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

14 de noviembre 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 a Big Data y Streaming de Datos

2.1.1. Sistemas Distribuidos

Un sistema distribuido es una colección de elementos computacionales autonomos que para su usuario parecen un sistema único y coherente. (Tanenbaum & van Steen, 2020)

Los sistemas distribuidos tienen dos caracteristicas que pueden regularse para escalar: Procesamiento y Almacenamiento.

En el último tiempo, ha habido una tendencia a preferir que la escala de ambas propiedades sea individual. Es decir, que se pueda escalar por un lado la potencia de procesamiento y por otro la capacidad de almacenamiento.

Consistencia

La Consistencia es la propiedad que tiene un sistema distribuido en la que todos los nodos ven los mismos datos al mismo tiempo. Esto significa que cualquier lectura en cualquier momento deberá devolver el valor más reciente escrito para ese dato. Si un sistema es consistente, una vez que se realiza una escritura, todas las lecturas subsiguientes deben reflejar esa escritura; sin importar desde que nodo se hagan. Esta propiedad garantiza que los clientes de los sistemas nunca vean datos desactualizados o inconsistentes.

Disponibilidad

La Disponibilidad es la propiedad que tiene un sistema distribuido para responder a todas las peticiones, ya sean de lectura o escritura, sin fallos. Un sistema disponible garantiza que cada solicitud reciba una respuesta sin importar el estado individual de cada nodo que lo compone. Esto significa que incluso si algunos nodos estan caidos, el sistema en su conjunto debe poder seguir dando servicio a las peticiones que recibe.

Tolerancia a Particiones

La Tolerancia a Particiones es la propiedad que tiene un sistema distribuido en la que continua funcionando a pesar de la perdida de conectividad entre nodos. Una partición ocurre cuando hay una ruptura en la comunicación dentro de la red, lo que resulta en que dos o más segmentos de la red no puedan comunicarse entre sí. Un sistema tolerante a particiones puede seguir operando incluso cuando estas particiones ocurren, lo que significa que puede manejar retrasos o pérdidas de mensajes entre nodos sin fallar por completo.

2.1.2. 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 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.

2.1.3. 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)

2.1.4. Teorema CAP

El Teorema CAP es un concepto fundamental en el diseño de sistemas distribuidos. Este establece que es imposible garantizar al mismo tiempo, tanto la Consistencia (Consistency), Disponibilidad (Availability) y la Tolerancia a las Particiones (Partition Tolerance).

Según esto, un sistema distribuido sólo es capaz de garantizar dos de estas propiedades al mismo tiempo. En general, para los sistemas de Big Data de Streaming, la disponibilidad es una propiedad obligatoria, ya que cualquier inactividad puede resultar en la pérdida de datos valiosos o en la imposibilidad de realizar acciones.

Por otro lado, la Tolerancia a Particiones es también indispensable para estos sistemas, que por su naturaleza requieren que su capacidad de procesamiento este distribuida a través de múltiples nodos dispersos en una red no confiable; por lo que son suceptibles a que se genere una partición. Por lo tanto, si no tuviera esta propiedad el servicio podría dejar de ser disponible.

Entonces, como corolario, un sistema de Big Data de Streaming debe tambien ser tolerante a las particiones para poder ser disponible. Esto nos deja con una única opción: relajar el "grado de consistencia" hasta un punto razonable que permita que el sistema siga siendo eficáz.(Muñoz-Escoí et al., 2019)

Consistencia Eventual

La consistencia eventual es un modelo de consistencia en sistemas distribuidos que garantiza que, si no se realizan nuevas actualizaciones a un objeto, en algun momento (eventualmente) todos los accesos a ese objeto devolverán el último valor actualizado. La consistencia eventual se alinea con las compensaciones descritas por el teorema CAP, permitiendo que estos sistemas prioricen la disponibilidad y la tolerancia a particiones. Además, facilita la escalabilidad horizontal, crucial para manejar el crecimiento continuo de datos y clientes de los sistemas. Por último, es importante diseñar cuidadosamente el sistema para manejar las posibles inconsistencias temporales y asegurar que la aplicación pueda tolerar y resolver estas situaciones de manera apropiada (Muñoz-Escoí et al., 2019)

2.1.5. 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 eventual 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).

2.1.6. 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).

2.1.7. 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.

2.1.8. 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. Tecnologias para Streaming en Big Data

2.2.1. Mensajería Distribuida

Las tecnologias de mensajería distribuidas en tiempo real cumplen el crucial rol de actuar como intermediarios entre las fuentes de datos y los sistemas que efectivamente procesan estos datos. (Kleppmann, 2018)

Deben funcionar como un conducto de alta capacidad de almacenamiento y baja latencia, capturando y canalizando los flujos de información desde sus múltiples orígenes y hacia sus diversos destinos en tiempo real. (Marz & Warren, 2015)

Su papel es fundamentalmente el de un sistema nervioso central, coordinando y distribuyendo datos a través de complejas arquitecturas distribuidas. Actúan como amortiguadores, absorbiendo picos en el flujo de datos y garantizando un procesamiento constante y eficiente. Además, estas tecnologías sirven como una capa de abstracción, desacoplando los productores de datos de los consumidores, lo que permite una mayor flexibilidad y escalabilidad en el diseño del sistema.

Apache Kafka

Apache Kafka es el estandar de facto de este tipo de sistemas. Utiliza un modelo de publicación-subscripcion (pub/sub) basado en logs, donde los datos se envían a "topics" y se almacenan en particiones distribuidas. Las particiones tienen una garantía de orden de los mensajes y permiten la retención de datos a largo plazo.

Además, proporciona conectores para integración con diversos sistemas y un amplio ecosistema. A nivel de seguridad ofrece encriptación en tránsito mediante TLS y permite ser configurado para soportar encriptación en reposo (aunque esto debe hacerse a nivel de sistema de archivos).

Por último, su arquitectura distribuida y replicada permite una alta disponibilidad y tolerancia a fallos.

Apache Pulsar

Apache Pulsar es también una plataforma de mensajería y streaming distribuida, al igual que Apache Kafka, pero que se distingue por tener una arquitectura basada en capas, separando la capa de almacenamiento de la capa de procesamiento; lo que permite escalar cada uno independientemente.

Apache Pulsar soporta modelos de entrega como colas, publicación y suscripción y puede ofrecer garantías de entregar un mensaje exactamente una vez ("exactly-once"). Ofrece encriptación a nivel de mensaje, lo que permite una granularidad fina en cuanto a que encriptar. También soporta TLS para la encriptación en tránsito y permite la configuración de encriptación en reposo.

Por último, también soporta almacenamiento de mensajes a largo plazo y soporte nativo para esquemas: Esto es, permite definir la estructura y el tipo de datos de los mensajes, lo que a su vez permite una validación automática de los datos y una serialización/deserialización más eficiente. Esto trae consigo además, la capacidad de evolucionar estos esquemas de mensajes, de forma que productores y consumidores evolucionen independientemente.

Amazon Kinesis

Amazon Kinesis es un servicio de streaming de datos administrado en la nube de AWS. Está diseñado para recopilar, procesar y analizar datos de streaming en tiempo real a gran escala. Kinesis, en realidad, se compone de varios servicios:

1. Kinesis Data Streams para ingestión de datos en tiempo real
2. Kinesis Data Firehose para cargar datos en los servicios de almacenamiento disponibles de AWS
3. Kinesis Data Analytics para procesar datos con SQL o Java
4. Kinesis Video Streams para streaming de video

Adicionalmente ofrece capacidades de auto-escalado, replicación entre zonas de disponibilidad para alta durabilidad, encriptación en reposo (y puede habilitarsele la encriptación en tránsito) y permite la retención de datos hasta 365 días.

Como se puede ver, al ser tan completo permitiría implementar, al menos en principio, una gran parte de un sistema de Big Data en tiempo real.

Azure Event Hubs

Azure Event Hubs es un servicio de ingestión de datos en tiempo real administrado en la plataforma Microsoft Azure. Se supone que está diseñado para soportar millones de eventos por segundo con baja latencia. Al igual que Kinesis, ofrece una muy buena con otros servicios de Microsoft Azure, lo que permite por ejemplo capturar directamente los eventos en los servicios de almacenamiento disponibles en este proveedor de nube.

Event Hubs es compatible con el protocolo Kafka, lo que permite a las aplicaciones existentes de Kafka conectarse sin cambios de código. Ofrece una retención de mensajes por defecto de 1 dia que puede aumentado hasta 7. En caso de necesitar una retención más a largo plazo se recomienda guardar los eventos en Azure Blob Storage o Azure Data Lake para su posterior procesamiento. Cuenta tambien con encriptación en transito con TLS y en reposo.

Proporciona además, características como el procesamiento batch para optimizar el rendimiento, control de acceso basado en roles, y encriptación en reposo y en tránsito.

Comparación

Escalabilidad

- **Apache Kafka:** Alta escalabilidad horizontal, millones de mensajes/segundo.
- **Apache Pulsar:** Muy alta escalabilidad horizontal, millones de mensajes/segundo con separación de almacenamiento y cómputo.
- **Amazon Kinesis:** Buena escalabilidad, con 1.000 mensajes por segundo con la configuración por defecto aunque con configuración adicional puede llegar al millon por segundo hipotético.
- **Azure Event Hubs:** Buena escalabilidad, con 1.000 mensajes por segundo. También puede escalar con configuración adicional a los 20.000 mensajes. Existe la posibilidad de tener una instancia dedicada que permite escalar a millones de eventos por segundo de forma hipotética.

