UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

POSTGRADO EN INFORMÁTICA

TESIS MAGISTER SCIENTIARUM

PROGRAMA DE MAESTRÍA EN ALTA GERENCIA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN E INNOVACIÓN PARA EL DESARROLLO (MAG-TIC) VERSIÓN 2 GESTIÓN 2015-2016

MODELO DE COMPUTACIÓN EN STREAMING PARA REDUCIR LA COMPLEJIDAD COMPUTACIONAL DEL SOFTWARE USADO EN EL SECTOR RETAIL

POR: Lic. Víctor Alfonso Ramos Huarachi

TUTOR: M.Sc. Marcelo Pinto Macedo

La Paz - Bolivia
2020

DEDICATORIA

Este logro es para ti mamita, Victoria Huarachi, tú que te has esforzado para que yo pueda crecer profesionalmente y personalmente, mis logros son tuyos y ahora que estás en la presencia de Dios quiero decirte no me apartaré de tus enseñanzas ni de los valores que me diste, volaré muy alto como alguna vez me dijiste.

¡Gracias por todo mamá!

AGRADECIMIENTOS

Gracias a ti Lizet por estar a mi lado y apoyarme, quizá no hubiera completado esto si no fuera por tu empuje y comprensión. Gracias a mi familia que me dan fuerzas para seguir adelante. Y finalmente, gracias a mi tutor, Marcelo, por creer en mí, en mi tema y por ayudarme a evolucionar esta tesis a su estado de excelencia final.

Atte. Victor Alfonso Ramos Huarachi

ÍNDICE DE CONTENIDO

RESUMEN 8

INTRODUCCIÓN 10

CAPÍTULO I 11

ASPECTOS GENERALES 11

1.1 ANTECEDENTES 11
1.2 PLANTEAMIENTO DEL PROBLEMA 12
1.3 OBJETIVO GENERAL Y ESPECÍFICO DE LA INVESTIGACIÓN 15
1.3.1 OBJETIVO GENERAL 15
1.3.2 OBJETIVOS ESPECÍFICOS 16
1.4 HIPÓTESIS Y OPERACIONALIZACIÓN DE VARIABLES DE INVESTIGACIÓN 16
1.4.1 VARIABLE DEPENDIENTE 16
1.4.2 VARIABLE INDEPENDIENTE 16
1.4.3 OPERACIONALIZACIÓN DE VARIABLES 16

CAPÍTULO II
MARCO TEÓRICO 18

2.1 MARCO CONCEPTUAL 18
2.1.1 TEORÍA DE LA COMPLEJIDAD 18
2.1.2 TEORÍA DE LA COMPLEJIDAD COMPUTACIONAL 19
2.1.3 INTERNET DE LAS COSAS 21
2.1.4 TÉCNICAS DE PROCESAMIENTO PARA BIG DATA 23
2.1.5 COMPUTACIÓN EN STREAMING 24
2.1.6 FRAMEWORKS DE PROCESAMIENTO EN STREAMING 26
2.1.6.1 APACHE STORM 26
2.1.6.2 APACHE SPARK 27
2.1.6.3 APACHE FLINK 27
2.1.6.4 APACHE KAFKA STREAMS 28
2.1.6.5 COMPARATIVA ENTRE FRAMEWORKS 29
2.1.6.6 SELECCIÓN DE LA TECNOLOGÍA 30
2.1.7 ARQUITECTURA DE KAFKA STREAMS 31
2.1.7.1 TÓPICOS 31
2.1.7.2 PRODUCTORES Y CONSUMIDORES 33
2.1.7.3 TOPOLOGÍA DE LOS PROCESADORES DE DATOS 34
2.1.7.5 PARTICIONAMIENTO 36

2.1.7.6 MODELO DE ESCALABILIDAD 37
2.1.7.7 MODELO DE TOLERANCIA A FALLOS 39
2.1.7.8 APLICACIÓN EN EL SECTOR RETAIL 42
2.1.8 CARACTERÍSTICAS DEL PROCESADOR DE STREAMING 43
2.1.8.1 ESCALABILIDAD 44
2.1.8.2 RENDIMIENTO 45
2.1.8.3 DISPONIBILIDAD 45
2.1.8.4 TOLERANCIA A FALLOS 46
2.1.8.5 SEGURIDAD 47
2.2 MARCO REFERENCIAL 48
2.2.1 INTERNET DE LAS COSAS Y SU IMPACTO EN RETAIL 48
2.2.2 ARQUITECTURA DE SOFTWARE DEL SECTOR RETAIL 50

CAPÍTULO III
METODOLOGÍA DE LA INVESTIGACIÓN 55
3.1 MÉTODO DE INVESTIGACIÓN 55
3.1.1 FASES METODOLÓGICAS 55
3.2 TIPO DE INVESTIGACIÓN 56
3.2.1 DISEÑO DE INVESTIGACIÓN 56
3.3 UNIVERSO O POBLACIÓN DE ESTUDIO 56
3.3.1 DETERMINACIÓN Y ELECCIÓN DE LA MUESTRA 57
3.4 SUJETOS VINCULADOS A LA POBLACIÓN 58
3.5 FUENTES Y DISEÑO DE LOS INSTRUMENTOS DE RELEVAMIENTO DE INFORMACIÓN 58
3.5.1 FUENTES DE LA INVESTIGACIÓN 58
3.5.2 DISEÑO DE LOS INSTRUMENTOS DE RELEVAMIENTO DE INFORMACIÓN 58

CAPÍTULO IV
MARCO PRÁCTICO 62
4.1 METODOLOGÍA DE DESARROLLO DE LA SOLUCIÓN 62
4.1.1 VISTA DE DESARROLLO 62
4.1.1.1 DEFINICIÓN DE TÓPICOS 62
4.1.1.2 COMPONENTES DE SOFTWARE 64
4.1.1.3 TOPOLOGÍA DEL PROCESADOR DE REGLAS 66
4.1.2 VISTA DE PROCESO 68
4.1.3 VISTA FÍSICA 70
4.1.3.1 MODELO DE DESPLIEGUE 70

4.1.3.2 MODELO DE REPLICACIÓN EN MÚLTIPLES CENTROS DE DATOS 73
4.2 VALIDACIÓN DEL MODELO 74
4.2.1 PRUEBAS DE RENDIMIENTO 74
4.2.2 PRUEBAS DE ESCALABILIDAD 78
4.2.3 PRUEBAS CARGA 80
4.2.4 PRUEBAS DE TOLERANCIA A FALLOS 83
4.2.5 COSTOS 84

CAPÍTULO V
MARCO DE RESULTADOS 87
5.1 ESTADO DE LOS OBJETIVOS 87
5.2. ESTADO DE LA HIPÓTESIS 90
5.3 CONCLUSIONES 91
5.4 RECOMENDACIONES 91

BIBLIOGRAFÍA 102

INDICE DE FIGURAS

Figura 1.1: Cadena de Suministro en el Sector Retail 14
Figura 1.2: Software de diseño multiregión 15
Figura 2.1: Modelo de consumo para IoT, Retail 24
Figura 2.2: Procesamiento tradicional vs procesamiento en streaming 27
Figura 2.3: Anatomía de un tópico 35
Figura 2.4: Modelo de consumo de datos de un tópico 35
Figure 2.5: Asignación de particiones a consumidores 37
Figura 2.6: Topología de procesamiento en streaming 38
Figura 2.7: Modelo de asignación de particiones de un tópico a tareas 40
Figura 2.8: Agrupación de tareas en hilos de procesamiento 41
Figura 2.9: Modelo de escalabilidad 42
Figura 2.10: Replicación de una partición en el clúster 43
Figura 2.11: Reasignación de tareas en el modelo de tolerancia a fallos 44
Figura 2.12: Bus de eventos del software basado en streaming 46
Figura 2.13: Composición de un mensaje en un stream de datos. 47
Figura 2.14: Interacción de procesadores de streaming con el clúster de Kafka 47
Figura 2.15: Beneficios de IoT en el sector retail 54
Figura 2.16: Capas Básicas del Modelo IoT 55
Figura 2.17: Componentes básicos del software para el sector de retail 58
Figura 2.18: Despliegue de software en el sector retail 59
Figura 4.1 Componentes Propuestos por el Modelo para el Sector Retail 70
Figura 4.2: Topología de procesamiento de datos 73
Figura 4.3: Diagrama de componentes de software para el sector retail aplicando computación en streaming 75
Figura 4.4: Modelo de Despliegue de Componentes 78
Figura 4.5: Diseño del replicador de datos 79
Figura 4.6: Infraestructura de centros de datos distribuidos 80
Figura 4.7: Gráficos de rate utilizando Kafka Exporter, Prometheus y Graphana 81
Figura 4.8: Linealidad del escalamiento de una aplicación en streaming 86
Figura 4.9: Uso de CPU de la aplicación en streaming 87
Figura 4.10. Uso de Memoria RAM de la aplicación en streaming 87
Figura 4.11 Comparativa de uso de recursos entre una aplicación streaming vs aplicación tradicional 88
Figura 4.12: Balanceo de carga en prueba de tolerancia a fallos 90
Figura 5.1 Resumen comparativa de rendimiento entre software en streaming y software tradicional 94
Figura 5.2 Comparativa de uso de memoria RAM entre software en streaming y software tradicional 95
Figura 5.3 Comparativa de uso de CPU entre software streaming y software tradicional 95

INDICE DE TABLAS

Tabla 1.1 Operacionalización de variables 18
Tabla 2.1: Comparativa de rendimiento entre frameworks de streaming 32
Tabla 2.2: Estadísticas de latencia, promedios, mínimos y máximos en segundos 32
Tabla 3.1: Casos de prueba planteados para la simulación 65
Tabla 4.1. Definición de tópicos 68
Tabla 4.2 Responsabilidades de los Componentes de Software Propuestos 70
Tabla 4.3 Responsabilidades de los nodos de la topología del procesador de streaming 73
Tabla 4.4: Descripción de nodos para el modelo de despliegue 77
Tabla 4.5: Asignación de recursos para el modelo de despliegue 77
Tabla 4.6: Resultados de rendimiento entre aplicación en streaming y tradicional 83
Tabla: 4.7: Porcentaje de mejora entre aplicación streaming vs tradicional 84
Tabla: 4.8: Prueba de escalabilidad de una aplicación en streaming 85
Tabla 4.9 Pruebas de carga de una aplicación streaming y tradicional 88
Tabla 4.10 Costo de despliegue del software basado en streaming 91
Tabla 4.11 Costo de despliegue del software tradicional 91

INDICE DE FÓRMULAS

Fórmula 1: Cálculo de rate de una aplicación en streaming 81
Formula 2: Cálculo del lag en una aplicación en streaming 82
Fórmula 3: Porcentaje de mejora de rendimiento 83
Fórmula 4. Cálculo de reducción de coste de procesamiento 91

RESUMEN

La complejidad computacional de un software está determinada por el uso de recursos que utiliza para llevar a cabo una determinada tarea, es decir cuánto CPU y RAM consume. En situaciones donde la cantidad de datos que se tienen que procesar es demasiada y continua, la computación en streaming es ideal para reducir la complejidad computacional que supone su procesamiento.

Este es el caso del software del sector retail, un sector compuesto por centros comerciales, tiendas departamentales, supermercados y almacenes, que tras el advenimiento de nuevos paradigmas como el Internet de las Cosas y Big Data, ha requerido adaptarse para competir con las demás empresas en ofrecer una mejor experiencia al cliente y en mejorar la eficiencia de sus operaciones, haciendo que el software tenga que procesar grandes cantidades de información en tiempo real.

El software utilizado por estas empresas debe adaptarse a los nuevos paradigmas y una aplicación de procesamiento en streaming no solo permite procesar grandes cantidades de datos, sino que lo hace a gran velocidad, en tiempo real, son aplicaciones escalables, altamente disponibles y tolerantes a fallos.

ABSTRACT

Software computational complexity is determined by the resources needed to execute a task, this means how much CPU and RAM it consumes. In situations where the amount of data to be processed is too much and continuous, stream computing is excellent for reducing the computational complexity involved in its processing.

This is the case of the software developed for retailers, constituted by shopping centers, department stores, supermarkets and warehouses, which after the advent of new paradigms such as the Internet of Things and Big Data, has required to be adapted to compete with other companies in offering a better customer experience and improving the efficiency of its operations, making the software have to process large amounts of information in real time.

The software used by these companies must be adapted to the new paradigms, and in this situation a streaming application not only allows processing large amounts of data, but it does so with a high throughput, in real time, they are scalable, highly available and fault-tolerant.

INTRODUCCIÓN

La información tiene mucho más valor en función al tiempo en el que se recibe, es por eso que muchas aplicaciones buscan procesar los datos en tiempo real, por ejemplo, una red social es capaz de detectar tendencias en cuestión de minutos; un motor de búsqueda intentará predecir qué sitios web contienen aquello que el usuario desea encontrar lo más rápido posible; y un operador de servicios busca monitorear logs para detectar fallas en cuestión de segundos.

El software diseñado para llevar a cabo estas tareas es una realidad gracias a la gran cantidad de datos disponibles, pero procesar tanta cantidad de datos se ha vuelto un reto de ingeniería en el área de software. La cantidad de datos disponibles es demasiada, específicamente en el sector de retail la cantidad de datos ronda los miles de millones para una sola empresa haciendo que el software diseñado para procesarlo se vuelva muy complejo, ya que se requieren cada vez más equipos de cómputo, se necesita más memoria RAM y CPU, se requiere hacer segmentación de los datos y otras técnicas que permitan un procesamiento confiable y en tiempo real.

Pero, así como crece la necesidad de procesar grandes cantidades de información también se crean nuevas técnicas y tecnologías que permiten el procesamiento eficiente de la información, tecnologías que se centran en reducir la complejidad computacional del software y permitiendo la construcción de sistemas escalables y de alto rendimiento.

CAPÍTULO I

ASPECTOS GENERALES

1.1 ANTECEDENTES

El sector de retail compuesto principalmente por supermercados, centros comerciales y grandes almacenes, la rápida toma de decisiones es imperante para superar a la competencia en brindar una mejor experiencia e innovadores productos y servicios. Pero, ante la gran cantidad de información existente, es necesario disponer de software capaz de procesar tal cantidad de información de manera eficiente, reportando eventos a tiempo para actuar ante situaciones cotidianas o prediciendo el comportamiento del mercado.

Pero diseñar software capaz de cubrir esa necesidad es altamente complejo, computacionalmente hablando es un reto (Teruel, K. P., Yelandi, M., Vásquez, L., Karel, I., Cedeño, F., Jimenez, S. V., & Mustelier, I. D., 2012), debido a que la escala necesaria para los softwares más grandes, como por ejemplo procesamiento de logs en tiempo real o analítica de datos, requieren de cientos de nodos de procesamiento.

Al ser un mercado altamente rentable, empresas como Amazon, Microsoft y Google proveen soluciones para el procesamiento de datos en tiempo real en la nube, pero estas no encajan perfectamente en las necesidades de las empresas del sector de retail puesto que estas son de propósito general.

El problema ha sido tratado por investigadores de todo el mundo en la búsqueda de algoritmos de alta disponibilidad y de procesamiento distribuido para el mundo del internet de las cosas, las soluciones generadas han sido ampliamente utilizadas en el mundo del análisis de datos, la inteligencia artificial y aplicaciones para “big data” (C. Hochreiner, M. Vogler, S. Schulte, S. Dustdar, 2016)

1.2 PLANTEAMIENTO DEL PROBLEMA

La cadena de suministro, desde la fábrica, transporte, almacenes, hasta los estantes en las tiendas, pasando por sensores digitales, balizas, quioscos digitales y estantería inteligente, los datos generados en las empresas del sector de retail es tanta que es muy complicado para los empleados de una tienda satisfacer la alta demanda para incluso el servicio al cliente más básico como ser: cotizaciones, disponibilidad en inventario, localización de productos o validación de autenticidad de un producto. Y de la misma forma, realizar tareas recurrentes es muy complicado, como por ejemplo realizar el inventario de una tienda al iniciar y al finalizar el día, control antirrobos, seguimiento de productos desde la fábrica, pasando por los empaquetados, envíos, recepción y venta.

Figura 1.1: Cadena de Suministro en el Sector Retail
Fuente Elaboración Propia

Los sistemas desarrollados para el "Internet de las Cosas" (IoT), son una de las mayores iniciativas que han revolucionado el sector de retail, pero dichos sistemas son

altamente costosos, debido a que se deben iniciar varias instancias de la misma para ser capaces de procesar grandes cantidades de datos, lo más rápidamente posible y entregar la información de manera oportuna.

Como dato adicional, un solo retailer puede tener miles de millones de ítems, los cuales puede que no estén siendo procesados al mismo tiempo, pero al existir un movimiento o inventario en cualquier punto de la cadena de suministros genera la necesidad de procesamiento por el orden de miles por segundo.

La solución más simple por la que optan muchas empresas es la separación de datos y servidores de procesamiento por tienda y por ubicación en el mundo, esto implica un crecimiento en el costo del hardware y en tiempo para realizar tareas de análisis de datos y de inteligencia de negocios de manera global. La solución más compleja es tener un software multi región, el cual sea una sola plataforma a los ojos del usuario final como indica la figura 2. Esta solución conlleva retos de ingeniería que las empresas tendrían que asumir y minimizar el esfuerzo es deseable.

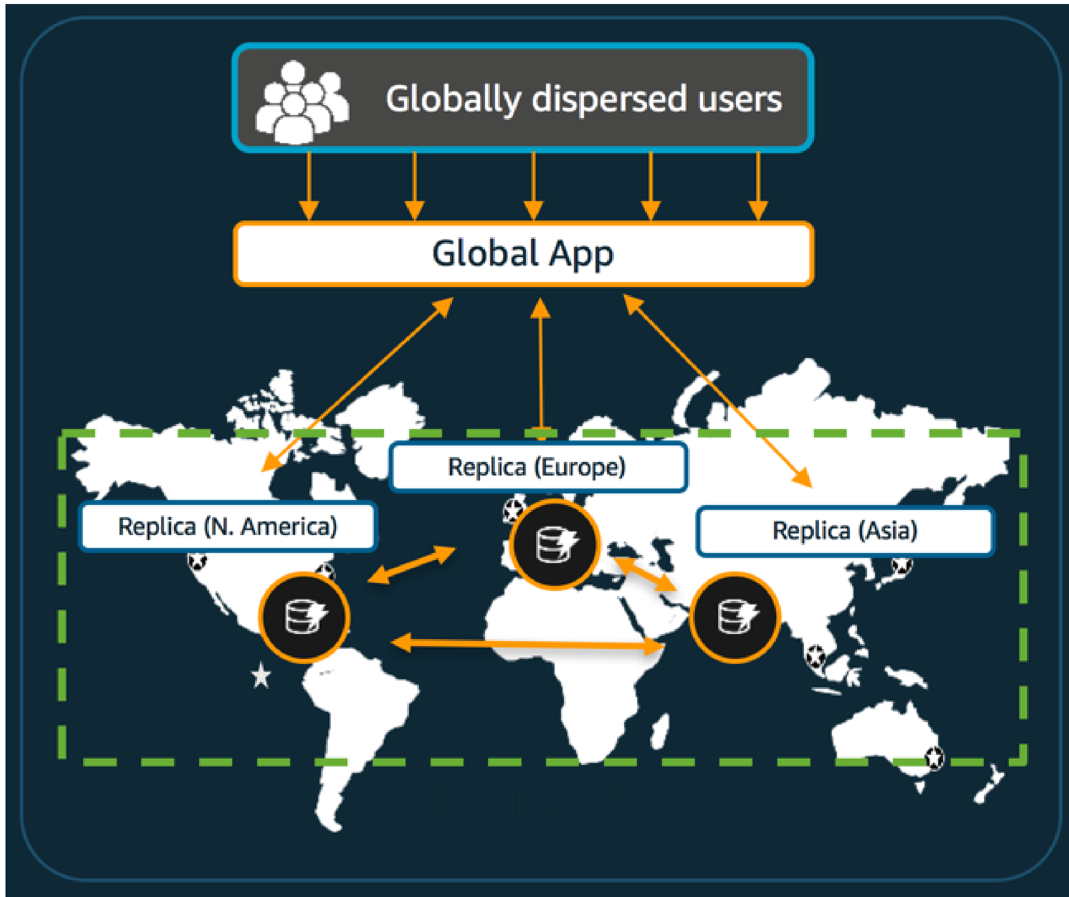


Figura 1.2: Software de diseño multiregión
Fuente: Elaboración propia

Pensar en grandes cantidades de datos lleva a pensar también en big data, este concepto, explicado más extensamente en el marco teórico de este perfil de tesis, no puede ser procesado por bases de datos ni software tradicionales, puesto que realizarlo de esa manera resulta en una complejidad computacional muy elevada, lo cual implica costos también elevados. La razón es simple, las bases de datos tradicionales son relacionales lo que permiten mantener la consistencia de los datos, pero eso tiene un costo en tiempo antes de escribir en el disco, con este tipo de base de datos, el software procesa o prepara los datos posteriores a su guardado, o sea cuando el usuario realiza una consulta a través de, por ejemplo, un servicio

web. A razón de ello nacieron las bases de datos NoSQL que son más veloces en la escritura y consulta de datos, pero requieren un procesamiento anterior a su guardado implicando un cambio de paradigma en el software de procesamiento.

Específicamente en el sector de retail se necesita que los datos sean procesados en tiempo real, por ejemplo, se muestran alertas cuando un producto está siendo robado, o también una tienda no debería cerrar por inventario sino este debería ser recurrente para tener un control más efectivo de lo que ocurre en ella para tomar más rápidas y mejores decisiones.

La alta cantidad de información requiere de capacidad de procesamiento masiva y en paralelo, llegar a obtener esta capacidad de cómputo requiere de varios nodos de procesamiento lo que conlleva a diferentes problemas como ser (Oracle Corporation, 2008):

- Fallos en algún nodo que llevaría a una pérdida de datos.
- Struggling o bajo rendimiento de algunos nodos.
- Alta latencia en el procesamiento.
- Coste elevado.

Finalmente, el problema de investigación se puede realizar de la siguiente manera:

¿Se puede reducir la complejidad computacional que supone el procesamiento de grandes cantidades de datos generados en una empresa del sector retail que permita tener un software eficiente y de gran velocidad?

1.3 OBJETIVO GENERAL Y ESPECÍFICO DE LA INVESTIGACIÓN

1.3.1 OBJETIVO GENERAL

Desarrollar un modelo basado en la computación en streaming para reducir la complejidad computacional de los sistemas de procesamiento de datos del sector de retail que permita procesar grandes cantidades de datos de manera eficiente y con gran velocidad.

