



# Big Data Streaming - Kafka

Texto base: Juan Esquivel Rodríguez  
Luis Alexander Calvo Valverde



# Streaming

- Técnica de procesamiento en tiempo real
- Comúnmente basado en colas de procesamiento
  - Productores: Agregan datos a la cola
  - Consumidores: Extraen datos de la cola
- Conceptualmente son bitácoras sin límite con procesamiento continuo
- Garantías y rendimiento varían según tecnología



# Apache Kafka

- Proyecto originado en LinkedIn con los siguientes objetivos
  - Bajo retardo
  - Manejo de grandes volúmenes de mensajes
  - Reutilizar ideas de frameworks establecidos
- Diferenciadores
  - Garantías de entrega
  - Enfoque en ancho de banda
  - Naturaleza distribuida
  - Política de retención de mensajes



# Diferenciadores - Garantías de Entrega

- Alta certeza requiere sistemas altamente complejos
- El caso de uso que generó Kafka no requería alta certeza de entrega
- Los autores pueden cumplir con sus requerimientos aún si se pierden algunos mensajes
- Kafka decide no enfocarse en garantías de entrega
- Qué pasa si el caso de uso requiere altas garantías?
  - Responsabilidad de desarrolladores agregar capas adicionales
  - O no usar Kafka!



## Diferenciadores - Ancho de banda

- Diseñado para enviar y recibir grandes cantidades de datos al mismo tiempo
- Optimizar el total de transmisiones TCP/IP
- Evitar "overhead" asociado a múltiples transmisiones pequeñas
- Tratar de enviar tantos mensajes en una sola transmisión como sea posible



# Diferenciadores - Naturaleza distribuida

- Descentraliza el almacenamiento
- Se introduce el concepto de particiones desde el diseño
- El almacenamiento está altamente acoplado a dichas particiones

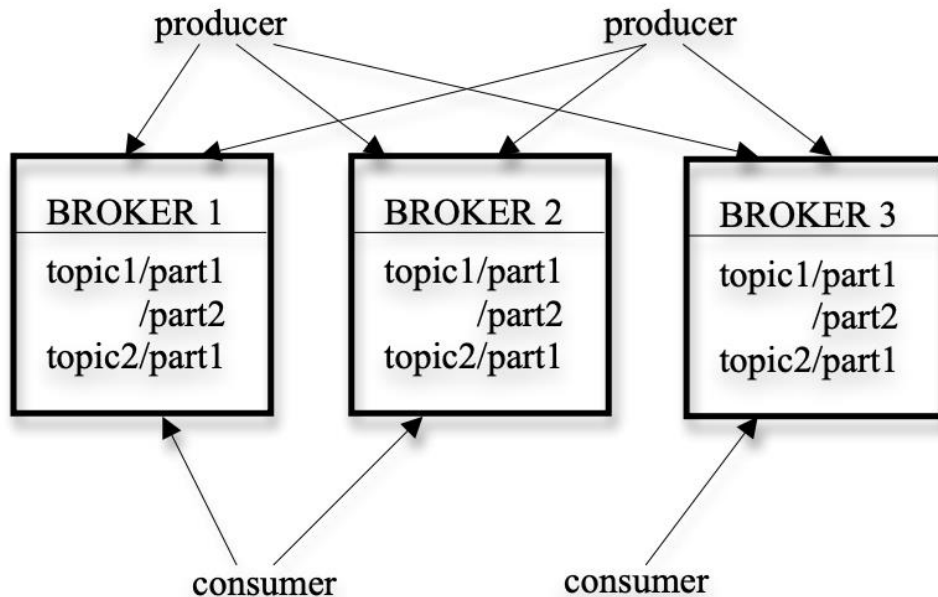


# Diferenciadores - Retención de Mensajes

- Otros manejadores de colas poseen buffers pequeños
- Se asume que consumidores extraen mensajes rápidamente
- Desventajas de buffers pequeños
  - Recuperación de fallos tiene ventana de oportunidad más pequeña
  - Función de cola como bitácora de actividad reciente se vuelve limitada
- Kafka se diseña para proveer acceso a los mensajes en cola por tiempos más prolongados
  - Hasta 7 días
  - Alta flexibilidad para que consumidores soliciten datos basados en sus propios puntos de control
  - Facilita la labor de los servidores

