Framework para consultas *Top*-K sobre sistemas P2P estructurados

Oscar Gilberto Herrera Sánchez

Universidad de los Andes
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas y Computación
2008

Framework para consultas Top-K sobre sistemas P2P estructurados

Oscar Gilberto Herrera Sánchez

Trabajo de Grado Presentado como Requisito para Optar por el Título de Maestría en Ingeniería de Sistemas y Computación

Asesora
María del Pilar Villamil Giraldo

Universidad de los Andes
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas y Computación
2008

Tabla de Contenidos

1. Introducción

Las empresas actualmente se enfrentan al reto de utilizar gran cantidad información que se encuentra generalmente distribuida para tomar decisiones y ser competitivas. Es necesario entonces, contar con mecanismos de apoyo para acceder a la información y clasificarla según criterios definidos por los usuarios. Esta necesidad se expande hasta contextos aplicativos como los motores de búsqueda en Internet y las consultas en redes de sensores.

Por ejemplo, una consulta en un motor de búsqueda en Internet puede tener como resultado millones de candidatos de registros, normalmente solo se muestran unos 100 de ellos, esto es equivalente a hacer una consulta Top-*k* con los 100 resultados más relevantes con respecto a los parámetros (cadena de búsqueda) ingresados por el usuario.

Estas consultas también suelen ser comunes en redes de sensores como en el caso de querer determinar las temperaturas máximas que se han registrado en una planta nuclear, o tomar decisiones de Inteligencia de Negocios (BI) como en el caso de un sitio de comercio electrónico que quiere conocer cuáles son los productos más vistos por los usuarios.

Estos casos se caracterizan por el hecho de tener que agregar la información para poder desplegar un resultado, pues cada uno de los nodos (sensores o servidores) solo conoce su información local.

A medida que va creciendo la cantidad de información disponible se requieren sistemas que permitan almacenarla e indexarla, y mecanismos de consulta que maximicen su utilidad. Las consultas Top-*k*, permiten aprovechar la información almacenada en un sistema de datos, agregándola, proveyendo respuestas en un tiempo corto y consumiendo la menor cantidad de recursos computacionales (tráfico de red y procesamiento local).

Estas consultas son comunes hoy en día pues proveen los resultados más relevantes con respecto a los criterios definidos por el usuario sobre un conjunto de datos que puede estar centralizado o ampliamente distribuido y que puede tener pocos o miles de millones de registros.

Cada vez que se realiza una consulta Top-*k* sobre un sistema, se establece un puntaje estandarizado que mide la relevancia de cada entrada del conjunto de datos con respecto a las demás para seleccionar entonces solo las más relevantes para el usuario de acuerdo a los criterios establecidos.

Los sistemas distribuidos tradicionales son estáticos y suelen estar compuestos por decenas de sitios. Otro estilo de sistemas, los P2P, están compuestos por cientos, miles o millones de sitios autónomos que exigen nuevos retos. Estos últimos se dividen en sistemas con supernodos que actúan como puntos centrales de consulta y

almacenamiento, y sistemas sobre DHT (Tabla de Hash Distribuida) que distribuyen la información en todo el sistema según una o más funciones de hash que se evaluan sobre cada elemento para determinar en qué nodo debe ser almacenado y/o consultado.

En la actualidad existen varios algoritmos que proveen soluciones para este tipo de consultas en sistemas P2P con supernodos que sirven como repositorios centrales, sin embargo la cantidad de algoritmos disponibles para sistemas DHT son muy pocos.

Una de las principales ventajas de estos sistemas es que permiten almacenar la información de manera uniforme, de esta forma se puede tomar provecho de la capacidad de almacenamiento y procesamiento del sistema ampliamente distribuido como un todo. Por tanto existe la posibilidad de escalar tanto vertical como horizontalmente, lo cual requiere costosas implementaciones en sistemas tradicionales.

Existen varias implementaciones de sistemas P2P sobre DHT, entre ellas Chord (Stoica, et al., 2001), Pastry (Rowstron, et al., 2001), Tapestry (Zhao, et al., 2002), o CAN (Ratnasamy, et al., 2001), sin embargo según el conocimiento del autor hasta el día de hoy solo esta existe un algoritmo Top-*k* que describe cómo manejar los índices y el acceso a datos en estos sistemas.

Integrar un nuevo operador para evaluar consultas Top-*k* en sistemas como los descritos en el párrafo anterior requiere tener en cuenta que los datos y estructuras de índices disponibles (si existen) seguramente son utilizados por operadores ya implementados, y que debido a las características de estos sistemas realizar una actualización completa de los clientes que se conectan a él puede resultar imposible.

La propuesta descrita en este documento busca solucionar este problema, proveyendo un *Framework* que permite migrar fácilmente un algoritmo Top-*k* implementado en un sistema P2P sobre DHT a otro sistema de las mismas características que no cuente con él, sin afectar la información que tiene almacenada y sin requerir costosas actualizaciones.

Este documento continúa de la siguiente manera. En el capítulo 2 se presenta la problemática y los objetivos de este trabajo, en el capítulo 3 se presentan las consultas Top-*k* y sus aplicaciones comunes, en el capítulo 4 los algoritmos tradicionales para resolver consultas Top-*k*. En el capítulo 5 se tratan los tipos de sistemas de datos distribuidos y se tratan algoritmos para resolver Top-*k* en sistemas P2P con supernodos, en P2P sobre DHT y se hace una síntesis de sus características. En el capítulo 6 se propone PTop un *Framework* para implementar y migrar algoritmos Top-*k* en sistemas P2P sobre DHT, en el capítulo 7 se realiza una evaluación tanto teorica como practica sobre el algoritmo propuesto. Finalmente en el capítulo 8 se presentan las conclusiones y trabajo futuro.

2. Problemática y Objetivos

Existe un gran potencial en los sistemas P2P DHT para almacenar grandes volúmenes de información de manera durable y segura, sin embargo, la mayoría de sistemas existentes de este tipo que proveen un mecanismo para realizar consultas no implementan consultas tipo Top-*k*. Uno de los principales usos del almacenamiento de la información es poder consultar a manera de resumen tendencias sobre los diferentes atributos de interés para el usuario. El reto es, proveer un mecanismo eficiente para extraer esta información agrupada de manera confiable, aquí es donde las consultas Top-*k* toman importancia.

La dificultad para realizar estas consultas en sistemas P2P DHT -a diferencia de los sistemas tradicionales- es que la información se encuentra fragmentada tanto vertical como horizontalmente y distribuida geográficamente según sus atributos en varios lugares.

Estas características exigen hacer un adecuado uso de los recursos disponibles como lo son el tráfico de red, la capacidad de procesamiento requerida por los nodos que participan en la consulta, y el tiempo que toma obtener los resultados requeridos por el usuario según los atributos definidos en la función objetivo. Esto motiva los objetivos de trabajo que se presentan a continuación.

2.1 Objetivos generales

- Diseñar e implementar una estrategia para evaluar consultas Top-*k* en sistemas P2P sobre DHT.

2.2 Objetivos específicos

- Analizar diferentes algoritmos *Top-k* existentes, sobre sistemas P2P DHT para identificar características que permitan conceptualizar este tipo de soluciones.
- Diseñar e implementar un Framework que permita incorporar diferentes estilos de algoritmos para resolver consultas *Top-k* en sistemas P2P DHT.
- Validar el Framework propuesto tomando como base las restricciones impuestas por los sistemas P2P DHT y los estilos de soluciones Top-*k* estudiadas.

3. Consultas Top-*k*

Las consultas Top-*k* permiten seleccionar los objetos más relevantes dentro de un conjunto de datos con respecto a una función de optimización descrita por el usuario. Un ejemplo de una consulta en SQL que responde éste operador es:

```
select pv.product_id, count(pv.view_date)
from product_views pv
where pv.view_date > (sysdate - 1)
group by pv.product_id
order by pv.view_date
limit 10
```

En este caso el usuario estaría interesado en conocer cuáles son los 10 productos más vistos en las últimas 24 horas. Tenga en cuenta que el comportamiento esperado del modificador *limit* es que primero se ordenen todos los productos por el número de visitas para luego elegir solo las 10 primeras filas.

Una consulta Top-*k* está compuesta de dos partes: una función de agregación que determina el orden en el cual se van a acceder los índices y con la cual se elegirán los objetos más relevantes, y una función de optimización que relaciona los atributos de interés cuyos valores determinarán el puntaje para cada objeto. Durante el resto del documento expresa una consulta Top-*k* de la siguiente manera:

```
TopK 5 max(estatura / peso)
```

Un factor importante a tener en cuenta es que las consultas en las que puede estar interesado un usuario pueden involucrar atributos cuyos valores no son numéricos, por tanto se les debe aplicar una función definida que les asigna un valor para poder ser evaluados dentro de la función y comparados contra los demás elementos almacenados en el sistema.

3.1 Tipos de aplicaciones con interés en las consultas Top-*k*

Entre las aplicaciones que comúnmente utilizan este operador se encuentran (Zeinalipour, 2006):

1. Un conjunto de servidores en clúster como una tienda de comercio electrónico que tiene los mismos productos ofrecidos en cada uno de sus servidores, se quieren conocer los cinco productos más vistos por los usuarios en un día.

En este caso se conoce que todos los servidores tienen el mismo listado de productos, pero al estar todos detrás de un balanceador de carga no se puede conocer de antemano que producto se consultó más veces en total ya que cada servidor atendió una parte del tráfico del día y cada uno de ellos solo conoce su información.

2. Se tiene información clínica de pacientes almacenada en un sistema de datos, se conocen los atributos disponibles y se quieren relacionar varios de ellos para seleccionar los pacientes que maximizan una función con respecto a ellos.

   Ej: `TopK 15 Max((`n_r`+`n_c`)/e)& edad >= 10 & edad <= 15`
   n_r: número de enfermedades respiratorias padecidas
   n_c: número de enfermedades cardiacas padecidas
   `e`: estatura del paciente

   Esta consulta es más compleja que la anterior pues se requiere conocer los valores que toma cada atributo relacionado en la consulta para cada paciente con el fin de evaluar la función descrita y seleccionar el conjunto de pacientes buscado. En este caso puede que algunos pacientes listados en el sistema no tengan información para uno o más atributos descritos en la función de optimización. Adicionalmente, en este caso de seguro la cantidad de información que debe ser evaluada es mayor a la del caso anterior.

3. Se quiere relacionar la información médica del caso anterior con otra base de datos que contiene información de viajes realizados 3 meses antes de padecer la enfermedad.

   En este caso la complejidad radica en establecer la unión entre dos o más conjuntos de datos inicialmente aislados y en descartar rápidamente elementos que pertenecen únicamente a un conjunto ya que al final la respuesta estará en términos de los elementos en la intersección de los conjuntos de datos involucrados.

4. Indexar el contenido de sitios de Internet para hacer consultas sobre él y encontrar las páginas que tengan información más relevante con respecto a los criterios de búsqueda.

   En este caso el conjunto de objetos es demasiado grande (cada palabra utilizada en un artículo indexado) al igual que la cantidad de índices y referencias a documentos para cada llave.

4. Algoritmos tradicionales para evaluar el operador Top-k

Anteriormente se mencionó la solución más simple a este problema: evaluar la función de optimización sobre cada uno de los elementos contenidos en el sistema para posteriormente realizar un ordenamiento sobre su puntaje y obtener los Top-*k* resultados deseados. Aunque esta aproximación resuelve el problema implica iterar sobre todo el conjunto de datos una vez por cada elemento para completar la información y poder evaluar la función deseada teniendo una complejidad de n^2.

Adicional al desperdicio de recursos locales como memoria y procesador al tener que iterar sobre todos los elementos, si esta consulta se hiciera en un sistema P2P estructurado se requerirían mínimo n^2 mensajes en la red lo cual hace impráctico este algoritmo en estos ambientes.

