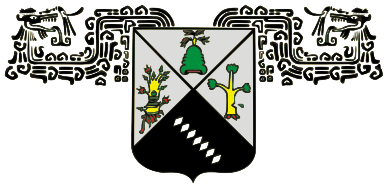


**UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS**

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO
Maestría en Optimización y Cómputo Aplicado

# Almacenamiento y procesamiento de grandes volúmenes de datos en una arquitectura Big Data: aplicaciones en la investigación e industria eléctrica

# T E S I S

**PRESENTA:**

**Javier Alberto Pérez Garza**

PARA OBTENER EL GRADO DE:
Maestro en Optimización y Cómputo Aplicado

**DIRECTOR DE TESIS:**
**Dra. Lorena Díaz González**

**COASESOR DE TESIS:**
**Dr. Benjamín Eddie Zayas Pérez**

**Cuernavaca, Morelos** **Octubre 2019**

# RESUMEN


**Almacenamiento y procesamiento de grandes volúmenes de datos en una arquitectura Big Data: aplicaciones en la investigación e industria eléctrica**

Javier Alberto Pérez Garza

El término Big Data es utilizado para definir datos que, por su complejidad, es inadecuado o imposible el uso de sistemas tradicionales para su almacenamiento, procesamiento y análisis. En particular, en Big Data los datos se caracterizan por aparecer en grandes volúmenes, ser generados con gran velocidad, requerir cortos tiempos de procesamiento, provenir de diversas fuentes y presentarse en una gran variedad de formatos. Los avances tecnológicos y la aparición de nuevos dispositivos capaces de generar datos, ha empezado a generar problemas Big Data en múltiples áreas en empresas, industrias y de investigación. La importancia de generar valor de los datos ha llevado a utilizar soluciones computacionales basadas en sistemas distribuidos que permiten escalar las arquitecturas de hardware de acuerdo con las necesidades del problema.

En este trabajo, se introducen los fundamentos teóricos y metodológicos de una arquitectura Big Data basada en el ecosistema de Hadoop para el almacenamiento y procesamiento de grandes volúmenes de datos. Además, se presenta el desarrollo de dos soluciones Big Data en áreas donde el crecimiento en el volumen de datos está impulsando la búsqueda de métodos computacionalmente eficientes para el análisis de datos. La primera, en el área de investigación en genómica, utiliza una metodología escalable para procesar bases de datos de ADN de referencia y crear nuevas bases de datos de tamaño reducido mediante el uso de la información filogenética de las secuencias. La segunda, en la industria eléctrica, se utilizan datos de consumo eléctrico generados por medidores inteligentes, sociales y climatológicos para generar un *pipeline* de algoritmos de aprendizaje de máquina con el objetivo de crear modelos escalables para el pronóstico del consumo eléctrico de usuarios de la red eléctrica inteligente.

# Agradecimientos


- A mis asesores, que me apoyaron constantemente durante todas las etapas de este proyecto de investigación.
- A los profesores de la MOCA, Instituto de Matemáticas y del Centro de Investigación en Ciencias, por su gran labor de formar futuros investigadores.
- A mis compañeros y amigos, que siempre me apoyan y me ayudan a aprender algo nuevo cada día.

# Índice general

**CAPÍTULO**

# Abreviaturas

- HDFS: *Hadoop Distributed File System* o Sistema de Archivos Distribuido de Hadoop.
- RAM: *Random Access Memory*.
- RDD: *Resilient Distributed Datasets*.
- API: *Application Programming Interface*.
- SQL: *Structured Query Language*.
- UDF: *User Defined Function*.
- ML: *Machine Learning*, aprendizaje de máquinas o aprendizaje automático.
- BD o BDs: Base de datos o Bases de datos.
- ADN: Ácido DesoxirriboNucleico.
- GB: *GigaByte*.
- CFE: Comisión Federal de Electricidad.
- TB: *TeraByte*.
- CSV: *Comma Separated Values*.
- OHE: *One Hot Encoding*.
- RMSE: *Root Mean Square Error*.

# Índice de tablas

# Índice de figuras

Capítulo 1

# Introducción

## 1.1. El contexto de Big Data

La acelerada evolución tecnológica observada en los últimos años ha revolucionado múltiples aspectos en la sociedad, ciencia e industria. En la sociedad, la tecnología se ha integrado en la vida diaria de las personas, transformando los medios de comunicación, entretenimiento, transporte y muchos otros más. En la ciencia e industria, el uso de la tecnología ha permitido generar avances de gran magnitud, al lograr optimizar y volver más eficientes los procesos para la generación de conocimiento, productos o servicios. Uno de los factores con gran impacto en esta evolución tecnológica es el inicio de la comercialización de la Internet en la época de los 90's, donde se permitió a la academia, empresas, instituciones, computadoras personales y dispositivos móviles conectarse a la red para compartir y recibir información. En la actualidad, existen cada vez más dispositivos y servicios que están basados en el uso de Internet. Además de los dispositivos, sistemas y tecnologías tradicionales, la aparición de comercios electrónicos, redes sociales, Internet de las cosas o *Internet Of Things* y nuevos dispositivos de bajo costo, han conllevado a un crecimiento exponencial en la capacidad de generación de datos.

El aumento en la cantidad de datos disponibles ha generado la oportunidad y necesidad de extraer la información y conocimiento relevante que se encuentra en los mismos, sin embargo, cuando la complejidad de los datos supera la capacidad de los sistemas de manejo de datos tradicionales, es necesario recurrir a nuevos métodos que sean escalables y eficientes. Grandes empresas como Google y Yahoo fueron las primeras en emplear soluciones alternativas para el manejo de este tipo de datos. No obstante, no solo las grandes empresas se han encontrado con problemas de grandes volúmenes de datos, sino también han aparecido o iniciado a aparecer en múltiples industrias y áreas de investigación, por lo cual existen las mismas necesidades y oportunidades para generar valor con los datos.

El término Big Data se utiliza para definir a los datos que por sus propiedades, vuelven imposible o inadecuado utilizar sistemas computacionales tradicionales. Para problemas tipo Big Data, es indispensable el uso de sistemas distribuidos, que permitan almacenar y procesar grandes cantidades de datos de forma eficiente, confiable y a bajo costo. Las propiedades de esta clase de sistemas permiten escalar la arquitectura de acuerdo con el tamaño del problema. Para resolver problemas tipo Big Data, es necesarios obtener el mayor valor del sistema, al desarrollar aplicaciones que aprovechen las capacidades de *Hardware* de los equipos de cómputo que formen parte del sistema distribuido.

## 1.2. Organización de la tesis

En este trabajo de investigación, se implementaron dos soluciones escalables para resolver problemas tipo Big Data en dos áreas de aplicación de diferentes ámbitos. En el Capítulo 2 de este trabajo se presentan en detalle los fundamentos teóricos y metodológicos de Big Data, sistemas distribuidos y la arquitectura para el almacenamiento, procesamiento y análisis distribuido de datos.

El Capítulo 3 documenta la primera solución a un problema tipo Big Data en un área de investigación, la reducción de bases de datos de ADN. En el área de genómica, múltiples herramientas para el análisis de secuencias utilizan bases de datos (BDs) de referencia como parte de su metodología. El inmenso tamaño de las BDs y su constante crecimiento, en conjunto con la complejidad computacional de los algoritmos de análisis, han creado la necesidad de utilizar nuevas técnicas para mejorar la calidad y velocidad de los análisis. En este trabajo, se desarrolló una aplicación para la reducción de BDs de ADN. Ésta aplicación utiliza una arquitectura Big Data para almacenar y procesar de manera eficiente las secuencias de ADN, generando nuevas BDs de secuencias que pueden ser utilizadas por nuevas y existentes herramientas de análisis.

En el Capítulo 4 se presenta una solución a un problema tipo Big Data en la industria eléctrica, el pronóstico del consumo eléctrico de hogares con datos de medidores inteligentes. En la industria eléctrica, la transformación de la red eléctrica tradicional hacia una red eléctrica inteligente implica la inclusión de nuevos dispositivos para el monitoreo y control. Entre éstos, los medidores inteligentes, son dispositivos que son capaces de realizar mediciones del consumo eléctrico de usuarios en cortos periodos de tiempo. La gran cantidad de datos generados por éstos y otros dispositivos de la red eléctrica, así como por fuentes externas, ocasionan que sea necesario utilizar métodos escalables, seguros y eficientes para obtener valor de los datos. Con este objetivo, se desarrolló un flujo de trabajo dinámico para el pronóstico del consumo de usuarios de medidores inteligentes con algoritmos de aprendizaje automático.

Finalmente, el Capítulo 5 proporciona las conclusiones generales de este trabajo de investigación.

Capítulo 2

# Fundamentos teóricos y metodológicos

## 2.1. Definición de Big Data

Big Data es el término utilizado para definir problemas donde la complejidad de los datos ocasiona que métodos tradicionales no puedan ser utilizados para su almacenamiento, procesamiento y análisis. En Big Data los datos aparecen en grandes cantidades, son generados en cortos periodos de tiempo, requieren mayor velocidad de procesamiento y/o se encuentran en diferentes tipos de formatos. Por lo general, Big Data suele ser definido con las 3 Vs, representadas visualmente en la Figura 2.1 y descritas a continuación:

- Volumen. La gran cantidad de datos supera las capacidades de sistemas tradicionales de almacenamiento y procesamiento, los datos requieren almacenamiento eficiente y confiable.
- Velocidad. Los datos pueden ser generados en cortos lapsos de tiempo y estos pueden requerir cortos tiempos de procesamiento, adecuados para la problemática.
- Variedad. Los datos pueden provenir de múltiples fuentes y en diferentes tipos de formatos, los datos pueden ser estructurados, semi-estructurados o no estructurados.

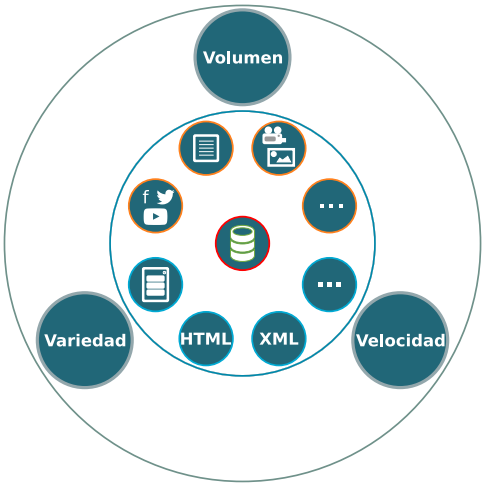


Figura 2.1: El panorama de datos en Big Data.

Además de las tres V's principales, algunas definiciones de Big Data incluyen a una cuarta y quinta V. La cuarta V, valor, es indicativa de que los datos por si solos no son la parte importante del concepto de Big Data, sino la posibilidad de obtener información relevante que genera conocimiento que será utilizado para la toma de decisiones. La quinta V, veracidad, indica que la información contenida en los datos puede ser incierta y no siempre puede ser comprobada, ya que ésta puede contener sesgos, anomalías o ruido.

## 2.2. La ley de Moore y sistemas distribuidos

En 1975, Gordon Moore, cofundador de Intel, pronosticó que la cantidad de transistores en un microprocesador se duplicaría cada dos años. La predicción de Moore resultó correcta durante décadas y es mejor conocida como la Ley de Moore. El comportamiento de desarrollo e investigación en la industria de semiconductores ha seguido esta ley de manera regular. Una de las consecuencias directas de la ley de Moore es la reducción de los costos de microprocesadores, haciéndolos generalmente más accesibles. La rápida adopción ha permitido generar grandes avances tecnológicos que impulsan el cambio social, el crecimiento económico y la productividad.

Aun cuando el número de transistores en los microprocesadores ha seguido la tendencia marcada por la ley de Moore, las empresas de microprocesadores encontraron desafíos para aumentar el rendimiento. Durante muchos años, los procesadores utilizaban una arquitectura de un solo núcleo, los avances en su rendimiento eran dependientes de aumentos en la frecuencia con la que éstos trabajaban. Sin embargo, aumentar la frecuencia del procesador demanda aumentar el consumo energético del mismo, causando altas temperaturas y volviéndolos poco eficientes. Esta problemática llevó a la industria a cambiar de estrategia. Se empezó a utilizar los transistores adicionales para colocar un mayor número de núcleos en un solo procesador (procesadores multinúcleo). La arquitectura multinúcleo de los procesadores permitió aumentar el volumen de trabajo computacional que éstos pueden ejercer.

Los avances en el rendimiento de los microprocesadores y la tendencia en la reducción de sus costos, ha causado que cada vez más dispositivos se integren en múltiples ámbitos de la vida humana. La constante integración y aparición de nuevos dispositivos ha ocasionado que exista un crecimiento exponencial en el volumen de datos que se generan día con día, superando las capacidades de almacenamiento y procesamiento de soluciones tradicionales basadas en una sola computadora. Para resolver este problema, se ha optado a utilizar sistemas distribuidos, que otorgan la escalabilidad necesaria para esta clase de problemas.

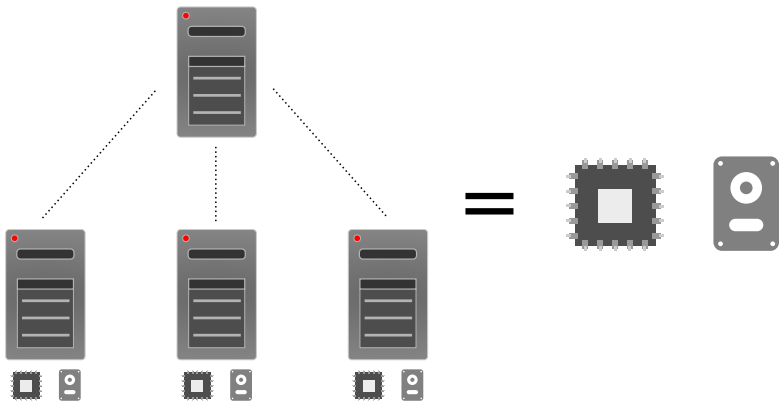

Figura 2.2: Escalabilidad horizontal de recursos de almacenamiento y procesamiento en un sistema distribuido mediante la conexión de computadoras con una red.

Un sistema distribuido, es un sistema conformado por un conjunto de computadoras interconectadas por una red, trabajando en conjunto para lograr un mismo objetivo (Figura 2.2). Este tipo de sistema permite resolver problemas computacionales de gran escala gracias a las propiedades que provee, ya que es sencillo aumentar su rendimiento al agregar más computadoras al sistema, esto es conocido como escalabilidad horizontal. Una ventaja principal en este tipo de escalabilidad es el costo, ya que puede ser más económico adquirir múltiples computadores de bajo costo que una sola computadora de gama alta. Una ventaja más es su tolerancia a fallas, los fallas en algún equipo no causarán que el sistema completo deje de funcionar, ya que cada computadora trabaja de forma independiente a las otras.

