OPTIQ: Componente de optimización de consultas para sistemas P2P

OPTIQ: A query optimizer component for P2P systems

Juan Pablo Gallardo Forero[1] I.S. & María del Pilar Villamil Giraldo[2] Ph.D.
1. Estudiante de Maestría en Ingeniería, Universidad de los Andes
2. Doctor en Informatique, Institut National Polytechnique de Grenoble, Profesor asistente. Universidad de Los Andes
jp.gallardo85@uniandes.edu.co; mavillam@uniandes.edu.co

Recibido para revisión 14 de abril de 2009, aceptado 4 de junio de 2010, versión final 30 de junio de 2010

***Resumen*— Los sistemas P2P – DHT ofrecen características muy atractivas para el manejo de grandes volúmenes de datos y procesamiento. Múltiples trabajos permiten ejecutar diversos tipos de consultas en dichos sistemas, sin proponer un componente de optimización adecuado para este contexto. Este artículo detalla las estrategias de optimización y ejecución de consultas utilizadas por dichos trabajos y propone OPTIQ, un componente para la optimización de consultas sobre sistemas P2P – DHT. Existe un prototipo funcional de OPTIQ que permite demostrar que los lenguajes diseñados son suficientemente expresivos y que la solución no afecta la precisión de los resultados ni la escalabilidad.**

***Palabras Clave*—Peer to Peer, Optimizador, Consultas Distribuidas.**

***Abstract*—The P2P - DHT systems offer very attractive features for managing large volumes of data and processing. Several works propose different styles of queries in these systems, but they do not include optimization components appropriate for this context. This article describes the strategies for optimization and execution query process used by these works. Additionally, it proposes OPTIQ, a Query Optimizer component for P2P – DHT systems. A prototype of OPTIQ is operational and it allows to demonstrate that the designed languages are appropriately expressive, and it does not affect neither the scalability the solution nor the precision of the query results.**

***Keywords*— Peer to Peer, optimizer, distributed queries.**

I. INTRODUCCIÓN

En las aplicaciones para manejo de datos, la necesidad de entregar al usuario los resultados a sus consultas de forma rápida y eficiente, impulsó desde sus orígenes la implementación de componentes de optimización. Los sistemas *Peer to Peer* (P2P) no han sido ajenos a esta necesidad. Múltiples trabajos han propuesto estrategias de ejecución de consultas de igualdad (KSS [1], PinS [2], [3]), consultas de rango (PinS [2], MAAN [4], [3], [5], [6], Mercury [7], Armada [8]), consultas top-K ([9], KLEE [10]) y consultas tipo JOIN ([11], PinS [2], PIER [12], [3]). Sin embargo, cada trabajo plantea un modelo propio de optimización, altamente acoplado a la ejecución misma de la consulta, con una visión limitada del estado del sistema y con poca flexibilidad al momento de generar y seleccionar un plan de ejecución.

Para resolver esta problemática, este artículo: (1) caracteriza los diferentes trabajos relacionados con la optimización de consultas sobre sistemas P2P – DHT buscando identificar heurísticas que apoyen el proceso de selección de una estrategia de ejecución apropiada para un contexto específico. (2) propone OPTIQ, un componente configurable de optimización de consultas sobre sistemas P2P – DHT, que funciona de forma desacoplada del sistema sobre el cual se implementa, capaz de proponer y ejecutar un Plan de Ejecución óptimo basado en el conocimiento del estado del sistema. (3) define un lenguaje, usado por OPTIQ, para expresar reglas de generación de planes de ejecución y estrategias de optimización en Sistemas P2P – DHT y (4) propone un componente para el chequeo semántico de

la consulta. Este componente también es configurable mediante un lenguaje definido para tal fin (Reglas de Chequeo Semántico).

Este artículo está organizado de la siguiente manera. La sección 2 define el contexto donde se enmarca la problemática. La sección 3 caracteriza los trabajos existentes relacionados con la optimización de consultas en sistemas P2P – DHT. La sección 4 propone OPTIQ y detalla sus mecanismos para optimizar consultas. La sección 5 presenta la evaluación de la propuesta y sus resultados. Por último, la sección 6, concluye y presenta el trabajo futuro.

II. CONTEXTO

Los sistemas P2P – DHT han despertado un gran interés en la comunidad académica de las bases de datos, por características inherentes a su arquitectura como escalabilidad, flexibilidad, robustez y tolerancia a fallos [13]. Estos sistemas están diseñados para, potencialmente, aumentar su capacidad de almacenamiento y procesamiento en función lineal al número de nodos que lo conforman (N), mientras que la complejidad de las búsquedas, medida en términos de la cantidad de nodos a contactar, crece apenas en función de log(N). Asimismo, sofisticados algoritmos para manejar el ingreso, salida y caída de nodos, le permiten al sistema mantener su consistencia en una red altamente dinámica y flexible.

Si bien la necesidad de optimizar consultas en sistemas P2P es una problemática novedosa, la optimización de consultas en Sistemas de Manejo de Datos se remonta a sus propios orígenes. Es así como la sección 2.1 habla del concepto clásico de optimizador, y la sección 2.2 revisa su evolución en el tiempo.

2.1. El Optimizador Clásico

Un optimizador se puede ver como un conjunto de componentes que van procesando la consulta en pasos sucesivos, desde el ingreso por parte del usuario en lenguaje no procedimental, hasta la ejecución de un plan óptimo seleccionado de forma automática basándose en un modelo de costos ([15], [16], [17], [14], [18]). Los componentes que conforman esta arquitectura tradicional son: un *Parser* que transforma la consulta que ingresa el usuario en una representación interna [15] que sea fácilmente procesable por las siguientes fases. La *Optimización Independiente del Estado* donde se transforma el árbol a un plan lógico de consulta seleccionando operadores algebraicos que remplacen las diferentes expresiones del árbol original para después aplicar transformaciones basadas en las propiedades de estos operadores. Estas transformaciones generarán un plan lógico que producirá el mejor plan físico de ejecución, sin importar las condiciones de la BD. En la *Optimización de la Consulta Dependiente del Estado* es donde se ejecutan optimizaciones que dependen del estado físico del sistema [14]. Como resultado de este proceso, el optimizador genera el plan físico de ejecución óptimo. Para generarlo, el optimizador enumera una serie de planes de ejecución alternativos y selecciona uno basándose en un modelo de estimación de costos. Por último en la *Generación del Plan de Ejecución y Ejecución de la Consulta* se transforma el plan de ejecución en código ejecutable de bajo nivel, se ejecutan las diferentes operaciones sobre la base de datos y se retornan los resultados al usuario.

2.2. Evolución de los Optimizadores

A medida que ha evolucionado el concepto de base de datos se han generado nuevas arquitecturas como las que se presentan en la Tabla 1. Es así como podemos hablar de arquitecturas de Bases de Datos Distribuidas, Paralelas, en Clúster, Heterogéneas y un nuevo concepto: Bases de Datos P2P.

Mientras que las bases de datos centralizadas mantienen el modelo clásico de optimizador, cada una de las nuevas arquitecturas ha agregado ciertos elementos a algunos de sus componentes con el fin de soportar las particularidades y desafíos de cada contexto. Por ejemplo, las Bases de Datos Distribuidas para enfrentar la distribución de los datos, incluyen la selección de la partición de las tablas a utilizar en el componente de Optimización Independiente del Estado; asimismo encapsulan la lógica de comunicación en operadores especiales (SEND, RECIEVE) utilizados para generar el plan físico de ejecución el cual será seleccionado con un modelo que incluya los costos de comunicación y será ejecutado de forma distribuida mediante anotaciones a cada operador del plan. Los mecanismos especiales para optimizar la comunicación en ejecución se pueden ver en la Tabla 1. Por otra parte en las bases de datos paralelas se incluye una nueva lógica en los operadores físicos para dividir el procesamiento entre los diferentes procesadores (SPLIT) y para fusionar los resultados (MERGE) según se requiera en el plan de ejecución planteado [19]. En las bases de datos en clúster se incluyen sofisticados algoritmos para el manejo de la concurrencia sobre recursos compartidos en el componente de ejecución [20], mientras que en las bases de datos heterogéneas la necesidad de integrar múltiples bases de datos implica tener en cuenta en la optimización de la consulta las particularidades en cuanto a esquemas, funcionalidades y costos de cada BD integrada [14]. Para logarlo se tiene una capa Mediador, que mantiene un esquema global, hace el *parsing*, determina cual BD está en capacidad de ejecutar cada parte de la consulta (mediante reglas de enumeración) y en qué capa será integrada cada parte de la respuesta. El mejor plan es seleccionado mediante un modelo de costos que tiene en cuenta las particularidades de cada BD. La ejecución en cada BD es llevada a cabo mediante la capa *wraper* que traduce las peticiones y respuestas.

Tabla 1. Componentes del optimizador en las diferentes arquitecturas de Sistemas de Manejo de Datos. En las Centralizadas se presenta el modelo clásico, mientras que en las demás se detallan los aportes de cada arquitectura.