Retención de datos

- **Apache Kafka:** Configurable, potencialmente indefinida.
- **Apache Pulsar:** Ilimitada por diseño.
- **Amazon Kinesis:** Hasta 365 días, configurable.

- **Azure Event Hubs:** Hasta 7 días, opción de Capture para largo plazo.

Garantías de entrega

- **Apache Kafka:** At-least-once por defecto, exactly-once configurable.
- **Apache Pulsar:** Exactly-once nativo.
- **Amazon Kinesis:** At-least-once.
- **Azure Event Hubs:** At-least-once.

Encriptación

- **En reposo:**
  - **Apache Kafka:** Configurable.
  - **Apache Pulsar:** Nativo.
  - **Amazon Kinesis:** Por defecto (AWS KMS).
  - **Azure Event Hubs:** Por defecto.
- **En tránsito:** Todos soportan TLS/SSL.

Observaciones clave

- Pulsar destaca en escalabilidad y retención ilimitada.
- Kafka ofrece mayor flexibilidad en configuración.
- Servicios gestionados (Kinesis, Event Hubs) tienen encriptación en reposo por defecto, pero retención limitada.
- Pulsar ofrece exactly-once nativo, Kafka lo requiere configurable.

Característica	**Kafka**	**Pulsar**	**Kinesis**	**Event Hubs**
Escalabilidad	Alta	Muy alta	Alta	Alta
Retención	Configurable	Ilimitada	365 días máx.	7 días máx.
Garantía de entrega	Exactly-once	Exactly-once	At-least-once	At-least-once
Encriptación en reposo	Configurable	Sí	Sí (AWS KMS)	Sí
Encriptación en tránsito	Sí	Sí	Sí	Sí
Throughput	Muy alto	Muy alto	Alto	Alto

Cuadro 2.2: Comparación de Sistemas de Mensajería

2.2.2. Motores de Procesamiento

Si las tecnologías de mensajería distribuida son el sistema nervioso de un sistema de Big Data de Streaming, los motores de procesamiento podrían considerarse su cerebro.

Actúan como la capa que transforma, enriquece y analiza los datos cumpliendo varios roles:

- Aplicar la lógica de negocio sobre los datos mientras estos fluyen
- Detectar patrones y anomalías sobre el flujo de datos
- Mantener el contexto y estado necesario para las operaciones con históricos o agregaciones
- Garantizar la consistencia de las operaciones incluso ante fallos del sistema
- Distribuir la carga de trabajo entre diferentes nodos, paralelizando tareas

Estos componentes pueden enlazarse y programarse de diversas maneras. Permitiendo ensamblarlos de forma de cumplir con los requisitos de negocio.

Apache Spark

Apache Spark se destaca como uno de los motores de procesamiento más populares y versátiles en el ecosistema de Big Data. Ofrece dos APIs principales para el procesamiento en tiempo real: Spark Streaming y Structured Streaming, siendo esta última la más moderna y recomendada. Implementa el procesamiento en tiempo real tratando los datos streaming como microbatches y su enfoque de procesamiento es en memoria, siendo capaz de mantener su estado distribuido a través de checkpoints, que son archivos que se guardan en un sistema de almacenamiento al que todos los nodos pueden acceder. Su modelo de procesamiento le permite ofrecer un cierto equilibrio entre latencia y throughput. La abstracción fundamental de Spark son los RDDs (Resilient Distributed Datasets), que son colecciones inmutables de datos distribuidos que pueden ser procesadas en paralelo, y los DataFrames, que proporcionan una abstracción de más alto nivel similar a una tabla de base de datos. Spark destaca por su amplio soporte de lenguajes de programación:

- Scala
- Python
- Java
- R
- SQL

Su API unificada y extenso catálogo de bibliotecas incluye MLlib para machine learning, GraphX para procesamiento de grafos, y Spark SQL para procesamiento estructurado.

Apache Flink

Apache Flink adopta un enfoque nativo de streaming, tratando al procesamiento batch como un caso especial de streaming con límites finitos. Su arquitectura está diseñada para mantener estado distribuido con garantías de consistencia muy fuertes y latencias extremadamente bajas. El motor gestiona automáticamente la distribución del estado y los checkpoints, asegurando semánticas de exactly-once y permitiendo recuperación exacta ante fallos sin duplicados. Su modelo de procesamiento se basa en dos conceptos fundamentales:

- Marcas de agua (Watermarks): Son metadatos que fluyen en el stream de datos indicando el progreso del tiempo del evento, permitiendo manejar datos desordenados.
- Ventanas de tiempo (Windows): Permiten agrupar y procesar datos en intervalos temporales definidos, soportando diversos tipos como tumbling, sliding y session windows.

Flink proporciona APIs de diferentes niveles:

- ProcessFunction: API de bajo nivel que ofrece máximo control sobre tiempo, estado y ventanas
- DataStream API: API de alto nivel para operaciones de streaming comunes
- Table API y SQL: APIs declarativas para operaciones relacionales

Los lenguajes soportados son:

- Java
- Scala
- Python
- SQL

Apache Beam

Apache Beam, por su lado, se distingue por proporcionar un modelo de programación unificado que abstrae el motor de ejecución subyacente. Su potencia radica en la capacidad de escribir la lógica de procesamiento una vez y ejecutarla en diferentes motores de procesamiento (runners) como Spark o Flink. Esta capacidad de abstracción es particularmente valiosa en escenarios donde la portabilidad y la flexibilidad de despliegue son requisitos clave, permitiendo cambiar de motor de procesamiento según evolucionen las necesidades y sin reescribir el código de procesamiento.

Apache Samza

Apache Samza se distingue por su estrecha integración con Apache Kafka y su arquitectura diseñada para mantener el estado de procesamiento de forma distribuida con un modelo de particionamiento que permite escalar horizontalmente. Samza proporciona un modelo de procesamiento simple pero potente, con fuerte énfasis en la gestión de estado local y la tolerancia a fallos. Se utiliza en Linkedin y la arquitectura Kappa fué propuesta inicialmente pensando en la utilización de este motor de procesamiento.

Apache NiFi

Apache NiFi aborda el procesamiento de datos desde una perspectiva de orquestación y gobierno de datos; centrándose en la automatización del flujo de datos entre sistemas. Su arquitectura está orientada a la trazabilidad y auditabilidad de cada dato que fluye por el sistema, manteniendo un registro detallado de todas las transformaciones y movimientos. Los datos fluyen a través de un grafo de procesadores que pueden transformar, enrutar y mediar entre diferentes protocolos y formatos. NiFi se destaca por su capacidad para garantizar la entrega confiable de datos, proveer linaje de datos completo y permitir modificaciones de flujos en tiempo real sin necesidad de detener el sistema. Por último, NiFi proporciona una interfaz visual para diseñar, controlar y monitorizar flujos de datos.

Apache Kafka Streams

Apache Kafka Streams es una biblioteca de procesamiento de streaming que forma parte del ecosistema de Apache Kafka, diseñada para construir aplicaciones y microservicios de procesamiento en tiempo real. Opera con un modelo de procesamiento que permite operaciones con manejo de estado, incluyendo agregaciones por ventanas, manejo de múltiples streams de datos y transformaciones complejas mediante APIs de alto y bajo nivel. Tiene un manejo de estado distribuido que se gestionan con los mismos logs de Kafka. En cuanto a rendimiento, Kafka Streams alcanza una latencia típica de decenas de milisegundos a segundos, dependiendo de la complejidad del procesamiento y la configuración, mientras que su throughput puede escalar linealmente añadiendo más instancias. También cfrece grantías de procesamiento at-least-once con posibilidad de exactly-once mediante configuración.

Apache Pulsar Functions

Apache Pulsar Functions ofrece capacidad de cómputo integrandose directamente en la infraestructura de Apache Pulsar. Este framework permite implementar funciones livianas que procesan mensaje a mensaje. El manejo del estado es distribuido y se realiza mediante un almacenamiento basado en RocksDB, que permite mantener información entre invocaciones de funciones de manera consistente y tolerante a fallos. En términos de rendimiento, está optimizado para baja latencia, típicamente en el rango de milisegundos, gracias a su modelo de procesamiento directo. El throughput puede escalar horizontalmente añadiendo más instancias de funciones, y el sistema proporciona garantías de procesamiento exactly-once La arquitectura de este sistema está diseñada para ser simple y eficiente, permitiendo casos de uso como enriquecimiento de datos, filtrado, y transformaciones en tiempo real.

