IMPLEMENTACIÓN DE UN SISTEMA DE ALMACENAMIENTO DE DATOS MASIVOS PARA MONITOREO ESTRUCTURAL

Johann Llanos F.
Departamento de Ingeniería Informática
Universidad de Concepción

Nombre Profesor Guía: Gonzalo Rojas D., PhD
Comisión: Guillermo Cabrera V., PhD, Julio Godoy D., PhD

2 de Abril de 2018

Abstract

El presente trabajo aborda el problema de almacenamiento de datos masivos o Big Data aplicado a un caso de estudio específico: la captura de datos desde estructuras de Ingeniería Civil a través de sensores. A partir de dicho caso de estudio se busca definir, diseñar e implementar un sistema de almacenamiento apropiado al caso, que cumpla con los requerimientos identificados, para luego realizar pruebas pertinentes de rendimiento sobre consultas y análisis de los datos almacenados.

Índice

1. INTRODUCCIÓN

El presente informe expone el procedimiento llevado a cabo para enfrentar un Caso de Estudio correspondiente a una empresa de Monitoreo de Salud Estructural, área que implementa detección de daño en infraestructuras civil, mecánica o aeroespaciales. Para el caso se identificó un problema de gestión de datos, presente en el proceso de adquisición de datos desde sensores. Esto, fue seguido por un proceso de definición, planificación y desarrollo de un prototipo de sistema de almacenamiento, el cual se trabajó sobre una herramienta seleccionada en base a las necesidades y limitantes identificadas para tratar dicho problema de información y el masivo volumen de datos que éste involucra.

Actualmente existe una difusión considerable en el uso de Datos Masivos – o Big Data –, pues se valora cada vez más el alcance de la información que es posible extraer desde dichos datos, a partir de una creciente variedad de fuentes presentes en el medio. El área de Monitoreo de Salud Estructural – desde ahora SHM, *Structure Health Monitoring* – implica una presencia inherente de Big Data, pues se basa en utilizar una red de sensores instalada en una estructura que interese monitorear para detectar la aparición de daño. El flujo continuo de datos entrante tiene un volumen y velocidad considerables, haciendo que el *dataset* sea complejo de manejar. Teniendo presentes los potenciales riesgos que puede ocasionar una falla estructural y la complejidad técnica del área, es claro que SHM y el caso de estudio representan un gran desafío de información en todos sus niveles y etapas.

1.1 Objetivos del Proyecto

El trabajo realizado en este informe persigue como objetivo general presentar una alternativa viable respecto al estado actual de almacenamiento de datos en el caso de estudio y evaluar comparativamente el rendimiento de tal alternativa al ser aplicada.

Los objetivos específicos del proyecto son:

I. Recopilar un trasfondo teórico del paradigma Big Data de almacenamiento y de las últimas herramientas disponibles.
II. Planificar, diseñar e implementar un prototipo del sistema seleccionado para el almacenamiento de los datos asociados a los sensores.

El trabajo descrito en este informe está dividido en las siguientes secciones: En el capítulo de Caso de Estudio, se presenta la situación actual de la empresa estudiada, los puntos relevantes extraídos de su análisis, el problema principal que se identificó y trabajó, incluyendo además los conceptos importantes del área trabajada y las tareas realizadas. Posteriormente, el capítulo de Discusión Bibliográfica profundiza en los conceptos fundamentales de almacenamiento de datos masivos, las arquitecturas típicas para gestionar Big Data y las potenciales herramientas de manejo de bases de datos apropiadas al caso. A continuación, en el capítulo de Desarrollo del Sistema, se detalla el procedimiento de diseño e implementación ejecutado en el trabajo del prototipo. Luego, se exponen los resultados obtenidos a partir de los experimentos realizados y finalmente se entregan las conclusiones obtenidas en base al trabajo realizado.

2. CASO DE ESTUDIO

El caso de estudio corresponde a una empresa de SHM, área especializada en la captura y análisis de datos provenientes desde redes de sensores instalados en estructuras de interés, propensas a experimentar daño debido a las fuerzas que experimentan o por haber superado su vida útil. En ambos casos, debe realizarse un estudio de los costos y de la efectividad de realizar reparaciones y/o refuerzos en la estructura para así extender su vida útil y operacional, pues generalmente tales estructuras tuvieron un costo de inversión inicial muy alto. Por esto se debe evaluar dicha inversión en comparación al continuo mantenimiento y monitoreo de la estructura para que ésta pueda continuar en operación. Esto debe contrastarse además con los riesgos o posibles costos humanos que cualquier fallo en la integridad de la estructura pueda tener.

2.1 Estado Actual del Caso

La empresa estudiada se enfoca en el monitoreo de infraestructuras de Ingeniería Civil que sean propensas a experimentar daño basado en vibraciones, tales como puentes, stockpiles[1] o silos[2]. Las características del daño o desgaste de las estructuras varían según las propiedades y trabajo realizado en éstas, aunque los problemas más comunes son grietas, deformación de elementos, corrosión y otros tipos de agentes de degradación estructural.

Se monitorean principalmente puentes de tránsito vehicular, estructuras que experimentan un trabajo de carga variable con naturaleza cíclica, es decir, periodos repetitivos con fuerzas de mayor y menor intensidad. En la Figura 1 se muestra un ejemplo de una red de sensores:

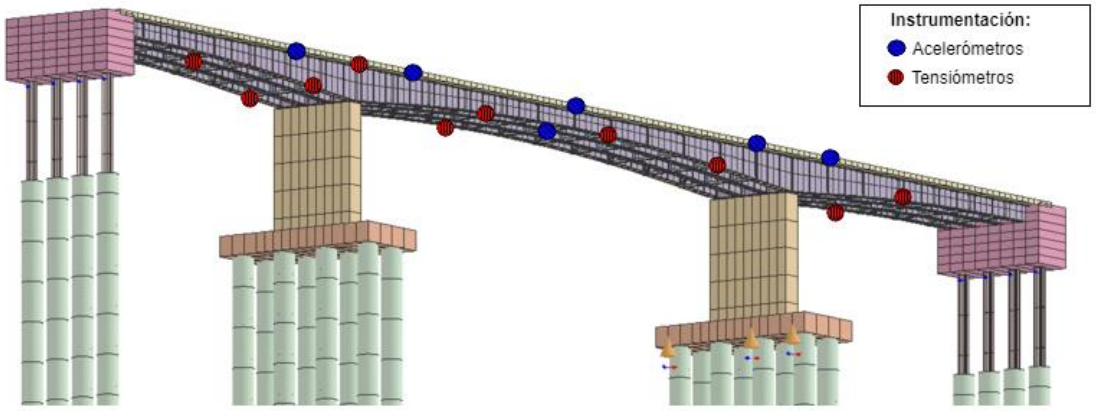


Figura 1: Diagrama ejemplificando una red de sensores instalada sobre una estructura genérica a monitorear [Fuente: http://bimandbeam.typepad.com].

[1]Estructuras de almacenamiento de materias primas, que ingresan el material a través de una correa transportadora hacia la zona de depósito o Stock.
[2]Estructuras cilíndricas de almacenamiento, el material es depositado desde tubos conectados a la parte superior.

Esta actividad requiere analizar un gran volumen de datos provenientes de múltiples sensores instalados en la estructura que se debe monitorear. En particular el caso de estudio presenta alrededor de 120 sensores instalados en una estructura sometida a constantes esfuerzos, con proyección a aumentar la cantidad de sensores y estructuras monitoreadas en el futuro. Los sensores realizan un *streaming* que es capturado en paquetes de 5 minutos, donde cada paquete se compone de 2 canales por sensor, con un valor en bruto compuesto de 200 valores por segundo (Frecuencia de muestreo de 200 Hz) y alrededor de 25 valores o estructuras de datos obtenidos del posterior procesamiento de éstos. Los datos procesados a almacenar corresponden a estructuras de datos como matrices, vectores y valores numéricos, todos con necesidad de alta precisión.

Para dimensionar el volumen de información que se trabaja, debe resaltarse que los paquetes de datos sin procesar consisten de muestras con 300 segundos – o 5 minutos – y una frecuencia de 200 datos por segundo, lo que entrega alrededor de 60.000 datos crudos atómicos para cada uno de los 120 sensores, en cada paquete. Considerando la captura de datos para un día completo, se reciben en total alrededor de 2 mil millones de valores crudos atómicos sin procesar al día. Los datos procesados son de un orden numérico menor, pero el vector de mayor dimensión por sensor sigue siendo de 1025 valores – para la estructura que contiene la FFT –, lo que resulta en alrededor de un máximo de 36 millones de datos procesados diarios, sin considerar el resto de valores y estructuras de datos.

El proceso de instalación de los sensores requiere una definición de elementos estructurales relevantes y la posición en ellos que se desea medir, así como el tipo de datos que se desea obtener a partir de ellos. Luego, se instalan los sensores en los puntos críticos identificados para la adquisición, seguimiento y análisis de los datos. Desde el análisis de los datos se busca identificar patrones y rasgos de interés que son utilizados, principalmente, en obtener información útil para la detección, prevención y predicción de daño estructural.

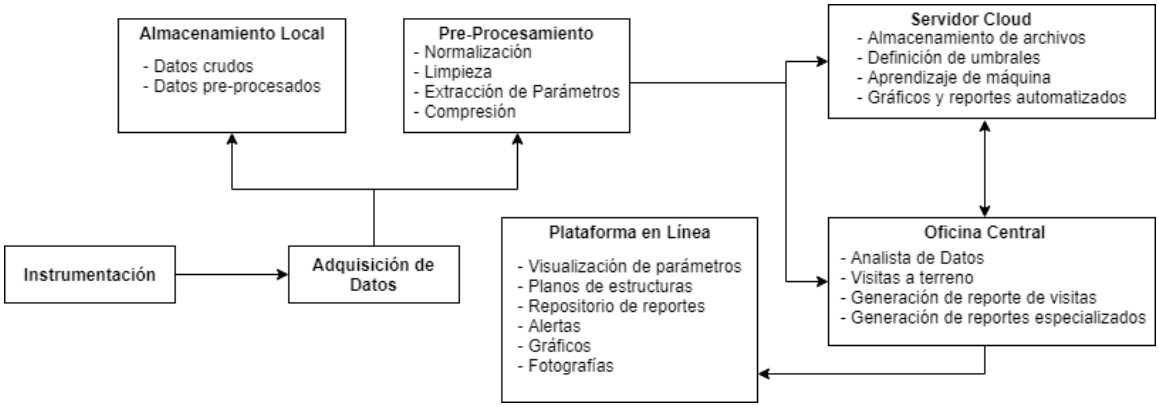


Figura 2: Diagrama del proceso general en el caso de estudio.

La Figura 2 muestra el proceso general que la empresa estudiada ejecuta para monitorear una estructura cualquiera. La instrumentación implica la instalación de la red de sensores, que puede incluir sensores tales como acelerómetros, tensiómetros y otros dispositivos que realicen captura de datos como una estación meteorológica. Los sensores emiten datos

continuamente a un dispositivo adquisidor de datos que los recibe y organiza en paquetes de datos en fracciones de tiempo fijo, actualmente definido en 5 minutos, para que el tamaño de cada paquete sea lo suficientemente pequeño y así puedan ser enviados sin que se solape el envío de dos paquetes consecutivos debido al ancho de banda limitado.

Luego, el dispositivo adquisidor envía estos paquetes a un servidor en terreno a través de red cableada. En dicho servidor se almacenan los datos crudos y se realiza un pre-procesamiento de éstos, generando paquetes de datos procesados que atravesaron normalización, limpieza, extracción de parámetros y compresión. Ambos conjuntos de datos, crudos y procesados son almacenados como archivos del software especializado Matlab, para luego ser ingresados a repositorios en la nube – en el caso de los datos procesados – o a discos duros físicos dentro del servidor en terreno, los que después son extraídos y almacenados en un rack local en la oficina de la organización.

Los archivos Matlab de paquetes procesados son almacenados en un servidor gestor de archivos en la nube, desde donde pueden ser accedidos y utilizados por personal especializado en el análisis, a través de scripts del mismo software. Luego, los resultados de los análisis son almacenados en bases de datos tipo SQL como información no permanente. Esta base de datos contiene información como el estado de salud estructural en los sensores, gráficos de datos y resúmenes extraídos para períodos fijos de tiempo, extendiéndose desde el presente hacia atrás en cierto número de días. Toda esta información es presentada y visualizada a través de *dashboards* en la plataforma web de la organización.