# Arquitectura

- Similar a otros sistemas basados en productores y consumidores







# Arquitectura - Otros componentes

- Topic
  - Define un flujo de mensajes (stream)
  - No hay restricción sobre qué puede ser un topic
  - Es posible definir cualquier cantidad de ellos por servidor
- Particiones
  - Cada topic puede dividirse en muchas secciones
  - Conceptualmente, cada partición es una sección del espacio de todos los mensajes
  - Su objetivo primario es balancear carga
- Broker
  - Servidor Kafka
  - Permite escritura y lectura de alguno (o todas) las particiones de un topic



## Consecuencias del diseño

- Es posible tener la misma partición en muchos brokers
- No es necesario tener todas las particiones en todos los brokers
- Productores y consumidores pueden comunicarse con múltiples brokers
- La comunicación no queda completamente encapsulada por el framework
- Los clientes deben configurar explícitamente múltiples servidores



# Distribución de mensajes

- Kafka particiona cada topic
  - Mediante el uso de llaves y funciones de partición
  - Se asigna los mensajes a diferentes particiones, dentro de un topic
- Consumidores recorren todas las particiones para consumir todos los datos
- Se organizan mediante grupos de consumidores (consumer groups)
  - Un grupo de consumidores distribuye las particiones equitativamente (en el caso ideal) entre cada consumidor individual
  - Idealmente, se tiene muchas más particiones que consumidores en el grupo (no se puede paralelizar una partición)
  - Cada mensaje sólo se envía a un consumidor dentro del grupo
  - Si hay más de un grupo de consumidores, cada mensaje será enviado una vez a cada grupo



# Coordinación descentralizada

- Kafka no usa un nodo maestro para coordinar las lecturas
- Los consumidores deben coordinar entre sí mientras procesan los datos de un topic
- **Zookeeper:** servicio que permite administrar jerárquicamente configuraciones de consenso mediante 4 operaciones
  - Crear una ruta
  - Leer el valor de una ruta
  - Borrar una ruta
  - Ver los hijos de una ruta
- Kafka utiliza las operaciones para
  - Detectar la creación o eliminación de brokers y consumidores
  - Iniciar un proceso de rebalanceo cuando cambian los nodos
  - Actualizar la información de consumo y desplazamientos de cada partición



## Garantías de entrega en detalle

- Kafka garantiza que los mensajes se entregan al menos una vez
  - Si existen errores y un proceso diferente debe encargarse de mensajes pendientes, podría suceder que se envíe más de una vez
  - No cumple con "deliver exactly once semantics"
- Los mensajes que vienen de una sola partición se entregan en orden
  - No hay garantía de orden entre particiones diferentes usadas como fuente
- Existen bits de chequeo para detectar corrupción de datos
- No hay garantías si un broker deja de funcionar y había eventos sin consumir



# Referencias

- Kreps, J; Narkhede, N; Rao, J. Kafka: a Distributed Messaging System for Log Processing. <https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/Kafka.pdf>
- <https://kafka.apache.org/quickstart>
- <https://spark.apache.org/docs/latest/streaming-programming-guide.html>



# Transferencia de datos

- Maximiza la cantidad de transmisiones hechas
  - Acumulación de múltiples mensajes
- La lectura no utiliza cache explícita de Kafka
  - Asume que la infraestructura de cache del sistema de archivos ayuda
  - E.g. páginas utilizadas frecuentemente son mantenidas en memoria
- Según los autores esto tiene varias ventajas:
  - Se evita tener dos buffers de datos
  - La cache puede estar lista aún si el proceso reinicia (cache warming)
  - Reduce la recolección de basura interna al proceso porque en realidad no carga a memoria para reutilizar