Comparación

Cuadro 2.3: Comparativa de Motores de Procesamiento Big Data

	Modelo	**Estado**	**Latencia**	**Throughput**	**Garantías**
Apache Spark	Micro-batches	En memoria	Media	Muy alto	At-least-once
Apache Flink	Streaming	Distribuido	Muy baja	Alto	Exactly-once
Apache Beam	Modelo unificado	Según runner	Variable	Variable	Según runner
Apache Samza	Streaming	Local por partición	Baja	Alto	At-least-once
Apache NiFi	Flujo dirigido	Local por procesador	Media-Alta	Medio	At-least-once
Kafka Streams	Streaming	Distribuido	Baja	Alto	Exactly-once
Pulsar Funcitons	Mensaje	Distribuido	Muy Baja	Alto	Exactly-once

2.2.3. Almacenamiento de Datos

Formatos de Almacenamiento

Los formatos de datos son un componente fundamental en cualquier arquitectura de sistemas de información moderna, ya que determinan no solo cómo se almacena la información, sino también cómo se procesa, transmite y analiza. La elección adecuada del formato de datos puede tener un impacto significativo en el rendimiento, la escalabilidad y la eficiencia del sistema en su conjunto. Para un sistema de Big Data, donde se manejan grandes volúmenes de información, la importancia de estos formatos se magnifica, ya que pueden significar la diferencia entre un sistema eficiente y uno que consume recursos excesivos. Además, los formatos de datos actúan como un lenguaje común entre diferentes componentes, facilitando la interoperabilidad y la integración de tecnologías.

Formatos Orientados a Filas

Los formatos orientados a filas representan la forma tradicional de almacenamiento de datos, donde cada registro se almacena de manera secuencial. Este enfoque ha sido la base de los sistemas de gestión de bases de datos durante décadas y sigue siendo crucial en muchos escenarios.

- Los registros completos se almacenan de manera contigua en disco
- Cada fila contiene todos los campos de un registro
- Optimizado para acceder a registros completos
- Los nuevos registros se añaden secuencialmente de forma eficiente
- Óptimo cuando las consultan necesitan todos los campos
- Fácil modificación de registros individuales
- Debe leer datos innecesarios cuando solo se necesitan algunas columnas
- Los datos heterogéneos juntos reducen la efectividad de los métodos de compresión
- Menos eficiente para análisis de columnas específicas

Formatos Orientados a Columnas

Los formatos de almacenamiento columnar representan un paradigma fundamental en el manejo de datos masivos, especialmente en entornos analíticos. A diferencia del almacenamiento tradicional orientado a filas, donde los registros se almacenan secuencialmente, el almacenamiento columnar organiza los datos por columnas, lo que ofrece ventajas significativas en ciertos escenarios.

- En lugar de almacenar registros completos de manera contigua, los datos se organizan por columnas
- Cada columna se almacena en bloques separados de memoria o disco
- Los valores similares se almacenan juntos, mejorando la compresión
- Los datos similares almacenados juntos permiten mayores tasas de compresión
- Solo se leen las columnas necesarias para una consulta
- Facilita operaciones como SUM, AVG, COUNT sobre columnas específicas
- Permite procesamiento eficiente de datos en hardware

Comparativa con Almacenamiento por Filas:

Consideremos una tabla simple de usuarios:
Formato por Filas:

```
[ID1, "Juan", 25] -> [ID2, "Ana", 30] -> [ID3, "Pedro", 28]
```

Formato Columnar:

```
IDs:     [ID1 -> ID2 -> ID3]
Nombres: ["Juan" -> "Ana" -> "Pedro"]
Edades:  [25 -> 30 -> 28]
```

Aspecto	Formato Columnar	Formato por Filas
Lectura parcial	Muy eficiente	Menos eficiente
Inserción de registros	Más lenta	Más rápida
Compresión	Alta	Moderada
Consultas analíticas	Excelente	Regular
Consultas transaccionales	Regular	Excelente

Formatos Específicos

JSON (JavaScript Object Notation) se ha convertido en el estándar de facto para el intercambio de datos en aplicaciones modernas, especialmente en entornos Web y APIs. Su popularidad se debe a su simplicidad, legibilidad humana y amplia compatibilidad con prácticamente todos los lenguajes de programación. A pesar de no ser el más eficiente en términos de espacio y rendimiento (ya que es un formato basado en filas), su flexibilidad para representar datos estructurados y semiestructurados lo hace invaluable en sistemas donde la interoperabilidad y la facilidad de desarrollo son prioritarias. Es particularmente útil en aplicaciones donde las transacciones individuales y la flexibilidad del esquema son más importantes que la eficiencia en el procesamiento de grandes volúmenes de datos.

Apache AVRO destaca como un formato de serialización de datos binario que combina la eficiencia del almacenamiento binario con la flexibilidad de esquemas evolutivos. Su característica más distintiva es su capacidad para manejar cambios en el esquema de datos a lo largo del tiempo sin requerir cambios en el código o reescritura de datos existentes. Para esto, AVRO almacena el esquema junto con los datos, lo que permite una deserialización precisa y eficiente. Es especialmente valioso en sistemas de mensajería y streaming de datos, donde la evolución del esquema y la eficiencia en la transmisión son cruciales. Su formato binario compacto y su capacidad de compresión lo hacen ideal para sistemas distribuidos donde el ancho de banda y el almacenamiento son consideraciones importantes.

Apache Parquet se ha establecido como el formato columnar dominante en el ecosistema de Big Data, especialmente para cargas de trabajo analíticas. Su diseño columnar permite una compresión altamente eficiente y un muy buen rendimiento en consultas que involucran solo un subconjunto de columnas. Parquet destaca particularmente en escenarios de análisis de datos, donde su capacidad para manejar esquemas complejos anidados y su integración con casi todas las herramientas lo hacen indispensable. La adopción generalizada de Parquet en la industria, lo ha convertido en el estándar de facto para almacenamiento de datos analíticos.

Optimized Row Columnar (ORC) inicialmente fué desarrollado para optimizar Hive, y aunque ofrece excelentes capacidades de compresión y rendimiento en consultas, su relevancia ha disminuido significativamente en los últimos años frente a Parquet. Aunque ORC sigue siendo relevante en sistemas legacy y específicos de Hive, la tendencia de la industria se ha movido claramente hacia Parquet como el formato columnar preferido para análisis de datos a gran escala.

Almacenamiento de Objetos en la Nube

El almacenamiento de objetos en la nube constituye un paradigma de almacenamiento donde los datos se organizan y gestionan como objetos independientes dentro de una estructura plana. Cada objeto almacenado comprende tres elementos fundamentales: los datos en sí mismos, un conjunto extenso de metadatos que describen y categorizan la información, y un identificador único global que permite su localización y recuperación. Dicho paradigma, se caracteriza por su naturaleza distribuida y su capacidad para manejar tanto datos estructurados como no estructurados, permitiendo almacenar desde documentos, archivos multimedia y hasta flujos de datos en tiempo real.

Los sistemas de almacenamiento de objetos se destaca por su capacidad para satisfacer las demandas contemporáneas de procesamiento de datos a gran escala. Como por ejemplo, ofrece una escalabilidad prácticamente ilimitada, pudiendo crecer según las necesidades sin preocuparse por restricciones de capacidad. Por otro lado, la durabilidad y disponibilidad de los datos se garantiza a través de la replicación automática en múltiples ubicaciones de forma automática y transparente. Adicionalmente, proveen APIs normalmente basadas en el protocolo HTTP que permite acceder de forma interoperable y estandarizada a los recursos almacenados

El uso de estos sistemas tambien conlleva sus propios desafíos. El más importante puede ser la latencia; pero también deben los grados de consistencia que ofrecen.

Formato de Tabla Analítica

Los formatos de tabla analítica son tecnologías diseñadas para resolver los desafíos del manejo de datos a gran escala. Surgen como respuesta a las limitaciones de los formatos de archivo tradicionales como Parquet y ORC cuando se trabaja con ellos en la nube.

Las características más importantes que aporta un formato de tabla analítica son:

- Proveen abstracciones sobre la metadata de archivos
- Permiten el uso de tablas con semántica SQL y evolución de esquema en las mismas
- Transacciones ACID
- Actualizaciones y Borrados
- Optimización de datos para mejoras de rendimiento
- Compatibilidad con múltiples motores de procesamiento
- Control de versiones en los datos

Los formatos de tablas analíticas dan a sus sistemas subyacentes estas características a través de diferentes mecanismos y estrategias. La principal es la gestión de metadatos, ya que implementan estructuras de datos altamente optimizadas que permiten rastrear eficientemente los archivos y sus cambios; mientras mantienen un historial detallado de transacciones que garantiza la consistencia de los datos, complementando con estrategias efectivas de particionamiento y organización.

Para la optimización del rendimiento, estos formatos emplean diversas técnicas como la compactación automática de archivos, estrategias de caching de datos para acceso rápido, utilización de formatos de archivo columnares como Parquet u ORC, y la incorporación de capacidades avanzadas de indexación y filtrado optimizado. La consistencia de los datos se garantiza mediante la implementación de transacciones ACID completas, que proporcionan un sólido aislamiento entre operaciones de lectura y escritura, manejan conflictos de manera automática y aseguran la consistencia en escenarios de operaciones concurrentes. En cuanto a la evolución y mantenimiento, estos formatos facilitan cambios de esquema sin interrupciones en el servicio.

Todas estas mecanismos trabajan en conjunto para proporcionar una solución completa para el manejo de datos a escala masiva, que como subproducto permite tratar la escritura de los archivos como un canal de mensajes sobre el que se puede hacer streaming.

Los exponentes más importantes de estos formatos son: Delta Table, Apache Iceberg y Apache Hudi

Delta Lake es un sistema de almacenamiento de datos diseñado por Databricks que utiliza archivos Parquet como base, organizándolos en una estructura de directorios con dos componentes principales: los archivos de datos en formato Parquet y el directorio _delta_log para metadatos y registro de transacciones. Esta arquitectura mantiene las ventajas de Parquet mientras añade capacidades transaccionales y de control de versiones, implementando un sistema de checkpoints para optimizar el rendimiento y un manejo de concurrencia que combina control optimista con serialización de escrituras.