1.3.2 OBJETIVOS ESPECÍFICOS

1. Diseñar una arquitectura de software que permita el procesamiento en streaming, basado en los componentes y aplicaciones actuales del sector retail.
2. Mejorar la eficiencia en el procesamiento de datos en el sector retail, incrementando la tasa de procesamiento por segundo a un menor coste de recursos computacionales.
3. Reducir el costo económico que supone el despliegue de infraestructura generados por el software del sector retail.

1.4 HIPÓTESIS Y OPERACIONALIZACIÓN DE VARIABLES DE INVESTIGACIÓN

"Un nuevo modelo basado en la computación en streaming reduce la complejidad computacional del software usado por las empresas del sector de retail"

1.4.1 VARIABLE DEPENDIENTE

Complejidad computacional de los sistemas de procesamiento de datos de las empresas del sector de retail.

1.4.2 VARIABLE INDEPENDIENTE

Modelo basado en la computación en streaming aplicada al software de empresas del sector de retail.

1.4.3 OPERACIONALIZACIÓN DE VARIABLES

Variable	Definición Conceptual	Componen-tes	Indicadores
Dependiente Complejidad computacional de los sistemas de procesamiento de datos de las empresas del sector de retail.	La complejidad computacional es una rama de la teoría de la computación que tiene por objetivo medir cuántos recursos en tiempo y espacio son necesarios para resolver un problema (Sipser, M. 2006)	Complejidad en tiempo	Rate de procesamiento por segundo para procesar una cantidad determinada de transacciones.
		Complejidad en espacio	Memoria RAM necesaria para procesar una cantidad determinada de datos Costos de despliegue y procesamiento de datos

Independiente Modelo basado en la computación en streaming aplicada al software de empresas del sector de retail.	Se refiere a las arquitecturas de software que potencialmente pueden resolver los problemas de procesamiento de flujo de datos (streaming) de alto volumen con baja latencia (Stonebraker, M., Çetintemel, U. & Zdonik, 2005). La aplicación de este tipo de arquitectura en el sector de retail es promovida por la necesidad de estas empresas en innovar para tomar decisiones rápidas y acertadas.	Frameworks de procesamiento en streaming.	Métricas de rendimiento de frameworks de procesamiento en streaming
			Métricas de latencia de frameworks de procesamiento en streaming
		Procesador de streaming	Tópicos creados para entrada y salida de una aplicación de procesamiento en streaming con particiones y réplicas definidas.
			Aplicaciones procesando en streaming con una topología definida
			Modelo de escalabilidad
			Modelo de tolerancia a fallos

Tabla 1.1 Operacionalización de variables
Fuente Elaboración Propia

CAPÍTULO II

MARCO TEÓRICO

2.1 MARCO CONCEPTUAL

2.1.1 TEORÍA DE LA COMPLEJIDAD

La teoría de la complejidad es una rama importante de la teoría de la computación cuyo objetivo es medir cuántos recursos son necesarios para resolver un problema.

La teoría de la complejidad estudia cómo crece el coste computacional en espacio y tiempo de resolver un determinado problema en relación al crecimiento de dicho problema (Gao, Q., & Xu, X., 2014). Por ejemplo, ordenar números puede suponer una tarea fácil para, incluso, una pequeña computadora. Ahora comparemos el anterior problema con el problema de una agenda, agendar todas las clases que una persona toma en la universidad evitando solapamiento en el horario entre clases ni con las actividades extracurriculares de la persona. El problema de la agenda parece mucho más complicado que ordenar números. Qué pasa si se debe agendar miles de clases para miles de personas; resolver de la mejor manera el problema podría tomar décadas, incluso para las supercomputadoras.

¿Qué es lo que hace a unos problemas computacionalmente difíciles y a otros fáciles? Esta es la pregunta más importante de la teoría de la complejidad. Aunque esta pregunta no tiene respuesta, la teoría de la complejidad se encarga de clasificar estos problemas de acuerdo a la complejidad computacional que implica resolverlas.

2.1.2 TEORÍA DE LA COMPLEJIDAD COMPUTACIONAL

Aunque se conozca la solución a un problema y computacionalmente es factible resolverlo, aun así, puede que no sea posible de resolver en la práctica.

Para medir si un problema del cual se conoce una solución es posible de ser resuelto se encarga la teoría de la complejidad computacional. Estas mediciones se realizan tomando en cuenta las dimensiones de espacio y tiempo.

Sipser define la complejidad de espacio de la siguiente manera:

Sea M una máquina de Turing determinista que lee todas sus entradas. La complejidad de espacio de M es la función f : N → N, donde f(n) es el máximo número de celdas de la cinta que M escanea para cada entrada de longitud n. Si la complejidad de espacio de M es f(n), podríamos decir que M corre en el espacio f(n).

Si M es una máquina de Turing no determinista donde todas las ramas se detienen en todas las entradas, se define el espacio de complejidad f(n) como el máximo número de celdas de la cinta que M escanea para resolver cada rama para cualquier longitud de M (Sipser, M, 2013).

En otras palabras, la complejidad de espacio se refiere a que para problemas deterministas el espacio requerido para resolverlo es lineal, mientras que para problemas no deterministas el espacio requerido podría ser polinomial o exponencial. Por ejemplo, supongamos que para ordenar 10 números se requieren 10 MB de memoria, para ordenar 100 número se requieren 100 MB de memoria, este problema sería clasificado como de complejidad de espacio lineal.

Si en el anterior ejemplo, en lugar de requerir 100 MB de memoria para ordenar 100 números, se requieren 1000 MB estaríamos hablando de un problema de complejidad computacional exponencial.

De la misma forma Sipser define la complejidad de tiempo de la siguiente manera:

Sea M una máquina de Turing determinista que se detiene en todas sus entradas. El tiempo de proceso o complejidad de tiempo de M es la función f: N → N, donde f(n) es el máximo número de pasos que usa M para cualquier entrada de longitud n. Si f(n) es el tiempo de proceso de M, entonces decimos que M corre en un tiempo f(n) (Sipser, M, 2013).

De la misma manera, la complejidad de tiempo se refiere a cuánto tiempo se requiere para procesar un problema utilizando un determinado algoritmo. Pueden existir problemas que pueden ser resueltos en un tiempo razonable y otros no.

Por ejemplo, algunos problemas como el ordenamiento de números podrían tener un algoritmo deficiente y complejidad de espacio polinomial pero las computadoras actuales pueden resolverlo en un tiempo razonable. Pero, por otro lado, si afrontamos un problema de desencriptación estaríamos frente a un problema no asumible por ser complejo en términos del tiempo necesario para resolverlo.

2.1.3 INTERNET DE LAS COSAS

Internet de las cosas o IoT, es una terminología genérica que describe la conectividad máquina a máquina, en donde los sensores o dispositivos de hardware comunican entre sí, información de las condiciones y ubicación de cada uno de ellos. Por sí mismo no es algo inteligente, pero cuando técnicas y tecnologías como el machine-learning y la inteligencia artificial entran en juego, toman los datos generados y los vuelven en algo con significado y con mucho valor para los negocios.

De acuerdo a un estudio realizado por Beecham Research e Intel, “IoT está cambiando de manera fundamental el cómo operan los grandes negocios del sector de retail” (Doug D., 2016). Aquellas empresas que abrazan estas tecnologías dejaron de ser empresas horizontales y se están comenzando a volver más verticales, esto quiere decir, que conocen más de cada aspecto del negocio, desde las preferencias de los clientes, pasando por los productos y llegando completamente a la cadena de suministros, esto le permite no lo solo retener a sus clientes sino captar nuevos. Algunos modelos, como el que se muestra en la figura 1, ejemplifica cómo IoT es enriquecido a tal punto de apoyar e influir en las decisiones tomadas como parte del marketing estratégico de la empresa.

Entonces, IoT es simple pero poderoso si se sabe procesar la información de manera eficiente. Tal cantidad de datos generado en toda la cadena de suministros, ha dado lugar a la creación de software altamente complejo, donde se requiere algoritmos eficientes, escalables y tolerantes a fallos; en este punto donde son de gran ayuda los sistemas basados en la computación en streaming.

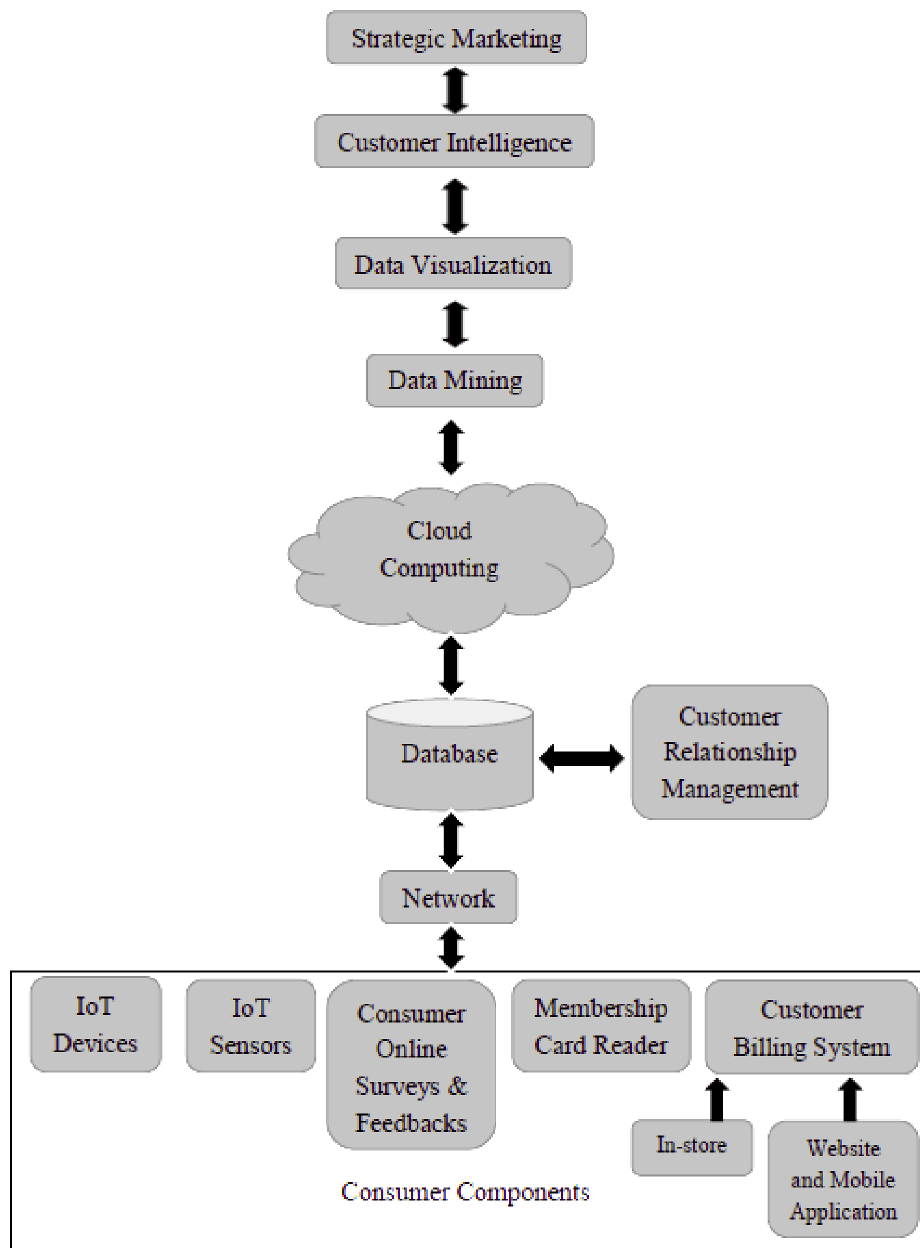


Figura 2.1: Modelo de consumo para IoT, Retail

Fuente: Jayaram, A., (2017)

2.1.4 TÉCNICAS DE PROCESAMIENTO PARA BIG DATA

Al hablar de grandes cantidades de información, una de las primeras cosas que se viene a la mente es big data. Big data se puede definir como una colección de datos tan grande que las bases de datos convencionales no pueden procesarlas en el tiempo deseado (Casado, R., & Younas, M., 2014), similar situación ocurre con el software tradicional, puesto que se tiene que tratar con grandes volúmenes de datos, velocidad y variedad, lo que le aumenta cierto grado de complejidad

Las principales estrategias de procesamiento de big data son las que se mencionan a continuación:

1. Procesamiento en paralelo, aplicado en todo tipo de aplicaciones, inclusive las tradicionales, aprovechando las características de los lenguajes de programación se pueden crear varios hilos de ejecución que permitan aprovechar al máximo los núcleos del procesador. Su principal desventaja es la facilidad de perder el control del código y de tratar con problemas de concurrencia.
2. Procesamiento por lotes, procesar un gran conjunto de datos es mejor que procesar los datos de uno en uno, puesto que se reduce la latencia producida por la red y/o escritura en disco. La principal desventaja, aparte de las configuraciones necesarias en cliente y servidor, es que se necesita la memoria suficiente para almacenar todos los datos mientras son procesados, y una vez liberada la memoria el sistema queda en un estado de reposo “desperdiciando” recursos.
3. El procesamiento en streaming, muy útiles en escenarios donde se requiere procesar no solamente muchos datos sino también en tiempo real; este tipo de procesamiento es muy veloz y eficiente, puesto que procesa pequeños lotes en intervalos constantes de

tiempo lo cual le permite ahorrar memoria, la latencia es menor por la misma razón. Otra ventaja que tiene es que, al procesar pequeños lotes, estos pueden ser distribuidos en varios nodos dándole una cualidad innata de escalabilidad. El principal problema es que al separar un todo se aumenta la probabilidad de fallos y se tiene que pensar en el concepto de consistencia eventual.

4. Procesamiento híbrido o arquitectura lambda, consta de dos capas, una capa de procesamiento por lotes y otra para velocidad usando técnicas de streaming, pensada para realizar consultar por lotes y procesar nuevos datos con streaming combinando lo mejor de ambos tipos de procesamiento. Su principal desventaja es que existe un desfase cuando se realiza consultas, puesto que al mismo tiempo pueden estar siendo procesados datos en tiempo real que no son tomados en cuenta.

2.1.5 COMPUTACIÓN EN STREAMING

La computación en streaming se refiere a una arquitectura de software diseñada para aplicaciones que requieren sofisticadas y oportunas capacidades de procesamiento de grandes volúmenes de un flujo continuo de datos (Balazinska, M., Rasin, A., Cetintemel, U., Stonebraker, M., Zdonik, S., Hwang, J. H., Zdonik, S., 2005). En este tipo de aplicaciones, los datos son enviados en forma de streams o conjuntos de datos al que le son aplicadas un conjunto de funciones o consultas de manera continua.

Un stream o flujo de datos, es una secuencia de tuplas generadas de manera continua en tiempo real el cual debe ser procesado a su llegada. En este modelo de procesamiento, los datos sufren transformaciones antes de ser guardados en un DBMS al contrario de modelos

tradicionales de procesamiento de datos en el que los datos se almacenan en base de datos primeramente y posteriormente son procesados.

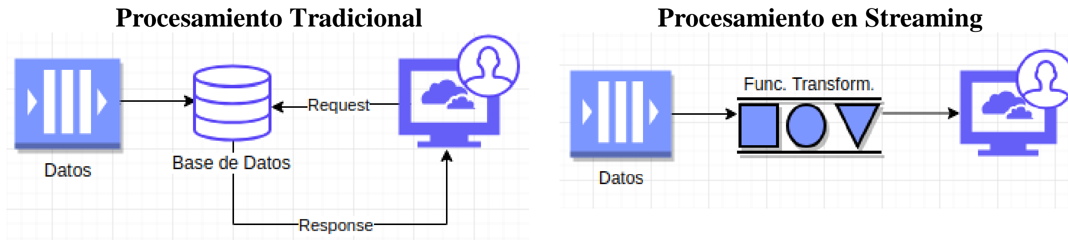


Figura 2.2: Procesamiento tradicional vs procesamiento en streaming
Fuente Elaboración propia

Al ser conjuntos de datos capaz de ser procesados deterministamente, entonces también son capaces de ser distribuidos en muchos nodos de procesamiento lo cual le da una cualidad innata de escalabilidad. Esta cualidad hace que su uso haya sido extendido a entornos distribuidos el cual conlleva al problema de posibles fallos y la gestión de la disponibilidad. Para eliminar este problema, los distintos frameworks para procesamiento de streams utilizan estrategias como la replicación de datos o copias de seguridad, ambas opciones tienen ventajas y desventajas, mientras la primera tiene un costo doble en hardware, el segundo tiene un costo mayor en tiempo de procesamiento. Por ello es imperante que se deba clasificar los datos que se quieran procesar para balancear los siguientes cuatro aspectos:

- Disponibilidad: Sistemas 24/7.
- Durabilidad: Sistemas con datos que no se pueden perder.
- Rendimiento: Sistemas de gran velocidad de procesamiento.
- Latencia: Sistemas de procesamiento en tiempo real.

También existe una categoría de computación en streaming como “Flujo de datos discretos”, los cuales plantean la ejecución de un proceso como un conjunto de tareas

deterministas, pequeñas y sin estado (Zaharia, M, 2016), los cuales a diferencia de los flujos de dato continuo y con estado, son capaces de guardar su propio estado en memoria en estructuras tolerantes a fallos que pueden ser recalculadas deterministamente. La principal ventaja es que, al descomponer un proceso en pequeñas tareas, no solamente expone las dependencias existentes, sino que permite aplicar la aplicación de algoritmos de recuperación más eficientes y procesamiento por lotes y en paralelo.

Existen frameworks que permiten la computación en streaming, entre los cuales los más conocidos son Apache Storm, Apache Flink, Apache Flume, Apache Spark, Apache Kafka, mismos que se desarrollarán posteriormente.

Estos frameworks son similares en la idea de procesar flujos de datos y procesarlos en tiempo real, pero tienen distintas capacidades en cuanto a tolerancia a fallos, paralelismo, escalabilidad y fuentes de datos; mientras que algunos son de propósito general otros están enfocados a segmentos más específicos como la analítica de datos.

2.1.6 FRAMEWORKS DE PROCESAMIENTO EN STREAMING

Las tecnologías o frameworks que permiten la computación en streaming más importantes son los descritas a continuación:

2.1.6.1 APACHE STORM

Es un framework libre y de código abierto que permite la computación distribuida en tiempo real, de baja latencia, basado en una arquitectura maestro-esclavo. Es relativamente sencilla de aplicar y es posible de utilizar con varios lenguajes de programación populares. Los principales casos de uso de Apache Storm son el análisis de datos, machine learning y

sistemas de procesamiento continuo. Apache Storm puede conectarse con sistemas de colas como ser RabbitMQ, Kafka, JMS o Amazon Kinesis además de bases de datos.

En la arquitectura maestro-esclavo existe una máquina maestra encargada de la coordinación, asignación y monitoreo de las tareas distribuidas entre los nodos esclavos los cuales son encargados de recoger y procesar las tareas, en caso de que algún nodo falle la tarea es retomada por otro nodo esclavo.

2.1.6.2 APACHE SPARK

Es un sistema de computación distribuida que permite procesar grandes cantidades de datos de forma masiva, distribuyendo las tareas en varios nodos de procesamiento proporcionando escalabilidad horizontal. La principal característica es que es capaz de ejecutar una tarea de manera distribuida, donde cada nodo procesa una parte de la tarea.

La arquitectura de spark igualmente sigue un patrón maestro-esclavo, donde un nodo es maestro y encargado de la asignación y control de las tareas y otros nodos son esclavos, aunque en cada nodo esclavo son capaces de correr varios “workers” que finalmente son los encargados de ejecutar las tareas.

Spark es ideal para el análisis de datos, operaciones de tipo map-reduce o el procesamiento de tareas extremadamente complejas. No es streaming real, por lo que no es recomendable cuando se busca baja latencia en el procesamiento de datos.

2.1.6.3 APACHE FLINK

Es un motor nativo de procesamiento de streaming de baja latencia. Tiene características de ser distribuido, tolerante a fallos, posee librerías de procesamiento de

eventos complejos o CEP, baja latencia, soporta lotes de milisegundos y por lo tanto tiene un alto rendimiento.

Su arquitectura es cliente-servidor en el que Flink levanta un gestor de trabajo o Job Manager que se encarga de coordinar todas las tareas con componentes llamados Task Manager o gestores de tareas los cuales se encargan de ejecutar partes del código en paralelo.

2.1.6.4 APACHE KAFKA STREAMS

Kafka es una plataforma de streaming distribuida. La principal diferencia con otras tecnologías de procesamiento en streaming es que Kafka es un clúster que corre independientemente a las aplicaciones, entonces una aplicación es capaz de recibir los beneficios de la computación en streaming solamente por utilizar e implementar las librerías proveídas por Kafka.

Su arquitectura sigue el patrón publish/subscribe similar al protocolo MQTT, en donde los clientes publican o se suscriben a streams de registros. Nativamente estos registros son durables y tolerantes a fallos.

Kafka es ideal para aplicaciones que requieren procesar flujos de datos en tiempo real que sean capaces de interactuar con varios otros tipos de aplicación basadas o no en Kafka. También es ideal para desarrollar aplicaciones que reaccionen a eventos de datos y sea capaz de transformarlos en tiempo real.

2.1.6.5 COMPARATIVA ENTRE FRAMEWORKS

Para la selección del framework a utilizar se revisaron investigaciones y estudios comparativos entre los diferentes frameworks (Karimov, J., Rabl, T., & Katsifodimos, A., Samarev R., Heiskanen H., Markl V., 2019), del cual se obtiene el siguiente estudio.

La comparación se realizó con CPUs de 2.40GHz Intel(R) Xeon(R) con 16 cores, 16GB RAM y un ancho de banda de 1 Gbps. La primera prueba consiste en medir el rendimiento en megabytes por segundo de salida de procesamiento.

	2 nodos	**4 nodos**	**8 nodos**
Storm	0.4 M/s	0.69 M/s	0.99 M/s
Spark	0.38 M/s	0.64 M/s	0.91 M/s
Flink	1.2 M/s	1.2 M/s	1.2 M/s
Kafka Streams	1.2 M/s	1.2 M/s	1.2 M/s

Tabla 2.1: Comparativa de rendimiento entre frameworks de streaming
Fuente: Karimov, J., Rabl, T., & Katsifodimos, A., Samarev R., Heiskanen H., Markl V., 2019

En términos de rendimiento Storm y Spark son comparables, siendo Spark un 8% más rápido. En el caso de Flink y Kafka Streams, su rendimiento está limitado al ancho de banda de la red.

