Módulo 8: Ingeniería avanzada de Big Data

Introducción

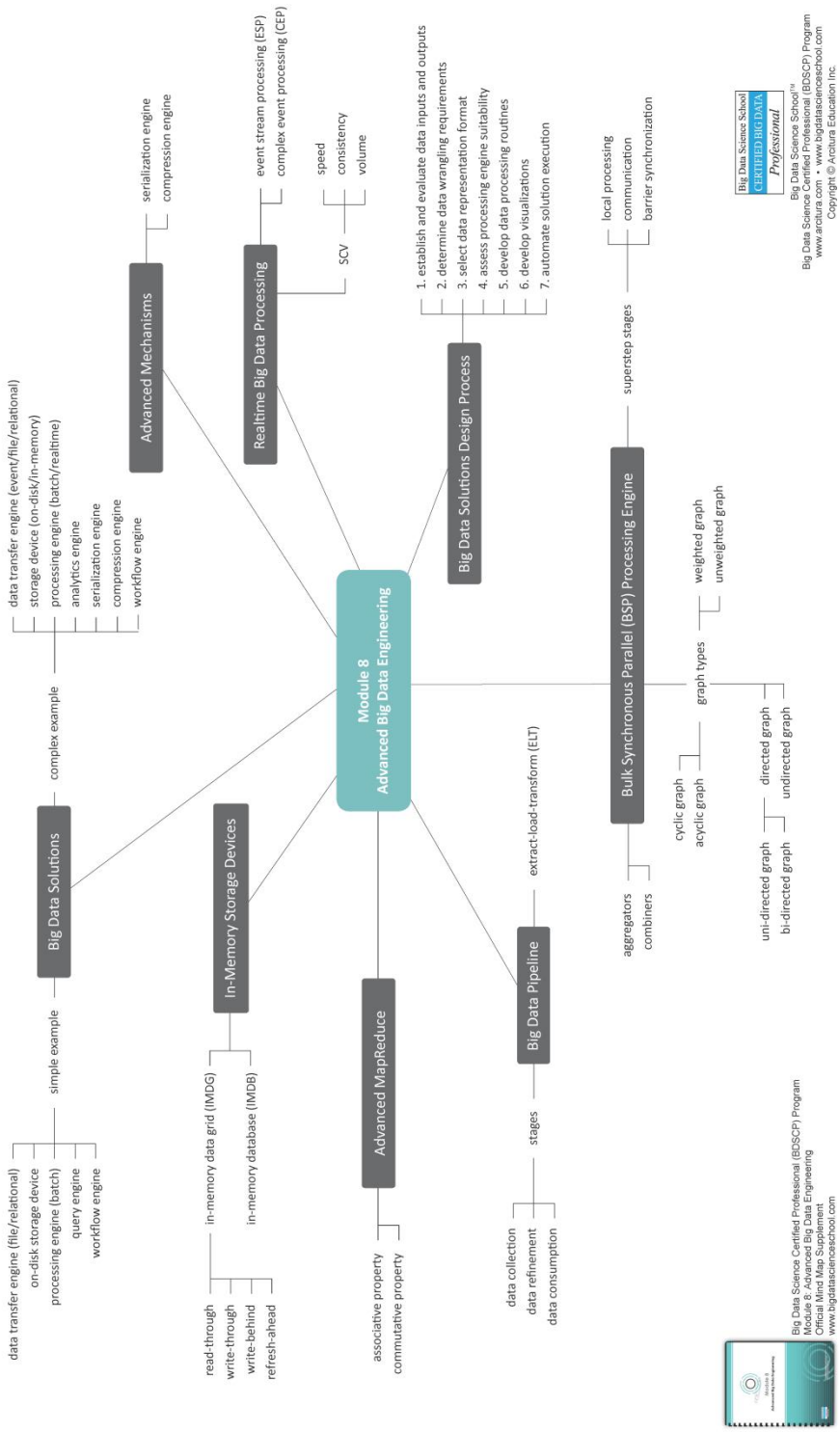
Este es el cuaderno de trabajo oficial del **Módulo 8: Ingeniería avanzada de Big Data** para el curso del BDSCP y su respectivo **Examen B90.08** de Pearson VUE.

El objetivo de este documento es brindar conocimientos sobre la ingeniería avanzada de Big Data, lo cual incluye, entre otros:

- Mecanismos de ingeniería avanzada de Big Data
- Dispositivos de almacenamiento en memoria
- Persistencia políglota
- Procesamiento de Big Data en tiempo real
- Diseño avanzado del algoritmo de MapReduce
- Motor de procesamiento Bulk Synchronous Parallel
- Big Data Pipeline y ELT
- Diseño de solución de Big Data

Póster del mapa mental

El *Póster del mapa mental del Módulo 8 del BDSCP* adjunto a este cuadernillo ofrece una representación visual alternativa de los principales temas abordados en este curso.

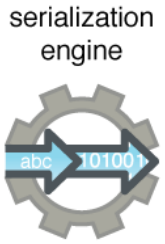


Mecanismos de ingeniería avanzada de Big Data

El Módulo 2 presentó los mecanismos fundamentales de Big Data y el Módulo 8 presenta dos mecanismos avanzados adicionales que son parte de un entorno típico de solución de Big Data. Estos son:

- Motor de serialización
- Motor de compresión

Motor de serialización



Un motor de serialización tiene la capacidad de serializar y deserializar datos en una plataforma de Big Data. La serialización es el proceso de transformar objetos o entidades de datos en bytes para conservarlos (en una memoria o en disco) o para transportarlos de una computadora a otra en una red. El proceso opuesto, la transformación de bytes a objetos o entidades de datos, se llama deserialización.

La serialización proporciona formas de estructurar bytes sin procesar en un formato que puede ser manipulado fácilmente. Los bytes serializados se pueden codificar usando un formato binario o un formato de texto plano. Los diferentes motores de serialización ofrecen diferentes niveles de velocidad, reducción de tamaño, extensibilidad e interoperabilidad.

Idealmente, un motor de serialización debe serializar/deserializar datos a alta velocidad con una máxima reducción de tamaño, ser susceptible a cambios futuros y trabajar con una variedad de productores y consumidores de datos. Sin embargo, dependiendo de los requerimientos de procesamiento de datos, es posible que se deba elegir entre un motor de serialización rápido y uno que proporcione datos de una forma más comprimida.

Un motor de serialización rápido requiere menos recursos de procesamiento, pero produce un volumen mayor de datos. Esta opción es ideal cuando hay menos recursos de procesamiento disponibles y la red es rápida. Un motor de serialización que soporta un alto nivel de compresión requiere mayores recursos de procesamiento, pero produce un volumen menor de datos y por lo general es lento. Esta opción es ideal si hay más recursos de procesamiento disponibles y la red es lenta.

En la Figura 8.1, el Nodo A debe enviar un objeto del cliente al Nodo B. Se ilustran los siguientes pasos:

1. Un motor de serialización en el Nodo A serializa el objeto usando un formato binario.
2. Después es enviado al Nodo B en la red.
3. En el Nodo B, otro motor de serialización deserializa el objeto del cliente a su forma original.

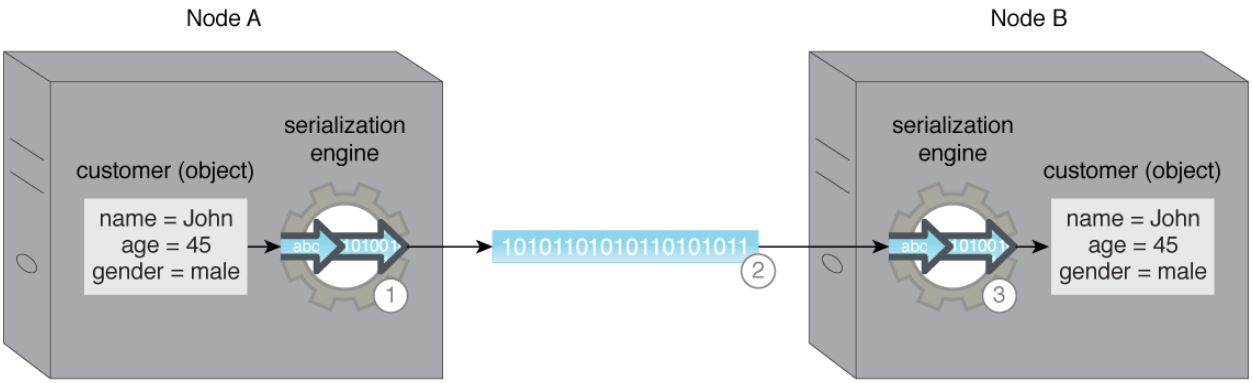


Figura 8.1 – Ejemplo de la serialización y deserialización por medio del uso de motores de serialización.

Así como los datos por cable deben transmitirse rápidamente en un espacio pequeño, lo mismo sucede con el almacenamiento de datos en disco y en memoria. Se debe elegir un motor de serialización apropiado al serializar datos para un entorno de almacenamiento de datos. La elección depende de los requerimientos de conservación de datos, ya que algunos motores de serialización pueden ser más extensibles e interoperables que otros.

Además, algunos motores de serialización pueden ser más apropiados para serializar datos binarios —tales como imágenes— que para serializar datos textuales —como archivos de registro (log files). Algunos ejemplos de motores de serialización en el contexto de Hadoop incluyen Writables, Thrift, SequenceFile, Protocol Buffers y Avro.

A menudo, los dispositivos de almacenamiento proporcionan almacenamiento de bytes sin procesar, y sin imponer ninguna estructura por medio de un esquema. Los datos son almacenados como pares llave-valor (key-value) en dispositivos de almacenamiento llave-valor (key-value) NoSQL y en grillas en memoria (IMDG). Los sistemas de archivos distribuidos también permiten el almacenamiento de valores como bytes sin procesar basados en archivos, en donde el nombre del archivo actúa como la llave. Las bases de datos relacionales tradicionales y las bases de datos relacionales en memoria (IMDB) requieren un esquema. En la sección *Dispositivos de almacenamiento en memoria* se presentan las IMDG e IMDB.

Por consiguiente, los datos deben ser serializados usando una codificación apropiada, ya que trabajar con bytes sin procesar no solo es inconveniente, sino que también puede causar problemas de interoperabilidad. Por ejemplo, es posible que los diferentes clientes que fueron desarrollados usando distintas tecnologías no tengan la posibilidad de interpretar datos almacenados por otros. Para los dispositivos de almacenamiento con sistema de archivos distribuido, la serialización debe ser aplicada a los datos sin procesar usando un motor de serialización.

Para alcanzar un alto rendimiento, por lo general se almacenan grupos de datos en el mismo archivo físico en donde se les trata como pares individuales de llave-valor (key-value). Al elegir entre dos motores de serialización distintos, se debe considerar el tamaño, la velocidad, la interoperabilidad y el tiempo de búsqueda de pares individuales llave-valor (key-value) dentro del archivo. Aunque los dispositivos de almacenamiento llave-valor (key-value) ofrecen un

almacenamiento binario sin procesar, por lo general se les aplica un grado de serialización para almacenar datos más estructurados.

La serialización puede ser aplicada automáticamente por dispositivos de almacenamiento llave-valor (key-value) que soporten una serialización personalizada. Generalmente, los dispositivos de almacenamiento de documentos serializan automáticamente los datos usando un formato como JSON o BSON, el cual permite que la búsqueda se realice en campos individuales del documento. Los dispositivos de almacenamiento basados en columnas almacenan datos como un array de bytes, y por consiguiente, requieren un motor de serialización compatible para serializar los datos. En un dispositivo de almacenamiento de grafos, los nodos y bordes son en su mayoría objetos livianos compuestos por atributos de tipo string que son serializados automáticamente. En caso de usar tipos binarios como atributos, se debe llevar a cabo una serialización manual.

En la Figura 8.2, un objeto del cliente debe ser conservado en el disco. Se ilustran los siguientes pasos:

1. Un motor de serialización serializa el objeto del cliente.
2. La serialización se lleva a cabo usando un formato de texto plano JSON.
3. A continuación se guarda en una base de datos NoSQL documental.

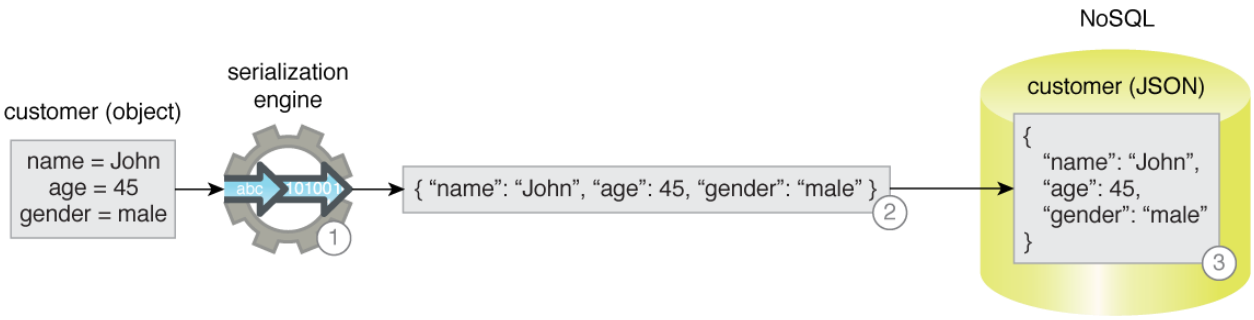


Figura 8.2 – Un motor de serialización serializa el objeto del cliente usando un formato de texto plano JSON, que después se guarda en una base de datos NoSQL documental.

Motor de compresión

Un motor de compresión tiene la habilidad de comprimir y descomprimir datos en una plataforma de Big Data. La compresión es el proceso de compactar datos con el fin de reducir su tamaño, mientras que la descompresión es el proceso de descomprimir datos con el fin de regresarlos a su tamaño original.

Debido a las características de volumen, velocidad y variedad de los datasets de Big Data, en los entornos Big Data existe la necesidad de almacenar cantidades de datos cada vez mayores. A pesar de que, en el caso de los dispositivos de almacenamiento en memoria y en disco, el costo se haya reducido considerablemente y la capacidad de almacenamiento haya aumentado, esta reducción y este aumento no son proporcionales al incremento correspondiente de volumen de datasets de Big Data.

Otro factor que hace que el espacio de almacenamiento sea más valioso y más escaso es la forma en la que los dispositivos de almacenamiento permanecen altamente disponibles y tolerantes a errores mediante la redundancia de datos. Además, a diferencia de los entornos tradicionales de procesamiento de datos, en donde los datos históricos se archivan en un almacenamiento offline, en entornos Big Data los datos históricos se mantienen online y pueden generar más valor a través de la aplicación de analítica de profundidad, con el fin de hallar información oculta. Como resultado, la compresión de datos es necesaria en los entornos Big Data para que se puedan guardar más datos en un dispositivo de almacenamiento, con el fin de conservar espacio valioso en la memoria o el disco y de mantener los costos al mínimo.

La compresión también ayuda a lograr tasas de transmisión de datos más rápidas en la red, como se analiza en la sección del motor de serialización. Distintos motores de compresión pueden proporcionar diferentes niveles de compresión de datos.

A la hora de elegir un motor de compresión, es importante entender la relación entre la rapidez de la compresión, el nivel de compacidad y los recursos de procesamiento. Generalmente, un motor de compresión rápido proporcionará datos menos compactos y requerirá menos recursos de procesamiento. En contraste, un motor de compresión lento proporcionará datos más compactos, pero requerirá más recursos de procesamiento. Algunos motores de compresión pueden ofrecer una descompresión más rápida que la compresión. Otro factor que vale la pena considerar es si el motor de compresión proporciona la capacidad de crear divisiones de un archivo almacenado en un sistema de archivos distribuido.

A manera de recordatorio de lo visto en el Módulo 7, como parte de la primera etapa de MapReduce, el archivo es dividido en varias partes más pequeñas y cada parte es enviada a un mapeador independiente. Si los datos fueron comprimidos usando un motor de compresión que no permite la división, el archivo no se puede dividir y será procesado en su totalidad por un solo mapeador, renunciando de este modo a los beneficios del procesamiento de datos distribuido y paralelo.

Normalmente, se puede lograr una reducción del tamaño de datos del 70% al 80% cuando se usa un motor de compresión. Algunos ejemplos de motores de compresión incluyen gzip, bzip2, LZO y LZ4.

Ejemplo de motor de compresión

En un sistema de archivos distribuido se debe almacenar una gran cantidad de datos sin estructurar o semiestructurados. En la parte superior de la Figura 8.3, el almacenamiento de datos en su forma sin comprimir requiere seis discos. En la parte inferior, se usa un motor de compresión para comprimir los datos. Como resultado, solo se requieren dos discos para almacenar la misma cantidad de datos en forma comprimida.

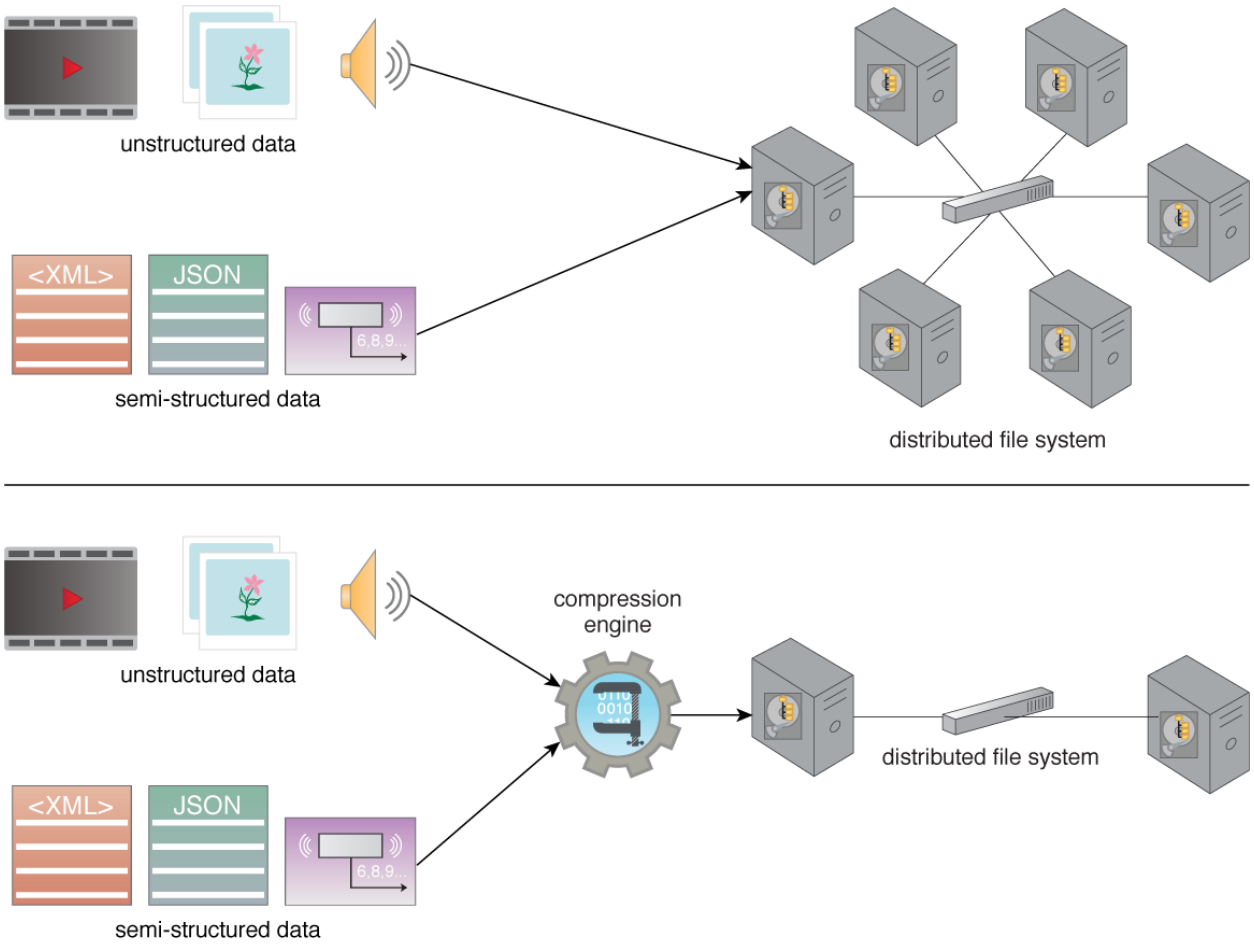


Figura 8.3 – Ejemplo que muestra datos sin comprimir en la parte superior, y un motor de compresión que se usa para comprimir datos en la parte inferior.

Ejercicio 8.1: complete los espacios en blanco

1. En las plataformas Big Data, la _____ es necesaria para permitir el intercambio de mensajes entre máquinas y guardar datos.

2. Cuando se elige un mecanismo de motor de compresión, es importante entender la relación entre _____, _____ y los _____ necesarios.

3. Preferiblemente, un mecanismo de motor de serialización debe serializar y deserializar datos a _____ velocidad con una reducción de tamaño _____, ser _____ a cambios futuros y trabajar con una variedad de productores y consumidores de datos.

4. Por lo general, un motor de compresión _____ proporcionará datos más compactos que requieren más recursos de procesamiento, mientras que un mecanismo de motor de compresión _____ proporcionará datos menos compactos y requerirá menos recursos de procesamiento.

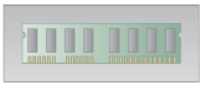
5. Cuando se usa un sistema de archivos distribuido, los datos deben serializarse, ya que trabajar con bytes _____ no solamente es difícil, sino que también causa problemas de _____.

Notas

Notas / Bocetos

Dispositivos de almacenamiento en memoria

NOTA
En este Módulo, cualquier referencia a los dispositivos de almacenamiento en memoria es una referencia implícita a los dispositivos de almacenamiento en memoria basados en cluster que abarcan múltiples computadoras, al contrario de una sola computadora.



in-memory storage device

El Módulo 7 presentó los dispositivos de almacenamiento en disco y sus diferentes clases como medios fundamentales de almacenamiento de datos. El Módulo 8 hace uso de estos conceptos y presenta el almacenamiento en memoria como una forma avanzada de almacenamiento de datos.

Normalmente, un dispositivo de almacenamiento en memoria utiliza la RAM —la memoria principal de una computadora— como su medio de almacenamiento para proporcionar acceso rápido a los datos. La capacidad en aumento y el costo cada vez menor de la memoria principal (RAM), junto con una mayor velocidad de lectura y escritura de los discos duros en estado sólido, han hecho posible desarrollar soluciones de almacenamiento de datos en memoria.

El almacenamiento de datos en la memoria elimina la latencia relacionada con la escritura o lectura del disco y la transferencia de datos entre la memoria principal y el disco duro. Esta reducción generalizada en la latencia de operaciones de lectura y escritura de datos hace que el procesamiento de datos sea mucho más rápido. La capacidad del dispositivo de almacenamiento en memoria se puede incrementar de forma masiva creando clusters de varios nodos conectados entre sí.

La memoria basada en cluster permite el almacenamiento de grandes cantidades de datos, incluyendo datasets de Big Data, a los que se puede acceder de una forma considerablemente más rápida en comparación con un dispositivo de almacenamiento en disco. Esto reduce significativamente el tiempo total de ejecución de la analítica de Big Data, lo que permite ejecutar la analítica de Big Data en tiempo real.

En la parte superior de la Figura 8.4, una lectura secuencial de 1 MB de datos en un dispositivo de almacenamiento en memoria tarda cerca de 0,25 ms. En la parte inferior, la lectura de la misma cantidad de datos en un dispositivo de almacenamiento en disco tarda cerca de 20 ms, lo que muestra que la lectura de datos en un dispositivo de almacenamiento en memoria es aproximadamente ocho veces más rápida que la lectura en un dispositivo de almacenamiento en disco. Cabe señalar que se asume que el tiempo de transferencia de datos en red es el mismo en los dos escenarios y por lo tanto se ha excluido del tiempo de lectura.

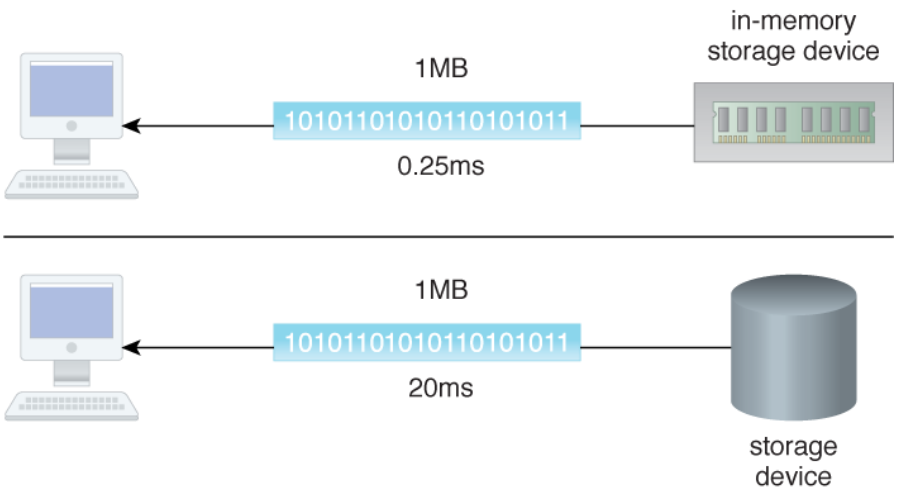


Figura 8.4 – Ejemplo de un dispositivo de almacenamiento en memoria, comparado con un dispositivo de almacenamiento en disco.

Un dispositivo de almacenamiento en memoria posibilita la analítica en memoria, lo que se refiere al análisis de datos (Data Analysis) en memoria; por ejemplo, la generación de ciertas estadísticas por medio de la ejecución de consultas de datos que están almacenados en la memoria en lugar del disco. La analítica en memoria facilita la analítica operacional (operational analytics) y la inteligencia de negocios (BI) operativa por medio de una rápida ejecución de consultas y algoritmos.

Principalmente, el almacenamiento en memoria permite darle sentido a la rápida afluencia de datos en un entorno Big Data (característica de velocidad) al proporcionar un medio de almacenamiento que facilita la generación de información en tiempo real. Esto permite tomar decisiones empresariales rápidas para mitigar una amenaza o para tomar ventaja de una oportunidad.

Un dispositivo de almacenamiento en memoria de Big Data es implementado en un cluster, lo que ofrece alta disponibilidad y redundancia. Por lo tanto, se puede lograr la escalabilidad horizontal simplemente agregando más nodos a la memoria. Cuando se compara con un dispositivo de almacenamiento en disco, un dispositivo de almacenamiento en memoria es más costoso debido al costo más alto de la memoria en comparación con un dispositivo de almacenamiento basado en disco.

Aunque una computadora de 64 bits puede hacer uso de 16 exabytes de memoria, la memoria instalada es considerablemente menor debido a las limitaciones físicas de la computadora, tales como el número de bahías de memoria. Para lograr la escalabilidad, no solo se debe adicionar más memoria, sino también agregar nodos que se requieren una vez se alcance el límite de memoria por nodos. Esto incrementa el costo del almacenamiento de datos.

Además de ser costosos, los dispositivos de almacenamiento en memoria no proporcionan el mismo nivel de soporte para el almacenamiento de datos duradero. Igualmente, el precio afecta la capacidad que puede lograr un dispositivo de almacenamiento en memoria, en comparación con un dispositivo de almacenamiento en disco. Por consiguiente, solo los datos actualizados y

recientes o los datos que tienen mayor valor son almacenados en la memoria, mientras que los datos anteriores son reemplazados por datos más nuevos y recientes.

Dependiendo de cómo se implemente, un dispositivo de almacenamiento en memoria puede soportar un almacenamiento sin esquema o compatible con esquemas. El soporte de almacenamiento sin esquema se proporciona por medio de la conservación de datos de llave-valor (key-value) (explicado brevemente).