Las características fundamentales de Delta Lake incluyen transacciones ACID completas, capacidad de acceso a versiones anteriores, evolución de esquema controlada, operaciones de Merge sofisticadas y optimización automática de datos mediante compactación de archivos y mantenimiento de estadísticas. El sistema también proporciona soporte para procesamiento de datos en tiempo real con semántica exactly-once y una integración robusta principalmente con Spark.

Apache Hudi desarrollado inicialmente por Uber, es una plataforma enfocada en crear una plataforma analítica transaccional, disponibilizando dos tipos principales de formatos de tablas: Copy On Write (optimizado para lecturas) y Merge On Read (optimizado para escritura).

Utiliza una Timeline para gestionar metadatos y registrar cronológicamente todas las acciones, implementando un sistema de indexación que permite optimizar operaciones y facilitar búsquedas rápidas, además maneja control de concurrencia optimista que mantiene lecturas sin bloqueos mientras serializa escrituras. Las características principales de Hudi incluyen procesamiento incremental para manejar streams de datos, gestión sofisticada de registros individuales con versionado a nivel de registro, borrado lógico y evolución de esquemas. También proporciona garantías de consistencia con transacciones ACID y semántica exactly-once, ofreciendo buena integración con motores de procesamiento como Spark y Flink.

Hudi, al ser una plataforma, no ofrece solo su formato de tabla analítica, sino también "Table Services" que son componentes computacionales que permiten la optimización como compactación automática y limpieza de almacenamiento.

Apache Iceberg diseñado inicialmente por Netflix, implementa una arquitectura de almacenamiento que se caracteriza por un modelo de metadatos enfocado en la evolución del esquema.

Utiliza una estructura jerárquica que separa completamente los metadatos de los datos, implementando control de versiones basado en snapshots atómicos e inmutables, donde cada snapshot representa un punto en el tiempo de la tabla y contiene referencias a todos los archivos de datos válidos para esa versión, permitiendo operaciones concurrentes sin necesidad de bloqueos pesados.

Entre sus características principales destacan una gestión de esquemas flexible que permite evolucionar tanto el esquema como la estrategia de particionamiento sin reescribir datos, una optimización de consultas avanzada basada en estadísticas detalladas a nivel de columna y archivos, y un sistema de control de concurrencia optimista.

Además, disponibiliza herramientas de mantenimiento como expiración de snapshots, compactación de archivos y reescritura de datos para optimización física, junto con una robusta integración con diferentes motores de procesamiento como Spark y Flink.

Bases de Datos Analíticas

Las bases de datos analíticas, también conocidas como bases de datos OLAP (Online Analytical Processing) orientadas a tiempo real, son sistemas especializados diseñados para procesar y analizar grandes volúmenes de datos con énfasis particular en consultas complejas y agregaciones. Estos sistemas están arquitecturalmente optimizados para procesar rápidamente consultas que involucran múltiples dimensiones y métricas sobre conjuntos masivos de datos, permitiendo análisis en tiempo real o casi real.

Estas bases de datos se distinguen por su capacidad de manejar cargas de trabajo analíticas complejas mientras mantienen latencias bajas y consistentes; especializandose en resultados en segundos o milisegundos.Esta característica se debe a su arquitectura orientada específicamente al análisis, que contrasta con los sistemas diseñados primariamente para transacciones (OLTP) o almacenamiento general de datos.

Por diseño permiten un análisis multidimensional eficiente, soportan alta concurrencia de usuarios, y pueden integrarse efectivamente con fuentes de datos en streaming. Además, su diseño orientado a columnas permite una compresión más eficiente y mejor rendimiento en consultas analíticas que típicamente involucran solo un subconjunto de ellas. Proporcionando capacidades avanzadas de agregación y pueden manejar eficientemente tanto datos históricos como en tiempo real.

Ninguno de estos beneficios vienen sin sus propios desafios, ya que se requiere una cuidadosa planificacion, operación y mantenimiento para ,amtemer su rendimiento. Además, más que nunca, es necesario organizar correctamente los indices y los esquemas de particionamiento son claves.

Ejemplos de estos sistemas son:

Apache Druid es una base de datos analítica distribuida diseñada principalmente para análisis en tiempo real de grandes volúmenes de datos de series temporales. Su arquitectura se distingue por su capacidad de ingesta en tiempo real combinada con consultas de baja latencia, utilizando un modelo de almacenamiento columnar híbrido que combina datos en memoria con almacenamiento en disco.

Apache Pinot fué desarrollado incialmente por LinkedIn y se enfoca en proporcionar análisis en tiempo real con latencias extremadamente bajas, incluso en escenarios de alta concurrencia de usuarios. Su arquitectura está optimizada para consultas de lectura masivas y paralelas. Además, destaca por su modelo de consistencia eventual y su capacidad para manejar esquemas dinámicos.

Apache Doris inicialmente fue desarrollado por Baidu, integra capacidades de almacenamiento columnar MPP (Procesamiento Paralelo Masivo) con funcionalidades OLAP, ofreciendo una solución más cercana a una base de datos tradicional pero con capacidades analíticas avanzadas. Su arquitectura es más simple en comparación con Druid y Pinot, lo que facilita la operación y mantenimiento, manteniendo un buen rendimiento para consultas analíticas.

2.2.4. Tecnologías Complementarias

Kubernetes

Kubernetes es una plataforma open-source diseñada para automatizar la implementación, el escalado y la gestión de aplicaciones en contenedores. Su principal fortaleza radica en su capacidad de escalado: puede escalar automáticamente las aplicaciones según la demanda en tiempo real, añadiendo o eliminando contenedores según sea necesario. Esto significa que puede aumentar los recursos cuando hay picos de tráfico y reducirlos en momentos de baja actividad, optimizando así los costos de infraestructura. Además, maneja automáticamente la distribución de carga entre contenedores, la recuperación ante fallos y actualizaciones continuas sin tiempo de inactividad, lo que lo hace especialmente valioso para aplicaciones que requieren alta disponibilidad y escalabilidad como las necesarias para Big Data.

Prometheus

Prometheus es un sistema de monitorización y alerta open-source, Es especialmente eficaz para monitorizar entornos dinámicos como los que se encuentran en arquitecturas de microservicios y contenedores. Permite recolectar y analizar métricas en tiempo real de manera altamente eficiente y tiene un potente lenguaje de consultas. Destaca por su capacidad para manejar millones de métricas simultáneamente y su arquitectura pull-based, que le permite escalar horizontalmente sin problemas. Además, cuenta con un sistema de alertas flexible y puede integrarse fácilmente con Kubernetes y otras herramientas de orquestación, lo que lo convierte en una herramienta fundamental para monitorizar aplicaciones modernas a gran escala.

Grafana

Grafana es una plataforma de visualización y análisis de datos de open-source que destaca especialmente cuando se combina con Prometheus. Su principal valor radica en su capacidad para transformar datos complejos en visualizaciones claras e interactivas a través de tableros personalizables. Cuando se utiliza junto con Prometheus, Grafana actúa como la capa de visualización, permitiendo crear paneles intuitivos que muestran en tiempo real el estado y rendimiento de los sistemas. Esto facilita la detección temprana de problemas, el análisis de tendencias y la toma de decisiones basada en datos. Una de sus características más poderosas es la capacidad de combinar datos de múltiples fuentes en un único dashboard, proporcionando una vista unificada del sistema.

K6

K6 es una herramienta open-source diseñada para realizar pruebas de rendimiento y carga con enfoque en desarrolladores, permitiendo escribir pruebas de carga en JavaScript. Puede exportar sus métricas directamente a Prometheus y visualizarlas en Grafana, creando así un ecosistema completo de pruebas y monitorización. Esto permite no solo ejecutar pruebas de carga, sino también analizar los resultados en tiempo real y mantener un histórico del rendimiento del sistema.

Apache Airflow

Apache Airflow es una plataforma open-source de orquestación que permite programar, ejecutar y monitorizar flujos complejos de tareas. En el contexto de Big Data, destaca especialmente por su capacidad para automatizar tareas de mantenimiento y procesos de manera robusta y programable. Su modelo de programación está basado en DAGs (Directed Acyclic Graphs), lo que permite definir dependencias claras entre tareas y asegurar que se ejecuten en el orden correcto. Una de sus mayores fortalezas es su capacidad para automatizar tareas rutinarias como la limpieza de datos antiguos, compactación de tablas, rebalanceo de particiones y regeneración de índices. Airflow puede orquestar estas tareas entre diferentes sistemas, manejando automáticamente los reintentos cuando las tareas fallan y permitiendo programar tareas condicionales basadas en el éxito de tareas previas.

2.2.5. Desafíos

Las tecnologías que se han mencionado son parte de los bloques que componen un sistema de Big Data para Streaming. Sin embargo, solo con el hecho de usarlas no alcanza para construir estos sistemas; debido a que existen desafíos que si bien las tecnologías ayudan a mitigar, deben tenerse en cuenta para cumplir con éxito el propósito que tienen dichos sistemas.

A continuación se describirán brevemente los desafíos más importantes que deben tenerse en cuenta a la hora de diseñar estos sistemas.

