PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

COLOCACIÓN ÓPTIMA DE DATOS MEDIANTE PARTICIÓN DE GRAFOS

BERTTY ALFREDO CONTRERAS ROJAS
RODRIGO ALEXIS PARDO MEZA

INFORME FINAL DE PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL INFORMÁTICA

NOVIEMBRE, 2016

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

Colocación óptima de datos mediante partición de grafos

Bertty Alfredo Contreras Rojas
Rodrigo Alexis Pardo Meza

Profesor Guía: **Wenceslao Palma Muñoz**

Profesor Co-referente: **Héctor Allende Cid**

Carrera: **Ingeniería Civil Informática**

Noviembre, 2016

Resumen

El mundo al conectarse a la nube genera datos de forma masiva. Las empresas tienen acceso a la nube donde hay recursos ilimitados, como lo es un disco infinito que requiere ser administrado. En este contexto no pueden existir cuellos de botella que perjudiquen la velocidad de respuesta al usuario. El beneficio del escalamiento horizontal disminuye mientras más maquinas requieren ser comunicadas. En este punto el flujo de peticiones al sistema distribuido produce una saturación en la red. Para remediar esto se propone analizar las relaciones entre los datos en base a la carga de trabajo. Logrando un distribución que permita buenas velocidades de respuesta. El desagrupamiento se enfocó en la similitud de los datos necesarios en la construcción de las respuestas a las peticiones. Esto provoca la jerarquización del modelo de datos, dado que un grupo de datos es más relevante para algunas solicitudes que para otras. Se crea el concepto de fragmento para el grupo de datos administrado y una estructura de datos que permita la gestión estos fragmentos aportando flexibilidad a la arquitectura de jerarquización. Esta estructura llamada árbol diamante tiene la capacidad de disminuir el tráfico en la red, para esto necesita una subdivisión efectiva del modelo. Se utiliza la minería de texto para dividir los datos que serán solicitados en las peticiones, de esta manera los fragmentos de cada árbol estarán estrechamente relacionados. Al haber exceso de datos en una maquina se realiza el particionamiento controlado de este árbol, que involucra una porción conocida del modelo.

Palabras claves: Base de datos distribuida, Shared-nothing, Partición de grafos, Estructura de Datos, Text mining.

Abstract

The world when connecting to the cloud the data is generated massively. Companies have access to the cloud where there are unlimited resources, like an infinite disk that needs to be managed. In this context, bottlenecks cannot exist that hinder the speed of response to the user. The benefit of horizontal scaling decreases, as more machines require communication. At this point, the flow of requests to the distributed system produces a saturation in the network. To remedy this, it is proposed to analyze the relationships between the data based on the workload. Achieving a distribution that allows good response speeds. The debundling focuses on the similarity of the data needed in the construction of the responses to the requests. This causes the hierarchical ranking of data model, since one data group is more relevant to some requests than others are. The concept of fragment is created for the managed data group and a data structure that allows the management of these fragments providing flexibility to the hierarchical architecture. This structure, called diamond tree has the ability to reduce traffic on the network, for this, it needs an effective subdivision of the model. Text mining is used to divide the data that will be requested in the requests, in this way the fragments of each tree will be closely related. Excessive data on a machine performs controlled partitioning of this tree, which involves a known part of the model.

Keywords: Distributed database, Shared-nothing, Graph partition, Data structure, Text mining.

Índice

Lista de figuras

1 Introducción

Hoy en día las transacciones que ocurren en las organizaciones conllevan el movimiento de grandes flujos de datos entre varias máquinas. Este flujo de datos debe mantenerse controlado para evitar la saturación de la red y mantener el buen funcionamiento del sistema.

Un modelo de datos de gran envergadura, que debe responder a una gran cantidad de peticiones, difícilmente puede mostrarse satisfactorio si no se dispone de un número considerable de servidores para distribuir el procesamiento de su carga de trabajo. Dado que los datos se relacionan de una forma compleja, es una tarea no trivial decidir cómo las tablas se distribuirán en las máquinas disponibles.

El resultado de esta subdivisión determina cómo se moverán finalmente los datos al realizar una consulta sobre la información que tiene el sistema. Este trabajo busca automatizar la detección de una forma de disminuir la comunicación entre los servidores, al analizar la carga de trabajo de la organización, estudiar estas relaciones y distribuirlas en forma inteligente.

Para llevar a cabo el análisis, diseño e implementación de esta herramienta; fue necesario estudiar las formas anteriores en que se intentó abordar esta problemática. Se encontró mucha diversidad, diferentes métodos que intentan generalmente agrupar las tablas más relacionadas a través del análisis de consultas, utilizando diferentes métricas y métodos de división. Gran parte de este proyecto se enfatizó en encontrar una forma de agrupación que tome en cuenta la mayor cantidad de ventajas de los métodos estudiados en el estado del arte.

Haciendo referencia al contenido del documento, primero se llevó a cabo un estudio sobre la problemática, describiendo lo que se requiere solucionar. En este punto se evaluó la relevancia de este proyecto en el contexto actual y sus diversos ámbitos de aplicación. Luego fue desarrollado el estado del arte, mostrando en detalle las distintas formas en que el problema ha sido abordado, se analizaron los pros y contras de cada alternativa a solución y como su estudio afectó al diseño de la solución actual. Luego, se definió formalmente la propuesta de solución, abarcando todos los conceptos y herramientas utilizados para elaborar el sistema final. Finalmente se expondrán los resultados experimentales y sus respectivas conclusiones.

2 Definición del problema

Como se mencionaba anteriormente, la apertura de las redes al mundo, junto con el aumento en la capacidad de cálculo y almacenamiento han hecho posible que las bases de datos crezcan enormemente. Debido a esto han surgido un sin número de técnicas que permiten explotar de forma efectiva la obtención de información a partir de estos datos. Pero es recién en los últimos años en donde se ha promovido la optimización del almacenamiento y manejo de este gran volumen de datos, lo cual ha sido denominado BigData.

A pesar de que actualmente se pueden obtener discos con gran capacidad de almacenamiento, estos no son capaces de almacenar este tipo de base de datos por su gran tamaño, por lo que se ha recurrido al almacenamiento distribuido, que trae consigo un número considerable de problemas asociados a su uso. Uno de los más importantes problemas es la saturación de las redes que se origina por la forma en que están distribuidos los datos al utilizar este tipo de arquitectura. Por lo general, los datos se distribuyen de forma aleatoria entre las máquinas de las organizaciones, esto gatilla que el movimiento de datos a través de la red sea arbitrario y no obedezca ninguna lógica que minimice las comunicaciones.

Debido a que las relaciones existentes entre los datos del esquema no son estudiadas y por consiguiente no se puede estimar a priori la transferencia total que se tenga que llevar a cabo para poder ejecutar una consulta, tampoco se puede predecir el tiempo total comprometido en su ejecución. Es por esto que la colocación de los datos debe ser realizada de forma estratégica ya que gracias a esto se podrán minimizar tanto el tiempo como la saturación en la red. Sin embargo, minimizar ambas variables a la vez es un problema del tipo NP-Hard como lo menciona Golab et al.[13], por lo tanto es una tarea no trivial determinar cómo se debe distribuir el modelo de datos para satisfacer de una mejor manera, sin saturar la red, la carga de trabajo de una organización según su capacidad de almacenamiento.

Otro problema que ha traído consigo este tipo de soluciones distribuidas, es la necesidad de replicación de la arquitectura del software. Esto se debe a que cada elemento del sistema distribuido debe replicarse para poder mantener los niveles de servicio acordados con el cliente (SLA). Esto conlleva al uso de muchas máquinas que permitan replicar los datos críticos, además esta aplicación no permite un mejor desempeño, sino que es una necesidad para que el sistema siga en funcionamiento cuando una de las máquinas no puede seguir en línea, por distintas circunstancias.

Para poder solucionar este problema se ha comenzado a utilizar los cluster de servidores Shared-nothing, que en el último tiempo han proliferado en empresas que están ofreciendo estos sistemas u objetos distribuidos, como lo es Amazon Web Service. Dichas empresas ofrecen una alta disponibilidad, fiabilidad y rendimiento, lo cual permite ocupar

técnicas similares a MapReduce. A pesar de esto, la relación entre la carga de trabajo y el transporte de datos entre los distintos componentes del sistema, no siempre son calculadas de forma eficiente para un uso exitoso del modelo de programación MapReduce. Por lo general estos errores de cálculo que ocasionan congestiones en las redes, provocan una disminución considerable en el rendimiento del sistema en general.

Con lo indicado anteriormente, surge el problema de la colocación de datos, el cual ha sido abordado por diferentes autores como se profundizará en el estado del arte. Cada uno de ellos lo definió dependiendo de los objetivos que se propusieron lograr. Un método que fue ocupado para definir satisfactoriamente este problema fue la programación lineal, mérito de los autores Baev et al.[6]. La cualidad más destacada de este modelo es la sencillez de la representación matemática. Sin embargo, se encuentra acotada a las restricciones propuestas para un almacenamiento no escalable.

Con la intención de presentar el problema de colocación de datos en ambientes distribuidos de una forma simple, en este trabajo se explicará de una forma descriptiva, de modo que se entiendan de forma clara las variables y objetivos que tiene la problemática. Esto se realizará dado que la cantidad de variables y restricciones involucradas son muchas.

Como se presentaba en los párrafos anteriores, los datos que se guardan en la base datos son sumamente importantes por la información que está presente dentro de ellos, al consolidarse como conjunto. Es por esto que es vital tener en consideración la relación existente entre distintos datos al momento de comenzar a describirlos como una variable más del problema. A pesar de poder considerarse cada dato como una variable independiente de las demás, su relación en el conjunto de datos es lo relevante, por lo que la forma de tratar cada elemento será decisivo para el resto de las variables. Sin embargo, como variable independiente simplemente la consideraremos como un registro, tupla u objeto que es parte de una tabla, nodo, clase u valor del tipo de base de datos que se esté analizando. Se definirá así, por la forma en la que los datos serán generados en la solución y por ser un elemento atómico que se genera sin tomar en cuenta lo previamente existente en la base de datos.

Por otra parte, una variable no menos importante es la carga de trabajo. Para cualquier tipo de base de datos, esta se expresa como todas las consultas que se realizan sobre los datos que la organización almacena en sus tablas. De esta forma, la carga de trabajo es el procesamiento que se ejecuta sobre los datos que la organización estima conveniente gestionar.

Las consultas se encargan de relacionar distintos grupos de datos, que procesados bajo una la lógica de negocios determinada, arrojan un conjunto reducido de los mismos

cuya nueva configuración se denomina respuesta de la consulta. Este conjunto de datos resultante no representa en forma clara la gran cantidad de datos, que tuvieron que ser analizados y procesados. En este aspecto se hace clave la capacidad de poder identificar el real impacto de las consultas y el costo que produce su ejecución en la base de datos. La forma de medir este costo se basa en el estudio previo de todos los elementos procesados en la ejecución de las consultas. Poder clasificar los datos y agrupar las consultas es una acción que puede mejorar en forma significativa el rendimiento final del sistema, lo cual se describe en el paper de Curino et al. [9].

Es relevante el estudio de la carga de trabajo de las organizaciones debido a que si dos consultas utilizan un conjunto de datos similar; vale la pena invertir en investigación para que ambas consultas se ejecuten en una misma máquina, que contenga ese conjunto de datos que las relaciona, de esta forma la comunicación entre máquinas se ve reducida directamente y la saturación de la red es prevenida. Para hacer explícito el estudio de la carga de trabajo, se debe hacer una revisión de todas las consultas que la organización objetivo ejecuta en sus sistemas. Las soluciones que se discutirán en la siguiente sección proponen diferentes formas de lidiar con la carga de trabajo para dividir el modelo de datos de forma inteligente.

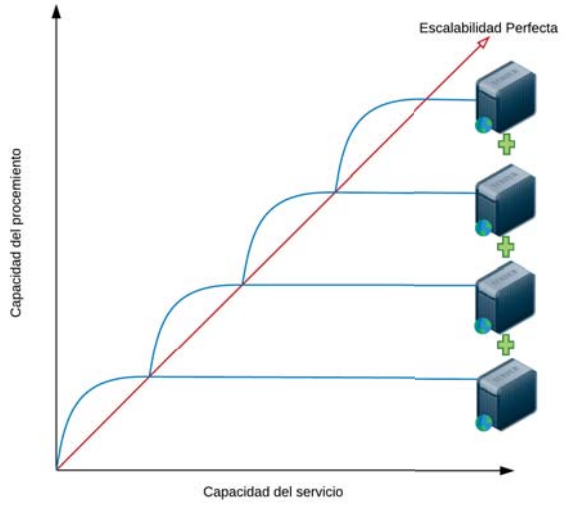


Figura 1: Escalabilidad Horizontal

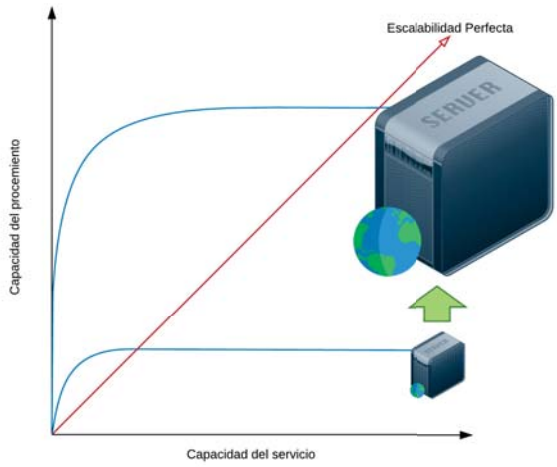


Figura 2: Escalabilidad Vertical

Los servidores son una variable que afecta directamente la calidad del servicio entregado. La capacidad de acceder a los datos con un buen tiempo de respuesta es un factor clave para realizar diversos tipos de operaciones. Es dificil mantener una performance constante cuando las bases de datos crecen más rapido que los servidores. La necesidad de almacenar datos es una constante que obliga a adquirir más capacidad de almacenamiento periódicamente, para esto se requiere comprar más discos o servidores; el problema es que por cada servidor añadido los costos de mantención son cada vez mayores. Por consecuencia, en un punto el beneficio obtenido no cumple las expectativas de la inversión realizada.

En la siguiente ilustración se muestra el comportamiento de la escalabilidad de los servidores 1 y 2. La escalabilidad perfecta es difícil de conseguir. A medida que se aumenta la calidad del servicio se deben aumentar la cantidad máquinas involucradas en el procesamiento, lo que muestra la figura 3 entre más máquinas se incorporan al sistema el costo eléctrico de las máquinas se comienza a disparar por la cantidad de energía que se necesita para poder mantener dicha tasa de escalabilidad, por lo que agregar más servidores no siempre va a ser una opción rentable económicamente.