	Centralizadas	Distribuidas	Paralelas	Clúster	Heterogéneas
Parser	Representación interna.	-	-	-	Ejecutado en el Mediador.
Optimización Independiente del Estado	Plan de consulta en álgebra relacional.	Selección Particiones.	-	-	-
Optimización Dependiente del Estado	Enumeración de planes físicos de ejecución. Selección basada en costo.	Operadores lógica de comunicación. Distribución de los operadores del plan. Modelo de costos con costo de comunicación	Operadores SPLIT y MERGE.	-	Definir en qué BD ejecutar cada sub-consulta mediante Reglas Enumeración. Definir cuál capa ejecutará cada parte de la consulta. Costos calculados para cada BD integrada.
Generación de Código y Ejecución de la Consulta	Transformación código de bajo nivel y ejecución.	Implementación de los nuevos operadores. Optimización comunicaciones: datos en bloque, multicast, semijoins, bloomfilters	Implementación de los nuevos operadores.	Algorítmica para manejo concurrencia sobre recursos compartidos.	Traducción de las operaciones en el wraper al API de la BD. Ejecución sub - consultas en cada BD integrada. Traducción resultados wraper al esquema mediador.

III. TRABAJOS RELACIONADOS CON LA OPTIMIZACIÓN EN SISTEMAS P2P

La optimización de la consultas en sistemas P2P ha sido objeto de múltiples investigaciones teniendo en cuenta la complejidad del problema, relacionada con las características de esta arquitectura. Los desarrolladores de optimizadores P2P deben lidiar con una gran cantidad de nodos interconectados (sin precedentes en sistemas de manejo de datos), con la imposibilidad de contar con catálogos centralizados y con la dificultad para mantener y disponer de información estadística. Teniendo en cuenta esta problemática se han desarrollado diversos trabajos con variados enfoques, desde la aplicación de un modelo tradicional de optimización automática hasta un enfoque donde la optimización corre por cuenta del usuario, y donde éste define, mediante un lenguaje procedimental, cómo se divide la consulta y en qué nodo se va a llevar a cabo cada una de sus operaciones. A continuación se presenta una clasificación de estos trabajos y se exploran los aportes de cada uno al concepto de optimización en Sistemas P2P.

3.1 Optimizadores

La idea de un optimizador tradicional de consultas, donde la optimización se hace de forma automática, ha sido desarrollada por [21] cuyo optimizador es capaz de generar planes de ejecución, que pretenden ser óptimos, basándose en el conocimiento distribuido de los esquemas presentes en la red y en la existencia índices, también distribuidos. La ejecución se basa en la distribución de la consulta sobre súper-nodos de la red, quienes haciendo uso de los índices son capaces de determinar cuáles partes de la consulta pueden ejecutarse localmente y cuales deben reenviarse a otro súper-nodo. Se plantea la enumeración de planes posibles de ejecución y la selección de un plan sobre otro basándose en los costos calculados a partir de las estadísticas del sistema.

En el otro lado está el enfoque dirigido a la optimización definida por el usuario, que, según [22] es una tendencia que viene cobrando fuerza en las aplicaciones de manejo intensivo de datos. En [22], el usuario define una plantilla (CQP – *Correlated Query Processing*) que determina para cada paso de la ejecución cuáles son los nodos a participar y cómo va a ser el flujo de proceso y de datos entre estos nodos. La ejecución de una consulta consiste en la ejecución concurrente de los diferentes pasos por cada nodo responsable y la sincronización mediante mensajería.

PIER, de forma similar, usa un lenguaje que define de manera explícita el plan físico de ejecución [23]. Este lenguaje, UFL (*Unnamed Flow Language*), está compuesto por operadores lógicos y físicos, los cuales son seleccionados por un usuario experto en el momento de ingresar una consulta. No obstante, PIER contempla como trabajo futuro la construcción de un optimizador tradicional.

3.2 Trabajos sobre Consultas Declarativas

Más frecuentes que los trabajos con los enfoques anteriormente mencionados, están las propuestas que implementan diversos tipos de consultas sobre sistemas P2P (consultas de igualdad :KSS [1], PinS [2], [3], de rango: PinS [2], MAAN [4], [3], [5], [6], Mercury [7], Armada [8], top-K: [9], KLEE [10], tipo JOIN: [11], PinS [2], PIER [12], [3]). Prácticamente en cada uo de estos trabajos se ha implementado un modelo propio de optimización acoplado a las estrategias de consulta.

Aún cuando en la mayoría de estos trabajos no se ha desarrollado de forma completa un componente de optimización que seleccione de forma automática una estrategia óptima de ejecución de la consulta, los autores evidencian la importancia de las condiciones del sistema en el momento de ejecutar la consulta ya que afectan el desempeño de cada implementación.

A continuación se presenta una descripción rápida de las conclusiones más importantes identificadas en la evaluación de los diferentes trabajos, que permiten identificar heurísticas en la selección de una estrategia de ejecución de consultas.

En MAAN [4], existen dos estrategias de ejecución para consultas de varios términos involucrando varios rangos: la iterativa y la dominada por un atributo.

Los estudios teóricos y prácticos permiten concluir que el modelo dominado por un atributo siempre es más eficiente que el iterativo. Asimismo la selectividad de los términos de consulta juega un papel decisivo en cuanto al orden en el que se deben ejecutar dichos términos en la consulta dominada por atributo. Incluso, si esta selectividad es muy alta (mayor a un 20%, el número de mensajes puede llegar a ser igual al 50% de la cantidad de nodos en la red) la cantidad de mensajes necesarios se acerca a la de una consulta por inundación en la red.

La implementación de las consultas Join en PIER [12] se logra mediante 4 algoritmos, análogos a sus contrapartes en la optimización tradicional: Symmetric Hash Join (SHJ), Fetch Matches Join (FMJ), Symmetric Hash Semi Join y Bloom Filter – Symmetric Hash Join.

Los resultados experimentales permiten concluir que el algoritmo que siempre consume más recursos de red es el SHJ. El FMJ se mantiene de forma constante por debajo de éste (consumiendo en promedio 20% menos), mientras que las dos optimizaciones del SMJ, dependiendo de la selectividad de los predicados sobre la tabla con menos registros (la llamaremos S) consumen menos recursos, aunque este consumo crece de forma lineal, hasta que se alcanza la selectividad del 60% y 80% respectivamente. En ese punto el FMJ es más eficiente.

PIERSearch [24], cuyo enfoque es la localización de archivos mediante palabras clave sobre una red de intercambio de archivos estilo Gnutella [25], usa una estrategia mixta de consulta por inundación para objetos comunes y uso de índices sobre una DHT para objetos con pocas réplicas.

Se demostró que la completitud de las consultas depende directamente del horizonte de las consultas por inundación (TTL) junto con el umbral para indexar los objetos escasos en la red. La precisión aumenta significativamente cuando se aumenta el umbral, aunque cuando este umbral sobrepasa los 3 objetos, la precisión no crece demasiado: con 3 objetos es, en promedio, del 95%, mientras que con 10 aumenta hasta el 99%.

Mercury [7] ejecuta las consultas de rango manteniendo un anillo (hub) por cada atributo, donde las tuplas se encuentran indexadas con relación a dicho atributo. En el momento de ejecutar una consulta se debe determinar sobre cuál atributo - anillo se debe iniciar la búsqueda.

Se puede concluir que las consultas que inician por el atributo con menor selectividad obtienen resultados con una reducción del 25% – 30% en la inundación de los nodos con respecto a consultas que seleccionan el primer anillo de forma aleatoria. Asimismo, las consultas que usaron caché siempre obtuvieron mejores desempeños: tiempos de consulta entre 30% - 50% menores.

El framework propuesto por [3] basa su modelo de optimización para consultas de rango y Join en el uso de índices distribuidos. Los índices pueden ser distribuidos en toda la red o almacenados únicamente en súper-nodos llamados Range Guards (RG).

Para consultas de desigualdad, al usar los índices comunes, la complejidad del algoritmo es de O(logN + m) donde m es la cantidad de nodos que contienen el rango de búsqueda. Con el uso de RG estas consultas de rango reducen su complejidad a un valor máximo de logN + l, donde l es la cantidad de RG en el sistema. Los algoritmos de Join tienen un comportamiento similar a las consultas de rango, y de la misma forma se pueden implementar mediante el uso de los índices o mediante los RG, obteniendo complejidades de O(n) y O(l) respectivamente.

La extensión de PinS detallada en [11], **se enfoca en las consultas tipo JOIN** implementando tres estrategias: Index Based Join – IBJ (cuando existe un índice sobre la condición del Join), Nested Loop Join – NLJ y NLJ con reducción del espacio de búsqueda.

El IBJ requiere constantemente una cantidad menor de mensajes. El NLJ con reducción resultó ser más eficiente cuando la cardinalidad del Join es mucho menor que el espacio de búsqueda (cantidad de tuplas resultado de la evaluación de todas las demás condiciones diferentes al Join), aunque cuando dicha cardinalidad se acerca al tamaño del espacio (cardinalidad mayor o igual al 15% del espacio de búsqueda), el procesamiento necesario para eliminar duplicados sobrepasa el beneficio obtenido posteriormente en la evaluación del Join.

Armada [8] ofrece estrategias de consulta determinísticas cuya aplicabilidad depende de la existencia previa del índice-tupla correspondiente al atributo involucrado en la condición.

3.3. Síntesis

En la Tabla 2 se presenta una síntesis de las estrategias anteriormente mencionadas, así como de las posibles heurísticas para determinar qué algoritmo usar según las condiciones del sistema.

Teniendo en cuenta el "estado del arte" podemos concluir que en este momento no existe un optimizador propiamente dicho para sistemas P2P. Si bien [21] se acerca a lo que pretendemos, su alto acoplamiento a la capa de red P2P con súper-nodos impide su implementación sobre otros sistemas de consultas P2P – DHT más eficientes, escalables y robustos. Por otra parte este trabajo no aborda temas relevantes, tales como las estrategias de generación de planes, mecanismos de recopilación de estadísticas y la falta de resultados experimentales no permite comprobar la eficiencia de la propuesta.