Procesamiento fuera de Orden

Los sistemas distribuidos tienen la característica de que los eventos pueden no ser procesados en el orden en que fueron generados. Por ejemplo, podría pasar que dos eventos generados en lugares geográficamente distintos viajen por dos conexiones de red con velocidades dispares y lleguen al sistema desordenados. Más aún, los componentes de un sistema distribuido estan

asimismos distribuidos por lo que podría pasar que aunque lleguen en ordern el procesamiento de dichos mensajes podría darse fuera de orden.

Esto provoca, fundamentalmente, una posible inconsistencia en los datos. Como se describió previamente, se puede decir que de todas formas eventualmente el sistema será consistente. El problema radica en que se necesita asegurar que el sistema es tan consistente como sea posible, tan seguido como sea posible. Para esto, lo que se suele hacer es tener una ventana de espera para los mensajes de forma de recibirlos si llegan un poco más tarde que otros mensajes o por otro lado complejizar los algoritmos de procesamiento utilizados para soportar procesamiento fuera de orden. En cualquier caso ambas soluciones llevan a un aumento de la latencia, por lo que se debe llegar a un compromiso entre una propiedad del sistema y la otra.

Manejo de Archivos Pequeños

Un desafío que puede parecer contraintuitivo es el Manejo de Archivos Pequeños. En este caso, los procesadores de datos reciben muchos archivos o eventos de poco tamaño, que no permiten aprovechar adecuadamente la capacidad de procesamiento y almacenamiento del sistema. Esto se debe a que leer y escribir muchos archivos pequeños es menos eficiente que leer y escribir pocos archivos grandes. Esto puede generar que se pase más tiempo gestionando estos archivos que procesando los datos en si mismos. Esto por su parte genera un problema de latencia.

Gestión del Estado

La gestión del estado en representa un gran desafío técnico en el procesamiento de datos. Estos sistemas deben manejar un flujo constante de datos entrantes mientras mantienen y actualizan información histórica necesaria para realizar cálculos y análisis significativos. Esta tarea se vuelve particularmente desafiante debido al volumen de datos que debe procesarse, junto con la necesidad de mantener tiempos de respuesta bajos y garantizar la precisión de los resultados.

La naturaleza distribuida de estos sistemas añade una capa adicional de complejidad ya que el estado debe distribuirse entre múltiples nodos de procesamiento, lo que introduce desafíos en términos de coordinación y consistencia. Cada nodo debe ser capaz de acceder y modificar su porción del estado de eficientemente, mientras mantiene una vista coherente del sistema en su conjunto.

Adicionalmente la tolerancia a fallos y la durabilidad del estado son aspectos críticos que no pueden ignorarse. Las limitaciones en los recursos re-

presentan otro desafío ya que la memoria disponible es finita, pero el estado puede crecer indefinidamente.

Todos estos desafíos deben abordarse pensando en mantener la disponibilidad, el rendimiento y la latencia del sistema en niveles aceptables.

Backpressure

El backpressure es un mecanismo que aborda el desafío del desequilibrio entre las tasas de producción y consumo de datos. Cuando los productores generan datos más rápidamente de lo que los consumidores pueden procesarlos, surge el riesgo de sobrecargar el sistema. El backpressure actúa como un sistema de control de flujo que permite evitar el desbordamiento de recursos.

La implementación efectiva de backpressure requiere coordinación entre los diferentes componentes del sistema de manera dinámica y adaptativo, ajustándose continuamente a las condiciones cambiantes del sistema. Sin embargo, implementar estos mecanismos de señalización de manera eficiente en sistemas distribuidos presenta desafíos significativos, especialmente cuando hay múltiples productores y consumidores involucrados.

La gestión del backpressure tiene implicaciones directas en la latencia y el rendimiento del sistema, ya que se puede generar un aumento en la latencia de procesamiento mientras el sistema se ajusta para manejar la carga. Esta compensación entre latencia y throughput debe manejarse cuidadosamente para mantener el sistema dentro de sus objetivos de nivel de servicio.

Reprocesamiento

El reprocesamiento surge cuando se necesita volver a procesar datos históricos debido a diversos factores como errores en el código, cambios en la lógica de negocio, o la necesidad de actualizar resultados anteriores. Este proceso, es complejo debido a la naturaleza continua y distribuida de los sistemas de streaming, donde el procesamiento de datos nuevos no puede detenerse mientras se realiza el reprocesamiento de datos históricos.

La gestión de recursos durante el reprocesamiento presenta desafíos particulares, ya que se debe balancear la carga entre el procesamiento de datos en tiempo real y el reprocesamiento de datos históricos. Esto es crítico para mantener el rendimiento del sistema y que el reprocesamiento termine en un tiempo razonable. La consistencia de los datos durante el reprocesamiento es otro aspecto crucial ya que se debe garantizar la coherencia con el estado actual del sistema y que no se degrade la calidad de los datos.

Conformidad Normativa

La conformidad normativa representa un desafío que va más allá de los aspectos puramente técnicos, abarcando requisitos legales, éticos y de privacidad que deben cumplirse rigurosamente. De las más importantes son HIPAA para datos de salud y GDPR para datos personales en Europa. Los sistemas deben implementar mecanismos robustos que garanticen la protección de datos sensibles mientras mantienen la funcionalidad y el rendimiento del procesamiento en tiempo real.

La gestión del consentimiento y los derechos de los usuarios presenta desafíos particulares en sistemas de tiempo real. Bajo GDPR, los usuarios tienen el derecho de acceder, modificar y eliminar sus datos personales (llamado "derecho al olvido"), lo que requiere que los sistemas puedan rastrear y gestionar estos datos a través de toda su estructura de procesamiento. En el caso de HIPAA, se requiere mantener registros detallados de quién accede a la información médica protegida y con qué propósito.

La seguridad y el cifrado de datos plantean otro conjunto de desafíos. Los datos sensibles deben estar cifrados tanto en tránsito como en reposo. El sistema debe ser capaz de cifrar y descifrar datos rápidamente sin introducir latencias significativas, mientras mantiene la integridad y la trazabilidad de los datos. Además, se deben implementar controles de acceso granulares y mecanismos de autenticación robustos que cumplan con los requisitos específicos de cada regulación, como las salvaguardas administrativas, físicas y técnicas requeridas por HIPAA.

Por su parte la retención y eliminación de datos sensibles presentan desafíos únicos. Las regulaciones pueden requerir que ciertos datos se conserven durante períodos específicos mientras otros deben eliminarse tan pronto como ya no sean necesarios. Se debe mantener un equilibrio entre cumplir con los requisitos de retención y mantener el rendimiento del sistema, especialmente cuando se trata de datos históricos que pueden ser necesarios para análisis o auditorías.

La auditoría y el monitoreo son aspectos cruciales para demostrar conformidad. Se debe mantener registros detallados de todas las operaciones de procesamiento de datos, especialmente aquellas que involucran datos sensibles o protegidos. Estos registros deben ser inmutables y seguros, permitiendo reconstruir quién accedió a qué datos, cuándo y con qué propósito. En el caso de HIPAA, esto incluye mantener registros de acceso a información médica protegida durante al menos seis años, mientras que GDPR requiere la capacidad de demostrar el cumplimiento en cualquier momento a través de registros detallados de procesamiento.

En sistemas distribuidos que procesan datos continuamente, implementar estas capacidades sin interrumpir el flujo de procesamiento o comprometer la integridad del sistema se vuelve extremadamente complejo.

2.3. Arquitecturas de Referencia

Una arquitectura de referencia representa una plantilla abstracta y probada que encapsula las decisiones arquitectónicas fundamentales de un sistema, mejores prácticas y experiencias acumuladas en un dominio específico. Esta proporciona un vocabulario común, además de componentes estandarizados y patrones de interacción que sirven como base para el desarrollo de los sistemas concretos. La arquitectura de referencia no solo define la estructura y comportamiento base del sistema, sino que también establece los principios de diseño, restricciones técnicas y mecanismos de extensibilidad que guiarán estas implementaciones.

2.3.1. Instancias de Arquitectura

Una plataforma que soporte el Monitoreo Remoto de Pacientes, necesita soportar grandes volúmenes de datos analizados como streaming. Además, para poder dar soporte al uso de la plataforma se requiere poder analizar el histórico de dichos datos. Por último, el tiempo en que el análisis de los datos de streaming es disponibilizado debe ser lo más cercano a tiempo real para poder tomar decisiones a tiempo para la salud del paciente.

Existen tres grandes familias de arquitecturas de referencia que cubren estos tres casos:

- Lambda
- Kappa
- Delta

De entre ellas, se instanciarán y se compararán Kappa y Delta; ya que son evoluciones propuestas sobre Lambda que siguen distintos caminos para alcanzar el mismo objetivo.

Realizar esta instanciación implica seleccionar las tecnologías que se usarán para cumplir las condiciones de la arquitectura de referencia; así como también componentes que si bien no son descritos por la arquitectura de referencia, son necesarios para cumplir con las características de calidad necesarias para el caso de uso propuesto.

Por último, las dos instancias de arquitectura implementadas utilizarán las mismas tecnologías, o tanto como sea posible, de modo que los resultados sean comparables.

2.4. Arquitectura Lambda

2.4.1. 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.