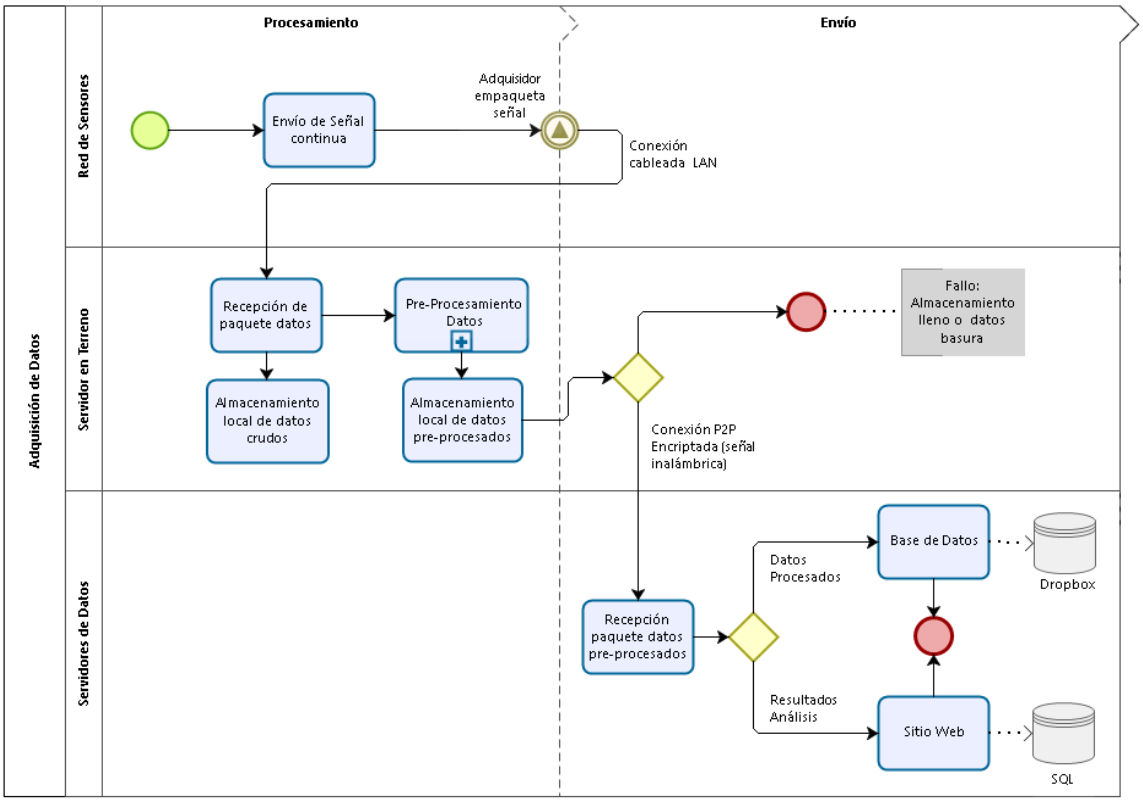


Figura 3: Proceso de Negocio para la adquisición de datos en el caso de estudio.

La Fig. 3 muestra un diagrama de procesos de negocio desarrollado para el estado actual descrito anteriormente, al nivel de infraestructura hardware y software presentes en el sistema. En la figura queda en evidencia que los datos crudos se dejan fuera de la gestión de sistemas de bases de datos y sólo se continúa el proceso con los datos procesados. Los datos procesados contienen principalmente información especializada de parámetros tales como la energía de la señal, la Transformada de Fourier (FFT), momentos temporales y frecuencias fundamentales. También se incluyen valores estadísticos como medias, mínimas y máximas, promedios, media cuadrática (RMS) y desviación estándar de los datos por sensor.

Algunas de las consultas y análisis realizados sobre el conjunto de datos procesados son:

- Obtener el estado actual de salud estructural presente en los sensores
- Para un intervalo de tiempo dado:
  - Obtener los peaks de FFT de entre todos los sensores
  - Concatenar las medias de un sensor específico
  - Cálculo de tensiones por sensor
  - Concatenar aceleraciones de sensor específico.
  - Visualizar frecuencias naturales de la estructura
  - Obtener ciclos de carga por sensor

Además, la organización realiza gestión de hallazgos provenientes de las visitas en terreno realizadas, incluyendo fotografías, coordenadas, comentarios y clasificación de estos hallazgos.

El caso de estudio hace necesario aclarar la definición de los siguientes conceptos:

a. **FFT:** La Transformada de Fourier lleva una función de tiempo, en este caso una señal proveniente de los sensores, al dominio de las frecuencias que la componen. Es decir, transforma una señal en el tiempo a valores de amplitud en un rango de frecuencias.
b. **Tensión:** Los sensores utilizados para medir deformación (Tensiómetros o *Strain Gauges*) en la estructura, registran diferenciales de voltaje que indican la variación ΔL para un elemento de largo L. La tensión o estrés del elemento se calculan a partir de esta deformación utilizando las propiedades del material.
c. **Frecuencia Natural:** En términos estructurales se refiere a los valores de frecuencias en los que la estructura o sus elementos tienden a vibrar. Esto ayuda a evitar dichas frecuencias en la operación de las estructuras, pues un trabajo que ingrese a tales frecuencias produce resonancia, lo que causa oscilaciones mayores y un más alto riesgo de daño estructural.
d. **Ciclos de Carga:** Se refiere a los niveles de carga operacional en la estructura para los cuales se percibe una tendencia cíclica. Es decir, las magnitudes medidas presentan un comportamiento repetitivo en sus valores máximos y mínimos. Éstos son discriminados según un valor umbral que debe ser definido y ajustado como parámetro.

2.2 Problemas Detectados

Los problemas del modelo de arquitectura actual se concentran principalmente en el almacenamiento de datos procesados y crudos. El almacenamiento de archivos Matlab presenta un conjunto de limitaciones:

- Necesidad de software especializado de licencia comercial para utilizar los archivos adecuadamente.
- Acceso manual o programado por script para acceder las rutas de las carpetas.
- Se requiere revisar constantemente el ordenamiento y estado de los paquetes de datos, pues es posible que algunos de éstos lleguen con retraso.
- Cuando los paquetes de datos simplemente no llegan, se generan paquetes de relleno para tener continuidad en la línea de tiempo.
- Creación, manejo y envío de archivos genera latencia en la manipulación de los datos.
- Necesidad de abrir y cargar archivos a memoria cada vez que el conjunto de datos consultado supere la fracción de tiempo que incluyen de los paquetes.
- Dificultad para respaldar, replicar y escalar el sistema.

- Posibilidad de pérdida irrecuperable de datos crudos, al estar alojados todos en discos duros en una misma ubicación física, sin respaldarse.
- La velocidad en que se incrementan las necesidades de almacenamiento requerido, a largo plazo superará las capacidades de un servidor focalizado de entregar resultados rápidos o almacenar el flujo entrante de datos a la velocidad necesaria.
- El estado actual del sistema previene el acceso eficiente a conjuntos de datos acotados por sensor, ya que los paquetes contienen datos para todos los sensores a la vez.

En base a los problemas y las causas identificadas, se propone transformar el estado actual del modelo de arquitectura con el uso de sistemas de bases de datos que puedan manejar adecuadamente el volumen y flujo de datos que se percibe en el caso de estudio. Se busca implementar una herramienta de almacenamiento compatible con infraestructuras actuales para datos Masivos y que permita un crecimiento escalable con los requerimientos de almacenamiento, procesamiento y análisis de datos presentes en el área de SHM.

El objetivo de transformar el estado actual de la infraestructura es mejorar la gestión de datos en los siguientes aspectos:

1. Desligar el sistema de almacenamiento utilizado, del software especializado utilizado en la organización.
2. Hacer que el conjunto de datos, al ser temporal, deje de ser fragmentado en unidades de una fracción de tiempo fija – actualmente de cinco minutos por paquete de datos– y en vez de esto, sea organizado como un registro histórico.
3. Minimizar el riesgo de pérdida de datos, pues al almacenarse un subconjunto de los datos como discos duros físicos en un rack que actúa como almacén de datos existe posibilidad de daño o extravío de dispositivos.
4. Agilizar el acceso a los datos y paralelizar el procesamiento en el manejo y análisis de la información.
5. Promover una infraestructura adecuada para el manejo de Datos Masivos, que tenga trasfondo teórico o experimental.

3. DISCUSIÓN BIBLIOGRÁFICA

La solución al problema anteriormente descrito requiere el estudio de conceptos, arquitecturas y herramientas que permitan encontrar una alternativa compatible con el caso de estudio y que puedan dar solución al problema encontrado. A continuación se exponen los contenidos que constituyen el marco teórico necesario para el proyecto:

3.1 SISTEMAS DE ALMACENAMIENTO BIG DATA

El almacenamiento del volumen de datos que implica Big Data hace necesario el uso de sistemas distribuidos que adopten y saquen provecho a la necesidad de utilizar múltiples servidores comunicados entre sí para almacenar un mismo conjunto de datos (Hwang et. al 2012). Los dos grandes paradigmas de almacenamiento de datos en la actualidad (Özsu & Valduriez, 2011)[6] son:

1. **Bases de Datos Relacionales Centralizadas (RDBMS):** Sistemas basados en el modelo Relacional de datos. Organizan los datos en tablas y generalmente utilizan el lenguaje de consultas SQL o similares. Extensa presencia en el mercado pero limitados en el manejo de grandes volúmenes de datos.
2. **Bases de Datos Distribuidas (DDBMS):** Corresponden a Sistemas de manejo de bases de datos en donde los dispositivos de almacenamiento no están todos ligados a una unidad de procesamiento común, en cambio, se encuentran propagados y se comunican a través de una red. Esta tecnología existe aplicada en diferentes tipos de sistemas:
    a. **Distributed SQL data warehouses -** Sistemas que permiten el procesamiento distribuido sobre los datos, bajo una infraestructura SQL de almacenamiento.
    b. **Hadoop -** Tecnología de manejo de datos que se define como (Shvachko et. al 2010): “Sistema de archivos distribuido y Framework de software open-source para el almacenamiento, transformación y análisis de conjuntos de datos muy grandes, utilizando el paradigma MapReduce”.
    c. **NoSQL –** Se refiere a un grupo de sistemas de manejo de datos No-Relacionales. Estos sistemas No-Relacionales son distribuidos y diseñados para almacenamiento de datos a gran escala y para procesamiento de datos masivamente paralelo a través de un gran número de servidores (Moniruzzaman & Hossain, 2013).
    d. **NewSQL –** Son sistemas de Bases de Datos Relacionales que utilizan características de estilo NoSQL tales como arquitecturas distribuidas y almacenamiento columnar para mejorar el rendimiento de las RDBMS.
    e. **Distributed Ledgers (DL) –** Sistemas distribuidos que utilizan criptografía para proveer un mecanismo de control de concurrencia descentralizado, permitiendo que las transacciones tengan “testigos” para validarse. Cada

nodo participante puede acceder al registro compartido a través de su red y poseer una copia de dicho registro.

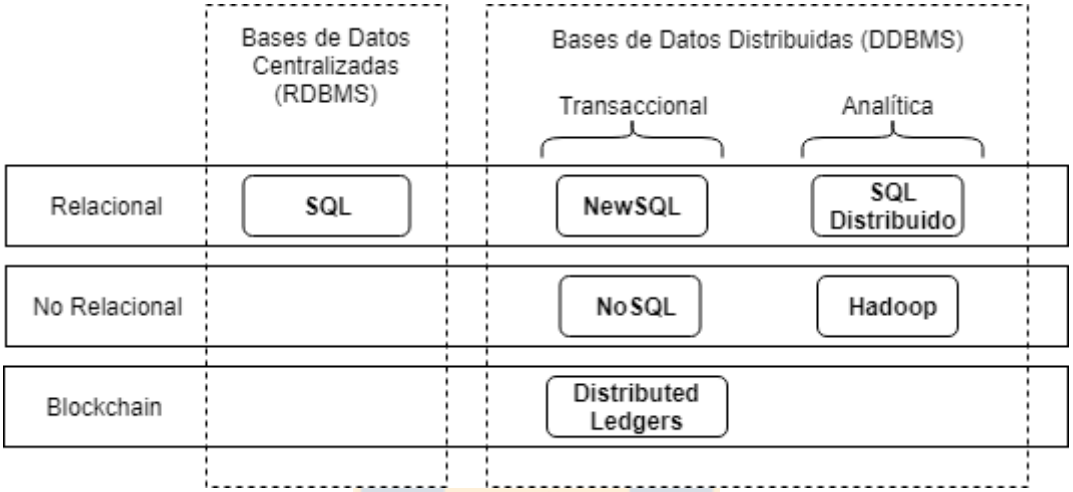


Figura 4: Esquema gráfico de las categorías de DBMS [Fuente: https://juliandontcheff.wordpress.com/2017/10/].

En la figura 4 se pueden ver las categorías principales en los sistemas descritos: centralizados, distribuidos, relacionales y no relacionales. A esto, además, se agregan las categorías de Bases de Datos Transaccionales, capaces de deshacer operaciones de escritura que no se hayan completado apropiadamente en el sistema. Esta categoría incluye a la mayoría de las bases de datos más comunes. En cambio, las bases de datos analíticas son diseñadas para Big Data, Inteligencia de Negocios y Análisis de Datos, con un enfoque en distribuir las tareas de procesamiento entre los nodos. Generalmente contienen funciones incorporadas, herramientas de minería de datos y otras características especializadas [6]. La principal diferencia entre las BD Transaccionales y Analíticas es su paradigma de procesamiento; las BD transaccionales son de consistencia instantánea, es decir, reflejan cambios en los datos automáticamente para ser vistos por el usuario. Por otro lado, las BD Analíticas son de procesamiento periódico latente con resultados entregados luego de completar las tareas de análisis y procesamiento sobre el conjunto de datos. Por último, Blockchain es la tecnología subyacente a los Distributed Ledgers, con un foco a la criptografía y seguridad en transacciones que además son completamente descentralizadas [22].

3.2 BASES DE DATOS DISTRIBUIDAS NO-RELACIONALES

NoSQL representa un *framework* de bases de datos que permiten procesamiento de información ágil y de alto rendimiento a una escala masiva, es decir, está bien adaptada a los fuertes requerimientos de Big Data. Los casos de uso de NoSQL frecuentemente involucran interactividad de usuario final, como en aplicaciones web, pero más ampliamente son sobre leer y escribir datos con baja latencia.