Tabla 2. Resumen de estrategias de optimización implementadas por los trabajos sobre consultas declarativas en P2P.

	Estilo de Consulta	Estrategias de Consulta	Condiciones del Estado	Heurísticas de Selección
MAAN [4]	Rangos	• Iterativa. • Dominada por Atributo.	• Término más selectivo	[Selectividad del término menos selectivo < 20%] => Usar estrategia dominada por atributo [else] => inundación?

PIER [12], [23]	Join	• Symmetric Hash Join.
		  ○ Semijoin.
		  ○ Bloom Filter.
		• Fetch Matches.
			• PK de la tabla.
			• Tamaño de la relación más pequeña.
			• Selectividad.
				{R join S / |S| < |R|}
				[Selectividad(S) > 80% ^ PK(s) = Condicion de Join] => Usar Fetch Matches Join
				[else] => Usar Symmetric Hash Semi Join

PIERSearch [24]	Palabra Clave	• Inundación. • Índices.	• Cantidad de resultados que traerá una consulta. • Presencia de índice sobre los objetos.	[|Resultados de una consulta| < 3] => Usar Índices [else] => inundación.

[3]	Rangos Join	• Simple
		• Usando Índices RangeGuards
			• Presencia de índices RG sobre los atributos.	{Q(A1,A2,…,An) / Q es una consulta con condiciones sobre los atributos A1, A2, …, An}
				FOR[i=1 - n]{
				[ExisteRG(A1)] => Usar estrategia RG.
				[else] => Usar estrategia simple.
				}

PinS [11]	Join	• Index Join • Nested Loop Simple Join • Nested Loop Simple Join con Reducción del Espacio de Búsqueda.	• Presencia de Índice sobre los atributos del Join. • Selectividad de términos individuales • Cardinalidad del Join.	[Existe un Índice sobre el atributo de la condición de Join] => Usar Index Join [Cardinalidad del Join < 15% del espacio de búsqueda] => Usar Estrategia NLJ con reducción [else] => Usar NLJ simple
Mercury [7]	Rangos	• Basada en hubs de atributos. • Basada en hubs de atributos - Caché.	• Selectividad de cada condición. • Presencia de Caché.	{Q(A1,A2,…,An) / Q es una consulta con condiciones sobre los atributos A1, A2, …, An} [Existe Caché] => Consultar con caché. [MinSelectividad(A1,A2,…,An) = Ai] => Iniciar consulta sobre el hub Ai
Armada [8]	Rangos	• Basado en Índices	• Presencia de Índices	[Existe Índice] => Usar estrategia basada en Índices [else] => inundación?

El enfoque de PIER [23] y [22] basa su modelo de optimización en la definición por parte del usuario mediante un lenguaje procedimental, de cómo se divide la consulta y en qué nodo se va a llevar a cabo cada uno de los pasos de la consulta. Sin embargo, estas aproximaciones requieren un conocimiento experto por parte del usuario en relación a dónde y cómo están localizados los datos. Dicho conocimiento no es trivial en sistemas P2P – DHT.

Por otra parte existe una gran variedad de trabajos que abordan el tema de consultas sobre sistemas P2P – DHT y que implementan su propio esquema de optimización: MAAN [4], PIER ([12] y [23]), PIERSearch [24], [3], PinS [11], Mercury [7] y Armada [8]. Sin embargo, en estos trabajos los planes de ejecución están implementados en el código mismo de la aplicación, existe un conjunto muy limitado de estrategias y la selección de una u otra por lo general la hace el implementador en el momento de la experimentación. Lo anterior implica un alto acoplamiento del optimizador con la implementación del sistema, y una escasa flexibilidad al momento de optimizar.

IV. OPTIQ: COMPONENTE DE OPTIMIZACIÓN DE CONSULTAS SOBRE SISTEMAS P2P – DHT

Éste trabajo propone OPTIQ, un componente de optimización de consultas sobre sistemas P2P – DHT que le permite al usuario ingresar consultas en un lenguaje declarativo, estilo SQL, y obtener su resultado de forma optimizada, teniendo en cuenta el contexto del sistema. A continuación se presenta una descripción detallada de OPTIQ. La sección 4.1 presenta las generalidades de OPTIQ. La sección 4.2 describe la arquitectura por capas propuesta. La sección 4.3 describe los lenguajes usados por OPTIQ para la expresión de reglas de generación de planes y para el chequeo semántico de las consultas. Por último, la sección 4.4 presenta los detalles de implementación del prototipo de OPTIQ.

4.1. Generalidades

OPTIQ es un componente de optimización desacoplado del sistema DHT subyacente. El usa los servicios provistos por cualquier sistema DHT completo. Esto es, un sistema que incluye servicios de localización (*Distributed Lookup Service*

– DLS), almacenamiento (*Distributed Storage Service* – DSS) y búsquedas complejas (*Distributed Data Service* – DDS) [2]. Ejemplos de estos sistemas son MAAN [4], PIER [23], PIERSearch [24], [3], PinS [11], Mercury [7] y Armada [8].

Las consultas que soporta son las comúnmente provistas por estos sistemas, es decir, consultas declarativas sobre metadatos, para seleccionar los objetos compartidos en el sistema. Estas consultas pueden involucrar la conjunción o disyunción de varios atributos, incluir uno o más términos de igualdad, desigualdad o Join.

Una vez recibida la consulta, OPTIQ es capaz de ejecutarla y presentar al usuario el resultado de forma eficiente, esto es, en el menor tiempo y / o con el menor uso de recursos posible. OPTIQ se basa en una serie de heurísticas para la generación de planes óptimos de ejecución, modeladas en el sistema en forma de reglas configurables por un usuario administrador. Estas reglas utilizan la estructura de la consulta, los recursos de optimización disponibles en el sistema, como estadísticas, índices o caché, para seleccionar el plan que más se ajuste a tales características del entorno.

OPTIQ es un componente de optimización configurable que permite definir nuevas reglas o modificar las existentes para generar mejores planes de ejecución.

4.2. *Arquitectura*

Como se anotó en la sección 3.3, la arquitectura tradicional de los Sistemas P2P - DHT presenta un alto acoplamiento del optimizador con la implementación misma del sistema, lo cual afecta la flexibilidad al momento de optimizar.

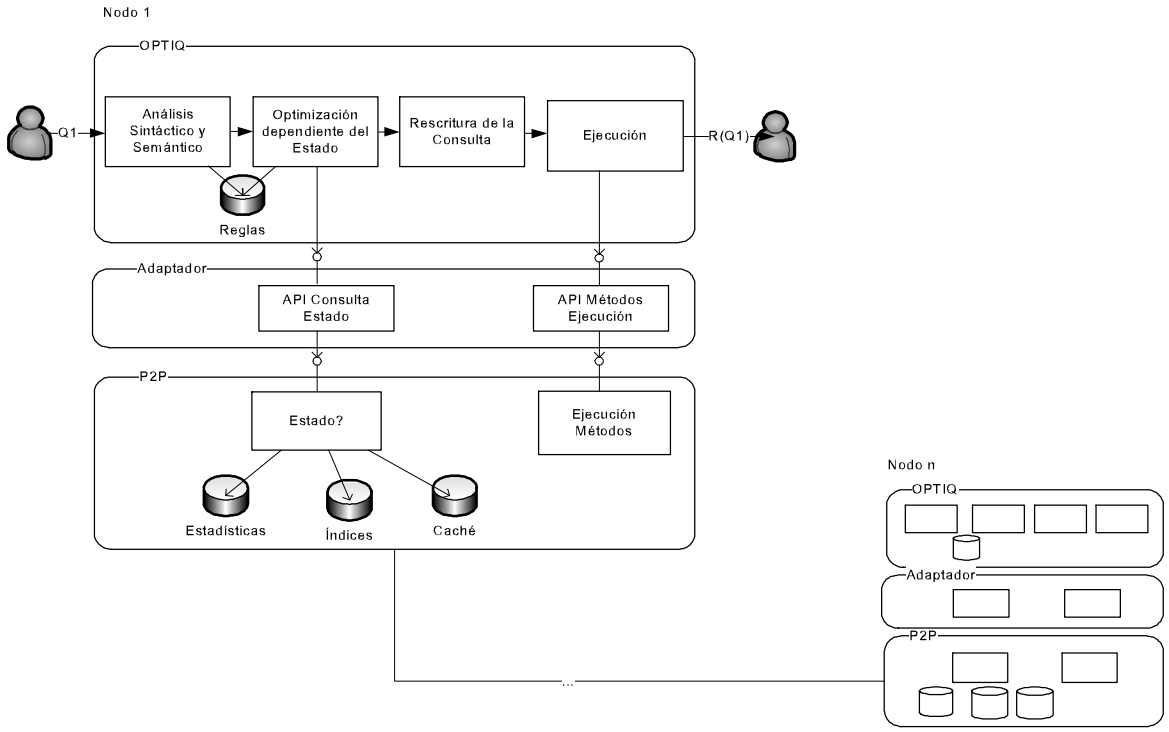


Figura 1. Arquitectura general OPTIQ.