## 2.3. Arquitectura Big Data con el ecosistema de Hadoop

Apache Hadoop [2] es un proyecto de software de código abierto diseñado para el almacenamiento y procesamiento distribuido de grandes volúmenes de datos. Dos elementos principales conforman la base de Hadoop, un sistema de archivos distribuido para el almacenamiento de datos y un *framework* para el procesamiento paralelo y distribuido de datos usando el modelo de programación MapReduce. El objetivo de Hadoop es el proporcionar un conjunto de herramientas de software para llevar a cabo cómputo distribuido confiable y altamente escalable al utilizar eficientemente las capacidades de almacenamiento y procesamiento de múltiples nodos de un *cluster* de computadoras. Además de la base de Hadoop, existe una gran variedad de herramientas que trabajan en conjunto con ésta para proveer capacidades especificas para resolver problemas Big Data. Este conjunto de herramientas de Hadoop también es conocido como el ecosistema Hadoop. La base de Hadoop y algunas de las herramientas de su ecosistema son descritas en las siguientes secciones.

### 2.3.1. Sistema de archivos distribuido de Hadoop

El sistema de archivos distribuido de Hadoop o Hadoop Distributed File System (HDFS), es un sistema de archivos diseñado para el almacenamiento de grandes volúmenes de datos en un *cluster* de computadoras. El HDFS está inspirado en *Google File System* [7], un sistema de archivos distribuido diseñado por Google para cubrir sus necesidades crecientes de almacenamiento de datos. El HDFS provee métodos de tolerancia a fallos y alto rendimiento al dividir los archivos en múltiples bloques para almacenarlos en distintos nodos del sistema. Los nodos de un *cluster* podrán almacenar datos y ejecutar tareas de los usuarios, mejorando la eficiencia del procesamiento en paralelo. Las propiedades de la arquitectura distribuida del HDFS permiten al sistema escalar horizontalmente, solo es necesario agregar más nodos para aumentar el rendimiento o capacidad de almacenamiento.

La interfaz de interacción del HDFS está basada en el sistema de archivos de UNIX, pero algunos de los estándares han sido sacrificados a cambio de mejorar la eficiencia para las aplicaciones. Todos los nodos que forman parte del sistema están interconectados y se comunican mediante protocolos basados en TCP (*Transmission Control Protocol*).

En *clusters* de gran tamaño, es común que se presenten errores en los nodos, por lo que es necesario utilizar métodos para la tolerancia a fallos. El HDFS no hace uso de mecanismos de protección de datos como RAID (*Redundant Array of Independent Disks*), sino que proporciona dos tipos de métodos para garantizar la durabilidad de los datos, la replicación de bloques o la codificación de borrado (*Erasure Coding* o EC).

#### 2.3.1.1. Arquitectura

El HDFS utiliza una arquitectura [8] [5] tipo maestro/esclavos (Figura 2.3). De manera general, un *cluster* de HDFS se conforma de un nodo maestro denominado *NameNode*, éste se encarga del manejo del espacio de nombres y de regular el acceso de clientes a los archivos en el sistema. Los nodos restantes (esclavos), denominados *DataNodes*, se encargan de almacenar los datos. En los siguientes párrafos se proporcionan más detalles del funcionamiento y rol de estos dos tipos de nodos.

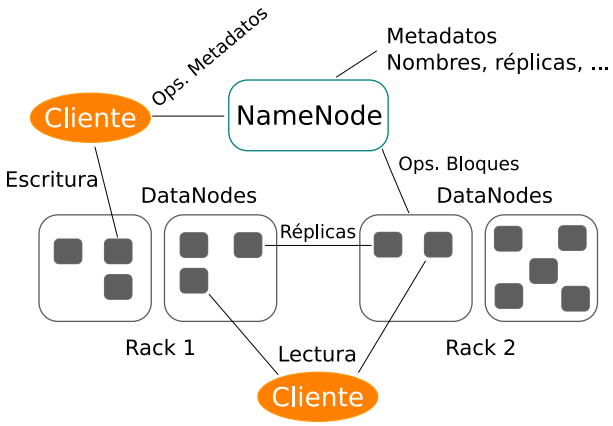


Figura 2.3: Arquitectura maestro/esclavos del HDFS.

(a) **NameNode.** El nodo maestro o *NameNode*, es el encargado del manejo del espacio de nombres del sistema de archivos; el espacio de nombres del HDFS es una jerarquía de directorios y archivos. Los directorios y archivos se representan en el *NameNode* como inodos, que almacenan atributos como permisos, modificaciones, tiempos de acceso y cuotas de espacio. El HDFS divide el contenido de los archivos en bloques que se almacenan en múltiples *DataNodes* del sistema. En el modelo de replicación, los bloques que conforman a un archivo son replicados en diferentes nodos del *cluster* (usando un factor de tres por defecto, pero configurable por archivo). Bajo el modelo de codificación de borrado, los bloques son almacenados en conjunto con las celdas de paridad que pueden ser utilizadas para la recuperación de bloques en caso de presentarse errores. El *NameNode* se encarga de mantener el árbol de nombres y mapeo de los bloques de un archivo a su localización física en los *DataNodes*. Un cliente del HDFS que requiere acceso de lectura a un archivo, contactará al *NameNode* para obtener las localizaciones de los bloques que forman parte del archivo y posteriormente accederá a ellos directamente desde los *DataNodes*. Un cliente que busca acceso de escritura, contactará al *NameNode*, que nominará a múltiples *DataNodes* para el almacenamiento de los bloques.

(b) **DataNode.** Los nodos esclavos o *DataNodes*, son los responsables de realizar las operaciones de lectura y escritura solicitadas por los clientes del sistema de archivos. Éstos además llevan a cabo las operaciones de creación, eliminación, replicación o recuperación de bloques, cuando son solicitadas por el *NameNode*. Cada bloque en un *DataNode* está representado por dos archivos en el sistema de archivos nativo del nodo. El primero contiene los datos del bloque y el segundo sus meta-datos (por ejemplo, la suma de comprobación o *checksum* y la estampa de tiempo de su creación).

Cuando un *DataNode* se conecta por primera ocasión al *NameNode*, éste le asigna un identificador, el cual es almacenado en el *DataNode* y utilizado para subsecuentes conexiones. Solo los nodos debidamente identificados por el *NameNode* podrán conectarse al *cluster*, manteniendo la integridad del sistema de archivos.

Los *DataNodes* mantienen el registro de los bloques que almacenan mediante reportes de bloques. Los reportes son enviados constantemente al *NameNode*, lo que le permite mantener una visión actualizada del estado de los bloques en el *cluster*. Durante su operación regular, los *DataNodes* mandan señales de vida al *NameNode* para confirmar su estado operacional, éstas son utilizadas para conocer el estado de los bloques contenidos en cada *DataNode*. Si el *NameNode* no recibe respuesta de un *DataNode* durante cierto lapso de tiempo, el *NameNode* considera al *DataNode* fuera de servicio y marca como no disponibles a los respectivos bloques. El *NameNode* entonces se hará responsable de comunicar a los *DataNodes* las acciones a llevarse a cabo para evitar la pérdida de datos. Las señales periódicas también son utilizadas para enviar información acerca de la capacidad de almacenamiento de los nodos, la fracción del almacenamiento utilizada, así como el número de operaciones de transferencia de datos que se estén llevando a cabo. Tal información es utilizada por el *NameNode* para el balanceo de cargas. El *NameNode* aprovecha las respuestas a estas señales para enviar instrucciones a los *DataNodes*.

#### 2.3.1.2. Lectura/Escritura de datos

El HDFS provee operaciones de lectura, escritura y eliminación de archivos, así como operaciones para la creación y eliminación de directorios. Cuando una aplicación busca leer un archivo del sistema, primeramente debe hacer contacto con el *NameNode* para conocer la lista de *DataNodes* que contienen bloques que conforman al archivo. Después, el contacto es directo con los *DataNodes*, que le transfieren el acceso a los bloques deseados. Cuando una aplicación busca escribir, ésta hace contacto con el *NameNode* el cual selecciona a los *DataNodes* que serán anfitriones de los bloques del archivo, los cuales proceden a almacenar los bloques hasta que todos éstos han sido almacenados exitosamente.

El HDFS implementa un modelo de un solo escritor, con varios lectores. Cuando una aplicación ha obtenido privilegios de escritura de un archivo, ningún otro cliente podrá modificar este archivo, la aplicación mandará una señal periódica para renovar los privilegios; cuando el archivo ha sido cerrado o no se ha recibido una señal de renovación de privilegios, se revocarán los privilegios, permitiendo que otros clientes puedan obtenerlos. Los bloques de un archivo en el HDFS, no podrán ser modificados ni eliminados, únicamente se podrá concatenar información a la ya existente. El diseño de lecturas y escrituras del HDFS está particularmente optimizado para el procesamiento por lotes, en el cual se requiere alto ancho de banda secuencial tanto en lecturas como escrituras.

#### 2.3.1.3. Tolerancia a fallas

En un *cluster* con grandes cantidades de nodos, las fallas son de esperarse. Los bloques almacenados en un *DataNode* pueden corromperse a causa de fallas en la memoria, discos o en la red. El HDFS provee dos métodos para la tolerancia a fallos [9], la replicación de bloques o la codificación al borrado. La mejor opción metodológica para la tolerancia a fallas variará dependiendo de las necesidades de implementación. Por ejemplo, la replicación, además de proveer protección a fallas, permite aprovechar la localidad de los datos para acelerar tareas de procesamiento por lotes para archivos de gran tamaño. Su principal desventaja es su alto consumo de recursos de almacenamiento, ya que por defecto, el HDFS almacena tres replicas de cada bloque, causando que su almacenamiento sea costoso. En el otro caso, la codificación de borrado permite utilizar una menor cantidad de recursos de almacenamiento, a diferencia de la replicación, los bloques de un archivo son utilizados para calcular bloques de paridad que pueden ser usados para la recuperación en caso de fallas. Sin embargo, la codificación y recuperación de bloques es costosa en términos de CPU y red para los clientes y en los *DataNodes*, además, la codificación de borrado no es aplicable a todas las configuraciones de *clusters*, sino es más adecuada cuando para configuraciones con gran número de nodos o *racks*.

El *NameNode* es el encargado de que los bloques en el HDFS siempre cumplan con los requisitos para garantizar su disponibilidad. El HDFS genera y almacena sumas de comprobación para cada bloque de los archivos en el sistema. Éstas son enviadas al cliente y verificadas en cada lectura para identificar daños en los datos. Además, el *NameNode* recibe reportes periódicos de los *DataNodes* para conocer el estado de los bloques. Cuando existe pérdida de bloques o bloques dañados, se iniciarán los procesos necesarios para la recuperación y se regrese a un estado en el cual el sistema garantice la tolerancia a fallos.

Para *clusters* con gran número de nodos, una práctica común es separar los nodos en múltiples *racks*. Los nodos en un *rack* comparten un *switch* y los *racks* del *cluster* se conectan mediante uno o más *switches*. Hadoop está diseñado para trabajar en sistemas de éste tipo. En el HDFS, se tomará en cuenta el número de *racks* para posicionar de manera inteligente los bloques de un archivo, garantizando la disponibilidad de los datos aún cuando se presentan fallas que impiden la comunicación entre los nodos en diferentes *racks* del sistema.

(a) **Replicación.** Cuando el HDFS utiliza la replicación de bloques que conforman un archivo como método de tolerancia a fallas (Figura 2.4), el *NameNode* se encargará de que los *DataNodes* siempre tengan el número especificado de replicas, dando prioridad a los bloques con el mayor riesgo a causar pérdida de datos. Las replicas son posicionadas en los *DataNodes* de tal manera que se maximice la eficiencia, confiabilidad y disponibilidad de los datos de acuerdo a la configuración del sistema. Las sumas de comprobación son utilizadas para verificar el estado de los bloques. Cuando un cliente encuentra incongruencias en éstas al querer acceder a una replica de un bloque, el *NameNode* es notificado de las réplicas con errores, permitiéndole proporcionar al cliente un nuevo acceso a una replica localizada en un *DataNode* distinto.

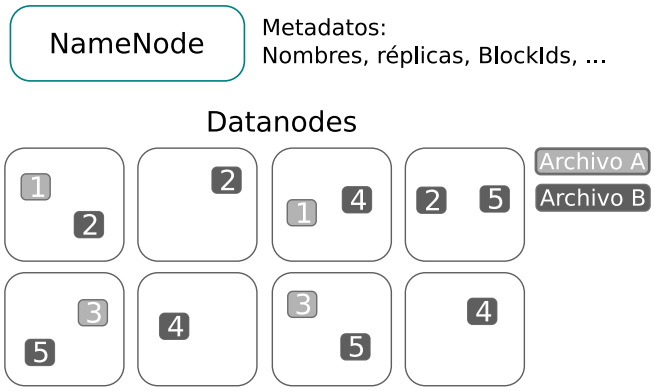


Figura 2.4: Ejemplo de la replicación de bloques en el HDFS. Los bloques que conforman a los archivos A y B son replicados y almacenados en distintos nodos del sistema.

(b) **Codificación de borrado.** Cuando el HDFS utiliza la codificación de borrado como método de tolerancia a fallos, el *NameNode* se encargará de que los bloques que conforman a un archivo y sus respectivos bloques de paridad siempre se encuentren en buen estado para evitar la pérdida de datos. Un algoritmo de codificación es utilizado para calcular bloques de paridad a partir de los bloques que componen a los archivos, los bloques de paridad y bloques en buen estado pueden ser utilizados para reconstruir bloques perdidos. Los bloques consecutivos de un archivo son almacenados en diferentes *DataNodes* del sistema, garantizando la disponibilidad y aumentando la eficiencia. Si un cliente encuentra incongruencias en la suma de comprobación al acceder a un bloque, éste podrá utilizar los bloques de paridad para reconstruirlo.

### 2.3.2. MapReduce

MapReduce es un modelo de programación diseñado por Google [6] para el procesamiento de conjuntos de datos de gran tamaño. Este modelo está inspirado en las operaciones `Map` y `Reduce` de un lenguaje de programación funcional. Un programa escrito en este modelo es paralelizado de manera automática y puede ser ejecutado en múltiples nodos de un *cluster*. El *framework* se encargará de los detalles de las particiones de datos, la calendarización de la ejecución de los programas en múltiples nodos, así como la comunicación entre ellos y fallas en las máquinas. Esto permite que el programador pueda utilizar todos los beneficios del sistema distribuido, enfocándose únicamente en la tarea a realizar y no en los detalles de la paralelización y distribución del procesamiento.

El modelo de programación MapReduce fue diseñado como una abstracción para el procesamiento paralelo y distribuido, que permite realizar una gran cantidad de tareas cotidianas de cómputo, ocultando los detalles de paralelización, distribución de datos y manejo de fallos. Muchas tareas de procesamiento de datos pueden ser simplificadas en operaciones `map` y `reduce`, como las que un lenguaje de programación funcional ofrece; una función `map`, genera pares de clave/valor intermedios; una función `reduce`, realiza una operación conjunta a los valores que comparten una misma clave. El uso de este modelo facilita la paralelización de operaciones en grandes conjuntos de datos, al dividir automáticamente el problema en múltiples tareas, además, ésto permite usar la re-ejecución de tareas como método principal de tolerancia a fallos.

