Comparación de las arquitecturas Kappa y Delta para el monitoreo remoto de pacientes basado en datos de sensores

Entregado como requisito para la obtencion del titulo de Master en Big Data

Emiliano Conti -

Tutor: Alejandro Bianchi

Universidad ORT

10 de octubre de 2024

Disclaimer

Yo, Emiliano Conti declaro que el trabajo que se presenta en esta obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba el Proyecto Final del Master en Big Data;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mi;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Firma: _____

Fecha: _____

Abstract

Aquí va tu resumen...

Índice general

Capítulo 1

Introducción

1.1. Descripción del Proyecto

Este proyecto compara las arquitecturas Kappa y Delta en el contexto del monitoreo remoto de pacientes mediante sensores IoT. En la era de la salud digital, estos sistemas generan grandes volúmenes de datos en tiempo real que requieren procesamiento eficiente. Se analizarán ambas arquitecturas, se definirán métricas de comparación y se implementarán en un caso de uso de monitoreo de pacientes. El objetivo es determinar la arquitectura más adecuada, considerando factores como latencia, escalabilidad, complejidad de implementación y manejo de datos históricos y en tiempo real.

1.2. Objetivos

1. Realizar un análisis teórico exhaustivo de las arquitecturas Kappa y Delta, detallando sus componentes, flujos de datos y casos de uso típicos.

2. Definir un conjunto de métricas y criterios para la comparación objetiva de ambas arquitecturas en el contexto del monitoreo remoto de pacientes.

3. Implementar ambas arquitecturas utilizando un conjunto de datos simulado de sensores de monitoreo de pacientes.

4. Ejecutar pruebas de rendimiento y funcionalidad en ambas implementaciones.

5. Analizar los resultados obtenidos y determinar la arquitectura más adecuada para el caso de uso específico de monitoreo remoto de pacientes.

6. Proporcionar recomendaciones para la selección e implementación de arquitecturas de procesamiento de Big Data en el ámbito de la salud digital.

1.3. Caso de Estudio: Big Data en Sistema de Salud

1.3.1. Contexto del Sistema

Sistema de salud integral que incluye perfiles de pacientes, telemedicina e integración con dispositivos IoT. El objetivo es mejorar la atención médica, con foco en prevencion, utilizando tecnologia.

1.3.2. Descripción del Caso de Uso

Monitoreo continuo y en tiempo real de la salud del paciente mediante el uso de Big Data. Se espera ademas, tener la capacidad de identificar patrones y tendencias en los datos medicos. Asi como tambien proporcionar recomendaciones personalizadas.

1.3.3. Proceso

1. **Recopilación de Datos**:
    - Dispositivos IoT (datos en tiempo real)
2. **Almacenamiento y Gestión**:
    - Almacenamiento de datos centralizada, segura y escalable
3. **Análisis de Datos**:
    - Procesamiento en tiempo real
    - Análisis histórico
    - Modelos predictivos (machine learning)
4. **Generación de Insights**:
    - Tableros
    - Alertas en tiempo real
5. **Intervención y Seguimiento**:
    - Monitoreo continuo
    - Feedback y mejora continua del sistema

Capítulo 2

Marco Teórico

2.1. Introducción al Big Data y Streaming de Datos

2.1.1. Definición y características del Big Data

Big Data es un término paraguas que se usa en la industria de IT para denominar a un conjunto de tecnologías que manejan grandes volúmenes de datos. La pregunta que se presenta entonces es: ¿qué tan grandes deberían ser estos volúmenes para ser considerados Big Data? O incluso, ¿existen otras características que definan lo que es Big Data? Una definición generalmente aceptada es la siguiente:

> Las tecnologías de Big Data (Datos Intensivos) están orientadas a procesar datos (conjuntos/activos) de alto volumen, alta velocidad y alta variedad para extraer el valor de datos previsto y asegurar una alta veracidad de los datos originales y la información obtenida, lo que demanda formas de procesamiento de datos e información (análisis) rentables e innovadoras para mejorar el conocimiento, la toma de decisiones y el control de procesos; todo esto exige (debe ser apoyado por) nuevos modelos de datos (que soporten todos los estados y etapas de los datos durante todo su ciclo de vida) y nuevos servicios y herramientas de infraestructura que permitan obtener (y procesar) datos de una variedad de fuentes (incluidas las redes de sensores) y entregar datos en una variedad de formas a diferentes consumidores y dispositivos de datos e información. (Demchenko et al., 2014)

Por lo que podríamos considerar que es Big Data todo aquello que esté orientado a datos cuyo volumen, velocidad y variedad no puedan ser tratados por un modelo de procesamiento de datos tradicional (como podrían ser las bases de datos relacionales). Con el objetivo de generar valor, asegurando la veracidad de los datos originales y la información obtenida.