Se mencionó anteriormente que NoSQL engloba un conjunto de sistemas de manejo de datos, los principales son [9][23]:

1. **Key-Value:** El tipo más trivial de almacén de datos NoSQL es el modelo de Key-Value. Éstos almacenan cada ítem en la BD como un nombre de atributo o Key, al cual va asociado un valor.
2. **Modelo de Documento:** Estos sistemas cuentan con la habilidad de consultar sobre cualquier campo dentro de "documentos" en la base de datos. Los documentos son objetos que consisten en uno o más campos autónomos en cada instancia, con una estructura similar a JSON [23].
3. **Modelo de Grafo:** Se utilizan en presencia de consultas abundantes, basándose en inferencias directas e indirectas realizadas sobre los datos y sus relaciones. Los datos son modelados como una red de relaciones entre elementos específicos, con nodos, arcos y propiedades para representarlos como grafos.
4. **Almacenamiento Wide-Column (o Column Family/Extensible record store):** Se asimila a las tablas encontradas en bases de datos tradicionales pero el almacenamiento es generalmente basado en columnas en vez de filas, es decir, las columnas de datos son almacenadas juntas, en vez de filas de datos como en una BD tradicional y las filas se reparten entre distintos servidores.

3.3 ARQUITECTURAS DE SISTEMAS BIG DATA

Para presentar una alternativa a la arquitectura del Caso de Estudio, se busca profundizar en las arquitecturas de Big Data más apropiadas para el problema abordado. A continuación se presentan dos arquitecturas que pueden ser aplicadas bajo las condiciones del caso estudiado:

3.3.1 Arquitectura Lambda

Uno de los modelos de arquitectura que más expansión ha logrado en el medio, diseñado para tratar casos de uso con presencia de *Big Data*, es la llamada Arquitectura Lambda. Este diseño presenta un enfoque en que se trabaja en frameworks independientes el procesamiento en tiempo real y en *batch* – o por lotes –, con un tercer componente global que unifica los resultados de ambos procesamientos. Fue propuesta por Nathan Marz el año 2013 (Marz, N. & Warren)[15], como parte de un proyecto desarrollado en Twitter para manejar las estadísticas de los mensajes a nivel de tiempo real y además realizar procesamiento histórico de éstos. Busca ajustarse a casos en que hay desfase de tiempo entre la adquisición de datos y la disponibilidad de éstos en *dashboards*, con el requerimiento de validación y procesamiento en línea de los datos a medida que llegan (Kiran et. al 2015). En cierta forma, corresponde a dos sistemas independientes que colaboran en sus tareas, uno para manejar información archivada históricamente y otro encargado del flujo entrante de datos nuevos.

Esta arquitectura se compone de las siguientes capas (Marz, N. & Warren, 2013) [15]:

1. **Batch Layer**
   - Gestiona el master dataset, un conjunto de datos crudos inmutables que sólo permiten anexado.
   - Pre-computa funciones de consultas arbitrarias, llamadas Batch Views.
2. **Speed Layer**
   - Acomoda todas las peticiones que están sujetas a requerimientos de baja latencia. Usando algoritmos incrementales y rápidos, la speed layer lidia solamente con datos recientes.
3. **Serving Layer**
   - Indexa las vistas de ambas capas anteriores para puedan ser consultadas en baja latencia de forma ad hoc.

En otras palabras, la arquitectura Lambda presenta un enfoque en que la capa de batch posee veracidad en su versión de los datos, pero con considerable latencia – que incluso puede llegar a tener un día completo de retraso – para entregar resultados. Por otro lado, la speed layer va reflejando resultados instantáneamente para los datos que van ingresando al sistema en tiempo real, sujeto a la posibilidad de recibir datos erróneos. Los datos en la speed layer son transitorios y pueden ser descartados una vez que la batch layer finalice el periodo de tiempo de retraso que necesitaba manejarse con el framework de tiempo real.

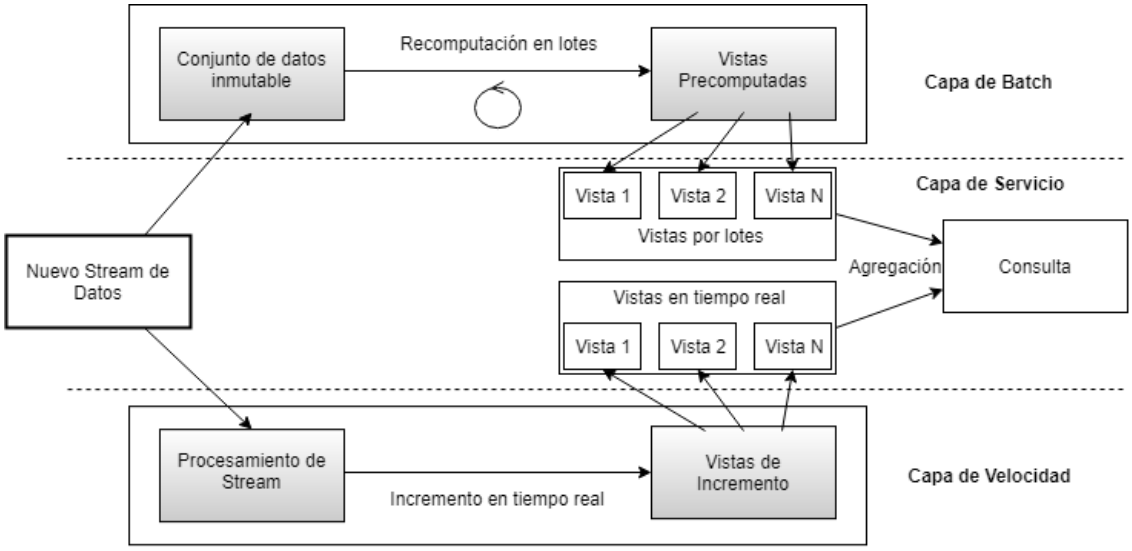


Figura 5: Visualización de los elementos de la Arquitectura Lambda [Fuente: https://dzone.com/articles/lambda-architecture-with-apache-spark].

La Figura 5 muestra un diagrama de la arquitectura Lambda, en donde se puede ver el ingreso de un *stream* de datos que se divide para llegar a ambas capas de procesamiento. En el Batch Layer, los datos son almacenados de manera inmutable a un conjunto de sólo lectura. Sobre estos datos archivados se realiza el procesamiento en *batch* que toma mayor tiempo en obtener resultados. Otra copia de los datos nuevos que ingresan es ingresada por *stream* para obtener resultados en tiempo real, que no son permanentes y sólo muestran la última información disponible.

Algunos de los beneficios de la Arquitectura Lambda son:

- Mantiene los datos de entrada inalterados, haciéndolo apropiado para procesar transformaciones de datos (Series de estados de los datos desde la entrada original).
- La mantención del *log* original de datos permite reprocesar datos originales en caso de introducción de bugs o evolución de algoritmos.
- Esta arquitectura toma en cuenta el problema de reprocesar los datos, una tarea que se realiza todo el tiempo, que puede requerir reprocesar toda la información en algún instante.

Los problemas que pueden presentarse con este diseño:

- Necesita implementarse la lógica de aplicación dos veces, en las tareas de procesamiento en Batch y en la implementación de procesamiento en Stream.
- La mantención del código que requiere producir el mismo resultado en dos sistemas distribuidos diferentes y complejos es de complejidad operacional muy alta (por ejemplo, el código de *MapReduce* y *Apache Storm/Spark*, utilizados para batch y tiempo real respectivamente, son completamente diferentes entre sí).
- Difícil de mantener, costoso y propenso a bugs.

- Requiere conocimiento profundo de ambos subsistemas: Tiempo Real y *Batch*, al nivel de requerir equipos de desarrollo independientes para cada caso.

3.3.2 Arquitectura Kappa

La arquitectura Kappa es una modificación de la arquitectura Lambda con una propuesta que busca simplificarla y hacerla más manejable a nivel de desarrollo, mantención y estabilidad general, eliminando la característica de tener dos complejos sistemas que deben atenderse independientemente y realizar todo el procesamiento de forma acumulativa en un mismo *framework*. Fue propuesta por Jay Krepps el año 2014 [17] y se basa en los siguientes principios:

1. Utilizar un sistema de *logging* que permita retener el registro completo de datos que se requiera reprocesar.
2. En el re-procesamiento, comenzar una segunda instancia de la tarea de procesamiento de *stream* que comience a trabajar desde el principio de los datos retenidos, pero dirigir estos datos a una nueva tabla de salida
3. Cuando la segunda tarea haya alcanzado a la principal, cambiar la aplicación a lectura desde la nueva tabla
4. Detener la versión antigua del trabajo, y borrar la tabla de salida anterior.

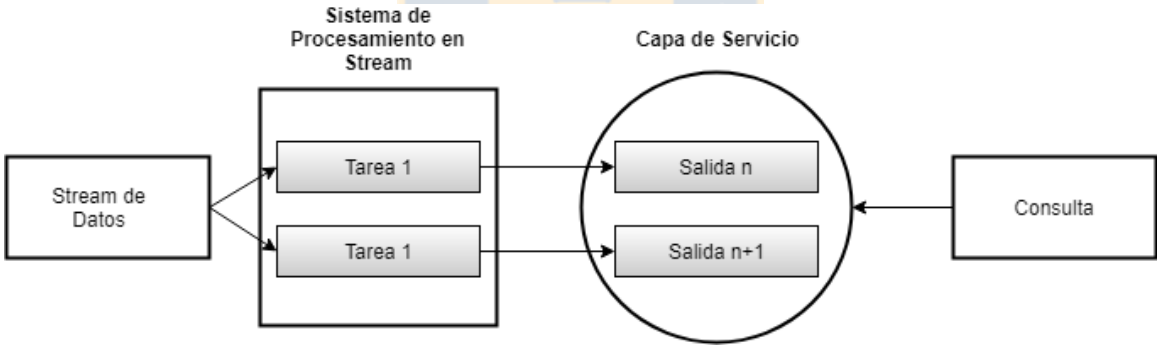


Figura 6: Diseño de Arquitectura Kappa [Fuente: https://dzone.com/articles/lambda-architecture-with-apache-spark].

Como puede verse en la Fig.6, la arquitectura Kappa utiliza tres etapas en su aproximación al manejo de datos masivos, la primera parte de la infraestructura se ocupa de recibir el flujo de datos entrantes y organizarlos como eventos temporales, luego se tiene una capa de procesamiento de tareas y finalmente una capa de servicio que se ocupa de proveer respuestas optimizadas a las consultas de usuario.

Para la recepción del flujo entrante de datos es apropiado el uso de almacenes de registro de datos – también llamados Log Data Stores – que se ocupan de almacenar un conjunto inmutable de los datos recibidos por la arquitectura.

La segunda etapa requiere el uso de un sistema de computación por *streaming* distribuido, que permita repartir la carga de trabajo entre múltiples dispositivos y optimice el rendimiento del trabajo. Finalmente la capa de servicio requiere de un sistema de manejo de Bases de Datos con la capacidad de generarse rápidamente a partir del registro canónico inmutable, es decir, se requiere una velocidad de escritura rápida y una respuesta optimizada a consultas específicas.

Una implementación simple de la arquitectura Kappa utiliza Apache Cassandra (C*) como sistema de almacenamiento, pues una de las principales características de esta BD es su alta velocidad de inserción y lectura, que se ajusta tanto a la arquitectura Kappa como a el caso de estudio, que presenta un flujo de datos de alta velocidad proveniente de sensores. Además, existe documentación (End Point Corporation, 2015) [12] de Benchmarking especializada e independiente que demuestra un rendimiento muy superior de C* respecto de otros sistemas NoSQL en operaciones de escritura y lectura.
Luego, debido a sus características, Cassandra presenta una alternativa apropiada para el sistema de almacenamiento, cuyo rendimiento además ha sido evaluado comparativamente ante otras bases de datos NoSQL en múltiples benchmark [12][24][25].

Las ventajas de la arquitectura Kappa son:

- Permite hacer desarrollo, *testing*, *debugging* y operación de los sistemas sobre un único framework de procesamiento.
- Puede ser implementada bajo un conjunto de tecnologías, como por ejemplo Apache Storm, Spark, Kafka, HBase, HDFS o Samza.
- Mayor flexibilidad de adaptación al cambio de requerimientos de procesamiento y análisis, al mitigar la complejidad de utilizar un segundo framework distribuido.

Las desventajas de esta arquitectura son:

- Para utilizar conjuntos datos que no pertenezcan a las consultas para las que ha sido optimizado el sistema, se requiere hacer un acceso secuencial al Log, lo que puede reducir el rendimiento.

3.4 APACHE CASSANDRA

Apache Cassandra [13] – Abreviado como C* – es una DDBMS de alto rendimiento, utilizada para almacenar grandes cantidades de datos y realizar consultas para las que ha sido optimizada. Es un sistema orientado a proveer gran operación transaccional, escalabilidad y ser altamente disponible – Es decir, siempre entrega una respuesta a las consultas –, pues está diseñado para ser tolerante a fallos. Además, posee un modelo de datos versátil, lenguaje de consulta simple y declarativo, rutas de acceso de escritura y lectura muy eficientes, alta escalabilidad y la capacidad de manejar fallos de nodos e incluso *datacenters* completos.

Es un sistema con infraestructura *NoSQL* y un modelo de datos *Wide-Column*, con una característica importante: Consistencia Eventual, la cual se decide asumiendo que el último valor ingresado es el correcto, pero que puede demorar en verse reflejado a nivel de usuario al realizar un cambio.

