

UNIVERSIDAD SIMÓN BOLÍVAR

DECANATO DE ESTUDIOS DE POSTGRADO

COORDINACIÓN DE COMPUTACIÓN

INGENIERÍA DE LA COMPUTACIÓN

TRABAJO DE GRADO

**JULIET: FRAMEWORK PARA EL ANÁLISIS DEL CONSUMO ENERGÉTICO DE BASE DE DATOS EN SISTEMAS DISTRIBUIDOS**

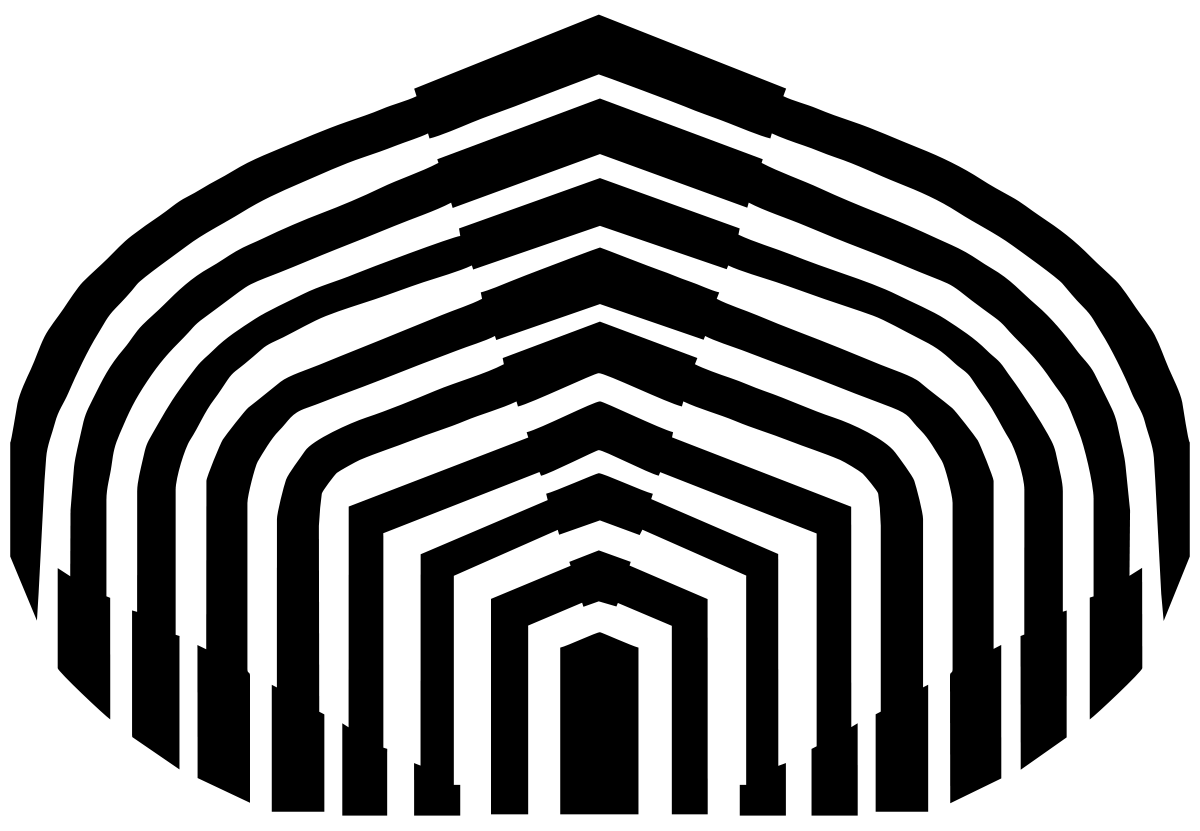
Por

Eros Cedeño

Septiembre de 2024

USB

|  |  |  |  |
| --- | --- | --- | --- |
| 2024 | EROS CEDENO | JULIET: FRAMEWORK PARA EL ANÁLISIS DEL CONSUMO ENERGÉTICO DE BASE DE DATOS EN SISTEMAS DISTRIBUIDOS |  |



UNIVERSIDAD SIMÓN BOLÍVAR

DECANATO DE ESTUDIOS DE POSTGRADO

COORDINACIÓN DE COMPUTACIÓN

INGENIERÍA DE LA COMPUTACIÓN

**JULIET: FRAMEWORK PARA EL ANÁLISIS DEL CONSUMO ENERGÉTICO DE BASE DE DATOS EN SISTEMAS DISTRIBUIDOS**

Trabajo de grado presentado a la Universidad Simón Bolívar Por

Eros Cedeño

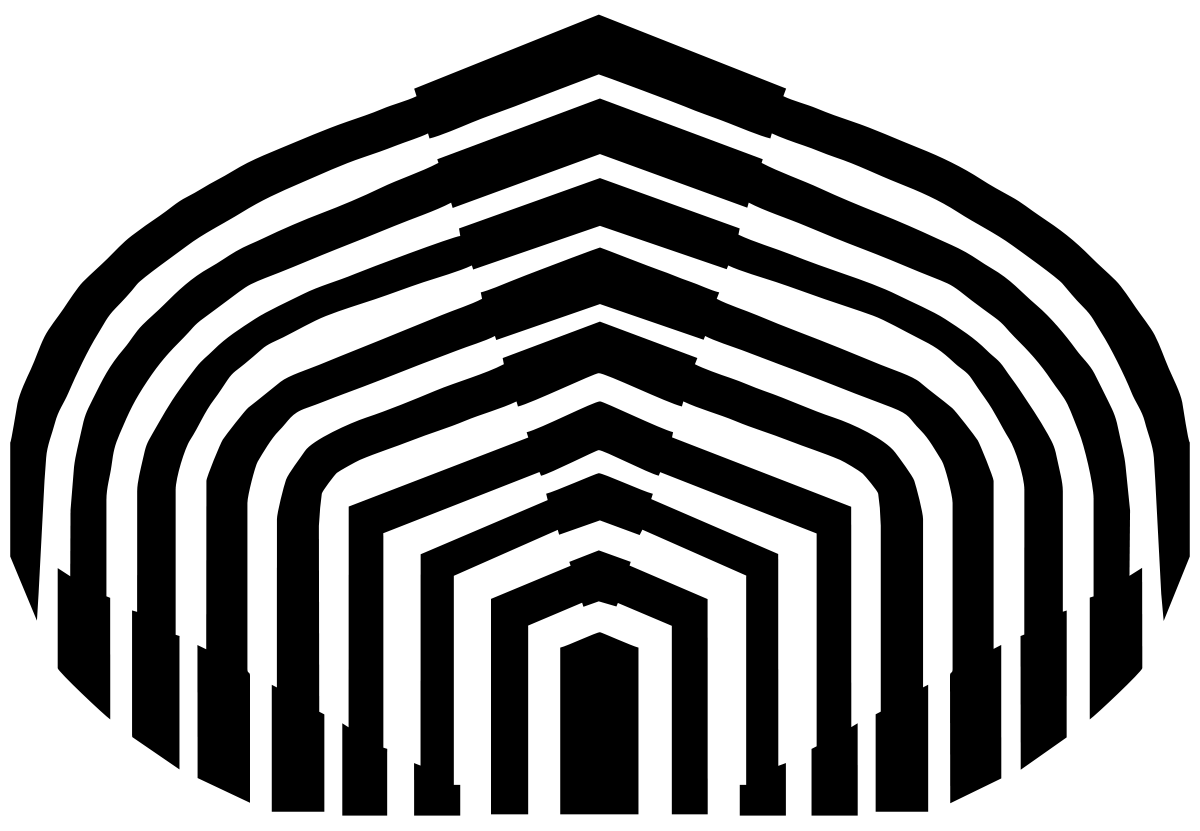
Como requisito parcial para optar al grado académico de

Ingeniero en Computación

Con la asesoría de los profesores

Yudith Cardinale

Septiembre de 2024



UNIVERSIDAD SIMÓN BOLÍVAR

DECANATO DE ESTUDIOS DE POSTGRADO

COORDINACIÓN DE COMPUTACIÓN

INGENIERÍA DE LA COMPUTACIÓN

**JULIET: FRAMEWORK PARA EL ANÁLISIS DEL CONSUMO ENERGÉTICO DE BASE DE DATOS EN SISTEMAS DISTRIBUIDOS**

Por: Eros Cedeño

Carnet: 16-10216

Este Trabajo de Grado ha sido aprobado en nombre de la Universidad

Simón Bolívar por el siguiente jurado examinador:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JURADO

Presidente(a)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JURADO JURADO

Miembro Principal Miembro Principal

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Yudith Cardinale

Miembro principal-tutor

Septiembre de 2024

# DEDICATORIA

A la comunidad de investigadores, desarrolladores y administradores de sistemas que buscan un futuro más sostenible en la era de la información:

Este trabajo de grado se dedica a ustedes, con la esperanza de que el framework Juliet se convierta en una herramienta valiosa y un punto de partida para optimizar el consumo energético de las bases de datos en sistemas distribuidos.

A mis profesores y mentores:

Gracias a las Profesoras Yudith Cardinale, Dennise Munante y Ana Aguilera por su oportuno apoyo y aliento durante el desarrollo de este proyecto. Sus conocimientos y experiencia fueron mi brújula en este camino.

A mi familia y amigos:

A mis padres y hermanas, su comprensión, paciencia y apoyo incondicional me permitieron dedicar el tiempo y esfuerzo necesarios para completar este trabajo.

A mí mismo:

Por la perseverancia, la dedicación y el compromiso con la búsqueda de soluciones innovadoras para los desafíos de la computación moderna.

# AGRADECIMIENTOS

La realización de este proyecto de grado ha sido posible gracias al apoyo y colaboración de diversas personas e instituciones a quienes quiero expresar mi más sincero agradecimiento:

**A mi tutora de tesis, Yudith Cardinale,** por su invaluable guía, asesoramiento y paciencia durante todo el proceso de investigación. Su experiencia y conocimiento fueron fundamentales para el desarrollo exitoso de este proyecto.

**A las profesoras Denisse Munante y Ana Aguilera,** por sus valiosas sugerencias, guías, comentarios y retroalimentación, que contribuyeron significativamente a mejorar la calidad de este trabajo.

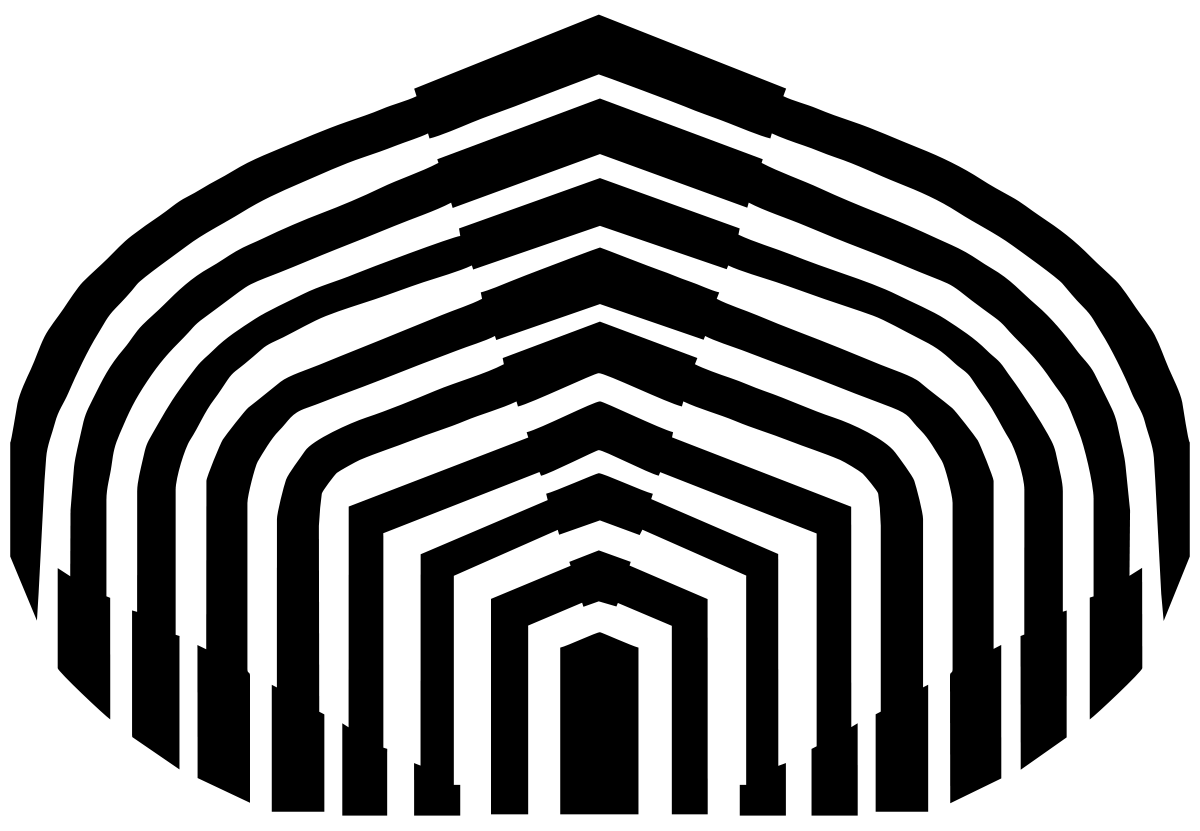
**A mi familia,** por su comprensión, paciencia y apoyo incondicional durante todo el proceso. Su amor y aliento me motivaron a seguir adelante en los momentos difíciles y celebrar los éxitos alcanzados.

**A la comunidad de software libre,** por compartir su conocimiento, herramientas y recursos de forma gratuita. Su trabajo colaborativo ha sido fundamental para el avance de la tecnología y la innovación.

**A mis profesores de la excelentísima Universidad Simón Bolívar**, por su impecable trabajo en la formación y recursos que aportan al crecimiento de sus estudiantes.

Finalmente, quiero expresar mi más profundo agradecimiento a todas las personas que han inspirado mi interés en la informática y la investigación. Su pasión y dedicación han sido la base sobre la cual he construido mi propia trayectoria académica y profesional.

¡Muchas gracias a todos!



UNIVERSIDAD SIMÓN BOLÍVAR

DECANATO DE ESTUDIOS DE POSTGRADO

COORDINACIÓN DE COMPUTACIÓN

INGENIERÍA DE LA COMPUTACIÓN

**JULIET: FRAMEWORK PARA EL ANÁLISIS DEL CONSUMO ENERGÉTICO DE BASE DE DATOS EN SISTEMAS DISTRIBUIDOS**

Por: Eros Cedeño

Carnet: 16-10216

Tutor: Yudith Cardinale

Septiembre 2024

# RESUMEN

La masificación y capacidad de cómputo de las computadoras ha crecido exponencialmente, y junto a este crecimiento los sistemas demandan una mayor cantidad de recursos para su correcto funcionamiento. Se propone Juliet Framework, un framework para el análisis del consumo energético en sistemas distribuidos en la red que requieren una recolección a altas frecuencias de muestreo. Juliet Framework se basa en las interfaces RAPL de los procesadores Intel y en los sistemas operativos basados en Unix para acceder a las métricas de consumo energético en micro Joules de cada computadora y mediante comunicaciones de tipo Socket y múltiples protocolos transmiten las métricas recolectadas a un servidor encargado de la recolección de las métricas dentro de la red. Haciendo uso de Juliet Framework se implementa un caso de uso específico para la recolección de métricas de energía y tiempo de ejecución de los sistemas manejadores de bases de datos PostgreSQL y MongoDB. Con esta implementación se recolectan datos acerca del impacto energético y de rendimiento al aplicar un conjunto definido de optimizaciones sobre los sistemas de bases de datos.

