Modelo de Computación en Streaming para Reducir la Complejidad Computacional del Software

Victor Alfonso Ramos Huarachi
Postgrado en Informática
Universidad Mayor de San Andrés
La Paz – Bolivia
victor.ramos.h@gmail.com

***Resumen*—La complejidad computacional de un software está determinada por el uso de recursos que utiliza para llevar a cabo una determinada tarea, esto quiere decir cuánto CPU y RAM consume. En situaciones donde la cantidad de datos que se tienen que procesar es de gran tamaño y continua, la computación en streaming es ideal para reducir la complejidad computacional que supone su procesamiento. Una aplicación de procesamiento en streaming no solo permite procesar grandes cantidades de datos, sino que lo hace a gran velocidad y en tiempo real, son aplicaciones escalables, altamente disponibles y tolerantes a fallos.**

Palabras clave—stream, streaming, complejidad computacional, Kafka

I. Introducción

Al hablar de grandes cantidades de información, una de las primeras cosas que se nos viene a la mente es big data que se puede definir como una colección de datos tan grande que las bases de datos convencionales no pueden procesarlas en el tiempo deseado [1] y en la actualidad desarrollar sistemas capaces de procesar esta data se ha vuelto una necesidad que ha traído nuevos retos tecnológicos a las empresas, retos que consisten en crear software cuidando que la solución no sea tan compleja que implique que el costo computacional sea superior a los beneficios obtenidos.

II. Estado del arte

Las principales estrategias de procesamiento de big data utilizadas actualmente son:

1) Procesamiento en Paralelo: Se crean varios hilos de ejecución aprovechando de mejor manera los núcleos del procesador. Su principal desventaja es la facilidad de perder el control del código y los problemas de concurrencia.

2) Procesamiento por Lotes: Se procesa un gran conjunto de datos en una sola operación, reduciendo la latencia producida por la red y/o escritura en disco. Su principal desventaja es que necesita la memoria suficiente para almacenar todos los datos mientras son procesados, una vez liberada la memoria, el sistema queda en un estado de reposo "desperdiciando" recursos.

3) Procesamiento en Streaming: La estrategia consiste en recoger pequeños lotes de datos en intervalos de tiempo para ahorrar memoria y procesarlos casi inmediatamente. El principal problema es que la probabilidad de fallos es alta y se tiene que pensar en una consistencia eventual.

4) Procesamiento híbrido o arquitectura lambda: Se consulta por lotes y se procesa los datos con streaming combinando lo mejor de ambos tipos de procesamiento. Su desventaja es el desfase que existe al consultar información mientras la data está siendo procesada.

III. Complejidad computacional

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 [2].

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 [3]. De un problema del cual se conoce una solución, saber si es posible ser resuelto se encarga la teoría de la complejidad computacional. Estas mediciones se realizan tomando en cuenta las dimensiones de espacio y tiempo.

A. Complejidad en tiempo

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.

B. Complejidad en espacio

La complejidad de espacio en computación se refiere a los recursos de hardware, memoria RAM o disco. Para problemas deterministas el espacio requerido es lineal, mientras que para problemas no deterministas el espacio requerido podría ser polinomial o exponencial.

IV. 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 [4].

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.

Para referenciar este artículo (IEEE):

[N] V. Ramos, «Modelo de Computación en Streaming para Reducir la Complejidad Computacional del Software», *Revista PGI. Investigación, Ciencia y Tecnología en Informática*, nº 8, pp. 196-200, 2020.

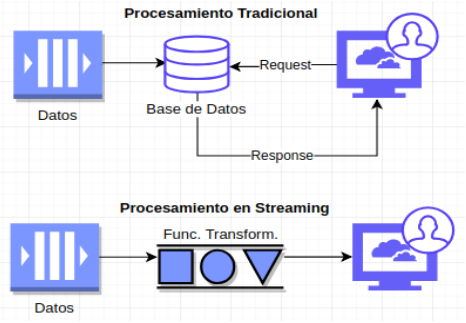

Fig. 1. Procesamiento Tradicional vs Streaming

Los frameworks que permiten la computación en streaming más populares nacieron con Apache como ser Storm, Flink, Spark, Kafka Streams, Samza y Flume.

