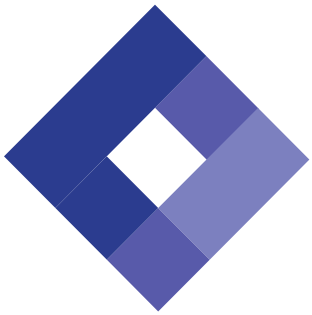

Instituto
International de
Ciencia de datos

Arquitectura Lambda vs Arquitectura Kappa

¿Cuál es el mejor enfoque para implementar un ambiente de trabajo para procesar big data?

Josué Careaga

El mundo de la industria y los negocios se transforma a un paso acelerado. Así también las necesidades que surgen de la operación y de quienes toman las decisiones referentes a la táctica y la estrategia. Para resolver dichas necesidades existen diversas herramientas técnicas y tecnológicas, tal es el caso de las nuevas infraestructuras y el software asociado que las administra. Ambos están diseñados para responder en latencias muy bajas, y dan la oportunidad de escalar la capacidad de cómputo y memoria, acompañando el constante crecimiento de la actividad económica e industrial que se desarrolla en el seno de la industria.

Hoy existen varias perspectivas de arquitecturas o pipelines que permiten realizar análisis estadísticos sobre un conjunto de datos de gran tamaño, conocido como Big Data. Entre esas opciones se encuentran las que utilizan herramientas de software libre, las que utilizan sólo software de paga y aquellas que combinan la implementación de herramientas de ambos tipos. La decisión de considerar un alto porcentaje de un tipo o de otro o cual es la mejor combinación, depende mucho de las necesidades de procesamiento que se desee cubrir y de los recursos con los que se cuenten. Este artículo comentará acerca de los enfoques de dos arquitecturas que ya han tomado renombre cuando se trata de procesamiento de grandes cantidades de información.

Primero, se hará una revisión de las características principales de cada una, enseguida se describirán sus ventajas y desventajas y finalmente, se concluirá considerando una mejor perspectiva y entendimiento de los componentes principales de cada una, así como sus objetivos.

La arquitectura Lambda es una arquitectura de procesamiento genérica, sin embargo, posee algunas características que la han hecho ser una de las arquitecturas mayormente implementadas cuando se busca procesar información de grandes volúmenes. Algunas de sus características son: es escalabe, tolerante a fallas y está orientada a satisfacer las necesidades de un sistema robusto, el cual, por sí mismo, es tolerante a fallas de dos tipos: infraestructura y errores humanos. Dicha arquitectura permite ejecutar una gran cantidad de cargas de trabajo y de casos de uso para los cuales son requeridas lecturas y actualizaciones de baja latencia. El diseño de una arquitectura lambda permite su escalabilidad lineal, es decir, su escalamiento es del tipo scale-out y

no scale-up, dado que el enfoque de esta arquitectura es el procesamiento de información de alta demanda usando archivos “siempre abiertos”.

La arquitectura Lambda está conformada por tres capas principales que son consideradas las responsables de realizar la ejecución de las tareas más relevantes del procesamiento de datos y las que entregan los resultados de

dicho procesamiento: Capa de Segmentos, Capa de Servicios y Capa de Velocidad (Batch Layer, Serving Layer, Speed layer).

