UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

MONITOREO EN TIEMPO REAL DE DNS UTILIZANDO HERRAMIENTAS OPEN SOURCE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

FELIPE ALEJANDRO ESPINOZA ROSALES

PROFESOR GUÍA:
JAVIER BUSTOS JIMÉNEZ

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
RODRIGO ARENAS ANDRADE

SANTIAGO DE CHILE
2018

Resumen

El sistema de nombres de dominio (DNS, por sus siglas en inglés, Domain Name System) ha pasado a ser una parte fundamental en la infraestructura de Internet, permitiendo el acceso a los diferentes recursos disponibles de manera fácil y rápida. Este sistema es utilizado para localizar los diferentes sistemas críticos encontrados en internet, permitiendo traducir nombres de dominio memorables en una dirección IP numérica.

Dada la gran importancia que este sistema ha obtenido, asegurar su correcto funcionamiento ha pasado a ser un punto fundamental, lo cual ha llevado a implementar sistemas de monitoreo que permitan detectar anormalidades en los servidores DNS. Las herramientas de monitoreo actuales son de carácter privado, o no aprovechan los avances que se han generado para realizar un análisis y alertas más detalladas sobre el estado de estos.

Esta memoria describe el análisis, desarrollo e integración de diferentes herramientas open source para realizar el monitoreo de servidores DNS, permitiendo visualizar su estado actual y detectar anormalidades, activando alertas para notificar a los administradores. Además, se busca permitir el análisis de eventos pasados a través del almacenamiento de la información desagregada, permitiendo una visión detallada para el desarrollo de medidas de prevención y mitigación de eventos futuros. La utilización de herramientas open source permite facilitar el acceso al proceso de monitoreo a los administradores de los servidores DNS, y así mejorar la velocidad de reacción ante eventos producidos en esta crítica pieza de internet.

El proceso de monitoreo se dividió en las capas de captura, almacenamiento y visualización, en las cuales se establecieron los diferentes requisitos que debe cumplir cada una de estas, además de las métricas a medir basándose en trabajos relacionados.

Los softwares públicamente disponibles analizados en la capa de captura no lograron satisfacer todos los requisitos planteados, por lo cual, basándose en software similares, se realizó el desarrollo del software *DnsZeppelin*, el cual logro presentar una mejora en la velocidad de procesamiento de paquetes DNS en comparación a las otras herramientas analizadas.

La capa de captura presento ser el punto más complejo de esta arquitectura, dado el gran volumen de datos que debe ser almacenado y agregado para su visualización. Para realizar una selección que se adecue a los requerimientos definidos, se procedió a realizar un benchmark de los diferentes sistemas, realizando una simulación de los datos que se generarán y las consultas a realizar, con lo cual finalmente se seleccionó el software *ClickHouse*.

En la capa de visualización se realizó la selección del software según el cumplimiento de los requerimientos y su compatibilidad con el sistema de almacenamiento seleccionado, lo cual llevo a seleccionar el software *Grafana*, en el cual se implementó la compatibilidad con el sistema de alertas que este posee.

Finalmente, utilizando los softwares *DnsZeppelin*, *ClickHouse* y *Grafana*, se desarrolló su integración y se realizaron pruebas con datos simulados y reales provistos por el laboratorio de investigación de NIC Chile *NICLabs*, validando el correcto funcionamiento de este sistema sobre cargas mayores a diez veces a las hoy generadas.

Agradecimientos

A mi madre, Marisol Rosales Lecaros, por estar presente en todo momento con su cariño y apoyo.

A todos mis amigos y compañeros, antiguos y nuevos que siempre estuvieron apoyándome para dar un paso más adelante.

A Javier Bustos Jiménez, por brindarme esta oportunidad, además de su apoyo y motivación en todo este proceso.

A todo el equipo de NIC Labs por su compañerismo y gran ambiente que permitió un muy buen desarrollo de este trabajo.

Tabla de Contenido

Capítulo 1

Introducción

1.1. Sistema de Nombres de Dominio

El sistema de nombres de dominio (DNS, por sus siglas en inglés, *Domain Name System*), es un sistema que permite acceder a los diferentes recursos disponibles en internet a través de la traducción de nombres de dominios a direcciones IP. Estos nombres se dividen de manera jerárquica, leyéndose de derecha a izquierda y separando cada nivel por un punto.

Los dominios de nivel superior (TLD, por sus siglas en inglés, Top Level Domain) se refieren al primer dominio de la jerárquica, tales como *.com* y *.cl*. En específico, existen los dominios de nivel superior de código de país (ccTLD, por sus siglas en inglés, country code Top Level Domain), los cuales son usados y reservados para un país o territorio dependiente, y donde cada uno de estos designa los gestores y establece las reglas para conceder dominios. Esto permite que cada país tenga autoridad sobre los sus dominios y establezca su infraestructura crítica de funcionamiento bajo su propio ccTLD.

Los administradores de ccTLD se encargan de administrar y resolver todas las consultas a los dominios registrados bajo su ccTLD (por ejemplo, *NIC Chile* administra los dominios *.cl*). Estos deben asegurar el correcto funcionamiento bajo las diferentes cargas que estos puedan presentar, lo cual hace necesario que muchos de estos servidores DNS sean distribuidos en diferentes localizaciones en el planeta. Estos servidores reciben miles de consultas por segundo que deben respondidas a alta velocidad para asegurar una buena calidad de servicio y experiencia de usuario.

1.1.1. Importancia del sistema DNS

El sistema DNS es utilizado por una gran cantidad de personas para realizar el acceso a los diferentes recursos disponibles en la red, permitiendo utilizar nombres memorables para realizar la localización de los diferentes servicios. Por otro lado, este sistema es utilizado por diferentes softwares como medio de localización de otros servicios, para así establecer

comunicaciones y automatizar procesos complejos.

Dada la gran importancia que este sistema ha obtenido, los administradores de los servidores DNS han centrado sus esfuerzos en asegurar su correcto funcionamiento y disponibilidad en todo el planeta, realizando la distribución de estos servidores en diferentes zonas geográficas, para así asegurar una respuesta rápida y una tolerancia a fallos mucho mayor.

Sin embargo, esto ha generado un gran aumento en la complejidad operacional, donde es necesario asegurar el correcto funcionamiento de una cantidad de servidores mucho mayor, lo cual ha generado una gran necesidad de realizar el monitoreo de estos en todo momento, para así reaccionar de la manera más rápida posible ante diferentes anormalidades que pueden ocurrir en el servicio.

Hoy *NIC Chile* realiza un monitoreo en vivo de sus servidores DNS, realizando una agregación de la información previa a su almacenamiento utilizando procesos batch. Esta información es luego transferida de manera periódica a un visualizador para mostrar el estado actual de los servicios. Este monitoreo se desea complementar con un sistema capaz de almacenar, analizar y visualizar información desagregada sobre las diferentes métricas capturadas en tiempo real, para así tomar acciones ante anormalidades de una forma más rápida y eficiente, generando alertas ante cualquier irregularidad que pueda ser detectada, permitiendo un análisis posterior de eventos para implementar medidas de prevención y asegurar el correcto funcionamiento del servicio en todo momento.

Esta memoria se centra en la investigación de la gran gama de herramientas open source que pueden ser utilizadas para realizar un monitoreo efectivo de este tipo de servidores en tiempo real, y su forma de integrarse para generar visualizaciones y alertas al detectar anormalidades en el servicio.

1.2. Motivación

Hoy en día existe una gran variedad de software open source enfocados en el monitoreo en vivo y generación de alertas de servidores y servicios. Sin embargo, estas soluciones no proveen todas las funcionalidades deseadas para realizar el monitoreo efectivo de servidores DNS, los cuales deben adecuarse a las diferentes necesidades que estos poseen en términos de escala, infraestructura y funcionalidades tales como análisis y alertas.

Esta deficiencia nace por la dificultad en la captura y almacenamiento de los datos generados por los servidores DNS para su posterior visualización, lo cual no puede realizarse directamente sobre los paquetes DNS del servicio, dado que su exportación a un servidor externo para su almacenamiento y visualización produciría una duplicación de la carga en la red, lo cual resulta inviable dada la cantidad de datos que se maneja actualmente en cada servidor.

Para realizar el almacenamiento de estos tipos de métricas se deben utilizar bases de datos especializadas en el manejo de información agregada en series de tiempo, es decir, pares de tiempo e información. Los softwares que se han desarrollado en esta categoría se centran en

el manejo de llaves de baja cardinalidad, tales como el nombre del servidor o nombre de proceso, no mostrando como caso principal en análisis de strings que involucran el servicio DNS tales como los nombres de dominio incluidos en una consulta.

Además de esto, la utilización de un software de almacenamiento especializado en este tipo de datos permite realizar un análisis sobre los datos históricos de manera directa, para así analizar eventos y ataque específicos de los servicios DNS y generar planes de acción y alertas en caso de que estos vuelvan a repetirse en un futuro.

Tomando lo anterior es necesario realizar la evaluación de las diferentes herramientas que se encuentran disponible, donde específicamente nos centraremos en las herramientas open source para la resolución de este problema.

Actualmente los servidores DNS de NIC Chile utilizan como solución de monitoreo *DNS Statistics Collector* (*DSC*) [2], un software open source con sus propias capas de captura, reducción, almacenamiento y visualización. Este software permite la exportación de los datos capturados a la base de datos relacional PostgreSQL [15] para realizar un análisis con herramientas externas, sin embargo, dada las grandes cantidades de información que se debe capturar, esta no logra escalar para manejar el volumen de información generado.

Por otro lado, este software no aprovecha los avances que se han desarrollado en el almacenamiento de este tipo de datos, con los cuales se pueden realizar consultas mucho más complejas para analizar eventos que ocurran en la red y almacenar los datos con una mayor precisión. Por último, este software no permite la configuración de alertas que reporten anormalidades en el sistema, con las cuales se podrían realizar acciones en caso de detectar anormalidades.

1.3. Objetivos

1.3.1. Objetivos generales

El objetivo general de la presente memoria consiste en el análisis de diferentes herramientas open source para los procesos de captura, almacenamiento y visualización de información específica de los servicios DNS, definiendo requisitos y métricas para realizar una evaluación y comparación de las distintas herramientas, con el fin de realizar un monitoreo DNS efectivo. Además de esto, se generará la integración de los softwares analizados para realizar un monitoreo en vivo de los servidores DNS, generando visualizaciones y alertas al detectar anormalidades en los servidores DNS.

1.3.2. Objetivos específicos

- Realizar una comparación entre los diferentes softwares open source para realizar el monitoreo de servidores DNS de alta carga, definiendo y evaluando métricas para los procesos de captura de datos, almacenamiento y visualización.

- Realizar la integración de los diferentes softwares seleccionados para realizar el monitoreo de servidores DNS, desarrollando las funcionalidades necesarias para cubrir las métricas que serán medidas desde estos.
- Proveer una visualización en tiempo real del estado de los servidores DNS monitoreados.
- Proveer detección y alertas de anormalidades básicas de los servicios DNS utilizando los datos capturados por el sistema.

1.4. Trabajos Relacionados

Uno de los softwares utilizados por diferentes administradores de servidores DNS corresponde a *DNS Statistics Collector* (*DSC*) [2], sistema desarrollado para recolectar y visualizar estadísticas de servidores DNS de alta carga. Este software realiza una agregación previa de la información y no permite realizar un análisis más profundo de los eventos que se producen en el sistema.

Otro sistema desarrollado de manera reciente corresponde a *ENTRADA* [38], sistema utilizado por los administradores del ccTLD *.nl* (Países Bajos) desde diciembre del año 2015, el cual realiza el consumo, análisis y almacenamiento de una gran cantidad de información de red. Este sistema logra manejar los grandes volúmenes de información generados desde los servidores DNS, sin embargo, requiere de una gran infraestructura y soporte para su correcto funcionamiento. Este trabajo busca generar un sistema capaz de manejar la carga de los servidores DNS utilizando la menor cantidad de recursos posibles, permitiendo su despliegue tanto en ambientes altamente distribuidos, como en ambientes de menor tamaño.

Capítulo 2

Marco Teórico

2.1. Funcionamiento del DNS

El *Sistema de Nombres de Dominio* (*DNS*, por sus siglas en inglés, *Domain Name System*) es un sistema que busca generar un espacio de nombres consistentes para referirse a diferentes recursos, de manera distribuida, posibilitando su utilización en diferentes aplicaciones con diferentes protocolos, y siendo posible utilizarlo tanto en computadores personales como de gran tamaño [25]. Este es generalmente utilizado hoy para realizar un mapeo entre un nombre de dominio y una dirección IP (por ejemplo, el nombre de dominio *www.gob.cl* y su dirección IPv4 asociada *163.247.172.147*).

El sistema DNS se compone de tres componentes principales:

- *Espacio de nombre de dominio* y *Registros de recurso*, los cuales son una especificación en forma de árbol de nombre e información asociados a estos.
- *Servidor de nombres*, el cual es un programa servidor que almacena información acerca de la estructura del árbol de dominio.
- *Resolver*, programa que extrae información de un servidor de nombres en respuesta a una consulta de un cliente.

En internet, los servidores de nombres son accedidos a través de su dirección IP, ya sea en su versión IPv4 o IPv6, utilizando los protocolos de transporte TCP y UDP, en el puerto 53 en ambos casos.

2.2. Administración de Nombres de dominio

Los nombres de dominio del sistema DNS se administran de manera jerárquica, permitiendo autonomía administrativa, autoritativa y control en internet. Los nombres de dominio se organizan en subdominios del *DNS raíz*, el cual es innominado, delimitando los subdominios

por el delimitador punto. La administración del *DNS raíz* actualmente es realizada por la *Internet Corporation for Assigned Names and Numbers* (*ICANN*) [27].

Un ejemplo de esta organización se muestra a continuación analizando el dominio "*www.example.com.*":

- [.]: Dominio DNS raíz.
- [*com*]: Dominio de nivel superior (o TLD por sus siglas en inglés *Top Level Domain*)
- [*example*]: Primera etiqueta que subdivide a *com.*
- [*www*]: Segunda etiqueta que subdivide a subdivide a *example.com.*

Los dominios de nivel superior son administrados por diferentes organizaciones. En particular, los dominios asignados a países tales como el *cl.* o *us.* son administrados por los ccTLD (dominio de nivel superior de código de país, o por sus siglas en inglés, country code top-level domain).

2.3. Paquetes DNS

La estructura de los paquetes DNS fue inicialmente definida por el RFC1035 [26], y ha sido actualizada a medida en que fueron agregándose más requisitos e información a internet.

La cabecera de un paquete DNS posee la estructura mostrada en la figura 2.1 y la información de un recurso se puede apreciar en la figura 2.2.

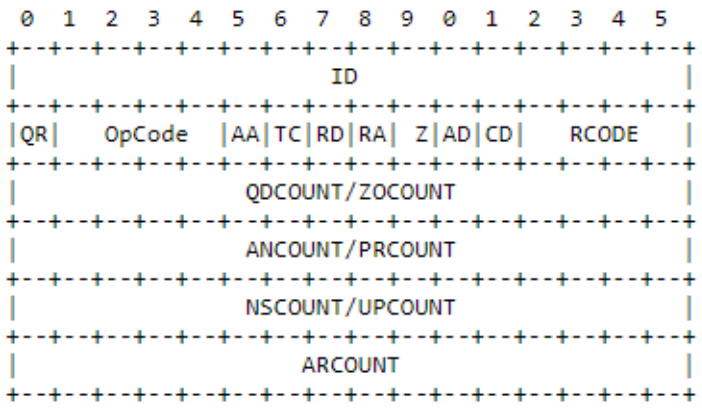

Figura 2.1: Estructura de la cabecera de un paquete DNS definida actualizada por el RFC6895 [1]. La información del paquete se encuentra luego de estos 12 bytes, la cual contiene en forma ordenada las consultas, las respuestas, información de autoridades de servidores de dominio e información adicional.

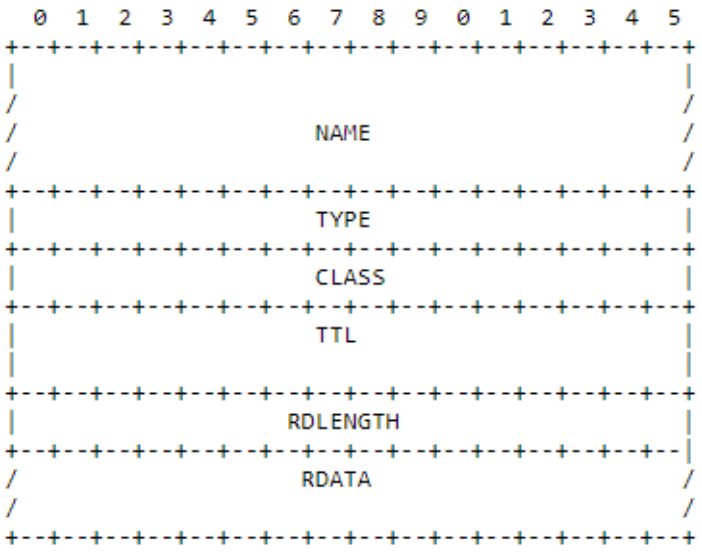

Figura 2.2: Estructura de un registro de recurso DNS definido por el RFC6895 [1], el cual contiene el nombre del recurso, tipo, clase, tiempo en segundos que puede ser almacenado en caché, largo del recurso y su información.

2.4. Captura de paquetes DNS

La captura de los paquetes DNS se pueden realizar de diferentes maneras, donde las más comunes se muestran a continuación:

- Captura desde software DNS: La información o estadísticas de los paquetes son expuestas por el mismo software, permitiendo una vista consistente con lo que ocurre en el servidor DNS. Sin embargo, estos softwares generalmente no presentan información suficientemente detallada para obtener el estado de este servició, entregando solo estadísticas muy general de este. Por ejemplo, el software de servidor DNS *BIND* [18] solo expone el número de consultas realizado al proceso actual separado por tipo y clase de recurso consultado, evitando la muestra de otras estadísticas tales como código de respuesta, TLD consultado, entre otras.
- Captura desde interfaz de red: Este tipo de captura obtiene los paquetes DNS directamente desde la interfaz de red de un servidor, escuchando todos los paquetes que entran y salen de este. La captura se puede realizar de manera online, realizando el procesamiento de los paquetes en conjunto con la captura, y de manera offline, guardando los paquetes capturados y enviándolos a un servidor externo para su procesamiento.

  La captura desde la interfaz de red se puede realizar en dos lugares diferentes

  - Captura desde computador donde se ejecuta el software DNS: Los paquetes DNS son leídos directamente desde el servidor DNS, permitiendo una visión completa de los paquetes que son recibidos y procesados por este. Esto trae la desventaja de agregar carga al servidor DNS para realizar el procesamiento online u offline de estos datos.
  - Captura de copia a través de un switch o router: Los paquetes DNS son copiados desde uno de los routers en la red, para realizar su procesamiento en un segundo

servidor. Esto permite separar la carga del monitoreo y el procesamiento de las consultas DNS.