4.1 Fagin

Este algoritmo al igual que todos los posteriores parte de un ordenamiento sobre los índices disponibles para cada atributo involucrado en la función de optimización.

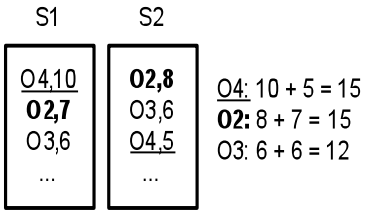

Figura 1: Algoritmo de Fagin

En su primer paso se obtiene de cada índice que contenga información sobre un atributo involucrado en la función de optimización la tupla <identificador de objeto, valor> para el elemento en la primera posición, a continuación se completa la información para los demás atributos involucrados en la función de optimización para cada objeto recibido de cada índice por medio de accesos aleatorios (consultar la información del objeto directamente) con el fin de obtener su puntaje agregado y compararlo contra los demás (Fagin, et al., 2001).

Este proceso se repite hasta que se hayan recorrido *k* elementos en cada uno de los índices y se haya evaluado completamente la función de optimización para cada uno de ellos. De estos se seleccionan los objetos que obtuvieron el mejor puntaje entre todos los objetos vistos.

4.2 Threshold Algorithm (TA)

Este algoritmo a diferencia del de Fagin accede a la información de cada índice en paralelo de tal forma que se obtiene una tupla <objeto, atributo> a la vez para cada atributo de cada índice y para cada iteración el nodo iniciador P_{init} establece un umbral/puntaje mínimo evaluando la función de optimización con los valores obtenidos en cada fila.

El valor obtenido para cada atributo en cada iteración se almacena en memoria para poder evaluar la función de optimización sobre el objeto una vez se tenga un valor para cada uno de sus atributos.

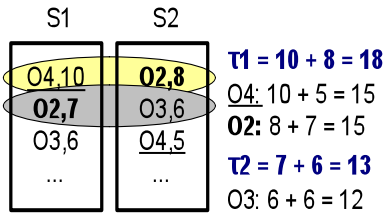

Figura 2: Threshold Algorithm

Cuando se tiene información completa para k elementos que superan el umbral establecido para la iteración en ejecución, se completa la información por medio de accesos aleatorios de los objetos vistos no completos (aquellos objetos para los que se ha obtenido valor para algunos de sus atributos pero para otros no) y se ordenan los objetos para obtener el conjunto resultado. Se garantiza que este algoritmo en el peor de los casos realiza tantas iteraciones como Fagin.

4.3 Otros algoritmos basados en TA

Existen varios algoritmos basados en TA que reducen el número de accesos a los índices para completar la información de los elementos que han sido vistos en las iteraciones pero para los que no se ha podido evaluar la función de optimización ya que hacen falta valores para algunos de sus atributos. Algunos de ellos:

- NRA (no accesos aleatorios) (Fagin, et al., 2001): en TA una vez se han encontrado k objetos con puntaje superior o igual al umbral se completa la información para los elementos vistos no completos, éste algoritmo sigue realizando TA hasta que logra completar la información para todos los objetos vistos hasta ese momento y puede ordenar los objetos según su resultado.

- BPA (algoritmo de la mejor posición) (Akbarinia, et al.): se acceden las listas en paralelo como en TA con la diferencia que para cada elemento encontrado se realiza el acceso aleatorio en ese momento como en Fagin pero se obtiene además del valor que toma el objeto la posición en la que se encuentra en ese índice. El umbral se calcula con los objetos que se encuentran en la mejor posición para cada uno de los índices no por filas como en TA. El objeto en la mejor posición para cada índice corresponde a la posición mayor consultada hasta ese momento mientras todas los objetos en las posiciones superiores hayan sido ya consultados. Se detiene cuando se han encontrado k objetos con puntaje superior al umbral establecido.

- TJA (algoritmo del umbral conjunto) (Zeinalipour, et al., 2005): el nodo iniciador P_{init} solicita a cada uno de los *m* nodos involucrados sus Top-*k* objetos y establece un umbral tal cual como se hace en TA. Se toma el puntaje resultado de evaluar la función con los valores de la fila *k* y se divide entre el número de nodos involucrados *m, T = puntajePosK/m*. Éste valor *T* se envía nuevamente a los *m* nodos y se solicita que envíen todas las tuplas <objeto, valor> para todos los objetos cuyo valor sea superior a *T*. P_{init} evalúa nuevamente la función de optimización con los nuevos datos obtenidos, si se puede evaluar la función para por lo menos *k* objetos se envían estos como el conjunto resultado, de lo contrario se completa la información para todos los objetos incompletos por medio de accesos aleatorios sobre cada índice.

Existen otros algoritmos como TPAT o TPOT (Yu, et al., 2005) basados en TJA que varían en la forma como calculan *T*, sin embargo no se encontraron documentos que hagan referencia a ellos a diferencia de TPUT expuesto a continuación.

Varios experimentos se han hecho también sobre los accesos aleatorios que se realizan en el último paso de TA y otros algoritmos que completa la información de los elementos vistos no completos. Este último paso requiere muchos mensajes de red y procesamiento de la maquina. Eliminar este último paso resulta en una respuesta aproximada al Top-*k* con mucho mejor desempeño.

4.4 TPUT

Éste algoritmo es una evolución a TA y en especial a TJA que garantiza que se puede obtener una respuesta Top-*k* en tres rondas de mensajes/fases. Inicialmente este algoritmo fue diseñado para funcionar sobre redes en topología de estrella, sin embargo los autores (Cao, et al.) también hacen referencia a cómo implementar el algoritmo en sistemas jerárquicos y sistemas P2P no estructurados.

Los autores definen el límite superior como el resultado de evaluar la función de optimización teniendo un valor para cada uno de los atributos relacionados en la consulta de interés.

Las 3 fases propuestas por el algoritmo son:

1. Determinar un límite inferior: el nodo iniciador P_{init} solicita a los *m* nodos que contienen información sobre los atributos de interés que envíen el conjunto de tuplas con los identificadores de objeto y valores correspondientes en las *k* primeras posiciones locales.

   Una vez se tienen estos valores, se evalúa la función de optimización con estos datos, asumiendo un valor constante (E.g. c=0) en los atributos para los cuales no se tiene información. Se organizan entonces los objetos con respecto a la función deseada (ascendente/descendentemente) y se toma el puntaje del objeto en la posición $k(\tau_1)$, luego se divide este valor entre el número *m* de nodos que

reportaron información y se establece éste valor como el umbral mínimo ($T = (\tau_1/m)$)

2. Eliminar los objetos no-candidatos y refinar el umbral: el nodo iniciador envía a los *m* nodos involucrados el valor *T* obtenido en la fase anterior y solicita que se le envíen todos los objetos y sus valores que localmente estén por encima de este valor.

   Los objetos que no aparezcan en esta fase definitivamente estarán por fuera de los Top-*k* buscados ya que se puede asegurar que su valor es menor que τ_1. Ahora, se vuelve a evaluar la función de optimización con la información disponible para cada objeto reportado, con esto se calcula el nuevo umbral mínimo (τ_2). Se eliminan del conjunto resultado todos los objetos cuyo límite superior sea menor que τ_2.

3. Identificar el conjunto de objetos resultado: se completa la información de los objetos para los que no se ha podido calcular su límite superior, para esto el nodo iniciador realiza una consulta sobre los nodos que contienen los objetos y solicita la información de los atributos faltantes para cada uno de ellos. Con esta información se evalúa la función de optimización para cada objeto, se ordena de acuerdo a la función de interés y se retornan los *k* elementos que obtuvieron el mejor puntaje.

Aunque este algoritmo obtiene el resultado más rápido que el TA, es dependiente de los valores que toman los atributos para cada objeto, en el paso 2 se calcula *T* asumiendo una distribución uniforme de los atributos por tanto de acuerdo a la selectividad de cada uno de ellos este valor puede resultar inadecuado trayendo o excluyendo información para objetos relevantes.

Lo anterior implica que si *T* da como resultado un valor muy bajo en la segunda fase el mensaje enviado desde los *m* nodos al nodo iniciador será muy grande. Esto requiere bastante consumo de red y procesamiento innecesario en el nodo iniciador.

5. Algoritmos Top-*k* en sistemas distribuidos a gran escala

Existen varios tipos de sistemas distribuidos donde se puede almacenar información hoy en día, entre ellos bases de datos en clúster, sistemas P2P con supernodos y sistemas P2P sobre DHT como se explicará con mayor detalle en la sección 4.1.

Todas las soluciones Top-*k* expuestas hasta éste punto en este documento funcionan sobre sistemas en bases de datos y sistemas P2P con supernodos ya que los índices y los datos en estos sistemas están centralizados y por tanto la aplicación de los algoritmos es inmediata. En el caso de aplicar alguna de estas soluciones en sistemas P2P sobre DHT se requiere crear estructuras sobre el sistema que permitan localizar la información en cada uno de sus pasos para poder procesar el algoritmo.

La única propuesta existente sobre sistemas P2P sobre DHT es DHTop el cual se presenta en la sección 4.4.Los otros dos algoritmos presentados en esta sección (KLEE, PJoin) corresponden a algoritmos que resultan deseables en estos sistemas debido a sus características y que pueden ser adaptados.

5.1 Sistemas distribuidos

Entre los diferentes sistemas que se utilizan hoy en día para almacenar datos en forma distribuida existen los sistemas de bases de datos en clúster, los sistemas P2P con supernodos y los sistemas P2P sobre DHT.

Los sistemas de datos en clúster son un conjunto de servidores con características similares donde cada uno de ellos toma un rol específico sobre el manejo, acceso y procesamiento de la información. Estos esquemas proveen escalabilidad horizontal, haciendo ver todo el sistema como un solo nodo.

Los sistemas P2P con supernodos se caracterizan por contar con repositorios centrales donde almacenan los datos (*meta-data*) e índices sobre la información contenida. Los nodos que actúan como clientes ven el sistema como un solo nodo debido a que las consultas se realizan sobre estos repositorios centrales.

En el caso de los sistemas P2P sobre DHT la localización de los datos es una tarea más compleja ya que estos por defecto no cuentan con índices sobre la información que contienen. Por otro lado, el hecho que la información no se almacene en un conjunto limitado de nodos sino de manera totalmente distribuida sobre el sistema implica que acceder a los datos requiere más mensajes de red y por tanto tiempos más altos de

respuesta. Estas características hacen difícil la implementación de cualquier algoritmo Top-*k* sobre estos sistemas.

5.1.1 Sistemas de Bases de Datos en clúster

Las bases de datos en clúster son un conjunto de componentes que proveen escalabilidad y estabilidad a bajo costo (Aycock, 2006). Estas características le permiten a aplicaciones empresariales de alta demanda estar disponibles y contar con mayor capacidad de procesamiento que una base de datos de un solo nodo.

En la actualidad, existen dos tipos comunes de implementaciones de clúster sobre bases de datos. Una está basada en instancias independientes, donde cada nodo guarda y es responsable por una parte de la información (shared-nothing); en la otra se comparte el acceso al medio de almacenamiento entre todos los nodos (shared-everything/shared-disk).

En el primer caso, se define en qué parte del sistema se almacenará la información correspondiente a cada atributo o sus índices. En el segundo caso, toda la información se almacena en un conjunto de discos que funcionan como uno solo. Esto último suele implementarse con sistemas RAID.

En ambos casos en estos sistemas se puede acceder a los índices para cada atributo de forma transparente, en el orden requerido y obteniendo solo una fracción de los datos por iteración, de tal forma que implementar cualquier algoritmo para resolver consultas Top-*k* es una tarea sencilla.

5.1.2 Sistemas P2P no estructurados

Los sistemas P2P no estructurados (Sung, et al., 2005) se caracterizan porque la información almacenada en ellos no tiene un orden particular ni se reparte entre los nodos conectados al sistema de alguna forma específica.

