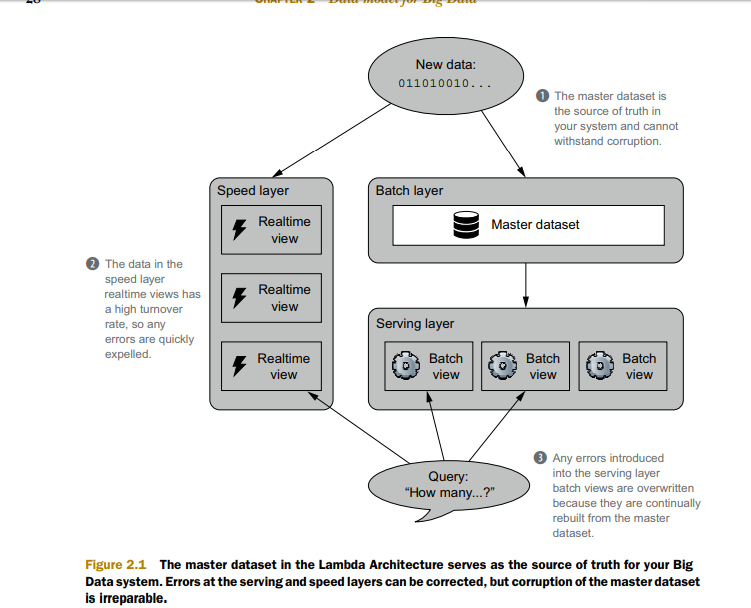
**2. Data model for Big Data**

El principal conjunto de datos es la única parte de la arquitectura Lambda que absolutamente debe ser protegida de la corrupción. Máquinas de sobrecargadas, falta discos y cortes de energía podrían provocar errores, y errores humanos con los sistemas de datos dinámicos es un riesgo intrínseco y inevitable eventualidad. Cuidadosamente debe el Ingeniero principal ver el conjunto de datos para prevenir la corrupción en todos los casos, como tolerancia a errores es esencial para la salud de un sistema de datos de larga duración.



Propiedades de la Data

Crudeza, inmutabilidad y perpetuidad (o la "eterna veracidad de datos").

Crudeza: Un sistema de datos responde a las preguntas sobre la información que has adquirido en el pasado. Al diseñar su sistema de información grande, quiere ser capaz de responder a tantas preguntas como sea posible.

Data es inmutable: Datos inmutables pueden parecer un concepto extraño, si estás versado en bases de datos relacionales. Después de todo, en el mundo de base de datos relacional — y la mayoría otras bases de datos así: update es una de las operaciones fundamentales. Pero para inmutabilidad no actualizar o eliminar datos, añades sólo mas mediante el uso de un esquema inmutable para sistemas grandes de datos, se obtendrá dos ventajas vitales: Tolerancia a fallos humanos y simplicidad.

Data es eternamente veraz: La consecuencia clave de inmutabilidad es que cada dato es verdadero a perpetuidad. Es decir, un pedazo de datos, una vez que es verdad, debe ser siempre verdadero. Inmutabilidad no tendría sentido sin esta propiedad, y vio cómo marcado cada pedazo de datos con una marca de tiempo es una forma práctica para hacer datos eternamente verdadero.

El modelo basado en hechos para representar datos

Datos están el conjunto de información que no pueden derivar de cualquier otra cosa, pero hay muchas maneras que se podría elegir representar dentro del conjunto de datos principal. Además de tradicionales tablas relacionales, XML estructurada y semiestructurados documentos JSON son otras posibilidades para almacenar datos. Sin embargo, se recomienda el modelo de hechos para este propósito. En el modelo basado en los hechos, reconstruyen los datos en unidades fundamentales llamadas (como era de esperar) hechos.

Como modelar un conjunto de datos complejo creando las bases para un ecosistema de Big Data. Las decisiones tomadas sobre el conjunto de datos maestro determinan el tipo de análisis que se puede realizar en nuestros datos y como se va a consumir esos datos. La estructura del maestro de datos es el conjunto de datos que admite la evolución de los tipos de datos almacenados, debido a que los tipos de datos pueden cambiar considerablemente con los años.

El modelo basado en hechos proporciona una representación simple pero expresiva de los datos naturalmente manteniendo una historia completa de cada entidad en el tiempo. Su naturaleza facilita su implementación en un sistema distribuido y puede evolucionar fácilmente a medida que sus datos y las necesidades cambian. No solo se está implementando un sistema relacional de una manera más escalable: también se está agregando capacidades completamente nuevas al sistema.