Para realizar esta captura, generalmente se utiliza la librería *pcap* [36], la cual provee captura y filtro de paquetes, y es utilizada por monitores de red, detectores de intrusiones y generadores de tráfico.

Generalmente los softwares de monitoreo realizan una captura desde la interfaz de red, para así no realizar modificaciones en los softwares DNS que se encuentran funcionando actualmente, además de separar e independizar ambos sistemas. Los softwares que se analizaron para realizar la captura de información de los servidores DNS se muestran a continuación.

2.4.1. Fievel

Fievel [22] es un software desarrollado en C desde el año 2015, el cual captura paquetes de red, analiza y extrae información utilizando la librería *pcap* [36]. Este software fue desarrollado por NICLabs como una herramienta para realizar una inspección en tiempo real de los paquetes DNS, siendo parte del framework RaTA-DNS [23].

Fievel realiza una reducción inicial sobre los datos extraídos, para así reducir la cantidad de información que debe exportarse, y así evitar duplicar la carga de red que los servidores ya poseen. Estos reductores se encuentran desarrollados en el lenguaje de programación *Lua*, los cuales son extensibles y actualizables de manera sencilla.

Fievel puede realizar la exportación de sus datos, a través de la salida estándar, un archivo, o un canal Redis [33], lo cual permite extender este para otros protocolos de manera sencilla.

2.4.2. Packetbeat

Packetbeat [9] es un software analizador de paquetes open source, desarrollado por Elastic desde el año 2015, el cual provee soporte a múltiples protocolos tales como HTTP y DNS. La información extraída de cada paquete es enviada directamente a una base de datos Elasticsearch [6] o a un servidor de procesamiento Logstash [8] para realizar algún tipo de procesamiento.

2.4.3. Collectd

Collectd [28] es un software open source desarrollado activamente desde el año 2005, el cual recolecta, almacena y transfiere información desde los diferentes sistemas y aplicaciones de un computador. Collectd realiza este monitoreo a través de diferentes plugins, los cuales son desarrollados por la comunidad. Para el caso de paquetes DNS, este realiza la captura de los paquetes desde la interfaz de red, generando métricas sobre los *OpCode*, *QType*, *RCode* y *Octets* de estos paquetes. Además de esto, Collectd permite conectarse directamente a

algunas aplicaciones tales como BIND [18] para la extracción directa de las estadísticas de este servidor DNS.

Gracias a la madurez de este software, existe una gran compatibilidad con múltiples softwares para el almacenamiento de las métricas extraídas, lo cual permite su integración en diferentes ambientes de manera sencilla.

2.4.4. DNS Statistics Collector

DNS Statistics Collector (DSC) es un software desarrollado inicialmente por el *Internet Systems Consortium, Inc.* y *The Measurement Factory, Inc.* el año 2007, y luego continuado por *OARC, Inc.* el año 2016. Este software es un sistema para recolectar y explorar estadísticas desde servidores DNS, directamente desde los paquetes de red. Algunas de las métricas capturadas por este son las siguientes:

- Tipo de consulta
- Código de respuesta
- Código de operación
- Dirección fuente o subred
- Nombre de TLD consultada
- Parámetros EDNS
- Tipos comunes de polución DNS
- Tamaño de mensajes
- Transporte IP
- Puertos TCP/UDP

Las métricas capturadas se guardan en un archivo XML, el cual cada 60 segundos realiza una copia al servidor presentador, donde se utiliza el software DNS Statistics Presenter (DSP) [3] para convertir los archivos XML y presentar los datos utilizando un script CGI.

2.4.5. gopassivedns

gopassivedns [29] es un software open source desarrollado desde el año 2016, el cual busca entregar una mayor seguridad en los procesos de monitoreo de paquetes DNS utilizando el lenguaje *Go*, previniendo errores por corrupción de memoria que pueden ocurrir al utilizar otros lenguajes.

Este software realiza un procesamiento previo de la información contenida en los paquetes DNS, donde se asocian los paquetes de consulta y respuesta antes de generar la salida de los datos, permitiendo realizar un análisis más simple y rápido de la información capturada.

La captura de información se realiza desde la interfaz de red utilizando la librería *pcap* [36], soportando la capturan de paquetes IPv4 en los protocolos de red TCP y UDP. Luego de

procesar la información se genera una salida en formato *JSON* con la información siguiente.

- Id de consulta
- Código de operación
- Nombre de TLD consultada
- Código de respuesta
- Nombre de TLD respondida
- Tipo de respuesta
- Tiempo de vida (TTL)
- IP de servidor
- IP de cliente
- Fecha de consulta

2.5. Almacenamiento de datos

Para realizar el almacenamiento de las métricas capturadas desde los servidores DNS, es preferible la utilización de bases de datos de series temporales (TSDB, por sus siglas en inglés, Distributed Time Series Database). Estas bases de datos se pueden apreciar como un mapa con una llave, y una lista ordenada de fechas y métricas, donde cada una de estas puede ser una serie de puntos. Los datos que manejan estas bases de datos, generalmente se ingresan de manera ordenada e incremental en el tiempo, nunca son modificados, y la consulta de estos son periódicas sobre las fechas más cercanas a la actual, y muy poco recurrentes sobre fechas muy antiguas. Esto permite a las TSDB optimizar su estructura de almacenamiento sobre estas características, para obtener el mejor rendimiento posible.

Dada las características de los datos, y a la gran magnitud de estos, no es recomendado utilizar una base de datos relacional. Estas generalmente manejan sus datos utilizando transacciones ACID (por sus siglas en inglés, *Atomicity, Consistency, Isolation, Durability*), las cuales son propiedades que permiten mantener la consistencia de los datos a medida que estos se insertan, actualizan y eliminan de la base de datos. Dado que nuestras métricas son ingresadas de manera permanente, perder la consistencia de los datos de manera temporal, no afecta en gran manera al análisis agregado de estos. Por otro lado, dado que no se posee la necesidad de manejar la actualización de los datos, es posible eliminar gran parte de la complejidad asociada este tipo de operación. Por estas razones, solo se recomienda utilizar este tipo de base de datos si se posee una carga baja en el ingreso de métricas, para así aprovechar el mayor poder de consulta y agregaciones que pueden realizarse sobre los datos, gracias a la gran madurez que poseen bases de datos tales como MySQL [5] y PostgreSQL [15].

Para realizar el almacenamiento de los datos generados por los servidores DNS, se realizó el análisis de múltiples sistemas de base de datos, los cuales se presentan a continuación.

2.5.1. Prometheus

Prometheus [31] es un sistema de monitoreo y alertas, originalmente desarrollado por SoundCloud [35] desde el año 2012, el cual hoy, es un proyecto open source independiente mantenido por su comunidad y colaboradores. Este es utilizado actualmente por empresas como *Ericsson*, *Quobyte* y *Maven Securities*.

Prometheus presenta un modelo de datos multi-dimensional, donde se almacena una serie temporal a través de un nombre de métrica y una serie de pares llave/valor llamados *labels*. Para el almacenamiento de índices, este utiliza LevelDB [14], una base de datos llave/valor desarrollada por Google que provee un mapeo ordenado de llaves string a valores string. Por otro lado, para los datos almacenados, Prometheus implementa su propia capa de almacenamiento, la cual organiza los datos capturados en trozos de tamaño constante (1024 bytes). Estos trozos son almacenados en un archivo por cada serie temporal.

Prometheus posee cuatro tipos de métricas:

1. Counter: Esta es una métrica acumulativa que representa un único valor numérico creciente. Este generalmente es utilizado para contar peticiones servidas, tareas, errores, etc. Estos no deben ser utilizados para valores que decrecen, tales como número de procesos, threads, etc.
2. Gauge: Esta métrica representa un valor numérico que puede tanto crecer como decrecer. Estos son utilizados generalmente para valores como temperatura o memoria.
3. Histogram: Este tipo de métrica toma las muestras obtenidas y las cuenta en un *bucket* (balde) de tamaño configurable. De esta manera, se provee un conteo de los valores observados, un contador acumulado de las observaciones y la suma de todos los valores observados.
4. Summary: Similarmente a *Histogram*, *Summary* procesa las muestras en *buckets*, proveyendo además el cálculo de cuantiles (puntos tomados en intervalos regulares de la función de distribución en una variable aleatoria) configurable de cantidades sobre una ventana móvil de tiempo. A través de esto, este puede transmitir φ-cuantiles de eventos observados.

Prometheus provee un sistema de federación, el cual permite a un servidor de Prometheus obtener valores en un periodo de tiempo seleccionado desde otro servidor Prometheus. Esto permite lograr configuraciones escalables de manera horizontal, permitiendo a Prometheus funcionar en millones de nodos. Esta federación funciona de manera jerárquica, donde su estructura es representada como un árbol, donde el nodo más alto en este recolecta las series temporales agregadas de un gran número de servidores subordinados.

Prometheus solo permite la medición de valores numéricos, y solo soporta la medición de strings a través de *labels*. Estas deben ser de una baja cardinalidad, dado que cada combinación llave-valor, representa una serie temporal única. Si se posee una gran cantidad de *labels*, la cantidad de datos almacenados puede aumentar de manera considerable, dado que cada medición representara una nueva medida, y no podrá comprimirse la información obtenida [30].

2.5.2. Druid

Druid [19] es una base de datos open source diseñada para realizar consultas en tiempo real sobre información histórica, proveyendo de un ingreso de baja latencia, exportación de datos flexible y rápida agregación de datos. Su principal uso es sobre consultas de business intelligence (Inteligencia de negocio), escalando a peta bytes de información. Druid comenzó a desarrollarse el año 2011 por la compañía *Metamarkets*, y fue liberada como un proyecto open source el año 2012. Esta base de datos es utilizada en empresas como *Airbnb*, *Paypal* y *Cisco*.

Druid es una base de datos orientada en columnas escrita en Java, donde cada tabla se separa en tres tipos de columnas.

- Columna de marca temporal (Timestamp Column): Guarda la fecha y hora del dato almacenado.
- Columnas de dimensiones (Dimension columns): Esta columna corresponde a strings de atributos de un evento, las cuales son utilizadas generalmente para filtrar la información.
- Columnas de métricas (Metric columns): Estas columnas son utilizadas para realizar agregaciones y cálculos. Estas pueden corresponder a números enteros o punto flotante.

Druid organiza su información en segmentos temporales, los cuales son almacenados de forma auto contenida y comprimida, en conjunto con los índices de las columnas.

Es posible utilizar Druid como un clúster, donde se dividen los nodos en diferentes tipos, cada uno diseñado para realizar un conjunto especifico de tareas.

- Nodo Histórico: Estos nodos guardan un conjunto de segmentos inmutables y responden a las consultas de estos.
- Nodo Agente: Estos nodos son los que interactúan directamente con el cliente para obtener información desde Druid. Además, son responsables de guardar la dirección de los segmentos guardados, distribuir las consultas y agrupar los resultados.
- Nodo Coordinador: Estos nodos manejan los segmentos en los nodos históricos en el clúster, indicando cuando se deben cargar nuevos segmentos, eliminar antiguos, o mover estos para balancear la carga.
- Nodo de procesamiento en tiempo real: El procesamiento en tiempo real de Druid se puede realizar usando estos nodos o utilizando servicios de indexado. Los procesos de tiempo real se refieren al ingreso de datos, indexado de datos (creación de segmentos) y el manejo de segmentos a los nodos históricos.

Para el correcto funcionamiento de un clúster Druid, este requiere como dependencias un servidor Zookeeper para comunicación en el clúster, un almacén de metadatos como MySQL o PostgreSQL y un método de almacenamiento permanente. Esto permite a Druid funcionar de forma altamente disponible (High Availability), donde no existe un punto único de fallo.

2.5.3. ClickHouse

ClickHouse [24] es una base de datos columnar open source diseñado para la generación en tiempo real de reportes analíticos utilizando consultas SQL. Esta base de datos es desarrollada por la compañía *Yandex* desde el año 2009, y fue liberada como un software open source el año 2016. ClickHouse es utilizada por compañías como *CloudFlare* para su producto *Cloudflare DNS Analytics*, realizando el almacenamiento de métricas simples sobre las consultas DNS realizadas a sus clientes.

ClickHouse permite al usuario elegir el motor utilizado por una tabla, donde los más comunes corresponden a la familia de los MergeTree (Estructura que permite el acceso indexado a archivos con un alto rendimiento a grandes volúmenes de inserción de datos).

ClickHouse provee un escalamiento horizontal y tolerancia a fallos, dividiendo el clúster en fragmentos, cada uno con un grupo de réplicas. ClickHouse utiliza una estructura de replicación multi-maestro asíncrono, donde la información es escrita a cualquier replica disponible, y luego distribuida a las réplicas restantes. Este clúster utiliza ZooKeeper para la coordinación de los procesos, pero no para la ejecución ni procesamiento de consultas.

2.5.4. InfluxDB

InfluxDB [17] es una base de datos construida en el lenguaje *Go* para almacenar métricas y eventos (información en forma de series temporales) y realizar análisis en tiempo real sobre estos datos. Influxdb se encuentra desarrollada por *InfluxData* desde el año 2013, como un proyecto open source, y es utilizada por compañías como *IBM*, *Autodesk* y *Cisco*.

InfluxDB organiza las métricas en diferentes series de tiempo, la cuales contienen un valor medido. Cada serie posee cero o más puntos con mediciones de la medida. Estos puntos consisten en una fecha y hora, un valor medido, una serie de *tags* que poseen metadatos de la información, tales como el nombre del servidor, y el valor de la medición.

InfluxDB organiza la información en fragmentos, donde cada uno de estos posee una política de retención, lo cual permite la eliminación de datos antiguos de forma eficiente.

El sistema de almacenamiento de InfluxDB es semejante a un MergeTree. Utilizando un registro de escritura adelantado y una colección de archivos solo lectura, los cuales son ordenados y comprimidos. La información es dividida en fragmentos por cada bloque de tiempo, según una política de retención definida, con un máximo de siete días, lo cual permite la consulta y eliminación de datos en forma eficiente.

InfluxDB solo posee una implementación de clustering, en su oferta comercial *InfluxEnterprise*.

2.5.5. ElasticSearch

ElasticSearch [6] es un motor de búsqueda basado en *Lucene*, escrito en Java y desarrollada por la compañía *Elastic* desde el año 2010. Esta, puede ser utilizada como una base de datos NoSQL, y proveer un sistema distribuido y tolerante a fallos de forma genérica. Este motor fue desarrollado en conjunto con un software de recolección de datos y procesamiento de logs llamado *Logstash*, y una plataforma de visualización llamada *Kibana*. Estos tres productos se denominan *Elastic Stack*. ElasticSearch es utilizada por una gran cantidad de compañías tales como *Wikimedia*, *Facebook*, *GitHub* y *Stack Exchange*.

ElasticSearch se encuentra orientada al almacenamiento de documentos, por lo cual no requiere un esquema de datos fijo. Un esquema puede ser definido para realizar un indexado personalizado, y realizar optimizaciones en el almacenamiento de los datos. Esto permite a ElasticSearch realizar agregaciones complejas para realizar el análisis de una gran cantidad de información.

ElasticSearch provee un escalamiento horizontal y tolerancia a fallos. Esto se realiza, a través de la distribución de los datos en fragmentos, donde cada fragmento posee una o más réplicas. La lectura de los datos puede realizarse en cualquiera de las réplicas, donde la información se encuentra disponible.

2.5.6. OpenTSDB

OpenTSDB [4] es una base de datos de series temporales construida sobre *HBase*. Esta comenzó su desarrollo el año 2007 por la compañía *StumbleUpon*, y el mantenedor actual es *Yahoo!*. Actualmente OpenTSDB es utilizada por compañías como *Clodflare*, *Atlassian* y *Ticketmaster*.

OpenTSDB consiste en un *demonio de series de tiempo* (TSD, por sus siglas en inglés, Time Series Daemon) y un conjunto de utilidades para la línea de comando. Las interacciones con OpenTSDB se logran ejecutando uno o más TSDs, donde cada uno de ellos es independiente y no poseen un estado compartido entre múltiples instancias. Esto permite ejecutar una gran cantidad de TSDs, como sean necesarios para manejar la carga requerida. Cada TSD puede utilizar las bases de datos HBase o Cassandra para almacenar y obtener la información consultada. El esquema de datos se encuentra altamente optimizado para realizar agregaciones de series de tiempo similares, minimizando es espacio de almacenamiento utilizado.

En OpenTSDB, un punto de una serie temporal consiste en:

- Nombre de métrica.
- Fecha y hora en formato UNIX
- Un valor entero, punto flotante, un evento JSON o un histograma.
- Un conjunto de tags (pares llave/valor) que describe a qué serie de tiempo pertenece cada punto.

Los tags deben poseer una baja cardinalidad. Por defecto, solo se puede generar un máximo de 16 millones de IDs para cada métrica, nombre y valor de tag, por lo que no es recomendable utilizar una gran cantidad de estos. Utilizar una alta cardinalidad afectara de manera directa la velocidad de las consultas y el almacenamiento de estos, por lo cual, se recomienda pre-agregar los datos antes de utilizarlos.

OpenTSDB provee un escalamiento horizontal, donde se pueden agregar nuevos nodos de manera directa e independiente, gracias a que cada uno de estos no posee un estado fijo. El almacenamiento que este sistema posee depende directamente de HBase o Cassandra, según haya sido seleccionado, donde ambos pueden ser escalados, para soportar el ingreso de una gran cantidad de datos.

2.6. Visualización de datos

La visualización de los datos capturados permite a un usuario visualizar el estado actual de los servicios de manera rápida y concreta, permitiendo detectar en forma general anormalidades que puedan estar ocurriendo. La selección de un software de visualización se realiza generalmente por su compatibilidad con el sistema de almacenamiento elegido, dado que estos no difieren en gran manera en términos de funcionalidad. Generalmente estos softwares incluyen sistemas básicos de alertas, que permiten de manera visual generar límites entre las diferentes métricas que se poseen.

Para visualizar los datos de los servidores DNS, se realizó el análisis de tres softwares de visualización según su compatibilidad con los diferentes sistemas de almacenamiento analizados anteriormente, y se presentan a continuación.

2.6.1. Kibana

Kibana [7] es una herramienta de visualización open source desarrollada para *Elasticsearch*. Esta provee una visualización sobre un clúster Elasticsearch, donde el usuario puede generar diferentes tipos de gráficos o mapas, para visualizar una gran cantidad de datos. Kibana es un software desarrollado por la empresa *Elastic* desde el año 2013, y se encuentra desarrollado en *JavaScript*.