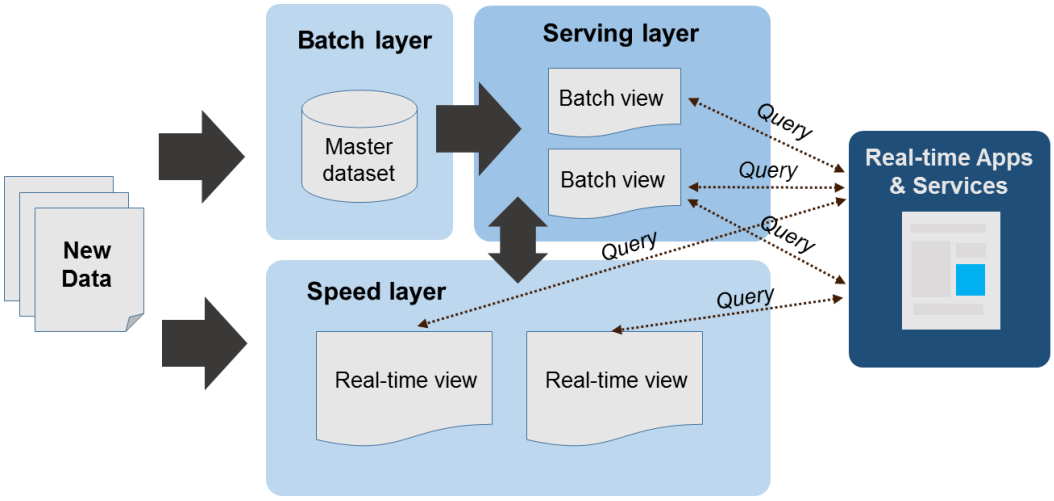

Figura 1. Diseño a gran escala de la arquitectura Lambda

Existen otros componentes que complementan la arquitectura desde el punto de vista de conectividad. Sin embargo, no son consideras elementos centrales de la arquitectura. Son módulos que permiten la comunicación de los componentes de las capas con componentes externos o con los mismos conectores que hacen posible el insumo de los datos a ser procesados.

La idea principal de la arquitectura Lambda es que toda la información que entra al sistema sea replicada en ambas, en la capa de velocidad (speed layer) y en la capa de segmentos (batch layer) para que la información esté disponible para generar vistas en tiempo real en la capa de velocidad (speed layer) y vistas batch en la capa de servicios (serving layer). De esta manera, cualquier consulta realizada al sistema puede ser resuelta combinando resultados de las vistas batch (batch views) y de las vistas en tiempo real (real-time views).

La capa de segmentos (batch layer) tiene dos funciones principales:

i) Administrar el dataset maestro, que es un conjunto de información en bruto (inmutable y solamente de almacenamiento).
ii) Llevar a cabo un pre-cómputo que genere las vistas por segmentos (batch views) que serán utilizadas en la capa de servicios (serving layer).

Por su parte, la capa de servicios (serving layer) indexa las vistas por segmento (batch views) generadas por la capa de segmentos (batch layer) para que estén disponibles para ser consultadas con baja latencia y con consultas ad-hoc.

Finalmente, la capa de velocidad (speed layer) o también conocida como capa de procesamiento entrega actualizaciones de información con alta latencia a la capa de servicios (serving layer) utilizando solo información recientemente actualizada en la fuente de datos y creando vistas de tiempo real (real-time views) a través de algoritmos incrementales.

La arquitectura Lambda está orientada a aplicaciones que han sido construidas alrededor de transformaciones asíncronas complejas que necesitan ejecutarse con latencia baja (segundos a minutos). Un buen ejemplo es un sistema de recomendación de noticias, el cual necesita extraer las noticias de diferentes fuentes, procesarlas y normalizarlas, indexarlas, ponderarlas y almacenarlas para que estén disponibles.

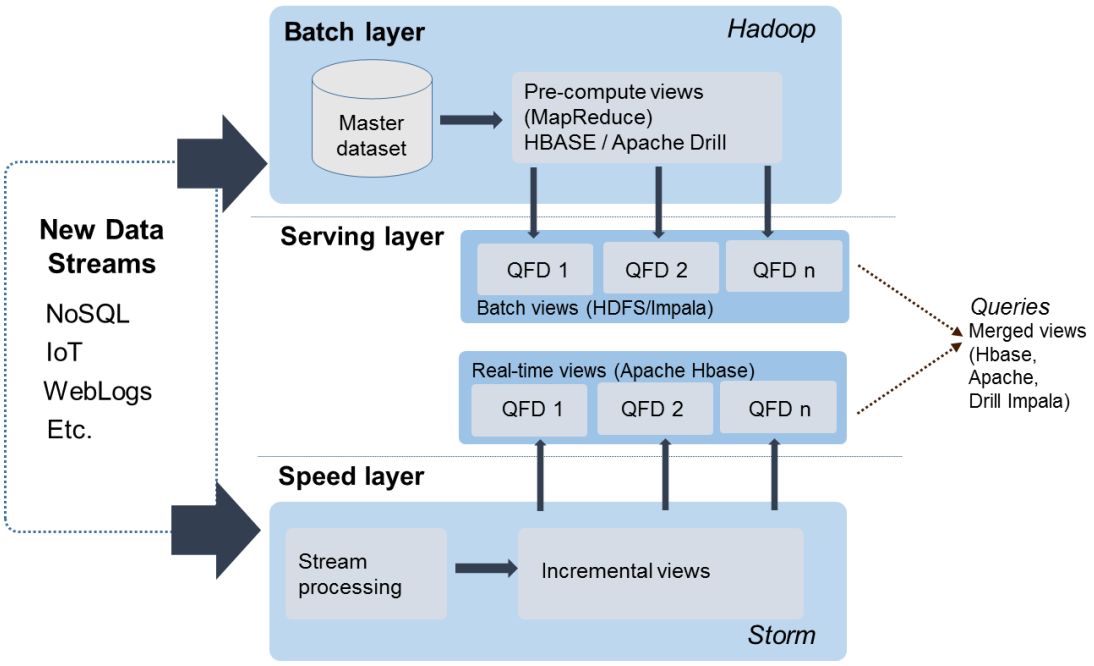

Figura 2. Composición detallada de la Arquitectura Lambda

Ventajas:

- Los datos de entrada prevalecen intactos en una parte de almacenamiento inicial (master dataset).

