flexible y principalmente en memoria.

* + - * 1. *GPU-Accelerated Computer*

La computación acelerada por *GPU* es el empleo de una unidad de procesamiento gráfico (ver figura 7), juntamente con una unidad de procesamiento de computadora, 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.

Las tecnologías de computación acelerada como *GPU* reciben una atención considerable en la *HPC* (*High performance Computing* o Computación de alto rendimiento) moderna. En comparación con los aceleradores clásicos y las CPU tradicionales, la computación acelerada exhibe una mayor densidad de cómputo, sino que también poseen un ancho de banda de memoria significativo y capacidades similares a las de los vectores para transmitir datos.

Por supuesto la aceleración a través de hardware especializado o procesadores no es nueva: varias áreas de aplicación como gráficos, multimedia, redes, criptografía, etc., generalmente se acelera en *PC* y dispositivos integrados (Matsuoka et al., 2009).

Matsuoka et al.(2009) afirma:

“Existen tres desafíos clave en el uso de *GPU* para aplicaciones de uso intensivo de datos, el cual los clasifica de la siguiente manera: Ejecución, Tubería de Transmisión, Coalescencia de Memoria.”



*Figura 7.* Procesamiento tradicional GPU.

*Fuente:*(Srinath & Kraus, 2019)*.*

*Elaboración:* Autor.

La figura 7 representa el proceso de trabajo de Procesamiento tradicional de *GPU*, *HDFS Read* se lee el archivo pasan a *GPU* *Read*, la *GPU* realiza múltiples tareas en paralelo, pasan a 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* 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, un lenguaje rígido y sustancialmente debe realizarse las principales tareas en *GPU*.

Cabe mencionar que en la sección 2.2 nos enfocaremos en el funcionamiento, diferencias entre *GPU* y *CPU* y las principales arquitecturas que utilizan la misma GPU.

* + - * 1. *RAPIDS*

*RAPIDS* es el punto fuerte de esta temática (ver figura 8), se enfoca en escribir en memoria del sistema, donde se mantendrá cosas en la *GPU* siempre como sea posible, tiene formatos comunes que permite obtener un ecosistema acelerado.



*Figura 8.* RAPIDS.

*Fuente:* (Srinath & Kraus, 2019).

*Elaboración:* Autor.

En la figura 8 representa el proceso de trabajo de *RAPIDS*, obtiene resultados beneficiosos, permite una simplificación significativa y obtiene algunas características principales como: mismo código, lenguaje flexible y principalmente en *GPU*; comienza con *Arrow* *Read*, después pasa por *Query* que son interpretadas como una cadena de consulta en las respectivas bases de datos y pasan a *ETL* (Extracción, transformación y carga), realiza un proceso de reformateo, limpiarlos y cargarlos, por último, realizan *ML Train,* enfocado en realizar un proceso de entrenamiento de un modelo, implica utilizar algoritmos de aprendizaje con los respectivos datos para obtener predicciones sobre nuevos datos. La librería RAPIDS ya simplifica algunos pasos que han sido mencionados anteriormente, RAPIDS es una librería de computación acelerada.

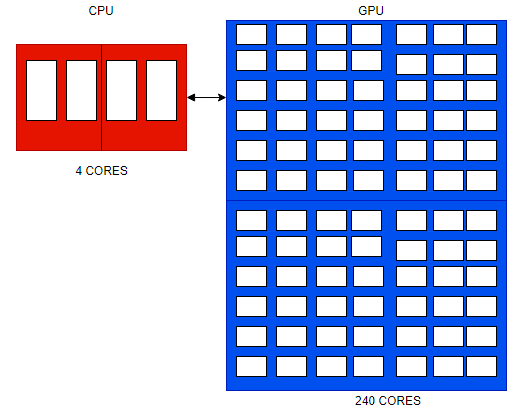
* 1. *GPU*

Se produjo la evolución en la informática de masas, en el mundo entero de la computación profesional la empresa *Sillicon Graphics* dedico muchos esfuerzos durante la década de 1980 a desarrollar soluciones orientadas a gráficos tridimensionales, con el fin de obtener una excelente visualización en cuanto a entornos gráficos*. Sillicon Graphics* popularizó el uso de tecnologías para *3D* en diferentes sectores como el gubernamental, defensa, visualización científica y técnica, además, de proporcionar herramientas para crear efectos cinematográficos nunca vistos.

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 inversivos en primera persona, como *Doom, Duke Nukem 3D o Quake*, que acercaban a sector de los videojuegos para ordenadores personales para entornos *3D*. 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. Estos primeros desarrollos representaron el principio de una nueva era de gráficos *3D* (Guim & Rodero, 2019).

*GPU* se ha convertido en una parte integral de los principales sistemas informáticos actuales. En los últimos seis años, ha habido un mercado en aumento para el rendimiento y las capacidades de las *GPU*. La *GPU* moderna no es solo un potente motor de gráficos (ver figura 9), es un procesador programable altamente paralelo con un ancho de banda de memoria y aritmética que supera a su contraparte de *CPU*. Además, es un procesador de propósito específico, permite optimizar grandes cantidades de datos y realizar las mismas operaciones una y otra vez (Owens et al., 2008).

La figura 9 demuestra que la *GPU* contiene tarjetas gráficas y realiza cálculos científicos de propósito general, el modelo de computación sobre tarjetas gráficas es usar conjuntamente una *CPU* (*Central Processing Unit*) y una *GPU* de manera que formen un modelo de computación heterogéneo.



*Figura 9.* Modelo de computación heterogéneo CPU + GPU.

*Fuente:* (Pérez Represa et al., 2016).

*Elaboración*: Autor.

Owens et al. (2008) afirma:

“La *GPU* está diseñada para una clase particular de aplicaciones, una comunidad en crecimiento ha identificado aplicaciones con características similares y las ha mapeado con éxito en la *GPU*:

* Requisitos computaciones son grandes: El renderizado en tiempo real requiere de miles de millones de pixeles por segundo y cada pixel requiere cientos o más operaciones. La *GPU* debe entregar una enorme cantidad de rendimiento informático para satisfacer la demanda de aplicaciones complejas en tiempo real.
* Paralelismo Sustancial: La canalización de gráficos es adecuada para el paralelismo. Operaciones en los vértices y fragmentos se adaptan bien a unidad de cómputo programables paralelas de grano fino y estrechamente acopladas, que a su vez son aplicables a muchos otros dominios computacionales.
* Rendimiento más importante que la latencia: Las implementaciones de *GPU* de la tubería de gráficos priorizaron el rendimiento sobre la latencia. El sistema visual humano opera en escalas de tiempo de milisegundos, mientras que las operaciones dentro de un procesador moderno toman nanosegundos.”
  + 1. *GPU vs CPU*

La *CPU* está diseñada para el procesamiento en serie, se compone de pocos núcleos muy complejos que pueden ejecutar unos pocos programas al mismo tiempo; *GPU* tienes cientos o miles de núcleos sencillos que pueden ejecutar cientos miles de programas específicos a la vez (ver figura 10). La *GPU* requieren un alto grado de paralelismo tradicionalmente una instrucción y múltiples datos, la *GPU* se vuelve más apta en el trabajo de procesamiento en paralelo, es por esto por lo que es diseñada para satisfacer los procesos informáticos, que necesitan de un procesamiento ligeramente rápido.

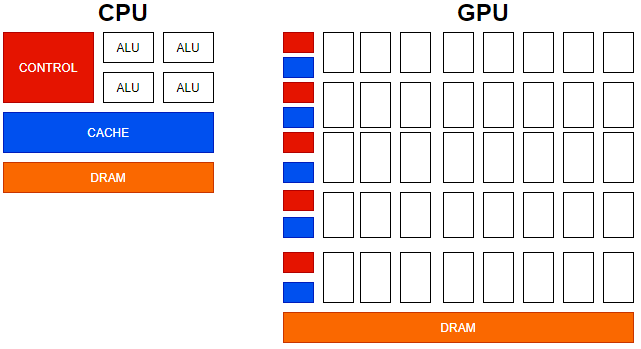
Guim & Rodero (2019) afirma:

“Las diferencias tan grandes entre el rendimiento de *CPU* y *GPU* se deben principalmente a una cuestión de filosofía de diseño. Mientras que las *GPU* están pensadas para explotar el paralelismo a nivel de datos con el paralelismo masivo y una lógica bastante simple, el diseño de una *CPU* esta optimizada para la ejecución eficiente de código secuencial. Las *CPU* utilizan lógica de control que permite un paralelismo a nivel de instrucción y fuera de orden y utilizan memorias cache bastante grandes para reducir el tiempo de acceso a los datos en memoria. También hay otras cuestiones, como el consumo eléctrico o el ancho de banda de acceso a la memoria. Las *GPU* actuales tienen anchos de banda a memoria en torno a diez veces superiores a los de la *CPU*, entre otras cosas porque las *CPU* deben satisfacer requisitos heredados de los sistemas operativos, de las aplicaciones o dispositivos de entrada y salida.”

Por medio de la *GPU* ha existido una evolución inmensa y muy rápida desde el punto de vista de la programación de las *GPU*, han cambiado el propósito de los dispositivos. Desde el año 2000 las *GPU* utilizaban unidades aritméticas programables para devolver el color de cada pixel de a pantalla.

Guim & Rodero (2019) afirma:

“Las apariciones aritméticas que se aplican a los colores de entrada y texturas las podía controlar completamente el programador, los investigadores observaron que los colores de entrada podían ser cualquier tipo de dato. Los datos de entrada eran datos numéricos que tenían algún significado más allá de un color, los programadores podían ejecutar cualquiera de los cálculos que necesitaban sobre esos datos de entrada.”

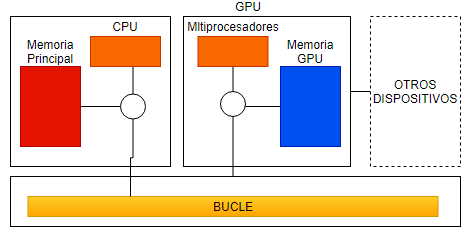


*Figura 10.* Comparativa de memoria y lógica de control para *CPU* y *GPU*.

*Fuente:* (Guim & Rodero, 2019).

*Elaboración*: Autor.

En la figura 10 se demuestra la comparativa de *CPU* y *GPU*, se requiere de menos lógica de control para cada flujo de ejecución. Al mismo tiempo dispone de una pequeña memoria cache, permite que los flujos compartan memoria y tengan el respectivo ancho de banda.



*Figura 11.* Interconexión entre *CPU* y *GPU* mediante *PCle*.

*Fuente:* (Guim & Rodero, 2019).

*Elaboración*: Autor.

Guim & Rodero (2019) afirma:

“La figura 11 es la comunicación entre *CPU* y *GPU* se lleva a cabo por medio de un puerto dedicado. En la actualidad el *PCI Exprres* o *PCIe* (*peripheral component interconnect express*) es el estándar para ejecutar esta comunicación”

Hay que tener en cuenta que una *GPU* no puede acceder a la memoria principal, es decir directamente, en cambio la *CPU* no puede acceder directamente a la memoria de una *GPU,* por lo tanto, hay que copiar los datos entre *GPU* y *CPU* de manera explícita, es decir en ambas direcciones.

* + 1. Arquitecturas *GPU*

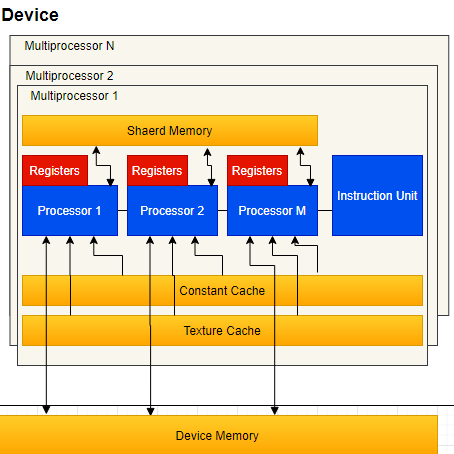
La arquitectura *GPU* (ver figura 12) durante las primeras generaciones tenían una cantidad de núcleos bastante reducida, ámbitos de consultas y dedicación por parte de los desarrolladores rápidamente se incrementó hasta hoy en día, cuando hablamos de dispositivos de tipo *many-core* con centenares de núcleos en un único chip.

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

La *GPU* siempre ha sido un procesador con amplios recursos computacionales, ha evolucionado desde una función fija, a un procesador de propósito especial en un procesador programable paralelo con funcionalidad adicional y función de propósito especial.

En general, una arquitectura *GPU* está hecha de un número escalable de multiprocesadores de transmisión (*SM* el nombre, se utiliza en una arquitectura de *GPU* de *Fermi*. Las versiones mejoradas de la arquitectura *SM* se denominan *SMX* en un *Kleper* y *SMM* en una arquitectura *Maxwell*), y cada *SM* contiene varios núcleos de procesamiento en paralelo (procesadores de flujo, *SP*), planificadores de deformación, unidades de despacho, unidades de funciones especiales (*SFU*), memoria local, memoria compartida, memoria de textura, L1, *Caches* L2 y cache constante.

Una placa *GPU* tiene un ancho de banda alto, memoria global sin chip, conocida como *DDR* gráfica, hasta varios *gigabytes* de tamaño. Se anuncia que la *GPU* incluye una memoria global de 12 *GB*. Una memoria compartida en el chip proporciona alta velocidad de acceso a la memoria tan rápido como un registro y una parte de la memoria compartida se puede configurar manualmente en un *Cache* L1. La memoria constante como la memoria de textura es colocada en una memoria global, pero se almacenan en una cache (Thi Yen et al., 2017).



*Figura 12.* Arquitectura básica de *GPU*.

*Fuente:* (Huang et al., 2013)*.*

*Elaboración*: Autor.

En la figura 12 representa la arquitectura *GPU*, tiene su propia memoria llamada memoria global. A excepción de la memoria global, la *GPU* tiene caches especiales como cache constante, cache textura y memoria compartida. Dentro de la *GPU* hay varios multiprocesadores y dentro del multiprocesador hay varios procesadores de flujo. El programa ejecutado por *GPU* se dividirá en dos niveles de paralelismo. El primer es el paralelismo grueso con respecto a los bloques controlados por multiprocesadores. El segundo es el paralelismo final con respecto a los subprocesos ejecutados en procesadores de flujo por cada bloque, se puede asignar una memoria especial llamada memoria compartida, es rápida y limitada (Huang et al., 2013).

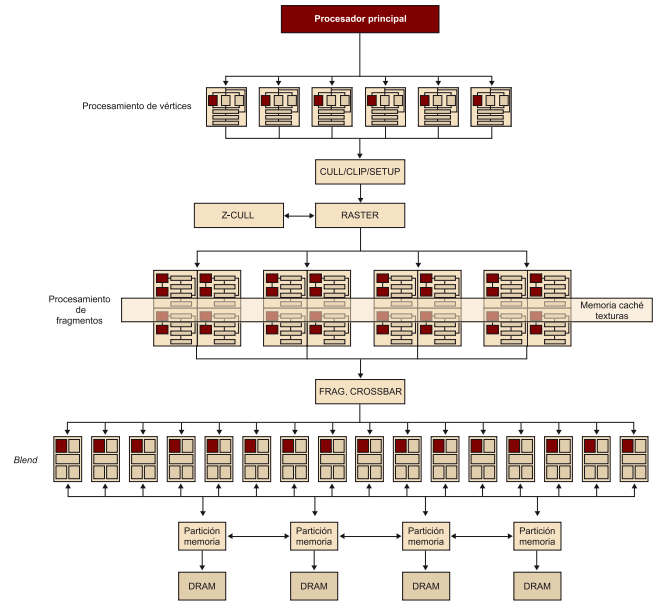
A continuación, en las secciones siguientes se detallará las arquitecturas que utilizan *GPU*, las cuales desde un punto de vista científico y experimental de desarrollo son muy primordiales:

* + - 1. Arquitectura *NVIDIA* *GeForce*

La Arquitectura *NVDIA* *GeForce* (ver figura 13) brinda diferentes productos, se dividen en tres fases principales, las siguientes:

* *GeForce*: Enfocada a un mercado de consumo multimedia como videojuegos, edición de video, fotografía digital.
* Cuadro: Enfocada para soluciones profesionales las cuales requieren modelos *3D*, como por ejemplo en el ambiente de ingeniería y arquitectura.
* *Tesla*: Enfocada para la computación de altas prestaciones, como por ejemplo el procesamiento de información sísmica, simulaciones bioquímicas, modelos meteorológicos y cambio climático, computación financiera y análisis de datos.

En este apartado nos enfocaremos en la serie *NVIDIA* *GeForce*, es de uso de *GPU* pensada para tratamiento de gráficos. A pesar de que no es la arquitectura muy actual nos permitirá interpretar y conocer de mejor manera cómo funciona (Guim & Rodero, 2019).



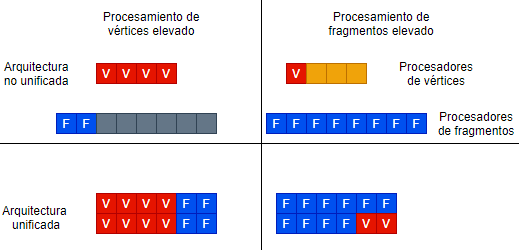
*Figura 13.* Esquema de la Arquitectura de la *GPU* *GeForce* de *NVIDIA*.

*Fuente*: (Guim & Rodero, 2019).

En la figura 13 se envía tres tipos de datos, estos son: instrucciones, texturas y vértices. Los vértices se encargan de aplicar un programa, tiene que ser específico porque ejecuta transformaciones sobre cada uno de los vértices de entrada. *GeForce* es el primer programa ejecutado en el procesador de los vértices.

* + - 1. Arquitectura Unificada

En las arquitecturas unificadas (ver figura 14) no existe la división a nivel de hardware entre procesadores de vértices y procesadores de fragmentos, son capaces de trabajar a nivel de vértice como a nivel de fragmento, sin estar especializada en un tipo concreto. Con arquitecturas unificadas, no hay partes específicas del chip asociadas a una etapa concreta del pipeline, sino un único tipo de unidad se encarga de ejecutar todas las operaciones, sea cual sea su naturaleza. También ofrecen un mayor potencial para hacer computación de propósito general (Guim & Rodero, 2019).



*Figura 14.* Comparativa de asignación de procesadores de una *GPU* en el procesamiento de vértices y de fragmentos en arquitecturas unificadas y no unificadas.

*Fuente:* (Guim & Rodero, 2019).

*Elaboración*: Autor.

La figura 14 muestra la respectiva ejecución de dos aplicaciones no balanceadas y como la arquitectura unificada puede ofrecer una solución más eficiente y factible.

* + - 1. Arquitecturas orientadas a computación de propósito general sobre *GPU*

Los procesadores gráficos pueden ser utilizados para ejecutar aplicaciones que son tradicionalmente ejecutados en *CPU*. La computación de propósito general se conoce como *GPGPU* (*general-purpose computing on Graphics processing unit*) (Guim & Rodero, 2019).

La *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 bases de datos, etc. El principal argumento de *GPGPU* es utilizar el procesador de fragmentos como una unidad de cómputo, también hay que tener en cuenta que la entrada y salida es limitada, es decir, pueden hacer lecturas arbitrariamente, pero hay restricciones para las escrituras (Guim & Rodero, 2019).

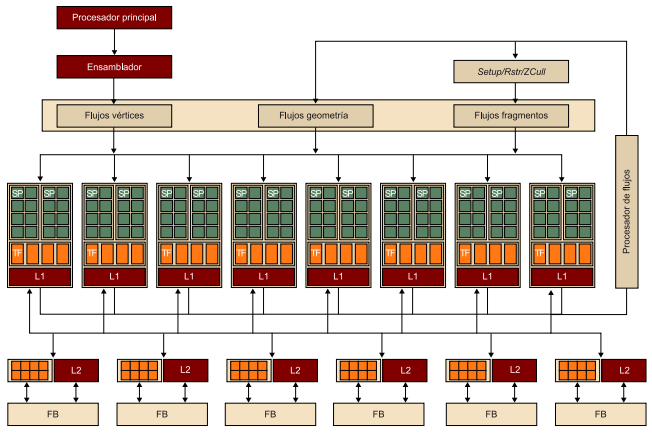
En *GPU* la computación se realiza normalmente por un flujo de procesador de fragmentos, tendrán que ejecutarse en las unidades funcionales de la *GPU* correspondientes. Se establece un símil entre el programa en *CPU* y la rasterización en *GPU*. La rasterización determina los pixeles de flujo de datos, estos se ven afectados por primitivas generadas a partir de los vértices y generan un fragmento para cada uno (Guim & Rodero, 2019).