También se realizaron pruebas de latencia cuando un sistema está saturado y cuando se encuentra al 90% de su capacidad.

	2 nodos			4 nodos			8 nodos		
	Prom	Min	Max	Prom	Min	Max	Prom	Min	Max
Storm	1.4	0.07	5.7	2.1	0.1	12.2	2.2	0.2	17.7
Storm 90%	1.1	0.08	5.7	1.6	0.04	9.2	1.9	0.2	11
Spark	3.6	2.5	8.5	3.3	1.9	6.9	3.1	1.2	6.9
Spark 90%	3.4	2.3	8	2.8	1.6	6.9	2.7	1.7	5.9
Flink	0.5	0.004	12.3	0.2	0.004	5.1	0.2	0.004	5.4
Flink 90%	0.3	0.003	5.8	0.2	0.004	5.1	0.2	0.002	5.4
Kafka	0.4	0.003	11.5	0.25	0.003	5.5	0.2	0.004	5.5
Kafka 90%	0.3	0.003	5.5	0.2	0.003	5.1	0.3	0.003	5.5

Tabla 2.2: Estadísticas de latencia, promedios, mínimos y máximos en segundos

Fuente: Karimov, J., Rabl, T., & Katsifodimos, A., Samarev R., Heiskanen H., Markl V., 2019

En la tabla se puede ver que Spark tiene las mayores latencias, pero la menor variación entre el promedio, mínimos y máximos. Storm incrementa su latencia a medida que crece el clúster. Finalmente, Flink y Kafka con las menores latencias, con una ligera ventaja de 5% aproximadamente para Kafka, ambos debido a su característica de proceso por micro lotes de información.

2.1.6.6 SELECCIÓN DE LA TECNOLOGÍA

Para la elaboración de la presente tesis se escogió Apache Kafka Streams como tecnología base de procesamiento de streaming por las siguientes razones:

1. Es tecnología madura, siendo su primer reléase en enero de 2011 y cuenta con amplia documentación de la comunidad y de su versión empresarial llamada

Confluent, la cual aporta con soporte y mejoras constantes de la tecnología además de servicios en la nube.

2. Es una tecnología desarrollada por LinkedIn la cual le permite procesar trillones de datos por segundo. Aspecto deseado ya que para IoT en el sector retail se requiera el soporte para el procesamiento de miles o millones de datos.
3. Es escalable, de baja latencia, tolerante a fallos y permite procesamiento de flujo de datos en tiempo real.
4. Permite la creación de un ecosistema de microservicios lo cual permitirá tener un impacto mínimo en las aplicaciones actualmente corriendo en el sector retail, además de poder ir migrando gradualmente los componentes actualmente en producción a la arquitectura basada en streaming.
5. En el sector retail existen sistemas heterogéneos, por lo que contar con sistemas capaces de acoplarse de manera gradual es deseable.
6. Es simple, por lo que la curva de aprendizaje es muy baja y se pueden crear aplicaciones basadas en la computación en streaming muy rápidamente.

2.1.7 ARQUITECTURA DE KAFKA STREAMS

2.1.7.1 TÓPICOS

La principal abstracción del procesador de streaming propuesto son los tópicos.

Un tópico se define como una categoría de datos bien concreta, en donde los registros son publicados y de donde múltiples suscriptores pueden consumir los datos de manera paralela.

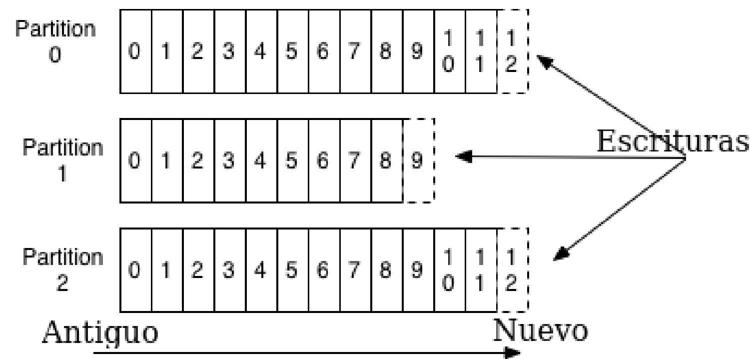


Figura 2.3: Anatomía de un tópico

Fuente: Documentación de Apache Kafka, 2019

Una partición se describe como una ordenada e inmutable secuencia de registros constantemente guardada como un log. A cada registro en una partición se le identifica con un offset, que es un número secuencial y único. Cuando un consumidor recoge datos de una partición, guardará el offset de consumo como una metadata más de ese consumer en el log.

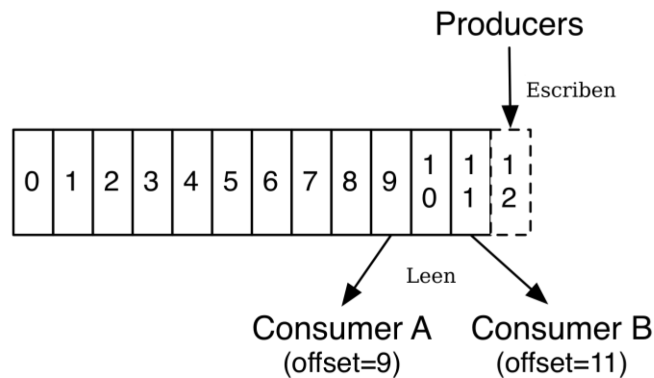


Figura 2.4: Modelo de consumo de datos de un tópico

Fuente: Documentación de Apache Kafka, 2019

En la gráfica se muestra como varios consumidores pueden leer datos de una partición independientemente, de la misma manera de un sistema de procesamiento publish/subscribe. Los clientes que guardan datos en Kafka son conocidos como productores o producers.

La partición de un tópico el aspecto más importante; permite que un tópico escale más allá de los límites de un solo nodo del clúster, una partición tiene que estar en un servidor, pero varias particiones pueden tener diferente cantidad de datos que pueden almacenarse en muchos servidores. Finalmente, lo más importante, una partición se convertirá en la unidad de paralelismo del modelo propuesto, pues determinará la cantidad de hilos que se pueden ejecutar en una aplicación.

Cada una de las particiones se distribuyen a lo largo del clúster, pero además cada partición puede ser replicada en otro miembro del clúster para darle la característica de tolerancia a fallos.

Cada partición tiene un server el cual actúa como líder de esa partición, este se encarga de sincronizar los datos de la partición en los otros nodos conocidos como seguidores. Si un líder falla entonces un seguidor asumirá el rol de líder, entonces, cada nodo del clúster es líder de un conjunto de particiones y seguidor de otros.

2.1.7.2 PRODUCTORES Y CONSUMIDORES

Los productores publican datos a los tópicos y son responsables de la asignación de la partición dentro del tópico. Esta asignación puede realizarse siguiendo un algoritmo round-robin o un algoritmo particular de particionamiento que usualmente se basa en la llave del registro o algún otro que se le puede proveer.

Los consumidores son clientes que recogen los datos del clúster donde están guardados los registros. La característica principal de estos es que tiene una etiqueta llamada “nombre de grupo de consumidores”. Los datos que el clúster entrega son a un grupo de consumidores y

no a uno solo, entonces, cada consumidor puede ser un proceso diferente corriendo en diferentes máquinas.

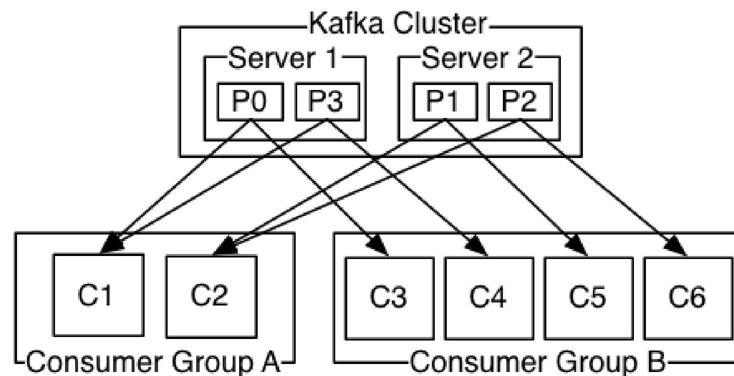


Figure 2.5: Asignación de particiones a consumidores

Fuente: Documentación de Apache Kafka, 2019

Si todas las instancias de consumidores tienen el mismo nombre de grupo o "consumer group" entonces los registros serán efectivamente balanceados entre todas las instancias de consumidores.

Si, por el contrario, todos los consumidores tienen diferente "consumer group" entonces cada registro será enviado a todas las instancias de consumidores.

2.1.7.3 TOPOLOGÍA DE LOS PROCESADORES DE DATOS

Se define como topología a la lógica de procesamiento de streaming definida para una aplicación, es decir, el como una entrada de datos es procesada para producir una salida. En otras palabras, una topología es un grafo de procesadores de flujo de datos conectados entre sí.

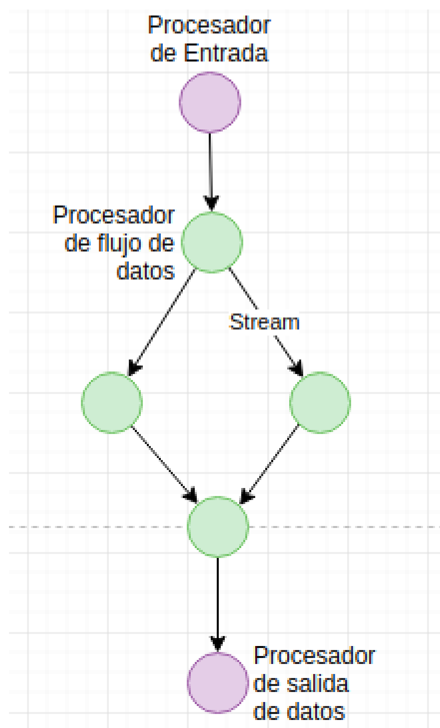


Figura 2.6: Topología de procesamiento en streaming

Fuente: Elaboración propia

En el modelo planteado, los procesadores de entrada y salida con procesadores especiales puesto que se encargan solamente de leer y guardar datos en un tópico. Un procesador de entrada puede escuchar muchos tópicos, pero recogerá los datos de las particiones que tenga asignado. Un procesador de salida escribe datos a un solo tópico, pero una topología puede tener varios procesadores de salida.

Una topología de procesamiento equivale a una abstracción lógica del código de procesamiento de la aplicación. Esta topología es la que es replicada en cada hilo de procesamiento lo que permite el paralelismo.

El software que aplique este modelo deberá definir una topología crean nodos de procesamiento de datos e indicando nodos de entrada y de salida que corresponden a tópicos de entrada y de salida.

2.1.7.5 PARTICIONAMIENTO

El particionamiento permite el balanceo del guardado de los datos, el balanceo del transporte en la red y el procesamiento de los mismos. El particionamiento es lo que define la ubicación de los datos, la elasticidad, escalabilidad, alto rendimiento y tolerancia a fallos.

Los conceptos de particionamiento del flujo de datos y tareas del flujo de datos son las unidades lógicas del modelo de paralelismo.

- Una partición de flujo de datos es una secuencia ordenada de registros que corresponden a un tópico.
- Un registro de datos de stream proviene de un mensaje de un tópico.
- La llave de un registro de datos determina la partición en que se guardará el registro.

Un procesador de streaming escala dividiéndose en múltiples tareas. En otras palabras, el framework crea un número fijo de tareas basadas en la cantidad de particiones de entrada. Cada tarea crea independiente las mismas instancias de objetos definidas en la topología de la aplicación, esto hace que cada tarea pueda ser procesada de manera independiente y en paralelo.

Cabe mencionar que el máximo nivel de paralelismo que una aplicación puede lograr esta determinado la cantidad de tareas que se pueden ejecutar, y esto a su vez está determinado por la cantidad de particiones a los que la aplicación está suscrita.

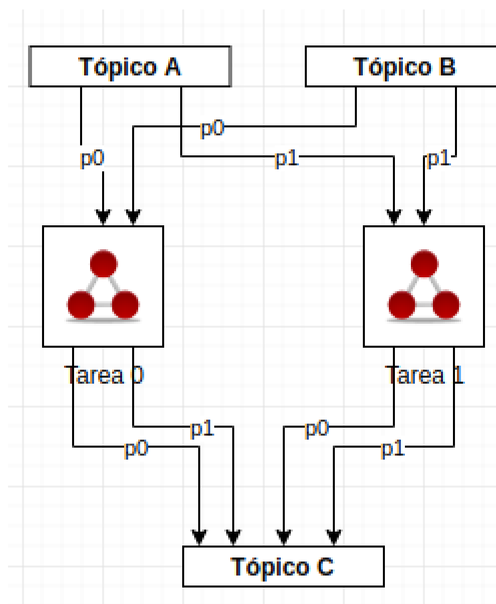


Figura 2.7: Modelo de asignación de particiones de un tópico a tareas

Fuente: Elaboración propia

El gráfico ejemplifica cómo se realiza la asignación de particiones a las tareas. Se nota que cada tarea tiene los mismos objetos instanciados por lo que son capaces de ejecutarse en paralelo.

2.1.7.6 MODELO DE ESCALABILIDAD

Para escalar una aplicación se incremente el número de hilos que corren en una instancia es paralelizar el procesamiento, donde cada hilo puede ejecutar varias tareas independientemente.

Es decir, escalar una aplicación basada en streaming es simple, solo basta con levantar más instancias de la aplicación y el framework se encarga de la distribución de las particiones a distintas tareas donde el grado de paralelismo es igual a la cantidad de particiones existentes.

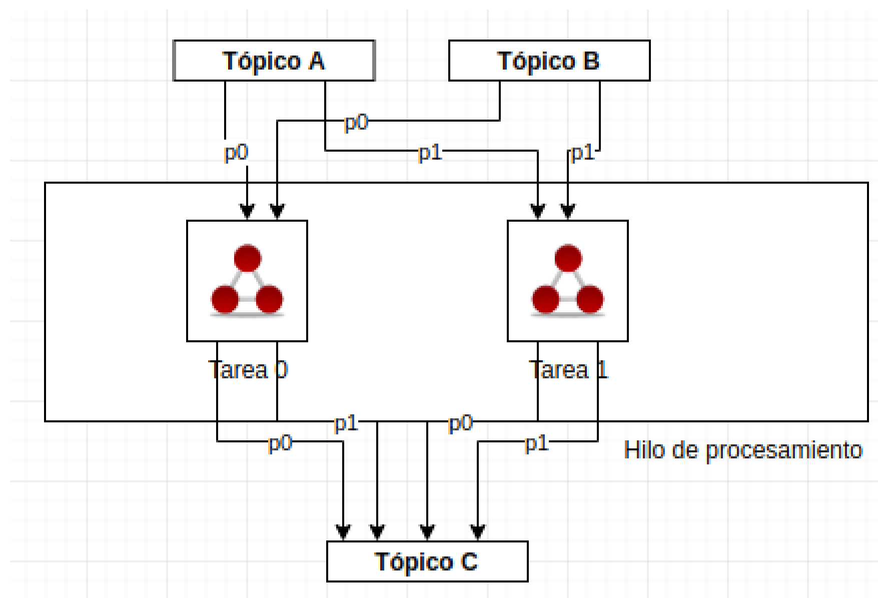


Figura 2.8: Agrupación de tareas en hilos de procesamiento

Fuente: Elaboración propia

Iniciar varios hilos de procesamiento equivale a duplicar la topología del programa y cada una procesando datos de distintas particiones.

Pero algo importante para el procesamiento eficiente de los datos es que no debe existir un estado compartido entre los hilos; los programas realizados para procesar en streaming deben cumplir esta condición, puesto que no existe coordinación inherente entre los hilos de proceso.

El siguiente gráfico ejemplifica la escalabilidad provista por el modelo de hilos:

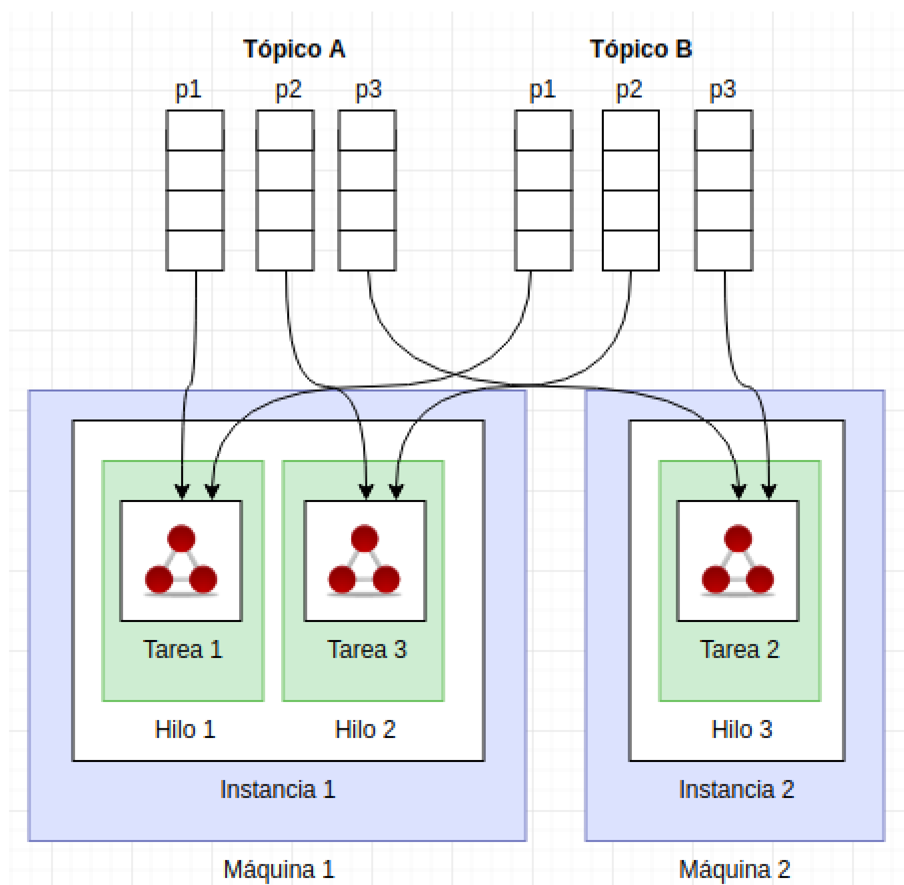


Figura 2.9: Modelo de escalabilidad

Fuente: Elaboración propia

La figura muestra cómo se realiza la asignación de particiones a las tareas, mismas que están distribuidas en las instancias existentes y finalmente, estas instancias pueden estar en varias máquinas del clúster.

2.1.7.7 MODELO DE TOLERANCIA A FALLOS

Todas las tecnologías que permiten el procesamiento en streaming proveen capacidades de tolerancia a fallos. En el caso del framework seleccionado, Kafka, de la misma manera la tolerancia a fallos se ofrece de manera nativa.

1. Las particiones se replican en el clúster, por lo que los datos tienen una alta disponibilidad incluso si un nodo del clúster falla.

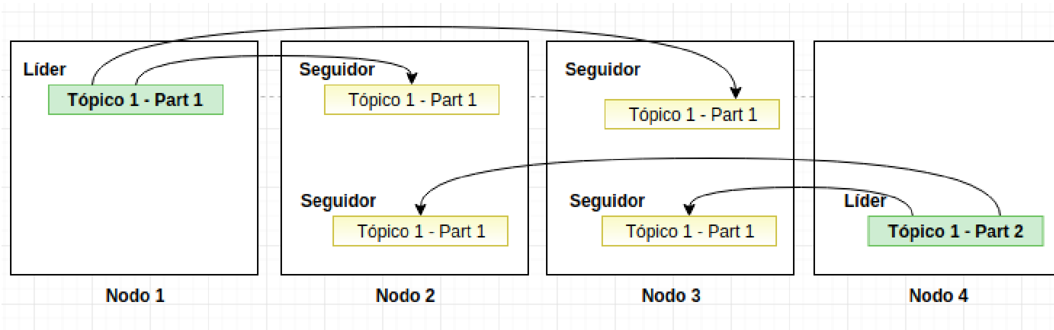


Figura 2.10: Replicación de una partición en el clúster

Fuente: Elaboración propia.

Para este modelo propuesto, se plantea un factor de replicación de tres, es decir, que cada partición de un tópico tendrá tres copias en el clúster, una sola será la partición líder y dos copias en otros nodos.

2. Si la máquina donde están corriendo alguna tarea cae, entonces la tarea es delegada a otra instancia de la aplicación que esté disponible.

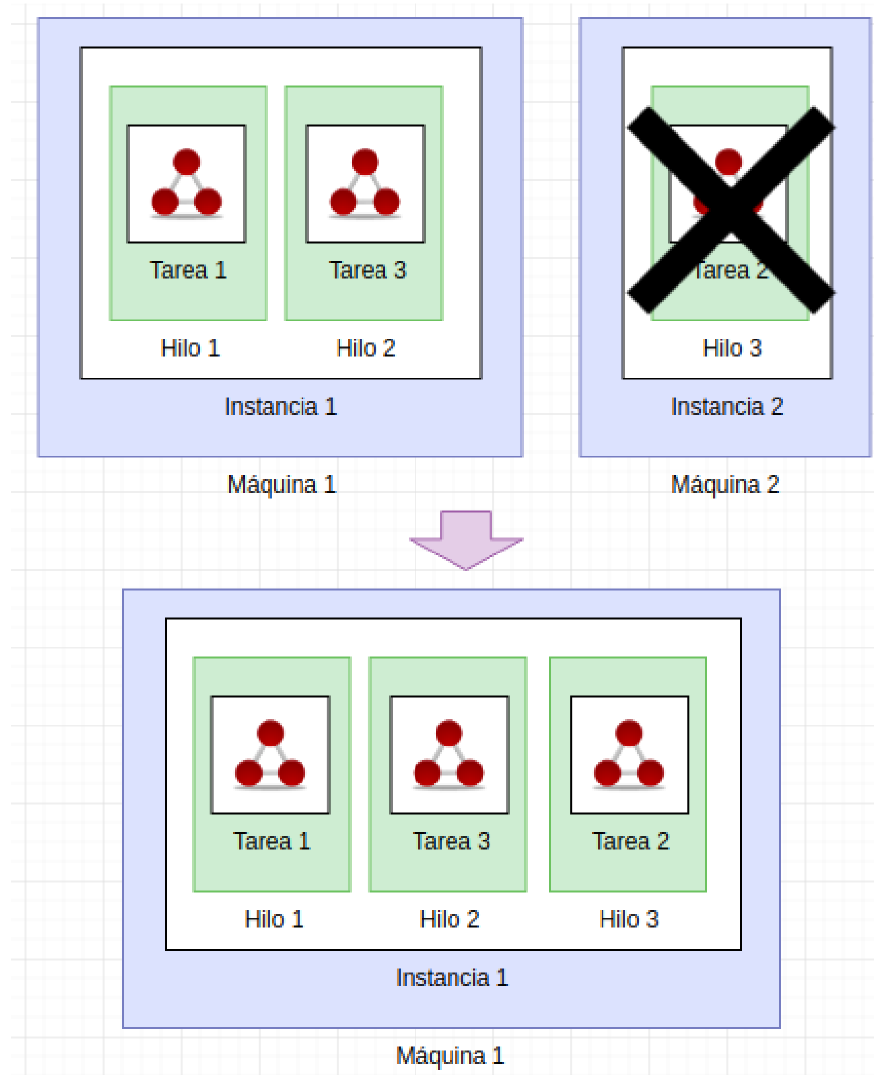


Figura 2.11: Reasignación de tareas en el modelo de tolerancia a fallos
Fuente: Elaboración propia

3. En el caso extremo donde caigan todas las instancias de la aplicación, el dato a ser procesado sigue guardado en el clúster, cuando una instancia se levante este comenzará a procesar desde la posición del último registro procesado.