Un dispositivo de almacenamiento en memoria es apropiado sí:

- los datos son recibidos gran velocidad y requieren analítica o procesamiento de flujo de eventos (ESP) en tiempo real
- es necesaria la analítica continua o permanente, como la Inteligencia de negocios (BI) operativa y la analítica operacional (operational analytics)
- se debe llevar a cabo el procesamiento de consultas interactivas y la visualización de datos en tiempo real, incluyendo el análisis “qué tal si” y las operaciones drill-down
- varios trabajos de procesamiento de datos requieren el mismo dataset
- se llevan a cabo Análisis Exploratorios de Datos, ya que el mismo dataset no necesita recargarse desde el disco si el algoritmo cambia
- el procesamiento de datos implica el acceso reiterado al mismo dataset, como la ejecución de algoritmos basados en grafos
- se desarrollan soluciones de Big Data de baja latencia compatibles con transacciones ACID

Un dispositivo de almacenamiento en memoria no es adecuado sí:

- el procesamiento de datos consiste en el procesamiento por lotes (Batch Processing)
- se deben conservar cantidades muy grandes de datos durante un periodo más prolongado para llevar a cabo análisis profundos de datos (Data Analysis)
- se lleva cabo una Inteligencia de negocios (BI) estratégica o una analítica estratégica que involucra el acceso a cantidades muy grandes de datos e implica el procesamiento de datos por lotes (Batch Processing)
- los datasets son extremadamente grandes y no caben en la memoria disponible
- se realiza la transición del análisis de datos (Data Analysis) tradicional al análisis de datos (Data Analysis) Big Data, ya que la incorporación de un dispositivo de almacenamiento en memoria puede requerir capacidades adicionales e implica una configuración compleja
- La empresa tiene un presupuesto limitado, ya que configurar un dispositivo de almacenamiento en memoria puede requerir la actualización de nodos, lo cual podría hacerse reemplazando los nodos o agregando más RAM

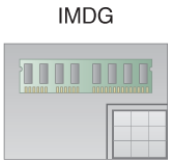
Tipos de dispositivos de almacenamiento en memoria

Los dispositivos de almacenamiento en memoria se pueden implementar como:

- Grilla de datos en memoria (IMDG)
- Base de datos en memoria (IMDB)

Aunque ambas tecnologías usan la memoria como su medio de almacenamiento de datos básico, lo que las hace distintas es la forma en la que se almacenan los datos en la memoria. Las características clave de cada una de estas tecnologías se analizan a continuación.

Dispositivos de almacenamiento en memoria: IMDG



Las IMDG almacenan datos en memoria como pares llave-valor (key-value) en múltiples nodos, en donde las llaves y los valores pueden ser cualquier objeto o datos de aplicación empresarial en forma serializada. Esto permite el almacenamiento de datos sin esquema por medio del almacenamiento de datos sin estructurar o semiestructurados. Por lo general, el acceso a los datos es proporcionado por medio de las API.

En la Figura 8.5:

1. Una imagen (a), datos XML (b) y un objeto del cliente (c) se serializan primero usando un motor de serialización.
2. Después se almacenan como pares llave-valor (key-value) en una IMDG.
3. Un cliente solicita el objeto del cliente por medio de su llave.
4. Después, la IMDG devuelve el valor en forma serializada.
5. El cliente entonces utiliza un motor de serialización para deserializar el valor con el fin de obtener el objeto del cliente...
6. ...para manipular el objeto del cliente.

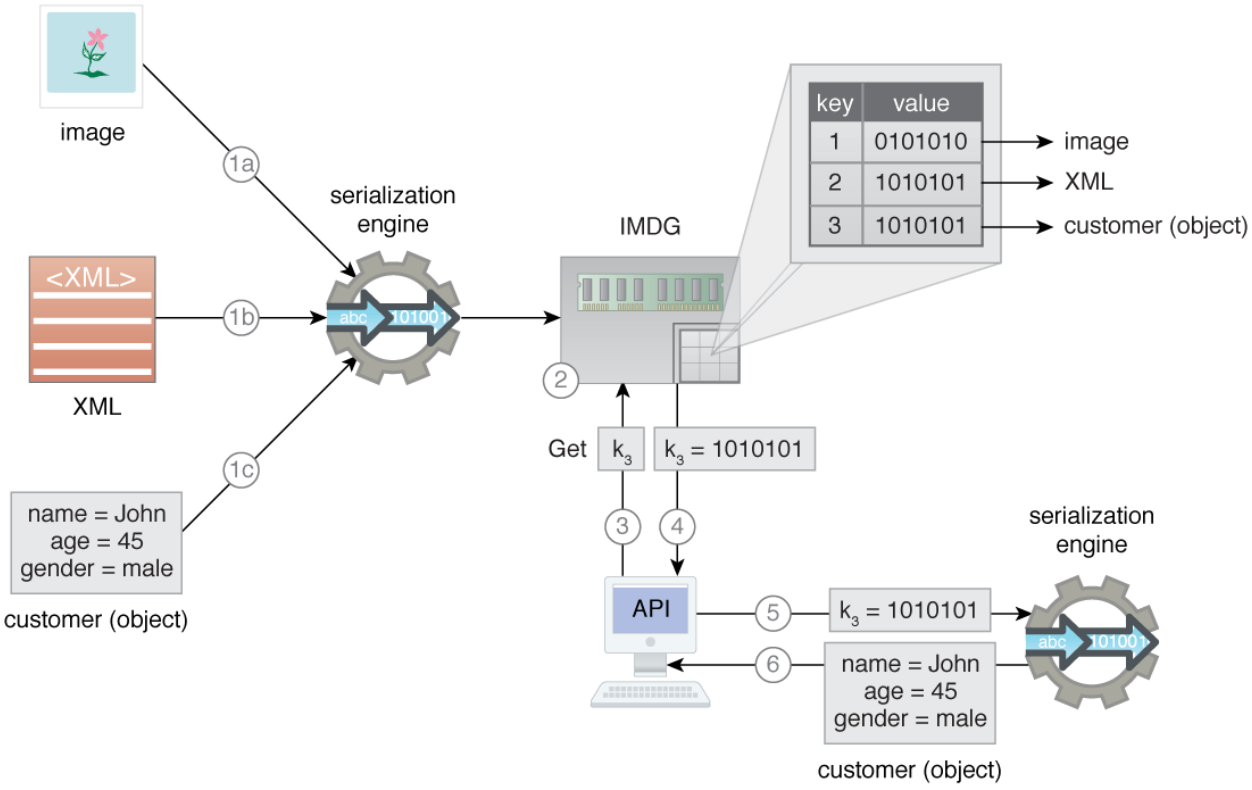


Figura 8.5 – Ejemplo de un dispositivo de almacenamiento IMDG.

Los nodos en las IMDG se mantienen sincronizados entre sí y colectivamente proporcionan alta disponibilidad, tolerancia a errores y consistencia. En comparación con el enfoque de consistencia eventual NoSQL, las IMDG proporcionan una consistencia inmediata.

En comparación con las IMDB relacionales (analizadas en la sección de IMDB), las IMDG proporcionan acceso rápido a los datos, ya que almacenan datos no relacionales como objetos. Por esto, a diferencia de las IMDB relacionales, no se requiere el mapeo objeto-relación, y los clientes pueden trabajar directamente con los objetos específicos del dominio.

Las IMDG son escaladas horizontalmente implementando la partición y replicación de datos, y además ofrecen mayor fiabilidad por medio de la replicación de datos en un nodo extra, como mínimo. En caso de que falle una computadora, las IMDG recrean automáticamente las copias perdidas de datos a partir de réplicas como parte del proceso de recuperación. Son bastante utilizadas para la analítica en tiempo real, ya que las IMDG soportan el Procesamiento de Eventos Complejos (CEP) por medio del modelo de mensajería publicación-subscripción. Esto se logra por medio de una función llamada *consultas continuas*, también conocida como consultas activas, en donde se registra una vez un filtro (filtering) para eventos de interés con la IMDG. El CEP es presentado en la sección *Procesamiento en tiempo real de Big Data*, más adelante.

La IMDG evalúa constantemente el filtro; y en el momento en que algún resultado de operaciones de inserción, actualización o eliminación cumpla con los requisitos del filtro, se les informa a los clientes suscritos (Figura 8.6). Las notificaciones son enviadas de forma asíncrona a medida que cambian los eventos —como los eventos *agregados*, *eliminados* y *actualizados*— con información acerca de pares llave-valor (key-value) —como valores *antiguos* y *nuevos*.

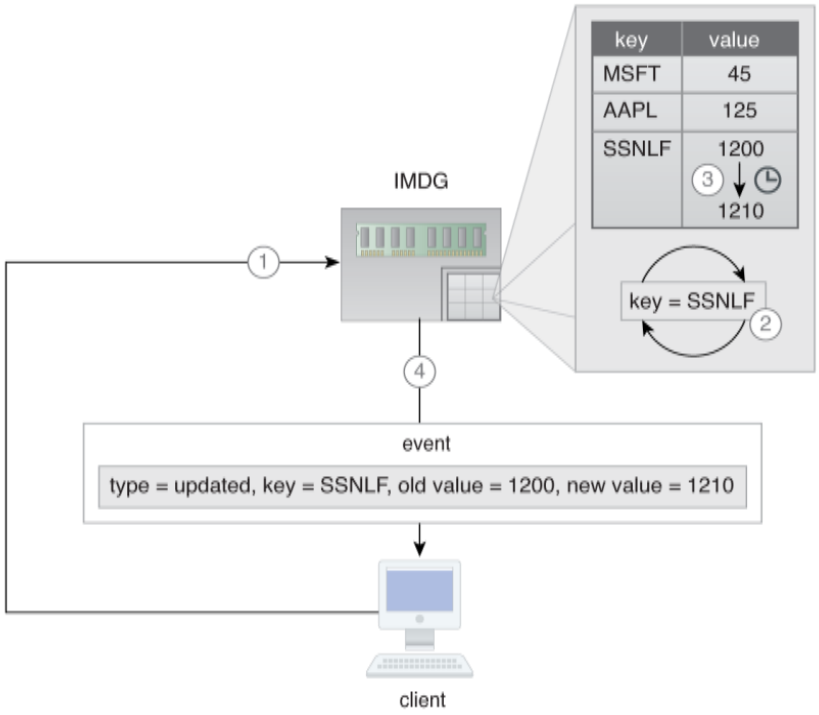


Figura 8.6 – Una IMDG almacena los precios de las mercancías, en donde la llave es el símbolo de la mercancía y el valor es el valor de dichas mercancías (se muestra como texto para facilitar la lectura). Un cliente emite una consulta continua (llave=SSNLF) (1) que está registrada en la IMDG (2). Cuando el precio de la mercancía de SSNLF cambia (3), se envía un evento actualizado al cliente suscrito que contiene varios detalles acerca del evento (4).

Desde el punto de vista de la funcionalidad, una IMDG es semejante a un caché distribuido, ya que ambos proporcionan acceso con base en memoria a los datos a los que se accede con frecuencia. Sin embargo, a diferencia de un caché distribuido, una IMDG proporciona compatibilidad integrada con la replicación y alta disponibilidad.

Los motores de procesamiento en tiempo real pueden hacer uso de las IMDG, en donde los datos de alta velocidad son almacenados a medida que son recibidos en la IMDG y son procesados allí antes de ser almacenados en el dispositivo de almacenamiento en disco, o son copiados desde el dispositivo de almacenamiento en disco en la IMDG. Esto hace que las órdenes de procesamiento de datos de gran magnitud sean más rápidas, y además permite reusar datos en caso de que haya múltiples trabajos o se ejecuten algoritmos reiterados en función de los mismos datos. Las IMDG también pueden soportar MapReduce en memoria, lo que ayuda a reducir la latencia del procesamiento MapReduce basado en disco, especialmente cuando se debe ejecutar el mismo trabajo varias veces.

Una IMDG también puede implementarse en un entorno basado en nube, en donde proporciona un medio de almacenamiento flexible que es escalable horizontal o verticalmente de forma automática, dependiendo de si la demanda de almacenamiento aumenta o disminuye, como se muestra en la Figura 8.7.

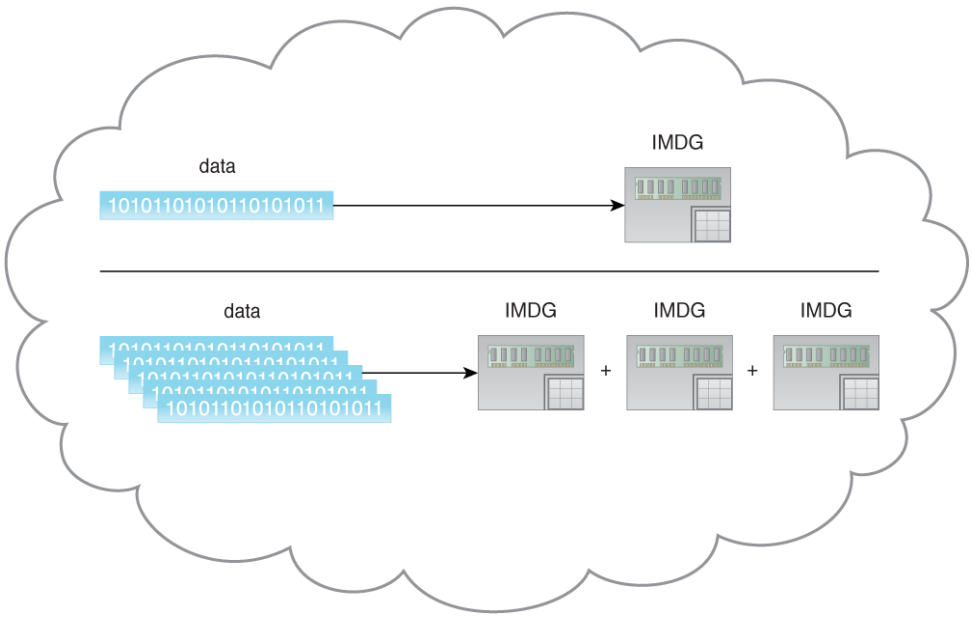


Figura 8.7 – Una IMDG implementada en una nube puede ser escalada horizontalmente de forma automática a medida que aumenta la demanda de almacenamiento de datos.

Las IMDG pueden añadirse a soluciones de Big Data existentes, implementándolas entre el dispositivo de almacenamiento en disco existente y la aplicación de procesamiento de datos. Sin embargo, esta implementación por lo general requiere cambiar el código de la aplicación, lo que se traduce en una implementación de API.

Cabe señalar que algunas implementaciones de IMDG también pueden proporcionar soporte SQL limitado o completo. Algunos ejemplos son In-Memory Data Fabric, Hazelcast y Oracle Coherence.

En un entorno de solución de Big Data, las IMDG con frecuencia son implementadas a la par con dispositivos de almacenamiento en disco que actúan como el almacenamiento backend. Esto se logra por medio de los siguientes enfoques que pueden combinarse según los requerimientos de rendimiento, consistencia y simplicidad de las operaciones de lectura y escritura:

- Lectura completa
- Write-through
- Write-behind
- Actualización posterior

Lectura completa

Si un valor solicitado para una llave no se encuentra en la IMDG, entonces se lee simultáneamente en el dispositivo de almacenamiento en disco backend; por ejemplo, una base de datos (Figura 8.8). Luego de la lectura exitosa en el dispositivo de almacenamiento en disco backend, el par llave-valor (key-value) se inserta en la IMDG y el valor solicitado es devuelto al cliente. Cualquier solicitud posterior para la misma llave será procesada directamente por la

IMDG y no por el almacenamiento backend. Aunque sea un enfoque simple, su naturaleza simultánea puede causar latencia en las operaciones de lectura.

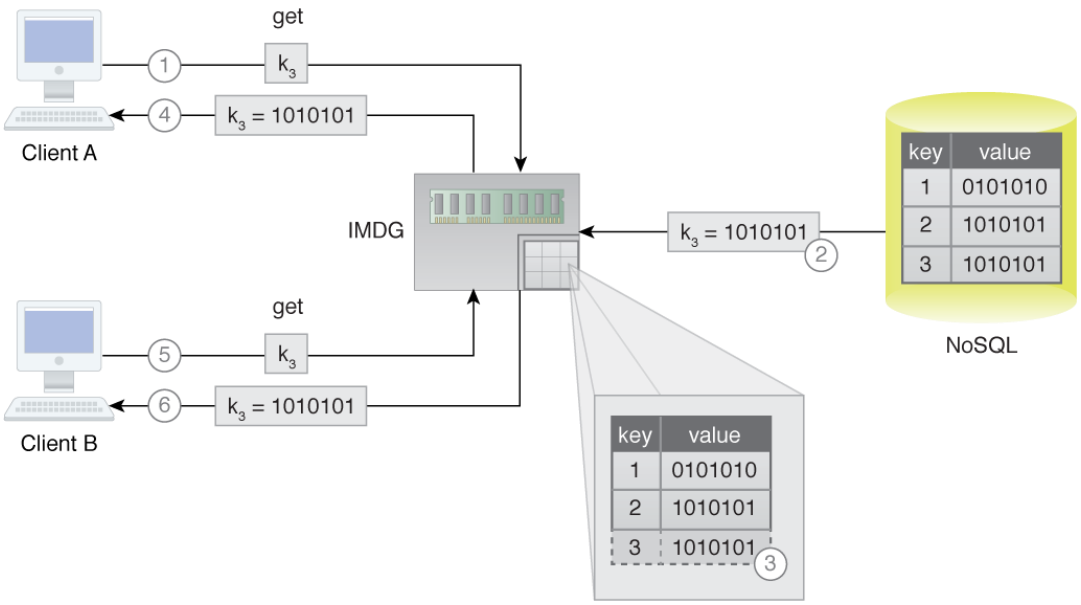


Figura 8.8 - El Cliente A trata de leer la llave K_3 (1), la cual no existe en el momento en la IMDG. Por consiguiente, esta se lee en el almacenamiento backend (2), se inserta en la IMDG (3) y luego es enviada al Cliente A (4). Entonces, la IMDG (6) procesa directamente una solicitud posterior hecha por el Cliente B (5) para la misma llave.

Write-through

Cualquier operación de escritura (inserción, actualización, eliminación) en la IMDG se escribe simultáneamente de forma transaccional en el dispositivo de almacenamiento en disco backend; por ejemplo, una base de datos. Si la operación de escritura en el dispositivo de almacenamiento en disco backend falla, la actualización se revierte en la IMDG. Debido a su naturaleza operativa, la consistencia de datos se alcanza inmediatamente entre los dos lugares de almacenamiento de datos. Sin embargo, este soporte operativo es proporcionado a costa de la latencia en operaciones de escritura, ya que cualquier operación de escritura se considera completa solo cuando se recibe la retroalimentación (de éxito o error en las operaciones de escritura) por parte del almacenamiento backend (Figura 8.9).

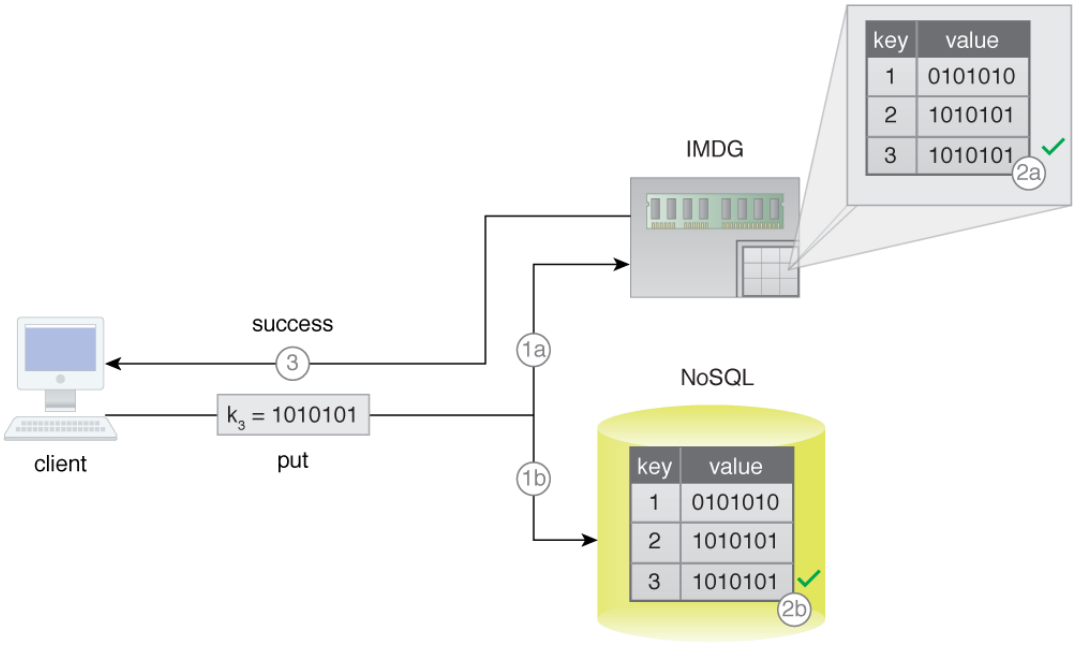


Figura 8.9 - Un cliente inserta un par nuevo llave-valor (key-value) (K_3, V_3), el cual es ingresado tanto en la IMDG (1a) como en el almacenamiento backend (1b) de forma transaccional. Luego de la inserción exitosa de los datos tanto en la IMDG (2a) como en el almacenamiento backend (2b), se le informa al cliente que los datos han sido ingresados exitosamente (3).

Write-behind

Cualquier operación de escritura en la IMDG se escribe de forma asíncrona como un lote en el dispositivo de almacenamiento en disco backend; por ejemplo, una base de datos.

Por lo general, entre la IMDG y el almacenamiento backend se ubica una cola para hacer seguimiento de los cambios requeridos al almacenamiento backend. Esta cola se puede configurar para escribir datos en el almacenamiento backend en diferentes intervalos.

La naturaleza asíncrona aumenta el rendimiento de las operaciones de escritura (la operación de escritura se considera completa tan pronto como se escribe en la IMDG), el rendimiento de las operaciones de lectura (los datos se pueden leer desde la IMDG tan pronto como se escriban en la misma), la escalabilidad y la disponibilidad en general.

Sin embargo, la naturaliza asíncrona causa inconsistencias hasta que el almacenamiento backend es actualizado al intervalo especificado.

En la Figura 8.10:

1. El Cliente A actualiza el valor de K_3, el cual se actualiza en la IMDG (a) y también se envía a la cola (b).
2. Sin embargo, antes de que se actualice el almacenamiento backend, el Cliente B hace una solicitud para la misma llave.
3. Se envía el valor antiguo.
4. Después del intervalo configurado...

5. ...el almacenamiento backend se actualiza finalmente.
6. El Cliente C hace una solicitud para la misma llave.
7. Esta vez, se envía el valor actualizado.

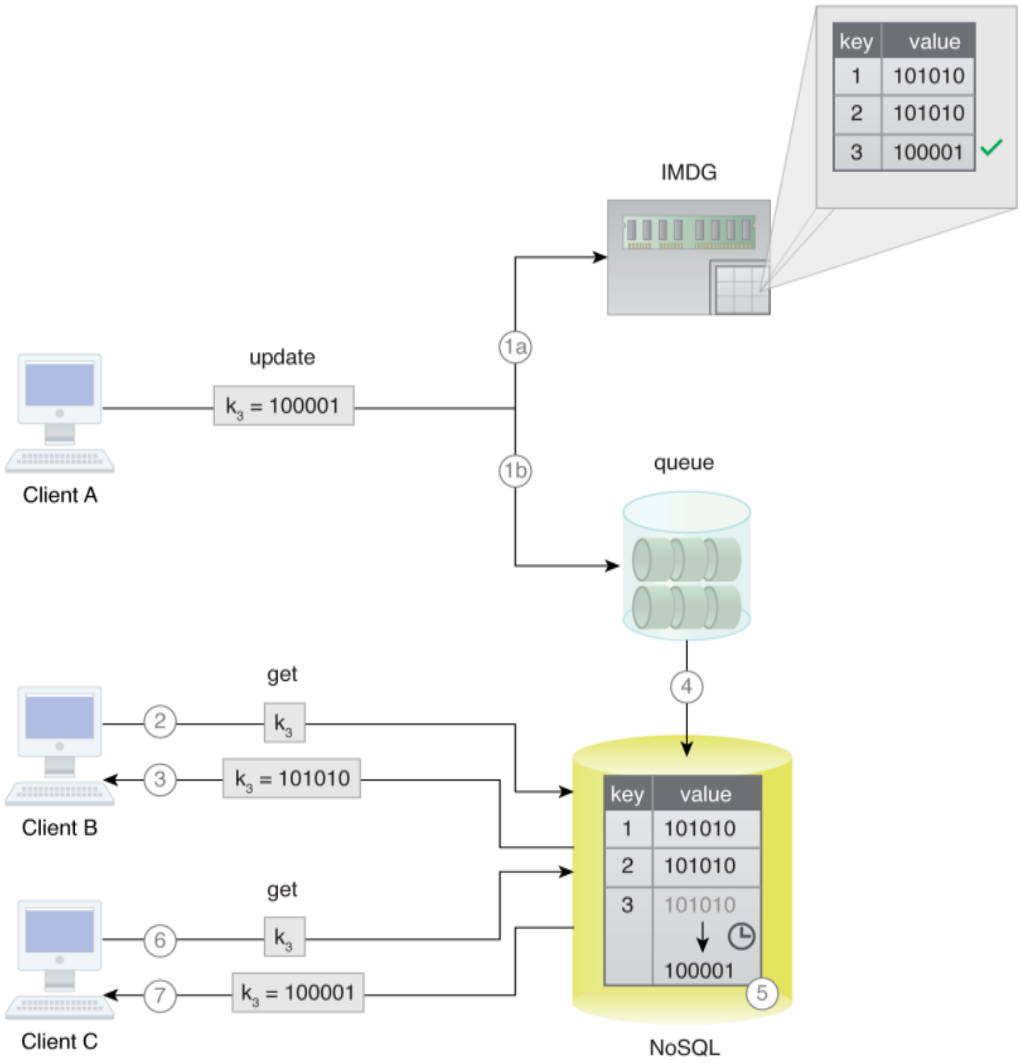


Figura 8.10 – Ejemplo del enfoque write-behind.

Actualización posterior

La actualización posterior es un enfoque proactivo en el que cualquier valor al que se haya tenido acceso de forma frecuente es actualizado automáticamente y de forma asíncrona en la IMDG, dado que se accede al valor antes de su fecha de expiración configurada en la IMDG. Si se accede a un valor después de su fecha de expiración, el valor es leído simultáneamente desde el almacenamiento backend —como en el caso del enfoque de lectura completa— y es actualizado en la IMDG antes de que sea devuelto al cliente.

Debido a su naturaleza asíncrona y prospectiva, este enfoque ayuda a alcanzar un mejor rendimiento de lectura y es especialmente útil cuando se accede a los mismos valores frecuentemente o cuando varios clientes tratan de acceder a los valores.

En comparación con el enfoque de lectura completa, en el cual los valores son procesados por la IMDG hasta su expiración, la inconsistencia de datos entre la IMDG y el almacenamiento backend se mantiene al mínimo, ya que los valores son actualizados antes de su fecha de expiración.

En la Figura 8.11:

1. El Cliente A solicita K_3 antes de su fecha de expiración.
2. El valor actual es devuelto desde la IMDG.
3. El valor es actualizado desde el almacenamiento backend.
4. El valor es actualizado en la IMDG de forma asíncrona.
5. Después de la fecha de expiración configurada, el par llave-valor (key-value) es emitido por la IMDG.
6. Ahora, el Cliente B hace una solicitud para K_3.
7. Ya que la llave no existe en la IMDG, se solicita de forma simultánea en el almacenamiento backend...
8. ...y es actualizada.
9. Entonces el valor es devuelto al Cliente B.