La arquitectura de OPTIQ es la evolución de este modelo, donde el sistema se divide en tres capas (ver Figura 1), en un esquema similar al de los optimizadores en Bases de Datos Heterogéneas, con los cuales se comparte la necesidad de desacoplar la funcionalidad de optimización de la funcionalidad de integración con sistemas heredados. Las tres capas son: el Optimizador (OPTIQ), El Adaptador y El Sistema P2P – DHT. Note que OPTIQ es una capa adicional al componente de Software instalado en cada uno de los nodos de un sistema P2P – DHT.

Optimizador. La capa superior es el Optimizador, componente análogo al Mediador, encargado de llevar a cabo los procesos de optimización de la consulta de forma desacoplada, mediante transformaciones sucesivas. Los componentes de esta capa son: *Análisis Sintáctico y Semántico*, donde se aplica una transformación con el fin de generar una representación interna y se hace un chequeo semántico; la *Optimización Dependiente del Estado*, donde se genera un árbol de ejecución óptimo, según las reglas del sistema P2P - DHT, basándose en el estado del sistema y en heurísticas definidas por el usuario administrador; la *Reescritura de la Consulta*, donde se transforma el plan de ejecución al lenguaje de ejecución del optimizador; y el *componente de Ejecución*, donde se invocan las operaciones sobre el Adaptador y se reciben los resultados para presentarlos al usuario.

Adaptador. La capa intermedia es el Adaptador, el cual se encarga de uniformizar los diferentes servicios de consulta de estado (*API* Consulta de Estado) y de ejecución de operaciones (*API* Ejecución de Operadores) ofrecidos por el sistema P2P – DHT de tal forma que sean aprovechables por el Optimizador. La API de Consulta de Estado está conformada por aquellos servicios provistos por el Adaptador para conocer aspectos relevantes del sistema tales como estadísticas, tamaño de las relaciones, presencia de índices, etc. La API de Ejecución de Operadores está conformada por aquellos servicios que le permiten al sistema ejecutar operadores de un plan de ejecución, tales como *SCAN*, *NLJ_JOIN* o *INTERSECT*. En este componente se concentra el acoplamiento con las capas inferiores, evitándolo en la capa *Optimizador*.

Sistema P2P – DHT. La capa inferior corresponde al sistema P2P – DHT. Esta capa es responsable de consultar el estado y ejecutar las operaciones del Plan de Ejecución generado por el Optimizador.

4.3. Lenguajes propuestos

El componente de optimización pretende ejecutar las consultas que ingresen al sistema en el menor tiempo y con el menor uso posible de recursos. Como se vio en la tabla 2 existen diferentes estrategias que son más (o menos) eficientes según las condiciones de la consulta y del sistema. Un componente que contenga una lógica de selección de un plan de ejecución basada en dichas condiciones debería generar planes cercanos al óptimo. Con este fin, el Componente de Optimización contiene un motor de reglas de generación de planes de ejecución, el *Optimizador Dependiente del Estado*, el cual se encarga de aplicar la lógica encapsulada en dichas reglas. Asimismo la necesidad de aplicar diferentes reglas sobre diferentes sistemas implica la necesidad de hacer de éstas un componente configurable. Con el fin de poder definir y configurar estas reglas se definió un lenguaje de generación de planes de ejecución.

Las reglas de Generación de Planes de Ejecución utilizan la estructura de la consulta y los recursos de optimización disponibles en el sistema, con el fin de definir cuáles operadores "físicos" se usarán y en qué orden se ejecutarán sobre la capa DHT.

Por otra parte, la necesidad de validar en ciertos contextos la sintaxis y la semántica de las consultas ingresadas hace necesaria la definición de reglas en el componente de *Análisis Sintáctico y Semántico*. Estas reglas a su vez deben ser configurables con el fin de aplicar en cada sistema sobre el que se implemente el optimizador las validaciones particulares al mismo. De esta forma se definió un lenguaje para expresar las Reglas de Chequeo Semántico.

Las reglas de Chequeo Semántico buscan definir las relaciones existentes en el sistema P2P – DHT, así como los atributos y sus tipos de datos. En estas reglas se permite deshabilitar partes del proceso de verificación semántico.

4.3.1. Reglas de Generación de Planes de Ejecución

Las reglas de generación de planes de ejecución permiten generar un plan de ejecución óptimo según el estilo de la consulta ingresada y el estado del sistema en el momento de la optimización. De esta forma las reglas se pueden ver como una secuencia de parejas conformadas por un *condicional* y el *patrón de plan de ejecución* a generar cuando se cumple el condicional. Los *condicionales* están conformados por uno o más predicados los cuales pueden contener constantes o variables en función del estado del sistema. Un *patrón de plan de ejecución* define las operaciones que se usarán para ejecutar la consulta, el lugar donde se ejecutarán y las relaciones de producción-consumo de resultados entre dichas operaciones.

Figura 2. Estructura de lenguaje para definición de reglas de generación de planes de ejecución

```
<declaración de variables>
if(<condición 1>){
  <Regla 1 para generación Plan de
ejecución>
}[elsif(<condición 2>){
  <Regla 2 para generación Plan de
ejecución>
}[elsif(<condición 3>){
  <Regla 3 para generación Plan de
ejecución>
} …
elsif(<condición N>){
  <Regla N para generación Plan de
ejecución>
}]][else{
  <Regla E para generación Plan de
ejecución>
}]
```

(a)

```
<declaración de variables>:
<nombre variable 1> := <literal>;
[<nombre variable 2> :=
(<variable>|<literal>);
[<nombre variable 3> :=
(<variable>|<literal>);
…
<nombre variable n> :=
(<variable>|<literal>);]]
```

(b)

```
<condición i>:
<predicado 1>
[(AND|OR)<predicado 2>
[(AND|OR)<predicado 3> …
(AND|OR)<predicado N>]]
```

(c)

```
<predicado i>:
(<boolean>
|
(<variable>|<literal>) <op. lógico>
(<variable>|<literal>))
```

(d)

```
<Regla i para generación Plan de
ejecución>:
<OPERADOR_UNARIO>[(sub-regla|relación)]
| <OPERADOR_BINARIO>[(sub-regla|relación),
(sub-regla|relación)]
```

(e)

```
<sub-regla>:
<OPERADOR_UNARIO>[(sub-regla|relación)]
| <OPERADOR_BINARIO>[(sub-regla|relación),
(sub-regla|relación)]
```

(f)

En la Figura 2 (a) se muestra el estilo mediante el cual se definen las reglas de generación de planes. Estas reglas se presentan en una estructura de casos condicionales. Las Figuras 2 (e) y 2 (f) muestran la estructura del contenido de cada condicional. En éste se especifica cómo debe generarse el plan para esa condición usando OPERADORES organizados de forma jerárquica. Esta jerarquía se expresa en el lenguaje mediante el anidamiento de reglas. Este anidamiento define la manera en la cual se producirán y consumirán los resultados entre los OPERADORES. El parámetro *p* indica si se habilita el *pipe-lining* de los resultados entre el operador y su padre en la jerarquía.

Los OPERADORES, así como el resto de elementos de este lenguaje se presentan en la tabla 3. Entre estos elementos se destacan tres estilos de funciones: las funciones de consulta del estado del sistema, identificadas con el prefijo 'ST_', las funciones de revisión de la estructura de la consulta, identificadas con el prefijo 'Q_' y las de producción de OPERADORES para el plan de ejecución (escritas en mayúscula), que permiten expresar los pasos de una estrategia específica de evaluación de consultas. Para cada uno de estos grupos de funciones, se identificó un subconjunto mínimo que permite expresar todas las estrategias analizadas en la sección 3.2. Este subconjunto puede ser extendido sin alterar el comportamiento de OPTIQ.

Tabla 3. Elementos del lenguaje para expresión de la lógica de las estrategias de ejecución en OPTIQ.

Funciones ST_	Estructura	*_index_over(a1,a2)*	*_pk(a1)*	
	Estadísticas	*_join_cardinality(a1)*		
		_less_cardinality_table(a1[])		
		_selectivity(a1,a2[])	*_result_cardinality*	
		_less_selective_term(a1[])	*_cache_over(a1)*	
	Q_	*_join_term*	*_relations*	
		_join_attribute(a1)	*_having_equality_terms*	
		_terms	*_having_inequality_terms*	
		_terms_over(a1)	*_equality_terms*	
		_other_terms(a1[])	*_inequality_terms*	
		_other_join_relation(a1)	*_all_attributes*	
		_join_relation1	*_down_limit*	
		_join_relation2	*_up_limit*	
		_join_term		
	OPERADORES	*LOCAL_SCAN(a1,s, p)[r1]*	*INDEX_JOIN(r1,s, p)[r2]*	
		INDEX_SCAN(a1,s, p)[r1]	*NESTED_LOOP_JOIN(r1,s, p)[r2]*	
		INTERSECT(s, p)[r1, r2]	*PROJECT(a1[],s, p)[r1]*	
		REDUCTION(s, p)[r1]	*BROADCAST(s, p)*	
		FULL_SCAN(s, p)[r1]	*CACHE(s, p)*	
		SCAN(s, p)[r1]		
Variables		Las variables se declaran asignándoles un valor (mediante el símbolo ':='). La variable toma el tipo del valor asignado.		
Literales		Se permite el uso de literales o constantes de tipo numérico, cadena de caracteres, fecha o lógico (*true* o *false*).		
Instrucciones de Control		*if elsif else loop*		
Símbolos	Asociados a Instrucciones de Control	( ) { }		
	Para asignación de variables	:=		
	Operadores aritméticos	+ - * /		
	Operadores lógicos	< > = != <= >= *AND OR*		
	Para denotar Jerarquías	[ ]		

Aunque el Plan de Ejecución se genera de forma local en el nodo originador de la consulta, la ejecución de la misma se llevará a cabo en una variedad de nodos de forma distribuida. La forma en la cual se representa esta distribución en las reglas de generación de planes es mediante la anotación de los diferentes OPERADORES con su lugar de ejecución. Esta anotación a los OPERADORES se hace mediante un parámetro (parámetro *s* en los OPERADORES de la tabla 3) que expresa el nodo donde se debe ejecutar la operación. Este parámetro puede ser definido por el usuario administrador en el momento de la configuración del plan o puede ser delegado al optimizador para que éste decida donde se debe ejecutar.