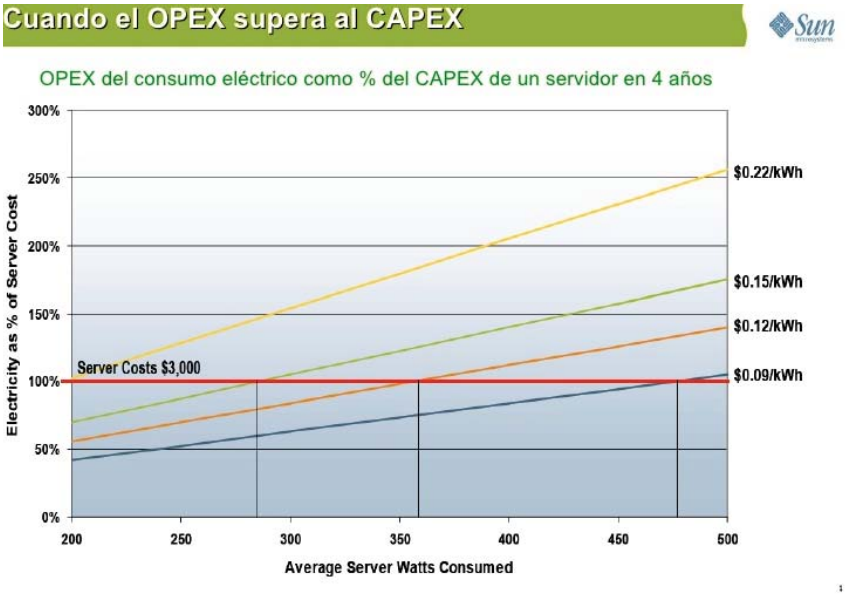


Figura 3: Costo vs Consumo eléctrico

Una variable que se debe tomar en cuenta en la actualidad, debido a que los datos llegan desde distintos puntos del planeta gracias a la red global que interconecta los cinco continentes, es la tasa de llegada de los datos. Esta variable es la que determina el tiempo y buffer de entrada que deben poseer los sistemas distribuidos. Esto debe ser así para poder determinar los tiempos de respuesta que deben presentarse al ejecutar cada una de las consultas que se realicen sobre la base de datos distribuida, dentro de los equipos que componen el sistema. Cabe destacar que la tasa de llegada de los datos tiene un máximo, que está dado por las características del sistema distribuido y su escalabilidad respecto a esta variable. Este aspecto del sistema no se puede descuidar porque al aumentar y superar el máximo, los tiempos de ejecución se vuelven prohibitivos para los cálculos y reestructuración del modelo en tiempo de ejecución, como lo mencionan en su escrito los autores Zeitler et al.[20].

El problema de colocación de datos puede ser formalmente definido, como lo hizo Baev en [6], como un problema de programación lineal. En aquel escrito se definen restricciones que se adecúan a la representación. Para comprender en esta ocasión el problema de la colocación de datos este será simplificado, de esta forma, en la siguiente definición sólo consideraremos las tablas de la base de datos y las consultas que se realizan sobre ellas.

Con la idea de facilitar la comprensión de la problemática, al mismo tiempo de formalizar lo que se desea resolver, definamos las siguientes variables:

1. Sea n tablas serán definidas como $T = (T_1, T_2, \ldots, T_n)$, en donde la j-esima tabla debe tener un tamaño dado por t_j, donde $t_j > 0$

2. Sea m consultas serán definidas como $Q = (Q_1, Q_2, \ldots, Q_m)$, cada una de las consultas puede hacer referencia a una o más tablas, donde $Q_i \subseteq T$

3. Sea l servidores serán definidos como $M = (M_1, M_2, \ldots, M_l)$, y la capacidad de almacenamiento del k-esimo servidor está dado por μ_k, donde $\mu_k > 0$

4. Para cada Q_i y $T_j \in Q_i$, existe un costo de comunicación que se denotará C_i^j, donde $C_i^j > 0$. C_i^j representa el costo incurrido en la transferencia de cualquier parte de la tabla T_j que se necesite para poder evaluar la consulta Q_i

Lo que se debe hacer para poder resolver este problema es poder asignar cada una de las tablas T en alguno de los l servidores que se encuentra en M, no violando la capacidad del servidor dado por μ_k y mientras se minimiza el costo de comunicación que se debe pagar al momento de procesar las consultas.

Por lo anterior, se debe hacer que la colocación de las tablas T a los servidores M sean una función $f : T \to M$. Para que la función f sea legal se debe cumplir que:

$$(\sum_{j:f(T_j=M_k)} t_j) \leq \mu_k \text{ para } 1 \leq k \leq l.$$

Lo anterior se puede describir como la suma de los tamaños de cada una de las tablas colocadas en el servidor M_k no sea mayor a la capacidad μ_k.

Para poder hacer más entendible el modelo que se está presentado, se permitirá que cualquier consulta se pueda procesar en cualquiera de los l servidores. El costo de procesamiento se entenderá como el costo asociado al realizar cualquier consulta Q_i en el servidor M_k y será el costo de transportar a M_k a todas aquellas tablas $T_j \in Q_i$ que no se encuentran almacenadas en el servidor M_k, lo que se puede expresar de la siguiente forma:

$$\sum_{T_j \in Q_i : f(T_j) \neq M_k} C_i^j$$

A este modelo de procesamiento, los autores del paper de Golab et al [13] lo denominaron *query-site execution*. Esto se debió a que el modelo surgió para poder minimizar el costo global de comunicación al momento de realizar el procesamiento de consultas.

En primera instancia el problema siempre ha sido abordado con un enfoque en el cual los datos son lo más importante y lo que impulsa la forma en la que quedan las diferentes colocaciones. Pero en el presente las consultas, la carga de trabajo será quien determine las colocaciones. Debido a que en el mundo real los datos llegan en forma de *stream*, se adaptará el problema para funcionar en ambientes dinámicos.

Bajo este contexto se definen las siguientes variables, para poder establecer de manera formal el problema de la colocación de datos:

1. Sea S_i los stream que llegan al sistema, para lo cual λ_i será la tasa de llegada de los datos y τ_i como el tamaño de cada tupla que llega al sistema por el S_i y la cantidad de S_i será dada por $|S|$.

2. Sea F_k los fragmentos generados por el procesamiento del algoritmo que se explicará más adelante su funcionamiento, cada F_k tiene un tamaño denotado por $\Upsilon(F_k)$ además que un fragmento se relaciona con una o más Q_j, para poder representar esta relación se dirá que $\Xi(F_k, Q_j)$ es 1 si F_k responde la consulta Q_j y entregará 0 para los demás casos. La cantidad de F_k será representada por $|F|$.

3. Sea M_l el conjunto de máquinas disponibles para poder llevar acabo el almacenamiento y procesamiento de la carga de trabajo. Cada M_l tiene una capacidad denotada por κ_l, a forma de restricción no se podrá modificar μ_l esto se debe a que al agrandar la capacidad se pierde el sentido de particionar la información y distribuirla en un conjunto de servidores; y al reducirla obligaría a generar particiones que no estaban consideradas lo que generaría una restructuración de la colocación de los datos y obligaría reparticionar lo que también hacer perder el sentido del algoritmo que se explicará más adelante.

Como restricción se establecerán los siguientes supuestos, derivados de los explicados descriptivamente en los párrafos anteriores:

1. Las consultas son un conjunto conocido, el cual no varía en el tiempo.

2. Las consultas que llegan son del conjunto ya conocido, y son procesadas durante todo el $T(Q_j)$.

3. la tasa de llegada λ_i no puede aumentar pues de ese modo se hacen prohibitivos los tiempos de cálculos como lo dice Zeitler en [20].

3 Estado del arte

Diversos grupos de matemáticos y profesionales de la informática han intentado dar solución al problema descrito. Dentro de ellos pueden reconocerse distintas formas y diseños de solución. En esta sección serán presentadas las orientaciones de solución más importantes, todas ellas ayudaron en el planteamiento de la solución propuesta en este documento.

3.1 Particionamiento de grafos

Un gran grupo, por no decir la mayoría de las soluciones al problema de colocación de datos, abordó la problemática utilizando la técnica de partición de grafos. Este método implica confeccionar una representación nodo-arista, que permite ocupar los algoritmos de particionamiento de grafos. Esta conversión del problema igualmente implica un problema NP-Hard, pero tiene un set de algoritmos que permiten llegar a una aproximación bastante eficiente y de esta forma se mapea una problema NP-Hard que no tiene una solución eficiente hasta la fecha, por otro problema NP-Hard que cuenta con una aproximación bastante aceptable para el tiempo que involucra la ejecución de los algoritmos.

El gran problema de este tipo de soluciones es que funcionan únicamente de manera estática, los tiempos necesarios para la partición de grafos son excesivos si se supone la llegada de un flujo de datos, además según este modelo de solución la llegada de un nuevo dato supone volver a consumir nuevamente el costo asociado a partir el grafo completamente, ni siquiera funciona de manera localizada al dato entrante. En conclusión si bien la solución final es una buena aproximación al óptimo, está descontextualizada a lo que son las bases de datos, las cuales se caracterizan por ser entidades altamente dinámicas que deber ir creciendo junto con las transacciones de la organización.

3.1.1 Schim

Esta solución propuesta por Curino et al.[9], en donde se presenta la minimización del número de transacciones distribuidas para una determinada carga de trabajo, mediante la asignación y replicación de las tuplas en forma individual en los distintos servidores presentes en el ambiente distribuido. Para esto ellos presentaron la base datos y la carga de trabajo, para este caso serán las consultas que se utilizan, estos dos componentes se presentan como un grafo, cuyos nodos corresponden a las tuplas y los bordes son las tuplas que son accedidas en una misma transacción. En este caso particular los algoritmos de partición de grafos se aplican para poder encontrar las partes equilibradas, donde cada una de las partes será colocada en un servidor distinto, la idea es minimizar los pesos de

los bordes, dado que estos bordes son los que deben ser posteriormente replicados, y entre menor sea el peso de los bordes cortados, menor será el factor de replicación de la base de datos.

Cabe agregar que este método permite separar de forma horizontal las tablas lo que trae como beneficio que la base de datos, pueda dividir grandes tablas que en pequeños conjuntos que son más fáciles de manipular y de trasladar entre servidores. Pero a su vez trae consigo la replicación de muchos datos que quizás no sea necesario replicar y aumentan el coste de la base de datos, además de que imposibilita la opción de cortar en forma vertical las tablas lo que permitiría un acceso a las claves de indexación y así reducir el costo de comunicación entre los distintos servidores. Esta solución tampoco está pensada para ambientes dinámicos con un gran flujo de datos entrantes.

3.1.2 Minimizando el coste de comunicación

En el trabajo de Kayyoor et al.[14], ellos representaron la colocación de datos como un conjunto de tablas, servidores y consultas, ellos decidieron estudiar qué tabla es conveniente replicar a partir del esquema de base de datos y además en que servidor colocar dichas tablas replicadas a partir de las consultas, de este modo reducir al mínimo las consultas que necesitan de distintos servidores, alojando los datos necesarios en las máquinas de cada consulta aunque esto implique que los datos se repitan. Como resultado de tal estrategia lograron reducir el coste de comunicación para la carga de trabajo dada. Esto se puede ver claramente en la figura 6 dado que las consultas provocan distintos movimientos de datos como resultado de sus ejecuciones, si una réplica es inteligentemente administrada puede ahorrarnos el trabajo de mover gran cantidad de datos a través de la red. Es por esto que no da lo mismo en donde se coloquen las réplicas de las tablas.

Figura 4: Ejemplo de Colocación de datos

Kayyoor et al.[14] muestra que vale la pena replicar tablas y aumentar el tamaño de la base de datos con información repetida, al obtener beneficios importantes en el tiempo de ejecución. Esto nos muestra que lo fundamental de los métodos estudiados es encontrar la mayor cantidad de relaciones de datos entre las consultas de la carga de trabajo y explotarlas de la mejor forma posible. De esta forma se puede dar cumplimiento a los tiempos de ejecución de una mejor manera, esta característica es importante en el diseño de la solución posteriormente planteada.

3.1.3 Hipergrafos

Esta representación se puede encontrar en los escritos de Kayyoor et al.[14] y Curino et al.[9]. Este modelo se basa en representar las tablas o tuplas como nodos y las consultas o transacciones son representadas como hiperaristas. Cada una de las hiperaristas es

un conjunto de nodos que representa un conjunto de tablas que están relacionados por una consulta o transacción. En ambos casos los objetivos de optimización son capturados exactamente con el particionamiento de las hiperaristas. Algo que se debe agregar es el hecho de que las hiperaristas que son partidas son las tuplas que se replican en al menos dos máquinas. Una característica que vuelve bastante útil el estudio de la colocación de datos con esta técnica, es el hecho de que las hiperaristas que se cortan corresponden exactamente al número de transacciones distribuidas, de esta forma es muy fácil graficar y ver los beneficios de esta representación.

Un inconveniente que tiene esta solución es que a pesar de poder capturar exactamente los objetivos de optimización, éste no puede capturar más variables como lo es el costo de comunicación de datos. Esto se debe a que asignar de forma adecuada los pesos de las hiperaristas es muy compleja, por no decir imposible debido a la forma en que se describe una hiperarista. No es posible asignar un peso a la comunicación, dado que ésta se corresponde con una comunidad de nodos y no a uno solo como en el caso del particionamiento de grafos convencional.

3.1.4 Grafos bipartitos

La propuesta de solución descrita por Golab et al.[13], es evitar el uso de los hipergrafos por la pérdida de información que tiene dicho método y para evitar su uso hace el siguiente mapeo del problema, en donde las tablas son los nodos del grafo y las consultas son las aristas, lo que resulta muchos más intuitivo al momento de estudiar el problema como un problema de partición de grafos. A pesar de parecer una solución fácil de entender y ser capaz de mostrar dos heurísticas para la replicación de los datos, y que además logra minimizar el coste comunicación entre las distintas máquinas para una carga de trabajo dada. No es capaz de soportar el ingreso de un grupo de datos nuevos por lo que solo funciona cuando los datos no crecen en el tiempo, y esto no es aplicable en la realidad con las base de datos que tenemos creciendo segundo a segundo.

3.1.5 DinPartGroup

El documento escrito por Liroz-Gistau et al.[15], se basa en lo mencionado por Curino et al.[9]. Defendiendo el hecho de que los datos tienen una relación entre sí que puede ayudar a determinar cuán efectivo es manejarlos de forma conjunta, esto les permitió a los autores de dicho paper poder determinar una ecuación que ayuda a determinar la afinidad de un nuevo dato con un grupo de datos denominado fragmento, el cual internamente tiene una afinidad alta. Es por esto que se puede determinar en qué fragmento conviene colocar el nuevo dato.

El concepto de afinidad aportado por los autores, se determina en función de la pérdida de eficiencia del fragmento si se aloja un nuevo dato en él. Esta pérdida radica en la cantidad de consultas que el fragmento como unidad es capaz de satisfacer, si se agrega un nuevo dato al fragmento este número de consultas puede aumentar, el desarrollo de Liroz-Gistau et al. busca el fragmento que sufra menos pérdida de eficiencia al agregar algún hipotético nuevo dato. Si un mismo fragmento es necesario para dar cumplimiento a un gran número de consultas, para los autores es algo negativo, ya que implica directamente que en el futuro ese fragmento debe ser potencialmente transportado para la ejecución de varias consultas diferentes.

Dividir los datos relevantes para un conjunto de consultas en fragmentos que posteriormente deben ser localizados, fue finalmente parte importante de la solución entregada en este proyecto. Sin embargo, tratar de que los fragmentos satisfagan la menor cantidad de consultas posibles, es algo que se puede obviar si se agrupan las consultas que se relacionan de mejor manera, de esta forma resulta en algo positivo esta agrupación debido a que se ahorra espacio en el disco. Además como veremos más tarde, cuando un fragmento satisface un gran número de consultas, lo vuelve un fragmento muy representativo para un conjunto de datos. Esto lo vuelve un objetivo deseable para separar la carga y distribuirla. Tomando en cuenta la estructura de datos creada en este proyecto, el rendimiento del sistema no se hace más bajo.

Una ventaja importante del algoritmo DinPartGroup que se presenta en el escrito, es que permite la carga de un conjunto de nuevos elementos de información dentro de los fragmentos ya existentes, lo cual permite que la base de datos pueda agregar nuevos datos sobre ella en tiempo de ejecución. Esta variante es fundamental, ya que es posible aplicar este algoritmo en forma dinámica, debido a que a diferencia del particionamiento de grafos, los cambios necesarios al agregar un nuevo dato tienen un impacto local, y no es necesario reconfigurar todo el mapeo como en el caso del particionamiento de grafos. Esto da tiempos de ejecución que pueden tolerar un flujo constante de datos.