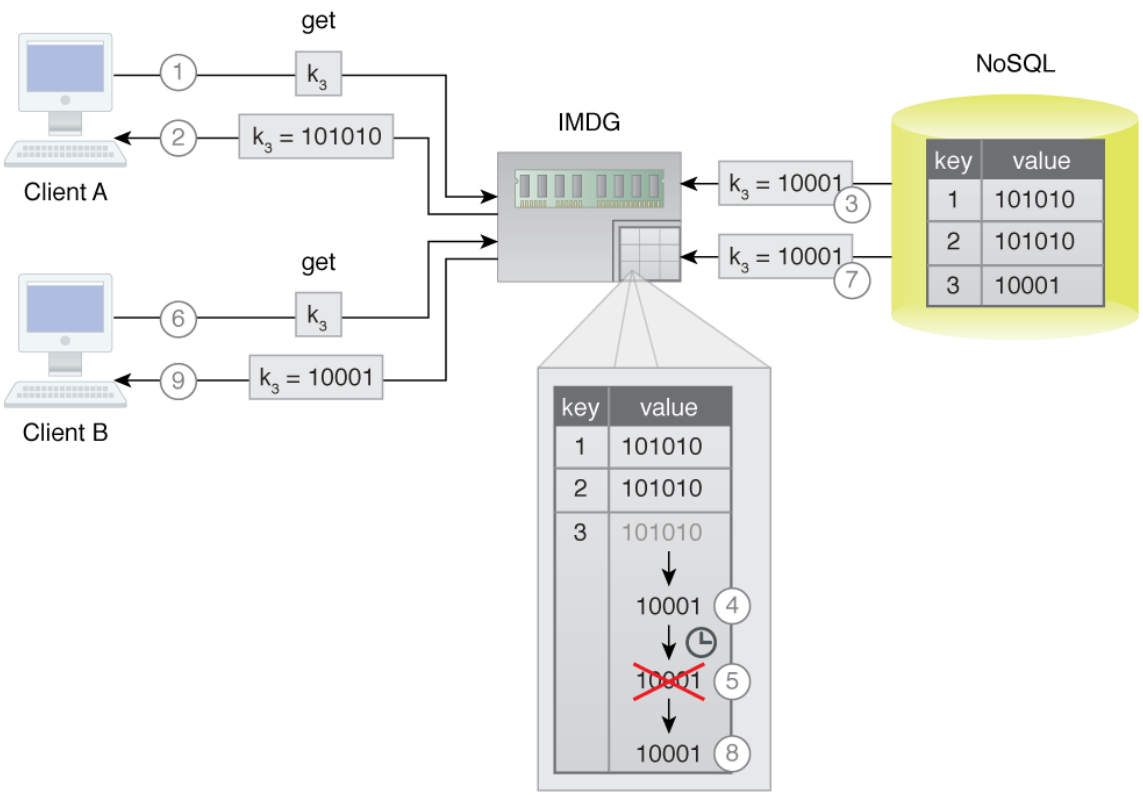


Figura 8.11 – Ejemplo del enfoque de actualización posterior.

Un dispositivo de almacenamiento IMDG es apropiado sí:

- los datos deben ser accesibles fácilmente en forma de objetos y con latencia mínima
- los datos almacenados no son relacionales; por ejemplo, los datos sin estructurar y semiestructurados
- se añade compatibilidad en tiempo real a una solución de Big Data que actualmente usa almacenamiento en disco
- el dispositivo de almacenamiento existente no puede ser reemplazado, pero la capa de acceso de datos se puede modificar
- la escalabilidad es más importante que el almacenamiento relacional; aunque las IMDG son más escalables que las IMDB (las IMDB son bases de datos funcionalmente completas), no son compatibles con el almacenamiento relacional

Dispositivos de almacenamiento en memoria: IMDB

Las IMDB son dispositivos de almacenamiento en memoria que usan la tecnología de base de datos y aprovechan el desempeño de la RAM para superar los problemas de latencia en el tiempo de ejecución, los cuales son un verdadero inconveniente en los dispositivos de almacenamiento en disco.

En la Figura 8.12:

1. Un dataset relacional es almacenado en un IMDB.
2. Un cliente solicita un registro de cliente (id = 2) por medio de SQL.
3. La IMDB arroja el registro de cliente relevante, el cual es directamente manipulado por el cliente sin necesidad de ningún tipo de deserialización.

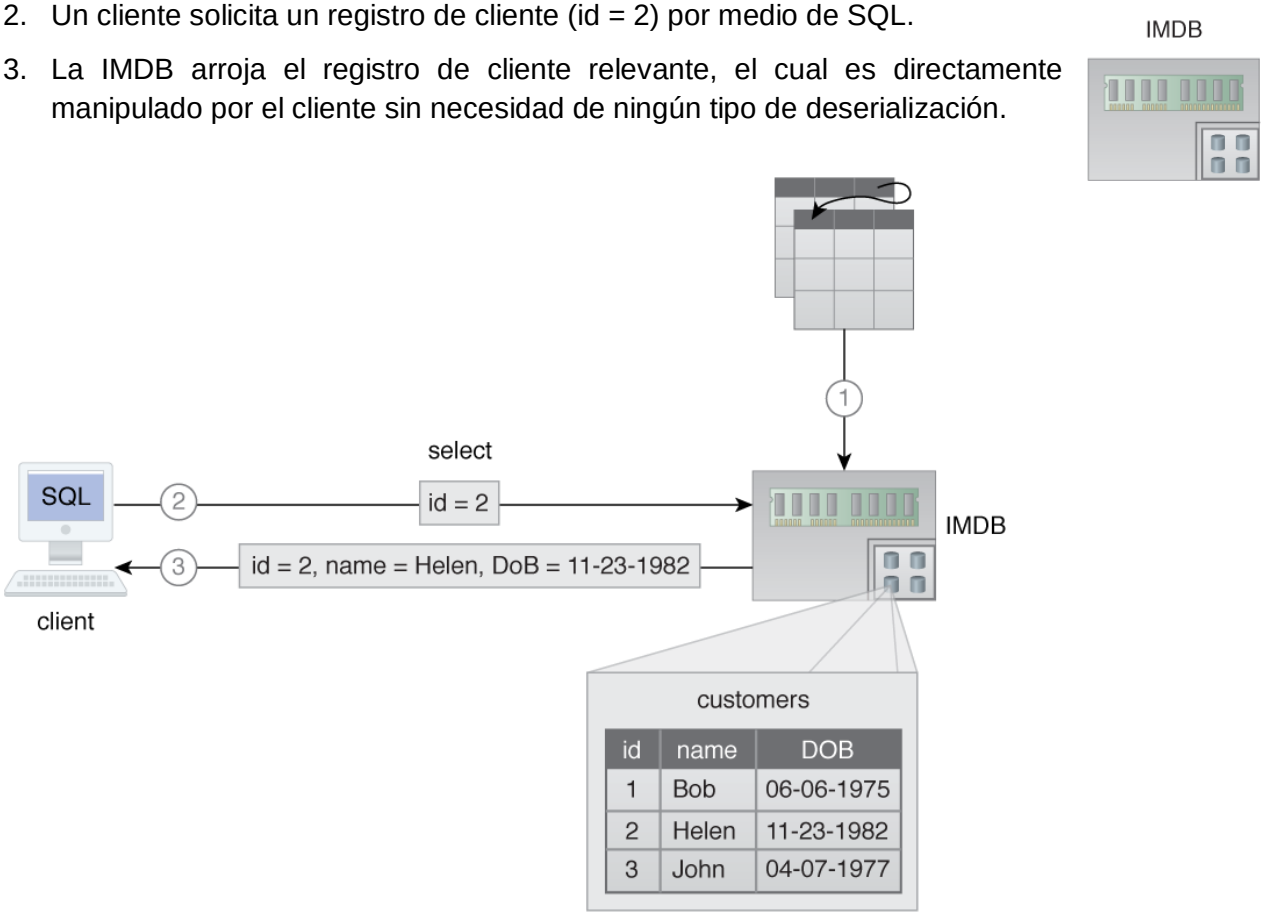


Figura 8.12 – Ejemplo que representa una IMDB.

Una IMDB puede ser relacional (IMDB relacional) para el almacenamiento de datos sin estructurar, o puede hacer uso de la tecnología NoSQL (IMDB no relacional) para el almacenamiento de datos sin estructurar y semiestructurados.

A diferencia de las IMDG, las cuales por lo general proporcionan acceso a los datos por medio de las API, las IMDB relacionales hacen uso de SQL, que es un lenguaje más conocido, lo cual ayuda a los analistas o científicos de datos que no tienen habilidades avanzadas de programación. Normalmente, las IMDB basadas en NoSQL proporcionan acceso basado en las API, que puede estar sencillamente limitado a operaciones de inserción, adquisición y

eliminación. Dependiendo de la implementación básica, algunas IMDB soportan la escalabilidad horizontal, mientras otras soportan la escalabilidad vertical.

No todas las implementaciones de IMDB son directamente duraderas; en lugar de eso, hacen uso de varias estrategias para proporcionar durabilidad frente a las fallas de la computadora o la corrupción en la memoria, tales como:

- La RAM no volátil (NVRAM) puede usarse para almacenar datos permanentemente.
- Los logs de operación de bases de datos pueden ser almacenados periódicamente en un medio no volátil, como un disco.
- Los archivos de instantánea, que capturan el estado de una base de datos en un momento determinado, se guardan en el disco.
- Una IMDB puede hacer uso del sharding y la replicación para soportar mayor disponibilidad y fiabilidad.
- Las IMDB pueden usarse conjuntamente con dispositivos de almacenamiento en disco, tales como bases de datos NoSQL y RDBMS, para un almacenamiento duradero.

Al igual que una IMDG, una IMDB también puede soportar la función de consulta continua, en donde un filtro en forma de una consulta de datos de interés se registra una vez con la IMDB. Entonces, la IMDB ejecuta continuamente la consulta de forma reiterada. Cuando el resultado de la consulta se modifique a causa de operaciones de inserción, actualización o eliminación, los clientes suscritos son informados de forma asíncrona al enviar los cambios como eventos; por ejemplo, eventos *agregados*, *eliminados* y *actualizados*, con información acerca de los valores de registro, tales como los valores *antiguos* y *nuevos*.

En la Figura 8.13, una IMDB almacena valores de temperatura de varios sensores. Se ilustran los siguientes pasos:

1. Un cliente emite una consulta continua (seleccionar * de los sensores en donde la temperatura es > 75).
2. Esto se registra en la IMDB.
3. Cuando la temperatura de cualquier sensor exceda los 75 °C…
4. ... Se envía un evento actualizado al cliente suscrito, el cual contiene varios detalles acerca del evento.

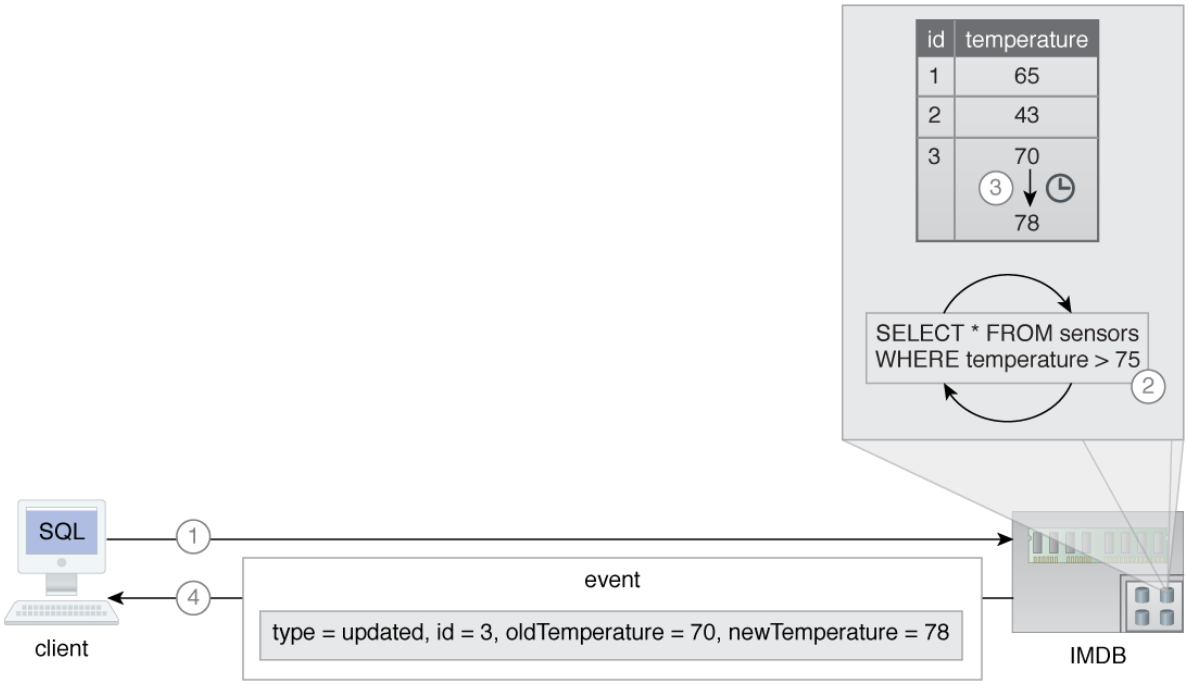


Figura 8.13 – Ejemplo del almacenamiento en IMDB.

Las IMDB son ampliamente utilizadas en analítica en tiempo real y pueden también ser usadas para desarrollar aplicaciones de baja latencia que requieren ser completamente compatibles con las transacciones ACID (IMDB relacional). En comparación con las IMDG, las IMDB proporcionan una opción de almacenamiento de datos en memoria fácil de configurar, ya que las IMDB por lo general no requieren dispositivos de almacenamiento backend en disco.

Por lo general, la introducción de las IMDB a las soluciones Big Data existentes requiere reemplazar los dispositivos actuales de almacenamiento en disco, incluyendo cualquier RDBMS, en caso de que sea usada. En caso de reemplazar un RDBMS con una IMDB relacional, se requieren pocos o ningún cambio en el código, debido a la compatibilidad con SQL que proporciona la IMDB relacional. Sin embargo, cuando se reemplaza con una IMDB NoSQL, tal vez sea necesario hacer cambios el código debido a la implementación de las API.

En caso de reemplazar una base de datos NoSQL en disco con una IMDB relacional, por lo general se requieren cambios en el código para establecer el acceso basado en SQL. Sin embargo, cuando se reemplaza con una IMDB NoSQL, es posible que siga siendo necesario hacer cambios en el código debido a la implementación de nuevas API.

Las IMDB relacionales son por lo general menos escalables que las IMDG, ya que deben soportar las consultas y transacciones distribuidas en el cluster. Algunas implementaciones de IMDB pueden beneficiarse de la escalabilidad vertical, lo cual ayuda a abordar los problemas de latencia que ocurren cuando se ejecutan consultas y transacciones en un entorno de escalabilidad horizontal. Algunos ejemplos son Aerospike, MemSQL, Altibase HDB, eXtreme DB y Pivotal GemFire XD.

Un dispositivo de almacenamiento IMDB es adecuado sí:

- los datos relacionales deben ser almacenados en una memoria compatible con ACID
- se añade compatibilidad en tiempo real a una solución de Big Data que actualmente usa almacenamiento en disco
- el dispositivo de almacenamiento en disco existente puede ser reemplazado por una tecnología equivalente en memoria
- es necesario minimizar los cambios a la capa de acceso a los datos del código de aplicación; por ejemplo, cuando la aplicación consta de una capa de acceso a los datos basada en SQL

El almacenamiento relacional es más importante que la escalabilidad; aunque muchas IMDB soportan el almacenamiento relacional, son menos escalables que las IMDG.

NOTA
Algunos proveedores proporcionan implementaciones con características compatibles parcial o totalmente con las IMDG e IMDB. Por consiguiente, tales implementaciones no pueden ser diferenciadas como IMDG o IMDB; en cambio, podemos referirnos a ellas como dispositivos de almacenamiento en memoria.

Lectura

En las páginas 222 a 224 y 229 a 230 del libro *Imperativos de Big Data* incluido en este módulo, encontrará un análisis más detallado sobre las IMDG y los IMDB.

Ejercicio 8.2: encuentre el término correspondiente para cada enunciado

Responda las siguientes preguntas completando los espacios en blanco con uno de los términos a continuación (tenga en cuenta que no se utilizarán todos los términos):

- Dispositivo de almacenamiento en disco
- Dispositivo de almacenamiento en memoria
- Dispositivo de grilla de datos en memoria
- Dispositivo de base de datos en memoria
- enfoque de lectura completa
- enfoque de write-through
- enfoque de write-behind
- enfoque de actualización posterior

1. Bob necesita agregarle capacidad de operación en tiempo real a una aplicación de procesamiento de datos. La aplicación actual usa un RDBMS, y los datos almacenados incluyen datos relacionales provenientes de varios sistemas de OLTP. Debido a la naturaleza obsoleta de la aplicación, no se pueden hacer cambios en la capa de acceso a datos. ¿Qué dispositivo de almacenamiento puede usar Bob para añadir un soporte que opere en tiempo real?

   _____

2. Suzan debe desarrollar un prototipo de solución que ejecute un análisis de sentimientos (Sentiment Analysis) basado en los comentarios hechos por los clientes en los diferentes sitios web de social media. Los datos que deben almacenarse son semiestructurados. Suzan tiene un presupuesto limitado, y quiere mantener un prototipo sencillo, pues este es su primer proyecto de Big Data. ¿Qué tipo de dispositivo de almacenamiento puede usar Suzan para implementar la funcionalidad requerida?

   _____

3. Keith está desarrollando un tablero de control (Dashboard) operativo que muestre en tiempo real el estado de mantenimiento de varios activos de TI de la empresa, incluidos servidores y switches. Desde los archivos de registro (log files) se generan múltiples datos estadísticos, los cuales se importan cada cinco minutos. La arquitectura actual de

almacenamiento está integrada por una base de datos NoSQL en disco, la cual Keith desea conservar, ya que varias aplicaciones la usan frecuentemente. ¿Qué tipo de dispositivo de almacenamiento en memoria puede ser utilizado en este caso?

_____

4. Anthony ha implementado soporte en tiempo real para una aplicación de procesamiento de datos, aumentando el nivel actual de almacenamiento de datos con una grilla de datos en memoria. Recientemente, el número de aplicaciones de clientes que tienen acceso a la grilla de datos en memoria ha aumentado considerablemente, y cada una de ellas mantiene múltiples conexiones con la grilla de datos en memoria. Los usuarios han comenzado a quejarse del rendimiento en relación con los procesos de lectura y escritura desde y hacia la grilla de datos en memoria. ¿Qué método de integración puede usar Anthony para reducir la latencia de los datos, tanto para procesos de lectura como de escritura?

_____

5. Justin está implementando una grilla de datos en memoria sobre una base de datos NoSQL existente. Él no tiene mucha experiencia en el manejo de grillas de datos en memoria, y quiere iniciar con una arquitectura simple que utilice un método sencillo para integrar la grilla de datos en memoria con la base de datos NoSQL. Existen varias aplicaciones que leen los datos a partir de la base de datos NoSQL existente, y tales aplicaciones requieren tener acceso actualizado a los nuevos datos, a medida que estos se escriben en la grilla de datos en memoria. ¿Qué método de integración puede usar Justin para cumplir con los requisitos actuales?

_____

Notas

Notas / Bocetos

Persistencia políglota

Los dispositivos de almacenamiento en disco (explicados en el Módulo 7) y en memoria, y sus subtipos, son empleados para diferentes propósitos. Cada uno está diseñado para almacenar un tipo específico de datos (estructurados, semiestructurados y sin estructurar) y patrones de acceso a datos (mediante streaming y por lotes) y soportar distintos tipos de cargas de trabajo (en tiempo real y por lotes). Por ejemplo, las bases de datos tipo grafo están optimizadas para almacenar datos con tendencia relacional, mientras que las bases de datos tipo llave-valor (key-value) son más adecuadas para almacenar datos binarios.

A fin de manejar el procesamiento por lotes (Batch Processing) y en tiempo real de los datasets de Big Data y satisfacer los requisitos de almacenamiento de datos estructurados, semiestructurados y sin estructurar, se requiere un entorno de almacenamiento heterogéneo, en el que se usen dispositivos especializados de almacenamiento aptos para guardar los datos. Dicho entorno se conoce como persistencia políglota (Figura 8.14), un enfoque híbrido que implica el uso de varios tipos de dispositivos de almacenamiento de acuerdo con los requerimientos individuales de almacenamiento de datos. Esto facilita el desarrollo de una arquitectura de almacenamiento óptima, en la que cada subtipo de dispositivo de almacenamiento se especializa en almacenar los datos según los casos de uso particulares.

Se puede usar un dispositivo de almacenamiento en memoria para ejecutar análisis en tiempo real sobre los datos basados en eventos, mientras que se puede usar un dispositivo de almacenamiento en disco para ejecutar análisis de datos (Data Analysis) por lotes sobre los datos sin estructurar. De la misma forma, los dispositivos de almacenamiento tipo llave-valor (key-value), tipo documento, basados en columnas, y las bases de datos de grafos NoSQL se pueden usar dependiendo del tipo de datos que se almacenarán y de la forma en que se accederá a los mismos (mediante campos individuales o por un BLOB), y también si los datos están relacionados de alguna forma.

Además de los tipos de datos compatibles con un dispositivo de almacenamiento, los requisitos de ACID y BASE hacen aún más específico el tipo de dispositivos de almacenamiento que pueden utilizarse en un entorno de almacenamiento políglota. Tenga en cuenta que la implementación de la persistencia políglota también puede incluir el uso de bases de datos relacionales. En los entornos Big Data, lo anterior se refiere al uso de los sistemas de OLTP o bodegas de datos digitales (Data Warehouses) para almacenar datos estructurados, mientras que los datos semiestructurados y sin estructurar son almacenados en las bases de datos NoSQL.

Usar una base de datos relacional puede ser ideal en caso de que solo se acceda o se actualice un subconjunto de un dataset y no haya necesidad de utilizar un método NoSQL basado en clusters y escalable.

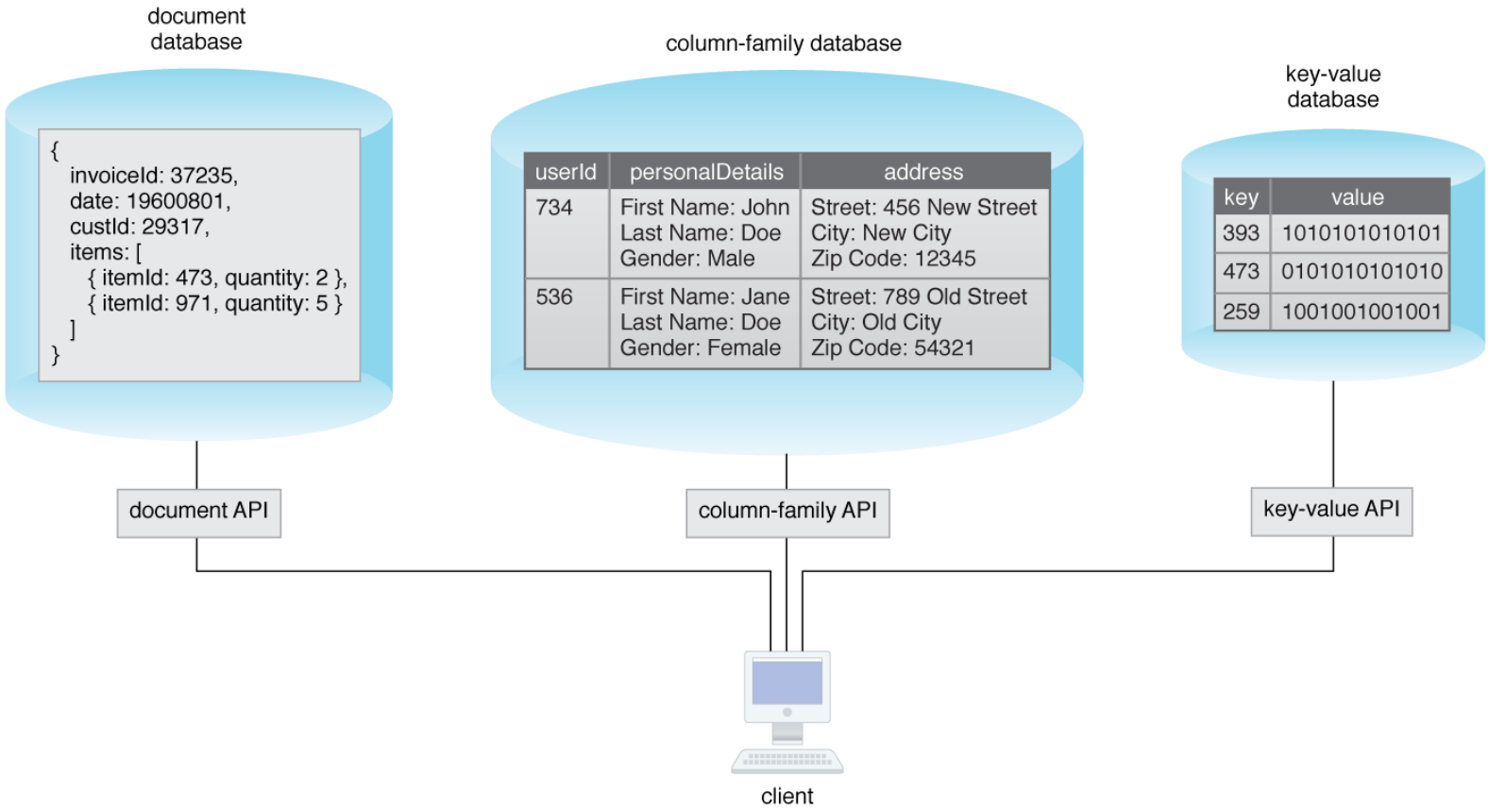


Figura 8.14 – El diagrama muestra un entorno de persistencia políglota, en el cual las facturas son almacenadas en una base de datos de tipo documento; la información del cliente es almacenada en una base de datos basada en columnas, y las imágenes de los productos son almacenadas en una base de datos de tipo llave-valor (key-value). Un cliente recupera los datos necesarios mediante la API proporcionada por cada dispositivo de

Persistencia políglota: problemas

A pesar de que la persistencia políglota ayuda a manejar múltiples requisitos de almacenamiento, también hace más compleja la arquitectura de almacenamiento y el diseño de la solución de Big Data. Por ejemplo, crear un informe que comprenda los productos (almacenados en bases de datos basadas en columnas), sus imágenes (almacenadas en bases de datos tipo llave-valor (key-value)) y las facturas (almacenadas en bases de datos tipo documento) requerirá la integración de los datos desde tres dispositivos de almacenamiento diferentes. En segundo lugar, esto expone varios puntos de fallo potenciales donde la fiabilidad y disponibilidad del sistema de procesamiento de datos son aquellas del dispositivo de almacenamiento con el nivel más bajo de fiabilidad y disponibilidad.

Con la persistencia políglota, el manejo de los diferentes tipos de dispositivos de almacenamiento puede convertirse en una preocupación, y haría falta contar con personal que tenga competencias especializadas, tales como administradores de bases de datos (DBA) independientes para cada base de datos, o un solo DBA con múltiples competencias. De forma similar, en el caso de muchos dispositivos de almacenamiento que presentan poco o ningún soporte de SQL, integrar cada dispositivo de almacenamiento requerirá adaptadores personalizados o tareas de joining y filtrado (filtering) sobre los datos según la aplicación.