2.6.2. Grafana

Grafana [21] es un software de visualización y alertas compatible con una gran cantidad de bases de datos, a través de diferentes plugins, los cuales agregan más funcionalidades a este software. Grafana se encuentra escrito en lenguaje *Go* y *JavaScript*, y desarrollado por *Grafana Labs* desde el año 2013, siendo utilizado por empresas como *StackOverflow*, *Intel* y *eBay*.

2.6.3. Graphite Web

Graphite Web [32] comenzó su desarrollo como un proyecto personal el año 2006, el cual fue lanzado por Graphite el año 2008 como una herramienta open source. Graphite Web posee su propia base de datos, para el almacenamiento de series de tiempo, y además provee conexiones a bases de datos externas para la generación de visualizaciones. Actualmente Graphite Web se encuentra escrito en lenguaje Python utilizando el framework *Django*, y es utilizado por empresas como *Lyft*, *Reddit* y *GitHub*.

Capítulo 3

Definición del Problema y Análisis de Softwares Open Source

3.1. Especificación del problema

El problema principal a resolver consiste en la extracción de diferentes métricas desde los servidores DNS para su visualización de manera rápida y eficiente.

Dada la gran cantidad de información con la cual se debe trabajar para realizar el monitoreo efectivo de DNS, se dividió el problema en tres partes: Captura, Almacenamiento y Visualización. A continuación, se muestran las métricas definidas a extraer, y los problemas y requisitos que posee cada una de las partes definidas anteriormente.

3.1.1. Métricas DNS a medir

Las métricas a medir sobre el DNS se definieron en base a las métricas generadas por *DSC* [2], software utilizado actualmente por *NIC Chile* para realizar el monitoreo de sus servidores DNS. Las métricas que se medirán a través de la captura de paquetes son las siguientes.

- Los 5 dominios más consultados.
- Número de paquetes por subred.
- Número de paquetes DNS que poseen habilitado EDNS.
- Largo promedio de los paquetes de consulta DNS.
- Número de paquetes por código de operación (OPCODE).
- Número de paquetes por clase de consulta (QCLASS).
- Número de paquetes por tipo de consulta (QTYPE).
- Número de paquetes por código de respuesta.
- Número de paquetes por protocolo de transporte (UDP o TCP).

3.1.2. Captura de datos

El proceso de captura de datos no debe implicar la modificación del funcionamiento actual del software utilizado para servir las consultas DNS, dado los posibles errores que se pueden generar en el desarrollo de este tipo de aplicaciones, y su impacto en internet en caso de errores.

Dada esta limitación, la captura de los paquetes DNS se debe realizar a nivel de la interfaz de red, ya sea desde el mismo servidor o uno cercano a este como se describe en el punto 2.4. La captura debe disponer suficiente información para medir las métricas definidas posteriormente, y debe adaptarse ya sea de forma genérica, o desarrollando las partes faltantes, al sistema de almacenamiento definido.

Dado que esta captura se realiza a través de la interfaz de red, es necesario que el proceso de captura soporte los diferentes protocolos por los cuales es posible transmitir información DNS, es decir, el protocolo de red IPv4 e IPv6 y los protocolos de transporte TCP y UDP.

3.1.3. Almacenamiento

El sistema de almacenamiento debe ser capaz de almacenar las métricas definidas utilizando la menor cantidad de recursos posible, ya sea utilización de CPU, memoria RAM y disco duro. Por otro lado, el tiempo de consulta debe ser el menor posible, para así tener una visualización lo más actualizada posible en todo momento.

El sistema de almacenamiento debe poder escalar a futuro para soportar una carga mayor, y debe priorizar la disponibilidad y tolerancia a fallos antes de la consistencia, permitiendo su funcionamiento continuo en un ambiente distribuido según el teorema CAP [34].

Por último, se debe poder consultar de manera continua todas las métricas definidas en la última hora, con una resolución y periodicidad mínima de 1 minuto.

Para verificar el cumplimiento de estos requerimientos, se debe realizar una comparación entre los sistemas de almacenamiento seleccionados, a través de un benchmark que simule la carga esperada de estos sistemas, utilizando como base las métricas que se definieron anteriormente, observando su comportamiento en relación a los tiempos de respuesta, uso de cpu, memoria y disco duro.

3.1.4. Visualización

La visualización de los datos debe lograr mostrar todas las métricas definidas a lo largo de una ventana de tiempo definida por el usuario, la cual debe soportar como mínimo la última hora. El sistema de visualización debe adaptarse al sistema de almacenamiento, desarrollando los componentes faltantes para su correcto funcionamiento.

3.2. Análisis de softwares a utilizar

Para realizar una correcta selección de los softwares a utilizar para desarrollar el monitoreo de los servidores DNS, se generó un análisis de las diferentes características de cada uno de estos, realizando comparaciones según las especificaciones definidas en el capítulo anterior.

3.2.1. Captura de datos

Para realizar la extracción de la información desde los servidores DNS, se evaluaron cinco softwares open source que realizan una captura de paquetes directamente desde la interfaz de red. En estos, se buscó la compatibilidad con todos los protocolos utilizados por DNS (IPv4, IPv6, TCP y UDP) y la posibilidad de extracción de las métricas definidas en el punto 3.1.1.

- Fievel [22]: Este software captura y permite procesar los paquetes de manera individual, sin embargo, este no soporta el procesamiento de paquetes IP fragmentados y el protocolo TCP.
- Packetbeat [9]: Este software logra capturar los paquetes DNS y reconstruye los flujos de los protocolos TCP y UDP. Al realizar pruebas de este software sobre una alta carga de paquetes, se encontró que este añade una gran carga al sistema, dado que este realiza la obtención de la mayor cantidad de información posible, produciendo finalmente que un 90 % de los paquetes recibidos no sean analizados. Por otro lado, este software no soporta la desfragmentacion de paquetes IP, lo cual produce que los paquetes fragmentados no sean analizados.
- Collectd [28]: Se realizó el análisis sobre este software y se determinó que no logra generar las métricas definidas, logrando soportar solo el código de operación, tipo y código de respuesta. Además de esto, solo soporta la captura a través del protocolo UDP.
- DNS Statistics Collector [2]: Este software logra generar la mayoría de las métricas que se definieron, lo cual lo hace apto como una solución para la extracción de información. Sin embargo, los datos que este software entrega son pre-procesados, mostrando la agregación de estos cada cierto periodo de tiempo definido por el usuario. Esto hace que realizar un análisis de eventos a futuro sea un trabajo más difícil, dado que no se posee la información unitaria de las consultas.
- gopassivedns [29]: Este software realiza la captura de información básica de los paquetes DNS, soportando los protocolos TCP y UDP. Sin embargo, no logra generar las métricas que se definieron y no soporta la desfragmentacion de paquetes IP.

Dado el análisis realizado y las características de cada capturador mostradas en la tabla 3.1, se logró apreciar que ninguno de los software analizados permite la generación de las métricas requeridas soportando los protocolos necesarios, por lo cual se decidió desarrollar una solución a la medida basada en *gopassivedns* y *Packetbeat*, donde se aprovechara la seguridad del lenguaje *Go* para trabajar sobre información externa potencialmente peligrosa y las diferentes librerías que esta posee para realizar la captura e interpretación de los paquetes DNS, implementando la desfragmentación de paquetes *IPv4* e *IPv6* para así soportar todas

las formas en que un cliente interactúa con un servidor DNS.

Este desarrollo tomo como base principal para su implementación a *gopassivedns* [29], dado que este se encuentra totalmente enfocado al monitoreo de DNS, permitiendo así poseer una baja complejidad y un alto rendimiento. Esto permitió tomar las funciones y estructuras más importantes, e implementar las funcionalidades faltantes sin agregar una gran complejidad.

	IPv4	IPv4 Fragmentado	IPv6	IPv6 Fragmentado	TCP	UDP	Datos crudos
Fievel	x		x			x	x
Packetbeat	x		x		x	x	x
collectd	x		x			x	
dsc	x	x	x	x	x	x	
gopassivedns	x				x	x	x

Tabla 3.1: Tabla que marca la disponibilidad de diferentes características que son deseadas de los capturadores de información DNS.

3.2.2. Almacenamiento de datos

El sistema de almacenamiento a utilizar debe adecuarse a las métricas que serán analizadas, proveyendo una utilización de recursos y velocidad de respuesta suficiente para soportar la carga esperada.

Para seleccionar este de manera adecuada, se desarrollará un benchmark a través del cual se obtendrán mediciones para verificar la factibilidad del uso de los diferentes sistemas de almacenamiento, y los recursos necesarios para su utilización, buscando la más apta para realizar el almacenamiento de las métricas definidas.

3.2.2.1. Benchmark de sistemas de almacenamiento

En este benchmark, se desarrollarán pruebas utilizado diferentes softwares de base de datos, obteniendo mediciones sobre su utilización de recursos y velocidad de respuesta a consultas sobre los datos agregados, utilizando información con estructuras semejantes a las esperadas luego de realizar la integración con las capas de captura y visualización.

3.2.2.1.1. Bases de datos a evaluar

Las bases de datos a evaluar, en conjunto con su versión se pueden observar a continuación.

- Prometheus 1.7.1
- Druid 0.9.1.1
- ClickHouse 1.1.54292

- InfluxDB 1.3
- ElasticSearch 5.5.1
- OpenTSDB 2.3.0

3.2.2.1.2. Datos de prueba

Los datos de prueba generados por este benchmark fueron desarrollados basándose en las métricas que se definieron anteriormente, estableciendo estructuras similares a las esperadas por la capa de captura y visualización. Los datos de prueba se limitaron a tres conjuntos, con el fin de visualizar y comparar el manejo de información de alta y baja cardinalidad, los cuales se muestran a continuación.

- 5 Dominios más consultados: Se realizará la generación de 300.000 nombres de dominio utilizando 10 caracteres aleatorios, agregando el sufijo *.cl* a cada uno de estos. Se generarán consultas tal que se obtendrán los 5 dominios más consultados en ventanas de un minuto en los últimos 10 minutos.
- Máscaras de red IPv4: Se generarán en forma aleatoria direcciones IPv4 clase A en el formato *X.0.0.0*, tal que X es un numero entre 0 y 255. Se realizará una consulta tal que se obtenga la cantidad de consultas por cada máscara de red en los últimos diez minutos de ejecución.
- Largo de paquete de respuesta: Se generarán tamaños de paquetes DNS aleatorios entre 10 y 500 bytes, y luego se consultará la suma de estos largos en ventanas de un minuto en los últimos diez minutos de ejecución.

A través de estos conjuntos de datos, se espera simular la estructura básica de los datos que serán insertados y consultados en las bases de datos a evaluar.

3.2.2.1.3. Mediciones a realizar

Para cada uno de estos conjuntos de datos, se realizará la obtención de las siguientes métricas para realizar la comparación entre los diferentes sistemas.

- Tiempo CPU utilizado: Tiempo utilizado en la CPU por el sistema de almacenamiento.
- Cantidad de memoria principal utilizada: Memoria residente ocupada por los procesos almacenada en RAM.
- Cantidad de memoria secundaria utilizada: Memoria ocupada por los procesos almacenada en disco duro.
- Tiempo requerido para consultar datos: Tiempo entre el inicio de una consulta sobre los datos y la obtención de la respuesta. La consulta varía según el conjunto de datos utilizado.

Las mediciones de estas métricas serán realizadas de forma periódica según la siguiente lista.

- Tiempo CPU utilizado: Cada 10 segundos.

- Cantidad de memoria principal utilizada: Cada 10 segundos.
- Cantidad de memoria secundaria utilizada: Cada 10 segundos.
- Tiempo requerido para consultar datos: Cada 1 minuto.

Por último, se realizará una verificación de los datos insertados, contando la cantidad de elementos insertados y comparándolos con los esperados, para así observar posibles pérdidas de información o errores en la inserción de elementos.

3.2.2.1.4. Carga de prueba

Se realizará una medición inicial realizando el ingreso de *3.000* datos por segundo, cantidad de paquetes cercana a la recibida actualmente por *NIC Chile*. Esta prueba poseerá una duración de 6 horas, con un total de 21.600.000 datos ingresados al sistema de almacenamiento. A través de esta prueba, se espera descartar los sistemas que no poseen el desempeño suficiente para ser utilizados como monitores para DNS.

En caso de encontrar dos o más sistemas de almacenamiento con un desempeño semejante, se repetirán las pruebas con *5.000* y *10.000* datos por segundo utilizando el conjunto de datos "*5 Dominios más consultados*" para observar el comportamiento de esto en presencia de una carga mayor.

3.2.2.1.5. Ejecución de pruebas

Para realizar las pruebas, se desarrolló una herramienta [10] que realiza la ejecución de los distintos sistemas de almacenamiento y todos los softwares necesarios para su correcto funcionamiento. Esta realiza la ejecución de estas a través de Docker para así obtener ambientes independientes y reproducibles.

Las pruebas fueron ejecutadas en un computador con las siguientes especificaciones.

- CPU: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz.
- Núcleo(s): 2.
- Hilo(s) por núcleo: 2.
- Memoria principal: 8GiB SODIMM DDR3 Síncrono 1600 MHz M471B1G73QH0-YK0.
- Memoria secundaria: 931GiB ATA HGST HTS541010A9.
- Sistema Operativo: Ubuntu 14.04.3 LTS
- Arquitectura: x86_64.
- Versión Docker: 17.06.0-ce, build 02c1d87.

3.2.2.1.6. Resultados

El benchmark que produjo la mayor cantidad de variaciones corresponde al que utiliza el conjunto de datos "*5 Dominios más consultados*", donde algunas bases de datos no lograron un rendimiento suficiente para soportar la carga entregada. Podemos ver esto en el tiempo de consulta que se muestra en la figura 3.1. Los otros resultados de las pruebas y mediciones se pueden observar en el Apéndice 1: Resultado benchmarks.

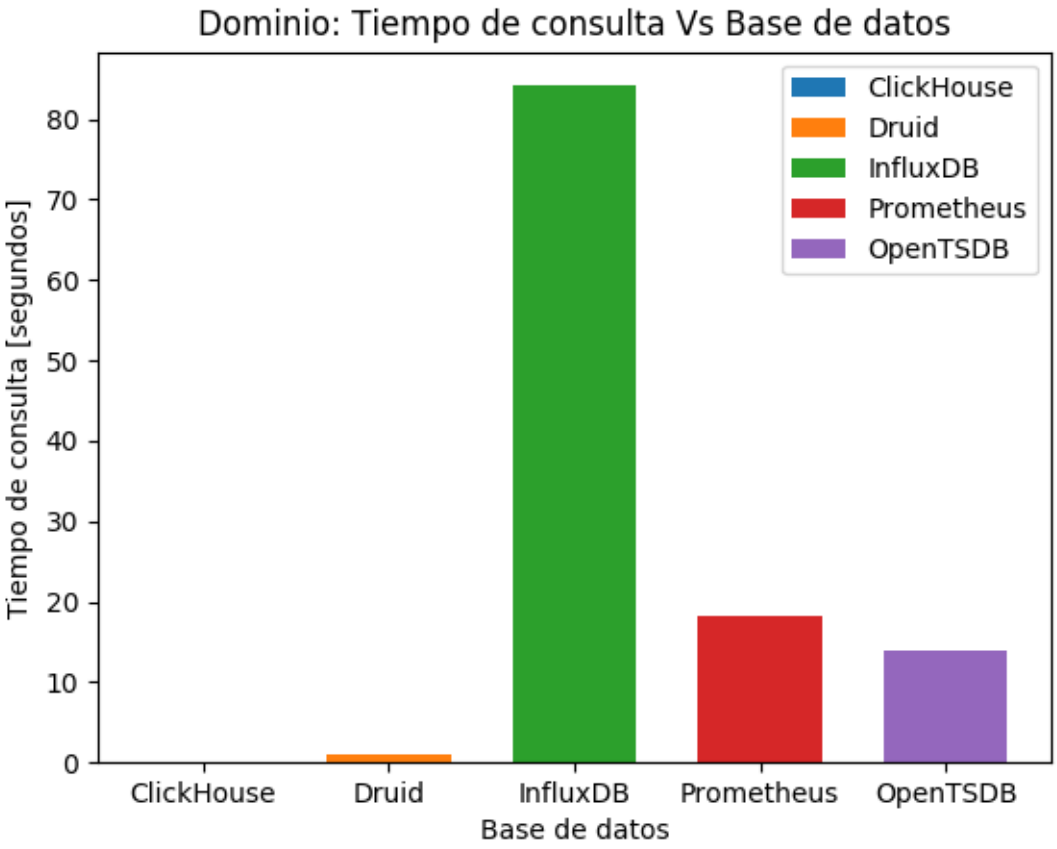

Figura 3.1: Gráfico que muestra el tiempo promedio requerido para obtener el resultado del conjunto de datos *5 dominios más consultados* utilizando los datos de las últimas dos horas de ejecución. Se puede observar que Druid y ClickHouse lograron responder en un tiempo mucho menor en comparación a las otras bases de datos. Por otro lado, ElasticSearch no se ha agregado dado que esta no realizó respuestas de las consultas luego del minuto 200 de ejecución.

A través de los resultados obtenidos, logramos observar que las bases de datos que mejor manejan la información con la cual se trabajara corresponden a Druid y ClickHouse, por lo cual se repitieron las pruebas aumentando la carga a *5.000* y *10.000* datos por segundo para realizar una mejor comparación entre ellas, la cual se puede apreciar en la figura 3.2.