En su forma pura, un nodo solo conoce a sus vecinos de acuerdo a la topología aplicada. Sin embargo, la implementación más común de estos sistemas es aquella que cuenta con supernodos y nodos clientes.

Realizar una consulta Top-*k* en un sistema P2P no estructurado puro es una tarea que requiere su propio estudio y para la cual no existen soluciones en este momento.

Por otro lado están los sistemas P2P estructurados con supernodos o repositorios centrales y otros nodos que actúan como clientes del sistema. En estas arquitecturas, al conectarse un cliente al sistema, éste envía la meta-data de los objetos que contiene al supernodo al cual se conecta.

Los supernodos funcionan como repositorios centrales donde los demás clientes pueden realizar búsquedas a las cuales responden con el identificador de los nodos que registraron información según la meta-data recibida. Los nodos clientes establecen conexiones entre ellos para completar la información de los objetos buscados.

En estos sistemas con repositorios centrales hacer una consulta Top-*k* es una tarea equivalente a una base de datos donde los índices pueden estar particionados entre varios supernodos, pero en todo caso se cuenta con toda la información de valores que toma un atributo y objetos que los referencian de primera mano, esto permite al igual que en el caso anterior, utilizar cualquier algoritmo de forma automática.

5.1.3 Sistemas P2P sobre DHT

Los sistemas P2P sobre DHT (Sung, et al., 2005), almacenan los objetos en cada nodo según su rango asignado dentro de una función de hash predefinida. Estos sistemas a diferencia de los anteriores, no cuentan con supernodos. Esto quiere decir que no cuenta con repositorios centrales de información.

Una DHT (tabla de hash distribuida) funciona tal cual como lo hace una tabla de hash en una máquina local: cada objeto es asignado a un nodo, el cual es determinado con la función de hash aplicada sobre la llave de dicho objeto y es el responsable de almacenarlo.

Al conectarse un nuevo nodo al sistema se le asigna un rango de la función de hash utilizada, todos los objetos para los que el resultado de la función de hash del sistema sobre su llave corresponda a éste rango serán almacenados por este nodo. De la misma forma todos los objetos que contenía al momento de conectarse se almacenarán en diferentes nodos de acuerdo al resultado de aplicar la función de hash sobre sus llaves.

Existen varias implementaciones de sistemas basados en DHT, tales como Chord (Stoica, et al., 2001), Pastry (Rowstron, et al., 2001), Tapestry (Zhao, et al., 2002), o CAN (Ratnasamy, et al., 2001). Las operaciones comunes que proveen estos sistemas son `get(key)` -que localiza y retorna un objeto dada su llave-, y `put(key, data)`-que almacena un objeto en la DHT según la llave dada-.

Estos sistemas debido a sus características, almacenan cada objeto en varios nodos usando varias funciones de hash sobre su llave con el fin de garantizar disponibilidad de la información. Esto implica que la información en estos sistemas es almacenada dinámicamente en un grupo de nodos variables, donde ninguno de estos es realmente propietario de la información, ya que al agregar o eliminar nodos o balancear nuevamente el sistema la asignación del rango de valores de hash que almacena cada nodo puede variar.

Estos sistemas proveen entonces solo un tipo de sistema de localización por defecto: igualdad. Si se quiere consultar un objeto cuya llave es *x,* el sistema es entonces capaz de encontrar el nodo que almacena ese objeto y recuperarlo.

Para evaluar una consulta Top-*k* se requiere comparar los valores que toman los atributos de cada objeto con los demás, lo cual implica que se requieren índices. Esto es resuelto por sistemas de interrogación de datos como PinS (Villamil, et al., 2004) que permiten realizar consultas que incluyen términos de igualdad, desigualdad y *joins*.

De lo mencionado anteriormente se puede deducir que el principal problema para implementar estos algoritmos en estos sistemas es la fragmentación de los datos, y aún más, contar con índices sobre los atributos de los objetos almacenados en el sistema.

Por otro lado, es deseable que la consulta se realice con la menor cantidad de mensajes posibles. Esto con el fin de disminuir el tráfico en la red, ya que es el recurso común compartido entre todos los nodos.

5.2 Sistemas P2P

5.2.1 KLEE

Este algoritmo toma como base a TPUT y propone optimizaciones en el uso de la red y el procesamiento local de cada nodo involucrado en la búsqueda. KLEE (Michel, et al., 2005) provee respuestas aproximadas no respuestas exactas como los demás algoritmos, por esto KLEE evita los accesos aleatorios que se realizan en los demás algoritmos los cuales garantizan exactitud en la respuesta.

Una de las principales características de este algoritmo es que utiliza información estadística con el fin de aproximarse al límite superior de cada objeto candidato desde su primera fase y descartar rápidamente la mayor cantidad de objetos con baja probabilidad de estar en el conjunto resultado. Esto disminuye el tráfico sobre la red siendo este uno de los principales problemas de TPUT. Para lograr esto KLEE resume la información almacenada en cada nodo por medio de histogramas. Utilizando *BloomFilters* (Broder, et al., 2004) prueba la pertenencia de un objeto a una celda específica dentro del histograma.

Los BloomFilters funcionan como un conjunto de funciones de hash que se aplican a cada objeto almacenados en el sistema y se almacenan en un arreglo binario donde por defecto cada posición está apagada (marcada en 0).Los bits se van encendido (marcando en 1) como resultado de aplicar la función de hash sobre la llave de cada objeto, cada resultado corresponde a una posición dentro del arreglo para cada función de hash. Se tiene entonces un arreglo de *b* columnas determinadas según el espacio de valores que puede tomar la función de hash y *h* filas de acuerdo al número de funciones de hash que se utilicen en el BloomFilter. Se utiliza más de una función con el fin de disminuir la probabilidad de obtener falsos negativos.

Cada nodo almacena para cada celda del histograma creado su límite inferior (*lb*), límite superior (*ub*), promedio (*avg*), cantidad de documentos contenidos en esa celda ($freq_i$), y el BloomFilter ($filter_i$) correspondientes por sus nombres o siglas en inglés.

Estos filtros son la base en todas las fases del algoritmo además de ser el principal tráfico enviado entre los nodos. Probar si un objeto pertenece a una celda especifica es evaluar la llave del objeto con las funciones de hash definidas y buscar el bit correspondiente en cada una de las *h* filas del filtro para cada celda. Sí se encuentra el bit correspondiente en cada una de las filas encendido entonces el objeto pertenece a esa celda, o es un falso positivo (Figura 3).

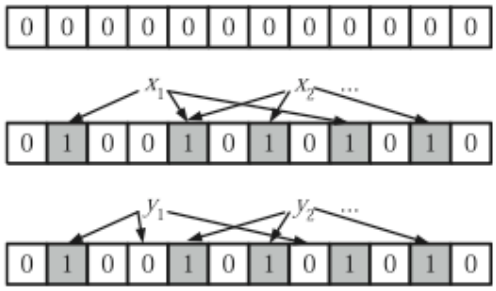

Figura 3 (Broder, et al., 2004): Prueba de existencia de un objeto en un BloomFilter

Éste algoritmo se desarrolla en 3 o 4 fases y esto se decide en tiempo de ejecución en la segunda fase de acuerdo a lo buena o mala que sea la aproximación del resultado parcial obtenido en la primera fase y a la simulación que se realiza para determinar cómo se logra reducir más el tráfico en la red, siendo éste último el criterio de mayor interés para los autores.

Para reducir el número de elementos no significativos enviados entre nodos, el algoritmo crea una CLF (lista de filtros candidatos), esta lista contiene para cada índice el resumen de objetos para los cuales el valor del atributo es superior al umbral establecido. Cada nodo utilizando una sola función de hash envía a P_{init} su resumen. El tamaño del arreglo (número de posiciones *R*) sobre el cual cada nodo enviará su resumen es el mismo para todos los nodos y es determinado por P_{init}. Cuando el nodo iniciador tenga el resumen recibido de cada uno de los nodos tendrá entonces una matriz de *R* columnas según el número de posiciones establecido para el arreglo resumen y *m* filas según el número de índices consultados (normalmente igual al número de atributos usados en la función de optimización).

Las fases en las que ocurre el algoritmo son exploración, optimización, reducción de la lista de candidatos y recolección de la información de la lista de candidatos como se presenta a continuación:

1. Exploración: El nodo iniciador P_{init} envía la consulta a los nodos que contienen información sobre los atributos involucrados. Cada uno de estos *m* nodos retorna para sus primeras celdas (según el histograma creado), las tuplas <identificador de objeto, valor>, su límite inferior, límite superior,

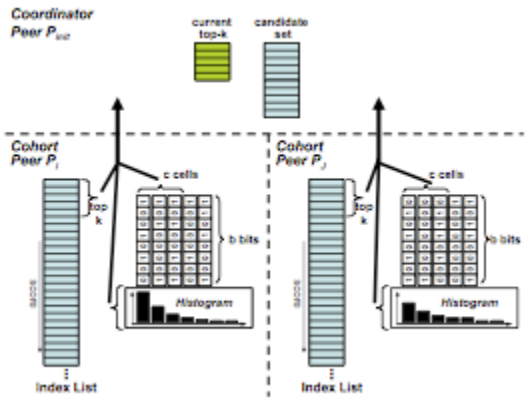

Figura 4 (Michel, et al., 2005): KLEE fase de exploración

promedio, frecuencia y filtro. Para las demás celdas solo envía el promedio, la frecuencia y el filtro (Figura 4).

Con la información obtenida de los m nodos P_{init} evalúa la función de optimización sobre cada objeto con los valores recibidos en cada índice para cada atributo. Para los objetos para los cuales un índice P_i no haya retornado un valor se utilizan los BloomFilters para determinar a cual celda pertenece y utilizar el valor promedio de la celda para evaluar la función de optimización. Si no se encuentra el objeto en alguno de los índices entonces se utiliza un promedio de los promedios de las celdas más bajas para ese atributo.

2. Optimización: no implica comunicación con los demás nodos, ésta fase solo se realiza en el nodo iniciador P_{init}. Esta fase busca determinar si se debe ejecutar la fase de reducción de lista de candidatos o no. Esto lo hace estimando el tamaño R de la CLF que va a recibir de cada nodo de acuerdo al umbral establecido en el paso anterior $T = TopK/m$. Estos valores, el tamaño esperado de la lista y el umbral establecido son enviados en el paso siguiente si se determina que se debe ejecutar.

3. Reducción de la lista de candidatos: ésta fase es opcional como se describió en el punto anterior siendo este quien determina si se ejecuta o no. En esta fase se solicita a cada uno de los m nodos que envíe en un arreglo binario del tamaño estimado R en el paso anterior los documentos que están por encima del umbral establecido. Cada uno de los m nodos envía a P_{init} su CLF. Al final de esta fase P_{init} tendrá una matriz de tamaño R x m donde las columnas contienen los objetos con valor para sus atributos superior al umbral y las filas corresponden a cada uno de los índices consultados.

4. Recolección de información de la lista de candidatos: en esta fase, P_{init} sobre la matriz formada en el paso anterior, busca objetos que tengan una cantidad alta de bits encendidos (varios bits en la misma columna), lo cual determina el conjunto de objetos candidatos para el resultado. Esto garantiza que el valor para varios de los atributos de los objetos en S está por encima del umbral (Figura 5 (Michel, et al., 2005): KLEE fase de recolección de información).

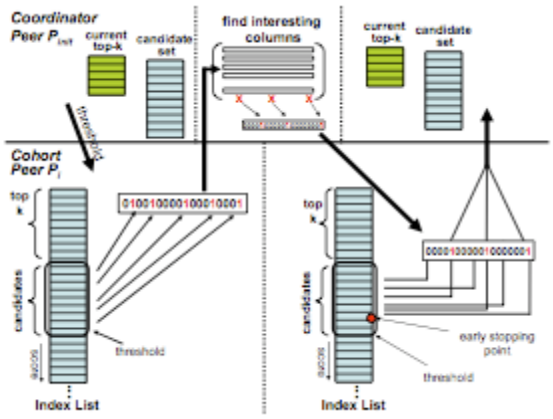

Figura 5 (Michel, et al., 2005): KLEE fase de recolección de información

La última ronda de comunicación con los demás nodos solicita a cada uno de ellos enviar la información completa para los objetos candidatos en el conjunto S, sobre estos se evalúa la función de optimización, se ordenan y se obtiene el resultado.