Sin embargo, para sacar provecho a la arquitectura y beneficios de C* es imprescindible que el diseño y modelo de datos sea apropiado al caso de uso y al paradigma de la arquitectura, el cual presenta diferencias significativas respecto a las bases de datos relacionales y el modelamiento de datos tradicional. Además, el modelo de almacenamiento de datos distribuido hace difícil (y muy ineficiente) realizar ciertos tipos de consultas para análisis de datos que son mucho más sencillas en una RDBMS.
Típicamente no puede dependerse de Cassandra para hacer agregaciones, análisis de datos y similares. Puede utilizarse para capturar todos los datos que sea necesario, pero por sí mismo, el sistema no tiene capacidades suficientes para aprovechar estos datos correctamente.

A partir de sus características, se han identificado las siguientes fortalezas y debilidades de la herramienta:

3.4.1 Fortalezas de C*

1. Soporta un muy alto flujo de datos. Permite alto rendimiento en la velocidad de escritura.
2. El rendimiento y capacidad de Lectura/Escritura escala linealmente a medida que nuevos nodos son agregados.
3. Agregar nuevo hardware no requiere tiempo de baja en servidores o perturbación a los usuarios.
4. Carece de puntos individuales de fallo. Cada nodo en el cluster es idéntico y puede correr en hardware de commodity. No hay un servidor maestro.
5. Los datos son automáticamente replicados a múltiples nodos – que pueden estar distribuidos entre *racks* y *datacenters*. Esto puede ser utilizado para asegurar alta disponibilidad a través de regiones geográficas.

6. CQL (Cassandra Query Language) es sintácticamente similar a SQL, simplificando la comprensión del lenguaje.

3.4.2 Limitaciones de C*

1. Su rendimiento queda sujeto a tener un cierto nivel de memoria disponible en los nodos o a la existencia de múltiples nodos.
2. Requiere de conocimiento previo de las consultas que se realizarán para diseñar el modelo de datos en torno a ellas y aprovechar el rendimiento potencial del sistema.
3. Debido a su modelo *Wide-Column* no es posible realizar ciertas transacciones y consultas de datos tradicionales, tales como Join.
4. Mover grandes cantidades de datos dentro y fuera de Cassandra puede ser problemático, pues cuando el conjunto de datos crece demasiado, recorrer distintas particiones y cargar estos datos en memoria puede provocar *timeouts*.

3.4.3 Componentes de C*

Los componentes clave de Cassandra son los siguientes:

- **Nodo –** Es el lugar donde los datos son almacenados
- **Datacenter –** Es una colección de nodos relacionados
- **Cluster –** Es un conjunto que contiene uno o más datacenters
- **Commit Log –** Mecanismo de recuperación ante caídas del sistema en Cassandra. Cada operación es escrita en el commit log.
- **Mem-table –** Estructura de datos residente en memoria. Después del commit log, los datos son escritos a la mem-table. Algunas veces, para una familia de columnas única, pueden haber múltiples mem-tables.
- **SSTable –** Es un archivo en disco al cual se vierten los datos desde la mem-table cuando sus contenidos llegan un valor de umbral determinado.

Finalmente, en base a los objetivos planteados inicialmente y el estudio realizado durante este capítulo se estima que una solución a al problema de gestión de datos identificado es la adopción del paradigma de arquitectura Kappa, con las herramientas Kafka, Spark y Cassandra en los niveles de captura, procesamiento y almacenamiento de datos, respectivamente. Apache Kafka es una plataforma para manejo de *streams* con alto volumen de datos, diseñada como un *Log* de transacciones distribuido. Apache Spark es un *framework* de procesamiento distribuido en clusters para computaciones paralelas de tareas. Así, estas herramientas pueden utilizarse para manejar el stream entrante de datos, realizar el procesamiento de las consultas y posteriormente hacer uso de cassandra para almacenar los resultados optimizados para tales consultas.

4. DESARROLLO

Para solucionar el problema identificado anteriormente en el capítulo de caso de estudio, se propuso la adopción de la arquitectura Kappa estudiada en la discusión bibliográfica, debido a su naturaleza de mayor simplicidad comparada a la arquitectura Lambda, la cual tiene una mayor complejidad en su desarrollo, *testing*, mantención, *debugging* y operación, pues básicamente funciona comunicando dos sistemas distribuidos complejos que deben trabajarse por separado. Así, para mantener el trabajo sobre un único *framework* se propuso utilizar la arquitectura Kappa.

4.1 ARQUITECTURA PROPUESTA

En la Fig. 7 se muestra un esquema con la arquitectura descrita y detallada en mayor profundidad, a la izquierda se presenta un conjunto de fuentes de datos que incluyen sensores, dispositivos y aplicaciones.

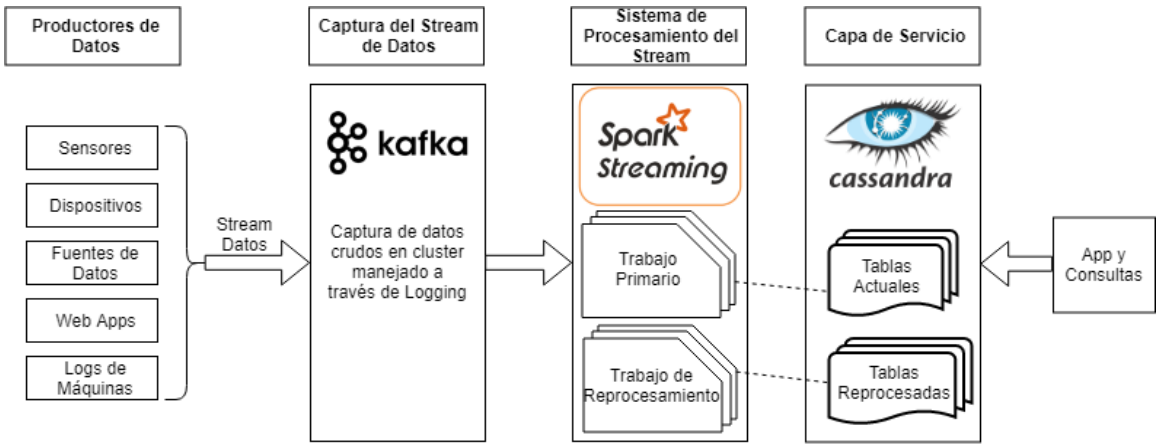


Figura 7: Arquitectura propuesta para el problema de gestión de datos del caso estudiado.

Como puede verse en la figura, se hizo uso de tres sistemas comunicándose entre sí: Apache Kafka para el *logging* o manejo del flujo de datos entrantes, que es recibido en un *cluster* que almacena los datos crudos de forma inmutable en un registro histórico, con el objetivo de recibir los datos en tiempo real desde el dispositivo adquisidor de los sensores y almacenarlos como un *dataset* canónico. Luego, se utiliza la tecnología Apache Spark para el procesamiento, agregación y análisis de datos, dando además la capacidad de realizar tareas automatizadas sobre el conjunto de datos almacenado y haciendo uso de sus conectores para comunicarse a los sistemas de almacenamiento. Finalmente, se tiene un *cluster* de Apache Cassandra que se utilizó como capa de servicio en la infraestructura, haciéndose cargo del almacenamiento de datos relevantes en consultas específicas de la aplicación. Para implementar este modelo de arquitectura se requiere de múltiples nodos interconectados para Cassandra y Spark, que tengan una capacidad de memoria suficiente para soportar las exigencias que ambos sistemas ponen sobre este recurso de los dispositivos, es decir, se debería utilizar servidores independientes para cada una de estas

tecnologías, pues debido a los requerimientos de memoria de los sistemas utilizados se volvería costoso tener en los servidores nodos con niveles de *hardware* muy altos.
Sin embargo, debido a las limitaciones de recursos que se enfrentan en el desarrollo de este trabajo, se limitó el prototipo del sistema a un enfoque sólo de la capa de servicio, es decir, utilizando la herramienta Apache Cassandra para almacenar el conjunto de datos procesados, sobre el cual se realizan las consultas de usuario. El conjunto de datos crudos se considerará en el diseño, pero de acuerdo a la arquitectura seleccionada debería ser almacenado en un sistema de Logging como Apache Kafka.

4.2 REQUISITOS DEL SISTEMA

El caso de estudio requirió un sistema de almacenamiento capaz de soportar lecturas y escrituras de alta velocidad, debido a las características del conjunto de datos asociado al manejo de datos provenientes de sensores. Además, el registro desde sensores tiene como objetivo llevar un archivo histórico de los datos recibidos, en otras palabras, las modificaciones a los datos luego de ser insertados al sistema son prácticamente nulas. Considerando las características presentes en el caso de estudio; Alta velocidad de inserción y lectura, baja prioridad de las operaciones de Update y Delete en la base de datos, y la capacidad de escalabilidad en el sistema, se pueden descartar con seguridad las bases de datos relacionales, por lo que el sistema a desarrollar pertenecerá al paradigma No Relacional – NoSQL –. A continuación se enumera el conjunto de requisitos estudiados:

4.2.1 Requisitos Funcionales:

- Almacenar datos provenientes de los paquetes de datos generados, de forma secuencial
- Permitir consultas con selección por rangos de fechas para valores asociados a sensores, tales como FFT, medias, RMS, máximos y mínimos.
- Calcular Peaks de FFT sobre todos los sensores en rango de fechas.
- Obtener Ciclos de Carga para un sensor dado y rango de fechas.

4.2.2 Requisitos No Funcionales:

- Permitir la inserción y lectura de datos a alta velocidad, con tal de soportar el flujo de datos proveniente de las redes de sensores.
- El sistema debe ser fácilmente escalable, para adaptarse al crecimiento de incorporar nuevas estructuras monitoreadas y sus respectivas redes de sensores.
- Entregar resultados compatibles con otros softwares y sistemas, permitiendo disponibilidad de consultas directas sobre el conjunto de datos total.
- Organizar los datos como registro histórico inalterable, manteniéndose como un conjunto de datos de sólo lectura.
- Tener la capacidad de respaldar los datos automáticamente, creando copias de los datos.

- Agilizar el procesamiento de consultas y análisis, reduciendo el tiempo requerido en las tareas realizadas.

Los requerimientos funcionales fueron extraídos y acotados según las necesidades del Caso de Estudio, por otro lado los requisitos no funcionales corresponden a características presentes en la mayoría de los sistemas software de mediana o gran envergadura, pues deben cumplirse ciertos estándares y políticas de calidad, respaldo de datos y eficiencia en el desarrollo software a nivel de mercado competitivo.

4.3 DISEÑO

Como se mencionó anteriormente, el diseño del modelo de datos en Cassandra es muy importante para sacar provecho a sus características y rendimiento. Para diseñar el sistema se utilizó la metodología propuesta por Chebotko (2015)[19], donde se describe un conjunto de etapas para diseñar sistemas en Cassandra, descritas en la Figura 8:

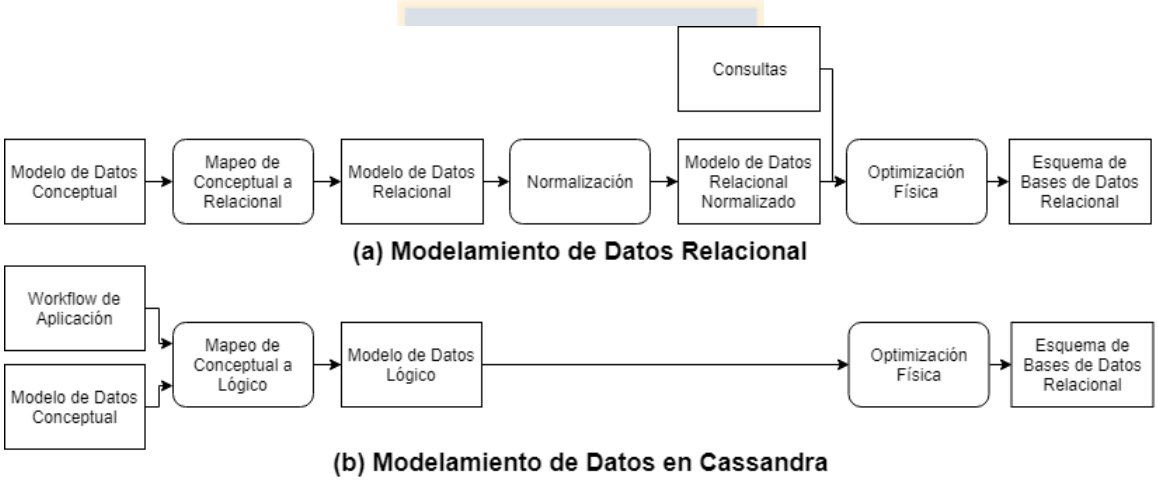


Figura 8: Modelamiento de datos tradicional versus la metodología adoptada.

Como se muestra en la Figura 8, el diseño de bases de datos en C* involucra no sólo los datos y sus características analizadas en el modelo conceptual, sino que pasa a primer plano el flujo de trabajo (workflow) de la aplicación y los patrones de acceso a los datos; en otras palabras, el modelamiento de datos en C* requiere una nueva forma de pensar el diseño, un cambio de paradigma que lleve el proceso de modelamiento desde un enfoque centrado únicamente en los datos, hacia un enfoque guiado también por las consultas que se requiera realizar sobre el conjunto de datos.