Todos los frameworks comparten conceptos importantes como ser el particionamiento, que consiste en dividir los datos en distintas colas de procesamiento para luego distribuirlas en tareas. Estas colas generalmente están alojadas en un sistema colas de mensajes conocidos como brokers.

La principal diferencia entre los frameworks es que algunos tienen un clúster de procesadores y las aplicaciones corren en el clúster, en cambio otros solo necesitan utilizar librerías dentro del propio código fuente de la aplicación para ya comenzar a procesar en streaming.

V. Modelo de Streaming

El modelo propuesto está basado en el framework de Kafka Streams debido a que tiene un alto grado de madurez, documentación, tiene una comunidad activa, es capaz de procesar millones de mensajes por segundo, se integra con otros tipos de sistemas y tiene varias aplicaciones en la creación de microservicios.

A. *Tópicos*

La principal abstracción del procesador de streaming son los tópicos. Un tópico es una categoría de datos bien definida dividida en particiones, donde cada partición es una secuencia ordenada de registros los cuales se publican por clientes llamados productores y se procesan por clientes llamados consumidores.

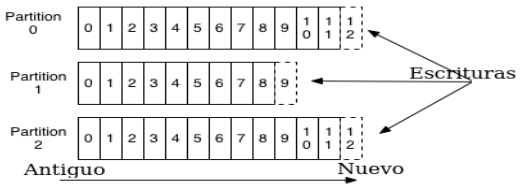

Fig. 2. Anatomía de un tópico [5]

La partición es la unidad de paralelismo de un procesador de streaming.

B. *Paralelismo*

En streaming se inician varios hilos de procesamiento donde cada uno procesa una o varias particiones de un tópico, 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.

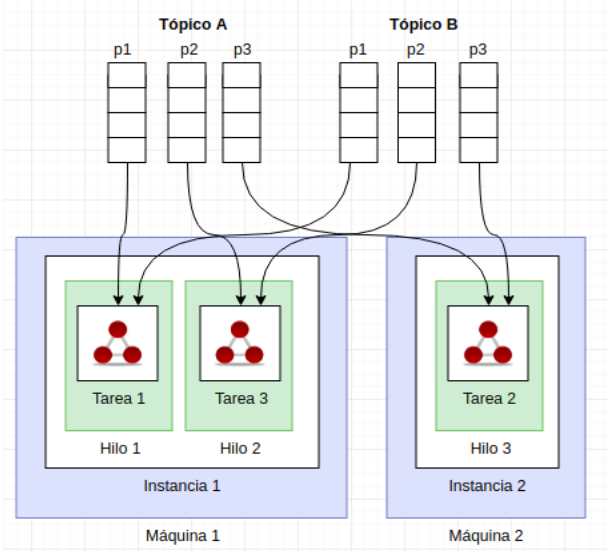

Fig. 3. Escalamiento de una aplicación en streaming [6]

C. *Tolerancia a Fallos*

La tolerancia a fallos de una aplicación en streaming está determinada por la replicación de las particiones en el clúster.

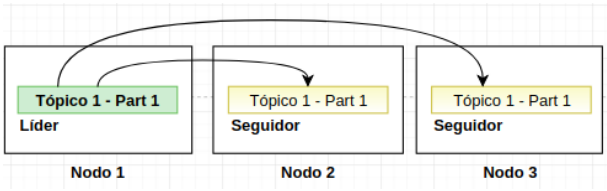

Fig. 4. Replicación de una partición

Si una máquina del clúster falla otra máquina que tenga el dato replicado se encargará de seguir procesando el dato. 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.

D. *Arquitectura de Software*

Para describir la arquitectura de software se partirá del modelo de Vistas de Arquitectura 4+1 propuesta por Philippe Kruchten [7] ajustada a los requerimientos de un software basado en la computación en streaming.

1) Arquitectura lógica: El procesador de streaming es capaz de procesar un flujo de datos continuo en tiempo real y lógicamente es solo una instancia de la aplicación, la cual puede adicionarse o removerse en cualquier momento, en caso de adicionarse una nueva instancia las tareas de balancean con el nuevo componente y en caso de removerse las tareas se asignan a una instancia de este libre.