2.4.2. 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 Lógica

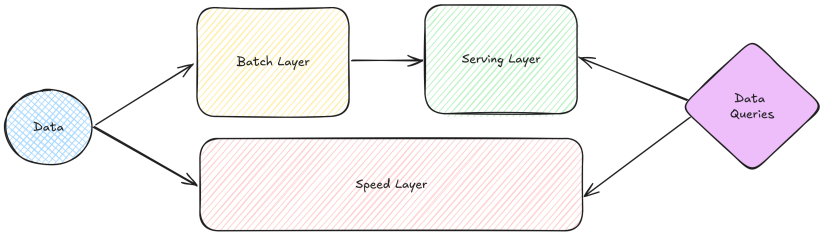

Figura 2.1: Diagrama de la Arquitectura Lambda

Implementación Típica

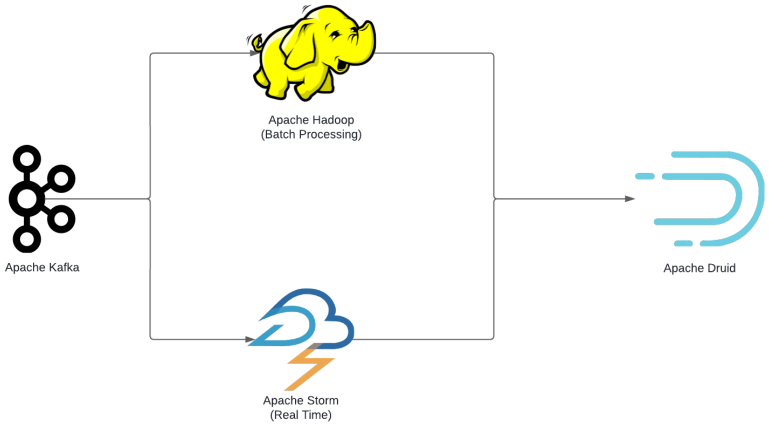

Figura 2.2: IMplementacion de la Arquitectura Lambda

2.4.3. 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.

2.4.4. Desafíos

- **Complejidad**: La implementación y mantenimiento pueden ser complejos debido a la duplicación de lógica en las capas de lotes y velocidad.
- **Latencia**: El procesamiento batch 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.

2.5. Arquitectura Kappa

2.5.1. Descripción General

La Arquitectura Kappa surge en 2014 como respuesta de parte de Jay Kreps a la Arquitectura Lambda. Si bien Lambda puede describirse de forma "sencilla" como una serie de transformaciones y además pone mucho enfasis en la posibilidad y facilidad de reprocesar los datos; tiene la desventaja de obligar a mantener código que debe producir el mismo resultado en dos sistemas distribuidos. Esto implica que cualquier cambio o mejora que reciba uno debe recibir un tratamiento de reingenieria para que el otro también lo tenga. Y, según argumenta Jay Kreps, la tendencia es a optimizar el código para uno de los motores (incluso si un mismo motor soporta dos modos de trabajo, la semántica que maneja será distinta por lo que terminará siendo una base de código distinta). (Kreps, 2014)

La Arquitectura Kappa busca responder la pregunta "Por qué un sistema de procesamiento de Streams no podría incrementar su paralelismo y reprocesar su historia muy rápido?"

La intuición detrás de esta arquitectura es la siguiente:

- Usar Kafka o algún otro sistema que permita tener la traza completa de los datos que se quieren reproducir para múltiples suscriptores
- Cuando se quiera hacer reprocesamiento, iniciar una segunda instancia de procesamiento que comience del principio de la historia
- Redirigir el procesamiento de la nueva historia a una tabla auxiliar
- Cuando el segundo procesamiento haya alcanzado al anterior, hacer que empiece a usarse la tabla auxiliar en lugar de la anterior
- Parar el procesamiento anterior y eliminar la tabla anterior

De todas maneras, el resultado del procesamiento o los estados intermedios pueden llegar a ser guardados a su vez por alguna otra herramienta para realizar procesamiento en batch.

2.5.2. 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.

2.5.3. Vista Lógica

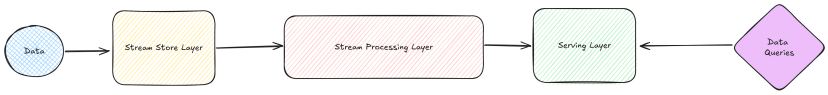

Figura 2.3: Diagrama de la Arquitectura Kappa

2.5.4. Capacidades

La Arquitectura Kappa ofrece varias capacidades clave:

- Reduce la complejidad del sistema al unificar el procesamiento batch y streaming
- Mejora la mantenibilidad al no tener que duplicar lógica para los distintos esquemas de procesamiento
- Garantiza la coherencia entre los resultados del streaming y el reprocesamiento.

2.5.5. Desafíos

A pesar de sus ventajas, la Arquitectura Kappa presenta algunos desafíos:

- Requiere un incremento en el uso de recursos muy grande cuando es necesario reprocesar los datos históricos.
- El costo de retención de eventos a largo plazo son enormes y vuelven prohibitiva esta arquitectura sin cambios.
- Es necesario reprocesar toda la historia en caso de que exista la necesidad de borrado de información

2.6. Arquitectura Delta

La Arquitectura Delta surge desde Databricks, al igual que la Arquitectura Kappa, como una respuesta a los desafíos que presenta la arquitectura Lambda. A diferencia de la suposición que hace Kappa de que todo puede ser tratado como un Stream, Delta por su parte, utiliza el almacenamiento de objetos en la nube como base y agrega por encima tecnología de metadata que otorga la posibilidad de utilizar este sistema de almacenamiento como un canal de mensajes de modo que se puede hacer Streaming sobre él; además de agregar otras capacidades. Esto permite utilizar un único flujo de datos tanto para análisis en tiempo real como análisis histórico.

2.6.1. Descripción General

Delta surge de la necesidad de procesar datos masivos a bajo costo; intentando aprovechar la estructura existente en las organizaciones que utilizan almacenamiento en la nube. Los formatos de tabla analítica permiten abstraer el almacenamiento y tratarlo como si fuera una tabla, por lo que se ingestan y luego, mediante el uso de motores de procesamiento se realiza el análisis de dichos datos, que se vuelcan en otras tablas dentro de la misma infrastructura. Esto permite no tener que distinguir entre batch y streaming, ya que los formatos de tabla analítica proveen mecanismos para detectar los cambios en las tablas y transmitirlos, generando un stream interno que puede ser aprovechado para realizar un análisis incremental.

Por lo general se utiliza un patrón de diseño de datos llamado Medallion, que define tres niveles de calidad de datos:

- Bronze: Donde se almacenan los datos que llegan en crudo
- Silver: Donde se filtran, limpian y enriquecen los datos de Bronze
- Gold: Donde se analizan los datos para generar información valiosa para el negocio

2.6.2. Componentes Principales

Stream Store Layer

- Similar a su función en la Arquitectura Kappa
- Recibe eventos y los envía al Data Lakehouse Layer

Data Lakehouse Layer

- Es una capa montada sobre almacenamiento barato como los servicios de almacenamiento de objetos en la nube
- Los formatos de tabla analítica se montan sobre este almacenamiento
- Recursos de cómputo llamados Table Services pertenecen a esta capa y dan mantenimiento al almacenamiento
- Los motores de procesamiento analizan los datos en varias etapas y las vuelvan nuevamente sobre el almacenamiento
- Generalmente se opta por una sub-arquitectura en niveles, cuyo último nivel son los datos procesados disponibles para el negocio

Catalog Layer

- Provee una capa de governanza permitiendo acceso granular a los datos, auditoría y políticas de retención
- Permite interoperar con terceros, permitiendoles descubrir tablas en base a metadatos
- Ofrece estadísticas de las tablas y herramientas para optimizar las consultas sobre ellas
- Es necesario para acceder a los datos históricos

Serving Layer

- Almacena los resultados procesados del stream por un periodo de tiempo.
- Proporciona acceso de baja latencia a los resultados del procesamiento y al histórico de datos disponibles.

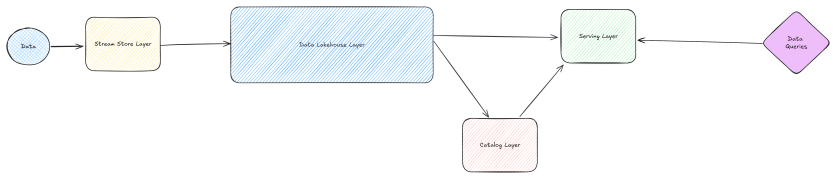

Figura 2.4: Diagrama de la Arquitectura Delta

2.6.3. Vista Lógica

2.6.4. Capacidades

- Garantiza características ACID para un sistema distribuido
- Reduce los costos de almacenamiento y procesamiento de la información
- Define una fuente de verdad única que puede ser usada por todos los procesos de análisis
- Reduce la cantidad de código que se debe mantener
- Permite agregar nuevas fuentes de datos sin necesidad de cambios en los procesos de análisis

2.6.5. Desafíos

- La latencia es un problema si se necesitan capacidades de análisis en tiempo real
- Se requieren compromisos de latencia y rendimiento por el problema de "Manejo de archivos pequeños"
- Si el Catalog Layer no se construye correctamente no es posible que la arquitectura escale

2.7. Monitoreo Remoto de Pacientes

Los Sistemas de Monitorización Remota de Pacientes (RPM, por sus siglas en inglés) constituyen un paradigma tecnológico de salud en el área de la Telemedicina que permite la adquisición, transmisión y análisis, idealmente en tiempo real, de datos fisiológicos del paciente fuera de los entornos clínicos tradicionales, mediante una red de dispositivos médicos y sensores de dispositivos inteligentes.

