Filtros Bloom

Son una estructura de datos porbabilística y optimizada. Se usan para encontrar si un objeto pertenece o no a un dataset. Optimiza este tipo de peticiones usando funciones hash en los elementos a procesar. Cuando el resultado de una petición es positivo, entonces el objeto posiblemente pertenezca al dataset en cuestión, de todas formas pueden ocurrir falsos positivos. Cuando el resultado es negativo, entonces el objeto no pertenece al dataset, no hay falsos negativos. Esta pensado para volúmenes de datos a gran escala.

Un filtro bloom puede ser definido como una tabla o array compuesta por m bits. Inicialmente todos los bits están inicializados a 0. Para añadir un elemento x a la tabla, se usan funciones hash k para encontrar su posición en la tabla y se establecen dichos bits a 1. En un filtro bloom clásico no se pueden eliminar items.

Parametrización de los filtros de bloom

La probabilidad de falsos positivos para un elemento que no pertenece al set es:

$$ε = (1-(1-\frac{1}{m})^{nk})^k ≈ (1 - e^{-kn/m})^k$$

Por lo tanto, el numero de funciones hash óptimo es:

$$k = \frac{m}{n}ln2$$

Y el tamaño del flitro de bloom puede ser determinado como:

$$m = - \frac{n ln ε}{(ln 2)^2}$$

n es el número de objetos almacenados dentro del filtro de bloom.

Propiedades de los Filtros de Bloom

Podemos estimar el número de elementos en un filtro de bloom F como:

$$|F| ≈ - \frac{m}{l}ln(1-\frac{∑^m_{i=1}F_i}{m})$$

La unión de 2 filtros de bloom A y B puede ser computada aplicando una operación OR:

$$|A ∪ B| ≈ -\frac{m}{k}ln(1 - \frac{∑^m_{i=1}(A ∪ B)_i}{m})$$

La intersección de 2 filtros de bloom A y B puede ser computada aplicando una operación AND:

$$|A ∩ B| = |A| + |B| - |A ∪ B|$$

Consideraciones sobre los filtros de Bloom

- No son una estructura que almacena datos por sí misma, pero puede ser usada como un mecanismo de optimización para mejorar el rendimiento de muchas aplicaciones.
- La tasa de falsos positivos debe ser medida y monitorizada. El rendimiento de los filtros de bloom se puede desplomar si hay demasiados elementos insertados.

Funciones Hash

En teoría, se deben seleccionar k funciones hash diferentes para implementar en los filtros de bloom. En la práctica las funciones hash son generadas por un esquema de doble hasing:

$$h_i(x) = h_1(x) + i*h_2(x)$$

En este caso, dos funciones hash diferentes son requeridas. Tambiñen es común suar una función hash con valores de entrada divididos en dos partes.

Diccionarios y Descomposición

Muchas palabras pueden ser descompuestas usando múltiples reglas, pero algunas requieren buscar su separación en una tabla de descomposición. Un filtro bloom es usado para almacenar la representación de dicha tabla. Si la palabra no está almacenada en el bloom filter, su descomposición puede ser resuelta usando reglas simples. Si la palabra fue almacenada en el filtro, entonces se ha realizado una búsqueda de tabla. Los falsos positivos significan que la tabla fue buscada cuando no era necesario. Extensiones de esta aplicación pueden ser usadas para sistemas de revisión de pronunciación y diccionarios de contraseñas no válidas y nombres de usuario ya en uso.

Bases de datos

Optimiza operaciones JOIN en sistemas distribuidos. Un nodo A envía un Filtro de Bloom que representa un subset de datos del nodo B. El nodo B devuelve al nodo A pares que han coincidido con el bloom filter. Finalmente, los falsos positivos son eliminados de A. Usando esta aproximación se reducen el consumo de recursos y las comunicaciones.

Proxies de Cacheo distribuidos

Cuando hay un fallo de cache, entonces el proxy intenta adivinar si hay otro proxy que tenga la página web en cache. Intercambiar todo el contenido en cache entre todos los proxies es muy caro. Un filtro de Bloom que representa los contenidos de la cache puede ser usado. Cuando un proxy tiene que encontrar otro que tenga la web en caché, simplemente revisa su Filtro de Bloom. Un falso positivo significa que la petición será hecha al proxy para descubrir que no tiene la página en cache. En esquema, los falsos negativos son posibles ya que la cache pudo haber sido actualizada durante el proceso.

Problemas de seguridad

Los filtros de bloom no son una estructura de datos segura. No pueden ser considerados una medida de seguridad activa. Pueden ser utilizados para intentar preservar la privacidad de los datos que representan hasta cierto punto. Son vulnerables a ciertos tipos de ataques.

- Es fácil manipular y modificar sus contenidos para incrementar los falsos positivos (Ataque de polución)
- El análisis probabilístico usando frecuencias puede resultar en un enlace de registros con bases de datos externas para revelar datos.

El uso de funciones hash criptográficas no reduce el número de falsos positivos. Pueden ser usadas si se desea, pero pueden afectar considerablemente el rendimiento del filtro. Normalmente se usan funciones hash no criptográficas. Si se usa un esquema de doble hasheo basado en slicing, las funciones criptogáficas pueden ser una buena elección.

