Proyecto de Grado

Presentado ante la ilustre Universidad de Los Andes como requisito parcial para obtener el Título de Ingeniero de Sistemas

Diseño e implementación de un sistema de monitoreo de redes orientado a la recolección masiva de datos.

Por

Br. Jesús Alberto Gómez Pérez

Tutor: Dr. Andrés Arcia-Moret

Noviembre 2015

Diseño e implementación de un sistema de monitoreo de redes orientado a la recolección masiva de datos.

Br. Jesús Alberto Gómez Pérez

Proyecto de Grado — Sistemas Computacionales, 131 páginas

Resumen: En el presente proyecto se plantea el desarrollo de un sistema de monitoreo distribuido de enlaces críticos de redes a través de un servicio web centralizado. Este servicio además pretende hacer énfasis en la visualización de los datos recolectados a partir de pruebas periódicas. El sistema está planteado como una herramienta para facilitar el entendimiento del funcionamiento de la red y ofrecer una solución centralizada, de muy bajo costo y con componentes de hardware que puedan estar desatendidos.

Palabras clave: Monitoreo de redes, Calidad de servicio, Big Data, Benchmarking, Cloud

Índice

Capítulo 1

Introducción

1.1 Antecedentes

Existen dos estrategias de monitoreo de redes: monitoreo activo o benchmarking que consiste en generar tráfico para realizar medidas y comprobar la respuesta de la red y monitoreo pasivo que consiste en escanear el tráfico de la red en ciertos puntos estratégicos para censar el tráfico en la red. El benchmarking tiene la desventaja de tener que inyectar tráfico lo cual puede entorpecer el funcionamiento normal de la red ejemplos de herramientas de benchmarking son ping, iperf y traceroute.

El monitoreo pasivo tiene la ventaja de que nos puede dar una muy buena idea del uso de la red sin embargo para mayor efectividad debe realizarse en nodos intermedios a los que muchas veces no tenemos acceso, ejemplos de herramientas de monitoreo pasivo son tcpdump y wireshark; en ambos casos hay que resaltar que es difícil tener una imagen completa de la realidad de la red.

Uno de los trabajos más importantes en el área de monitoreo de redes es el Protocolo Simple de Administración de Red o SNMP (del inglés Simple Network Management Protocol) este emergió como una de las primeras soluciones al problema de manejo de redes y se ha convertido en la solución más ampliamente aceptada debido a su diseño modular e independiente de productos o redes específicas. SNMP consiste de (1) un administrador de red, (2) una serie de dispositivos remotos monitoreados (3) bases de información de administración (MIBs) en estos dispositivos, (4) agentes remotos que

reportan la información de las MIBs al administrador de red y toman acciones si les indica y (5) un protocolo de comunicación entre los dispositivos [1].

SNMP no solo ofrece al administrador de red reportes sobre cada uno de los dispositivos administrados sino que también permite tomar acción sobre ellos proactivamente antes de que ocurran problemas o de forma reactiva para solucionar problemas cuando ocurren de forma inesperada.

La red de TVWS (TV White Spaces o Espacios blancos en el espectro radioeléctrico) de Malawi fue implementada en el marco de un proyecto para llevar Internet a áreas rurales en países en vías de desarrollo utilizando soluciones de bajo costo a través de los espacios en blanco en el espectro radioeléctrico específicamente en la banda UHF (siglas del inglés Ultra High Frequency, 'frecuencia ultra alta') [2].

Muchas veces esta red debe dejarse desatendida durante largos periodos de tiempo ya que sus nodos son de difícil acceso o es muy costoso tener a profesionales dedicados que se encarguen de su mantenimiento, este escenario hace evidente la necesidad de una solución de monitoreo de redes a distancia y de mínimo mantenimiento. Además es atractivo para el centro de monitoreo de la red poder añadir, modificar o eliminar nodos de interés de manera sencilla a la interfaz de monitoreo.