En la figura 3 se muestra un ejemplo de cómo se puede expresar una de las heurísticas (en este caso de PinS) descrita en la tabla 2 mediante el lenguaje de *Reglas de Generación de Planes de Ejecución*. Se puede observar cómo se expresan los

condicionales de la heurística mediante los condicionales del lenguaje (*if(...), elsif(...), else(...)*) y mediante funciones tipo ST_ (eg. *ST_index_over, ST_join_cardinality*) y Q_ (eg. *Q_join_term, Q_relations*) teniendo en cuenta que las decisiones dependen del estado del sistema y del estilo de la consulta. Asimismo se observa que las estrategias de ejecución de la heurística son expresadas mediante los patrones de planes de ejecución que usan OPERADORES físicos (eg. *INDEX_JOIN, NLJ_JOIN, REDUCTION, INDEX_SCAN*) anidados, lo cual es una forma de representar un árbol de ejecución similar al de los optimizadores tradicionales. Note la presencia del parámetro *s* en los diferentes OPERADORES.

Figura 3. Representación de las heurísticas de selección de PinS (ver tabla 2) mediante el lenguaje de generación de planes de ejecución de OPTIQ. La heurística fue simplificada eliminando ciertas condiciones con el fin de hacer el ejemplo más legible.

Heurísticas de Selección	Reglas de Generación de Planes OPTIQ
[Existe un Índice sobre el atributo de la condición de *JOIN*] => Usar *Index Join*	`if(ST_index_over (Q_join_term)) {` `  INDEX_JOIN(Q_join_term, s=local)[]`

[Cardinalidad del Join < 15% del espacio de búsqueda] => Usar Estrategia NLJ con reducción	`}elsif (ST_join_cardinality (Q_join_term, Q_relations) < 15%*ST_selectivity (Q_other_terms (Q_join_term))) {`
	`  NLJ_JOIN(Q_join_term, s=local)[`
	`  REDUCTION(s=local)[`
	`  INDEX_SCAN(INEQ_JOIN_INDX, Q_inequality_terms, s)[`
	`  INDEX_SCAN(EQU_JOIN_INDX, Q_equality_terms,s)`
	`  ]]]]`

[*else*] => Usar NLJ simple	`} else {`
	`  NLJ_JOIN(Q_join_term, s=local)[`
	`  INDEX_SCAN(INEQ_JOIN_INDX, Q_inequality_terms, s)[`
	`  INDEX_SCAN(EQU_JOIN_INDX, Q_equality_terms, s)`
	`  ]]]`
	`}`

4.3.2. Reglas de Chequeo Semántico

Las reglas de chequeo semántico le permiten validar al Componente de Análisis Sintáctico y Semántico que las consultas ingresadas estén acordes a la estructura de los datos almacenada en el Sistema P2P - DHT. De esta forma existen tres elementos a validar sobre las consultas: Que las relaciones involucradas existan en el sistema, que los atributos utilizados pertenezcan a la relación a la que se asocia en la consulta y que la forma en que se utilizan dichos atributos sea consistente con el tipo de dato que almacena.

De esta forma las reglas de chequeo semántico buscan definir el conjunto de relaciones permitidas, el conjunto de atributos para cada una de estas relaciones y el tipo de cada uno de estos atributos. Para lograr esto, se definió un lenguaje en el que existen dos tipos de reglas. En el primero se define el conjunto de relaciones que están presentes en el sistema y que se validarán sobre una consulta ingresada al Análisis Sintáctico y Semántico. Si este conjunto se reemplaza por la palabra reservada ANY, se deshabilita de inmediato el chequeo semántico. El segundo tipo de regla define para cada relación, el conjunto de atributos que la conforman y sus tipos. El conjunto se puede reemplazar por la palabra reservada ANY con lo cual no se validarían los atributos para esta relación. La misma palabra ANY en lugar del tipo de dato de un atributo deshabilita el chequeo de tipos de datos.

4.4. Prototipo

Con el fin de validar esta propuesta se implementó un prototipo de OPTIQ que incluye las principales funcionalidades del sistema. Adicionalmente el prototipo incluye la configuración e implementación del Adaptador sobre PinS [11]. Sobre este prototipo se hicieron las evaluaciones cuyos resultados se presentan en el capítulo 5.

4.4.1. Componentes de la Aplicación Optimizador

La figura 4 muestra los componentes del Optimizador de OPTIQ. El Parser, PlanBuilder, Compiler y Executer están encargados de llevar a cabo cada uno de los pasos del proceso de optimización descrito en la sección 4.2. De esta forma, el Parser ejecuta el Análisis Sintáctico y Semántico de la consulta, el PlanBuilder se encarga de la Optimización Dependiente del Estado, mientras que el Compiler y Executer se encargan respectivamente de la Reescritura de la Consulta y de su Ejecución.

El componente Rules tiene como finalidad mantener el modelo de reglas que requiere el optimizador para su funcionamiento. Es así como en este componente contiene la representación de las Reglas de Generación de Planes de Ejecución y las Reglas de Chequeo Semántico.

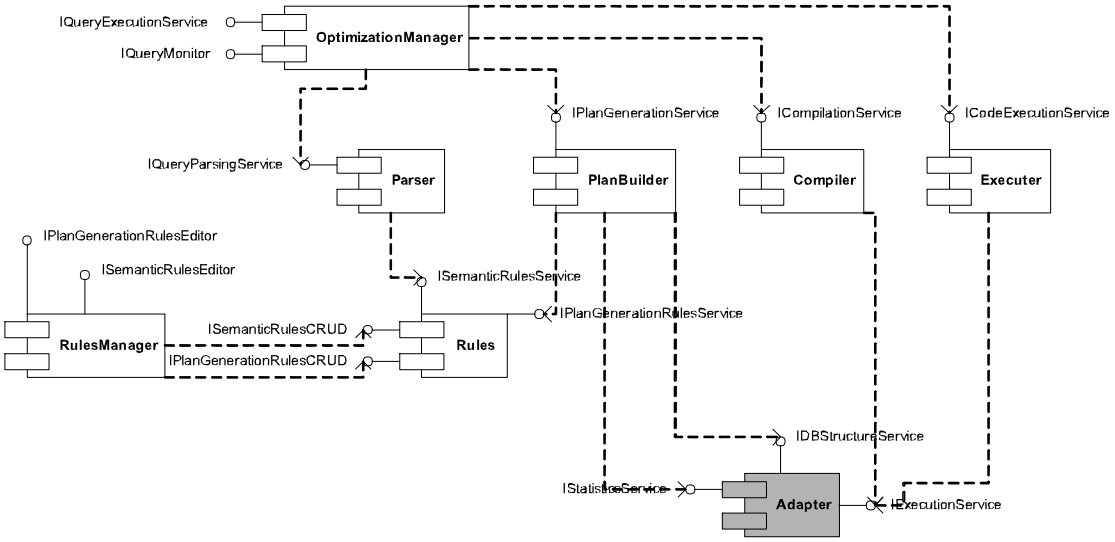


Figura 4. Componentes del Optimizador OPTIQ.

El componente OptimizationManager recibe las peticiones de optimización y ejecución desde la interfaz de usuario y dirige los pasos sucesivos de optimización a través de los componentes Parser, PlanBuilder, Compiler y Executer. Otra de las responsabilidades del OptimizationManager consiste en ofrecer un servicio de consulta de estadísticas e información de control de las consultas ingresadas, representado en la figura 4 como IQueryMonitor. Este servicio no se encuentra implementado en la versión inicial del prototipo.

Por último, el RulesManager ofrece los servicios necesarios para permitir al usuario configurar las reglas del componente Rules. Esto incluye la lógica para recibir del usuario las reglas expresadas en los lenguajes descritos en la sección 4.3, validarlas y procesarlas para persistirlas en el modelo.

Adaptador

El objetivo del Adaptador, es servir de intermediario entre el Optimizador y el Sistema P2P - DHT. Para lograr esto, el Adaptador cuenta con tres componentes principales: DBCatalog donde se implementa la lógica de consulta del estado del sistema; OperatorExecution donde se implementa la lógica de ejecución de operadores de los Planes Ejecutables; y Environment que representa el ambiente de ejecución de las funciones de consulta y los operadores.

4.4.2. Funcionalidad del Prototipo

La funcionalidad del prototipo se puede dividir en dos categorías principales, dependiendo de la etapa del ciclo de vida en la que se encuentre el desarrollo: Preparación del Sistema y Ejecución de Consultas Optimizada.