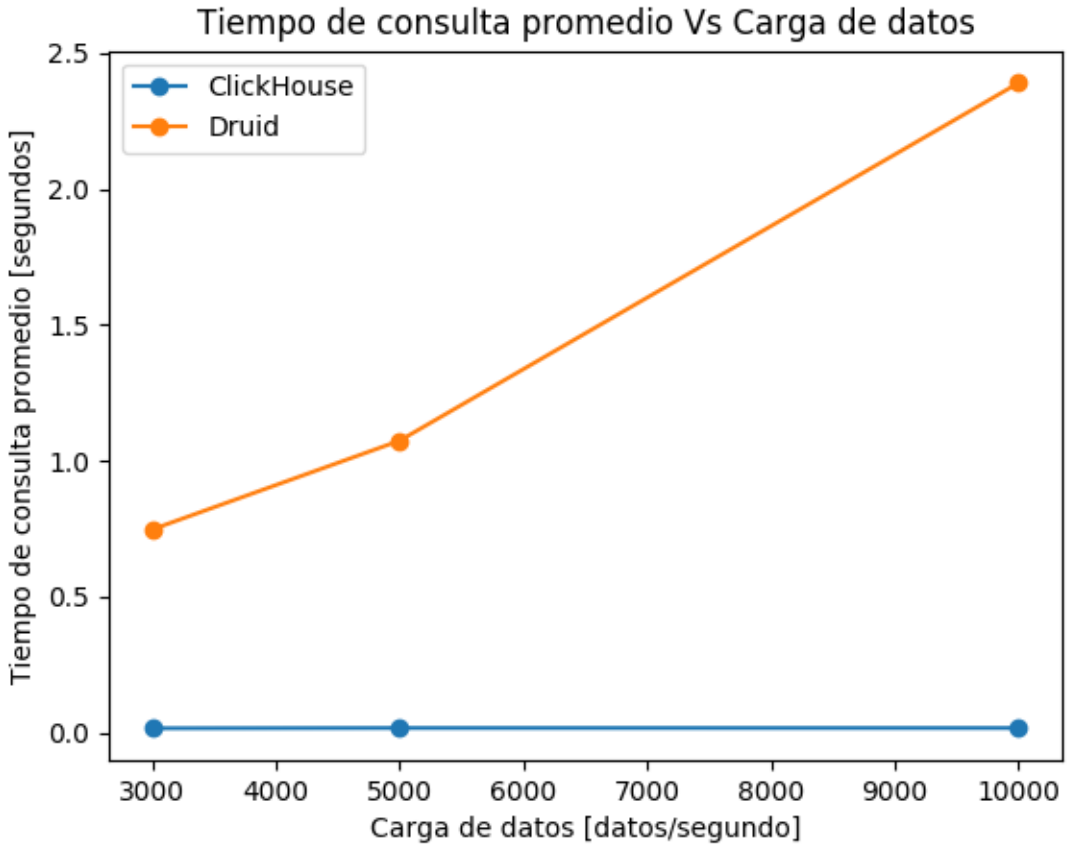

Figura 3.2: Línea de tendencia sobre el tiempo de consulta para diferentes cargas de datos por segundo, obteniendo el tiempo de consulta promedio de las últimas dos horas de ejecución de los benchmark. Se puede apreciar que el tiempo de consulta se mantiene prácticamente constante para *ClickHouse*, en comparación con *Druid* que aumenta de manera exponencial.

3.2.2.2. Análisis de resultados

Utilizando la información obtenida en los benchmarks realizados, logramos apreciar que Druid y ClickHouse logran soportar la información que deseamos almacenar, donde ClickHouse logra escalar de una mejor manera ante el aumento de la información ingresada.

Por otro lado, comparando la arquitectura de ambas bases de datos, podemos ver que Druid presenta una mayor complejidad operacional, donde esta se encuentra dividida en cinco servicios que interactúan entre sí, además de necesitar la mantención de un cluster Hadoop, ZooKeeper y MySQL. Por último, Druid solo presenta una interfaz SQL de manera experimental, la cual no podemos utilizar de manera confiable para realizar el análisis de la información.

ClickHouse funciona utilizando un único binario de manera centralizada, solo requiriendo un cluster ZooKeeper para utilizar tablas replicadas de manera consistente. Esto permite un manejo operacional mucho más sencillo, donde agregar un nuevo nodo solo involucra iniciar un nuevo proceso ClickHouse, realizando las configuraciones pertinentes. Además de esto,

ClickHouse presenta una interfaz SQL muy semejante a otras bases de datos más populares tales como MySQL y PostgreSQL, permitiendo la creación de tablas y consultas utilizando el mismo lenguaje. Esto además permite generar vistas que realicen agregaciones de la información recibida sin necesitar la modificación del software de captura de datos.

Por los motivos mencionados, se decidió trabajar con ClickHouse, para así aprovechar su simplicidad para el almacenamiento de los datos y su desempeño como lo demostró en los benchmarks realizados, además de otras características deseables que esta posee tales como federación, replicación, distribución y sharding, lo cual permite una gran escalabilidad, disponibilidad y tolerancia a fallos.

3.2.3. Visualización

Para realizar la selección del sistema de visualización a utilizar, se realizó un análisis entre los sistemas de almacenamiento y su compatibilidad con los sistemas de visualización, el cual se puede observar en la tabla 3.2.

	Prometheus	Druid	ClickHouse	InfluxDB	ElasticSearch	OpenTSDB
Kibana	x				x	
Grafana	x	x	x	x	x	x
Graphite				x		x

Tabla 3.2: Tabla que muestra la compatibilidad entre los sistemas de almacenamiento y los sistemas de visualización analizados.

Dado el análisis realizado, para realizar la visualización de la información almacenada en ClickHouse, se decidió utilizar el software *Grafana* [21], dada la ya existencia de implementaciones que permiten realizar la conexión entre estos softwares, además de la amplia compatibilidad que esta posee con otros sistemas de almacenamiento.

Luego de realizar pruebas de la conexión entre estos, se detectó que la versión actual no posee compatibilidad con el sistema de alertas de *Grafana*, por lo cual será necesario generar una versión modificada de esta, implementando de manera directa la conexión a ClickHouse, en vez de utilizar el sistema de plugins externos que esta posee.

Capítulo 4

Implementación de la Solución

4.1. Trabajo a desarrollar

A través del análisis realizado, logramos determinar que el trabajo a desarrollar corresponde a los siguientes puntos.

- Desarrollo de aplicación de captura de paquetes DNS desde interfaz de red basada en *gopassivedns*.
- Integración de software de captura con base de datos ClickHouse.
- Implementación de reducción de información en ClickHouse con el fin de mostrar las métricas definidas en el punto 3.1.1, ya sea a través de consultas directas o vistas a una tabla con datos sin procesar.
- Generar versión modificada de *Grafana* compatible con ClickHouse y con el sistema de alertas habilitado para esta.

4.2. Arquitectura de la solución

La arquitectura definida para realizar el monitoreo se estableció en tres capas: *Captura*, *Almacenamiento* y *Visualización*. Esta división permite escalar de manera horizontal cada capa de manera independiente, permitiendo incrementar los recursos solo los servicios que poseen problemas procesando los datos. Por otro lado, esta arquitectura permite separar los roles y características de cada uno de los servicios que se utilizan, donde por ejemplo, se permite al sistema continuar funcionando aun cuando la capa de captura o visualización deje de funcionar.

El punto más importante de esta arquitectura se centra en el almacenamiento de los datos, dado que, en caso de una falla en esta, los servicios de captura y visualización dejaran de funcionar, por lo que asegurar su disponibilidad pasa a ser de gran importancia para este sistema.

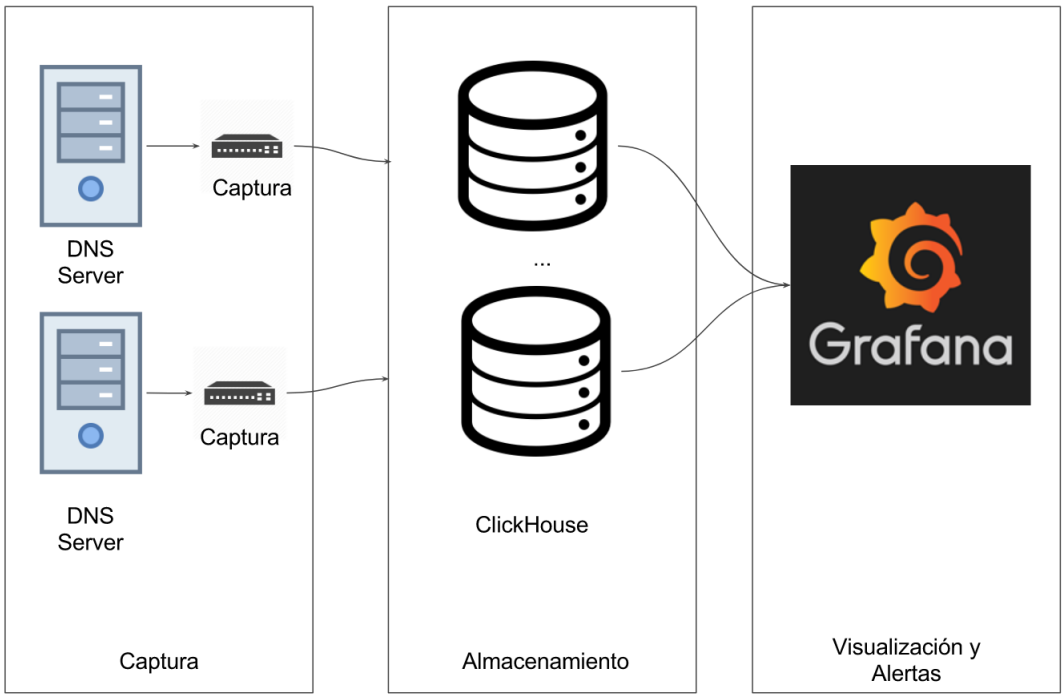

Figura 4.1: Arquitectura de la solución a desarrollar, dividiendo esta en tres partes, *Captura*, *Almacenamiento* y *Visualización*. Esta división permite una separación clara de roles para así realizar un buen escalamiento de cada capa según sea necesario.

4.3. Implementación

El desarrollo se dividió según las tres capas definidas en la arquitectura, Captura, Almacenamiento y visualización, en las cuales se obtuvieron los siguientes resultados.

4.3.1. Captura de datos

Para realizar la captura de la información, se desarrolló una librería llamada *dnszeppeling* [12], la cual utilizando la librería *pcap* realiza una captura de los paquetes enviados y recibidos desde una interfaz de red.

Esta librería se encarga de realizar la deserialización de la información recibida, iniciando desde los protocolos de red IPv4 e IPv6, realizando el re-ensamblaje en caso de recibir paquetes fragmentados para ambos protocolos. Luego de esto se realiza la deserialización de los protocolos de transporte TCP y UDP, soportando el re-ensamblaje del flujo TCP almacenando de manera temporal el estado de las conexiones según se observa dado los paquetes capturados.

Para realizar el envío de la información a la base de datos ClickHouse, se implementó sobre esta librería una aplicación llamada *dnszeppeling-clickhouse* [11], la cual obtiene los paquetes DNS ya procesados y procede a enviar en baches la información de estos para su almacenamiento.

4.3.2. Almacenamiento y agregación

La información obtenida desde la capa de captura es almacenada en una tabla central, en la cual la información es almacenada en diferentes columnas. Esta tabla utiliza el motor *MergeTree* para estructurar la información utilizando como índice de partición la fecha de consulta, lo cual permite a la base de datos optimizar la estructura de almacenamiento para obtener información de fechas específicas de manera rápida y a bajo costo. Por otro lado, esto permite un control sobre el tiempo de vida de la información almacenada, donde es posible eliminar información antigua a partir de su fecha de generación.

Para realizar la agregación de la información, se implementaron diferentes vistas materializadas sobre la tabla principal, lo cual permite trabajar sobre información ya procesada y presentar a la capa de visualización la información de manera mucho más simple y rápida al no requerir realizar el procesamiento de la información en cada consulta. Por otro lado, realizar esta separación nos permite utilizar motores de tablas especializados según el tipo de información con la cual se trabaja, entre los cuales podemos encontrar motores de suma, media, conteo de información única, entre otros.

Por último, la utilización de vistas nos permite desacoplar las capas de almacenamiento y visualización, al utilizar interfaces diferentes desde donde realizamos la inserción de la información y donde obtenemos las métricas agregadas de esta. Esto nos permite realizar cambios en la capa de captura, tal como agregar nueva información, sin afectar a la capa de visualización, donde solo se deben respetar las interfaces ya definidas.

Las consultas para generar la tabla principal y las vistas materializadas se pueden ver en el Apéndice 2: Consultas para generación de base de datos.

4.3.3. Visualización

Para permitir la visualización de la información en *Grafana* y utilizar del sistema de alertas, se tomó un plugin ya desarrollado que genera la conexión entre *Grafana* y *ClickHouse*, integrándose directamente en el binario de *Grafana*. Este plugin [20] fue actualizado de la versión *4.1.1* a la versión *4.5.2* [13] de Grafana para obtener las mejoras y actualizaciones generadas entre estas versiones.

Luego de esto, se procedió a generar un panel que muestre las diferentes métricas definidas en el punto 3.1.1, generando las diferentes consultas a las vistas materializadas con la información ya procesada. Las consultas que se realizan se pueden ver en el Apéndice 3: Consultas para obtención de métricas para Grafana.

4.3.4. Integración y ejecución

Para realizar la integración y prueba de los servicios, se utilizó la plataforma *Docker* para generar los tres contenedores que encapsularán los servicios de captura (*dnszeppelin-*

clickhouse), almacenamiento (*ClickHouse*) y visualización (*Grafana*).

Con el fin de permitir la comunicación entre estos contenedores, se utilizó *Docker Compose* [16] para realizar la definición de los servicios prestados por los contenedores, estableciendo la arquitectura básica para el correcto funcionamiento del monitoreo.

Utilizar *Docker* nos permite ejecutar la aplicación en diferentes ambientes sin depender del sistema en el cual se encuentra instalada la aplicación, lo cual facilita la realización de pruebas del sistema antes de realizar un paso a sistemas de producción.

Para ambientes productivos, es posible utilizar los contenedores o los binarios de cada capa de manera independiente, permitiendo mover cada uno de estos a distintos servidores, para así lograr escalar cada uno de estos según sea necesario. De esta manera, es posible asegurar el funcionamiento del sistema ante errores en los servidores en los cuales se encuentran instalados.

Las instrucciones para iniciar los servicios a través de *Docker Compose* se pueden encontrar en el Apéndice 4: Ejecución de los servicios a través de Docker Compose

4.4. Pruebas de funcionamiento

4.4.1. Datos de prueba

Para la realización de las pruebas de funcionamiento, se obtuvo una captura de datos de los servidores de NIC Chile generada el año 2016, la cual nos permite reproducirla en una interfaz local para realizar la simulación de emisión y recepción de paquetes.

4.4.2. Validación de captura

Para realizar la validación del proceso de captura y almacenamiento, se realizó la ejecución en paralelo del sistema de monitoreo desarrollado y el sistema de recolección y procesamiento de información actualmente utilizado por NIC Chile *dsc* [2].

Luego de realizar la reproducción de los datos de prueba por una hora, se realizó la comparación del número de paquetes capturados, tipo de consulta y respuesta capturados por cada sistema, con lo cual se encontró que los valores coincidían perfectamente. Gracias a esto, logramos verificar que el sistema de captura y almacenamiento logran procesar los paquetes de manera correcta.

4.4.3. Prueba de inserción paralela de datos y visualización

Para verificar el correcto funcionamiento del sistema desarrollado ante la arquitectura en producción esperada, se realizó la ejecución de múltiples procesos de captura para realizar la simulación de diferentes servidores insertando información de manera paralela. Esta prueba produjo los resultados presentados en la tabla 4.1.

Tiempo de ejecución	36 Horas
Paquetes por segundo	7.000 pps
Cantidad total de paquetes almacenados	927.000.000
Tamaño total de información almacenada sin comprimir	52.1 GB
Tamaño total de información almacenada comprimida	7.1 GB
Tamaño de información almacenada comprimida por paquete	8.3 bytes

Tabla 4.1: Resultado de pruebas realizadas reproduciendo una captura de datos real de NIC Chile. Los valores se encuentran aproximados para una mejor visualización.

A través de los resultados presentados en esta tabla, podemos apreciar que el sistema logra realizar una buena compresión de los datos almacenados, permitiendo reducir el espacio utilizado en disco en un total de 86.3 %, lo cual nos muestra una alta compresión de los datos.

Además de esto, se realizó la visualización en vivo de los datos para verificar el soporte de la cantidad de datos ingresada, presentando tiempos de actualización menores a los 5 segundos para una visualización de la última hora de ejecución refrescándose cada 30 segundos. Se puede apreciar el resultado final del proceso de captura en la figura 4.2 y la figura 4.3, donde se muestra el resultado de dos horas de captura de la prueba realizada.

Figura 4.2: Visualización de una ventana de dos horas luego de finalizar la reproducción de los datos de prueba, mostrando los dominios más consultados, cantidad de dominios únicos consultados, tamaño medio y total de paquetes, número de paquetes por versión IP y por protocolo de transporte.

Figura 4.3: Visualización de una ventana de dos horas luego de finalizar la reproducción de los datos de prueba, mostrando la cantidad de paquetes por prefijo *IPv4* e *IPv6*, tipo y clase de consulta, código de operación y respuesta.

4.4.4. Prueba de carga por inundación

Utilizando la herramienta *dns-flood* [37], se realizó una inundación de paquetes DNS realizando el envió de consultas a la interfaz local de la forma más rápida posible. En el mismo equipo se realizó la captura de datos, enviando la información capturada a un segundo equipo para su almacenamiento y visualización.

El proceso de captura de datos reportó una captura de aproximadamente 70.000 paquetes por segundo, aumentando a 120.000 si se realiza la ejecución de un segundo proceso de la herramienta *dns-flood*. No se realizaron pruebas con una mayor cantidad de procesos dado que el equipo no poseía recursos suficientes para ejecutar un nuevo proceso de *dns-flood* sin llevar la utilización de CPU sobre el 100 %.

En la figura 4.4 logramos apreciar como aumenta de manera drástica las consultas a uno de los dominios, lo cual nos permite detectar este tipo de ataque de manera rápida. Si fueran realizarse consultas de dominios con nombres aleatorios, se logrará observar el cambio en el gráfico que muestra el conteo de dominios únicos consultados, con lo cual logramos observar características básicas del flujo que se está recibiendo.

Figura 4.4: Visualización de los dominios más consultados ante una inundación de paquetes con un único dominio. Se realizó el envío de 70.000 paquetes por segundo, los cuales fueron almacenados y procesados en conjunto a la reproducción de los datos de prueba.

4.4.5. Prueba de alertas

Para probar el sistema de alertas, se estableció una alerta que se activará cuando el promedio de la cantidad de paquetes recibidos en los últimos cinco minutos sea mayor a 5.000 paquetes por segundo, realizando el envío de un mensaje a través de *telegram* (Version 0.6.0; Telegram Messenger LLP; Telegram Messenger LLP, noviembre 2017). Luego de aumentar la carga por sobre este valor, se logró apreciar la activación de la alerta como se muestra en la figura 4.5 y se recibió un mensaje como se muestra en la figura 4.6 con el inicio y fin del evento.