#### 2.3.2.1. Arquitectura de ejecución

Los nodos de un sistema distribuido que ejecutan un programa en un *framework* MapReduce, trabajan bajo un modelo maestro/esclavos. El nodo Maestro se encarga de la asignación de tareas a los nodos. Los demás nodos del sistema, denominados nodos *Worker*, se encargan de la ejecución de las tareas `Map` y `Reduce` asignadas a ellos por el nodo Maestro. La Figura 2.5 muestra un diagrama de la estructura maestro/esclavos para la ejecución de un programa de MapReduce.

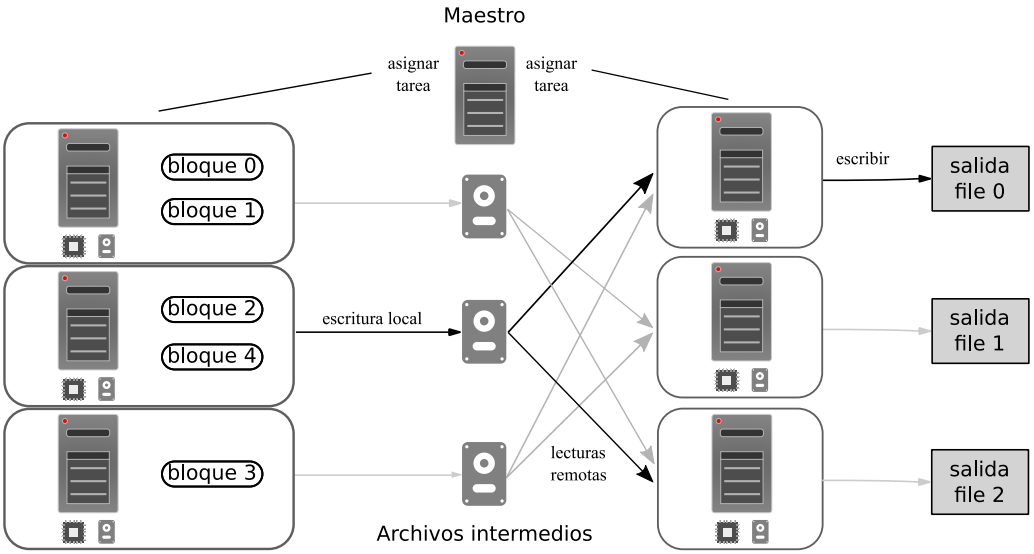


Figura 2.5: Arquitectura maestro/esclavos en la ejecución de un programa MapReduce.

#### 2.3.2.2. Modelo de programación

El usuario de una librería MapReduce expresará las operaciones de cómputo mediante dos funciones principales: la función `Map` y la función `Reduce`. El sistema recibirá un conjunto de pares como entrada y retornará un conjunto de pares clave/valor como salida (Figura 2.6). Los siguientes párrafos describen a detalle las funciones que componen un programa MapReduce.

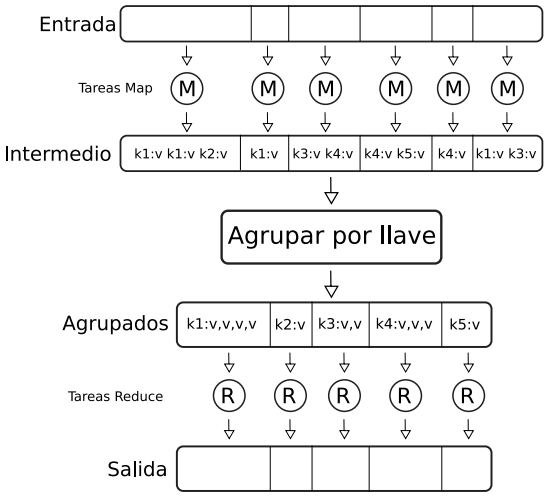


Figura 2.6: Flujo de datos en un programa MapReduce.

(a) `Map`. La función `Map`, escrita por el usuario, divide la entrada en fragmentos y asigna a cada uno de ellos una tarea `Map`, lo que permite que se lleven a cabo de manera paralela y distribuida en distintos nodos del *cluster*. Cada tarea `Map` recibe como entrada el conjunto de pares del fragmento asignado y retorna como salida pares intermedios de elementos clave/valor. Para cada par $(K, V)$ la tarea `Map` usará la función definida para transformar y generar un nuevo par $(K', V')$. Una vez finalizadas las tareas `Map`, el sistema agrupará todos los pares intermedios que contienen una misma clave y los enviará como entrada a la función `Reduce`.

(b) `Reduce`. La función `Reduce`, escrita por el usuario, recibe pares intermedios de valores con una misma clave y los agrega para generar como salida un par $(K, V)$. Cada tarea `Reduce` recibe como entrada el conjunto de pares clave/valor intermedios $(K', V'*)$ que comparten a una misma clave y retorna como salida un nuevo elemento clave/valor. Las tareas `Reduce` son distribuidas a los nodos del *cluster* y el *framework* se encarga de asignar los datos intermedios correspondientes. La etapa `Reduce` generalmente creará un nuevo conjunto de pares de menor tamaño que la entrada.

(c) `Combiner`. La función opcional `Combiner`, permite combinar pares con la misma clave de manera local, antes de ser enviados por la red para realizar la operación `Reduce`. Existen casos donde se pueden encontrar múltiples pares intermedios que comparten una misma clave en un mismo nodo, el utilizar una función `Combiner` permite combinar parcialmente a estos elementos de manera local, reduciendo la cantidad de datos que deberán ser enviados por la red, lo que acelera significativamente algunas de las tareas MapReduce.

Un programa escrito bajo el modelo MapReduce podrá ser ejecutado de forma paralela en múltiples nodos de un *cluster* (Figura 2.7) sin necesidad de ajustes, dado que el *framework* se hará cargo de los detalles de ejecución.

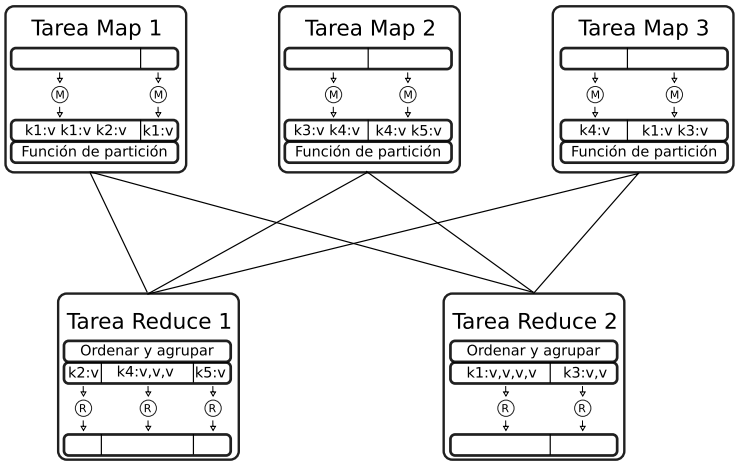


Figura 2.7: Ejecución en paralelo de un programa MapReduce.

De manera general, el proceso de ejecución de un programa MapReduce está definido de la siguiente manera:

1. Los datos de entrada de un programa MapReduce serán divididos en $M$ bloques. El programa entonces iniciará múltiples copias de si mismo en los nodos del *cluster*.

2. Un nodo `Worker`, al que le fue asignada una tarea tipo `Map`, leerá el contenido del respectivo bloque asignado. Éste entonces generará los pares de clave/valor intermedios de acuerdo a la función `Map` especificada por el programa. Los pares intermedios producidos serán almacenados en un *buffer* en memoria.

3. De forma periódica, los pares en el *buffer* serán almacenados de manera local en el nodo. Éstos serán divididos en $R$ regiones definidas por una función de partición. La información de la localización en el almacenamiento local de los pares es enviada al nodo maestro, el cual es responsable de enviarla a los nodos *Worker* asignados a tareas `Reduce`.

4. Cuando un nodo *Worker* asignado con una tarea `Reduce` recibe información por parte del nodo maestro de la localización de bloques, éste procederá a realizar la lectura remota de los bloques contenidos en los nodos *Worker* que realizaron las tareas `Map`. Cuando el nodo ha leído toda la información, éste la ordenará usando sus claves intermedias, agrupando todas las ocurrencias con la misma clave. Cuando la cantidad de información intermedia es demasiado grande, el ordenamiento se realizará de manera externa (en disco).

5. Los nodos *Worker* asignados a tareas `Reduce`, reciben los pares intermedios agrupados por su clave y los pasan como entrada de la función `Reduce` del programa. La salida de la función se concatena a un archivo de salida final de la partición respectiva.

6. Cuando todas las funciones `Map`/`Reduce` han finalizado, el nodo maestro despierta el programa del usuario en el mismo nodo y regresa al código de usuario.

7. La salida de un programa MapReduce serán $R$ archivos de salida (uno por cada tarea `Reduce`).

#### 2.3.2.3. Tolerancia a fallos

En un *cluster* que trabaja bajo una arquitectura maestro/esclavos las fallas pueden presentarse tanto en el nodo maestro como en los nodos esclavos. El *framework* MapReduce provee distintos mecanismos para la tolerancia a fallos para ambos casos. Los siguientes puntos describen estos mecanismos.

(a) Fallas en nodos Worker. El nodo maestro envía señales periódicas a los nodos *Worker* del *cluster*, si un nodo deja de responder a estas señales durante un rango de tiempo, el nodo maestro determinará que el nodo ha fallado. Cualquier tarea `Map` completada por el nodo *Worker* fallido será reiniciada a su estado inicial y la tarea será elegible para su ejecución en otros nodos *Worker*. Reiniciar la tarea es necesario ya que la salida de una tarea `Map` es almacenada localmente en el nodo, por lo que ésta queda inaccesible si el nodo falla.

Cualquier tarea `Map` o `Reduce` en progreso en un nodo *Worker* fallido, será reiniciada a su estado inicial y la tarea será elegible para ser ejecutada en otros nodos *Worker*. Una tarea `Reduce` completada por un nodo *Worker* fallido no será reiniciada, ya que la salida de ésta es guardada en el sistema de almacenamiento del *cluster* (por ejemplo, HDFS), por lo que no existe pérdida de datos.

Cuando una tarea `Map` ha fallado en un nodo $A$, y ha sido asignada a un nuevo nodo $B$, todos los nodos *Worker* que realicen tareas `Reduce` serán notificados y éstos reiniciarán la ejecución de la tarea de así ser necesario, leyendo los datos directamente desde el nodo $B$.

Gracias a esta re-ejecución de tareas, MapReduce es resistente a fallos, ya que el nodo maestro simplemente re-asignará las tareas de nodos fallidos a nuevos nodos para su ejecución, progresando de manera general, hasta que el programa MapReduce sea ejecutado en su totalidad.

(b) Falla en nodo Maestro. Fallas en el nodo maestro pueden ser solventadas guardando copias periódicas de la información contenida por el nodo. Si el nodo maestro falla, una nueva copia puede ser inicializada desde el último punto de control guardado. Para tareas no sensibles, es posible reiniciar completamente la tarea MapReduce cuando el nodo Maestro vuelva a estar disponible. El nodo Maestro, por ser un nodo único en el *cluster*, podrá utilizar técnicas tradicionales para la tolerancia a fallos, garantizando en mayor medida su disponibilidad.

#### 2.3.2.4. Hadoop MapReduce

Hadoop MapReduce [11] es un *framework* que implementa el modelo de programación MapReduce para el desarrollo de aplicaciones para el procesamiento de grandes volúmenes de datos. Hadoop MapReduce permite trabajar en conjunto con el sistema distribuido de archivos de Hadoop (HDFS) y otras herramientas del ecosistema. En Hadoop, los nodos de procesamiento *Worker*, también podrán ser *DataNodes* del HDFS, permitiendo que el procesamiento se lleve a cabo en el sitio donde se encuentran los datos, resultando en una mejora significativa en el desempeño de las aplicaciones.

### 2.3.3. Apache Spark

Apache Spark [3] [17] es un *framework* para el procesamiento distribuido de datos, enfocado en aplicaciones que reutilizan los conjuntos de datos durante su ejecución. La principal característica de Spark, es su capacidad de mantener los datos en la memoria de los nodos del sistema, incrementando la velocidad de procesamiento para este tipo de aplicaciones.

MapReduce ha sido utilizado exitosamente para el procesamiento de datos a gran escala, al proveer un modelo para el procesamiento en paralelo en múltiples nodos de un *cluster*, tolerancia a fallos y balanceo de cargas. Este modelo permite al usuario crear un flujo de datos acíclico, donde los datos de entrada pasan por un conjunto de operaciones, lo que permite al sistema encargarse de la calendarización y fallas sin intervención alguna del usuario. Sin embargo, existen múltiples aplicaciones que no pueden ser expresadas eficientemente mediante este modelo acíclico. Apache Spark fue diseñado para esta clase de aplicaciones, que reutilizan el conjunto de datos durante su ejecución. En Spark es posible cargar y mantener en memoria los conjuntos de datos, permitiendo a las aplicaciones realizar eficientemente operaciones de manera repetida. En particular, algunas de los tipos de tareas que se benefician por el uso de memoria son las tareas iterativas e interactivas, éstas se describen de la siguiente manera:

- **Tareas iterativas.** Tareas que aplican una misma función a un conjunto de datos de manera iterativa, como algoritmos de aprendizaje automático que iteran con el fin de optimizar un parámetro (por ejemplo, descenso de gradiente). Al expresar estas tareas en el modelo MapReduce, cada iteración deberá de cargar los datos desde el almacenamiento, lo que afecta la velocidad de procesamiento.
- **Análisis interactivo.** Tareas donde el usuario utiliza operaciones exploratorias en grandes conjuntos de datos. Estas operaciones expresadas en el modelo MapReduce suelen ser poco eficientes, ya que cada operación deberá ser expresada como una nueva tarea MapReduce, leyendo desde el almacenamiento en cada operación.

El *framework* Apache Spark, provee capacidades de tolerancia a fallos y escalabilidad similares a las de MapReduce. Spark trabaja en una arquitectura maestro/esclavos, el nodo maestro denominado *Driver* se encargará de asignar y coordinar los procesos en el sistema. Los nodos esclavos, denominados *Worker*, se encargarán de llevar a cabo las tareas respectivamente asignadas y de almacenar los datos en memoria RAM o en disco. La arquitectura distribuida de Spark (Figura 2.8) otorga la escalabilidad necesaria para resolver problemas Big Data.

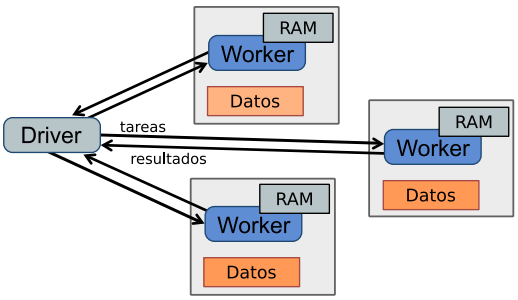


Figura 2.8: Arquitectura maestro/esclavos en Spark.