2.1.2. Streaming de datos

El streaming de datos, también conocido como procesamiento de flujo, es un paradigma de procesamiento de datos en el que los datos se tratan como un flujo continuo e ilimitado de eventos discretos. En el contexto de Big Data, el streaming permite procesar y analizar grandes volúmenes de datos en tiempo real o casi real, a medida que se generan o llegan al sistema. (Hueske & Kalavri, 2019)

2.1.3. Desafíos en el manejo de datos de streaming

1. **Procesamiento en tiempo real y baja latencia**

   El procesamiento de datos debe ocurrir con un retraso mínimo para proporcionar resultados en tiempo real.

   Un desafío clave en el procesamiento de streams es lograr equilibrar la latencia, el costo y la correctitud simultáneamente (Akidau et al., 2015).

2. **Manejo de datos fuera de orden**

   Los datos pueden llegar en un orden diferente al que fueron generados, lo que complica el procesamiento.

   El procesamiento de eventos fuera de orden es un desafío fundamental en los sistemas de procesamiento de streams (Hueske & Kalavri, 2019, p. 87).

3. **Escalabilidad**

   Los sistemas deben poder manejar volúmenes crecientes de datos sin degradación del rendimiento.

   La escalabilidad en sistemas de streaming implica la capacidad de aumentar el rendimiento añadiendo recursos computacionales (Preuveneers et al., 2016).

4. **Tolerancia a fallos y consistencia**

   El sistema debe poder recuperarse de fallos sin pérdida de datos y mantener la consistencia de los resultados.

   Garantizar la semántica de "exactamente una vez" en presencia de fallos es un desafío significativo en el procesamiento de streams (Carbone et al., 2015).

5. **Procesamiento de ventanas temporales**

   Definir y procesar eficientemente ventanas de tiempo sobre streams de datos continuos.

   El procesamiento de ventanas temporales es fundamental en aplicaciones de streaming y requiere consideraciones cuidadosas en cuanto a la semántica del tiempo y la completitud de los datos (Akidau et al., 2015).

6. **Integración con sistemas batch**

   Combinar eficazmente el procesamiento de streams con sistemas batch existentes.

   La integración de paradigmas batch y streaming, a menudo referida como 'procesamiento híbrido', presenta desafíos únicos en términos de consistencia de datos y modelos de programación (Carbone et al., 2015).

2.1.4. Conceptos clave en el procesamiento de streaming

El procesamiento de streaming se refiere al análisis y manipulación de datos en tiempo real a medida que se generan o reciben. Según Carbone et al. (Carbone et al., 2015), los conceptos fundamentales incluyen:

- **Flujo de datos**: Una secuencia potencialmente infinita de registros que llegan continuamente (Akidau et al., 2015).
- **Latencia**: El tiempo entre la llegada de un dato y su procesamiento, crucial para aplicaciones en tiempo real (Akidau et al., 2015).
- **Ventanas**: Mecanismos para agrupar datos en intervalos finitos para su procesamiento (Akidau et al., 2015).
- **Estado**: Información que se mantiene entre eventos para cálculos incrementales (Carbone et al., 2015).
- **Watermarks**: Indicadores de progreso del tiempo en el flujo de datos (Akidau et al., 2015).

2.1.5. Comparación entre procesamiento por lotes y en tiempo real

La elección entre procesamiento por lotes y en tiempo real depende de los requisitos específicos de la aplicación:

Característica	**Procesamiento por lotes**	**Procesamiento en tiempo real**
Latencia	Alta (minutos a horas)	Baja (milisegundos a segundos)
Throughput	Alto	Moderado a alto
Complejidad	Menor	Mayor
Consistencia	Fuerte	Eventual
Uso típico	Análisis histórico, reportes	Monitoreo, alertas, decisiones inmediatas

Cuadro 2.1: Comparación de procesamiento por lotes y en tiempo real

Como sugiere Stonebraker et al. (Stonebraker et al., 2005), el procesamiento en tiempo real es esencial para aplicaciones que requieren decisiones inmediatas, mientras que el procesamiento por lotes es más adecuado para análisis profundos de grandes volúmenes de datos históricos.