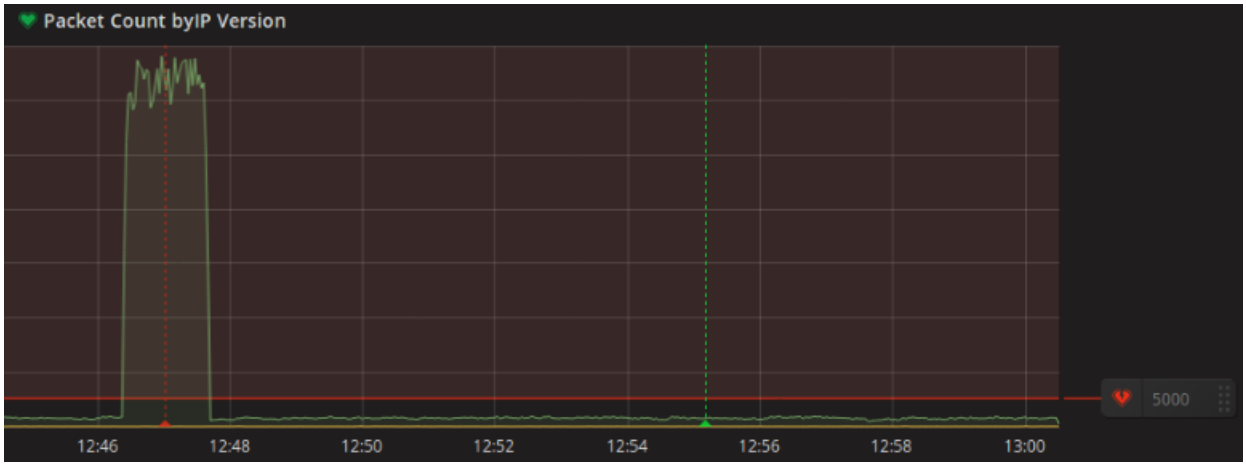

Figura 4.5: Visualización de una alerta detectada en *Grafana* al recibir una cantidad mayor a 5.000 paquetes por segundo en promedio en los últimos cinco minutos, donde el inicio de la alerta se estableció a las 12:47 y finalizo a las 12:55, representando el inicio con una línea vertical roja, y el fin con una línea horizontal verde.

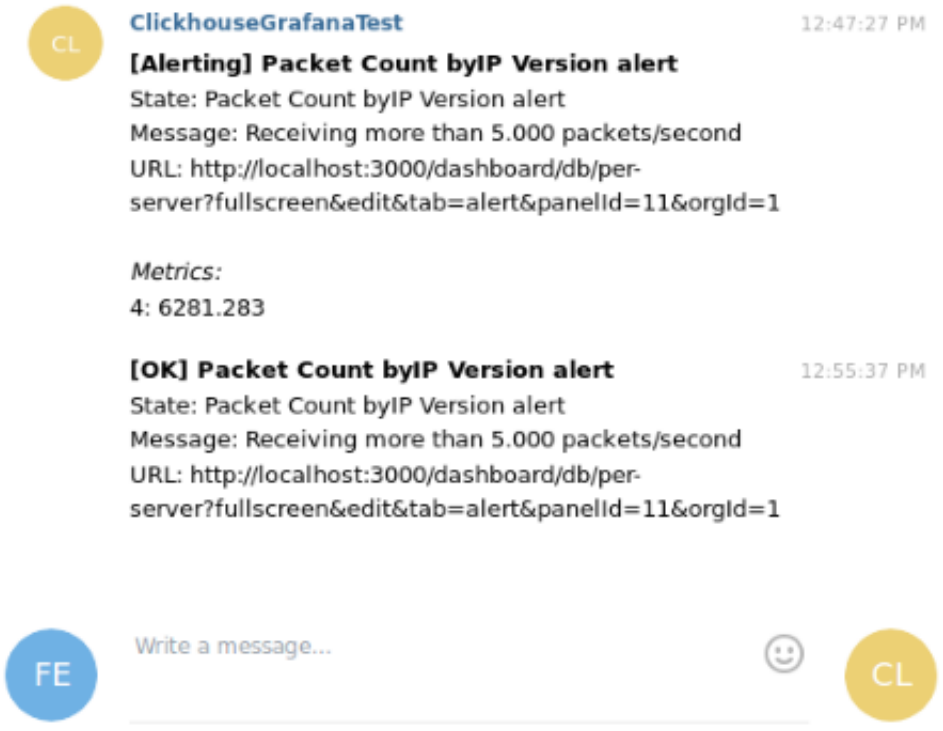

Figura 4.6: Mensajes recibidos al inicio y fin de la alerta, mostrando la etiqueta de la métrica que pasa sobre el limite definido por la alerta.

4.4.6. Utilización de ancho de banda y encriptación

Actualmente, la aplicación de captura de datos *dnszeppelin-clickhouse* no realiza la compresión y encriptación de los datos, por lo cual cuando el sistema de captura y almacenamiento se encuentran en servidores separados, los datos se transmiten de manera transparente en la

red que lo conecta.

Realizando las pruebas de la aplicación sobre la inundación de paquetes, se encontró que al utilizar un túnel *ssh* comprimido entre los servidores de almacenamiento y captura de datos, se logra reducir el ancho de banda necesario en un 99.07 % (26 MBit/s a 246 kBit/s), logrando mejorar en gran manera la velocidad de transferencia de los datos, disminuyendo el ancho de banda e impacto en la red que este proceso produce, además de la seguridad provista por la encriptación de los datos. Esta disminución se produce dada la baja entropía en los datos de la inundación, lo cual permite comprimir los datos en gran manera tal como muestran los resultados.

Se logro observar que el impacto en la utilización de recursos de realizar este proceso es mínimo, utilizando aproximadamente un 5 % de tiempo de procesamiento, por lo cual se recomienda utilizar este método para asegurar la transferencia de los datos de manera segura y disminuir el impacto en la red utilizada.

Capítulo 5

Conclusión

El trabajo realizado en esta memoria busco definir, analizar e integrar diferentes softwares open source para la realización de un monitoreo en tiempo real de servidores DNS de alta carga, permitiendo visualizar el estado de los servicios en todo momento y alertar ante posibles anormalidades.

Para esto, inicialmente se definieron las métricas principales a medir desde los servidores DNS, basándose en trabajos relacionados que buscan cumplir objetivos similares. Además de esto se definieron nuevas métricas que logren dar una mayor visión del comportamiento actual del sistema.

El proceso de monitoreo se dividió en tres capas, captura, almacenamiento y visualización, donde se definieron y analizaron diferentes requerimientos para los softwares de cada una de las etapas.

En primer lugar, se realizó el análisis de cinco diferentes softwares para realizar la captura de información desde los servidores DNS, determinando su capacidad para extraer las diferentes métricas que se definieron. A través de este análisis, logramos observar que ningún software open source lograba extraer todas las métricas definidas, por lo que se determinó la necesidad de desarrollar una nueva solución basada en las ya existentes para extender sus funcionalidades y cubrir las diferentes métricas importantes de los servidores DNS.

El segundo análisis desarrollado corresponde a la capa de almacenamiento, para el cual se desarrolló un benchmark sobre seis distintos sistemas de almacenamiento. A través de este benchmarks se logró determinar el rendimiento de estos sistemas para los datos que se esperan generar por las métricas definidas, con lo cual se determinó que el sistema de almacenamiento más apto para nuestro caso de uso corresponde a *ClickHouse* [24].

El análisis final correspondió al sistema de visualización, en el cual se buscó una compatibilidad con la visualización de las métricas definidas, donde se seleccionó para su utilización a *Grafana* [21], dado que existen implementaciones de este software compatibles con el sistema de almacenamiento *ClickHouse*. Además de esto, se realizó la modificación de la implementación actual para permitir al sistema de alertas de *Grafana* funcionar con el sistema de

almacenamiento *ClickHouse*, funcionalidad que no se encontraba implementada.

Luego de realizar este análisis, se procedió a realizar el desarrollo del software de captura de información *dnszeppeling*, con el cual se lograron capturar todas las métricas definidas en la etapa de análisis, a través de los diferentes protocolos que deben ser compatibles con el sistema DNS. Además de esto, se realizó la integración con el sistema de almacenamiento *ClickHouse*, generando tablas para almacenar la información generada, y el sistema de visualización *Grafana*, generando un panel que permite la visualización de la información de manera agregada y establecer alertas para detectar anormalidades.

Para validar el funcionamiento del sistema, se realizaron pruebas sobre diferentes tipos de carga, tales como tráfico normal de un servidor DNS y pruebas de carga realizando consultas lo más rápido posible, demostrando el correcto funcionamiento del sistema por sobre treinta veces a la carga normal esperada por estos servidores.

Considerando lo anterior, se logró cumplir el objetivo general de realizar el monitoreo de un sistema DNS, manejando la gran carga que es generada por estos sistemas y alertando en caso de anormalidades. Sin embargo, dado a restricciones de tiempo, aún es posible realizar mejoras al sistema que permitan un análisis más detallado de posibles eventos y anormalidades.

Finalmente, el sistema final fue presentado a NIC Chile, administradores del ccTLD de Chile (dominios *.cl*), los cuales se mostraron interesados en realizar pruebas con este sistema, para así poseer una nueva alternativa para realizar el monitoreo de sus servidores y asegurar el correcto funcionamiento de sus servicios en todo momento.

5.1. Trabajo Futuro

El trabajo desarrollado en esta memoria solo realiza la captura de información sobre los datos necesarios para la visualización de las métricas definidas, por lo que ampliar la información capturada a todos los campos de una consulta *DNS* ayudara a realizar un análisis más preciso sobre los diferentes eventos que ocurren en estos sistemas.

Sobre la visualización de la información, un dato importante que no fue incluido es la geolocalización y detección de los ASN desde donde se generan las consultas. Esta información es importante dado que permiten a los operadores de los servidores DNS posicionar sus servidores de manera estratégica según el origen de las consultas, además de visualizar las rutas utilizadas para acceder a los servidores, detectando rutas mal configuradas que pueden llevar a una degradación de los servicios.

Esta memoria solo trabajó utilizando una única instancia de la base de datos *ClickHouse*, y no realizó pruebas en un ambiente distribuido de estas. Si poseer un ambiente tolerante a fallos es de carácter crítico, es posible cambiar los tipos de tablas utilizadas a replicadas, para así lograr replicar la información en diferentes servidores, asegurando el correcto funcionamiento del sistema en caso del fallo de uno de estos.

Por último, el sistema de alertas actualmente implementado, no logra reaccionar de manera

activa frente a los diferentes eventos que son detectados, y realizar el desarrollo de una herramienta que permita mitigar algunos de estos podría mejorar la disponibilidad de los servidores DNS en gran manera.

Capítulo 6

Bibliografía

[1] D. Eastlake 3rd. *Domain Name System (DNS) IANA Considerations. RFC 6895.* Internet Engineering Task Force, Abril 2013.

[2] The DNS Operations, Analysis, and Research Center. "dns-oarc/dsc: Dns statistics collector". [En línea]. Disponible en: `https://github.com/DNS-OARC/dsc` [Accedido: 16/10/2017].

[3] The DNS Operations, Analysis, and Research Center. "dns-oarc/dsp: Dns statistics presenter". [En línea]. Disponible en: `https://github.com/DNS-OARC/dsp` [Accedido: 16/10/2017].

[4] The OpenTSDB Authors. "opentsdb - a distributed, scalable monitoring system". [En línea]. Disponible en: `http://opentsdb.net/` [Accedido: 16/10/2017].

[5] Oracle Corporation. "mysql". [En línea]. Disponible en: `https://www.mysql.com/` [Accedido: 16/10/2017].

[6] Elasticsearch. "elasticsearch: Restful, distributed, search & analytics". [En línea]. Disponible en: `https://www.elastic.co/products/elasticsearch` [Accedido: 16/10/2017].

[7] Elasticsearch. "kibana: Explore, visualize, discover data | elastic". [En línea]. Disponible en: `https://www.elastic.co/products/kibana` [Accedido: 16/10/2017].

[8] Elasticsearch. "logstash: Collect, parse, transform logs | elastic". [En línea]. Disponible en: `https://www.elastic.co/products/logstash` [Accedido: 16/10/2017].

[9] Elasticsearch. "packetbeat: Network analytics using elasticsearch | elastic". [En línea]. Disponible en: `https://www.elastic.co/products/beats/packetbeat` [Accedido: 16/10/2017].

[10] Felipe Espinoza. "fdns/tsdb-benchmarks". [En línea]. Disponible en: `https://github.com/fdns/TSDB-benchmarks/` [Accedido: 16/10/2017].

[11] Felipe Espinoza. "niclabs/dnszeppelin-clickhouse: A go binary that send dns statistics to a clickhouse database". [En línea]. Disponible en: https://github.com/niclabs/dnszeppelin-clickhouse/ [Accedido: 14/11/2017].

[12] Felipe Espinoza. "niclabs/dnszeppelin: Go library to capture dns packets". [En línea]. Disponible en: https://github.com/niclabs/dnszeppelin/ [Accedido: 18/01/2018].

[13] Felipe Espinoza. "niclabs/grafana: The tool for beautiful monitoring and metric analytics & dashboards for graphite, influxdb & prometheus & more". [En línea]. Disponible en: https://github.com/niclabs/grafana/tree/clickhouse-datasource/ [Accedido: 18/01/2018].

[14] Google. "google/leveldb: Leveldb is a fast key-value storage library written at google that provides an ordered mapping from string keys to string values". [En línea]. Disponible en: https://github.com/google/leveldb [Accedido: 16/10/2017].

[15] PostgreSQL Global Development Group. "postgresql: The world's most advanced open source database". [En línea]. Disponible en: https://www.postgresql.org/ [Accedido: 16/10/2017].

[16] Docker Inc. "docker compose | docker documentation". [En línea]. Disponible en: https://docs.docker.com/compose/ [Accedido: 14/11/2017].

[17] InfluxData, Inc. "influxdata/influxdb: Scalable datastore for metrics, events, and realtime analytics". [En línea]. Disponible en: https://github.com/influxdata/influxdb [Accedido: 16/10/2017].

[18] Internet Systems Consortium, Inc. "bind". [En línea]. Disponible en: https://www.isc.org/downloads/bind/ [Accedido: 16/10/2017].

[19] Metamarkets Group Inc. "druid | interactive analytics at scale". [En línea]. Disponible en: http://druid.io/ [Accedido: 16/10/2017].

[20] Roman Khavronenko. "hagen1778/grafana at clickhouse-datasource". [En línea]. Disponible en: https://github.com/hagen1778/grafana/tree/clickhouse-datasource/ [Accedido: 14/11/2017].

[21] Grafana Labs. "grafana - the open platform for analytics and monitoring". [En línea]. Disponible en: https://grafana.com/ [Accedido: 16/10/2017].

[22] NIC Chile Research Labs. "niclabs/ratadns-fievel: Rata dns - preliminary reducers framework". [En línea]. Disponible en: https://github.com/niclabs/ratadns-fievel/ [Accedido: 16/10/2017].

[23] NIC Chile Research Labs. "rata-dns development blog". [En línea]. Disponible en: http://niclabs.cl/ratadns/ [Accedido: 16/10/2017].

[24] YANDEX LLC. "clickhouse - open source distributed column-oriented dbms". [En línea]. Disponible en: https://clickhouse.yandex/ [Accedido: 16/10/2017].

[25] P. Mockapetris. *DOMAIN NAMES - CONCEPTS AND FACILITIES. RFC 1034.* Internet Engineering Task Force, Noviembre 1987.

[26] P. Mockapetris. *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. RFC 1035.* Internet Engineering Task Force, Noviembre 1987.

[27] Internet Corporation For Assigned Names and Numbers". "icann. [En línea]. Disponible en: `https://www.icann.org/` [Accedido: 23/10/2017].

[28] Florian octo Forster. "collectd – the system statistics collection daemon". [En línea]. Disponible en: `https://collectd.org/` [Accedido: 16/10/2017].

[29] Philip. "phillipmartin/gopassivedns: Passivedns in go". [En línea]. Disponible en: `https://github.com/Phillipmartin/gopassivedns/` [Accedido: 23/10/2017].

[30] Prometheus. "metric and label naming". [En línea]. Disponible en: `https://prometheus.io/docs/practices/naming/#labels` [Accedido: 16/10/2017].

[31] Prometheus. "prometheus - monitoring system & time series database". [En línea]. Disponible en: `https://prometheus.io/` [Accedido: 16/10/2017].

[32] Salvatore Sanfilippo. "graphite-project/graphite-web: A highly scalable real-time graphing system". [En línea]. Disponible en: `https://github.com/graphite-project/graphite-web` [Accedido: 16/10/2017].

[33] Salvatore Sanfilippo. "redis". [En línea]. Disponible en: `https://redis.io/` [Accedido: 16/10/2017].

[34] Nancy Lynch Seth Gilbert. *Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services".* ACM SIGACT News, 2002.

[35] SoundCloud. "soundcloud - listen to free music and podcastson soundcloud". [En línea]. Disponible en: `https://soundcloud.com/` [Accedido: 16/10/2017].

[36] Tcpdump/Libpcap. "tcpdump/libpcap public repository". [En línea]. Disponible en: `http://www.tcpdump.org/` [Accedido: 23/10/2017].

[37] Nick Winn. "nickwinn/dns-flood: Original dns-flood tool found on code.google.com". [En línea]. Disponible en: `https://github.com/fike/dns-flood` [Accedido: 14/11/2017].

[38] M. Wullink, G. C. M. Moura, M. Müller, and C. Hesselman. Entrada: A high-performance network traffic data streaming warehouse. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 913–918, April 2016.

Capítulo 7

Apéndice

7.1. Apéndice 1: Resultado benchmarks

7.1.1. 5 Dominios más consultados

7.1.1.1. Utilización total de CPU

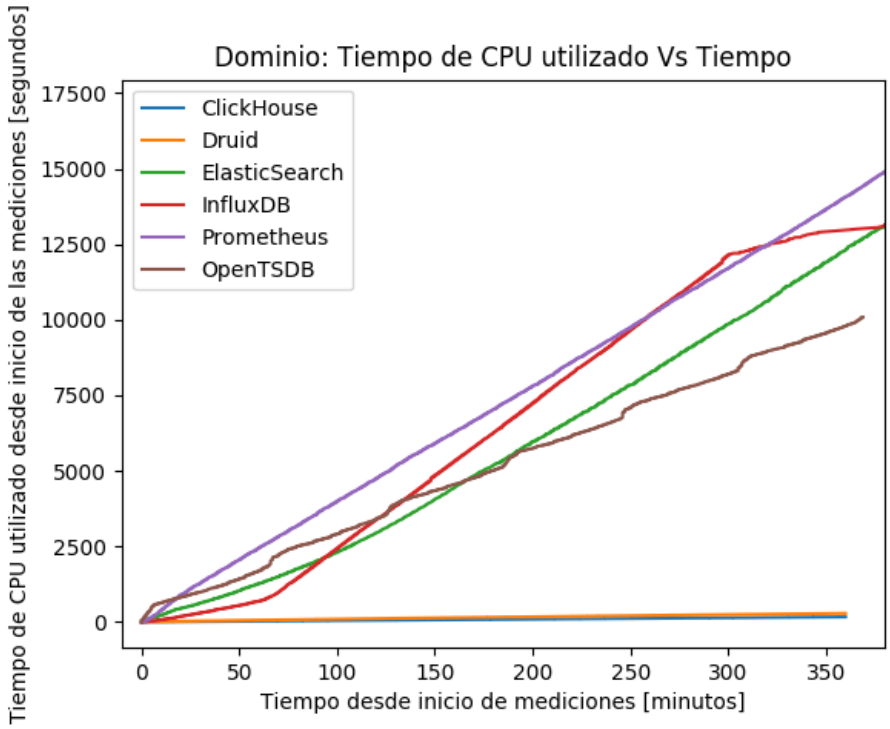

Figura 7.1: Gráfico que muestra la comparación del tiempo de CPU total utilizado por cada base de datos. Los datos fueron obtenidos cada 10 segundos por cada benchmark.