Una característica importante de este algoritmo es que provee varios parámetros para su configuración, entre ellos, la cantidad de funciones de hash utilizadas por el BloomFilter con el fin de reducir la posibilidad de falsos positivos, la cantidad de celdas recibidas en el primer paso con la información completa para disminuir la cantidad de objetos para los que toca usar el promedio para aproximar su valor y evaluar la función de optimización y evitar el paso 3, y el tamaño *R* del arreglo para la CLF que se utiliza en el paso 3.

Una implementación de este algoritmo es usada en Minerva (Michel, 2007) para resolver el problema de consultas Top-*k* en Internet a modo de motor de búsquedas. Como se describió anteriormente el principal problema en éste caso es la gran cantidad de términos que deben ser indexados y debido a las características del sistema P2P sobre el cual lo implementan, nodos que contengan información critica de un fragmento del índice podría desconectarse, para lo cual proponen redes de índices de términos (TINs) que garantizan disponibilidad y accesibilidad a los índices en cualquier momento.

5.2.2 PJoin

PJoin (Zhao, et al., 2005) es un algoritmo propuesto sobre sistemas P2P con supernodos que evalúa consultas Top-*k* que incluyen *joins* en dos fases usando el algoritmo *PSel* (Zhao, et al., 2007). Una operación de *join* puede relacionar varios tipos de objetos en un mismo sistema de datos o en varios.

PJoin funciona como se describe a continuación:

1. Se selecciona un conjunto de objetos candidatos comunes para los conjuntos de datos involucrados. Para esto se aplica una función de hash común sobre los valores que toman los atributos relacionados en cada uno de los conjuntos y se selecciona por medio de *PSel* un conjunto de objetos candidatos que pertenezcan a todos los conjuntos involucrados. Estos se copian a un nuevo sistema donde almacenaran temporalmente mientras dura la ejecución del algoritmo.

2. Se envía la consulta original a cada uno de los sistemas en paralelo que el conjunto de objetos candidatos para ser evaluado en cada uno de ellos, una vez más al obtener los resultados se utiliza el algoritmo *PSel* para seleccionar los objetos con mejor puntaje y darlos como resultado.

El algoritmo *PSel* envía la consulta a cada uno de los supernodos del sistema, cada uno de estos calcula el puntaje más alto para los objetos que tiene almacenados y lo envía al nodo iniciador junto al identificador del nodo, éste selecciona los *k* nodos con los puntajes más altos recibidos y les solicita calcular sus *k* puntajes locales y enviarlos nuevamente al nodo iniciador. Finalmente éste ordena y retorna el resultado.

5.2.3 DHTop

Este algoritmo está basado en TA y es una adaptación para sistemas P2P sobre DHT. Los autores proponen un esquema de almacenamiento para los índices sobre los datos que permiten evaluar el algoritmo en estos sistemas, al igual que optimizaciones para aprovechar mejor los recursos disponibles.

Debido a que los sistemas P2P sobre DHT solo proveen la capacidad de localizar un objeto por su llave, los autores (Akbarinia, et al., 2006) proponen un esquema donde los índices para cada atributo son almacenados como valores dentro de un conjunto de objetos con llaves conocidas indexados en el sistema. Los índices son un arreglo de tuplas *[<valor, [identificadores de los objetos que toman ese valor]>]* para cada valor encontrado en los objetos indexados en el sistema.

Ej. [<15, [o1, o3, o7]>, <13, [o2, o6]>, <9, [o4, o10]>, …]

Los autores asumen que se conoce la distribución de los valores que toman los atributos indexados y de acuerdo a esto el índice para cada atributo se parte en subconjuntos llamados subdominios.

Ej. $sd_{\text{atrib}1}$: [v1-v2), sd_{atrib2}: [v2, v3), sd_{atrib3}: [v3, v4)…

Teniendo estos índices ejecutan el TA con la única diferencia que se trae no solo un valor por iteración sino un subdominio completo para cada atributo, luego en el nodo iniciador se ejecuta el algoritmo original tal cual se describe completando la información de los atributos por cada objeto para evaluar la función de optimización y compararla contra el umbral establecido para cada iteración.

Si con el primer subdominio no se logran encontrar los *k* elementos buscados se pide entonces el segundo subdominio para cada atributo y así sucesivamente hasta encontrar los *k* elementos comunes en todos los índices. Posteriormente se completa la información para los objetos vistos no completos usando la llave de estos para obtener el valor de todos los atributos involucrados en la función de optimización, poder evaluar la función y comparar su resultado con los demás.

5.3 Síntesis

En todos los sistemas y algoritmos descritos anteriormente existen elementos en común que resultan críticos al momento de realizar una consulta; los factores más analizados son el uso de la red y el tiempo para obtener una respuesta.

Al momento de adaptar e implementar estos algoritmos sobre un sistema P2P sobre DHT es deseable contar con una organización de objetos y características físicas adecuadas que provean tolerancia a fallas, disponibilidad de la información, estabilidad y mantenibilidad de acuerdo a las necesidades requeridas en cada caso.

- Índices

  Aunque cada tipo de consulta que se quiera realizar requiere un tipo de índice diferente, se puede generalizar que el conjunto de atributos sobre el cual se permite hacer consultas es de dominio conocido.

  En general los índices pueden tener cualquier forma mientras se mantenga un concepto de orden sobre los valores que toman y se relacione al objeto que lo describe, en otras palabras el índice debe contener por lo menos una tupla <identificador de objeto, valor>.

  Un índice que además provea información estadística sobre los contenidos del nodo consultado como el propuesto en KLEE puede ser de gran utilidad porque permite descartar un gran conjunto de valores no significativos al realizar aproximaciones sobre los valores que toma cada atributo.

  Almacenar la lista de objetos que toman un valor para un atributo en un mismo índice como lo propone DHTop puede resultar muy costoso en caso al insertar nuevos datos al sistema. Por otro lado, almacenar los objetos que toman un valor por aparte del índice aumenta la cantidad de mensajes requeridos para relacionar los objetos en el caso de consultas que impliquen múltiples atributos.

- Capacidad de procesamiento local

  De todos los algoritmos vistos anteriormente al momento de ejecutar una consulta existen dos roles que toman los nodos, nodo iniciador y nodo de consulta, el primero es quien despliega la consulta en el sistema y realiza la mayoría del procesamiento local en la mayoría de algoritmos, el segundo tipo contiene los índices sobre los valores que toman los atributos de los objetos indexados.

  Algoritmos como KLEE ponen una gran responsabilidad a los nodos involucrados en la consulta, al tener que mantener sus índices actualizados y seleccionar dinámicamente los objetos relevantes en cada paso. Esto le da la ganancia de procesamiento distribuido (en este caso parcial) lo cual es una característica deseable en estas consultas más si se tiene una gran cantidad de datos.

  En los demás algoritmos todo el procesamiento se hace en el nodo iniciador P_{init}, la función de los demás nodos que contienen los índices y que almacenan los objetos es solo proveer esta información al nodo que solicita la consulta. Esto implica bastante procesamiento en el nodo iniciador.

  La mayoría de los autores realizan pruebas sobre sus algoritmos aumentando el número de atributos involucrados en la consulta Top-k, el resultado suele ser que el número de mensajes de red aumenta como es de esperarse, sin embargo también suele aumentar el tiempo requerido para obtener el resultado, esto debido a que todo el filtrado de objetos y evaluación de la función de optimización se

hacen en el nodo iniciador sin hacer uso de la capacidad de procesamiento de los nodos involucrados.

- Volatilidad de la información

  En este punto es importante determinar cómo se afectan los índices al agregar, modificar, ó eliminar objetos del sistema. Si se tiene implementado un algoritmo como DHTop realizar cualquiera de las operaciones anteriores puede resultar muy peligroso puesto que la distribución de los valores que toma un atributo puede variar y un supuesto del índice propuesto por ellos es conocer de antemano esta distribución.

  El problema al afectar la cantidad de datos almacenados por cada subdominio es que el tamaño de los mensajes enviados en la red al nodo iniciador aumenta y por tanto la cantidad de datos que debe procesar. Si no se tiene en cuenta este requerimiento con el tiempo un nodo puede convertirse en un cuello de botella para las consultas Top-*k* que se realicen sobre el sistema.

  En un sistema que funcione con BloomFilters como el propuesto en KLEE el principal problema es eliminar un elemento del sistema, pues cada nodo contiene un índice según las funciones de hash seleccionadas construido sobre la llave de los objetos que tiene almacenados. Eliminar un objeto implica eliminar el registro de cada uno de los filtros creados con las funciones de hash, sin embargo no se tiene garantía que no se está borrando información para otro elemento cuyo resultado al aplicar una función de hash puede ser el mismo que el del elemento que está siendo eliminado.

  Para permitir eliminar elementos en un BloomFilter se requiere cambiar la estructura del arreglo, no almacenando datos binarios sino contadores que indiquen el número de objetos que dan como resultado ese mismo valor a cada función (Broder, et al., 2004). Eliminar un objeto entonces implica disminuir el contador en 1. Por otro lado esta solución tiene el problema que hace que el tamaño de los mensajes aumente.

- Seguridad de la información

  Ninguno de los algoritmos presentados anteriormente trata el tema de la seguridad como factor relevante. Dentro del estudio realizado solo se encontró referencia a un algoritmo que considera que solo el resultado debe ser identificable (Vaidya, et al.), ninguna de la información intercambiada en los pasos intermedios para su cálculo.

  Otro enfoque que puede ser tenido en cuenta es restringir el acceso a ciertos atributos para algunos usuarios, en cuyo caso la información sobre los privilegios de acceso con los que cuenta un usuario debería ser manejada por la DHT.

Si se quisiera implementar un sistema que tuviera en cuenta este requerimiento se podría almacenar como otro objeto información sobre la lista de usuarios que pueden acceder a cada índice o recuperar información sobre un conjunto de objetos. Éste objeto tendría la restricción que debe estar firmado digitalmente por un administrador del sistema. Una vez un usuario fuera autenticado se generaría un *token* que le permitiría acceder a los datos autorizados en un esquema como el manejado por Globus Alliance con sus certificados Proxy (Alliance, 2008).

- Funciones soportadas por los algoritmos

  Una característica común a todas las soluciones existentes es que solo funcionan sobre funciones monótonas ó funciones IOD-EV (crecientes o decrecientes con respecto a cada variable por sus siglas en inglés en DHTop) (Akbarinia, et al., 2006), lo cual es un prerrequisito importante pues se parte del hecho que se acceden las listas de atributos individuales en orden ascendente o descendente según sea el caso lo cual debe garantizar que se encontrarán primero los mejores puntajes.

- Propuestas de solución

  En los algoritmos vistos se identifican tres mecanismos de solución de consultas Top-k: consultar directamente sobre los valores que toman los atributos e intercambiarlos para su evaluación como lo hacen Fagin, TA, TJA, TPUT, DHTop, entre otros; consultar la posición dentro del índice en la cual se encuentra el valor para el atributo de cada objeto como lo propone BPA; utilizar información estadística generada con base a los índices existentes como hace KLEE al utilizar histogramas, promedios, máximos, mínimos y conteo de elementos para obtener el resultado.

  Los algoritmos vistos que utilizan los valores o la posición en la cual se encuentra el valor para un atributo de un objeto dan como resultado el conjunto de Top-*k* exacto, KLEE por su lado utilizando información estadística da como resultado un conjunto Top-*k* aproximado con pequeñas perdidas en exactitud según describen sus autores.