Preparación del Sistema. La preparación del sistema consiste en las actividades de implementación y configuración, llevadas a cabo por el usuario administrador, necesarias para el correcto funcionamiento del componente de optimización. Estas actividades se llevan a cabo previamente al despliegue del sistema y son:

1. Definición y Configuración de Reglas de Generación de Planes de Ejecución: Consiste en la caracterización de las estrategias de ejecución de consultas que apliquen al sistema sobre el que se implantará OPTIQ, así como las condiciones en las cuales cada una de estas estrategias es óptima. El producto de este análisis es un conjunto de heurísticas de selección de planes de ejecución como las detalladas en la tabla 2, las cuales son ingresadas al sistema mediante el lenguaje descrito en la sección 4.3.1.
2. Definición y Configuración de Reglas de Chequeo Semántico: Consiste en definir y configurar las relaciones, atributos y tipos presentes en el sistema mediante el lenguaje descrito en la sección 4.3.2. En caso de querer mantener una consistencia relajada en la cual no se validen partes de la semántica del sistema, se configuran dichas excepciones mediante las respectivas palabras clave provistas por el lenguaje.
3. Implementación de Operadores y Funciones de Consulta del Estado del Sistema: En este paso se implementan en el componente Adaptador, los operadores y las funciones de consulta del estado que intervienen en las Reglas de Generación de Planes de Ejecución previamente caracterizadas. La implementación de cada una de estas funciones y comandos consiste en la codificación de una clase que contenga la lógica específica de ejecución o de consulta del estado sobre la capa P2P - DHT.

Optimización y Ejecución de las Consultas. Una vez finalizada la preparación y despliegue del sistema, una consulta ingresada en OPTIQ se procesará en los siguientes pasos sucesivos:

1. Análisis Sintáctico y Semántico: En este paso a la consulta ingresada se le aplica una validación sintáctica y es transformada a una representación interna (Árbol de Consulta) para que sea fácilmente procesable por los siguientes pasos de la optimización. Sobre esta representación se aplican las Reglas de Chequeo Semántico (sección 4.3.2).
2. Optimización Dependiente del Estado: Basándose en el Árbol de Consulta y las Reglas de Generación de Planes de Ejecución, el sistema selecciona el plan de ejecución óptimo. Con este fin se consulta el estado del sistema mediante la ejecución de las Funciones de Consulta implementadas en el Adaptador.
3. Rescritura de la Consulta: Basándose en el Plan de Ejecución seleccionado y el Árbol de Consulta se genera un Plan Ejecutable con código de bajo nivel. Este plan está conformado por operadores específicos implementados sobre el Adaptador con todos los parámetros de la consulta, el cual está listo para ser ejecutado directamente sobre el ambiente de ejecución.
4. Ejecución de la Consulta: En este paso el Plan Ejecutable es ejecutado sobre el Adaptador, el cual redirige los requerimientos hacia la capa P2P - DHT. Los resultados recibidos son presentados al usuario.

Para entender mejor las etapas del procesamiento de una consulta, supongamos un escenario de implementación de la consulta Q1 (figura 5). En este escenario se implementa OPTIQ sobre PinS, sistema en el cual existe implícitamente una única relación, nombrada objeto. Los objetos manejados por el sistema en este escenario, son documentos que manejan los metadatos autor, pub (fecha), type y publisher. El atributo publisher no se encuentra indexado y la selectividad de los términos de la consulta en las líneas 3, 4 y 5 no supera el 15% de la cardinalidad del JOIN.

```
Q1.
1. select *
2. from objeto o1, objeto o2
3. where o1.type = "articulo"
4.    and o2.type = "libro"
5.    and o1.pub > '01/01/2009'
6.    and o1.publisher = o2.publisher;
```

Figura 5. Ejemplo de consulta ingresada en un lenguaje declarativo.

En este escenario se configuran las reglas de chequeo semántico de la figura 6 donde existe la relación *objeto* y se permite cualquier atributo para dicha relación, en este caso se deshabilita la verificación semántica asociada a los atributos de la relación.

```
RELATIONS: {objeto}
objeto: {ANY}
```

Figura 6. Reglas de chequeo semántico donde se permite cualquier atributo para una única relación "objeto".

Asimismo se configuran las reglas de generación de planes de ejecución basadas en las heurísticas y estrategias de ejecución propuestas por PinS presentadas en la figura 3.

Al ingresar **Q1**, el procesamiento inicia en el *Análisis Sintáctico y Semántico* con la generación de la representación interna y el chequeo semántico el cual, al permitirse cualquier atributo, arroja un resultado positivo.

El siguiente paso es la generación del árbol de ejecución en el componente de *Optimización Dependiente del Estado*. Basándose en los métodos *ST_index_over*, *ST_join_cardinality* y *ST_Selectivity* (que determinan respectivamente la presencia de un índice, la cardinalidad de una término tipo join y la selectividad de un predicado), el Optimizador opta por el plan de ejecución basado en la estrategia *NLJ* con reducción, el cual se implementa a través de los operadores *INDEX_SCAN*, *REDUCTION* y *NLJ_JOIN* de la siguiente manera:

```
NLJ_JOIN("o1.publisher=o2. publisher", s)[
  REDUCTION(local)[
    INDEX_SCAN(INEQ_JOIN_INDX, {"o1.pub>01/01/
      2009"}, s)[
       INDEX_SCAN(EQU_JOIN_INDX, {"o2.type=libro",
        "o1.type=articulo"}.s)[
]]]]
```

En el anterior plan de ejecución, los operadores *INDEX_SCAN*, *REDUCTION* y *NLJ_JOIN* son representaciones de los algoritmos de búsqueda por índice, reducción y NLJ de PinS descritos en [2] y en [11].

El siguiente paso de la optimización es rescribir la consulta en términos del lenguaje de ejecución del optimizador. El lenguaje de ejecución usa las operaciones provistas por el Adaptador en su componente de ejecución, los cuales, para nuestro ejemplo son:

```
adapter.execution.PinSNestedLoopJoin("o1.publisher=o2.publisher")
adapter.execution.Reduction()
adapter.execution.PinSIndexScan
(INEQU_JOIN_INDX,{o1.pub>01/01/2009})
adapter.execution.PinSIndexScan
(EQU_JOIN_INDX,{o2.type=libro,o1.type=articulo})
```

Una vez se ha generado el plan de ejecución se pasa a su aplicación por parte del componente de *Ejecución*. Éste, invoca las operaciones sobre el Adaptador el cual dirige los requerimientos hacia las capas inferiores, y recibe los resultados para retornárselos al componente de *Ejecución*.

V. EVALUACIÓN

Este capítulo presenta la evaluación de la solución, midiendo que cumpla con los siguientes requerimientos:

R1 - Respuestas Correctas y Completas: Al ejecutar las consultas con OPTIQ se deben obtener los mismos resultados que con el Sistema P2P – DHT sobre el cual se está implementando. Esto se demostrará en la sección 5.2.

R2 - Balanceo de carga y distribución: La aplicación no debe sobrecargar ningún nodo ni crear esquemas donde se concentre el procesamiento ni el almacenamiento. Se validará en la sección 5.3.

R3 - Eficiencia: El sistema no debe afectar de manera significativa los tiempos de respuesta del Sistema P2P – DHT sobre el que se implementa. Se validará en le sección 5.4.

R4 - Expresividad y Flexibilidad: Los lenguajes deben ser suficientes para implementar cualquier regla que desee configurar el administrador de la aplicación. Esto se ejemplificó en la figura 3 de la sección 4.3.1. A su vez, y como parte de nuestra investigación, se lograron expresar todas las reglas de la Tabla 2 mediante el lenguaje de *Reglas de Generación de Planes de Ejecución*, aunque no se incluyen en este artículo dado su extensión.

R5 - Bajo Acoplamiento y Flexibilidad: OPTIQ debe presentar un bajo acoplamiento con el Sistema P2P – DHT con el fin de permitir su posterior adaptación a otros sistemas. Esto se cumple mediante la arquitectura propuesta en la sección 4.2.

5.1. Metodología de Experimentación

Los resultados presentados en las secciones 5.2, 5.3 y 5.4 se obtuvieron mediante el despliegue de OPTIQ sobre la plataforma Grid5000 [26]. Para la experimentación se emplearon 60 máquinas IBM eServer 326m, distribuidas uniformemente en dos *cluster*, con procesadores AMD Opteron 246 (2.0 GHz) o AMD Opteron 250 (2.4 GHz), memoria de 2 Gb y 80 Gb en disco. En cada máquina se ejecutaron 20 nodos, para un total de 1200 nodos conectados al sistema. Los objetos eran documentos con los metadatos: EXP (llave primaria), PUB (fecha de publicación), AUTOR, TYPE y PUBLISHER. Los índices se configuraron en PinS sobre EXP, PUB y AUTOR.

5.2. Exactitud y Completitud de los Resultados

La verificación de la exactitud y completitud de los resultados se realizó cargando para cada escenario de pruebas, cierta cantidad de objetos en el sistema y ejecutando consultas en dos nodos diferentes. En el primer nodo se ejecutaban las consultas sobre PinS independiente, mientras que en el segundo se ejecutaba sobre OPTIQ. En cada escenario se ejecutaron dos consultas:

Qa: o1.autor=o2.autor *AND* o1.type=article *AND* o2.type=book

Qb: o1.type=o2.type *AND* o1.autor=GaoXingjian *AND* o2.autor=GabrielaMistral

Para la ejecución de la consulta **Qa** se configuró PinS de tal forma que usara la estrategia *Index Based Join*, mientras que para **Qb** se configuró para que usara *Nested Loop Join*. Por su parte OPTIQ seleccionaba una u otra estrategia de forma automática. Es así como, teniendo en cuenta las reglas definidas para PinS en la tabla 2, OPTIQ seleccionaba siempre la estrategia *Index Based Join* para **Qa**, y la estrategia *Nested Loop Join* para **Qb**. Los resultados mostraron que los resultados obtenidos por los dos nodos son idénticos.