Además de su componente base para el procesamiento de datos, Apache Spark está conformado por cuatro componentes principales para el procesamiento de datos:

- Spark SQL. Permite operaciones tipo SQL en datos representados en tablas, similares a una tabla en una base de datos relacional.
- Spark Streaming. Permite el procesamiento de datos en *streaming*, esto es, el procesamiento de un flujo continuo de datos.
- ML. Un módulo conformado por múltiples algoritmos distribuidos de aprendizaje de máquina. Cuenta con algoritmos supervisados, no supervisados y herramientas para realizar transformaciones, evaluación de modelos y para la creación de *pipelines*.
- GraphX. Un módulo de procesamiento distribuido de grafos. Permite representar y procesar datos en forma de grafos. Incluye distintos algoritmos distribuidos para el análisis de grafos.

#### 2.3.3.1. Spark RDD

Los *Resilient Distributed Datasets* [16] (RDDs), son la principal abstracción distribuida en memoria de Apache Spark, éstos forman parte de la base de Spark para el procesamiento distribuido de datos. Un RDD representa una colección inmutable de objetos particionados en múltiples máquinas de un sistema distribuido. Los RDD proveen mecanismos de tolerancia a fallos, que permiten que las particiones puedan ser reconstruidas en caso de que se presenten fallas en los nodos que las contienen. Los usuarios de un RDD pueden invocar una operación de `cache` de manera explicita, lo que ocasiona que las particiones que lo conforman se almacenen en la memoria RAM de las máquinas del sistema distribuido. Esto evita que los datos deban ser cargados nuevamente desde disco, agilizando el procesamiento gracias a la baja latencia y mayor ancho de banda de la memoria RAM.

**Operaciones.** Spark proporciona la abstracción RDD a los programadores mediante una API. El conjunto de datos en un RDD se representa como un objeto inmutable, la API proporciona métodos para aplicar transformaciones y acciones para crear y procesar RDDs. Inicialmente, el programador define uno o más RDD mediante transformaciones a datos que se encuentran en un medio de almacenamiento o a RDDs existentes. Después, el usuario puede hacer uso de los RDD definidos mediante acciones, las cuales son operaciones que inician el plan de trabajo generado por las transformaciones para regresar valores de salida al programa de usuario o para exportarlos a un sistema de almacenamiento.

- Transformaciones. Un RDD, al ser una colección de objetos inmutable, únicamente puede ser creado a partir de operaciones deterministas aplicadas a datos en un sistema de almacenamiento o a otros RDDs. Ejemplos de estas transformaciones incluyen `Map`, `filter` y `join`.

- Acciones. Un RDD es computado de manera pasiva, de tal forma que múltiples transformaciones puedan ser concatenadas formando un plan de ejecución. Las acciones son operaciones que inician el plan de ejecución generado para un RDD, devolviendo valores al programa o escribiendo resultados en un sistema de almacenamiento. Algunos ejemplos de acciones son las operaciones `count`, `collect` y `save`.

- Persistencia. Las operaciones de persistencia de un RDD, posibilitan al usuario indicar una estrategia de persistencia, el usuario podrá seleccionar la mejor estrategia de almacenamiento para los datos de cada RDD. Por ejemplo, para un RDD que será reutilizado, el almacenamiento en memoria será la mejor estrategia para alcanzar la mayor eficiencia. En Spark, la operación `cache` indicará al sistema que los datos del RDD deben colocarse memoria. Si se excede la capacidad de memoria RAM en las máquinas del sistema, las particiones excedentes se almacenarán en disco. Otros tipos de persistencia también podrán ser seleccionados, el usuario podrá elegir que un RDD únicamente se almacene en disco, que sea replicado en múltiples máquinas del sistema o podrá elegir prioridades para la persistencia de diferentes RDDs.

- Partición. Existen operaciones de partición que pueden ser utilizadas en RDDs para elegir como son particionados los datos de un RDD. El usuario podrá ingresar una *key* para llevar a cabo el particionado. Las operaciones de partición permiten optimizar la localización de los datos, logrando que operaciones que harán uso de éstos sean lo más eficientes posible.

**Tolerancia a fallos.** La tolerancia a fallos de un RDD se logra mediante el concepto de linaje. En el caso de que una partición de un RDD sea perdida, existirá la información suficiente de cómo fue generada desde otros RDD para que ésta pueda ser reconstruida. Este método permite que la recuperación de datos se agilice, evitando métodos más costosos, como lo es la replicación de datos. Además, gracias a la inmutabilidad de los RDD, el sistema podrá mitigar los efectos de nodos lentos iniciando copias de las tareas lentas, sin afectar a otras tareas, ya que no se comparte información entre éstas.

**Expresividad de los RDD.** Los RDD en Spark ofrecen una interfaz para realizar transformaciones que realizan una misma operación a un conjunto de datos. Aunque esto puede parecer una limitación, los RDDs son lo suficientemente expresivos para desarrollar aplicaciones que aplican una misma operación a todos los datos de una entrada. Los RDD pueden representar de manera eficiente múltiples modelos de programación distribuida, tales como MapReduce, SQL, Pregel, DryadLINQ, *Iterative* MapReduce y *Batched Stream Processing*. Los RDDs pueden producir las mismas salidas que programas escritos en los modelos mencionados, incorporando las optimizaciones que estos *frameworks* realizan.

#### 2.3.3.2. Spark SQL

Spark SQL [4], es un módulo de Apache Spark que integra capacidades de procesamiento relacionales al paradigma funcional de Spark. Spark SQL mezcla el modelo de consultas relacionales con el paradigma por procedimientos, logrando una interfaz dinámica e intuitiva que permite a los usuarios expresar de manera natural las operaciones a realizar en las distintas etapas del flujo de trabajo.

El módulo SQL de Spark se caracteriza por proveer una interfaz de programación amigable a los usuarios, manteniendo y mejorando las propiedades de las interfaces tradicionales (RDD, MapReduce). Spark SQL expone interfaces tipo SQL, que pueden ser accedidas por el usuario mediante conectores ODBC/JDBC, la línea de comandos interactiva y la API DataFrame integrada en los lenguajes de programación soportados por Spark.

**DataFrame.** La principal abstracción en la API de Spark SQL son los DataFrame, una colección distribuida de filas que comparten un mismo esquema, equivalente a una tabla de una base de datos relacional. Un DataFrame puede ser manipulado usando operaciones relacionales (`groupBy`, `where`), de manera similar a dataframes existentes en otras APIs (R, Python, etc.) o usando operaciones nativas de una colección distribuida (`map`), similares a las de un RDD. Los DataFrame en Spark son evaluados de manera pasiva, esto es, cada objeto DataFrame representa un plan lógico para computar un conjunto de datos, pero la ejecución de éste no ocurre hasta que se realiza una operación especial de salida (por ejemplo la operación `save`). Esta evaluación pasiva permite que el plan lógico generado pueda ser optimizado antes de ser iniciado.

Un DataFrame puede ser creado a partir de una fuente de datos externa o desde DataFrames/RDDs existentes. Además, existe interoperabilidad entre el formato RDD y DataFrame, lo que permite a los usuarios elegir la representación que mejor se adapte a las necesidades de la tarea.

Los DataFrames utilizan de manera automática un formato de almacenamiento columnar, que permite sean más compactos que los objetos nativos. Además, al igual que los RDD, la operación de `cache()` puede ser aplicada a un DataFrame, lo que permitirá que los datos de éste sean colocados en memoria, y gracias a su formato, el uso de memoria se reduzca significativamente mediante métodos de compresión.

**UDF (User Defined Functions).** Los DataFrames de Spark SQL permiten el registro de funciones definidas por el usuario pasando funciones en Scala/Java/Python, éstas pueden utilizar la API de Spark internamente. Las UDF son útiles para proveer funcionalidades avanzadas al lenguaje SQL, como comúnmente se hace en sistemas de bases de datos tradicionales. En Spark SQL, el registro de UDFs se realiza en el código del programa, facilitando su uso. Una vez registradas pueden ser utilizadas por operaciones en DataFrames, así como en conectores JDBC/ODBC.

**Integración con el módulo ML de Spark.** Los DataFrames de Spark, se integran con el módulo de aprendizaje máquina de Spark (MLlib), mediante una API que utiliza a los DataFrames de manera nativa (recibe y retorna DataFrames). Esta API se basa en el concepto de *pipelines*, usados en otras librerías de aprendizaje máquina (por ejemplo, scikit-learn). Un *pipeline* es una secuencia de transformaciones aplicadas a los datos, donde cada etapa genera un conjunto de datos que serán recibidos por la siguiente etapa. La representación columnar de los DataFrames es ideal para la creación de *pipelines* de aprendizaje máquina, ya que cada columna representará una característica de los datos. Los algoritmos que forman parte de un *pipeline* recibirán los nombres de las columnas de entrada y salida, permitiendo mantener los datos originales para cada entrada.

**Optimizador Catalyst.** Catalyst es un optimizador que forma parte del módulo SQL de Spark, éste evalúa y optimiza el conjunto de operaciones del plan lógico de un DataFrame. Además, Catalyst puede ser utilizado para extender Spark SQL, por ejemplo, mediante la creación de métodos para leer nuevas fuentes de datos, reglas de optimización y tipos de datos definidos por el usuario (*User Defined Type*).

### 2.3.4. Otras herramientas del ecosistema de Hadoop

Además de las herramientas base de Hadoop, el ecosistema de Hadoop consta de un conjunto de herramientas de código libre y comerciales especializadas para cubrir las necesidades de problemas tipo Big Data. Algunas de las herramientas principales del ecosistema de Hadoop se listan a continuación:

- YARN. *Yet Another Resource Negotiator* es una herramienta para la administración, monitoreo y control de la seguridad de los recursos que utilizan las aplicaciones de un *cluster* de Hadoop.
- Ambari. Es una herramienta para el monitoreo, configuración, instalación y control de herramientas del ecosistema de Hadoop, así como el *cluster* de Hadoop mediante una interfaz web.
- HBase. Es una base de datos NoSQL columnar que utiliza el sistema de archivos distribuido de Hadoop para el almacenamiento de los datos.
- Apache Hive. Es un motor SQL para leer, escribir y administrar grandes volúmenes de datos almacenados en Hadoop HDFS.
- Pig. Es un lenguaje de scripting para transformar el formato de conjuntos de datos en Hadoop.
- Sqoop. Es una herramienta para importar datos desde bases de datos relacionales hacia Hadoop o exportar datos desde Hadoop hacia bases de datos relacionales.
- Flume. Es una herramienta para la importación en *streaming* de datos no estructurados a Hadoop.
- Kafka. Es una herramienta para el manejo de datos en *streaming*.
- Oozie. Es una herramienta para la definición y calendarización de flujos de trabajo compuestos por aplicaciones que corren en Hadoop mediante Grafos aciclicos dirigidos.
- Zeppelin. Provee un servicio de notebooks para el desarrollo y visualización de aplicaciones en herramientas del ecosistema Hadoop.
- Zookeeper. Es un coordinador de herramientas del ecosistema de Hadoop.

## 2.4. Bases de datos NoSQL

Las bases de datos NoSQL (*Not only* SQL), permiten el almacenamiento escalable y flexible de datos mediante el uso de sistemas distribuidos. A diferencia de las bases de datos relacionales, las bases de datos NoSQL utilizan una variedad de estructuras para almacenar, acceder y administrar los datos; la Tabla 2.1 describe los modelos de datos principales utilizados en bases de datos NoSQL.

Tabla 2.1: Principales modelos de datos en NoSQL.

| Modelo de datos | Descripción |
|---|---|
| Clave/valor | Para este modelo los datos son representados como un conjunto de pares clave/valor. Este tipo de modelo es utilizado como base de otros modelos más complejos. |
| Columnar | Las bases de datos columnares utilizan tablas, filas y columnas para almacenar los datos. A diferencia de bases de datos relacionales, los nombres y formatos de las columnas podrán variar en las filas de una misma tabla. |
| Documentos | Estas bases de datos almacenan la información en forma de documentos. Los formatos comúnmente utilizados incluyen XML, JSON, PDF, etc. |
| Grafos | En éstas se utilizan conjuntos de vértices y aristas para representar y almacenar los datos. Son útiles para expresar relaciones entres los datos almacenados. |

De acuerdo al teorema CAP, es imposible que una base de datos pueda proveer más de dos de las siguientes propiedades:

- **Consistencia (Consistency).** Todos los clientes siempre reciben la información más reciente o un error.
- **Disponibilidad (Availability).** Los clientes siempre reciben una respuesta a sus peticiones, sin importar si éstas contienen la información más reciente.
- **Tolerancia al particionado (Partition tolerance).** El sistema continuará funcionando sin importar si existen fallas en los nodos o la red.

Dado que es esencial que cualquier base de datos distribuida proporcione tolerancia al particionado, esto deja dos posibilidades para las bases de datos NoSQL. Algunas bases de datos NoSQL sacrifican la consistencia en favor de la tolerancia al particionado y la disponibilidad, éstas utilizan el concepto de consistencia eventual para describir la forma en la que los cambios efectuados en los datos se propagan de manera ”eventual” a todos los nodos del sistema, por lo que es posible que no todos los clientes lean la información más reciente, pero siempre recibirán una respuesta. Las bases de datos que sacrifican la disponibilidad en cambio de la tolerancia al particionado y la consistencia, aseguran que la información leída por los clientes siempre está actualizada, tal comportamiento puede afectar el rendimiento de las lecturas, ya que los clientes deben esperar a que la información se propague a todos los nodos o recibirán un error.
Las bases de datos NoSQL son utilizadas para problemas donde se tienen grandes volúmenes de datos, se requieren cortos tiempos de respuesta o cuando es necesario un modelo flexible de datos. Las propiedades de las bases de datos NoSQL las convierten en una herramienta indispensable para una solución Big Data.

Capítulo 3

# Reducción de bases de datos de ADN

## 3.1. Introducción

El ácido desoxirribonucleico (ADN) es un tipo de molécula que contiene la información genética que proporciona instrucciones para el desarrollo, funcionamiento, crecimiento y reproducción de todos los organismos conocidos y algunos tipos de virus. Una cadena de ADN está conformada por unidades llamadas nucleótidos. Cada nucleótido en el ADN contiene una de cuatro posibles bases nitrogenadas: adenina (A), guanina (G), citosina (C) o timina (T).

La secuenciación del ADN es el conjunto de procesos utilizados para determinar la secuencia de nucleótidos que forman a una muestra de ADN. Los avances tecnológicos han permitido que las técnicas de secuenciación sean cada vez más rápidas y a su vez menos costosas. El volumen de datos generados por instrumentos de secuenciación masiva de ADN es cada día más grande, lo que ha ocasionado que métodos tradicionales se vuelvan computacionalmente infactibles.