* + - 1. Arquitectura *NVIDIA*

*NVIDIA* permite analizar enormes cantidades de datos y realizar predicciones comerciales precisas a una velocidad sin precedentes.

*NVIDIA* a finales del año 2006 presento un hardware orientado a computación general de prestaciones llamada *Tesla*, esta se empezó a desarrollar a mediados del año 2002. *Tesla* ofrece un hardware de prestaciones muy altas, por ejemplo, bloques de procesadores gráficos. Esta sección se centrará en la arquitectura *G80* (ver figura 15), la arquitectura *G80* representa un importante salto tecnológico por el hecho de implementar una arquitectura unificada (Guim & Rodero, 2019).

La figura 15 representa la arquitectura *G80* de *NVIDIA* es una arquitectura unificada en toda su totalidad, se orienta en la ejecución masiva de flujos y se ejecuta en un estándar *IEEE* 754 (Guim & Rodero, 2019).

*Figura 15.* Esquema de arquitectura *G80* de *NVIDIA*.

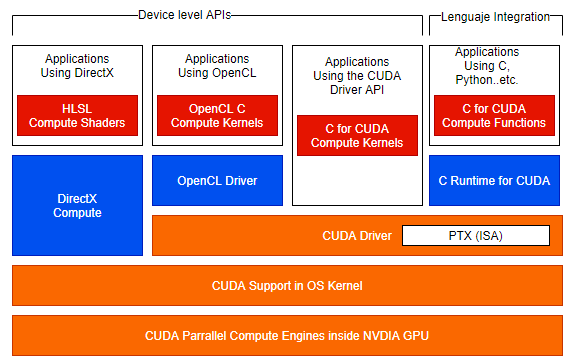
*Fuente:* (Guim & Rodero, 2019).

La arquitectura *G80* de *NVIDIA* presento una mejora en el procesamiento gráfico e incremento de prestaciones, la clave principal es la mejora de la capacidad de cálculo. El funcionamiento básico de la arquitectura *G80* para ejecutar un programa es: Los datos de entrada se procesan mediante un hardware especializado, se encarga de distribuir datos que puedan ser utilizados al máximo número de unidades para obtener la capacidad máxima de cálculo durante la ejecución del programa. También un controlador de flujos se encarga de controlar la ejecución de flujos que ejecutan cálculos de manera coordinada (Guim & Rodero, 2019).

* + - 1. Arquitectura *CUDA*

La arquitectura *CUDA* (ver figura 16) es de computación paralela y revolucionaria ya que ofrece el rendimiento de la mundialmente conocida como tecnología de procesamiento gráfico de *NVIDIA* para aplicaciones de uso general de cálculo en la *GPU*.

Las aplicaciones que son ejecutadas en una arquitectura *GPU* aprovechan una base instalada de cien millones de *GPUs* habilitadas para *CUDA* en ordenadores de sobremesa y pórtatiles, estaciones de trabajo profesionales y clusters de superordenadores (NVIDIA, 2009).



*Figura 16.* Arquitectura *CUDA*.

*Fuente:* (NVIDIA, 2009).

*Elaboración*: Autor

En la figura 16 representa la arquitectura *CUDA* consta de algunos componentes principales, son:

* Motores de cálculo paralelo de las *GPUs*.
* Soporte a nivel de kernel del Sistema Operativo.
* Controlador en modo Usuario, y esta proporciona una *API* en nivel de dipositivo.
* Arquitectura de conjuntos de instrucciones para núcleos y funcionaes de cálculo paralelo.

Pérez Represa et al (2016) afirma:

“En la arquitectura clásica de una tarjeta gráfica podemos encontrar la presencia de dos tipos de procesadores, los procesadores de vértices y los procesadores de fragmentos, dedicados a tareas distintas e independientes dentro del cauce gráfico y con repertorios de instrucciones diferentes. Esto presenta dos problemas importantes, por un lado, el desequilibrio de carga que aparece entre ambos procesadores y por otro la diferencia entre sus respectivos repertorios de instrucciones. De este modo, la evolución natural en la arquitectura de una *GPU* ha sido la búsqueda de una arquitectura unificada donde no se distinguiera entre ambos tipos de procesadores. Así se llegó a la arquitectura *CUDATM*, donde todos los núcleos de ejecución necesitan el mismo repertorio de instrucciones y prácticamente los mismos recursos.”

A continuación en la tabla 1 se refleja los respectivos núcleos de procesamiento de un dispostivo *CUDA* en función de su capacidad de cómputo.

*Tabla 1.* Núcleos de procesamiento de un dispositivo *CUDA*.

|  |  |  |
| --- | --- | --- |
| Capacidad de Cómputo | Nombre de la Arquitectura | Núcleos por multiprocesadór |
| 1.x | *Tesla* | 8 |
| 2.0 | *Fermi* | 32 |
| 2.1 | *Fermi* | 48 |
| 3.x | *Kepler* | 192 |
| 5.x | *Maxwell* | 128 |

*Fuente*: (Pérez Represa et al., 2016).

*Elaboración*: Autor.

En la tabla 1 se indica el nombre de las diferentes arquitecturas así como también el número de núcleos de cómputo o *Streaming* *Processors* característicos de cada una de ellas en función de la capacidad de cómputo (Pérez Represa et al., 2016).

Dentro de la terminología de *CUDA* recibe el nombre de capacidad de cómputo, se indica mediante dos números de la forma M.m, estos representan la revisión mayor y la revisión menor de la arquitectura del dispositivo (Pérez Represa et al., 2016).

* + 1. Funcionamiento *GPU*

Las *GPU* suelen tener grandes cantidades de núcleos de procesamiento a frecuencias de reloj relativamente bajas. Se basa en las primitivas *NVIDIA* *CUDA* (ver figura 15) para la optimización de cómputo de bajo nivel, pero expone ese paralelismo de *GPU* y la velocidad de la memoria de alto ancho de banda a través de interfaces *Python* fáciles de usar.

*NVIDIA* *CUDA* (*Unified Device Architecture*) es como un mini procesador que se encarga de cierto tipo de instrucciones, que suelen poder ejecutarse de manera paralela, incluye un compilador, también un conjunto de herramientas de desarrollo por *NVIDIA*. Además, proporciona un entorno de desarrollo para crear aplicaciones aceleradas por *GPU* de alto rendimiento.

* + - 1. *CUDA*

*CUDA* permite explotar todas las ventajas de la *GPU* utilizando el paralelismo, realiza un sin número de cálculos complejos que se implementan en cantidades enormes de núcleos en las *GPU*.

*CUDA* tiene la posibilidad de paralelizar operaciones de computación mucho más avanzadas, llevan tiempo empleándose en máquinas de tipo *Deep* *Learning* para inteligencia artificial, procesa millones de datos por ciclo de reloj y para la obtención de resultados en tiempo real o el menor tiempo posible dependiendo de la tarea asignada.

Las bibliotecas *CUDA* aceleradas por *GPU* permiten la aceleración directa en múltiples dominios, tenemos: Algebra lineal, procesamiento de imágenes y videos, aprendizaje profundo y análisis de gráficos.

Dentro del desarrollo de algoritmos personalizados, puede usar integraciones disponibles con idiomas y paquetes numéricos de uso común, así como *API* de desarrollo bien publicadas.

Las aplicaciones hechas en *CUDA* se implementan en todas las familias de *GPU* *NVIDIA* disponibles en las instalaciones y en instancias de *GPU* en la nube, utilizan capacidades integradas para distribuir cálculos a través de configuraciones múltiples *GPU*, los científicos e investigadores pueden desarrollar aplicaciones que escalen desde estaciones de trabajo de *GPU* como también instalaciones en la nube con miles de *GPU*.

* + - * 1. Ventajas y Desventajas *CUDA*

*CUDA* presenta algunas ventajas importantes frente a otros tipos de computación en *GPU* utilizando *APIs* gráficas, las principales ventajas son:

* Realizar lecturas disperas: consultar cualquier posición de memoria (El et al., 2015).
* Memoria compartida: disposición del programador en un área de memoria que sea compartida entre hilos. Dado su tamaño y rapidez pueden ser utilizados como cache (El et al., 2015).
* Permite lecturas más rápidas desde y hacia la *GPU* (El et al., 2015).
* Ofrece soporte para enteros y operadores a nivel de bit (El et al., 2015).

*CUDA* sin embargo presenta algunas desventajas, son:

* No se puede utilizar recursividad, punteros o funciones, variables estáticas dentro de funciones o funciones con numero de parámetro de variable (El et al., 2015).
* No soporta el renderizado de texturas.
* No soporta números desnormalizados.
* Puede existir un cuello de botella entre la CPU y la GPU por los anchos de banda de los buses y sus latencias. (El et al., 2015).
* Los hilos deben lanzarse en grupos de al menos 32.
  1. *Open* *Source* *RAPIDS*

La biblioteca de código abierto *RAPIDS* brinda la capacidad de ejecutar canalizaciones de ciencia de datos y análisis de extremo a extremo en *GPU*.

*RAPIDS* se enfoca en tareas comunes de preparación de datos para análisis y ciencia de datos. Incluye una *API* de trama de datos familiar que se integra con una variedad de algoritmos de aprendizaje automático para aceleraciones de canalización de extremo a extremo sin pagar los costos de serialización típicos.

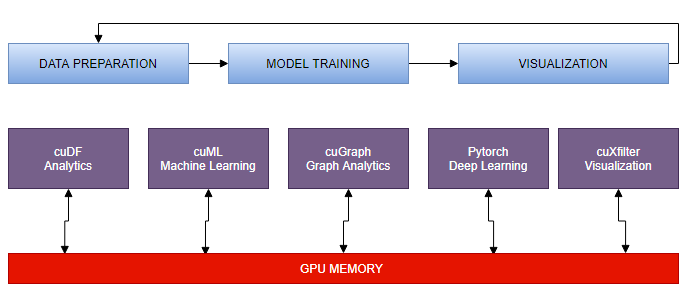
*RAPIDS* permite un procesamiento y capacitación enormemente acelerados en tamaños de conjuntos de datos mucho más grandes, brinda a los científicos de datos un enorme salto de rendimiento para resolver los desafíos comerciales más complejos como predecir fraudes con tarjetas de créditos, pronosticar inventarios de venta minorista y comprender el comportamiento de los clientes, etc.

*RAPIDS* comenzó a partir de los proyectos *Apache Arrow****,* una plataforma de desarrollo en varios idiomas para datos en memoria**y *GoAi* 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 (RAPIDS Development Team, 2018).

El principal objetivo de la librería RAPIDS no es solo acelerar las partes individuales del flujo de trabajo típico de la ciencia de datos, sino acelerar el flujo de trabajo completo de extremo a extremo, se mencionará algunas características muy importantes del porque acelera el flujo de trabajo.

* Integración sin complicaciones:Acelera su cadena de herramientas de ciencia de datos de Python con cambios mínimos de código y sin nuevas herramientas para aprender.
* Precisión del modelo:Aumenta la precisión del modelo de aprendizaje automático iterando en los modelos más rápido y desplegándolos con mayor frecuencia.
* Código abierto:Personalizable, extensible, interoperable; el software de código abierto es compatible con *NVIDIA* y se basa en *Apache* *Arrow*.
* Tiempo de entrenamiento reducido:Mejora drásticamente su productividad con ciencia de datos casi interactiva.
  + 1. Ambiente de Trabajo *RAPIDS*

El ambiente de trabajo de la librería *RAPIDS* (ver figura 17) expone de manera determinada como es el proceso de trabajo aplicando los diferentes puntos como *“Data Preparation”, “Model Training”* y *“Visualization”* y el proceso interno que hay detrás.



*Figura 17.* Ciencia de Datos acelerado de *GPU* de extremo a extremo con *RAPIDS*.

*Fuente:* (Srinath & Kraus, 2019).

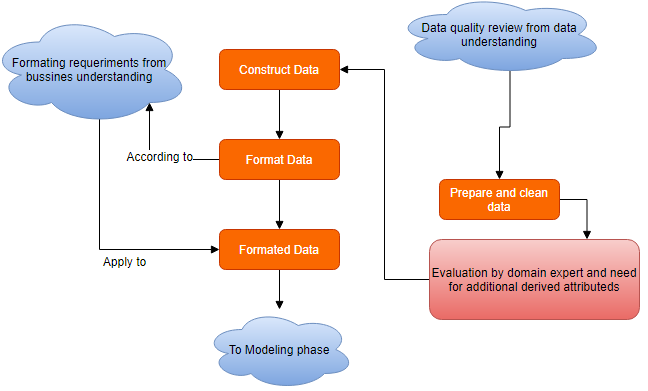
*Elaboración*: Autor.

La figura 17 representa el trabajo de *RAPIDS*, cabe destacar que se menciona pricipales bibliotecas que nos permitira realizar la preparación, entrenamiento y visualización, estas bibliotecas serán explicadas de manera más detalladas en las secciones 2.3.3, 2.3.4, 2.3.5 y 2.3.6 y además hablar de *Apache* *Arrow* que sera explicado en la sección 2.3.2.

* + - 1. Preparación de la Data

La preparación de datos es la etapa fundamental del análisis de datos (ver figura 18), existen demasiada información de baja calidad en varias fuentes de datos, web y diferentes organizaciones, están sumamente enfocadas en cómo transformar los datos en formularios limpios que puedan usarse con fines de alto beneficio.

El enfoque de la fase de preparación de datos es identificar datos de calidad y formatearlos adecuadamente, lo que puede conducir a la generación de patrones de calidad por los algoritmos de minería de datos elegidos. La preparación de datos genera un conjunto de datos más pequeño que el conjunto de datos originales, pero con mejor calidad y datos relevantes que pueden mejorar significativamente la eficiencia de la fase de modelado (Mansingh et al., 2017).



*Figura 18.* Fases de Preparación de Datos.

*Fuente:* (Mansingh et al., 2017).

*Elaboración*: Autor.

La figura 18 representa la principal interacción de preparación de datos y la fase de modelado, por lo tanto, mientras se preparan los datos, se debe realizar un modelado preliminar, es decir, realizar un modelo que permita determinar que la preparación de los datos queda de manera definitiva para después proceder a utilizar algoritmos de *Machine* *Learning*.

*RAPIDS* se enfoca en la preparación de la data utilizando la biblioteca específica *cuDF*, esta permitirá realizar un proceso de configuración a los datos que van a hacer analizados, por ende, realizará una integración, transformación y reducción, más adelante se detallará el funcionamiento de la biblioteca cuDF.

* + - 1. Entrenando el Modelo

El entrenamiento de *Machine* *Learning* debe obtener una precisión predictiva con los distintos datos, debe ser esencial para los científicos de datos, estos se describen como técnicas básicas y avanzadas para entrenar un modelo, algunos puntos deben ser nombrados en este apartado:

* Entrenar
* Evaluar
* Usar la ingeniería de características para crear características que se utilicen para el entrenamiento del modelo.

*RAPIDS* entrena el modelo utilizando la biblioteca *cuML*, para proporcionar datos de entrenamiento para aprender de algoritmos de aprendizaje, estos datos previamente deben estar preparados y deben contener los datos ya clasificados y listos para aplicar el conjunto de los algoritmos de Machine Learning que tiene la biblioteca *cuML.*

* + - * 1. *Machine Learning*

Es un subcampo de las ciencias de la computación y rama de la inteligencia artificial, para desarrollar técnicas que permitan que las computadoras aprendan de manera inteligente.

Además, es la ciencia que nos permite que los computadores aprendan y actúen como humanos, de una forma autónoma, alimentándolas con información, datos en forma de observaciones e interacciones con el mundo real.

El aprendizaje automático (*Machine* *Learning*) utiliza algoritmos programados que reciben y analizan millones de datos para predecir valores de salida.

La aplicación de métodos de aprendizaje automático a grandes bases de datos se denomina minería de datos. La analogía es que un gran volumen de tierra y materia prima se extrae de una mina, que cuando se procesa conduce a una pequeña cantidad de material muy valioso; de manera similar, se procesa un gran volumen de datos para construir un modelo simple con uso valioso, por ejemplo, que tenga una alta precisión predictiva (Alpaydin, 2009).

Algoritmos *Machine* *Learning*

Un algoritmo es una secuencia de instrucciones que deben llevarse a cabo para transformar la entrada en salida, los algoritmos aprenden y optimizan sus operaciones para mejorar el rendimiento, desarrollando “inteligencia” con el tiempo. Los algoritmos encuentran patrones en los datos de entrenamiento que asignan los atributos de los datos de entrada el destino y generan un modelo que captura dichos patrones.

Se necesita algoritmos eficientes para resolver el problema de optimización, así como para almacenar y procesar la gran cantidad de datos que generalmente se tiene. Una vez que se aprende un modelo, su representación y solución algorítmica (Alpaydin, 2009).

Tipos de Algoritmos de *Machine* *Learning*

Realizare un listado de los principales tipos de algoritmos que son utilizados en la actualidad y también una breve descripción de estos.

* **Clasificación y Regresión:** Clasificación se enfoca en los procesos de identificar una nueva categoría de observación sobre la base de formación del conjunto de datos que contiene observaciones de las cuales las categorías son desconocidas. El resultado es una clase, entre un número limitado de clases, con clases nos referimos a categorías arbitrarias según el tipo de problema.

Regresión consiste que el resultado es un número, el resultado será un valor numérico dentro de un conjunto infinito de posibles resultados.

* **Inferencias Estadísticas:** Cada sistema a su vez necesita o requieren de un análisis estadístico simple o complejo, gran parte de los datos que son recopilados por organizaciones exhiben valores perdidos múltiples, interacciones complejas de datos, interacciones complejas de datos.
* ***Clustering*:** El proceso clustering, permite medir la similitud entre objetos, se suelen utilizar diferentes formas de distancia. *Clustering* es una técnica de aprendizaje automático, el aprendizaje realizado es no supervisado, además, juega un papel muy importante en aplicaciones de minería de datos, aplicaciones sobre bases de datos espaciales, aplicaciones web, marketing, diagnóstico médico, análisis de *ADN* en biología computacional, entre otras más (Garre et al., 2007).
* **Descomposición y Dimensión de reducción:** La reducción de dimensión es una metodología principalmente utilizada en distintos campos ligados al procesamiento de datos, representa una etapa de preproceso o ser un elemento esencial para la representación y clasificación de datos, obtener una nueva representación de los datos originales en un espacio de menor dimensión, de forma que se produzca información más depurada, reduzca el tiempo del procesado y genere representaciones visuales entendibles para el ser humano (Castro, 2018).

Cabe recalcar que en la sección 2.3.4 abarcaremos más información de *cuML,* esta biblioteca está conformada por un conjunto de algoritmos de *Machine Learning*, además, indicar el funcionamiento, conceptos y uso de esta biblioteca.

* + - 1. Visualización con *RAPIDS*

La visualización es una representación gráfica de datos, presenta los datos como una imagen o gráfico para facilitar la identificación de patrones y comprender conceptos difíciles, la tecnología permite que los usuarios interactúen con los datos cambiando los parámetros para ver más detalles y crear nuevas ideas (RAPIDS Development Team, 2018).

*RAPIDS* permite que se juegue con un enfoque muy dinámico con las bibliotecas de visualización de ciencia de datos. Para un excelente rendimiento aún mayor, para proporcionar capacidades de visualización de datos de alto rendimiento y alto *FPS*, incluso con conjuntos de datos muy grandes.

Los datos que son visualizados tienden a tener más valor cuando se procesan, analizan, este proceso lo realiza la biblioteca *cuXfilter*, se hablara de manera más específica en la sección 2.3.6; esto lo hace aún más valioso para qué es más fácil consumir información, que a su vez se convierte en conocimiento.

* + 1. *GPU* *Memory* *Apache* *Arrow*

*Apache* *Arrow* es una plataforma en varios idiomas para datos en memoria; especifica un formato de memoria columnar estandarizado independiente del lenguaje para datos planos y jerárquicos, organizado para operaciones analíticas eficientes en hardware moderno. Proporciona bibliotecas computacionales y mensajes de transmisión de copia cero y comunicación entre procesos (Arrow, 2016).

*Apache* *Arrow* especifica un formato en memoria dirigido a grandes conjuntos de datos y bibliotecas para varios idiomas para interactuar con los datos.

*Arrow* evita la necesidad de serialización entre diferentes tiempos de ejecución del lenguaje y comunicación entre conjuntos de datos entre procesos sin copia (Peltenburg et al., 2019).