Para intentar solucionar este problema y recolectar información sobre la red de Malawi se implementó un sistema en dos partes: un monitor de red instalado en la estación base (BS) en Malawi y un servicio web remoto, el monitor en la estación base se conecta a cada uno de los nodos de la red y determina el tiempo de ida y vuelta (RTT por sus siglas en ingles) de forma automatizada en ciertos intervalos de tiempo, guarda los resultados en archivos y los coloca en una carpeta que se sincroniza a través de un servicio en la nube de tipo PaaS (Plataforma como servicio) con el servidor web, que a su vez escanea la carpeta compartida y actualiza su base de datos que puede usarse para generar gráficas de RTT promedio y determinar tiempos de actividad continuos y porcentaje de disponibilidad de servicio [3].

A pesar de que este sistema recolecta información útil y presenta gráficas muy sencillas de entender, su programación no permite agregar nuevos nodos, esto trae como consecuencia que debe ser modificado manualmente cuando la red se expande, dando lugar a la necesidad de que el servicio web se pueda expandir para dar servicio

a múltiples monitores remotos simultáneamente.

Por estos motivos proponemos la construcción de un sistema de monitoreo a gran escala que llamaremos Octopus Monitor, para hacerlo totalmente configurable y robusto además de agregar una interfaz de configuración vía web que permita manejar usuarios, agregar monitores remotos, agregar nodos y modificar los parámetros de las pruebas, todo esto apoyándonos en un sistema de archivos compartidos a través de la nube.

Mientras que el mayor valor de Malawinet Monitor es la visualización de grandes volúmenes de datos que se pueden obtener a partir de pruebas de bajo impacto de tráfico (ping, traceroute), se han realizado otros trabajos en el área de monitoreo de redes como Bowlmap; este es un sistema de monitoreo de redes a través de la visualización de mediciones para el Laboratorio Abierto Inalámbrico de Berlín (BOWL, por sus siglas en ingles). Este sistema tiene una alta flexibilidad ya que permite realizar cambios en sus pruebas existentes así como agregar pruebas totalmente nuevas y a su vez generar las visualizaciones necesarias para el análisis de dicha información, además tiene la ventaja de solo transmitir la información necesaria para cada actualización lo que acelera las peticiones y permite la visualización de data en tiempo real [4].

1.2 Planteamiento del Problema

Las redes de computadoras se componen de un conjunto de nodos interconectados sujetos a numerosos factores que escapan de nuestro control y sobre los que muchas veces no tenemos conocimiento, en otras palabras la red puede llegar a ser impredecible y no ofrece garantías sobre el servicio que ofrece; algunas aplicaciones dependen de una alta disponibilidad y estabilidad por lo que es esencial para un administrador de red tener información del estado de la red, para diagnosticar, solucionar problemas y asegurar la calidad de servicio.

Existen muchos otros ejemplos en los que es importante tener datos del estado de la red como en redes de bajo costo en las que pueden ocurrir largas interrupciones de servicio o para un cliente de un servicio de alojamiento web que desea saber si su sitio web está disponible y que tan rápido responde; plataformas como Pingdom [5] o

UptimeRobot [6] permiten monitorear distintos servicios en Internet y generan alertas cuando encuentran problemas, sin embargo no son gratuitas y no permiten la inclusión de nuevos tipos de pruebas o la visualización masiva de datos históricos.

Mantener estas mediciones con las herramientas existentes se vuelve una tarea compleja mientras crece el número de nodos a monitorear (es decir, las fuentes de información) y la cantidad de datos aumenta a través del tiempo, sumado a esto solo podemos capturar información a partir de los nodos externos de la red, por lo que en la mayoría de los casos no es posible tener una imagen completa de los enlaces a monitorear.

A pesar de que Malawinet Network Monitor podría ofrecer estadísticas de tiempo de ida y vuelta (RTT) y disponibilidad de un enlace solo a partir de las trazas capturadas con Ping, se desea además implementar un marco de trabajo que permita agregar nuevas pruebas automatizadas que ayuden a obtener una imagen más completa de la red.

1.3 Justificacion

Ante la aparición de nuevos paradigmas en la distribución del acceso a Internet, tales como redes comunitarias caracterizadas por despliegues caóticos y descoordinados o redes inalámbricas con enlaces de larga distancia que pueden ser poco confiables, se evidencia la necesidad de tener herramientas de bajo costo, sencillas de desplegar y mantener que puedan ayudar a los interesados a tener conocimiento del estado de la red.