2.1.6. Evolución de las arquitecturas de procesamiento de datos

La evolución de las arquitecturas de procesamiento de datos ha sido impulsada por la necesidad de manejar volúmenes cada vez mayores de datos en tiempo real:

1. **Arquitecturas por lotes**: Sistemas tradicionales como Hadoop MapReduce, diseñados para procesar grandes volúmenes de datos estáticos (Dean & Ghemawat, 2008).

2. **Arquitecturas de streaming puro**: Como Apache Storm, enfocadas en el procesamiento en tiempo real pero con limitaciones en la consistencia y exactitud (Toshniwal et al., 2014).

3. **Arquitectura Lambda**: Propuesta por Marz (Marz, 2011), combina procesamiento por lotes y en tiempo real para balancear latencia, throughput y tolerancia a fallos.

4. **Arquitectura Kappa**: Introducida por Kreps (Kreps, 2014), simplifica la Lambda tratando todos los datos como streams.

5. **Arquitectura Delta**: Desarrollada por Databricks, combina las ventajas de las arquitecturas Lambda y Kappa, optimizando el procesamiento de datos tanto en batch como en streaming (Armbrust et al., 2020) (Leano, 2020).

2.2. Arquitectura Lambda

2.2.1. Descripción General

La Arquitectura Lambda es un paradigma de procesamiento de datos diseñado para manejar grandes cantidades de información en sistemas de Big Data. Propuesta por Nathan Marz en 2011, esta arquitectura busca abordar las limitaciones de los sistemas de procesamiento por lotes (batch) y en tiempo real, combinando ambos enfoques para proporcionar una vista completa y actualizada de los datos.

2.2.2. Componentes Principales

La Arquitectura Lambda se compone de tres capas fundamentales:

Batch Layer

- Almacena el conjunto completo de datos históricos.
- Procesa periódicamente volúmenes arbitrarios de datos.
- Genera vistas pre-computadas para consultas eficientes.

Serving Layer

- Almacena las vistas pre-computadas de la capa de lotes.
- Proporciona acceso de baja latencia a los resultados.

Speed Layer

- Procesa datos en tiempo real.
- Genera vistas de estos datos.
- Mantiene los datos guardados unicamente hasta que la Batch Layer haya hecho el reprocesamiento de los datos historicos.

Vista Logica

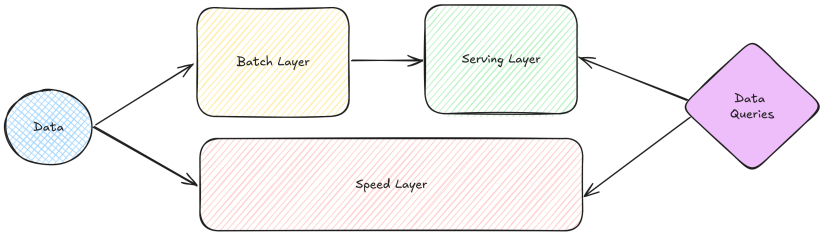

Figura 2.1: Diagrama de la Arquitectura Lambda

Implementacion Tipica

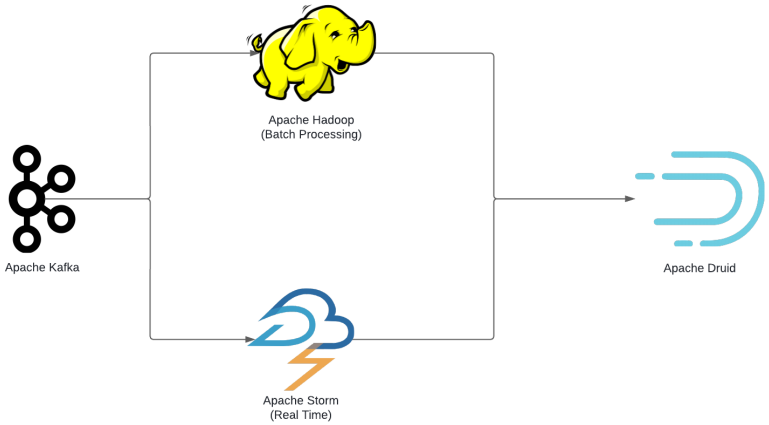

Figura 2.2: IMplementacion de la Arquitectura Lambda

2.2.3. Capacidades

- **Procesamiento de datos a gran escala**: Maneja eficientemente volúmenes masivos de datos.
- **Baja latencia**: Proporciona resultados en tiempo real para consultas.
- **Tolerancia a fallos**: Mantiene la integridad de los datos incluso en caso de fallos del sistema.
- **Escalabilidad**: Se adapta fácilmente al crecimiento del volumen de datos.