Otro punto a considerar a partir de la Fig. 8 es que las consultas fueron incluídas en el workflow de la aplicación al comienzo del diseño, mientras que en el modelamiento relacional se consideran sólo en la etapa final de optimización. También se deja fuera el proceso de normalización de datos, debido a que la naturaleza del sistema C* es la desnormalización de los datos. Se busca generar tablas que se comporten de forma eficiente

ante las consultas, además se diseña en torno a desnormalización pues actualmente el costo de almacenamiento es mucho menor que el de otros componentes hardware en la infraestructura.

4.3.1 Modelo de Datos Conceptual

El proceso de diseño comenzó con un diagrama conceptual de los datos, éste fue diseñado de acuerdo a las características de los datos que fueron capturadas en el Caso de Estudio y su construcción fue desarrollada de acuerdo al procedimiento utilizado en el modelado destinado a esquemas relacionales.

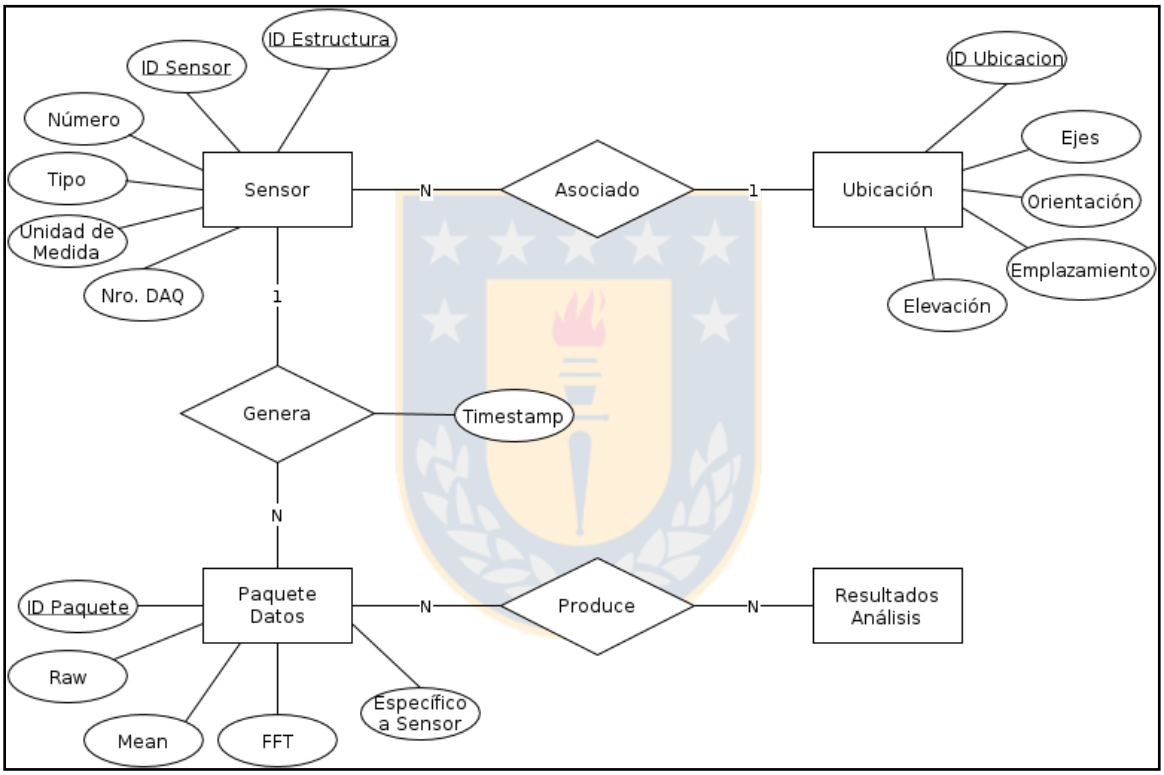


Figura 9: Modelo conceptual de los datos generales asociados a los sensores.

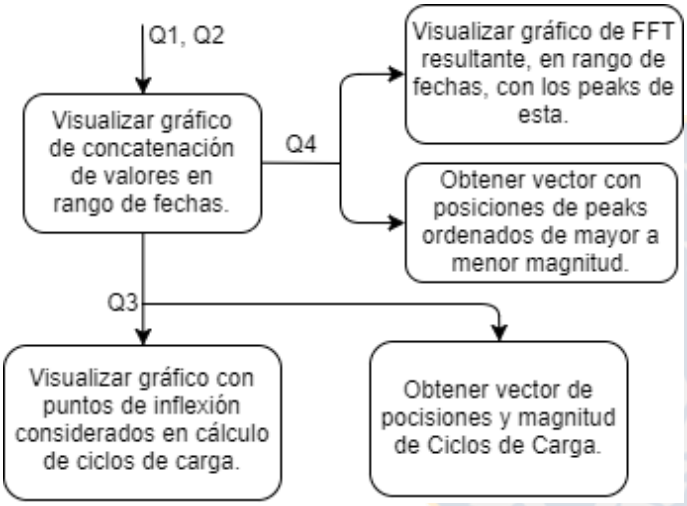
La figura 9 muestra el modelo conceptual diseñado en torno a la situación actual, definiendo inequívocamente las entidades, relaciones, cardinalidades, claves y otras restricciones, para los sensores y el conjunto de datos asociado en torno a ellos. Se construyó el diagrama base a los paquetes de datos generados, pero buscando lograr una mayor generalización de las estructuras de datos. Se agruparon los datos crudos y procesados dentro del mismo paquete para el sensor.

Además, se incluyó en el diagrama la entidad de Ubicación para los sensores, pero a un nivel de Big Data no tiene sentido almacenar las ubicaciones de los sensores, pues son fijas y presentan un crecimiento mucho más lento que los datos capturados por los dispositivos,

por lo que se dejó como contexto respecto del Caso de Estudio. Luego, considerando además las consultas descritas anteriormente y el procedimiento de Chebotko, se centrará el trabajo en las consultas entre las entidades de Sensor, Paquete Datos y Resultados Análisis.

4.3.2 Workflow de Aplicación

Para el desarrollo del prototipo se consideran sólo las consultas asociadas a las tareas de análisis de datos que se van a trabajar, es decir, la obtención de los datos necesarios para tales tareas desde la base de datos y el procesamiento de éstos para obtener los resultados. Se consideran las tareas de análisis descritas en los requerimientos funcionales capturados: Obtención de peaks diarios de FFT y de ciclos de carga, respectivamente.



Q1: Encontrar Medias de un sensor dado, en un rango de fechas. Orden de fecha desc.

Q2: Encontrar FFT de un sensor dado, en un rango de fechas. Orden de fecha desc.

Q3: Obtener Ciclos de carga para un sensor, en un rango de fechas.

Q4: Obtener peaks de FFT para una estructura, en un rango de fechas.

Figura 10: Workflow del prototipo, con las respectivas consultas especificadas a la derecha.

La figura 10 muestra el diagrama de flujo de trabajo para el prototipo, con las consultas de encontrar los valores de media y FFT como entrada inicial. Cada bloque representa un resultado percibido por el usuario a partir de la consulta, en el caso de Q1 y Q2 sólo pueden mostrarse gráficos con los valores obtenidos, pero las consultas asociadas al análisis de datos entregan además valores de vectores que pueden ser utilizados en otras tareas por el usuario especializado en análisis.

El objetivo de este modelo es entender y organizar las consultas sobre las cuales deben construirse las tablas que van a soportar tales consultas. Luego, en conjunto con el modelo conceptual se puede generar el conjunto de columnas que componen cada tabla, para ser utilizado en el modelo lógico.

4.3.3 Modelo lógico de Datos

La sección central en el modelamiento de datos para Cassandra es el modelo lógico, que toma el modelo conceptual y lo mapea al modelo lógico, basándose en las consultas definidas en el Workflow de aplicación.

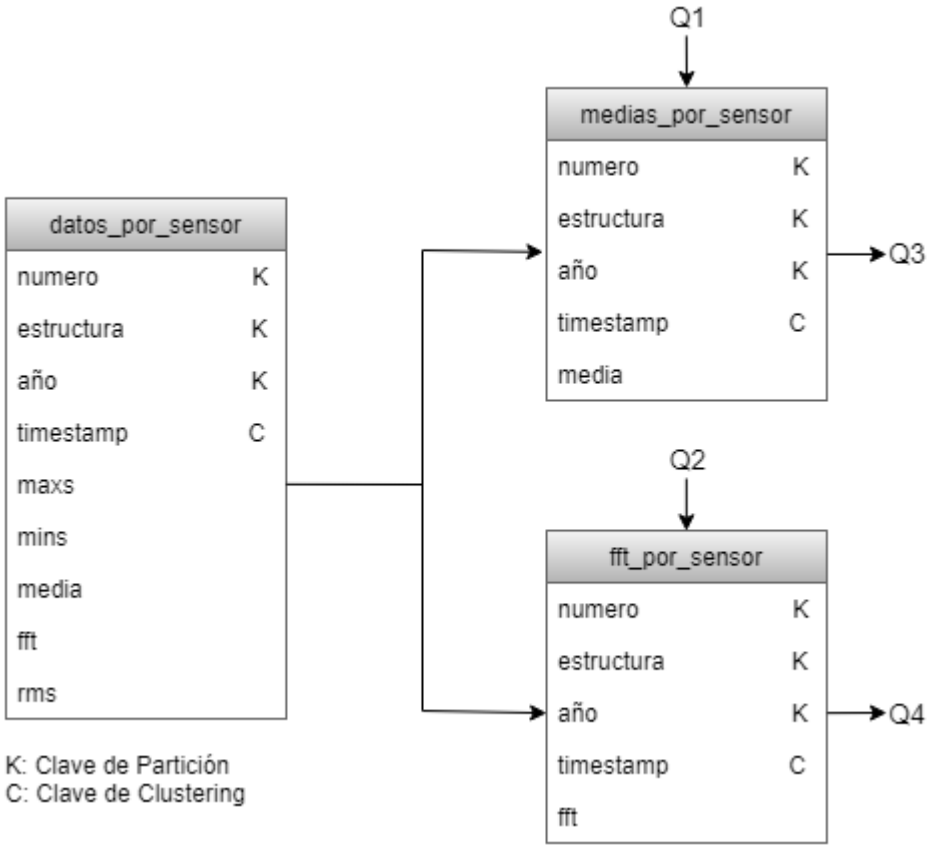


Figura 11: Modelo lógico de Datos.

El modelo lógico de la Fig.11 muestra una tabla general, con todos los datos (izq.), que se diseñó en base a las reglas de mapeo descritas por la metodología de Chebotko, generada directamente desde el diseño conceptual. Esta tabla luego se deriva a dos tablas específicas a las consultas que fueron enumeradas en el Workflow de aplicación. Para todas las tablas descritas se eligió una Clave de Partición compuesta por Número de Sensor, Nombre de Estructura y Año; Teniendo en mente la distribución equitativa de datos entre nodos, considerando además que no se acumulen demasiados datos en un único nodo, pues es más fácil agregar un nodo extra al cluster que aumentar la capacidad de nodos existentes. Esto, bajo el análisis de que si se utilizara sólo el número de sensor como Clave de Partición, cada nodo contendría la información de un sensor específico, lo que llevaría a un modelo desbalanceado que requeriría un crecimiento en el almacenamiento de cada nodo, en vez de distribuir los datos nuevos entre todo el cluster.

Ya que las consultas Q1 y Q2 son de acceso a datos, y las consultas Q3 y Q4 de análisis sobre el mismo grupo de datos, sólo es necesaria la creación de dos tablas en las cuales trabajará el prototipo. Sin embargo, también se incluirá en la sección de Experimentos las pruebas comparativas de rendimiento entre el acceso a los mismos datos en la tabla general y las tablas específicas.

4.3.4 Optimización Física del Modelo

Debido a las limitantes de recursos disponibles para realizar la implementación del prototipo, éste fue finalmente desarrollado en una máquina virtual. Por esto, no es posible realizar una optimización física en profundidad respecto al número de nodos, estimación de tamaños de partición, factores de replicación y políticas de consenso posibles. Sin embargo, se tomaron consideraciones extra en las tablas descritas anteriormente para mejorar el rendimiento de las consultas:

Datos por Sensor	Medias por Sensor
CREATE TABLE datos_por_sg (number int, structure text, year int, fecha timestamp, fft frozen<list<double>>, maxs frozen<list<double>>, mean double, mins frozen<list<double>>, rms double, PRIMARY KEY ((number, structure,year), fecha)) WITH CLUSTERING ORDER BY (fecha DESC);	CREATE TABLE medias_por_sensor (number int, structure text, year int, ts timestamp, mean double, PRIMARY KEY ((number, structure, year), fecha)) WITH CLUSTERING ORDER BY (fecha DESC);

Tabla 1: Código de Cassandra Query Language utilizado en la creación de las tablas usadas en la experimentación.

En la tabla se especifican los tipos de datos para cada columna definida en el modelo lógico y se ingresan las claves primarias compuestas por las claves de partición y de clustering descritas también en la lógica del modelo. Para las estructuras de datos del tipo lista se hace uso de colecciones estáticas de datos o *frozen*, este atributo serializa los múltiples componentes de la colección en un valor único. Esto optimiza el acceso y camino de datos al realizar consultas, en especial para el caso de colecciones con largo considerable como las frecuencias FFT. Las colecciones que no son de este tipo permiten la actualización de campos individuales y deteriora el rendimiento para acceder a estos datos. A nivel de cluster, se configuró un factor de replicación de 1, al no existir más nodos a los cuales enviar réplicas, esto indica que cada dato existe en única instancia. También se definió una política de consenso local única, es decir, las consultas son respondidas por un único nodo, que no busca validación de pares en el resultado.