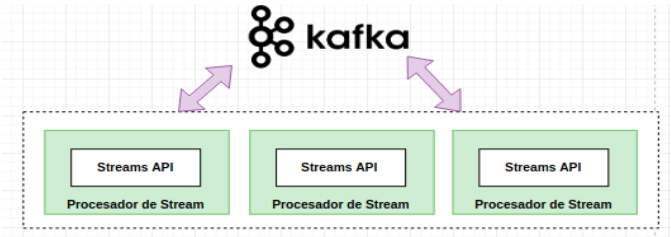

Fig. 5. Replicación de una partición

La vista lógica es sencilla, una aplicación que utiliza Kafka Streams sigue siendo una aplicación normal, pero al incluir las librerías de Kafka Streams ya posee capacidades de escalabilidad, disponibilidad y tolerancia a fallos.

2) Requisitos No Funcionales

- Escalabilidad, que está determinada por la cantidad de particiones de un tópico.
- Escalabilidad, se pueden levantar varias instancias de una aplicación a demanda.
- Rendimiento, definida por el nivel de paralelismo que está limitado por la cantidad de particiones de un tópico.
- Disponibilidad, sujeto al factor de replicación de una partición en un clúster.
- Tolerancia a fallos, que se obtiene gracias al clúster ya que cuando un nodo cae, otro toma su lugar.
- Seguridad, proveído por el framework como ser el cifrado, la autenticación y listas de acceso ACL.

3) Componentes de Software: La arquitectura de software planteada utiliza 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.

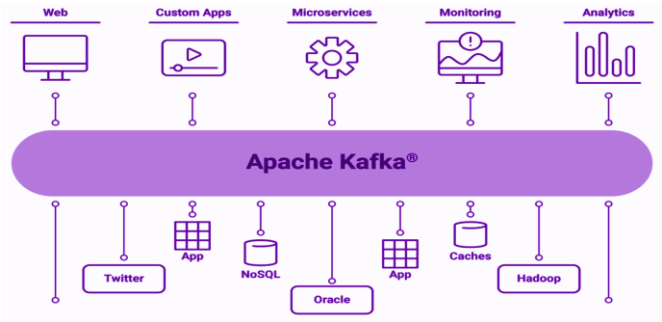

Fig. 6. Bus de Eventos basado en Kafka

Este modelo simplifica los canales de comunicación entre componentes y ofrece los beneficios de la computación en streaming.

4) Arquitectura de Despliegue: Para el modelo de despliegue se consideran los siguientes aspectos:

- Los componentes de software se encapsulan en containers usando docker como tecnología.
- Los containers se orquestan en el clúster de servidores utilizando Kubernetes.

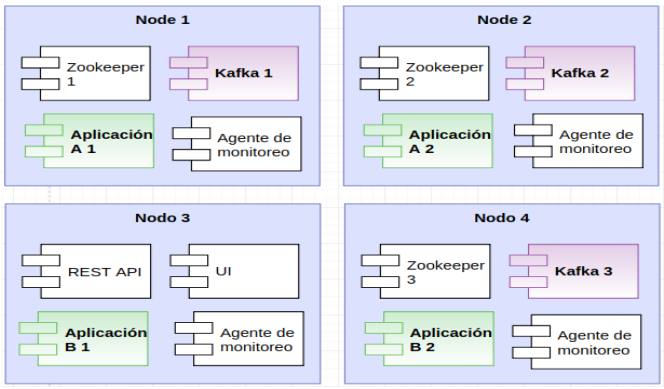

Fig. 7. Diagrama de Despliegue

Los componentes en streaming son pequeños comparados a un sistema monolíticos, pero trabajan como una sola unidad de procesamiento.

VI. Validación del Modelo

El modelo propuesto fue aplicado en sistemas que requieren gran capacidad de procesamiento, por el orden de miles de transacciones por segundo.

