Actualización reactiva de bases de datos usando cadenas de procesadores de flujo de datos

Miguel Algorri, José Mª Drake y Marta Zorrilla

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria, Santander (España)
{miguel.algorri,jose.drake,marta.zorrilla}@unican.es

Resumen. Este trabajo en curso explora el uso de cadenas de procesadores de flujos de datos como medio para proporcionar a aplicaciones con requisitos de tiempo real (TR) un acceso a la información del entorno bajo una perspectiva de base de datos (consultas continuas consistentes). En este trabajo se formulan las características que han de ofrecer las cadenas de procesadores de flujos de datos para este caso de uso, se define la arquitectura de procesado a utilizar y se asignan responsabilidades a cada uno de los elementos de la arquitectura.

Palabras clave: Flujos de datos · Base de datos en memoria · Tiempo real · Consultas sobre flujos de datos.

1 Introducción

Las aplicaciones de tiempo real (TR) que operan sobre entornos complejos han de procesar con baja latencia intensos flujos de datos procedentes de él. Ejemplos de estos entornos son aquellos basados en el internet de las cosas (IoT), las redes inteligentes de gestión de la energía, o los sistemas militares con multitud de elementos coordinados a través de redes *wireless*. Todos ellos tienen actualmente una enorme importancia estratégica y, además, representan un desafío a las tecnologías disponibles. En estas aplicaciones interesa modelar el estado del entorno utilizando metodologías y tecnologías de bases de datos (BD), ya que con ello se contribuye a su simplificación, escalabilidad, exportabilidad y facilidad de mantenimiento. Hasta comienzos de esta década [1], las soluciones se buscaban con el diseño de BD con bajas latencias de actualización y lectura y que tuvieran la capacidad de operar con las limitaciones de recursos que son inherentes a los sistemas embebidos. Con el actual cambio tecnológico, esta estrategia está siendo superada por nuevas arquitecturas basadas en cadenas de procesadores de flujos de datos en memoria que ofrecen la información requerida por las aplicaciones sin necesidad de tener que ser previamente persistida. Estas arquitecturas se suelen denominar "*Real-time Data Pipeline*" [2].

Los casos de uso que originariamente han motivado la generación de la tecnología que da soporte a las *RT-Data Pipeline* están relacionados con el comercio electrónico, las redes sociales, el análisis de datos en tiempo real, etc. en los que es más relevante el análisis de grandes volúmenes de datos (*BigData*) que la latencia y su predictibilidad

temporal. En este trabajo, se aborda el montaje de dos *pipelines* a fin de explorar su capacidad y prestaciones en sistemas de TR. Para ello se ha elegido dos tecnologías con enfoques distintos que pueden ser aplicadas a este contexto [7]: *Spark Streaming*, un *Stream Proccessing Engine* (SPE) y *S-Store* [3] el primer *memory-based DBMS* extendido con directivas para soportar *streaming*.

2 Antecedentes

Los primeros sistemas que se enfrentan a la gestión de datos con restricciones de TR datan de los años 80, donde la investigación se centró en extender los gestores relacionales para soportar características de TR [1]. En los 90, se realizaron propuestas sobre arquitecturas orientadas a objetos y se implementaron soluciones que, para satisfacer las restricciones de tiempo, hacían uso de técnicas de BD en memoria. Posteriormente, la investigación se centró en aplicaciones prácticas surgiendo, por ejemplo, los gestores para redes de sensores [4]. La mayoría de estos desarrollos fueron concebidos con una perspectiva centralizada.