Múltiples herramientas de análisis de secuencias utilizadas en distintas ramas de la genómica hacen uso de bases de datos (BDs) de secuencias como parte de su metodología de análisis. El número de secuencias disponibles en las BDs de referencia ha presentado un crecimiento exponencial debido a la aparición de nuevas tecnologías de secuenciación masiva. Un ejemplo es GenBank, una de las BDs más utilizada alrededor del mundo. Actualmente está conformada por más de doscientos millones de secuencias y trescientos mil millones de pares de bases, lo que se traduce a 953 GB de datos. Además, se reporta que GenBank duplica su tamaño cada 18 meses, lo que implica una gran velocidad en la que se generan los datos.

Por otro lado, la complejidad temporal (tiempo de ejecución) y espacial (cantidad de memoria) de muchas de las herramientas existentes para el análisis de secuencias las vuelve poco eficientes al trabajar con grandes volúmenes de datos. Las propiedades de las BDs de secuencias y la complejidad de las herramientas que las utilizan generan grandes retos computacionales con gran relevancia en múltiples áreas de investigación. Una arquitectura Big Data puede proveer la escalabilidad necesaria para almacenar y procesar estos grandes volúmenes de datos de manera eficiente y confiable.

En este trabajo, se desarrolló una aplicación para la reducción de BDs de secuencias usando una arquitectura Big Data. Ésta utiliza las capacidades de almacenamiento distribuido y procesamiento paralelo de la arquitectura Big Data, lo que la vuelve altamente escalable.

## 3.2. Descripción del problema

Múltiples herramientas para el análisis de secuencias utilizan BDs de referencia como parte de su metodología. La complejidad temporal y espacial de los algoritmos utilizados por muchas de las herramientas de análisis, ocasiona que éstas sean inaplicables para grandes cantidades de datos. Nuevos métodos de análisis han intentado mejorar la velocidad de procesamiento al sacrificar la precisión o exactitud de los análisis en cambio de acelerar el procesamiento. Existe una necesidad de desarrollar métodos computacionales que permitan mejorar la eficiencia mediante el uso de técnicas escalables para el almacenamiento, procesamiento y análisis de secuencias.

Las BDs de referencia pueden ser altamente redundantes, ya que regularmente contienen múltiples secuencias con alta similitud. Esto ocasiona que su tamaño aumente de manera considerable, afectando la velocidad de procesamiento de las herramientas que las utilizan. La redundancia en las BDs también puede llevar a errores en los análisis. Por ejemplo, algunos métodos de análisis de similitud de secuencias aceleran sus búsquedas al asignar a la primera secuencia encontrada en la BD, sin explorar otras asignaciones posibles. Esto puede conducir a los investigadores a llegar a conclusiones incorrectas.

Existen diferentes motivos que explican la alta redundancia en las BDs de referencia. Es común que éstas contengan múltiples secuencias con alta similitud debido a que representan a organismos o microorganismos de una misma especie. Por ejemplo, secuencias generadas desde diferentes cepas de una especie. Además, las relaciones filogenéticas entre los diferentes organismos o microorganismos representados en las BDs también son susceptibles a altos porcentajes de redundancia. Esto es debido a las regiones conservadas en las secuencias de organismos o microorganismos que divergieron de un mismo ancestro, lo cual expresa la relación de similitud taxonómica (ordenación jerárquica) que hay entre éstos.

## 3.3. Objetivos

- Desarrollar un algoritmo escalable utilizando una arquitectura Big Data para reducir el tamaño de BDs de secuencias de ADN.
- Generar nuevas bases de datos que puedan ser utilizadas en herramientas de análisis existentes.
- Obtener bases de datos reducidas que contengan la información taxonómica en sus metadatos, minimicen la pérdida de información y mejoraren la velocidad de los análisis.

## 3.4. Metodología

### 3.4.1. Reducciones con sentido biológico usando información taxonómica

Con la finalidad de obtener BDs que utilicen menor espacio y reducir la pérdida de información, se desarrolló un algoritmo distribuido (Figura 3.1) que utiliza la información taxonómica de las secuencias para agrupar las secuencias que provienen de un ancestro común y eliminar regiones o secuencias redundantes en dichos grupos. La relación filogenética entre las secuencias de estos grupos permite aumentar la probabilidad de encontrar información similar.

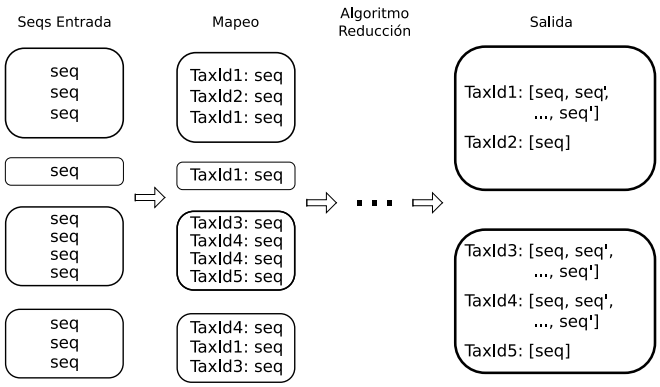


Figura 3.1: Flujo de datos del algoritmo de reducción.

El algoritmo desarrollado utiliza alineamientos globales para encontrar regiones similares en las secuencias que pertenecen a una misma categoría taxonómica (Figura 3.2). Las sub-regiones similares encontradas son almacenadas una sola vez y son catalogadas como información redundante de la categoría taxonómica correspondiente. Las regiones únicas de cada secuencia son almacenadas en nuevas sub-secuencias identificadas por su secuencia de origen, evitando la pérdida de información.

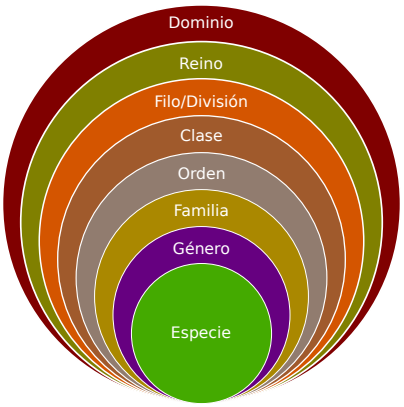


Figura 3.2: Categorías taxonómicas principales.

### 3.4.2. Información única y redundante en las secuencias

Para encontrar las regiones redundantes y únicas de las secuencias, el algoritmo lleva a cabo de forma paralela alineamientos por pares para encontrar su alineamiento global óptimo. Las tareas de alineamiento pueden llevarse a cabo de forma paralela automáticamente, al expresar el programa con un modelo de procesamiento distribuido. El resultado de cada alineamiento es utilizado para encontrar tanto las regiones similares, como las regiones distintas de las secuencias. Una ventana de tamaño y margen de identidad configurable es utilizada para recorrer la cadena de salida del algoritmo de alineamiento y hallar tales regiones. La Figura 3.3 muestra la salida de un algoritmo de alineamiento global, donde se pueden observar regiones similares y únicas de dos secuencias.

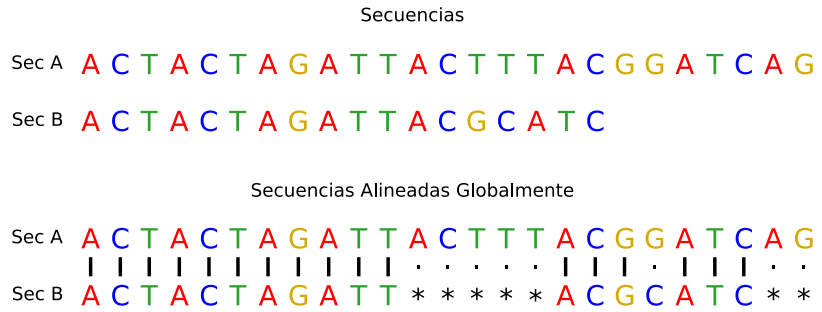


Figura 3.3: Ejemplo de la entrada y salida al aplicar un algoritmo de alineamiento global para un par de secuencias.

Para reducir cada par de secuencias, el algoritmo de Myers and Miller [12], una modificación del algoritmo tradicional Needleman-Wunsch [13], fue utilizado para hallar el alineamiento global óptimo de las secuencias en espacio lineal. A diferencia de algoritmos de alineamiento tradicionales, el algoritmo de Myers and Miller únicamente utiliza espacio lineal para encontrar el alineamiento global óptimo. Esto permite realizar un mayor número de alineamientos en paralelo sin saturar la memoria del sistema. Por otro lado, la capacidad de ejecutar en paralelo y de manera independiente los alineamientos de cada par de secuencias acelera la velocidad general del algoritmo.

### 3.4.3. Algoritmo distribuido

El framework de procesamiento distribuido Apache Spark fue utilizado para desarrollar este algoritmo, ya que este provee la escalabilidad y eficiencia necesaria para esta clase de aplicaciones. Los siguientes párrafos describen en detalle las etapas de procesamiento de los datos del algoritmo usando funciones de Spark.

En primera instancia, el algoritmo recibe como entrada una BD de secuencias. Los datos de secuencias son cargados a un RDD para ser procesados. Después, el algoritmo añade a cada secuencia su información taxonómica, es decir, mapea a cada secuencia con su árbol taxonómico, generando un RDD de pares clave-valor. En la siguiente etapa, se selecciona como clave el identificador de la categoría taxonómica seleccionada como base para la reducción.

El proceso de reducción utiliza la función `combineByKey` de Spark para combinar y reducir las secuencias con la misma clave, en este caso, el identificador de la categoría taxonómica correspondiente. Por ejemplo, si en la BD se tienen las secuencias de los virus de Influenza A e Influenza D, y se selecciona la familia como la categoría taxonómica para realizar la reducción, entonces el identificador taxonómico correspondiente a la familia (Orthomyxoviridae) es asignado como clave en ambas secuencias. La Figura 3.4 muestra un ejemplo del árbol taxonómico que contiene al virus de Influenza A e Influenza D a nivel de especie, ambos pertenecientes a la familia Orthomyxoviridae.

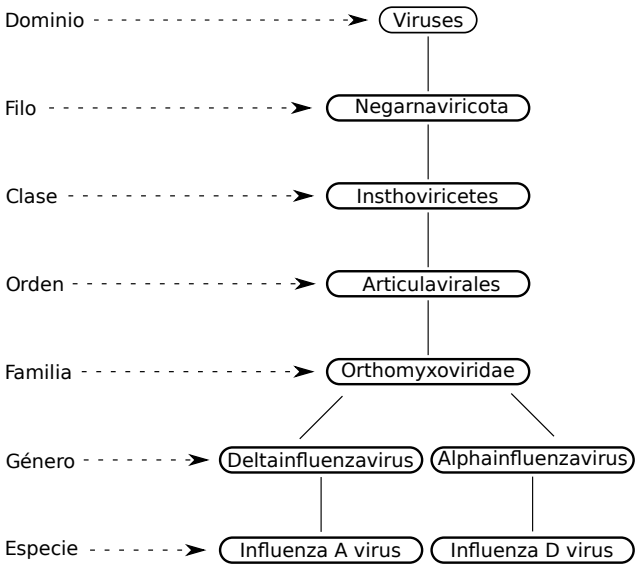


Figura 3.4: Árbol taxonómico que contiene dos especies del virus de Influenza

La función `combineByKey` recibe como parámetros tres funciones de agregación personalizadas `createCombiner`, `mergeValue` y `mergeCombiners` para combinar los elementos de cada clave de un RDD$[(K, V)]$ y generar un nuevo RDD$[(K, C)]$.

La función `createCombiner` recibe un valor $V$ y lo transforma en $C$ (*combiner*). Para esta aplicación, un par (identificador, secuencia) es transformado en una lista con un elemento que contiene el mismo par (Figura 3.5).

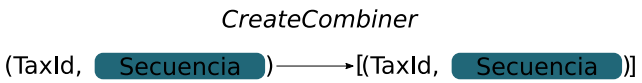


Figura 3.5: Función `createCombiner` en el algoritmo de reducción de secuencias.

La función `mergeValue` recibe un valor $V$ y un *combiner* $C$, y retorna como salida un nuevo $C$ que integra a $V$. Para esta aplicación, se recibe un par (identificador, secuencia) y una lista de secuencias reducidas y, como salida, devuelve una nueva lista con la información única de la secuencia recibida como entrada (Figura 3.6).

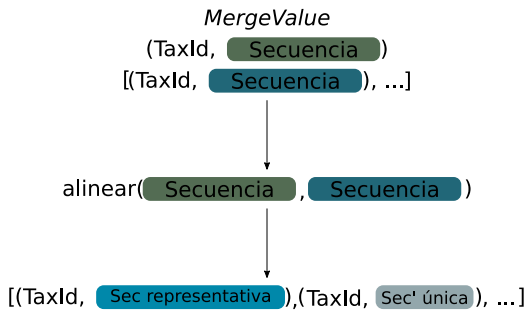


Figura 3.6: Función `mergeValue` en el algoritmo de reducción de secuencias.

La función `mergeCombiners`, recibe como entrada dos *combiners* $C$ y genera un solo *combiner*. Para esta aplicación, la función recibe dos listas de secuencias reducidas, alinea las secuencias representativas para eliminar información redundante y almacena las secuencias únicas en la lista (Figura 3.7).

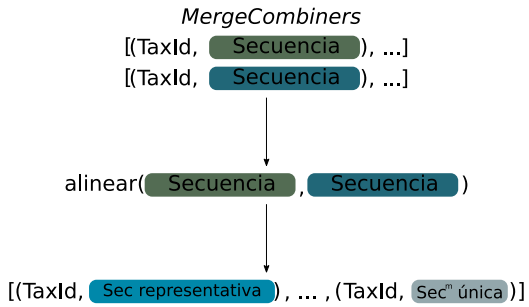


Figura 3.7: Función `mergeCombiners` en el algoritmo de reducción de secuencias.

Como resultado de la función `combineByKey`, se combinan las secuencias con un mismo identificador a una secuencia consenso y un conjunto de sub-secuencias con información única, lo que evita la pérdida de información y reduce el tamaño final de la base de datos. Además, cada secuencia y sub-secuencia generada contiene en sus meta-datos información relevante sobre su taxonomía y la secuencia original que se utilizó para generarla. Para ejemplificar el comportamiento del algoritmo, si la aplicación recibe como entrada una BD de secuencias que contiene $n$ secuencias de 100 géneros distintos y se selecciona al género como la categoría taxonómica para la reducción, entonces la BD reducida generada estará conformada por 100 secuencias representativas, una por cada género, que contienen la información conservada a dicho nivel. Adicionalmente, la nueva BD contendrá $m$ secuencias que representan la información única de cada secuencia en la BD original.

La categoría taxonómica (especie, genero, familia, etc.) que será utilizada durante la fase de reducción es un parámetro configurable del algoritmo. La elección de éste tendrá repercusiones sobre el tamaño y la calidad de la base de datos reducida que se obtenga al ejecutar el algoritmo.