* + - 1. Funcionamiento de *Apache Arrow*

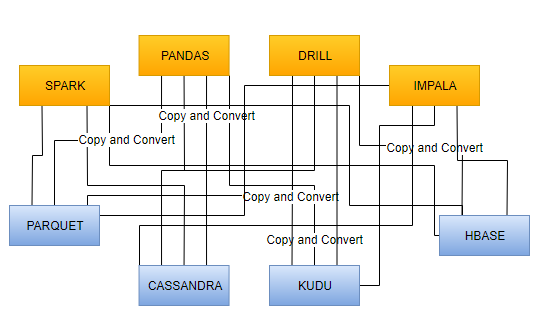
Los conjuntos de datos de *Apache* *Arrow* suelen ser tabulares y almacenados en una abstracción llamada *RecordBatch*. Un *RecordBatch* contiene varias columnas para cada campo de un registro, que están en *Arrow* llamadas matrices. Estas respectivas matrices contienen todo tipo de datos, desde cadenas hasta listas de enteros, listas de marcas de tiempos y otros. Las matrices consisten en varios buffers contiguos de *Arrow*, están relacionados para almacenar los datos de un tipo específico (B et al., 2019).

El funcionamiento de *Apache* *Arrow*, lo detallamos por las principales características, las siguientes son:

Rápido:*Apache* *Arrow* 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 nativa al procesamiento de datos analíticos. El diseño en columnas está optimizado por la localidad de los distintos datos para un respectivo rendimiento en hardware moderno tanto para *CPU* y *GPU* (Arrow, 2016).

Flexible:*Apache Arrow* actúa como una interfaz de alto rendimiento entre varios sistemas, es decir, dentro de cada sistema *Apache Arrow* se visualiza como una interfaz nueva para cada sistema. También se centra en adquirir una amplia variedad de lenguajes de programación estándar. Las implementaciones de *C, C ++, C#, Go, Java, JavaScript, MATLAB, Python, R, Ruby y Rust* están en progreso y se aceptan más idiomas (Arrow, 2016).

**Sin *Apache Arrow:***



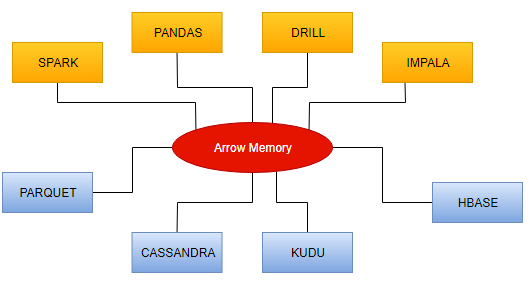
*Figura 19.* Sin *Apache Arrow*.

*Fuente:* (Arrow, 2016).

*Elaboración*: Autor.

La figura 19 representa la no utilización de *Apache Arrow*, por ello existen un flujo muy extenso de copiar y convertir datos en diferentes sistemas, cada sistema tiene su propio formato de memoria interna, un 70 a 80 % de cálculo desperdiciado en serialización y deserialización, hay un cargo excesivo de flujo de los datos, donde se pierde tiempo.

**Con *Apache Arrow:***



*Figura 20.* Utilizando *Apache Arrow.*

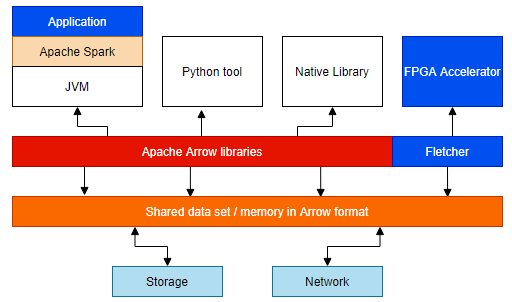
*Fuente:* (Srinath & Kraus, 2019).

*Elaboración*: Autor

La figura 20 representa la utilización de *Apache Arrow,* se demuestra de manera especifica y eficiente que no hay un gasto excesivo de comunicación entre los sistemas, es decir, *Apache Arrow* se encarga de realizar un acceso común para los mismo, utilizan el mismo formato de memoria, además, se destaca el procesamiento paralelo y pueden compartir funcionalidad.

* + - 1. Arquitectura de *Apache Arrow*

En la figura 21 representa la arquitectura de *Apache Arrow,* indica una representación estandarizada como capa de datos común, formato de columnas, estas son compatibles con el *hardware* al iterar entradas en una sola columna (*SIMD*, *caches*, etc.), mejor para muchos algoritmos; bibliotecas y *API* para más de 10 idiomas para construir y acceder a conjuntos de datos.



*Figura 21.* Arquitectura *Apache* *Arrow*.

*Fuente:* (Peltenburg et al., 2019).

*Elaboración*: Autor

* + - 1. Ventajas de *Apache Arrow*
* Proporciona una capa de acceso a datos común a todas las aplicaciones.
* Optimiza la localidad de datos y procesamiento paralelo.
* *Apache* *Arrow* es adecuado para los tipos de procesamiento vectorizado de *SIMD* (instrucción única, datos múltiples).
* Admite *GPU*.
* Todos los sistemas utilizan el mismo formato de memoria.
* Sin gastos generales para la comunicación entre sistemas.
* Los proyectos pueden compartir funcionalidad.
  + 1. Biblioteca *cuDF*

*cuDF* es la biblioteca de *RAPIDS*, de manipulación de *DataFrame* basada en *Apache Arrow*, está enfocada 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\_*, y las funciones de procesamiento de datos pueden operar en esas respectivas columnas (Aramburo, 2019). *cuDF* actualmente tiene dos *API*: *API* de *C ++,* permite el acceso a todas las funciones de alto nivel y una *API* de *Python*, imita frecuentemente a *Pandas*.

Hablando un poco más acerca de la *API* de *Python*, esta es la más fácil de usar, los científicos de datos pueden interactuar con ella casi exactamente como lo harían con *Pandas* para las respectivas transformaciones de datos, la limpieza y el análisis respectivo que se requiere a los datos a evaluar (Aramburo, 2019).

* + - 1. Características de *cuDF*

Conformado por dos aspectos principales *libGDF, PyGDF,* hace que la biblioteca trabaje de manera eficiente para el procesamiento y análisis de los datos.

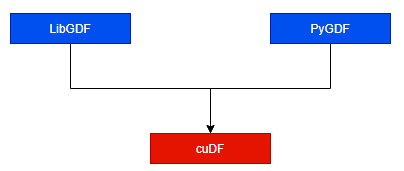


Figura 22: Estructura de biblioteca cuDF.

*Fuente*: (Patterson, 2019).

*Elaboración*: Autor.

La figura 22 representa como está estructurado la biblioteca cuDF, se enfoca en puntos primordiales que son *libGDF* y *PyGDF*, esto hace que la biblioteca se enfoque en realizar un trabajo de manera satisfactoria, además, es un repositorio único que contiene tanto la implementación de bajo nivel como envolturas y *API* de alto nivel, futuros enlaces de idiomas de alto nivel basados en la demanda, comentarios y contribuciones de la comunidad. En fin, en las secciones siguientes incorporaremos las características más principales que abarcan cada una de ellas.

* + - * 1. *LibGDF*
* Biblioteca de bajo nivel que contiene implementaciones de funciones y *API C, C++.*
* Importar y exportar un *GDF* utilizando el mecanismo de *CUDA IPC.*
* Núcleos *CUDA* para realizar operaciones matemáticas basadas en elementos en columnas *DataFrame GPU.*
* Operaciones de clasificación, unión, agrupación y reducción *CUDA* en marcos de datos *GPU*.
  + - * 1. *PyGDF*
* Una biblioteca de *Python* para manipular marcos de datos de *GPU*.
* Interfaz de *Python* para la biblioteca *LibGDF* con funcionalidad adicional.
* Crear *GDF* a partir de matrices *Numpy*, *Pandas* *DataFrame* y *PyArrow* *Tables*.
* Compilación *JIT* de funciones definidas por el usuario usando *Numba*.
  + - 1. Funcionamiento de *cuDF*

En la sección anterior se mencionó acerca de funciones de procesamiento de datos, estas pueden operar en respectivas columnas, dentro de esta sección se hablará de las funciones que tiene *cuDF* como también el funcionamiento de esta biblioteca.

Las funciones de procesamiento de datos cubren un rango, desde simples primitivas de manipulación de transformación de datos tales como concatenación, funciones aritméticas y clasificación, hasta respectivas funciones de análisis de datos que son de nivel superior como unir, agrupar y agregar.

*cuDF* contiene otras funciones que permiten muchas operaciones adicionales con los usuarios que están familiarizados con *Pandas*, *Numpy* o *SQL.* La biblioteca *cuDF* contiene funciones de ingreso de datos, estos permiten a los usuarios leer archivos *csv, json, txt,* entre otros o *Apache Parquet* (*Apache ORC* próximamente) directamente en un *GDF* (Aramburo, 2019).

* + 1. Biblioteca *cuML*

*cuML* es un conjunto de bibliotecas que implementan algoritmos de aprendizaje automático (ver tabla 2) aceleradas por *GPU* y funciones primitivas matemáticas que comparten *API* compatibles con otros proyectos de *RAPIDS*. *cuML* permite a los científicos de datos, investigadores e ingenieros de software ejecutar tareas tradicionales de *ML* en *GPU*. *cuML* presenta operaciones con múltiples *GPU*, múltiples nodos y también una lista creciente de algoritmos.

* + - 1. Algoritmos soportados por *cuML*

Tabla 2. Algoritmos cuML.

|  |  |  |
| --- | --- | --- |
| Categoría | Algoritmo | Notas |
| *Clustering* | *Density-Based Spatial Clusterind of Applications with Noise (DBSCAN).* |  |
| *K-means.* | Multi-nodo multi-GPU a través de Dask |
| *Dimensionality Reduction* | *Principal Componentes Analysis (PCA).* | Multi-nodo multi-GPU a través de Dask |
| *Truncated Singular Value Decomposition (tSVD).* | 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 conda CUDA 10. |
| *Linear Regression with Lasso or Ridge Regularization.* |  |
| *ElasticNet Regression* |  |
| *Logistic Regression.* |  |
| *Stochastic Gradient Descent (SGD), Coordinate Descent (CD), and Quasi-Newton (QN) (including L-BFGS and OWL-QN) solvers for linear models.* |  |
| *Nonlinear Models for Regresion or Classification* | *Random Forest (RF) Classification.* | 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 versión 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 versión 0.12 y paquetes conda nocturnos. Utiliza Faiss para la consulta de vecinos más cercanos. |
| *Support Vector Machine Classifier (SVC).* |  |
| *Time Series* | *Support Vector Machine Classifier (SVC).* |  |
| *Linear Kalman Filter.* |  |
| *Holt-Winters Exponential Smoothing.* |  |
| *Auto-regressive Integrated Moving Average (ARIMA).* |  |

*Fuente*: (RAPIDS Development Team, 2018).

*Elaboración*: Autor.

La tabla 2 representa los principales conjuntos de algoritmos que tiene *cuML*, están agrupados por categoría y dentro de cada 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 trabaja juntamente con RAPIDS.

A continuación, se hablará de manera concisa y muy explícita de que trata algunos algoritmos de *cuML,* características y funcionalidades:

* *Linear Regression:* Se usa en tareas de regresión donde uno quiere predecir, por ejemplo, ventas o precios de la vivienda, además, se usa en tareas de extrapolación o series de tiempo, modelado de sistemas dinámicos y muchas otras tareas de aprendizaje automático.
* *Ridge Regression:* Se usa igualmente como *Linear Regression*, se usa con más frecuencia ya que no sufre problemas de multicolinealidad, se utiliza en la predicción de primas de seguros, análisis de bolsa y mucho más.
* *Stochastic Gradient Descent:* Se enfoca en optimizar alguna función de costo mediante pasos de gradiente, esto hace que *SGD* sea muy atractivo para grandes problemas difíciles o incluso imposible de encontrar.
* *Nearest Neighbors:* Incluyen sistemas de recomendación donde se utiliza el contenido o el filtrado colaborativo, dado que *Nearest Neighbors* es un modelo generativo relativamente simple, se usa en tareas de visualización de datos y regresión o clasificación.
* *K-Means Clustering:* La ventaja de *K-Means* es su velocidad y simplicidad esto hace que sea la primera opción de muchos profesionales de un algoritmo de agrupamiento, cuando se conoce aproximadamente la cantidad de clústeres, como las tareas de agrupación de datos grandes, la segmentación de imágenes y la agrupación médica.
* *DBSCAN:* Puede encontrar grupos de forma no lineal, permite que sea resistente al ruido, *DBSCAN* se aplicado para el análisis de “*particle collisons in the Large Hadron Collider”*, la segmentación de clientes en análisis de marketing y mucho más.
* *Principal Component Analysis:* Se usa en la práctica para la visualización y compresión de datos, además, visualiza incrustaciones de palabras extremadamente grandes como *Word2Vec* y *GloVe* en 2 o 3 dimensiones y se ha utilizado para distinguir células cancerosas de células sanas.
* *Truncated SVD:* Se utiliza en tareas de recuperación de información, sistemas de recomendación y comprensión de datos.

Se planean más algoritmos de *ML* en *cuML* y más primitivas de *ML* para futuras versiones que incluyen:

* *Spectral* *embedding* (Inclusión espectral).
* *Sprectral clustering* (Agrupación espectral).
* *Support vector machines* (Máquinas de vectores de soporte).
* *Additional time series methods* (Métodos de series de tiempo adicionales).
  + 1. 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 llamar fácilmente a algoritmos de gráficos utilizando los datos almacenados en un marco de datos de *GPU*.

*cuGraph* hace 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.

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 *NetworkX:* es una biblioteca que se enfoca en estudiar gráficos y redes, para 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* en el apartado 2.3.7 nos enfocaremos en *DASK-cuGraph*; por medio de *cuGraph* distribuirá de manera inteligente la carga de trabajo entre los recursos disponibles (RAPIDS Development Team, 2018).

* + - 1. Algoritmos soportados por *cuGraph*

Tabla 3: Algoritmos soportados por *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 versión BSP |
| *Traingle Counting* | *Single-GPU* |  |
| *K-Cores* | *Single-GPU* |  |
| *Core Number* | *Single-GPU* |  |
| *Subgraph Extraction* | *Single-GPU* |  |
| *Spectral Clustering-Balanced-Cut* | *Single-GPU* |  |
| *Spectral Clustering-Modularity Maximization* | *Single-GPU* |  |
| *Louvain* | *Single-GPU* |  |
| *Renumbering* | *Single-GPU* |  |
| *Basic Graph Statistics* | *Single-GPU* |  |
| *Weakly Connected Components* | *Single-GPU* |  |
| *Strongly Connected Components* | *Single-GPU* |  |

*Fuente*: (RAPIDS Development Team, 2018).

*Elaboración*: Autor

La tabla 3 representa todos los principales conjuntos de algoritmos que contiene *cuGraph*, cada algoritmo obtiene una escala en la cual es *Multi-GPU* o *Single-GPU.* Todos los algoritmos demostrados en la tabla sirven para poder analizar distintas gráficas.

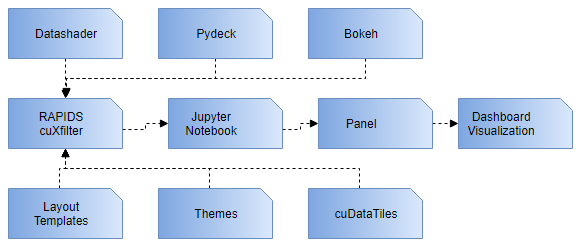
A continuación, se explica de manera concisa y explícita de que trata algunos algoritmos de *cuGraph.* Estos algoritmos están diseñados para ejecutarse en una sola GPU con conjuntos de datos de alrededor de 500 millones de bordes o menos:

* *Jaccard Similarity:* Mide la similitud de vecindario entre vértices conectados; dentro de los sistemas de recomendaciones, esto es muy útil para encontrar clientes con un comportamiento similar o igual (Rees, 2019).
* *Weighted Jaccard:* Es similar a Jaccard, con la excepción que el algoritmo suma los pesos de los vértices.
* *Page Rank:* Utilizada por motores de búsqueda, sin embargo, tiene aplicaciones en análisis de redes sociales, sistemas de recomendación y para nuevos usos en ciencias naturales al estudiar la relación entre proteínas y en redes ecológicas (Rees, 2019).
* *Single Source Shortest Path (SSSP):* Se enfoca en identificar la ruta más contra entre un par de vértices; por ejemplo, dentro de una red de carreteras se puede utilizar para encontrar la ruta más rápida de A a B, además, SSSP se puede utilizar para optimizar una amplia gama de problemas logísticos (Rees, 2019).
* *Breadth-First Search (BFS):* Es un algoritmo de búsqueda clásico que explora iterativamente el gráfico, comenzado en un punto inicial, el algoritmo da un salto por iteración.
* *Spectral Clustering:* Agrupa vértices en función de sus características, de modo que haya una gran similitud intragrupo y una baja similitud entre grupos. *Spectral Clustering* construye una matriz, resuelve un problema de valor propio asociado y extrae la información de división de los vectores propios calculados (Rees, 2019).
* *Louvain Clustering:* Usa la modalidad como la métrica para combinar iterativamente vértices en grupos, comienza con cada vértice en su propio grupo y funciona iterativamente grupos basados en el modularidad (Rees, 2019).
  + 1. Biblioteca *cuXfilter*

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

*cuXfilter*.*py* resuelve los problemas al aprovechar el poder de la pila rapids.ai, principalmente *cuDF*, los datos se mantienen en una *GPU* como un marco de datos de *GPU* y las operaciones como agregaciones grupales, clasificación y consulta se realizan en la propia *GPU*, solo devolviendo el resultado como salida a los gráficos.

*cuXfilter*.*py* 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.



*Figura 23.* Arquitectura *cuXfilter*.*py*

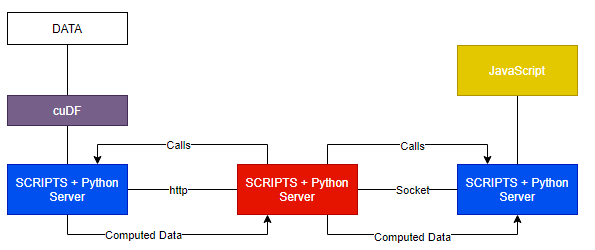
*Fuente*: (RAPIDS Development Team, 2018).

*Elaboración*: Autor.

La figura 23 representa la arquitectura de *cuXfilter*.*py*, aprovecha el portátil de *jupyter* y el servidor *bokeh* para reducir en gran medida la complejidad del *backend*.

* + - 1. Arquitectura *cuXfilter*

En la figura 24 representa la arquitectura de *cuXfilter*, necesita 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.*



*Figura 24.* Arquitectura general *cuXfilter*.

*Fuente:* (Enemark, 2018)*.*

*Elaboración*: Autor.

* + 1. *DASK*

Los científicos de datos utilizan a menudo herramientas como *Pandas, Scikit-Learn, Numpy* y el resto del ecosistema *Python* para analizar datos en su computadora personal, sin embargo, cuando elijen analizar un conjunto de datos más grandes, descubren que esas herramientas no fueron diseñadas para escalar más allá de una sola máquina.

*DASK* proporciona formas de escalar los flujos de trabajo de Pandas, Scikit-*Learn*, *Numpy* de forma más nativa, con una reescritura mínima, descubre como dividir grandes cálculos y enrrutar 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 (Team, 2016).

*DASK* es un programador de cómputo distribuido, es extremadamente modular con la programación, transferencia de datos y manejo fuera del núcleo, todo desunido, lo que permite conectar nuestras propias implementaciones (Patterson, 2019).

* + - 1. *DASK* a Computadoras Portátiles

Las computadoras portátiles y las estaciones de trabajo de hoy en día son sorprendentemente útiles y se usan correctamente, además, pueden manejar conjuntos de datos y cálculos para los que anteriormente se dependía de los clústeres.

*DASK* puede capacitar a los científicos de datos y analistas para manipular conjuntos de datos de 100GB en la computadora portátil o conjuntos de datos de 1TB sin molestarse en absoluto con un clúster.sz, cálculos paralelos eficientes en máquinas individuales al aprovechar sus *CPU* de múltiples núcleos y transmitir datos de manera eficiente desde el disco, *DASK* cambia el clúster por planificadores de una sola máquina que son livianos, no requieren configuración y pueden ejecutarse completamente dentro del mismo proceso de sesión del usuario (Team, 2016).

* + - 1. *DASK*-*RAPIDS*

*RAPIDS* utiliza *DASK* para escalar cargas de trabajo de *Python* desde computadores portátiles a clústeres de supercomputadoras.

Puede ejecutar fácilmente varios trabajadores *DASK* por nodo para permitir un modelo de desarrollo más sencillo de un trabajador por *GPU*, independientemente del entorno único o de múltiples nodos.