El inconveniente que posee esta idea es el hecho de que los fragmentos tienen una tamaño que no puede superar al resto de los fragmentos por un δ, es por esta razón que se genera una pérdida de afinidad en los fragmentos lo que obliga a juntar un conjunto de fragmentos de datos para que interactúen por esta pérdida de afinidad producida por el δ. Otro problema que posee tal solución es que la pérdida de afinidad ocasiona un aumento en el tráfico de red, que además no está considerado al momento de separar en distintas máquinas el conjunto de fragmentos. En conclusión, la solución para tener un tamaño parejo de sus fragmentos, deja de asignar la mejor afinidad y comienza a disparar datos al resto de las opciones sin mayor gestión. En la solución aquí planteada, para no tener este déficit de eficiencia, los fragmentos que se manejan no tienen un límite para su crecimiento,

sin embargo, cuando ya no son almacenables en un único servidor, estos son divididos con una herramienta de particionamiento de grafos.

3.1.6 Otras herramientas

Luego de observar lo que se ha estudiado por parte del problema de colocación de datos, es importante mencionar que los algoritmos de particionamiento de base de datos relacionales se han estudiado arduamente en el pasado. No obstante, el esfuerzo de estos trabajos se ha concentrado en el estudio de los datos desagrupados y la afinación del diseño físico, con el fin de acelerar la evaluación de las consultas a través del aprovechamiento de la paralización de la carga de trabajo. Algunas de las estrategias que se pueden destacar son el particionamiento de rango o espectro, particionamiento hash y el particionamiento basado en modelos de costo de consultas [5, 21, 18, 16, 12, 19, 17]. A todo esto se debe agregar que los nuevos sistemas distribuidos, como por ejemplo BigTable[7], no se han optimizado para cargas de trabajo de múltiples tablas relacionales. Además, los almacenes distribuidos con clave/valor, como lo son Amazon Dynamo[10], HDFS[8], RIAK[3] y Quantcast QFS[2] se centran principalmente en la distribución al azar de los datos redundantes. Esto se realiza con el objetivo de poder aumentar la fiabilidad y disponibilidad.

En el último tiempo la herramienta Co-hadoop[11] se ha estado desarrollando para tomar ventaja en el ámbito de colocación no aleatoria de los datos, esto se realiza con el fin de poder acelerar la evaluación de cierta clase de consultas. Pero a pesar de este esfuerzo, los sistemas antes mencionados se centran en los aspectos técnicos y dejan toda la responsabilidad de elegir la ubicación de los datos al administrador de la base de datos.

4 Propuesta de solución

Al momento de escribir una propuesta de solución se presentan una serie de interrogantes, las cuales están asociadas a las variables, restricciones e interacción de estas dos componentes para hacer entrega de una solución sólida y cuya implementación sea factible. A pesar de que cualquier alternativa no podrá encontrar la solución óptima del problema, debido a la complejidad del mismo, por lo menos se puede obtener una solución que se aproxime de forma razonable al óptimo.

Para poder presentar en forma clara la propuesta diseñada y confeccionada, ésta se explicará en forma de componente-solución-unión. En primer lugar se definirá que es lo que hace cada componente, detallando como realiza sus funciones y la lógica que maneja internamente. Luego se describe su aporte a la solución, indicando que aspecto del problema planteado se espera resolver con su uso, esta sección también establece el nexo entre lo que el módulo hace y lo que se busca resolver con él. Para terminar con la unión, se describe cómo éste componente conforma parte de la estructura, que permite de forma conjunta la resolución del problema final, explicando la retroalimentación existente con los demás módulos. Este problema tiene un grupo de variables y restricciones difíciles de apreciar, pero están involucradas dentro de los subsistemas del problema, que se ha propuesto resolver en este documento.

Para comprender y poder visualizar de mejor manera los componentes se presenta la Figura 5. En esta ilustración se aprecian cinco grandes componentes entre los cuales destaca el pre-procesamiento de las consultas, la clusterización de las mismas, el almacén de datos, la estructura de datos Árbol Diamante y finalmente el algoritmo de particionamiento de grafos. Otro componente que se encuentra en la ilustración pero no se comentará es el stream de datos, que es simplemente el canal por el cual llegan de forma fluida los datos pero sin sobrepasar la tasa de llegada de los mismos para de esta forma evitar que se viole una de las restricciones. Esta restricción indica que hay cierta tasa de llegada de datos que involucra tiempos prohibitivos para resolver este problema de forma dinámica, dado que el tiempo necesario para procesar los datos será mayor que el tiempo en que se reciban nuevos datos.

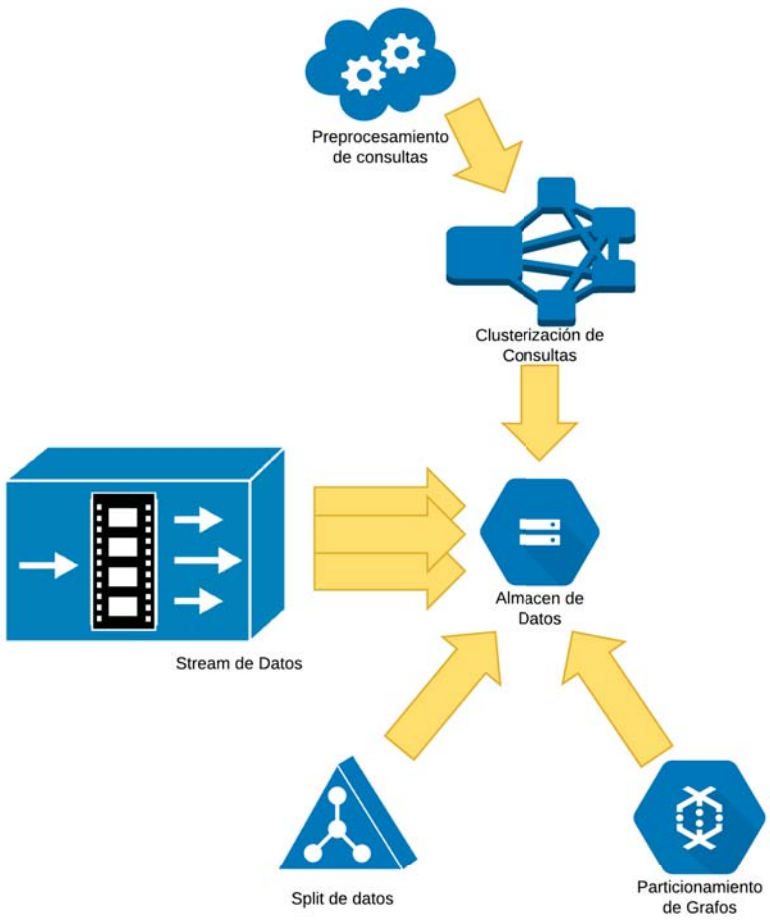


Figura 5: Diagrama Diseño de la Solución

A continuación se presentarán cada uno de los componentes presentes en la imagen, como se hizo referencia en los párrafos anteriores.

4.1 Generación de consultas

Utilizando el benchmark TPC-DS[4], se procedió a la generación automática de consultas SQL, aquellas están diseñadas para trabajar sobre el modelo de datos de TPC-DS. El conjunto de consultas resultante se encuentra en lenguaje SQL Server. Al tener este conjunto de consultas SQL, que conformarán la carga de trabajo del cliente para las pruebas, se hace necesaria la búsqueda de alternativas para separar lo que es propio del modelo

de datos usado y lo que es SQL nativo, además de palabras reservadas del motor de base de datos que la organización utilice.

Con la finalidad de hacer este proceso más genérico para cualquier modelo de datos, se realizó el estudio de las funciones SQL que sirven para pivotear y totalizar los datos en los sistemas de gestión. Estas palabras serán distintas dependiendo del motor de base de datos que use la organización, esta herramienta también hace diferencia en la sintaxis que se debe utilizar al escribir la consulta, por ejemplo, dicta donde se puede utilizar una subconsulta. Es por esto que no haremos uso de un autómata formal para procesar el interior de la consulta, sino que las analizaremos, reconoceremos patrones, y extraeremos las palabras objetivo que pertenezcan a la lógica de negocios del cliente.

Dependiendo de la calidad de la solución otorgada, se definirá si es una mejor alternativa basarse en un template para un motor de bases de datos en particular. De esta forma será responsabilidad del usuario buscar la forma de procesar su carga con otro motor en esta herramienta. Para las pruebas del programa, recordar que se utilizará el benchmark TPC-DS, motivo por el cual las pruebas se basarán en el lenguaje SQL Server, sobre Windows Server 2012 Estándar.

En primer lugar el preprocesador carga las consultas en texto plano, cada consulta es leída y almacenada como un String. Luego se define un conjunto de palabras reservadas, este conjunto funcionará como stop words en la siguiente etapa del marco de trabajo del sistema. Finalmente se quitarán en un proceso recursivo, haciendo uso de expresiones regulares para detectar funciones, números y todo tipo de expresiones, con la finalidad de no perder en ningún caso una palabra clave del modelo de datos en una consulta y evitar el procesamiento de una palabra restringida de SQL, operadores lógicos y todo tipo de palabras del léxico propio del lenguaje.

El objetivo de esta operatoria es cargar una colección de términos en un diccionario. Cada término está en N consultas y en cada una de estas está N_i veces. Esto demanda que cada término en nuestro diccionario se represente con un vector de largo N, donde i=1,2...,N; cada una de estas posiciones representa la cantidad de veces que se repite el término en la i-ésima consulta. Esto es fundamental para poder representar vectorialmente la carga de trabajo.

4.2 Clusterización de consultas

Una vez que se tiene cada término desagregado de las consultas, pero guardando cuantas veces acontece en cada una de ellas, se puede proceder a analizar y relacionar las

consultas en función de los términos. Para hacer esto posible se requieren dos cosas; una forma adecuada de cuantificar la relevancia de un término para una consulta, tomando en consideración que se trabaja con texto; y una métrica para poder comparar este conjunto de relevancias que se presentan en una consulta, con los términos más relevantes que se presentan en las demás.

El primer problema descrito se resuelve con minería de texto, es un proceso que busca desprender de una gran cantidad de documentos, la información que trasciende a lo que se podría tener en forma directa en una base de datos. Esto es, buscar cadenas de implicaciones causales las cuales pueden indicar nuevas relaciones, hipótesis y otras formas de conocimiento.

En general, la minería de texto requiere que los documentos objetivo sean representados de una manera que facilite su estudio. La representación vectorial de documentos, busca extraer los términos de todos los documentos y formar una matriz que de N x M, representando N términos provenientes de M documentos. Luego se evalúa la relevancia de cada término en cada documento.

De todas las técnicas para realizar este trabajo sobresalió la Frecuencia de los Términos y Frecuencia de Documentos Inverso (TF-IDF), es un método para evaluar qué tan importante es una palabra dentro de un documento. Este modelo es una forma de representar las características de un texto en forma vectorial, la cual permite centrarse en los elementos que concierne analizar del texto.

Esta técnica parece especialmente atractiva para trabajar con textos similares a consultas SQL. Generalmente en los documentos de este tipo, los términos que rara vez se presentan suelen ser más informativos y determinantes que los términos que ocurren con mayor frecuencia entre textos. Es común que una misma tabla sea utilizada por todas las consultas, pero cuando solo dos consultas hacen uso de una tabla poco común, la relación de estas consultas resulta mucho más estrecha por estar vinculadas por una tabla inusual que por la más popular que aparece en un gran número de documentos. Es por esto que TF-IDF toma en cuenta no sólo el término aislado en un documento, sino también el término y su relevancia dentro de la colección de documentos como un todo. Además hace uso de una escala logarítmica con la cual un término que aparece 100 veces más que otro, está lejos de ser un factor 100 veces más relevante a la hora de relacionar los textos.

El uso de TF-IDF otorga una matriz que representa la relevancia de cada término en cada documento, tomando en cuenta el uso de este término en el resto de los documentos. La relevancia viene dada por:

$$T_f(t,d) = \sum_{x \in d} fr(x,t)$$

Suma cuantas veces está el término t en el documento d.

$$fr(x,t) = \left\{ \begin{array}{l} 1, x \in t \\ 0, x \notin t \end{array} \right\}$$

Si el término t es igual a la palabra x presente en el documento, vale 1.

$$Idf(t) = \lg \left(\frac{|D|}{1 + |\{d : t \in d\}|} \right)$$

$|\{d : t \in d\}$: es el número de documentos en que t aparece al menos 1 vez.

D : Conjunto de documentos

$$TF - IDF = TF(t,d) x IDF(t)$$

Luego estos vectores son normalizados, para evitar un sesgo hacia los documentos con mayor cantidad de términos, que configurarían un vector más grande que es potencialmente más fácil de relacionar con el resto. Al tener todos los vectores normalizados no hay imparcialidades de este tipo.

Al contar con esta representación de las consultas, se puede proceder a agruparlas según alguna métrica. Es en este sentido que se ha investigado cuál es la forma más apropiada para medir las distancias entre los vectores.

De los algoritmos estudiados sobresalió el clustering jerárquico ascendente. Este método de análisis de grupos busca formar subclusters, que se aglomeran formando nuevos grupos, configurando finalmente una jerarquía de grupos. El método que es preferible utilizar para medir la similitud entre clusters se denomina complete linkage. Este define que la distancia entre dos clusters viene dada por la máxima distancia entre sus componentes.

Se consideró que este algoritmo es superior no debido a la precisión de la clasificación entregada, sino a que el conjunto de valores recibidos al ejecutar el programa tienen una varianza mucho menor que sus demás homólogos derivados de k-means clustering. En los otros casos la ubicación inicial aleatoria del centroide tiene un efecto mucho más decisivo en el resultado final entregado por la herramienta de agrupamiento. Es más importante para este sistema que la medición proporcione un valor poco variable, a tener más relevancia en la reducción de la interacción entre los clusters. Se cree que la relevancia de la métrica es despreciable si se compara con el impacto que tiene la calidad de la representación de

las consultas entregada por la minería de texto TF-IDF.

Los primeros ejercicios de extracción de características desde las consultas, se hicieron tratando de podar lo menos posible los datos presentes en ellas. De esta forma al haber una función de SQL, se captura con una expresión regular y se rescatan las columnas y relaciones que la consulta busca obtener. A esta extracción se le denomino Tabla – Columna, ésta buscó procesar las consultas como si se trataran de cualquier otro tipo de documento, quitando el menor número de términos posible.

Al evaluar el resultado de la clusterización de las consultas utilizando este principio, el equipo se percató de que el lenguaje SQL interfiere bastante en la clasificación de los documentos. Para dar con este hecho se requirió de la construcción de un programa, que funcionó como evaluador de la calidad de la clusterización. La métrica para evaluar si el agrupamiento de dos consultas fue certero o no fue observar las tablas en común que dos consultas utilizan. De esta forma, si dentro de un cluster las consultas poseen muchas tablas en común, y en relación a las consultas afuera de su cluster, las consultas tienen pocas tablas en común, se puede inferir que la calidad de la clusterización fue buena.

Al utilizar este evaluador con los clusters, nos percatamos de que la clasificación no fue correcta según el algoritmo. Estudiando las relaciones, se puede ver que las tablas muy relevantes en el modelo de datos fueron demasiado determinantes a la hora de clasificar las consultas. Analizando detenidamente el problema, se hizo evidente que si bien la minería utilizando TF-IDF dada su escala logarítmica, evita crear un sesgo hacia stepdocumentos largos; existe el problema de que una sola tabla con muchas columnas, crea muchas dimensiones distintas en el vector, que finalmente repercutirán severamente en la clasificación de la consulta. Por esta razón, una tabla muy predominante en el modelo, no significa una dimensión, sino varias según las columnas de la tabla que la consulta ocupe, creando un sesgo hacia un cluster en específico. Este hecho quedará demostrado en la sección de pruebas.