Filtros Bloom para enlace de registros que preservan la privacidad

El enlace de registros empareja registros de diferentes bases de datos que se refieren a la misma entidad, se suelen usar para soportar procesos como la realización de decisiones. Este problema suele surgir en ambientes relacionados con la salud. Se necesita publicar los datos para ayudar a la cominidad a realizar su propio análisis de datos de la mejor manera posible. Para ello se recomienda seguir los siguientes pasos:

1. Preprocesado: Se sanean y estandarizan los datos.
2. Filtrado: Optimiza el proceso de búsqueda reduciendo el espacio de búsqueda.

3. Comparación: Computa una puntuación similar
4. Clasificación: Hace decisiones de acuerdo a la métrica.

Intercambiar información sensibla con terceras partes tiene problemas de privacidad. La idea es que la tercera parte sea capáz de enlazar los registros sin revelar la información. Usando codificación clásica o mecanismos de cifrado puede resultar en computación segura muy cara que garantiza una privacidad robusta.

Respuesta Ordinal preservadora de la privacidad aleatoriamente agregable (RAPPOR)

Es un mecanismo para recoger estadísticas anónimamente desde el cliente de software del usuario, garantizando la privacidad y permitiendo el análisis de los datos sobre la información recolectada. Está basado en el mecanismo de respuesta aleatorizada. Este método permite recolectar estadísticas más de una vez preservando la privacidad. Los datos son representados por una cadena de bits que contiene una respuesta aleatorizada a una característica.

- Para valores categóricos, cada bit puede ser representado como la presencia o ausencia de dicha característica
- Para valores numéricos, cada bit puede ser asociado con un predicado para un rango de valores.
- Otras características no categoricas o no numéricas son tratadas con cadenas de caracteres.

Dichos valores son codificados usando filtros bloom y se le sañade salt con ruido para hacerlos más robustos contra ataques longitudinales. Este método tiene los siguientes parámetros:

- Cada usuario es asignado a uno de los M grupos distintivos
- Un flitro bloom de tamaño N con K funciones hash se usa para representar os predicados que van a ser reportados al servidor
- Se añade ruido al filtro de bloom creado de acuerto con 3 parámetros de privacidad: p, q y f

Procedimiento

- Dado un valor v, se hashea en el $FB(B)$ de forma que ciertos bits serán establecidos a 1 y el resto como 0.
- Se añade ruido a B para obtener una Respuesta Permanente Aleatorizada (PRR) B' que será guardada para futuras respuestas para el mismo valor v

$$B'_i = \begin{cases} 1, & \text{con probabilidad } \tfrac{1}{2}f \\ 0, & \text{con probabilidad } \tfrac{1}{2}f \\ B_i, & \text{con probabilidad } 1 - f \end{cases}$$

- Cada vez que el cliente envía un reporte al servidor, una respuesta aleatorizada instantanea (IRR) S es construida basada en B' estableciendo cada bit i en S a 1 con probabilidades:

$$P(S_i = 1) = \begin{cases} q, & \text{if } B'_i = 1 \\ p, & \text{if } B'_i = 0 \end{cases}$$

En este procedimiento B nunca se usa para generar un reporte, se usa B' aleatorizada para evitar ataques que puedan quitar le ruido para desenmascarar B. Debido a la forma en la que está construida, B' puede o no contener información sobre B. B' Ninca se envía directamente en los reportes, se le añade rudio previamente en S de forma que localizar al cliente basándose en B' sea más difícil. Los adversarios pueden aprender algo sobre B' con el tiempo pero no van a ser capaces de distinguir entre los bits verdaderso y el ruido, por lo que B no va a ser nunca revelada. A pesar del rudio introducido pro este procedimiento, es posible agregar reportes individuales y extraer información útil mediante el uso de análisis estadístico.

Análisis de datos

Como no se reportan datos a través de los filtros de bloom, debemos construir un set candidato compuesto por Z elementos de acuerdo a los datos que estamos recolectando. Para cada uno de los valores en los candidatos debemos identificar cual de los bits será activado cuando se aplican las funciones hash en ellos. Debido a la privacidad diferencial no podemos identificar los sets originalmente establecidos a 1, pero podemos estimar la media de cuanto ruido se ha añadido.

$$t_{i,j} = \frac{c_{i,j} - \left(p + \tfrac{1}{2} f q - \tfrac{1}{2} f p\right) N_j}{(1 - f)(q - p)}$$

Tras eso, se crea un sistema de ecuaciones lineales para computar el conteo de cada elemento en el set de caditadots.

$$X=\begin{bmatrix} X_{i,j,z} & \cdots & X_{i,j,z} \\ \vdots & \ddots & \vdots \\ X_{i,j,z} & \cdots & X_{i,j,z} \end{bmatrix},\quad \text{donde } X_{i,j,z}=\begin{cases} 1, & \text{Si } B_i=1 \text{ para el elemento } z \text{ en el grupo} j \\ 0, & \text{En caso contrario} \end{cases}$$