***DASK-cuDF:***DASK 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 los de hardware utilizando un marco de comunicaciones llamada *OpenUCX* (Patterson, 2019)*.*

***DASK-cuML:*** Una integración nativa con *DASK-cuDF*, puede usar fácilmente a los trabajadores de DASK para inicializar avances *NCCL* para operaciones de recolección o dispersión optimizadas, además, proporciona primitivas fáciles de usar y de alto nivel para la sincronización de trabajadores, es necesario para muchos algoritmos de *ML* (Patterson, 2019)*.*

***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.

* + 1. Integración con Bibliotecas de *Deep Learning*

*RAPIDS* se integra a la perfección con bibliotecas de aprendizaje profundo, para la computación acelerada, en este aparatado nos enfocaremos en algunas librerías que tiene integración con *RAPIDS*.

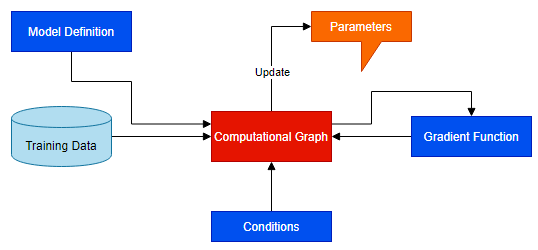
* + - 1. *Chainer*

*Chainer* es un *framework* de *Python*, permite principalmente a los investigadores, entrenar y evaluar rápidamente modelos de aprendizaje profundo.

*Chainer* es un marco de código abierto diseñado principalmente para la investigación eficiente y el desarrollo de *Deep Learning*. El enfoque de *Chainer* es único: construir el grafico computacional “sobre la marcha” durante el entrenamiento. Permite a los usuarios cambiar el gráfico en cada iteración o por cada muestra, según las condiciones.

Chainer proporciona formas imperativas de declarar redes neuronales para admitir operaciones compatibles con *Numpy* entre matrices y también incluye computación numérica basada en *GPU* (Shohei, 2016)*.*

Para entrenar una red neuronal en *Chainer* se necesitan tres pasos muy importantes: construir un gráfico computacional a partir de la definición de red, ingresar datos de entrenamiento, calcular la función perdida y por último actualizar los parámetros usando optimizador y repetir hasta la convergencia (Shohei, 2016).



*Figura 25.* Proceso de *Chainer Define-by-Run*.

*Fuente:* (Shohei, 2016).

*Elaboración:* Autor

La figura 25 representa cómo funciona la biblioteca *Chainer* para *Deep* *Learning*, el gráfico computacional no se proporciona antes del entrenamiento, sino que se obtiene en el curso del entrenamiento. El cálculo directo corresponde al gráfico computacional y la propagación hacia atrás de él, cualquier modificación en el respectivo gráfico computacional se hará el cálculo directo en cada interacción e incluso para cada muestra (Shohei, 2016).

* + - * 1. Características *Chainer*

Está conformado por tres puntos principales, que permite que este *framework* sea utilizado por bastantes científicos e investigadores, los siguientes son:

**Poderoso**

Tabla 4: *Fast Chainer.*

|  |  |  |
| --- | --- | --- |
| **Fast** | *CUDA* | *Supports GPU acceleration using CUDA wint CuPy* |
| *CuDNN* | *High-speed traininginference with cuDNN optimixzed Deep Learning functions with CuPy.* |
| *NCCL* | *Supports a fast, multi-GPU Learning using NCCL with CuPy.* |

*Fuente*: (Chainer, 2019).

*Elaboración*: Autor

En la tabla 4 admite computo *CUDA*; requiere de unas pocas líneas de código para aprovechar una *GPU*, también se ejecuta en múltiples *GPU* con poco esfuerzo (Chainer, 2019).

**Intuitivo**

Tabla 6: Intuitivo Chainer.

|  |  |  |
| --- | --- | --- |
| ***Intuitive*** | *Define-by-Run* | *Easy and intuitive to Write a network. Supports Dynamic graphs.* |
| *High debuggability* | *User-friendly error messages. Easy to debug using pure Python debuggers.* |
| *Easy to use APIs* | *Well-abstracted common tools for varios NN Learning, easy to write set of Learning flows.* |
| *Low Learning curve* | *No needs to learn a new tensor API since Chainer uses Numpy and CuPy.* |
| *Maintainable codebase* | *Written in pure and well documented.* |

*Fuente*: (Chainer, 2019).

*Elaboración*: Autor

En la tabla 6 incluye cualquier declaración de flujo de control de *Python* sin carecer de la capacidad de propagación hacia atrás, además, el código es intuitivo y fácil de depurar (Chainer, 2019).

* + - 1. *MXNet*

*MXNet* surgió de una colaboración de los autores de *cxxnet, minerva y purine2; MXNet* combina aspectos de cada uno de estos proyectos para lograr la flexibilidad, velocidad y eficiencia de memoria (MXNET, 2019).

*MXNet* es una librería de aprendizaje profundo, verdaderamente de código abierto adecuado para la producción y creación de prototipos de investigación flexible.

*MXNet* es un marco diseñado para la eficiencia y la flexibilidad, permite mezclar programación simbólica e imperativa para poder maximizar la eficiencia y la productividad, contiene un programador de dependencia dinámico que paraleliza automática las operaciones simbólicas e imperativas sobre la marcha; *MXNet* hace que la ejecución sea simbólica, rápida y eficiente en la memoria. *MXNet* es portátil y liviano, escalable de manera efectiva a múltiples *GPU* y múltiples máquinas (MXNET, 2019).

* + - * 1. Características *MXNet*

**Frontal Híbrido:** Un *front*-*end* hibrido pasa a la perfección entre el modo imperativo ansioso de *Gluon* y el modo simbólico para proporcionar flexibilidad y velocidad (MXNET, 2019).

**Entrenamiento distribuido:** La capacitación es distribuida escalable y la optimización del rendimiento en investigación y producción están habilitadas por el servidor dual de parámetros y el soporte de *Horovod* es un entorno de entrenamiento distribuido popular y de código abierto para escalar entrenamiento con *TesorFlow* en múltiples *GPU* (MXNET, 2019).

***Lenguage* *Bindings:*** Integraciónprofunda con el lenguaje *Python*, tiene soporte para otros lenguajes como *Scala, Julia, Clojure, Java, C++, R y Perl* (MXNET, 2019)*.*

**Herramientas y Librerías:** Consta de un próspero ecosistema de herramientas y librerías, permite casos de usos de visión por computadora, *PNL*, series de tiempo y más (MXNET, 2019).

* + - 1. *PyTorch*

*Pytorch* está diseñado para ser intuitivo, lineal en pensamiento y fácil de usar por los programadores, cuando se ejecuta una línea de código, está se ejecuta, no existe visión asincrónica del mundo. Cuando se ingresa un depurador o recibe mensaje de error y rastrea fallos, estos son fáciles de comprenderlos y son sencillos (PyTorch, 2019).

Tabla 7. Componentes *Pytorch*.

|  |  |
| --- | --- |
| Componente | Descripción |
| *Torch* | Una biblioteca Tensor con *Numpy*, con un fuerte soporte de GPU. |
| *Torch.autograd* | Una librería de diferenciación automática basada en cinta que admite todas las operaciones de Tensor diferenciables en *Torch*. |
| *Torch.jit* | Una pila de compilación (*TorchScript*) para crear modelos con *Autogrand* diseñada para la máxima flexibilidad. |
| *Torch.nn* | Una librería de redes neuronales profundamente integrada con Autograd diseñada para la máxima flexibilidad. |
| *Torch.multiprocessing* | Multiprocesamiento de Python, pero con un intercambio mágico de memoria de los tensores de Torch en los procesos. |
| *Torch.utils* | *DataLoader* y otras funciones de utilidad para mayor comodidad. |

Fuente: (PyTorch, 2019).

Elaboración: Autor.

En la tabla 7 *Pytorch* está conformado por algunos componentes principales y su descripción de cada componente, ayudan que el trabajo sea más eficiente para los procesos de *Deep Learning.*

Pytorch netamente lo usan porque es un remplazo de *NumPy* para poder usar el poder de las *GPU* y también como una plataforma de investigación de aprendizaje profundo que proporciona la máxima flexibilidad y velocidad.

*Pytorch* proporciona una amplia variedad de rutinas tensoras para acelerar y ajustar sus necesidades de computación científica, como segmentación, indexación, operaciones matemáticas, algebra lineal y reducciones (PyTorch, 2019).

* + - * 1. Característica *PyTorch*

***TorchScript:*** Proporciona una transición perfecta entre el modo ansioso y el modo gráfico para acelerar el camino hacia la respectiva producción (PyTorch, 2019).

**Entrenamiento distribuido:** Capacitación distribuida escalable y la optimización del rendimiento en investigación y producción habilitadas por el *backend* distribuido (PyTorch, 2019).

**Herramientas y librerías:** Ecosistema de herramientas y librerías Pytorch, admite el desarrollo en visión por computadora (PyTorch, 2019).

**Socios en la nube: S**oportado en las principales plataformas en la nube, proporciona un desarrollo sin fricción y escalado fácil (PyTorch, 2019).

* + - 1. *Numba*

El compilador *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 (Crist, 2016).

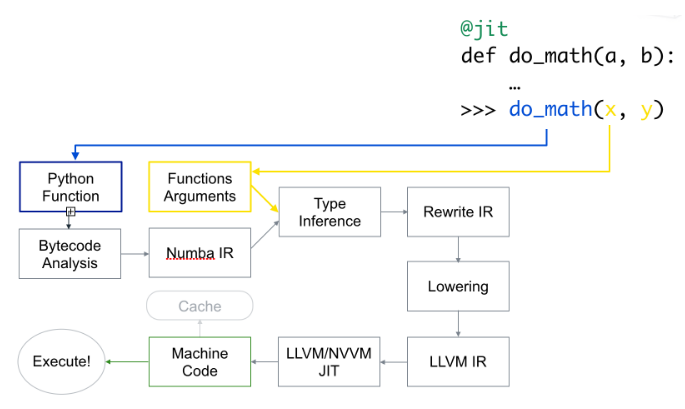
La forma más común de utilizar *Numba* es por medio de una colección de decoradores que se pueden aplicar a sus respectivas funciones para indicarle a *Numba* que las compile. Cuando se realiza una respectiva llamada a una función propia de *Numba*, esta compila el código de maquina *“just-in-time”* para su ejecución y todo o parte de su código puede ejecutarse posteriormente a la velocidad del código de maquina nativo (Numba, 2019).

* + - * 1. *Python-Numba*

*Numba* hace que el código de *Python* sea rápido, realiza una llamada a un compilador de código abierto, se traduce un subconjunto de código *Python y Numpy* en código rápido de máquina (ver figura 26).

Numba puede acelerar todas las funciones escritas en *Python* centradas principalmente en cálculo y pesadas computacionales, tiene soporte para biblioteca *Numpy*, por lo tanto, realiza cálculos y acelera el cálculo general ya que los bucles en Python son muy lentos.

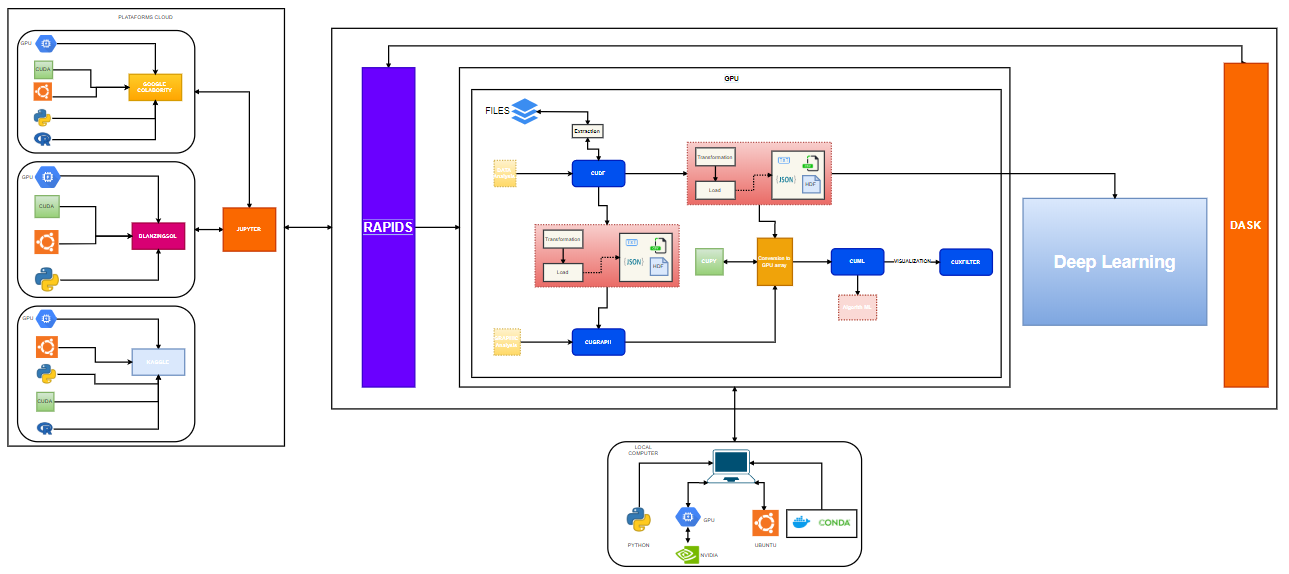
En la figura 26 se demuestra el proceso de trabajo de *Python-Numpy*, la función *Python* se toma, optimiza y se convierte en la representación intermedia de *Numba*, después de la inferencia de tipo, que es como la inferencia de tipo *Numpy*, se convierte en código interpretable LLVM, este código se envía al compilador justo a tiempo de LLVM para entregar el código de la máquina (Puneet, 2018).



*Figura 26.* Proceso de Trabajo *Python-Numpy.*

*Fuente:* (Puneet, 2018)*.*

* + - * 1. Características *Numba*
* Generación de código sobre la marcha (tiempo de importación o tiempo de ejecución, según las preferencias del usuario).
* Generación de código nativo para el hardware de la *CPU* (predeterminado) y *GPU*.
* Integración con la pila de software científico de *Python*.
  + 1. Análisis de la Arquitectura y Características de *RAPIDS*



*Figura 27.* Arquitectura RAPIDS.

*Elaboración*: Autor.

La figura 27 representa el análisis extenso e implementación de la arquitectura propuesta para *RAPIDS*, se explica de manera dinámica, concisa y sólida, a través de los siguientes puntos:

1. *RAPIDS* trabaja de forma correcta, eficiente y eficaz, necesita estar en un ambiente dedicado a Ciencia de Datos, para construir un ambiente de tal magnitud se debe realiza un proceso respectivo:
   1. RAPIDS permite leer y analizar archivos tipo *csv, json, txt, hdf,* entre otros.
   2. El conjunto de bibliotecas principales de RAPIDS permitirá llevar a cabo un flujo de trabajo, *ETL* (Extracción, transformación y carga), este proceso lo realiza la biblioteca cuDF para el procesamiento y análisis de los datos. Además, RAPIDS permite realizar el análisis de gráficos, con la biblioteca cuGraph, está biblioteca trabaja juntamente con cuDF y cuML.
   3. Una vez realizado el proceso de ETL, los datos se encuentran totalmente limpios, cuando existen datos categóricos se debe realizar un preprocesamiento a estos, hay que convertirlos o transformarlos a datos numéricos, la transformación se realiza por medio de la librería *cupy (*trabaja conjuntamente con *RAPIDS)*, se encarga de transformar los valores categóricos a numéricos y los guarda como array ambientado a GPU.
   4. ML proceso de entrenamiento y prueba lo realiza la biblioteca cuML a los datos previamente seleccionados y analizados por cuDF y con el respectivo preprocesamiento por *cupy*.
   5. Visualización, cuXfilter para la respectiva representación de resultados en forma de gráficos estadísticos y filtrados de datos.
2. *RAPIDS* trabaja en GPU, permite que el procesamiento de los datos sea más ágil, reduce tiempos de ETL, ML y para la representación de resultados trabaja de forma dinámica.
3. *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 algoritmos, para trabajar con este tipo de bibliotecas debe pasar por el proceso de cuDF para la extracción, transformación y carga de los datos.
4. *RAPIDS* para mejorar su rendimiento y tiempos de esfuerzo, tiene una fusión con *DASK*, particularmente si estas analizado y procesando bastante información, *DASK* con *RAPIDS* permiten acelerar las cargas de trabajo de *Python* desde tu computadora portátil y ejecutar fácilmente varios trabajadores.
5. *RAPIDS* está basado en *Apache* *Arrow*, cabe recalcar que RAPIDS se ha basado en este “*development plataforma”*, permite minimizar las conversaciones de datos y las serializaciones de datos cuando una tubería de procesamiento de datos incluye diferentes marcos de cómputo.
6. RAPIDS debe trabajar en el lenguaje de programación de *Python*, algunos puntos precisos del porque trabajar con este lenguaje de programación:
   1. 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 (Aguerzame et al., 2019).
   2. Python reúne características necesarias para la ciencia de datos, permite simplificar muchas cosas como también mejorar el código sea legible a través de la sintaxis.
   3. 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 dedica *RAPIDS*.
   4. Python permite equipar a los científicos de datos para implementar soluciones factibles, al mismo tiempo sigue los estándares de los algoritmos requeridos del proceso de análisis, desarrollo y solución.
7. RAPIDS trabaja en dos ambientes diferentes, con plataformas en la nube (*cloud*) y en un computador personal (localmente). Como plataformas en la nube tenemos las siguientes:
   1. RAPIDS con *Google* *Colabority.*
   2. RAPIDS con *BLAZINGSQL.*
   3. RAPIDS con *Kaggle*.

Las plataformas mencionadas ya cuentan con una GPU, con sistema operativo Ubuntu, lenguaje de programación Python y los drivers de CUDA.

En el capitulo 4 abarcaremos la instalación de RAPIDS en las plataformas en la nube, se detallará el proceso de funcionamiento y entornos de configuración en *Google* *Colabority* y *BLAZINGSQL.* Además, el proceso de instalación en computador personal (localmente), también se detallará en este capítulo.

**CAPÍTULO 3 TRABAJOS RELACIONADOS**

1. Trabajos Relacionados
   1. Fuentes de Información

* *Google Scholar*

Buscador que permite a los investigadores, científicos, etc. Localizar varios documentos académicos como artículos, trabajos de titulación, libros, etc.

* *RAPIDS*

Fuente de información que permite a los científicos de datos, estudiantes y desarrolladores a localizar varios documentos con respecto a la Librería *Open Source RAPIDS*.

* 1. Cadenas de Búsqueda

|  |  |  |
| --- | --- | --- |
| **Bases de Datos** | **Cadena de Búsqueda** | **Trabajo** |
| Scholar | “GPU” +” using NVDIA RAPIDS” +” Computing Acelerated” | Accelerated Simulation of Air Pollution Using NVIDIA RAPIDS. |
| Scholar | “GPU” +” using NVDIA RAPIDS” +” Computing Acelerated” | GPU Acceleration of PySpark using RAPIDS AI. |
| Scholar | “GPU” +” using NVDIA RAPIDS” +” Computing Acelerated” | Accelerating recommender system training 15x with RAPIDS. |
| RAPIDS | Ninguna | Security Alert Analysis Using GPUs. |
| RAPIDS | Ninguna | Accelerating Random Forest up to 45x using cuML. |
| RAPIDS | Ninguna | Combining Speed & Scale to Accelerate K-Means in RAPIDS cuML |

* 1. Criterios de inclusión y exclusión

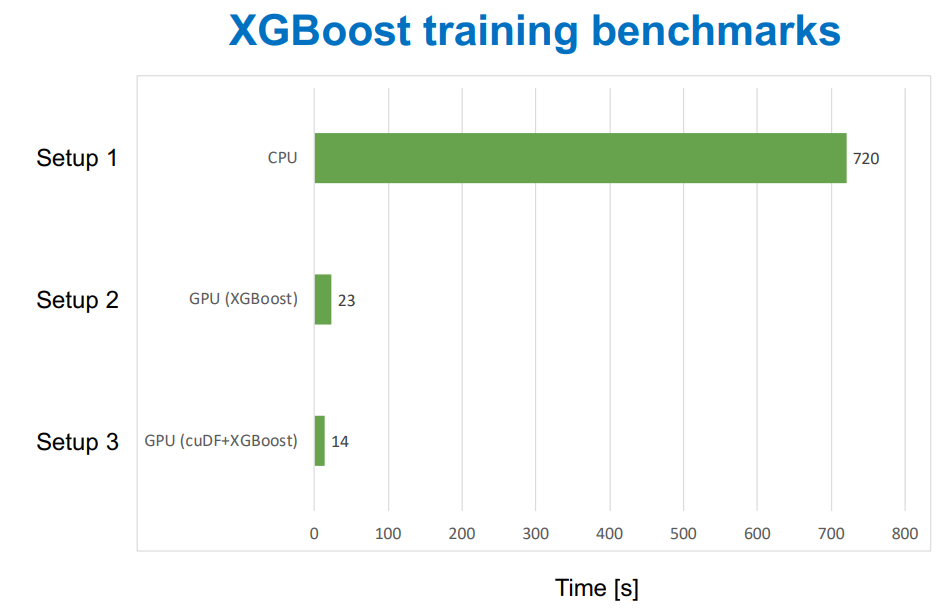
|  |  |
| --- | --- |
| **Criterio de Búsqueda** | **Parámetro** |
| Rango de Fecha | 2019 al presente |
| Tipo de Documento | Full papers, books, article |
| Tipo de Acceso | Todos |
| Idioma | Inglés |

* 1. Análisis de Trabajos Relacionados

***Accelerated Simulation of Air Pollution Using NVIDIA RAPIDS***

La simulación presenta un enfoque alternativo para el cálculo de la química atmosférica basado en el aprendizaje automático. Utilizan un conjunto de datos de entrenamiento, se produce utilizando el modelo del Sistema de Observación de la Tierra *Goddard* de la *NASA* (*GEOS*) con química *GEOS-Chem,* se ejecuta en el centro de simulación climática de la NASA para descubrir el clúster de supercomputación en 384 núcleos *Intel Xeon Haswell.*

El conjunto de datos contiene las concentraciones de contaminación del aire, parámetros físicos como la temperatura y la intensidad del sol. La aplicación usara *Dask-cuDF* y *Dask-XGBoost* en la plataforma de *NVDIA RAPIDS* en 8 *GPU Tesla V100* (ver figura 28), para el entrenamiento de los datos utilizan modelos de árboles de decisión impulsados por gradiente que pueden reproducir la simulación de la cinética química, además la aplicación aprovecha al máximo los recientes avances en *Dask-XGBoost*, como el escalado de múltiples nodos y múltiples *GPU* para la distribución de grandes datos. Muestran los beneficios de este enfoque y se discute la posible aceleración de este modelo de química atmosférica acelerada de aprendizaje automático. Incorporan modelos de árboles potenciados en el modelo de referencia *GEOS* utilizando la *API C* de *XGBoost*, lo que permite una integración perfecta para los modelos entrenados por GPU en *GEOS*-*Chem* (Keller et al., 2019).