**3. Data model for Big Data: Illustration**

En este capítulo se implementó el modelo de datos de SuperWebAnalytics.com que utiliza Apache Thrift, un marco de serialización, que incluso en una tarea tan sencilla como escribir un esquema, hay fricción Entre la teoría idealizada y lo que se puede lograr en la práctica.

Por qué un marco de serialización?

Muchos desarrolladores siguen el camino de escribir sus datos en bruto en un formato sin esquemas como JSON. Esto es atractivo debido a lo fácil que es comenzar, pero este enfoque rápidamente conduce a los problemas. Ya sea debido a errores o malentendidos entre diferentes desarrolladores, la corrupción de datos inevitablemente se produce. Es nuestra experiencia que la corrupción de datos Los errores son algunos de los que requieren más tiempo para depurar.

Los problemas de corrupción de datos son difíciles de depurar porque tiene muy poco contexto en Cómo ocurrió la corrupción. Normalmente solo notará que hay un problema cuando hay un error downstream en el procesamiento, mucho después de que se escribieron los datos corruptos. Por ejemplo, puede obtener una excepción de puntero nulo debido a un campo obligatorio estar desaparecido Te darás cuenta rápidamente de que el problema es un campo faltante, pero tendrás No hay absolutamente ninguna información sobre cómo llegaron esos datos allí en primer lugar.

Limitaciones a los marcos de serialización

Los marcos de serialización solo comprueban que todos los campos requeridos están presentes y son de la Tipo esperado. No pueden verificar propiedades más ricas como "Las edades no deben ser negativas" o “las marcas de tiempo reales no deben estar en el futuro”. Los datos no coinciden estas propiedades indicarían un problema en su sistema, y no querría escrito a su conjunto de datos maestro.

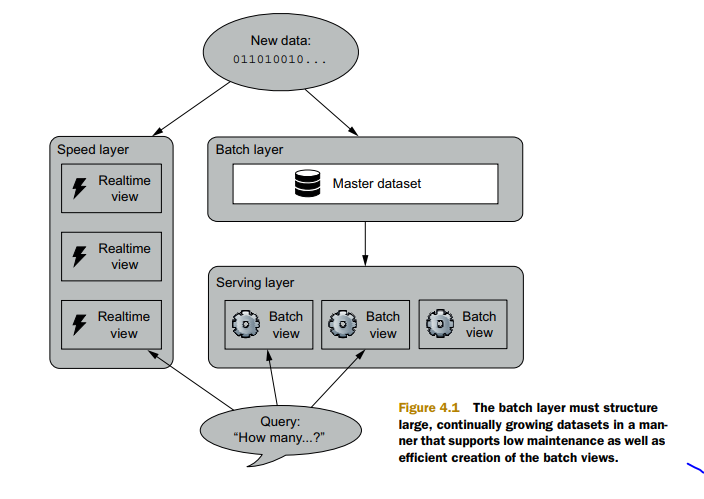
Esto puede no parecer una limitación porque los marcos de serialización parecen algo similar a cómo funcionan los esquemas en bases de datos relacionales. De hecho, puede haber encontrado Los esquemas de bases de datos relacionales son un dolor para trabajar y se preocupan de que hacer esquemas incluso Más estricto sería aún más doloroso. Pero te instamos a que no confundas lo incidental. complejidades de trabajar con esquemas de bases de datos relacionales con el valor de los esquemas sí mismos. Las dificultades para representar objetos anidados y hacer migraciones de esquemas. con bases de datos relacionales no existen al aplicar marcos de serialización Representar objetos inmutables mediante esquemas gráficos.

En la mayor parte, la implementación del esquema grafico ejecutable para SuperWebAnalytics.com es sencilla, se vio la fricción que aparece cuando se usa una serialización marco para este propósito, es decir, la incapacidad de hacer cumplir todas las propiedades. Las herramientas rara vez capturan los requisitos a la perfección, pero es importante saber que sería posible con las herramientas ideales.

**4. Data storage on the batch layer**

El siguiente paso es aprender físicamente almacenar esos datos en la capa de lote. La Figura 4 resume en dónde estamos Arquitectura lambda.