Para solucionar este problema, se determinó otra forma de caracterizar las consultas, esta vez solo evaluando las tablas que las consultas ocupan. De esta forma no se crean varias dimensiones por cada tabla, terminando con el sesgo hacia tablas grandes, con muchas columnas relevantes.

Al evaluar la clusterización utilizando solo las tablas, el evaluador muestra que esta vez los clusters si poseen un conjunto importante de tablas en común, esto es, las consultas están mejor relacionadas. Además, ya no existe un cluster gigantesco, que aglomere un gran conjunto de consultas según una única tabla grande, esta vez al separar en 10 clusters, el grupo más numeroso comprende 32 consultas, utilizando el principio anterior el cluster

más grande contaba con 58 consultas. Téngase en cuenta que el conjunto de prueba usado es de 99 consultas.

4.3 Almacén de datos

Dentro de todo sistema cuya tarea sea la administración de un conjunto de datos, como lo es este proyecto, se tiene un lugar donde se guardan todos los datos que ingresan al sistema. Cada almacén de datos se encarga de guardar la información de la manera más conveniente según las futuras peticiones que recibirá del programa, esto incide de forma directa en el rendimiento general que el sistema tendrá. Para evitar que el rendimiento de la solución se vea afectada se optó por un almacenamiento de tipo estructurado que permite definir una forma de guardar, modificar y eliminar los datos que contiene dicha estructura.

A pesar de que el funcionamiento que tiene este componente de la solución es sencillo, requiere una arquitectura de diseño compleja que permita mejorar el rendimiento, además de aumentar la capacidad de almacenamiento para ciertas circunstancias que se pueden presentar tanto en el ambiente de pruebas como en el de producción. De esta forma se estableció la solución a través de grid computing, esta tecnología se puede describir como la nueva computación distribuida la cual permite la interacción de forma coordinada de distintos componentes.

Este tipo de infraestructura permite que se pueda contar con un componente, el cual se denominará en el resto del trabajo como "Balanceador de cluster", éste elemento es el encargado de tener las referencias a cada cluster del sistema, además de ser quien se encarga de recibir los datos provenientes desde el exterior, verificando si se trata de consultas o datos. Esta distinción es necesaria dado que permite discriminar las acciones futuras que se deben tomar sobre los clusters, esto se debe a que los datos son enviados a cada uno de los cluster de forma simultánea. Sin embargo, las consultas que llegan deben ser dirigidas a un cluster en específico, para esto el balanceador es capaz de decidir a cuál cluster pertenece la consulta que acaba de llegar y enviarla al cluster correspondiente para que se efectúe la acción objetivo sobre la consulta.

Otro elemento importante dentro de este almacén de datos son los cluster, que además de contener sus respectivas consultas y saber los servidores que poseen todos los datos para responder a las peticiones; es la entidad encargada de administrar la ejecución o adhesión de una consulta, esto dependerá del estado en ese instante de la consulta entrante.

Finalmente llegamos al elemento que ocasionó todo este estudio, los servidores. Estos pertenecen a un cluster y son los que contienen una estructura de datos que consigue dis-

tribuir la carga de la forma deseada, la cual se explicará en la sección 4.4. Los servidores, dependiendo del estado en que se encuentre la ejecución del sistema, incluso pueden llegar a almacenar un solo fragmento y en el peor de los casos una parte de un fragmento, esto se profundizará posteriormente cuando se termine de explicar el funcionamiento de cada una de las partes.

4.4 Estructura de datos: Árbol Diamante

Al momento de diseñar la solución se observó una característica particular del problema, que es la relación existente entre los datos. El estudio de esta relación permite que cada vez que se inserta un dato, este se relacione y agrupe con un conjunto de ellos. Esto se ve claramente al momento de responder una consulta, dado que el conjunto de datos debe pasar por un proceso de selección y validación para verificar la pertenencia de cada dato almacenado a la respuesta de una consulta en particular.

Para mejorar la velocidad de respuesta de la solución planteada en este trabajo fue necesario buscar una forma de organizar esta información y hacer posible un acceso más eficaz a algún conjunto de datos en cuestión. Considerando que la distribución de información busca dar respuesta a un conjunto de consultas que utilizan un mismo modelo de datos, resulta evidente que una buena forma de disminuir el espacio en disco y solapar los recorridos de búsqueda es mezclar los conjuntos de solución. Esta idea tiene como consecuencia que los fragmentos que constituyen una respuesta se multiplicarán a medida que se ingresen nuevos datos, sin embargo se debe tener en cuenta que lo que se desea minimizar es el movimiento de datos, de tal forma que es más importante tener muchos fragmentos fáciles de distribuir que pocos que tengan un gran peso y por si solos saturen la red.

Es por este motivo que surgieron pequeños grupos de datos llamados Fragmentos, estos simplemente consisten en un grupo de datos que tienen una alta relación dado que todos satisfacen un mismo conjunto de consultas. Se pueden encontrar casos en los cuales un grupo de datos satisface un grupo de consultas, estos fragmentos se denominarán de nivel N, en donde N es la cantidad de consultas que puede satisfacer el fragmento, así mismo hay fragmentos de primer nivel que solo responderán a una única consulta. De esta forma la manera de contestar a una consulta, será navegar por este conjunto de resultados solapados hasta encontrar a todos los constituyentes de una solución. Se debe agregar que los fragmentos representan la configuración total de las respuestas y son los únicos que pueden dar respuesta a una consulta.

El orden en que se llega a los fragmentos no es relevante en cuanto a cómo se construye

la respuesta. Una consulta solo se ejecutará una vez los fragmentos que necesita lleguen a su servidor principal, el cual corresponde a la máquina con mayor cantidad de datos relevantes para ella. Se reconocerá esta máquina comparando la suma del peso total de los fragmentos relevantes entre los servidores involucrados con la consulta. En conclusión, la manera de recorrer el árbol, al cambiar la configuración de sus nodos, no tiene por qué llevarnos a los nodos en un mismo orden; este orden puede ser aleatorio y solo importa llegar a todos los fragmentos con posibilidad de satisfacer a la consulta.

Bajo el contexto anterior se presenta una jerarquía intrínseca que obliga a que los fragmentos que responden menos consultas estén sobre los que responden más, esto se puede apreciar en la siguiente representación

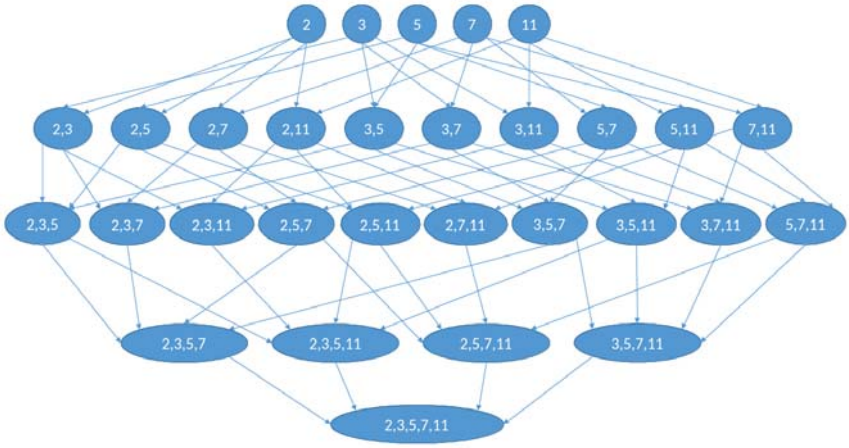


Figura 6: Estructura de árbol diamante

4.4.1 Búsqueda a través del árbol

Este proceso de búsqueda se caracteriza por no tener claridad de los niveles, ni de los nodos que se desean encontrar, mucho menos de la cantidad de fragmentos que deberemos analizar para encontrarlos. Para dar con los datos que configuran la respuesta de una consulta se realiza un recorrido exhaustivo sobre los descendientes del fragmento de primer nivel que representa a la consulta requerida, haciendo extracción de los datos de sus hijos hasta que no quede ninguno.

La búsqueda se basa en un proceso recursivo, en donde se desconoce la cantidad de fragmentos a encontrar dado que su formación depende de las relaciones entre los datos y

la carga de trabajo. La complejidad de este algoritmo se mantiene igualmente controlada, ya que a partir de un nodo conocido del árbol se rescatarán todas las combinaciones posibles para dar una respuesta completa a la consulta solicitada.

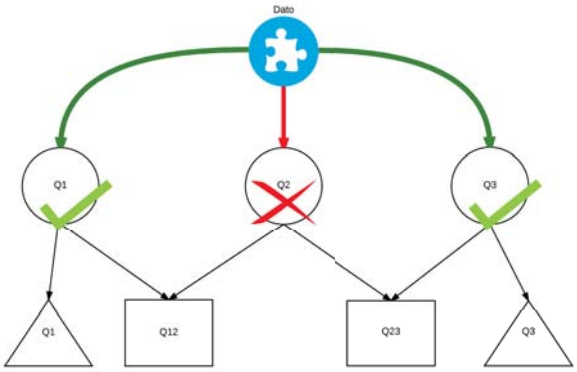
4.4.2 Inserción a través del árbol

Al insertar un dato hay que tener en cuenta que este puede satisfacer un número indeterminado de consultas. El conjunto a recorrer se puede acotar estudiando las raíces del árbol, validando sobre qué fragmentos se debe realizar posteriormente la extracción. Si el dato a insertar no satisface ninguna de las consultas pertenecientes a la raíz, evidentemente no es necesario recorrer el árbol y el dato se puede descartar al no ser relevante para la carga de trabajo de la organización.

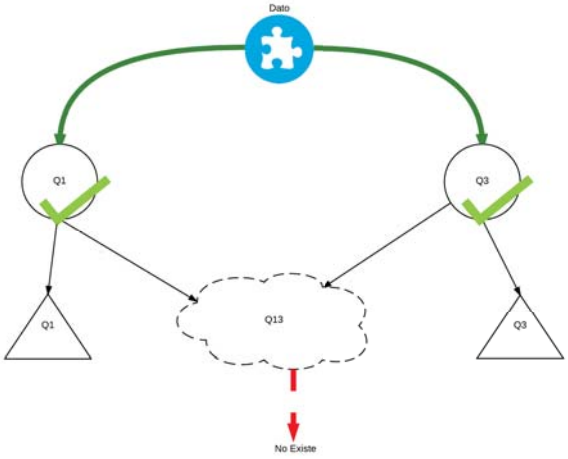
Si el dato satisface a una consulta, éste debe ingresarse en la raíz asociada a la consulta. Sin embargo, si el dato satisface más de una consulta, el fragmento que le corresponde es de un nivel superior. Este fragmento intermedio puede existir previamente en el árbol o no, de no existir debe crearse para su almacenamiento y vincularse con el resto de los nodos del árbol. Para encontrar su ubicación dentro de la estructura de datos, se recorren los nodos a partir de las raíces de las consultas a las cuales el dato responde; luego, se recorren sus hijos como posibles padres directos del dato a insertar. Si alguno de los hijos responde a todas las consultas que el mismo dato responde, allí se insertará el dato; si responde a un subconjunto de las consultas que el dato responde, se almacena como posible padre del nodo buscado y se elimina el padre de este ya que el nuevo candidato responde a todas las consultas del dato y a una más, acercándonos al nodo que responde a todas las consultas asociadas al dato.

Una vez se encuentra el fragmento que satisface todas las consultas del dato, éste se inserta en el fragmento. Si al recorrer se llega a un padre que satisface un subconjunto de las consultas, pero no tiene más hijos; éste se guarda como un padre directo. Lo que sucede en estos casos es que no existen los fragmentos intermedios, ya que no hay otros datos que satisfagan todas esas consultas, ni un subconjunto cercano a ellas. Para estos casos se crea un enlace directo entre el padre existente y un fragmento nuevo para el dato a ingresar. Una vez lleguen estos fragmentos intermedios faltantes, el padre actual se asigna como padre de ellos y se desvincula del fragmento actual, el nuevo padre de este será el fragmento intermedio a generar en el futuro. De esta forma se mejoran los tiempos de respuesta y se mantiene la lógica del árbol diamante.

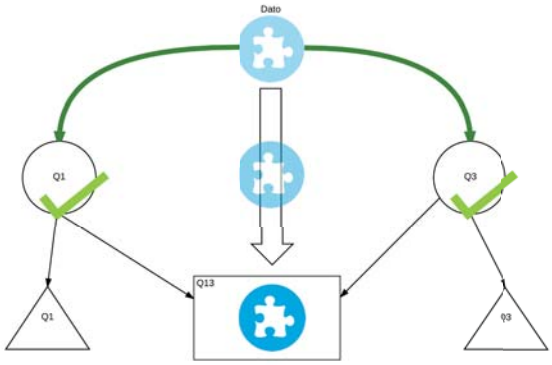
Algorithm 1 Inserción a través del árbol



$D \leftarrow$ Dato nuevo
$lista\_padres \leftarrow \phi$
while $Q_i \in Q$ **do**
  if $\Xi(D, Q_i)$ **then**
    add_fragmento(lista_padres, Q_i)
  end if
end while



if lista_padres $= \phi$ **then**
  delete(D)
  return
end if
if size(lista_padres) $= 1$ **then**
  add(pop(lista_padres, D))
  return
end if
$F \leftarrow$ lista_padres
lista_hijos $\leftarrow \phi$
while $F \neq \phi$ **do**
  $F_i \leftarrow$ pop(F)
  add(F, F_i->getHijos())
  if F_i->getConsultas() $\subset$ getConsultas(lista_padres) **then**
    add(F_i, D)
    return
  else if getConsulta(lista_padres) $\subseteq$ F_i->getConsultas() **then**
    add(lista_hijos, F_i)
  end if
end while

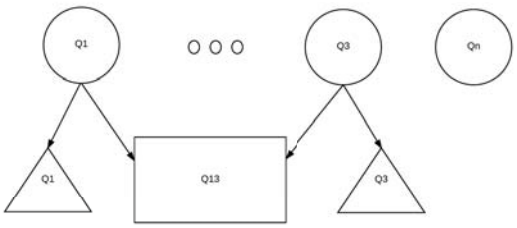


$F_{tmp} \leftarrow$ new Fragmento()
add(F_{tmp}, D)
add_padres(F_{tmp}, F)
add_hijos(F_{tmp}, lista_hijos)

4.4.3 Ingreso de consultas en el árbol

Además de llegar datos, la solución también requiere que lleguen consultas como estímulo para el árbol. Al recibirse una consulta por primera vez, se empiezan a ingresar los datos necesarios para contestarla dentro del árbol. Luego, la segunda vez que llega la consulta, el árbol interpreta que se está solicitando la información guardada entre la primera y segunda recepción de la consulta, ejecutando la funcionalidad descrita como búsqueda dentro del árbol. Finalmente, la tercera vez que llega la consulta, se procede a eliminar a la consulta del árbol junto con todos los datos que únicamente sirven a ésta. De esta forma según llega la consulta, el árbol adopta distintos estados para recibir nuevos estímulos.

Algorithm 2 Ingreso de consulta en el árbol



$Q_{tmp} \leftarrow$ new Query(Nueva Consulta)
add(Q, Q_{tmp})
return

4.4.4 Eliminación de una consulta dentro del árbol

Con el tercer arribo de una consulta en el árbol, se procede a la eliminación de sus datos. Recordar que los fragmentos de un nivel mayor a uno, poseen datos que son relevantes para más de una consulta. Este hecho es importante a la hora de abordar la necesidad de eliminación, debido a que una mala gestión de los fragmentos traerá problemas evidentes a la integridad de la información que se entregará en el futuro. Para hacer más rápido y seguro este proceso, se definió que los fragmentos cuenten con una referencia a sus hijos y otra a su padre. De esta forma se puede aplicar la eliminación con un solo recorrido al árbol a partir del nodo base de la consulta a eliminar.