# Almacenamiento

- Cada partición debe manejar su copia de los mensajes enviados a Kafka
  - Corresponda a una bitácora completa (por partición)
  - Cada bitácora se divide en segmentos de aproximadamente el mismo tamaño
  - En el orden de los GB en la publicación original
- Los segmentos sólo permiten insertar al final
- Política de materialización
  - Basado en cantidad de mensajes
  - Basado memoria es almacenada
- Los clientes no tienen disponible esos mensajes hasta que se materialice
  - Operación de vaciado (flush)

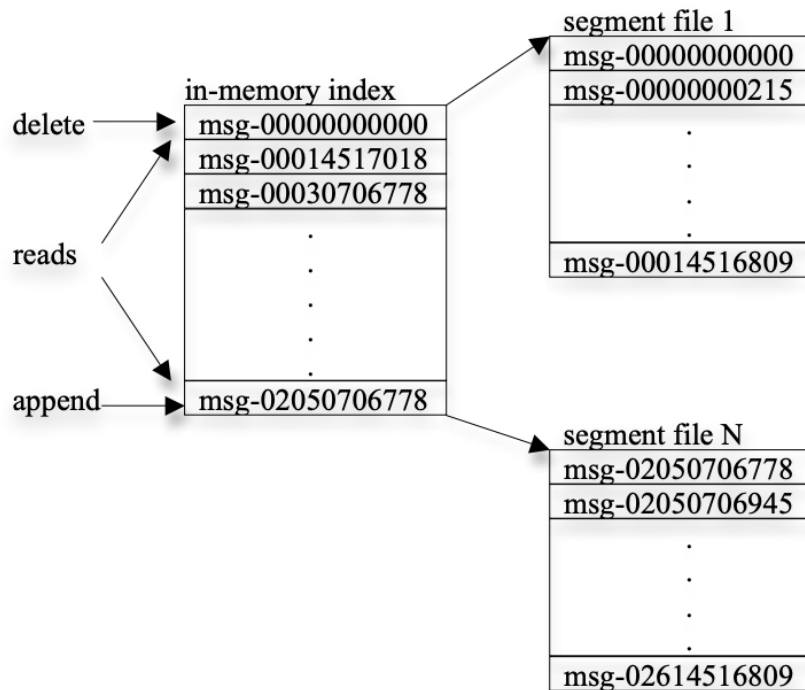




# Manejo de índices

- Kafka considera innecesario asignar un identificador por mensaje
- Se maneja implícitamente
  - Cada mensaje tiene un identificador dado por el desplazamiento en memoria desde el inicio de la bitácora hasta el mensaje
  - Evita tener que obtener los desplazamientos basado en tablas de otros índices.
- Consumidores leen secuencialmente
  - Solicitar un mensaje implica que el cliente ya ha leído todos los mensajes anteriores
  - Útil para llevar una marca del flujo
- Se mantienen las propiedades de incremento monotónico de los identificadores
- Para encontrar el siguiente se toma el identificador más el tamaño del mensaje
- Brokers pueden mantener en memoria una lista ordenada de desplazamientos
  - Podrían ser los inicios de cada archivo de segmento.

# Manejo de índices






## Arquitectura - Producer

```
producer = new Producer (...);
```

```
message = new Message(  
    "test message str".getBytes());
```

```
set = new MessageSet(message);
```

```
producer.send("topic1", set);
```