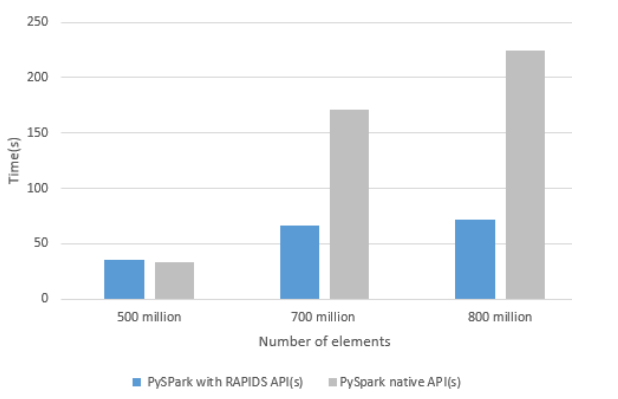


*Figura 28.* Evaluación comparativa de XGBoost con CPU, GPU y XGBoost cuDF + GPU.

*Fuente*: (Keller et al., 2019).

***GPU Acceleration of PySpark using RAPIDS AI.***

La investigación e implementación se centra en aprovechar muchos núcleos y una gran plataforma de memoria, con un enfoque de escalamiento en mente, el enfoque es implementar en algunos servidores grandes *BullSequana* con muchos núcleos y memoria grande, además, está dedicado a exponer la solución que aprovecha las *GPU* en el flujo de trabajo de *PySpark* utilizando la biblioteca *RAPIDS* (ver figura 29*).* Utilizan *Docker* acoplable que va a contener todas las bibliotecas necesarias, también la etiqueta de imagen “*cuda9.2-runtime-ubuntu16.04*” y un portátil *Jupyter* y le agregan *Spark*, el servidor consta con 8x12 núcleos y memoria *RAM* de 4 *terabytes*. El proceso de verificación para asegurar la conexión entre *RAPIDS* y *Spark* sea posible y delegar tareas de *Spark* en la *GPU*, abre nuevas posibilidades para que *Apache* *Spark* sea capaz de aprovechar servidores de gama alta como *BullSequana* acoplados a *GPU* (Aguerzame et al., 2019).



*Figura 29.* *PySpark* + *RAPIDS* y *PySpark* nativo.

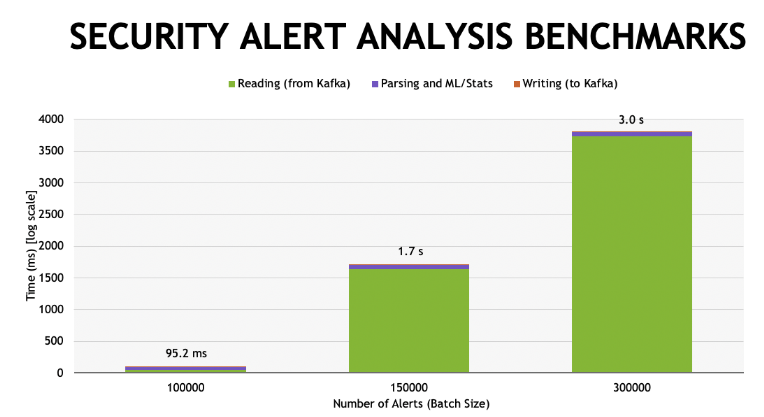
*Fuente*: (Aguerzame et al., 2019).

***Accelerating recommender system training 15x with RAPIDS*.**

El documento proporciona una aceleración de la generación de funciones y el tiempo de entrenamiento del modelo, aceleran la capacitación del modelo en un factor de 15.6x, desde un flujo de trabajo de 891.8s a 57.2s, a través de una combinación de la biblioteca *RAPIDS*, *cuDF* para preprocesamiento, un cargador de datos por lotes, *LAMB* y tamaños de lotes extremos y una respectiva actualización del kernel responsable de calcular el gradiente de incrustación en *PyTorch*. Usando *cuDF* aceleran a un factor de 9.7x al realizar cálculos en la *GPU*, reduciendo el tiempo necesario para generar características sumamente primordiales de 51 min a 5 min (Rabhi et al., 2019).

***Security Alert Analysis Using GPUs.***

El documento muestra cómo se puede procesar más de 300.000 alertas sin procesar en menos de 3s con un solo *Tesla V100*. Además, permitir que los equipos de seguridad ingieran y analicen más tipos de registros y las cantidades de registros aumenta la visibilidad en toda la red y proporciona meta-alertas complejas en tiempo real (ver figura 30). Al utilizar la aceleración de *GPU*, se puede proporcionar simultáneamente análisis en tiempo real de estas alertas cada vez mayores. Las alertas se agregan por tipo y por día, las normalizar por el número total de alertas por tipo y la traza como un mapa de calor para que los analistas puedan ver rápidamente las tendencias en sus alertas y compararlas entre sí. Los componentes del flujo de trabajo del análisis de alertas son tanto el análisis del registro de alertas como el flujo de trabajo analítico. *CLX* tiene un módulo útil que analizar formatos de registros comunes usando *RAPIDS*. *Regex* para el análisis de alertas se almacena en un archivo *yaml* y el analizador notable *Splunk* lee ese archivo y ejecuta la funcionalidad de análisis. Debido a que *CLX* se construye utilizando *RAPIDS*, se puede analizar una gran cantidad de alertas muy rápidamente (Allen & US, 2020).



*Figura 30.* Análisis comparativo de alertas con una sola *GPU*.

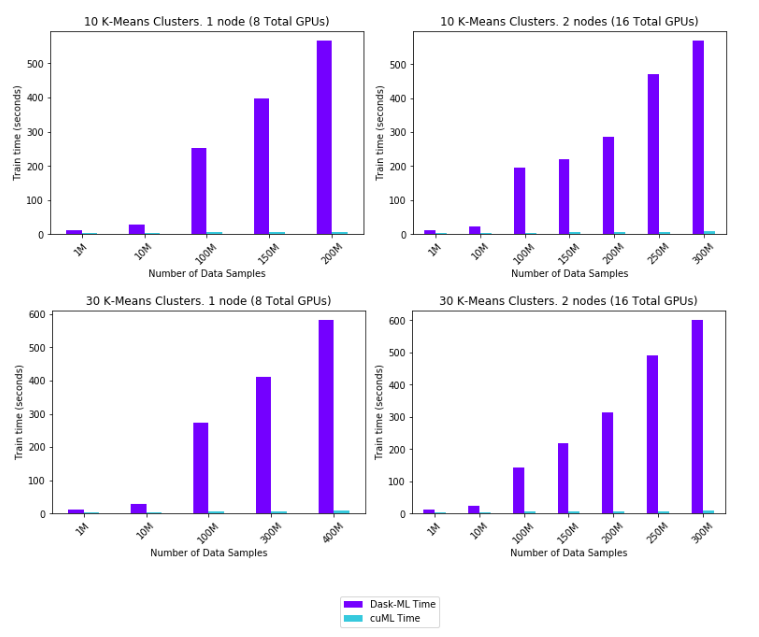
*Fuente*: (Allen & US, 2020).

***Accelerating Random Forest up to 45x using cuML.***

El documento proporciona una revisión de algoritmos básicos de *Random* *Forest*, como su entrenamiento puede ser paralelo a las *GPU* *NVIDIA* y números de referencia que demuestran el rendimiento. Cada árbol de decisión se entrena sobre una muestra diferente con reemplazo del conjunto de datos original. La biblioteca *Random* *Forest* de *cuML* contiene dos algoritmos divididos de alto rendimiento para seleccionar que valores exploran para cada combinación de característica más nodo: histogramas mínimos / máximo y cuántiles, el algoritmo *cuML* puede construir varios árboles en paralelo en una sola *GPU*, cada árbol está construido en su propio flujo *CUDA* controlado por un hilo *OpenMP* en la *CPU*. Para garantizar el mejor rendimiento, utiliza un marco de datos de *GPU* como entrada para *cuML* y una matriz *numpy* como entrada para *sklearn*. Además, para comprender completamente el mejor rendimiento que se puede obtener del entrenamiento forestal aleatorio utilizando *GPU*, se amplía la prueba a las ejecuciones de múltiples *GPU* utilizando el enfoque distribuido basado en *Dask* usando un conjunto de datos de 8.8M filas para entrenar y 1000 filas para probar, hardware de servidor *DGX-1* con ocho *GPU* *V100*-*16GB*, para las pruebas múltiples *GPU* se utiliza 1000 árboles por modelo y una profundidad máxima a 8, 12 o 16 (Vishal, 2019).

***Combining Speed & Scale to Accelerate K-Means in RAPIDS cuML***

*K-Means* de *cuML* proporciona el método inherentemente secuencial, combina muestreo aleatorio junto con distribuciones de distancias de puntos a cada muestra para dispersar mejor los centroides iniciales a través del espacio donde existen puntos reales. Muchos algoritmos de *cuML* se construyen a través de las primitivas de *CUDA*. Los algoritmos de multi-nodo *multi-GPU* se ejecutan dentro del entorno de *Dask*, lo que facilita la carga de conjuntos de datos de gran tamaño en un marco de datos distribuido por *cuDF* y utiliza modelos de aprendizaje automático en *GPU* (ver figura 31). Una vez entrenados los centroides, se mantienen en un solo trabajador Dask, la predicción se realiza de manera paralela trasmitiendo los centroides a los trabajadores, *cuML* puede escalar a un gran número de *GPU* y nodos. Se comparó dos algoritmos de *K-Means-cuML* con *K-Means* de *Dask-cuML*, ambos algoritmos fueron comparados en dos máquinas, utilizando 8 trabajadores por nodo (Nolet, 2019).



*Figura 31.* Evaluación comparativa de múltiples nodos, múltiples *GPUs cuML* y *Dask-CUML*.

*Fuente*: (Nolet, 2019).

**CAPITULO 4 IMPLEMENTACIÓN CON RAPIDS**

1. Implementación
   1. Instalación RAPIDS

Para el enfoque de instalación de RAPIDS se puede instalar de dos maneras muy dinámicas, es necesario explicarlas en este apartado, para el proceso de instalación, además, dar a conocer a los desarrolladores y estudiantes que RAPIDS al momento de instalar va a obtener la misma funcionalidad.

* + 1. Instalación en Máquina Personal

Dentro del proceso de instalación de *RAPIDS* en un computador portátil localmente se necesita ciertos prerrequisitos para ser uso de la librería, se incorpora los principales puntos para la respectiva instalación.

Tabla 8. Prerrequisitos de Instalación *RAPIDS*.

|  |  |  |  |
| --- | --- | --- | --- |
| **GPU Nvidia** | **SO. Ubuntu** | **Docker** | **Cuda and Nvidia Drivers** |
| Titan RTX  Tesla  GeForce | Versión 16.04  Versión 18.04 | Versión 19.03 | Versión 10.0  Versión 10.1  Versión 10.2 |

Fuente: Autor.

Elaboración: Autor.

La tabla 8 representa los prerrequisitos que debe tener un computador personal para instalar la librería *Open Source RAPIDS.*

La máquina personal donde se realizó la respectiva instalación de *RAPIDS* cumple con los prerrequisitos mencionados, se detalla las especificaciones que tiene el mismo.

Tabla 9. Especificaciones computador portátil.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Marca Portátil** | **Intel** | **Sistema Operativo** | **Tipo de Sistema Operativo** | **GPU** |
| DELL | Core I7 8th Gen | Ubuntu 18.04 | De 64 bits | Nvidia GeForce MX150 |

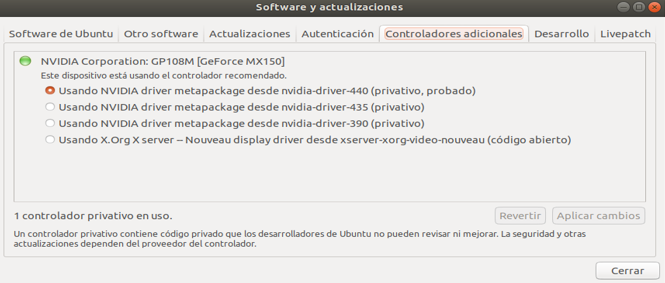
Fuente: Autor.

Elaboración: Autor.

La tabla 9 representa las características que tiene la máquina personal, se detalla los requerimientos que tiene el mismo y donde se procedió a instalar la librería *RAPIDS*.

Cabe destacar para instalar la librería debe tener como base el sistema operativo *Ubuntu* y *GPU de Nvidia*, si no contiene estos dos aspectos la librería no se podrá instalar.

Primeramente, debemos verificar si el sistema operativo tiene instalado y actualizado los controladores o *drivers* de *Nvidia* (ver figura 32*)*, esto lo hacemos para determinar que nuestro computador cuenta con *GPU* y los drivers están activos en toda su totalidad.

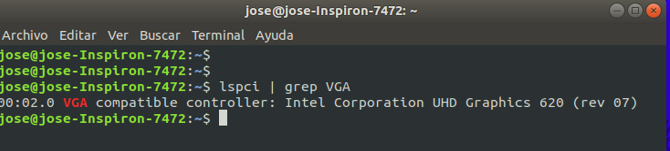


*Figura 32.* *Nvidia* activa.

Elaboración: Autor.

La figura 32 representa, dentro del sistema operativo, se encuentra en la máquina personal de marca *DELL* tiene instalado los controladores o drivers de Nvidia, están activados y funcionando de manera correcta.

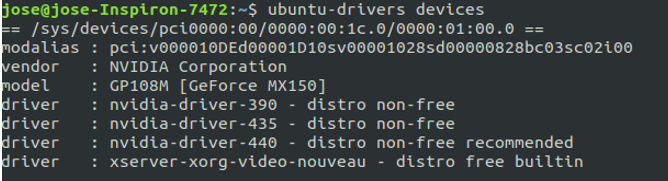
Seguidamente abrimos un terminal, para conocer la información del modelo de la tarjeta gráfica que tiene la máquina.



*Figura 33.* Modelo tarjeta gráfica.

Elaboración: Autor.

Dentro del mismo terminal debemos verificar que modelo y controlador(ver figura 34) está disponible a través de los canales oficiales del sistema operativo *Ubuntu*, según la versión que estemos utilizando y seguidamente se realiza la instalación del controlador(ver figura 35).



*Figura 34.* Drivers disponibles.

*Elaboración*: Autor.

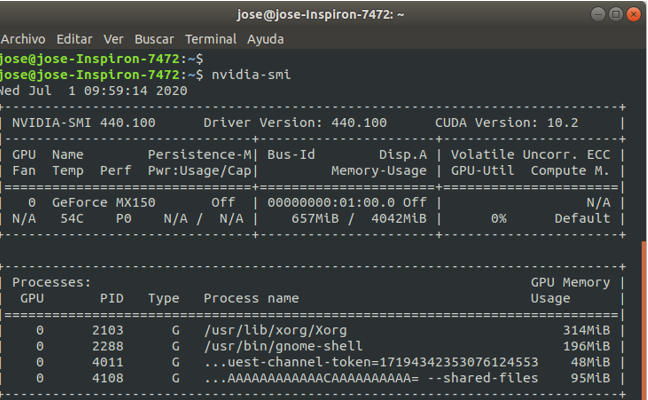


*Figura 35.* Instalarcontrolador*.*

*Elaboración*: Autor.

La figura 34 y 35 corresponde al proceso de verificación de controladordisponible y el comando para instalar el controladorque necesita el sistema operativo *Ubuntu*, con el fin de que pueda funcionar correctamente la *GPU* y no tener problemas futuros con la instalación de la librería *RAPIDS*.

Una vez instalado el controlador, abrir nuevamente otro terminal para verificar si ya se encuentra activa la GPU actualizada y funcionando correctamente.



*Figura 36.* *GPU Nvidia* activa.

*Elaboración*: Autor.

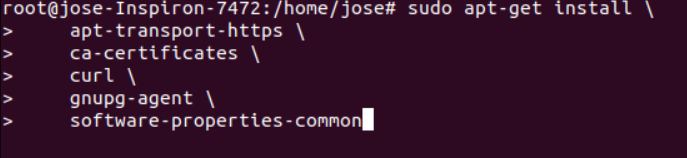
La figura 36 representa la máquina personal marca DELL ya esta constando con la GPU Nvidia en toda su totalidad y funcionando con todas las características de esta.

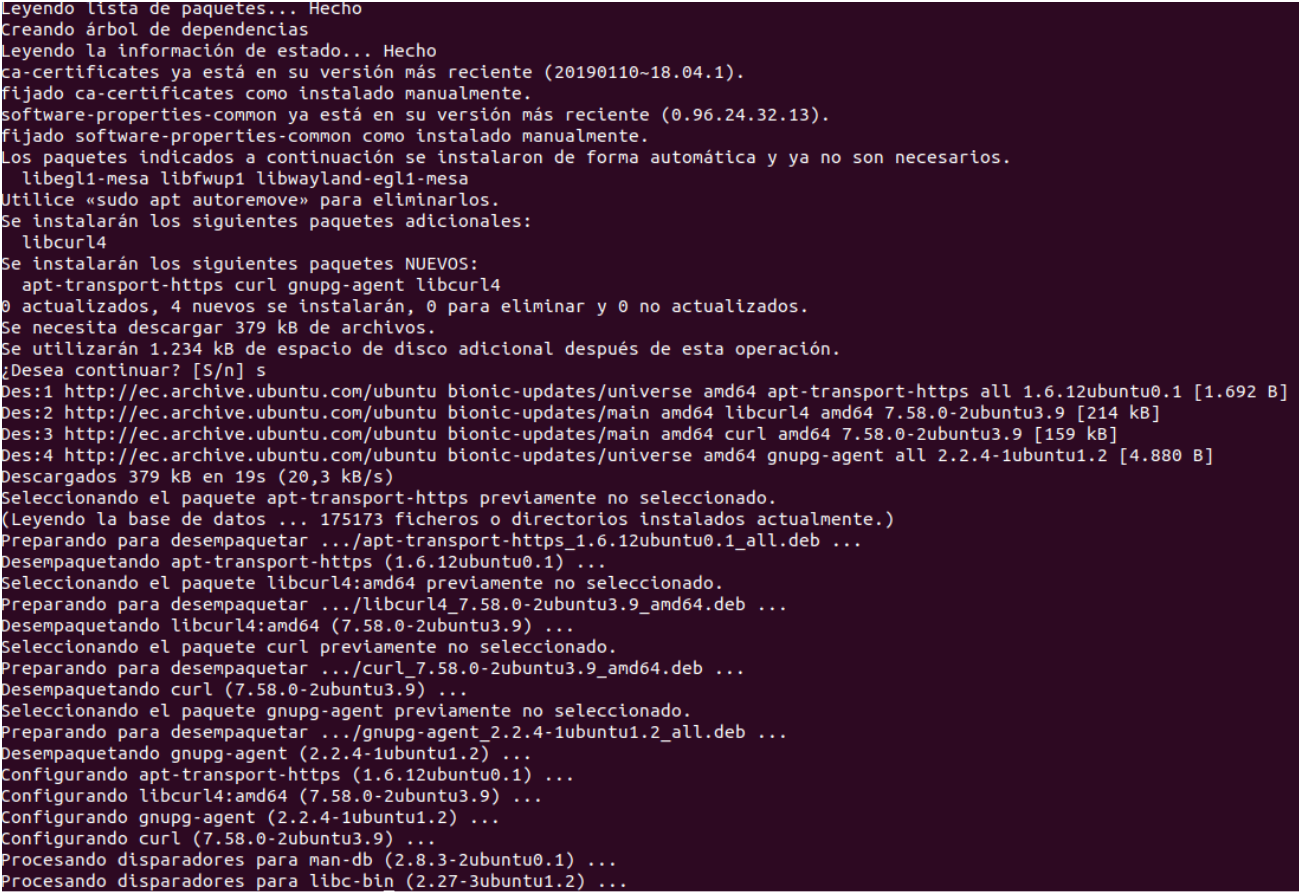
* + - 1. Instalación con *Docker*.