A. *Entrada de datos*

Se escribieron scripts de simulación de carga y se prepararon pruebas para estresar el sistema. Los datos se enviaron durante media hora de la siguiente forma:

TABLA I. Pruebas

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	3	2
10	2000	50	4	4

Los objetivos de las pruebas fueron los siguientes:

- Comprobar la linealidad del paralelismo, esto quiere decir que dos instancias de una aplicación duplican la capacidad de procesamiento de una.
- Comprobar el impacto del tamaño de un mensaje.
- Tomar métricas de 'rate' y de 'lag', donde rate se define como la cantidad de mensajes procesados por segundo y lag como la cantidad de mensajes en espera.
- Tomar métricas de uso de CPU y memoria RAM por instancia de la aplicación.

Las pruebas están orientadas a medir la complejidad computacional en tiempo y espacio, es decir, el rendimiento de la aplicación y el uso de recursos para procesar dicha tarea.

B. *Resultados en cuanto a rendimiento*

El rendimiento de la aplicación se obtiene de las métricas de rate y lag obtenido por las herramientas de monitoreo.

TABLA II. Resultados de Rendimiento

Caso	Rate Promedio	Lag Máximo
1	250	380
2	500	910
3	1000	2,100
4	1220	1,070,000
5	2000	3,200
6	3000	8,200
7	750	405,000
8	985	54,000
9	1000	1,900
10	1880	21,000

El caso 4 muestra que para procesar mensajes de 10 KB se tiene un rate máximo de 1220 mensajes procesados por segundo

por instancia de 4 hilos de ejecución. Las pruebas 1, 2 y 3 al recibir menor carga pueden procesar al mismo rate que la entrada de datos.

La prueba 5 y 6 muestra que dos instancias de la aplicación son capaces de procesar 2000 y 3000 mensajes por segundo gracias al paralelismo de una aplicación en streaming.

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, pero como se verá en los siguientes resultados es debido al factor de ancho de banda.

C. *Resultados en cuanto a escalabilidad*

La prueba realizada para verificar la escalabilidad consiste en incrementar gradualmente el número de hilos para verificar el rate de procesamiento por segundo. Después se incrementan instancias de la aplicación en otro servidor para verificar la escalabilidad en un clúster.

TABLA III. RESULTADOS DE ESCALABILIDAD

Nro. de Instancias	Nro. de Hilos por Instancia	Rate
1	1	310
1	2	602
1	3	925
1	4	1190
2	1	595
2	2	1175
2	3	1870
2	4	2200
3	1	910
3	2	1905

Los resultados demuestran que al incrementarse el número de hilos de una aplicación en streaming, la tasa de procesamiento casi se duplica obteniendo un resultado lineal que obedece a la fórmula “rate = número de hilos * rate base”.

D. *Resultados en cuanto a tolerancia a fallos*

Gracias al uso del framework de Kafka Streams se ha podido comprobar que una aplicación en streaming es capaz de recuperarse si algún nodo del clúster falla tras un proceso llamado “rebalanceo” el cual se encarga de reasignar las particiones de un tópico a las instancias disponibles.

E. *Resultados en cuanto a uso de recursos*

Se han instalado agentes de monitoreo en cada uno de los miembros del clúster donde se ejecutan las aplicaciones en streaming. Las muestras se tomaron durante la ejecución de las anteriores pruebas y los resultados muestran los siguiente:

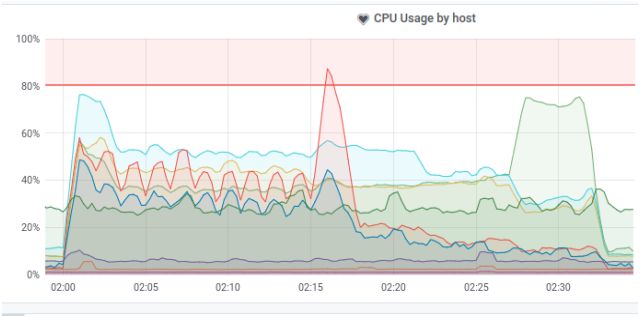

Fig. 8. Uso de CPU

El uso de CPU de las aplicaciones es normal y no llega a superar el 90% de uso.