2.1.7.8 APLICACIÓN EN EL SECTOR RETAIL

El sector retail, al aplicar las tecnologías IoT, cuenta con una estructura de procesamiento como se mostró en el capítulo 2, estado del arte. Los datos son recogidos por las antenas a través de diferentes protocolos, como ser HTTP o MQTT, estos pasan a ser procesador por la capa de ingestión de datos y posteriormente son guardadas una base de datos NoSQL la cual permite a sistema externos realizar el análisis de los datos y presentarlos en forma de tablas y gráficos.

En este sentido la arquitectura de software planteada tiene la característica principal de utilizar Kafka como "bus de eventos", esto quiere decir que el medio de comunicación entre los distintos componentes se realizará a través del clúster de Kafka.

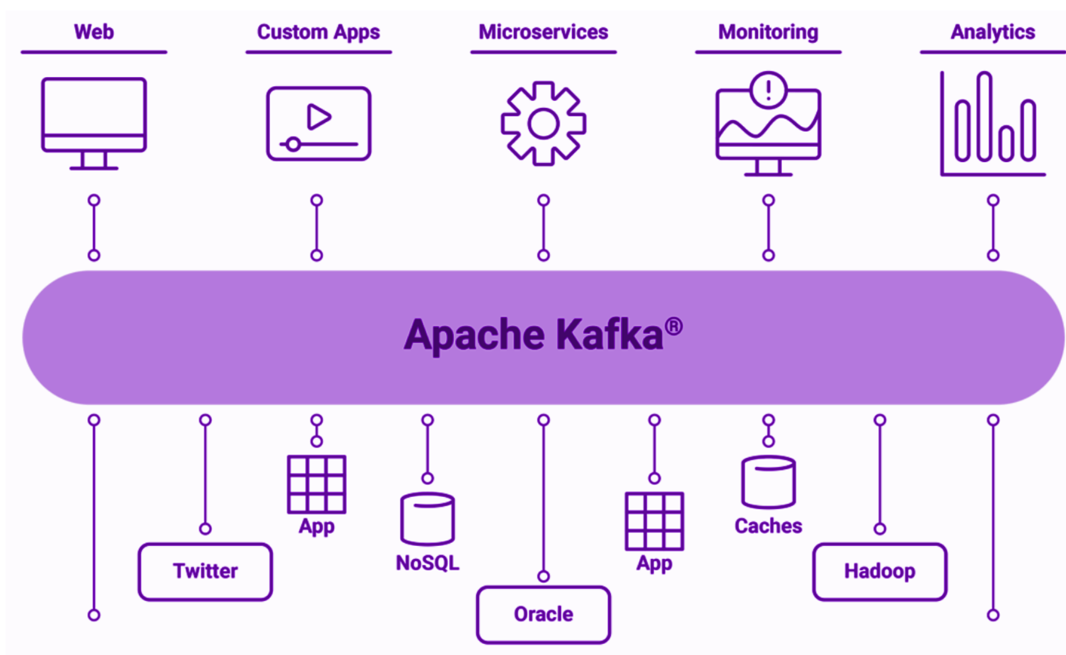


Figura 2.12: Bus de eventos del software basado en streaming
Fuente: Documentación de Confluent, 2019

De esta forma cada uno de los componentes de la aplicación tienen las capacidades de la computación en streaming y se simplifica en gran medida los canales de comunicación existente previamente.

2.1.8 CARACTERÍSTICAS DEL PROCESADOR DE STREAMING

El procesador de streaming es responsable de procesar un flujo de datos continuo en tiempo real. La facilidad que ofrece el framework utilizado, es que muchas cosas ya son heredadas por la aplicación de streaming, como por ejemplo, ya no se necesita pensar en obtener nuevos registros del clúster, simplemente se reciben y se procesan.

Cada mensaje está compuesto por clave y valor, como se muestra en la siguiente gráfica:

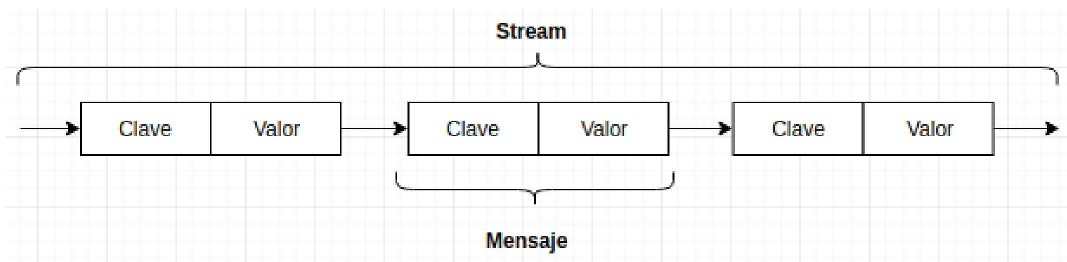


Figura 2.13: Composición de un mensaje en un stream de datos.
Fuente: Elaboración propia

Un stream de datos es procesado dentro de una misma instancia de la aplicación, no se necesitan sistemas separados para ejecutar la aplicación, sino solamente una conexión con el clúster de Kafka a donde se escriben y de donde se reciben datos siguiendo el patrón publish/subscribe.

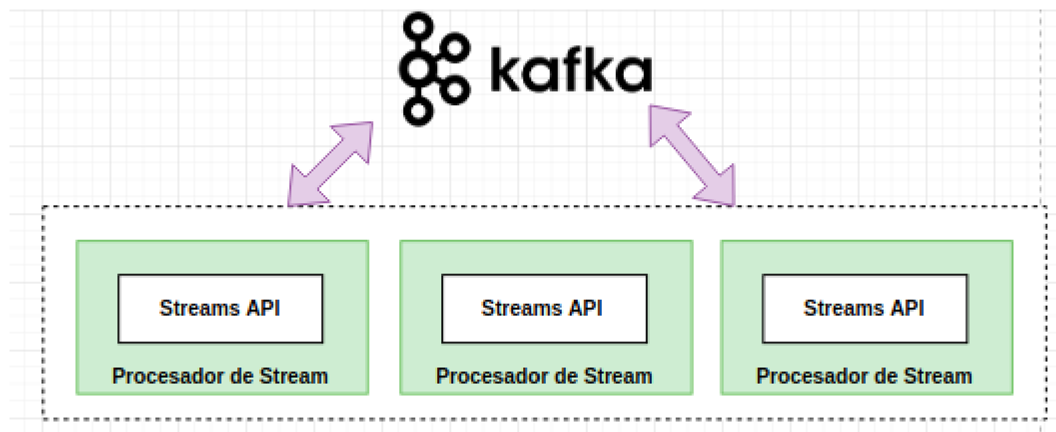


Figura 2.14: Interacción de procesadores de streaming con el clúster de Kafka
Fuente: Elaboración propia

Un procesador de stream es una instancia de la aplicación, la cual puede adicionarse o removerse en cualquier momento, en caso de adicionarse las tareas se balancean con el nuevo componente y en caso de removerse las tareas se asignan a una instancia de este libre (Confluent Documentation, 2019).

La vista lógica es tan sencilla como esa, una aplicación que utiliza Kafka Streams sigue siendo una aplicación normal, escrita en Java, en Scala, Python o algún otro, pero donde se pueden adicionar cualquier otra característica como un servidor web, herramientas de monitoreo, pues es solo una aplicación más, pero al incluir las librerías de Kafka Streams ya posee capacidades de escalabilidad, disponibilidad y tolerancia a fallos.

2.1.8.1 ESCALABILIDAD

La arquitectura de software basada en la computación en streaming es inherentemente escalable como se mostró en capítulo 3, marco del modelo. La capacidad de escalabilidad de una aplicación está definida por la cantidad de particiones con las que cuenta un tópico.

Una aplicación en streaming será capaz de escalar horizontalmente en el momento que se necesite, como por ejemplo, cuando se reciba mayor cantidad de datos. Cuando una

aplicación escale se producirá un evento de rebalanceo de carga, la cual se encarga de asignar particiones a la nueva instancia creada.

2.1.8.2 RENDIMIENTO

El rendimiento de una aplicación basada en streaming está definida por el nivel de paralelismo, este a su vez está limitado a la cantidad de tareas que se pueden ejecutar al mismo tiempo que a su vez está limitado por la cantidad de particiones de un tópico.

Escalabilidad y rendimiento son proporcionales, esto quiere decir que, si se escala una aplicación de una a dos instancias, entonces el rendimiento también crecerá en un factor de dos. Esto quiere decir que, si una instancia de una aplicación procesa a 1000 mensajes por segundo, entonces dos instancias de la aplicación procesarán 2000 mensajes por segundo. Para ello las dos instancias de la aplicación deben estar configuradas con el mismo nombre de grupo de consumidor.

2.1.8.3 DISPONIBILIDAD

El factor disponibilidad está sujeto a la replicación de los datos de una partición. El diseño de una aplicación basada en streaming requiere de un clúster de al menos tres miembros, de forma que los datos puedan estar replicados por lo menos en un factor de tres; de esta forma se asegura que cuando caiga un miembro del clúster siempre existirá una réplica de la información en otro miembro el cual puede ser consumido.

De la misma manera el concepto de replicación se aplica a las instancias de la aplicación. Para asegurar que una aplicación esté siempre arriba, se requiere de un factor de escalamiento de tres, por la misma razón, en caso de una instancia de la aplicación cae, otra

instancia se encargará de procesar sus tareas hasta que se reponga y se realice un nuevo balanceo de carga.

2.1.8.4 TOLERANCIA A FALLOS

Como ya está claramente especificado por los anteriores puntos, tener una aplicación funcionando en clúster les ofrece varios beneficios y la tolerancia a fallos es algo implícito a la arquitectura basada en la computación en streaming. Cuando una aplicación funciona en clúster y existe coordinación de las tareas que se están realizando entonces es muy difícil tener tiempos muertos de procesamiento. Incluso para hacer una actualización del sistema se puede hacer de manera gradual de forma que para el usuario final una actualización es transparente.

Otro aspecto que ofrece mayor robustez en cuanto a tolerancia a fallos es que los datos se encuentran persistidos en una cola en el clúster. En el caso catastrófico que todas las instancias de una aplicación caigan, entonces solamente la cola deja de ser procesada pues no hay consumidor que recoja los datos de él. Cuando se repongan las instancias de la aplicación entonces sólo se comenzará a procesar los datos desde el último offset procesado.

Para la tolerancia a fallos a través de múltiples data centers se puede crear un replicador de datos capaz de llevar datos de tópicos de un clúster a otro preservando las mismas configuraciones del clúster de origen.

La replicación de un centro de datos tiene los siguientes beneficios:

1. Ambientes activos geo localizados en diferentes áreas, lo cual permite a los usuarios un acceso de baja latencia y alto rendimiento.

2. Protección contra desastres, en caso de una catástrofe que afecte todo un centro de datos, el sistema no caería pues seguiría funcionando haciendo uso del centro de datos alterno más cercano.
3. Analítica de datos centralizada, el cual recoge los datos generados en distintas regiones para generar un análisis de datos global.
4. Interacción cloud-edge, que quiere decir que es capaz de integrar sistemas funcionando en la nube con sistemas funcionando en instalación locales del cliente.

2.1.8.5 SEGURIDAD

Por defecto cualquier framework de procesamiento en streaming no cuenta con seguridad habilitada, pero cuando la seguridad es un aspecto crucial de la aplicación entonces esta se puede configurar para cumplir con los requerimientos de seguridad necesarios como ser: (Apache Kafka Documentation, 2019)

- Encriptación. Se pueden asignar llaves SSL y certificados digitales.
- Autenticación. Vía SASL, cuyos mecanismos disponibles en el framework son: GSSAPI (Kerberos), OAUTHBEARER, Scram, Plain o tokens.
- ACL o listas de acceso. ACLs como mecanismo de autorización también es soportado el cual permite la creación de usuario y la asignación de roles y permisos.

Este aspecto no se está cubriendo para el desarrollo de esta tesis, pero sí es un aspecto que se puede extender a esta investigación.

2.2 MARCO REFERENCIAL

2.2.1 INTERNET DE LAS COSAS Y SU IMPACTO EN RETAIL

El Internet de las cosas o IoT, es un nuevo paradigma que busca llevar el mundo real al mundo digital, y en la actualidad ya lo vemos presente, con toda una gama de dispositivos que cierran la brecha entre lo digital y lo físico. Según algunos investigadores, se predicen que para 2020 habrá 30 millones de dispositivos IoT con un potencial impacto de 11 billones de dólares por año para el 2025. Tal oportunidad de negocio ya está en la mira de empresarios, investigadores, medios de comunicación y público en general, especialmente en países desarrollados.

El concepto básico de IoT consiste en la identificación única de objetos y que estos puedan ser inteligentemente enlazados con otros objetos y luego reconocidos por dispositivos como ser sensores. Tecnologías como RFID o identificación por radio frecuencia, códigos de barras o QR, son capaces de identificar objetos y proveer información a través de dispositivos inteligentes. La idea es simple, que por ejemplo una tienda es capaz de proveer a sus clientes información interactiva y en tiempo real de los objetos que, en una tienda, todo esto gracias al uso de sensores e identificadores únicos, esta información puede ser utilizada para extender la información de un producto, mostrar productos relacionados, hacer compras sin necesidad de esperar en una fila.

La aplicación de IoT es muy variada, aplicada en la creación de ciudades inteligentes, domótica o en fábricas por dar unos ejemplos, pero donde ha tomado mayor importancia es en la industria de retail, es decir tiendas y centros comerciales (Pantano & Timmermans,2014), no solo ha hecho que puedan ofrecer una mejor experiencia al cliente sino que el cliente forme parte del proceso de toma de decisiones (Chen, 2014), aunque otros investigadores sostienen

que aún se han hecho poco para entender la percepción del cliente hacia la tecnología IoT (Gao, Bai, 2014) y otros que se necesita aún más investigaciones relacionadas a la aceptación del cliente de la tecnología IoT (Evanschitzky, H., Iyer, G. R., Pillai, K. G., Kenning, P., & Schütte, R. 2014).

El internet de la cosas ofrece a las tiendas y comercios oportunidades en tres áreas críticas: la experiencia del cliente, la cadena de suministros y nuevas fuentes de ganancia. Lo que antes parecía ciencia ficción hoy en día es una realidad en el siguiente gráfico se ejemplifica los beneficios de la aplicación de IoT tanto para el cliente como para los comercios en sus operaciones.

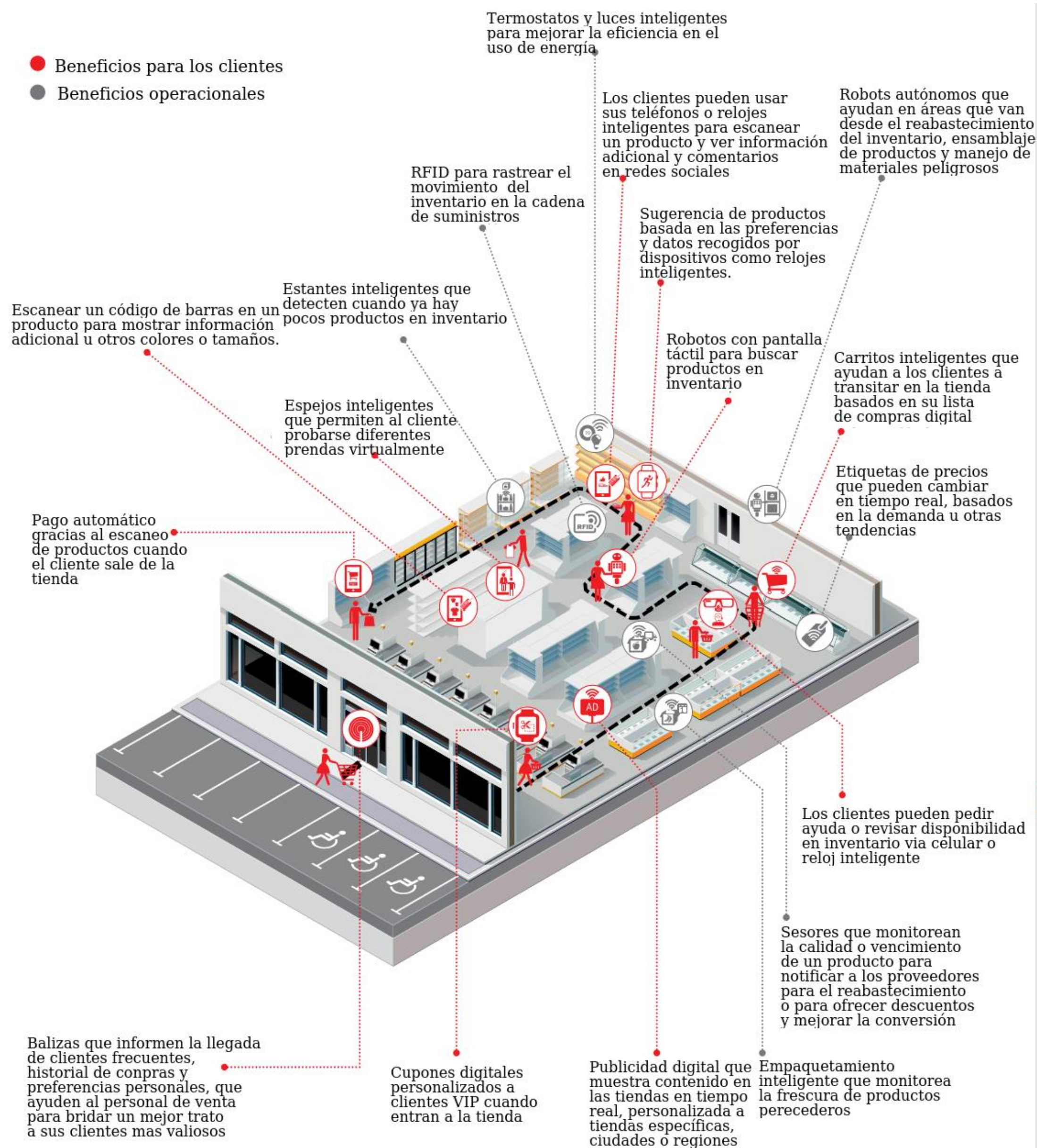


Figura 2.15: Beneficios de IoT en el sector retail
Fuente: Gregory, J. (2015). IoT: Revolutionizing the Retail Industry

2.2.2 ARQUITECTURA DE SOFTWARE DEL SECTOR RETAIL

Hasta hace algunos años, el modelo propuesto por el protocolo TCP/IP era la base de la comunicación y la construcción de aplicaciones, sin embargo, IoT ha creado la necesidad de procesar grandes cantidades de datos (Kraijak, S., & Tuwanut, P., 2015). Los nuevos

estándares y protocolos para procesar IoT necesitan resolver problemas tales como la sostenibilidad, confiabilidad, confidencialidad, calidad de servicio e integridad entre otros, todo debido a que IoT básicamente propone la conexión de todo contra todo lo que hace que el tráfico de información a través de la red crezca exponencialmente. Por ello nacen arquitecturas de software y protocolos que son ampliamente utilizados por el sector de retail.

2.2.2.1 Arquitectura Básica

La arquitectura típica en IoT se divide en cinco capas como muestra la siguiente figura.

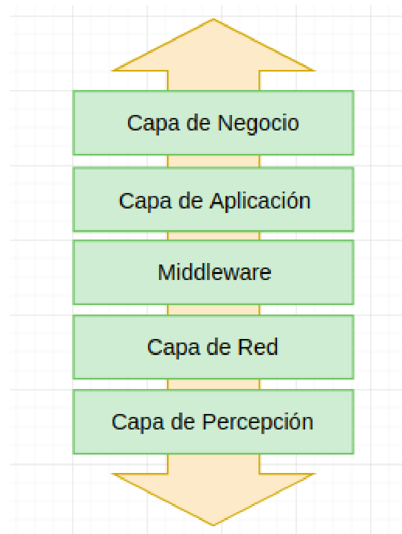


Figura 2.16: Capas Básicas del Modelo IoT
Fuente Elaboración Propia

1. La capa de percepción es similar a la capa física del modelo OSI, pero en este caso consiste de sensores, antenas, identificadores de radio frecuencia RFID, Zigbee, códigos de barra, códigos QR, infrarrojos y otros elementos del entorno. Los datos obtenidos por esta capa puede ser ubicación, velocidad del viento, vibración, nivel de pH, humedad, cantidad de polvo en el aire.
2. La capa de red no solo se encarga de la transmisión de los datos recogidos por los sensores, sino de proveer la seguridad necesaria de datos confidenciales.

3. El middleware tiene dos funciones importantes, almacenar los datos recogidos de las capas inferiores y proveer la información a las capas superiores. En este punto, los datos se convierten en información.
4. La capa de aplicación es la capa que aglomera el software que utiliza la información y lo convierte en algo usable. Las aplicaciones IoT pueden ser buzones inteligentes, hogares inteligentes, carros inteligentes, etc.
5. La capa de negocio es responsable de la creación de gráficos, modelos de negocio, diagramas de flujo, reportes ejecutivos y otros resultados del proceso de análisis e inteligencia de negocios.