A pesar de que existe una gran variedad de herramientas para el monitoreo de despliegues de redes arbitrarias, se desea construir una plataforma que facilite el monitoreo a través de una interfaz clara y metáforas intuitivas que abstraigan las entidades monitoreadas y que permita extender rápidamente el sistema cuando se deseen monitorear características de la red que no fueron consideradas de antemano.

1.4 Objetivos

1.4.1 Objetivos Generales

Construir un servicio web de monitoreo de redes de bajo costo para países en vías de desarrollo con almacenamiento de datos en la nube de tipo PaaS que sea de fácil instalación y permita configurar múltiples monitores remotos para ajustarse a cambios en las características de las redes a monitorear y la carencia de personal in sitio.

1.4.2 Objetivos Específicos

- Desarrollar un servicio de monitoreo de bajo costo para países en vías de desarrollo que de servicio a múltiples monitores de red remotos, presente visualizaciones gráficas a partir de los datos recogidos y ofrezca un marco de trabajo para agregar nuestros tipos de pruebas a los monitores de red existentes.
- Desarrollar un cliente monitor para desplegar en nodos desatendidos con dispositivos recolectores de muestra de bajo costo (ej. Raspberry PI, Alix boards, APU) para observar el comportamiento de los enlaces a través de aplicaciones de monitoreo sencillas y de consola.
- Utilizar de sistemas de bajo costo y alta disponibilidad en la nube para almacenamiento y transferencia de datos.
- Integrar los distintos subsistemas que conforman el servicio de monitoreo.
- Desarrollar un modulo de calculo asíncrono de gráficas que permita mejorar los tiempos de interacción del usuario final con el sistema utilizando técnicas para agilizar cómputo como caching, prefetching, threads, etc.

1.5 Metodología

En este trabajo se seguirá una metodología en espiral; el modelo en espiral es un modelo del ciclo de vida del software donde el esfuerzo del desarrollo es iterativo. Cada ciclo de

la espiral representa una fase del desarrollo de software, cada uno de los ciclos consiste de los siguientes pasos:

1. Determinar o fijar los objetivos. En este paso se definen los objetivos específicos para posteriormente identifica las limitaciones del proceso y del sistema de software, además se diseña una planificación detallada de gestión y se identifican los riesgos.

2. Análisis del riesgo. En este paso se efectúa un análisis detallado para cada uno de los riesgos identificados del proyecto, se definen los pasos a seguir para reducir los riesgos y luego del análisis de estos riesgos se planean estrategias alternativas.

3. Desarrollar, verificar y validar. En este tercer paso, después del análisis de riesgo, se eligen un paradigma para el desarrollo del sistema de software y se lo desarrolla.

4. Planificar. En este último paso es donde el proyecto se revisa y se toma la decisión si se debe continuar con un ciclo posterior al de la espiral. Si se decide continuar, se desarrollan los planes para la siguiente fase del proyecto.

Se realizarán cuatro ciclos, el primero corresponde a la realización de un monitor remoto básico que realice mediciones de RTT de la red con almacenamiento en la nube y visualizaciones de los datos obtenidos.

El segundo ciclo consiste en permitir el monitoreo de una cantidad arbitraria de monitores remotos permitiendo a múltiples usuarios manejar sus monitores remotos desde el servicio web y observar las visualizaciones.

El tercer ciclo corresponde en diseñar e integrar una prueba con otras herramientas (traceroute, iperf, etc) para conseguir puntos comunes y generar un enfoque de integración sencillo de los wrappers futuros a las aplicaciones. (ej. Lidiar con aplicaciones que requieren enfoque cliente solo [ping] o cliente-servidor [iperf]).

El cuarto ciclo consiste en hacer análisis del rendimiento del sistema y hacer las optimizaciones necesarias para ofrecer una calidad de servicio apropiada, determinar costos, limitaciones y requisitos mínimos para implementar en países en vías de desarrollo.

1.6 Alcance

El alcance de este trabajo remite al diseño y detalles de implementación tanto de una aplicacion web que sirva como plataforma central de coordinación del monitoreo, y un agente monitor de redes ligero y robusto que pueda ser controlado remotamente. Se pretende diseñar un sistema flexible y un esquema de coordinación entre sus distintos entes a través de la nube.