5.3. Distribución de los Objetos

OPTIQ no inserta nuevos tipos de objetos ni metadatos al sistema, ni afecta los mecanismos de distribución de los objetos. Teniendo en cuenta lo anterior, es de esperarse que mantenga el balanceo de carga de almacenamiento del Sistema P2P – DHT sobre el cual se implementa. Con el fin de comprobar este enunciado, se midió la cantidad de objetos almacenados en cada nodo del despliegue del Sistema en Grid5000. En estos se incluyen objetos de tipo datos, objetos de tipo metadatos y las réplicas que el sistema genera.

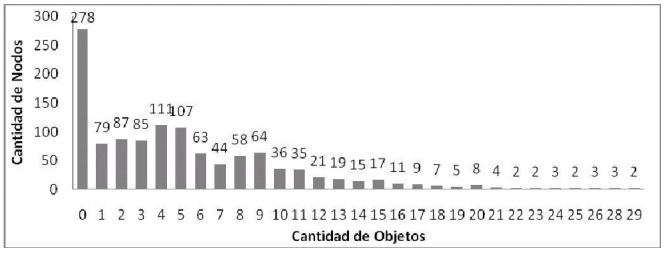


Figura 7. Distribución de los Objetos en los Nodos del Sistema.

Para hacer esta comprobación se cargaron 3000 objetos en el sistema obteniendo una distribución uniforme observada en la Figura 7. En relación a los datos, el 86% de los nodos contienen entre 0 y 10 objetos, llegando a un máximo de 29 objetos en 2 nodos. Se obtuvieron resultados similares en cuanto a distribución de los metadatos en la red.

5.4. Desempeño del Optimizador

El desempeño del Optimizador se mide en términos de la diferencia entre el tiempo de respuesta de una consulta ejecutada con cierta estrategia en OPTIQ y el tiempo de respuesta de la misma consulta ejecutada con la misma estrategia en PinS. Se espera que el tiempo agregado por OPTIQ a la ejecución de una consulta, esto es, el tiempo que se demora en construir el plan

de ejecución, sea mínimo en comparación al tiempo total de la consulta y además se mantenga estable sin importar que tanto escale el sistema P2P – DHT en cuanto a cantidad de objetos consultados.

Para esto, de forma similar a las pruebas realizadas en la sección 5.1, se implementaron diferentes escenarios de pruebas variando la cantidad de objetos en el sistema. Sobre cada escenario se ejecutaron **Qa** y **Qb** usando las estrategias *Index Based Join* y *Nested Loop Join* respectivamente.

La Figura 8 (a) muestra los tiempos de respuesta para la ejecución de Qa con la estrategia *Index Based Join* sobre PinS Independiente y OPTIQ. Los resultados demuestran que la diferencia entre tiempos de respuesta se mantiene en promedio en 35 ms. y no sobrepasa los 85 ms. La Figura 8 (b) muestra resultados similares para Qb ejecutada con la estrategia *Nested Loop Join*.

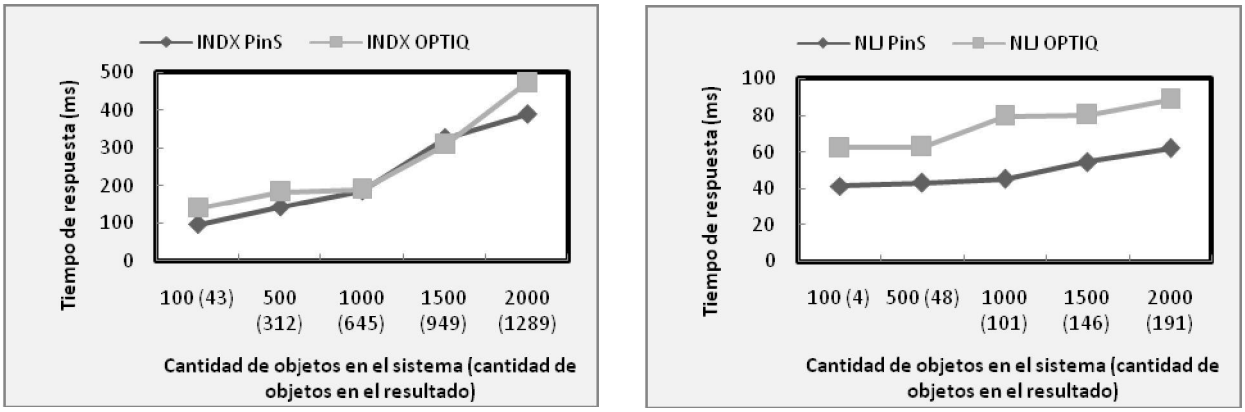


Figura 8. (a) Tiempos de Respuesta de ejecución en OPTIQ y PinS Independiente para la consulta Qa usando la estrategia Index Based Join. (b) Tiempos de Respuesta de ejecución en OPTIQ y PinS Independiente para la consulta Qb usando la estrategia Nested Loop Join.

La Figura 9 muestra los tiempos que emplea OPTIQ en procesar los dos tipos de consultas en cada escenario. Se puede observar que los tiempos se mantienen constantes por debajo de 4 ms. con lo que el tiempo de optimización no llega a sobrepasar el 6% del tiempo total de la consulta. Esto se debe a que sin importar el tamaño de la red en términos de nodos u objetos, el procesamiento de una consulta es el mismo, con lo cual se demuestra que la solución es escalable. Este tiempo no justifica la diferencia de 35 ms. en promedio entre los tiempos de ejecución de las consultas en OPTIQ y en PinS, los cuales son atribuibles a la implementación de los comandos de ejecución en el Adaptador.

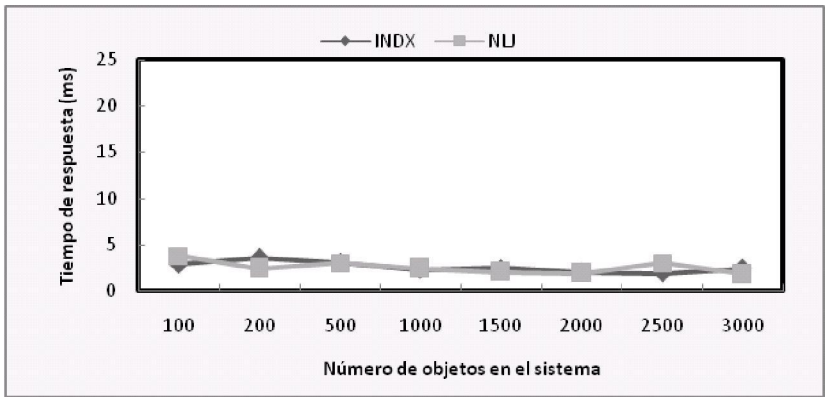


Figura 9. Tiempos de procesamiento de las consultas Qa (Index Based Join) y Qb (Nested Loop Join) en el componente de optimización de OPTIQ. Este tiempo no incluye la ejecución de los operadores en el sistema.

VI. CONCLUSIONES Y TRABAJO FUTURO

Los sistemas P2P - DHT han demostrado su potencial para el manejo de grandes volúmenes de datos. Sobre estos sistemas se han construido muchas soluciones (KSS [1], PinS [2], [3], MAAN [4], [5], [6], Mercury [7], Armada [8], [9], KLEE [10], PIER [12]) que brindan consultas declarativas e implementan estrategias que dependen en gran medida del contexto (presencia de índices, selectividad de los términos, tamaño de las relaciones, etc.). Sin embargo, la capacidad de seleccionar una u otra estrategia en el momento de ejecución de una consulta es limitada por el fuerte acoplamiento que hay entre las

funcionalidades de consulta y de optimización.

Otros trabajos ([21], [22], [23]) permiten la selección de diversas estrategias en tiempo de ejecución. Sin embargo, la responsabilidad de seleccionar una estrategia recae en gran medida en el usuario, quien debe tener un conocimiento acerca de los conceptos de optimización y del funcionamiento interno del sistema.

Este trabajo propone OPTIQ, un Optimizador de Consultas para Sistemas P2P, configurable y desacoplado, el cual es capaz de seleccionar en tiempo de ejecución la mejor estrategia para una consulta dependiendo del contexto. Para esto se vale de las Reglas de Generación de Planes de Ejecución, las cuales son configurables por el usuario administrador mediante un lenguaje también propuesto en este trabajo.

Adicionalmente en la arquitectura de OPTIQ se incluye un componente para el chequeo semántico de la consulta. Este componente también es configurable mediante un lenguaje definido para tal fin (Reglas de Chequeo Semántico). Como parte del trabajo de validación del lenguaje, se caracterizaron las diferentes estrategias de ejecución de los diferentes tipos de consultas sobre los trabajos revisados (MAAN [4], PIER [12], PIERSearch [24], [3], PinS [2,11], Mercury [7] y Armada [8]). Asimismo se identificaron las condiciones en las cuales cada estrategia es considerada como la óptima y se expresaron estas reglas de selección y estrategias de ejecución en el lenguaje propuesto.