INDICE GENERAL

[DEDICATORIA 3](#_Toc178521023)

[AGRADECIMIENTOS 4](#_Toc178521024)

[RESUMEN 5](#_Toc178521025)

[ÍNDICE DE FIGURAS 11](#_Toc178521026)

[Ilustraciones 11](#_Toc178521027)

[Códigos y Comandos 11](#_Toc178521028)

[Tablas 12](#_Toc178521029)

[INTRODUCCION 13](#_Toc178521030)

[Planteamiento del problema 13](#_Toc178521031)

[Objetivos 14](#_Toc178521032)

[Aportes 16](#_Toc178521033)

[CAPÍTULO I CONCEPTOS PRELIMINARES 18](#_Toc178521034)

[1. Conceptos Básicos de Energía 18](#_Toc178521035)

[1.1. Definición de Energía 18](#_Toc178521036)

[1.2. Definición de Potencia 18](#_Toc178521037)

[2. Sistemas Manejadores de Bases de Datos (SGBD): 19](#_Toc178521038)

[2.1. Definición 19](#_Toc178521039)

[2.2. Funciones principales de un SGBD: 19](#_Toc178521040)

[2.3. Tipos de SGBD: 19](#_Toc178521041)

[2.4. Optimizaciones de Bases de Datos: 20](#_Toc178521042)

[3. Consumo Energético de Sistemas Computacionales 23](#_Toc178521043)

[3.1. Definición 23](#_Toc178521044)

[3.2. Medición del Consumo Energético 23](#_Toc178521045)

[3.3. Software de Monitoreo de Consumo Energético 23](#_Toc178521046)

[3.4. Sistemas Distribuidos 26](#_Toc178521047)

[3.5. Consumo Energético de Sistemas Manejadores de Bases de Datos en Sistemas Distribuidos 27](#_Toc178521048)

[4. Interfaz RAPL de los Procesadores Intel en Sistemas Operativos Linux 27](#_Toc178521049)

[4.1. Definición de RAPL (Running Average Power Limit) 27](#_Toc178521050)

[4.2. En Qué se Basa RAPL 27](#_Toc178521051)

[CAPÍTULO II TRABAJOS RELACIONADOS 29](#_Toc178521052)

[1. Análisis del consumo energético en sistemas computacionales 29](#_Toc178521053)

[2. Análisis del consumo energético a nivel de software 30](#_Toc178521054)

[3. Análisis del consumo energético a nivel de software en sistemas distribuidos 32](#_Toc178521055)

[CAPÍTULO III METODOLOGÍA DE TRABAJO 36](#_Toc178521056)

[1. FASE DE INVESTIGACIÓN Y FAMILIARIZACIÓN 36](#_Toc178521057)

[1.1. Revisión de Literatura 36](#_Toc178521058)

[1.2. Análisis Crítico 37](#_Toc178521059)

[1.1. Definición del Problema 37](#_Toc178521060)

[1.2. Weekly de Desarrollo Ágil 37](#_Toc178521061)

[2. FASE DE DISEÑO 37](#_Toc178521062)

[2.1. Diseño Metodológico 37](#_Toc178521063)

[2.2. Selección de Herramientas 38](#_Toc178521064)

[2.3. Planificación del Trabajo 38](#_Toc178521065)

[3. FASE DE IMPLEMENTACIÓN 38](#_Toc178521066)

[4. FASE DE EXPERIMENTACIÓN 39](#_Toc178521067)

[4.1. Pruebas y Validación 39](#_Toc178521068)

[4.2. Análisis de Resultados 39](#_Toc178521069)

[CAPÍTULO IV JULIET: FRAMEWORK PARA EL MONITOREO DEL CONSUMO ENERGÉTICO DISEÑO Y ARQUITECTURA 40](#_Toc178521070)

[1. Motivacion 40](#_Toc178521071)

[2. Arquitectura del Framework 41](#_Toc178521072)

[2.1. Abstracciones de instancias 42](#_Toc178521073)

[2.2. Canales de comunicación 43](#_Toc178521074)

[2.3. Protocolos de comunicación 43](#_Toc178521075)

[2.4. Componentes 44](#_Toc178521076)

[2.5. Configuración centralizada en archivo único 45](#_Toc178521077)

[CAPÍTULO V JULIET: FRAMEWORK PARA EL MONITOREO DEL CONSUMO ENERGÉTICO IMPLEMENTACION 46](#_Toc178521078)

[1. CANALES Y PROTOCOLOS DE COMUNICACIÓN 47](#_Toc178521079)

[2. ARCHIVO DE CONFIGURACIÓN CENTRALIZADO 49](#_Toc178521080)

[3. COMMAND LINE INTERFACE (CLI) 49](#_Toc178521081)

[4. CONSOLE 50](#_Toc178521082)

[5. ORCHESTRATORS 51](#_Toc178521083)

[6. EXECUTOR 51](#_Toc178521084)

[7. OBSERVER 53](#_Toc178521085)

[8. COLLECTOR 55](#_Toc178521086)

[CAPÍTULO VI EXPERIMENTOS Y RESULTADOS 56](#_Toc178521087)

[1. ESCENARIO DEL EXPERIMENTO 56](#_Toc178521088)

[2. BENCHMARK TPC-H TRADUCCION A MONGODB 56](#_Toc178521089)

[2.1. Diferencias Fundamentales entre SQL y NoSQL 56](#_Toc178521090)

[2.2. Traducción Directa 57](#_Toc178521091)

[2.3. Proceso de Migración de Tablas a Colecciones 57](#_Toc178521092)

[2.4. Estrategias de Traducción 61](#_Toc178521093)

[3. CONFIGURACIONES EXPERIMENTALES 65](#_Toc178521094)

[3.1. Computadoras con Rol de Servidor 66](#_Toc178521095)

[3.2. Computadoras con Rol de Cliente 66](#_Toc178521096)

[3.3. Computadora con Rol de Analista y Colector 66](#_Toc178521097)

[3.4. Experimentos en PostgreSQL 67](#_Toc178521098)

[3.5. Experimentos en MongoDB 67](#_Toc178521099)

[4. ALGORITMO DE EJECUCIÓN 67](#_Toc178521100)

[CONCLUSIONES 69](#_Toc178521101)

[REFERENCIAS 70](#_Toc178521102)

[ANEXO A 71](#_Toc178521103)

[TABLAS DE RESULTADOS 75](#_Toc178521104)

[1. Lista de Abreviaturas en las Tablas de Resultados 75](#_Toc178521105)

[2. Tablas de Resultados PostgreSQL 76](#_Toc178521106)

[3. Tablas de Resultados MongoDB 87](#_Toc178521107)

# ÍNDICE DE FIGURAS

## Ilustraciones

[Ilustración 1 Vista General 41](#_Toc178519591)

[Ilustración 2 Configuración Posible de Juliet Framework 46](file:///C:\Users\Admin\Study\PFG\Book\Eros%20Cedeño%20-%20proyecto%20de%20grado.docx#_Toc178519592)

[Ilustración 3 Ejemplo o de buffer de comunicación 48](#_Toc178519593)

[Ilustración 4 52](#_Toc178519594)

[Ilustración 5 Diagrama de Capa de Red 53](file:///C:\Users\Admin\Study\PFG\Book\Eros%20Cedeño%20-%20proyecto%20de%20grado.docx#_Toc178519595)

[Ilustración 6 Arquitectura Monitor Juliet 54](#_Toc178519596)

[Ilustración 7 Algoritmo de experimentación 68](#_Toc178519597)

## Códigos y Comandos

[Comando 1 Leer el consumo de energía del dominio de la CPU 28](#_Toc178519912)

[Comando 2 Leer el consumo de energía actual en micro Joules 28](#_Toc178519913)

[Comando 3 Leer Nombre del dominio 28](#_Toc178519914)

[Comando 4 Leer consumo de energía del dominio de la CPU 28](#_Toc178519915)

[Comando 5 Tabla orders en PostgreSQL 58](#_Toc178519916)

[Comando 6 Tabla Orders in Mongo 59](#_Toc178519917)

[Comando 7 Equivalencia Relación: orders y customer 59](#_Toc178519918)

[Comando 8 PostgreSQL Consulta para Obtener Información de orders y customer 60](#_Toc178519919)

[Comando 9 MongoDB Consulta para Obtener Información de orders y customer 60](#_Toc178519920)

[Comando 10 Consulta SQL de ejemplo de traducción 63](#_Toc178519921)

[Comando 11 Consulta Mongo de ejemplo de traducción 64](#_Toc178519922)

## Tablas

[Tabla 1 Trabajos Relacionados 33](#_Toc178519855)

[Tabla 2 Equivalencia de conceptos PostgreSQL MongoDB 57](#_Toc178519856)

# INTRODUCCION

En la historia pocas cosas han transformado tanto a la sociedad como la computación. El comienzo de la era digital trajo consigo cambios radicales y acelerados en la concepción del mundo actual, transformándolo, moldeándolo y sosteniéndolo. Sin embargo, este crecimiento acelerado ha dejado de lado asuntos fundamentales como el impacto energético y ambiental que la tecnología causa en el planeta, un crecimiento desbocado e inconsciente no es sostenible a lo largo del tiempo. La tecnología se enfrenta a una dificultad natural: Los recursos son finitos, pero las necesidades son infinitas. También se enfrenta a la naturaleza del hombre quien cuanto más devora, más hambre tiene.

En este contexto, surge la necesidad de desarrollar software con conciencia, es imperante considerar el consumo energético del software y ajustar su diseño en busca de un equilibrio entre rendimiento y consumo. Una de las piezas más importantes dentro de los sistemas de computación a gran escala encontramos a las bases de datos. Las Bases de Datos son componentes cruciales de los sistemas informáticos por lo es se debe contar con herramientas que permitan a los administradores de sistemas y desarrolladores de aplicaciones comprender el consumo energético de sus bases de datos.

## Planteamiento del problema

El consumo energético desbordado de las bases de datos puede tener un impacto significativo en los costos operativos y la huella ambiental de las infraestructuras. Los sistemas distribuidos conciben los sistemas como un conjunto de piezas simples y atómicas que separan los procesos y gestiones en nodos independientes distribuidos en ambientes heterogéneos. Por lo que el análisis y monitoreo se encuentra con los siguientes desafíos:

* **Falta de visibilidad:** La complejidad y diversidad de los sistemas distribuidos dificulta la monitorización y comprensión del consumo energético de las bases de datos que estas contienen. Los administradores de sistemas carecen de herramientas para identificar las configuraciones y operaciones que consumen más energía.
* **Dificultad en la optimización:** Sin una comprensión clara del consumo energético, es difícil tomar decisiones informadas para optimizar el rendimiento energético de las bases de datos. Las configuraciones inadecuadas, las consultas ineficientes y los patrones de acceso no optimizados pueden contribuir significativamente al consumo excesivo de energía.
* **Ausencia de predicción:** La falta de herramientas de recolección de datos energéticos dificulta la planificación a largo plazo y la toma de decisiones estratégicas relacionadas con la infraestructura de las bases de datos. Es necesario recolectar adecuadamente estos datos para que en el futuro se pueda predecir el consumo energético bajo diferentes escenarios de carga para garantizar la eficiencia energética a medida que crecen las demandas del sistema.

Estos problemas tienen consecuencias negativas que impactan tanto a las organizaciones como al medio ambiente:

* **Costos operativos elevados:** El consumo energético excesivo se traduce en mayores costos para las organizaciones. Para empresas apoyadas en servicios en la nube como AWS se traduce a un mayor costo de cómputo y en consecuencia a un mayor gasto operativo.
* **Huella ambiental:** La generación de energía a partir de fuentes no renovables libera gases de efecto invernadero y contribuye al calentamiento global.
* **Rendimiento del sistema:** El consumo energético descontrolado puede afectar negativamente el rendimiento del sistema, aumentando la latencia y reduciendo la capacidad de respuesta.

## Objetivos

El problema del consumo energético excesivo de las bases de datos en sistemas distribuidos exige soluciones innovadoras que permitan a los administradores de sistemas y desarrolladores de aplicaciones:

* Monitorear el consumo energético a diferentes niveles de granularidad.
* Identificar patrones de consumo y detectar anomalías.
* Analizar los factores que contribuyen al consumo energético.
* Tomar decisiones informadas para optimizar el consumo energético de las bases de datos.

Este proyecto de grado tiene como objetivo desarrollar un framework integral para el análisis del consumo energético de bases de datos en sistemas distribuidos, denominado **Juliet**, y plantear un diseño de software escalable para el análisis del consumo energético de cualquier pieza de software en sistemas distribuidos. Adicionalmente, haciendo uso del **Framework Juliet** se desea generar una base de conocimiento para los desarrolladores, a fin identificar diversas estrategias de optimización de base de datos y estudiar su posible impacto en el consumo energético en sistemas distribuidos. Con los resultados obtenidos con el framework se busca hacer un análisis comparativo entre el desempeño energético de diferentes tipos de sistemas manejadores de base de datos y estudiar si un conjunto dado de optimizaciones causa algún efecto en el consumo total de hacer consultas sobre las mismas. Los objetivos específicos del proyecto se mencionan a continuación:

1. Desarrollar una propuesta de diseño de framework modular y escalable para la monitorización del consumo energético de piezas de software específicas en sistemas distribuidos. La propuesta de diseño debe contemplar la escalabilidad de los sistemas distribuidos, su heterogeneidad, y la interdependencia de los nodos dentro de la red.
2. Desarrollar un framework modular y escalable para la monitorización del consumo energético de bases de datos en sistemas distribuidos. El framework debe ser capaz de recopilar datos de consumo energético de la base de datos a diferentes niveles de granularidad, incluyendo nodos, procesos, consultas y operaciones sobre la base de datos.
3. **Documentar el framework Juliet en detalle, incluyendo su arquitectura, componentes, funcionalidades y uso.** La documentación debe ser clara, concisa y accesible para usuarios con diferentes niveles de experiencia técnica.
4. Liberar el framework Juliet como software de código abierto para que la comunidad de investigación y desarrollo pueda contribuir a su mejora y adopción.
5. **Crear una base de conocimiento sobre el impacto que puede o no generarse al aplicar optimizaciones sobre las bases de datos:** A fin de identificar las configuraciones de optimización de bases de datos más eficientes energéticamente, se llevará a cabo un estudio empleando el framework Juliet, el cual evaluará el consumo energético asociado a la ejecución de consultas.
6. Realizar un análisis comparativo del consumo energético entre sistemas manejadores de base de datos relacionales y no relacionales. Se deben realizar experimentos de desempeño y consumo energético de sistemas manejadores de base de datos relaciones y comparar su eficiencia energética con al menos un sistema manejador de base de datos no relacional.

Una vez alcanzados estos objetivos, Juliet se convertirá en un aliado estratégico para quienes buscan optimizar el consumo energético de sus bases de datos. Al ofrecer una herramienta para identificar las configuraciones más eficientes, Juliet contribuirá a la reducción de costos operativos, a la mitigación del cambio climático y a la mejora continua de los sistemas informáticos, promoviendo un futuro digital más sostenible.

## Aportes

Las contribuciones principales de este trabajo son:

* **Juliet**, framework modular y escalable para la monitorización del consumo energético de bases de datos en sistemas distribuidos, su objetivo principal es convertirse en una herramienta de análisis del consumo energético capaz de reportar datos de consumo energético en sistemas distribuidos tanto a nivel de cliente como de servidor, así los desarrolladores pueden determinar estrategias de optimización para disminuir el consumo energético en sus aplicaciones.
* **Juliet Monitor**, herramienta de monitoreo de consumo energético a nivel de proceso para sistemas operativos basados en linux y procesadores Intel con interfaz RAPL disponible.
* **Benchmark TPC-H para base de datos MongoDB**, Se realizará una traducción directa del ampliamente conocido benchmark **TPC-H** diseñado específicamente para evaluar el rendimiento de sistemas de gestión de bases de datos para realizar pruebas de estrés similares a las que se realizan con dicho benchmark sobre base de datos relacionales.
* Una publicacion:
  + Eros Cedeño, Ana Isabel Aguilera, Denisse Muñante, Jorge Correia, Leonel Guerrero, Carlos Sivira, Yudith Cardinale:

An Evaluation of the Impact of End-to-End Query Optimization Strategies on Energy Consumption. ENASE 2024: 657-665

Organización del documento

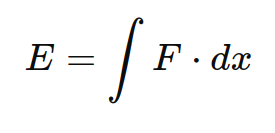
Este trabajo está estructurado de la siguiente manera: en el capítulo I se presentan conceptos preliminares fundamentales de las áreas de estudio a tratar, se abordan conceptos tales como Consumo energético, Software en sistemas Distribuidos, Optimizaciones en bases de datos y monitoreo de energía a través de hardware y software especializado. En el capítulo II se presentan trabajos relacionados en las áreas de estudio, análisis de consumo energético mediante dispositivos de hardware, herramientas de software enfocadas al monitoreo del consumo energético del sistema operativo y herramientas especializadas a nivel de proceso. En el capitulo III se plantea la metodología utilizada en la elaboración del trabajo, las implementaciones y análisis. En el capítulo IV se plantea la el diseño general y la conceptualización del framework propuesto, se describe el métodos de recolección de los datos de consumo energético a nivel de proceso, se definen las bases matemáticas sobre la que se sustenta el cálculo, se describen los componentes fundamentales, se explican los métodos de sincronización de los componentes del framework y finalmente los requerimientos de este. En el capítulo V y VI se presenta a Juliet como un Framework que implementa el diseño propuesto junto a dos casos de uso de monitoreo del consumo energético en dos tipos de sistemas manejadores de bases de datos configurados en ambientes distribuidos y concurrentes, adicionalmente se realiza un análisis comparativo de la eficiencia energética de ambos sistemas al ejecutar un benchmark específico y se reportan los resultados obtenidos durante los experimentos realizados.

# CONCEPTOS PRELIMINARES

Para establecer los cimientos que sustentan este trabajo se definirán conceptos básicos de la energía desde un punto de vista físico, luego se abordarán conceptos tales como consumo energético de sistemas computacionales, sistemas distribuidos, y consumo energético de sistemas manejadores de bases de datos en sistemas distribuidos, entre otros.

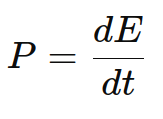
## Conceptos Básicos de Energía

### Definición de Energía

La energía desde el punto de vista físico se define como la capacidad de realizar trabajo. La energía (*E*) es medida en Joules (*J*) según el Sistema Internacional de Unidades (SI). 

Donde F es la fuerza aplicada y dx es el desplazamiento en la dirección de la fuerza. Sin embargo, este concepto se puede transportar a la electrónica si se entiende esta como la capacidad de un sistema electrónico para realizar trabajo, es decir, para mover cargas eléctricas a través de un circuito.

### Definición de Potencia

La potencia es una magnitud física que representa la rapidez con la cual se realiza trabajo o se transfiere energía. En otras palabras, es la tasa a la cual se consume o se produce energía en un sistema. La potencia se mide en Watts (W) en el Sistema Internacional de Unidades (SI), donde un watt equivale a un Joule por segundo (1 W = 1 J/s).

Matemáticamente, la potencia (P) se define como la derivada de la energía en el tiempo. Donde dE es la energía transferida en un intervalo de tiempo dt.

La ecuación anterior en este contexto puede entenderse de la siguiente manera: La potencia de un sistema electrónico en un tiempo *t* es igual a la energía consumida por el sistema en un diferencial de tiempo inmediatamente anterior.

## Sistemas Manejadores de Bases de Datos (SGBD):

### Definición

Un Sistema Manejador de Bases de Datos (SGBD), también conocido como DBMS (por sus siglas en inglés), es un software que permite crear, administrar y utilizar bases de datos de manera eficiente y segura. Es un elemento fundamental en la gestión de información, ya que proporciona una estructura organizada para almacenar, recuperar y manipular datos de forma confiable.