Este enfoque contribuye a mejores resultados para los pacientes, disminuye costos para las instituciones de salud y permite dar un acceso más generalizado a los servicios médicos. Es especialmente para el seguimiento de condiciones pre-existente, población anciana y monitoreo luego de intervenciones quirúrgicas.(V et al., 2024)

2.7.1. Monitoreo de Signos Vitales

Frecuencia Respiratoria

- Número de ciclos respiratorios (inspiración/espiración) por minuto
- Valores normales adulto: 12-20 respiraciones/min

Saturación de Oxígeno

- Porcentaje de hemoglobina unida a oxígeno en sangre arterial
- Valores normales: 95-100 %

Presión Sistólica

- Presión máxima ejercida por la sangre sobre las paredes arteriales durante la sístole
- Valores normales: 90-120 mmHg
-

Frecuencia Cardíaca

- Número de contracciones cardíacas por minuto
- Valores normales adulto: 60-100 latidos/min

Temperatura

- Medida del calor corporal
- Valores normales: 36.5-37.5°C

Escala Glasgow

- Escala neurológica que evalúa nivel de consciencia
- Evalúa la apertura ocular, la respuesta verbal y la respuesta motora en distintos rangos
- Valores normales: 15
- Es difícil de automatizar

Nivel de Conciencia

- Sistema simplificado de evaluación del estado de consciencia utilizado en valoración inicial y monitoreo
- Utiliza el sistema APVU: Alerta, Respuesta a estímulos verbales, Respuesta a estímulos dolorosos, Sin respuesta
- Valores normales: 0
- Al ifual que la escala Glasgow es dificil de automatizar

2.7.2. Identificación de Riesgo en Pacientes

En un entorno hospitalario, la monitorización de los signos vitales constituye un pilar fundamental en la evaluación del estado clínico de un paciente. Estos parámetros fisiológicos esenciales incluyen la presión arterial (PA), la saturación de oxígeno en sangre (SpO2), la temperatura corporal (T), la frecuencia cardíaca (FC) y la frecuencia respiratoria (FR), los cuales son registrados sistemáticamente en intervalos de 4 a 6 horas como parte del protocolo estándar de vigilancia para detectar posibles deterioros en la condición del paciente.

En diversos establecimientos sanitarios a nivel global, el personal médico y de enfermería implementa metodologías estandarizadas de evaluación, conocidas como sistemas de alerta temprana (SAT). Estos sistemas utilizan algoritmos validados que asignan puntuaciones específicas a las desviaciones

de los rangos normales de los signos vitales, permitiendo la activación de alertas cuando se detectan patrones que indican un deterioro clínico.

Esta práctica sistemática facilita la identificación a tiempo de pacientes en riesgo y permite la intervención terapéuticas a tiempo, contribuyendo significativamente a la reducción de eventos adversos y a la optimización de los resultados clínicos.

Existen diferentes estándares para la detección, muchos dependientes del contexto de la unidad donde se atienda al paciente. (Arora et al., 2024)

MEWS

MEWS (Modified Early Warning Score) es un sistema de puntuación fisiológica validado para la detección temprana del deterioro clínico en pacientes hospitalizados, que evalúa cinco parámetros vitales fundamentales: frecuencia respiratoria, frecuencia cardíaca, presión arterial sistólica, temperatura y nivel de consciencia.

Cada parámetro recibe una puntuación de 0 a 3 según la gravedad de su alteración, siendo 0 el valor normal y 3 el más patológico; La suma total de estos valores genera una puntuación que oscila entre 0 y 14, categorizando el riesgo del paciente en bajo (0–1), medio (2–3), alto (4–5) o crítico ($\geq$6), lo que determina la frecuencia de monitorización necesaria y las intervenciones requeridas, desde una vigilancia rutinaria cada 8-12 horas en puntuaciones bajas hasta la activación inmediata del equipo de respuesta rápida y posible traslado a UCI en puntuaciones críticas.

NEWS2

El NEWS2 (National Early Warning Score 2) es una versión mejorada y actualizada del sistema de alerta temprana, adoptado como estándar por el Servicio Nacional de Salud del Reino Unido, que evalúa siete parámetros fisiológicos: frecuencia respiratoria (3-0 puntos), saturación de oxígeno (con dos escalas distintas según el riesgo de insuficiencia respiratoria hipercápnica, 3-0 puntos), uso de oxígeno suplementario (2 puntos si requiere), temperatura (3-0 puntos), presión arterial sistólica (3-0 puntos), frecuencia cardíaca (3-0 puntos) y nivel de consciencia utilizando la escala ACVPU; la puntuación total varía de 0 a 20, estratificando el riesgo en bajo (0-4), medio (5-6), alto (7 o más, o cualquier parámetro individual con puntuación de 3) y determinando la respuesta clínica necesaria, desde monitorización estándar hasta evaluación urgente por equipo de cuidados críticos, representando una mejora significativa respecto al MEWS al incluir la saturación de oxígeno y la confusión como nuevo nivel de consciencia.

SOFA

El SOFA (Sequential Organ Failure Assessment Score) es un sistema de puntuación diseñado para evaluación diaria (cada 24 horas) de la disfunción/fallo multiorgánico en unidades de cuidados intensivos, evaluando seis sistemas orgánicos: respiratorio (mediante la relación PaO2/FiO2 evaluada con cada gasometría, 0-4 puntos), cardiovascular (mediante presión arterial media y requerimiento de vasopresores monitorizados continuamente, 0-4 puntos), hepático (mediante bilirrubina sérica medida diariamente, 0-4 puntos), coagulación (mediante recuento plaquetario diario, 0-4 puntos), renal (mediante creatinina sérica diaria o gasto urinario horario, 0-4 puntos) y neurológico (mediante la escala de Glasgow evaluada cada 4 horas o con cambios clínicos, 0-4 puntos); cada sistema recibe una puntuación de 0 (normal) a 4 (máxima disfunción), con una puntuación total que varía de 0 a 24 puntos, calculándose cada 24 horas o antes si hay deterioro clínico significativo, siendo especialmente relevante el cambio en la puntuación a lo largo del tiempo.

qSOFA

El qSOFA (quick Sequential Organ Failure Assessment) es una versión simplificada del SOFA, diseñada para la identificación rápida de pacientes con sospecha de sepsis y alto riesgo de mortalidad fuera de la UCI, evaluando únicamente tres parámetros clínicos que se pueden medir de manera inmediata a pie de cama, sin necesidad de pruebas de laboratorio: alteración del estado mental (escala de Glasgow $\leq$13 puntos, 1 punto), frecuencia respiratoria elevada ($\geq$22 respiraciones/minuto, 1 punto) y presión arterial sistólica baja ($\leq$100 mmHg, 1 punto); la puntuación total varía de 0 a 3 puntos, donde una puntuación $\geq$2 indica alto riesgo de mortalidad y la necesidad de evaluación más exhaustiva, monitorización estrecha y consideración de traslado a un nivel superior de cuidados; el qSOFA debe reevaluarse con cada valoración del paciente o ante cualquier cambio en su estado clínico, típicamente cada 1-2 horas en pacientes inestables o con sospecha de sepsis, siendo una herramienta especialmente útil en servicios de urgencias, plantas de hospitalización y entornos extrahospitalarios.

2.7.3. Desafíos

Los sensores empleados en la monitorización remota permiten un monitoreo continuo de los parámetros fisiológicos correspondientes, proporcionando una frecuencia de recolección de datos significativamente superior a los intervalos tradicionales establecidos en entornos convencional. Sin embargo, la implementación de estos sistemas de monitorización remota presenta diversos desafíos técnicos y operativos que requieren consideración.

Entre las principales se encuentran:

- Movilidad del usuario que puede afectar la calidad de la señal
- Uso correcto del dispositivo
- Variabilidad en las condiciones ambientales
- Fallos intermitentes de los sensores
- Agotamiento de la batería de los dispositivos
- Pérdida de conectividad en la transmisión de datos
- Falta de datos por dificultad de medición como en el caso de la escala Glasgow o niveles de conciencia

Esto genera la necesidad de procesar los datos adecuadamente y desarrollar técnicas para la limpieza y validación de datos. A su vez, se deben desarrollar estrategias para el manejo de datos faltantes e incompletos. (Arora et al., 2024)

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 y Rendimiento

Se implementarán mediciones a través de puntos de instrumentación estratégicos a lo largo de los componentes de la arquitectura. Estos puntos de medición incluirían timestamps en los mensajes, métricas de procesamiento en los componentes intermedios, y el tiempo de escritura/lectura en la capa final.

Las métricas serán tomadas utilizando Prometheus y mostradas en un tablero de Grafana.

Las métricas a considerar serán:

- Tiempo desde ingesta hasta visualización
- Tiempo de ingesta
- Tiempo de procesamiento
- Tasa de mensajes procesados por segundo
- Volúmen de datos procesados por segundo

3.1.2. Escalabilidad

Se realizarán pruebas de carga en los sistemas. Las métricas serán tomadas utilizando Prometheus y mostradas en un tablero de Grafana.