1.7 Estructura del Documento

El presente trabajo se estructura de la siguiente manera:

Capítulo 1. Introducción, este capítulo consiste de los antecedentes relevantes al tema a tratar, planteamiento del problema, justificación, objetivos, metodología a emplear y el alcance de este trabajo.

Capítulo 2. Marco Teórico, define brevemente los conceptos y herramientas usados durante el desarrollo de este trabajo y que son esenciales para su comprensión; tales como conceptos de alto nivel de monitoreo de redes, principio fin-a-fin, métricas de la calidad de un enlace, visualización de datos y los y sistemas de software usados para su implementación.

Capítulo 3. Aplicacion Web, contiene el diseño general del sistema de monitoreo, así como la arquitectura de la aplicacion web, sus sub-sistemas, componentes relevantes, casos de uso y finalmente análisis de rendimiento.

Capitulo 4. Monitor de Red, explica el diseño del monitor de red remoto, su arquitectura, componentes y los módulos de prueba implementados.

Capitulo 5. Framework de Integracion de Pruebas, describe los requisitos del marco de trabajo para la implementación de nuevas pruebas, así como todos sus componentes genéricos, finalmente explica como usar dichos componentes para integrar nuevas características a un sistema en producción.

Capitulo 6. Conclusiones y Recomendaciones, contiene las conclusiones obtenidas a partir del diseño e implementación del sistema, así como recomendaciones y posibles mejoras a incluir en trabajos futuros.

Finalmente se presentan las referencias bibliogr aficas.

Capítulo 2

Marco Teórico

En este capitulo se presentan los conceptos necesarios para la comprensión de este documento y se describen las herramientas de software, métodos y formatos que se emplearán para el desarrollo del sistema propuesto.

2.1 Monitoreo de Redes

El monitoreo de redes se refiere al uso de sistemas computacionales para determinar el estado de redes de computadoras, el estado de la red puede ser visto como el conjunto de factores o métricas que describen la red en un momento dado, se hablará a detalle de algunas de estas métricas en la sección 2.3.

Mientras una red tiene ciertos elementos relativamente invariables como la posición de sus nodos, la longitud de los enlaces, el tipo de enlace (fibra óptica, cable, satelital, etc), el ancho de banda de los enlaces, la velocidad de procesamiento de los enrutadores, entre otros, el estado de la red viene determinado por elementos generalmente impredecibles, como condiciones climatológicas, problemas de configuración, equipos en mal estado y congestión, determinar todo este tipo de problemas para generar alertas, aplicar acciones correctivas o al menos tener conocimiento de ellas es el objetivo de un sistema de monitoreo de redes.

Existen dos paradigmas fundamentales en el monitoreo de redes, el monitoreo activo que consiste en generar tráfico y observar la respuesta de la red y el monitoreo pasivo,

que consiste en observar el tráfico que atraviesa un cierto enlace o nodo. Para tener una imagen completa de la red es conveniente usar ambos paradigmas ya que ambos pueden ofrecer distintas perspectivas.

2.2 Principio Fin-a-Fin

El principio de fin-a-fin (también llamado argumento de fin-a-fin) sugiere que las funciones en los niveles bajos de un sistema pueden ser redundantes o de poco valor comparadas con el costo de proveerlas a bajo nivel[7], en otras palabras este argumento recomienda que la complejidad de un sistema de computación distribuido debe estar en los nodos mas externos o de "mayor nivel".

Este principio es valioso para el diseño de cualquier tipo de sistema distribuido sin embargo ha sido fundamental en la expansión y desarrollo de redes de computadoras de propósito general y se sustenta en que los nodos intermedios de una red deben proveer solo las funciones necesarias para permitir la comunicación entre los nodos finales, de esta manera es posible implementar nuevas funcionalides en los extremos de la red sin necesidad de hacer cambios a los nodos intermedios.

En el contexto de monitoreo de redes, se denomina tomografía de redes al proceso de inferir aspectos internos de una red a partir de mediciones realizadas en sus nodos externos.

2.3 Métricas de calidad de un enlace

A continuación se explican las métricas mas comúnmente utilizadas para determinar la calidad de un enlace.