6. PTop: Framework para consultas Top-*k* sobre DHT

Como se describió anteriormente existen muchos algoritmos que resuelven el problema de Top-*k* en sistemas con P2P con supernodos y con repositorios centrales. El principal reto es adaptar estos algoritmos a un sistema P2P DHT permitiendo portabilidad con respecto a las implementaciones hechas. Esto implica resolver el problema de utilizar estos algoritmos bajo un esquema de fragmentación vertical de datos.

Con el objetivo de resolver éste problema en este capítulo se presenta PTop. La sección 6.1 presenta la metodología a utilizar para proponer el Framework. La sección 6.2 hace una introducción a PTop y a su funcionamiento. La sección 6.3 da un recorrido sobre sus características no funcionales. La sección 6.4 presenta el *Framework* propuesto y se describen sus componentes. La sección 6.5 analiza la propuesta con respecto a procesamiento distribuido y la sección 6.6 con respecto a integración con otros operadores. En la sección 6.7 se describen algunos componentes que servirían de soporte al *Framework* propuesto. Finalmente en el capítulo 7 se presenta un caso de estudio sobre el cual se evalúa la propuesta y se presenta un ejemplo de su uso implementando un algoritmo que permite la evaluación del operador Top-*k* sobre un sistema que no cuenta con él.

6.1 Metodología

El proceso que se ha seguido (Fayad, et al., 1999) para proponer PTop se basa en primera instancia en las conclusiones obtenidas en los capítulos anteriores donde se ha buscando caracterizar los problemas que implica la implementación de algoritmos para consultas Top-*k* sobre sistemas P2P DHT. Partiendo de esto, se han establecido los siguientes requerimientos al momento de proponer PTop.

- Generar y mantener los índices faltantes y requeridos para soportar el algoritmo Top-*k* implementado.
- Distribuir los índices a través del sistema P2P DHT de manera confiable.
- Mantener la información almacenada previamente en el sistema sin cambios, permitiendo que las operaciones ya implementadas sigan funcionando.
- Interactuar con diferentes versiones de la aplicación en los nodos conectados al sistema sin requerir su actualización.
- Portabilidad de las implementaciones realizadas sobre estos sistemas, permitiendo aprovechar las características de los resultados obtenidos por estas consultas en sistemas con datos preexistentes sin realizar cambios sobre ellos.

Con estos requerimientos y siguiendo el proceso propuesto, a través de este capítulo se explica por medio de un diseño orientado por objetos una solución a este problema, el dominio sobre el que actúa y sus diferentes características. En el siguiente capítulo se

presenta una implementación de ejemplo de un algoritmo para consultas Top-*k* utilizando PTop sobre un sistema P2P DHT existente.

6.2 Descripción general de la solución

PTop surge con la idea de resolver los problemas asociados a realizar consultas Top-*k* sobre la información en sistemas P2P sobre DHT. Entre ellos: varias implementaciones P2P DHT con estructuras de índices diferentes, generación y el mantenimiento de índices alternos que soportan el algoritmo implementado, versiones diferentes en los participantes del sistema P2P DHT que brinda consultas declarativas, y el procesamiento distribuido de la búsqueda.

PTop requiere conocer las estructuras existentes en el sistema P2P DHT para implementar el algoritmo. PTop requiere extender el protocolo existente para comunicarse entre los nodos haciendo las fases y tareas de búsqueda definidos en el algoritmo implementado para ejecutarse en paralelo en diferentes nodos.

La implementación de un algoritmo Top-*k* en PTop parte por extender los índices existentes en el sistema; en los capítulos pasados se describen las estructuras de datos que permiten realizar las consultas para cada algoritmo, lo cual impone un problema de entrada al implementar un algoritmo portable debido a las varias implementaciones de sistemas P2P sobre DHT existentes, también expuestas anteriormente. PTop abstrae en un lenguaje portable la generación de índices requiriendo la descripción de los índices y estructuras con las que actualmente cuenta el sistema donde se está migrando el algoritmo.

La primera tarea que realiza PTop entonces, es validar que el sistema cuenta con la información tal cual se describió en el paso anterior. Una vez esta validación es completada exitosamente PTop genera los nuevos índices según las descripciones realizadas por el implementador. Debido a que se espera que la cantidad de información en el sistema sea bastante grande, PTop realiza una partición sobre el espacio de datos según la distribución estadística de los mismos, y registra el segmento en el que está trabajando en un índice accesible a todos los clientes con el nuevo operador implementado que potencialmente pueden participar en la generación de los índices, de esta forma se evita que más de un nodo trabaje en la generación del índice para un mismo espacio de datos. Al finalizar la generación del índice del espacio asignado se marca en el espacio compartido el índice como finalizado y la hora en la que termino.

Una vez PTop ha generado los índices requeridos en el nuevo sistema, se pueden realizar consultas utilizando el algoritmo Top-*k* desde los nodos clientes que cuentan con el nuevo operador implementado. Si la implementación del sistema P2P DHT soporta extender el protocolo de comunicación entre los nodos y la implementación del algoritmo Top-*k* soporta tareas a ejecutarse en paralelo, PTop hará uso de ellas.

El implementador hace uso de las clases definidas por PTop para enviar la consulta al sistema y para obtener los resultados y presentarlos al usuario.

6.3 Características de PTop

Entre las principales características que ofrece PTop se encuentran

- Portabilidad: el algoritmo implementado para resolver la consulta Top-*k* es independiente del sistema DHT de base y del sistema manejador de datos P2P.
- Adaptabilidad: constantemente son propuestos nuevos algoritmos Top-*k* que hacen mejor uso de los recursos del sistema y que al mismo tiempo proveen respuestas más rápidas. *PTop* permite intercambiar el algoritmo implementado según sea la necesidad del sistema o el estado del arte en Top-*k*.
- Facilidad de implementación: al implementar un nuevo operador no se afectan los datos, las estructuras de índices ni los operadores existentes. *PTop* se basa en la idea de adaptarse al nuevo ambiente, y no que el ambiente se adapte a él.
- Manejo de diferentes versiones del cliente de forma simultánea: el nuevo operador Top-*k* puede ejecutarse aún si cada nodo tiene una versión diferente de la aplicación por medio de la cual se conecta a la DHT. Si varios nodos involucrados en una consulta cuentan con la misma versión y el algoritmo se implementa de tal forma que permite ser paralelizado, PTop hace uso de la capacidad de procesamiento distribuido del sistema.
- Extensibilidad: extender nuevas funcionalidades sobre las implementaciones existentes.

6.4 Descripción del Framework

El *Framework* propuesto *PTop* cuenta con tres componentes principales: manejador del operador, manejador de la consulta y manejador de índices del sistema, organizados como se muestra en la figura a continuación (Figura 6).

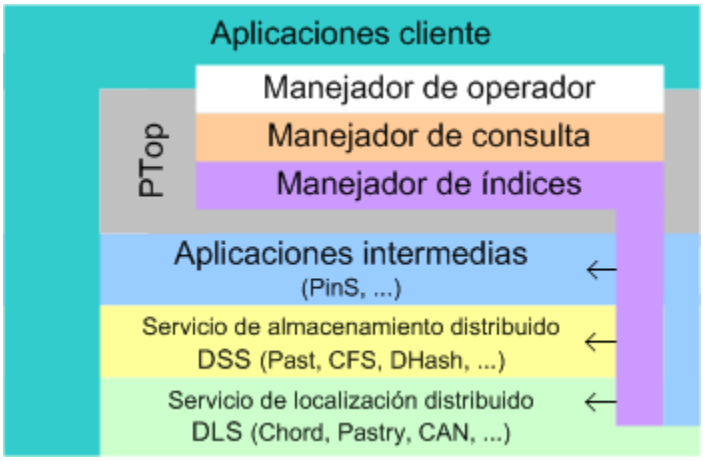

Figura 6: Esquema de PTop

Las características principales de cada componente son:

- Manejador de operador: Corresponde a la representación dada del operador Top-*k* al usuario final (aplicación o una persona). Se encarga de la gramática asociada al nuevo operador y de la representación del resultado. Este componente delega el procesamiento de la consulta sobre el manejador de consulta.

- Manejador de consulta: Es el componente encargado de implementar la funcionalidad del nuevo operador. Describe el conjunto de pasos necesarios para realizar la operación y obtener el resultado. Es el componente con el que el desarrollador de un nuevo algoritmo debe interactuar y donde se encuentra toda la lógica de la nueva operación. Todas sus operaciones se hacen con base en el manejador de índices.

- Manejador de índices: Es el API común encargado de localizar y extender la información registrada en el sistema P2P sobre DHT donde se ejecute. Provee servicios para extender los índices existentes en el sistema de acuerdo a la descripción hecha por el implementador.

Al momento de iniciar la aplicación con la implementación del nuevo operador, éste se carga en el sistema y se ejecutan todas las tareas que estén definidas como tareas de inicialización. Estas tareas pueden incluir procesamiento adicional con los índices que el manejador de índices no puede crear, dadas las descripciones en el manejador de índices.

El manejador del operador debe definir la consulta especificando los parámetros ingresados por el usuario y con los que se ejecutará la operación. Este objeto se le entregará al manejador de consulta, el cual tendrá la implementación del algoritmo para su ejecución.

El algoritmo implementado puede estar formado por fases, las cuales se ejecutarán en el orden descrito al ser cargadas en el operador. Cada una de estas fases a su vez puede contener un conjunto de tareas individuales que pueden ser ejecutadas secuencialmente o en paralelo según la especificación del desarrollador.

Note que al ser el manejador del operador quién crea el objeto de consulta, es éste quien establece la precedencia de cada operador dentro de una consulta ingresada por el usuario. De esta forma, el manejador de consulta puede determinar el orden en que se debe ejecutar la consulta.

6.4.1 Manejador del operador

La consulta típica en este contexto está compuesta por selecciones sobre atributos. Para evaluar consultas Top-*k* se incorpora el uso de criterios definidos por un usuario por medio de una función de score.

La función de score asociada a una consulta Top-*k* puede partirse en dos: la función de agregación y la función de optimización. La primera define el criterio de orden con el cual se van a seleccionar los resultados (e.g., *max*, *min*). La segunda establece una

función con respecto a los atributos definidos en el sistema (e.g., salario / edad) sobre la cual se aplica la función de agregación.

El manejador del operador le permite al usuario interactuar con el nuevo operador y es responsable de integrar el nuevo operador dentro de la gramática soportada por el sistema. Sin embargo, el análisis gramatical queda fuera del alcance de esta propuesta. Cada aplicación cliente que lo quiera implementar es responsable de su integración.

En este documento se asume que el nuevo operador se llama *TopK* y se define a continuación.

```
topK         := topOp [restriccion]
topOp        := "TopK" NUMERO"(" func ")"
func         := agregacion "(" optimizacion ")"
agregacion   := max | min
optimizacion := atributo (oper atributo)*
restriccion  := ("&" (atributo comparador NUMERO))+
atributo     := (LETRA | DIGITO)+
oper         := "+" | "-" | "/" | "*"
comparador   := "<" | "<=" | ">" | ">=" | "<>" | "="
NUMERO       := DIGITO+
DIGITO       := [0-9]
LETRA        := [a-zA-Z_]
```

Se debe tener en cuenta que la definición anterior es ilustrativa y puede estar incompleta. Como se explicó anteriormente, no es el objetivo ni está dentro del alcance de este documento definir la operación ni su conjunto de posibles variaciones.

Dada la descripción anterior, la consulta - encontrar las 50 personas con mejor ingreso para su edad - utilizando el nuevo operador sería:

Ej: `TopK 50 (max(salario / edad))`

El usuario podría agregar la restricción - personas entre 25 y 35 años -:

Ej: `TopK 50 (max(salario / edad)) & edad >= 25 & edad <= 35`

El resultado retornado por PTop es el conjunto de objetos y su puntaje en el orden requerido (descendente si la función de agregación es *max*, ascendente si es *min*). Los detalles de presentación (e.g., {<obj, score>}) deben ser ajustados para cada implementación.