## Arquitectura - Consumidor

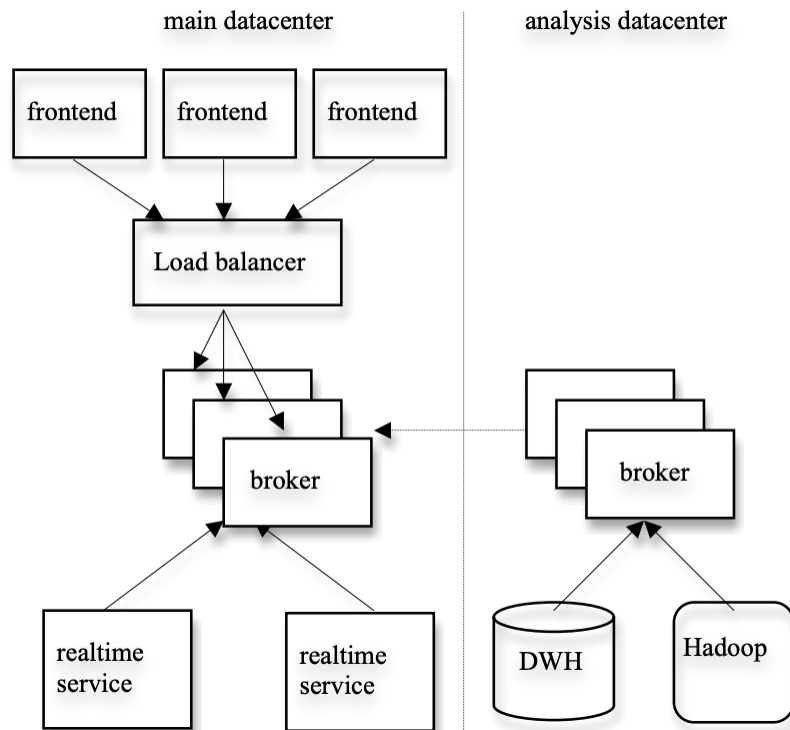
```
streams[] =  
    Consumer.createMessageStreams(  
        "topic1",  
        1  
    )  
for (message : streams[0]) {  
    bytes = message.payload();  
    // procesar  
  
}
```



# Caso de estudio

- Detalles de implementación en LinkedIn
- Clusters Kafka colocalizados con servicios de usuarios
  - Capturan eventos y distribuyen "localmente" a consumidores en tiempo real
- Existen clusters Kafka localizados en otros datacenters sin aplicaciones de usuario en ellos
  - Objetivo es proveer capacidades de análisis con infraestructura Hadoop
  - Estos clusters se alimentan de los clusters Kafka "primarios"
- Al momento de la publicación, los clusters Kafka se reporta que acumulaban cientos de GB de datos por día y una cantidad de mensajes cercano a los mil millones