En esta década, la necesidad de procesar grandes flujos de datos ha conducido al desarrollo de entornos de procesado directo de flujos con baja latencia, que se basan en arquitecturas de procesamiento distribuido sobre grupos de servidores y BD en memoria [5]. Buscando una transición gradual y segura de la tecnología, inicialmente se ha propuesto arquitecturas híbridas que se han denominado arquitectura *Lambda* [6] en la que se combinaban procesamiento en *batch* de los datos persistidos (p.e. HDFS) el cual proporciona una información consolidada, junto con un segundo procesamiento "al vuelo" de los flujos de datos (p.e. *Spark Streaming*) que proporcionaba una información con muy baja latencia, pero pendiente de consolidar. La fiabilidad de estas tecnologías y la existencia de aplicaciones que solo requieran mantener los valores más recientes o una versión agregada de los mismos, ha inducido una evolución hacia arquitecturas basadas únicamente en tecnología de flujo continuo de datos que se ha denominado arquitectura *Kappa* [5]. En ella, el procesamiento se realiza de forma continua sobre el propio flujo de datos, y sólo se repite la computación de los datos en los transitorios de cambio de la lógica de negocio, o cuando se detectan datos erróneos o fuera de orden (*bounded retention time*).

3 Arquitectura de experimentación

La Fig. 1 muestra la arquitectura tipo *kappa* que se evalúa. El entorno contiene múltiples dispositivos que generan un flujo de datos heterogéneo. Cada tipo de datos está caracterizado por su tipo (*topic*) que describe su estructura interna (*schema*), su caducidad temporal y la forma que ha de ser gestionado por el sistema. Todo el flujo de datos que genera el entorno se integra en un bus conceptual de información (*Enterprise Service Bus* (ESB)), y a través de él, la información generada por el entorno queda accesible a las aplicaciones presentes y futuras que lo gestionan, lo supervisan o lo controlan. Las aplicaciones RT se diseñan con dos módulos, uno destinado a mantener actualizada la vista del modelo del entorno en base a la información que se recibe de él,

y la otra, implementa la funcionalidad de negocio que le es propia. El objetivo del trabajo es evaluar el primer módulo. Este se construye con un *streaming processor*, que se subscribe a la fracción específica del flujo del ESB que es útil a la aplicación, filtra la información que es relevante, la procesa para adaptarla al modelo del entorno que la aplicación espera y, con una estrategia de actuar cuando el dato llega (*push update*), mantiene actualizada la base de datos local que representa la vista del entorno con la que opera la aplicación. En la parte inferior de la Fig. 1 se describen los componentes específicos con los que se implementan las dos configuraciones a evaluar:

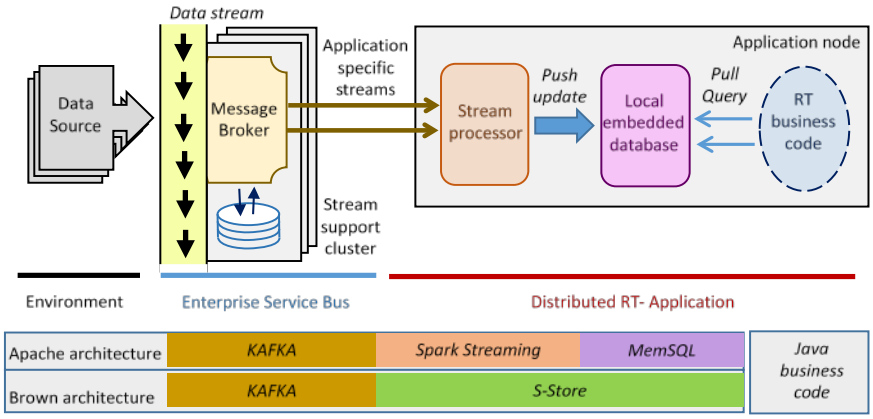

Fig. 1. Arquitecturas de experimentación tipo kappa

- *Enterprise service bus*: En ambas soluciones se implementa mediante el servicio de intermediación de mensajes *Kafka*, plataforma distribuida, de alto rendimiento, de baja latencia y escalable para la manipulación en TR de fuentes de datos. En el diseño de *Kafka* se da una solución unificada a la heterogeneidad inherente de un entorno complejo y del sistema que lo gestiona. *Kafka* ofrece dos características especialmente adecuadas para TR: la transmisión de baja latencia entre la fuente y la aplicación sin requerir almacenamiento, y la facilidad que ofrece para gestionar de forma reactiva datos que llegan fuera de orden, atrasados o caducados.
- Actualización reactiva (*push update*) de la base de datos embebida: Esto supone dos tareas complementarias: 1) seleccionar, filtrar y transformar los datos para que a la aplicación sólo lleguen aquellos que corresponden a la vista del entorno que se ha especificado, y 2) mantener actualizada la base de datos embebida local con la que la aplicación accede asíncronamente y con baja latencia a los datos del entorno. La implementación de esta segunda sección en cada una de las implementaciones es diferente. En la configuración *Apache*, se utiliza *Spark Streaming* que es un potente SPE, con capacidad de ser distribuido y con plena escalabilidad que se complementa con el gestor relacional en memoria *MemSQL* ya que dispone de adaptadores para su actualización reactiva desde *Spark*. Además, opera a nivel de micro-baches (en vez de tuplas) lo que facilita la gestión del estado (*windowing*) al tiempo que garantiza el orden en que estos son procesados a costa de incrementar ligeramente la latencia, diferencias principales con respecto a su competidor *Storm*. En el caso de *S-*

Store se trabaja con una BD distribuida capaz de operar en memoria, *H-Store* (comercializada como *VoltDB*), a la que han añadido una capa para gestionar su actualización reactiva en base a flujos de datos de entrada que llegan, usando lenguaje SQL y con garantía transaccional (ACID)[3].

4 Conclusiones y líneas futuras

La arquitectura propuesta satisface las características básicas que requieren las aplicaciones basadas en la gestión de flujos de datos que operan en TR [7]. Los objetivos que se persiguen son evaluar estas configuraciones y extenderlas para satisfacer los requisitos que se derivan de las restricciones temporales en aplicaciones TR cuasi estrictas (*firm real-time*): a) latencias en el rango de las décimas de segundo y un caudal acotado de los flujos de datos específicos de la aplicación; b) predictibilidad del comportamiento temporal, el punto que presenta mayor problema porque toda la tecnología utilizada está desarrollada para plataformas Linux y en lenguajes como Java, Python o Scala que no son compatibles con TR estricto (*hard RT*); c) bases de datos que ofrezcan consultas continuas consistentes con requisitos de caducidad en los datos afectados en el cumplimiento de requisitos temporales.

Actualmente se están comparando las arquitecturas utilizando el *Leaderbooard maintenance benchmark* [3] ejecutado sobre un Intel Core i5-4570 3.2GHzx4 y 32GB de RAM. Dado que S-Store solo puede acceder a la BD desde un solo nodo, se ha configurado el acceso a MemSQL del mismo modo. Todos los productos han sido configurados con sus parámetros por defecto. Los resultados de los experimentos realizados hasta el momento no nos permiten, por ahora, valorar de manera justa cada tecnología.

Agradecimientos. Este trabajo ha sido financiado por el Gobierno de España con referencia TIN2014-56158-C4-2-P (M2C2).

Referencias

1. Lindström, J.: Real Time Database Systems. John Wiley & Sons, Inc. (2007)
2. Orenstein, G., Doherty, C., White, K., Camiña, S.: Building Real-Time Data Pipelines. O'Reilly Media, Inc (2015)
3. Meehan, J. et al: S-Store: Streaming meets transaction processing. Proc. VLDB Endow. 8(13), 2134-2145 (2015)
4. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: An acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30(1), 122-173 (2005)
5. Wingerath, W., Gessert, F., Friedrich, S., Ritter, N.: Real-time stream processing for big data. Information Technology 58(4), 186-194 (2016)
6. Marz, N., Warren, J.: Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co., Greenwich, CT, USA, 1st ed. (2015)
7. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 Requirements of Real-Time Stream Processing. Newsletter ACM SIGMOD 34(4), 42-47 (2005)