Los distintos dispositivos de almacenamiento poseen controles de acceso y seguridad diferentes, lo que facilita una autenticación consistente y una autorización rigurosa. Debido a esta característica de variedad, para ejecutar análisis de datos (Data Analysis) detallados en entornos Big Data es necesario unir los datos provenientes de distintos dispositivos de almacenamiento, cada uno con su propio modelo de almacenamiento, en un dispositivo de almacenamiento provisional. Esto genera una latencia en el acceso a los datos, y dificulta aún más la forma en qué se accede a los mismos.

Es posible que múltiples aplicaciones o productos de datos interactúen con el mismo dispositivo de almacenamiento, cada uno con su propia implementación de la API del dispositivo de almacenamiento. Sin embargo, esto genera códigos duplicados para las bases, dificultando su mantenimiento. Un producto de datos es la instancia de un modelo de aprendizaje automático (Machine Learning) o estadístico diseñado durante el análisis de datos (Data Analysis). Se presenta en forma de una aplicación, y genera valor a partir de los datos para alcanzar una meta empresarial.

Persistencia políglota: recomendaciones

En un entorno de persistencia políglota, la comunicación con los diferentes dispositivos de almacenamiento se puede estandarizar al encapsular la lógica de implementación API en un servicio. Esto no solo ayuda a lograr una interfaz estandarizada de comunicación, sino que también protege las aplicaciones de cualquier cambio en el dispositivo de almacenamiento subyacente, como la sustitución de dicho dispositivo.

La seguridad es otra preocupación al momento de usar la persistencia políglota, pues los dispositivos de almacenamiento diferentes a RDBMS o Big Data, tales como las bases de datos

NoSQL generalmente no tienen patrones de seguridad por defecto, o podrían implementar esquemas de seguridad individuales o heterogéneos.

Encapsular el acceso a los dispositivos de almacenamiento en un servicio puede añadir una capa de seguridad a los dispositivos de almacenamiento que no tienen características de seguridad integradas (Figura 8.15), o puede homogeneizar las interacciones de seguridad heterogéneas para dispositivos de almacenamiento que presentan distintos mecanismos de seguridad. Con esto se puede conseguir un punto final de seguridad estandarizado, que ofrezca características de autenticación y autorización para múltiples dispositivos de almacenamiento en un entorno de almacenamiento políglota.

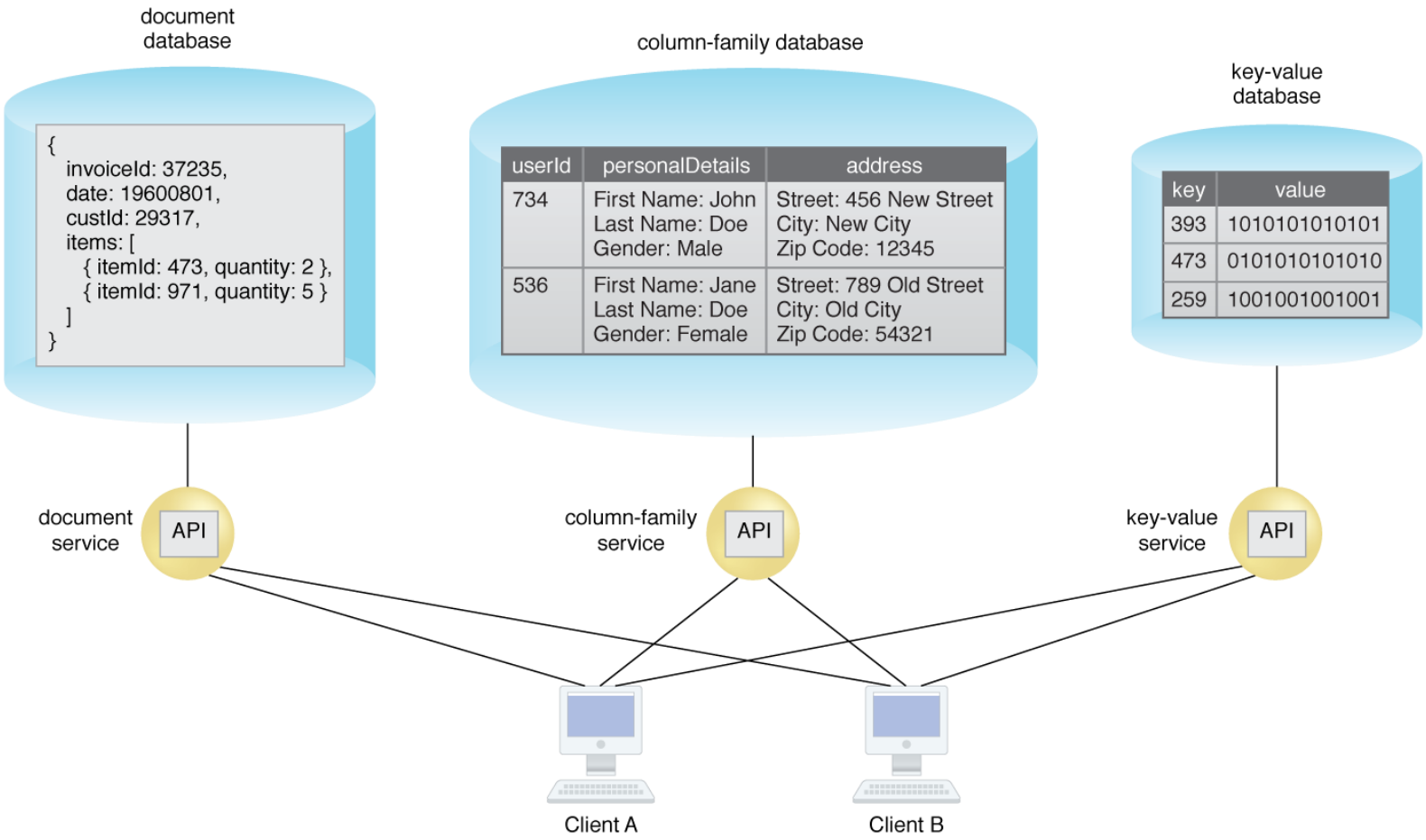


Figura 8.15 – Este diagrama muestra un entorno de persistencia políglota. Con el fin de estandarizar el acceso a los datos, y para asegurar cada dispositivo de almacenamiento, el acceso API correspondiente a cada dispositivo de almacenamiento es encapsulado en un servicio. En lugar de tener acceso individual a los datos, todos los clientes usan el mismo punto final para acceder a un

Algunos dispositivos de almacenamiento son compatibles con el almacenamiento de múltiples tipos de datos; por ejemplo, datos orientados a documentos y datos de llave-valor (key-value) (Figura 8.16), y su introducción en un entorno de persistencia políglota puede eliminar el uso de varios dispositivos de almacenamiento, lo cual simplifica la arquitectura de almacenamiento.

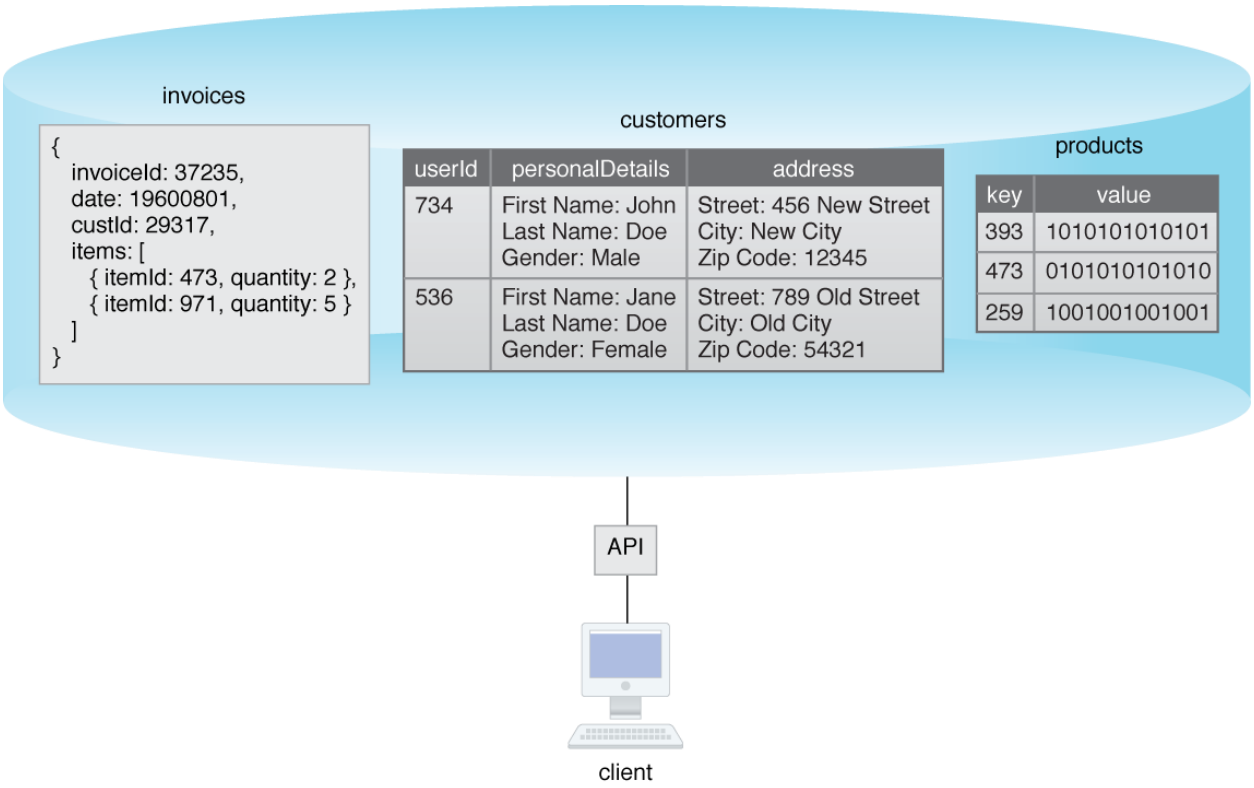


Figura 8.16 – Un único dispositivo de almacenamiento NoSQL permite el almacenamiento de datos sin estructurar, semiestructurados y estructurados.

Lectura

En las páginas 98 a 102 del libro *Imperativos de Big Data* incluido en este módulo, encontrará un análisis más detallado sobre la persistencia políglota.

Procesamiento de Big Data en tiempo real

En los módulos 2 y 7 se examinaron varios conceptos fundamentales y la terminología relacionada con el procesamiento de Big Data. Estos incluyeron clusters, sistemas de archivos distribuidos, procesamiento de datos distribuidos, procesamiento de datos paralelos, procesamiento de cargas de trabajo, paralelismo de datos y paralelismo de tareas. Todo lo anterior conforma una base sólida para la comprensión del procesamiento de datos por lotes (Batch Processing) en los entornos Big Data.

Esta sección presenta el procesamiento en tiempo real de Big Data y su funcionamiento, y analiza si el uso de MapReduce es necesario o no. Antes de analizar el procesamiento en tiempo real de Big Data, se aborda un principio fundamental del procesamiento de Big Data, denominado velocidad, consistencia y volumen (SCV por sus siglas en inglés).

Velocidad, Consistencia, Volumen (SCV)

Mientras que el teorema CAP analiza las propiedades del almacenamiento de datos, el teorema SCV (Figura 8.17) está relacionado con el procesamiento de datos distribuidos, y establece que un sistema de procesamiento de datos distribuidos puede estar diseñado de manera que cumpla solo dos de los siguientes tres requisitos:

- Velocidad: se refiere a qué tan rápidamente pueden ser procesados los datos después de ser generados. En el caso del análisis en tiempo real, los datos se procesan relativamente más rápido que los análisis por lotes. Esto generalmente excluye el tiempo empleado para capturar los datos y se centra únicamente en el procesamiento de datos reales, tal como la generación de estadísticas o la ejecución de un algoritmo.
- Consistencia: se refiere a la exactitud y precisión de los resultados. Se considera que los resultados son exactos cuando están cerca del valor correcto, y precisos si son cercanos entre sí. Un sistema más consistente hará uso de todos los datos disponibles, lo que tendrá como resultado alta precisión y exactitud, en comparación con un sistema menos consistente que utilice técnicas de muestreo, lo cual puede generar menor exactitud y un nivel apenas aceptable de precisión.
- Volumen: es la cantidad de datos que pueden ser procesados. La característica de velocidad de Big Data genera datasets de rápido crecimiento, lo que lleva a que se deban procesar volúmenes enormes de datos de manera distribuida. Procesar tal volumen de datos completamente, garantizando su velocidad y consistencia, se convierte en un problema.

Si la velocidad (S) y la consistencia (C) son necesarias, quizás no sea posible procesar grandes volúmenes de datos, puesto que el procesamiento de enormes volúmenes de datos ralentiza dicho proceso.

Si la consistencia (C) y el procesamiento de grandes volúmenes de datos (V) son necesarios, entonces quizás no sea posible procesar los datos a alta velocidad (S), dado que lograr un procesamiento de datos a alta velocidad requiere de volúmenes de datos más pequeños.

Si el procesamiento de datos de gran volumen (V) junto con el procesamiento de datos a alta velocidad (S) son necesarios, entonces quizás los resultados procesados pueden no ser

consistentes (C) , dado que el procesamiento a alta velocidad de grandes cantidades de datos implica el muestreo de los datos, lo que puede reducir la consistencia.

Cabe señalar que, la decisión sobre a cuáles dos de las tres dimensiones se les debe dar prioridad depende completamente de los requisitos del sistema.

En los entornos Big Data, es obligatorio disponer de la máxima cantidad de datos para realizar un análisis profundo; por ejemplo, identificar un patrón. Así, se debe considerar cuidadosamente si prescindir del volumen (V) para favorecer la velocidad (S) y la consistencia (C), es posible que aún se sea necesario el procesamiento por lotes (Batch Processing) de los datos para obtener información adicional.

En el caso del procesamiento de Big Data, suponiendo que la pérdida de datos (V) sea inaceptable, generalmente un sistema de análisis de datos en tiempo real será SV o CV, dependiendo de si se necesita contar con velocidad (S) o resultados consistentes (C).

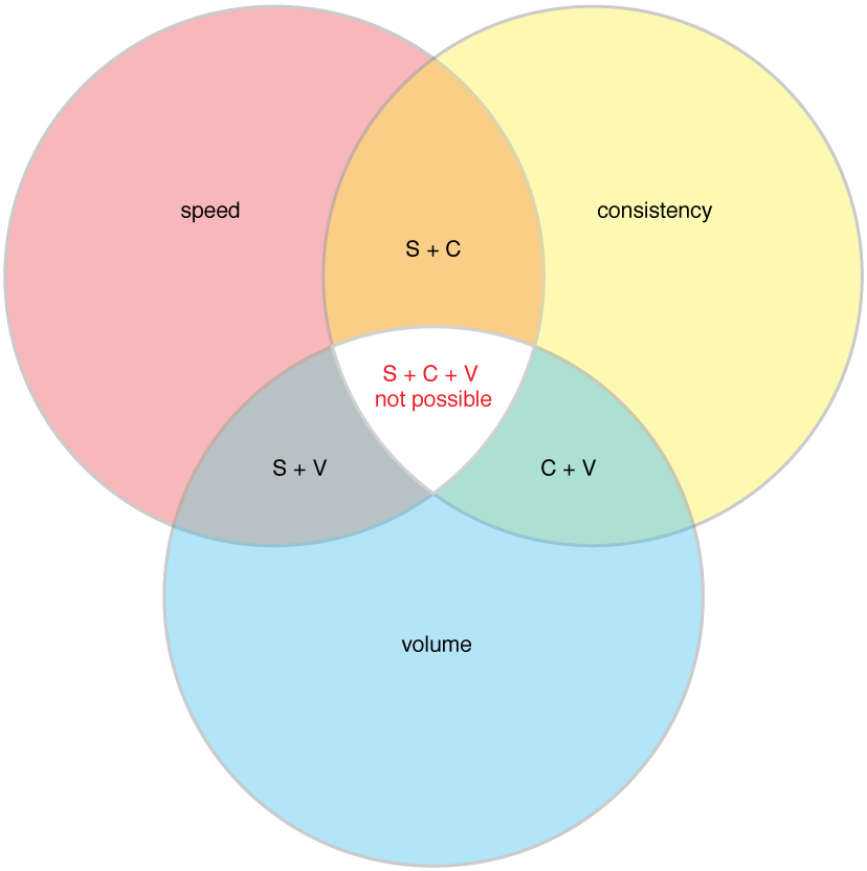


Figura 8.17 – Diagrama de Venn que resume el principio SCV.

Procesamiento de Big Data en tiempo real

El procesamiento en tiempo real de Big Data generalmente se refiere a un análisis (prácticamente) en tiempo real. Los datos son procesados a medida que son recibidos en las empresas sin retrasos injustificados. En lugar de almacenar inicialmente los datos en el disco,

por ejemplo en una base de datos, los datos primero son procesados en la memoria y, a continuación, son almacenados en el disco para su uso futuro o con fines de archivo. Este proceso es contrario al método de procesamiento por lotes (Batch Processing), en el que los datos se almacenan en el disco primero, y posteriormente son procesados, lo que puede llevar a retrasos significativos.

El análisis de Big Data en tiempo real requiere el uso de dispositivos de almacenamiento en memoria (IMDG o IMDB). Una vez en la memoria, los datos pueden ser procesados en tiempo real sin incurrir en ningún tipo de latencia de E/S por parte del disco. El procesamiento en tiempo real puede implicar cálculos estadísticos simples, la ejecución de algoritmos complejos o la actualización del estado de los datos en memoria como resultado de los cambios en algunas métricas de datos.

Para un análisis de datos (Data Analysis) mejorado, los datos en memoria pueden ser combinados con datos previamente procesados por lotes (Batch Processing) o datos no normalizados cargados desde dispositivos de almacenamiento en disco. Esto ayuda a mejorar el procesamiento en tiempo real de los datos, dado que los datasets pueden integrarse en la memoria.

Aunque el procesamiento en tiempo real de Big Data generalmente se refiere a los nuevos datos recibidos, también puede incluir consultas ejecutadas en datos antiguos ya conservados, lo cual requiere una respuesta interactiva. Una vez que se han procesado los datos, los resultados del procesamiento pueden ser mostrados a los consumidores interesados, por ejemplo mediante una aplicación de tablero de control (Dashboard) en tiempo real o una aplicación web que proporcione actualizaciones en tiempo real para el usuario.

Dependiendo de los requisitos del sistema, los datos procesados y sin procesar pueden ser descargados en el dispositivo de almacenamiento en disco para fines de un análisis complejo de datos por lotes, utilizando el enfoque síncrono write-through o el enfoque asíncrono write-behind.

La Figura 8.18 ilustra los siguientes pasos:

1. Se capturan los datos de streaming a través de un motor de transferencia de datos.
2. Posteriormente se guardan de forma síncrona en un dispositivo de almacenamiento en memoria (a) y un dispositivo de almacenamiento en disco (B).
3. Posteriormente, se usa un motor de procesamiento para procesar los datos en tiempo real.
4. Finalmente, los resultados se envían a un tablero de control (Dashboard) para su análisis operativo.

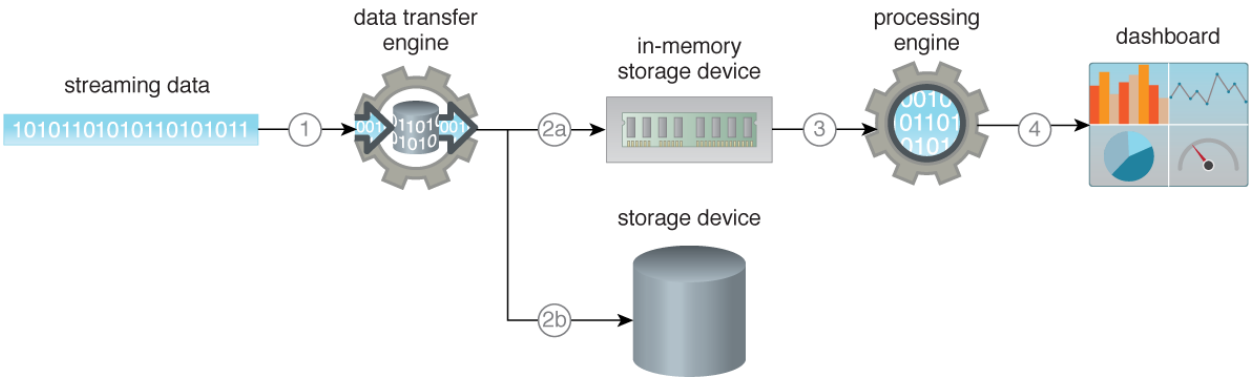


Figura 8.18 – Ejemplo de procesamiento de datos en tiempo real en un entorno Big Data.

Dos conceptos importantes en el procesamiento en tiempo real de Big Data son:

- Procesamiento de flujo de eventos (ESP)
- Procesamiento de eventos complejos (CEP)

Procesamiento de flujo de eventos (ESP)

El flujo entrante de eventos, generalmente originados por una sola fuente y ordenados cronológicamente, se analiza constantemente a través de la aplicación de algoritmos que se basan principalmente en fórmulas o a través de la rápida ejecución de consultas, antes ser almacenados en un dispositivo de almacenamiento en disco.

Otras fuentes de datos (residentes en memoria) también pueden ser incorporadas para realizar un análisis más detallado, al tiempo que los resultados del procesamiento pueden integrarse en el tablero de control (Dashboard) o pueden actuar como un activador de otra aplicación que realice una acción preconfigurada o análisis posterior. Esto se centra más en la velocidad que en la complejidad; la operación que se ejecuta es relativamente simple para fines de una ejecución más rápida.

Procesamiento de eventos complejos (CEP)

Una serie de eventos en tiempo real, a menudo provenientes de fuentes distintas y distribuidos entre diferentes intervalos de tiempo, son analizados simultáneamente para descubrir patrones e iniciar alguna acción. Se aplican los algoritmos basados principalmente en reglas y las técnicas de estadística, y la lógica y procesos empresariales son tomados en cuenta para descubrir patrones transversales de eventos complejos.

El CEP se centra más en la complejidad, proporcionando un análisis detallado. Sin embargo, como resultado, la velocidad de ejecución puede verse afectada negativamente. En general, el CEP se considera un superconjunto de ESP, y en muchas ocasiones los datos de salida de ESP tienen como resultado la generación de eventos sintéticos que pueden ser enviados al CEP.

Procesamiento en tiempo real de Big Data y SCV

Al diseñar un sistema de procesamiento en tiempo real de Big Data, se debe tener en cuenta el principio de SCV. En vista de este principio, consideremos un sistema de procesamiento de Big Data en tiempo real y otro en tiempo prácticamente real. Para ambos casos, tanto en tiempo real como en tiempo prácticamente real, suponemos que la pérdida de datos es inaceptable; en otras palabras, se requiere del procesamiento de grandes volúmenes de datos (V) en ambos sistemas.

Tenga en cuenta que el requisito de que no debe ocurrir la pérdida de datos no significa que todos los datos serán realmente procesados en tiempo real (eso se explicará en breve). En lugar de eso, significa que el sistema captura todos los datos de entrada y que los datos siempre se conservan en el disco, bien sea indirectamente desde el almacenamiento en memoria o directamente en el dispositivo de almacenamiento en disco.

En el caso de un sistema de tiempo real, es necesaria una respuesta rápida (S); por lo tanto, se pondrá en riesgo la consistencia (C) si se requiere procesar un gran volumen de datos (V) en la memoria, debido a la utilización de algunas técnicas de muestreo o de aproximación (resultados menos exactos con una precisión tolerable).

En el caso de un sistema en tiempo prácticamente real, se requiere una respuesta relativamente rápida (S); por lo tanto, se puede garantizar la consistencia (C) si se necesita procesar un gran volumen de datos (V) en la memoria. Los resultados serán más precisos si se comparan con el sistema en tiempo real, puesto que el algoritmo puede ser aplicado para el dataset completo, en lugar de recopilar muestras o emplear cualquier técnica de aproximación.

Esto demuestra que, en el contexto del procesamiento de Big Data, un sistema en tiempo real requiere poner en riesgo la consistencia (C) para garantizar una respuesta rápida (S), mientras que en un sistema en tiempo prácticamente real, es necesario poner en riesgo la velocidad (S) para garantizar resultados consistentes (C).

Procesamiento en tiempo real de Big Data y MapReduce

MapReduce es un motor de procesamiento basado en lotes, que es ideal para el procesamiento de grandes cantidades de datos en reposo. Sin embargo, no puede procesar datos de manera gradual y solo puede procesar datasets completos. Además, requiere que todos los datos de entrada estén disponibles en su totalidad antes de ejecutar la función de procesamiento de datos. En otras palabras, los datos tienen que ser estáticos, mientras que el procesamiento de datos en tiempo real comprende datos que a menudo son incompletos y continuos, como en el caso de los flujos.

En MapReduce, una tarea de reduce generalmente no puede empezar antes de que finalicen todas las tareas de mapeo. En primer lugar, el mapeo de salida se almacena localmente en cada nodo que ejecute la función de mapeo. A continuación, el mapeo de salida se copia a través de la red a los nodos que ejecutan la función reduce, incorporando la latencia de procesamiento. Asimismo, los resultados de un reductor no pueden ser enviados directamente a otro reductor; en lugar de eso, los resultados deben pasar primero por un mapeador.

Las deficiencias mencionadas, junto con la sobrecarga asociada con la creación de trabajos y coordinación, hacen que MapReduce no sea por lo general óptimo para el procesamiento en tiempo real de Big Data. Sin embargo, existen algunas estrategias que pueden permitir el uso de MapReduce en casos de procesamiento en tiempo prácticamente real de Big Data.

Una estrategia es utilizar un dispositivo de almacenamiento de memoria, a fin de evitar que se ejecuten consultas interactivas compuestas por trabajos de MapReduce en los datos de entrada. Alternativamente, se pueden implementar trabajos de MapReduce por microlotes que estén configurados para ejecutarse en datasets relativamente más pequeños, almacenados en la memoria o en un disco, a intervalos frecuentes de quince minutos. Otro método consiste en ejecutar continuamente trabajos de MapReduce en datasets completos almacenados en dispositivos de almacenamiento en disco para crear *vistas materializadas*, que luego pueden ser combinadas con resultados de poco volumen obtenidos a partir de los nuevos flujos de datos en memoria para procesar las consultas de forma interactiva.

Cabe destacar algunos proyectos de código abierto, por ejemplo, Apache Spark, Apache Tormenta y Apache Tez, que proporcionan capacidades de procesamiento en tiempo real de Big Data.

Lectura

En la sección *Análisis en tiempo real*, en las páginas 227 y 228 del libro de texto *Imperativos de Big Data* incluido en este módulo, encontrará un análisis más detallado sobre este tema.