*Docker* consiste en crear contenedores ligeros y portables, para distintas aplicaciones de software, es decir, dentro de cada contendor se puede instalar librerías, herramientas, plataformas, bases de datos, entre otras, independientemente del sistema operativo que el computador tenga por debajo, con el fin de facilitar los despliegues.

El proceso de instalación de la librería *RAPIDS* con *Docker* se detalla de la siguiente manera, a continuación, el proceso siguiente es:

* Instalación de paquetes, para permitir que *apt* use un repositorio sobre *HTTPS*.





*Figura 37.* Repositorio sobre *HTTPS*.

*Elaboración*: Autor.

La figura 37 representa la instalación de los principales paquetes para apt, esto quiere decir que apt dentro de *Ubuntu* permitirá tomar acciones sobre el repositorio *HTTPS*, con el fin de poder entrar al mismo, con los principales cuadernos de *Jupyter*, entrando con el puerto *localhost:8888* (ver figura 47).

* Agregar la clave *GPG* oficial de *Docker*.

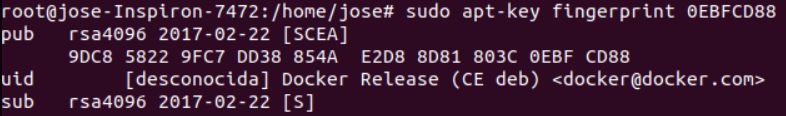


*Figura 38.* Agregar clave

*Elaboración*: Autor.

La figura 38 permite agregar la clave oficial de *Docker* para *apt*, como resultado debe salir *OK*, esto quiere decir que la clave fue agregada con éxito y no hubo complicaciones en el proceso.

* Verificar clave.

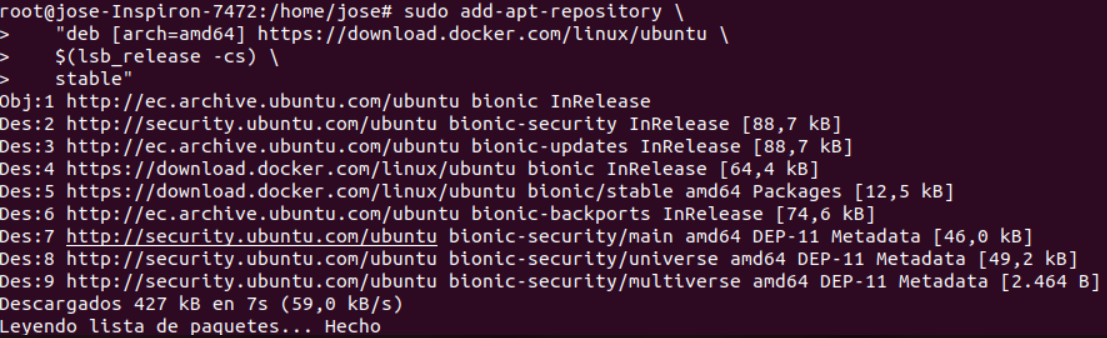


*Figura 39.* Clave huella digital.

*Elaboración*: Autor.

La figura 39 verifica que tiene la clave con la huella digital 9DC8 5822 9CF7 DD38 854A E2D8 8D81 803C 0EBF CD88, buscando los últimos 8 caracteres de la huella 0 0.

* Configurar el repositorio a estable.

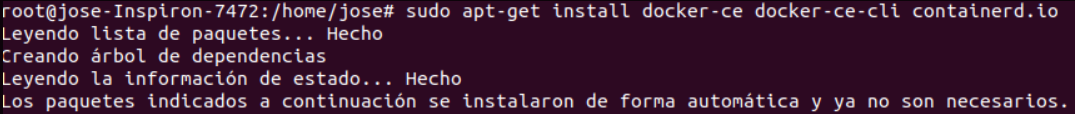


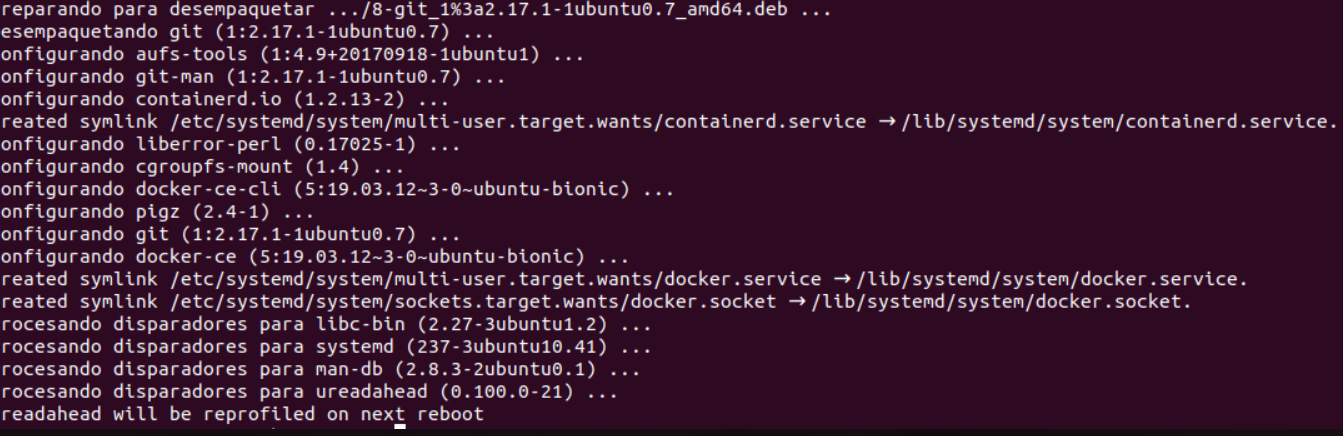
*Figura 40.* Repositorio estable.

*Elaboración*: Autor.

La figura 40 permite configurar el repositorio a un estado estable, con el fin de no tener problemas futuros y además al momento de ingresar al repositorio, en este caso a los cuadernos de *Jupyter* con *Docker* y *RAPIDS*, se puede ingresar de manera correcta (ver figura 47).

* Instalar *Docker* en la máquina personal.



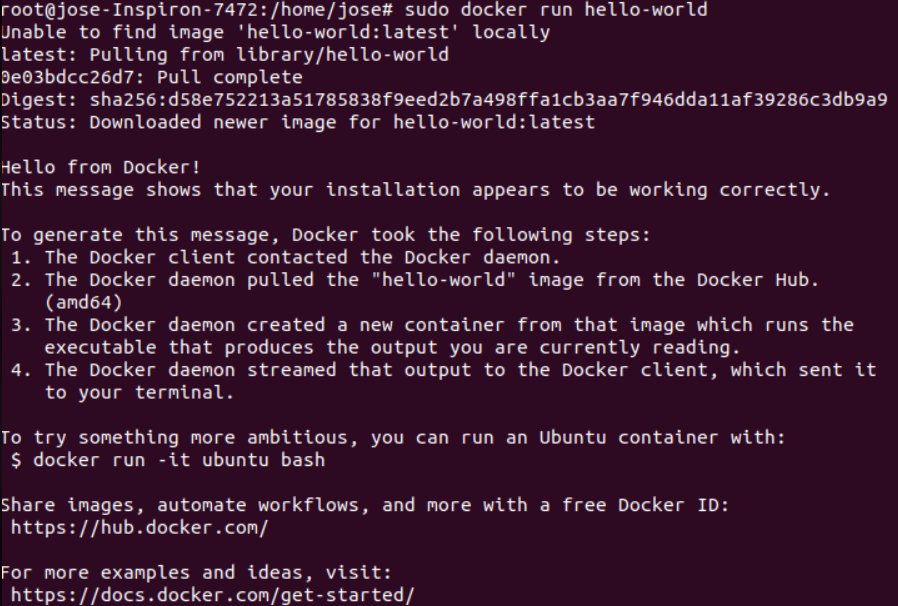


*Figura 41.* Instalar Docker.

*Elaboración*: Autor.

La figura 41 consiste en instalar *Docker* en su última versión 19.03, permite poner en marcha el contenedor en la máquina personal, *docker* ya se encuentra instalado.

* Realizar prueba de *Docker* en máquina personal.

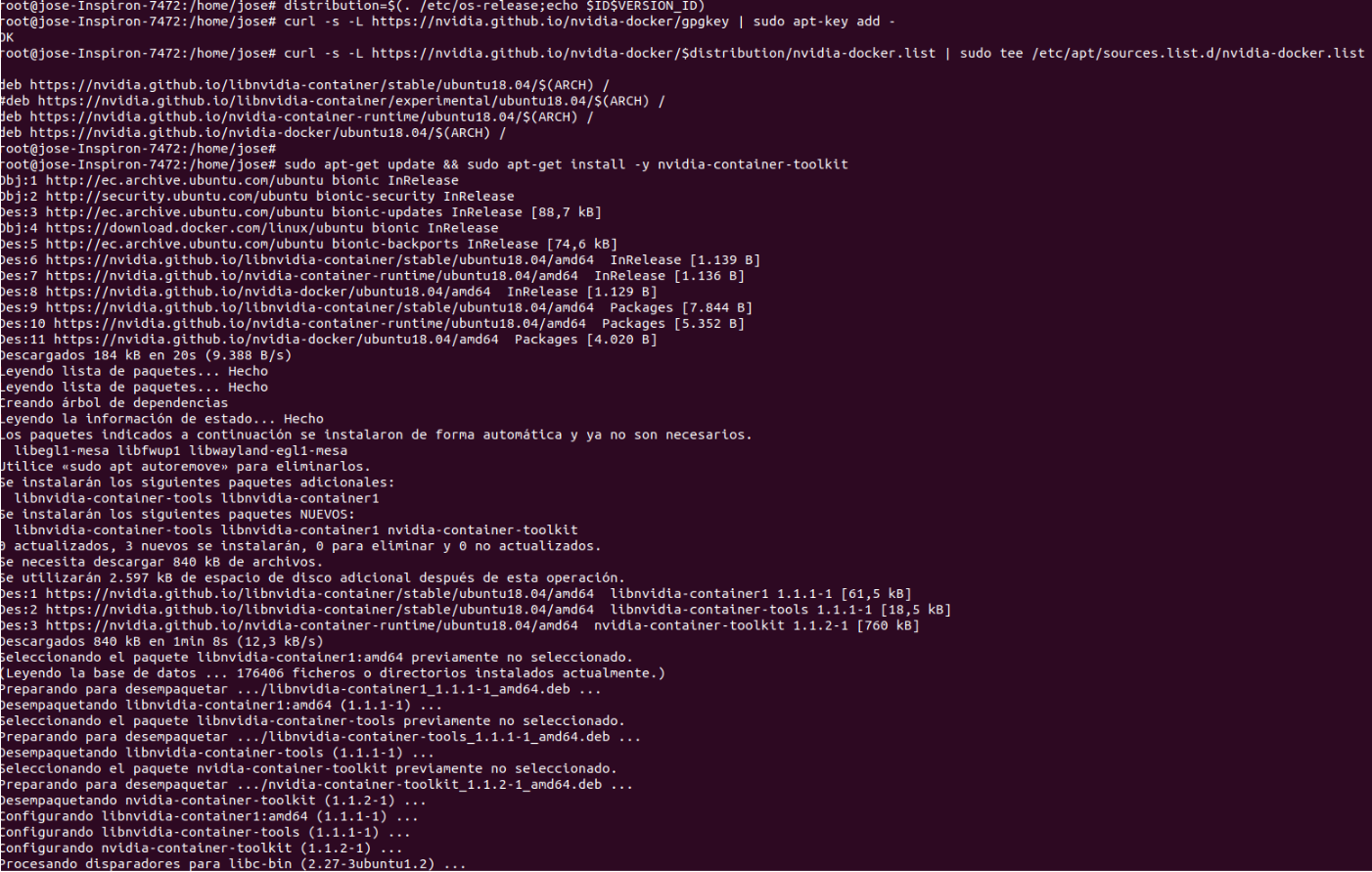


*Figura 42.* Prueba *Docker*.

*Elaboración*: Autor.

La figura 42 permite verificar con un simple ejemplo como *“hello-world”* si *Docker* se instaló correctamente y si está funcionando con las principales características de este.

* Crear y ejecutar contenedores *Docker* acelerados por *GPU*.



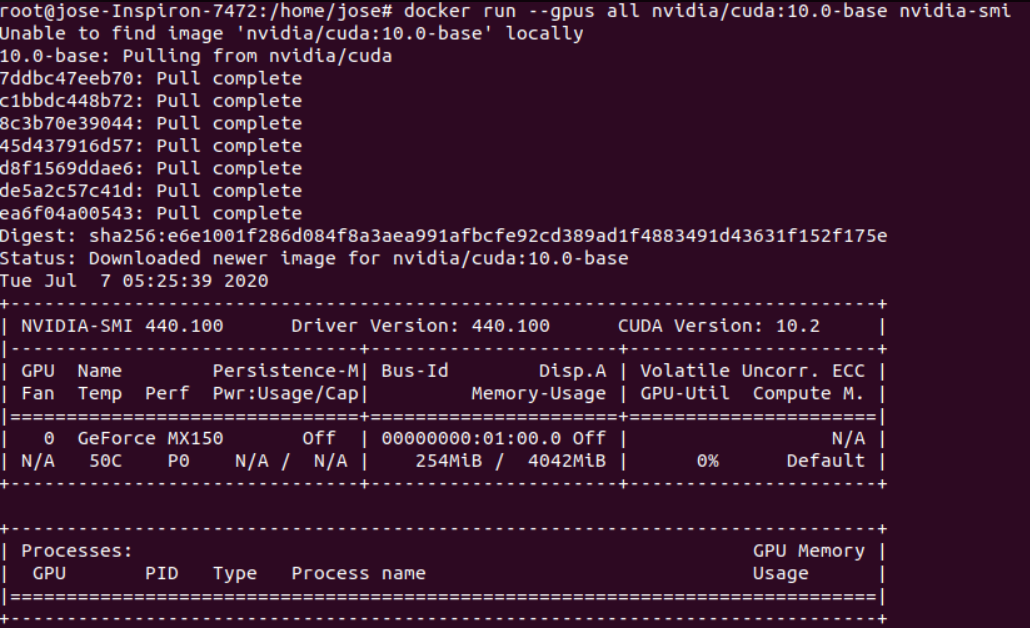


*Figura 43.* Contenedores *Docker* con *GPU*.

*Elaboración*: Autor.

La figura 43 incluye una biblioteca de tiempo en ejecución de contendor y utilidades para configurar automáticamente los contenedores para aprovechar las *GPU* *NVIDIA* de nuestra máquina personal, dentro del contenedor creado y seguidamente reiniciamos el contenedor.

* Prueba de *GPU* *Nvidia* en *Docker*.

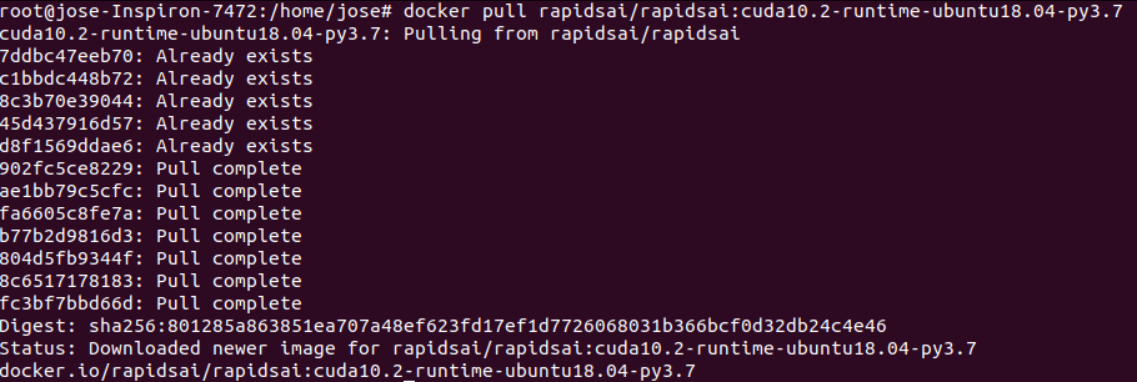


*Figura 44.* Prueba *GPU* en *docker*.

*Elaboración*: Autor.

La figura 44 representa una prueba *nvidia-smi* con la última imagen oficial de *CUDA*, cabe recalar que el contendor ya tiene las características de la *GPU* del computador personal (ver figura 36).

* Instalación de la librería *RAPIDS* al contendor creado en la máquina personal.

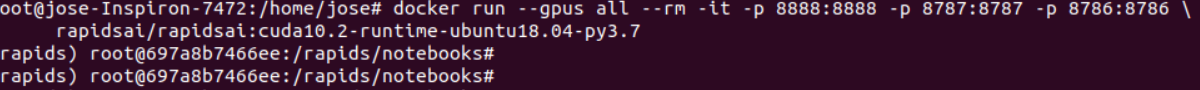


*Figura 45.* Instalación *RAPIDS*.

*Elaboración*: Autor.

La figura 45 representa la instalación de *RAPIDS* con todo el conjunto de bibliotecas que tiene, características y su funcionamiento, además, se instala el lenguaje de programación Python a su última versión 3.7.

* Ejecutar *Docker*.

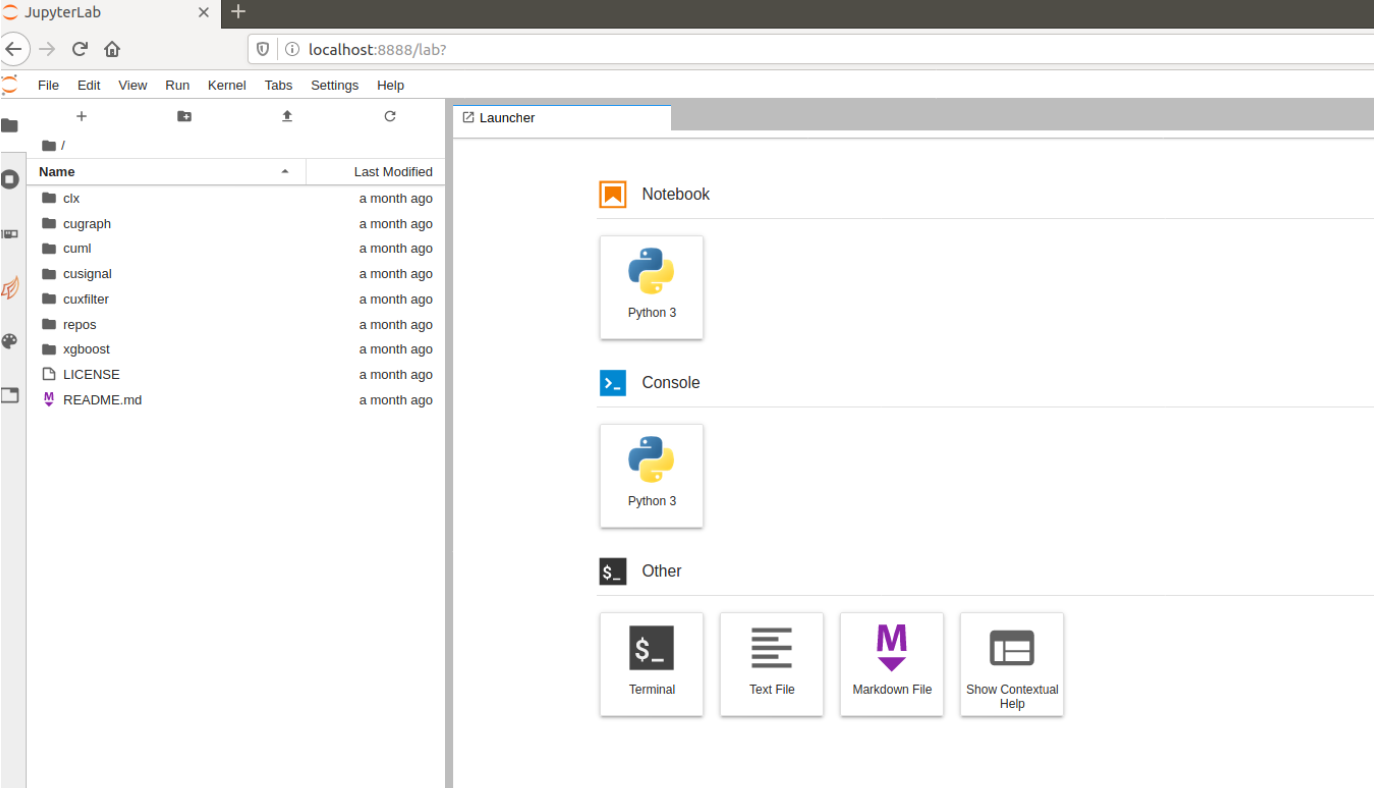


*Figura 46.* Ejecutar *Docker* y *RAPIDS*.