La forma de realizar esta acción corresponde a mover un nivel hacia arriba a todos los fragmentos de nivel mayor a uno, desvinculándolos de la consulta eliminada. Por consiguiente, los datos de la consulta solapados a otras respuestas no se perderán y sus referencias seguirán siendo correctas. Una incidencia que se puede presentar es que ya exista un nodo que represente a alguno de los fragmentos movidos. Esto es, si al eliminar una consulta de un fragmento, en el nivel superior ya existe un fragmento que almacene la relación entre las consultas restantes; en este caso los dos fragmentos se juntan en uno nuevo,

que guarda las referencias de ambos hacia los demás fragmentos. Los datos pertenecientes al fragmento de nivel 1 de la consulta son los únicos datos realmente removidos del sistema.

4.4.5 Partición de grafos

Cuando uno de los árboles formados es sobrecargado, se divide su carga valiéndonos de un nuevo servidor auxiliar. Con el objetivo de que la división de la estructura de datos sea cercana al óptimo se utiliza una herramienta de particionamiento de grafos que dividirá los fragmentos entre las nuevas máquinas a disposición. El producto utilizado se llama Metis[1], este se basa en k-way multinivel, algoritmo que intenta agrupar nodos minimizando la comunicación entre grupos. Se eligió esta aplicación debido a su gran rapidez y su facilidad de adaptación a este sistema.

Al partir un árbol se copian todos los fragmentos de primer nivel en ambas particiones. Esto solo se trata de un vínculo simbólico para no perder las referencias entre los fragmentos, siendo estos vínculos necesarios para las operaciones antes comentadas. Por lo tanto, un árbol puede estar almacenado en más de un servidor, y los procesos de inserción, búsqueda y eliminación; pueden comprometer a varias máquinas.

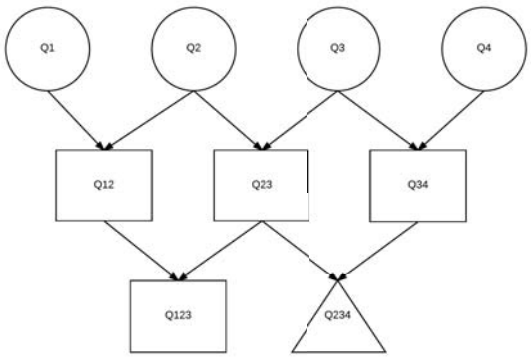
4.5 Cálculo de complejidad

Todas las operaciones anteriores se pueden ver un poco engorrosas y difíciles de entender, por lo que se puede generar un pensamiento de que la complejidad de implementación y los tiempos de respuestas no sean los mejores y puedan afectar de forma considerable el desarrollo de todo este modelo que se está planteando. Para demostrar que esto no es así, se ha incluido el cálculo de la complejidad que se basará en la comparación con otras estructuras de datos similares para que de esta forma sea más fácil entender todo lo que está pasando al momento en que se gatillan estos procesos que modifican la estructura del árbol diamante.

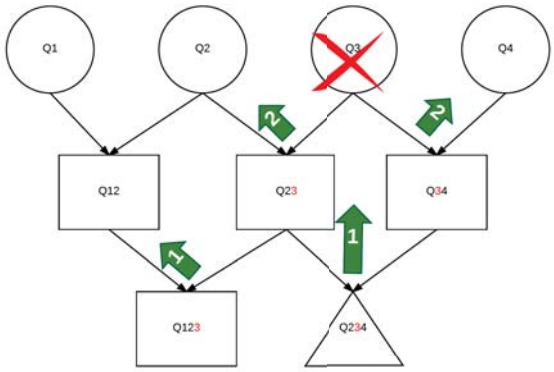
4.5.1 Cálculo de Cantidad de hijos

Para poder entender de una forma clara de donde vienen los valores reales asociados al cálculo de la complejidad, y de esta manera evitar la necesidad de tratar con una serie de ecuaciones que pueden traer confusión; se ha decidido realizar una comparación con el árbol B ya que existe similitud en cuanto a los recorridos y búsquedas. Estas funciones se ocupan en la inserción y eliminación del árbol diamante.

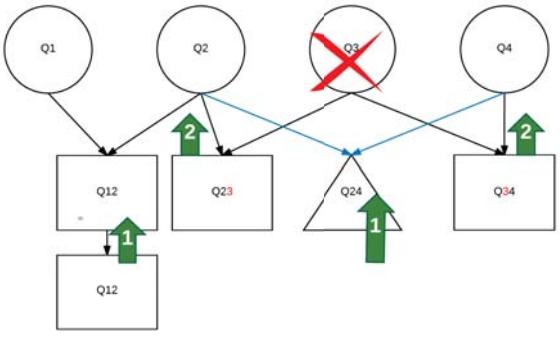
Algorithm 3 Eliminación de una consulta dentro del árbol (Parte 1)




```
Q ← Consulta_eliminada
F_q ← Q->getFragmento
cola_hijos ← F_q->getHijos()
pila_recorrido ← φ
while cola_hijos ≠ φ do
    F ← pop(cola_hijos)
    push(pila_recorrido, F->getHijos)
    push(cola_hijos, F->getHijos)
end while
```

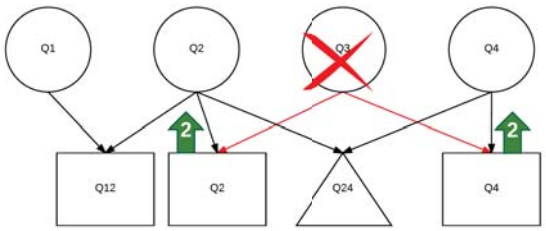



```
while pila_recorrido ≠ φ do
    F ← pop(pila_recorrido)
    F->remove(Q)
    if ! NuevoNivel(F) then
        SubirNivel(F)
    end if
end while
```

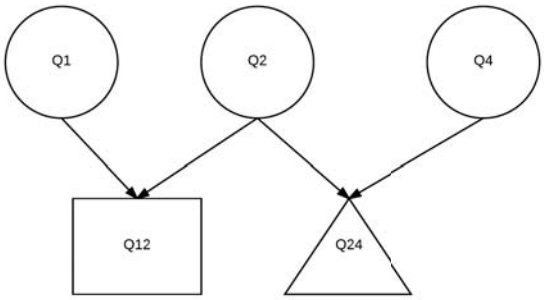



```
function PUEDESUBIR(Fragmento:F)
    cola_padres ← F->getPadres()
    while cola_padres ≠ φ do
        F_padres ← pop(cola_padres)
        if F_padres->getConsultas() = F->getConsultas() then
            return F_padres
        end if
    end while
    return Null
end function
```

Algorithm 4 Eliminación de una consulta dentro del árbol (Parte 2)




```
function NuevoNivel(Fragmento:F)
    F_tmp ← puedeSubir(F);
    if F_tmp ≠ Null then
        return True
    end if
    return False
end function
```




```
function SubirNivel(Fragmento:F)
    F_tmp ← puedeSubir(F);
    if F_tmp ≠ Null then
        add(F_tmp, F->getDatos())
    end if
end function
```

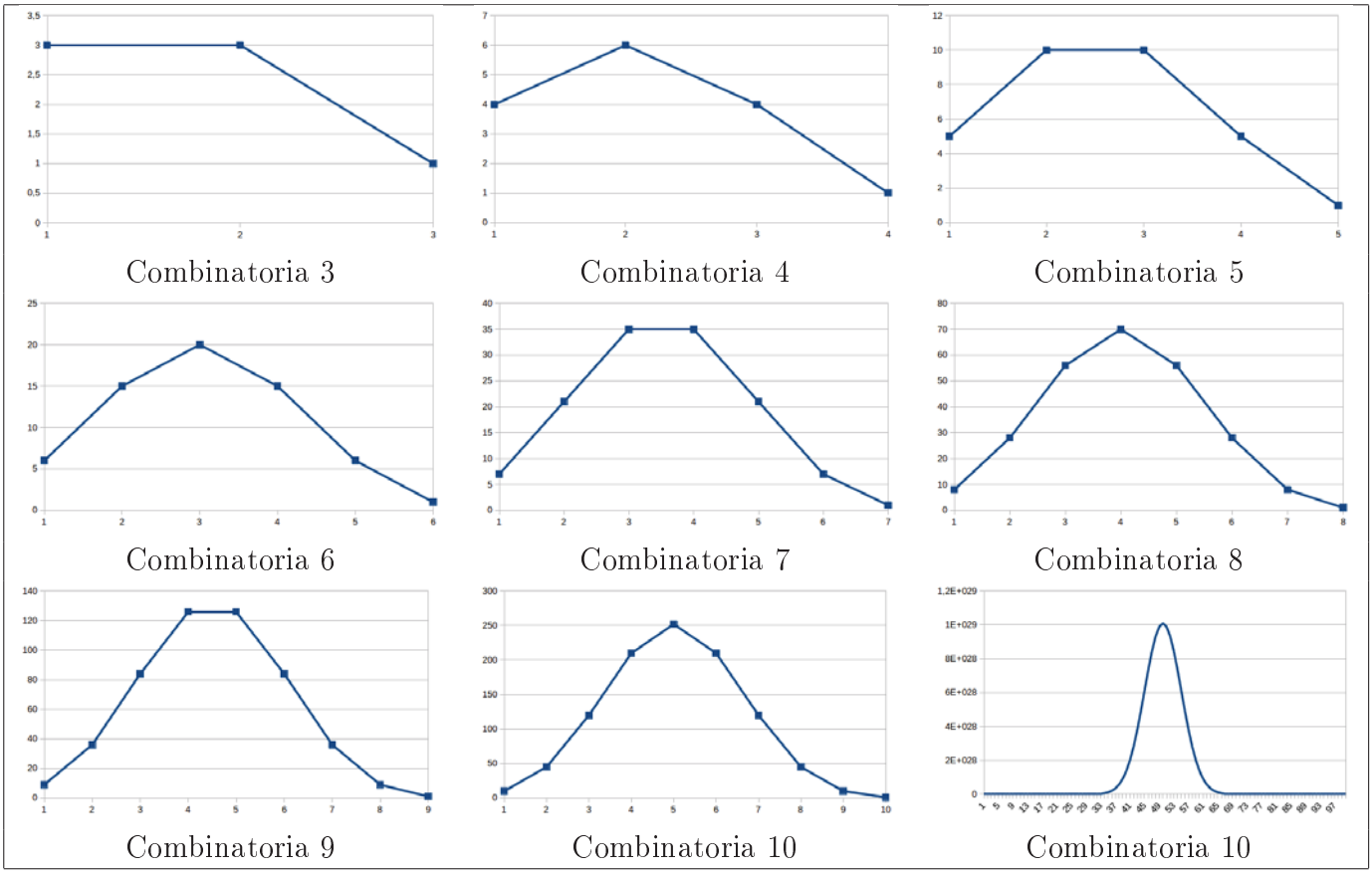
Para comenzar con esta demostración del cálculo de complejidad del árbol diamante, recordemos que esta estructura de datos permite mantener dentro de sí misma la intersección de un conjunto de reglas, para el caso estudiado estas reglas son las consultas. Para indagar sobre este conjunto de intersecciones posibles, la herramienta matemática nos ofrece la función de combinatoria, esta se define como sigue: dottedtocline

$$C(n,r) = \frac{n!}{(n-r)!r!}$$

Desde esta ecuación se pueden obtener la cantidad de hijos máximos que se pueden tener para un nodo en su proximo nivel, en donde n es la cantidad de reglas que se están procesando en el árbol diamante y en donde r es el próximo nivel. Esto hace que la cantidad de hijos por nivel varíe nivel a nivel lo que hace que la comparación con un árbol B se torne complicada. Es por esto que se tomará como referencia el nivel que genera más nodos hijos. Para obtener dicho nivel se puede observar en los gráficos de combinatoria.

En el gráfico se puede observar que a partir de $r_m = \frac{n}{2} \pm 1$ esto depende de si es par o impar. Se encuentra el nivel que posee más nodos hijos por nivel. A partir de estos antecedentes se definirá la cantidad de nodos hijos como

$$C(n,r_m) = \frac{n!}{(n-r_m)!r_m!} = \frac{n!}{(n-\frac{n}{2}\pm 1)!(\frac{n}{2}\pm 1)!}$$



Cuadro 1: Gráficos de combinatorias

4.5.2 Complejidad: Insertar una consulta

Para insertar una consulta no se recorre ni se modifica la estructura del árbol, por lo que la inserción de una consulta en el árbol diamante tiene un orden $O(1)$.

4.5.3 Complejidad: Insertar un dato

Al insertar un dato cambia la situación, dado que ahora es necesario recorrer el árbol. Como es sabido, el Árbol B requiere para encontrar un elemento dentro de su configuración, incurrir en un costo de $\log_b N$, en donde b es la cantidad de nodos hojas que tiene por nivel y N es la cantidad de datos insertados en el árbol. Bajo esta premisa se puede afirmar que el orden que tiene insertar un dato en el árbol diamante esta descrito por la siguiente ecuación:

$$O\left(\log_{C(n,r_m)} \sum_{i=0}^{R} C(n, i+R)\right)$$

En donde R es la cantidad de niveles que tiene el árbol, pero dado que para un $n > r$, no se pueden realizar combinaciones, el árbol nunca tendrá más niveles que la cantidad de consultas insertadas, por lo que se puede decir que R también representa la cantidad de consultas insertadas.

4.5.4 Complejidad: Eliminación de una consulta

La eliminación de una consulta al igual que la inserción de un dato conlleva el recorrido del árbol, esta se puede describir de la misma forma pero con la variante de que ya no se suma la cantidad de consultas insertadas sino que al recorrer los elementos buscados, estos son modificados. Solo se debe realizar el recorrido exhaustivo para encontrar y alterar las relaciones de todos los fragmentos relevantes para la consulta eliminada. El orden queda simplemente dado por

$$O\left(\log_{C(n,r_m)} \sum_{i=0}^{R} C(n, i)\right)$$

4.5.5 Complejidad: Ejecución de una consulta

La ejecución de una consulta conlleva un costo similar al de la eliminación de la misma, esto debido a que se recorre la totalidad de los fragmentos involucrados en dicha consulta, sin embargo, en este proceso no se reestructura el árbol por lo que el orden queda de la siguiente forma

$$O\left(\log_{C(n,r_m)} \sum_{i=0}^{R} C(n, i)\right)$$

5 Arquitectura de la solución

A partir de este punto se comenzará a describir la forma en que cada uno de los componentes que se describieron anteriormente, interactúan entre ellos y llevan a cabo la labor de reducir el tráfico dentro de la red. Para poder realizar esta labor se explicarán en dos partes las tareas realizadas, la primera de ellas será la explicación de la arquitectura en general, describiendo a grandes rasgos las interacciones, posteriormente se procederá a explicar el ambiente de simulación y las consideraciones que se tomaron en cuenta al momento de implementar la arquitectura. Todo esto se hará para poder mostrar de la mejor manera posible la forma en que se obtuvieron los datos y las métricas de la sección siguiente en donde se analizarán los datos.

5.1 Descripción de la arquitectura

Nuestra entrada inicial son las consultas de la organización objetivo. El preprocesador transforma las consultas SQL en un vector para ordenar los términos dentro de ella con el número de veces que cada uno de estos se repite, quitando todos los elementos que no sean relevantes de medir. Esta tarea es aprovechada por el clusterizador de consultas, que recibe este vector y lo usa para cargar una matriz de representación a través de la cual será posible describir las consultas de forma cuantitativa y realizar mediciones. Valiéndose de la matriz generada, el clusterizador hace uso de clustering jerárquico ascendente, este método de análisis de grupos busca formar subclusters que se aglomeran formando nuevos grupos. La similitud entre clusters se determinó por la máxima distancia entre sus componentes. En esta etapa se debe elegir el número de clusters a utilizar en la solución final.