Ejercicio 8.3: identifique cuáles afirmaciones son falsas y cuáles son verdaderas

Indique cuáles de las siguientes afirmaciones son verdaderas y cuáles son falsas:

1. La persistencia políglota proporciona un entorno de almacenamiento óptimo. Sin embargo, cuando se utiliza la persistencia políglota, el manejo de los diferentes tipos de dispositivos de almacenamiento se puede convertir en un problema. (Verdadera/Falsa)

2. La persistencia políglota se refiere a un entorno de almacenamiento heterogéneo en el que es necesario tener más de un dispositivo de almacenamiento. (Verdadera/Falsa)

3. La persistencia políglota se refiere a utilizar solamente tipos diferentes de bases de datos NoSQL. (Verdadera/Falsa)

4. En un entorno de persistencia políglota, la implementación de los servicios no solo estandariza el acceso de las API a múltiples dispositivos de almacenamiento, sino que también ofrece medios para proteger los dispositivos de almacenamiento. (Verdadera/Falsa)

5. El principio SCV solo aplica para sistemas de procesamiento de datos distribuidos. (Verdadera/Falsa)

6. En vista del principio de SCV, dentro de un entorno Big Data, una aplicación de procesamiento de datos solo puede ser SC o CV. (Verdadera/Falsa)

7. El procesamiento en tiempo real de Big Data se refiere exclusivamente al procesamiento de datos que acaban de ser procesados, y excluye datos históricos procesados anteriormente. (Verdadera/Falsa)

8. El procesamiento de flujo de eventos (ESP) generalmente se centra en el análisis de una única fuente y los eventos ordenados cronológicamente, e implica también la aplicación de algoritmos relativamente simples. (Verdadera/Falsa)

9. El Procesamiento de Eventos Complejos (CEP) implica el análisis de una serie de eventos provenientes de distintas fuentes, y utiliza algoritmos relativamente complejos. (Verdadera/Falsa)

10. En MapReduce, una tarea de reduce generalmente no puede empezar antes de que finalicen todas las tareas de mapeo. Los resultados de la tarea de mapeo se conservan localmente en cada nodo que ejecute la función de mapeo, antes de ser copiados desde la red hasta los nodos mediante la función reduce, incorporando la latencia de procesamiento. (Verdadera/Falsa)

Notas

Notas / Bocetos

Diseño avanzado del algoritmo de MapReduce

El módulo 7 presentó pautas de diseño fundamentales para crear algoritmos útiles para el framework de MapReduce. El módulo 8 hace uso de estos conceptos y explora más a fondo las características de los problemas para los cuales es apropiada una solución basada en MapReduce.

En general, el problema que debe ser solucionado a través de MapReduce es un problema lamentablemente paralelo. Este es un problema que puede dividirse fácilmente en sub-problemas, a fin de que estos puedan ser solucionados espontáneamente sin requerir ningún tipo de comunicación entre ellos.

Las tareas que se ejecutan como parte del algoritmo no deberían requerir un estado compartido, ni deberían requerir la transmisión de mensajes entre ellas a fin de calcular el resultado. Los algoritmos que requieren acceso a los resultados intermedios o al conjunto de valores previos, tales como los algoritmos iterativos, generalmente no son adecuados para el framework de MapReduce. Los algoritmos de varias fases necesitan ejecutarse como una serie de varios trabajos, lo cual puede lograrse usando un motor de flujo de trabajo (Workflow) o encadenando múltiples trabajos a través de un programa controlador.

Los problemas que deben ser solucionados se basan generalmente en la agregación, de tal manera que los resultados de las tareas individuales se combinan de alguna manera, aunque algunos podrían no estar relacionados con la agregación (una función de solo mapeo que no incluya ningún reductor). Un ejemplo es el reconocimiento facial, donde un algoritmo de reconocimiento de imagen se aplica a cada imagen en el mapeador.

Los problemas que se deben solucionar no deben ser iterativos y no deben requerir la actualización de estados ni la ejecución de un procesamiento condicional con base en el estado actual. Por ejemplo, para calcular la distancia entre entidades en datos basados en grafos donde cada vértice pueda tener un estado diferente (no procesado, en procesamiento, procesado).

Algunos algoritmos iterativos de aprendizaje automático (Machine Learning), como el agrupamiento (clustering) y la clasificación, se implementan como trabajos de MapReduce, dado que un solo trabajo de MapReduce puede acomodar solamente una iteración a la vez. Por ejemplo, la clasificación mediante el algoritmo apriori implica encontrar elementos individuales que se repiten frecuentemente, y posteriormente parejas de elementos, entre otros, antes de finalmente combinar los resultados de cada iteración para hallar las reglas de generación. La iteración requerida por este algoritmo requiere múltiples trabajos de MapReduce.

El tipo de datos de las llaves y los valores especificados como datos de entrada de la función reduce deben coincidir con el tipo de datos de las llaves y los valores escritos a partir de la función de mapeo. Todos los trabajos de MapReduce deben tener un mapeador. En otras palabras, los resultados de salida de un reductor no pueden convertirse en los datos de entrada de otro reductor, lo que requiere la creación de un mapeador ficticio para que los datos puedan ser transmitidos al reductor.

MapReduce funciona bien para tareas de procesamiento de datos donde se realizan operaciones sobre los datos contenidos en el mismo agregado (el registro leído por el mapeador), tales como

el precio unitario multiplicado por el número de unidades vendidas. Por otro lado, las operaciones que requieren acceso a los valores contenidos en otros registros no son adecuadas para MapReduce. Por ejemplo, encontrar la correlación entre dos valores de llave diferentes (cada llave en registros separados), por ejemplo, al tratar de hallar la correlación entre los tableros de cotizaciones.

Lograr que los algoritmos sean adecuados para MapReduce puede requerir cambios en la estructura de datos o alguna clase de preprocesamiento, de forma que todos los valores requeridos estén en el mismo registro, lo cual podría lograrse mediante un trabajo de MapReduce.

Al momento de diseñar el algoritmo, hay que comprobar si la operación realizada sobre los datos cumple con la ley distributiva, que establece que una operación es distributiva si el resultado obtenido al realizar una operación sobre un número es el mismo que al realizar la operación en partes de ese número y, a continuación, agregar el resultado. Por ejemplo, tal como se ilustra en la figura 8.19, 2 x 7 tiene el mismo resultado que 2 x (3+4) o (2 x 3) + (2 x 4).

$$2 \times 7 = 2 \times (3+4) = (2 \times 3) + (2 \times 4)$$

Figura 8.19 – Ejemplo de una ecuación que cumple la ley distributiva.

Según la propiedad distributiva, en lugar de aplicar una función a un campo en todos los registros, podemos primero aplicar la función a los subconjuntos de los registros, y luego aplicarla a los resultados individuales obtenidos de cada subconjunto. Por ejemplo, consideremos un archivo distribuido entre dos nodos compuestos por lecturas hipotéticas de un medidor inteligente tomadas cada quince minutos, representados en los campos de la Tabla 8.1:

node	meter id	timestamp	load (watts)	min voltage	max voltage
1	A	20120515:0600	500	107.80	112.50
1	B	20120515:0600	800	108.75	111.15
2	A	20120515:0615	650	109.25	113.50
2	B	20120515:0615	750	107.15	112.35

Tabla 8.1– Tabla compuesta por lecturas de un medidor inteligente.

Para calcular la carga máxima para todos los hogares, si todas las lecturas están en un solo archivo, podemos hallar fácilmente la carga máxima mediante el procesamiento simultáneo de todos los valores de carga (Figura 8.20).

$$\text{Max}(500, 800, 650, 750)=800$$

Figura 8.20 – La carga máxima se calcula procesando simultáneamente todos los valores de carga.

Sin embargo, en el caso de MapReduce, podemos encontrar la carga máxima para las lecturas dentro de un único nodo primero en el trabajo de mapeo, antes de encontrar la carga máxima real con base en los resultados intermedios de carga máxima de los dos nodos en la función reduce (Tabla 8.2).

	node 1	node 2
map	Max(500, 800)=800	Max(650, 750)=750
reduce	Max(800, 750)=800	

Tabla 8.2 – Carga máxima encontrada para las lecturas en cada nodo.

Aunque en este escenario funciona para hallar la carga máxima, no es posible distribuir el cálculo de la carga media de manera similar, ya que al analizar la media de los promedios no se obtiene el mismo resultado que al tomar los promedios de todos los números simultáneamente. En el caso de hallar la carga promedio, se debe modificar el diseño del algoritmo, a fin de que la función reduce no solo tenga acceso a la suma de todos los números, sino también a la cantidad de números que integran esa suma.

Es importante reducir la cantidad de datos enviados desde los mapeadores hasta los reductores tanto como sea posible, puesto que la comunicación dentro del cluster representa una de las principales sobrecargas que contribuyen a la latencia general en la ejecución de trabajos. La cantidad de datos enviados desde los mapeadores hacia los reductores puede disminuirse con sumas locales, realizando esta operación localmente dentro del mapeador y enviando únicamente el resultado al reductor, en donde todos los resultados pueden ser agregados de alguna manera para llegar al resultado deseado.

Por ejemplo, para un algoritmo de conteo de palabras, en lugar de simplemente extraer cada palabra y escribir un conteo de *uno* (una variable constante con un valor de uno), el mapeador puede ser optimizado en todos los registros de entrada para que escriba el conteo total para cada palabra (como un combinador). En la mitad superior de la figura 8.21 se presenta un mapeador sin optimizar que genera nueve pares llave-valor (key-value), en comparación con la mitad inferior, donde un mapeador optimizado genera siete pares llave-valor (key-value).

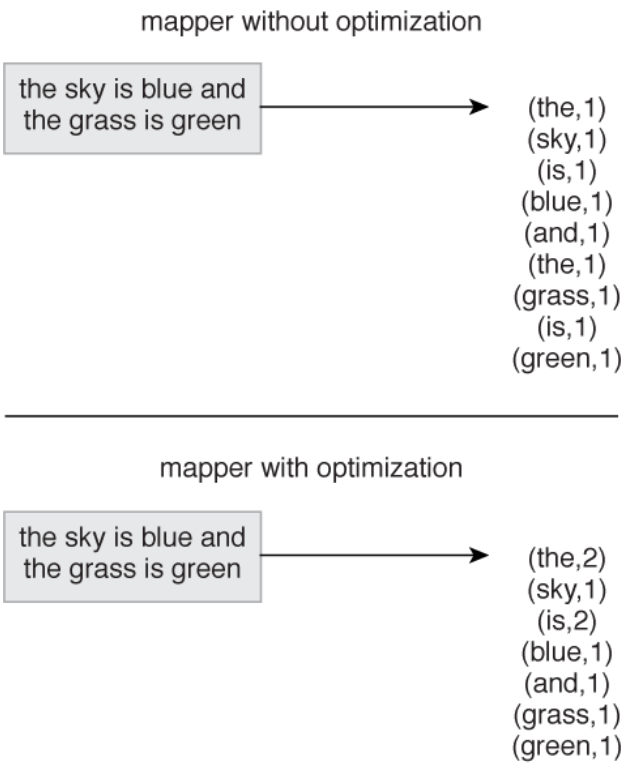


Figura 8.21 - Algoritmo de conteo de palabras de MapReduce, con una versión de mapeador optimizado y una versión de mapeador sin optimizar.

La agregación local también puede aplicarse usando un combinador que intercepte los datos de salida del mapeador y proporcione acceso a todos los valores para cada llave única en los datos de salida del mapeador, de modo que los valores puedan agregarse de cierta manera a través de una operación, antes de enviar el resultado agregado al reductor como un valor único para cada llave.

El tipo de datos de las llaves y los valores especificados como datos de entrada y de salida de la función combinar deben coincidir con el tipo de datos de las llaves y los valores escritos en la función de mapeo. Cuando se usa un combinador, la operación aplicada al dataset debe cumplir las propiedades conmutativa y asociativa.

Una operación que cumple la propiedad asociativa es aquella cuyo resultado sigue siendo el mismo, independientemente de la forma en que estén agrupados los números. Por ejemplo, (2 x 3) x 4 tiene el mismo resultado que 2 x (3 x 4), como se muestra en la figura 8.22.

$$(2 \times 3) \times 4 = 2 \times (3 \times 4)$$

Figura 8.22 – Ecuación que cumple la propiedad asociativa.

Una operación que cumple la propiedad conmutativa es aquella cuyo resultado sigue siendo el mismo, independientemente del orden en el que aparezcan los números. Por ejemplo, 3 x 4 tiene el mismo resultado que 4 x 3, como se muestra en la figura 8.23.

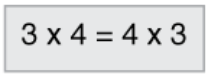


Figura 8.23 – Ecuación que cumple la propiedad conmutativa.

A manera de recordatorio de lo visto en el Módulo 7, no está garantizado totalmente que el combinador se ejecute en el framework de MapReduce. Si el combinador es ejecutado, las agrupaciones y el orden de los valores tomados por el reductor pueden ser diferentes. Sin embargo, los datos de salida del reductor deben ser iguales, independientemente de si el combinador es o no ejecutado. Esto se logra realizando operaciones que cumplan las propiedades conmutativa y asociativa.

Cada combinador aplicará la operación a un grupo de valores disponibles localmente (un solo resultado del mapeador), que corresponde a un único subconjunto del grupo de valores procesados por el reductor. Si el combinador es ejecutado, las agrupaciones vistas por el reductor no serán las mismas. Sin embargo, en caso de que la operación cumpla la propiedad asociativa, su resultado final no se verá afectado por las variaciones en las agrupaciones de datos.

El orden en el que los valores son procesados por el combinador puede ser diferente del orden de los valores en el reductor. Esto se debe a la vista localizada del combinador (un solo resultado del mapeador), en contraposición a la vista global del reductor. Sin embargo, en caso de que la operación cumpla la propiedad conmutativa, su resultado final no se verá afectado por el orden de los valores en los datos.

Lectura

En la sección Patrones de MapReduce en las páginas 162 a 165 del libro *Imperativos de Big Data* incluido en este módulo, encontrará un análisis más detallado sobre este tema.

Motor de procesamiento Bulk Synchronous Parallel

El modelo Bulk Synchronous Paralell (BSP por sus siglas en inglés) es un motor de procesamiento de computación paralela/distribuida altamente escalable, el cual permite la ejecución eficiente de algoritmos de carácter iterativo y que requieren la transferencia de mensajes para procesar grandes cantidades de datos. El modelo BSP fue propuesto por Leslie Valiant en 1990, se ejecuta en un cluster de máquinas y generalmente es usado para el procesamiento de datos de tipo grafos y para ejecutar algoritmos de aprendizaje automático (Machine Learning).

BSP se usa comúnmente como un motor de procesamiento por lotes (Batch Processing), pero también puede usarse para tareas de procesamiento en tiempo real y prácticamente real, ofreciendo de esta manera soporte para múltiples cargas de trabajo. También es compatible con tareas de redundancia y tolerancia a errores mediante puntos de control periódicos y reinicio automático de procesos con errores. Además, siendo un motor de procesamiento exclusivo para la transmisión de mensajes, es compatible con el procesamiento de datos sin esquema.

Bulk Synchronous Parallel: conceptos

Un solo procesamiento de datos en BSP consiste en una serie de *superpasos*, y cada uno comprende tres etapas:

- procesamiento local: mediante múltiples tareas se ejecuta una función definida por el usuario sobre los elementos de datos, ya sea de manera paralela o asíncrona sobre los diferentes nodos, a fin de procesar los datos de entrada de carácter local para cada nodo. En esta etapa también se recopilarán y procesarán los mensajes enviados por otras tareas durante el superpaso anterior.
- comunicación: los datos procesados son transmitidos como mensajes a tareas específicas o a todas las tareas que se ejecuten en el mismo nodo o en diferentes nodos.
- sincronización de la barrera: cualquier tarea que ha finalizado el procesamiento de los datos y el envío de mensajes espera hasta que todas las demás tareas hayan finalizado el procesamiento de datos y el envío de mensajes.

Este superpaso se repite durante un número determinado de veces, o hasta que se cumpla la condición anterior. Los mensajes que se envían durante el mismo superpaso no están disponibles para su uso hasta llegar al próximo superpaso. Por ejemplo, los mensajes enviados en el superpaso s-1 estarán disponibles en el superpaso s, mientras que los mensajes transmitidos en el superpaso s solamente podrán estar disponibles en el superpaso s+1.

Tenga en cuenta que, generalmente, una tarea es ejecutada por un proceso alojado en un procesador lógico. Un procesador físico alberga múltiples procesadores lógicos. Por lo tanto, un cluster de un solo nodo alberga varias instancias de procesamiento, en las que cada proceso ejecuta una tarea de forma paralela y asíncrona para otros procesos.

En la Figura 8.24 se resume la ejecución de un único superpaso en BSP. Varias tareas ejecutan cálculos de forma paralela y distribuida durante la etapa de cálculo local. Los mensajes se envían durante la etapa de comunicación, una vez que cada tarea ha finalizado el cálculo. Durante la etapa de sincronización de la barrera, las tareas que ya finalizaron esperan a las tareas pendientes.

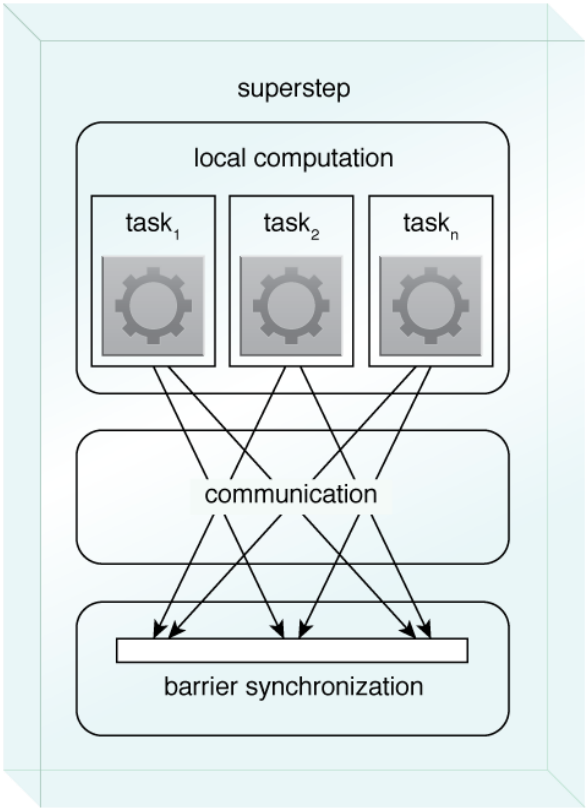


Figura 8.24 – La ejecución de un solo paso en BSP.

Los siguientes dos mecanismos ayudan a mejorar el uso y la eficiencia del motor de procesamiento BSP:

- agregadores
- combinadores

Agregadores

Son tareas con visibilidad global, a partir de las cuales cualquier otra tarea puede enviar y recopilar mensajes. Básicamente se usan para crear resúmenes de datos, tales como mínimos, máximos y sumas. También actúan como un medio para lograr un procesamiento selectivo; por ejemplo, la ejecución condicional del código de acuerdo con el valor actual del agregador y la coordinación global entre tareas.

Combinadores

Múltiples mensajes enviados a una tarea se pueden combinar en un solo mensaje (dependiendo del tipo de algoritmo, como la suma), para así reducir la comunicación en la red y el uso de la memoria. Los combinadores integran los mensajes en un mensaje único. Al igual que MapReduce, no siempre se puede ejecutar una función de combinador, como resultado los combinadores solo deben usarse en caso de que la operación que está siendo aplicada cumpla las propiedades asociativa y conmutativa.

Comparación entre BSP y MapReduce

En MapReduce, un algoritmo iterativo debe ser ejecutado realizando varias veces el mismo trabajo de MapReduce. Esto genera una sobrecarga considerable, puesto que no solo los datos de salida de los mapeadores deben ser guardados en un disco antes de ser enviados a los reductores, sino que los datos de salida intermedios de los reductores también deben ser guardados en un disco como preparación para la próxima iteración del trabajo de MapReduce.

En MapReduce, la sincronización existente entre los mapeadores y reductores se logra de forma implícita al solicitar que todos los mapeadores terminen su procesamiento antes de que los reductores puedan leer sus datos de salida. La sincronización se refiere a alcanzar la uniformidad del sistema, garantizando que las tareas sean ejecutadas en el orden correcto y que una tarea finalice completamente antes de ser ejecutada nuevamente o de que se ejecute una tarea diferente.

MapReduce no permite que se transfieran los mensajes entre mapeadores o reductores. Para obtener los datos de salida de otros mapeadores, durante la etapa de mapeo se escriben los datos de salida intermedios de cada mapeador, los cuales solamente estarán disponibles para otros mapeadores en la próxima iteración del trabajo de MapReduce. Aunque se pueden enviar mensajes, la comunicación entre los mapeadores y reductores es únicamente indirecta. Esto se hace de manera implícita a través del motor de MapReduce, mediante la agrupación de las mismas llaves generadas por los mapeadores, a fin de que puedan ser enviadas en forma de mensajes a los reductores correspondientes. No existe compatibilidad con la comunicación inversa de los reductores a los mapeadores. Una forma de solucionar este problema es primero guardar los datos de salida de los reductores, antes de que puedan ser leídos por los mapeadores durante la próxima iteración de MapReduce.

Enviar un mensaje en MapReduce es algo similar a definir la misma llave en la función de mapeo, de manera que los datos de salida sean transmitidos al mismo reductor. Sin embargo, en medio de las distintas ejecuciones de un trabajo de MapReduce, la misma función reduce podría ser ejecutada en un nodo distinto, debido a que MapReduce no tiene estados.

MapReduce no tiene estados; una vez finalizan las tareas de reduce y mapeo, el resultado se guarda en el disco, y el estado local —por ejemplo, la entrada asignada de las tareas de mapeo y reduce— es eliminado. Cuando se ejecuta varias veces el mismo trabajo de MapReduce, los datos de salida provenientes de cada trabajo ejecutado deben guardarse en un dispositivo de almacenamiento, y posteriormente son releídos como datos de entrada en la tarea de mapeo, durante la próxima ejecución del trabajo de MapReduce.

En MapReduce, la transferencia de datos desde los mapeadores hasta los reductores, así como el requisito de guardar los datos de salida en el disco después de una iteración y antes de la relectura de los datos al comienzo de cada ejecución del trabajo, contribuyen a que el procesamiento de los datos sea ineficiente. Por otro lado, BSP tiene estados; las particiones de los datos de entrada que contienen los elementos de datos son asignadas a cada tarea al inicio, y son actualizadas de forma local a través de múltiples superpasos. Las mismas tareas siguen procesando los mismos elementos de datos en los mismos nodos a través de múltiples superpasos hasta que se logra la condición de parada. Estas tareas también actualizan localmente cualquier dato de estado entre los superpasos.

Un superpaso en BSP equivale a la ejecución de un trabajo de MapReduce. Sin embargo, a diferencia de un trabajo, el superpaso no termina y puede ser iterado varias veces. No obstante, la eficiencia del motor de BSP depende de la capacidad de conservar la partición de los datos de entrada en su totalidad al interior de la memoria de los nodos de procesamiento. Si la partición de datos de entrada no se puede conservar en su totalidad en la memoria, el excedente de los datos se descarga al disco, lo que genera una sobrecarga, que a su vez conlleva a un procesamiento de datos menos eficiente.

Bulk Synchronous Parallel

El motor de procesamiento BSP se puede usar para:

- procesar datasets orientados en relaciones
- procesar datos que requieren una amplia transmisión de mensajes
- procesar datos de forma repetida
- procesar datos rápidamente o en tiempo prácticamente real

El motor de procesamiento de BSP no debe ser usado en los siguientes casos:

- en el procesamiento de datasets conformados por registros basados en datos agregados que no tengan relación con otros registros
- en el procesamiento de datos que impliquen leer los datos de entrada una vez, y en el cual el algoritmo pueda completarse como una sola ejecución
- en caso de que el estado de los datos de entrada permanezca igual durante el procesamiento
- en caso de que los datos asignados a una tarea no pueden guardarse completamente en la memoria principal de los nodos del cluster durante los distintos superpasos

Datos de grafo

Como se analizó brevemente en el módulo 7, los datos de grafos se refieren a un conjunto de entidades conectadas mediante alguna forma de relación. Cada entidad es comúnmente conocida como vértice o nodo, mientras que cada conexión se denomina borde.

En el contexto de los datos de grafo, un nodo representa un elemento de datos o un registro, y no es igual a un nodo de computadora presente en un cluster. Generalmente, cada vértice tiene

una identificación y puede estar conformado por un solo valor, o por múltiples pares llave-valor (key-value). De manera similar, cada borde puede estar conformado de un solo valor, o por múltiples pares llave-valor (key-value).

Tipos de datos de grafo

Un borde puede ser dirigido o no dirigido. En el caso de un borde dirigido, entre los vértices puede existir un borde unidireccional (un recorrido) o un borde bidireccional (dos recorridos). Los bordes unidireccionales son analizados de la misma forma que los bordes bidireccionales.

Se denomina grafo dirigido a un grafo conformado solamente por bordes dirigidos, mientras que se denomina grafo no dirigido a un grafo conformado por bordes no dirigidos, tal como se muestra en la Figura 8.25. Un grafo puede estar conformado por una combinación de bordes dirigidos y no dirigidos.

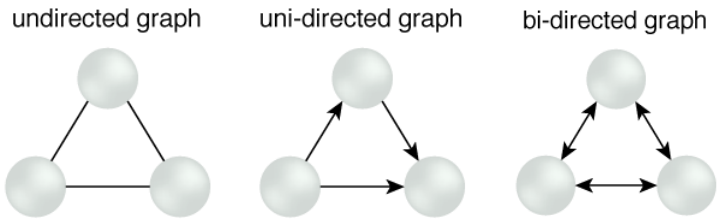


Figura 8.25 – Ejemplos de grafos no dirigido, unidireccional y bidireccional.

Un grafo cíclico es aquel en el que los bordes están distribuidos de tal manera que un vértice puede recorrerse nuevamente desde otro vértice (formando un loop que puede incluir más de dos vértices), tal como se muestra en la Figura 8.26. A un grafo que no presenta loops se le denomina grafo acíclico.

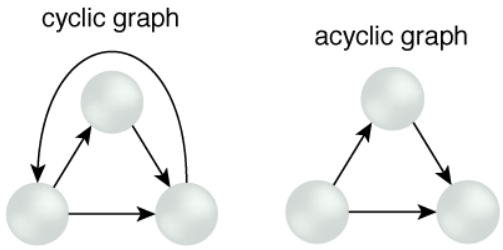


Figura 8.26 – Ejemplos de grafos cíclico y acíclico.

Un grafo ponderado está conformado por bordes que tienen un valor numérico correspondiente que representa la cantidad de recorridos entre ambos vértices. Un grafo que no tiene dichos valores se denomina grafo sin ponderar, tal como se muestra en la Figura 8.27.

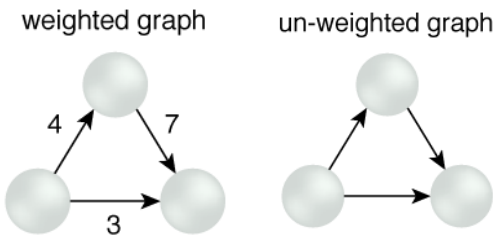


Figura 8.27 – Ejemplos de grafos ponderado y sin ponderar.

En las Figuras 8.28 y 8.29, los círculos y flechas representan los vértices y bordes, respectivamente. Las letras indican la identidad de los vértices, mientras que el valor de los vértices se representa con el número que aparece dentro del círculo. La Figura 8.28 muestra un ejemplo de grafos dirigido, acíclico y ponderado. El número que acompaña cada flecha indica la ponderación del borde. La Figura 8.29 muestra ejemplos de grafos uni/bidireccional, cíclico y sin ponderar.

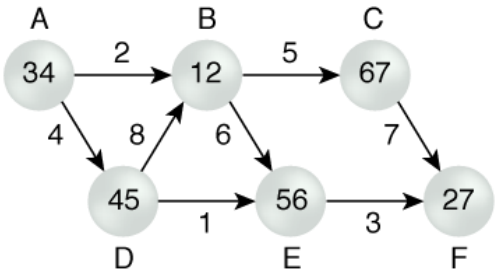


Figura 8.28 – Ejemplos de grafos dirigido, acíclico y ponderado.

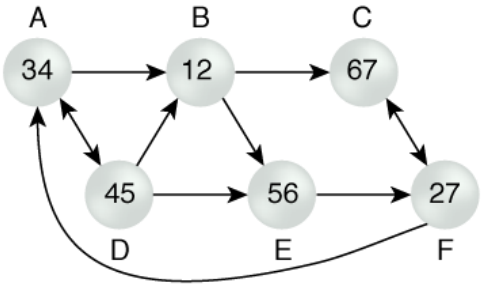


Figura 8.29 – Ejemplos de grafos uni/bidireccional, cíclico y sin ponderar.