Un factor que puede ser tenido en cuenta con el fin de evitar procesamiento de consultas para las cuales no se puede obtener respuesta es contar con índices sobre atributos de los objetos indexados en el sistema. Por otro lado también aportaría conocer el tipo de dato de cada atributo: no tendría sentido realizar una consulta que incluya como atributo de interés una variable con valores no cuantitativos si no se especifica una función que le dé un valor.

6.4.2 Manejador de consulta

De acuerdo a las descripciones hechas sobre los algoritmos Top-*k* existentes, se puede notar que todos ellos funcionan en un conjunto de pasos bien definidos. Con base en esto, el manejador de consulta permite implementar fases y tareas asociadas al nuevo operador.

La principal idea de esta capa es que una vez implementado un algoritmo, éste no tenga que ser ajustado para migrarse a otro sistema de datos, ya que las tareas internas se deben describir con base al manejador de índices.

A continuación se presenta el modelo de clases del manejador de consulta.

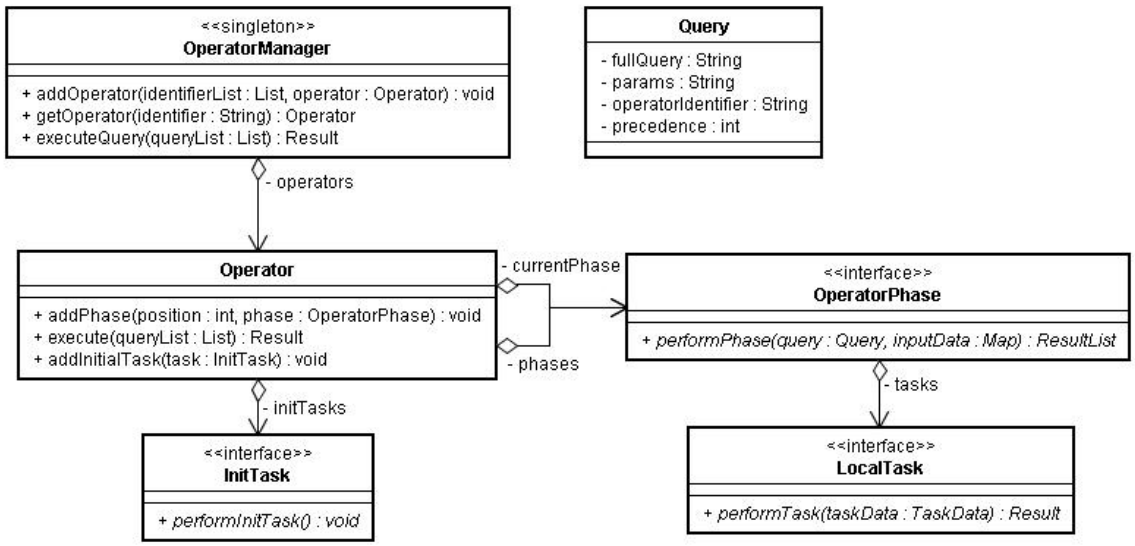

Figura 7: Modelo de clases Manejador de Consulta

El `OperatorManager` es el encargado de incorporar la implementación del nuevo operador `Operator` al sistema. Al momento de realizar una consulta, éste debe recibir una lista de operaciones `Query` las cuales ejecuta según la precedencia establecida en cada una de ellas. Estas a su vez contienen los parámetros ingresados por el usuario que serán delegados al operador para su ejecución.

Un operador `Operator` está compuesto de fases `OperatorPhase` y estas pueden o no estar compuestas de tareas individuales `LocalTask`. Ésta partición le permite al implementador describir los pasos en los que se ejecuta un algoritmo y a su vez determinar que tareas de estas son paralelizables.

Por otro lado un operador puede implementar un conjunto de tareas `InitTask` que se ejecutan al momento de iniciar la aplicación. Estas tareas dan flexibilidad al implementador para definir información que las solas descripciones del manejador de índices no pueden determinar (e.g., crear un histograma).

6.4.3 Manejador de índices

Como se explicó anteriormente, la tarea de este componente es conocer los objetos e índices con los que cuenta el sistema actualmente, permitir consultar sobre ellos y extenderlos para proveer la información que requiere el operador implementado para realizar la correcta evaluación del algoritmo. Esta capa funciona entonces como una máquina virtual sobre el sistema donde se quiera ejecutar el nuevo operador.

Para ejecutar su función, este componente requiere conocer los índices con los que cuenta actualmente el sistema, con el fin de acceder a ellos y usarlos como base de los nuevos índices. Estos nuevos índices son generados dinámicamente y son requeridos por el manejador de consultas para la ejecución del nuevo operador.

La primera tarea que realiza este componente es -determinada la información faltante- generar los índices sin afectar las operaciones ya existentes en el sistema. Estos índices pueden ser generados *ad-hoc* o como objetos que se almacenan en el sistema, tal como se explica a continuación:

- Generar los índices completando la información solo cuando se ejecute la consulta y descartarla cuando se obtenga el resultado: es una aproximación no invasiva en el sistema, entendiendo por esto que no deja rastros sobre las operaciones realizadas. La principal desventaja es que requiere generar los índices cada vez que se ejecute la operación, lo cual puede tomar bastante tiempo y resultar tedioso para el usuario.

- Generar los índices completando la información desde que se inicializa el nodo que proveerá la consulta y almacenarlos en el sistema: tal vez es la aproximación más completa debido a que se debe suponer que varios usuarios realizaran consultas del mismo tipo sobre el sistema una vez se implemente un nuevo operador. Sin embargo, esta aproximación impone resolver problemas adicionales como particionar los nuevos índices adecuadamente, mantenerlos actualizados, evitar que dos o más nodos generen la misma porción de un índice, entre otras. Más adelante se tratarán de algunas estrategias para resolver estos problemas.

Para describir los índices, el implementador cuenta con un conjunto básico de tipos de datos:

Descriptor	Descripción
`id`	Describe la llave de un objeto almacenado en el sistema
`info`	Describe un campo que contiene información alfanumérica
`score`	Describe un campo que contiene el puntaje que se debe usar para la evaluación de un algoritmo

`obj`	Describe un objeto almacenado en el sistema
`atrib`	Describe el nombre de un atributo
`rel(reader, …)`	Describe una relación de información que puede ser interpretada por medio de una expresión regular (`reader`).
`list(reader, …)`	Describe una lista de información que puede ser interpretada por medio de una expresión regular (`reader`).

Una vez se hayan descrito los índices, el componente estará listo para proveer los datos requeridos al manejador de consulta por medio de las operaciones expuestas a continuación:

- **`get(key): Object:`** retorna el objeto almacenado en el sistema con la llave de objeto dada. Si no existe, se retorna un objeto vacío.
- **`getAttributeTopScores(attribute, number, offset, sortOrder): List<IResultObject>`**: retorna una lista de `number` objetos de acuerdo a la descripción hecha para ellos con valores más altos/bajos (`sortOrder` es *asc* ó *desc* respectivamente) con respecto al atributo solicitado, iniciando en `offset`, sobre los objetos almacenados en el sistema.
- **`getObjectCountOnRange(attribute, lowerBound, upperBound): long`**: retorna el número de objetos almacenados en el sistema cuyo atributo está entre los límites superior e inferior `(lowerBound, upperBound]`.
- **`getIndex(name, attribute, lowerBound, upperBound): IndexData`**: retorna el fragmento de un índice generado dinámicamente para el atributo descrito que contiene los objetos con valores entre los valores establecidos.
- **`setIndex(name, attribute, IndexData): void:`** agrega un índice denominado `name` al sistema para el atributo `attribute` descrito con la información de `IndexData`.
- **`put(key, Object): void`**: agrega el objeto al sistema bajo la llave descrita.

Considerando que los volúmenes de información almacenados en el sistema son grandes, y queriendo tomar provecho de la capacidad de procesamiento distribuida del sistema se crean dos índices adicionales, el primero describe el estado de los índices que deben ser creados según la descripción realizada de las estructuras de datos requeridas para soportar el nuevo algoritmo Top-*k*; los segundos corresponden a la información generada por el sistema de acuerdo a las descripciones realizadas por el usuario.

El primer tipo de índices se puede representar de la siguiente forma:

Nombre del índice	<nombre_indice>_atributo
Descripción	Corresponde a un índice de soporte para conocer el número de segmentos que se crearon o se van a crear con respecto a un atributo.

Campos	Grupos	Número de grupos/fragmentos definidos para el índice
	Orden	Índica si el índice se debe crear en orden ascendente o descendente

El segundo tipo de índices esta descrito a continuación:

Nombre del índice	<nombre_indice>_<parámetros extra>	
Descripción	Corresponde a un nuevo índice descrito para soportar el algoritmo implementado. Contiene la información descrita por el índice solicitado según la definición realizada.	
Campos	Atributo	Nombre del atributo sobre el cual se generó el índice
	Orden	Ascendente ó Descendente
	Estado	En creación, en actualización, ó listo
	Timestamp	Marca de tiempo en la cual se completó la última actualización
	Nodo	Identificador del nodo que esta generando este fragmento del índice
	Datos	Lista de datos que contiene este fragmento del índice

6.5 Procesamiento distribuido

Una característica común en todos los algoritmos presentados para Top-*k* es que consultan los datos remotamente, pero todo el procesamiento se realiza de forma local. Sin embargo, los autores de los algoritmos presentados suelen dejar como tema a estudiar la posibilidad de procesar el resultado en más de un nodo, tomando ventaja de la capacidad de procesamiento disponible en el sistema.

En términos del *Framework* propuesto, esto implica que otros nodos puedan procesar tareas remotamente durante la ejecución de una fase, lo cual sería equivalente a extender las funcionalidades provistas por la DHT, ya que ésta solo se encarga de almacenar y recuperar objetos en el sistema.

Sistemas como PinS extienden la funcionalidad del sistema DHT para realizar acciones específicas en otros nodos. En sistemas como estos, el enfoque propuesto se puede extender, permitiendo paralelizar la ejecución de una tarea (e.g. evaluando la consulta sobre fragmentos de los datos), consultando un índice (e.g. aplicando un filtro sobre la información sobre la cual se evalúa la consulta), o preprocesando la información (e.g.

generando los BloomFilters de KLEE si no están generados) antes de *enviarla* al nodo iniciador.

Para implementar esta funcionalidad el manejador de consultas debe ser extendido, permitiéndole al implementador definir qué tareas o fases son paralelizables y bajo qué criterios. A continuación se muestra el modelo propuesto que permite enviar mensajes a otros nodos y ejecutar acciones sobre ellos.

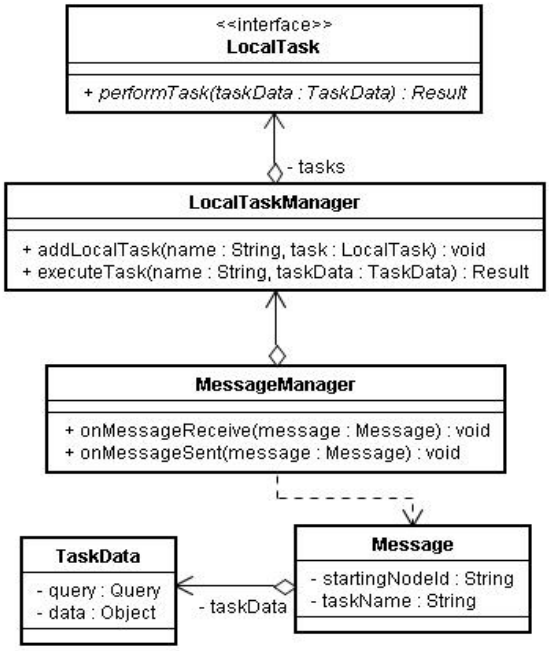

Figura 8: Modelo de clases para procesamiento distribuido en PTop

Al recibirse una solicitud de procesamiento remoto, este mensaje `Message` es delegado al `MessageManager` el cual determina por medio del `LocalTaskManager` si puede evaluar la tarea solicitada y la delega sobre su implementación `LocalTask`. Éste último se ejecuta y por medio del `MessageManager` envía el resultado de su procesamiento al nodo que lo solicitó.