Este capítulo está dedicado al conjunto de datos maestro. El conjunto de datos maestro suele ser demasiado grande para existir en un solo servidor, por lo que se debe elegir Cómo distribuirá sus datos en múltiples máquinas. La forma en que almacena su el conjunto de datos maestro afectará la forma en el que se consumes, por lo que es vital diseñar su estrategias de almacenamiento con sus patrones de uso en mente



Requisitos de almacenamiento para el conjunto de datos maestro

Para determinar los requisitos para el almacenamiento de datos, debe considerar cómo sus datos se escribirán y cómo se leerá. El papel de la capa de lotes dentro de la arquitectura lambda afecta a ambas áreas. Discutiremos cada una a un alto nivel antes de proporcionar una Lista completa de requisitos.

La capa de lotes también es responsable de las funciones de cálculo en el conjunto de datos para producir Las vistas por lotes. Esto significa que el sistema de almacenamiento de capa por lotes debe ser bueno en leyendo muchos datos a la vez. En particular, el acceso aleatorio a datos individuales es no requerido

Sistemas de archivos distribuidos

Resulta que hay un tipo de tecnología con la que ya estás familiarizado. es un ajuste perfecto para el almacenamiento de capas por lotes: sistemas de archivos.

Los archivos son secuencias de bytes, y la forma más eficiente de consumirlos es escaneando a través de ellos. Se almacenan secuencialmente en el disco (a veces se dividen en bloques, pero la lectura y la escritura sigue siendo esencialmente secuencial). Tienes el control total sobre los bytes de un archivo, y tiene total libertad para comprimirlos sin embargo querer. A diferencia de un almacén de clave / valor, un sistema de archivos le brinda exactamente lo que necesita y no más, al tiempo que no limita su capacidad para ajustar el costo de almacenamiento en comparación con el costo de procesamiento.

Además de eso, los sistemas de archivos implementan sistemas de permisos específicos, que son perfectos para imponer la inmutabilidad.

Almacenamiento de un conjunto de datos maestro con un sistema de archivos distribuido

Los sistemas de archivos distribuidos varían en los tipos de operaciones que permiten. Algunos distribuidos los sistemas de archivos le permiten modificar los archivos existentes, y otros no. Algunos te permiten adjuntar a archivos existentes, y algunos no tienen esa característica. En esta sección veremos cómo puede almacenar un conjunto de datos maestro en un sistema de archivos distribuido con solo el más deshuesado de características, donde un archivo no puede ser modificado en absoluto después de ser creado.

Los requisitos de alto nivel para almacenar datos en la capa de proceso por lotes de la arquitectura lambda son sencillos. Se observó que estos requisitos podrían ser asignados a una lista de verificación para una solución de almacenamiento, y se observó que un sistema de archivos distribuido es un ajuste natural para este propósito.

**5. Data storage on the batch layer: Illustration**

-Los archivos se dividen en bloques que se reparten entre muchos nodos cluster.

-Los bloques se replican entre muchos nodos, por lo que los datos están disponibles incluso cuando las maquinas bajan

- los nodos realizan un seguimiento de los bloques para cada archivo y donde se encuentran esos bloques se almacenan.

**5.1.1 The small-files problema**

Cada tarea requiere una sobrecarga para planificar y coordinar su ejecución, y como cada archivo pequeño requiere una tarea separada, el costo es repetidamente incurrido esta propiedad de MapReduce significa que querrá consolidar sus datos. Si los archivos pequeños se vuelven abundantes dentro de su conjunto de datos. Se puede lograr esto ya sea por escribir código que use la API de HDFS o mediante un trabajo MapReduce personalizado, pero ambos enfoques requieren un trabajo considerable y conocimiento de los aspectos internos de Hadoop.

**5.2.2 Towards a higher-level abstraction**

**5.4 Summary**

Mantener un conjunto de datos dentro del ecosistema HDFS implica tareas comunes de agregar datos maestros, dividir verticalmente los datos en muchos files, y consolidar archivos pequeños.

Estas tareas se logran usando el API de HDFS, directamente es tediosa y propensa a errores humanos. Se vio la abstracción de Pail. Pail aisla de los formatos de archivos y la estructura de directorios de HDFS, lo que facilita la realización de procesos verticales sólidos y exigentes. Particionar y realizar operaciones comunes en su conjunto de datos. Usando la abstracción de cubo. En definitiva toma muy pocas líneas de código. La partición vertical ocurre automáticamente, y las tareas como los apéndices y la consolidación son simples de una sola línea.

Esto significa que uno puede centrarse en como desea procesar sus registros con HDFS Y Pail, se ha representado una forma de almacenar el conjunto de datos maestro que cumple todos los requisitos y es elegante de usar.