2.3.1 Latencia

La latencia es el tiempo que le toma a un paquete o mensaje viajar desde su origen hasta el punto destino [8]. Teóricamente la latencia está relacionada directamente con la distancia entre los puntos finales de una comunicación, en la practica los paquetes viajan a través de una red de enrutadores que retransmiten el mensaje hasta su destino,

la latencia total será la suma de cada uno de los siguientes factores para cada uno de los enrutadores que atraviese el paquete[8]:

- **Retraso de Propagación:** es el tiempo que tarda un mensaje en viajar del emisor al receptor y es función de la distancia por la velocidad a la que la señal se propaga.
- **Retraso de Transmisión:** es el tiempo que tarda el emisor en poner todos los bits de un mensaje en el medio de transmisión y es función de la longitud del paquete y el ancho de banda del enlace.
- **Retraso de Procesamiento:** tiempo requerido para revisar la cabecera del paquete, buscar errores y determinar el destino del paquete.
- **Retraso de Colas:** Cantidad de tiempo que un paquete pasa en la cola de una interfaz esperando su turno por ser procesado.

2.3.2 Tiempo de ida y vuelta

El tiempo de ida y vuelta (RTT por sus siglas es ingles) es el tiempo que le toma a un paquete viajar desde el origen a su destino y de vuelta, podría pensarse que el RTT es aproximadamente el resultado de multiplicar la latencia por dos, sin embargo existen factores como el retraso por procesamiento en el equipo destino o diferencias en el enrutamiento del paquete en su camino de vuelta que hace que sea arriesgado establecer esa proposición.

Uno de los métodos mas populares para medir el RTT es enviar un paquete ICMP Echo Request y esperar el correspondiente ICMP Echo Reply sin embargo también es posible medir el RTT pasivamente durante la transmisión de un flujo TCP usando marcas de tiempo en la cabecera de los mensajes TCP[9].

Es importante notar que no es lo mismo medir el RTT desde la capa de red con el protocolo ICMP que hacerlo en la capa de transporte con TCP o en la capa de aplicación con un protocolo como HTTP, evidentemente el RTT será mayor en las capas superiores, sin embargo estas métricas también son útiles ya que se acercan mas fielmente a la experiencia del usuario.

2.3.3 Perdida de paquetes

La pérdida de paquetes ocurre cuando los paquetes en una transmisión fallan en llegar a su destino, ya sea por interferencias en el medio de transmisión (por ejemplo obstáculos físicos en un enlace WiFi), o cuando son desechados por un nodo de la red, los paquetes pueden ser desechados si se determina que están corruptos o mas comúnmente debido a escenarios de congestión en los que un enrutador está recibiendo paquetes a una tasa mayor de la que puede retransmitirlos y no tiene mas opción que desechar los paquetes que desborden la cola.

La perdida de paquetes afecta dramáticamente la latencia percibida en la capa de aplicación, según [9] una perdida de paquetes del 5% puede introducir un retraso de medio segundo en la aplicación.

Mientras puede parecer que la perdida de paquetes es siempre producto de un problema, esta juega un papel vital en el algoritmo de prevención de congestión (congestion avoidance) del protocolo TCP, ayudándolo a detectar congestión en la red el cual responderá limitando su tasa de envío de datos, esto ha sido esencial para evitar el colapso de las redes ip NOTA: Nosotros alguna vez hablamos de esto, valdría la pena nombrar el RFC que lo explica.

2.3.4 Jitter

Se llama jitter a la fluctuación de la latencia durante la transmisión de un conjunto de paquetes, estas fluctuaciones vienen dadas principalmente por la variación del retraso que los paquetes experimentan en las colas en los enrutadores[1], sin embargo todos los factores mencionados en la sección 2.3.1 pueden contribuir en menor medida.

El término jitter puede tener distintas connotaciones sin embargo es usado frecuentemente por científicos en el área de la computación como la variación de una métrica (comúnmente latencia) con respecto a otra métrica de referencia (como latencia mínima o promedio), a esto también se le llama variación en el retraso de los paquetes (PDV por sus siglas en ingles), este termino es a veces preferido por ser mas preciso [10].