Una característica importante para la implementación de este esquema es que se debe adicionar un protocolo que permita conocer si los nodos involucrados tienen la capacidad de ejecutar tareas distribuidas. En caso de no ser soportado el operador, debe procesarse secuencialmente en el nodo iniciador.

El procesamiento distribuido es responsabilidad del implementador del operador en todo nivel, pues el sistema no puede determinar que fases son paralelizables por defecto. Las tareas por su lado, de acuerdo al orden con que se establezcan se ejecutarán en paralelo (e.g., dos o más tareas definidas en la posición tres para una fase).

6.6 Integración del operador Top-*k*

Es de esperarse que el nuevo operador se pueda integrar con un conjunto de restricciones sobre los datos en el sistema. De hecho, limitar el rango de valores que puede tomar un atributo involucrado en la función de optimización implica interactuar con un operador de filtro de datos.

En una DHT sin extensiones, la única solución a este problema es realizar un pre-procesamiento de los datos sobre los cuales se va a aplicar el nuevo operador, pues no se pueden delegar los filtros a los nodos involucrados. De esta forma, en el caso de una consulta Top-*k*, el otro operador con el que se integra tendría que ejecutarse primero y darle su resultado como parámetro de entrada al operador `Topk`.

Para esto, las operaciones descritas en el manejador de índices se deben extender, recibiendo el conjunto de datos sobre el cual se va a aplicar el operador Top-*k*. De esta forma, el operador se ejecuta solamente sobre el conjunto de datos -ya filtrado- relevante para el usuario.

En PTop éste requerimiento es tenido en cuenta al permitirle al manejador del operador definir el orden en el cual se debe ejecutar cada parte de la consulta (la precedencia de los operadores) al momento de solicitar la ejecución del operador al manejador de consulta.

En un sistema DHT extendido donde los nodos no solo almacenan y permiten consultar su información sino que también pueden ejecutar tareas individuales según solicitud, se puede delegar la evaluación de los operadores de la consulta a los nodos involucrados para que sean ejecutados según su precedencia.

6.7 Componentes de soporte

Otros componentes que podrían integrarse al *Framework* propuesto como servicios de soporte con el fin de apoyar los procesos dentro de los operadores que se implementen son:

- Manejador de Estadísticas: sería interesante un controlador que proveyera información sobre el número de objetos que tienen un valor para un atributo (a_1:x_1, a_2:x_2, …, a_n:x_n donde a_i es el atributo y x_i es el número de objetos con ese valor). Esto facilitaría construir índices como los propuestos por KLEE, donde se almacenan histogramas con información para cada celda.

- Manejador de Cache: en el caso que el operador tenga que procesar grandes cantidades de información en un nodo específico sería importante tener un espacio donde almacenar los resultados temporales y obtenerlos eficientemente.

- Controlador de Localización: en el caso de hacer una consulta que involucre relacionar varios sistemas de datos establece conversiones entre las diferentes localizaciones/culturas con las cuales se pudo haber almacenado la información en el sistema, estableciendo un marco común para evaluar un resultado.

También sería bueno contar con un esquema de descomposición de consultas por factores algebraicos sobre la consulta solicitada. Una primera aproximación para esto son las reglas descritas a continuación (requiere un estudio mayor al respecto) que buscan maximizar los atributos que más aportan a la función y minimizar el valor que toman los atributos que disminuyen el resultado su evaluación:

***f*(x)**	**Condición**	**Algoritmo**
K + x	x > 0	max(*x*)
	x < 0	min(x)
K – x	x > 0	min(x)
	x < 0	max(x)
K * x	x > 1	max(x)
	-1 <= x <= 1	ver el caso de división (K / x)
	x < -1	min(x)
K / x	x > 1	min(x)
	-1 <= x <= 1	ver el caso de multiplicación (K * x)
	x < -1	max(x)

7. Evaluación de PTop

A través de éste capítulo se presenta un ejemplo de implementación de DHTop usando PTop sobre PinS con el objetivo de evaluar su completitud en términos del API propuesto, comportamiento del *Framework* y funcionamiento en ambientes P2P DHT donde solo algunos de los nodos/clientes cuentan con el nuevo operador implementado.

Para esto, en la sección 7.1 se describen las estructuras de datos/índices con los que cuenta PinS actualmente. En la sección 7.2 se describen las estructuras de datos adicionales que se requieren para implementar DHTop. En la sección 7.3 se describe el proceso que sigue PTop para la generación de las estructuras de datos adicionales requeridas basado en las estructuras de datos con las que cuenta el sistema actualmente. En la sección 7.4 se presenta una posible implementación del algoritmo DHTop utilizando PTop. Finalmente en la sección 7.5 se presentan resultados de la evaluación practica de la implementación de este algoritmo.

7.1 Estructuras de datos en PinS

A continuación se presenta un ejemplo que describe los índices disponibles en PinS con el fin de implementar el algoritmo propuesto por DHTop sobre él.

Asumiendo que la gramática para describir los índices de un sistema es de la forma `indexName: key: values`, los índices disponibles en PinS se describirían de la siguiente manera[1]:

idx1: id: obj
Asociación llave-objeto

idx2: atrib "=" score: list("(1,)+", id)
La llave se describe como el nombre de un atributo igualado a un valor **atrib "=" score** y como resultado se obtiene una lista de objetos **list(id)** para los cuales el atributo descrito toma el valor dado.

idx3: atrib: list("(1,)+", score)
La llave se describe como el nombre del atributo **atrib** y como resultado se obtiene una lista que contiene el conjunto de valores que toma ese atributo en los diferentes objetos almacenados en el sistema **list(score)**.

7.2 Estructuras de datos requeridas por DHTop

[1] Para facilidad de lectura se eliminan los caracteres de escape en las expresiones regulares descritas

Para implementar DHTop se requiere también describir un índice de la siguiente forma:

idx4: "sd" atrib info: list("(1,)+", "rel(2,3)", score, "list((4,)+)", id)
La llave se describe como una cadena compuesta por un prefijo "sd" seguida del nombre del atributo y un valor variable que debe ser dado al momento de crear el índice **"sd" atrib info.** Como resultado se obtiene una lista de relaciones donde su primer elemento es el puntaje o valor que toma el atributo y el segundo la lista de objetos almacenados en el sistema que toman ese valor **list(score, list(id)).**

7.3 Creación de estructuras de datos adicionales por PTop

Para crear `idx4`, primero se debe determinar el punto al que se quiere llegar, para lo cual no es relevante usar la llave sino solo el valor que se almacenará; segundo, a partir de los índices disponibles, determinar cómo se puede llegar a este valor. Este procedimiento se describe en el macro algoritmo a continuación:

idx4: list("(1,)+", "rel(2,3)", score, "list((4,)+)", id)

Función	Detalles
list(rel(score,list(id,)),)	simplifique la lista
rel(score,list(id,))	simplifique la relación
score,list(id,)	relación buscada
score,id	relación básica de idx4

idx3: atrib: list("(1,)+", score)

Función	Detalles
atrib: list(score,)	simplifique la lista
atrib: score	relación básica de idx3

idx2: atrib "=" score: list("(1,)+", id)

Función	Detalles
atrib "=" score: list(id,)	relación básica de idx2

Usando la relación básica de idx3 como llave de idx2 se obtiene la relación buscada y se puede pasar a generar el índice buscado.

Con esta información a partir de `idx2` e `idx3` se generaría `idx4` con el siguiente macro algoritmo[2]:

```
createIndex(lowerBound, upperBound, sectionId) {
   List idx4values = new List();
```

[2] Las operaciones `get()` y `put()` usadas en el macro algoritmo descrito corresponden a las funciones disponibles en el sistema de datos en el cual se está implementando el nuevo operador.

```
   List values = getValuesBetween(idx3.values,
      lowerBound, upperBound);
   foreach (score in values) {
      key = idx3.key + "=" + score;
      idx4values.add(new Relation(score, get(key));
   }
   newKey = "sd" + idx3.key + sectionId;
   put(newKey, idx4values);
}
```

Note que en este caso `idx4` se debe crear de acuerdo a la distribución que siguen los valores para cada uno de sus atributos. Esta fuera del alcance de este documento explicar cómo determinar la función de distribución para un conjunto de valores. Para efectos de este ejemplo se asumirá que siguen una distribución normal.

En este punto el manejador de índices tiene la información completa y lista para proveerla al manejador de consulta. Los pasos descritos a continuación corresponden a la implementación del algoritmo DHTop con las clases descritas en el manejador de consulta.

7.4 Implementación de DHTop usando PTop

Una vez se han definido los índices requeridos el paso a seguir es implementar las fases que sigue el algoritmo. En el caso de DHTop se han identificado las siguientes fases para ser implementadas:

1. Solicitar a cada uno de los índices para los atributos relacionados en la función de optimización su primer subdominio.

```
performPhase(query, data) {
   Map<String, List> subdomains;
   foreach (att in query.getParams) {
      subdomains.put(att,
         getAttributeTopScores(att, 1, 0, SortOrder.ASC));
   }
   return subdomains;
}
```

2. Establecer el umbral con la fila de elementos recibidos de cada índice y evaluar el puntaje de los objetos para los que se tiene información completa. Si se agotan los elementos para alguno de los índices recibidos, solicite el siguiente subdominio para cada uno de los índices de los atributos relacionados. Deténgase al encontrar *k* elementos con puntaje superior al umbral establecido para la fila en evaluación.

```
long currentRow = 0;
Map<objectId, attribValues> objectScores;
performPhase(query, data) {
   int lowerCommonDataCount =
      getLowerCommonDataCount(data);
   for(int i = 0; i < lowerCommonDataCount; i++) {
```

```
            long threshold = 0;
            foreach (desc in data.getDescriptors) {
               ResultObject res = desc.getResults().get(i);
               QueryAtribs qAt = objectScores.get(res.getId());
               if (qAt == null) {
                  qAt = new QueryAtribs(query.getParams());
                  objectScores.put(res.getId(), qAt);
               }
               qAt.add(desc.getName(), res.getScore());
               threshold += res.getScore();
            }
            int fullEvalCount = 0;
            foreach (qAt in objectScores) {
               long fullScore = qAt.getFullScore();
               if (fullScore != -1 &&
                  fullScore >= threshold) {
                  fullEvalCount++;
                  if (fullEvalCount == query.getK())
                     goToNextStep(objectScores);s
               } else if (fullScore != -1 &&
                  fullScore < threshold) {
                  objectScores.drop(qAt.getObjectId());
               }
            }
         }
         retrieveNextSubdomainAndContinue();
      }
```

3. Solicite completar la información para cada uno de los elementos vistos no completos. Evalúe la función de optimización con sus atributos y ordene los resultados.

```
foreach (qAt in objectScores) {
   obj = get(qAt.getId());
   qAt.setParams(obj);
}
return sortResults(objectScores);
```

El manejador del operador debe integrar ahora el nuevo operador dentro de su gramática delegando el procesamiento sobre el manejador de consulta que incluye la implementación del operador Top-*k* con el algoritmo DHTop como se acaba de describir.

7.5 Evaluación práctica

A continuación se presentan los resultados de implementar la solución descrita en la sección anterior. Ésta implementación se hace con el objetivo de probar el *Framework* propuesto PTop en un ambiente real, haciendo énfasis en mantener desacoplado el nuevo operador del ambiente donde se implementa de tal forma que las estructuras existentes, los datos, y las operaciones que expone no se vean afectados.