### 3.4.4. Secuencias después de la reducción

El algoritmo de reducción genera una secuencia consenso para cada identificador de la categoría taxonómica elegida y, además, conserva la información no redundante de las secuencias en múltiples sub-secuencias. Los meta-datos de las secuencias reducidas contienen la información taxonómica de las secuencias que fueron utilizadas por el algoritmo para generarlas, lo que permite identificarlas de manera inequívoca. Si una herramienta analítica utiliza una BD reducida generada por este algoritmo, los meta-datos de cada secuencia permitirán que éstas puedan ser identificadas como sub-secuencia única del nivel taxonómico seleccionado o secuencia consenso del mismo nivel, lo que permite evitar posibles errores de interpretación. En la Figura 3.8 se muestra un diagrama de las secuencias reducidas generadas después de un alineamiento entre dos secuencias de una misma categoría.

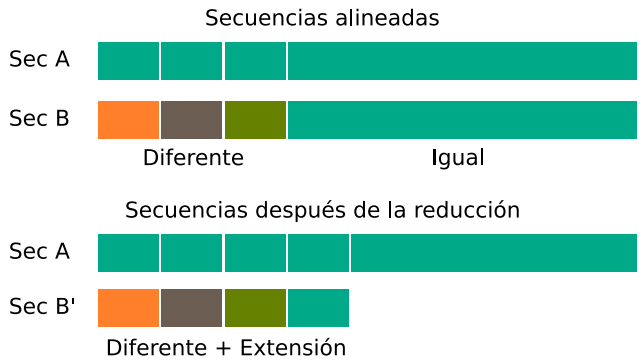


Figura 3.8: Ejemplo de la salida del algoritmo de reducción.

## 3.5. Resultados

Para mostrar las capacidades del algoritmo, se seleccionaron tres BDs de secuencias de genomas completos de ADN pertenecientes a tres familias virales. Se utilizó el algoritmo desarrollado para generar tres nuevas BDs de tamaño reducido y se comparó contra el tamaño de las BDs originales. Además, se utilizó Bowtie2 [10], una popular herramienta para el análisis de secuencias, para mostrar la capacidad de utilizar las BDs reducidas en herramientas existentes y demostrar el margen de pérdida de información al utilizar el algoritmo de reducción. El análisis comparativo de las BDs antes y después de la reducción con BowTie2 se llevó a cabo usando datos generados con Grinder [1].

### 3.5.1. Familia Parvoviridae

La BD de la familia Parvoriridae consta de 784 secuencias de genomas completos de 93 especies distintas, en total, las secuencias están compuestas por $3,920,391$ pares de bases. Después de aplicar el algoritmo de reducción, se generó una nueva BD con $1,676,896$ pares de bases, lo que se traduce a un factor de compresión de 2.33, esto puede visualizarse en la Figura 3.9.

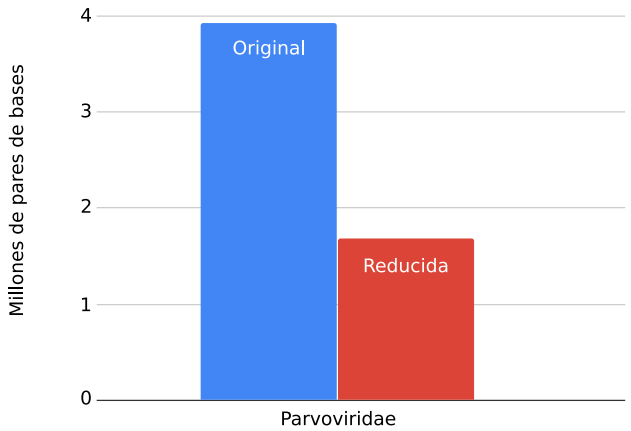


Figura 3.9: Comparativa del tamaño de la base de datos original contra la reducida para la familia Parvoviridae.

La Tabla 3.1 muestra las diferencias en los resultados de análisis obtenidos con BowTie2 al utilizar la BD original y la BD reducida para las secuencias de la familia. En ésta se puede observar que BowTie2 logró alinear $2,498$ secuencias con la BD reducida, esto es, una secuencia menos que con la base de datos original. Además, el porcentaje de alineamientos con un solo mapeo aumentó de 3.84 % a 28.76 % al utilizar la BD reducida, ya que se eliminó información redundante.

Tabla 3.1: Tabla comparativa de alineamientos encontrados usando BowTie2 con la base de datos original contra la reducida para la familia Parvoviridae.

| Aligned | Parvoviridae | |
|---|---|---|
| | Full | Reduced |
| 0 times | 1/2500 reads | 2/2500 reads |
| 1 time | 96 (3.84 %) | 719 (28.76 %) |
| >1 times | 2403 (96.12 %) | 1779 (71.16 %) |

### 3.5.2. Familia Virgaviridae

La BD de genomas completos de la familia Virgaviridae consta de 334 secuencias provenientes de 43 especies. En total, la BD contiene 2, 117, 792 pares de bases. Después de aplicar el algoritmo de reducción, se generó una nueva BD con 581, 691 pares de bases, lo que indica un factor de compresión de 3.64, estos resultados pueden observarse en la Figura 3.10.

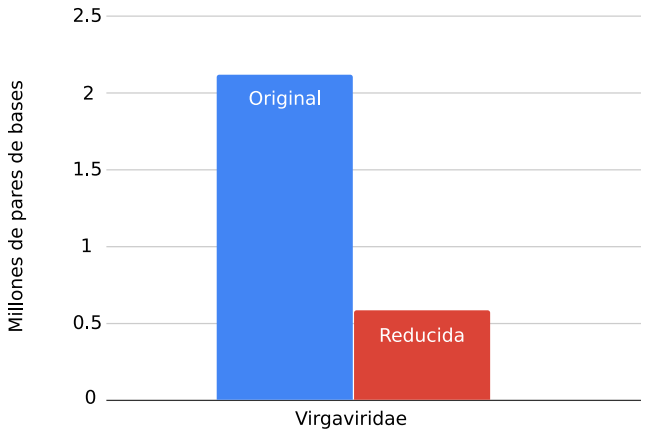


Figura 3.10: Comparativa del tamaño de la base de datos original contra la reducida para la familia Virgaviridae.

Para los resultados del análisis con BowTie2, la Tabla 3.2 compara los resultados obtenidos usando la BD original y la BD reducida. BowTie2 logró alinear 2, 499 secuencias con la BD reducida, lo que equivale al 99.96 % de la información. El porcentaje de alineamientos con un solo mapeo aumentó de 3.56 % a 42.56 % al utilizar la BD reducida.

Tabla 3.2: Tabla comparativa de alineamientos encontrados usando BowTie2 con la base de datos original contra la reducida para la familia Virgaviridae.

| Aligned | Virgaviridae | |
|---|---|---|
| | Full | Reduced |
| 0 times | 0/2500 reads | 1/2500 reads |
| 1 time | 89 (3.56 %) | 1064 (42.56 %) |
| >1 times | 2411 (96.44 %) | 1435 (57.4 %) |

### 3.5.3. Familia Myoviridae

La BD de la familia Myoviridae contiene 1,772 secuencias de genomas completos de 878 especies y en total se constituye por 243,188,662 pares de bases. El algoritmo redujo la BD con factor de compresión de 1.97. La nueva BD contiene 123,227,766 pares de bases. El factor de compresión para la BD de la familia Myoviridae puede observarse en la Figura 3.11.

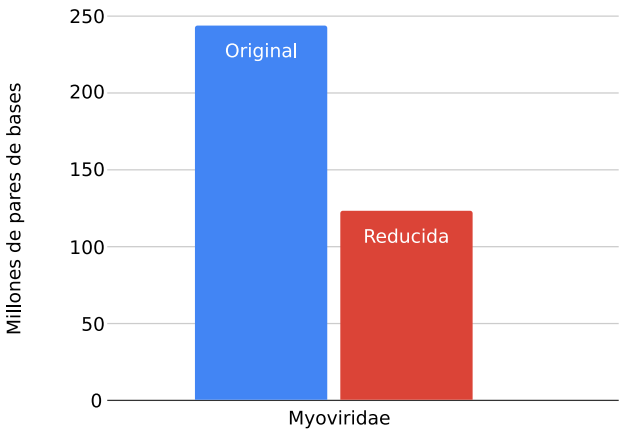


Figura 3.11: Comparativa del tamaño de la base de datos original contra la reducida para la familia Myoviridae.

La Tabla 3.3 muestra los resultados del análisis obtenidos con BowTie2 para la BD original y la BD reducida. Para esta familia, BowTie2 logró alinear 4,996 de 5,000 secuencias simuladas al utilizar la BD reducida. El porcentaje de alineamientos con un solo mapeo aumentó de 4.94 % a 32.56 % al utilizar la BD reducida.

Tabla 3.3: Tabla comparativa de alineamientos encontrados usando BowTie2 con la base de datos original contra la reducida para la familia Myoviridae.

| Aligned | Myoviridae | |
|---|---|---|
| | Full | Reduced |
| 0 times | 0/5000 reads | 4/5000 reads |
| 1 time | 247 (4.94 %) | 1628 (32.56 %) |
| >1 times | 4753 (95.06 %) | 3368 (67.36 %) |

## 3.6. Conclusiones

El algoritmo desarrollado fue utilizado para reducir el tamaño de tres BDs de secuencias de virus. Se utilizó el nivel especie como base para las reducciones realizadas. Se deberá explorar la posibilidad de utilizar niveles taxonómicos más altos como base para las reducciones. Las BDs reducidas mostraron bajos porcentajes de error al ser utilizadas en una herramienta existente para el análisis de secuencias. Los resultados de los análisis muestran que se presentó una gran eliminación de la redundancia al utilizar las BDs generadas con el algoritmo desarrollado, esto es reflejado en el número de pares de bases en la BD después de realizar la reducción.

La escalabilidad proporcionada por el algoritmo y la arquitectura distribuida posibilitarán la reducción de BDs de gran tamaño, lo cual no sería factible mediante una metodología de procesamiento tradicional. Las BDs reducidas podrán ser de gran utilidad en múltiples ámbitos de la genómica, ya que utilizarlas permitirá acelerar la velocidad de análisis y al mismo tiempo reducir errores de asignación. Se espera que la capacidad de utilizar BDs reducidas en herramientas existentes facilite su adopción.

Ya que el algoritmo de alineamiento utilizado está diseñado para recibir como entrada cualquier tipo de cadenas, será posible utilizar el algoritmo de reducción en BDs de proteínas. Será necesario mayor experimentación para confirmar la utilidad del algoritmo para estos tipos de secuencias.

Aún cuando la escalabilidad de la aplicación permite acelerar en gran medida el procesamiento, las BDs con secuencias de gran longitud se verán limitadas por la complejidad temporal del algoritmo de alineamiento global. El cuello de botella podrá presentarse en las tareas paralelas que utilizan alineamientos para identificar las regiones redundantes entre pares de secuencias. Para secuencias largas, una posible alternativa es el uso de algoritmos de alineamiento no exactos, que sacrifican la calidad del alineamiento para reducir la complejidad temporal del algoritmo. Las repercusiones en el tamaño y calidad de las BDs reducidas al utilizar alineamientos no óptimos deberán ser analizadas.

Capítulo 4

# Pronóstico del consumo eléctrico de hogares con datos de medidores inteligentes

## 4.1. Introducción

La red eléctrica es un sistema complejo conformado por múltiples equipos, dispositivos y tecnologías trabajando en conjunto con el objetivo de llevar energía eléctrica a los consumidores desde las fuentes generadoras. Tradicionalmente, la red eléctrica está compuesta por tres componentes principales. El primer componente, la generación, consta de las fuentes de generación de energía, que típicamente son centrales de gran tamaño y se encuentran lejos del consumidor, ejemplos de éstas son las centrales nucleares y centrales termoeléctricas. El segundo componente, la transmisión, se conforma de líneas que permiten transportar por grandes distancias la energía generada desde las fuentes. Por último, la distribución, se encarga de llevar la energía a los consumidores. En cada uno de estos componentes están involucrados múltiples sistemas y dispositivos que mantienen a la red en operación. En los próximos años, se espera que se integren a gran escala fuentes de energía renovable, fuentes de generación distribuida, sistemas de almacenamiento de energía y vehículos eléctricos, así como nuevos sistemas, dispositivos, y tecnologías, lo que transformará a la red eléctrica en un sistema mucho más dinámico y complejo.

Los avances tecnológicos y las necesidades actuales están impulsando a las empresas de la industria eléctrica a modernizar la red, creando una red eléctrica inteligente que incorpora nuevos dispositivos de control, sensores y sistemas que permiten mejorar la operación, diseño y mantenimiento de la red. Usar todos los recursos de manera oportuna y eficiente, permitirá a las empresas eléctricas y a los consumidores obtener grandes beneficios económicos y ambientales.

La cantidad de datos generados por sistemas y dispositivos que forman parte de una red eléctrica inteligente, así como de fuentes externas, representan un gran desafío para los sistemas computacionales tradicionales, sin embargo, la importancia de generar información útil con estos datos, la cual permita optimizar los recursos disponibles y futuros, es ahora más relevante que nunca, es por eso que es necesario utilizar nuevas tecnologías de comunicación y técnicas computacionales para generar conocimiento de esta gran cantidad de datos. Es aquí donde el análisis de datos y Big Data pueden ofrecer una solución para extraer valor de los datos, proveyendo técnicas para el almacenamiento y procesamiento que no se ven limitadas por el volumen, velocidad o la variedad de los datos.

Uno de los dispositivos que se integran a la red eléctrica como parte de la transformación a la red eléctrica inteligente son los medidores inteligentes. Un medidor inteligente, es un dispositivo de monitoreo y control que permite una comunicación bidireccional entre el medidor y un sistema central. Esto permite a la empresa proveedora obtener información sobre el consumo eléctrico de sus clientes en cortos lapsos de tiempo, así como controlar remotamente el dispositivo. La capacidad

de comunicación constante de estos dispositivos permite la recopilación de información del consumo eléctrico de cada cliente en periodos de horas, minutos o segundos, lo que supone un nuevo reto para su almacenamiento, procesamiento y análisis. De manera hipotética, podemos ejemplificar la cantidad de datos que se podrían generar en un futuro usando información actual de la Comisión Federal de Electricidad (CFE). Para diciembre del 2018, la CFE reportó que en la Ciudad de México existían $3,214,653$ medidores en servicio, si cada uno de éstos fuera reemplazado por un medidor inteligente, y éstos recopilaran datos sobre el consumo eléctrico, identificador del dispositivo, estampas de tiempo y meta-datos cada treinta minutos durante un año, se generarían 3.43 TB de datos en formato CSV; si se implementaran y recopilaran datos treintaminutales de medidores inteligentes durante un año en todo el país, que para diciembre del 2018 sumaban $43,365,753$ medidores, esto generaría 46.35 TB de datos. Además de los datos generados por medidores inteligentes, si se toman en cuenta datos generados por otras fuentes de la misma red eléctrica y fuentes externas, es claro que el volumen, variedad y la velocidad de los datos requieren nuevas soluciones escalables para su almacenamiento y procesamiento. Es indispensable desarrollar e implementar soluciones basadas en tecnologías Big Data, que permitan realizar tareas que herramientas tradicionales no son capaces de realizar.