El jitter puede afectar considerablemente transmisiones en tiempo real como VoIP

o streaming sin embargo la correcta implementación de políticas de buffering del lado del receptor puede minimizar e incluso mitigar este efecto[1].

2.3.5 Throughput

En términos de redes de computadoras, el throughput es la tasa en bits por segundo a la que fluyen los datos a través de un enlace [1], el throughput puede compararse al caudal de un rio, donde mientras mas ancho sea el rio mas agua puede fluir a través de el.

El máximo throughput teórico entre un par de nodos esta determinado principalmente por el segmento de la red con menor ancho de banda, este comportamiento se ilustra en la figura 2.1, donde el cable entre el punto de acceso wifi y el ISP resulta ser el cuello de botella en la comunicación. En la practica el throughput también viene determinado por varios factores como otros flujos de datos con los que se comparte la red o la velocidad a la que el receptor es capaz de procesar el flujo de datos entrante.

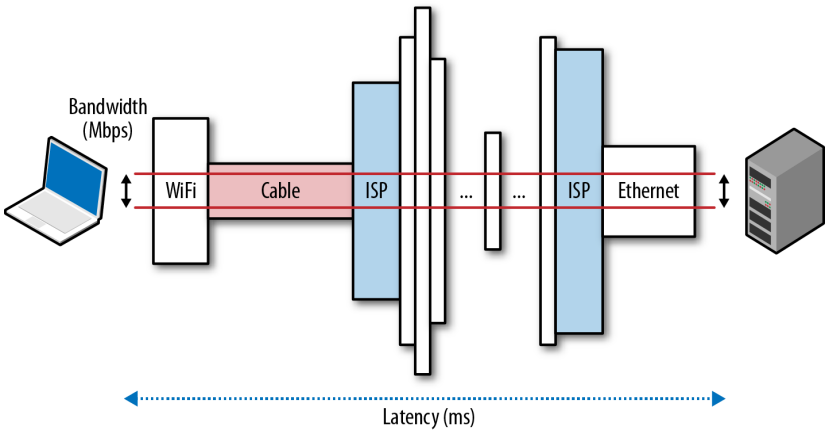

Figura 2.1: Latencia y ancho de banda – Imagen tomada del sitio: http://chimera.labs.oreilly.com/books/1230000000545/ch01.html

Para aplicaciones en tiempo real es esencial tener un throughput mínimo mayor a un cierto umbral que asegure que los datos se estén recibiendo a una tasa mayor o igual a la que se están consumiendo[1], es decir que a partir de esa tasa mínima un mayor throughput no implica una mejoría en la calidad de la comunicación, otras aplicaciones

como transferencias de archivos se benefician del mayor throughput posible ya que el tiempo de transferencia es una función del tamaño del archivo entre el throughput durante la transferencia.

No es posible medir el throughput de un enlace de forma pasiva, por lo que las pruebas que existen consisten en inyectar tráfico al enlace hasta saturarlo y determinar la velocidad a la que se reciben los datos del lado del receptor. Existen múltiples maneras de saturar el enlace, por ejemplo speedtest.net [11] utiliza hasta cuatro hilos paralelos transmitiendo mensajes aleatorios a través de HTTP, utilizar múltiples flujos es una forma efectiva de mitigar el efecto de la latencia sobre el throughput.

A diferencia de las métricas anteriores medir el throughput solo es posible con la colaboración de los nodos extremos del enlace, por lo que es necesario tener algún tipo de software preparado para aceptar la prueba e informar del resultado obtenido.

2.3.6 Uso del ancho de banda

Se le llama uso del ancho de banda a la medida de los datos que fluyen a través de un enlace, generalmente se busca optimizar el uso del ancho de banda para aprovechar al máximo los recursos de red.

Mas allá de solo determinar la cantidad de datos o la velocidad del flujo a través de un enlace, también es útil hacer análisis detallados del uso del ancho de banda para descubrir patrones de uso de la red, por ejemplo una universidad podría desear saber que porcentaje del ancho de banda esta siendo utilizado por streams de audio o video, a una empresa le gustaría saber cuanto tiempo pasan sus empleados en redes sociales, un analista desea saber cuales son los sitios web mas visitados en una red, etc.