# Caso de estudio - Arquitectura

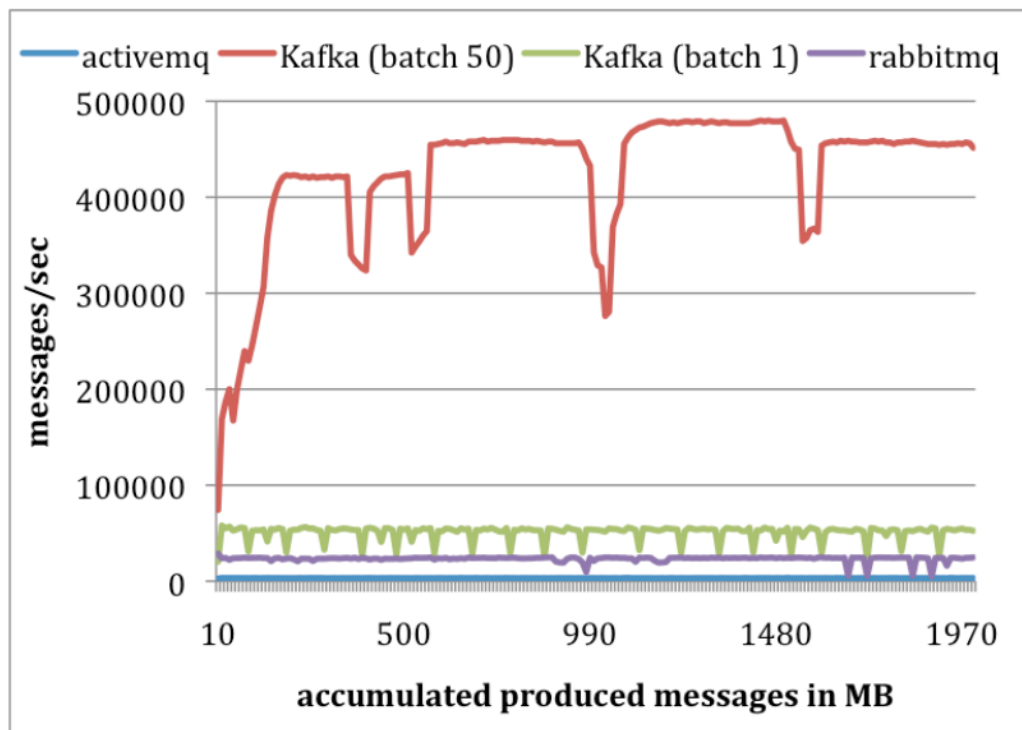




## Caso de estudio - Rendimiento

- Comparación de Kafka con otros sistemas de la época
  - Se utilizó dos máquinas Linux con 8 núcleos de la época, 16GB de memory y 6 discos con RAID 10, conectadas por un enlace de 1 Gigabit
  - Una máquina jugaba el papel de broker y la otra de consumer.
- En el productor se identifican ventajas gracias a la habilidad de hacer envíos grandes
  - Tasas de envío de 50-400 mil mensajes por segundo
- No se espera confirmación del broker
  - Esto puede ser una desventaja por integridad de los datos
  - Se justifica, según los autores, por el tipo de uso
- Se considera que Kafka tiene un almacenamiento más eficiente
  - En otros sistemas hay más *overhead* de bytes que no corresponden a datos

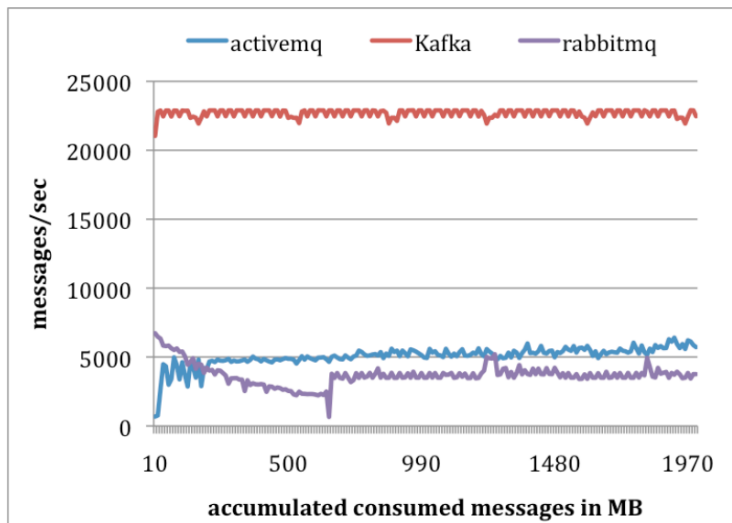
## Caso de estudio - Rendimiento





## Caso de estudio - Rendimiento

- En el consumidor Kafka logró obtener tasas de lectura de 22 mil mensajes por segundo
  - Almacenamiento más eficiente, debido a transmisiones más pequeñas
  - Los otros sistemas deben mantener un control sobre cuáles han sido entregados



# Distribución de carga

**Algorithm 1:** rebalance process for consumer  $C_i$  in group  $G$

For each topic  $T$  that  $C_i$  subscribes to {

remove partitions owned by  $C_i$  from the ownership registry

read the broker and the consumer registries from Zookeeper

compute  $P_T$  = partitions available in all brokers under topic  $T$

compute  $C_T$  = all consumers in  $G$  that subscribe to topic  $T$

sort  $P_T$  and  $C_T$

let  $j$  be the index position of  $C_i$  in  $C_T$  and let  $N = |P_T|/|C_T|$

assign partitions from  $j*N$  to  $(j+1)*N - 1$  in  $P_T$  to consumer  $C_i$

for each assigned partition  $p$  {

set the owner of  $p$  to  $C_i$  in the ownership registry

let  $O_p$  = the offset of partition  $p$  stored in the offset registry

invoke a thread to pull data in partition  $p$  from offset  $O_p$

}

}