### Funciones principales de un SGBD:

* **Almacenamiento de datos:** El SGBD organiza los datos en una estructura definida, permitiendo un acceso eficiente y seguro.
* **Recuperación de datos:** El SGBD permite a los usuarios recuperar datos de la base de datos de manera rápida y precisa, utilizando diferentes criterios de búsqueda y filtrado.
* **Manipulación de datos:** El SGBD permite a los usuarios realizar operaciones sobre los datos, como agregar, eliminar, modificar y consultar información.
* **Control de acceso:** El SGBD permite controlar el acceso a la base de datos, definiendo permisos y roles para los usuarios, garantizando la seguridad e integridad de la información.
* **Integridad de datos:** El SGBD implementa mecanismos para mantener la integridad de los datos, asegurando que sean consistentes, precisos y confiables.
* **Recuperación de fallos:** El SGBD proporciona mecanismos para recuperar la base de datos en caso de fallos o errores del sistema.
* **Concurrencia:** El SGBD permite que múltiples usuarios accedan y manipulen la base de datos simultáneamente de manera organizada y controlada.
* **Seguridad:** El SGBD implementa medidas de seguridad para proteger la base de datos contra accesos no autorizados, intrusiones y pérdida de información.

### Tipos de SGBD:

* **SGBD relacionales:** Organizan los datos en tablas con relaciones entre ellas, utilizando el modelo de datos relacional. Son los SGBD más comunes y utilizados, como MySQL, PostgreSQL y Oracle Database.
* **SGBD no relacionales:** No se basan en el modelo de datos relacional y pueden almacenar y gestionar datos de manera más flexible. Algunos ejemplos incluyen MongoDB, Cassandra y CouchDB.
* **SGBD distribuidos:** Distribuyen los datos en múltiples nodos de una red, permitiendo una mayor escalabilidad y disponibilidad. Ejemplos incluyen Apache Cassandra, Amazon DynamoDB y Google Cloud Spanner.
* **SGBD empotrados:** Se integran en aplicaciones específicas y se optimizan para un rendimiento y funcionalidad específicos. Ejemplos incluyen SQLite y Berkeley DB.

### Optimizaciones de Bases de Datos:

La **optimización de bases de datos** es el proceso de ajustar la estructura, el diseño y la configuración de una base de datos para mejorar su rendimiento y eficiencia. Esto implica el uso de diversas técnicas y estrategias para reducir el tiempo de respuesta de las consultas, mejorar la eficiencia del almacenamiento de datos, y minimizar el uso de recursos del sistema, como la memoria y la CPU.

#### Indexación

Implica crear estructuras adicionales, llamadas índices, que permiten un acceso rápido a las filas de una tabla en función de los valores de una o más columnas. Un índice funciona de manera similar a un índice en un libro, permitiendo encontrar datos específicos sin necesidad de revisar todas las filas de la tabla.

En **PostgreSQL**, los índices son fundamentales para mejorar el rendimiento de las consultas, especialmente en tablas grandes. Algunos de los tipos de índices más utilizados incluyen:

* **B-tree:** El tipo de índice más común, utilizado para búsquedas rápidas de igualdad y rango.
* **Hash:** Útil para búsquedas de igualdad, aunque menos común debido a algunas limitaciones en versiones anteriores de PostgreSQL.
* **GIN (Generalized Inverted Index):** Adecuado para búsquedas en columnas que contienen muchos valores, como arrays o documentos JSONB.
* **GiST (Generalized Search Tree):** Usado para tipos de datos complejos como puntos geográficos o datos geométricos.

En **MongoDB**, un sistema de base de datos NoSQL, la indexación es igualmente crucial. MongoDB soporta varios tipos de índices:

* **Índices de campo único:** Índices simples en un solo campo.
* **Índices compuestos:** Índices en múltiples campos, que permiten consultas más complejas.
* **Índices multikey:** Utilizados para campos que contienen arrays, creando un índice separado para cada valor del array.
* **Índices de texto:** Optimizados para búsquedas de texto completo.
* **Índices geoespaciales:** Para consultas de datos geoespaciales.

MongoDB utiliza automáticamente índices para mejorar la eficiencia de las operaciones de búsqueda, sorteo y agrupamiento.

#### Compresión

La **compresión** es una técnica para reducir el tamaño de los datos almacenados mediante la codificación de los datos de manera que ocupen menos espacio. Esto se logra eliminando redundancias y aplicando algoritmos de compresión. La compresión puede mejorar el rendimiento de una base de datos al reducir la cantidad de datos que se deben leer o escribir desde el disco, lo que es especialmente beneficioso en sistemas donde el almacenamiento es un cuello de botella.

En **PostgreSQL**, la compresión puede ser aplicada a nivel de tablas y columnas usando técnicas como:

* **TOAST (The Oversized-Attribute Storage Technique):** Utilizada para manejar columnas de gran tamaño, PostgreSQL almacena datos grandes fuera de la tabla principal en almacenamiento separado y comprimido.
* **Compresión de columnas en índices BRIN (Block Range INdexes):** BRIN utiliza compresión para almacenar datos de índices de manera eficiente, lo que es útil en tablas muy grandes donde las consultas suelen acceder a rangos de datos.

Sin embargo tambien es posible aplicar compresión a una base de datos postgres utilizando extensiones del sistema manejador. Una de estas es **Pg Squeeze** la cual está diseñada para manejar la fragmentación de tablas y la bloat (expansión innecesaria) de manera automática.

Pg Squeeze aborda estos problemas mediante las siguientes características principales:

1. Reducción del Bloat (VACUUM) Automática:
   * Pg Squeeze realiza operaciones de reducción de bloat sin necesidad de bloquear la tabla completa, lo que permite que la base de datos siga funcionando normalmente mientras se optimiza.
   * A diferencia del comando estándar VACUUM FULL de PostgreSQL, que requiere bloquear la tabla y puede ser costoso en términos de tiempo de inactividad, Pg Squeeze permite una "compresión" en línea de las tablas afectadas.
2. Reorganización de Tablas:
   * La extensión crea una nueva tabla con los datos reorganizados y optimizados, transfiriendo los datos de la tabla original a esta nueva tabla mientras minimiza la interferencia con las operaciones en curso.
   * Una vez que la transferencia se completa, Pg Squeeze cambia automáticamente las referencias a la nueva tabla, eliminando la tabla original.
3. Automatización y Programación:
   * Pg Squeeze puede ser configurado para ejecutarse automáticamente según un cronograma, lo que permite a los administradores de bases de datos optimizar sus tablas regularmente sin intervención manual.
   * También permite la personalización de cuándo y cómo se deben manejar las diferentes tablas, basándose en el tamaño de la tabla, la cantidad de bloat, o el uso de la tabla.
4. Soporte para Particiones:
   * Pg Squeeze soporta tablas particionadas, lo que significa que puede reorganizar particiones individuales sin afectar a las demás.

En **MongoDB**, la compresión se maneja a nivel del motor de almacenamiento. Uno de los motores más utilizados, **WiredTiger**, ofrece compresión nativa. Algunas opciones de compresión incluyen:

* **Snappy:** Un algoritmo de compresión ligero y rápido, utilizado por defecto.
* **zlib:** Proporciona una mayor tasa de compresión a costa de un mayor uso de CPU.
* **zstd:** Equilibrio entre la velocidad de compresión y la eficiencia.

La compresión en MongoDB ayuda a reducir el tamaño de los datos almacenados en disco y a mejorar la eficiencia de las operaciones de E/S (entrada/salida).

## Consumo Energético de Sistemas Computacionales

### Definición

El consumo energético de los sistemas computacionales se refiere a la cantidad de energía eléctrica utilizada por dispositivos de computación, como servidores, computadoras personales, y dispositivos móviles, durante su operación.

### Medición del Consumo Energético

El consumo energético de un sistema computacional puede medirse utilizando varios métodos, incluyendo:

* **Vatímetros (Wattmeters):** Dispositivos que miden el consumo en vatios (W). Nótese que estos dispositivos reportan unidades de potencia y no de energía. Los dispositivos de monitoreo de consumo energético mediante hardware son en realidad dispositivos medidores de potencia más que de energía, esto debido a que es una medida informativa de la energía que se está desplazando dentro del sistema. Sin embargo estos dispositivos no realizan una medición precisa de la energía total consumida a lo largo de la medición.
* **Software de monitoreo:** Son aplicaciones de software que estiman el consumo energético basándose en el uso de la CPU, la GPU, y otros componentes. Estos sistemas de monitoreo pueden elaborar el monitoreo a diferentes niveles dependiendo del objetivo de la aplicación.

### Software de Monitoreo de Consumo Energético

#### Definición de Software de Monitoreo de Consumo Energético

El software de monitoreo de consumo energético como ya se mencionó anteriormente es un conjunto de herramientas diseñadas para medir, analizar y gestionar el uso de energía en sistemas computacionales y dispositivos electrónicos. Estas herramientas permiten a los administradores de sistemas, ingenieros y usuarios finales optimizar el consumo energético, reducir costos operativos y minimizar el impacto ambiental. El monitoreo detallado proporciona datos valiosos sobre cómo se utiliza la energía, facilitando la implementación de estrategias para mejorar la eficiencia energética.

#### Tipos de Software de Monitoreo de Consumo Energético

##### Monitoreo de Consumo Energético a Nivel de Sistema

Estas herramientas miden el consumo de energía de todo el sistema computacional, incluyendo CPU, GPU, memoria, discos duros y otros componentes. Se enfocan en obtener métricas generales y globales del sistema objetivo. Sin embargo su nivel de granularidad es bajo.

##### Monitoreo de Consumo Energético a Nivel de Componente

Estas herramientas se centran en el consumo de energía de componentes específicos dentro de un sistema, como la CPU, GPU, y dispositivos de almacenamiento. Las métricas de energía reportadas son por cada uno de los componentes electrónicos objetivos, estos permiten una medición precisa del consumo energético de componentes individuales, Gráficos en tiempo real y datos históricos por componente, soporte específico para diferentes tipos de hardware entre otros. Su objetivo es reportar datos de consumo energético por cada pieza de hardware de interés.

##### Monitoreo de Consumo Energético a nivel de Aplicación y Software

Estas herramientas se enfocan en medir y optimizar el consumo energético a nivel de aplicaciones de software específicas. Su objetivo es ayudar a los desarrolladores a crear aplicaciones más eficientes. Esta presenta un análisis a mayor granularidad y detalle por aplicativo. Este tipo de software puede a su vez aumentar la especificidad de los reportes a nivel de procesos y aportar métricas de consumo energético para aplicaciones multiprocesos y multihilos.Estas herramientas permiten un análisis detallado del consumo energético a nivel de código, la identificación de partes del código que consumen más energía. Algunas incluso proporcionan sugerencias y herramientas para optimización.

El framework propuesto **Juliet** pertenece a esta categoría de Software de monitoreo.

##### Componentes Principales que Consumen Energía en las Computadoras

El consumo energético en sistemas computacionales se distribuye entre varios componentes clave:

* **Unidad Central de Procesamiento (CPU):** La CPU, también conocida como el cerebro de la computadora, es responsable de ejecutar instrucciones y procesar datos. Su consumo de energía varía según la velocidad del reloj, la complejidad de las instrucciones y la carga de trabajo. Los factores que influyen en el consumo de energía de la CPU incluyen:
* **Arquitectura de la CPU:** Las arquitecturas modernas, como x86 y ARM, utilizan técnicas para optimizar el consumo de energía, como el escalado de frecuencia y voltaje dinámicos, la desactivación de núcleos inactivos y la memoria caché de múltiples niveles.
* **Tecnología de fabricación:** Los procesos de fabricación más nuevos, permiten transistores más pequeños y eficientes como los de 7 nm y 5 nm, lo que reduce el consumo de energía.
* **Carga de trabajo:** Las tareas intensivas en computación, consumen mucha más energía que las tareas simples
* **Unidad de Procesamiento Gráfico (GPU):** Similar a la CPU, la GPU consume una cantidad significativa de energía, especialmente durante tareas intensivas como el procesamiento gráfico y los cálculos paralelos. La GPU es un procesador especializado en el procesamiento de gráficos y cálculos paralelos. Su consumo de energía es considerable, especialmente en tareas intensivas en gráficos. Los factores que influyen en el consumo de energía de la GPU incluyen:
* **Arquitectura de la GPU:** Las arquitecturas modernas de GPU, como NVIDIA CUDA y AMD Radeon, utilizan técnicas para optimizar el consumo de energía, como el escalado de frecuencia y voltaje dinámicos, la desactivación de multiprocesadores inactivos y la memoria caché de múltiples niveles.
* **Tecnología de fabricación:** Al igual que con los CPUs, los procesos de fabricación más nuevos, permiten transistores más pequeños y eficientes en las GPU, lo que reduce el consumo de energía.
* **Carga de trabajo:** Como se mencionó anteriormente, cargas de trabajo mayores incrementan el consumo energético.
* **Memoria (RAM):** La memoria dinámica de acceso aleatorio (DRAM) también consume energía tanto durante la lectura/escritura como en estado de reposo debido a la necesidad de refrescar sus celdas de memoria. La memoria dinámica de acceso aleatorio (DRAM) almacena datos temporalmente para que la CPU y la GPU puedan acceder a ellos rápidamente. Su consumo de energía se debe principalmente al proceso de "refresco" constante, para mantener los datos en las celdas de memoria.

### Sistemas Distribuidos

#### Definición

Un sistema distribuido es una colección de computadoras independientes que a ojos de sus usuarios simulan un sistema único y coherente. Los sistemas distribuidos son utilizados para mejorar la eficiencia, la escalabilidad, y la fiabilidad de los procesos computacionales.

#### Arquitecturas de Sistemas Distribuidos

Las arquitecturas de los sistemas distribuidos pueden ser clasificadas en varias categorías:

* **Cliente-Servidor:** Los clientes solicitan servicios y los servidores los proporcionan. Una red formada de este tipo de sistema será el ambiente de estudio de este trabajo de grado en el cual se realizarán los experimentos.
* **P2P (Peer-to-Peer):** Todos los nodos tienen roles equivalentes y pueden actuar tanto como clientes como servidores.
* **Sistemas de Multinivel:** Combinación de cliente-servidor y P2P.

#### Ejemplos de sistemas distribuidos:

* **Internet:** Internet es un sistema distribuido a gran escala que conecta miles de millones de computadoras en todo el mundo.
* **La nube:** La nube es un sistema distribuido que proporciona servicios de computación, almacenamiento y red a través de Internet.
* **Redes sociales:** Las redes sociales son sistemas distribuidos que permiten a los usuarios interactuar entre sí y compartir información.
* **Sistemas de comercio electrónico:** Los sistemas de comercio electrónico son sistemas distribuidos que permiten a los clientes comprar productos y servicios en línea.

### Consumo Energético de Sistemas Manejadores de Bases de Datos en Sistemas Distribuidos

#### Definición y Contexto

Los sistemas manejadores de bases de datos (DBMS) son software que se utiliza para crear, gestionar y manipular bases de datos. En el contexto de este trabajo se abordan sistemas de base de datos relacionales y no relacionales con data centralizada dentro de un sistema distribuido, estos sistemas permiten la consulta de datos desde múltiples nodos distribuidos en una red.

## Interfaz RAPL de los Procesadores Intel en Sistemas Operativos Linux

### Definición de RAPL (Running Average Power Limit)

RAPL (Running Average Power Limit) es una tecnología de gestión de energía desarrollada por Intel para monitorear y controlar el consumo energético de sus procesadores. Esta tecnología permite a los sistemas operativos y a las aplicaciones acceder a información precisa sobre el consumo de energía de diferentes componentes del procesador, como la CPU, la memoria DRAM y la GPU integrada. RAPL proporciona mecanismos para establecer límites de energía y reaccionar dinámicamente a las variaciones en el consumo de energía, asegurando un equilibrio entre rendimiento y eficiencia energética.

### En Qué se Basa RAPL

RAPL se basa en una serie de registros de modelo específicos (MSRs - Model Specific Registers) que están presentes en los procesadores Intel modernos. Estos registros proporcionan métricas detalladas sobre el consumo de energía y permiten configurar límites de energía para diferentes dominios de hardware. Los principales dominios controlados por RAPL incluyen:

* **Package Domain:** Representa todo el consumo de energía del paquete del procesador.
* **Power Plane 0 (PP0) Domain:** Normalmente se refiere al consumo de energía de los núcleos de la CPU.
* **Power Plane 1 (PP1) Domain:** Generalmente asociado con la GPU integrada.
* **DRAM Domain:** Corresponde al consumo de energía de la memoria DRAM.

Cada uno de estos dominios puede ser monitoreado y controlado de manera independiente, permitiendo una gestión detallada del consumo energético en diferentes partes del procesador.

Comando 1 Leer el consumo de energía del dominio de la CPU

En sistemas operativos Linux, la interfaz RAPL se puede consultar a través del subsistema powercap y el controlador intel\_rapl.El subsistema powercap expone la información RAPL a través del sistema de archivos sysfs. Se puede acceder a esta información navegando en el directorio /sys/class/powercap/intel-rapl. Dentro de este directorio, se pueden encontrar varios archivos y subdirectorios que representan los diferentes dominios y métricas de energía.

$ cat /sys/class/powercap/intel-rapl/intel-rapl:0/energy\_uj

Comando 2 Leer el consumo de energía actual en micro Joules

$ cat /sys/class/powercap/intel-rapl/intel-rapl:0/name

Comando 3 Leer Nombre del dominio

$ cat /sys/class/powercap/intel-rapl/intel-rapl:0:0/energy\_uj

Comando 4 Leer consumo de energía del dominio de la CPU

Para leer el consumo de energía y otros parámetros, se pueden utilizar comandos como cat para visualizar el contenido de los archivos relevantes. Por ejemplo:

Estos comandos proporcionan valores en microjoules (µJ), que pueden ser convertidos a otras unidades de energía según sea necesario.