- Permite que flujos de procesamiento sean rastreables y a la vez permite que se pueda hacer debug de cada etapa de manera independiente.

- Considera el problema de reprocesamiento de información una vez que la aplicación construida haya evolucionado y necesita procesar campos o variables que no eran requeridos anteriormente o simplemente cuando se quiera corregir un error.

- Combina resultados de ambos procesamiento de datos en batch y en tiempo real.

Desventajas:

- Necesidad de proporcionar mantenimiento a los códigos que generan el mismo resultado en dos sistemas distribuidos complejos, lo cual representa trabajo excesivo.

- La complejidad operacional que resulta una vez implementado un sistema distribuido como hadoop o storm es diferente incluso en el código.

- Al crear nuevas abstracciones del sistema, estas deben ser soportadas por la intersección de los dos sistemas.

- Si se hace una modificación en el código y se hace commit, se perdería la compatibilidad con la gran gama de herramientas y lenguajes que hace que Hadoop sea tan poderoso (Hive, Pig, Crunch, Cascading, Oozie, etc.)

- Necesidad absoluta de consolidar diferentes bases de datos a través de mapeos transparentes y de proveer una interfaz casi estandarizada.

- Dificultad de abstraer paradigmas de programación totalmente divergentes construidos sobre sistemas distribuidos que son ya muy estables.

Por su parte, la arquitectura Kappa, se sabe que una de las motivaciones más importantes para crear la arquitectura Kappa, fue el evitar dar mantenimiento a dos sistemas separados (la capa batch y la capa de velocidad). La idea principal es administrar ambos, el procesamiento de datos y el procesamiento continuo de datos usando un único motor de procesamiento en línea.

Partiendo de ese objetivo, la arquitectura Kappa, sólo cuenta con dos capas: capa de tiempo real y capa de servicios. La capa de procesamiento en línea ejecuta las actividades de procesamiento en línea y las actualizaciones de código que permiten reprocesar los datos y visualizar los cambios en los resultados. Normalmente, una tarea de procesamiento en línea es ejecutada para habilitar el procesamiento de información en tiempo real. Una tarea de procesamiento de datos en línea en particular, es la encargada de replicar toda la información previa una vez ejecutados los códigos que modifican información.

Estas tareas son creadas bajo demanda, es decir, las instancias de streaming que deberán reprocesar información son ejecutadas simultáneamente con las innstancias que ejecuta la capa de streaming con regularidad, posteriormente, deben replicar su ejecución sobre toda la información previa, para conservar la integridad de la misma y mantener las reglas de procesamiento actualizadas para cada instancia.

Cuando se desea reprocesar, se inicia una segunda instancia de la tarea que se encuentra ejecutando el procesamiento en línea (streaming processing) comenzando a procesar en el inicio de la información ya “retenida” pero enviando su resultado a una nueva tabla de salida. Cuando la segunda instancia de la tarea haya terminado de procesar la información se debe cambiar la aplicación para leer la información de la nueva tabla de salida.

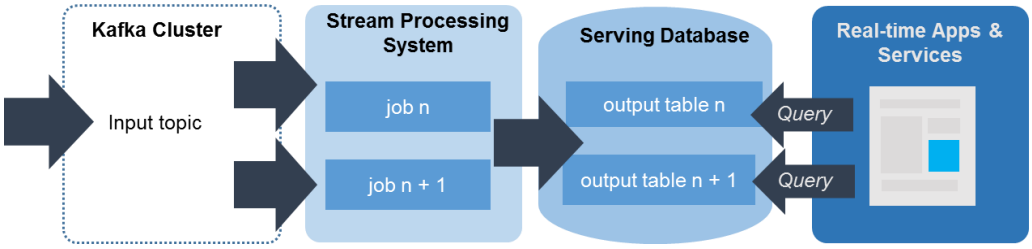

Figura 3. Composición de la arquitectura Kappa

Ventajas:

- Es una simplificación de la arquitectura Lambda, ya que se suprime el uso de la capa de batch.