Con el fin de validar la propuesta se realizó un prototipo de OPTIQ implementado sobre PinS, utilizado en Grid5000 [26] para verificar las buenas propiedades de la propuesta. El prototipo se construyó con una arquitectura que desacopla la funcionalidad de optimización de la funcionalidad del procesamiento de consultas del Sistema P2P - DHT mediante el uso de un componente Adaptador. Esta independencia entre las capas de Optimización y P2P - DHT permite que la primera pueda adaptarse a cualquier sistema P2P - DHT con un mínimo esfuerzo.

Los resultados obtenidos en Grid5000 demuestran que OPTIQ emplea menos del 6% del tiempo total de la consulta en la optimización. OPTIQ no afecta la escalabilidad del sistemas P2P DHT utilizado en términos de los objetos en el sistema y de número de objetos. Los tiempos de optimización están por debajo de 4 ms. mientras que los tiempos totales de consulta se mantuvieron alineados con los tiempos de consulta de PinS con un retardo promedio de 35 ms. Asimismo se demostró que OPTIQ mantuvo la precisión y exactitud de los resultados de PinS, así como las propiedades de distribución y balanceo de carga de este sistema.

Teniendo en cuenta los requerimientos de implementación para la evaluación sobre PinS, no fue necesaria la codificación de las funcionalidades de recopilación de estadísticas para la adaptación del optimizador; distribución de fragmentos del plan de ejecución, *Pipe-Lining*, enumeración de lugares alternativos para ejecución de operadores ni chequeo semántico. Estas funcionalidades merecen una revisión detallada y son materia de estudio futuro.

La caracterización de las Reglas de Generación de Planes de Ejecución sobre los trabajos en consultas declarativas P2P – DHT revisados en el capítulo 3.2 permitirá una implementación sencilla de OPTIQ sobre estos otros sistemas. Una nueva implementación aportaría valiosos elementos a la solución, tales como nuevos operadores o nuevos componentes para los lenguajes, lo que permitiría validar y garantizar la generalidad del Optimizador.

BIBLIOGRAFÍA

[1] Gnawali, Omprakash D., 2002. A Keyword-Set Search System for Peer-to-Peer Networks. Boston: Massachusetts Institute of Technology.

[2] Villamil M., Roncancio C. y Labbé C., 2006. PinS: Peer to Peer Interrogation and Indexing System. St. Martin d'Hères, France: LSR-IMAG Laboratory BP 72.

[3] Triantafillou P. and Pitoura T., 2003. Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. Patras, Grecia : Department of Computer Engineering and Informatics - University of Patras.

[4] Cai M., Frank M., Chen J. and Szekely P., 2003. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Marina del Rey, CA, USA: IEEE.

[5] Sahin O. D., Gupta A., Agrawal D. and El Abbadi A., 2002. Query Processing Over Peer-To-Peer Data Sharing Systems. Santa Barbara, CA, USA: Department of Computer Science - University of California at Santa Barbara.

[6] Andrzejak A. and Xu Z., 2002. Scalable, Efficient Range Queries for Grid Information Services. Palo Alto CA. USA: Hewlett-Packard Laboratories.

[7] Bharambe A., Agrawal M. y Seshan S., 2004. Mercury: Supporting Scalable Multi-Attribute Range Queries. En Proc. of ACM SIGCOMM.

[8] Li D., Cao J., Lu X., and Chan K., 2009. Efficient Range Query Processing in Peer-to-Peer Systems. ieeexplore.ieee.org. [En línea][Citado el: 15 de 02 de 2009.] http://ieeexplore.ieee.org.biblioteca.uniandes.edu.co:8080/stamp/stamp.jsp?arnumber=4527242&isnumber=4688924.

[9] Akbarinia R., Pacitti E. y Valduriez P., 2007. Processing Top-k Queries in Distributed Hash Tables. [En línea] [Citado el: 09 de 03 de 2009.] http://www.springerlink.com/content/p6r2422738512470/.

[10] Michel S., Triantafillou P. and Wei G., 2005. KLEE: a framework for distributed top-k query algorithms. [En línea] [Citado el: 09-03-2009.]http://netcins.ceid.upatras.gr/papers/Klee_VLDB.pdf.

[11] Prada C., Villamil M. and Roncancio C., 2008. Join Queries in P2P DHT Systems. Bogota, Colombia - Grenoble, Francia : Los Andes University,

[12] Huebsch R., Hellerstein J., Lanham N., Loo B., Shenker S. and Stoica I., 2003. Querying the internet with PIER. Berkeley, CA, USA: ACM,

[13] Bonifati A., Chrysanthis P., Ouksel A. and Sattler K., 2008. Distributed databases and peer-to-peer databases: past and present. s.l.: ACM.

[14] Kossmann, D., 2000. The State of the Art in Distributed Query Processing. s.l.: ACM Computing Surveys.

[15] Garcia H., Ullman J. and Widom J., 2000. Database System Implementation. New Jersey, USA: Prentice Hall.

[16] Chan I., 2008. Oracle Database Performance Tuning Guide 11g Release 1 (11.1). Redwood Shores, CA 94065 U.S.A.: Oracle Corporation.

[17] Richard S., 2008. Oracle Database Concepts, 11g Release 1 (11.1). Redwood Shores, CA 94065 U.S.A.: Oracle Corporation.

[18] Bell D. and Grimson J., 1992. Distributed Database Systems. Gran Bretaña: Addison Wesley.

[19] DeWitt D. and Gray J., 1992. Parallel database systems: the future of high performance database systems. [En línea] [Citado el: 23-03-2009.] http://portal.acm.org/citation.cfm?id=129894.

[20] Bauer M., 2002. Oracle9i Real Application Clusters Concepts, Release 2 (9.2). Redwood Shores, CA 94065 USA.: Oracle Corporation.

[21] Brunkhorst I., Dhraief H., Kemper A., Nejdl W. and Wiesner C., 2003 Distributed Queries and Query Optimization in Schema-Based P2P-Systems. [En línea] [Citado el: 31-01-2009.] http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2003/VLDB_2003.pdf.

[22] Chen Q. and Hsu M., 2008. Correlated Query Process and P2P Execution. Palo Alto, California, USA: HP Labs.

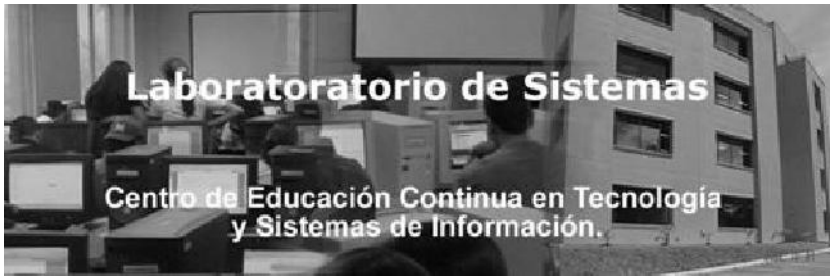
[23] Huebsch R, Chun B., Hellerstein J., Loo B, Maniatis P., Roscoe T., Shenker S., Stoica I. and Yumerefendi A., 2005. The Architecture of PIER: an Internet-Scale Query Processor. [En línea] [Citado el: 23 de 02 de 2009.] http://www.huebsch.org/papers/CIDR05.pdf. 67.

[24] Loo B., Hellerstein J., Huebsch R., Shenker S. and Stoica I., 2004. Enhancing P2P File-Sharing with an Internet-Scale Query Processor. [En línea] [Citado el: 15-02-2009.] http://db.cs.berkeley.edu/papers/vldb04-piersearch.pdf.

[25] Patrick K., 2003. Gnutella - A Protocol for a Revolution. [En línea] [Citado el: 29-03-2009.] http://rfc-gnutella.sourceforge.net/.

[26] Grid5000:Home - Grid5000, 2009. [En línea] [Citado el: 20-Julio-2009.] www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

Universidad Nacional de Colombia Sede Medellín
Facultad de Minas



Misión

Ofrecer servicios de apoyo a la docencia en cuanto a la operación de los computadores y del software adecuado con miras al desarrollo integral de los futuros ingenieros.

Visión

Avanzamos en la búsqueda de convertir el Laboratorio de Sistemas e Informática en una dependencia ágil, moderna, facilitadora de procesos y cambios, atenta a las necesidades de otras dependencias de la Universidad, cuya labor apoyamos y articulamos. Serviremos con dinamismo, amabilidad y efectividad a todos los integrantes de la comunidad universitaria y a la sociedad en general. Uniremos esfuerzos para construir un ambiente de trabajo cada vez más positivo que propicie la participación, la creatividad y el desarrollo profesional de los integrantes del equipo de trabajo. Propendemos por un Laboratorio como instrumento gestor y generador de proyectos de investigación sustentado en un equipo interdisciplinario de trabajo en torno a la informática aplicada a la ingeniería.

Cursos

- Lenguajes de Programación: Diseño de Páginas Web en ASP.NET con VB.NET, Programación Web PHP y MYSQL, MS-Visual Basic Básico y Avanzado, Java
- Generales: Latex, ARCGIS, MS-Office (Word, Excel y PowerPoint), Excel Financiero, Excel Avanzado, Mantenimiento de Hardware y Software Niveles I y II, MS-Access Básico, MS-Project Básico (Programación y Gerencia de Proyectos), AUTOCAD 2D Básico y 3D, Matlab, Moodle de Apoyo a los Procesos de Enseñanza y Aprendizaje.

Para mayor información, por favor comunicarse a los teléfonos: 4255313, 4255312, 4255355. Calle 59A No. 63 – 020 Medellín (Colombia), Bloque M7, quinto piso.
Email: labsis@unalmed.edu.co http://xue.unalmed.edu.co/cursos