7.1.1.2. Utilización de CPU media

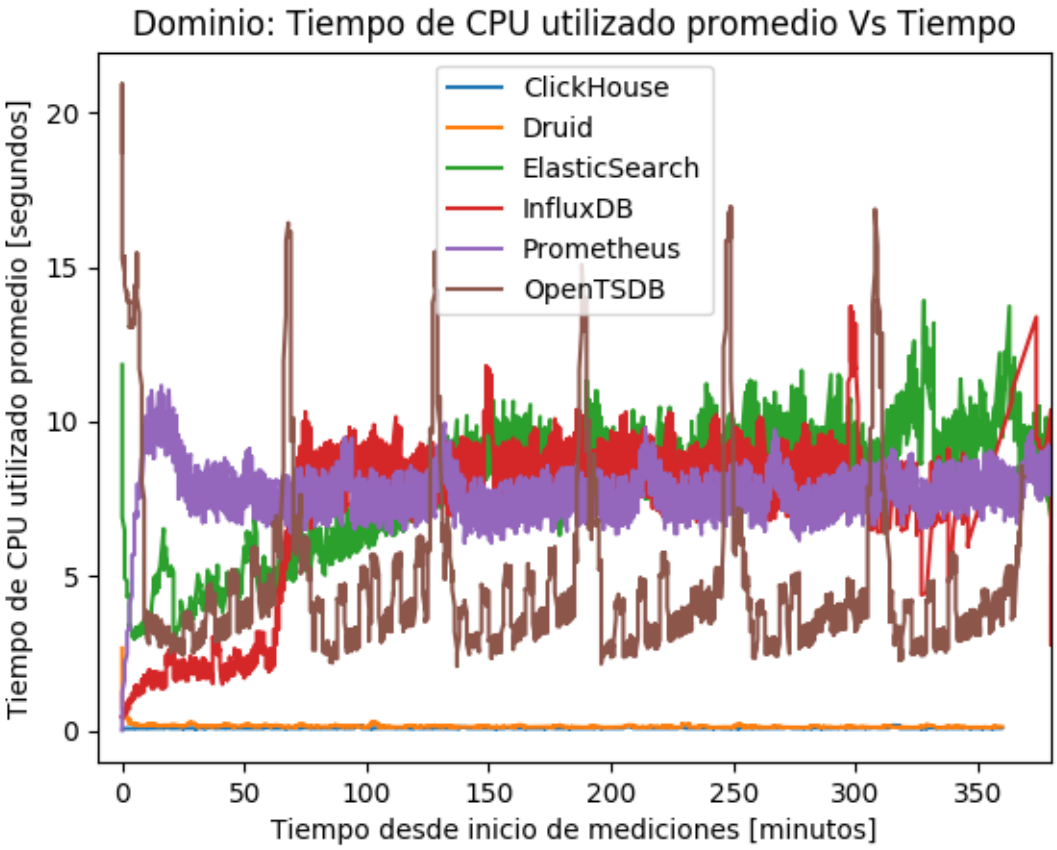

Figura 7.2: Gráfico que muestra la comparación del tiempo de CPU medio utilizado por cada base de datos. Los datos se obtuvieron calculando el promedio de las últimas veinte diferencias medidas.

7.1.1.3. Utilización de memoria principal

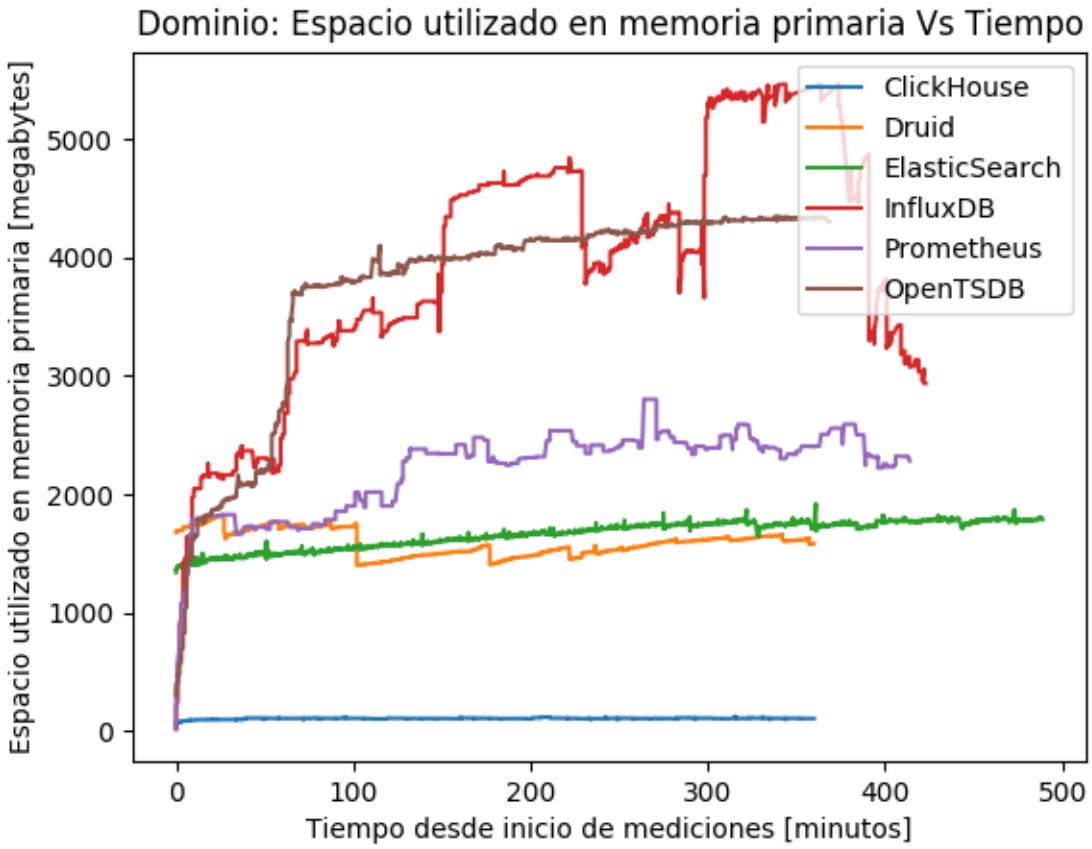

Figura 7.3: Gráfico que muestra la cantidad de memoria primaria (RAM) utilizada por la base de datos. La consulta fue realizada cada 10 segundos.

7.1.1.4. Utilización de memoria secundaria

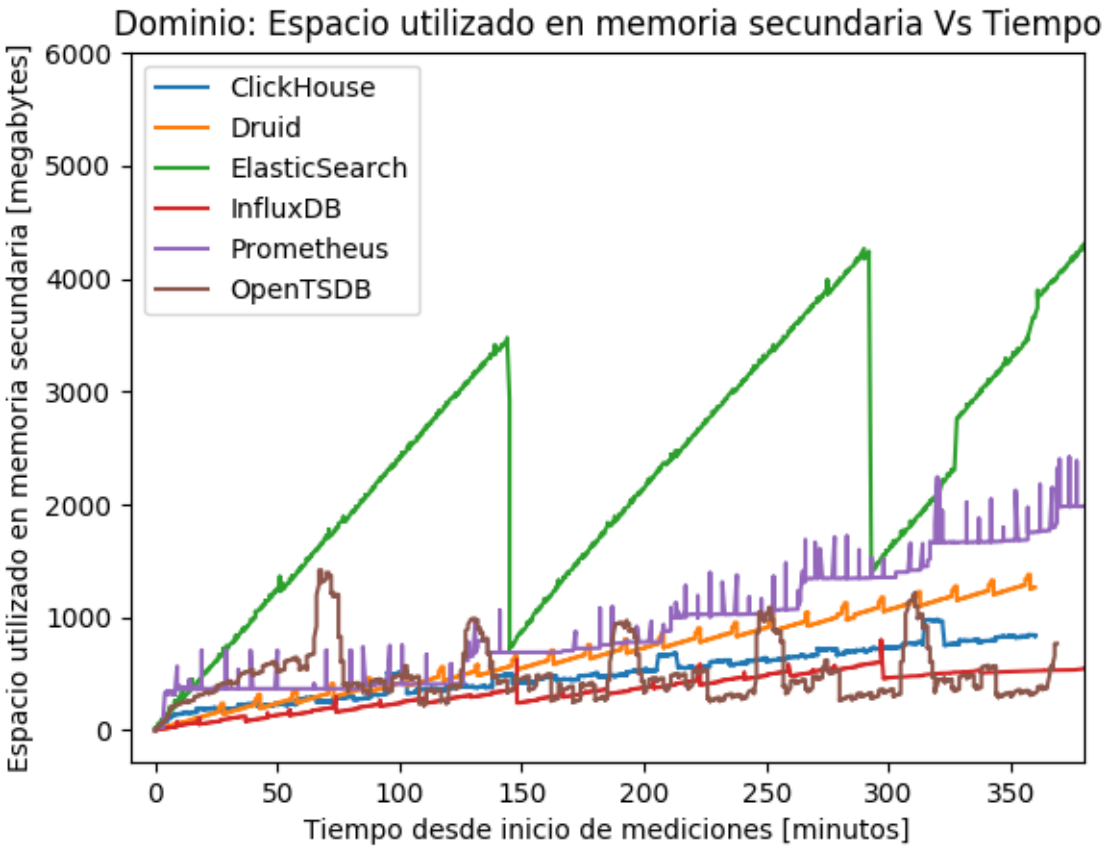

Figura 7.4: Gráfico que muestra la cantidad de memoria secundaria (Disco Duro) utilizada por la base de datos. La consulta fue realizada cada 10 segundos.

7.1.1.5. Tiempo de consulta

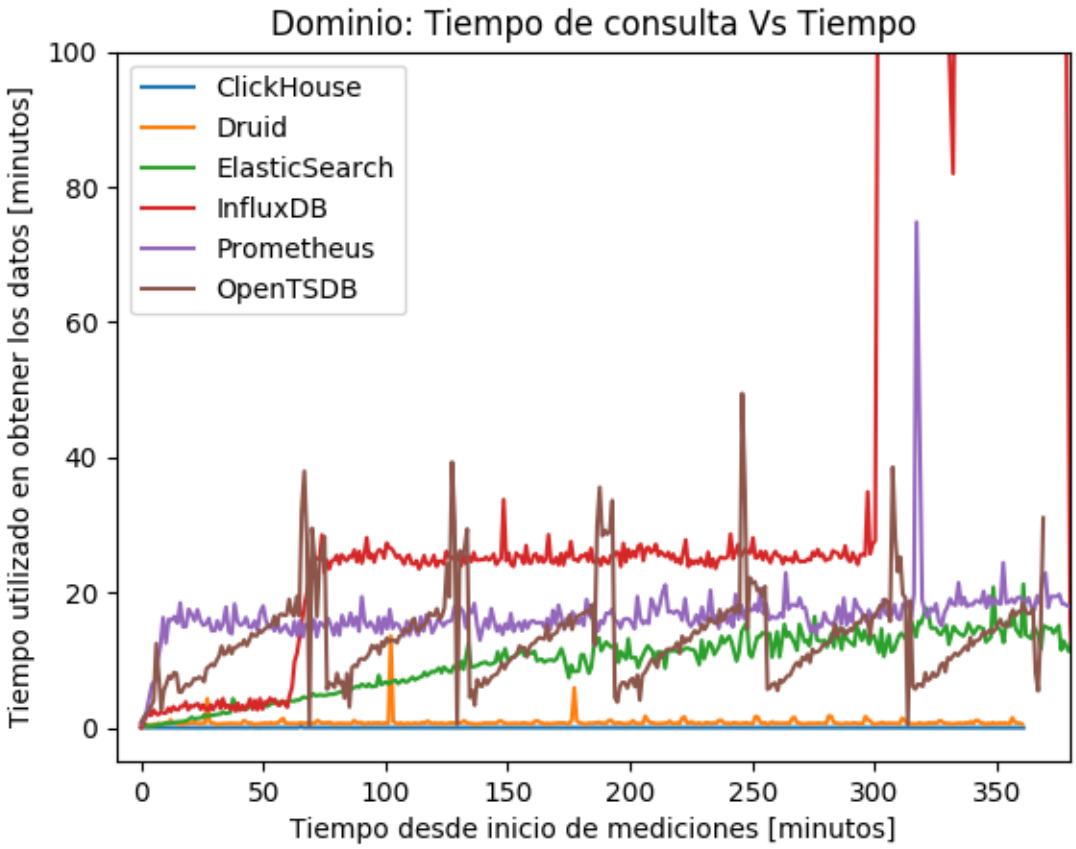

Figura 7.5: Gráfico que muestra el tiempo utilizado para realizar la consulta de los datos. La consulta fue realizada cada 1 minuto.

7.1.2. Máscaras de red IPv4

7.1.2.1. Utilización total de CPU

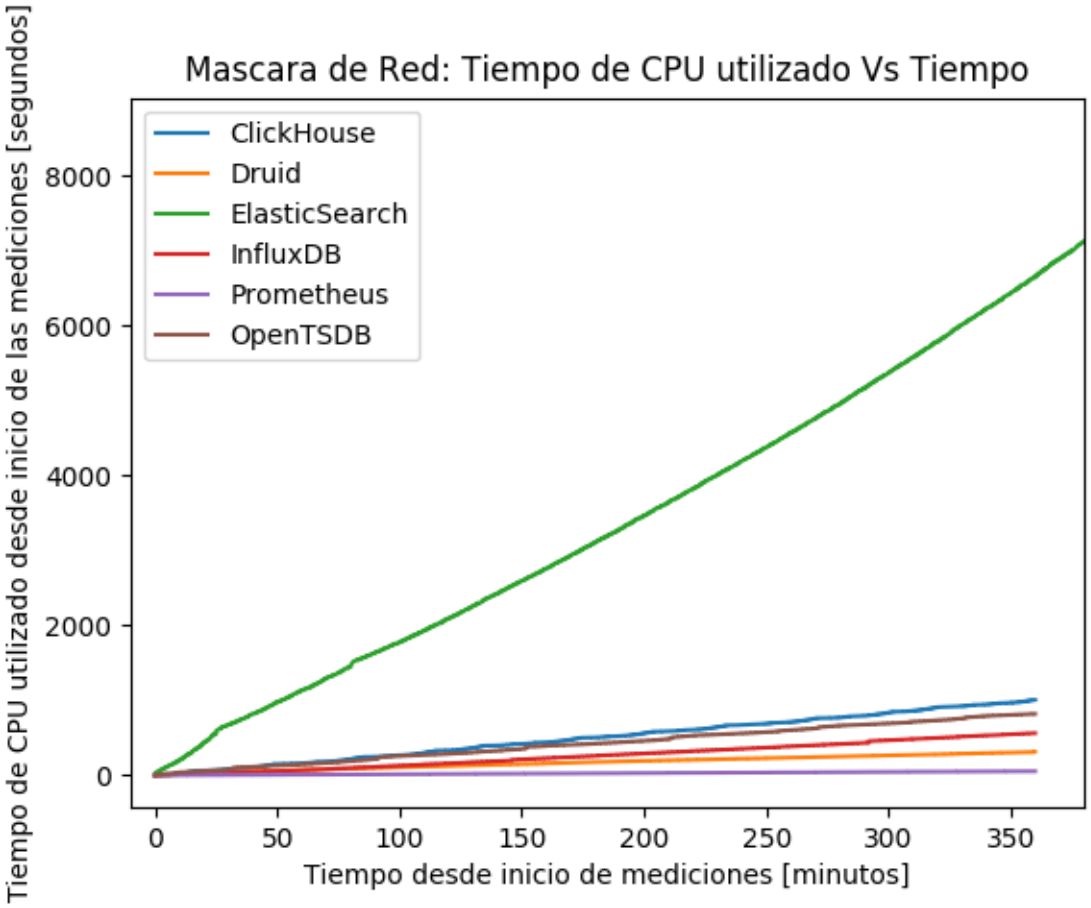

Figura 7.6: Gráfico que muestra la comparación del tiempo de CPU total utilizado por cada base de datos. Los datos fueron obtenidos cada 10 segundos por cada benchmark.

7.1.2.2. Utilización de CPU media

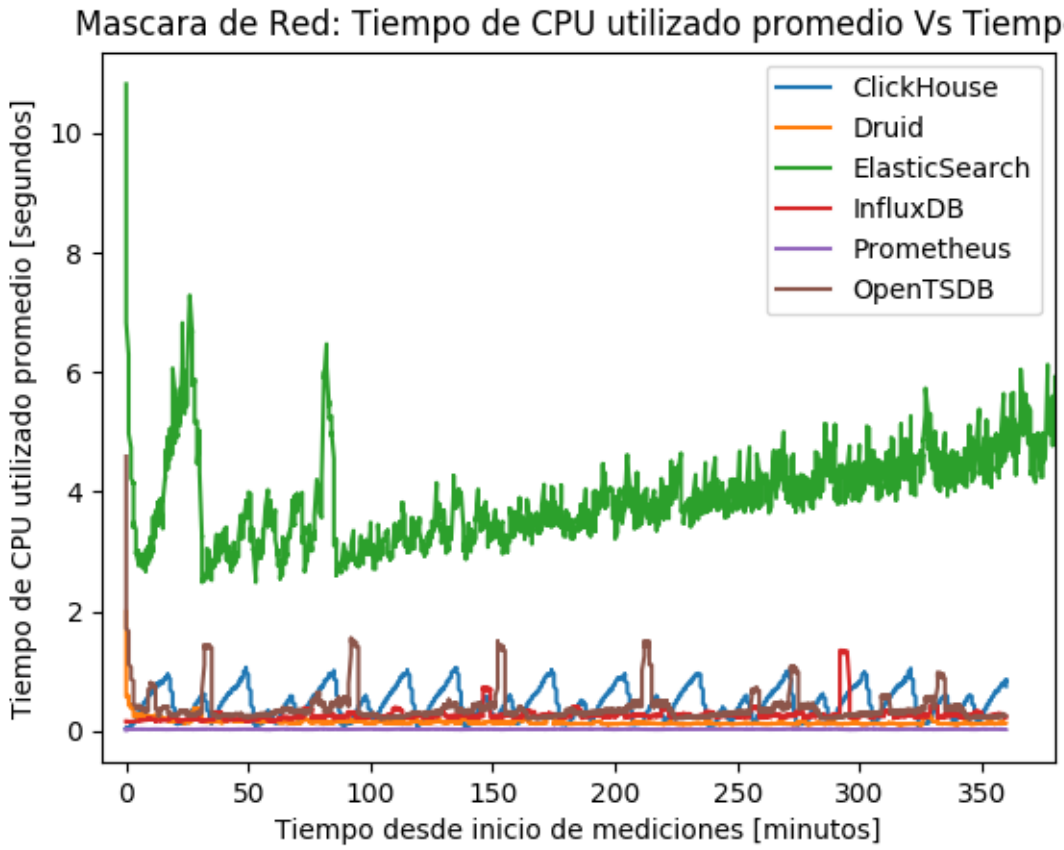

Figura 7.7: Gráfico que muestra la comparación del tiempo de CPU medio utilizado por cada base de datos. Los datos se obtuvieron calculando el promedio de las últimas veinte diferencias medidas.