2.2.2.2 Protocolos IoT

Un protocolo es un conjunto de reglas que permiten a dos entidades comunicarse de manera efectiva. Los más utilizados para IoT son:

- MQTT (Message Queue Telemetry Transport), es un protocolo de transporte de mensajes de tipo publish/subscribe, donde un cliente publica datos a un tópico y otro cliente suscrito a ese tópico consume los mismos. MQTT es un protocolo liviano, abierto, simple, diseñado para una implementación rápida y que corre sobre TCP/IP. El patrón de diseño publish/subscribe permite la transmisión de mensajes con cardinalidad "uno a muchos", además que permite la configuración de diferentes tipos de calidad de servicio.
- CoAP (Constraint Application Protocol), es un protocolo de transferencia web especializado en nodos y redes limitadas, como ser, baja energía, pérdidas aceptables de datos. Requiere muy poca RAM y velocidades de 10 kbps.

Soporta interacción de tipo request y response de la misma forma que servicios web además de conceptos básicos de identificación como URI y media types.

2.2.2.3 Diseño de Software

El software diseñado para procesar los datos provenientes de IoT contempla

- Recepción de datos, la cual se realiza por distintos protocolos, como ser HTTP, MQTT o CoAP, esta capa permite la recolección de datos de cualquier fuente y en cualquier formato, datos que pasarán a ser procesados por la capa de ingestión.
- Ingestión de datos, que consiste en el procesamiento de los datos para aplicar filtros y reglas necesarias para cumplir las necesidades del negocio. La salida de esta capa son datos útiles para el cliente y usuarios de las aplicaciones.
- Análisis de datos, la cual realiza procesos de minería de datos, aplica algoritmos de big data y otras técnicas que permitan mostrar datos enriquecidos a los usuarios que permitan la toma de decisiones.

La siguiente figura muestra un ejemplo de los componentes de software utilizados en el sector de retail que permite el procesamiento de datos en IoT

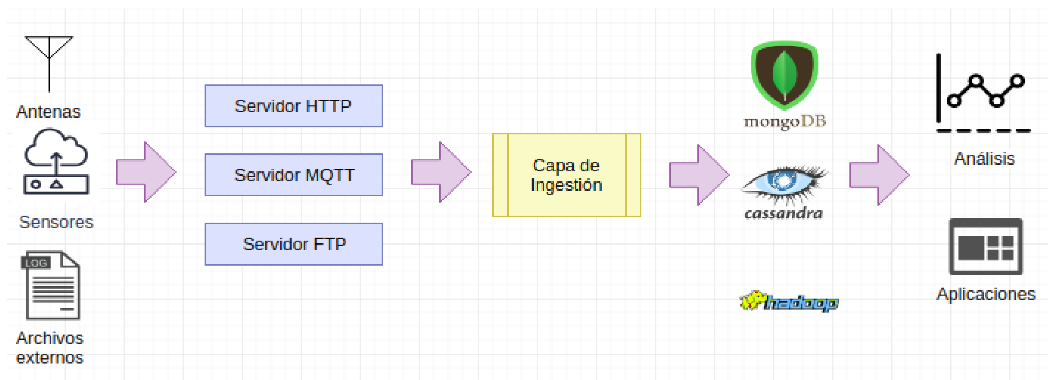


Figura 2.17: Componentes básicos del software para el sector de retail
Fuente: Elaboración propia

Este modelo de software básico para el procesamiento de datos es replicado en entornos cercanos a la ubicación de las tiendas para reducir la latencia y mejorar la experiencia del usuario al utilizar las aplicaciones.

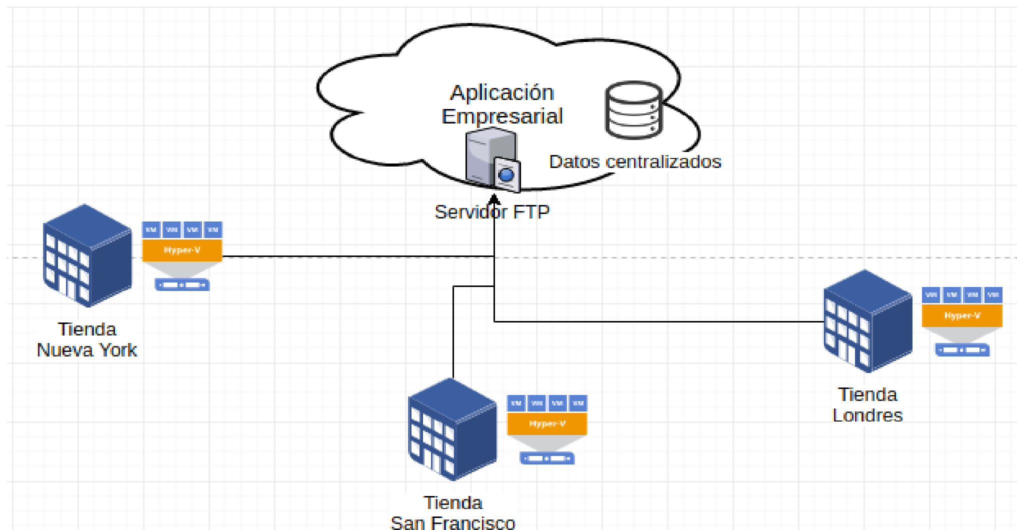


Figura 2.18: Despliegue de software en el sector retail
Fuente: Elaboración propia

Desde el momento que emergen las ideas propuestas por el internet de las cosas, las empresas e investigadores comenzaron a proponer arquitecturas de software capaz de procesar la información generada; las soluciones en la nube se volvieron una tendencia en la industria y con la tecnología 4G los retos de comunicación fueron altamente subsanados en términos de ancho de banda, confiabilidad y baja latencia, pero así como los canales de comunicación iban mejorando, la cantidad de datos recogidos y las necesidades iban creciendo.

Es por esto que la arquitectura de software más utilizada es la computación en la niebla o "Fog Computing", que es una simplificación o abstracción de las soluciones en la nube (Chang, Y. P., Chen, S., & Lee, Y., 2016). Un nodo en las mismas instalaciones de una tienda puede responder mucho más rápido y ser más dinámicas.

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

3.1 MÉTODO DE INVESTIGACIÓN

Se utilizará el método científico, puesto que se ha identificado un problema y planteado una hipótesis, se realizará la investigación de las teorías existentes al respecto y se aplicarán los principios conocidos de la computación en streaming para deducir si estas pueden reducir o no la complejidad computacional de los sistemas actualmente en explotación de las empresas del sector de retail.

3.1.1 FASES METODOLÓGICAS

1. Relevamiento de los principios de la complejidad computacional y la computación en streaming.
2. Descripción de las arquitecturas computacionales y complejidad actualmente en explotación para empresas del sector de retail.
3. Planteamiento de una arquitectura basada en streaming que satisfaga las demandas del sector de retail.
4. Implementación de la arquitectura en al menos dos empresas, manteniendo otras empresas a modo de control.
5. Medición transversal en el tiempo 1 de los indicadores descritos en la operacionalización de variables dependiente e independiente en la arquitectura experimental y la de control.
6. Validación de la experimentación en el tiempo 2 haciendo uso de las técnicas descritas en el siguiente punto.

3.2 TIPO DE INVESTIGACIÓN

Se utilizará un tipo de investigación exploratoria, puesto que se buscará conocer cómo impacta la computación en streaming en la complejidad computacional del software utilizado en el sector de retail. Los resultados de la investigación darán un panorama de las ventajas y desventajas de aplicar streaming para procesar grandes cantidades de datos de manera confiable, de esta manera encaminar nuevas investigaciones en áreas como ser la ciencia de datos, inteligencia de negocios en tiempo real o computación en la niebla (Yang, S., 2017).

3.2.1 DISEÑO DE INVESTIGACIÓN

Se plantea un diseño de investigación experimental y de corte transversal en donde una vez aplicada la arquitectura de streaming, en un tiempo corto, se tomarán las medidas necesarias de todas las dimensiones afectadas de la variable dependiente, para analizar el modo en el que se ha visto afectada, describiendo las causas de los cambios presentados para compararlos adecuadamente con las soluciones aplicadas actualmente en el sector de retail a modo de control.

3.3 UNIVERSO O POBLACIÓN DE ESTUDIO

El trabajo de investigación se orienta a empresas del sector de retail, empresas que venden bienes y servicios directamente a los consumidores o usuarios finales como ser: supermercados, almacenes, centros comerciales, etc.

Se han escogido 2 empresas para la aplicación de la computación en streaming, estas empresas cuentan con 150 y 1100 tiendas respectivamente, lo cual asegura una carga y estrés computacional apropiado para la experimentación. Por motivos de privacidad no se puede indicar los nombres de las empresas seleccionadas.

La experimentación se realizará de manera coordinada entre distintos equipos de la empresa boliviana “Coderoad” dependiente de Mojix Inc., empresa estadounidense especializada en soluciones empresariales IoT y Big Data.

3.3.1 DETERMINACIÓN Y ELECCIÓN DE LA MUESTRA

Para validar el modelo se escogió dos empresas del sector retail con tiendas distribuidas en diferentes ciudades.

- Mark & Spencer, es un retailer multinacional con base en Londres con más de 900 tiendas en el Reino Unido y otras tiendas en Canadá, Francia, Alemania, Bélgica, Holanda y Austria.
- Kohls, retailer con más de 1100 tiendas departamentales en todo Estados Unidos y Reino Unido.

Debido a las políticas de las empresas y los contratos de servicio suscritos la prueba no se realizó en la totalidad de las tiendas, sino en 50 y 100 respectivamente.

Para Mark & Spencer se utilizó un software tradicional para el procesamiento de datos con un socket abierto para la recepción de los mismos. Este sirvió de control para poder realizar las comparaciones necesarias.

Kohls en cambio, se preparó un ambiente preparado y configurado con una aplicación basada en la computación en streaming aplicando el modelo presentado en esta tesis.

Para ambos retailers se contó con ambientes de pruebas y producción. Se realizaron las simulaciones descritas en el siguiente punto sobre los ambientes de prueba para poder tener resultados más controlados. Los ambientes de producción sirven como referencia para validar los resultados de las simulaciones.

3.4 SUJETOS VINCULADOS A LA POBLACIÓN

Las empresas seleccionadas cuentan con tiendas en Estados Unidos y Europa, pero el diseño, desarrollo, implementación y recopilación de los datos se lo hará desde las oficinas de la empresa ubicadas en La Paz, Bolivia, gracias al apoyo de la empresa CODEROAD S.R.L. como sucursal de ingeniería de Mojix Inc. La empresa tiene 10 años de experiencia en desarrollo de software implementado a nivel mundial, además cuenta con ingenieros de software calificados en diferentes áreas vinculados al desarrollo de software.

Mojix es una empresa estadounidense que ofrece soluciones empresariales para la automatización y digitalización de operaciones y cadenas de suministro en los mercados minorista e industrial. El software que provee soluciones inteligentes para empresas del sector de retail a nivel de ítem y cuenta con clientes importantes alrededor del mundo.

3.5 FUENTES Y DISEÑO DE LOS INSTRUMENTOS DE RELEVAMIENTO DE INFORMACIÓN

3.5.1 FUENTES DE LA INVESTIGACIÓN

Los datos utilizados provinieron de dos fuentes, la tienda Kohls y Mark & Spencer, datos que actualmente se envían a los sistemas controlados por Coderoad Los datos provenientes de estas tiendas se simularon a través de scripts para estresar los sistemas de manera controlada, dichos scripts fueron escritos en Python para generar mensajes con el formato enviado por las tiendas del sector retail seleccionadas.

El script de simulación se encuentra en la sección anexos de esta tesis.

3.5.2 DISEÑO DE LOS INSTRUMENTOS DE RELEVAMIENTO DE INFORMACIÓN

El script de simulación diseñó con las siguientes características:

- Se puede configurar la cantidad de mensajes enviados.
- Se controla la cantidad de mensajes enviados por segundo.
- Se controla el tipo de mensajes enviados. El tipo de mensaje define los campos que se pueden enviar en el cuerpo del request.
- Se controla los campos que se envían y los datos enviados en cada campo. Esto permitirá controlar el tamaño del mensaje.
- Se pueden enviar mensajes de manera paralela creando hilos de ejecución.

3.5.3 PROCESAMIENTO Y ANÁLISIS DE LA INFORMACIÓN

El procesamiento consiste en enviar cantidades controladas de datos usando el script de simulación, tanto a la aplicación en streaming como a la aplicación tradicional y luego analizar los resultados obtenidos.

Caso	Msg. por Seg.	KB por Msg.	Nro. de Instancias	Nro. de Hilos por Instancia
1	250	10	1	1
2	500	10	1	2
3	1000	10	1	4
4	2000	10	1	4
5	2000	10	2	4
6	3000	10	3	4
7	1000	50	1	4
8	1000	50	2	4
9	1000	50	2	2
10	2000	50	4	4

Tabla 3.1: Casos de prueba planteados para la simulación
Fuente Elaboración propia

Para la investigación se utilizó ampliamente la observación científica de los fenómenos que sucedieron durante y después de la experimentación, se hizo uso de gráficos en tiempo real como apoyo, y diagramas que permitan describir el modelo propuesto. A continuación, se detalla lo anteriormente dicho.

Los objetos observados fueron los componentes de software utilizados, los recursos de hardware consumidos, los servidores donde se encuentran desplegados y la tasa de

procesamiento por segundo con el fin de obtener las métricas descritas en los indicadores de la variable dependiente y poder compararlos con un software tradicional.

Para medir la complejidad de tiempo, o, dicho en otras palabras, el rendimiento o velocidad de procesamiento, se hicieron uso de gráficos en tiempo real de uso de CPU, “rate” y “lag”, donde “rate” se define como la cantidad de transacciones por segundo y “lag” como la diferencia entre el índice del último registro a procesar y el índice del último registro procesado.

De la misma forma, para medir la complejidad de espacio, o, dicho de otra forma, el uso de recursos de hardware utilizados, se hará el uso de gráficos en tiempo real del uso de memoria RAM, disco y red.

Se monitoreó el ambiente utilizando herramientas que permiten la obtención de métricas desde Kafka, en este caso la herramienta se llama Kafka Exporter en combinación con Prometheus y Graphana permitieron la obtención de gráficas en tiempo real.

Para describir y modelar lo que se entiende por complejidad y complejidad computacional se harán uso de diagramas de máquinas de estado.

Para describir y modelar el procesamiento en streaming y los componentes que lo permiten se utilizarán diagramas de actividades, secuencia y comunicación.

Para modelar la arquitectura física del sistema, sus componentes y cómo se interrelacionan para hacer posible la computación en streaming se utilizarán diagramas de componentes; asimismo, diagramas de distribución de la arquitectura física del clúster de servidores e interconexiones; diagramas de despliegue para mostrar la arquitectura de distribución de los artefactos de software en los nodos del clúster.

CAPÍTULO IV

MARCO PRÁCTICO

4.1 METODOLOGÍA DE DESARROLLO DE LA SOLUCIÓN

El modelo diseñado siguió la metodología de Vistas de Arquitectura propuesta por Philippe Kruchten ajustada a los requerimientos de un software basado en la computación en streaming (Kruchten, P., 2006).

4.1.1 VISTA DE DESARROLLO

4.1.1.1 DEFINICIÓN DE TÓPICOS

Se definieron los siguientes tópicos necesarios para el procesamiento de datos con el software utilizado por el sector retail.

Descripción	Nombre del Tópico	Particiones	Réplicas
Datos en Bruto: datos tal y como llegaron de las distintas fuentes de datos, su objetivo es almacenar todos los datos capturado por los sensores, antenas, archivos u otra fuente de información.	**rawdata**	16	3
Datos de Entrada: son los datos leídos y sanitizados para el proceso de ingestión y ejecución de reglas del negocio.	**input**	64	3
Datos de Salida: son los datos procesados y que están listos para su guardado. Los datos también sirven de entrada para otras aplicaciones de	**output**	64	3

análisis de datos.			
Acciones: son las acciones a realizar, que como resultado del procesamiento de las reglas del negocio deben ejecutarse. Algunas acciones pueden ser prender alarmas, enviar correos electrónicos, notificar la ocurrencia de un evento a sistemas terceros.	**actions**	16	3
Cubos BI: O cubos de información para inteligencia de negocios, son los datos transformados y planos disponibles para su análisis por alguna herramienta BI.	**bicubes**	32	3

Tabla 4.1. Definición de tópicos
Fuente: Elaboración propia

En la tabla anterior se definen la cantidad de particiones y réplicas que se deben crear para el procesamiento óptimo de los datos.

- La cantidad de réplicas es 3 debido a que el software requiere que una alta disponibilidad y tolerancia a fallos.
- La cantidad de particiones es 16, 32 o 64 para tener un factor de escalamiento acorde a la importancia en rendimiento del software. Dicha importancia responde a las responsabilidades descritas en la tabla 4.2, responsabilidades de los componentes de software propuestos.

4.1.1.2 COMPONENTES DE SOFTWARE

El componente de ingestión definido en la arquitectura de software del sector de retail del punto 2.2.2.3, figura 2.17 de la presente tesis, se ha divido en 4 componentes.

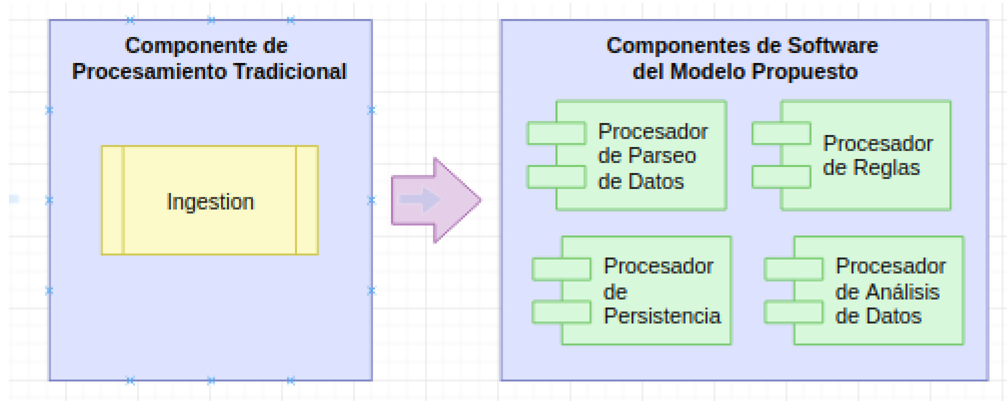


Figura 4.1 Componentes Propuestos por el Modelo para el Sector Retail
Fuente Elaboración Propia

Y para cada componente se ha determinado las siguientes responsabilidades:

Componente	Responsabilidad

Procesador de parseo datos	Componente encargado de la transformación de los datos en entrada en un formato estándar único para el procesamiento de reglas. Los datos de entrada pueden ser variados puesto que las entradas de datos son heterogéneas como ser:
	- Datos en formato CSV.
	- Datos en formato JSON.
	- Datos en formato XML.
	- Datos en base 64.
	- Datos en formato ALE.
	- Datos customizados por el usuario.
	La salida de este procesador es un formato JSON estándar con la siguiente estructura:
	{
	“meta”: {

	"sequence": 125
	"requestId": "SMD9F9MFJWEJ2",
	"origin": [2.4332, 0.003013, 1.234222],
	"source": "POE2_ANTENNA"
	},
	"serial": "22000934859034850",
	"timestamp": "153309929223,
	"typeCode": "ITEM",
	"fields": {
	"location": [2.52, 5.22, 0.0],
	"status": "available",
	"readPointCode": "AA00B23",
	"quantity": 19,
	"epc": "223909AB9C98F98E986CC"
	}
	}
	En donde la meta información son datos para la identificación y control del mensaje. Y por último los fields corresponden a los datos del elemento procesado, estos datos provienen de los datos recogidos por los sensores o antenas y son variables.

Procesador de reglas del negocio	Este procesador contiene todas las reglas de negocio del cliente, estas reglas están relacionadas con el procesamiento de ítems, puesto que todas siguen el estándar GS1 que dicta cómo se debe identificar, capturar y compartir información entre sistemas del sector de retail, esto quiere decir que existe un estándar de tipos de mensajes que se envían o reciben de los clientes, proveedores y terceros dentro de la cadena de suministro.
Procesador de persistencia de datos	Está encargado del guardado de la información en diferentes bases de datos. Estas bases de datos son NoSQL puesto que son las aconsejables para el procesamiento de Big Data, en este caso MongoDB. Este procesador es agnóstico al negocio, su única responsabilidad es asegurarse de la persistencia de la información.
Procesador de analítica de datos	Este componente es paralelo al procesador de persistencia y se encarga de transformar los datos en

	cubos de información. Es decir, aquí se aplican algoritmos de transformación propios de los sistemas de inteligencia de negocios. La salida de estos pueden ser bases de datos o también pueden ser otros tópicos. Cuando los datos se ponen en otros tópicos se pueden obtener gráficos y tableros de información en tiempo real.

Tabla 4.2 Responsabilidades de los Componentes de Software Propuestos
Fuente Elaboración Propia

4.1.1.3 TOPOLOGÍA DEL PROCESADOR DE REGLAS

Se ha diseñado la topología del procesador de reglas, el cual es el componente central de la arquitectura, que se encarga de procesar toda la lógica de negocio con la que cuenta Kohls.

Para ello, se ha identificado la lógica del software actual de Kohls como ser: validaciones, filtros, completar metadata, procesamiento de reglas de negocio y envío de datos, y se han organizado en una topología para procesamiento en streaming.