# TRABAJOS RELACIONADOS

En este capítulo se mencionan y describen los estudios, investigaciones y trabajos más recientes relacionados con el análisis del consumo energético en sistemas computacionales, se analizan los diversos enfoques y objetivos de cada trabajo. Luego se estudian trabajos relacionados al consumo energético en las bases de datos, el alcance de estas investigaciones conclusiones y limitaciones. Finalmente se abordan trabajos relacionados al monitoreo de energía en sistemas distribuidos y no distribuidos, para establecer los antecedentes a este proyecto.

## Análisis del consumo energético en sistemas computacionales

El análisis del consumo de energía ha cobrado gran interés en los últimos años, debido a la necesidad de reducir gastos energéticos que permitan la escalabilidad de productos y servicios. En general, es muy simple obtener las métricas de la energía consumida por una computadora en un rango de tiempo determinado, desde hace décadas existen herramientas de hardware especializadas para la recolección de métricas de energía como se ha mencionado anteriormente. Sin embargo el desafío verdadero es identificar el origen del consumo para generar predicciones, y visualizar adecuadamente los efectos que pueden causar las diferentes decisiones de diseño de software en el desempeño energético de un sistema.

Autores como X. Fan, W.-D. Weber, y L. A. Barroso abordan la gestión del consumo energético en centros de datos, específicamente en procesadores multi-core, y proponen un modelo detallado que permite estimar y analizar el consumo energético de los procesadores, basándose en una caracterización del hardware, esto incluye núcleos de CPU, unidades de control de memoria y subsistemas de entrada/salida, en función de esto utilizan herramientas de medición como medidores de potencia y software de monitoreo para capturar datos empíricos bajo diferentes condiciones operativas, y luego integran estos datos obtenidos en un modelo matemático que relaciona el consumo energético con variables operativas como la frecuencia de la CPU, el voltaje y la carga de trabajo.

Con esto se estima el consumo energético de sistemas computacionales y generan predicciones de consumo energético según los componentes de hardware involucrados, sin embargo los autores no toman en cuenta la optimización de energía de cada computador, únicamente se analiza el consumo como una variable estadística y analizan la correlación entre ella y otras variables vinculadas directamente al hardware como la frecuencia de CPU o cantidad de núcleos.

Por otros lado, otras investigaciones han puesto su enfoque en determinar estrategias de optimización de hardware para aminorar los consumos de energía, estos proponen la selección y configuración de hardware que maximice la eficiencia energética, como el uso de procesadores de bajo consumo y sistemas de almacenamiento eficientes. También abordan estrategias avanzadas para la gestión térmica, incluyendo el uso de enfriamiento adiabático y refrigeración líquida. Técnicas para la distribución eficiente de la carga de trabajo, como la migración de máquinas virtuales y la consolidación de servidores.

## Análisis del consumo energético a nivel de software

Dado que el enfoque de este trabajo es a nivel de software hay que establecer el estado del arte según este enfoque. Dado que el consumo de energía es una variable física, el software necesita un canal de comunicación con el hardware para obtener métricas de la carga de trabajo del computador y la cantidad de energía que se está consumiendo en un lapso de tiempo determinado, si se desea obtener las métricas directamente. Otra alternativa que se puede tomar en consideración sería que el software obtenga estas métricas de un modelo previamente entrenado para estimar el consumo de energía.

Ya se han realizado esfuerzos a nivel de hardware para monitorear, reportar y regular el consumo energético desde el software, este vínculo es propuesto en el artículo "RAPL: Memory Power Estimation and Capping", H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, y C. Le donde presentan la tecnología RAPL (Running Average Power Limit) de Intel, que permite justamente esto, estimar y limitar el consumo energético en procesadores modernos. RAPL utiliza Model Specific Registers (MSRs) para monitorear y controlar el consumo de energía en tiempo real, ajustando dinámicamente la frecuencia y el voltaje de la CPU para mantenerse dentro de límites predefinidos. Este sistema es esencial para evitar el sobrecalentamiento y reducir el desgaste de los componentes. La implementación de RAPL en sistemas operativos como Linux permite a los administradores del sistema utilizar herramientas de monitoreo como *powerstat* y *perf* para visualizar y controlar el consumo energético. Los resultados experimentales demuestran que RAPL puede reducir significativamente el consumo energético en aplicaciones intensivas en cálculo sin afectar negativamente el rendimiento, lo que la convierte en una herramienta valiosa para una amplia variedad de entornos, desde centros de datos hasta dispositivos de computación de alto rendimiento. La tecnología RAPL es una herramienta clave para el framework Juliet, como se verá más adelante.

Gracias a esta interfaz que proponen los procesadores Intel han surgido diferentes herramientas para monitorear el consumo energético, En el artículo "Linux Energy Management with PowerTOP", R. Ge, X. Feng, y K. Cameron presentan PowerTOP, una herramienta de software diseñada para diagnosticar y reducir el consumo de energía en sistemas Linux. PowerTOP utiliza contadores de hardware y software para monitorear el consumo de energía, identificando los componentes y procesos que consumen más energía. La herramienta ofrece recomendaciones prácticas para optimizar el uso de energía, como ajustes en la frecuencia de la CPU y la gestión del uso de la red. Los autores describen cómo PowerTOP mide el consumo de energía en tiempo real y sugiere cambios en la configuración del sistema para reducir el consumo energético. Los estudios de caso presentados demuestran que PowerTOP puede lograr reducciones significativas en el consumo energético con cambios relativamente simples en la configuración y el uso del sistema, haciendo de PowerTOP una herramienta esencial para los administradores de sistemas Linux que buscan optimizar el consumo energético.

PowerTop está destinado a ser una utilidad de monitoreo y uso rápido para determinar el consumo energético en tiempo real del host donde se ejecuta. Esta reporta métricas a intervalos regulares de todos los procesos que se están ejecutando en el computador. Pero el objetivo principal de PowerTop es ser una herramienta de acceso rápido y universal para monitorear un computador como su unidad absoluta. No admite análisis de consumo en procesos que coexisten en sistemas distribuidos en una red, lo que imposibilita analizar el consumo de energía a alta granularidad (a nivel de proceso) en sistemas distribuidos.

Paralelamente han surgido otras herramientas de recolección de métricas con mayor especificidad, un ejemplo de esto es JoularJx un software basado en agentes de java para monitorear la potencia en Watts y la energía en Joules consumida por una pieza de software a nivel de codigo, es decir permite conocer el consumo del software a nivel de procedimientos y bloques de codigo, esto haciendo uso de las funcionalidades nativas de la JVM que le permite obtener métricas de uso de CPU por función, hilo entre otros. Esta especificidad tan alta aporta a los programadores una herramienta valiosa para optimizar el código de Java que elaboren. Utiliza Intel RAPL para obtener lecturas de energía precisas en GNU/Linux, también cuenta con modelos de regresión basados ​​en investigaciones en dispositivos Raspberry Pi y un monitor de programa personalizado (usando un controlador RAPL) para lecturas de energía precisas en Windows. Sin embargo entre las limitaciones de JoularJx se tiene que el análisis del consumo energético está restringido a programas que se ejecutan dentro de la Máquina Virtual de Java (JVM) y esto es debido a que su diseño se sustenta con las herramientas de monitoreo nativas de la JVM, por lo que no permite analizar procesos genéricos del Sistema Operativo.

## Análisis del consumo energético a nivel de software en sistemas distribuidos

Para el análisis energético en sistemas distribuidos encontramos varias propuestas interesantes, un ejemplo de este tipo de software es Scaphandre.

Scaphandre es una herramienta de código abierto diseñada para el monitoreo y análisis del consumo energético en sistemas computacionales, especialmente en entornos distribuidos. Esta herramienta se utiliza principalmente para medir el consumo energético de procesos individuales, contenedores y máquinas virtuales, proporcionando información crucial para optimizar la eficiencia energética en centros de datos y otros entornos de computación de alto rendimiento. Scaphandre utiliza interfaces como RAPL (Running Average Power Limit) en procesadores Intel para obtener mediciones precisas del consumo energético. Además, puede integrarse con otras fuentes de datos para capturar información energética en diferentes componentes del sistema.

Una de las características más destacadas de Scaphandre es su capacidad para medir el consumo energético de contenedores (como Docker) y máquinas virtuales, lo que permite a los administradores de sistemas y desarrolladores entender mejor cómo las cargas de trabajo distribuidas impactan en el consumo energético. A su vez puede exportar los datos de consumo energético a diferentes sistemas de monitoreo y análisis, como Prometheus y Grafana, facilitando la visualización y el análisis de los datos en tiempo real. Esta integración permite a los usuarios crear dashboards personalizados para monitorear y optimizar el consumo energético de sus sistemas.

Aunque Scaphandre es una herramienta poderosa y versátil para el monitoreo del consumo energético en sistemas distribuidos, presenta algunas limitaciones que es importante tener en cuenta.

Tabla 1 Trabajos Relacionados

Scaphandre está optimizado para utilizar RAPL, que es específico de los procesadores Intel. Si bien esta es una interfaz común en muchos servidores, los sistemas que utilizan procesadores de otros fabricantes, como AMD, pueden no ser compatibles o pueden requerir configuraciones adicionales. Pero una de las limitaciones principales de esta herramienta es que la instrumentación y la recopilación de datos en tiempo real de una gran cantidad de métricas del sistema anfitrión pueden consumir recursos adicionales del sistema, lo que podría afectar el rendimiento de aplicaciones críticas en entornos de producción. Y algo muy importante es que la exportación y procesamiento de grandes volúmenes de datos de consumo energético usualmente introducen latencia en sistemas de monitoreo y análisis para entornos de alta frecuencia de muestreo.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Trabajo | Medida | Método | SMBD | Componente | Benchmark | Optimización / Estrategia/ Operación | Lado |
| (Yang et al., 2023) | Energy | LSSVM, sensors, predictive model | Oracle 11g | CPU, mem, hard drive | TPC-H | - | Client |
| (Karyakin and Salem, 2017) | Background power, active power | SIMM riser and linear power model | Shore-MT, MonetDB | DRAM, I/O | TPC-C, TPCH | - | Server |
| (Mahajan et al., 2019) | Performance, power, energy | Self-built power measurement tool | MySQL, MongoDB, Cassandra | CPU, DRAM | YCSB Twitter data | Index, ordered writes, agregations, row caching, compaction | Client |
| (Koc¸ak et al., 2018) | Average power | UPM EM100 Energy Meter | IBM DB2 | CPU, I/O | TPC-H | Index, compression, materialized query tables | Client |
| (Lella et al., 2023) | Energy, Mean Energy | PSUTIL,  python package | MySQL, PostgreSQL, MongoDB, & Couchbase | CPU, RAM | Netflix Userbase, SMS Spam collection | SELECT, INSERT, DELETE, UPDATE | Client |
| (Xu et al., 2015) | Energy | RLS, Wattsup power, and predictive model | PostgreSQL | CPU, I/O, communication | TPC-H, SDSS | - | Server |
| (Rodr´ıguez et al., 2013) | Energy, power, time | Wattsup power | PostgreSQL | Time | TPC-DS | - | Server |
| (Roukh et al., 2016) | Time, power (W) | Watts Up Pro | PostgreSQL | I/O, CPU | TPC-H, SSB | - | Server |
| (Guo et al., 2017) | Time, energy (J), power (W), energy efficiency | WT3000, power analyzer, and predictive model | Oracle 11g | CPU, disk memory | TPC-C, TPCH | Buffer Cache, Shared Pool | Server |
| (Liu et al., 2013) | Time, power | PM1000, power analyzer, and predictive model | PostgreSQL 9.1.8 | CPU, disk memory | TPC-H | - | Server |
| (Procaccianti et al., 2016) | Time, energy | Wattsup PRO meter | PLAIN SQL, Propel, and TinyQueries | Time | - | CREATE, READ, UPDATE, DELETE | Client |

# METODOLOGÍA DE TRABAJO

En este capítulo, se presenta la metodología Ad-Hoc utilizada para el desarrollo del framework Juliet, enfocado en el análisis del consumo energético en bases de datos distribuidas. La metodología se divide en cuatro fases: investigación y familiarización, diseño, implementación y experimentación. Cada una de estas fases es esencial para abordar de manera efectiva el problema del consumo energético en sistemas distribuidos y para alcanzar los objetivos establecidos.

La fase de investigación y familiarización permite entender el estado actual del conocimiento en el área, revisando literatura relevante y analizando trabajos previos. Posteriormente, en la fase de diseño, se establecen las bases teóricas y prácticas del framework, definiendo herramientas y metodologías de recolección de datos. La fase de implementación se centra en la ejecución de lo planificado, donde se realiza la recolección de datos sobre el consumo energético utilizando el Juliet Monitor y el benchmark TPC-H tanto en PostgreSQL como para MongoDB. Finalmente, la fase de experimentación permite evaluar la efectividad del framework y validar las hipótesis planteadas a través de pruebas controladas y análisis de resultados.

Este enfoque metodológico asegura un desarrollo estructurado y adaptable, garantizando que los hallazgos sean relevantes y útiles para la comunidad de desarrolladores interesados en optimizar el consumo energético de sus aplicaciones.

## FASE DE INVESTIGACIÓN Y FAMILIARIZACIÓN

### Revisión de Literatura

La fase inicial implica una revisión exhaustiva de la literatura existente sobre consumo energético en sistemas distribuidos, optimización de bases de datos y herramientas de monitoreo. Se consultan artículos académicos, tesis y estudios de caso que permitan identificar las mejores prácticas y los métodos utilizados por otros investigadores en este campo. La revisión se centra en:

* **Consumo energético en sistemas distribuidos:** Se analizan trabajos que exploran cómo diferentes arquitecturas y configuraciones afectan el consumo de energía.
* **Optimización de bases de datos:** Se estudian diversas estrategias y técnicas que han demostrado ser efectivas en la reducción del consumo energético.
* **Herramientas de monitoreo:** Se examinan soluciones existentes que permiten medir el consumo energético a nivel de software y hardware.

### Análisis Crítico

A partir de la información recopilada, se realiza un análisis crítico que permite identificar lagunas en la investigación y áreas donde el framework Juliet puede aportar valor. Este análisis se enfoca en:

* Evaluar la eficacia de las herramientas de monitoreo actuales.
* Identificar limitaciones en los estudios de consumo energético que puedan ser abordadas por el framework.
* Proponer un enfoque que integre las mejores prácticas de optimización en bases de datos.

### Definición del Problema

Con base en el análisis previo, se define el problema a investigar: la falta de un framework integral que no solo mida el consumo energético de bases de datos en sistemas distribuidos, sino que también brinde a los desarrolladores estrategias concretas para optimizar sus aplicaciones.

### Weekly de Desarrollo Ágil

Para garantizar una adaptación continua durante el desarrollo del proyecto, se adopta un enfoque ágil con ciclos semanales de trabajo. En cada reunión semanal se discuten los avances, se revisan los obstáculos encontrados y se ajustan las prioridades. Esta metodología permite una respuesta rápida a los cambios y una mejora continua del proceso de desarrollo.

## FASE DE DISEÑO

### Diseño Metodológico

El diseño metodológico se establece como un marco conceptual que guía el desarrollo del framework Juliet. Este diseño combina métodos cualitativos y cuantitativos para asegurar un análisis integral del consumo energético. Se definen los siguientes aspectos:

### Selección de Herramientas

Para la recolección de datos y la implementación del framework, se seleccionan herramientas que faciliten la monitorización del consumo energético:

* **Juliet Monitor:** Esta herramienta se diseñará para realizar un seguimiento del consumo energético a nivel de proceso en sistemas operativos Linux, aprovechando la interfaz RAPL disponible en procesadores Intel.
* **Benchmark TPC-H para PostgresSQL:** Se utilizará el benchmark TPC-H para establecer una muestra del desempeño de las bases de datos relacionales.
* **Benchmark TPC-H para MongoDB:** Se desarrollará una adaptación del benchmark TPC-H, permitiendo realizar pruebas de estrés y comparar el rendimiento energético de diferentes sistemas manejadores de bases de datos.

### Planificación del Trabajo

Se elabora un cronograma detallado que incluye:

* Fases de diseño y desarrollo del framework.
* Tiempos asignados para la implementación de las herramientas de monitoreo y el benchmark.
* Plazos para la recolección y análisis de datos.

Este cronograma se ajusta de manera dinámica en función de los resultados obtenidos en cada iteración semanal.

## FASE DE IMPLEMENTACIÓN

La fase de implementación del framework Juliet es un proceso fundamental en el que se lleva a cabo la materialización del diseño teórico planteado en la fase anterior. Esta fase está orientada a experimentar y analizar el consumo energético en dos sistemas manejadores de bases de datos diferentes: PostgreSQL y MongoDB. La implementación busca no solo establecer una infraestructura técnica robusta, sino también proporcionar un entorno de experimentación que facilite un análisis comparativo entre el desempeño energético de un sistema de bases de datos relacional y uno no relacional.

## FASE DE EXPERIMENTACIÓN

### Pruebas y Validación

En esta fase, se realizan experimentos controlados para evaluar el rendimiento del framework Juliet en diferentes escenarios de uso. Se aplican:

* Pruebas de carga para evaluar el consumo energético bajo condiciones de estrés.
* Comparativas entre los resultados obtenidos con y sin optimizaciones aplicadas.

### Análisis de Resultados

Los resultados de las pruebas se analizan para determinar la efectividad del framework en la monitorización y establecer un conocimiento sobre el impacto de un conjunto de configuraciones de bases de datos sobre el consumo energético. Se elaborarán gráficos y tablas que resuman los hallazgos y se presentarán de manera clara para facilitar la interpretación.

# JULIET: FRAMEWORK PARA EL MONITOREO DEL CONSUMO ENERGÉTICO DISEÑO Y ARQUITECTURA

En el capítulo anterior se describieron diversas investigaciones y herramientas de software orientadas a analizar el consumo energético de sistemas computacionales según diferentes enfoques. Cada una de estas herramientas presenta sus propias bondades y limitaciones. Algunas herramientas permiten el análisis de datos en tiempo real, otras posibilitan el monitoreo a altas frecuencias, y algunas están diseñadas para funcionar en sistemas distribuidos. Algunas destacan por su precisión, mientras que otras sobresalen por la robustez en la captura de gran cantidad de datos. Sin embargo, existen casos donde estas herramientas no cumplen con las necesidades específicas de captura de datos de energía, lo que limita su aplicabilidad en ciertos contextos de investigación y análisis.