- **Flexibilidad**: Permite el procesamiento tanto por lotes como en tiempo real.
- **Consistencia eventual**: Garantiza que los datos eventualmente reflejarán todos los cambios.
- **Reprocesamiento**: En caso de necesitar reprocesar los datos, este proceso es trivial, pues se tiene almacenado el histórico completo.

2.2.4. Debilidades

- **Complejidad**: La implementación y mantenimiento pueden ser complejos debido a la duplicación de lógica en las capas de lotes y velocidad.
- **Latencia**: El procesamiento por lotes genera latencia debido al tiempo de la actualización de vistas.
- **Costo**: Al utilizar recursos computacionales diferentes entre el procesamiento batch y en stream, esto puede requerir varios nodos computacionales, lo que incrementa los costos.

2.2.5. Conclusiones

La Arquitectura Lambda ofrece una solución robusta para el procesamiento de Big Data, combinando las ventajas del procesamiento por lotes y en tiempo real. Aunque presenta desafíos en términos de complejidad, latencia y costo.

2.3. Arquitectura Kappa

2.3.1. Descripción General

La Arquitectura Kappa es un patrón de arquitectura de procesamiento de datos propuesto por Jay Kreps en 2014 como una simplificación de la Arquitectura Lambda. Su objetivo principal es unificar el procesamiento por lotes y en tiempo real en un único flujo de datos, eliminando la necesidad de mantener códigos separados para estos dos tipos de procesamiento. La Arquitectura Kappa se basa en la premisa de que todo es un flujo de datos (Stream) y que el reprocesamiento se puede lograr simplemente reproduciendo este flujo desde el principio.

2.3.2. Componentes Principales

Stream Store Layer

- Actúa como un registro inmutable de todos los eventos de datos entrantes.
- Permite la reproducción de datos históricos para reprocesamiento cuando se actualiza la lógica de procesamiento.

Stream Processing Layer

- Ingiere datos en tiempo real desde diversas fuentes.
- Procesa estos datos utilizando un sistema de procesamiento de streams.
- Aplica la lógica de negocio y las transformaciones necesarias a los datos entrantes.

Serving Layer

- Almacena los resultados procesados del stream.
- Proporciona acceso de baja latencia a los resultados.

Vista Logica

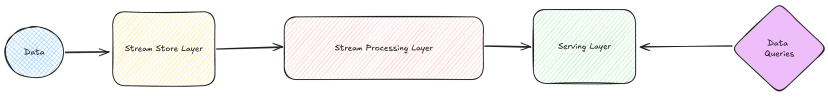

Figura 2.3: Diagrama de la Arquitectura Kappa

Implementacion Tipica

Figura 2.4: Implementacion de la Arquitectura Kappa

2.3.3. Capacidades

La Arquitectura Kappa ofrece varias capacidades clave:

- **Simplificación**: Al unificar el procesamiento batch y en tiempo real, reduce la complejidad del sistema.
- **Consistencia**: Garantiza la coherencia entre los resultados del procesamiento en tiempo real y el reprocesamiento.
- **Escalabilidad**: Se adapta fácilmente al crecimiento del volumen de datos.
- **Reprocesamiento**: Permite actualizaciones sencillas de la lógica de procesamiento mediante el reprocesamiento del stream.
- **Latencia reducida**: Proporciona resultados en tiempo real con menos latencia que Lambda para la mayoría de los casos de uso.

2.3.4. Debilidades

A pesar de sus ventajas, la Arquitectura Kappa tiene algunas limitaciones:

- **Complejidad tecnologica**: Requiere tecnologias con caracteristicas muy especificas en la capa de Stream Store.
- **Dependencia del almacenamiento**: Requiere un sistema de almacenamiento capaz de retener grandes volúmenes de datos históricos.
- **Complejidad**: Algunos análisis complejos pueden ser más difíciles de implementar en un modelo puramente basado en Streams.

2.3.5. Conclusiones

La Arquitectura Kappa representa una alternativa interesante en el diseño de sistemas de procesamiento de datos, ofreciendo una solución elegante para unificar el procesamiento batch y en tiempo real. Su enfoque en el procesamiento de streams como paradigma único simplifica la arquitectura general y reduce la complejidad del mantenimiento del código.

Mientras que es ideal para muchos casos de uso modernos de procesamiento de datos, especialmente aquellos que requieren resultados en tiempo real y flexibilidad en la actualización de la lógica de procesamiento, puede no ser la mejor opción para todos los escenarios.