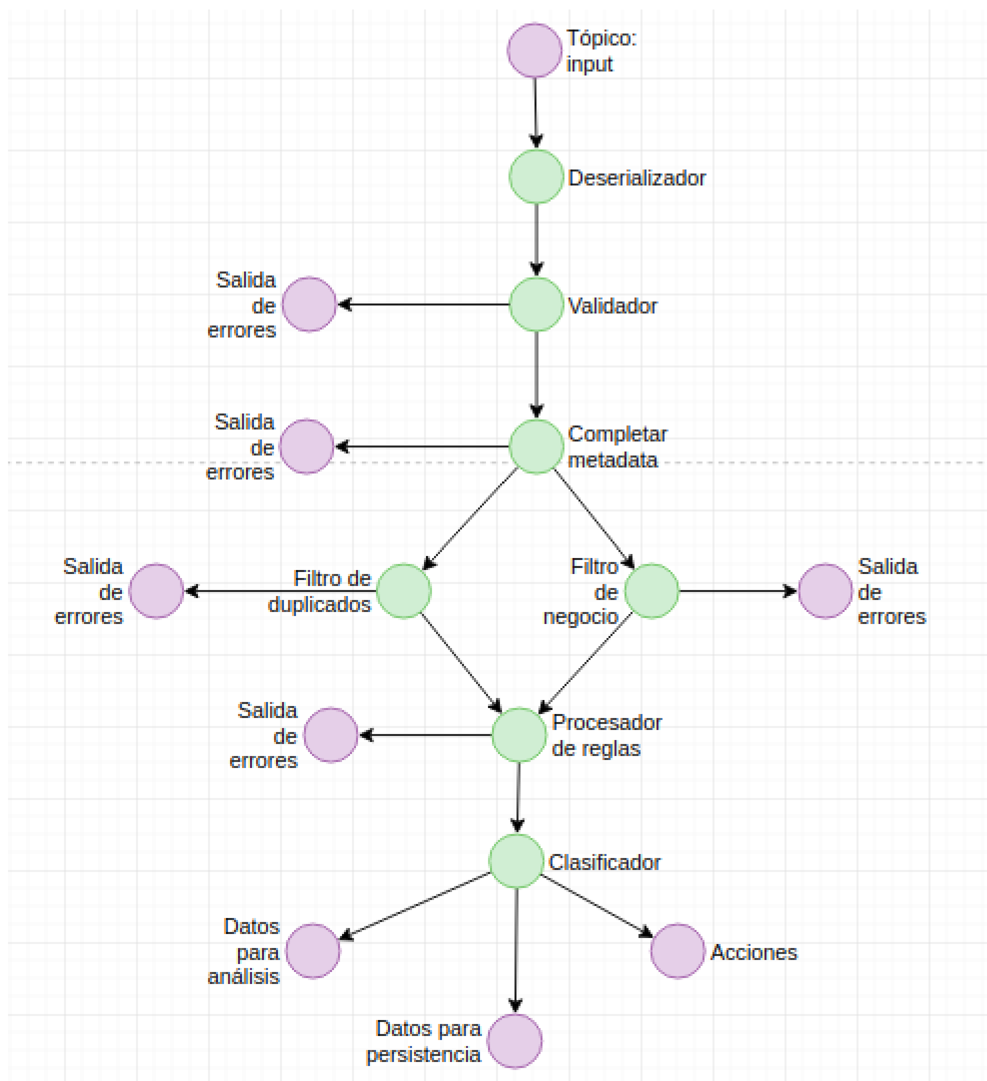


Figura 4.2: Topología de procesamiento de datos
Fuente: Elaboración propia

Para cada nodo de la topología se describieron las siguientes responsabilidades:

Nombre de nodo	Responsabilidad
Nodo de Entrada	Recibe los datos del tópico "input" y los asignan a un hilo de procesamiento. Define un nodo deserializador, el cual transforma los bytes en objetos entendibles por el resto de la aplicación.

Nodo de Validación	Verifica que un mensaje sea procesable, contenga todos los datos necesarios para su proceso y que cuente con las credenciales correspondientes.
Nodo de Completado de Datos	Completa la información con metadata, que permitirá la ejecución de reglas del negocio como ser fuente, usuario, tiempos e identificadores de tipo de dato
Nodo de Filtro	Remueve datos que no son procesables, en el caso de Kohls, se necesita filtros para datos duplicados o filtros de acuerdo a reglas del negocio.
Nodo de Proceso de Reglas de Negocio	Ejecuta las reglas del negocio como ser cambios de estado, procesos antirrobo, ventas de productos, movimiento de inventario, prendido de alarmas u otros.
Procesador de Errores	En cualquier paso del procesamiento pueden existir errores los cuales deben ser procesados y guardados en otros tópicos para su análisis o reprocesamiento.
Procesador de Salida o Clasificador	Clasifica la salida de datos y los envía a diferentes tópicos para su posterior persistencia o ejecución de acciones.

Tabla 4.3 Responsabilidades de los nodos de la topología del procesador de streaming
Fuente Elaboración Propia

4.1.2 VISTA DE PROCESO

Para la vista de proceso se ha diseñado la arquitectura del software de Kohls cuyo flujo de datos en tiempo de ejecución se describe en la figura siguiente.

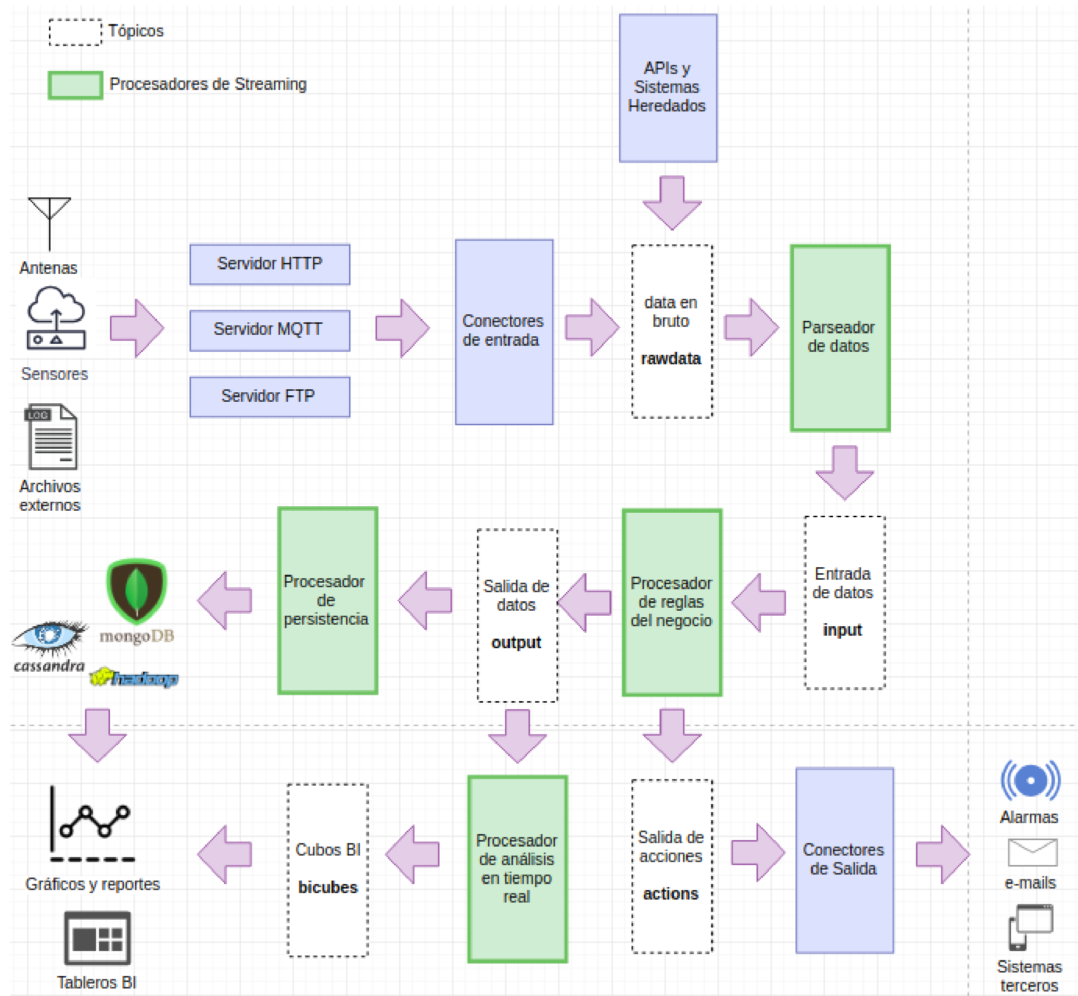


Figura 4.3: Diagrama de componentes de software
para el sector retail aplicando computación en streaming.
Fuente: Elaboración Propia

En la gráfica se pueden observar algunas características que permite el modelo propuesto obtenidas por el procesamiento en streaming:

- El procesador de reglas, componente central de la arquitectura, tiene más de una salida de datos que son consumidos por diferentes aplicaciones.

- El tópico output es consumido por 2 aplicaciones diferentes que realizan diferentes acciones.
- Los conectores de entrada y salida permiten la integración con sistemas terceros funcionando con distintos protocolos y formatos como ser: bases de datos, procesadores de inteligencia de negocio, sensores y antenas.

4.1.3 VISTA FÍSICA

4.1.3.1 MODELO DE DESPLIEGUE

Para desplegar todos los componentes se consideraron los siguientes aspectos:

1. Cada uno de los componentes de software se encapsularon en containers usando docker como tecnología.
2. Los container se orquestan en el clúster de servidores utilizando Kubernetes.
3. El clúster de servidores se desplegó en Google Cloud Platform (GCP).
4. Se cuentan con discos SSD para la base de datos de 3 TB y un ancho de banda de 4 Gbps.

El motivo por el cual se realizó tal configuración es para optimizar el uso de recursos y para facilitar la orquestación de componentes.

Los servidores en la nube utilizados tienen las siguientes características: (Google Cloud Platform - All Pricing, 2019)

Tipo de Nodo	**CPUs Virtuales**	**Mem. RAM (GB)**	**Disco (GB)**	**Cantidad**
n1-standard-4	4	15	128	6
n1-standard-8	8	30	128	1
TOTAL	**32**	**120**	**896 ***	**7**

Tabla 4.4: Descripción de nodos para el modelo de despliegue
Fuente Elaboración propia

Después de iniciar cada uno de los componentes y monitorear el uso de recursos de los mismos, se obtuvo los datos de uso descritos en la tabla siguiente.

Componente	**CPUs**	**RAM**	**Disco (GB)**	**Cantidad**
Zookeeper	0.5	512 MB	1 GB	3
Kafka	1	8 GB	64 GB	3
MongoDB	2	24 GB	2 TB	1
MySQL	1	1 GB	256 MB	1
FTP Server	1	512 MB	1 GB	1
HTTP Server	1	512 MB	256 MB	1
MQTT Server	1	512 MB	1 GB	1
Conectores de Kafka	1	2 GB	1 GB	1
Procesador de Parseo	1	2 GB	256 MB	1
Procesador de Reglas	2	4 GB	256 MB	3
Procesador de Persistencia	1	2 GB	256 MB	2
Procesador de Análisis	2	4 GB	256 MB	2
REST API del Negocio	2	8 GB	4 GB	1
Interfaces UI	0.5	512 MB	256 MB	1
Agentes de monitoreo	0.5	256 MB	1 GB	7
TOTAL (x Cantidad)	**30.5**	**90.25 GB**	**2.2 TB**	**29**

Tabla 4.5: Asignación de recursos para el modelo de despliegue
Fuente Elaboración Propia

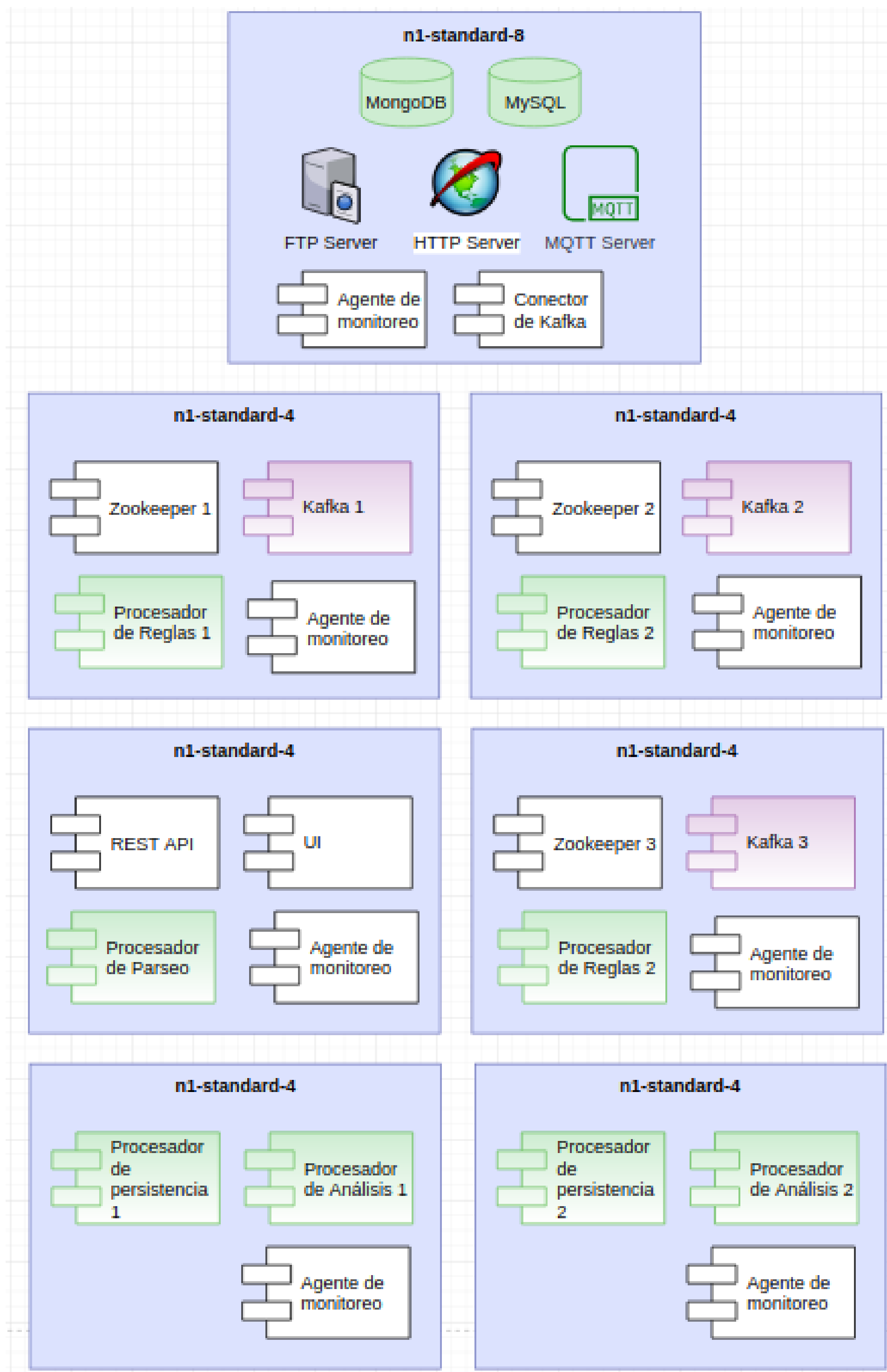


Figura 4.4: Modelo de Despliegue de Componentes
Fuente: Elaboración propia

4.1.3.2 MODELO DE REPLICACIÓN EN MÚLTIPLES CENTROS DE DATOS

Se ha diseñado un replicador de datos para cumplir con un requerimiento del software utilizado en el sector retail, el cual es la accesibilidad desde distintas zonas geográficas ofreciendo la misma calidad de servicio. Este componente permite la replicación de los datos en múltiples centros de cómputo, utilizando las mismas características de streaming.

El siguiente gráfico muestra la arquitectura del replicador.

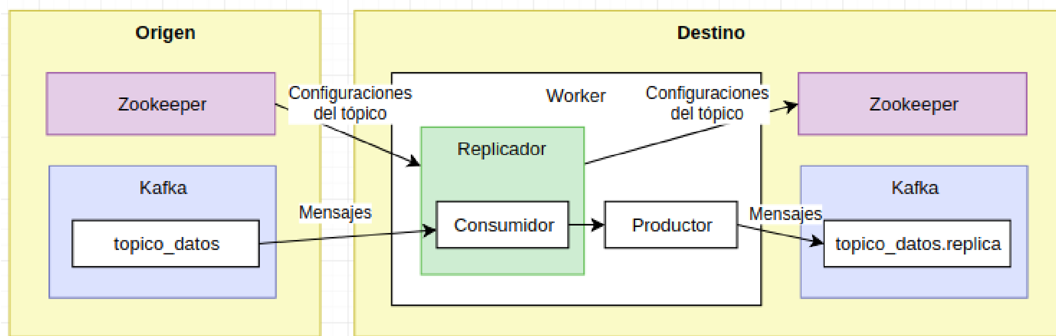


Figura 4.5: Diseño del replicador de datos
Fuente: Elaboración propia

Se aplicó la arquitectura en el software de Kohls para sus sucursales de Nueva York, San Francisco y Londres, definiendo un centro de datos principal y otros secundarios utilizando replicadores para centralizar la información.

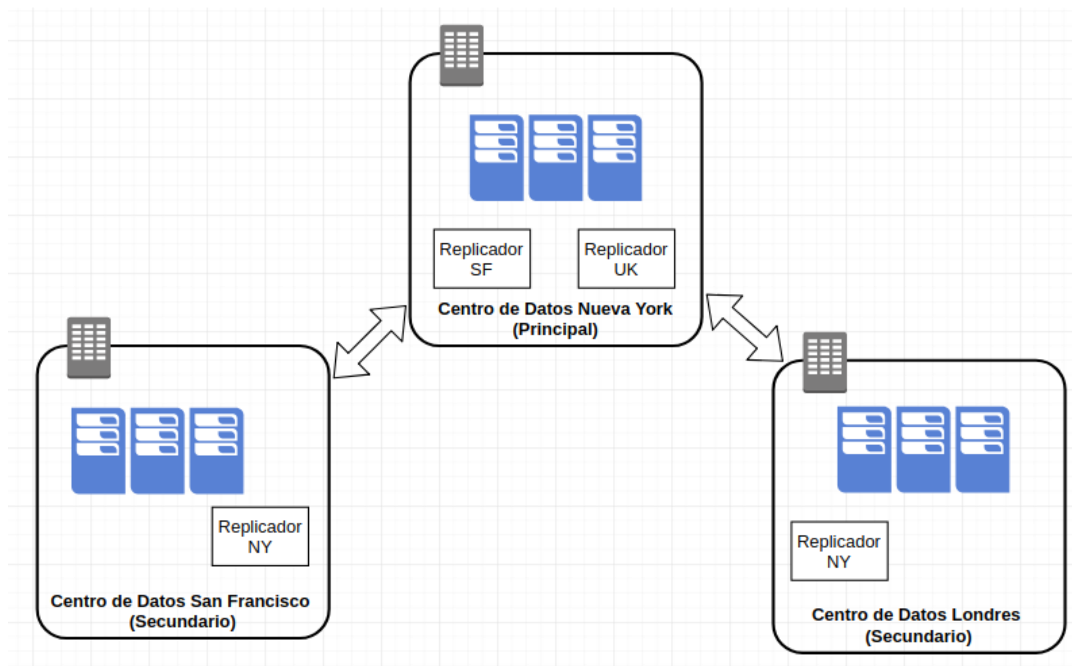


Figura 4.6: Infraestructura de centros de datos distribuidos
Fuente: Elaboración propia

4.2 VALIDACIÓN DEL MODELO

Se aplicó el modelo propuesto en el software de Kohls y se realizaron diferentes pruebas para verificar la reducción de la complejidad computacional en tiempo y espacio, lo que quiere decir que el software sea más eficiente realizando las mismas tareas con mejor cantidad de recursos.

4.2.1 PRUEBAS DE RENDIMIENTO

Los pasos realizados fueron los siguientes:

1. Se prepararon herramientas para el monitoreo del rate y el lag haciendo uso de Kafka Exporter, Prometheus y Graphana para obtener gráficos en tiempo real.

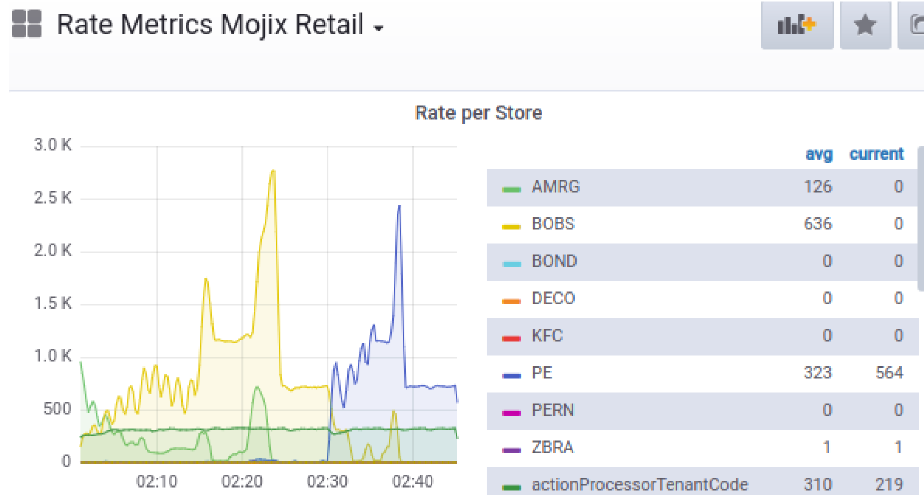


Figura 4.7: Gráficos de rate utilizando Kafka Exporter, Prometheus y Graphana
Fuente Elaboración propia

2. Se agregó un gráfico en graphana con la siguiente fórmula para el cálculo del rate:

```
sum by (topic) (
        rate (kafka_consumergroup_current_offset{
                consumergroup=~"streamProcessor(.*)"
        }[2m])
)
```

Fórmula 1: Cálculo de rate de una aplicación en streaming
Fuente: Elaboración propia

Donde:

- **Kafka_consumergroup_current_offset:** es el último offset procesado de un consumer group.
- **~"streamProcessor(.*):** Es una expresión regular que permite obtener los consumer groups que contienen en su nombre “streamProcessor”.

- **2m:** Indica la ventana de tiempo para el cálculo del rate. Para este caso, es 2 minutos en el que se recogen las métricas y se calcula el promedio.

Y esta fórmula para el cálculo del lag:

```
sum by (topic) (
        kafka_consumergroup_lag{consumergroup=~"streamProcessor(.*)"}
)
```

Formula 2: Cálculo del lag en una aplicación en streaming
Fuente: Elaboración propia

Dónde:

- **Kafka_consumergroup_lag:** Mide la diferencia entre el último offset registrado de una partición menos el último offset procesado por un consumer group. En otras palabras, mide cuando le falta por procesar a un hilo los datos de una partición.
- **~"streamProcessor(.*):** Es una expresión regular que permite obtener los consumer groups que contienen en su nombre "streamProcessor".

3. Para la aplicación tradicional se realizó un escalamiento vertical en las pruebas donde se requería un mayor rate de procesamiento.
4. Se prepararon los scripts de simulación y se enviaron cargas iguales a ambas aplicaciones.