## Motivacion

Juliet nace en el contexto de una investigación [] llevada a cabo por profesores, investigadores y estudiantes de la Universidad Simón Bolívar de Venezuela en colaboración con otras universidades e instituciones cuyo objetivo principal es realizar estudios y análisis del consumo energético tanto en el lado del cliente como del servidor del sistema manejador de base de datos PostgreSQL al aplicarles un conjunto de optimizaciones.

En la investigación antes mencionada, para recolectar las métricas del lado del cliente se recurre a la herramienta JoularJx la cual ya se definió con anterioridad, sin embargo para la recolección de datos del lado del servidor la investigación enfrentó un problema cúspide. Se quería encontrar la manera de recolectar los datos del consumo energético de un sistema manejador de base de datos en un intervalo de tiempo específico y sumamente corto, sin embargo JoularJx no es aplicable en este caso dado que no permite monitorear procesos fuera de la JVM, por lo tanto se recurre a software de monitoreo a nivel de procesos, específicamente a Scaphandre. Sin embargo Scaphandre a pesar de las grandes bondades con las que cuenta, como se mencionó anteriormente posee limitaciones que impiden la recolección adecuada en este escenario: durante la recopilación de datos, el objetivo principal de Scaphandre es reportar datos en tiempo real de una gran cantidad de métricas del sistema anfitrión, pero en este caso la mayoría de las métricas recolectadas no eran relevantes en la investigación, lo que afectaba el rendimiento del experimento. Y finalmente, el inconveniente principal fue que durante la experimentación la recolección de datos no era fiable. Debido a que cada iteración del experimento ocurre en intervalos de tiempos muy cortos (menos de un segundo), la exportación, transmisión, recolección y procesamiento de estos volúmenes de datos introducían alta latencia para tomar cada muestra dificultando la captación adecuada de los datos.

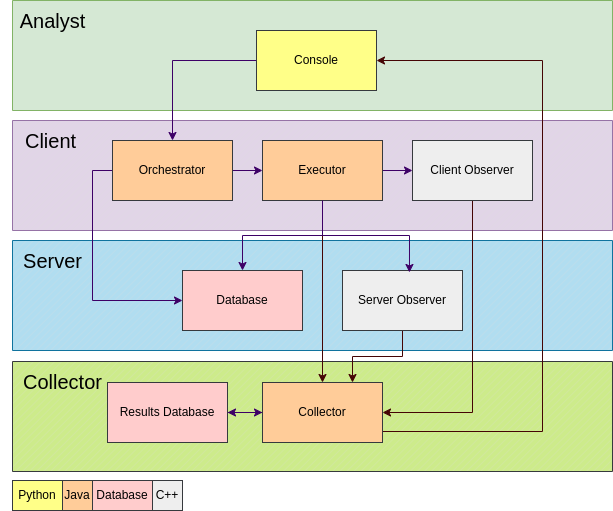
En este contexto se desarrolla un diseño de framework capaz de convivir en sistemas distribuidos, escalable y robusto. El objetivo principal de este diseño de framework es permitir el monitoreo a altas frecuencias del total de energía consumida por un conjunto de procesos distribuidos en diferentes host que comparten una red común.

## Arquitectura del Framework

La arquitectura de Juliet puede modelarse como una red de componentes distribuidos en diferentes instancias. Una instancia es una abstracción de una computadora destinada a cumplir una función específica dentro de la red.

Ilustración 1 Vista General

La figura a continuación muestra una estructura general de la arquitectura del framework Juliet



MEJORAR ESTA FIGURA

En esta figura se observan diferentes componentes (recuadros pequeños) conectados entre sí por flechas orientadas, la dirección de las flechas indica la dirección del flujo de los datos, y los cuadros de mayor tamaño delimitan el alcance y estructura de una instancia.

A continuación, se desglosa cada una de las partes de la arquitectura.

### Abstracciones de instancias

Una instancia es una abstracción para referirse a cualquier computadora dentro de la red que tenga un rol específico. En la figura se observa que la arquitectura de Juliet contempla 4 tipos de instancias diferentes, estas son Analyst (Analista), Client (Cliente), Server (Servidor) y Collector (Colector).

Las instancias son abstracciones, por lo que bien es posible superponer instancias en una misma computadora con las configuraciones adecuadas, o incluso configurar la ejecución de todas las instancias en la misma computadora. Sin embargo, crear el concepto de instancia da la capacidad de construir redes amplias con multitud de nodos, lograr clasificarlos y adicionalmente permite reconocer la estructura y capacidad disponible de cada nodo. Esto permite generar múltiples nodos por tipo de instancia y así hacer crecer la red de forma segura y adecuada.

* **Analyst:** esta instancia está destinada a ser el centro de operaciones del framework, desde aquí los investigadores pueden identificar las opciones disponibles en los nodos clientes de la red e iniciar cualquier acción que esté disponible.
* **Client:** es la instancia encargada de generar las configuraciones necesarias tanto sobre él mismo como sobre su servidor vinculado. Realiza las acciones recibidas como orden directa del analista (Analyst) y ejecuta la acción solicitada sobre el servidor cuando la configuración se haya completado. De igual forma es el encargado de dar las órdenes de inicio, final, reporte y cierre de conexión a todos los elementos conectados a ella que lo requieran, y reportar las métricas obtenidas durante la ejecución.
* **Server:** ejecuta las acciones solicitadas por el cliente. Adicionalmente esta instancia tiene la capacidad de analizar el consumo energético de los procesos en estudio y reportar las métricas colectadas en el servidor.
* **Collector:** esta instancia es un banco de datos, ella recibe la información de todas las métricas de todos los nodos de la red y las almacena en una base de datos para su posterior consulta por parte del analista.

### Canales de comunicación

Juliet está diseñado como un conjunto de componentes independientes que se comunican entre sí a través de diferentes canales. Durante el diseño se establece como regla general que cada canal de comunicación que se introduce entre componentes debe cumplir con las siguientes propiedades fundamentales:

* Cada canal de comunicación debe poder atravesar la red utilizando protocolos y estándares independientes del sistema donde se ejecutan.
* Para sincronización entre los componentes se requiere canales de comunicación basados en conexión con alta fiabilidad en la transmisión y baja latencia. Este tipo canal se nombraran **Canales de Tipo I** en éste y capítulos siguientes.
* Para comunicación entre componentes que no requieren baja latencia se prefiere simplicidad y fiabilidad de implementación y uso. Se llamarán **Canales de Tipo II**.

Estos principios se establecen para lograr componentes distribuibles en una red, para que puedan realizar ejecuciones independientes con canales de comunicación de alta velocidad y con alta fiabilidad en la transmisión cuando sea necesario (Canales de Tipo I) y adicionalmente mantener la simpleza en los canales cuya velocidad de comunicación no responde a una necesidad del análisis (Canales Tipo II).

### Protocolos de comunicación

Por definición un protocolo de comunicación es un conjunto de normas, pautas o instrucciones que sirven para guiar las acciones durante el intercambio de información, en otras palabras es el conjunto de reglas que permiten a los miembros comunicarse efectivamente.

En el caso de las comunicaciones a través de Canales de Tipo I se establecieron los siguientes protocolos para asegurar la sincronización entre los componentes comunicantes:

* **Espera Activa**: Cada mensaje enviado debe esperar una confirmación de recepción antes de continuar con su ejecución.
* **Confirmación de Recibo:** Cada mensaje recibido debe ser reportado como recibido al emisor.
* **Recibo Efectivo:** Cuando un mensaje recibido inicia una acción en el proceso receptor, este deberá reportar como recibido el mensaje únicamente cuando la acción se inicie adecuadamente, exceptuando la acción de clausura.
* **Clausura:** En caso de cierre del canal es necesario que ambos extremos cierren adecuadamente la conexión. Un proceso no cierra su conexión si no ha recibido la confirmación de recepción del mensaje de cierre del otro.

Sobre los canales de Tipo II se establece como requisito único la simplicidad de implementación y facilidad de lectura.

### Componentes

Juliet Framework pone a disposición un conjunto de componentes de software que son piezas independientes con función específica que interactúan entre sí por medio de los canales y protocolos de comunicación para ejecutar las acciones requeridas. A continuación se definirán las características y funciones de cada componente.

* **Console**: esta es la interfaz única de comunicación entre el usuario y el framework. La consola descubre los diferentes orquestadores que están escuchando en la red y se conecta a ellos mediante canales de tipo I. Al conectarse recibe de cada orquestador el conjunto de acciones y procedimientos que estos pueden ejecutar permitiéndole al usuario iniciar cualquiera de ellas. De igual forma desde la consola es posible obtener las métricas que almacena el colector y aplicar los tests estadísticos disponibles cuando el usuario así lo quiera. Esta consola reduce la complejidad de interacción con el framework y funciona como un nodo raíz con un rango superior dentro de la red a los cuales el resto de nodos están subordinados.
* **Orchestrator**: este componente es el encargado de iniciar un servicio de escucha de conexiones entrantes para el nodo analista y al establecer una conexión debe indicarle al analista el conjunto de acciones disponibles. Luego maneja las solicitudes de acción que el analista solicite, iniciando las configuraciones necesarias sobre el cliente y el servidor e invocando finalmente el inicio del ejecutor.
* **Executor**: el ejecutor es la pieza fundamental de la ejecución de de cada acción, a diferencia del orquestador ella no configura ninguna instancia, únicamente se encarga de ejecutar la acción solicitada iniciando y deteniendo oportunamente los monitores de energía tanto del cliente como del servidor.
* **Observer**: es el monitor de energía. Recolecta las métricas de un proceso dado, cuando recibe la orden del ejecutor de reportar las métricas obtenidas las transmite al colector. La arquitectura de Juliet Framework cuenta con un observador por instancia cliente y uno por instancia servidor como se observa en la figura #.
* **Collector**: el colector es el componente encargado de recibir las métricas colectadas y almacenarlas para su posterior consulta por parte de la instancia analista.

### Configuración centralizada en archivo único

Debido a la cantidad de componentes, la independencia de estos, la arquitectura distribuida, aunado a otros factores, la complejidad de configuración del framework crece. Por lo que se plantea en el diseño generar un archivo único de configuración del cual derivarán las configuraciones de todos los componentes del framework. Este archivo único se distribuye en todos los nodos de la red de tal manera que cada uno de estos pueda tomar las configuraciones que le correspondan y conocer las configuraciones del resto de nodos en caso de ser necesario.

# JULIET: FRAMEWORK PARA EL MONITOREO DEL CONSUMO ENERGÉTICO IMPLEMENTACION

**VISTA GENERAL**

En el capítulo anterior se planteó el diseño y arquitectura teórica de **Juliet Framework**. En este capítulo se describe y explica una implementación de este diseño teórico con el objetivo de experimentar y analizar el consumo energético en dos sistemas manejadores de base de datos diferentes, por un lado PostgreSQL y por el otro MongoDB. Esto para poder realizar un análisis comparativo entre el desempeño energético de un sistema de base de datos relacional como es Postgres y un sistema de base de datos no relacional como es un Mongo.

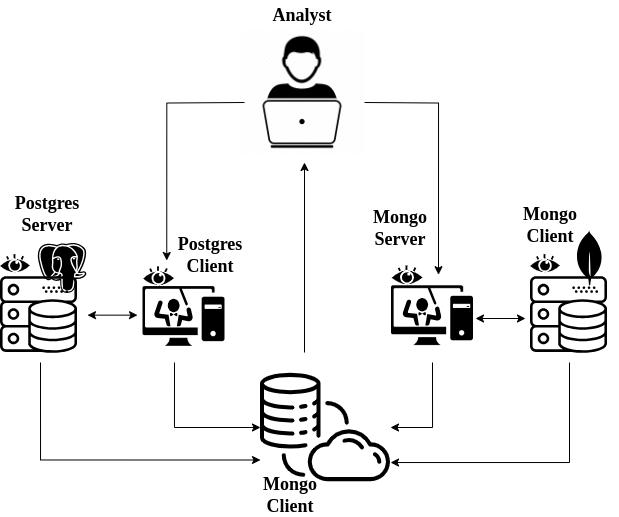
Para abordar estos retos y permitir llevar la experimentación a niveles mayores de complejidad, surge esta implementacion de Juliet Framework, ella busca ser un framework integral para el análisis del consumo energético de bases de datos en sistemas distribuidos con frecuencias altas de muestreo. Este framework no solo abordará los problemas de recolección de datos en intervalos cortos, sino que también permitirá una integración más fluida con diversos sistemas y herramientas de monitoreo, facilitando así una investigación más robusta y precisa en el ámbito del consumo energético.

Ilustración 2 Configuración Posible de Juliet Framework

La *Ilustración* 2 Configuración Posible de Juliet Frameworkmuestra un ejemplo de una posible configuración del framework en un sistema distribuido de 2 pares cliente servidor ejecutando pruebas con Postgres y Mongo, y todas reportando a un colector único y común que puede estar alojado en un servicio en la nube sin ningún inconveniente.

## CANALES Y PROTOCOLOS DE COMUNICACIÓN

En el diseño propuesto se enunciaron dos tipos de conexiones. Cada una de estas con diferentes características y principios. Para la implementación se seleccionó a los Sockets como Canal de Tipo I y a las comunicaciones por medio del protocolo HTTP como Canal de Tipo II.

* **Canales de Tipo I:** Para los canales de tipo I se optó por utilizar comunicación mediante Sockets utilizando el protocolo TCP, dado que permite un **flujo de datos bidireccional** **continuo** en tiempo real. Al ser orientados a conexión se **reduce el overhead** dado que se evita la necesidad de establecer una nueva conexión para cada intercambio de datos, lo que reduce la latencia y el consumo de recursos. En cuanto a escalabilidad se permiten
  + **Manejo de múltiples conexiones:** Los servidores pueden manejar múltiples conexiones de socket simultáneamente, lo que permite escalar las aplicaciones para atender a un gran número de usuarios.
  + **Arquitecturas distribuidas:** Los sockets son compatibles con arquitecturas distribuidas, donde múltiples servidores pueden comunicarse entre sí para ofrecer un servicio más robusto y escalable.
* **Canales de Tipo II:** Para los canales de Tipo II se seleccionó el protocolo HTTP debido a su **universalidad,** es el protocolo más utilizado en internet su diseño es relativamente **sencillo**, lo que facilita su implementación y comprensión. Para las comunicaciones sobre HTTP se eligió utilizar el estándar **REST**, por lo que en este canal los datos a intercambiar son de tipo JSON (Javascript Object Notation).

En resumen, el primer tipo (Canales de Tipo I) es para conexiones que requieran de alta velocidad y de sincronización por esto se utiliza comunicación a través de Sockets de tipo Socket\_Stream los cuales proporcionan una conexión bidireccional y confiable entre los dos puntos a comunicar, haciendo uso del protocolo TCP para garantizar la entrega ordenada y confiable de los datos. Y el segundo tipo de Canal se utilizará para aquellas comunicaciones cuya latencia no afecten en las métricas del experimento, mediante el protocolo HTTP se alcanzan estos objetivos además de no ser orientado a conexión lo que simplifica su implementación. Aunque su latencia es más alta que las de Tipo I su uso es muy simple y accesible.

En el intercambio de datos entre los extremos de una conexión de tipo Socket TCP se establecen dos códigos numéricos como estándares dentro de la comunicación:

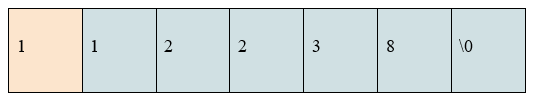
**0. Código de Confirmación de Recibo:** Este código es el estándar de confirmación ACKs que se maneja en la implementación. Cuando un mensaje es enviado a través del Socket se espera recibir el código de recepción ‘0’ del otro extremo

**9. Código de Clausura de Comunicación:**  Cuando una comunicación ha cumplido con su labor es necesario realizar la clausura del canal para la correcta finalización de los procesos y desocupación correcta del puerto TCP en uso. Para esto el extremo del Socket Cliente debe procurar enviar al Socket Servidor el código de clausura ‘9’ y esperar la confirmación de recepción antes de cerrar la conexión. El extremo servidor al recibir el código de clausura envía el mensaje de confirmación y procede a cerrar la conexión

El rango de códigos del **1-8** son libres entre las partes del túnel socket para acordar las acciones o procedimientos que requieran.

Ilustración 3 Ejemplo o de buffer de comunicación

Cada Stream de Datos cuenta con un Buffer de 1024 bytes donde el primer Byte indica el código de la acción a realizar seguido del argumento. Por ejemplo, el buffer de Stream de datos dentro de una comunicación con Socket luce de la siguiente manera:



En esta figura cada celda representa un byte de información. En este caso el código de la acción a realizar es el número 1. Como es un código de libre acuerdo la acción específica depende del contexto y acuerdo entre los extremos del Socket. Luego, los Bytes siguientes corresponden al stream de datos o argumentos a enviar de un extremo al otro. Un caso de uso posible del buffer que se observa en la figura puede ser: *“Inicia el monitoreo energético, del proceso con pid 12238”* suponiendo que el código “1” indique la acción de inicio de monitoreo.

## ARCHIVO DE CONFIGURACIÓN CENTRALIZADO

La configuración centralizada en un archivo único se definió como se puede observar en el anexo A. Es un archivo con formato JSON en el cual se especifican las configuraciones individuales de cada uno de los componentes del framework. Primero se tiene el console command en el cual se especifica qué emulador de consola de comandos se utilizará para ejecutar los diferentes componentes, luego está la configuración de los orquestadores de la que se tiene la identificación de cada uno de los orquestadores que están disponibles en la red, su dirección IP dentro de la red y el puerto de conexión.