## 4.2. Descripción del problema

La instalación y uso de medidores inteligentes para clientes de múltiples empresas en la industria eléctrica es ya una realidad en varias partes del mundo, otras empresas tienen entre sus planes iniciar este proceso o deberán iniciarlo para poder modernizar su red y ésta esté preparada para la inminente inclusión de nuevas tecnologías y fuentes de generación de energía. Es por esto, que es necesario desarrollar técnicas analíticas basadas en tecnologías de almacenamiento y procesamiento Big Data, que permitan el manejo de los grandes volúmenes de datos generados por medidores inteligentes, así como de otros dispositivos de la red eléctrica y datos generados por fuentes externas. Aunque la aplicación de técnicas analíticas avanzadas, como algoritmos de aprendizaje de máquina, en datos en la industria eléctrica no es un área inexplorada, estas aplicaciones no han sido pensadas con escalabilidad en mente, sino desarrolladas para conjuntos de datos de menor tamaño, por lo que no son trasladables en el caso de una problemática Big Data, como lo son los datos de la red eléctrica inteligente y en particular, los datos generados por medidores inteligentes. Dada esta situación, es necesario proponer y desarrollar técnicas de analítica avanzada escalables, que puedan ser aplicadas en la práctica y que permitan extraer valor de los datos generados por medidores inteligentes. Entre las aplicaciones analíticas de datos de medidores inteligentes que es posible encontrar en la literatura [15] se encuentran la identificación de datos erróneos, detección de pérdidas no técnicas, identificación de cargas, pronóstico del consumo, caracterización de clientes y aplicaciones para la respuesta a la demanda. Sin embargo, es necesario que los desarrollos que busquen explotar los datos de medidores inteligentes sean capaces de adaptarse al crecimiento de los conjuntos de datos, ya que, de no ser así, simplemente no serán aplicables en situaciones donde se presenten grandes volúmenes de datos.

## 4.3. Objetivo

Desarrollar una aplicación de analítica avanzada escalable para el pronóstico del consumo eléctrico de usuarios de medidores inteligentes usando una arquitectura Big Data.

## 4.4. Datos

Para esta aplicación, se utilizaron datos de consumo eléctrico de 5,567 hogares con medidores inteligentes de la ciudad de Londres, publicados por *UK Power Networks* [14], los datos se presentan en formato CSV, las mediciones fueron realizadas cada 30 minutos durante aproximadamente tres años y se tienen 167 millones de registros en total. Además, datos del clima y fechas festivas de los periodos respectivos fueron utilizados para complementar los datos de los medidores.

## 4.5. Metodología

En el área de aprendizaje automático, el término *pipeline* es utilizado para describir un flujo de trabajo que consiste en múltiples etapas que forman una secuencia de operaciones que conllevan a un objetivo deseado. Las etapas de un *pipeline* reciben datos procesados por una etapa anterior, realizan operaciones a los datos y retornan su resultado a una siguiente etapa. El uso de *pipelines* permite que las aplicaciones generadas puedan ser modificadas fácilmente, ya que las etapas pueden ser reemplazadas o alteradas sin afectar a otras etapas del *pipeline*; nuevas etapas pueden ser adicionadas para añadir nuevas operaciones; cada etapa puede ser desarrollada de manera independiente, de tal manera que pueden ser priorizadas de acuerdo con su importancia.

Para esta investigación, se desarrolló un *pipeline* (Figura 4.1) para generar pronósticos del consumo eléctrico de usuarios de medidores inteligentes. El *pipeline* está compuesto por múltiples etapas que transforman o generan datos que son utilizados durante el flujo de trabajo. Apache Spark fue utilizado para el desarrollo de esta aplicación, lo que provee escalabilidad en cada una de las etapas que la conforman.

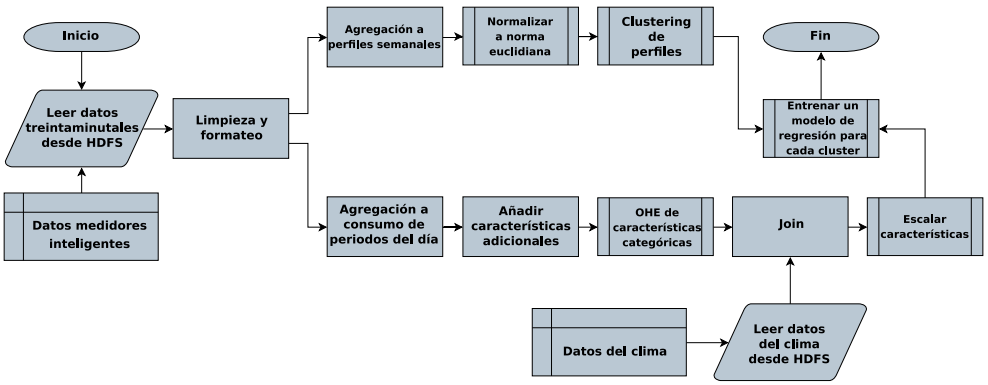


Figura 4.1: Flujo de trabajo para el pronóstico del consumo eléctrico.

El *pipeline* para esta aplicación se compone principalmente de dos tareas de aprendizaje de máquina, *clustering* y regresión. En la rama superior del *pipeline*, se generan los perfiles semanales normalizados para aplicar el algoritmo de *clustering* con el objetivo de generar grupos de usuarios que tienen patrones de consumo similares. En la rama inferior, se utilizan funciones de agregación para obtener datos de consumo a nivel de periodo del día. Después, se añaden características adicio-

nales y si es necesario, éstas son transformadas mediante el algoritmo de OHE (One Hot Encoding). Finalmente, a éstos vectores de características se le añaden datos del clima correspondientes. Los datos son ingresados a modelos de regresión, de acuerdo a la información generada por la etapa de *clustering*. El entrenar modelos de regresión que explotan las propiedades de los patrones de consumo de los usuarios permite obtener pronósticos de consumo particularizados.

### 4.5.1. Perfiles de consumo semanales

El perfil de consumo semanal de un cliente caracteriza su comportamiento promedio de consumo eléctrico durante los siete días de la semana. Un perfil de consumo es generado al promediar para cada día de la semana, los consumos agregados de 4 etapas del día. La Tabla 4.1 muestra las etapas del día aquí utilizadas. En resumen, cada medidor generó en promedio $52,560$ mediciones durante tres años con mediciones treintaminutales (365 días en un año x 3 años x 48 mediciones al día). Al agregar el consumo a un nivel de etapa del día, se pasa a tener $4,380$ mediciones en la serie de tiempo. Finalmente, se promedian estas mediciones a nivel día de la semana y etapa del día, por lo que cada perfil semanal de consumo contendrá 28 elementos (4 etapas del día x 7 días de la semana).

| Noche | Mañana | Tarde | Tarde-noche |
|---|---|---|---|
| 21 a 5 horas | 5 a 12 horas | 12 a 18 horas | 18 a 21 |

Tabla 4.1: Secciones del día como niveles de agregación.

### 4.5.2. Normalización de los perfiles de consumo

Después de generar un perfil de consumo semanal para cada medidor en el conjunto de datos, se normalizó cada perfil a su norma euclidiana (Ecuación 4.1). La norma euclidiana está definida como:

$$|x| = \sqrt{\sum_{k=1}^{n} |x_k|^2} \tag{4.1}$$

Normalizar los perfiles de consumo de la manera mencionada, antes de aplicar un algoritmo de *clustering* basado en distancias euclidianas, es equivalente a emplear la similitud coseno como métrica de distancia, lo que permite eliminar la magnitud de los vectores como factor de agrupamiento. Ya que el interés para esta aplicación es el encontrar perfiles con patrones similares sin importar las diferencias en las cantidades consumidas, es necesario eliminar la magnitud de los vectores antes de la etapa de agrupamiento. La Figura 4.2 y Figura 4.3 muestran un ejemplo de la diferencia entre aplicar y no aplicar normalización respectivamente para tres perfiles de consumo con patrones similares, pero con magnitudes diferentes.

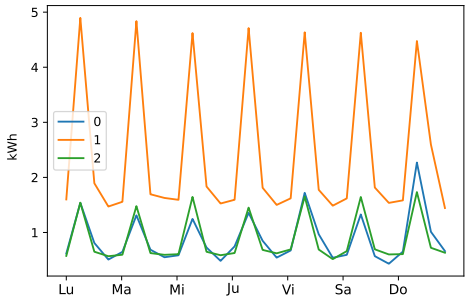


Figura 4.2: Perfiles de consumo semanales no normalizados

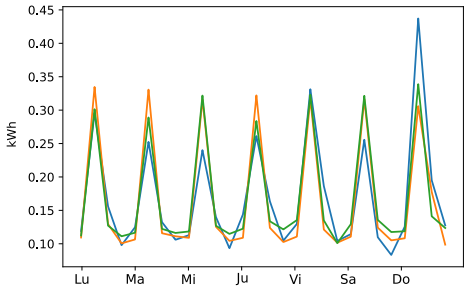


Figura 4.3: Perfiles de consumo semanales normalizados

### 4.5.3. Agrupamiento de perfiles

Un algoritmo de *clustering* permite particionar los objetos del conjunto de datos de entrada, asignando objetos similares a un mismo grupo. Para esta aplicación, un algoritmo de *clustering* es utilizado para encontrar agrupaciones de perfiles de consumo semanales. Cada grupo encontrado por el algoritmo contendrá los perfiles de clientes que comparten de manera cercana sus patrones de consumo eléctrico.

#### 4.5.3.1. $K$-Medias

$K$-medias es un algoritmo de *clustering* que recibe como entrada un conjunto de datos y un parámetro $K$, este parámetro indica el número de *clusters* o grupos en los que se desea agrupar los elementos del conjunto de datos. El algoritmo encuentra una solución al problema de optimización (Ecuación 4.2) que busca minimizar las distancias entre los elementos y el centroide del grupo al que fueron asignados. Tal problema de optimización se formula de la siguiente manera:

$$min \sum_{i=1}^{n} \sum_{j=1}^{K} w_{ij} ||x^{(i)} - \mu_j||^2 \tag{4.2}$$

sujeto a

$$\sum_{i=1}^{n} w_{ij} = 1 \, \forall j$$

donde

$n$ - es el número de ejemplos.
$K$ - es el número de centroides.
$||x^{(i)} - \mu_j||^2$ - es la distancia Euclidiana entre $x^{(i)}$ (el $i^{esimo}$ ejemplo) y $\mu_j$ (el $j^{esimo}$ centroide).
$w_{ij}$ - es una variable binaria que indica si el ejemplo $i$ fue asignado al centroide $j$.

El algoritmo de $K$-Medias, también conocido como algoritmo de Lloyd's, consiste en dos etapas. La primera etapa del algoritmo, genera el conjunto inicial de $K$ centroides. La segunda etapa trabaja de forma iterativa hasta cumplir con una condición de paro. Una iteración de la segunda etapa asigna a cada elemento del conjunto de datos al centroide más cercano y finalmente actualiza los centroides promediando sus respectivos elementos asignados. El algoritmo de K-medias se define como:

1. Inicializa los $K$ centroides $\mu_1, \mu_2, \cdots, \mu_K \in \mathbb{R}^n$.
2. Repetir hasta cumplir con condición de paro {
   Para cada $i$, asigna
   $$w_{ij} := \begin{cases} 1 & \text{if } j = \arg\min_j ||x^{(i)} - \mu_j||^2 \\ 0 & \forall j \neq \arg\min_j ||x^{(i)} - \mu_j||^2 \end{cases}$$
   Para cada $j$, asigna
   $$\mu_j := \frac{\sum_{i=1}^{n} w_{ij} x^{(i)}}{\sum_{i=1}^{n} w_{ij}}$$
   }

El algoritmo de $K$-medias fue utilizado para hallar las agrupaciones de perfiles de consumo semanales. Con el objetivo de encontrar grupos con patrones con alta similitud, el algoritmo fue ejecutado en múltiples ocasiones con diferentes valores de $K$, aumentando el valor de $K$ hasta hallar un *clustering* donde, para cada *cluster*, la varianza en cada dimensión de los elementos asignados al respectivo *cluster* sea menor o igual a la unidad. Dada la escalabilidad que la arquitectura Big Data provee, ejecutar múltiples modelos es factible, aún cuando el volumen de datos es inmenso.
Para la inicialización de centroides, se seleccionaron $K$ elementos del conjunto de entrada de forma aleatoria. Para cada valor de $K$, se entrenaron 10 modelos distintos con la intención de evitar mínimos locales causados por la inicialización aleatoria. La condición de paro del algoritmo fue cumplir con al menos $1,000$ iteraciones o converger a un mínimo local.

### 4.5.4. Regresión para el pronóstico del consumo

Para desarrollar modelos para el pronóstico del consumo eléctrico de usuarios con datos de consumo a nivel medidor, ciertos aspectos deberán ser considerados. Los datos de consumo a gran escala son complejos y contienen una variedad enorme de patrones de consumo, factores como el día de la semana, localización geo-espacial del hogar, tamaño del hogar, número de habitantes en la vivienda, entre otros, hacen que los patrones de consumo de un usuario puedan ser muy distintos a los de otros usuarios; por ejemplo, los patrones de consumo de un hogar donde habita un adulto que trabaja gran parte del día, será muy distinto al patrón de consumo de un hogar donde vive una familia de cuatro integrantes. De manera contraria, usuarios que comparten ciertos factores, tienen mayor probabilidad de tener patrones similares.

Una posible estrategia para el pronóstico es generar un único modelo de regresión que sea entrenado con el conjunto de datos completo. Este modelo sería utilizado para hacer pronósticos para cualquier usuario. Sin embargo, este modelo tendría que ser lo suficientemente complejo para entrenar con poco sesgo (*bias*) sin sacrificar la varianza, lo cual es indispensable dadas las propiedades de los datos.

Una segunda posible estrategia es generar un modelo de regresión personalizado para cada usuario. Cada modelo recibiría datos de entrenamiento generados por el medidor correspondiente a cada usuario. Los modelos serían utilizados para pronosticar el consumo del respectivo usuario. La desventaja de esta estrategia es la limitada cantidad de datos de entrenamiento con los que se contaría para cada modelo, lo que podría conllevar a modelos con alto sesgo.

Para esta aplicación, se recurrió a una estrategia alternativa que busca balancear las ventajas y desventajas de las estrategias antes mencionadas. La información generada por el algoritmo de agrupamiento fue utilizada para generar $K$ modelos de regresión para los diferentes grupos de perfiles de consumo encontrados, esto permite reducir la complejidad del problema, aumentar el tamaño de los conjuntos de datos de entrenamiento que recibe cada modelo y otorga flexibilidad para modificar de manera independiente cada modelo de pronóstico para los diferentes tipos de perfiles de consumo. Ya que los perfiles utilizados en la etapa anterior del *pipeline* fueron generados con datos agregados a etapas del día, los modelos de regresión de igual manera utilizan datos agregados al mismo nivel.