7.1.2.3. Utilización de memoria principal

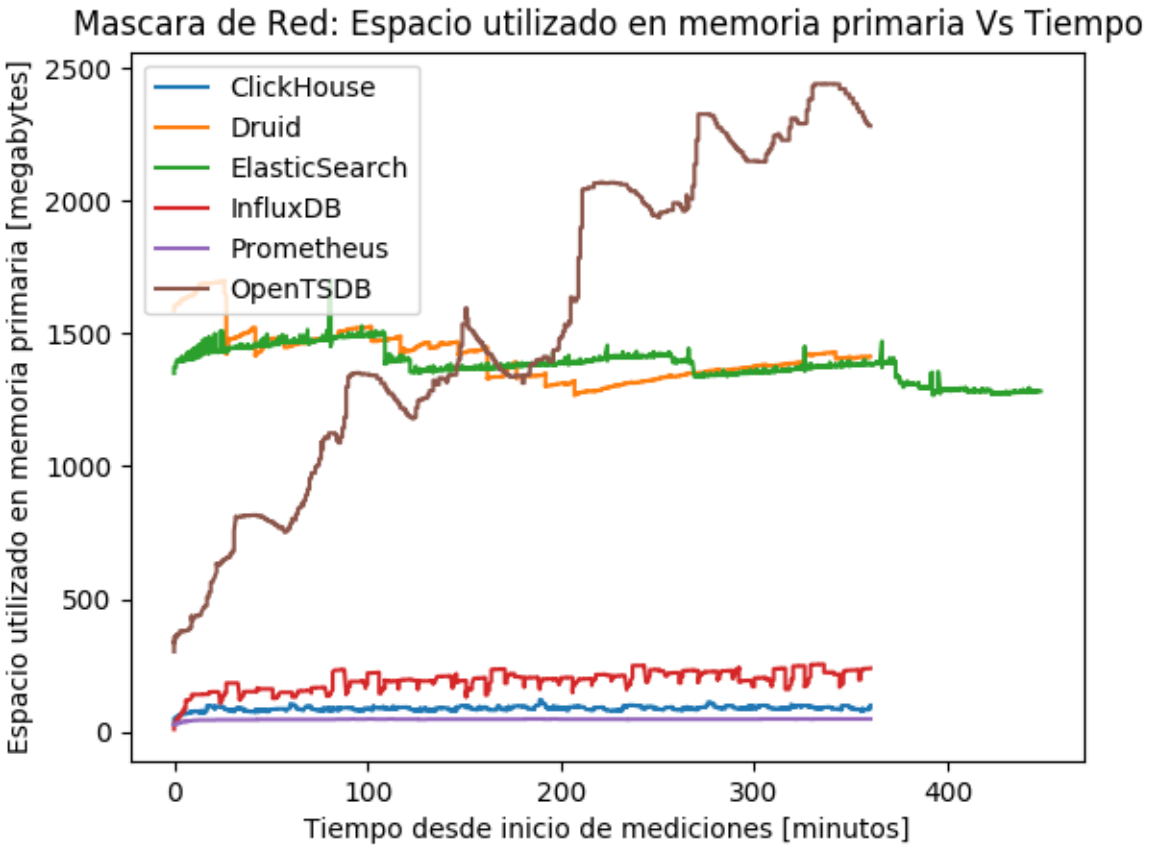

Figura 7.8: Gráfico que muestra la cantidad de memoria primaria (RAM) utilizada por la base de datos. La consulta fue realizada cada 10 segundos.

7.1.2.4. Utilización de memoria secundaria

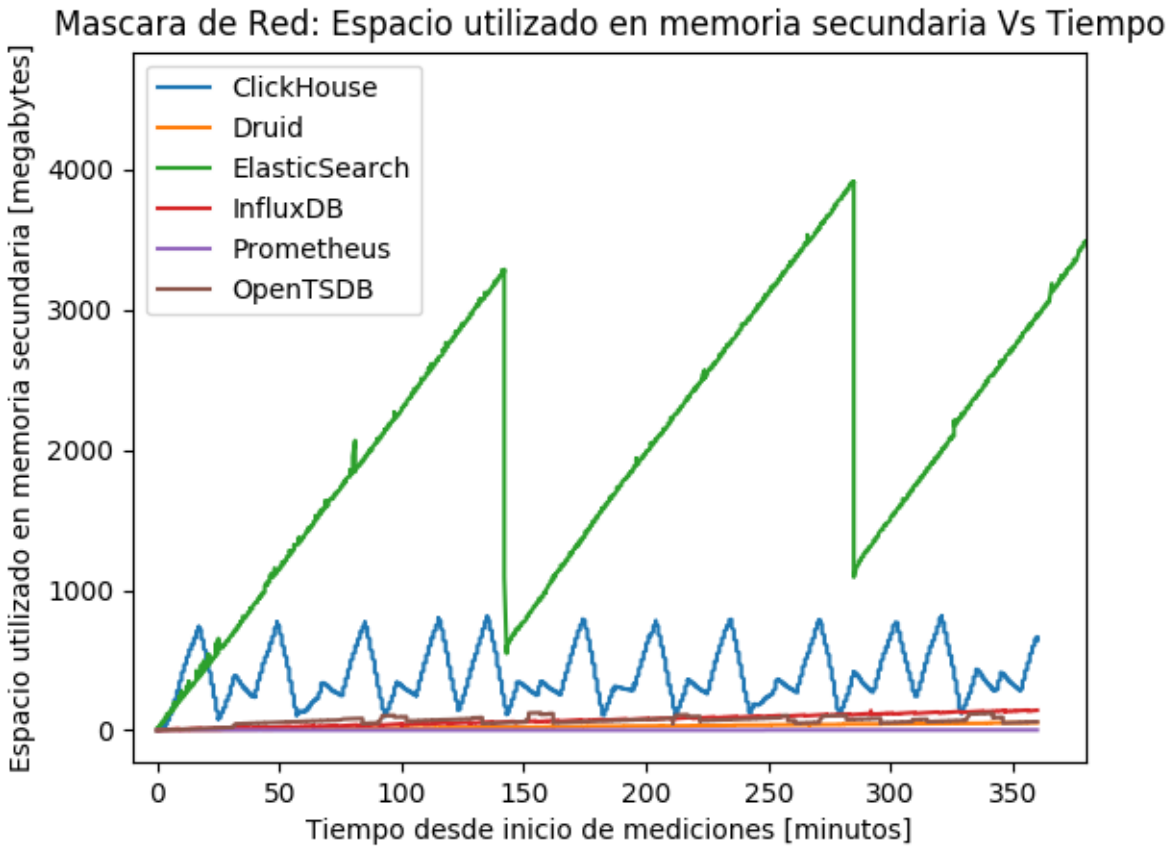

Figura 7.9: Gráfico que muestra la cantidad de memoria secundaria (Disco Duro) utilizada por la base de datos. La consulta fue realizada cada 10 segundos.

7.1.2.5. Tiempo de consulta

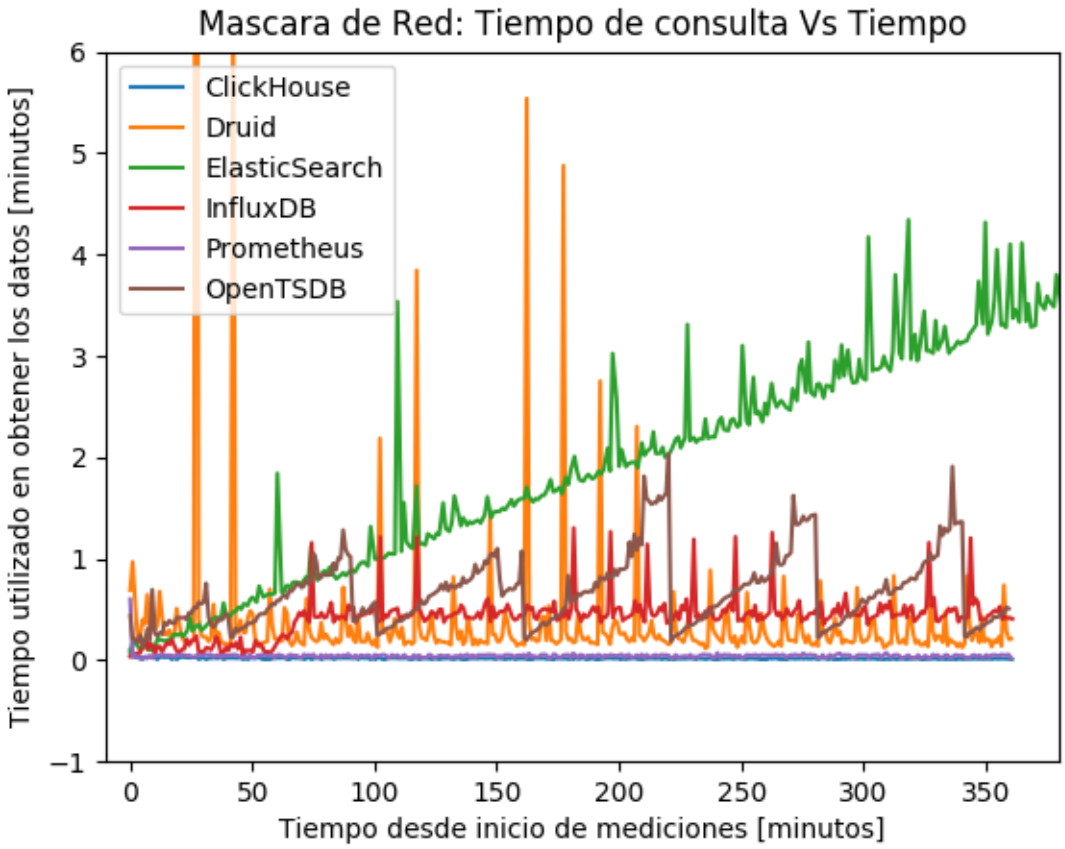

Figura 7.10: Gráfico que muestra el tiempo utilizado para realizar la consulta de los datos. La consulta fue realizada cada 1 minuto.

7.1.3. Largo de paquetes de respuesta

7.1.3.1. Utilización total de CPU

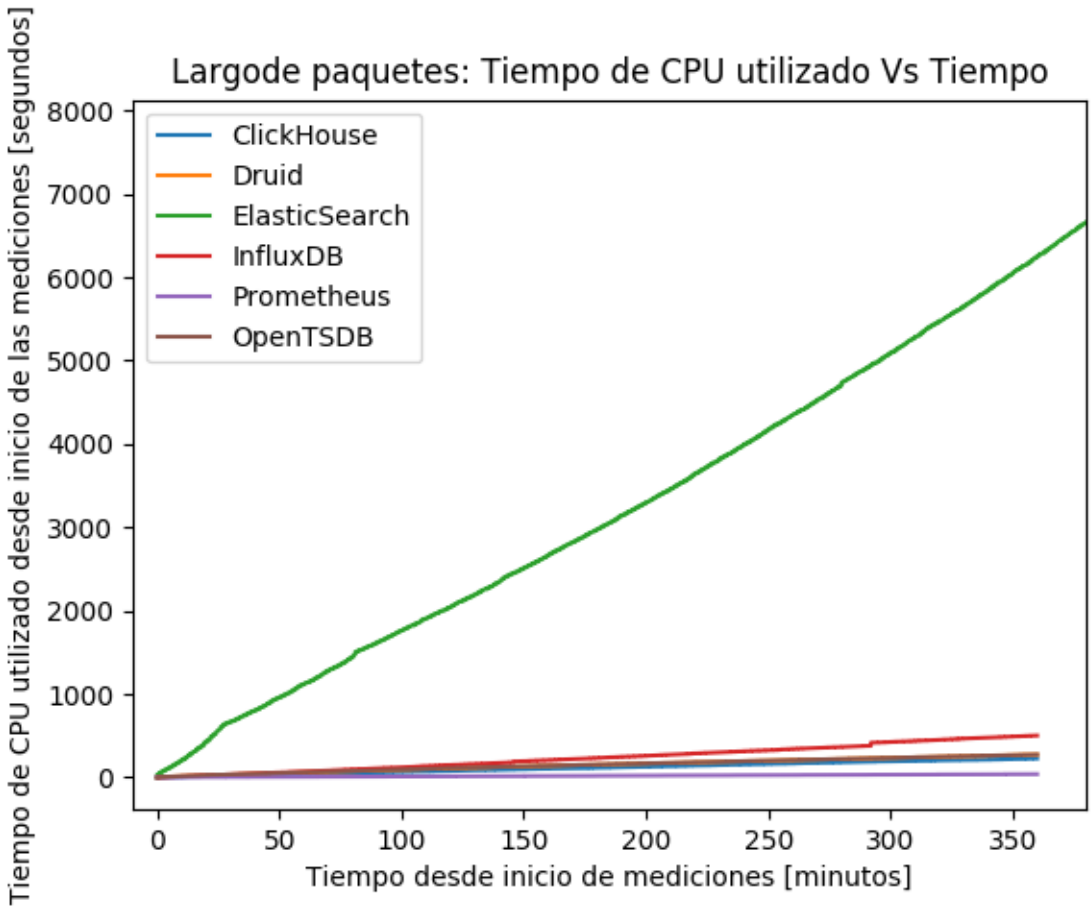

Figura 7.11: Gráfico que muestra la comparación del tiempo de CPU total utilizado por cada base de datos. Los datos fueron obtenidos cada 10 segundos por cada benchmark.

7.1.3.2. Utilización de CPU media

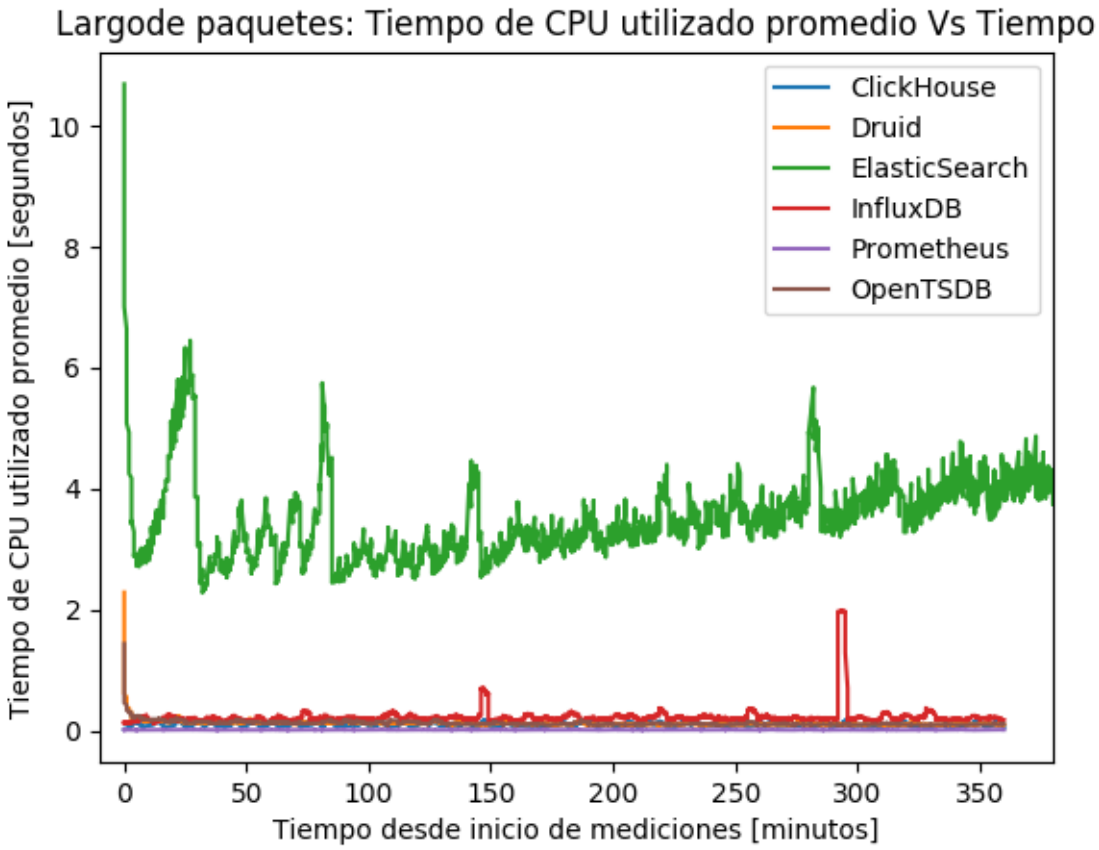

Figura 7.12: Gráfico que muestra la comparación del tiempo de CPU medio utilizado por cada base de datos. Los datos se obtuvieron calculando el promedio de las últimas veinte diferencias medidas.