Según el número deseado de clusters a utilizar, se aglomerarán los grupos de consultas similares hasta conformar el número de grupos escogido, estos grupos serán la base para construir el árbol diamante. La estructura de datos está diseñada pensando en que cada cluster sea contenido en un servidor diferente, entiéndase que como resultado de las operaciones anteriores, cada cluster es un conjunto de consultas que usa un grupo de tablas significativo en común. Cada servidor inicial funcionará como un almacén de datos independiente del resto y será la máquina raíz del cluster. En él se cargará el modelo de datos necesario para satisfacer a sus consultas relevantes. Sobre cada servidor y almacén de datos, se ejecutará un árbol diamante diferente. Cuando los datos guardados en la máquina para satisfacer a las consultas del cluster exceden la capacidad del servidor, se ejecuta una partición con la cual se divide el árbol diamante para poder redistribuirse utilizando un servidor adicional, este servidor se añade al árbol del cluster.

Sobre este conjunto de máquinas iniciales opera un balanceador, el cual constante-

mente recibe datos y consultas a través de un stream de datos. Cuando el stream hace entrega de un dato, el balanceador de carga hace envío de este dato a todas las máquinas raíces de los clusters. Estas máquinas son quienes analizan si el dato es relevante para las consultas del cluster que albergan. Si es relevante se cargan al árbol en la máquina que corresponda según las particiones que se puedan haber efectuado, sino es relevante ya que las consultas de cluster no lo utilizarán, se descarta sin almacenarse. Puede ocurrir incluso que el dato no se almacene en ningún cluster. Se decidió que serían las máquinas raíz de los cluster quienes analizarán la relevancia del dato y no el balanceador para que este no se transforme en un cuello de botella en la red.

Cuando el stream efectúa el envío de una consulta, el balanceador la envía directamente al servidor que representa al cluster donde la consulta fue asignada. El árbol diamante de ese servidor mantiene una monitorización del estado de todas las consultas y según este estado actual responde ante la consulta ingresada. Si la consulta entrante llega por primera vez al servidor, el árbol empieza a almacenar los datos relacionados a la consulta, antes de eso los datos entrantes relevantes a la consulta no eran guardados dentro del almacén de datos. Cuando la consulta arriba al servidor por segunda vez, esta debe ser ejecutada, por lo que el árbol procede a recorrer sus nodos buscando los fragmentos relevantes para la consulta por todas las máquinas por donde la estructura de datos se extienda. Los datos son llevados hasta la máquina con los fragmentos relevantes más pesados ya que involucran un mayor movimiento a través de la red, luego la consulta es ejecutada normalmente en ese servidor. Cuando la consulta es recibida por tercera vez desde el balanceador, se procede a la eliminación de la consulta en el árbol diamante que la administra.

El stream se caracteriza por tener una mayor posibilidad de enviar datos que consultas.

5.2 Ambiente de simulación

Para llevar a cabo la comprobación de todo lo explicado hasta el momento en este trabajo, se creó un ambiente de simulación para poder obtener un conjunto de métricas que nos permitan, en la sección de análisis de resultados, explicar los beneficios y desventajas de la propuesta planteada en el presente documento.

5.2.1 Simulación consideraciones

Como se explicaba anteriormente en la arquitectura de la solución, ésta fue simulada bajo un entorno de red que proporciona una serie de datos y consultas, estos son extraídos del conjunto de datos previamente obtenidos del brenchmark TPC-DS al igual que las consultas como se dijo anteriormente en este escrito. La red que se ocupó tiene una

tasa de llegada constante, esto se debe a lo que se mencionó en la descripción del problema.

Otra característica que se tomó en cuenta al momento de generar la red de datos, es que las consultas llegan en menor proporción que los datos, esto se debe a que los datos tienen una mayor frecuencia de llegada que las consultas por los costos de procesamiento que se incurren en los sistemas al momento que estas son ejecutadas.

Dentro de las posibilidades que se encontraban al momento de comenzar la simulación se tenían dos propuestas, una de ellas correspondía a que todos los cluster partieran sobre una misma máquina para reducir el gasto en servidores, pero luego del análisis de costos que será mostrado más adelante se optó por minimizar el tráfico en la red que implicaba el movimiento de clusters a otras máquinas cuando en las que se encontraban inicialmente se estaban quedando sin memoria. Para poder conseguir minimizar el tráfico de red se decidió que cada cluster estará alojado en una máquina inicialmente, esto para evitar el tráfico de red que implica mover al momento de recolocar los cluster en sus nuevas máquinas.

Otro aspecto que se consideró al momento de la simulación fue el hecho de que al momento de particionar un servidor se transportaba por la red un conjunto de fragmentos entre máquinas, y esto obligaba a particionar el árbol diamante. Es por esta razón, que el particionamiento del árbol debe estructurarse de manera lógica para evitar errores en los resultados. Sin embargo, esto no facilita los problemas procedimentales al momento de llevarlo a la práctica.

La siguiente consideración que se tomó en cuenta, fue el hecho de que el particionamiento se realiza con la herramienta Metis. Siempre la partición se realizó con el objetivo de generar dos conjuntos, entre los cuales siempre se elegía el de menor tamaño para mover, esto dado que el de mayor tamaño implicaría un mayor tráfico dentro de la red. En el caso de Metis siempre se particionaba los fragmentos que estaban dentro de una máquina y de todas las máquinas, porque esto implicaría mover muchos más datos, y a pesar de que quizás esta forma de particionar no es la óptima es la que permite mover la menor cantidad de datos al momento de aplicar una partición en los datos.

Para poder entender de una mejor manera esta última consideración se puede pensar en lo que ocurre al particionar el árbol completo. Puede ser una tarea muy compleja y costosa, por el simple hecho de que la cantidad de nodos que puede llegar tener un árbol que está dada por la siguiente fórmula:

$$N_t = \sum_{r=1}^{n} \frac{n!}{(n-r)!r!}$$

Sin embargo, dentro de una máquina hay un conjunto muy reducido de este número,

Cuadro 2: Tabla de costos se máquinas

Capacidad provedor	Proveedor 1	Proveedor 2	Proveedor 3
1 procesador, 1GB de ram y 30GB de memoria	US$10/mes	US$10/mes	US$10/mes
4 procesador, 8GB de ram y 80GB de memoria	US$80/mes	US$80/mes	US$80/mes

que puede llegar a contener a solo un par de ellos, bajo este contexto se tomó dicha decisión.

5.3 Cálculo de beneficio económico

En esta sección se intentará cuantificar el ahorro económico que surge gracias al uso de la computación distribuida. Una máquina centralizada que ejecute en paralelo todos los cálculos necesarios para el uso del sistema comprende elevados requisitos de sistema, necesitando gran capacidad, múltiples procesadores y una memoria RAM elevada. La herramienta presentada en esta tesis permite deshacernos de este despilfarro computacional y tener la capacidad de hacer frente al problema de distribución de carga de trabajo usando máquinas más pequeñas y menos poderosas en cuanto a poder de procesamiento.

Para realizar este cálculo se han estudiado distintos proveedores de soluciones de almacenamiento distribuido muy relevantes en el escenario actual. Para cada uno de ellos se buscó una máquina de 4 procesadores, 8GB de ram y 80GB de memoria como características, este servidor simula la potente máquina que sería necesaria para el procesamiento de los datos de una organización de forma centralizada. Por otra parte, se buscó para el mismo proveedor el costo que tendría el arriendo de un número 8 de servidores de una capacidad 1 procesador, 1GB de ram y 30GB de memoria, evidentemente menor que la máquina anterior.

Se sostiene que la herramienta presentada tendrá una performance igual o mayor al valerse de los 4 servidores más pequeños, en lugar de ejecutar las operaciones de la organización en la máquina de mayor envergadura. Se afirma esto debido a la subdivisión realizada al clusterizar la carga de trabajo del usuario en base a la sección del modelo de datos usada y la complejidad de la estructura de datos utilizada para rescatar los fragmentos. Se puede apreciar el beneficio en el rendimiento del sistema en la sección de resultados.

Los costos de las máquinas encontrados fueron los mostrados en el cuadro 2.

En base a esta tabla se procedió a calcular los costos promedio al arrendar máquinas de capacidad 1 procesador, 1GB de ram y 30GB de memoria y el promedio de arrendar la máquina de capacidad 4 procesador, 8GB de ram y 80GB de memoria, los datos resultaron ser los los mostrados en el cuadro 3.

Cuadro 3: Tabla de Costos Promedio

Capacidad	Promedio	Ahorro promedio especto del otro
1 procesador, 1GB de ram y 30GB de memoria	US$10/mes	US$70/mes
4 procesador, 8GB de ram y 80GB de memoria	US$80/mes	-US$70/mes

Con esta información podemos afirmar que el beneficio económico al usuario es seguro y el ahorro corresponde en promedio de un 87.5 %. Esta diferencia se mantiene aun cuando se escala a máquinas de mejores características, lo que nos dice que a mayor volumen de datos de la organización el ahorro es mayor; este beneficio es aún más relevante considerando la proliferación de datos inherente a los negocios de toda índole en la actualidad.

6 Análisis de resultados

Dentro de este proyecto se puede visualizar una gran cantidad de componentes, pero entre todos ellos resaltan dos que merecen ser analizados mediante métricas. El primero que analizaremos será la clusterización de las consultas, que es uno de los elementos principales de toda esta arquitectura de solución que se ha planteado. El segundo componente que analizaremos a través de éstas métricas será la reducción que tuvo el tráfico dentro de la red, a partir del uso tanto del árbol diamante como del algoritmo de particionamiento bajo la clusterización obtenida por TF-IDF.

6.1 Calidad de clusterización

Este proyecto basó gran parte de la calidad de la solución en lograr la buena subdivisión del modelo de datos. Esta distribución de la base de datos se realizó en función de qué datos serán solicitados en forma simultánea, haciendo un análisis en detalle de las consultas que se efectuarán sobre el sistema. La subdivisión realizada fruto de este análisis fue utilizada posteriormente para determinar la configuración de los árboles diamante, quienes representan finalmente en la solución a los grupos de consultas clusterizados.

La forma de clusterizar utilizada, como se expuso anteriormente, corresponde al clustering jerárquico. Para que la calidad resultante de este proceso sea lo más alta posible, se debía encontrar una buena forma de caracterizar a las consultas y poder realizar mediciones cuantitativas sobre estas dimensiones. Lograr que la medición final de distancias entre las consultas, usada para agrupar a las más cercanas y dejarlas en una misma máquina, tenga efectos prácticos en la performance del sistema; requería que las características escogidas tuvieran directa relación con el objetivo del proyecto. Recordemos que se espera minimizar la saturación de la red, por lo que se quiere agrupar consultas que usen un conjunto de datos estrechamente relacionado.

Dado que consultas no tan relacionadas pueden tener tablas en común, la separación de estas consultas en clusters distintos significa que estas tablas se presentarán al menos en dos máquinas diferentes. Así mismo, dentro de un cluster, hay tablas que no son utilizadas por todo el conjunto de consultas, pueden haber tablas que solo son relevantes para una consulta del conjunto, lo cual no es representativo para agrupar. Lo que se desea minimizar con la buena clusterización es evitar que estos problemas se produzcan. Minimizar la replicación y las tablas no representativas del cluster, son objetivos puramente relacionados a la solución, y son un indicio de que la clusterización logró sus cometidos con el proyecto, llamaremos a la medición de estos objetivos "medidas externas de calidad".

La calidad del resultado de la clusterización jerárquica también puede ser medido ma-

temáticamente. Existen coeficientes que indican que tan bien se logró que las consultas se agruparan con otras consultas similares a ellas. Estos coeficientes realizan este cálculo solo tomando en cuenta los valores de las consultas en el conjunto de características elegido y no tienen consideración acerca de si las dimensiones escogidas sirven o no a los objetivos del proyecto. Llamaremos a este conjunto de coeficientes "medidas internas de calidad".

Lo que se debía lograr caracterizar de las consultas era que parte de su código es más relevante medir para conocer qué elementos del conjunto de datos utilizan. Para lograr esto se usaron los ya antes descritos principios Tabla y Tabla-Columna, la diferencia entre ambos radica en los términos a utilizar como dimensiones en la representación vectorial de las consultas. El principio tabla asume que las columnas no son importantes de medir y que producen sesgos hacia tablas con muchos campos. El principio Tabla-Columna propone que la mejor forma de representar a la consulta es medir la mayor cantidad de codigo posible. Se basa en que como los vectores son normalizados no habrá problemas en que una misma tabla aparezca varias veces por campo.

6.1.1 Resultados de la clusterización

Para probar el sistema se generaron varios conjuntos de clusters distintos, usando los dos principios tabla y tabla-columna por separado. Generando un número de clusters que va de tres a diez. Esto dio como resultado un conjunto de 16 configuraciones distintas para nuestra plataforma. En esta sección del trabajo se realizará un análisis sobre cómo obtener una calidad de clusterización mayor, basándonos en las medidas de calidad externas e internas antes expuestas.

6.1.1.1 Evaluación de métricas externas

Las medidas externas analizan cuál de las configuraciones cumple mejor con los objetivos de la herramienta, por lo que nos interesa medir con ellas que tan bien la clusterización logró separar consultas que no poseen datos relevantes en común y dejar juntas las que sí. Estas métricas son un indicio de que la clusterización logró sus cometidos con el proyecto.

6.1.1.2 Precisión de clasificación

Mide el porcentaje de las tablas contenidas en la configuración de un cluster, que es relevante a todas las consultas que se satisfacen en él. Es decir, qué cantidad de tablas del almacén del árbol se deben utilizar al ejecutar cualquiera de las consultas del cluster. Este tipo de tablas se encontraría en el último nivel de nuestro árbol diamante.

Lograr un buen valor en este principio indicaría que la solución es muy deseable considerando cómo fue diseñado el árbol diamante, pero el que no consiga un buen valor no indica un mal tiempo de ejecución. Su estudio es importante porque indica el porcentaje de términos de las consultas que significó la relación más fuerte al clusterizar.

6.1.1.3 Intersección entre clusters

Esta métrica indica la suma de las intersecciones de tablas entre clusters. Esto es, por cada cluster se toman todas las tablas de su almacén. Luego, se recorren los demás clusters de la configuración y se van sumando las tablas que sirven al cluster actual y al inicial. Esto se repite por cada cluster en la configuración.

La suma total es comparable entre distintas configuraciones. Más abajo se muestran todas las sumas para las 16 alternativas diferentes generadas.

6.1.1.4 Evaluación de métricas internas

Existen coeficientes que indican qué tan bien se logró que las consultas se agruparan con otras consultas similares a ellas. Estos coeficientes miden la calidad de la operación de clusterización como tal y dejan de lado las implicaciones de este con el resultado del proyecto.