Procesamiento de datos de grafos

Los datos de grafos se presentan en diversidad de formas y figuras; por ejemplo, redes sociales, interacciones proteína-proteína, redes de dispositivos y redes de tuberías. Los datos relacionales presentes en un RDBMS también se pueden convertir en datos de grafos, para que así puedan ser procesados usando motores especializados de procesamiento de grafos. Cada fila en una

tabla se convierte en un vértice, y cada llave externa se convierte en un borde. También se pueden agregar bordes basados en llaves no externas.

Aunque el procesamiento de datos de grafos difiere según cada caso, generalmente se caracteriza por el procesamiento iterativo de vértices, en el cual el cálculo de un valor depende de los vértices conectados a un vértice, o de los vértices cercanos a un vértice (no conectados, pero sí cercanos).

Varios algoritmos de aprendizaje automático (Machine Learning) (como k-media para el agrupamiento (clustering)) y otros algoritmos (como el algoritmo de la ruta más corta y el algoritmo de la búsqueda de profundidad) permiten procesar datos de grafos para los cuales el cálculo de los resultados requiere un procesamiento recurrente de los vértices y la transmisión constante de mensajes entre los mismos.

De acuerdo con la característica de volumen, en los entornos Big Data, un grafo puede aumentar su extensión significativamente. Como resultado, no es posible almacenar y procesar grafos extensos con una sola máquina, ya que puede conducir a niveles de desempeño menores a lo ideal.

Debido a su naturaleza de conexión, los datos de grafos son guardados generalmente en la memoria, a fin de lograr un rendimiento de procesamiento óptimo. Esto elimina los cuellos de botella en el acceso frecuente a los discos necesarios para leer los vértices conectados.

Procesamiento de datos de grafos: BSP

Para procesar grafos extensos de manera escalable, es necesario separarlos en múltiples shards y almacenarlos en un cluster. Dicho grafo particionado se puede procesar usando un motor de procesamiento de grafos, que generalmente guarda cada shard en la memoria y los procesa en paralelo, de manera iterativa.

BSP es uno de esos motores que pueden ser utilizados para el procesamiento de grafos extensos de manera eficiente, debido a sus características de procesamiento de datos iterativos y transmisión de mensajes. Cuando se utiliza BSP para el procesamiento de grafos, cada tarea procesa los vértices preasignados a la par con sus bordes, los cuales son conservados en la memoria local durante el procesamiento.

Por lo general, el procesamiento de datos de grafos en el contexto del motor de procesamiento BSP comprende las siguientes etapas:

Preprocesamiento

Cada partición del grafo se divide en múltiples particiones, cada una conformada a su vez por datos que representan múltiples vértices. Los datos de las particiones son analizados a manera de vértices, y posteriormente son asignados a las tareas individuales. Generalmente, en cada nodo del cluster se ejecutan múltiples tareas. Normalmente, el particionamiento de los vértices se puede personalizar de forma que los vértices relacionados sean asignados a la misma tarea, o se mantengan en estrecha proximidad; por ejemplo, dentro de un mismo nodo. En esta etapa inicial, el estado de todos los vértices se fija como activo.

Cálculo local

Durante esta etapa, los vértices analizados se procesan mediante una función definida por el usuario, conocida generalmente como función de cálculo. La funcionalidad se define a nivel de cada vértice, y durante cada superpaso se ejecuta la función de cálculo para cada vértice activo.

El código en la función de cálculo generalmente sigue el patrón de lectura y procesamiento de los mensajes enviados al vértice durante el superpaso anterior (no durante el primer superpaso) y hace los procesamientos necesarios; por ejemplo, ejecutar una operación, cambiar el estado del vértice o sus bordes, o crear nuevos vértices.

Comunicación

Durante la etapa de comunicación, un vértice se comunica con otros enviando mensajes de forma asíncrona. Puede tratarse de vértices conectados al vértice actual por medio de un borde de salida, o puede ser cualquier otro vértice de identidad conocida. Generalmente, estos mensajes están conformados por cargas ligeras, como un único valor calculado. Los mensajes que se envían a un vértice no están disponibles inmediatamente en el mismo superpaso, sino únicamente hasta el superpaso siguiente.

Sincronización de barrera

Cuando un vértice finaliza el trabajo que le fue asignado (procesamiento de datos y transmisión de mensajes), se desactiva y su estado se fija como inactivo. Un vértice inactivo no se procesa en los superpasos posteriores, a menos de que se reactive a causa de los mensajes recibidos durante cualquiera de los superpasos posteriores. La sincronización de barrera se logra una vez todas las tareas hayan procesado sus vértices asignados antes de que se declare como completo el superpaso actual.

Las etapas de cálculo local, comunicación y sincronización de barrera se repiten hasta que ya no haya más vértices que procesar (todos los vértices estén inactivos) y tampoco haya mensajes que enviar. El motor de procesamiento BSP actualiza el estado de los vértices durante todo el procesamiento.

El estado de cada vértice incluye una etiqueta que identifica si dicho vértice está activo o inactivo, su valor de tiempo de ejecución, el valor de tiempo de ejecución para cada borde de salida junto con la identificación del vértice más próximo, además de todos los mensajes recibidos durante el superpaso anterior.

Procesamiento de datos de grafos: ejemplo de BSP

El siguiente es un ejemplo de un procesamiento de datos de grafos usando un motor de procesamiento BSP. Consideremos el grafo de la Figura 8.30, que representa una pequeña red de carreteras. Los vértices muestran las ciudades, mientras que los bordes muestran las carreteras que conectan dichas ciudades. Para fines de simplicidad, todas las carreteras van en una dirección (grafo unidireccional) y no hay una carretera que conecte ninguna ciudad de vuelta con las ciudades anteriores (grafo acíclico). El número que acompaña cada borde representa la distancia correspondiente entre las dos ciudades (grafo ponderado). Se debe calcular la distancia

más corta desde la ciudad de origen (Vértice A) hasta las otras ciudades (Vértices B a H), a las cuales se conecta de manera directa o indirecta.

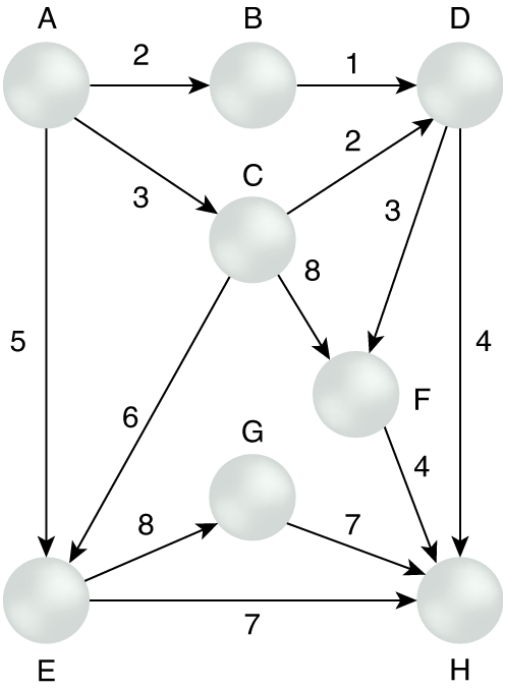


Figura 8.30 – Grafo que representa una pequeña red de carreteras.

El entorno de solución de Big Data incorpora un cluster conformado por dos nodos de procesamiento (Nodo X y nodo Y). Cada nodo de procesamiento ejecuta una tarea de BSP en paralelo. Los vértices desde A hasta D se asignan al nodo X, mientras que los vértices desde E hasta H se asignan al nodo Y.

Todos los vértices son inicializados con un valor infinito (∞), excepto el vértice A, que es inicializado con un valor cero, ya que la distancia con respecto a sí mismo es cero, como se muestra en la Figura 8.31. El valor infinito representa la máxima distancia posible existente entre el vértice y el vértice de origen A, además de indicar que el vértice actual no ha sido recorrido aún (sin procesar). Los vértices activos (en color verde) envían la suma del valor del vértice y el valor del vértice de salida como mensajes encapsulados a los demás vértices conectados. Los vértices inactivos se muestran en color rojo.

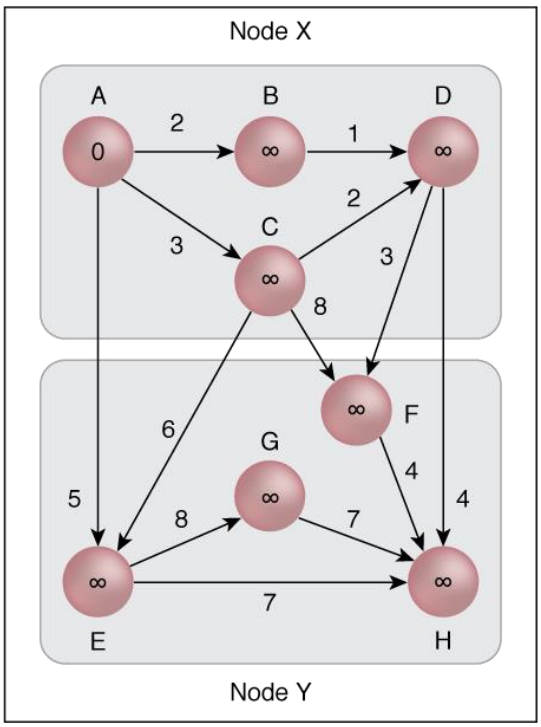


Figura 8.31 – El vértice A es inicializado con un valor cero.

La lógica definida por el usuario e incluida en la función de cálculo opera de la siguiente forma:

- Cada vértice activo procesa todos los mensajes enviados por los demás vértices durante el superpaso anterior (en el primer superpaso no se recibieron mensajes).
- Los valores de los mensajes recibidos (la suma del valor del vértice de origen y el correspondiente valor del borde de salida) se comparan con el valor actual del vértice. Si se encuentra un valor menor, el valor del vértice actual se actualiza con este nuevo valor.
- Posteriormente, todos aquellos vértices cuyos valores son actualizados envían mensajes (la suma del valor del vértice y el valor del borde de salida) a los demás vértices conectados por medio de sus bordes de salida.

Esta lógica se ejecuta nuevamente hasta que ya no haya más mensajes que enviar.

Superpaso S_0

En el primer superpaso, el Vértice A es el vértice activo, como se muestra en la Figura 8.32. Puesto que ya no hay mensajes recibidos que puedan procesarse, el Vértice A simplemente envía los mensajes 2, 3 y 5 a los Vértices B, C y E respectivamente. El Vértice A pasa a un estado inactivo después de enviar todos los mensajes requeridos.

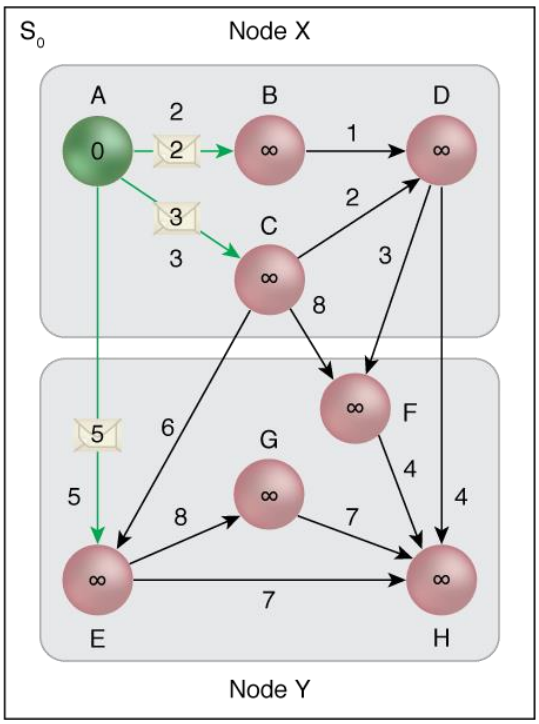


Figura 8.32 – Primer superpaso.

Superpaso S_1

Durante el segundo superpaso, los mensajes enviados por el Vértice A son transmitidos a los Vértices B, C y E, y en consecuencia, estos vértices se activan (Figura 8.33). Luego de la comparación, los valores 2, 3 y 5 son asignados a los Vértices B, C y E, respectivamente. Luego, estos vértices envían mensajes a los Vértices D, E, F, G y H, y pasan a un estado inactivo.

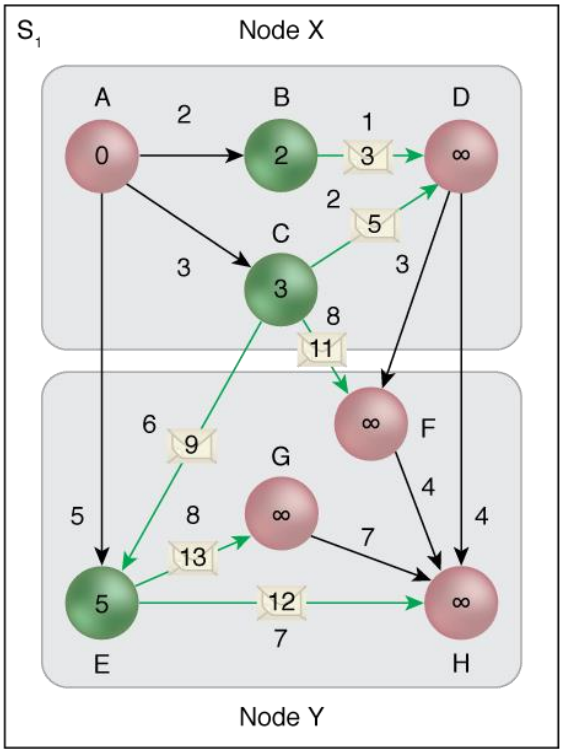


Figura 8.33 – Segundo superpaso.

Superpaso S_2

En el tercer superpaso, los Vértices D, F, G, E, y H se activan, como se muestra en la Figura 8.34. Observe que el Vértice E se reactiva debido al mensaje del Vértice C enviado durante el superpaso anterior. Después de la comparación, los valores 3, 5, 11, 13 y 12 son asignados a los vértices D, E, F, G, y H, respectivamente, tal como se muestra en la Figura 8.34. El valor del vértice E permanece sin cambios, y por lo tanto, no envía ningún mensaje. Aparte de E y H, otros vértices actualizados (D, F y G) envían posteriormente mensajes a los Vértices F y H, y luego entran en un estado inactivo.

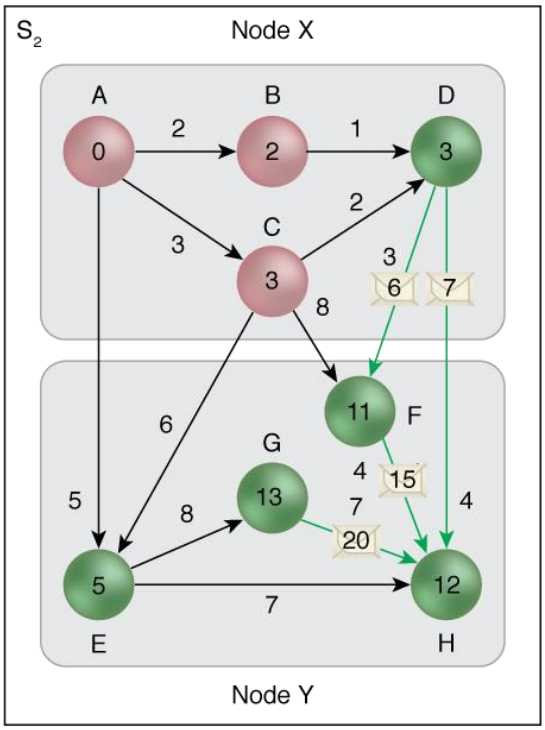


Figura 8.34 – Tercer superpaso.

Superpaso S_3

En el cuarto superpaso, los Vértices F y H se activan, como se muestra en la Figura 8.35. Observe que ambos vértices se reactivan debido a los mensajes que recibieron durante el superpaso anterior. Después de la comparación, los valores 6 y 7 son asignados a los Vértices F y H, respectivamente. Posteriormente, el vértice actualizado F envía un mensaje al Vértice H, y ambos vértices H y F pasan a un estado inactivo.

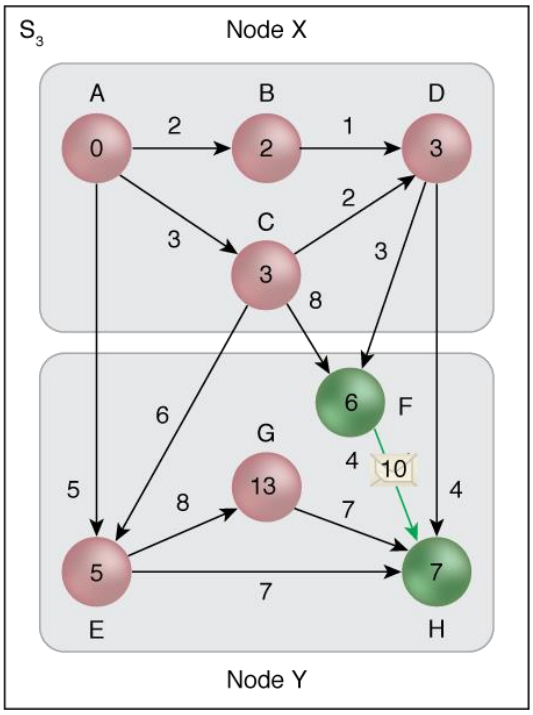


Figura 8.35 – Cuarto superpaso.

Superpaso S_4

En el quinto superpaso, el Vértice H se activa, como se muestra en la Figura 8.36. Observe que el Vértice H se reactiva debido al mensaje que recibió durante el superpaso anterior. Sin embargo, después de la comparación, su valor permanece sin cambios. Puesto que el valor del Vértice H no se actualiza, ni presenta bordes de salida, este simplemente entra en un estado inactivo, sin enviar ningún mensaje.

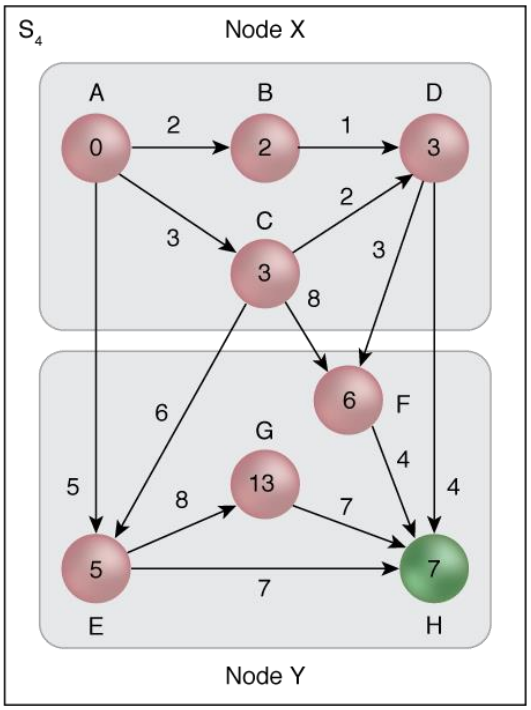


Figura 8.36 – Quinto superpaso.

Como no hay más mensajes que procesar, el procesamiento ha finalizado, y cada vértice transporta un valor que representa la distancia más corta desde la ciudad A hasta el vértice mismo, como se muestra en la Figura 8.37.

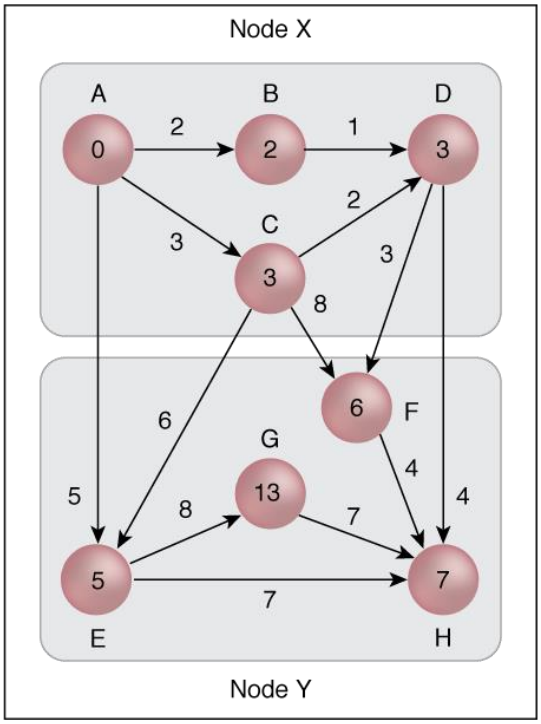


Figura 8.37 – La etapa de procesamiento finaliza cuando no hay mensajes que procesar.

Combinador

La lógica definida en la función de cálculo es ideal para usar la función de combinador. En lugar de reenviar todos los mensajes enviados a un vértice, solamente se reenvía el mensaje con el menor valor, el cual se compara con el valor del vértice. Por ejemplo, los tres mensajes enviados a los vértices H en S_2 son reemplazados con un único mensaje, como se muestra en la Figura 8.38. De forma similar, y aunque no se muestra, los dos mensajes enviados al Vértice D en S_1 son reemplazados con un solo mensaje.

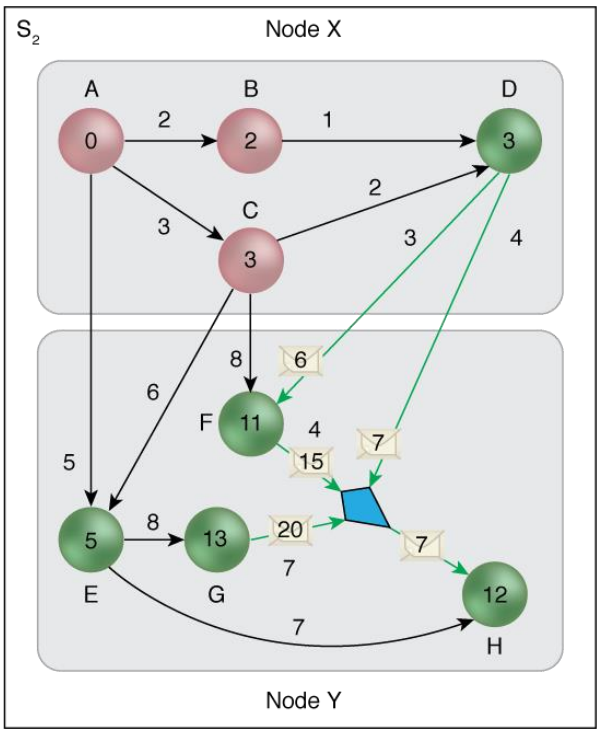


Figura 8.38 – Uso de una función de combinador.

Procesamiento de datos de grafos: ejemplo de BSP (comparación con MapReduce)

Comparado con BSP, el motor de procesamiento de MapReduce procesará el mismo grafo de la siguiente manera (se describe una sola ejecución del trabajo):

Función de mapeo

Los vértices activos pasan al estado inactivo, conservando su valor actual; y posteriormente son enviados a un reductor en forma de un par llave-valor (key-value). Por ejemplo, en S_0, se genera un par llave-valor (key-value) *{A, (0, inactivo)}* para el Vértice A. En cada vértice, se genera un par llave-valor (key-value) para cada uno de sus vértices conectados, en el cual la llave corresponde a la identidad del vértice de destino, y el valor corresponde a la distancia. Este valor también incluye el estado activo. Por ejemplo, en S_0, se generan los pares llave-valor (key-value) *{B, (2, activo)}, {C, (3, activo)}, {E, (5, activo)}* para el Vértice A. Todo vértice inactivo es enviado tal cual al reductor en forma de un par llave-valor (key-value). Por ejemplo, en S_0, se genera el par llave-valor (key-value) *{B, (∞, inactivo)}* para el Vértice B.

Función reduce

El reductor recibe una lista consolidada de los pares de llave-valor (key-value) para cada vértice, incluyendo los vértices activos e inactivos. Para cada vértice, si existe más de un valor en la lista, entonces su valor es actualizado con el menor valor de distancia, y su estado se fija como activo (ya que recibió mensajes desde otros vértices). Por ejemplo, en S_2, la llave H recibe los valores

{(7, activo), (15, activo), (20, activo), (12, inactivo)}, y como resultado, el valor del Vértice H es actualizado con 7 y estado activo.

Si solo hay un valor en la lista, esto significa que el vértice correspondiente no recibió ningún mensaje. Este vértice es escrito tal como está. Por ejemplo, en S_2, la llave C recibe un valor *{(3, inactivo)}*, y como resultado, el valor del Vértice C permanece como 3, como estado inactivo.

Ejercicio 8.4: organice las etapas de BSP

Organice las etapas de BSP a continuación en el orden correcto:

sincronización de barrera	1. _____
cálculo local	2. _____
preprocesamiento	3. _____
comunicación	4. _____

Notas

Notas / Bocetos

Big Data Pipeline y ELT

Pipeline de datos

Un pipeline de datos es un flujo de trabajo (Workflow) controlado por datos y compuesto por tareas, cada una de las cuales comprende los datos de entrada, la operación y los datos de salida, como se muestra en la Figura 8.39.

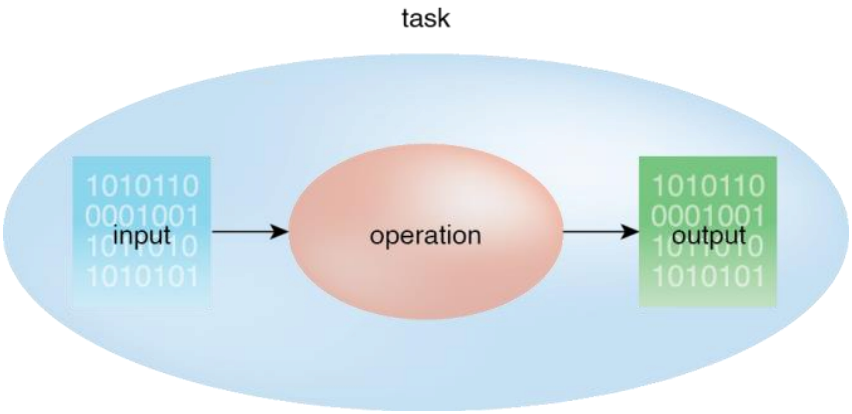


Figura 8.39 – Ejemplo de una tarea que comprende los datos de entrada, la operación y los datos de salida.

Cada pipeline de datos consta de varias tareas integradas de manera serializada, de modo que los datos de salida de una tarea se convierten en los datos de entrada de la tarea posterior, como se muestra en la Figura 8.40. Dicha combinación de tareas representa una etapa individual.

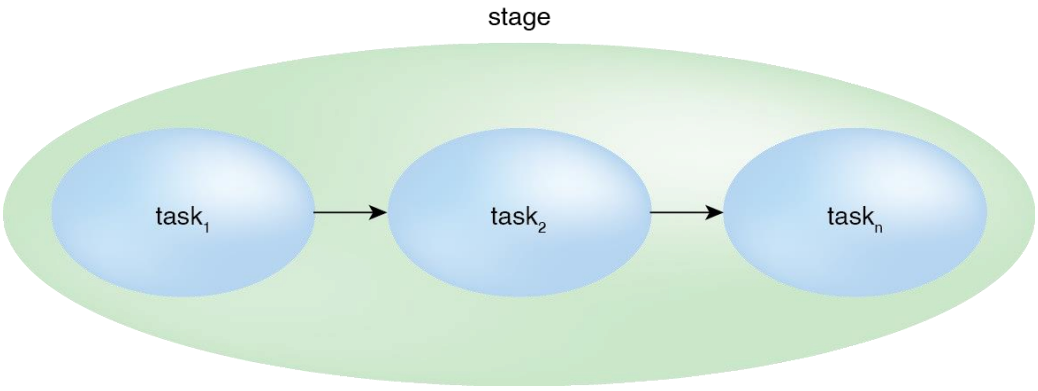


Figura 8.40 – Ejemplo de una etapa individual integrada por tareas.