Las métricas a considerar serán:

- Ratio de escalado, que relaciona recursos añadidos vs mejora en el rendimiento
- Eficiencia de recursos al escalar horizontalmente

3.1.3. Consistencia de Datos

Se implementarán técnicas para enviar mensajes de forma maliciosa, mediante K6 de Grafana, que puedan generar inconsistencias en el análisis. Las métricas serán tomadas utilizando Prometheus y mostradas en un tablero de Grafana.

Las métricas a considerar serán:

- Tiempo que le toma al sistema manejar el estado inconsistente
- Puntaje qualitativo indicando qué tan fácil resulta la implementación de las medidas necesarias

3.1.4. Manejo de Datos Históricos

Se medirán especificamente la efectividad con la que el sistema permite consultar datos históricos y reprocesar toda la historia. Las métricas serán tomadas utilizando Prometheus y mostradas en un tablero de Grafana.

- Tiempo de consulta al primer dato ingresado de un paciente
- Tiempo de reprocesamiento de la historia completa
- Uso de recursos en reprocesamiento de la historia completa
- Puntaje qualitativo indicando qué tan fácil resulta la implementación necesaria

3.1.5. Costos Operativos

La implementación del sistema en contenedores, permitirá un monitoreo del uso de los recursos del sistema. Además, se plantearán cálculos, utilizando las calculadoras de costo que proveen los sistemas de nube Azure y AWS, que permitan dimensionar el costo operativo de estos sistemas. Las métricas serán tomadas utilizando Prometheus y mostradas en un tablero de Grafana.

Las métricas a considerar serán:

- Uso de memoria por componente
- Uso de memoria total
- Uso de cpu por componente
- Uso de cpu total
- Operaciones de escritura/lectura en disco
- Uso de red entre componentes
- Consumo de recursos por reprocesamiento del histórico del sistema

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.

3.2. Stack de Tecnologías a Utilizar

Para que la implementación de las instancias de arquitecturas de referencia sean comparables es necesario mantener criterios similares para no favorecer a una de ellas. Estas instancias serán definidas mediante contenedores y desplegadas en Kubernetes de forma local utilizando la herramienta Minikube; lo que permitirá definir un ambiente Cloud-Native para la comparación de las mismas.

Se realizó un análisis exhaustivo de las tecnologías y se decidió por:

- Apache Pulsar como punto de ingestión de datos y ruteo de mensajes
- Apache Flink como motor de procesamiento distribuido
- MinIO como almacenamiento de objetos
- Apache Pinot como base de datos de análisis
- Apache Superset como tablero que consume de los sistemas implementados
- Apache Iceberg como formato de tabla analítica
- Apache Airflow como gestor de procesos de mantenimiento
- Prometheus para monitoreo de métricas de los sistemas
- Grafana para creación de tableros de métricas comparativas
- K6 para pruebas de carga y consistencia
- ChaosMonkey para pruebas de tolerancia a fallos

3.3. Conjunto de Datos

3.3.1. Proceso de Análisis

El objetivo es que se pueda utilizar el sistema como plataforma para poder darle mejor seguimiento a los pacientes ambulatorios, así como también ingresados a los centros de salud. Por esto, se propone tomar como referencia la implementación de los sistemas de alerta temprana para diversas áreas:

- MEWS como sistema de alerta de pacientes fuera de una instituición de salud
- NEWS2 como sistema de alerta de pacientes ingresados en una institución de salud
- qSOFA como sistema de alerta de pacientes en ingresados en una Unidad de Cuidados Intensivos

Se modularizará la lógica tanto como sea posible, de modo de reducir problemas de medición teniendo dos implementaciones diferentes.

3.3.2. Conjunto de Datos

El conjunto de datos utilizado para evaluar el sistema será de datos sintéticos. Se utilizará la plataforma Node-RED para la creación de una red de nodos de dispositivos virtuales que enviarán datos a los distintos sistemas. Se buscará generar un fuerte volúmen de datos, del orden de los 10 GB para poder tener una buena evaluación de los criterios de evaluación de los sistemas.

Existen dos razones principales para el uso de datos sintéticos:

- Los conjuntos de datos de sensores disponibles son heterogéneos y muy pequeños; lo que de todos modos implicaría generar datos sintéticos.
- No es necesario contar con datos que muestren tendencias reales, sino que se comporten como lo harían en realidad para demostrar los atributos de calidad.

Para esto, se definirá un flujo de datos para cada sensor que será interconectado pero definiendo comportamientos especificos que tengan que ver con la naturaleza de su medición. Los signos vitales tomados para esto son:

- Frecuencia respiratoria

- Presión sistólica
- Frecuencia cardíaca
- Temperatura
- Saturación de oxígeno
- Nivel de conciencia en escala AVPU
- Nivel de conciencia en escala Glasgow

Capítulo 4

Desarrollo

Contenido del capítulo...

4.1. Implementación Arquitectura Kappa

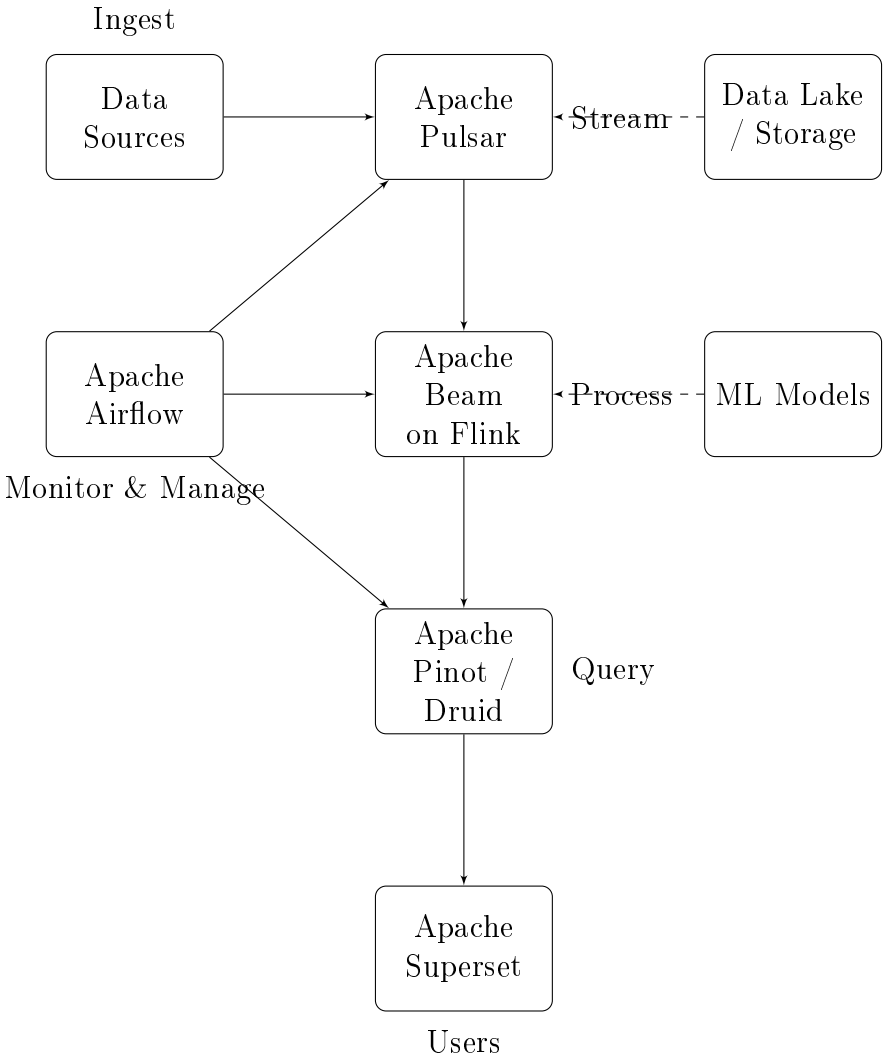

Figura 4.1: Pulsar-Centric Kappa Architecture

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).

Marz, N., & Warren, J. (2015). *Big Data: Principles and best practices of scalable realtime data systems* (1st). Manning Publications Co.

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

Kleppmann, M. (2018). *Designing Data-Intensive Applications* (1.ª ed., Vol. 1). O'Reilly Media Inc.

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

Muñoz-Escoí, F. D., Juan-Marín, R. d., García-Escrivá, J.-R., Mendívil, J. R. G. d., & Bernabéu-Aubán, J. M. (2019). CAP Theorem: Revision of Its Related Consistency Models. *Computer Journal*, *62*(6), 943-960. https://research.ebsco.com/linkprocessor/plink?id=3508200f-f883-34a0-a32c-3dea6a6209e0

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

Tanenbaum, A. S., & van Steen, M. (2020). *Distributed Systems: Principles and Paradigms* (Third edition). Pearson Education, Inc.

Arora, T., Balasubramanian, V., Stranieri, A., & Menon, V. G. (2024). Modified Early Warning Score (MEWS) Visualization and Pattern Matching Imputation in Remote Patient Monitoring. *IEEE Access*, *12*, 74784-74794. https://doi.org/10.1109/ACCESS.2024.3396274

V, D. G., Joshi, D., C, S. k., G, M. R., D, S., & Habeeb, M. (2024). Impact of IoT on Remote Patient Monitoring and Advancements in Telemedicine. *2024 Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI)*, 319-326. https://doi.org/10.1109/ICoICI62503.2024.10696558