Por otro lado se tiene la configuración de las bases de datos que se encuentran ejecutándose en cada instancia de tipo servidor. Cada una de estas identificadas con una etiqueta única en la que se específica primero la dirección del servidor de la base de datos el puerto de conexión el nombre de la base de datos sobre la cual se ejecutarán las pruebas nombre de usuario y contraseña de la base de datos en caso de ser requerido entre otras configuraciones necesarias por ejemplo para para ejecutar tanto el reinicio del servicio de la base de datos como configuraciones relacionados al benchmark y las optimizaciones disponibles.

Adicionalmente se tienen las configuraciones de los observadores que se ejecutan en las diferentes instancias, las de tipo cliente y las de tipo servidor

Finalmente se tienen las configuraciones del colector en la cual se especifica el puerto en el cual se expone el servicio REST del colector su dirección IP las configuraciones relacionadas a la base de datos del colector y las especificaciones de los endpoints de de recolección y consulta de las métricas de energía.

## COMMAND LINE INTERFACE (CLI)

Como se vio anteriormente en las abstracciones de las instancias, cada instancia cuenta con un conjunto de componentes específicos configurados para interactuar, colectar datos y reportarlos dependiendo del caso, sin embargo la configuración de estas instancias puede ser engorrosa lo que podría dificultar el uso del framework dado que exige un conocimiento profundo de la estructura interna de la red así como la adecuación completa del entorno en cada instancia.

Para corregir este problema se propone una Interfaz de linea de Comandos, Command Line Interface o CLI por sus siglas en inglés, programado en python que ofrece un repertorio de comandos para asistir en la configuración de un computador como una instancia específica y dar inicio a los componentes necesarios en cada caso.

Para ejecutar el CLI es necesario instalar las dependencias especificadas en el archivo de requerements.py y ejecutar el script juliet\_cli.py

$ python3 juliet\_cli.py <configuration file> <instance> [options]

Donde:

* Configuration file: es el path al archivo de configuración
* Instance: el tipo de instancia (Analyst, Collector, Client, Server)
* Options: Postgres o Mongo en caso de ser instancia de tipo servidor o cliente

## CONSOLE

A través de la CLI de Juliet es posible acceder al modo analista utilizando el comando:

**$ python3 juliet\_cli.py <configuration file> analyst**

Al ejecutar este comando se accede a la consola de comandos. Este es un programa escrito en python, al iniciar obtiene las configuraciones del archivo de configuración específicamente la configuración de los orquestadores. Por cada uno de los orquestadores allí presentes la consola intentará realizar una conexión con ellos. Luego de conectar a los orquestadores solicita las opciones disponibles a cada orquestador y se las presenta al usuario en un menú de texto para que este seleccione un orquestador objetivo y la acción que desea realizar sobre esta. Una vez que el usuario ha seleccionado la acción que desea ejecutar muestra mensajes de estatus de ejecución hasta que finalmente el orquestador indica la finalización de la acción.

## ORCHESTRATORS

El orquestador es un componente complejo dentro de la instancia del cliente. Se maneja un orquestador por tipo de base de datos a analizar. Este componente inicialmente expone un *socket server* que espera conexiones por parte del analista y ejecuta la configuración que fue seleccionada desde el analista por el usuario. Para el caso de interés de este trabajo los orquestadores exponen las distintas optimizaciones disponibles que son aplicables sobre el sistema manejador de base de datos en cada caso y al ser seleccionados este configura el benchmark TPC-H sobre la base de datos del servidor especificado con la optimización seleccionada por el usuario.

El TPC-H es un benchmark estándar diseñado por el Transaction Processing Performance Council (TPC) para medir el rendimiento de los sistemas de gestión de bases de datos (DBMS) en términos de consultas de procesamiento de transacciones y generación de informes. El principal objetivo del TPC-H es proporcionar una métrica de rendimiento objetiva y reproducible que permita comparar diferentes sistemas de bases de datos bajo condiciones de carga realista. Es ampliamente utilizado para analizar el desempeño de sistemas gestores de base de datos relacionales.

En la implementación del framework se utilizó la librería tpch-pgsql para generar y cargar el benchmark en Postgres y adicionalmente generar el repertorio de 22 queries o consultas que se utilizarán sobre este conjunto de datos.

Sin embargo este benchmark fue originalmente concebido para funcionar en sistemas gestores de base de datos relacionales, por lo que por fines investigativos se realizó un módulo de python específicamente para la traducción de este benchmark a Mongo y de igual manera se tradujo el repertorio de 22 queries para este sistema gestor de base de datos para ser ejecutados desde el driver de Java y para poder realizar un análisis comparativo del consumo energético entre Postgres y Mongo sobre el mismo conjunto de datos.

## EXECUTOR

El ejecutor es el componente con mayor interacción en el framework. Se encuentra en la librería principal desarrollada por el framework, está programado en Java y en total maneja dos conexiones de tipo socket TCP, una a cada observador (cliente y servidor) y una conexión de tipo HTTP hacia el collector.

El ejecutor maneja múltiples hilos en el proceso principal. Un hilo por conexion con los observadores tanto del servidor como del cliente, en cada uno de estos hilos se comunica con ellos e inicializa el monitoreo de los procesos específicos. Luego de haber iniciado los observadores ejecuta la query en turno para finalmente detener la ejecución de los observadores y reportar las métricas al colector.

A continuación se muestra una imagen ilustrativa de la interacción entre los componentes del framework cuya implementación ya fueron definidos.

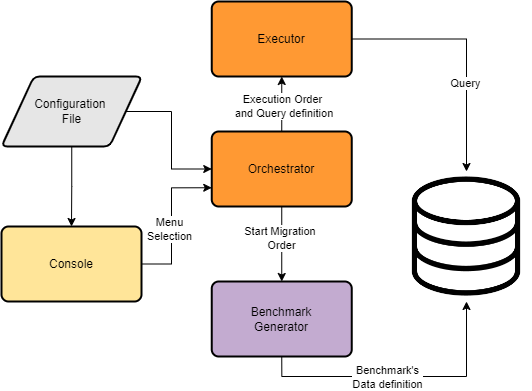


Ilustración 4

Interacción entre los componentes del framework

Las configuraciones nacen del archivo de configuración único. En este archivo de configuración se especifican las direcciones y puertos de cada una de las partes del framework. Esta información es utilizada por la consola para descubrir a los orquestadores dentro de la red y así poder interactuar con ellos. Cada orquestador haciendo uso del archivo de configuración así como de la selección del usuario, establece una configuración sobre la base de datos mediante el generador del benchmark. Luego de aplicada la configuración deseada sobre la base de datos, el orquestador procede con la ejecución secuencial de cada una de las queries del benchmark por la cantidad de iteraciones especificadas en el archivo de configuración.

## OBSERVER

Juliet Framework posee una pieza fundamental para la recolección de datos, este es su software monitor de energía llamado Juliet Observer. Está pieza de software está basada en la propuesta desarrollada por el autor para el trabajo de investigación "". Está construido en C++ y funciona en sistemas operativos basados en Linux con disponibilidad de las interfaces de monitoreo RAPL de los procesadores INTEL. Su función principal es acceder a las métricas de energia del sistema para dar una estimación del total de energía consumida por un proceso dado en el intervalo de tiempo de estudio.

Juliet Monitor está compuesto por dos capas independientemente, interconectadas y en constante comunicación.

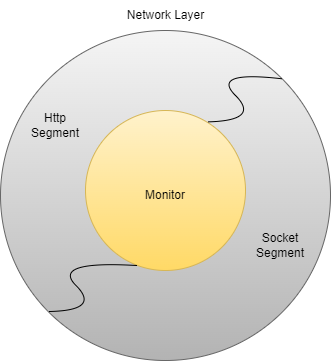
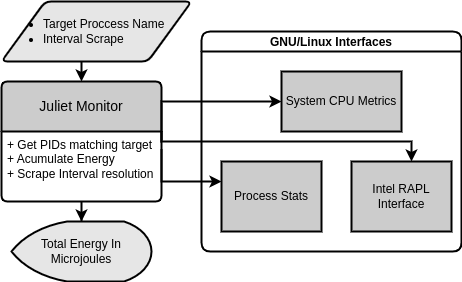
**Capa de Red:** la capa más externa es la capa de Red o la capa de comunicaciones. Esta capa le otorga a la herramienta la capacidad de establecer conexiones, enviar y recibir datos desde otros elementos de la red. Esta capa está dividida en dos segmentos: Segmento para comunicaciones tipo socket y segmento para comunicaciones tipo http. **El Segmento para comunicaciones tipo socket** está destinada exclusivamente a manejar las comunicaciones entrantes y salientes orientadas a Sockets con streams de datos. Esta capa hace uso de paquetes de confirmación ACKs para asegurar la entrega correcta de los datos entre las partes. Por otro lado, el **segmento para comunicaciones http** como su nombre lo indica maneja las comunicaciones entre la herramienta de monitoreo y el resto de componentes que requieran comunicación a través de los protocolos http. Finalmente en el centro de la estructura se encuentra el **Núcleo Monitor.** Este es el sistema monitor por sí mismo que recolecta la información del sistema operativo y se la entrega a la capa de Red para su posterior transmisión.

Ilustración 5 Diagrama de Capa de Red

Juliet Observer, al igual que herramientas como JoularJx, Scaphandra o PowerTop se apoya de las interfaces RAPL de los procesadores Intel para obtener los datos del consumo energético total del sistema, y en función de esto realiza los cálculos para determinar el consmo energético de un proceso que se ejecuta en el sistema operativo.

Ilustración 6 Arquitectura Monitor Juliet

A continuación, se muestra un gráfico de la arquitectura general del Monitor de Juliet



Se muestra que para iniciar el monitoreo, se recibe el nombre o PID del programa objetivo y adicionalmente el intervalo de muestreo en el cual se accede a la Interfaz RAPL, a las estadísticas de los procesos del sistema y a las métricas de uso de CPU. El observador recibe del ejecutor la orden de iniciar el monitoreo de un proceso en la instancia donde se está ejecutando y una vez iniciado el hilo de monitoreo del proceso objetivo queda a la escucha de futuras órdenes por parte del ejecutor.

Cuando llega la orden de detener el monitoreo, se detendrá el hilo de monitoreo obteniendo el total de energía consumido por el proceso objetivo y finalmente reportando estas métricas al colector, procediendo a la clausura de la conexión con el ejecutor y reiniciando la escucha a nuevas conexiones.

El observador se configura con un intervalo de muestreo el cual es el diferencial de tiempo en el que accede a las métricas del sistema anfitrión. El monitor accede a la interfaz de RAPL al inicio de un intervalo de muestreo para obtener la energía total consumida por el sistema hasta ese punto inicial, luego de finalizar el intervalo de muestreo vuelve a consultar la interfaz RAPL para obtener la diferencia entre el valor de la medición final y la inicial. Esta diferencia corresponde al consumo energético de todo el sistema en ese intervalo de muestreo.

Además de esta diferencia de energía, el observador consulta las estadísticas propias del sistema operativo para obtener el porcentaje total del uso del cpu en el intervalo de muestreo y el porcentaje total de uso de cpu del proceso objetivo de análisis.

## COLLECTOR

El colector es un componente del framework enfocado a ser un micro servicio del tipo REST que expone una API para la inserción y consulta de métricas dentro de la base de datos de resultados. Está escrito en Java con Spring Boot y el manejador de dependencias Maven.

# EXPERIMENTOS Y RESULTADOS

## ESCENARIO DEL EXPERIMENTO

Como objetivos se planteó realizar un estudio empírico del consumo energético entre sistemas manejadores de base de datos relacionales y no relacionales utilizando el **Framework Juliet** construido con tales fines. Como representante de los SMBD relacionales se analizó PostgreSQL y para las bases de datos NO-SQL se estudió el desempeño de MongoDB.

## BENCHMARK TPC-H TRADUCCION A MONGODB

El benchmark TPC-H es un conjunto estandarizado de consultas y especificaciones diseñadas para evaluar el rendimiento de sistemas de bases de datos transaccionales y analíticas. Estas consultas, diseñadas en SQL, abordan problemas complejos de análisis de datos en escenarios de negocios reales. Uno de los objetivos de este proyecto de grado es trasladar estas consultas de SQL a NoSQL con MongoDB, es crucial entender tanto las diferencias fundamentales entre los dos paradigmas como las capacidades específicas de MongoDB.

### Diferencias Fundamentales entre SQL y NoSQL

SQL es un lenguaje utilizado principalmente para gestionar y consultar bases de datos relacionales. Estas bases de datos están estructuradas en tablas con relaciones definidas entre ellas, lo que facilita operaciones como JOINs y subconsultas.

NoSQL, en contraste, abarca una variedad de tecnologías, siendo MongoDB una de las más populares para documentos orientados a JSON. Las bases de datos NoSQL son más flexibles en cuanto a la estructura de datos y están diseñadas para manejar grandes volúmenes de datos distribuidos. MongoDB, en particular, organiza los datos en documentos dentro de colecciones, permitiendo una estructura más anidada y flexible.

La migración del benchmark y en general de un esquema de base de datos relacional, como PostgreSQL, a un modelo NoSQL como MongoDB, generalmente implica la reorganización de tablas en colecciones. Una de las estrategias posibles es hacer una **traducción directa**, donde cada tabla de la base de datos relacional se convierte en una colección en MongoDB sin realizar embeddings (anidación) de documentos. A continuación se detalla cómo se lleva a cabo este proceso y los principios detrás de la traducción directa aplicada.

### Traducción Directa

En una **traducción directa** de tablas a colecciones:

* Cada tabla en PostgreSQL se convierte en una colección en MongoDB.
* Las filas de la tabla se convierten en documentos dentro de la colección.
* Las relaciones entre tablas (JOINs) no se materializan dentro de los documentos, sino que se mantienen como referencias utilizando campos de claves foráneas.

Equivalencias Básicas Utilizadas

Tabla 2 Equivalencia de conceptos PostgreSQL MongoDB

|  |  |
| --- | --- |
| PostgreSQL | MongoDB |
| Tabla | Colección |
| Fila | Documento |
| Columna | Campo |
| Primary Key | \_id (opcional) |
| Foreign Key | Referencia (id u otro identificador) |

### Proceso de Migración de Tablas a Colecciones

Para realizar el proceso de migración de las tablas a colecciones se siguió un conjunto de pasos generales que se describen a continuación

#### Paso 1: Identificación de las Tablas y su Estructura

Antes de realizar la migración, es esencial comprender la estructura de cada tabla, incluyendo sus columnas, tipos de datos, claves primarias y claves foráneas.

CREATE TABLE orders (  
 o\_orderkey SERIAL PRIMARY KEY,  
 o\_custkey INT REFERENCES customer(c\_custkey),  
 o\_orderstatus CHAR(1),  
 o\_totalprice DECIMAL,  
 o\_orderdate DATE,  
 o\_orderpriority VARCHAR(15),  
 o\_clerk VARCHAR(15),  
 o\_shippriority INT,  
 o\_comment VARCHAR(79)  
);

Comando 5 Tabla orders en PostgreSQL

**Ejemplo: Tabla orders en PostgreSQL:**

#### Paso 2: Creación de la Colección Correspondiente en MongoDB

Cada tabla se mapea a una colección. No es necesario definir el esquema de antemano en MongoDB, ya que es una base de datos sin esquema. Los documentos dentro de la colección reflejan las filas de la tabla original.

{

"\_id": ObjectId("..."), // Equivalente a o\_orderkey

"o\_custkey": 12345,

"o\_orderstatus": "F",

"o\_totalprice": 1234.56,

**"o\_orderdate":** ISODate("2023-08-18T00:00:00Z"),

"o\_orderpriority": "1-URGENT",

**"o\_clerk":** "Clerk#000000123",

"o\_shippriority": 0,

**"o\_comment":** "Comment here..."

}

Comando 6 Tabla Orders in Mongo

#### Paso 3: Mantenimiento de Relaciones (Foreign Keys)

En PostgreSQL, las relaciones entre tablas se gestionan mediante claves foráneas. En MongoDB, estas relaciones se mantienen utilizando referencias explícitas.

**Ejemplo de** **Relación: *orders* y *customer***

{

"\_id": ObjectId("..."), // Equivalente a o\_orderkey

"o\_custkey": ObjectId("..."), // Referencia a \_id en la colección customer

"o\_orderstatus": "F",

...

}

Comando 7 Equivalencia Relación: orders y customer

En PostgreSQL, la relación entre orders y customer se gestiona mediante la clave foránea o\_custkey. En MongoDB, este campo se conserva tal cual como referencia a la colección customer:

#### Paso 4: Manejo de Consultas y Operaciones Relacionales

SELECT o.\*, c.c\_name

FROM orders o

JOIN customer c ON o.o\_custkey = c.c\_custkey;

Comando 8 PostgreSQL Consulta para Obtener Información de orders y customer

Dado que no se realizan embeddings para la traducción del benchmark, las operaciones que requieren combinar datos de varias colecciones (equivalente a JOINs) se manejan con el uso de la operación $lookup en el pipeline de agregación de MongoDB.

db.orders.aggregate([

{

$lookup: {

from: "customer",

localField: "o\_custkey",

foreignField: "\_id",

as: "customer\_info"

}

},

{

$unwind: "$customer\_info"

},

{

$project: {

"o\_orderkey": 1,

"o\_orderstatus": 1,

"customer\_info.c\_name": 1,

...

}

}

]

);

Comando 9 MongoDB Consulta para Obtener Información de orders y customer

#### Paso 5: Indexación

Para mantener un rendimiento adecuado, es crucial definir índices en los campos que se utilizan comúnmente en filtros, ordenaciones y uniones ($lookup). En MongoDB, los índices se pueden definir de manera similar a cómo se haría en PostgreSQL.