Un pipeline de datos puede constar de un número de etapas, cuyo resultado es que los datos son leídos desde una fuente y se realizan algunas operaciones (durante varias etapas) en los datos de entrada y, a continuación, se escriben los datos procesados. De igual forma que un pipeline real, el pipeline de datos se usa para mover datos entre la fuente y el sink de manera automática, como se muestra en la Figura 8.41. Esto se logra mediante la ejecución de distintas operaciones, tales como limpieza (cleansing), validación, joining y transformación.

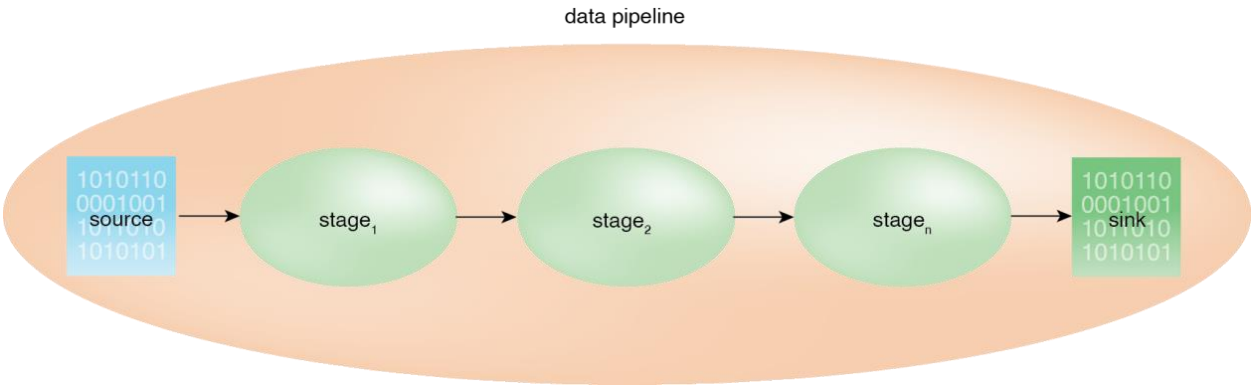


Figura 8.41 – Ejemplo de un pipeline de datos integrado por etapas.

Big Data Pipeline

Un Big Data Pipeline está conformado generalmente por múltiples etapas, en las que el procesamiento complejo está dividido en etapas modulares para facilitar la actualización y la adaptación de futuros requerimientos de procesamiento. Por lo general, cada tarea en un Big Data Pipeline utiliza un mecanismo de motor de procesamiento, tal como MapReduce o Spark, o un mecanismo de motor de consultas (Query Engine), tal como Hive o Pig, como se muestra en la Figura 8.42.

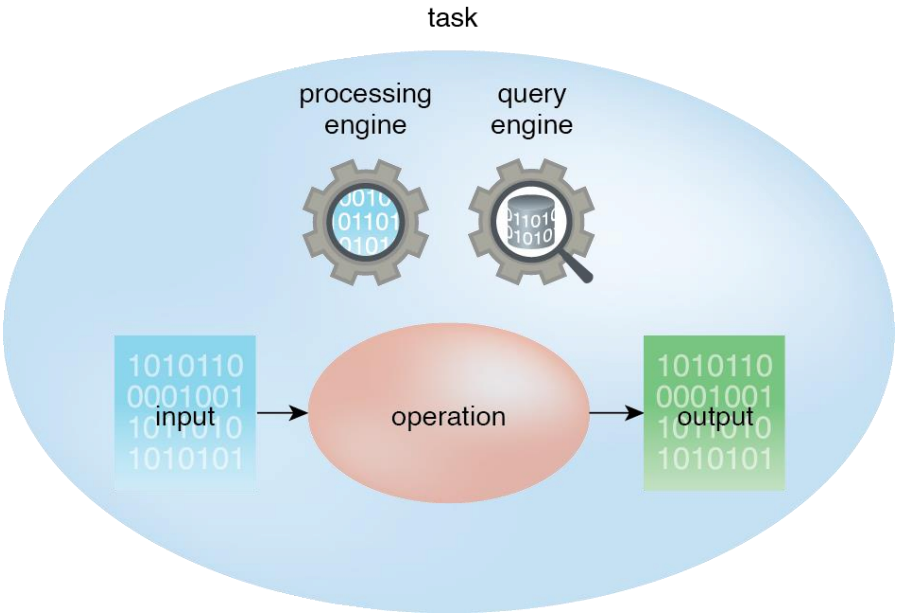


Figura 8.42 – Ejemplo que ilustra los componentes de una tarea en un Big Data Pipeline.

Los mecanismos habilitadores en un Big Data Pipeline incluyen los mecanismos del motor de transferencia de datos, el dispositivo de almacenamiento, el motor de procesamiento, el motor de consultas (Query Engine), el motor de comprensión, el motor de serialización y el motor del flujo de trabajo (Workflow), como se muestra en la Figura 8.43.

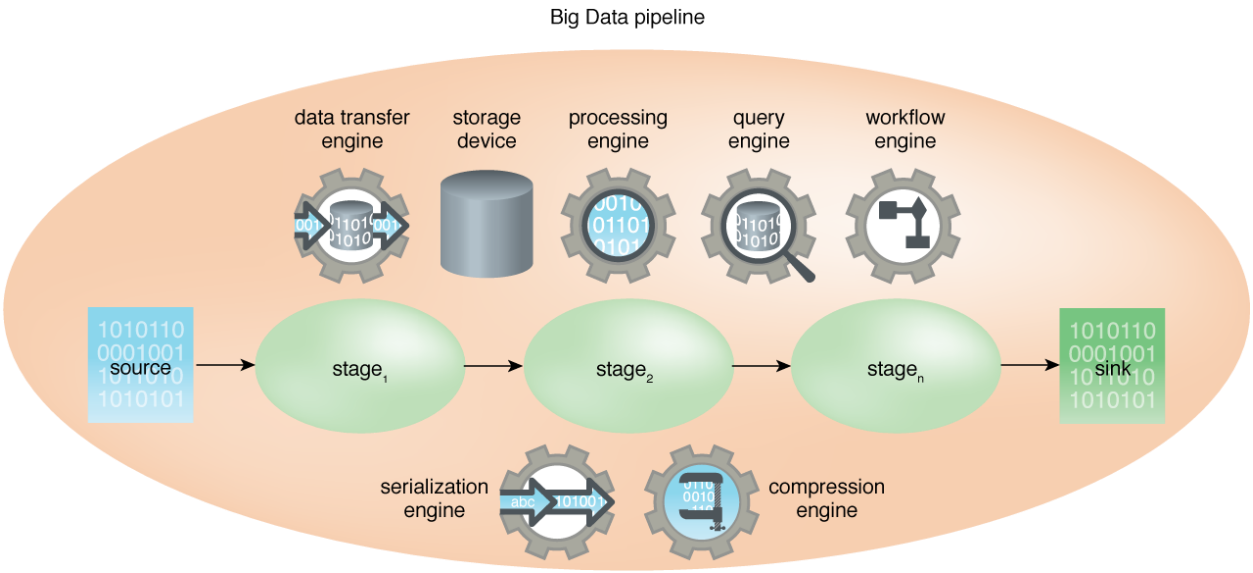


Figura 8.43 – Ejemplo de un Big Data Pipeline.

En un Big Data Pipeline, el motor del flujo de trabajo (Workflow) organiza el diseño del pipeline, activando las tareas posteriores una vez que se complete la tarea anterior. El desarrollo de soluciones de procesamiento de datos en entornos Big Data se articula en torno al concepto de pipelines de datos. Algunos de los retos encontrados durante el desarrollo de Big Data Pipelines incluyen el requisito de contar con un método escalable y confiable para ingerir datos estructurados, semiestructurados y sin estructurar, y combinar datos estructurados con datos sin estructurar de manera totalmente automática.

Un Big Data Pipeline puede ser muy simple —integrado por una sola etapa—, o también muy complejo —integrado por múltiples etapas. Dentro de los entornos Big Data, los pipelines de datos se usan cuando los datos deben ser modificados de alguna forma; por ejemplo, desde la extracción de los datos iniciales hasta el enriquecimiento de los datos posteriores, como en el caso de la geocodificación de direcciones IP y la desnormalización de datos. Generalmente, un Big Data Pipeline constituye una solución parcial o completa para el análisis de datos (Data Analysis) de Big Data.

Una de las principales razones para diseñar un Big Data Pipeline es convertir los datos sin estructurar a su forma estructurada, a fin de que sean útiles para los sistemas descendentes. Para simplificar y acortar el tiempo de desarrollo, se pueden usar muchas herramientas de desarrollo de pipeline que pueden combinar un subconjunto de tareas en un micropipeline, tales como la validación y separación de datos.

El Big Data Pipeline está totalmente automatizado gracias al uso de un motor de flujo de trabajo (Workflow). Sin embargo, antes de que el pipeline pueda ser implementado en el sistema en vivo, tiene que pasar rigurosas pruebas. Todo el pipeline debe ser primero probado de forma modular (prueba de unidad), garantizando que en cada etapa se generen los resultados deseados; y después mediante una prueba de integración, a fin de verificar el funcionamiento del Big Data Pipeline, de principio a fin.

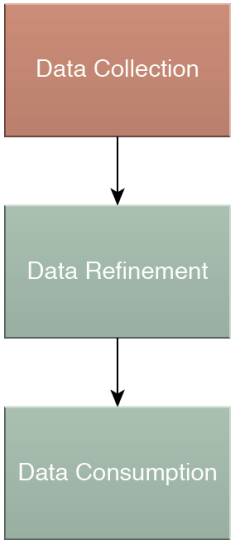
Big Data Pipeline: etapas

Un típico Big Data Pipeline está compuesto por las siguientes etapas:

1. Recolección de datos
2. Refinamiento de datos
3. Consumo de datos

NOTA
Cabe señalar que estas etapas representan un Big Data Pipeline de principio a fin. En la práctica, las etapas necesarias dependen de los requerimientos de procesamiento de datos.

Big Data Pipeline: recolección de datos



La etapa de recolección de datos está compuesta por las tareas de ingestión, filtración, compresión y almacenamiento de datos. Los datos recolectados son almacenados en su forma sin procesar usando un dispositivo de almacenamiento apropiado, que luego se recuperará y procesará de alguna manera antes de que los resultados sean almacenados de forma estructurada.

La recolección implica hacer las conexiones necesarias para adquirir los datos; en otras palabras, conectar fuentes de datos con colectores de datos usando los motores de transferencia de datos. Los datos pueden ser ingeridos por lotes o en modo de tiempo real.

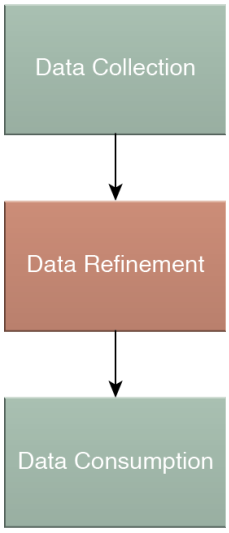
Normalmente, el modo por lotes se usa para datos relacionales y datos basados en archivos que pueden ser automatizados utilizando el motor de transferencia de datos relacional y el motor de archivos, respectivamente. Por lo general, el modo en tiempo real se usa para los datos basados en eventos, tales como datos de sensores o transaccionales, los cuales se pueden automatizar usando el motor de transferencia de datos de eventos. Usualmente, en esta etapa se lleva a cabo un filtrado (filtering) básico para eliminar los datos inválidos usando un motor de consultas (Query Engine) o un motor de procesamiento.

Antes del almacenamiento, los datos sin procesar pueden comprimirse usando un motor de compresión para ahorrar espacio de almacenamiento y para lograr tasas de transferencia de datos más rápidas para el procesamiento posterior. También es importante almacenar los datos sin perder la fidelidad y elegir un motor de serialización apropiado que proporcione un formato de almacenamiento al que puedan acceder varias bibliotecas y herramientas de software.

Los datos recolectados se almacenan usando un dispositivo de almacenamiento apropiado, tal como una base de datos NoSQL o un sistema de archivos distribuido. Dependiendo del (de los) tipo(s) de datos recopilados, se puede requerir más de una clase de dispositivos de almacenamiento. Asimismo, dependiendo de los requerimientos de procesamiento de carga de trabajo, pueden ser necesarios dispositivos de almacenamiento en memoria o en disco, o en algunos casos, una combinación de ambos.

Dependiendo del motor de procesamiento que se esté usando, podría ser necesario combinar múltiples archivos pequeños para optimizar el procesamiento con el fin de disminuir su tiempo de ejecución. Esto se debe al hecho de que un motor de procesamiento como MapReduce trabaja mejor con archivos más grandes pero en menor cantidad. Un gran número de archivos pequeños genera exceso en las operaciones de búsqueda en el disco y aumenta el consumo de memoria, ya que, por lo general, el motor de procesamiento genera procesos dedicados a cada archivo de manera individual.

Big Data Pipeline: refinamiento de datos



La etapa de refinamiento involucra la extracción, validación o limpieza (cleansing) y la unión o división de tareas, incluyendo la creación de modelos de datos y las tareas de exportación de datos. Los campos de datos requeridos se extraen de los datos sin procesar que fueron almacenados previamente.

Es importante entender que es posible no conocer cuáles campos deben ser extraídos, especialmente en el caso del análisis exploratorio. La información acerca de los requerimientos de datos estratégicos y operativos, y las metas empresariales pueden ayudar a identificar los campos requeridos.

Se debe usar un criterio significativo de extracción que logre un balance entre los casos de uso conocidos y los requerimientos exploratorios desconocidos. El objetivo no es almacenar datasets superficiales o demasiado pobres.

Normalmente, la extracción de datos requiere datos estructurados y semiestructurados. Sin embargo, con los datos sin estructurar —tales como imágenes o archivos de audio—, o no se requiere de ninguna clase de extracción, o se requiere una extracción parcial, tal como la extracción de atributos embebidos de imágenes. Asimismo, los datos extraídos se validan; y se eliminan aquellos que contengan valores inválidos.

La característica de variedad de Big Data establece que, de forma frecuente, todos los datos requeridos pueden estar ausentes en un solo dataset, lo cual justifica la fusión de datasets. Por otra parte, es posible que un solo dataset incluya grupos lógicos que pueden ser consumidos de forma separada para respaldar distintas tareas de análisis, lo cual justifica la división de un dataset en varios datasets.

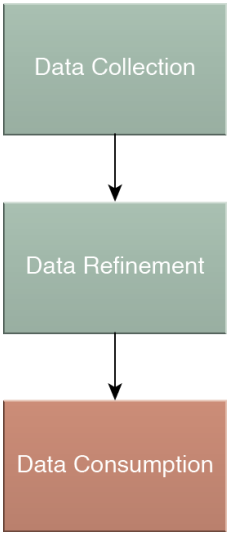
Otras tareas en esta etapa pueden incluir la generación de nuevos campos basados en datos existentes y la transformación del formato de los datos. Tales tareas se pueden llevar a cabo usando un motor de procesamiento, como MapReduce, o ya sea directa o indirectamente por medio de un motor de consultas (Query Engine). En caso de una solución de procesamiento de

datos en tiempo real, la etapa de refinamiento por lo general se lleva a cabo antes de que los datos sean almacenados en el disco. En otras palabras, las operaciones en esta etapa se llevan a cabo sobre los datos en memoria.

La toma de decisiones respecto a cuál tipo de dispositivo de almacenamiento se debe usar (para los datos refinados) y el diseño del esquema más óptimo también se llevan a cabo durante esta etapa, ya que pueden existir múltiples estrategias para almacenar el mismo tipo de datasets. Esto es relevante especialmente para los dispositivos de almacenamiento de grafos y basados en columnas, ya que los mismos datos se pueden almacenar de diferentes maneras.

Por ejemplo, los datos de sensor basados en series temporales pueden ser almacenados en una sola fila con una columna por lectura (filas amplias) o como múltiples filas divididas por tiempo —por ejemplo, por día— con menos columnas (filas estrechas). De forma similar, con los datos de grafos, se debe considerar el tamaño de los datos (cantidad de vértices y bordes), los patrones de acceso (transaccional y por lotes) y los casos de uso (algoritmos de búsqueda y repetitivos).

Big Data Pipeline: consumo de datos



La etapa de consumo consiste en las tareas que usan los datos refinados en la etapa anterior. En esta etapa, los datos refinados pueden ser consumidos directa o indirectamente.

El consumo directo implica poner en marcha modelos desarrollados por científicos de datos por medio de la ejecución de algoritmos subyacentes en datos activos (refinados). De forma similar, en casos de uso relativamente simples, se generan varias estadísticas tales como *totales* y *promedios* a partir de datos refinados, y se introducen en sistemas descendentes de reporte de inteligencia de negocios (BI), tableros de control (dashboards) y otros colectores de información, como los portales.

El consumo indirecto implica el suministro de datos refinados a múltiples funciones del negocio y sistemas descendentes. Específicamente, consiste en garantizar que los datos refinados están en el formato o estructura correcta para que el sistema final los pueda utilizar.

Por ejemplo, una herramienta de inteligencia de negocios (BI) requiere datos en formato tabular, mientras que los datos refinados existen en un formato de archivo delimitado por tabulaciones. Por lo tanto, los datos refinados deben ser procesados aún más para que cumplan con el formato necesario. Igualmente, es posible que los datos refinados deban exportarse a una bodega de datos digital (Data Warehouse) para realizar un análisis de datos (Data Analysis) más tradicional.

Independientemente de la forma en que sean consumidos los datos, por lo general se requiere un mayor procesamiento o transformación de los mismos, dependiendo del caso de uso individual; por ejemplo, para la creación de subconjuntos o agregados de datos para desarrollar modelos (estadísticos o de aprendizaje automático (Machine Learning)). Debido a la naturaleza diversa de esta etapa, se pueden usar mecanismos de un servidor de Big Data, incluyendo el motor de transferencia de datos, el motor de consultas (Query Engine), el motor de procesamiento y el motor analítico (Analytics Engine).

Extraer-cargar-transformar (ELT)

Al igual que el ETL (presentado en el Módulo 1), el proceso extraer-cargar-transformar (ELT) es un proceso de carga de datos desde un sistema origen a un sistema destino. Sin embargo, a diferencia del ETL, en el cual la transformación se lleva a cabo fuera del sistema destino y antes de ingresar datos en el sistema destino, en el proceso ELT los datos son cargados tal cual al sistema destino y se transforman dentro del sistema destino.

El uso de ELT es más frecuente en entornos Big Data, en donde los datos sin procesar se almacenan tal cual, en parte debido a que no se conocen los escenarios de uso de los datos. El ELT elimina la necesidad de un sistema de pruebas (base de datos) ya que los datos se pueden transformar internamente en la plataforma Big Data, tal como Hadoop. También ayuda a evitar la duplicación de datos y la necesidad de obtener hardware adicional para la etapa de transformación.

En el ETL, la etapa de transformación se une con la etapa de carga, y los datos se transforman de acuerdo con los escenarios de consumo conocidos; mientras que en el ELT, las etapas de transformación y carga están separadas y los datos se transforman de acuerdo con escenarios nuevos, previamente desconocidos.

En entornos Big Data, el ELT generalmente tiene un mejor desempeño que el ETL. La etapa de carga es más rápida debido a la importación de datos altamente escalables y la etapa de transformación es más rápida debido a la colocación de los datos. El ELT proporciona acceso a todos los datos en su forma sin procesar, la cual es ideal para el análisis exploratorio y la construcción de modelos.

Las primeras dos etapas del Big Data Pipeline presentadas anteriormente esquematizan de forma aproximada el concepto ELT, en el que los datos sin procesar se cargan en el almacenamiento distribuido sin ninguna transformación y el refinamiento se realiza dentro de la plataforma de Big Data, sin necesidad de exportar datos a otro sistema, como se muestra en la Figura 8.44. Normalmente, los mecanismos requeridos incluyen el motor de transferencia de datos, el dispositivo de almacenamiento, el motor de consultas (Query Engine) y el motor de flujo de trabajo (Workflow).

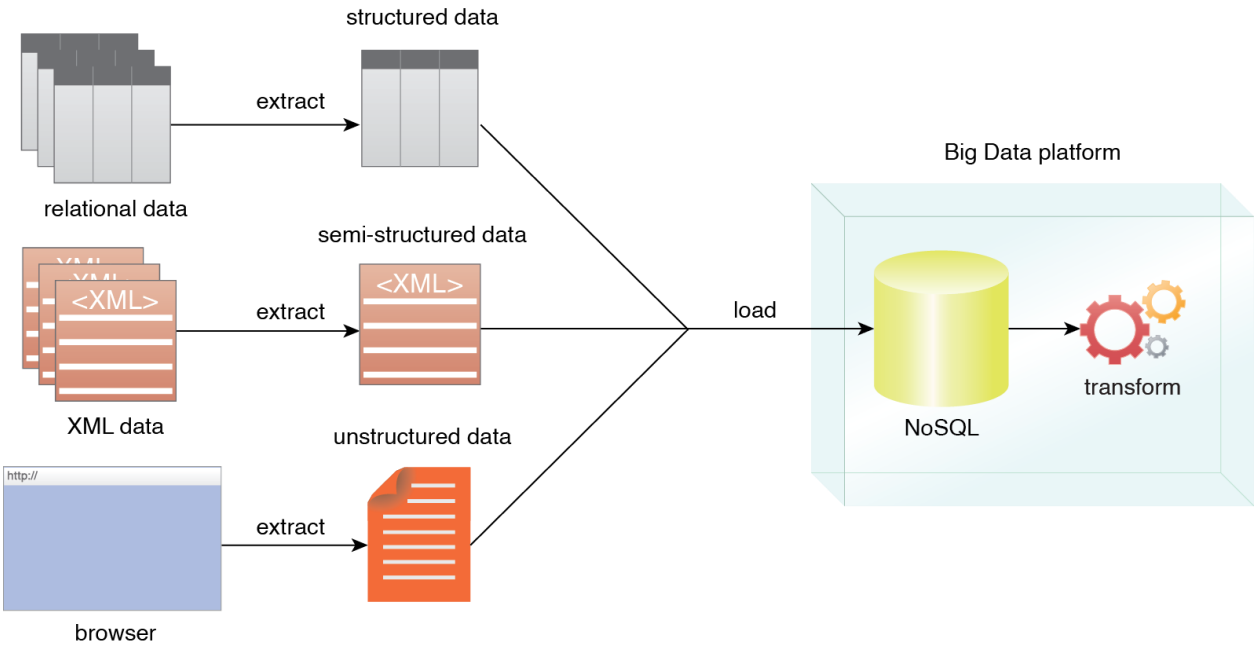


Figura 8.44 – Ejemplo de refinamiento dentro de la plataforma de Big Data.

Ejercicio 8.5: complete los espacios en blanco

1. Un pipeline de datos es un flujo de trabajo (Workflow) _____ compuesto por múltiples tareas en las que cada una comprende _____, _____ y _____.

2. Un pipeline de datos se usa para mover datos entre la _____ y el _____ de manera automática, mientras que se llevan a cabo diferentes _____.

3. Una de las principales razones para diseñar un Big Data Pipeline es convertir los datos _____ en su forma _____, a fin de que sean útiles para los sistemas descendentes.

4. Un típico Big Data Pipeline consiste en las etapas de _____, _____ y _____.

5. La etapa de _____ involucra las tareas de extracción, validación o limpieza (cleansing) y joining o división.

6. Como parte de la etapa de _____, por lo general se requiere mayor procesamiento o transformación de datos.

7. La etapa de _____ está compuesta por las tareas de ingestión, filtración, compresión y almacenamiento de datos.

8. Al igual que el ETL, el proceso _____ es un proceso mediante el cual los datos son cargados desde un sistema origen a un sistema destino.

9. El ELT elimina la necesidad de un _____, ya que los datos se pueden transformar internamente en la plataforma Big Data.

Notas

Notas / Bocetos

Diseño de solución de Big Data

Diseño de solución de Big Data: definición

Una solución de Big Data es una aplicación que opera basada en datos y que procesa datos estructurados, semiestructurados y sin estructurar adquiridos en diferentes fuentes, que incluyen fuentes de datos tradicionales (tales como aplicaciones de OLTP y bodegas de datos digitales (Data Warehouses)) así como fuentes de datos contemporáneas (tales como social media, internet de las cosas (IoT por sus siglas en inglés) y logs de máquinas).

Diseño de solución de Big Data: casos de uso

El principal caso de uso de las soluciones de Big Data puede ser considerado como una solución de principio a fin que ingiere datasets de Big Data, lleva a cabo algunos análisis y después comunica los resultados por medio de un tablero de control (Dashboard) o reenvía los resultados a otra aplicación.

El caso de uso secundario puede ser considerado como una solución que funciona en capacidad de respaldo y cuya función es facilitar otras aplicaciones o sistemas. Tal solución ingiere datasets de Big Data, los procesa para crear datos resumidos o los convierte a un formato estructurado, y después exporta los datos estructurados y resumidos a una aplicación o sistema más tradicional para un mayor procesamiento.

Solución de Big Data: características

A diferencia de las aplicaciones empresariales tradicionales —tales como los sistemas de relación con los clientes (CRM) y los Sistemas de Planificación de recursos empresariales (ERP)— en los que las aplicaciones son productoras de datos, las soluciones de Big Data en general son consumidoras de datos. Sin embargo, crean datos adicionales que pueden contribuir a las aplicaciones tradicionales y los productos de analítica.

A pesar de que las aplicaciones de reporte tradicionales —tales como las herramientas de Inteligencia de negocios (BI)— son consumidoras de datos, estas funcionan con datos estructurados y generalmente no crean más datos para que sean consumidos por otras aplicaciones. En la mayoría de casos, se combinan múltiples datasets heterogéneos para crear datasets significativos unificados que son más valiosos que los datasets individuales por separado.

A diferencia de las aplicaciones tradicionales, los datos de entrada y el resultado pueden evolucionar con el tiempo a medida que se dispone de más datasets. De forma similar, el alcance de una solución de Big Data puede evolucionar con el tiempo a medida que se descubren más casos de uso.

Aunque la naturaleza exacta de la solución de Big Data depende de las necesidades empresariales, su tarea principal es que los datos estén disponibles para varias aplicaciones descendentes, tales como los tableros de control (Dashboards) y los sistemas de OLAP/OLTP,

y permitir el análisis de datos (Data Analysis) por medio de la ejecución de varios modelos, tales como los modelos estadísticos y los algoritmos de aprendizaje automático (Machine Learning).

En algunas situaciones, la fuente puede ser la misma que el sink, como es el caso de la importación de datos de ventas diarias en un sistema de comercio electrónico, el cual genera las cifras de ventas por producto y posteriormente exporta los resultados de vuelta al sistema de comercio electrónico, con el fin de elaborar informes.

Las soluciones de Big Data enriquecen las aplicaciones tradicionales por medio de un loop de retroalimentación. Por ejemplo, las predicciones generadas por una solución de Big Data alimentan a una aplicación web y son refinadas por medio de la retroalimentación recibida a partir de las interacciones con los clientes.

Solución de Big Data: consideraciones de diseño

Una solución de Big Data debe cumplir con la característica de variedad de Big Data, brindando compatibilidad con el almacenamiento de datos que tengan varias estructuras y varios formatos, y que fueron capturados de diferentes fuentes de datos, además de contar con capacidad para procesar datos cuyo esquema pueda ser desconocido.