**6**. **Batch Layer**

**Summary:**

El objetivo de un conjunto de datos es responder preguntas arbitrarias sobre sus datos. En la arquitectura lambda, la capa de lotes se precomputa el conjunto de datos maestro en vistas por lotes para que las consultas se puedan resolver con baja latencia. Esto requiere un equilibro entre lo que se precomputara y lo que se computara en el tiempo de ejecución para completar la consulta. Haciendo un poco de cálculo sobre la marcha para completar las consultas, ahorrando tiempo para calcular grandes vistas de lotes.

La capa de lote es el núcleo de la arquitectura lambda. La capa de lote es de alta latencia por su naturaleza, y debe usarse la latencia como una oportunidad para realizar un análisis profundo y cálculos costosos que no se puede hacer en real time. Al diseñar vistar por lotes, hay una compensación entre el tamaño de la vista generada y la cantidad de trabajo que se requerirá en el momento de la consulta para finalizar.

La disyuntiva de Mapreduce proporciona primitivas consultas generales para precomputar consultas, funciona a través de todos sus datos de una manera escalable. Sin embargo, puede ser difícil pensar en un mapa reducido. Aunque más reduce proporciona tolerancia a fallos, paralelización y tareas programando, está claro que trabajar con Mapreduce en bruto es tedioso y limitante. Pensar en términos de diagramas de tuberías es una forma mucho más concisa y natural de pensar en el cálculo por lotes.

**7. Batch layer: Illustration**

**Summary:**

Un punto clave en el código de procesamiento de datos es no es diferente a cualquier otro código que escribas. Como tal, requiere buenas abstracciones. Que son reutilizables y compostables. La abstracción y la composición son las piedras angulares de la buena ingeniería de software.

En lugar de solo enfocarse en cómo JCascalog permite implementar diagramas de tuberías, iremos más allá de eso y se verá que Cascalog permite una amplia gama de abstracción y composición técnicas que simplemente no son posibles con otras herramientas. Se ha encontrado que la mayoría de los desarrolladores piensan en términos de que SQL es el estándar de oro de las herramientas de manipulación de datos, y nosotros encuentra que la mentalidad es severamente limitante. Muchas herramientas de procesamiento de datos sufren de incidental complejidades que surgen, no de la naturaleza del problema, sino del diseño de la propia herramienta. JCascalog habilita técnicas de programación. Eso permite escribir código muy conciso, muy elegantes.

La forma de expresar los cálculos es de crucial importancia si se quiere evitar la complejidad, prevenir errores, y aumentar la productividad. Las principales técnicas para combatir la complejidad son abstracción y composición, y es importante en el procesamiento de datos.

**8. An example batch layer: Architecture and algorithms**

**Summary:**

Especificamente aprender como crear la capa de proceso por lotes para el ejemplo de SuperWebAnalytics es bastante complejo para requerir una capa de lotes bastante sofisticada, pero no tan complejo para perderse en los detalles. En este capítulo se vio la arquitectura y los algoritmos para la capa de proceso por lotes (utilizando diagramas de tuberías) .

La capa de lotes para SuperWebAnalytics.com tiene una lógica sofisticada, pero es muy sencilla su implementación. Esto se debe enteramente a la naturaleza informática de la capa de lotes funcione en todos sus datos. Cuando se puede mirar todos los datos a la vez, y no está limitado por las limitaciones de los algoritmos incrementales: la creación de sistemas es a la vez más simple y fácil. El cálculo por lotes también le da una gran flexibilidad. Es realmente fácil de extender la capa de proceso por lotes para calcular nuevas vistas: cada etapa de flujo de trabajo es gratis para la ejecución arbitraria de todos los datos.

**9. An example batch layer: Implementation**

La capa de lotes para SuperWebAnalytcis.com es solo unos pocos cientos de líneas de código, pero la lógica de negocio involucrada es bastante sofisticada. Las diversas abstracciones encajan bien.

Respecto a la capa de lotes, los aspectos de inmutabilidad y recalculo la proporcionan tolerancia a fallas humanas, una propiedad no negociable de cualquier sistema de datos. Es sencillo de escribir el código para producir las vistas por lotes: la materia oscura como la tolerancia a fallas y la concurrencia se maneja para usted por el marco de cálculo. La capa de lote se simplifica enormemente. El problema de producir vistas en tiempo real porque las vistas en tiempo real solo necesitan representar una porción muy pequeña del conjunto de datos completo.