Se utilizó el coeficiente de silueta, de la librería de Python llamada sklearn. Este coeficiente se calcula utilizando la distancia media entre los componentes de un mismo cluster (A) y la distancia media entre una muestra y el cluster más cercano distinto al propio (B). El cálculo realizado es el siguiente:

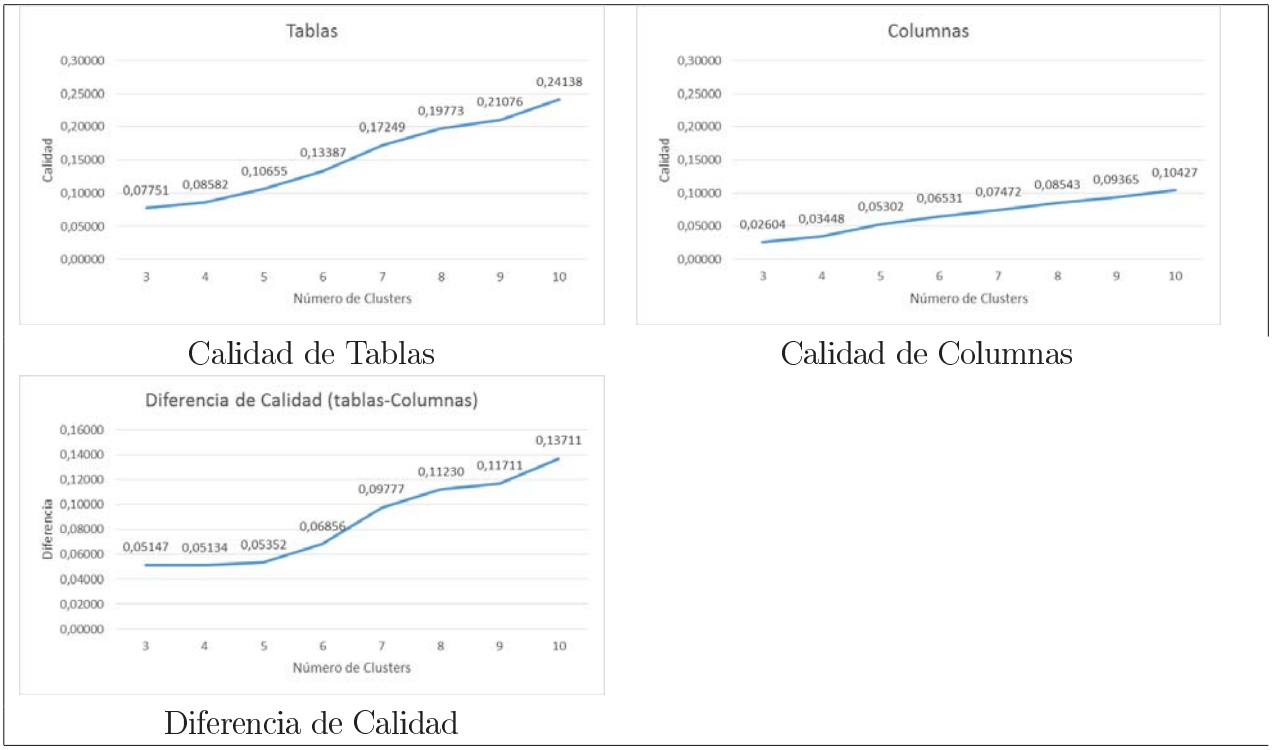
$$COEFSilueta = \frac{(B - A)}{MAX(A, B)}$$

Esta medida es calculada para cada muestra, obteniéndose la calidad de haber sido asignada a su cluster actual. Con este valor se puede tener una noción de cuan bien estan conformados los clusters.

El coeficiente de silueta sirve como una medida de qué tan bien el proceso de clusterización logró agrupar muestras que son similares entre ellas. Si se logra obtener un coeficiente de silueta alto, entre 0.2 y 1, se puede afirmar que los resultados son densos, donde las muestras en el mismo grupo resultan similares entre sí, y bien separados, logrando que entre diferentes grupos las muestras tengan datos diferentes en sus dimensiones. Valores muy cercanos a 0 indican que los grupos se solapan, lo cual significaría una gran intersección de tablas entre los clusters. Si los valores de coeficiente de silueta resultan

negativos indica que los clusters fueron mal asignados, esto quiere decir que el vecino más cercano representa una mejor opción a agrupar que el cluster actual de la muestra evaluada.

Los resultados generales que entregan a continuación, muestran el promedio del coeficiente de silueta para todas las muestras considerando su respectivo cluster. Estos resultados fueron los siguientes:



Calidad de Tablas

Calidad de Columnas

Diferencia de Calidad

Cuadro 4: Gráficos de combinatorias

Como se puede apreciar la calidad de la clasificación aumenta a medida que incrementamos la cantidad de clusters en la configuración, una tarea a realizar es buscar la cantidad óptima de clusters a generar. Los resultados fueron favorables en un principio para la clusterización de tabla, generándose siempre una diferencia positiva con respecto a la clusterización tabla-columna. Adicionalmente, esta diferencia resulta incremental.

A continuación, como información de referencia, entregamos la calidad de asignación de cada muestra para el principio tabla y tabla-columna al generar 10 clusters.

Al observar estos datos detalladamente descubrimos que los datos mal asignados per-

Cuadro 5: Tabla

Cluster	N°querys	$\overline{X}$	MIN	MAX	σ
1	2	0,99999999	0,99999999	0,99999999	0
2	2	0,915388505	0,91537338	0,91540363	2,139E-05
3	5	0,19069457	0,15331436	0,23948713	0,034911298
4	4	0,259507345	0,22405706	0,32487026	0,044672609
5	58	-0,022386829	-0,07688637	0,00780527	0,019323972
6	3	0,341995797	0,22833051	0,41427574	0,099641694
7	10	0,066637539	0,03410063	0,09312677	0,019696323
8	6	0,247932907	0,11327615	0,31655133	0,070950541
9	3	0,350067179	0,39910647	0,63650811	0,134509717
10	6	0,159333453	0,08144899	0,22100106	0,05579092

tenecen todos al mismo cluster. La información de asignación por cluster para el agrupamiento por tabla-columna con 10 clusters generados es el siguiente.

El cluster 5 agrupó a todos los clusters que no pudieron ser agrupados. Sus 58 muestras tienen una calidad baja de asignación, el cluster por tabla también posee un cluster de éstas características pero es significativamente más pequeño.

La calidad interna de la clusterización depende en gran medida del número de clusters generados y del tamaño del cluster que agrupa las consultas sobrantes. Si se trabaja en encontrar las medidas óptimas para estos dos factores, aumentará la calidad de la clusterización.

6.2 Tráfico en la red

Como se había mencionado anteriormente, el tráfico en la red será el siguiente elemento a analizar con métricas. Desde ahora en adelante, se hablará de dos situaciones. La primera, en donde se realizará la ejecución sin particionamiento, tendrá como característica que las máquinas en donde corre el algoritmo tienen una capacidad infinita, lo que trae como beneficio que no se produzcan particionamientos al momento de correr todos los algoritmos. Esto obliga a que la ejecución de las consultas se haga en una segunda máquina para evitar la caída de la misma, que genera un problema a nivel sistémico en el cual no se puede visualizar, ni consultar los datos. En la segunda, se realizará una ejecución con particionamiento y funcionará con pequeñas máquinas, al igual que el análisis de costos que se planteó anteriormente, pero con la salvedad de que las consultas se ejecutan sobre la máquina que tiene más datos para satisfacer a la misma con el menor movimiento de datos.

Entre las métricas que se mostrarán a continuación, se omite la ejecución de la consulta

al momento de tener todos los datos para su ejecución. Esto se debe a que el tiempo de ejecución, tanto en la primera como en la segunda situación sería el mismo. Esto resulta evidente considerando que los datos son los mismos, de tal forma que el tiempo de respuesta sería exactamente el mismo desde ese momento, por lo tanto, se ignora este tiempo.

La siguiente consideración que se debe tener en cuenta al momento de revisar los datos, es una característica que poseen actualmente los servicios que ofrecen los proveedores que se analizaron, esta corresponde a que el tiempo de comunicación entre dos máquinas en un mismo site son tendientes a cero.

Para entender cómo se analizaron los resultados, se debe tener en cuenta que la ejecución de una consulta es la suma de llevar todos los datos a la máquina donde se va a ejecutar dicha consulta. Además, se debe considerar que como todo fue hecho en un ambiente de simulación, tanto la corrida sin particionamiento como la que lo tenía, reciben los mismos datos y las mismas consultas exactamente en el mismo tiempo de simulación, lo que permite hacer una comparación de ambos casos de estudio sin tener un sesgo de diferenciación de la red.

El ambiente que se ocupó para la simulación considera una hora de tiempo de simulación; la ejecución de más de 500 consultas y más de veintisiete mil datos que son aproximadamente cinco gigabytes. Se debe considerar que el peso final de la ejecución de las consultas es mucho mayor a la carga de datos ingresadas en el servidor. Esto se debe a que se re-ocupan muchas veces datos que ya están ingresados y esto hace que la suma de la ejecución de las consultas sea mucho mayor a los datos ingresados.

En la primera métrica que se puede observar en la figura 7, se representa la cantidad de bytes promedio que se movieron por ejecución de consultas durante el tiempo de simulación. Esto representa prácticamente todo el consumo de red que tiene una empresa en la realidad. Todo esto sucede porque a pesar de que hay un ingreso de datos que genera un costo de inserción, éste es obviado para la métrica debido a que es un costo en la red que debe ser asumido siempre y no puede ser disminuido; sin embargo, el costo de la ejecución de las consultas debe ser y es el que se está tratando de reducir.

En la figura 7 se puede observar claramente que el consumo interno del tráfico de red se ve disminuido drásticamente en los primeros niveles del clustering, sin embargo, cuando la cantidad de cluster crece, éste se ve disminuido, pero siempre mantiene una diferencia promedio de un 30 % de ahorro de la simulación con particionamiento de la que no lo tiene. Esto demuestra que a pesar de la cantidad de cluster éste porcentaje se mantiene constante, lo que muestra que la partición de los árboles y los movimientos de los fragmentos se comportan de tal manera que permiten disminuir el tráfico.

Sin embargo, uno podría llegar a la conclusión anticipada de que separar en cluster no tiene sentido dado que siempre se ahorraría alrededor del 30 % del tráfico de la red, pero esto no es así, el que hayan más cluster implica un movimiento menor dentro de los mismos, lo que permitiría en la realidad dispersar el servicio por el globo dado que los servicios actuales lo permiten. Esto traería consigo el beneficio de que entre más centro de datos tengo para procesar y el tráfico dentro de este centro de datos sea menor me permite disponer de más servicios que abusen del tráfico de la red como lo son los mapas de logs o servicios de sincronización.

Pero la figura 7 no nos dice mucho respecto a cómo es tráfico de la red en cuanto al movimiento de datos cuando se particiona un árbol, es por esto que en la figura 8 y figura 9 se puede apreciar la cantidad de particiones promedio que se realizan con ambos métodos de clusterización.

Para entender el porqué de la forma en la figura 8 es necesario entender algo que se explicó anteriormente que es el hecho de a pesar de que uno trata de clusterizar de la mejor manera posible los grupos de consultas, siempre hay un grupo de consultas que no se puede agrupar, pero cuando este grupo de consultas se pueden separar en dos conjuntos que entre si están mejor agrupados sucede lo que pasa en el gráfico, la cantidad megabytes movidos disminuyen pero a medida que se comienzan a re-armar el grupo de consultas que no tiene nada en común permiten que la cantidad de datos movidos en las particiones aumente dado que se generan más particiones. Igualmente se puede ver claramente que los resultados obtenidos anteriormente se condicen con la disminución de tráfico en la red al momento de generar una partición del árbol.

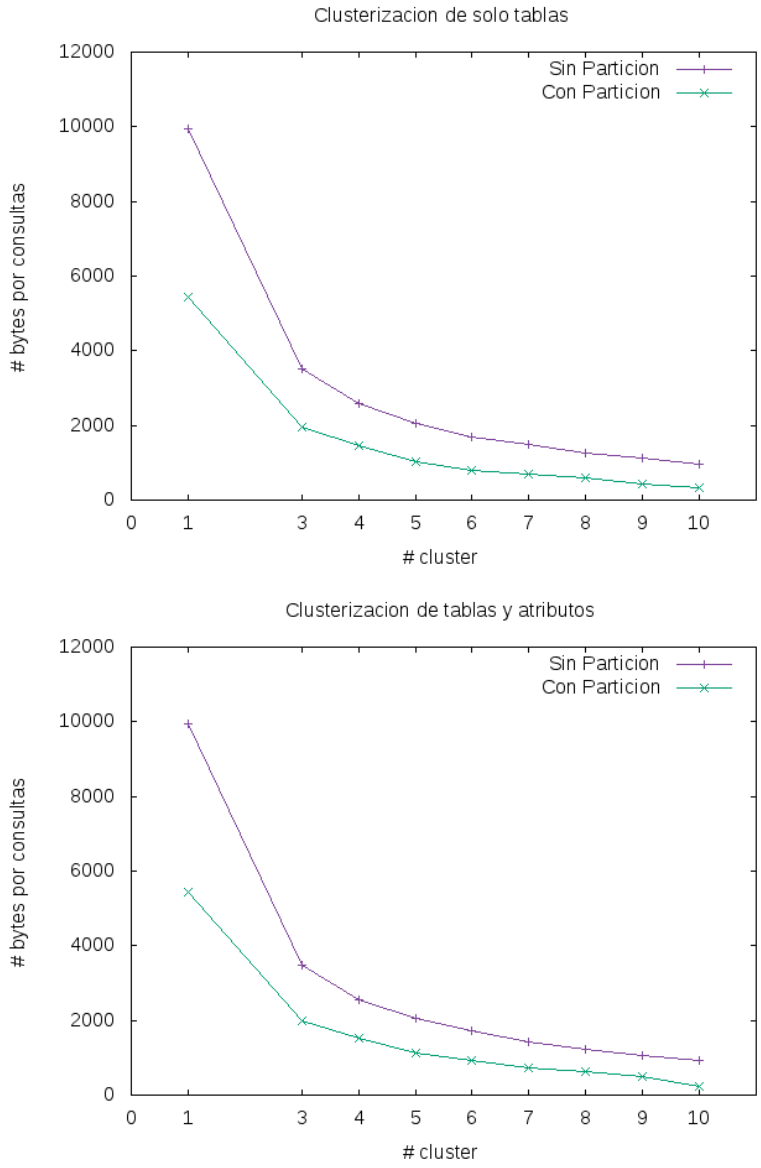


Figura 7: Gráfico 1

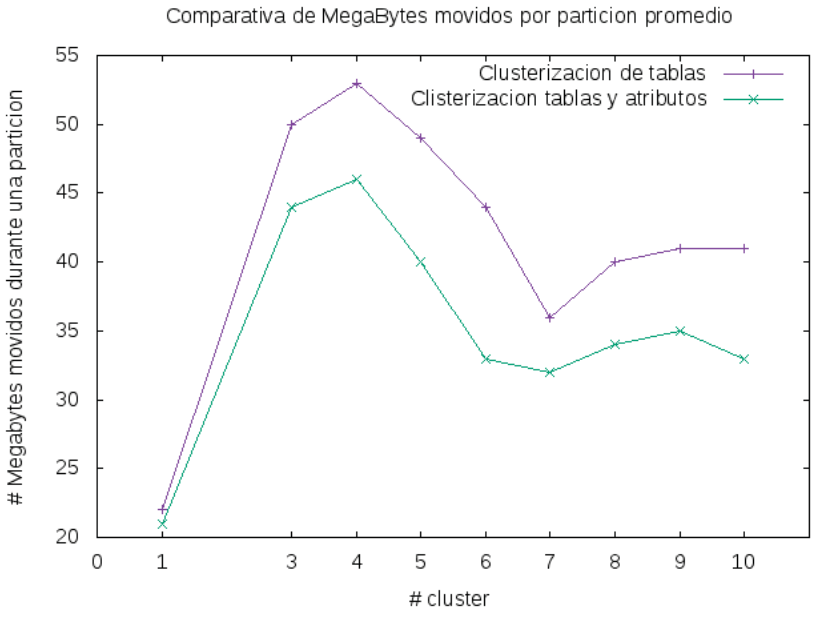


Figura 8: Gráfico 2

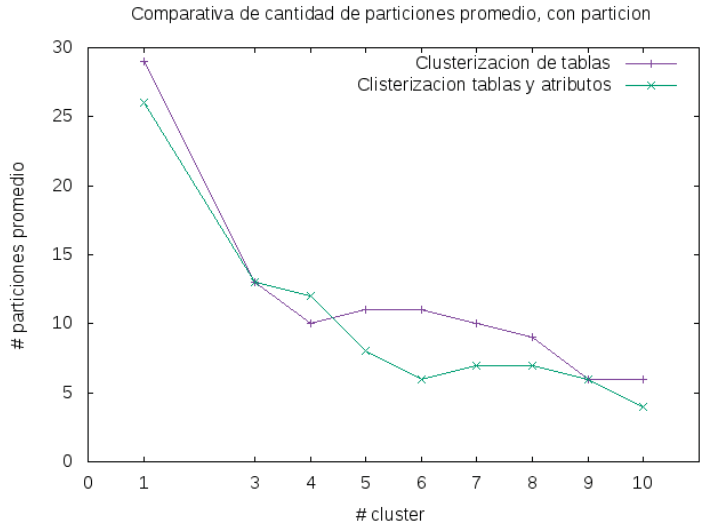


Figura 9: Gráfico 3

7 Conclusiones

Cuando se comenzó este trabajo, se planteó uno de los problemas que tienen actualmente las grandes empresas. El tema abordado fue la administración de los grandes volúmenes de datos que son ingresados a los sistemas transaccionales y miles de sensores puestos a nivel global que permiten que estas bases de datos crezcan a cada minuto y segundo, es por esto que el poder administrar estos datos ha sido una labor compleja para las empresas. La administración de estos datos requiere encontrar formas de aumentar el rendimiento de las bases de datos; al momento de pensar en particiones, dividir los datos, o hacer un sistema que permita tener una carga distribuida, es necesario pensar en la capa de datos, pues esta tiene un grave problema: es difícil de desagrupar. Al realizar el estudio de las soluciones previas que otros autores han dado a ésta problemática, se pudo llegar a una conclusión. Todos buscan una forma de separar los datos de una forma inteligente, siempre ocupando alguna regla, estudiando el dominio del problema, las consultas, o algún método que permita desagrupar esos datos para poder generar una base consistente.