Una solución de Big Data debe escalar masivamente estas tres áreas:

- la recolección de datos, especialmente en el caso de datos generados por máquinas o IoT
- el almacenamiento de datos
- el procesamiento de datos

Con el fin de permanecer escalables, las soluciones de Big Data son diseñadas como soluciones distribuidas que se extienden a través de múltiples máquinas en un cluster. Por consiguiente, las fallas de informáticas y de comunicación deben ser abordadas mediante el diseño de la solución, a fin de garantizar una alta disponibilidad.

Cabe señalar que, en el caso de los dispositivos de Big Data —plataformas patentadas de Big Data que incluyen máquinas e interconexión a nivel de toda la empresa—, la alta disponibilidad deja de ser un problema, y en cambio, el costo y la escalabilidad son las preocupaciones más inmediatas. Además, se deben implementar la replicación de datos y el sharding para proporcionar tolerancia a errores, alta disponibilidad y escalabilidad.

Dependiendo de los requerimientos de procesamiento de datos, tal vez sea necesario que la solución de Big Data sea compatible con el procesamiento por lotes (Batch Processing) y el procesamiento en tiempo real. Por ejemplo; una solución de Big Data analiza los comentarios de los clientes en tiempo real para brindar mejor atención al cliente, a la vez que estos mismos comentarios son combinados con el historial de compras para realizar la minería de patrones (un proceso por lotes (Batch Processing)) y así determinar las causas recurrentes detrás de la devolución de un producto.

Se deben seguir consideraciones especiales en relación con la prueba y depuración de soluciones de Big Data. Esto se debe a que las soluciones de Big Data por lo general consisten en pipelines de datos complejos que comprenden varias etapas. Por consiguiente, antes de

implementarla en un cluster, una solución de Big Data debe ser ejecutada en un modo seudodistribuido (un modo de ejecución de un solo nodo que imite un cluster). Así, cada etapa se puede probar o depurar de forma relativamente fácil, ya que todos los archivos de registro (log files) están ubicados en la misma máquina.

Tanto para el ajuste del rendimiento como la depuración, el modo de registro en las distintas etapas de una solución de Big Data compleja debe ser ejecutado y configurado a un nivel apropiado, tal como INFO o DEBUG, si se usa el log4j. Esto ayuda a conseguir tanta información como sea posible acerca del comportamiento del tiempo de ejecución de la solución de Big Data, ayudando de esta manera a que exista una depuración más rápida y efectiva.

Usar subconjuntos de grandes datasets como datos de entrada facilita aún más la depuración. Esto no solo ayuda a encontrar problemas lógicos sino también relacionados con los datos, tales como datos formateados incorrectamente o registros inservibles. El modo seudodistribuido también es ideal para desarrollar prototipos, especialmente si la empresa se está embarcando en un proyecto de Big Data por primera vez.

Los datos fuente deben mantenerse en la forma de común denominador más baja para atender múltiples casos de uso. Esto significa que los datos fuente se deben limpiar y validar. Sin embargo, los datos fuente deben guardarse antes de llevar a cabo cualquier operación posterior, tal como la transformación del modelo o formato de datos, o la unión de datasets.

Dependiendo de la estructura de datos, se debe usar el motor de procesamiento correspondiente. Por ejemplo, para datos de grafos, se debe usar un motor de procesamiento compatible con el envío de mensajes entre nodos, mientras que MapReduce se puede usar en situaciones en las que los datos no están relacionados unos con otros y cada registro existe como un agregado individual que puede procesarse independientemente.

Solución de Big Data: proceso de diseño

1. Establecer y evaluar los datos de entrada y salida

Identifique las fuentes de datos requeridas. Determine el acceso a las fuentes de datos por medio del (de los) motor(es) de transferencia de datos. Elija el tipo correcto de dispositivo(s) de almacenamiento. Identifique cómo se debe conservar el resultado evaluando la(s) aplicación(es) descendente(s). Establezca los criterios de acceso (autenticación y autorización) tanto para los datos de entrada guardados como para los datos de salida.

2. Determinar los requerimientos de manejo de datos

Evalúe la calidad de los datos de entrada. Desarrolle pipelines de datos para el análisis, limpieza (cleansing), filtrado (filtering), joining y conversión del formato o modelo de datos.

3. Seleccionar el formato de representación de los datos

Almacene los datos ajustados en el paso anterior, eligiendo una estructura de datos que pueda ser procesada de manera óptima por el (los) motor(es) de procesamiento de datos. Evalúe los patrones de consulta en caso de que el objetivo sea proporcionar acceso a datos limpios y sin procesar por medio de consultas similares a SQL; por ejemplo, durante el análisis exploratorio de datos.

4. Evaluar la idoneidad del motor de procesamiento

Evalúe el modo de procesamiento de datos en función del tiempo necesario para obtener resultados (modo en tiempo real, tiempo prácticamente real, o modo offline/por lotes). Considere tanto el modelo del dispositivo de almacenamiento como las características del algoritmo de procesamiento, con el fin de seleccionar un motor de procesamiento apropiado. Por ejemplo, los datos almacenados en una base de datos de grafos y la necesidad de un algoritmo repetitivo llevarán a que se seleccione un motor de procesamiento compatible con procesamiento basado en BSP.

5. Desarrollar rutinas de procesamiento de datos

Dependiendo de la naturaleza del procesamiento, escriba el código usando un lenguaje de desarrollo de software, tal como Java, Python, etc., o escriba scripts usando un lenguaje de scripts, tal como Pig Latin. Una secuencia típica de código involucra leer los datos de entrada en un dispositivo de almacenamiento, llevar a cabo algunas funciones de manejo de datos en los datos y después escribir los datos procesados en un dispositivo de almacenamiento. Dependiendo de las capacidades de la plataforma de Big Data, este puede ser un ejercicio sencillo; y en otros casos, en vez de escribir el código, se puede requerir la incorporación de un algoritmo por medio de sus API.

6. Desarrollar visualizaciones

Cree soluciones gráficas para mostrar los datos procesados en un formato sencillo y para respaldar la realización de varios análisis. Por lo general, las API incorporan varias bibliotecas gráficas para construir front end similares a un tablero de control (Dashboard).

7. Automatizar la ejecución de la solución

Por lo general, una solución de Big Data es ejecutada continuamente, como en el modo en tiempo real; o a intervalos regulares, como en el modo por lotes, de forma automática. Luego de desarrollar la solución de Big Data, se debe realizar la configuración necesaria o escribir el código que garantice las operaciones de ingreso, procesamiento y egreso de los datos automatizados. Esto puede involucrar una actualización automática de las visualizaciones de datos; por ejemplo, los tableros de control (Dashboards).

Como parte de la automatización de ejecución de la solución, también se puede requerir la creación de conexiones para establecer cualquier loop de retroalimentación necesario, como es el caso de algunos algoritmos de aprendizaje automático (Machine Learning). Aunque se describe como el último paso, este también puede realizarse mientras se establecen las conexiones necesarias de entrada y salida y mientras se desarrollan las rutinas de procesamiento necesarias.

Solución de Big Data: simple

Las aplicaciones tradicionales se desarrollan teniendo en mente un objetivo en particular. Sin embargo, debido a la naturaleza exploratoria del análisis de Big Data, las metas asociadas con la solución de Big Data deben cambiar con el tiempo, lo que dificulta el diseño de las soluciones de Big Data.

Una solución de Big Data puede comenzar su vida como una simple aplicación con muchos cálculos, pero después de un tiempo puede evolucionar y convertirse en un producto de datos. En general, una solución simple de Big Data es una aplicación de procesamiento de datos por lotes (Batch Processing) con un número relativamente pequeño de datos de entrada que genera ciertas estadísticas o que realiza tareas simples de transformación de datos.

Normalmente, los mecanismos de Big Data involucrados en el diseño de una solución simple de Big Data incluyen:

- El motor de transferencia de datos (archivo/relacional)
- El dispositivo de almacenamiento en disco
- El motor de procesamiento (por lote)
- El motor de consultas (Query Engine)
- El motor de flujo de trabajo (Workflow)

La Figura 8.45 muestra cómo estos mecanismos generalmente están conectados entre sí. Tenga en cuenta que, aunque es parte de la solución de Big Data, el gestor de recursos no está incluido, con el fin de mostrar un diseño más simple.

La Figura 8.45 ilustra los siguientes pasos:

1. (a, b). Un motor de transferencia de datos de tipo archivos y uno de tipo relacional se usan para obtener datos, generalmente del interior de la empresa, tales como datos transaccionales y archivos de registro (log file) de servidores web.
2. (a, b). Los datos se almacenan usando dispositivos de almacenamiento en disco, incluyendo un sistema de archivos distribuido y una base de datos NoSQL.
3. (b, a). Como soporte de los requerimientos computacionales necesarios, un motor de procesamiento por lotes (Batch Processing) se usa directa o indirectamente por medio de un motor de consultas (Query Engine).
4. Los resultados procesados se exportan después a una aplicación descendente por medio del motor de transferencia de datos, tal como un tablero de control (Dashboard).

Un motor de flujo de trabajo (Workflow) se usa para que las actividades de ingreso, procesamiento y egreso de datos se lleven a cabo repetitivamente según se requiera, sin ninguna intervención humana.

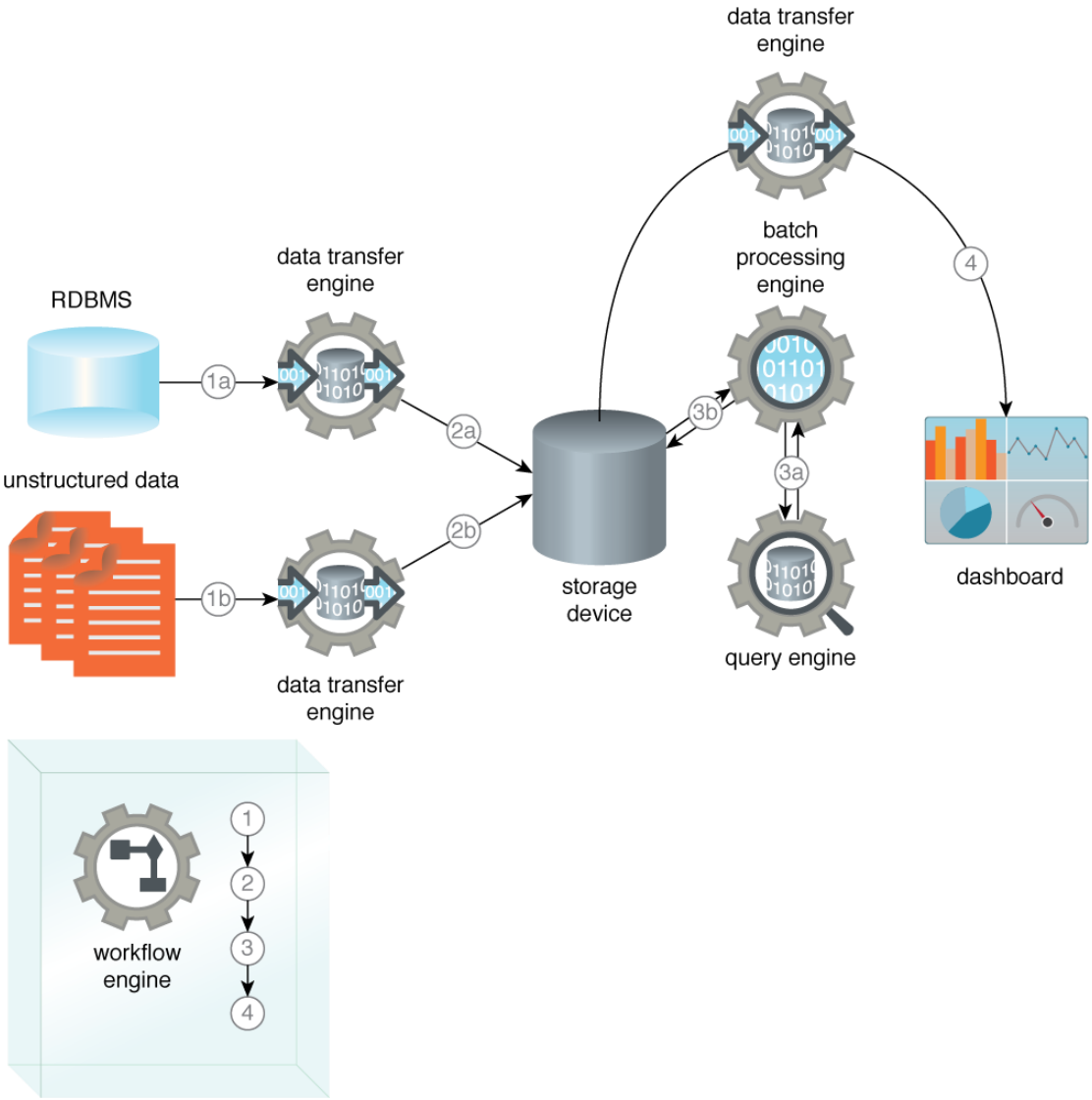


Figura 8.45 – Ejemplo de una solución simple de Big Data.

Solución de Big Data: compleja

En contraste con una solución de Big Data simple, una solución de Big Data compleja puede tener componentes tanto de lotes como de tiempo real; por ejemplo, dispositivos de almacenamiento en disco o en memoria y motores de procesamiento por lotes (Batch Processing) o en tiempo real. En general, una solución compleja de Big Data ingresa los datos desde múltiples fuentes de datos con el fin de realizar el procesamiento complejo de datos.

El procesamiento en sí puede involucrar la creación de pipelines de datos multietapa que ejecuten el aprendizaje automático (Machine Learning) u otros algoritmos complejos. El producto procesado se puede exportar a un tablero de control (Dashboard) o a alguna otra aplicación posterior que incorpore los resultados del procesamiento, tal como una aplicación de detección y mitigación de riesgos.

Normalmente, los mecanismos de Big Data involucrados en el diseño de una solución compleja de Big Data incluyen:

- El motor de transferencia de datos (evento/archivo/relacional)
- El dispositivo de almacenamiento (en disco o en memoria)
- El motor de procesamiento (por lotes o en tiempo real)
- El motor analítico (Analytics Engine)
- El motor de serialización
- El motor de compresión
- El motor de flujo de trabajo (Workflow)

La Figura 8.46 muestra cómo estos mecanismos generalmente están conectados entre sí. Se ilustran los siguientes pasos:

1. (a). Generalmente se usa un motor de transferencia de datos de eventos para obtener datos provenientes de la empresa, tales como datos de sensores, o de fuera de la empresa, tales como datos de social media.

   (b, c). Los motores de transferencia de datos de tipo archivos y de tipo relacional se usan para obtener datos, generalmente provenientes de la empresa, tales como datos transaccionales y archivos de registro (log file) de servidores web; o de fuera de la empresa, tales como datos de censos y datos relativos al genoma.

2. (a, b, c). Los datos importados se serializan en un formato que sea apropiado para el posterior procesamiento de datos.

3. (a). Los datos de eventos se conservan en un dispositivo de almacenamiento en memoria.

   (b, c). Los datos sin estructurar se comprimen primero y después se conservan en un dispositivo de almacenamiento en disco. Los datos relacionales también pueden comprimirse primero, antes de ser almacenados.

4. (a, b). Los datos en memoria se procesan directamente usando un motor de procesamiento en tiempo real, o indirectamente por medio de un motor analítico (Analytics Engine).

   (c). El motor de procesamiento en tiempo real también puede combinar datos de un dispositivo de almacenamiento en disco. (d). Los datos sin procesar en memoria se comprimen y luego se conservan en el dispositivo de almacenamiento en disco; por ejemplo, un sistema de archivos distribuido o una base de datos NoSQL, que puede unirse con otros datasets para un análisis más detallado.

5. (a, b). Por otro lado, los datos en disco se procesan directamente usando un motor de procesamiento por lotes (Batch Processing) o indirectamente por medio de un motor analítico (Analytics Engine) para ejecutar algoritmos complejos; por ejemplo, un algoritmo de agrupamiento (Clustering) para la segmentación de clientes de una campaña de mercadeo dirigido.

6. (a, b). Por medio del motor de transferencia de datos correspondiente, los resultados del procesamiento de datos son exportados desde el almacenamiento en memoria y en disco a una aplicación posterior, tal como un tablero de control (Dashboard) o alguna otra aplicación empresarial.

Un motor de flujo de trabajo (Workflow) automatiza las actividades de ingreso, procesamiento y egreso de datos, para que estas se lleven a cabo repetitivamente según se requiera, sin ninguna intervención humana.

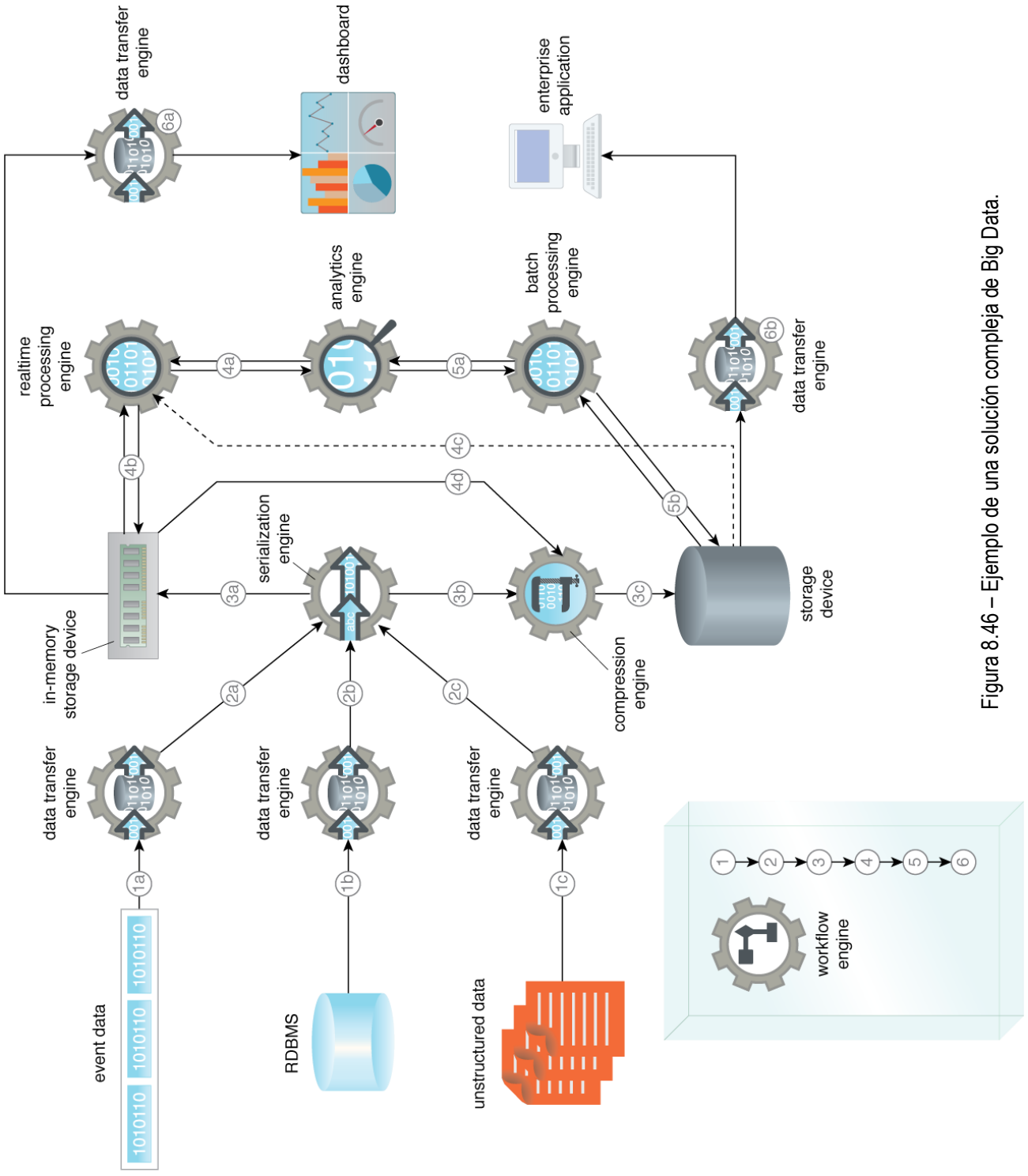


Figura 8.46 – Ejemplo de una solución compleja de Big Data.

Lectura

En la sección Desarrollo de un sistema de recomendación, en las páginas 244 a 249 del libro *Imperativos de Big Data* incluido en este módulo, encontrará un análisis más detallado sobre este tema.

Notas

Notas / Bocetos

Respuestas a los ejercicios

Respuestas al ejercicio 8.1

1. En las plataformas Big Data, es necesaria la **serialización** para permitir el intercambio de mensajes entre máquinas y guardar datos.

2. Cuando se elige un mecanismo de motor de compresión, es importante entender la relación entre **la rapidez de compresión**, **el nivel de compacidad** y los **recursos de procesamiento** necesarios.

3. Preferiblemente, un mecanismo de motor de serialización debe serializar y deserializar datos a **alta** velocidad con una reducción de tamaño **máxima**, ser **susceptible** a cambios futuros y trabajar con una variedad de productores y consumidores de datos.

4. Por lo general, un motor de compresión **lento** proporcionará datos más compactos que requieren más recursos de procesamiento, mientras que un mecanismo de motor de compresión **rápido** proporcionará datos menos compactos y requerirá menos recursos de procesamiento.

5. Cuando se usa un sistema de archivos distribuido, los datos deben serializarse, ya que trabajar con bytes **sin procesar** no solamente es difícil, sino que también causa problemas de **interoperabilidad**.

Respuestas al ejercicio 8.2

1. Dispositivo de base de datos en memoria
2. Dispositivo de almacenamiento en disco
3. Dispositivo de grilla de datos en memoria
4. enfoque de write-behind
5. enfoque de write-through

Respuestas al ejercicio 8.3

1. Verdadera
2. Falsa
3. Falsa
4. Verdadera
5. Verdadera
6. Falsa
7. Falsa
8. Verdadera
9. Verdadera
10. Verdadera

Respuestas al ejercicio 8.4

1. preprocesamiento
2. cálculo local
3. comunicación
4. sincronización de barrera

Respuestas al ejercicio 8.5

1. Un pipeline de datos es un flujo de trabajo (Workflow) **orientado por datos**, compuesto por múltiples tareas en las que cada una comprende **datos de entrada**, **operación** y **datos de salida**.

2. Un pipeline de datos se usa para mover datos entre la **fuente** y el **colector** de manera automática, mientras se llevan a cabo diferentes **operaciones**.

3. Una de las principales razones para diseñar un Big Data Pipeline es convertir los datos **sin estructurar** en su forma **estructurada**, a fin de que sean útiles para los sistemas descendentes.

4. Un típico Big Data Pipeline consiste en las etapas de **recolección de datos**, **refinamiento de datos** y **consumo de datos**.

5. La etapa de **refinamiento de datos** involucra las tareas de extracción, validación o limpieza (cleansing) y joining o división.

6. Como parte de la etapa de **consumo de datos**, por lo general se requiere mayor procesamiento o transformación de datos.

7. La etapa de **recolección de datos** está compuesta por las tareas de ingestión, filtración, compresión y almacenamiento de datos.

8. Al igual que el ETL, el proceso **extraer-cargar-transformar (ELT)** es un proceso mediante el cual los datos son cargados desde un sistema origen a un sistema destino.

9. El ELT elimina la necesidad de un **sistema de pruebas (base de datos)**, ya que los datos se pueden transformar internamente en la plataforma Big Data.

Examen B90.08

El curso que acaba de finalizar corresponde al Examen B90.08, que es un examen oficial del Programa Profesional Certificado de Ciencias de Big Data (BDSCP).

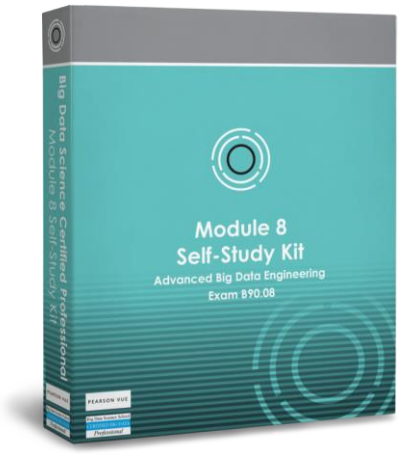
Este examen se puede tomar en los Centros de Examinación de Pearson VUE en todo el mundo, o a través de Pearson VUE Proctoring Online, que le permite tomar exámenes desde su casa o estación de trabajo y contar con supervisión en vivo. Si desea obtener más información, visite las siguientes páginas web:

www.bigdatascienceschool.com/exams/

www.pearsonvue.com/arcitura/

www.pearsonvue.com/arcitura/op/ (Online Proctoring)

Kit de autoaprendizaje del Módulo 8



Para este Módulo se encuentra disponible un kit oficial de autoaprendizaje de BDSCP, el cual le ofrece materiales y recursos de estudio adicionales, incluyendo una guía separada de autoaprendizaje, CD de audiotutoría y tarjetas de memorización.

Tenga en cuenta que las versiones de este kit de autoaprendizaje están disponibles con y sin un cupón para el Examen B90.08 de Pearson VUE.

Si desea obtener más información, visite la siguiente página web:
www.bigdataselfstudy.com

Información y recursos de contacto

Comunidad de AITCP

Únase a la creciente comunidad de Profesionales Certificados en TI de Arcitura (Arcitura IT Certified Professional, AITCP), conectándose mediante las plataformas oficiales de social media: LinkedIn, Twitter, Facebook y YouTube.

Los links de social media y de la comunidad están disponibles en:

- www.arcitura.com/community
- www.servicetechbooks.com/community

Información general del programa

Si desea conocer información general acerca del programa de BDSCP y los requisitos de certificación, visite la siguiente página web:

www.bigdatascienceschool.com y www.bigdatascienceschool.com/matrix/

Información general acerca de los Módulos del curso y los kits de autoaprendizaje

Si desea conocer información general acerca de los Módulos del curso BDSCP y los kits de autoaprendizaje, visite las siguientes páginas web:

www.bigdatascienceschool.com y www.bigdataselfstudy.com

Inquietudes acerca del examen de Pearson VUE

Si desea conocer información relacionada con la presentación de los exámenes de BDSCP en los centros de examinación de Pearson VUE o mediante la supervisión online de Pearson VUE, visite las siguientes páginas web:

www.pearsonvue.com/arcitura/
www.pearsonvue.com/arcitura/op/ (Online Proctoring)

Programación de talleres dirigidos al público y guiados por instructores

Si desea conocer la más reciente programación de los talleres de BDSCP guiados por instructores que están abiertos al público, visite la siguiente página web:

www.bigdatascienceschool.com/workshops

Talleres privados guiados por instructores

Los entrenadores certificados pueden realizar los talleres directamente en sus instalaciones, con la opción de supervisión de exámenes en el sitio. Si desea saber más acerca de las opciones y tarifas, envíe un correo electrónico a la siguiente dirección:

info@arcitura.com

o llame a la línea

1-800-579-6582

Convertirse en un entrenador certificado

Si usted está interesado en alcanzar el rango de Entrenador Certificado para este o cualquier otro curso o programa de Arcitura, puede obtener más información visitando la siguiente página web:

www.arcitura.com/trainerdevelopment/

Inquietudes generales sobre BDSCP

En caso de que tenga otras preguntas relacionadas con este Curso, o cualquier otro Módulo, Examen o Certificación que haga parte del programa BDSCP, envíe un correo electrónico a la siguiente dirección:

info@arcitura.com

o llame a la línea

1-800-579-6582

Notificaciones automáticas

Si desea que se le notifique automáticamente sobre cambios o actualizaciones al programa BDSCP y a los sitios de recursos relacionados, envíe un mensaje de correo electrónico en blanco a la siguiente dirección:

notify@arcitura.com

Retroalimentación y comentarios

Ayúdenos a mejorar este curso. Envíe su retroalimentación o comentarios a la dirección de correo electrónico:

info@arcitura.com