Análisis de Resultados

Tras ejecutar los scripts de simulación se obtiene la siguiente tabla:

					Aplicación en Streaming		Aplicación Tradicional	
Caso	**Msg. por Seg.**	**KB por Msg.**	**Nro. de Instancias**	**Nro. de Hilos por Instancia**	**Rate Promedio**	**Lag Máximo**	**Rate Promedio**	**Lag Máximo**
1	250	10	1	1	250	380	250	0
2	500	10	1	2	500	910	500	0
3	1000	10	1	4	1000	2,100	875	0
4	2000	10	1	4	1320	1,070,000	910	1,550,000
5	2000	10	2	4	2000	3,200	1710	4,200
6	3000	10	3	4	3000	8,200	2270	14,050
7	1000	50	1	4	860	405,000	580	596,880
8	1000	50	2	4	1000	54,000	850	55,000
9	1000	50	2	2	1000	1,900	825	2,052
10	2000	50	4	4	1880	21,000	1320	29,800

Tabla 4.6: Resultados de rendimiento entre aplicación en streaming y tradicional
Fuente: Elaboración propia

Para medir la mejora de rendimiento se ha tomado en cuenta las pruebas de la tabla 4.6 en las que la cantidad de mensajes enviados por segundo es superior al rate de procesamiento de las aplicaciones en streaming y tradicional, es decir los casos 4, 5 y 6, seguidamente se aplicó la siguiente fórmula:

$$\textbf{Mejora de Rendimiento} = \frac{\text{Rate App Streaming} - \text{Rate App Tradicional}}{\text{Rate App Streaming}} \times 100$$

Fórmula 3: Porcentaje de mejora de rendimiento
Fuente: Elaboración Propia

	Rate de App Streaming	Rate de App Tradicional	Mejora en %
Caso 4	1320	910	31.06
Caso 7	860	580	32.56
Caso 10	1880	1320	29.79
PROMEDIO	**1353.33**	**936.67**	**30.79**

Tabla: 4.7: Porcentaje de mejora entre aplicación streaming vs tradicional.
Fuente: Elaboración Propia

Entonces se comprueba un incremento de la mejor de rendimiento de un 30.79% de una aplicación en streaming comparado con una aplicación tradicional.

Otros aspectos observados en la prueba

- Las pruebas 1, 2 y 3 no muestra mejor alguna debido a la baja la carga de datos. Pero cuando la carga comienza a ser mayor, el rate de procesamiento de una aplicación en streaming también es mayor.
- La aplicación tradicional más lenta incluso con el escalamiento vertical aplicado.
- La prueba 7, 8, 9 y 10 demuestran que el tamaño de un mensaje influye en la capacidad de procesamiento de una aplicación en streaming, necesitando más instancias de la aplicación para poder procesar más mensajes por segundo. La explicación de este comportamiento es debido al factor de ancho de banda.

4.2.2 PRUEBAS DE ESCALABILIDAD

La prueba anterior verifica implícitamente que al incrementar una instancia de una aplicación basada en streaming se obtiene una mejor tasa de procesamiento, en

cambio, la presente prueba verifica de una forma más controlada la escalabilidad de los componentes al incrementarse paulatinamente la cantidad de hilos de procesamiento.

Para las pruebas se envió gran cantidad de datos con el script de simulación a los tópicos de entrada de datos, generando un lag de por lo menos media hora para cada prueba, de esta forma al iniciar la aplicación se comienza a procesar a su máximo rate.

Nro. de Instancias	**Nro. de Hilos por Instancia**	**Total Hilos**	**Rate**
1	1	1	310
1	2	2	602
1	3	3	925
1	4	4	1190
2	1	2	595
2	2	4	1175
2	3	6	1850
2	4	8	2280
3	1	3	910
3	2	6	1905

Tabla: 4.8: Prueba de escalabilidad de una aplicación en streaming
Fuente: Elaboración propia

La siguiente figura utiliza los datos de las pruebas con dos instancias de la aplicación. No se grafican los otros casos puesto que son similares y se ajustan a la función obtenida

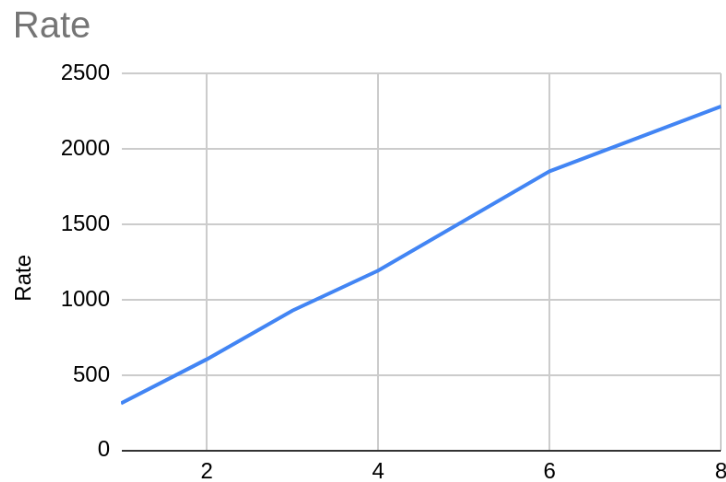


Figura 4.8: Linealidad del escalamiento de una aplicación en streaming

Fuente: Elaboración propia

A través de esta prueba se demostró la linealidad de rendimiento de una aplicación en streaming.

4.2.3 PRUEBAS CARGA

Los pasos realizados fueron:

1. Se prepararon ambientes con la misma cantidad de recursos en núcleos de procesador y memoria RAM para la aplicación streaming y tradicional.
    - **CPUs:** 4
    - **Memoria RAM:** 16 GB
    - **Tamaño del Mensaje:** 50 KB
2. Se enviaron cantidades controladas de mensajes utilizando el script de simulación, para verificar el uso de CPU y memoria consumida para el procesamiento de dichos mensajes.

3. Utilizando Docker, Promethus y Graphana, se generó gráficos del uso de recursos de CPU y memoria RAM.

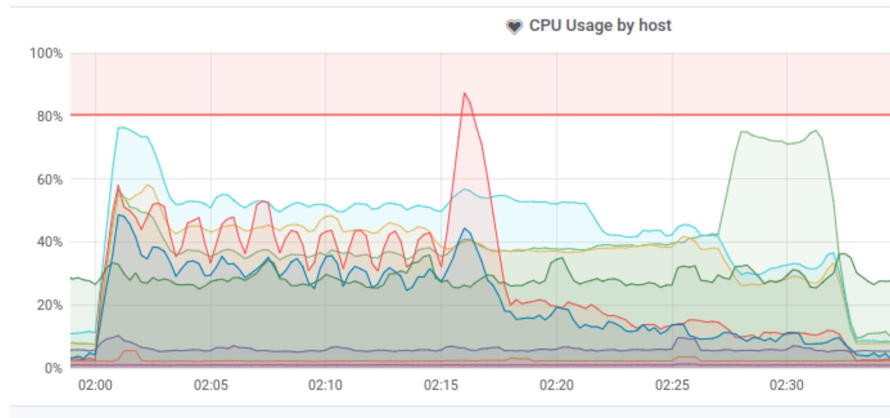


Figura 4.9: Uso de CPU de la aplicación en streaming
Fuente: Elaboración propia

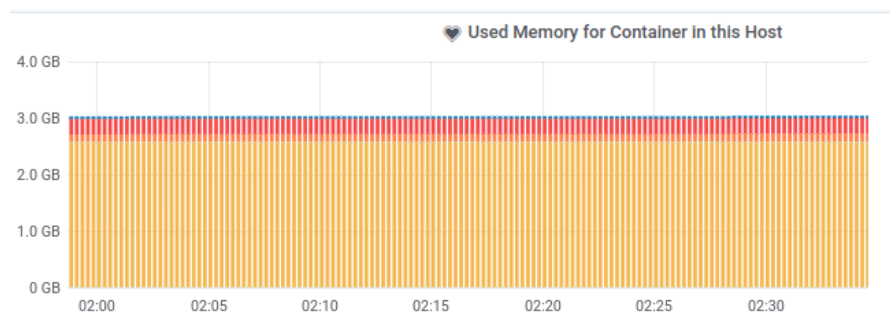


Figura 4.10. Uso de Memoria RAM de la aplicación en streaming
Fuente: Elaboración propia

Análisis de Resultados

Después de enviar datos a ambas aplicaciones se obtiene la siguiente tabla comparativa:

	Aplicación Tradicional		Aplicación en Streaming	
Cantidad de Mensajes	**Uso de CPU**	**Uso de Memoria (GB)**	**Uso de CPU**	**Uso de Memoria (GB)**
1,000	60%	5	90%	1.5
2,000	85%	10	100%	1.5
3,000	100%	15	100%	1.5
5,000	100%	25 *	100%	1.5

Tabla 4.9 Pruebas de carga de una aplicación streaming y tradicional
Fuente Elaboración Propia

La última prueba para una aplicación tradicional no se pudo realizar puesto que no se tenía la memoria suficiente para procesar 5 mil de mensajes y se tuvo que enviar lotes de datos uno después de otro.

Graficando ambas tablas se obtiene lo siguiente:

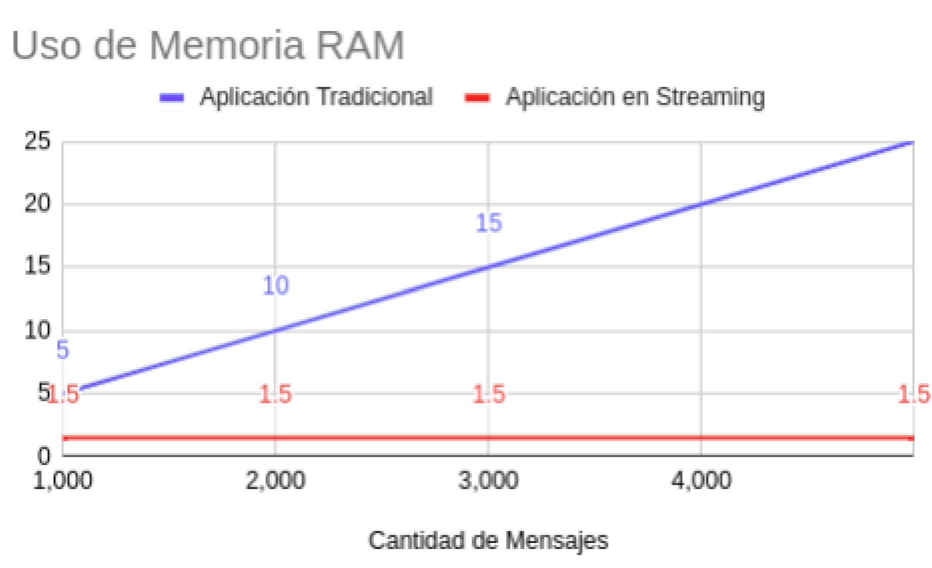


Figura 4.11 Comparativa de uso de recursos entre una aplicación streaming vs aplicación tradicional
Fuente Elaboración propia

La prueba muestra claramente que el uso de memoria RAM de una aplicación en streaming se reduce considerablemente. La explicación es simple, una aplicación

en streaming no procesa los datos guardándolos en memoria para procesarlos, en cambio obtiene los datos en micro lotes por lo que el uso de memoria es pequeño y se van procesando con el tiempo.

4.2.4 PRUEBAS DE TOLERANCIA A FALLOS

Esta prueba se realizó solamente con la aplicación en streaming debido a que una aplicación tradicional no es tolerante a fallos por sí misma siguiendo los siguientes pasos:

1. Se levantó 3 instancias de la aplicación.
2. Enviar gran cantidad de datos para que estos sean procesados tan rápido como sea posible. El objetivo es tener a la aplicación siempre ocupada e incluso con un lag aceptable.
3. Apagar forzosamente una de las instancias de la aplicación.
4. Verificar la continuidad del procesamiento de datos.
5. Levantar la aplicación previamente detenida.
6. Verificar la continuidad del procesamiento de datos.

Resultados Obtenidos

Se comprobó el rebalanceo de carga, donde los datos que se estaban procesando por instancia pasaron a ser procesados por las instancias que aún quedaban vivas.

```
Oct 19 09:53:44 ● kafka [2019-10-19 13:53:44,735] INFO [GroupCoordinator 1]: Preparing to rebalance
  group mm-c2f-RED-FRA082-A83 in state PreparingRebalance with old generation 11335 (__consumer_offsets-
  5) (reason: Adding new member mm-c2f-RED-FRA082-A83-0-76b9717f-0c82-4dec-acaf-70cebf922e9b)
  (kafka.coordinator.group.GroupCoordinator)
Oct 19 09:53:44 ● kafka [2019-10-19 13:53:44,736] INFO [GroupCoordinator 1]: Stabilized group mm-c2f-
  RED-FRA082-A83 generation 11336 (__consumer_offsets-5) (kafka.coordinator.group.GroupCoordinator)
Oct 19 09:53:45 ● kafka [2019-10-19 13:53:45,244] INFO [GroupCoordinator 1]: Preparing to rebalance
  group mm-c2f-RED-FRA082-A83 in state PreparingRebalance with old generation 11336 (__consumer_offsets-
  5) (reason: Adding new member mm-c2f-RED-FRA082-A83-4-8ed834fc-9c18-4e7c-a996-02f45663690c)
  (kafka.coordinator.group.GroupCoordinator)
Oct 19 09:53:46 ● kafka [2019-10-19 13:53:46,330] INFO [GroupCoordinator 1]: Stabilized group mm-c2f-
  RED-FRA082-A83 generation 11337 (__consumer_offsets-5) (kafka.coordinator.group.GroupCoordinator)
Oct 19 09:53:46 ● kafka [2019-10-19 13:53:46,331] INFO [GroupCoordinator 1]: Preparing to rebalance
  group mm-c2f-RED-FRA082-A83 in state PreparingRebalance with old generation 11337 (__consumer_offsets-
  5) (reason: Adding new member mm-c2f-RED-FRA082-A83-2-48f0721c-21f5-4342-8f8b-b6eae39456b8)
  (kafka.coordinator.group.GroupCoordinator)
Oct 19 09:53:48 ● kafka [2019-10-19 13:53:48,088] INFO [GroupCoordinator 1]: Stabilized group mm-c2f-
  RED-FRA082-A83 generation 11338 (__consumer_offsets-5) (kafka.coordinator.group.GroupCoordinator)
Oct 19 09:53:49 ● kafka [2019-10-19 13:53:49,778] INFO [GroupCoordinator 1]: Assignment received from
  leader for group mm-c2f-RED-FRA082-A83 for generation 11338 (kafka.coordinator.group.GroupCoordinator)
Oct 19 09:53:50 ● kafka-http-connector [2019-10-19 13:53:50,655] DEBUG Polling for new source records...
  url: https://dev.red.vizix.io/statemachine-api-configuration/rest/configuration/rules/fixtures/modes
  (com.tierconnect.riot.bridges.connectors.http.HttpSourceTask:67)
Oct 19 09:53:50 ● kafka-http-connector [2019-10-19 13:53:50,655] DEBUG Fetching data from url:
```

Figura 4.12: Balanceo de carga en prueba de tolerancia a fallos
Fuente: Elaboración propia

Al revisar los logs de la aplicación se puede verificar las siguientes características:

- El rebalanceo remueve las particiones asignadas a los miembros del consumer group.
- Se reasigna las particiones a los consumidores y una vez hecho esto, se considera al grupo como rebalanceado.
- Un consumidor recibe la asignación y comienza a recibir registros de las nuevas particiones.

4.2.5 COSTOS

Los costos económicos incurridos para el despliegue y procesamiento de datos de una aplicación streaming aplicando el modelo propuesto es el siguiente:

Tipo de Nodo	CPUs	Memoria RAM (GB)	Cantidad	Costo Unitario por mes (Dólares)	Costo Total por mes (Dólares)
n1-standard-4	4	15	6	97	582
n1-standard-8	8	30	1	194	194
				TOTAL	**776**

Tabla 4.10 Costo de despliegue del software basado en streaming.
Fuente Elaboración Propia

A su vez, los costos de despliegue de una aplicación tradicional para obtener la misma capacidad de procesamiento de datos es la siguiente:

Tipo de Nodo	CPUs	Memoria RAM (GB)	Cantidad	Costo Unitario por mes (Dólares)	Costo Total por mes (Dólares)
n1-standard-32	32	120	1	776	776
n1-standard-8	8	30	2	194	388
				TOTAL	**1164**

Tabla 4.11 Costo de despliegue del software tradicional.
Fuente Elaboración Propia

Para calcular la reducción de costo se aplicó la siguiente fórmula

$$\textbf{Reducción de costo} = \frac{\text{Costo Software Tradicional - Costo Software Streaming}}{\text{Costo Software Tradicional}} \times 100$$

Fórmula 4. Cálculo de reducción de coste de procesamiento
Fuente Elaboración Propia

$$\textbf{Reducción de costo económico} = \frac{1164 - 776}{1164} \times 100$$

Reducción de costo = 33.33 %

Pero aún más importante que los costos aplicados para este modelo, es que una aplicación en streaming puede escalar incrementando el presupuesto en 97$ según la tabla 4.10. De la misma manera se puede disminuir la capacidad de procesamiento si es necesario.

Los factores que permitieron la reducción del coste económico son:

- El escalamiento vertical de recursos de un servidor en la nube es generalmente en potencias de 2, lo que significa que si se necesita escalar un servidor con 64 de RAM y 16 CPUs lo siguiente que puedes acceder es un servidor de 32 de RAM y 32 CPUs y puede ser que este sobrepase los requerimientos reales de una aplicación. En el caso de prueba la aplicación en realidad necesitaba 90 GB de RAM y 24 CPUs.
- El incremento del rendimiento de una aplicación en streaming hace que se requieran menos recursos computacionales lo que implica adquirir menos servidores con el correspondiente ahorro que conlleva. En el caso de prueba 1 servidor n1-standard-32 equivale a 8 servidores n1-standard-4 pero se requerían sólo 6 ahorrando el coste de 2 servidores.

Adicionalmente el escalamiento vertical tiene un límite físico, mientras que el escalamiento horizontal es limitado solo por la capacidad de administración del clúster.

CAPÍTULO V

MARCO DE RESULTADOS

5.1 ESTADO DE LOS OBJETIVOS

1. Se ha diseñado una arquitectura de software basado en computación en streaming adaptada al software utilizado en el sector de retail, específicamente el software utilizado en Kohls. La arquitectura contempla 3 componentes descritos en el capítulo IV puntos 4.1.1, 4.1.2 y 4.1.3, donde:
   a. **Vista de desarrollo,** se definieron los tópicos necesarios (Tabla 4.1, pág. 68), componentes de software desarrollados (Figura 4.1p, pág. 68) y topología del procesador de reglas de negocio del software para Kohls (Figura 4.2, pág. 73).
   b. **Vista de proceso,** muestra la interacción de los componentes desarrollados (Figura 4.3, pág. 75), donde se ve el flujo de datos en tiempo de ejecución desde la recepción de datos de antenas y sensores, pasando por los tópicos y procesados de streaming hasta su guardado en base de datos, análisis de datos o interacción con alarmas o sistemas de notificaciones.
   c. **Vista Física,** describe el modelo de despliegue de la infraestructura (Figura 4.4, pág. 78) y replicación de datos en múltiples centros de datos (Figura 4.6, pág. 80).
2. Se ha mejorado la eficiencia en el procesamiento de datos del software del sector retail, es decir, se pueden procesar datos con un menor costo de recursos computacionales; para comprobarlo se han realizado las pruebas

respectivas descritas en el punto 4.2.1, 4.2.2, 4.2.3 y 4.2.4 obteniendo los siguientes resultados:

a. **Pruebas de Rendimiento**: donde aplicando la Fórmula 3 (pág. 81) sobre los datos de la Tabla 4.7 (pág. 84) se obtiene una mejora de 30.79% en el rate de procesamiento.

   Rate promedio del software en streaming: 1353.33 msg/seg

   Rate promedio del software tradicional: 936.67 msg/seg

   Mejora de Rendimiento $= \frac{1353.33 - 936.67}{1353.33} \times 100 = 30.79\ \%$

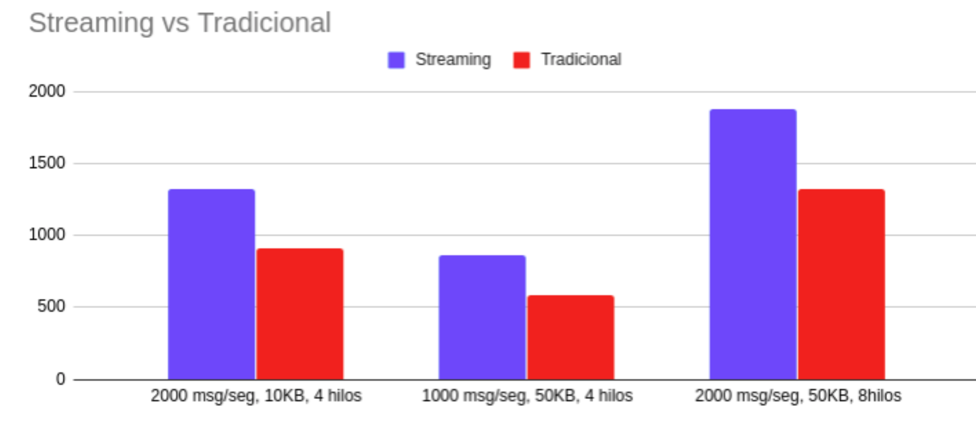


Figura 5.1 Resumen comparativa de rendimiento entre software en streaming y software tradicional
Fuente Elaboración propia

b. **Pruebas de Escalabilidad:** donde se verificó la escalabilidad de una aplicación en streaming demostrando que el rendimiento mejora tan solo con iniciar más instancias de la misma (Tabla 4.8 pág. 85, Figura 4.8 pág. 86).

c. **Pruebas de Carga:** donde se comprobó que el uso de memoria RAM es constante en una aplicación en streaming, mientras que en una aplicación tradicional es lineal, pero además se hace un mayor uso de la CPU (Tabla 4.9, pág. 88).

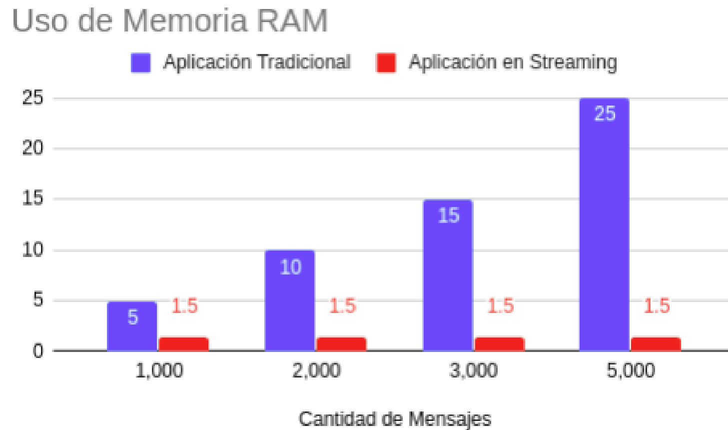


Figura 5.2 Comparativa de uso de memoria RAM entre software en streaming y software tradicional
Fuente Elaboración propia

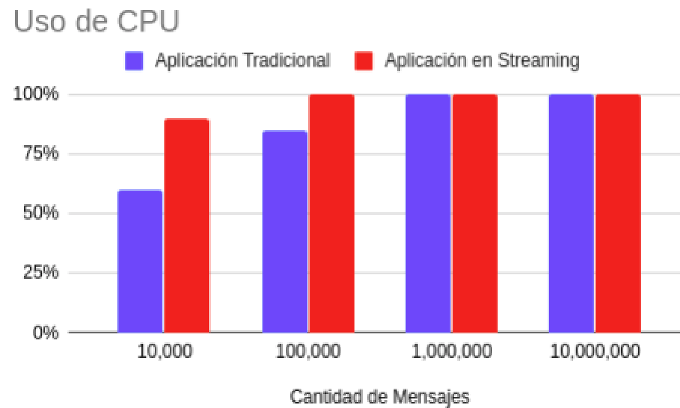


Figura 5.3 Comparativa de uso de CPU entre software en streaming y software tradicional
Fuente Elaboración propia

d. **Prueba de tolerancia a fallos**, donde a través de la figura 4.12 (pág. 90) se verifica que cuando una instancia de una aplicación en streaming cae sus tareas se asignan a otras instancias aún vivas.