*Elaboración*: Autor.

La figura 46 representa la ejecución de docker con *RAPIDS*, en un servidor local por medio del puerto 8888, además, el contendor ya proporcionada ejemplos del conjunto de bibliotecas que tiene *RAPIDS*.

* Plataforma de desarrollo para la *Data* *Science* con *Docker* y *RAPIDS*.



*Figura 47.* Plataforma *Jupyter* con *Docker* y *RAPIDS*.

*Elaboración*: Autor.

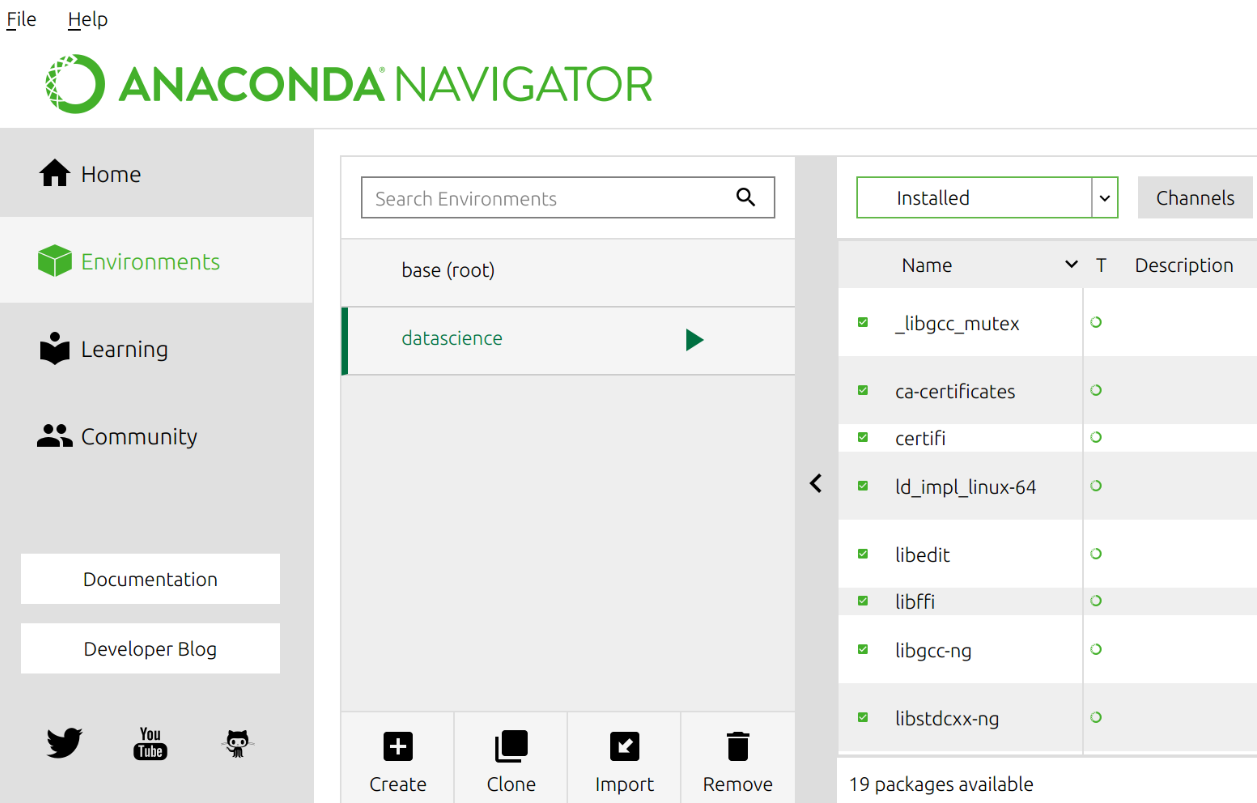
La figura 47 representa el entorno de desarrollo, en donde esta funcionando el contenedor localmente en la máquina personal, dentro del contendor ya se encuentra la librería *RAPIDS* instalada con todo su conjunto de bibliotecas, cuadernos Jupyter de ejemplos, además, se obtiene el funcionamiento de la librería *Dask*, cabe destacar que para ingresar nuevamente a la plataforma de desarrollo debemos ejecutar el comando indicado en la figura 46 (ver figura 46).

* + - 1. Instalación con *Conda*.

*Conda* es un sistema de gestión de paquetes y un sistema de gestión de entornos que puede ser ejecutados en sistemas operativos como *Linux*, *Windows*, *macOS*; utilizada para la *Data* *Science* y aprendizaje automático, incluye un procesamiento de grandes volúmenes de información, análisis predictivo y cómputos científicos impulsada por Python, además, es gratis porque es un proyecto de código abierto, orientado para poder simplificar el despliegue, administración de los paquetes de software, desarrollar aplicaciones de una manera más eficiente, rápida y sencilla.

El proceso de instalación de la librería *RAPIDS* con *Conda* se detalla de la siguiente manera, a continuación, el proceso siguiente es:

* Crear entorno en la plataforma de *Conda*.

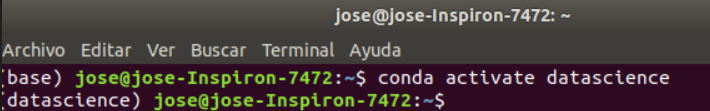


*Figura 48.* Crear entorno.

*Elaboración*: Autor.

La figura 48 representa el panel de administración de Conda, el mismo nos proporcionada una interfaz gráfica amigable, podemos crear el respectivo entorno de trabajo (*datascience*) y donde vamos a instalar la librería RAPIDS.

* Activar entorno de trabajo por el terminal.

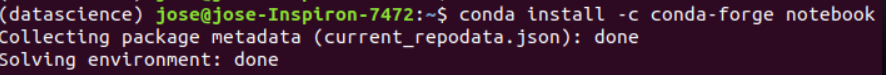


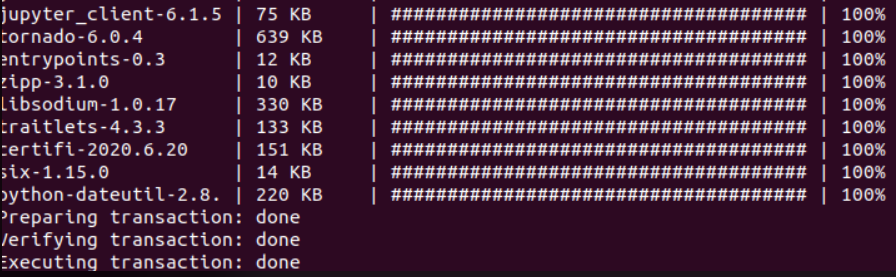
*Figura 49.* Activar entorno.

*Elaboración:* Autor.

La figura 49 permite activar el entorno de trabajo creado en *Conda*, abrimos un terminal y escribimos el comando para activar el entorno de *datascience.*

* Instalar *Jupyter*.



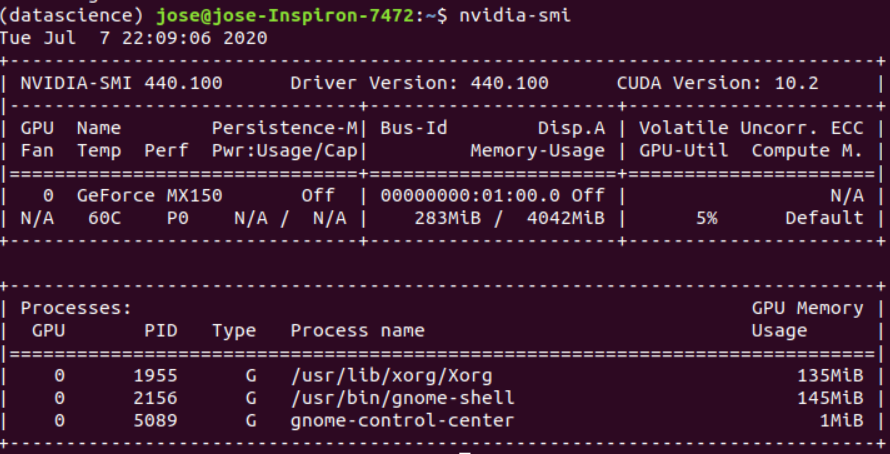


*Figura 50.* Instalación *Jupyter*.

*Elaboración:* Autor.

La figura 50 representa la instalación de *Jupyter* en nuestro entorno de trabajo *Conda*, permitirá trabajar en un ambiente de desarrollo más amigable e incorporaremos todo el código fuente en el respectivo cuaderno de *Jupyter*.

* Prueba de *GPU* *Nvidia* en *Conda*.

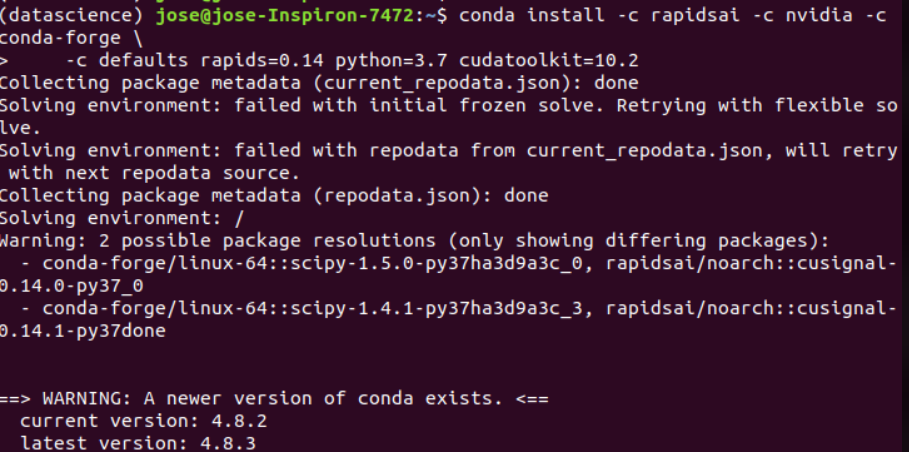


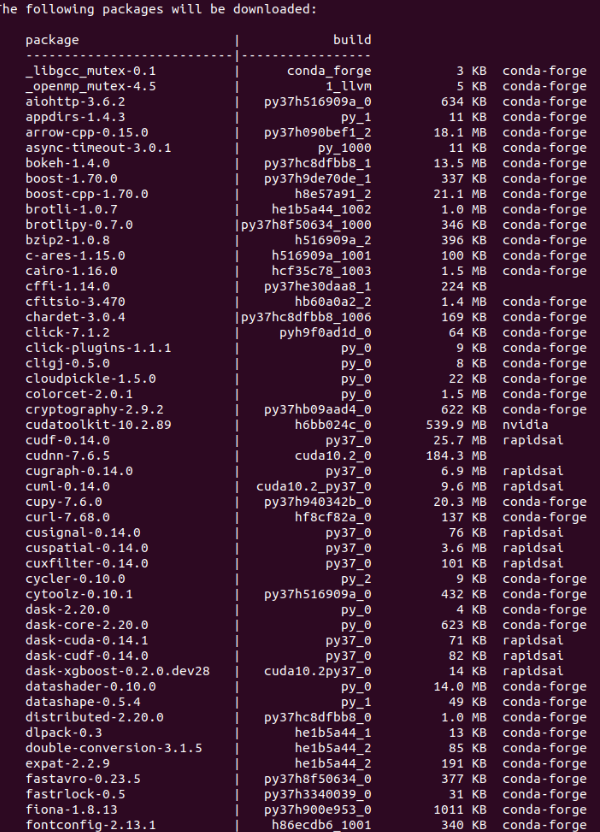
*Figura 51.* Prueba *GPU* en *Conda*.

*Elaboración:* Autor.

La figura 51 representa una prueba *nvidia-smi* para poder determinar si nuestro entorno de trabajo creado por *Conda* ya cuenta con las prestaciones de la *GPU* de nuestra máquina personal, cabe recalar que el entorno ya tiene las características principales de la *GPU* de la máquina personal (ver figura 36).

* Instalación de la librería *RAPIDS.*





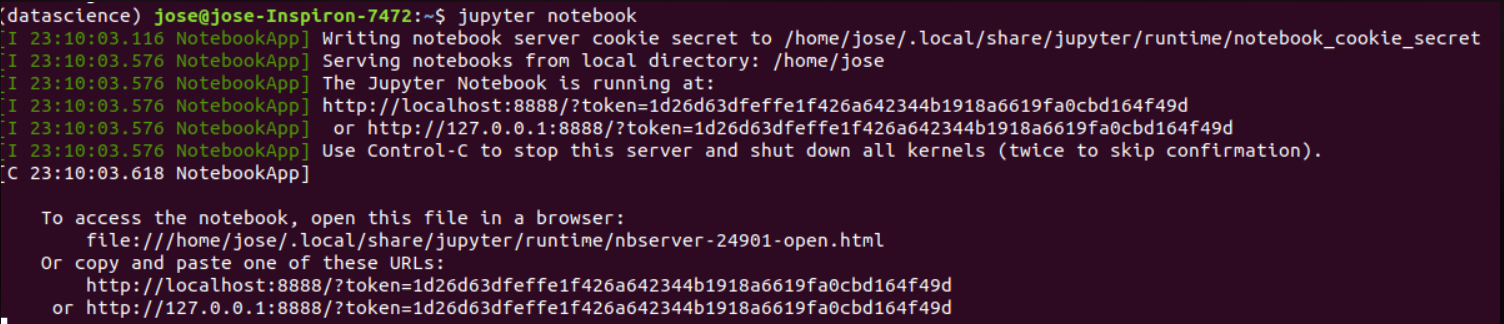


*Figura 52.* Instalación *RAPIDS* al entorno de trabajo *datascience* creado por *Conda*.

*Elaboración:* Autor.

La figura 52 representa la instalación de toda la librería RAPIDS en el entorno de trabajo creado por Conda (*datascience*), se instala todo el conjunto de bibliotecas que tiene RAPIDS con sus principales características y su funcionamiento.

* Activar y entrar al entorno Jupyter.

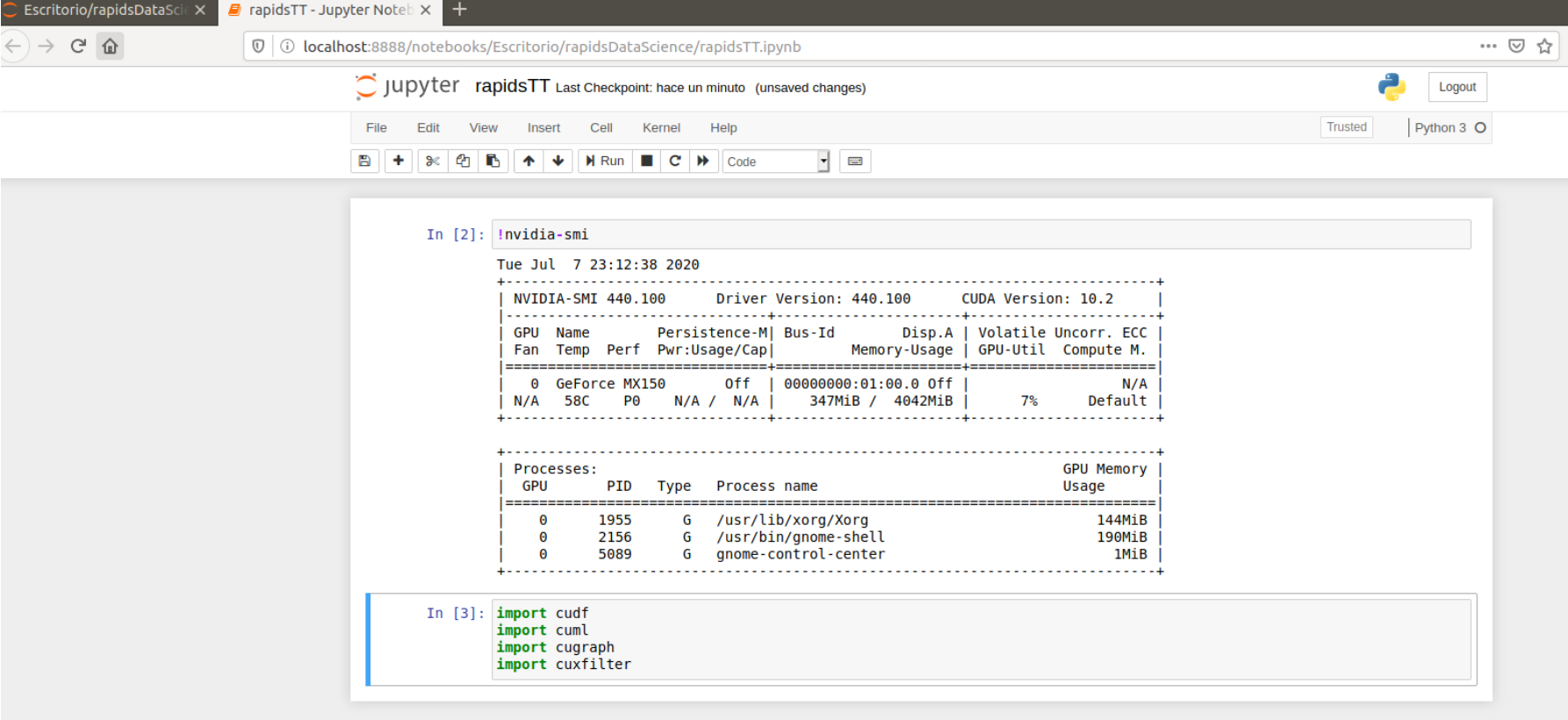


*Figura 53.* Entrar a *Jupyter*.

*Elaboración:* Autor.

La figura 53 permite entrar al entorno de desarrollo de *Jupyter*, con el fin de poder utilizar toda la librería *RAPIDS,* es decir que ya está activado nuestro servidor local por medio del puerto 8888.

* Plataforma de desarrollo *Jupyter* para la *Data* *Science* con *Conda* y *RAPIDS*.



*Figura 54.* Plataforma *Jupyter* con *Conda* y *RAPIDS*.

*Elaboración*: Autor.

La figura 54 representa la plataforma de desarrollo Jupyter instalada en el entorno de trabajo *datascience*, además, se puede observar que la librería RAPIDS ya está funcionando correctamente con todo su conjunto de bibliotecas, cabe destacar que para ingresar nuevamente a la plataforma de desarrollo debemos ejecutar el comando indicado en la figura 53 (ver figura 53).

* + 1. *Google Colabority*

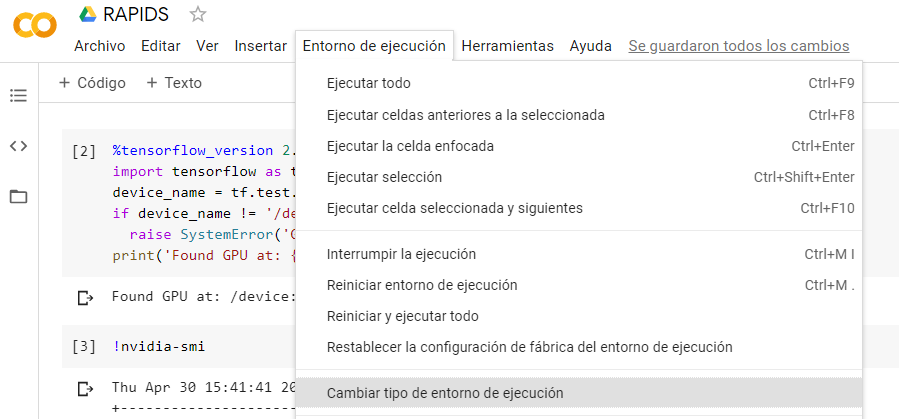
*Google* *Colabority* es una plataforma de entorno gratuito de *Jupyter* *Notebook* que permite escribir, ejecutar código en Python en un navegador como *Google* *Chrome*, dentro de los principales enfoques: no necesita de configuración requerida, acceso gratuito a *GPU* y facilidad de compartir a código con distintos desarrolladores.