NOTA PARA EL PROFESOR: ayuda, no encontré ninguna cita que me ayudara a definir estos conceptos, sin embargo me parece que es importante incluir esto en el marco teórico.

2.3.7 Disponibilidad

Un servicio esta disponible para un cliente cuando dicho cliente puede comunicarse con el, análogamente un servicio no esta disponible para un cliente cuando no puede

comunicarse con el, ya sea por un problema en el nodo final o en la red [12].

Generalmente la disponibilidad se mide a través de la disponibilidad promedio, que es la fracción del tiempo durante el cual un servicio esta disponible para un cliente promedio [12], sin embargo también se puede usar la disponibilidad continua, en este trabajo llamaremos actividad continua a un evento durante el cual un servicio está continuamente disponible para un cliente, y un inactividad continua a un evento durante el cual un servicio esta continuamente no disponible para un cliente.

Hemos definido la disponibilidad como un concepto meramente binario, en el que si un servicio es alcanzable entonces está disponible [12] sin embargo hay ciertos escenarios en los que se puede considerar que un nodo alcanzable está fallando en ofrecer un servicio adecuadamente, por ejemplo cuando un servidor web está respondiendo a las peticiones con un código de estado 500, o cuando la latencia es tal que hace que una comunicación en tiempo real no sea posible.

2.3.8 Bufferbloat

Bufferbloat es la existencia de buffers excesivamente grandes y generalmente llenos presentes en Internet [13]; puede parecer contra-intuitivo ya que buffers mas grandes implican que menos paquetes serán desechados al llegar a un enrutador congestionado, pero mientras la cola en la interfaz del enrutador crece también lo hace el tiempo de espera del paquete y a la vez interfiere (o invalida) los algoritmos de control de congestión de los protocolos mas comunes en la capa de transporte [13].

El bufferbloat puede ser mitigado configurando apropiadamente el hardware disponible, sin embargo es difícil de diagnosticar y es confundido frecuentemente con congestión en la red.

Recientemente se ha comenzado a medir el blufferbloat como el tiempo adicional que toma enviar paquetes a través de un enlace congestionado, algunas pruebas disponibles en linea [14][15] intentan determinar este retraso haciendo mediciones constantes de latencia al mismo tiempo que inundan el enlace para determinar la forma en que esta varía durante la prueba.

2.4 Big Data

Debido a la rápida evolución en la capacidad de almacenamiento y procesamiento de los sistemas computaciones y la adopción de los mismos por parte de miles de millones de usuarios en Internet, así como la creciente de popularidad de objetos inteligentes (Internet de las Cosas), sensores, cámaras, micrófonos, lectores biométricos, lectores de radiofrecuencia, entre muchos otros, día a día se están produciendo datos de forma rápida y masiva, esta tendencia creciente en la generación de datos ha hecho evidente la necesidad de tener sistemas capaces de manipular estos datos para obtener información y hacer descubrimientos que habrían sido imposibles anteriormente.

Big Data se refiere la tendencia reciente de recolectar, almacenar y hacer análisis sobre cantidades masivas de datos, Big Data es un termino relativamente impreciso ya que no existe un convenio sobre la cantidad de datos a la que se refiere, pero usualmente el termino se relaciona con datos en el orden de los petabytes (10^{15} bytes) y exabytes (10^{18} bytes) [16].

El aprovechamiento de este flujo de datos generalmente inmanejable por los sistemas previamente concebidos ha sido adoptado por muchos tipos de organizaciones, por ejemplo las redes sociales rutinariamente analizan a sus usuarios para descubrir sus gustos y preferencias y ajustar cuidadosamente la publicidad que estos ven, juegos en linea analizan a sus jugadores para entender que factores determinan su comportamiento y de esta manera optimizar finamente las experiencias que les ofrecen, un ejemplo conocido de uso de big data en el ámbito médico ha sido la exitosa predicción de transmisión de enfermedades como el dengue a partir de patrones de búsqueda en google[17], también existe preocupación sobre el uso del big data para violar la privacidad de usuarios de Internet por parte de organizaciones gubernamentales de seguridad y vigilancia que son capaces de conocer todo tipo de detalles personales como transacciones y compras, sitios web visitados, búsquedas realizadas, publicaciones en redes sociales, posición geográfica con el uso de GPS, etc.