- La información es almacenada utilizando un log inmutable de sólo almacenamiento, del cual se envía a almacenamientos auxiliares para la capa de serving.
- La arquitectura Kappa permite que la migración y la reorganización de la información a partir de diversas fuentes de información se ejecuten de manera eficiente proporcionando la información de manera rápida a través de la capa de streaming.
- Dada la ausencia de la capa de batch, sólo un código debe de ser actualizado en caso de necesitar mantenimiento.
- No utiliza un esquema de base de datos relacional o un almacenamiento basado en valores clave, como SQL o Cassandra, respectivamente.

Desventajas:

- Requiere del doble de espacio de almacenamiento en el sistema de base de datos de salida temporalmente, mientras exista la posibilidad de reprocesar la información.
- Los sistemas de bases de datos deben de soportar escritura de grandes volúmenes de registros para el reprocesamiento.
- Las características del hardware requerido para realizar el reprocesamiento de la información deben considerar un pequeño porcentaje adicional para los jobs que estarán reprocesando activamente la información en cualquier momento dado.

Una vez revisados los elementos primordiales de ambos enfoques (lambda y kappa) se concluye que, a pesar de encontrar diferencias en sus ventajas y desventajas, la elección de alguno para su implementación dependerá totalmente de la necesidad que se desee cubrir, del hardware por implementar, del personal a designar para su correcta implementación y la necesidad de tener la información procesándose en línea (el tiempo de respuesta o la latencia que se espera en los sistemas para que terminen sus ejecuciones). Lo anterior, debido a que los cuatro, son factores que determinan el presupuesto para la solución. Sin embargo, existen algunas recomendaciones que surgen del análisis del funcionamiento y de los componentes de cada arquitectura. Como lo son:

Si se desea implementar una arquitectura Lambda se recomienda utilizar un sistema de procesamiento en batch tipo Map Reduce (si no hay sensibilidad a la latencia). Por otro lado, en caso de que exista mucha dependencia a la latencia, es decir, al tiempo de respuesta del procesamiento de la información se sugiere utilizar un framework de procesamiento en stream. Si lo que se busca es contar con la posibilidad de desarrollar, probar, hacer debug y operar los sistemas sobre un solo flujo de procesamiento de información y seguir obteniendo los mejores resultados del mismo, se sugiere utilizar la arquitectura Kappa, ya que esta ahorra la implementación de la capa de batch. Sin dejar de considerar que debe existir un clúster que administre el catálogo de logs de los sistemas que suministran la información para detectar los cambios que ha sufrido la misma y de esta manera sólo reprocesar la que ha cambiado.

La arquitectura kappa ofrece mayores ventajas que desventajas, probablemente puede ser la mejor opción a implementar siempre que se adecúe a las necesidades de la solución esperada, a los recursos considerados para su implementación y la capacitación de los mismos en los frameworks que este enfoque utiliza.

Referencias:

http://lambda-architecture.net/

https://www.oreilly.com/ideas/questioning-the-lambda-architecture

http://www.ericsson.com/research-blog/data-knowledge/data-processing-architectures-lambda-and-kappa/

http://events.linuxfoundation.org/sites/events/files/slides/ASPgems%20-%20Kappa%20Architecture.pdf

http://milinda.pathirage.org/kappa-architecture.com/

http://storm.apache.org/index.html

https://kafka.apache.org/documentation.html#introduction

http://samza.apache.org/

Acerca del autor

Josué Careaga tiene experiencia como un arquitecto de soluciones y dirigiendo proyectos, aplicando sus habilidades en modelado de datos, inteligencia de negocios, minería de datos y procesamiento de lenguaje natural en varios proyectos, tanto en el sector privado como en el público.

Josué también formó parte de un grupo de investigación universitario centrado en el procesamiento del lenguaje natural donde desarrolló y fortaleció sus competencias en análisis de datos y algoritmos de inteligencia artificial para crear aplicaciones no supervisadas de la extracción automática de datos.

Actualmente se desempeña como gerente en PwC, construyendo soluciones estratégicas de arquitectura de infraestructura y de datos.

Acerca del Instituto

El Instituto Internacional de la Ciencia de Datos busca resolver preguntas importantes de la sociedad a partir de los datos que genera la misma. Construye sinergias entre la creciente comunidad de entusiastas y científicos de datos y fomenta la correcta ejecución de la apertura de datos y conocimiento.

https://www.linkedin.com/company/instituto-de-la-ciencia-de-datos

@the_i2ds