3. Se ha reducido el costo mensual de despliegue de la infraestructura en un 33.33%, demostrado al aplicar la fórmula 4 (pág. 91) que calcula la diferencia entre el software streaming y tradicional (Tabla 4.10 y 4.11, pág. 91).

   Costo mensual de software en streaming: 776 $us.

   Costo mensual de software tradicional: 1164 $us.

   Reducción de costo = $\frac{1164 - 776}{1164} \times 100 = 33.33\ \%$

   Pero, además, en caso de requerir mayor rendimiento, se puede escalar las aplicaciones a bajo costo, 97 $us según la tabla 4.10 (pág. 91).

5.2. ESTADO DE LA HIPÓTESIS

Tras la aplicación del modelo propuesto basado en la computación en streaming en un software para el sector retail, se comprobó la reducción de la complejidad en tiempo y espacio, esto quiere decir que el software desarrollado tiene un mejor rendimiento y utiliza menos recursos para procesar la misma cantidad de datos que un software tradicional.

- 33.79% de mejora en rendimiento (Tabla 4.7, pág. 84)
- Mayor aprovechamiento del CPU (Tabla 4.9 pág. 88, Figura 5.3 pág. 95)
- Uso de memoria RAM constante vs uso de memoria lineal (Tabla 4.9 pág. 88, Figura 5.2 pág. 95)

- Reducción de costos de despliegue de un 33.33% (Tabla 4.10 y 4.11, pág. 91)

5.3 CONCLUSIONES

- La aplicación de un modelo de procesamiento de datos basado en streaming en el software del sector de retail reduce la complejidad computacional del software del sector retail mejorando el rendimiento en 30,79% con un uso constante de recursos computacionales, pero además trae beneficios deseables para cualquier empresa y otros del área de tecnologías de información, como ser la escalabilidad, disponibilidad y tolerancia a fallos.
- Se reducen los costos económicos de despliegue de la infraestructura de software en un 33,33% y en caso de requerir mayor rendimiento de la aplicación se puede escalar pequeños servidores de bajo costo.
- La arquitectura en streaming permite que los datos puedan ser procesados por varias aplicaciones en cualquiera de los tópicos de la tabla 4.3, permitiendo que conceptos como ser la analítica en tiempo real sea posible. Además, está diseñada para que cualquier componente que requiera mayor capacidad de procesamiento pueda escalar de manera independiente.

5.4 RECOMENDACIONES

- Se recomienda realizar la evaluación del modelo propuesto en 3 meses, para ajustarlo a los requerimientos constantes de las empresas del sector retail, especialmente en la forma de despliegue de la aplicación en regiones geográficamente distantes, de forma que se pueda ofrecer la mejor experiencia a los clientes en cuanto a rendimiento. Para la validación del modelo en

anexos se incluyen los pasos necesarios para la ejecución de pruebas de complejidad computacional y los scripts necesarios para la simulación de carga.

- El modelo propuesto en esta tesis, se puede extender al campo de analítica de datos, machine learning e inteligencia artificial, por lo que se recomienda proponer soluciones a estos conceptos, pero en tiempo real aprovechando las características de la computación en streaming.
- Como frecuentemente los datos que se procesan en una aplicación empresarial, o parte de ellos, pueden ser críticos se recomienda aplicar mecanismos de seguridad proveídos por los frameworks de procesamiento en streaming como ser el cifrado de datos, la autenticación y la autorización con el fin de proteger la información generada.
- Finalmente, investigaciones posteriores pueden extender el alcance del modelo propuesto a soluciones que lleven el procesamiento de la nube a las propias instalaciones de las tiendas, concepto conocido como "Fog Computing" o "Edge Computing", el cual consiste en tener software de procesamiento similar a la nube pero en las propias tiendas, de forma que la latencia se pueda reducir al mínimo, se pueda trabajar incluso sin conexión a internet y que a través de un proceso de sincronización deseablemente en tiempo real se pueda seguir teniendo la ilusión de un solo macro sistema global en la nube.

ANEXOS

SCRIPT DE GENERACIÓN DE DATOS PARA PRUEBAS DE CARGA

Lenguaje: Python 3.6

Librerías requeridas: Kafka

```
# --------------------------------------------------------------------------------

#python3
import os
import sys
import time
import subprocess
import datetime
import random
from pathlib import Path
from Kafka import KafkaProducer
from kafka.errors import KafkaError
import queue
import threading

TENANT="KOHLS"

RATE = 100 # how many blinks per second, values like 0.5 are allowed

THREADS = 1

#producer = KafkaProducer(bootstrap_servers=['kafka:9092'], batch_size=2000000,
linger_ms=10000)
producer =
KafkaProducer(bootstrap_servers=['localhost:9093','localhost:9094','localhost:9095']
)
TOPIC = "___v2___data0___json"

# Total number of blinks
N = 1000000

OLDER_CHANCE = 0 # <----------------- OLDER_CHANCE % of all blinks will be
older
# Serial numbers involved in the blinks (cyclic)
FROM = 1 #0
```

```
TO   = 100000
# Allow to add an offset to the serial number using arguments
try:
   OFFSET = int(sys.argv[1])
except:
   OFFSET = 0

# Values for the contents of the blinks:
PREFIX = "RATEFILTER"
try:
   PREFIX=sys.argv[2]
except:
   PREFIX=PREFIX
BRIDGECODE= "/_TENANT/ALEB" #"/_TENANT/SM/ALEB"
#"/_TENANT/ALEB__TENANT" #"ALEB" #"ALEB_DISCOVERED"
#"ALEBBC" # 'ALEB' #
THINGTYPE= "/_TENANT/ASSET" #"/_TENANT/item" #
"default_rfid_thingtype" #"default_rfid_thingtype"
#"/_TENANT/SM/DEFAULT_RFID_THINGTYPE" # "item"
#THINGTYPE= "expressionthingtype"
ZONE_OPTS = ["/_TENANT/0000789/ZONE%s%d" % (letter,number) for letter in
"ABCD" for number in (1,2,3,4) ] #
#ZONE_OPTS= [ "/_TENANT/" + x.upper() for x in ZONE_OPTS ]
#ZONE_OPTS = ['/_TENANT/679/679_SalesFloor',
'/_TENANT/341/341_StockRoom', '/_TENANT/780/780_StockRoom',
'/_TENANT/432/432_StockRoom']
STATUS_OPTS = ['on','off','disabled','file_not_found']
#STATUS_OPTS = ['fast'] * 20 + ['slow']
#ZONE_OPTS = ['13','12','45' ]
RANDOM_OPTS =
["9SD8FM","23D23D2","5G45H55S","45G2F24","M91C29","N8C39NW","9NS8D
SFJ9","M98W9N823"]
XYZ = ["X","Y","Z"]
def pick_priority():
   return 3
   #return 1 + 2*random.randint(0,1)
def pick_location():
   return "-108.44395660357625;34.048117009863525;0.0"
# these udfs are added to the blink
def gen_epc(sqn):
   #6.5.8
   a = random.randint(100000,999999)
   x = "%013d" % sqn
   b = x[:5]
   c = x[5:]
```

```
    return ( "urn:epc:sgtin:%d.%s.%s" % (a,b,c) )
def gen_event(sqn):
    return "TransactionEvent-1553202829651-56cda0543974ceddaffdd31b3ab00001"
    #return [
    #    "TransactionEvent-1553202829651-56cda0543974ecddaffdd31b32200007",
    #    "TransactionEvent-1553202829651-56cda0543974ecddaffdd31b32200008",
    #    "TransactionEvent-1553202829651-56cda0543974ecddaffdd31b32200009",
    #    "TransactionEvent-1553202829651-56cda0543974ecddaffdd31b32200010",
    #][sqn%4]
def valuesOld(t, sqn):
    return [
        ("zone"  , random.choice(ZONE_OPTS) ),
        #("status", random.choice(STATUS_OPTS) ),
        #("location", pick_location() ),
        #("httpcode", "666" ),
        #("SA_Clearance", random.choice(XYZ) ),
        #(#"Reassociated", random.choice(XYZ) ),
        #(#"logicalReaderALE", random.choice(XYZ) ),
        #("FirstDetectedZone", random.choice(XYZ) ),
        #("FacilityID", random.choice(XYZ) ),
        ##("deviceId", random.choice(XYZ) ),
        #("CountType", random.choice(XYZ) ),
    ]
def values(t, sqn):
    #STATUS_OPTS[0] = rotate(STATUS_OPTS[0],1)
    return [
        #("autoThingType", "KOHLS" ),
        #("zone", random.choice(ZONE_OPTS) ),
        #("ean", random.choice(RANDOM_OPTS) ),
    ]
def generateAliasConfig(serial, sqn):
    return None
    #a = [serial+"X01", serial+"X02"]
    #a = [ '"%s"'%x for x in a ]
    #return "[" + ",".join(a) + "]"
THE_TIMES = dict()
def long_to_time(t):
    d=datetime.datetime.utcfromtimestamp(t/1000)
    d=d.replace(tzinfo=datetime.timezone.utc).isoformat()
    d=d[:len("2018-02-23T15:12:01.976")]+"Z"
    if  '+' in d:
        #print("ALERT %d" % t)
        return d[:len("2018-11-27T13:09:35")] + ".000Z"
    return d
def get_new_time(serial):
```

```
    if serial in THE_TIMES:
        (minv, maxv, vset) = THE_TIMES[serial]
    else:
        (minv, maxv, vset) = ( 10**20, -10**20, set() )
    t = None
    #
    if (len(vset) >= 2) and  (len(vset) < maxv - minv + 1) and  (random.randint(1,100)
<= OLDER_CHANCE):
        try:
            bad = True
            while bad:
                t = random.randint( minv, maxv - 1)
                bad = (t in vset )
        except:
            print('*')
            t = None
    if t == None:
        t = max(int(round(time.time() * 1000)), maxv + 1)
    vset.add(t)
    minv = min(minv, t)
    maxv = max(maxv, t)
    THE_TIMES[serial] = (minv, maxv, vset)
    return t
def value2json(s):
    if isinstance(s,int):
        return str(s)
    elif s == None:
        return "null"
    else:
        return '"%s"' % s
def blink(serial, sqn):
    t = get_new_time(serial)
    TIME = long_to_time(t)
    v = values(TIME, sqn)
    properties = ','.join( '"%s":{"value": %s,"time":"%s"}' %
(v[0],value2json(v[1]),TIME) for v in v )
    aliasConfig = generateAliasConfig(serial, sqn)
    #"aliasConfig" : %s,
    MSG='''{
        "bridgeCode": "%s",
        "serialNumber": "%s",
        "sqn" : "%s",
        "time": "%s",
        "priority": %d,
        "thingTypeCode": "%s",
```

```
        "udfs": {%s}
    }''' % (BRIDGECODE, serial,  str(sqn),  TIME, pick_priority(), THINGTYPE,
properties)
    MSG=MSG.replace("\n"," ")
    MSG=MSG.replace("_TENANT",TENANT)
    #KEY="%s-%s" % (BRIDGECODE, THINGTYPE)
    KEY=None
    return (TIME, KEY,MSG, " ".join(str(x[1]) for x in v) )
JOBS = [ queue.Queue() for i in range(THREADS) ]
time0 = time.time()
def serial_generator():
    ctime = -1.0
    i = FROM
    for j in range(N):
        t = (i + OFFSET) % THREADS #thread number for the job
        serial = "%s%09d" % (PREFIX, i + OFFSET)
        i += 1
        if i > TO:
            i = FROM
        ctime = time0 + (1.0 / RATE) * j
        JOBS[t].put( (ctime , serial, j) )

LOCK = threading.Lock()
COUNTER = [0]

def thread_fun(i):
    q = JOBS[i]
    while not q.empty():
        (wtime, serial, sqn) = q.get()
        time.sleep( max(0,wtime - time.time() ) )
        (blinktime, key, msg, value) = blink(serial,sqn)
        future = producer.send(TOPIC, key=(None if key == None else key.encode()),
value=msg.encode() )
        ctime = time.time()
        with LOCK:
            COUNTER[0] += 1
            real_counter = COUNTER[0]
        real_rate = real_counter / max(ctime - time0, 1)
        print( "%02d %s %d %s %.02f %s" % (i, blinktime, real_counter ,serial,
real_rate,value) )

current_milli_time = lambda: int(round(time.time() * 1000))
PRODUCER = threading.Thread(target=serial_generator)
def closure(i):
    return lambda : thread_fun(i)
```

```
WORKERS = [ threading.Thread(target= closure(i) )  for i in range(THREADS) ]
PRODUCER.start()
for t in WORKERS:
    t.start()
PRODUCER.join()
for t in WORKERS:
    t.join()
# flush ensures all messages are sent
producer.flush()
producer.close()
```

ANEXOS

SECUENCIA DE PASOS PARA LA EJECUCIÓN DE PRUEBAS DE COMPLEJIDAD COMPUTACIONAL

1. Se deben instalar componentes necesarios para correr la aplicación en streaming y aplicación tradicional en una infraestructura de servidores como el descrito en el modelo de despliegue, punto 4.1.3. Por motivos de seguridad no se incluyen los instaladores de los componentes en esta tesis.
2. Las herramientas de monitoreo utilizadas con Prometheus y Grafana, aplicaciones open source que permiten ver gráficamente los indicadores desarrollados. Los componentes de la aplicación desarrollada para el sector retail expone las métricas necesarias para medir el uso de recursos computacionales, además del "rate" y "lag" del software.
3. Para el cálculo del rate se debe crear un gráfico de tipo lineal en grafana con la siguiente fórmula:

```
sum by (topic) (
        rate (kafka_consumergroup_current_offset{
                consumergroup=~"streamProcessor(.*)"
        }[2m])
)
```

4. Para el cálculo del lag se debe crear un gráfico de tipo lineal con la siguiente fórmula:

   Y esta fórmula para el cálculo del lag:

```
sum by (topic) (
        kafka_consumergroup_lag{consumergroup=~"streamProcessor(.*)"}
```

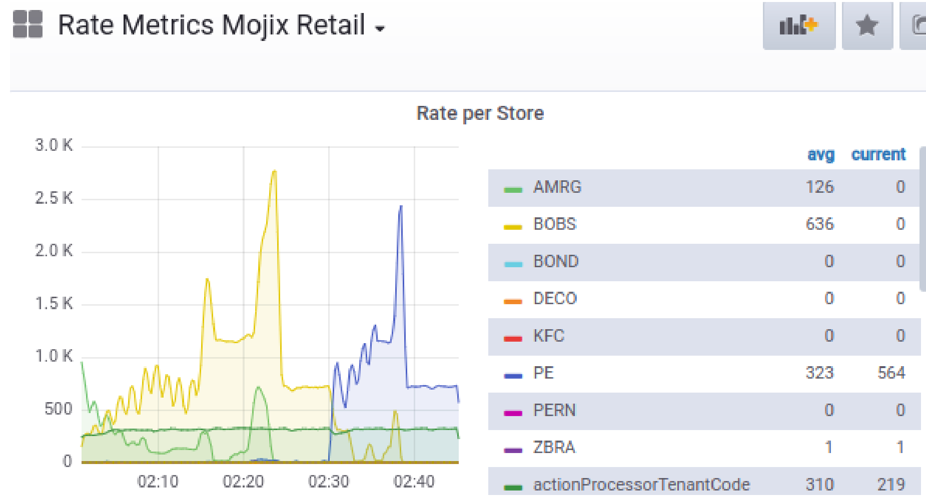
```
)
```

5. Una vez preparada la aplicación y las herramientas de monitoreo se debe proceder a configurar el script de generación de datos para pruebas de carga del anterior anexo a través de la variable “RATE” en las primeras líneas del script. El valor que muestra el script actualmente es del 100, que puede ser modificado a discreción. En caso de que no se pueda generar el rate de simulación deseado, se debe incrementar el valor de la variable “THREADS” o incrementar la capacidad de procesamiento del servidor de pruebas.
6. Se debe configurar correctamente la conexión al servidor, en este caso la conexión al servidor se describe en las primeras líneas del script en la variable “bootstrap_servers”.
7. Ahora se crean los casos de prueba de carga como los siguientes:
    a. Caso 1: 200 mensajes por segundo
    b. Caso 2: 400 mensajes por segundo
    c. Caso 3: 800 mensajes por segundo
    d. Caso 4: 1600 mensajes por segundo

   Se deben modificar los casos a los más adecuados según el criterio del encargado de pruebas.
8. Se guarda el script con un nombre a elección. Ejemplo: “load_test.py”
9. Se ejecuta el script con

   $ python3 load_test.py

10. Ahora en grafana se podrán ver en tiempo real el uso de recursos computacionales de la aplicación para la prueba enviada. Las gráficas obtenidas son parecidas a la siguiente:



11. Se anotan los resultados y se procede a la comparación con los diferentes casos y con pruebas anteriores.

BIBLIOGRAFÍA

Hochreiner, C., Vogler, M., Schulte, S. & Dustdar, S. (2016, Julio 2). Elastic Stream Processing for the Internet of Things. *2016 IEEE 9th International Conference on Cloud Computing, 9, 8.*

Oracle Corporation. (2008, Noviembre). Complex Event Processing Performance. *Oracle Corporation World Headquarters 500 Oracle Parkway Redwood Shores U.S.A*., p.16.

Gregory, J. (2015). The Internet of Things : Revolutionizing the Retail Industry. *Accenture, p.8.*

Qiang, G. , Xinhe, X. (2014, July 4). The Analysis and Research on Computational Complexity. *The 26th Chinese Control and Decision Conference, 1, 6.*

Sipser, M. (2013). *Introduction to the theory of computation*. Boston, MA: Cengage Learning.

Balazinska, M., Stonebraker, M., Çetintemel, U. & Zdonik, S. (2005, December 4). The 8 Requirements of Real-Time Stream Processing . *ACM SIGMOD Record, 34, pp.42-47*

Shusen, Y. (2017, Agosto 8). IoT Stream Processing and Analytics in the Fog. *IEEE Communications Magazine*, 55, 8. pp.21-27.

Doug, D. (2016, Febrero). The Future of Retail through the internet of things. *Beecham Research*, 1, pp.22.

Jayaram, A. (2017). Smart Retail 4.0 IoT Consumer Retailer Model for Retail Intelligence and Strategic Marketing of In-store Products.

Bossaerts, P., & Murawski, C. (2017, December 1). Computational Complexity and Human Decision-Making. Trends in Cognitive Sciences, Vol. 21, pp. 917–929. https://doi.org/10.1016/j.tics.2017.09.005

Casado, R., & Younas, M. (2014). Emerging trends and technologies in big data processing. Concurrency and Computation: Practice and Experience, 27(8), 2078–2091. doi:10.1002/cpe.3398

Gao, Q., & Xu, X. (2014). The analysis and research on computational complexity. 26th Chinese Control and Decision Conference, CCDC 2014, 3467–3472. https://doi.org/10.1109/CCDC.2014.6852777

Teruel, K. P., Yelandi, M., Vásquez, L., Karel, I., Cedeño, F., Jimenez, S. V., & Mustelier, I. D. (2012). Modelo matemático y procedimiento para evaluación por complejidad de los requisitos software . Wer, (March)

Yang, S. (2017). IoT Stream Processing and Analytics in the Fog. IEEE Communications Magazine, 55(8), 21–27. https://doi.org/10.1109/MCOM.2017.1600840

Pantano, E., & Timmermans, H. (2014). What is smart for retailing ? *Procedia Environmental Sciences*, *22*, 101–107. https://doi.org/10.1016/j.proenv.2014.11.010

Chen, C. (2014). *RFID-based intelligent shopping environment : a comprehensive evaluation framework with neural computing approach*. (250). https://doi.org/10.1007/s00521-014-1652-7

Balaji, M. S., & Roy, S. K. (2016). *Value co-creation with Internet of things technology in the retail industry*. *1376*(September). https://doi.org/10.1080/0267257X.2016.1217914

Gao, L., & Bai, X. (2014). *An empirical study on continuance intention of mobile social networking services network externalities and flow theory*. *2013*. https://doi.org/10.1108/APJML-07-2013-0086

Evanschitzky, H., Iyer, G. R., Pillai, K. G., Kenning, P., & Schütte, R. (2014). *Consumer Trial , Continuous Use , and Economic Benefits of a Retail Service Innovation : The Case of the Personal Shopping Assistant* *. *44*(1), 1–17. https://doi.org/10.1111/jpim.12241

Chang, Y. P., Chen, S., & Lee, Y. (2016). *Fog computing node system software architecture and potential applications for NB-IoT industry*. (Iii), 2–5. https://doi.org/10.1109/ICS.2016.149

Kraijak, S., & Tuwanut, P. (2015). *A survey on the internet of things, protocols, possible applications, security, privacy, real world implementations and future trends,* 26–31. https://doi.org/10.1109/ICCT.2015.7399787

Kruchten, P. (2006). *Planos Arquitectónicos : El Modelo de " 4 + 1 " Vistas de la La Arquitectura del Software ´. 12*(6), 1–16.

Karimov, J., Rabl, T., & Katsifodimos, A., Samarev R., Heiskanen H., Markl V. (2019). Benchmarking Distributed Stream Data Processing Systems.

Confluent Documentation (2019) KSQL and Kafka Streams. Revisado en Octubre de 2019. https://docs.confluent.io/current/streams/index.html

Apache Kafka Documentation (2019) Kafka 2.0. Revisado en Octubre de 2019. https://kafka.apache.org/20/documentation/

Google Cloud Platform (2019) All Pricing. Consultado en Diciembre de 2019. https://cloud.google.com/compute/all-pricing