Dentro de las bases consistentes, se pueden establecer tres consideraciones, el primero es que los datos deben utilizar el menor ancho de red posible; segundo, hay que validar que estos datos puedan seguir respondiendo a todas las consultas que la organización estime relevantes; y la tercera, es que quizás al separar los datos, se pueden perder relaciones importantes entre ellos, algunos autores plantearon que los datos no deben separarse. Sin embargo, al momento de revisar el trabajo de otros autores, se puede deducir que administrando los datos de forma correcta puede existir integridad al separar.

El estudio se centró en establecer un procedimiento que permitiera separar los datos, inicialmente de forma estática con particiones de grafos; sin embargo, esto hoy en día no se puede llevar a ambientes productivos ya que las bases de datos crecen demasiado rápido y, al crecer demasiado rápido, existen dos factores que marcan el tiempo. Estos factores son, la tasa de llegada y la capacidad que se tiene para responder a las consultas. Por lo que se decidió adoptar una solución dinámica, que en situaciones críticas utiliza un particionamiento de grafos para una porción reducida del modelo de datos.

Posteriormente al estudio de lo que se ha planteado en torno a esta problemática, el equipo se propuso encontrar una nueva manera de separar los datos, la decisión se basó en aprovechar al máximo las relaciones entre los datos. Es por esto, que la capacidad de desagrupar las consultas y generar clusters de datos, fue la llave para conseguir los fines del proyecto, logrando distribuir la carga de trabajo.

Al tener estos datos desagrupados y de forma estructurada, se presentó uno de los mayores desafíos abarcados por el proyecto; se hacía necesario encontrar una forma orde-

nada de llegar a los datos. La forma en que se administrará a la gran cantidad de datos a manipular, debe satisfacer una gran cantidad de necesidades propias del dominio del problema. La principal de ellas consistía en aprovechar una característica de la solución que se estaba diseñando. Los distintos conjuntos conformados, denominados fragmentos, tienen intersecciones jerarquizables y estos conjuntos de relaciones pueden resultar enormes. Se estudiaron diversas estructuras de datos que pudieran almacenar estos fragmentos. Los árboles de decisión no cumplen con la lógica de mantener una intersección, pues evalúa si el grupo cumple o no con las restricciones, por lo que no sirve para agrupar. Luego, se evaluaron los árboles B, los cuales al momento de evaluar cómo administrar las intersecciones no proponían ningún beneficio. Finalmente llegamos a los árboles diamante, estos permitieron una mejora cuantificable en el rendimiento de las consultas. Pese a que inicialmente este aumento en el rendimiento no fue tan considerable como se esperaba; permitió generar una base de conocimiento para comenzar a desarrollar.

Como problema paralelo a la estructura de datos a usar, existió la necesidad de encontrar una forma diferente de desagregar el modelo de datos. Se decidió buscar una forma alternativa debido a problemas encontrados en el algoritmo Dyn Part Group. Este mostró la posibilidad de distribuir los datos en fragmentos relacionables, asociar pequeños conjuntos de datos según una función de similitud y afinidad. Dicha ecuación tenía la desventaja de necesitar que los fragmentos crecieran a una misma velocidad, esto iba en desmedro de una separación de datos efectiva. El problema se refleja en que cuando llega un dato y los factores de carga se ven superados, el algoritmo no permite que se inserte este dato en el fragmento que logra la mejor afinidad, por lo que se genera un entrampamiento y se obliga a buscar en otros fragmentos con menor afinidad, esperando que el dato quepa conforme a la restricción, lo que podría haberse evitado al aglomerar nuevamente en más conjuntos y moverlos.

La lógica del algoritmo Dyn Part Group es permite un buen control de la tasa de llegada pero va en contra de los objetivos de la solución aquí planteada. Debido a este razonamiento se definió dejar que los fragmentos crezcan libremente y efectuar una partición de grafos en una parte específica del modelo de datos, para poder seguir brindando una solución factible en tiempos de ejecución.

El árbol diamante definido en esta tesis demostró tener la capacidad de aprovechar las relaciones entre fragmentos para realizar búsquedas con mejor rendimiento; que se le haya otorgado esta capacidad al diseñarlo no es casual. El objetivo primario de este trabajo es minimizar el tráfico en la red en base al estudio de la carga de trabajo del cliente. Los fragmentos y sus asociaciones representan explícitamente el análisis de la demanda de datos del cliente hacia el sistema en consultas SQL. Se puede afirmar que un fragmento es una subdivisión alternativa del modelo de datos, este nuevo nivel de abstracción está dise-

ñado para permitir el desglose del coste al transportar los datos para satisfacer consultas. Una vez se parta un árbol diamante, sus nodos se distribuirán en máquinas, los costes de comunicación estarán dados por las relaciones de fragmentos podadas.

La clusterización busca que cada árbol diamante ya intente satisfacer un conjunto de consultas fuertemente relacionado, es en este sentido que un árbol general ya viene podado según sus relaciones más fuertes, formando los árboles diamante que operan en cada servidor; así se evitan además sucesivos particionamientos de grafos al comienzo de la ejecución. Resumiendo, se asigna un árbol diamante por cluster, este administrará los conjuntos de datos relevantes para su grupo de consultas correspondiente. Además estos conjuntos de datos se pueden desagregar, son particionables. Esto quiere decir, que se habla de un sistema distribuido que finalmente en conjunto a todo lo realizado, tiene el potencial de dividir la carga en base a un análisis premeditado.

A pesar de lo anteriormente expuesto, el potencial de esta herramienta depende en gran medida del análisis de datos que se haga sobre las consultas del cliente. Que se tomen en cuenta ciertas características de la carga de trabajo y se ignoren otras produce que se determinen distintos clusters, a su vez estos conllevarán movimientos de datos totalmente diferentes en los árboles. Para estudiar este efecto se realizaron dos tipos de clusterización, uno basado en las tablas, y otro basado en las tablas-consultas. Las primeras aproximaciones, como se pudo ver en los gráficos, no fue tan decisivo como se esperaba, pero a pesar del bajo coeficiente se pudo apreciar una mejora en el rendimiento. Si se encuentra una forma de clusterizar con mayor efectividad se podrían obtener resultados aún mejores.

Respecto al árbol diamante se puede observar que funcionó de forma óptima en el tema de la partición, pese a que fue ya probado por Kayoor en el paper 1, se cambió un poco la técnica y evolucionó con el tema de los cluster, en cuanto a rendimiento estuvo bastante bien. A pesar de que el árbol tenía una escala logarítmica que no era de las mejores, se pudo llegar a algo que tenía un sentido y que finalmente se pudo mejorar con una programación dinámica para poder llegar así a un árbol n logn. El log es a partir de la base de la máxima cantidad de nodos que se pueden llegar a tener, al igual que un árbol b, sin embargo, su complejidad es baja para la cantidad de nodos que pueden tener, que en sus niveles medios, que es donde más temor se tiene que se expanda, puede tener cientos de miles de nodos, pero en sus niveles inferiores, que es donde se espera que tenga toda la carga, y esto por el nivel de clusterización hace que los nodos queden en la parte baja y no en la parte media, produce que este árbol tenga un potencial gigante sobre esta solución.

Al mirar los gráficos uno puede observar y llegar a la siguiente conclusión, las herramientas ocupadas y toda esta arquitectura que se formó a partir de un grupo de ideas de una serie de autores que se le sacó lo que se pensaba que estaba mal y se unieron con otras

ideas que tenían un tremendo potencial pero que se veían disminuidas por las restricciones o por el contexto que se les estaba dando, ayudó a que pudiera mejorar el sistema, y es esto lo que se ve en el gráfico. No se ve una mejora de rendimiento considerable, pero viéndolo desde el punto de vista del tráfico de la red, que era lo que se planteó desde un principio, se disminuyó cerca de un 30 % que en sí no se ve como un gran cambio, pero cuando se piensa que ese mismo 30 % fue dividido en diez cluster que entre ellos en principio no tienen comunicación, pero sí están haciendo réplicas, permiten que la base de datos puedan estar replicadas y puedan responder un conjunto de consultas asociadas a ellas, por lo que mejora el rendimiento tanto a la máquina como el rendimiento general del sistema.

Una teoría que queda en el aire es si la clusterización de consultas y esta forma de trabajo de poderla agrupar y efectuar consultas sobre un conjunto de datos desde una perspectiva o como una programación en todo aspecto tiene sentido en la vida cotidiana. Por parte de la experiencia de los autores en el manejo de grandes bases de datos, se puede pensar que sí, al agrupar las consultas por tipo y el trabajo que hacen internamente, ayuda a que las máquinas no se sobrecarguen en cuanto a su rendimiento. Por otra parte, la clusterización lo que busca es saber en qué máquinas debe correr cierto tipo de consultas, lo que permite tener indexado de diferentes maneras la información para poder sacar mejor rendimiento a los tipos de consulta que se uno quiera correr sobre estas bases de datos, incluso tener distintas bases de datos para el mismo conjunto de datos, sacándole provecho a la ejecución de las consultas y el rendimiento general del sistema se vería disparado porque ocuparía un tipo de base de dato específico para cada tipo de cluster que se genere, lo cual le da un potencial enorme, sobre todo en los sistemas actuales que intentan mantener una serie de restricciones y de datos asociados que permitirían tener una ventaja muy alta en las decisiones de diseño.

En otro aspecto, se puede plantear a futuro una comparación entre el árbol diamante y el árbol de decisión. El árbol de decisión se caracteriza porque se evalúan una sola vez, sin embargo, por ejemplo en el sistema bancario se ocupan sistemas batch que corren durante horas y podrían ser ayudados con este tipo de árboles para reducir la consulta y poder agrupar clientes ya procesados puesto que ya se saben las características de esos clientes y de esta manera ahorrar el reproceso que el sistema batch utiliza para poder llegar a la misma conclusión. A partir de estos dos trabajos futuros nace un tercero, el cómo reducir la cantidad de particionamiento, ya que pese a que cuando el árbol se particiona genera poco ruido en cuanto al ámbito de generación de información, genera un alto costo por el movimiento y la re-estructuración del árbol. Esta situación se puede ver de forma clara cuando los grupos de cluster son las consultas que quedaron y no se pueden agrupar sobre otro conjunto, son las que más daño hacen a todo el sistema en general. Por lo que se podrían hacer sistemas y poder agrupar este tipo de consultas en bases de datos más

específicas y con mayor rendimiento que el resto ya que estas últimas ya se encuentran agrupadas sobre los datos que más acceden, lo que permite, a nivel de capa de datos, mejorar el rendimiento general del sistema, donde actualmente la capa de datos es la más importante dentro de todas las capas que se ocupan dentro de un sistema, pese que actualmente se le está dando más importancia a la capa de presentación pero ésta no puede no ser alimentada por una capa de datos bien formada, porque pese a que se le pueda dar toda la tecnología a la capa de presentación o a la capa de negocios, la capa de datos es la que finalmente le da soporte al sistema y le permite entregar toda esa información al sistema y pueda llevar esa vista al cliente.

Referencias

[1] Metis. `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`. [Online; accessed 18-junio-2015].

[2] Quantcast. `https://www.quantcast.com/engineering/qfs`. [Online; accessed 24-april-2015].

[3] Riak. `http://basho.com/riak/`. [Online; accessed 24-april-2015].

[4] TPC-DS. `http://www.tpc.org/tpcds/default.asp`. [Online; accessed 18-junio-2015].

[5] Agrawal, S., Narasayya, V., and Yang, B. Integrating vertical and horizontal partitioning into automated physical database design. In *SIGMOD* (June 2004), Association for Computing Machinery, Inc.

[6] Baev, I. D., and Rajaraman, R. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2001), SODA '01, Society for Industrial and Applied Mathematics, pp. 661–670.

[7] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst. 26*, 2 (June 2008), 4:1–4:26.

[8] Chansler, R., Kuang, H., Radia, S., Shvachko, K., and Srinivas, S. The Hadoop Distributed File System. `http://www.aosabook.org/en/hdfs.html`, 2007. [Online; accessed 24-april-2015].

[9] Curino, C., Jones, E., Zhang, Y., and Madden, S. Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow. 3*, 1-2 (Sept. 2010), 48–57.

[10] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev. 41*, 6 (Oct. 2007), 205–220.

[11] Eltabakh, M. Y., Tian, Y., Özcan, F., Gemulla, R., Krettek, A., and McPherson, J. Cohadoop: Flexible data placement and its exploitation in hadoop. *Proc. VLDB Endow. 4*, 9 (June 2011), 575–585.

[12] Ghandeharizadeh, S., and DeWitt, D. J. Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor databases machines. In *Proceedings of the Sixteenth International Conference on Very Large Databases* (San Francisco, CA, USA, 1990), Morgan Kaufmann Publishers Inc., pp. 481–492.

[13] Golab, L., Hadjieleftheriou, M., Karloff, H. J., and Saha, B. Distributed data placement via graph partitioning. *CoRR abs/1312.0285* (2013).

[14] Kumar, K. A., Deshpande, A., and Khuller, S. Data placement and replica selection for improving co-location in distributed environments. *CoRR abs/1302.4168* (2013).

[15] Liroz-Gistau, M., Akbarinia, R., Pacitti, E., Porto, F., and Valduriez, P. Dynamic Workload-Based Partitioning Algorithms for Continuously Growing Databases. *Transactions on Large-Scale Data- and Knowledge-Centered Systems* (2014), 105.

[16] Navathe, S. B., and Ra, M. Vertical partitioning for database design: A graphical algorithm. *SIGMOD Rec. 18*, 2 (June 1989), 440–450.

[17] Nehme, R., and Bruno, N. Automated partitioning design in parallel database systems. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2011), SIGMOD '11, ACM, pp. 1137–1148.

[18] Rao, J., Zhang, C., Megiddo, N., and Lohman, G. Automating physical database design in a parallel database. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2002), SIGMOD '02, ACM, pp. 558–569.

[19] ren Liu, D., and Shekhar, S. Partitioning similarity graphs: A framework for declustering problems. *Information Systems Journal 21* (1996), 475–496.

[20] Zeitler, E., and Risch, T. Massive scale-out of expensive continuous queries. In *VLDB Endowment Vol. 4, No. 11* (2011).

[21] Zilio, D. C. Physical database design decision algorithms and concurrent reorganization for parallel database systems. Tech. rep., University of Toronto, 1997.