Ejemplo: Índice en o\_custkey en la colección orders:

***db.orders.createIndex({ o\_custkey: 1 });***

Consideraciones Adicionales

* **Escalabilidad:** Mientras que la traducción directa evita la complejidad de los embeddings, puede tener un impacto en la escalabilidad y el rendimiento, especialmente en grandes conjuntos de datos que requieren múltiples lookup.
* **Flexibilidad:** Este enfoque permite mayor flexibilidad para modificar el esquema en el futuro, ya que no hay dependencias rígidas entre documentos.
* **Mantenimiento:** Aunque más simple de implementar, puede requerir un mantenimiento adicional para gestionar la consistencia de las referencias entre colecciones.

### Estrategias de Traducción

#### Estrategia 1: Entender el Contexto Relacional

Las consultas SQL suelen estar diseñadas para operar sobre múltiples tablas relacionadas mediante claves foráneas. En MongoDB, estas relaciones deben gestionarse manualmente usando referencias o incorporaciones (embeddings). En este caso como se realiza una traducción directa de los modelos relacionales a colecciones de MongoDB se descartaron las incorporaciones o anidamientos por lo que, en las consultas las relaciones se manejan manualmente.

#### Estrategia 2: Uso del Pipeline de Agregación

MongoDB proporciona un framework robusto para realizar operaciones complejas de agregación que son fundamentales en las consultas TPC-H. Este pipeline permite realizar filtros (***$match***)***,*** proyecciones (***$project***), agrupaciones (***$group***), ordenamientos (***$sort***), y más.

Al traducir las consultas, la mayoría de las subconsultas y operaciones de agregación en SQL se traducen directamente al pipeline de agregación en MongoDB.

#### Estrategia 3: Uso de Subconsultas y Condiciones Complejas

En el benchmark es común encontrar subconsultas que se ejecutan dentro de otras consultas para filtrar resultados o calcular agregaciones específicas. En MongoDB, estas subconsultas se manejan mediante el uso de combinaciones de operaciones como ***$lookup***, ***$group***, y ***$cond***.

Especificaciones de los metodos de traduccion

##### Método 1: JOINs y Relaciones entre Tablas

En SQL, los JOINs son la forma primaria de combinar datos de diferentes tablas. En MongoDB, la operación $lookup permite realizar operaciones similares al JOIN de SQL.

**Ejemplo:** Como se mencionó anteriormente traducir un ***JOIN*** simple entre *orders* y *lineitem* se realiza mediante un ***$lookup*** que une las colecciones orders y lineitem basándose en la clave *orderkey*.

##### Método 2: Filtrado y Condiciones con *$match* y *$expr*

Las cláusulas ***WHERE*** de SQL se traducen directamente a ***$match*** en MongoDB, donde se definen las condiciones de filtrado. Para condiciones más complejas, como las expresiones en subconsultas, ***$expr*** permite construir condiciones lógicas y aritméticas que replican el comportamiento de SQL.

##### Método 3: Agrupaciones y Agregaciones con *$group*

Las operaciones de agrupamiento en SQL, realizadas con ***GROUP BY***, se manejan en MongoDB usando ***$group***. Cada agrupación puede incluir múltiples operaciones de agregación, como ***$sum, $avg, $min,*** y ***$max***.

##### Método 4: Ordenamiento con *$sort*

El ordenamiento en SQL, realizado con ***ORDER BY***, se traduce directamente a ***$sort*** en MongoDB. Es importante asegurarse de que los índices correctos estén en su lugar para optimizar la eficiencia del ordenamiento.

##### Método 5: Uso de Proyecciones con *$project*

SELECT

l\_shipmode,

SUM(case

when o\_orderpriority = '1-URGENT' or o\_orderpriority = '2-HIGH' then 1

else 0

end) as high\_line\_count,

SUM(case

when o\_orderpriority <> '1-URGENT' and o\_orderpriority <> '2-HIGH' then 1

else 0

end) AS low\_line\_count

FROM

orders,

lineitem

WHERE

o\_orderkey = l\_orderkey

and l\_shipmode in ('REG AIR', 'FOB')

and l\_commitdate < l\_receiptdate

and l\_shipdate < l\_commitdate

and l\_receiptdate between '1996-01-01' and '1996-12-31'

GROUP BY

l\_shipmode

ORDER BY

l\_shipmode;

Comando 10 Consulta SQL de ejemplo de traducción

Las cláusulas ***SELECT*** en SQL, que determinan qué campos incluir en los resultados, se traducen a ***$project*** en MongoDB. Esto permite incluir o excluir campos específicos en la salida final, similar a lo que se haría en SQL con ***SELECT*** especificando columnas. Para ilustrar estos métodos, tomemos un ejemplo de una consulta SQL del benchmark TPC-H y su traducción a MongoDB:

db.lineitem.aggregate([

{

$lookup:

{

from: "orders", localField: "l\_orderkey", foreignField: "o\_orderkey", as: "orderDetails"

}

},

{ $unwind: "$orderDetails" },

{

$match:

{

l\_shipmode:

{

$in: ["REG AIR", "FOB"]

}, l\_commitdate: { $lt: "$l\_receiptdate" }, l\_shipdate: { $lt: "$l\_commitdate" }, l\_receiptdate: { $gte: new Date("1996-01-01"), $lt: new Date("1997-01-01") }

}

},

{

$group: {

\_id: "$l\_shipmode",

high\_line\_count: { $sum: { $cond: [{ $or: [{ $eq: ["$orderDetails.o\_orderpriority", "1-URGENT"] }, { $eq: ["$orderDetails.o\_orderpriority", "2-HIGH"] }] }, 1, 0] } },

low\_line\_count: { $sum: { $cond: [{ $and: [{ $ne: ["$orderDetails.o\_orderpriority", "1-URGENT"] }, { $ne: ["$orderDetails.o\_orderpriority", "2-HIGH"] }] }, 1, 0] } }

}

},

{ $sort: { \_id: 1 } },

{ $project: { \_id: 0, l\_shipmode: "$\_id", high\_line\_count: 1, low\_line\_count: 1 } }

]);

Comando 11 Consulta Mongo de ejemplo de traducción

## CONFIGURACIONES EXPERIMENTALES

Se propone generar una base de conocimiento definida sobre el efecto de diferentes tipos de optimizaciones en consumo energético de ambos sistemas en una red multinodo.

El conjunto de configuraciones propuesto es el siguiente:

* **Base:** esta es la configuración de muestra o de referencia, se carga el benchmark TPCH en la base de datos sin ningún tipo de optimización.
* **Indexación:** se propone analizar el consumo energético del SMBD al crear índices de consulta.
* **Compresión:** adicionalmente se quiere analizar el desempeño energético cuando se le aplica compresión a las tablas o colecciones según sea el caso
* **Índice y Compresión:** combinar ambos métodos y determinar si es conveniente desde el punto de vista energético

Los experimentos fueron llevados a cabo en computadoras con procesadores Intel Core i7 de 8va generación con 12 núcleos, y 4 hilos por núcleo, una frecuencia de 3.75GHz, 16GB de RAM y 500GB de disco HDD. Todas con sistema operativo Linux Mint 21.1 GNU/Linux, Java 11, PostgreSQL 16 y MongoDB 7.0 En total se dispuso de 9 de estas computadoras, en una LAN cableada con cables RJ-45 cat 5e conectadas a un Switch central junto con un router que cumplía de servidor DHCP.

Para reducir al mínimo la demanda de recursos utilizados por los componentes del sistema operativo se aplicó sobre cada uno de los nodos un script de bash para la limpieza, y apagado de procesos innecesarios en nuestro experimento. Entre los procesos se puede mencionar la interfaz gráfica X11, y los servicios asociados al daemon de Docker. Por naturaleza los sistemas operativos son multiusuario y de multiacceso, sin embargo en este experimento se aseguró el uso exclusivo de cada computadora específicamente para la realización de los experimentos.

Como se mencionó anteriormente se utilizó Juliet para realizar dos (2) experimentos. Uno de ellos sobre el sistema manejador de Base de Datos PostgreSQL y otro sobre MongoDB. Estos experimentos se realizaron por separado. De esta forma minimizamos la posible interferencia en el uso de los recursos entre ambos sistemas.

Se distribuyeron las 9 computadoras de la siguiente manera:

* 4 computadoras cumpliendo los roles de servidores
* 4 computadoras haciendo los roles de Clientes
* 1 computadora configurada con Analista y Colector

Cada una de estas computadoras cuenta con un hostname que la identifica en la red lo que hace más sencilla la comunicación entre ellas, esta es una buena práctica si se desea extender aún más esta red de tal manera que la conexión sea a orientada al hostname y no por IPs.

### Computadoras con Rol de Servidor

Las computadoras con el Rol de Servidor están configuradas conforme al diseño propuesto en los capítulos anteriores. Cuentan con los dos sistemas manejadores de base de datos en estudio, inicialmente con los procesos de ambos sistemas detenidos. Cada uno de estos sistemas manejadores dentro de cada computadora fueron configurados para permitir las conexiones externas desde otros dispositivos de la red. Se configuró cada una con las dependencias necesarias para llevar a cabo su rol de Servidor, entre estas se pueden mencionar el compilador G++ para la compilación del Software Monitor Juliet. Adicionalmente se configuraron los permisos de acceso a la interfaz RAPL de los procesadores Intel dentro del sistema operativo para monitorear adecuadamente el consumo energético total del sistema y determinar adecuadamente el consumo energético de los procesos en cada base de datos asociados a la ejecución. De igual forma se cuenta con acceso de Superusuario en el sistema operativo para acceder a las interfaces de Metricas y Uso de CPU del sistema.

### Computadoras con Rol de Cliente

En las computadoras clientes se limitó al máximo los procesos que no formaban parte del experimento utilizando el mismo script mencionado anteriormente. Se configuró cada una de estas con las dependencias necesarias para la ejecución, al igual que en la configuración de las computadoras con rol servidor se instaló el compilador G++ para construir el Software Monitor Juliet, y se configuró el acceso a las interfaces de energía y estadísticas de uso de CPU de cada proceso en el sistema operativo.

### Computadora con Rol de Analista y Colector

Este es el centro de mando del experimento, los procesos de esta máquina no fueron limitados. Cuenta con interfaz gráfica y las dependencias de python y Java necesarios para este Rol, este nodo de la red NO está en estudio, únicamente cumple la labor de captación de resultados y es quien da inicio a cada experimento enviando las señales adecuadas a cada computadora cliente, por lo que para este experimento no se consideró relevante analizar el consumo energético de este Rol. El Colector está dotado con un sistema manejador de base de datos PostgreSQL y un microservicio construido en Java 11 con Spring Boot 3 que expone una REST Api en la red para la inserción de cada resultado reportado tanto por los servidores como por los clientes en una base de datos de resultados.

### Experimentos en PostgreSQL

Del conjunto de las 8 computadoras destinadas al experimento (entiéndase que la novena es el Analista-Colector) se tomaron 4 pares Cliente-Servidor, y a cada par se le aplicó una de las configuraciones planteadas al inicio del capítulo. Para la indexación se utilizaron índices de tipo B-tree, y para la compresión se utilizó pg squeeze.

### Experimentos en MongoDB

De igual manera se configuraron los 4 pares de computadoras con cada una de las optimizaciones propuestas. Los índices que se crearon sobre las colecciones fueron tanto de campo único como de campo compuesto y para generar la compresión se empleó el motor WiredTiger nativo en Mongo, y se comprimió con la estrategia **zst**

## ALGORITMO DE EJECUCIÓN

El Orquestador es el componente principal que coordina las ejecuciones de las interacciones del experimento. El orquestador posee un conjunto de funciones y tareas para controlar tanto la comunicación entre componentes como la lectura y escritura de las métricas colectadas en el colector dentro de la red. Sin embargo a continuación se presenta el algoritmo que el orquestador utiliza para asegurar la recolección de métricas de energía y tiempo de ejecución. El algoritmo presentado a continuación es una simplificación para fines didácticos. Debe tenerse en cuenta que cada instrucción del algoritmo en algunos casos puede traducirse en una funcionalidad de complejidad considerable.

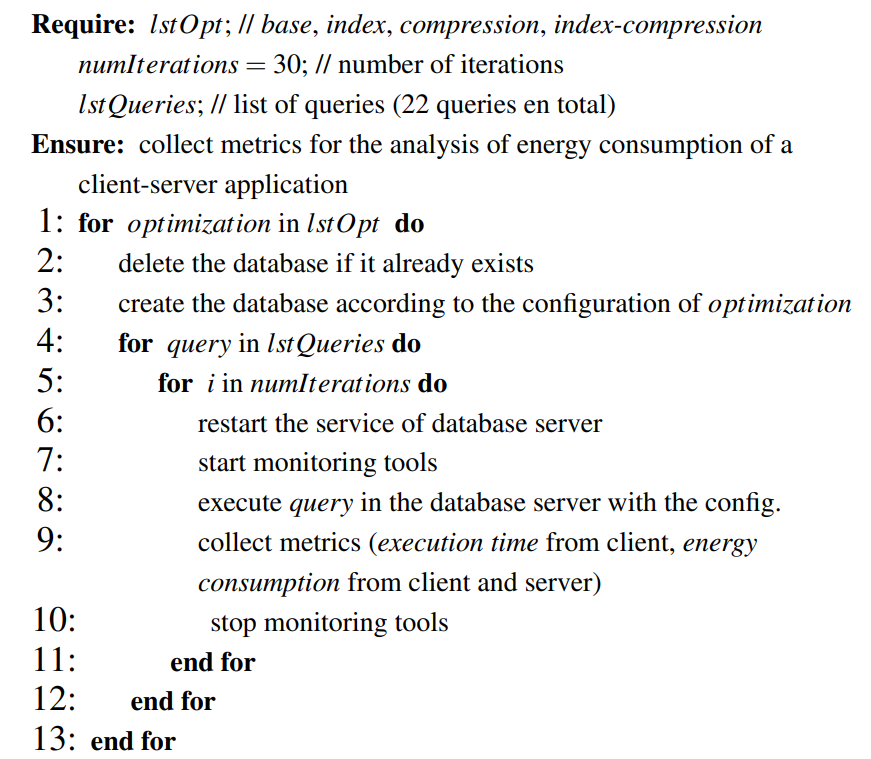


Ilustración 7 Algoritmo de experimentación

Inicialmente se requiere la lista de optimizaciones a aplicar durante el experimento, la lista puede contener un subconjunto de las configuraciones disponibles (*Véase CONFIGURACIONES EXPERIMENTALES* ). Adicionalmente se requiere la cantidad de iteraciones a realizar por cada configuracion y query, y la lista de las Queries del benchmark a utilizar, esta es una lista de números enteros, donde cada numero corresponde al numero de la query dentro del benchmark (*Vease: ORCHESTRATORS*).

Luego en cada optimización se ejecutan las queries seleccionadas la cantidad de iteraciones especificadas al inicio. Antes de ejecutar cada query se reinicia el servicio del SMBD y se inician las las herramientas de monitoreo, (los observadores) tanto del cliente como del servidor, estas herramientas colectaran las métricas tanto del tiempo de ejecución como de la energía total consumida una vez que la ejecución de la query haya finalizado y serán reportadas al colector.

# CONCLUSIONES

Desempeño energético

Tiempo de Ejecución

# REFERENCIAS

# ANEXO A

Ejemplo de archivo de Configuración Centralizado

{

"consoleCommand": "gnome-terminal",

"orchestrators": {

"postgres": {

"address": "192.168.2.109",

"port": "43317"

},

"mongo": {

"address": "192.168.2.109",

"port": "43318"

}

},

"databases": {

"postgres": {

"host": "localhost",

"port": "5432",

"name": "tpch",

"user": "postgres",

"password": "postgres",

"rootPassword": "123456",

"sshPort": "22",

"restartCommand": "systemctl restart postgresql",

"optimizationsDirectory": "postgres/client/optimization",

"queriesDirectory": "postgres/client/tpch-pgsql/query\_root/perf\_query\_gen"

},

"mongo": {

"host": "localhost",

"port": "27017",

"name": "tpch",

"user": "mongo",

"password": "mongo",

"rootPassword": "123456",

"sshPort": "22",

"restartCommand": "systemctl restart mongod",

"optimizationsDirectory": "",

"queriesDirectory": "mongo/client/tpch-mongo/queries"

}

},

"clientObservers": {

"postgres": {

"address": "192.168.2.109",

"port": "43319"

},

"mongo": {

"address": "192.168.2.109",

"port": "43320"

}

},

"serverObservers": {

"postgres": {

"address": "192.168.2.109",

"port": "43321"

},

"mongo": {

"address": "192.168.2.109",

"port": "43322"

}

},

"collector": {

"port": "43323",

"db\_host": "localhost",

"db\_port": "5432",

"db\_password": "\*\*\*\*\*\*\*\*",

"db\_user": "collector",

"directive": "http://192.168.2.109:43323/api/v1/directives/add",

"client-metrics": "http://192.168.2.109:43323/api/v1/client-metrics/add",

"server-metrics": "http://192.168.2.109:43323/api/v1/server-metrics/add",

"analyst-get-all": "http://192.168.2.109:43323/api/v1/analyst/getAll"

}

# TABLAS DE RESULTADOS

## Lista de Abreviaturas en las Tablas de Resultados

A continuación se presentan los resultados obtenidos durante la ejecución de los experimentos asociados al proyecto

Antes de esto se presenta una lista de abreviaturas necesarias para su correcto entendimiento

* **AEC:** (Average of energy consumption) Energia Promedio Consumida
* **RSEC:** (Relative standard deviation) Desviación Estándar Promedio
* **DEC:** (Difference of Energy Consumed) Diferencia entre la energía consumida en la optimización dada en comparación con la configuración base