Cada entrada en el conjunto de datos corresponde a un vector de características que incluyen una mezcla de variables numéricas y categóricas, cada una con una relación en el consumo eléctrico. Cada variable categórica fue transformada mediante OHE (*One Hot Encoding*) antes de ser pasada al modelo de regresión para el entrenamiento. Las variables categóricas utilizadas fueron:

- Día de la semana. Esta variable indica el día de la semana del respectivo vector de características.
- Horario del día. Indica el horario del día del respectivo vector de características.
- Día festivo. Una variable binaria indicativa si el respectivo día es o no uno festivo.

Las siguientes variables numéricas fueron generadas mediante funciones de agregación para coincidir con el día de la semana y horario del día respectivo de cada entrada:

- Temperatura promedio, máxima y mínima reportadas.
- Velocidad del viento promedio, máxima y mínima reportadas.
- Visibilidad promedio, máxima y mínima reportadas.

El algoritmo de regresión lineal fue utilizado para generar los modelos regresión (Ecuación 4.3). Cada modelo se entrenó durante 100 iteraciones y un parámetro de regularización $\alpha = 0.01$ fue utilizado para evitar el sobreajuste. El modelo de regresión lineal se define como:

Hipótesis:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \tag{4.3}$$

Parámetros:

$\theta$, un vector con $n + 1$ elementos.

Función de costo:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Cada modelo de regresión lineal fue entrenado mediante el algoritmo de descenso de gradiente que es definido por la Ecuación 4.4.

Repetir{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{4.4}$$

} para $j = 0, ..., n$

donde

$x_0 = 1$
$n$ - el número de características.
$m$ - el número de ejemplos de entrenamiento.
$x^{(i)}$ - el vector de características del $i$-ésimo ejemplo de entrenamiento.
$y^{(i)}$ - el valor de salida (*target*) del $i$-ésimo ejemplo de entrenamiento.
$\alpha$ - la tasa de aprendizaje.

## 4.6. Resultados

Durante la etapa de *clustering*, se generaron perfiles semanales de consumo para cada medidor en el conjunto de datos. Después, se aplicó el algoritmo de $K$-medias para obtener agrupaciones de usuarios con perfiles similares. Como resultado del algoritmo se obtuvieron $1,775$ grupos. Cada grupo generado contiene uno o más perfiles con una máxima desviación de 1kWh en cualquiera de las dimensiones de sus respectivos vectores.

La información de los agrupamientos fue utilizada en la etapa de regresión para generar $1,775$ modelos para el pronóstico del consumo eléctrico. Se utilizaron los datos de consumo eléctrico de los medidores asignados a cada uno de los grupos. Éstos fueron agregados a nivel etapa del día antes de ser utilizados para el entrenamiento. Dos conjuntos fueron generados para cada modelo, un conjunto de entrenamiento, que constó del $80\,\%$ de los datos, y uno de pruebas, que constó del $20\,\%$ restante. El entrenamiento de cada modelo fue realizado únicamente utilizando su conjunto de entrenamiento.

Para evaluar el comportamiento de los modelos de regresión entrenados, los conjuntos de entrenamiento y prueba fueron utilizados. El RMSE fue seleccionado como métrica de evaluación. En promedio, el RMSE de los modelos de pronóstico fue de 1.27 para los conjuntos de entrenamiento y de 1.29 en los conjuntos de pruebas. La Figura 4.4 muestra el pronóstico de una semana del consumo eléctrico de un medidor inteligente del conjunto de datos, obtenido usando su respectivo modelo y su conjunto de prueba, además, el pronóstico es comparado con el consumo eléctrico real del medidor para el mismo periodo.

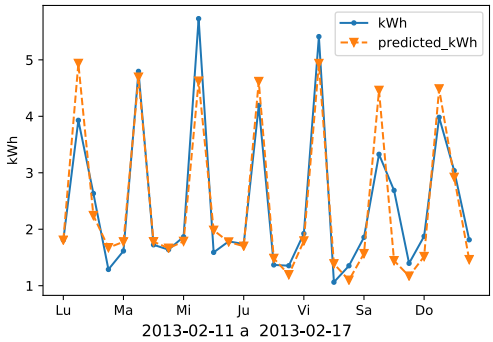


Figura 4.4: Pronóstico contra consumo eléctrico real del medidor MAC001450 asignado al grupo 4.

## 4.7. Conclusiones

Para poder implementar una red eléctrica inteligente, las empresas de la industria eléctrica tendrán que estar preparadas para solventar los problemas que conllevan la enorme cantidad de datos que generarán los nuevos dispositivos, así como los que generan los dispositivos existentes y fuentes de datos externas. El uso de arquitecturas Big Data permitirá a las empresas del sector eléctrico extraer valor de los datos relevantes a la red eléctrica.

Los medidores inteligentes son un ejemplo de los nuevos dispositivos de la red eléctrica inteligente, éstos son capaces de generar grandes cantidades de datos en cortos periodos de tiempo. La aplicación de analítica avanzada desarrollada muestra que es posible diseñar aplicaciones escalables que exploten la información que se encuentra en los datos. El conocimiento generado por este tipo de aplicaciones será utilizado para la toma de decisiones empresariales, lo que conllevará a beneficios económicos, al reducir costos en la generación de energía, infraestructura y mantenimiento; sociales, al automatizar procesos y proporcionar un mejor servicio, lo que se traduce a mejoras en la calidad de vida de los usuarios y empleados; ambientales, al poder utilizar de manera eficiente los sistemas de almacenamiento de energía, fuentes renovables y distribuidas, reduciendo emisiones contaminantes.

La implementación de un *pipeline*, permite a los desarrolladores obtener aplicaciones que son fácilmente configurables y modificables. La aplicación analítica desarrollada podrá ser utilizada como una base para nuevos desarrollos en el área.

Capítulo 5

# Conclusiones generales

La aparición e integración de nuevos dispositivos y tecnologías en múltiples áreas de investigación, en empresas e industrias, ha propiciado que la capacidad de generar datos presente problemas para su almacenamiento y procesamiento mediante sistemas tradicionales para el manejo de datos. Para poder obtener valor de los datos, es necesario utilizar arquitecturas Big Data, que permitan escalar de acuerdo a las necesidades del problema. En este trabajo, dos problemas relacionados con grandes volúmenes de datos fueron explorados para generar soluciones usando una arquitectura Big Data.

En el área de genómica, los avances en las tecnologías de secuenciación masiva están generando datos a una velocidad exponencial. Una aplicación fue desarrollada para el procesamiento de este tipo de datos. La metodología de la aplicación utiliza la escalabilidad de la arquitectura Big Data para procesar eficientemente BDs de referencia de secuencias de ADN y generar nuevas BDs de tamaño reducido. Las herramientas de análisis que utilizan BDs de referencia en su metodología podrán beneficiarse de utilizar las BDs generadas al mejorar la velocidad y calidad de los análisis.

En la industria eléctrica, la evolución de la red eléctrica tradicional a una red eléctrica inteligente implica la inclusión de nuevos dispositivos y sistemas capaces de generar grandes volúmenes de datos a gran velocidad. Entre éstos dispositivos, se encuentran los medidores inteligentes, dispositivos de monitoreo y control capaces de generar una gran cantidad de datos en cortos periodos de tiempo. Una arquitectura Big Data proporcionará la escalabilidad necesaria para almacenar y procesar eficientemente los datos generados en la red eléctrica inteligente. Un *pipeline* de algoritmos de aprendizaje de máquinas fue desarrollado para el pronóstico del consumo eléctrico usando datos generados por medidores inteligentes. El *pipeline* propuesto es altamente modular, lo que permitirá mejorar los componentes de éste o agregar nuevas funcionalidades. Esta aplicación podrá ser utilizada como una base para futuros desarrollos en el área.

Los fundamentos de la arquitectura Big Data descrita en este trabajo podrán ser utilizados para facilitar la adopción y despliegue de soluciones escalables para el almacenamiento y procesamiento de datos en áreas donde se está presentando un aumento sustancial en la cantidad y complejidad de éstos. La arquitectura, basada en el ecosistema de Hadoop, contiene múltiples herramientas para el almacenamiento, procesamiento y análisis de datos escalable mediante el uso de sistemas distribuidos. El HDFS y Hadoop MapReduce forman parte de la base de Hadoop, pero otras herramientas del ecosistema pueden trabajar en conjunto con ésta para extender las funcionalidades y cubrir necesidades especificas para varios tipos de problemas Big Data.

Se espera que este trabajo establezca una base e impulse el desarrollo de nuevas aplicaciones que aprovechen las capacidades computacionales de sistemas distribuidos en áreas donde las propiedades de los datos han superado o están empezando a superar las capacidades de la herramientas tradicionales.

# Bibliografía


[1] Florent E. Angly y col. "Grinder: a versatile amplicon and shotgun sequence simulator". En: *Nucleic Acids Research* 40.12 (mar. de 2012), e94-e94. ISSN: 0305-1048. DOI: 10.1093/nar/gks251. eprint: http://oup.prod.sis.lan/nar/article-pdf/40/12/e94/25343576/gks251.pdf. URL: https://doi.org/10.1093/nar/gks251.

[2] *Apache Hadoop*. URL: https://hadoop.apache.org/.

[3] *Apache Spark - Unified Analytics Engine for Big Data*. 2019. URL: https://spark.apache.org/.

[4] Michael Armbrust y col. "Spark SQL: Relational Data Processing in Spark". En: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD 15. Melbourne, Victoria, Australia: ACM, 2015, págs. 1383-1394. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742797. URL: http://doi.acm.org/10.1145/2723372.2742797.

[5] Dhruba Borthakur. *The hadoop distributed file system: Architecture and design*. 2007.

[6] Jeffrey Dean y Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". En: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. OSDI'04. San Francisco, CA, 2004.

[7] Sanjay Ghemawat, Howard Gobioff y Shun-Tak Leung. "The Google File System". En: *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. Bolton Landing, NY, 2003, págs. 20-43.

[8] *HDFS Architecture*. 2017. URL: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[9] *HDFS Erasure Coding*. 2015. URL: https://issues.apache.org/jira/browse/HDFS-7285.

[10] Ben Langmead y Steven L Salzberg. "Fast gapped-read alignment with Bowtie 2". English (US). En: *Nature Reviews Clinical Oncology* 9.4 (abr. de 2012), págs. 357-359. ISSN: 1759-4774. DOI: 10.1038/nmeth.1923.

[11] *MapReduce Tutorial*. 2017. URL: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

[12] Eugene W. Myers y Webb Miller. "Optimal alignments in linear space". en. En: *Bioinformatics* 4.1 (1988), págs. 11-17. ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/4.1.11. URL: https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/4.1.11.

[13] Saul B. Needleman y Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". En: *Journal of Molecular Biology* 48.3 (1970), págs. 443 -453. ISSN: 0022-2836. DOI: https://doi.org/10.1016/0022-2836(70)90057-4. URL: http://www.sciencedirect.com/science/article/pii/0022283670900574.

[14] *SmartMeter Energy Consumption Data in London Households*. 2015. URL: https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households.

[15] Y. Wang y col. "Review of Smart Meter Data Analytics: Applications, Methodologies, and Challenges". En: *IEEE Transactions on Smart Grid* 10.3 (2019), págs. 3125-3148. ISSN: 1949-3053. DOI: 10.1109/TSG.2018.2818167.

[16] Matei Zaharia y col. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”. En: *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, págs. 15-28. ISBN: 978-931971-92-8. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

[17] Matei Zaharia y col. “Spark: Cluster Computing with Working Sets”. En: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud 10. Boston, MA: USENIX Association, 2010, págs. 10-10. URL: http://dl.acm.org/citation.cfm?id=1863103.1863113.

Cuernavaca, Morelos a 21 de Agosto del 2019.

**DRA. LAURA PATRICIA CEBALLOS GILES**
**DIRECTORA DE LA FCAeI**
**PRESENTE**

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Pérez Garza Javier Alberto, con matrícula 10010402, con el título **Almacenamiento y procesamiento de grandes volúmenes de datos en una arquitectura Big Data: aplicaciones en la investigación e industria eléctrica** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además, construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente,

**Dr. Benjamin Eddie Zayas Pérez**
**Investigador**
**INEEL**

***Por una humanidad culta***
*Una universidad de excelencia*

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos a 21 de Agosto del 2019.

**DRA. LAURA PATRICIA CEBALLOS GILES**
**DIRECTORA DE LA FCAeI**
**PRESENTE**

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Pérez Garza Javier Alberto, con matrícula 10010402, con el título **Almacenamiento y procesamiento de grandes volúmenes de datos en una arquitectura Big Data: aplicaciones en la investigación e industria eléctrica** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además, construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente,

Blanca T.

**Dra. Blanca Itzel Taboada Ramírez**
**Investigador**
**Instituto de Biotecnología**

***Por una humanidad culta***
*Una universidad de excelencia*

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos a 21 de Agosto del 2019.

**DRA. LAURA PATRICIA CEBALLOS GILES**
**DIRECTORA DE LA FCAeI**
**PRESENTE**

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Pérez Garza Javier Alberto, con matrícula 10010402, con el título **Almacenamiento y procesamiento de grandes volúmenes de datos en una arquitectura Big Data: aplicaciones en la investigación e industria eléctrica** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además, construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente,

**Dr. Jorge Hermosillo Valadez**
**Profesor - investigador**
**Centro de Investigación en Ciencias**

***Por una humanidad culta***
*Una universidad de excelencia*

Cuernavaca, Morelos a 21 de Agosto del 2019.

**DRA. LAURA PATRICIA CEBALLOS GILES**
**DIRECTORA DE LA FCAeI**
**PRESENTE**

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Pérez Garza Javier Alberto, con matrícula 10010402, con el título **Almacenamiento y procesamiento de grandes volúmenes de datos en una arquitectura Big Data: aplicaciones en la investigación e industria eléctrica** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además, construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente,

**Dra. Lorena Díaz González**
**Profesor - investigador**
**Centro de Investigación en Ciencias**

***Por una humanidad culta***
*Una universidad de excelencia*

Cuernavaca, Morelos a 21 de Agosto del 2019.

**DRA. LAURA PATRICIA CEBALLOS GILES**
**DIRECTORA DE LA FCAeI**
**PRESENTE**

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado de Maestría en Optimización y Cómputo Aplicado, del estudiante Pérez Garza Javier Alberto, con matrícula 10010402, con el título **Almacenamiento y procesamiento de grandes volúmenes de datos en una arquitectura Big Data: aplicaciones en la investigación e industria eléctrica** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además, construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que el estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente,

**Dr. Outmane Oubram**
**Profesor- investigador**
**Facultad de Ciencias Químicas e Ingeniería**

***Por una humanidad culta***
*Una universidad de excelencia*