2.4. Arquitectura Delta

2.4.1. Fundamentos y principios de diseño

2.4.2. Componentes clave

2.4.3. Manejo de datos en la arquitectura Delta

2.4.4. Ventajas y limitaciones

2.4.5. Casos de uso ideales

Capítulo 3

Metodología

3.1. Criterios de Evaluación para Arquitecturas de Streaming

Para evaluar y comparar las arquitecturas Kappa y Delta en el contexto del monitoreo remoto de pacientes, se considerarán los siguientes criterios:

3.1.1. Latencia de Procesamiento

- Tiempo de respuesta para el procesamiento de datos en tiempo real
- Capacidad para manejar picos de datos sin aumentar significativamente la latencia

3.1.2. Escalabilidad

- Capacidad para manejar un aumento en el volumen de datos
- Facilidad de agregar recursos computacionales según sea necesario
- Rendimiento bajo diferentes cargas de trabajo

3.1.3. Consistencia de Datos

- Garantía de consistencia entre datos en tiempo real y datos históricos
- Manejo de datos fuera de orden o retrasados

3.1.4. Tolerancia a Fallos

- Capacidad de recuperación ante fallos del sistema
- Prevención de pérdida de datos en caso de interrupciones

3.1.5. Manejo de Datos Históricos

- Eficiencia en el acceso y análisis de datos históricos
- Capacidad para reprocesar datos históricos cuando sea necesario

3.1.6. Costo Operativo

- Requisitos de hardware y software
- Costos de mantenimiento y operación a largo plazo

3.1.7. Seguridad y Cumplimiento Normativo

- Capacidad para cifrar datos en tránsito y en reposo
- Cumplimiento con regulaciones de protección de datos en salud (por ejemplo, HIPAA)

3.1.8. Rendimiento en Análisis Complejos

- Capacidad para realizar análisis en tiempo real de múltiples fuentes de datos
- Eficiencia en la ejecución de modelos de machine learning

Estos criterios servirán como base para una evaluación exhaustiva y objetiva de las arquitecturas Kappa y Delta en el contexto del monitoreo remoto de pacientes, permitiendo una comparación detallada y fundamentada.

Capítulo 4

Desarrollo

Contenido del capítulo...

Capítulo 5

Resultados

Contenido del capítulo...

Capítulo 6

Conclusiones

Contenido del capítulo...

Apéndice A

Primer Anexo

Contenido del anexo...

Bibliografía

Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, *34*(4), 42-47.

Dean, J., & Ghemawat, S. (2008). MAPREDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS. *Communications of the ACM*, *51*(1), 107-113. https://research.ebsco.com/linkprocessor/plink?id=1a5fefb3-a714-300d-bc4c-94603fe83a6f

Marz, N. (2011). *How to beat the CAP theorem* [Accessed on 2024-10-08]. Nathan Marz's Blog. Consultado el 8 de octubre de 2024, desde http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

Demchenko, Y., Laat, C. D., & Membrey, P. (2014). Defining architecture components of the Big Data Ecosystem. *2014 International Conference on Collaboration Technologies and Systems (CTS)*, *1*(2), 104-112.

Kreps, J. (2014). *Questioning the Lambda Architecture*. Consultado el 7 de octubre de 2024, desde https://www.oreilly.com/radar/questioning-the-lambda-architecture/

Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al. (2014). Storm@ twitter. *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 147-156.

Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R. J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., et al. (2015). The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, *8*(12), 1792-1803.

Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, *36*(4).

Preuveneers, D., Berbers, Y., & Joosen, W. (2016). SAMURAI: A batch and streaming context architecture for large-scale intelligent applications and environments. *Journal of Ambient Intelligence, Smart Environ-*

ments, *8*(1), 63-78. https://research.ebsco.com/linkprocessor/plink?id=afac9a51-b38e-3302-b299-be23cea08349

Hueske, F., & Kalavri, V. (2019). *Stream Processing with Apache Flink : Fundamentals, Implementation, and Operation of Streaming Applications* (First edition). O'Reilly Media.

Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Łuszczak, A., undefinedwitakowski, M., Szafrański, M., Li, X., Ueshin, T., Mokhtar, M., Boncz, P., Ghodsi, A., Paranjpye, S., Senster, P., ... Zaharia, M. (2020). Delta lake: high-performance ACID table storage over cloud object stores. *Proc. VLDB Endow.*, *13*(12), 3411-3424. https://doi.org/10.14778/3415478.3415560

Leano, H. (2020, 20 de noviembre). *Delta vs. Lambda: Why Simplicity Trumps Complexity for Data Pipelines*. Databricks. https://www.databricks.com/blog/2020/11/20/delta-vs-lambda-why-simplicity-trumps-complexity-for-data-pipelines.html