## Tablas de Resultados PostgreSQL

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis PostgreSQL | | | | | | | |
| **Query Number** | **Base Configuration** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 4178.926 | 12061.078 | - | 5661113.610 | 539506.339 | - |  |
| 2 | 1884.837 | 1794.161 | - | 318178.606 | 11196.133 | - |  |
| 3 | 20605.111 | 33869.122 | - | 909481.866 | 20794.314 | - |  |
| 4 | 3488.395 | 2894.330 | - | 259773.921 | 306301.369 | - |  |
| 5 | 892.675 | 416.336 | - | 658101.295 | 22660.418 | - |  |
| 6 | 44079.352 | 100617.369 | - | 865956.050 | 22975.709 | - |  |
| 7 | 10554.403 | 25466.753 | - | 988634.518 | 15158.893 | - |  |
| 8 | 23597.668 | 29098.810 | - | 1061763.666 | 39340.659 | - |  |
| 9 | 3800.921 | 7325.865 | - | 1387179.578 | 116591.333 | - |  |
| 10 | 22109.229 | 52557.220 | - | 1504226.267 | 22002.223 | - |  |
| 11 | 754.092 | 132.248 | - | 120501.474 | 8679.400 | - |  |
| 12 | 1157.028 | 866.293 | - | 1304988.127 | 69154.870 | - |  |
| 13 | 4262.389 | 5728.730 | - | 1132142.856 | 16971.623 | - |  |
| 14 | 1870.046 | 1884.990 | - | 866623.024 | 18067.284 | - |  |
| 15 | 598.889 | 79.079 | - | 1955911.581 | 14842.874 | - |  |
| 16 | 26234.563 | 43931.315 | - | 789880.476 | 652420.056 | - |  |
| 17 | 11125.674 | 32717.917 | - | 6769017.103 | 43163.749 | - |  |
| 18 | 23646.982 | 40076.298 | - | 8424781.208 | 53852.241 | - |  |
| 19 | 49382.949 | 108084.866 | - | 1233248.341 | 11795.580 | - |  |
| 20 | 610.918 | 68.844 | - | 2526004.548 | 14198.020 | - |  |
| 21 | 1288.613 | 850.851 | - | 1646643.908 | 17362.672 | - |  |
| 22 | 3315.260 | 3312.905 | - | 177847844.008 | 1033306.873 | - |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis PostgreSQL | | | | | | | |
| **Query Number** | **Compression Optimization** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 4643.379 | 14737.708 |  | 6158819.031 | 573605.461 |  |  |
| 2 | 1258.687 | 887.684 |  | 437799.395 | 482042.925 |  |  |
| 3 | 14580.968 | 27919.431 |  | 877174.298 | 137046.470 |  |  |
| 4 | 1501.857 | 937.897 |  | 197817.351 | 256533.063 |  |  |
| 5 | 725.495 | 239.106 |  | 657871.198 | 10964.136 |  |  |
| 6 | 31757.517 | 81265.953 |  | 872765.044 | 20388.096 |  |  |
| 7 | 10124.670 | 23471.398 |  | 854191.546 | 23832.706 |  |  |
| 8 | 9438.074 | 23846.207 |  | 1098017.154 | 23575.959 |  |  |
| 9 | 4896.930 | 7397.345 |  | 1338601.073 | 15376.631 |  |  |
| 10 | 20167.094 | 51252.238 |  | 1471947.673 | 19682.207 |  |  |
| 11 | 767.981 | 197.636 |  | 113169.657 | 5177.758 |  |  |
| 12 | 862.889 | 448.784 |  | 1185554.317 | 12660.219 |  |  |
| 13 | 3439.565 | 5786.032 |  | 1188108.349 | 17990.806 |  |  |
| 14 | 1933.996 | 2322.917 |  | 853621.977 | 16873.254 |  |  |
| 15 | 700.438 | 301.350 |  | 1898350.169 | 14829.497 |  |  |
| 16 | 772.057 | 256.896 |  | 410202.855 | 16115.990 |  |  |
| 17 | 12830.608 | 31871.635 |  | 6477747.802 | 828093.040 |  |  |
| 18 | 11234.142 | 27607.064 |  | 8252975.535 | 35557.489 |  |  |
| 19 | 69361.046 | 116430.305 |  | 1183833.318 | 18923.852 |  |  |
| 20 | 699.211 | 210.084 |  | 2707266.128 | 263030.628 |  |  |
| 21 | 730.593 | 313.032 |  | 1582300.555 | 10790.208 |  |  |
| 22 | 3475.877 | 2407.815 |  | 189682366.852 | 1925191.439 |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis PostgreSQL | | | | | | | |
| **Query Number** | **Index Optimization** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 23542.480 | 68064.655 |  | 6750889.005 | 642128.683 |  |  |
| 2 | 1974.210 | 2093.163 |  | 365850.577 | 29947.952 |  |  |
| 3 | 29872.349 | 60403.232 |  | 896761.867 | 149450.501 |  |  |
| 4 | 1633.954 | 1435.984 |  | 226743.443 | 255304.903 |  |  |
| 5 | 3038.358 | 5560.774 |  | 968832.605 | 15214.370 |  |  |
| 6 | 19891.540 | 62024.764 |  | 906148.296 | 34107.879 |  |  |
| 7 | 7954.497 | 18273.651 |  | 1161749.473 | 84860.105 |  |  |
| 8 | 15693.510 | 24538.211 |  | 1710062.531 | 14544.062 |  |  |
| 9 | 5083.933 | 6700.036 |  | 190070.487 | 7714.284 |  |  |
| 10 | 34400.856 | 57604.872 |  | 818552.994 | 13877.868 |  |  |
| 11 | 579.258 | 48.035 |  | 51485.757 | 5358.998 |  |  |
| 12 | 626.112 | 213.176 |  | 770978.414 | 18937.084 |  |  |
| 13 | 4272.002 | 4815.560 |  | 1218227.898 | 8857.262 |  |  |
| 14 | 17199.918 | 38603.424 |  | 365163.831 | 340710.789 |  |  |
| 15 | 831.010 | 382.437 |  | 1712250.587 | 16820.255 |  |  |
| 16 | 835.800 | 209.099 |  | 407341.778 | 29708.967 |  |  |
| 17 | 16045.630 | 33331.892 |  | 5958463.558 | 49568.848 |  |  |
| 18 | 23868.429 | 41647.854 |  | 7533227.512 | 1347071.866 |  |  |
| 19 | 43425.741 | 85188.565 |  | 10734.663 | 1524.700 |  |  |
| 20 | 534.359 | 16.369 |  | 2888048.547 | 134.379 |  |  |
| 21 | 859.136 | 275.131 |  | 1679678.259 | 24591.655 |  |  |
| 22 | 5516.798 | 3454.802 |  | 32764.548 | 3662.607 |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis PostgreSQL | | | | | | | |
| **Query Number** | **Index & Compression Optimization** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 4274.160 | 10991.769 |  | 6477915.197 | 647257.238 |  |  |
| 2 | 1902.944 | 2041.831 |  | 309365.145 | 15581.204 |  |  |
| 3 | 19942.854 | 25234.069 |  | 859158.817 | 13183.670 |  |  |
| 4 | 965.091 | 804.123 |  | 230763.185 | 255751.730 |  |  |
| 5 | 861.340 | 523.161 |  | 921335.606 | 16042.932 |  |  |
| 6 | 20504.049 | 61045.810 |  | 928016.247 | 11645.247 |  |  |
| 7 | 9048.058 | 19486.071 |  | 1171790.274 | 24117.889 |  |  |
| 8 | 10256.747 | 21510.134 |  | 1552249.839 | 21446.148 |  |  |
| 9 | 6169.006 | 7430.516 |  | 179717.994 | 7990.984 |  |  |
| 10 | 14673.588 | 41919.871 |  | 740019.001 | 19668.385 |  |  |
| 11 | 541.712 | 103.489 |  | 52535.920 | 32098.604 |  |  |
| 12 | 1005.969 | 463.035 |  | 589728.606 | 7567.122 |  |  |
| 13 | 3314.340 | 5391.991 |  | 1103695.410 | 10451.544 |  |  |
| 14 | 1921.348 | 2843.585 |  | 219643.853 | 9123.925 |  |  |
| 15 | 553.398 | 173.813 |  | 1625036.657 | 25764.316 |  |  |
| 16 | 895.407 | 723.699 |  | 370624.692 | 13607.024 |  |  |
| 17 | 10375.671 | 26346.385 |  | 5674141.126 | 51046.007 |  |  |
| 18 | 7890.646 | 21037.307 |  | 8100672.222 | 69881.361 |  |  |
| 19 | 72523.146 | 122722.355 |  | 8316.220 | 1723.258 |  |  |
| 20 | 491.017 | 153.957 |  | 2669891.694 | 242511.694 |  |  |
| 21 | 2982.649 | 4239.218 |  | 1501310.609 | 25901.480 |  |  |
| 22 | 1934.297 | 3126.854 |  | 29261.490 | 3176.054 |  |  |

## Tablas de Resultados MongoDB

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis MongoDB | | | | | | | |
| **Query Number** | **Base Configuration** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 23725.723 | 39582.109 |  | 4773864.273 | 114137.018 |  |  |
| 2 | 9315.793 | 13924.742 |  | 964697.351 | 10399.978 |  |  |
| 3 | 4252.446 | 9684.243 |  | 21665987.448 | 987960.993 |  |  |
| 4 | 3772.859 | 3784.624 |  | 2477577.843 | 539634.753 |  |  |
| 5 | 19400.856 | 35761.051 |  | 421326202.454 | 6645416.163 |  |  |
| 6 | 3582.688 | 8601.049 |  | 5170464.392 | 155599.803 |  |  |
| 7 | 8178.329 | 3320.924 |  | 884124969.169 | 10289307.854 |  |  |
| 8 | 1492.875 | 912.774 |  | 13076411.209 | 280438.959 |  |  |
| 9 | 3733.779 | 10844.788 |  | 158110269.634 | 1901420.194 |  |  |
| 10 | 2069.825 | 2680.924 |  | 30972042.257 | 476183.781 |  |  |
| 11 | 2514.731 | 1797.737 |  | 165779783.679 | 1707394.971 |  |  |
| 12 | 872.353 | 789.718 |  | 5295828.337 | 93021.819 |  |  |
| 13 | 674.105 | 317.147 |  | 10343758.491 | 75968.841 |  |  |
| 14 | 3515.574 | 5881.840 |  | 5067737.972 | 44854.716 |  |  |
| 15 | 756.932 | 308.368 |  | 5644643.431 | 1621716.906 |  |  |
| 16 | 82431.850 | 36294.427 |  | 139487402.089 | 2063040.222 |  |  |
| 17 | 1076.136 | 893.758 |  | 31029156.165 | 428659.272 |  |  |
| 18 | 1033.958 | 763.236 |  | 23088649.149 | 984634.181 |  |  |
| 19 | 8079.784 | 4565.877 |  | 604104765.529 | 5033178.976 |  |  |
| 20 | 1500.531 | 843.008 |  | 5110651.103 | 64099.449 |  |  |
| 21 | 804.480 | 593.461 |  | 4860173.433 | 59114.965 |  |  |
| 22 | 3562.512 | 5576.967 |  | 633938.714 | 147329.589 |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis MongoDB | | | | | | | |
| **Query Number** | **Compression Optimization** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 22121.026 | 36904.656 |  | 4572205.522 | 271812.251 |  |  |
| 2 | 7205.105 | 10307.060 |  | 896643.440 | 15452.573 |  |  |
| 3 | 2508.639 | 5409.930 |  | 20044481.350 | 711720.339 |  |  |
| 4 | 2004.007 | 3728.473 |  | 2218528.541 | 273941.897 |  |  |
| 5 | 19542.970 | 54369.857 |  | 394135379.721 | 7764824.103 |  |  |
| 6 | 4957.170 | 10623.821 |  | 4953718.519 | 197874.568 |  |  |
| 7 | 17277.185 | 37366.267 |  | 840735461.002 | 10286385.802 |  |  |
| 8 | 1032.836 | 731.682 |  | 12047445.130 | 217505.772 |  |  |
| 9 | 4428.915 | 9784.194 |  | 148672628.385 | 1820190.339 |  |  |
| 10 | 1389.178 | 1870.296 |  | 29221907.809 | 253035.877 |  |  |
| 11 | 3487.486 | 2368.121 |  | 155768537.337 | 1807196.280 |  |  |
| 12 | 579.991 | 126.099 |  | 5030145.621 | 93867.777 |  |  |
| 13 | 926.806 | 782.282 |  | 9885495.816 | 310871.981 |  |  |
| 14 | 2141.821 | 4032.588 |  | 4789626.457 | 31754.018 |  |  |
| 15 | 915.727 | 556.894 |  | 4702151.509 | 1578718.041 |  |  |
| 16 | 88753.581 | 37663.103 |  | 131289082.710 | 1873244.774 |  |  |
| 17 | 2388.099 | 5086.073 |  | 29476440.012 | 377775.801 |  |  |
| 18 | 995.019 | 645.125 |  | 22068477.653 | 1227189.491 |  |  |
| 19 | 9698.242 | 4661.235 |  | 573889706.957 | 5237315.424 |  |  |
| 20 | 1883.793 | 1400.373 |  | 4874786.202 | 56681.562 |  |  |
| 21 | 803.249 | 489.279 |  | 4700628.708 | 43470.427 |  |  |
| 22 | 2586.472 | 4067.591 |  | 686368.736 | 10518.443 |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis MongoDB | | | | | | | |
| **Query Number** | **Index Optimization** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 34114.873 | 43948.030 |  | 684.905 | 164.306 |  |  |
| 2 | 6340.133 | 8429.323 |  | 962822.594 | 25514.469 |  |  |
| 3 | 2046.468 | 3380.133 |  | 26704772.287 | 1432966.361 |  |  |
| 4 | 2462.104 | 4355.327 |  | 1768907.030 | 28555.009 |  |  |
| 5 | 13857.157 | 32435.245 |  | 504454406.783 | 9400978.224 |  |  |
| 6 | 4563.824 | 8316.546 |  | 3693602.064 | 66446.559 |  |  |
| 7 | 7331.926 | 3896.895 |  | 1026445102.702 | 12866695.425 |  |  |
| 8 | 676.592 | 292.755 |  | 14567820.520 | 281395.601 |  |  |
| 9 | 1847.282 | 1715.873 |  | 181246000.518 | 2891554.080 |  |  |
| 10 | 2428.012 | 3690.688 |  | 19234695.404 | 269850.557 |  |  |
| 11 | 1592.708 | 1503.617 |  | 188366488.655 | 2244021.882 |  |  |
| 12 | 357.924 | 0.000 |  | 558.439 | 0.000 |  |  |
| 13 | 613.884 | 370.860 |  | 11443598.667 | 92842.918 |  |  |
| 14 | 1348.419 | 305.183 |  | 615.336 | 106.170 |  |  |
| 15 | 513135.000 |  |  | 664.006 |  |  |  |
| 16 | 71734.809 | 29149.019 |  | 163013255.826 | 3766773.426 |  |  |
| 17 | 6378.368 | 13288.322 |  | 36449954.853 | 266573.616 |  |  |
| 18 | 1351.754 | 2335.796 |  | 25531912.854 | 1105823.201 |  |  |
| 19 | 5147.628 | 3241.976 |  | 729855128.227 | 7298090.015 |  |  |
| 20 | 698.758 | 370.465 |  | 6132784.549 | 385735.442 |  |  |
| 21 | 628.790 |  |  | 870.020 |  |  |  |
| 22 | 14039.534 | 25019.743 |  | 1218404.496 | 14939.096 |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Energy Analysis MongoDB | | | | | | | |
| **Query Number** | **Index & Compression Optimization** | | | | | | |
|  | **Client** | | | **Server** | | | **Mean Execution Time** |
|  | AEC (μJ) | RSEC (μJ) | %DEC | AEC (μJ) | RSEC (μJ) | %DEC |  |
| 1 | 28505.122 | 28505.122 |  | 750.980 | 296.464 |  |  |
| 2 | 10678.429 | 10678.429 |  | 911809.058 | 26137.603 |  |  |
| 3 | 4458.371 | 4458.371 |  | 24687148.948 | 760504.279 |  |  |
| 4 | 4574.288 | 4574.288 |  | 1657962.300 | 27459.128 |  |  |
| 5 | 49005.215 | 49005.215 |  | 466552321.018 | 6416327.510 |  |  |
| 6 | 7411.193 | 7411.193 |  | 3429083.543 | 61497.976 |  |  |
| 7 | 2657.816 | 2657.816 |  | 953253666.174 | 11438164.396 |  |  |
| 8 | 1054.386 | 1054.386 |  | 13659989.664 | 199814.362 |  |  |
| 9 | 6205.744 | 6205.744 |  | 168121633.300 | 2635048.056 |  |  |
| 10 | 2179.227 | 2179.227 |  | 18043640.317 | 1016142.034 |  |  |
| 11 | 1135.765 | 1135.765 |  | 174647053.474 | 1336638.381 |  |  |
| 12 | 0.000 | 0.000 |  | 931.291 | 0.000 |  |  |
| 13 | 896.009 | 896.009 |  | 10559093.248 | 115388.620 |  |  |
| 14 | 6268.139 | 6268.139 |  | 636.992 | 27.864 |  |  |
| 15 | 0.000 | 0.000 |  | 660.030 | 0.000 |  |  |
| 16 | 36672.171 | 36672.171 |  | 151585089.715 | 2623235.030 |  |  |
| 17 | 1936.143 | 1936.143 |  | 33966338.937 | 238159.405 |  |  |
| 18 | 339.223 | 339.223 |  | 24264422.638 | 1593480.634 |  |  |
| 19 | 34307.900 | 34307.900 |  | 677518942.241 | 5844799.510 |  |  |
| 20 | 740.949 | 740.949 |  | 5562629.710 | 110606.642 |  |  |
| 21 | 0.000 | 0.000 |  | 635.492 | 0.000 |  |  |
| 22 | 29511.305 | 29511.305 |  | 1134770.391 | 9492.715 |  |  |