5. EXPERIMENTOS Y RESULTADOS

A continuación se presentarán los experimentos realizados en torno al prototipo. Se inician las pruebas comparando el rendimiento de ambos diseños de tablas presentados anteriormente en la optimización física del modelo. Luego se realizaron pruebas para medir el rendimiento de consultas básicas y rutas de acceso a los datos. Finalmente se describe el trabajo realizado para desarrollar las tareas de análisis de datos consideradas como consultas en los requisitos funcionales y se midió su rendimiento comparativo ante la situación actual del caso de estudio, bajo el software Matlab.

El desarrollo y experimentación se realizó en una máquina virtual, pues una implementación completa del prototipo requeriría múltiples servidores, cada uno con un alto requerimiento de memoria RAM, lo que eleva su costo y escapa a los recursos disponibles para el trabajo. Dicha máquina virtual funcionó como nodo único para el sistema, lo que limitó la cantidad de recursos disponibles del sistema y se centró el trabajo en comparar el funcionamiento del estado actual de acceso a datos en el caso de estudio con el acceso a datos cargados en el sistema y a comprobar el funcionamiento de las tareas de análisis descritas en el diseño con el conjunto de datos contenidos en el sistema.

Para los experimentos de esta sección se utilizó un *dataset* correspondiente a 1 mes de los datos procesados provenientes de todos los sensores instalados en una única estructura, para los cuales el elemento más significativo es el resultado de la transformada de Fourier o FFT, el cual corresponde a un vector de 1025 valores que se obtiene para cada sensor en cada paquete de muestreo, los cuales cubren 5 minutos de tiempo. Es decir, para un día se deben ingresar al sistema 295,200 valores unitarios para cada sensor. La estructura considerada tenía instalados 123 sensores al momento de la captura de los datos utilizados, lo que resulta en un total de 36,309,600 de valores unitarios diarios y el conjunto de valores de frecuencias FFT para el mes completo corresponde a 11,255,976,000 valores unitarios. En promedio, la inserción de un paquete de datos al sistema tardó 1.6 milisegundos.

5.1 RENDIMIENTO DEL DISEÑO

La experimentación comenzó con las pruebas comparativas entre el modelo de datos general que incluye todos los datos del sensor en una misma tabla, versus el modelo específico a las consultas que contiene sólo los datos requeridos. Para el experimento se realizó la misma consulta en ambas tablas, utilizando la misma Clave Primaria que contenía Clave de Partición y Clustering idénticas. El único factor diferenciador entre los experimentos fue el modelo lógico utilizado como fuente de los datos.

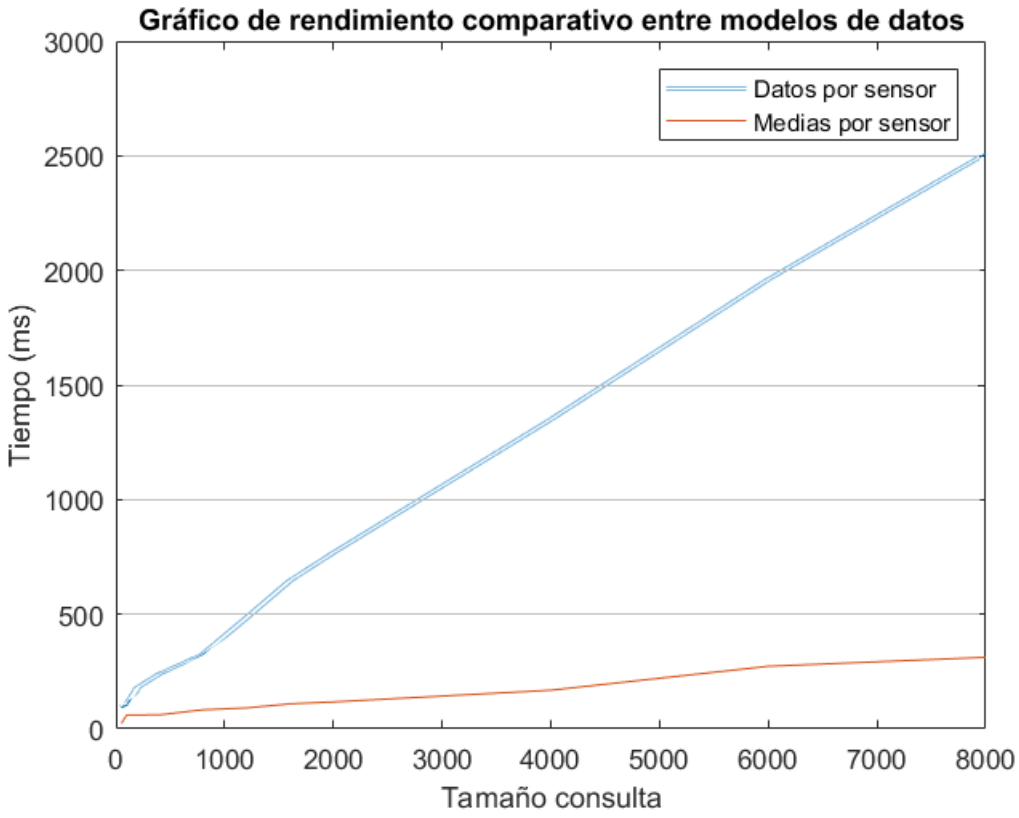


Figura 12: Gráfico de rendimiento de la tabla con todos los datos versus una tabla específica a la consulta.

La figura 12 muestra el rendimiento de los resultados del experimento, donde se ve una clara tendencia de crecimiento prácticamente lineal para la tabla “Datos por Sensor” y un crecimiento mucho menos pronunciado para la tabla “Medias por Sensor”. Evidentemente, es deseable un tiempo menor en las consultas de datos, y que este tiempo no crezca a tanta velocidad con conjuntos de resultados grandes, por lo tanto el comportamiento de la segunda tabla es mucho mejor que el de la primera.

5.2 EXPERIMENTOS

Los experimentos se realizaron bajo un mismo dominio de tamaño de la consulta, la que se fue incrementando acumulativamente para los días del mes, es decir, se fue aumentando el número de días incluidos en el conjunto de datos, partiendo con el primer día e incrementando el rango de días hasta incluir todos los días del mes en la última consulta. Sobre este conjunto se realizaron la mayoría de las consultas de los experimentos hechos.

Uno de los aspectos principales que interesa es comparar el rendimiento de la lectura de datos presente actualmente en el caso de estudio, que trabaja mediante la apertura y carga en memoria de archivos Matlab, versus el acceso a los datos a través de consultas realizadas sobre el prototipo del sistema implementado. A continuación se presentan los resultados obtenidos de los experimentos realizados para la comparación de acceso a datos:

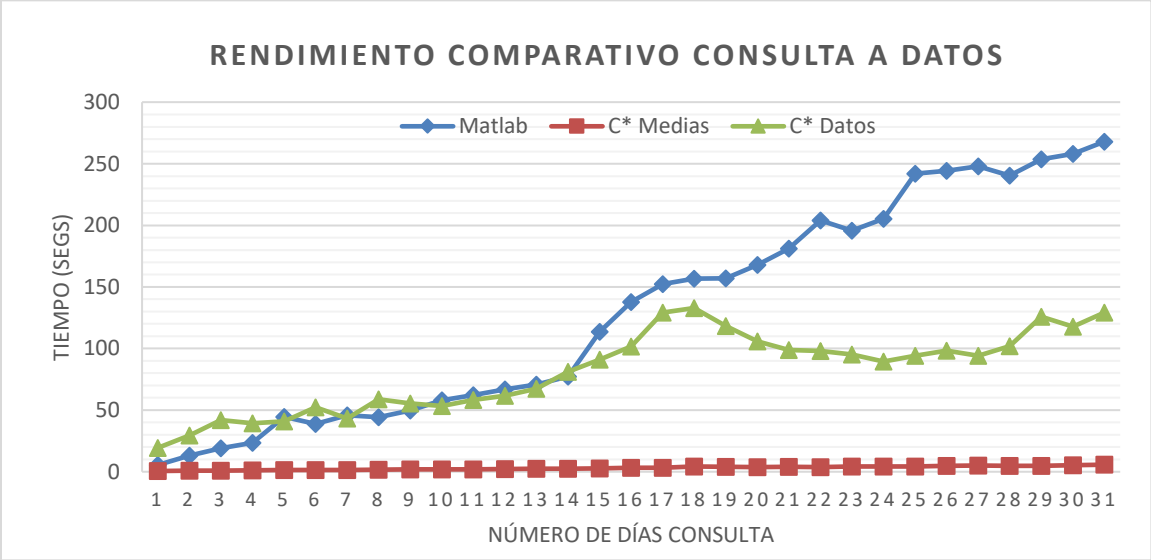


Figura 13: Gráfico de rendimiento en consulta a datos para el sistema actual y el prototipo implementado.

El experimento de la Figura 13 se llevó a cabo consultando datos relevantes para las dos tablas diseñadas y para los archivos Matlab. Puede observarse en los resultados que a partir de los 15 días como tamaño del conjunto, el software Matlab presenta un rendimiento peor que ambas tablas diseñadas.

Debido a que Matlab lee la estructura de datos completa para cada paquete de datos, existe una diferencia que debe considerarse al consultar la base de datos: fijar el rango de fechas en cada incremento y recorrer todo el conjunto de sensores o fijar el sensor y recorrer el rango de fechas.

Esta diferencia es evaluada en el experimento presentado en la Fig.14 a continuación:

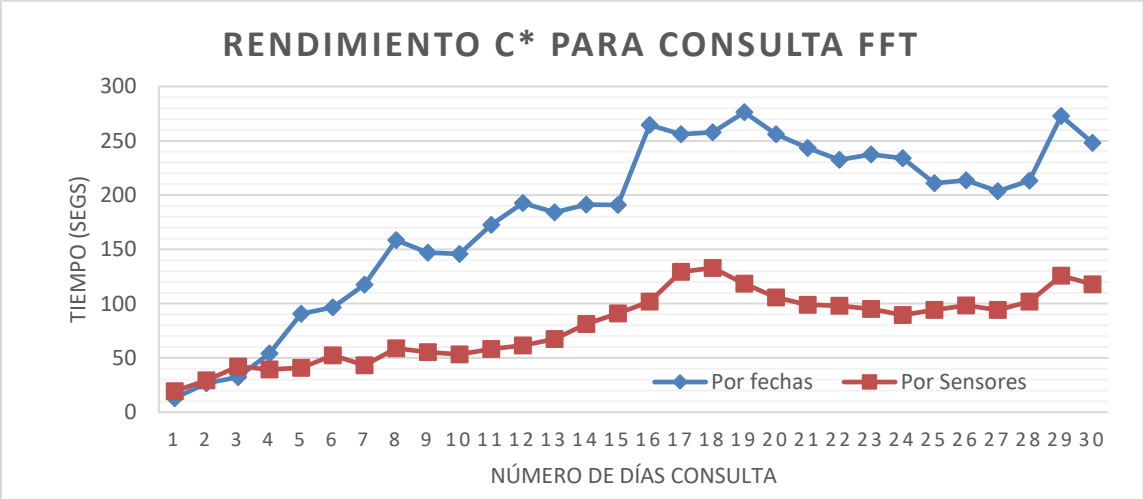


Figura 14: Rendimiento de consulta FFT incremental, recorriendo fechas y sensores.

La Figura 14 muestra el experimento de medir la diferencia en el rendimiento para una consulta sobre los mismos datos, variando solo el orden en que se recorren los elementos. En el primer caso se van fijando los intervalos de fechas y se recorren todos los sensores, mientras en el segundo se fijan los sensores y se recorren intervalos de fechas. Puede verse claramente que el rendimiento del primer caso es peor a partir de los 4 días de rango para la consulta.

El resultado anterior indica que es más eficiente moverse de partición una vez que ésta se ha terminado de utilizar y consultar el rango de fechas solicitado para cada una, por el contrario, saltar entre particiones múltiples veces vuelve mucho menos eficaz el acceso a los datos.

5.3 ANÁLISIS DE DATOS

En esta sección se realiza el trabajo de las tareas de análisis de datos presentadas como requisitos funcionales en la sección de Diseño. Éstas fueron implementadas en lenguaje Python, haciendo uso de las librerías SciPy y NumPy, para lograr funcionalidades concordantes a las de Matlab en el manejo de estructuras de datos.

El objetivo de este desarrollo es lograr un desempeño apropiado al nivel de complejidad del trabajo que se realiza sobre los datos, que mejore la situación actual del Caso de Estudio.

5.3.1 Especificación de Tareas de Análisis

A partir del estado actual de los procesos realizados en el caso estudiado, se describe a continuación el procedimiento de las tareas de análisis a realizar:

Tarea de análisis	**Input**	**Procedimiento**
Obtener peaks diarios de FFT	Rango de fechas	Para obtener los peaks de FFT se accede a los valores de FFT para cada día en el rango de fechas, se realiza un promedio sobre estos valores superpuestos. Luego, sobre este resultado se aplica un algoritmo de detección de peaks ajustando los parámetros al conjunto de datos.
Obtener Ciclos de Carga	Valor umbral Número Sensor Rango de fechas	Para obtener los ciclos de carga se realiza una concatenación de los valores de medias de deformación entre el rango de fechas para el sensor seleccionado, luego se recorre esa concatenación buscando valores de diferencia entre máximos y mínimos relativos que superen un umbral dado.