*Google* *Colabority* permite crear modelos de Ciencia de Datos, *Machine* *Learning* a través de la nube, libera a la máquina personal de tener un trabajo demasiado extenso, costoso en tiempo y potencia, incluso permite a la máquina trabajar de manera eficiente en la nube.

Para el proceso de instalación por *Colabority* se ha definido por los siguientes puntos principales, son necesarios para el uso de la librería *RAPIDS*.

* + - 1. *Entorno de Ejecución.*

Habilitar entorno de ejecución en *Colabority*, tiene que estar en *GPU* (Unidad de procesamiento gráfico).



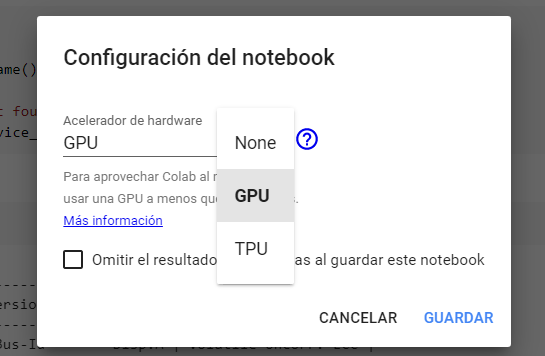


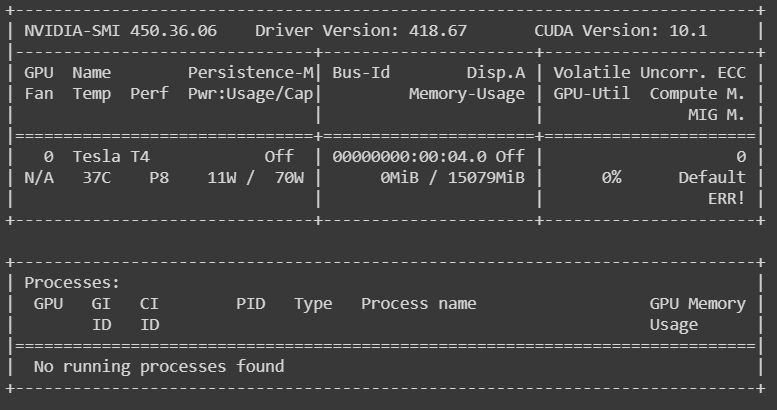
Figura 55. Entorno de ejecución *GPU*.

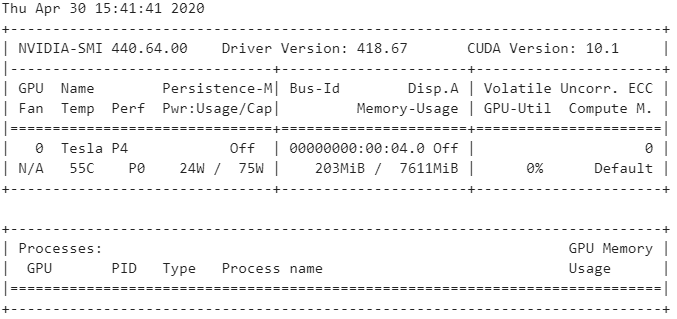
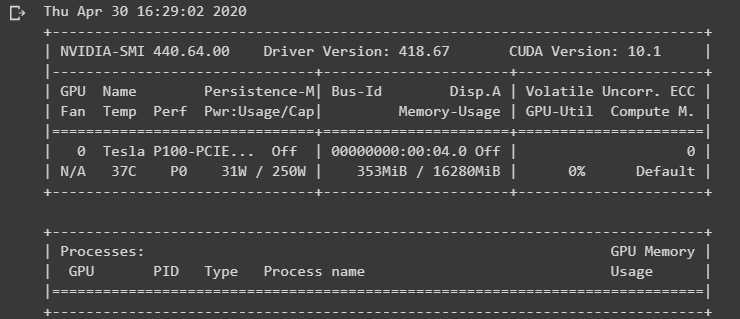
Elaboración: Autor.

La figura 55 representa la configuración de un entorno de ejecución en Colabority, el mismo ya proporciona un entorno GPU, para la aceleración de los datos utilizando las características de la GPU.

* + - 1. Instalación *RAPIDS.*

Verificamos el tipo de *GPU* (ver figura 56) con el que vamos a trabajar, *Google* *Colabority* lanza por defecto las siguientes arquitecturas de *GPU*. Debemos escribir el comando para la verificación de la misma: !nvidia-smi



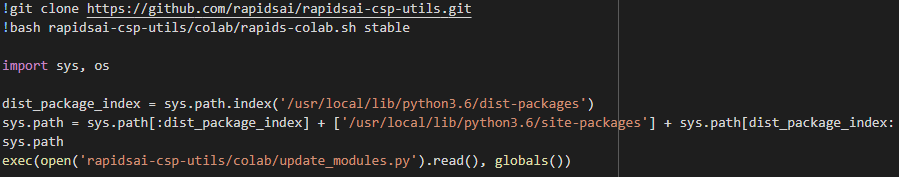
 

*Figura 56*. Tipos *GPU Tesla.*

Se debe tomar en cuenta, para que *RAPIDS* funcione en *Google* *Colabority* debe trabajarse con las arquitecturas de *GPU* siguientes:

* *Tesla T4*
* *Tesla P4*
* *Tesla P100*

La figura corresponde el comando principal para la instalación de *RAPIDS*, instalará todas las dependencias, características y funcionalidades de *RAPIDS*.



*Figura 57.* Instalación RAPIDS.

Elaboración: Autor.

* 1. Proceso de ETL
     1. Extracción cuDF

La figura representa la instalación o importación del conjunto de bibliotecas de RAPIDS y también otras librerías que nos ayudaran en el trascurso del proceso de ETL.

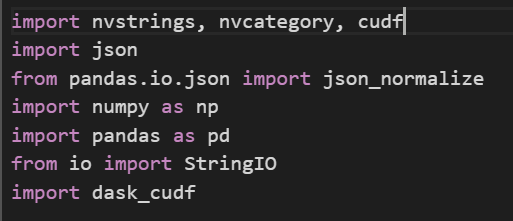


Figura. Importación de librerías

Elaboración: Autor.

La figura representa la carga de datos del computador personal a Google Colabority

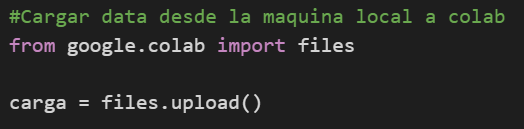


Figura. *Load Data.*

Elaboración: Autor.

La figura representa la lectura del archivo cargado, utilizando la biblioteca *cuDF*, en este caso el archivo es un CSV, utilizamos la funcionalidad de *read\_csv* que tiene *cuDF*. Además incorporamos *time.time* para medir el tiempo de proceso de lectura con el fin de poder comparar con la biblioteca PANDAS y comprender que RAPIDS trabaja de manera acelerada.

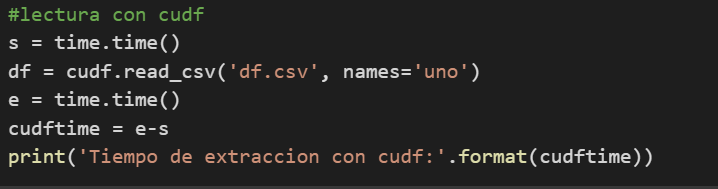


Figura. *Read with cuDF*.

Elaboración: Autor.

La figura representa la lectura del archivo cargado, utilizando la biblioteca *PANDAS*, en este caso el archivo es un CSV, utilizamos la funcionalidad de *read\_csv* que tiene *pandas*. Además incorporamos *time.time* para medir el tiempo de proceso de lectura con el fin de poder comparar con la biblioteca *RAPIDS*.

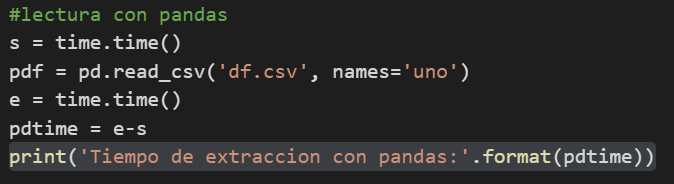


Figura. *Read with PANDAS.*

Elaboración: Autor.

La figura representa la creación del nuevo *DataFrame* con *cuDF*, en este caso la lectura del archivo se realizó con pandas (ver figura 39) y después procedimos a convertir la variable *pdf* a formato *cuDF* con la característica *from\_pandas* (pertenece a *cuDF*)*,* realiza la conversión a formato *cuDF* y crea el nuevo *DataFrame*.

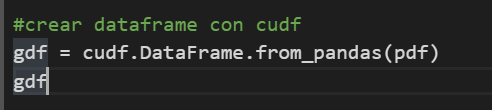


Figura. *Convert DataFrame cuDF.*

Elaboración: Autor.

La figura representa la salida de la creación del DataFrame con cuDF, como se puede observar en la imagen es un DataFrame no entendible, en este aspecto procedemos a crear un nuevo DataFrame para después jugar con los datos y seguidamente realizar el proceso de transformación de la Data.

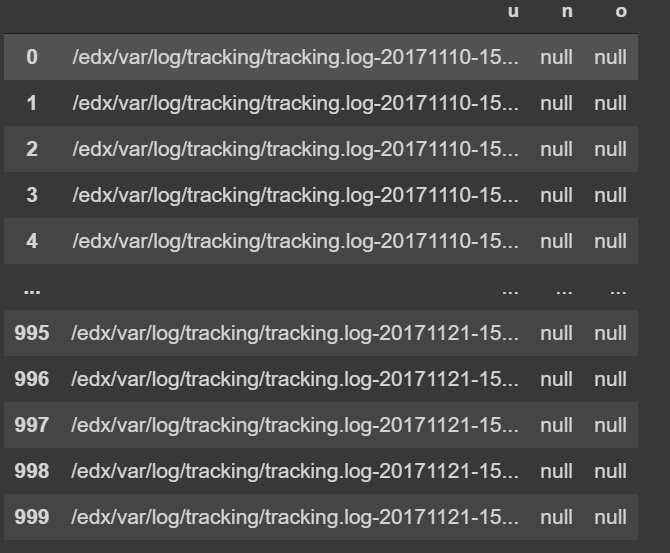
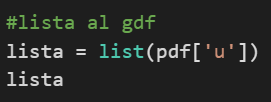


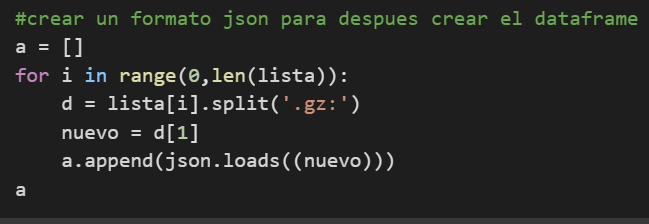
Figura.Salida del *DataFrame.*

Elaboración: Autor.

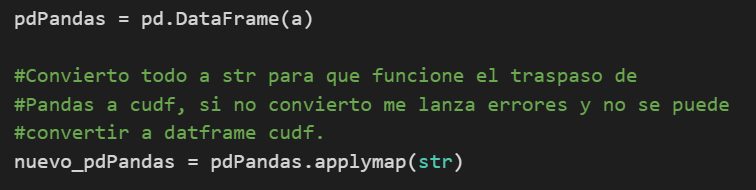
* + 1. Transformación cuDF

La columna u (ver figura 40) la transformamos a una lista para poder tratarla de mejor manera y a su vez entender como está estructurada la información.

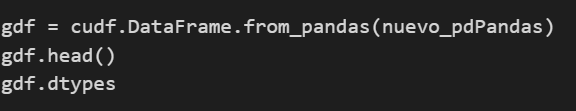




Procedemos a separar la lista y lo cargamos en formato json.



Creamos el *DataFrame* con *PANDAS* y utilizamos la función *applymap* (propia de *PANDAS*) para convertir toda la data en un formato *string*, se realizó para que puede funcionar el traspaso de *PANDAS* a *cuDF*.



Creamos el DataFrame con cuDF con la característica *from\_pandas* (pertenece a *cuDF*)yrealiza la respectiva conversión.

* + 1. Carga
  1. Proceso de Implementación de ML
  2. Presentación de Resultados

Referencias

Aguerzame, A., Pelletier, B., & Waeselynck, F. (2019). GPU Acceleration of PySpark using RAPIDS AI. *DATA 2019 - Proceedings of the 8th International Conference on Data Science, Technology and Applications*, *Data*, 437–442. https://doi.org/10.5220/0008191404370442

Allen, R., & US, B. R. (2020). *Security Alert Analysis Using GPUs.* https://medium.com/rapids-ai/security-alert-analysis-using-gpus-1a31270aa85e

Alpaydin, E. (2009). *Introduction to machine learning*.

Aramburo, R. (2019). BlazingSQL Parte 1: El GPU DataFrame (GDF) y cuDF en RAPIDS AI. *CuDF — GPU Data Processing for GDFs*, *0*(0). https://blog.blazingdb.com/blazingsql-part-1-the-gpu-dataframe-gdf-and-cudf-in-rapids-ai-96ec15102240

Arrow, A. (2016). *The Apache Software Foundation*. https://arrow.apache.org/

B, S. L., Chu, R. S. W., Wang, X., & Luk, W. (2019). *Image Classification on FPGAs* (Vol. 1, Issue 16). Springer International Publishing. https://doi.org/10.1007/978-3-030-17227-5

Castro, J. A. (2018). *Metodología de reduccion de dimensión de tipo espectral con representación interactiva de datos*. 28. http://bdigital.unal.edu.co/64456/1/1127938442.2018.pdf

Chainer. (2019). *A flexible framework for neural networks*. https://chainer.org/

Crist, J. (2016). Dask & Numba: Simple libraries for optimizing scientific python code. *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, 2342–2343. https://doi.org/10.1109/BigData.2016.7840867

El, E. N., Cuda, E., Aplicaciones, E. N., & Tecnolog, N. C. (2015). *Segunda Parte : TECNOLOGÍA CUDA*. 12–26.

Enemark, A. (2018). *Accelerating Cross Filtering with cuDF*. https://medium.com/rapids-ai/accelerating-cross-filtering-with-cudf-3b4c29c89292

Estrada, J. C. ., Silva, I. A. ., & Paéz, J. O. . (2018). Big Data: Ventajas y desventajas-aplicaciones y tecnologías para implementar el servicio. *COMITÉ CIENTÍFICO CICOM 2018*, *0*(0).

Garre, M., Cuadrado, J. J., Sicilia, M. A., Rodríguez, D., & Rejas, R. (2007). Comparación de diferentes algoritmos de clustering en la estimación de coste en el desarrollo de software. *Revista Espa？ola de Innovación, Calidad e Ingeniería Del Software*.

Guim, F., & Rodero, I. (2019). *Arquitecturas basadas en computación gráfica (GPU)*. *0*(0), 68.

Howard, K., Siddharth, S., & Sergei, V. (2010). Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms. *A Model of Computation for MapReduce*, *0*(0).

Huang, J., Lin, Z., Ma, C., & Yuan, X. (2013). *GPU SPEED-UP FOR THE IMPLICIT NAVIER-STOKES SOLVER*. *0*(0), 10.

Joyones Aguilar, L. (2016). *Big Data, Análisis de grandes volúmenes de datos en organizaciones*. https://books.google.com.ec/books?hl=es&lr=&id=1GywDAAAQBAJ&oi=fnd&pg=PT6&dq=Aguilar,+L.+J.+(2016).+Big+Data,+Análisis+de+grandes+volúmenes+de+datos+en+organizaciones.+Alfaomega+Grupo+Editor.&ots=\_WU9L37j\_Q&sig=YPC8eRaPaRv56HxrbY01I8IoRmk&redir\_esc=y#v=on

Keller, C. A., Clune, T. L., Thompson, M. A., Stroud, M. A., Evans, M. J., & Ronaghi, Z. (2019). *Accelerated simulation of air pollution using NVIDIA RAPIDS*. *November*, 4–6.

López García, D. (2012). Análisis de las posibilidades de uso de Big Data en las organizaciones. *Otros Conceptos Relacionados Con Big Data*, *0*(0), 18–62.

Mansingh, G., Osei-Bryson, K. M., Rao, L., & McNaughton, M. (2017). Data preparation: Art or science? *Proceedings of the 2016 International Conference on Data Science and Engineering, ICDSE 2016*. https://doi.org/10.1109/ICDSE.2016.7823936

Matsuoka, S., Aoki, T., Endo, T., Nukada, A., Kato, T., & Hasegawa, A. (2009). GPU accelerated computing–from hype to mainstream, the rebirth of vector computing. *Commoditization of HPC and Its Acceleration, but Niche Still Remains .*, 11.

MXNET. (2019). *A FLEXIBLE AND EFFICIENT LIBRARY FOR DEEP LEARNING*. https://mxnet.apache.org/

Nolet, C. (2019). *Combining Speed & Scale to Accelerate K-Means in RAPIDS cuML*. https://medium.com/rapids-ai/combining-speed-scale-to-accelerate-k-means-in-rapids-cuml-8d45e5ce39f5

Numba. (2019). *Numba makes Python code fast*. https://numba.pydata.org/

NVIDIA. (2009). NIVDIA CUDA Architecture. *NVIDIA CUDA Architecture Introduction & Overview*, *0*(0), 9.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). *GPU Computing*. *0*(0), 18.

Patterson, J. (2019). *The Platform Inside and Out*.

Peltenburg, J., van Straten, J., Wijtemans, L., van Leeuwen, L., Al-Ars, Z., & Hofstee, P. (2019). *Fletcher: A Framework to Efficiently Integrate FPGA Accelerators with Apache Arrow*. 270–277. https://doi.org/10.1109/fpl.2019.00051

Pérez Represa, C., Cámara Nebreda, J. M., & Sánchez Ortega, P. L. (2016). INTRODUCCIÓN A LA PROGRAMACIÓN EN CUDA. *INTRODUCCIÓN A LA PROGRAMACIÓN EN CUDA*, *0*(0), 74.

Puneet, G. (2018). Speed Up your Algorithms Part 2— Numba. *Get C++/Fortran like Speed for Your Functions with Numba*. https://towardsdatascience.com/speed-up-your-algorithms-part-2-numba-293e554c5cc1

PyTorch. (2019). *FROM RESEARCH TO PRODUCTION*. https://pytorch.org/

Rabhi, S., Sun, W., Perez, J., Kristensen, M. ., Liu, J., & Oldridge, E. (2019). Accelerating recommender system training 15x with RAPIDS. *In Proceedings of the Workshop on ACM Recommender Systems Challenge.*

RAPIDS Development Team. (2018). *RAPIDS: Collection of Libraries for End to End GPU Data Science*. https://rapids.ai

Rees, B. (2019). *RAPIDS cuGraph*. https://medium.com/rapids-ai/rapids-cugraph-1ab2d9a39ec6

Shilpi, S., & Saurabh, G. (2001). Practical Real-Time Data Processing and Analytics. In *Assiut Journal of Environmental Studies*.

Shohei, H. (2016). Complex neural networks made easy by Chainer. *A Define-by-Run Approach Allows for Flexibility and Simplicity When Building Deep Learning Networks.* https://www.oreilly.com/content/complex-neural-networks-made-easy-by-chainer/

Srinath, A., & Kraus, K. (2019). RAPIDS and cuDF: Accelerating DataFrames on GPUs. *Hadoop Processing, Reading from Disk. Spark In-Memory Processing. Traditional GPU Processing. RAPIDS.*, *0*(0), 46.

Team, D. D. (2016). *Dask: Library for dynamic task scheduling*. https://dask.org

Thi Yen, P., Deok-Young, L., & Jeong-Gun, L. (2017). Impacts of optimization strategies on performance, power/energy consumption of a GPU based parallel reduction. *GPU Architecture and CUDA*, *0*(0), 14.

Turner, J. (2011). *Hadoop: What it is, how it works, and what it can do*. *0*(0), 2.

Vishal, M. (2019). *Accelerating Random Forests up to 45x using cuML*. https://medium.com/rapids-ai/accelerating-random-forests-up-to-45x-using-cuml-dfb782a31bea

White, T. (2015). *Hadoop The Definitive Guide Fourth Edition* (Vol. 0, Issue 0).