7.1.3.3. Utilización de memoria principal

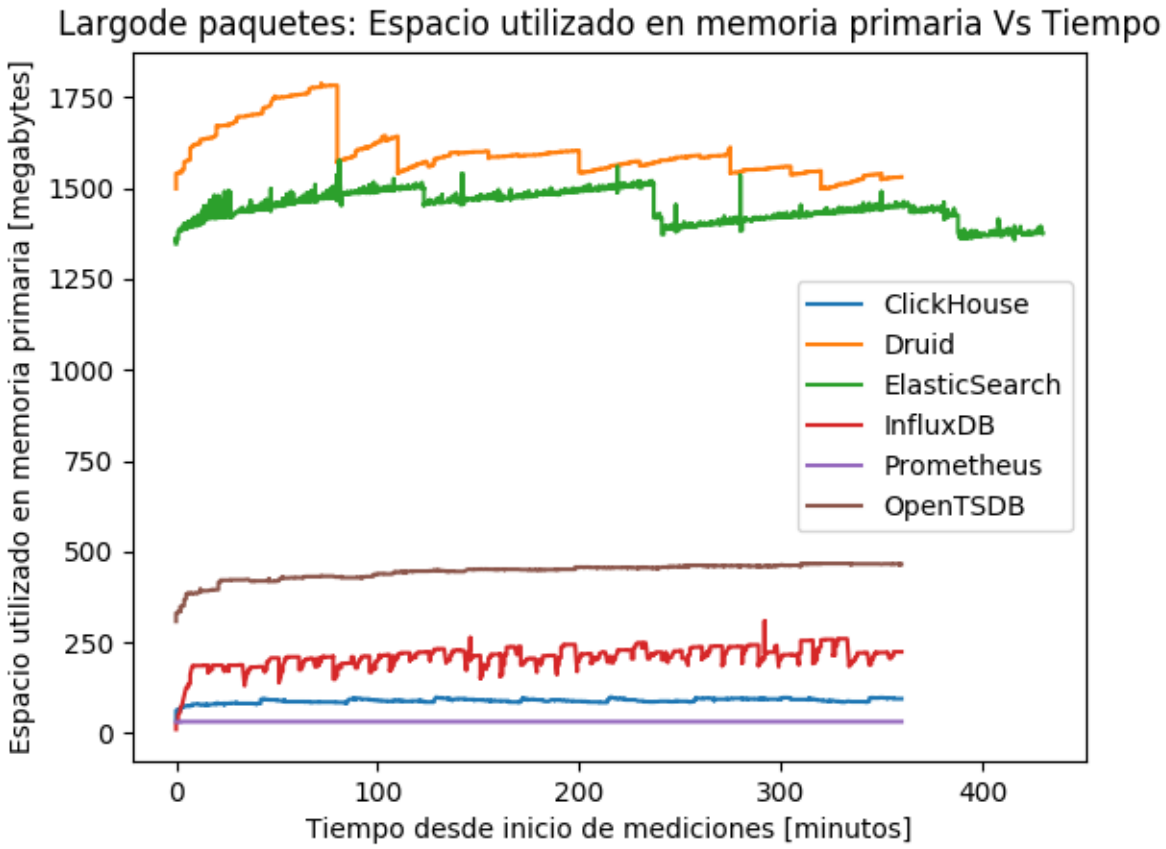

Figura 7.13: Gráfico que muestra la cantidad de memoria primaria (RAM) utilizada por la base de datos. La consulta fue realizada cada 10 segundos.

7.1.3.4. Utilización de memoria secundaria

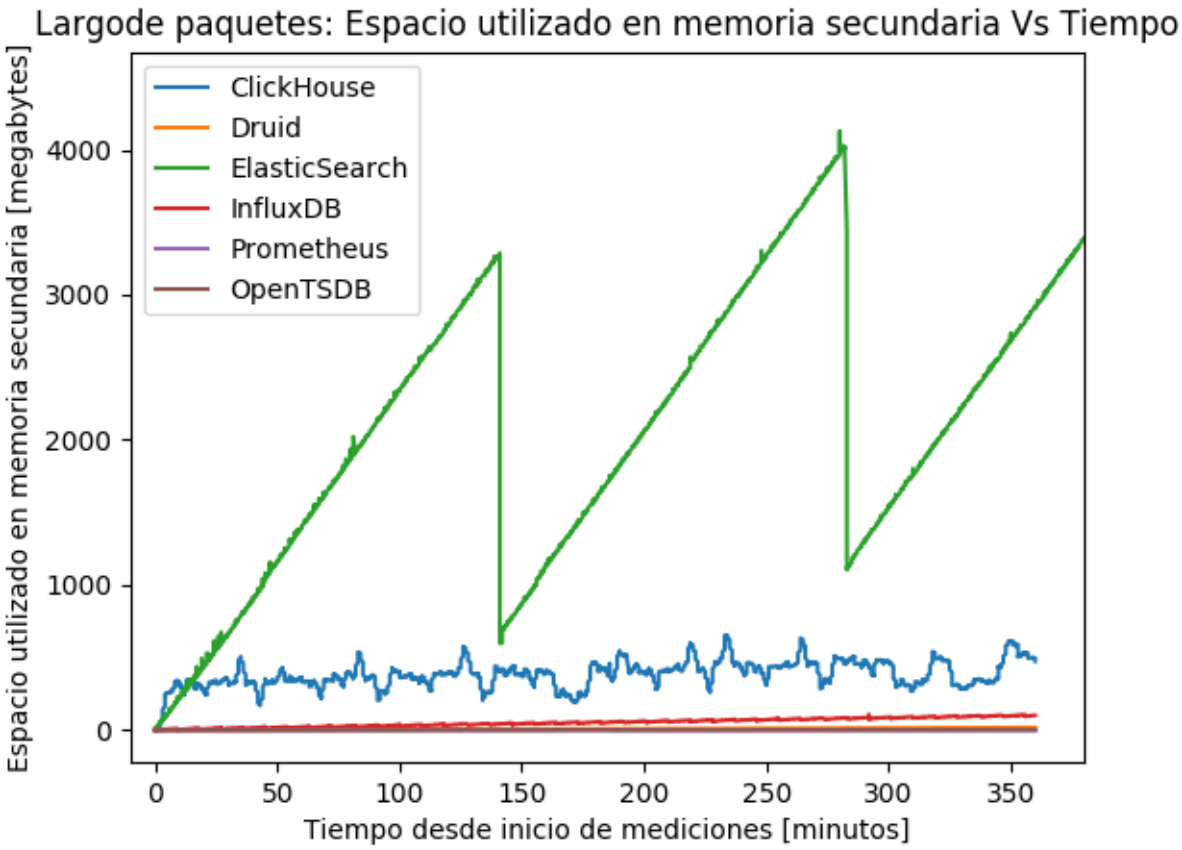

Figura 7.14: Gráfico que muestra la cantidad de memoria secundaria (Disco Duro) utilizada por la base de datos. La consulta fue realizada cada 10 segundos.

7.1.3.5. Tiempo de consulta

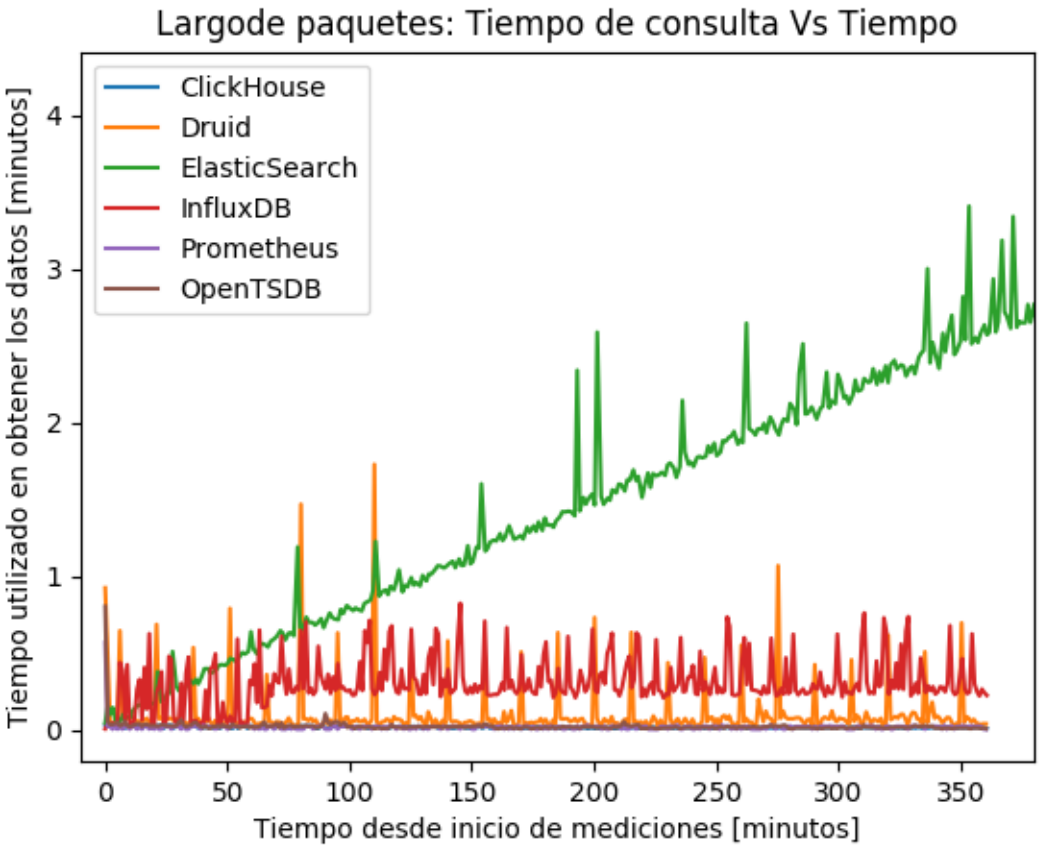

Figura 7.15: Gráfico que muestra el tiempo utilizado para realizar la consulta de los datos. La consulta fue realizada cada 1 minuto.

7.2. Apéndice 2: Consultas para generación de base de datos

```
CREATE TABLE IF NOT EXISTS DNS_LOG (
    DnsDate Date,
    timestamp DateTime,
    Server String,
    IPVersion UInt8,
    IPPrefix UInt32,
    Protocol FixedString(3),
    QR UInt8,
    OpCode UInt8,
    Class UInt16,
    Type UInt16,
    Edns0Present UInt8,
    DoBit UInt8,
    ResponceCode UInt8,
    Question String,
    Size UInt16
) engine=MergeTree(DnsDate, (timestamp, Server), 8192);

-- View for top queried domains
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_DOMAIN_COUNT
ENGINE=SummingMergeTree(DnsDate, (t, Server, Question), 8192, c)
    AS
    SELECT DnsDate, toStartOfMinute(timestamp) as t, Server,
        Question, count(*) as c FROM DNS_LOG WHERE QR=0 GROUP BY
        DnsDate, t, Server, Question;

-- View for unique domain count
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_DOMAIN_UNIQUE
ENGINE=AggregatingMergeTree(DnsDate, (timestamp, Server), 8192) AS
    SELECT DnsDate, timestamp, Server, uniqState(Question) AS
        UniqueDnsCount FROM DNS_LOG WHERE QR=0 GROUP BY Server,
        DnsDate, timestamp;

-- View for count by protocol
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_PROTOCOL
ENGINE=SummingMergeTree(DnsDate, (timestamp, Server, Protocol),
    8192, (c)) AS
    SELECT DnsDate, timestamp, Server, Protocol, count(*) as c FROM
        DNS_LOG GROUP BY Server, DnsDate, timestamp, Protocol;

-- View with packet sizes
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_GENERAL_AGGREGATIONS
ENGINE=AggregatingMergeTree(DnsDate, (timestamp, Server), 8192) AS
```

```
SELECT DnsDate, timestamp, Server, sumState(Size) AS TotalSize,
    avgState(Size) AS AverageSize FROM DNS_LOG GROUP BY Server,
    DnsDate, timestamp;

-- View with edns information
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_EDNS
ENGINE=AggregatingMergeTree(DnsDate, (timestamp, Server), 8192) AS
  SELECT DnsDate, timestamp, Server, sumState(Edns0Present) as
      EdnsCount, sumState(DoBit) as DoBitCount FROM DNS_LOG WHERE QR
      =0 GROUP BY Server, DnsDate, timestamp;

-- View wih query OpCode
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_OPCODE
ENGINE=SummingMergeTree(DnsDate, (timestamp, Server, OpCode),
    8192, c) AS
  SELECT DnsDate, timestamp, Server, OpCode, count(*) as c FROM
      DNS_LOG WHERE QR=0 GROUP BY Server, DnsDate, timestamp, OpCode
      ;

-- View with Query Types
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_TYPE
ENGINE=SummingMergeTree(DnsDate, (timestamp, Server, Type), 8192,
    c) AS
  SELECT DnsDate, timestamp, Server, Type, count(*) as c FROM
      DNS_LOG WHERE QR=0 GROUP BY Server, DnsDate, timestamp, Type;

-- View with Query Class
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_CLASS
ENGINE=SummingMergeTree(DnsDate, (timestamp, Server, Class), 8192,
    c) AS
  SELECT DnsDate, timestamp, Server, Class, count(*) as c FROM
      DNS_LOG WHERE QR=0 GROUP BY Server, DnsDate, timestamp, Class;

-- View with query responses
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_RESPONCECODE
ENGINE=SummingMergeTree(DnsDate, (timestamp, Server, ResponceCode)
    , 8192, c) AS
  SELECT DnsDate, timestamp, Server, ResponceCode, count(*) as c
      FROM DNS_LOG WHERE QR=1 GROUP BY Server, DnsDate, timestamp,
      ResponceCode;

-- View with IP Prefix
CREATE MATERIALIZED VIEW IF NOT EXISTS DNS_IP_MASK
ENGINE=SummingMergeTree(DnsDate, (timestamp, Server, IPVersion,
    IPPrefix), 8192, c) AS
  SELECT DnsDate, timestamp, Server, IPVersion, IPPrefix, count(*)
      as c FROM DNS_LOG GROUP BY Server, DnsDate, timestamp,
```

```
IPVersion, IPPrefix;
```

7.3. Apéndice 3: Consultas para obtención de métricas para Grafana

A continuación se muestras las consultas que se realizan en el panel de grafana, donde se aprecia como *$ServerName* los nombres de los servidores a analizar, *$timeSeries* la división de la serie y *$timeFilter* el rango de tiempo a consultar.

```
-- Top 5 queried domains
SELECT timestamp, groupArray((Question, count)) FROM (SELECT
    $timeSeries as timestamp, Question, sum(c) as count FROM
    DNS_DOMAIN_COUNT WHERE $timeFilter AND Server IN($ServerName)
    group by t, Question ORDER BY count desc limit 5 by timestamp)
    GROUP BY timestamp\n ORDER BY timestamp

-- Unique queried domain names count
SELECT $timeSeries as t, uniqMerge(UniqueDnsCount)/$interval as
    Count FROM DNS_DOMAIN_UNIQUE WHERE $timeFilter AND Server IN(
    $ServerName) GROUP BY t ORDER BY t

-- Average packet size in bytes
SELECT $timeSeries as t, avgMerge(AverageSize) AS Bytes FROM
    DNS_GENERAL_AGGREGATIONS WHERE $timeFilter AND Server IN(
    $ServerName) Group by t order by t

-- Total packet size in bytes
SELECT $timeSeries as t, sumMerge(TotalSize)*8/$interval as \"
    Bytes/s\" FROM DNS_GENERAL_AGGREGATIONS WHERE $timeFilter AND
    Server IN($ServerName) Group by t order by t

-- Query count by network protocol
SELECT t, groupArray((IPVersion, count/$interval)) FROM (SELECT
    $timeSeries as t, IPVersion, sum(c) as count FROM DNS_IP_MASK
    WHERE $timeFilter AND Server IN($ServerName) group by t,
    IPVersion ORDER BY t) group by t\n order by t

-- Query count by transport protocol
SELECT t, groupArray((Protocol, count/$interval)) FROM (SELECT
    $timeSeries as t, Protocol, sum(c) as count FROM DNS_PROTOCOL
    WHERE $timeFilter AND Server IN($ServerName) group by t,
    Protocol ORDER BY t) group by t\n order by t

-- Query count by IP range (IPv4)
SELECT 0, groupArray((IP, total)) FROM (SELECT IPv4NumToString(
    IPPrefix) AS IP,\nsum(c) as total FROM DNS_IP_MASK PREWHERE
```

```
    IPVersion=4 WHERE $timeFilter AND Server IN($ServerName)
    GROUP BY IPPrefix order by IPPrefix)

-- Query count by IP range (IPv6), filtering the top 20 values
SELECT 0, groupArray((IP, total)) FROM (SELECT IPv6NumToString(
    toFixedString(unhex(hex(IPPrefix)), 16)) AS IP,\nsum(c) as total
     FROM DNS_IP_MASK PREWHERE IPVersion=6 WHERE $timeFilter AND
    Server IN($ServerName)    GROUP BY IPPrefix order by IPPrefix
    desc limit 20)

-- Query count by type
SELECT t, groupArray((dictGetString('dns_type', 'Name', toUInt64(
    Type)), count/$interval)) FROM (SELECT $timeSeries as t, Type,
    sum(c) as count FROM DNS_TYPE WHERE $timeFilter AND Server IN(
    $ServerName)    group by t, Type ORDER BY t) group by t\n order
    by t

-- Query count by class
SELECT t, groupArray((dictGetString('dns_class', 'Name', toUInt64(
    Class)), count/$interval)) FROM (SELECT $timeSeries as t, Class,
     sum(c) as count FROM DNS_CLASS WHERE $timeFilter AND Server IN(
    $ServerName)    group by t, Class ORDER BY t) group by t\n order
    by t

-- Query count by responce code
SELECT t, groupArray((dictGetString('dns_responce', 'Name',
    toUInt64(ResponceCode)), count/$interval)) FROM (SELECT
    $timeSeries as t, ResponceCode, sum(c) as count FROM
    DNS_RESPONCECODE WHERE $timeFilter AND Server IN($ServerName)
    group by t, ResponceCode ORDER BY t) group by t\n order by t

-- Query count by operation code
SELECT t, groupArray((dictGetString('dns_opcode', 'Name', toUInt64
    (OpCode)), count/$interval)) FROM (SELECT $timeSeries as t,
    OpCode, sum(c) as count FROM DNS_OPCODE WHERE $timeFilter AND
    Server IN($ServerName)    group by t, OpCode ORDER BY t) group by
     t\n order by t

-- Query count which have EDNS0 bit set
SELECT $timeSeries as t, sumMerge(EdnsCount)/$interval as
    Edns0Present FROM DNS_EDNS WHERE $timeFilter AND Server IN(
    $ServerName)    GROUP BY t ORDER BY t

-- Query count which have Do bit set
SELECT $timeSeries as t, sumMerge(DoBitCount)/$interval as
    DoBitSet FROM DNS_EDNS WHERE $timeFilter AND Server IN(
    $ServerName)  GROUP BY t ORDER BY t
```

7.4. Apéndice 4: Ejecución de los servicios a través de Docker Compose

Para ejecutar los servicios, es necesario instalar *Docker*, *Docker-Compose* y *git* en un sistema operativo *Ubuntu*.

Primero es necesario obtener el código fuente, a través del comando

```
git clone https://github.com/niclabs/dnszeppelin-clickhouse.git
```

Luego para iniciar los servicios, se debe ejecutar el comando

```
(cd dnszeppelin-clickhouse/docker && docker-compose up)
```

Para generar las tablas utilizadas por el proceso de captura, es necesario ejecutar el siguiente comando luego de que la base de datos haya sido iniciada.

```
(cd dnszeppelin-clickhouse && cat tables.sql
 | docker run -i -a stdin --rm --net=host yandex/clickhouse-client
   --multiquery)
```

Finalmente, es necesario agregar el panel ya desarrollado a través de la interfaz web de Grafana, donde se debe agregar la base de datos ClickHouse en utilizando el hostname *clickhouse:8123*, y el archivo de configuración ubicado en *dnszeppelin-clickhouse/docker/grafana/panel.json*.