Por otro lado, es de gran interés probar que PTop es capaz de crear nuevas estructuras que sirvan de soporte para realizar las tareas descritas en el nuevo operador, y en caso que no las pueda determinar con las descripciones dadas, que se cuente con todas las herramientas para poder implementarlas.

Muy similar a como se describe en la bibliografía sobre el proceso de creación de un Framework (Fayad, et al., 1999), diseñar PTop tomo cerca de 8 meses y su implementación menos de un mes. Cabe anotar que esta es una primera evaluación de PTop que muy seguramente requerirá pequeños ajustes a medida que se vayan implementando más algoritmos Top-*k* y portándolos a diferentes sistemas.

La implementación de DHTop utilizando PTop requirió 5 clases que suman cerca a 170 líneas de código. Sin embargo adaptar PinS para utilizar PTop requirió, 7 clases, de las cuales 3 interactúan directamente con PinS proveyéndole un puente que le permite interactuar con las estructuras de datos existentes en el sistema y crear otras nuevas. Adicional a estas clases, hay que agregar las que se requieran modificar para soportar el nuevo operador en la interfaz de usuario.

En caso que se quisiera migrar la implementación realizada de DHTop utilizando PTop a otro sistema diferente de PinS, se deberían de cambiar solo las 3 clases que interactúan directamente con el sistema de base, de tal forma que PTop pueda acceder a la información almacenada en el sistema y agregar nueva información en el.

La prueba se ejecutó cargando 3 atributos con valores numéricos distribuidos uniformemente con selectividad de 2.5% para 10.000 elementos en 10 nodos simulados por medio de máquinas virtuales. La información fue cargada en el sistema antes de agregar el nodo que utiliza PTop con DHTop implementado. El tiempo que tomó PTop para crear y agregar las estructuras requeridas una vez se conectó al sistema fue de 16 minutos 31 segundos.

El tiempo promedio que tomó obtener el resultado de una consulta Top-*k* (con k=5) que incluye a los 3 atributos fue de 6,83 segundos. Aumentando el número de nodos a 100, manteniendo las demás variables iguales y realizando la misma consulta anterior el tiempo promedio que tomo el sistema en obtener el resultado fue de 8,76 segundos.

Comparando estos resultados con los presentados por la evaluación de DHTop (Akbarinia, et al., 2006), el tiempo que tomo llevar a cabo la consulta utilizando el algoritmo implementado y PTop fue 2 veces mayor. Esta diferencia en principio tan radical puede surgir de varios factores: los autores de DHTop utilizan una implementación plana de Chord (Stoica, et al., 2001), donde las estructuras de datos se ajustan desde el primer momento para realizar este tipo de consultas, en el caso de prueba presentado, los índices para soportar la consulta Top-*k* se generaron sobre datos existentes en el sistema, asumiendo siempre una distribución normal.

8. Conclusiones y trabajo futuro

PTop es un Framework que facilita portar la implementación de un algoritmo para resolver consultas Top-*k* entre diferentes sistemas P2P sobre DHT. Para esto, PTop garantiza un desacoplamiento entre las estructuras de datos del algoritmo para resolver consultas Top-*k* y los índices existentes en el sistema.

El *Framework* propuesto PTop hace uso del nuevo operador sin actualizar la versión de la aplicación cliente en todos los nodos que hacen parte del sistema. A medida que más nodos utilicen la versión con el nuevo operador, los índices sobre la información almacenada en el sistema se generaran y estarán disponibles más rápidamente.

Adicional a lo anterior, PTop permite utilizar las estructuras de datos e índices definidos por implementaciones previas para definir nuevas estructuras e implementar nuevos algoritmos Top-*k* sin afectar las funcionalidades provistas anteriormente. También acorde al diseño propuesto, se pueden reutilizar implementaciones de tareas o fases que sean comunes a más de un algoritmo.

PTop propone una primera aproximación para aprovechar la capacidad de procesamiento del sistema y paralelizar sus tareas en varios nodos. También se propone una aproximación simple para integrarse con otros operadores y restricciones sobre la consulta, sin embargo un análisis más detallado sobre este último tema debe ser realizado.

Se trabajó en sistemas P2P sobre DHT ya que estos proveen la capacidad de almacenar grandes cantidades de información siendo escalables no solo vertical sino horizontalmente. De acuerdo al ajuste que se haga en estos sistemas pueden ser tolerantes a fallas y ofrecer alta disponibilidad de la información de manera económica. Estos factores hacen que contar con los operadores con los que se cuenta en los sistemas de bases de datos tradicionales, en clúster, y P2P con supernodos sea una característica deseable que motive su rápida adopción en aplicaciones comerciales.

Como se describió anteriormente existen soluciones que permiten indexar y consultar la información almacenada en estos sistemas, sin embargo las operaciones que proveen no permiten agregar la información como si lo hacen las consultas Top-*k*. Estas consultas son de gran interés actual y continuamente se proponen nuevos algoritmos que hacen mejor uso de los recursos del sistema y que proveen respuestas en tiempos más cortos. Prueba de esto es la gran cantidad de algoritmos existentes hasta el día de hoy. Por éste motivo, no tiene sentido hacer la implementación de un algoritmo que permita resolver estas consultas en un sistema P2P sobre DHT especifico, sino proponer un esquema, como PTop, que permita implementar futuras propuestas y llevarlos de un sistema a otro fácilmente.

Como trabajo futuro se plantea validar el *Framework* para extender su funcionalidad soportando otros operadores, agregándole otros componentes que permitan migrar un gran conjunto de operaciones entre sistemas P2P sobre DHT de forma sencilla.

Por otro lado, queda pendiente proveer un mecanismo para que el implementador con conocimiento de la distribución estadística de los datos de interés almacenados en el sistema pueda describírsela a PTop de la misma forma como lo hace con las estructuras de datos existentes, con la finalidad que PTop pueda realizar una mejor clasificación y segmentación de los datos en los índices. También sería de gran utilidad integrar un componente de seguridad a PTop ya que ésta es una característica común en los sistemas de datos más usados actualmente

Finalmente, se puede validar el *Framework* propuesto sobre un sistema en Grid, comparando sus características y ajustándolo para soportar el operador Top-*k*, o tomando provecho de las características que se proveen en estos sistemas para agregarlas como características de PTop.

9. Referencias Bibliograficas

Akbarinia Reza, Pacitti Esther and Valduriez Patrick An Efficient Mechanism for Processing Top-K Queries in DHTs [Journal] // University of Nantes, France. - 2006.

Akbarinia Reza, Pacitti Esther and Valduriez Patrick Best Position Algorithms for Top-K Queries [Journal] // University of Nantes. - pp. 3-4.

Alliance The Globus [Online] // The Globus Alliance Wiki. - 06 30, 2008. - 07 27, 2008. - http://dev.globus.org/wiki/Security/ProxyCertTypes.

Aycock Christopher What is a clustered database? [Online] // insideHPC.com. - 07 14, 2006. - 06 03, 2008. - http://insidehpc.com/2006/07/14/what-is-a-clustered-database/.

Broder Andrei and Mitzenmacher Michael Network Applications of Bloom Filters: A Survey [Journal] // Internet Mathematics Vol 1. - 2004. - pp. 486-495.

Broder Andrei and Mitzenmacher Michael Network Applications of Bloom Filters: A Survey [Journal] // Internet Mathematics Vol 1. - 2004. - pp. 486-495.

Cao Pei and Wang Zhe Efficient Top-K Query Calculation in Distributed Networks [Journal] // Princeton University, Department of Computer Science. - pp. 3-5.

Crainiceanu Adina [et al.] A Storage and Indexing Framework for P2P Systems [Journal] // Cornell University, Department of Computer Science. - 2004. - pp. 1-2.

Fagin Ronald, Lotem Amnon and Naor Moni Top-K Query Processing. Optimal aggregation algorithms for middleware [Journal]. - 2001.

Fayad Mohamed E., Schmidt Douglas C. y Johnson Ralph E. Building Application Frameworks: object oriented foundations of framework design [Libro]. - [s.l.] : Wiley, 1999.

Hua Ming [et al.] Efficiently Answering Top-K Typicality Queries on Large Databases [Journal] // VLDB. - 2007. - pp. 5-9.

Michel Sebastian Top-K Aggregation Queries in Large-Scale Distributed Systems [Book]. - Saarbrücken : [s.n.], 2007.

Michel Sebastian, Triantafillou Peter and Weikum Gerhard KLEE: A Framework for Distributed Top-K Query Algorithms [Journal] // Proceedings of the 31st VLDB Conferfence. - 2005. - pp. 3-7.

Michel Sebastian, Triantafillou Peter and Weikum Gerhard KLEE: A Framework for Distributed Top-K Query Algorithms [Journal] // Proceedings of the 31st VLDB Conferfence. - 2005. - pp. 3-7.

Ratnasamy S [et al.] A scalable content-addressable network [Conference] // Proc. of SIGCOMM. - 2001.

Ratnasamy S [et al.] A scalable content-addressable network [Conference] // Proc. of SIGCOMM. - 2001.

Rowstron A I.T and Druschel P Patry: Scalable, Descentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems [Conference] // Proc. of ACM Int. Conf. on Distributed Systems Platforms (Middleware). - 2001.

Rowstron A I.T and Druschel P Patry: Scalable, Descentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems [Conference] // Proc. of ACM Int. Conf. on Distributed Systems Platforms (Middleware). - 2001.

Stoica I [et al.] Chord: A scalable peer-to-peer lookup service for internet applications [Conference] // Proc. of SIGCOMM. - 2001.

Stoica I [et al.] Chord: A scalable peer-to-peer lookup service for internet applications [Conference] // Proc. of SIGCOMM. - 2001.

Sung Alex [et al.] A Survey of Data Management in Peer-to-Peer Systems [Journal] // University of Waterloo, Web Data Management. - 2005.

Vaidya Jaideep and Clifton Chris Privacy-Preserving Top-K Queries [Journal] // Rutgers University and CIMIC; Purdue University. - pp. 2-6.

Vaidya Jaideep and Clifton Chris Privacy-Preserving Top-K Queries [Journal] // Rutgers University and CIMIC; Purdue University. - pp. 2-6.

Villamil Maria-Del-Pilar, Roncancio Claudia and Labbé Cyril PinS: Peer to Peer Interrogation and Indexing System [Journal] // IDEAS'04. - 2004. - pp. 3-8.

Yu Hailing [et al.] Efficient Processing of Distributed Top-K Queries. [Journal] // University of California, Santa Barbara. - 2005.

Zeinalipour D [et al.] The Threshold Join Algorithm for Top-K Queries in Distributed Sensor Networks [Journal] // University of California Riverside. - 2005. - pp. 4-5.

Zeinalipour Demetris Top-K Query Processing Techniques for Distributed Environments [Journal] // University of Cyprus. - 2006.

Zhao B.Y, Kubiatowicz J and Joseph A.D Tapestry: a Fault-tolerant Wide-Area Application Infrastructure [Journal] // Computer Communication Review 32. - 2002.

Zhao B.Y, Kubiatowicz J and Joseph A.D Tapestry: a Fault-tolerant Wide-Area Application Infrastructure [Journal] // Computer Communication Review 32. - 2002.

Zhao Keping [et al.] Supporting Ranked Join in Peer-to-Peer Networks [Journal] // Proceedings of the 16th International Workshop on Database and Expert Systems Applications. - 2005. - p. 3.

Zhao Keping, Zhou Shuigeng and Zhou Aoying Towards Efficient Ranked Query Processing in Peer-to-Peer Networks [Journal] // Fundan University. - 2007. - pp. 6-8.

Zhao Keping, Zhou Shuigeng and Zhou Aoying Towards Efficient Ranked Query Processing in Peer-to-Peer Networks [Journal] // Fundan University. - 2007. - pp. 6-8.