Tabla: Especificaciones de las tareas de análisis de datos que se implementaron.

5.3.2 Detección de Peaks de FFT

Para la detección de peaks diarios de la Transformada de Fourier se deben realizar los siguientes pasos principales:

1. Obtener un promedio de las FFT por cada paquete de datos, o sea, los valores de FFT para todos los sensores, dentro del rango de fechas ingresado como entrada.
2. Superponer todos los promedios obtenidos para el rango de fechas.
3. Calcular la envolvente de las señales superpuestas, es decir, los valores máximos de entre todas las FFT en cada punto del dominio de frecuencias.
4. Sobre la señal envolvente resultante, aplicar un algoritmo de detección de peaks
5. (Inicialmente) Ajustar los parámetros de detección de peaks para el conjunto de datos

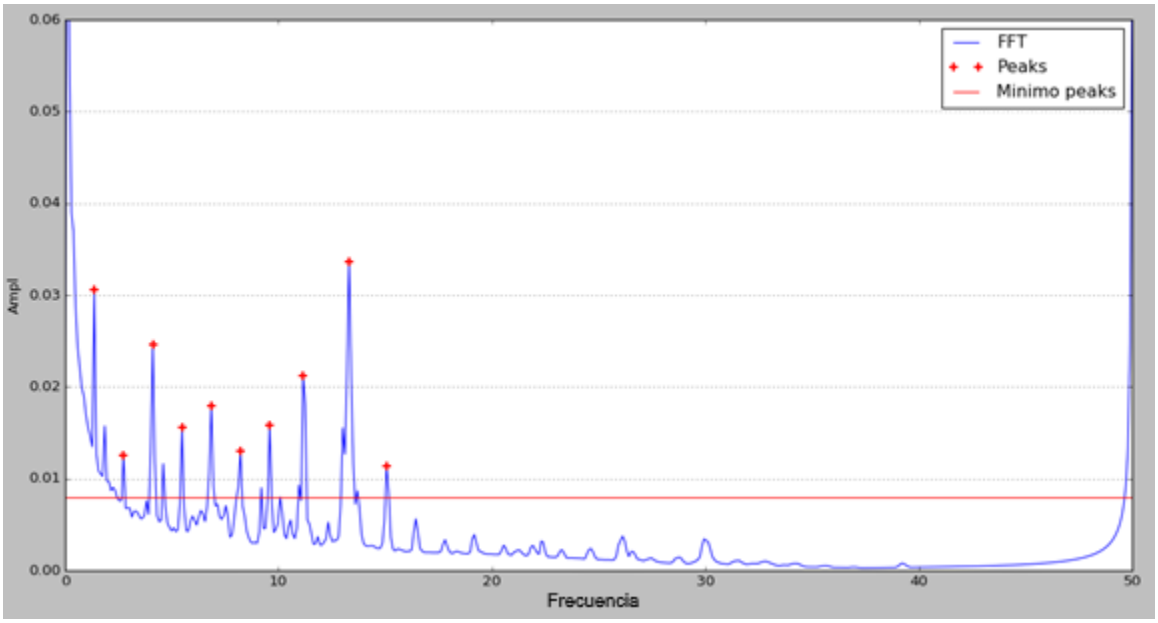


Figura 15: Ejemplo de resultado obtenido del cálculo de peaks de FFT en Python

La Figura 15 muestra una instancia de resultado para la tarea de análisis sobre los datos consultados en C*. En los experimentos para esta tarea interesa comparar el rendimiento del procedimiento actual del Caso de Estudio ante lo implementado en el prototipo, para ello se realizaron pruebas sobre los mismos conjuntos de datos, variando la plataforma y la construcción de los algoritmos, pero manteniendo el procedimiento descrito anteriormente.

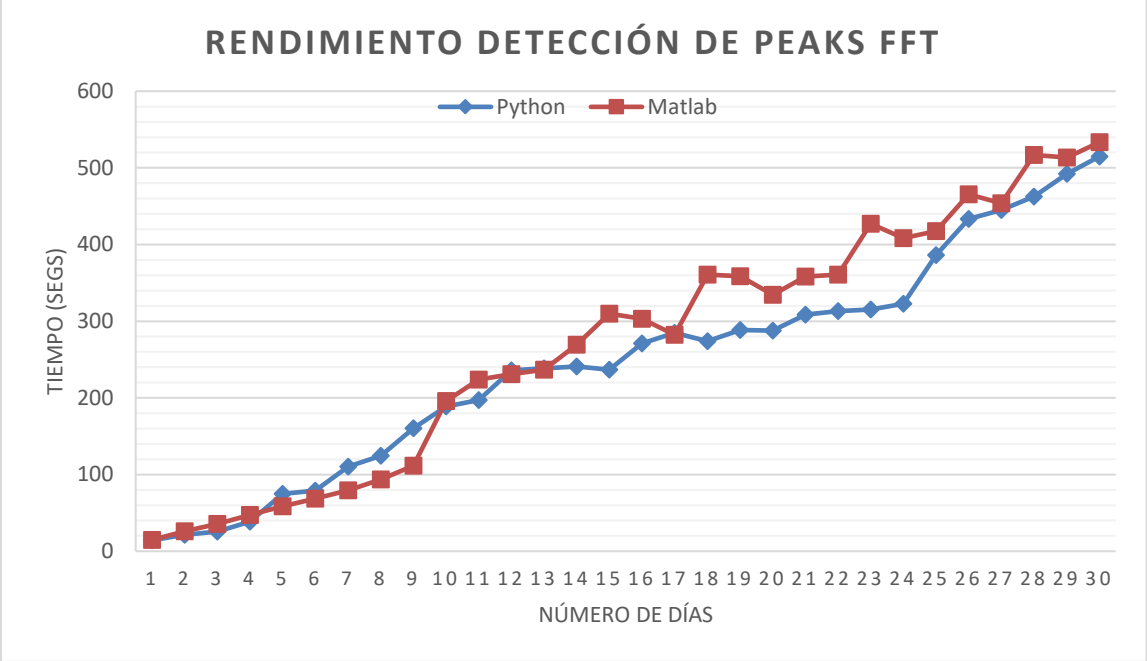


Figura 16: Rendimiento de la consulta implementada versus funcionamiento actual en Matlab.

En la Fig.16 puede apreciarse un comportamiento similar entre los procedimientos, siendo el implementado en el prototipo levemente mejor, pero con una tendencia de rendimiento prácticamente idéntico.

5.3.3 Obtener de Ciclos de Carga

Los ciclos de carga se obtienen, dado un sensor específico, concatenando los valores de medias de deformación entre el rango de fechas de interés y sobre ese resultado actualmente en el caso de estudio se aplica un algoritmo desarrollado para recorrer linealmente la señal resultante y detectar saltos entre máximos y mínimos relativos que sean mayores a un valor de umbral entregado.

Para implementar la tarea en Python se usa el algoritmo Ramer–Douglas–Peucker [20][21] – RDP, por sus siglas –, el cual genera una señal simplificada que filtra el ruido y construye fragmentos de líneas rectas entre los máximos y mínimos locales. Luego, a partir de dicha señal simplificada, se identifican los puntos de inflexión, que después son evaluados para encontrar distancias superiores al valor umbral y así encontrar los ciclos de carga.

Debido a la naturaleza del algoritmo, el cálculo de ciclos funciona como una aproximación al resultado obtenido de forma lineal, puesto que depende del ajuste de parámetros del algoritmo RDP y del cálculo de puntos de inflexión.

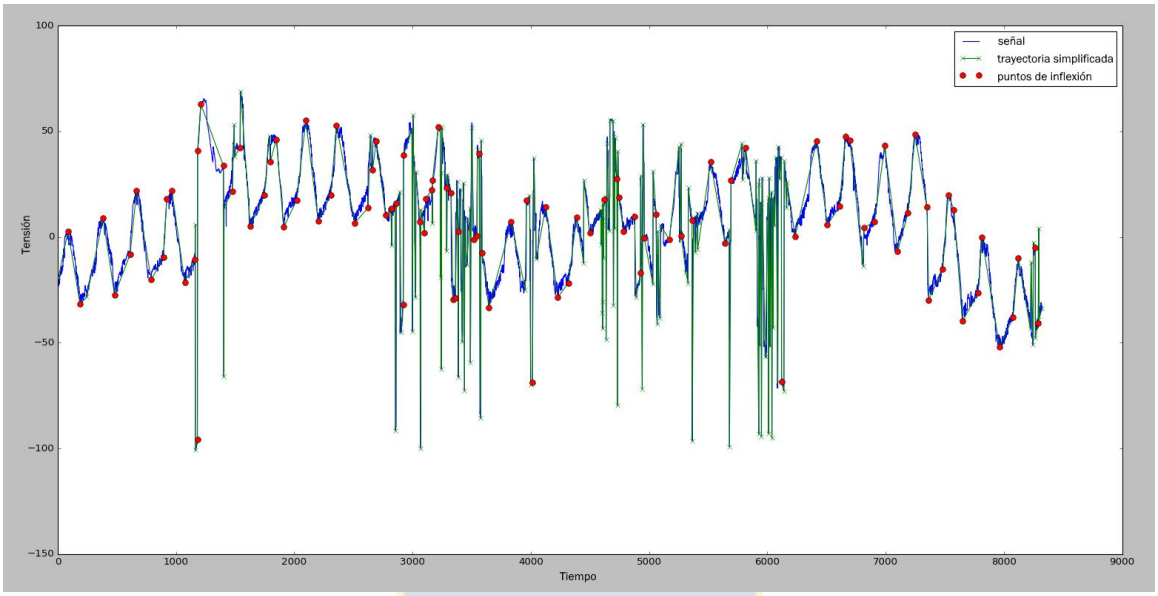


Figura 17: Ejemplo de resultado obtenido por el algoritmo implementado para el conteo de ciclos de carga.

En la figura 17 puede verse un resultado del cálculo de los ciclos de carga, en donde se grafica la señal analizada, la trayectoria simplificada obtenida desde el algoritmo RDP y los puntos de inflexión identificados a partir de dicha trayectoria. De este resultado puede aseverarse que la solución desarrollada se aproxima lo suficiente al resultado obtenido en Matlab, pues ambos entregan 23 ciclos de carga para el conjunto de datos analizado, sin embargo, los procedimientos seguidos son muy diferentes e interesa conocer el rendimiento de cada caso.

En los experimentos se compararon dos casos frontera, el cálculo de los ciclos de carga para todos los sensores y para un sensor único. Esto con la intención de comprobar el rendimiento en ambos casos, buscando encontrar las fortalezas y debilidades de ambas implementaciones en los casos extremos. Sin embargo, en la práctica el cálculo se realiza sobre un único sensor, pues correspondería en el caso de estudio a consultas realizadas a través de la plataforma web para revisar el comportamiento en el tiempo de un único sensor.

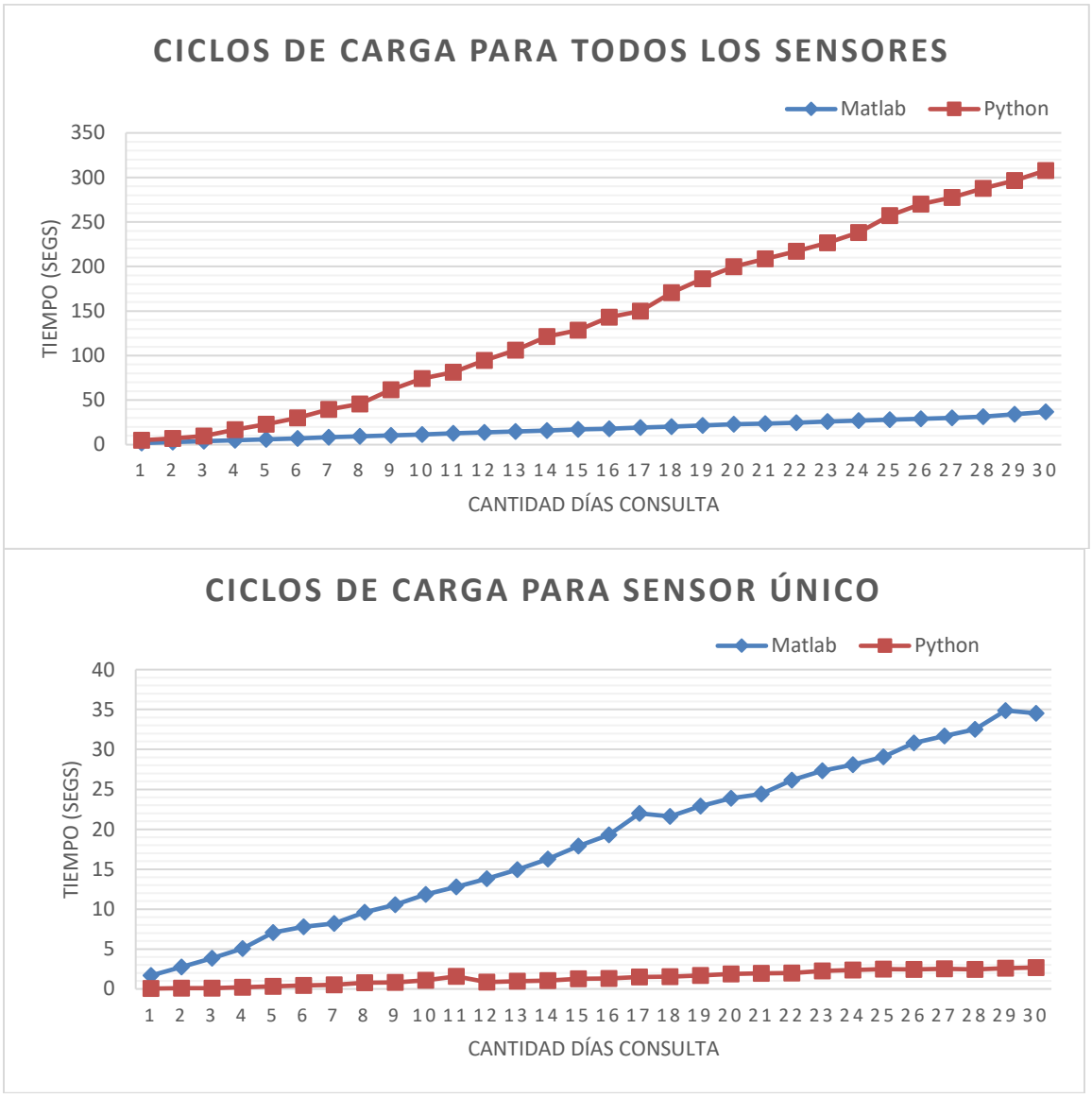


Figura 18: Rendimiento del procedimiento de cálculo de ciclos de carga, aplicado sobre todo el conjunto de sensores y sobre un sensor único, respectivamente.