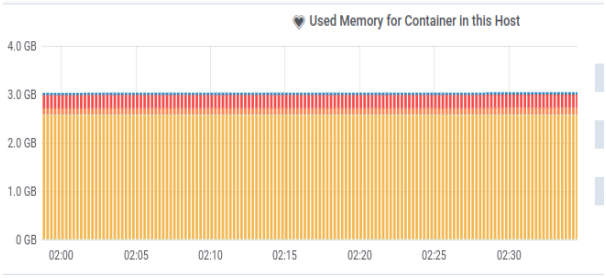

Fig. 9. Uso de Memoria RAM

El uso de memoria de las aplicaciones es aproximadamente 3 GB RAM, las aplicaciones están configuradas para no superar los 4 GB de RAM lo cual demuestra que la complejidad de espacio se ha visto reducida.

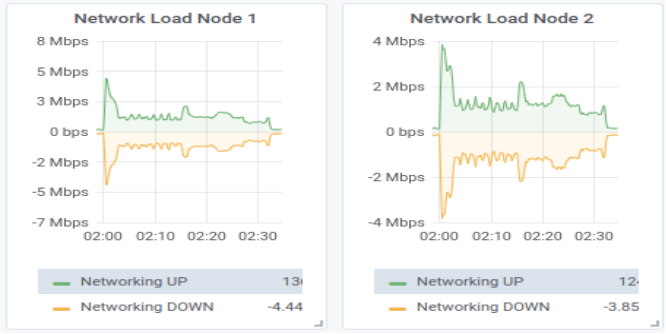

Fig. 10. Ancho de Banda usado

Por último, el ancho de banda utilizado es inferior al que se dispone en los ambientes en cada uno de los nodos del clúster, en la figura 10 muestra un aproximado de 3 Gbps de 10 Gbps disponibles.

VII. CONCLUSIONES

Los resultados de la aplicación de un modelo de computación en streaming a una aplicación que requiere procesar grandes cantidades de datos son:

- La computación en streaming reduce la complejidad de tiempo de una aplicación, puesto que se comprobó que el rendimiento en mensajes procesados por segundo es mucho mayor que un sistema de arquitectura tradicional.
- Una aplicación en streaming es escalable y permite la mejora del rendimiento de manera lineal. La escalabilidad está determinada por la cantidad de hilos de procesamiento y a su vez por la cantidad de particiones de un tópico.
- Una aplicación en streaming es tolerante a fallos, esto gracias al uso del framework de Kafka Streams que tiene mecanismos de rebalanceo de carga si algún nodo se añade o se quita del clúster.
- La complejidad computacional de espacio se reduce porque al procesar los datos en la medida que estos se van generando, elimina la necesidad de contar con grandes cantidades de memoria por el procesamiento de lotes de datos, al contrario, solo se necesita lo suficiente para procesar un pequeño lote de información el cual se va procesando en intervalos pequeños de tiempo.

REFERENCIAS

[1] Casado, R., & Younas, M. (2014). Emerging trends and technologies in big data processing. Concurrency and Computation: Practice and Experience, 27(8), 2078–2091.

[2] Gao, Q., & Xu, X. (2014). The analysis and research on computational complexity. 26th Chinese Control and Decision Conference, CCDC 2014, 3467–3472.

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

[4] 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

[5] (2019) Apache Kafka website [Online]. Available: https://kafka.apache.org/documentation/#introduction

[6] (2192) Confluent website. [Online]. Available: https://docs.confluent.io/current/

[7] Kruchten, P. (2006). Planos Arquitectónicos: El Modelo de "4 + 1"

Breve CV del autor

Victor Alfononso Ramos Huarachi es Ingeniero Informático por la Universidad Autónoma Juan Misael Saracho de Tarija; certificado Desarrollador Apache Kafka por Confluent con Credencial Nro. 12176372. Actualmente realiza la maestría en Alta Gerencia en TICs e Innovación para el Desarrollo MAGTIC en el Postgrado en Informática de la Universidad Mayor de San Andrés. Ejerce profesionalmente como Developer Architect en Mojix Inc, con base en Los Angeles - Estados Unidos, anteriormente como Profesional de Desarrollo de Sistemas en Aduana Nacional de Bolivia y la Agencia Nacional de Hidrocarburos. Es co-organizador del Java User Group de Bolivia y actualmente está interesado en la construcción de aplicaciones modernas de procesamiento en tiempo real, arquitecturas de software dirigidas por eventos, computación en streaming y analytics en tiempo real.
Email: victor.ramos.h@gmail.com.