La figura 18 muestra los resultados de realizar la obtención de ciclos de carga para todos los sensores y para un sensor único. El procedimiento incluye ambos la obtención de datos y los cálculos realizados sobre éstos. Puede observarse a partir de los resultados en la figura que en cada caso de frontera hay un rendimiento marcadamente favorable para cada sistema respectivo. Para el caso de procesar todos los sensores existe mejor rendimiento en Matlab y para un único sensor en Python, respectivamente. Esto indica que Matlab toma ventaja al momento de procesar todos los sensores pues sólo requiere abrir el archivo para cargar en memoria todos los sensores, mientras que el sistema implementado requiere repetir las consultas para cada sensor. Por otro lado, el comportamiento para un sensor único es mucho mejor en el prototipo, pues acota el resultado únicamente al sensor requerido,

mientras que Matlab debe seguir cargando en memoria todos los datos aunque sólo se esté solicitando el análisis de un sensor único.

La principal causa de la gran diferencia en rendimiento proviene del funcionamiento de la arquitectura subyacente de los sistemas. El acceso a los datos contenidos en archivos Matlab requiere las acciones de apertura del archivo, cargar toda la estructura de datos del archivo – incluyendo los datos que no se utilizarán – en la memoria de la máquina, para luego operar sobre la información cargada en memoria. Por otro lado, el acceso al prototipo del sistema consulta directamente por los datos requeridos únicamente, lo que acota el conjunto de datos a recorrer y agiliza el acceso a la información. Esto quedó demostrado en los resultados obtenidos en los experimentos realizados.

Un aspecto importante a considerar en esta comparación de rendimiento, es el acceso a las particiones de Cassandra. Existe una diferencia entre fijar un rango de fechas y recorrer todos los sensores midiendo el tiempo resultante para tal rango, versus fijar un sensor e ir incrementando el rango de fechas. Esta diferencia se debe considerar, pues la arquitectura de Cassandra almacena en espacios contiguos de memoria y disco los datos pertenecientes a una misma partición, optimizando la lectura de datos continuos en ella. Ya que el modelo lógico del prototipo utiliza como Clave de Partición los valores de número de sensor, estructura y año, consultar los datos recorriendo el conjunto de sensores múltiples veces ocasiona un continuo salto entre particiones que afecta considerablemente el rendimiento.

5.4 RESUMEN DE RESULTADOS

Considerando el objetivo principal en el trabajo de este proyecto, se puede concluir de los resultados que se presenta una alternativa viable de sistema de almacenamiento. El sistema se comporta considerablemente mejor en consultas sobre subconjuntos de datos o valores individuales. Sin embargo, se debe considerar que la herramienta sólo es adecuada para el almacenamiento de resultados del procesamiento de los datos crudos, pero no para los datos crudos en sí. Ya que desde los resultados con estructuras de datos tipo colección en los experimentos, se puede concluir que la baja en el rendimiento para el tamaño de estructura utilizada se haría insostenible con una que contenga los datos crudos, que presentaría un tamaño de orden muy superior. Además, la necesidad de construir las tablas o familias de columnas de acuerdo a las consultas implica duplicidad de datos y bajo el escenario de un cluster de varios nodos, si la familia de columnas no se distribuye correctamente entre las particiones puede traer problemas de rendimiento considerables.

Por último, para las tareas de análisis desarrolladas, se obtuvo en la identificación de peaks de FFT un rendimiento prácticamente idéntico al del estado actual del caso de estudio. La tarea de identificación de ciclos de carga obtiene un rendimiento mejor cuando la consulta se hace sobre los datos para un único sensor, pero no para todo el conjunto de sensores. Debido a que en el diseño del sistema se hicieron particiones por sensor, la familia de columnas no está optimizada para consultar múltiples veces sobre todos los sensores.

6. CONCLUSIONES

Los objetivos propuestos para este trabajo se han cumplido; se obtuvo una alternativa de arquitectura que se ajusta a la situación actual y mejora aspectos de rendimiento en ésta, se cumplieron los objetivos específicos de recopilar un trasfondo teórico y aplicar estos conocimientos para planificar, diseñar e implementar un prototipo de sistema de almacenamiento.

La alternativa de arquitectura propuesta fue seleccionada a través de la recopilación de información, para abordar el problema identificado en el caso de estudio. Posteriormente, en base a la información reunida, se seleccionó una herramienta adecuada al problema identificado, sobre la que se realizó el desarrollo. Luego, se continuó con el procedimiento de planificación, diseño e implementación del prototipo de la herramienta, identificando requisitos del sistema según el problema y caso estudiados. Se realizó un diseño del sistema de acuerdo a una metodología ajustada al paradigma de la herramienta seleccionada. A continuación, se implementó el almacenamiento y tareas para los datos correspondientes a un mes de información, bajo el modelo de datos diseñado. Finalmente, los experimentos realizados mostraron mejoras considerables en los casos en que se desea obtener un único valor del conjunto de datos o en consultas optimizadas. Esto ayuda a concluir que comparativamente, la alternativa propuesta es una buena opción a nivel de gestión de datos y almacenamiento frente a la situación actual del caso de estudio.

El principal problema que presenta el caso de estudio es la gestión de datos, debido a que se almacenan los paquetes de datos capturados directamente en archivos del software especializado Matlab, estos archivos son almacenados dentro de repositorios en la nube o simplemente en discos duros físicos. Esto no sólo hace insostenible el crecimiento y escalabilidad de la organización, sino que también crea limitaciones de rendimiento en el acceso a la información, como quedó demostrado en los resultados de este informe, la carga de archivos a memoria requiere mucho más tiempo que una consulta a una base de datos como el prototipo de sistema implementado.

En la experimentación se encontró evidencia de mejoras en múltiples casos al comparar el prototipo desarrollado respecto de la situación actual. Sin embargo, también se descubrió que existe un rendimiento peor en otros casos específicos. Esto se debe a que Matlab posee una ventaja para consultas en que se haga uso de todos los sensores, debido a su estructura de datos cargada en memoria. Esta ventaja se hace mayor debido al diseño del modelo utilizado, ya que se dividen las particiones por sensor, lo que ralentiza las consultas que requieran saltar múltiples veces entre estas particiones. Además, en una implementación completa dichas particiones estarían en diferentes nodos del cluster, lo que haría aún mayor la desventaja para Cassandra. Esto puede enfrentarse extendiendo el modelo para estas consultas específicas que requieren optimización para obtener datos desde todos los sensores, dejándolos juntos en una misma partición y buscando otra forma de fraccionar el conjunto de datos.

En base a lo anteriormente expuesto, puede considerarse el sistema seleccionado como una buena alternativa a lo implementado actualmente en el caso de estudio, pues mejora el rendimiento considerablemente en múltiples situaciones. Sin embargo, debe tenerse en cuenta que existe bastante duplicidad de datos y que se transa un buen rendimiento por mayor uso de disco y memoria. Es decir, el prototipo desarrollado puede calificarse como una buena alternativa de almacenamiento, siempre que la organización esté dispuesta a invertir en servidores que entreguen los recursos suficientes para manejar el volumen de datos a escala completa.

Otro resultado relevante del trabajo realizado es la considerable diferencia en rendimiento del sistema frente a una diferencia sólo en el modelo de datos. Esto demuestra la importancia del diseño en el desarrollo de un sistema que utilice la herramienta seleccionada y de una correcta captura de requerimientos al inicio del proceso de definición del proyecto.

Como trabajo futuro, queda pendiente el trabajo sobre el conjunto de datos crudos con un sistema de archivos inmutable en la capa de captura. También pueden realizarse pruebas con modelos de tablas especializados a otras consultas o diseñados para optimizar el rendimiento de, por ejemplo, el acceso a múltiples sensores en una misma partición. Además, se puede medir el rendimiento que logra el sistema con distribución de datos entre nodos, pues una de las características de Cassandra es ser linealmente escalable, por lo que es posible que con el diseño presentado en este informe se pueda mejorar los resultados de los experimentos que obtuvieron rendimiento inferior, al aumentar la cantidad de nodos e implementar procesamiento distribuido de consultas y procedimientos de análisis.

7. REFERENCIAS

[1] Cheok, A. (2016). **Social Impact of Hyperconnectivity.** In: Hyperconnectivity. Human–Computer Interaction Series. Springer, London.

[2] Stuart, J. & Baker, A. (2013). **Undefined by Data: A Survey of Big Data Definitions**. University of St. Andrews, UK.

[3] Rabl, T., Sadoghi, M. & Jacobsen H. (2012). **Solving Big Data Challenges for Enterprise Application Performance Management**. University of Toronto, Canada.

[4] Beyer, M. A. & Laney, D. **The importance of big data: A definition**. *Stamford, CT: Gartner*, 2012

[5] Hwang, K., Fox, G. & Dongarra, J. (2012). **Distributed and cloud computing: from parallel processing to the internet of things.** Waltham, MA: Morgan Kaufmann.

[6] Özsu, M. T., & Valduriez, P. (2011). **Principles of distributed database systems.** Springer Science & Business Media.

[7] DB-Engines Ranking (2018). Web: https://db-engines.com/en/ranking

[8] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). **The hadoop distributed file system**. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium*.

[9] Moniruzzaman, A. & Hossain, S. (2013). **NoSQL Database: New Era of Databases for Big Data Analytics – Classification, Characteristics and Comparison.**

[10] MongoDB Inc. (2016). **Top 5 Considerations When Evaluating NoSQL Databases.** *A MongoDB White Paper.*

[11] Gilbert, Seth, & Nancy Lynch (2012). **Perspectives on the CAP Theorem**. *Massachusetts Institute of Technology, USA.* Web http://hdl.handle.net/1721.1/79112

[12] Browne, J. (2009). **Brewer's CAP Theorem.** Web: http://www.julianbrowne.com/article/brewers-cap-theorem

[13] Apache Cassandra. Web: https://cassandra.apache.org/

[14] End Point Corporation. (2015). **Benchmarking Top NoSQL Databases, Apache Cassandra, CouchBase, HBase, and MongoDB.**

[15] Marz, N., & Warren, J. (2013). **Big Data: Principles and best practices of scalable realtime data systems.**

[16] Kiran, M., Murphy, P., Monga, I., Dugan, J., & Baveja, S. (2015). **Lambda architecture for cost-effective batch and speed big data processing.** In *Big Data (Big Data), 2015 IEEE International Conference.* IEEE.

[17] Krepps, J. (2014). **Questioning the Lambda Architecture.** Web: https://www.oreilly.com/ideas/questioning-the-lambda-architecture

[18] Kappa Architecture. Web: http://milinda.pathirage.org/kappa-architecture.com/

[19] Chebotko, A., Kashlev, A., & Lu, S. (2015). **A big data modeling methodology for Apache Cassandra**. In *Big Data (BigData Congress), 2015 IEEE International Congress.*

[20] Ramer, U., Douglas, D., Peucker, T., (1973). **Ramer-Douglas-Peucker poligonal simplification algorithm**. Enlace: https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm.html

[21] Hershberger, J., & Snoeyink, J. (1994). **An O (n log n) implementation of the Douglas-Peucker algorithm for line simplification.** In *Proceedings of the tenth annual symposium on Computational geometry*. ACM.

[22] Cachin, C. (2016). **Architecture of the Hyperledger blockchain fabric**. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*.

[23] Han, J., Haihong, E., Le, G., & Du, J. (2011). **Survey on NoSQL database**s. In *Pervasive computing and applications (ICPCA*). IEEE.

[24] Van der Veen, J. S., Van Der Waaij, B., & Meijer, R. J. (2012). **Sensor data storage performance: SQL or NoSQL, physical or virtual.** In *Cloud computing (CLOUD), 2012 IEEE 5th international conference on* (pp. 431-438). IEEE.

[25] Hadjigeorgiou, C. (2013). **Rdbms vs nosql: Performance and scaling comparison**. *MSc in High*.

[26] Kreps, J. (2017). **It's Okay To Store Data in Apache Kafka.** Web: https://www.confluent.io/blog/okay-store-data-apache-kafka/