2.5 Visualización de datos

La visualización de datos se refiere al aprovechamiento de elementos gráficos para representar información cuantitativa; cualquier conjunto de datos no tienen significado sin alguna manera de organizar y presentar los descubrimientos relevantes que se encuentran potencialmente ocultos dentro de estos.

Los humanos podemos comprender los datos de mejor manera cuando son presentados a través de imágenes y elementos gráficos que leyendo números en tablas y columnas[18], una visualización apropiada de los datos permite de forma efectiva preguntar y responder las preguntas relevantes a una organización, por ejemplo en el marco de un sistema de monitoreo de redes preguntas como "¿donde están apareciendo los cuellos de botella?" "¿que factores afectan el rendimiento de un enlace?" o "¿cuales son los patrones de uso de los usuarios de la red?"

Hay una variedad de métodos apropiados para visualizar distintos conjuntos de datos, por ejemplo, los datos discretos se pueden observar a través de gráficas de barras, los gráficos de redes pueden comunicar la relación entre distintos entes, los mapas son efectivos para desplegar información geográfica, los mapas de calor permiten comparar el rendimiento de una variable a través del tiempo, etc, cada una de estas visualizaciones puede ser enriquecida a través del uso creativo de colores y formas para agregar nuevas dimensiones a los datos representados. Es posible combinar distintos conceptos para lograr visualizaciones aun mas poderosas como mapas de calor superpuestos en mapas que pueden expresar datos de la densidad de una variable al mismo tiempo que se da una idea de su posición geográfica.

En la figura 2.2 se puede observar como se usan distintos métodos para representar datos sobre crímenes como robo de vehículos, homicidios y asaltos con armas letales en un área del distrito de Columbia, un mapa es usado para mostrar la localización en que se ocurrieron los crímenes, en esta representación se puede observar rápidamente que zonas son mas propensas a cada tipo de crimen y obtener una idea de su frecuencia, a la izquierda se puede ver una gráfica de barras mostrando la cantidad de ocurrencias coloreadas por tipo de crimen y separados por día de la semana, de esta manera es posible no solo observar que crímenes son mas frecuentes sino que días y semanas son las mas propensas. También se ofrecen herramientas para filtrar los crímenes por

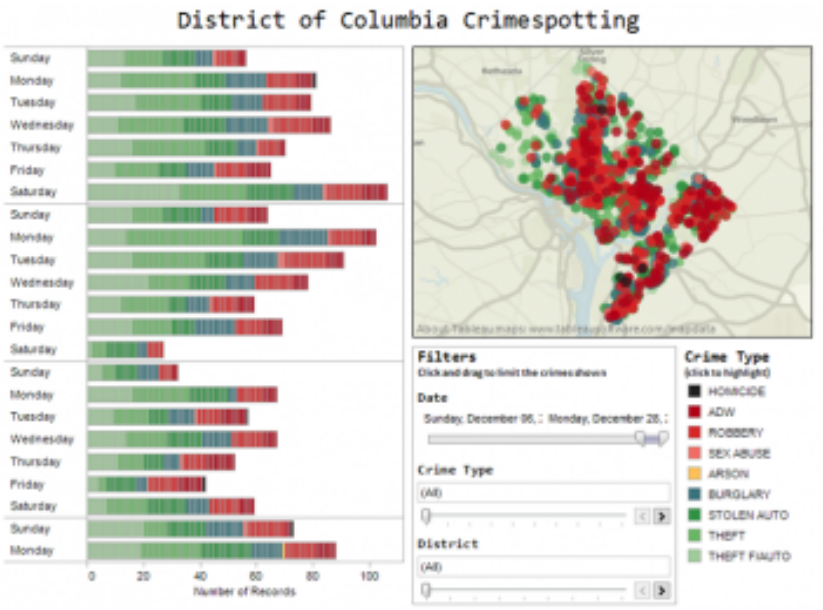

Figura 2.2: Visualizacion de datos sobre delincuencia en el distrito de Columbia – Imagen tomada del sitio:: http://www.pervasif.com/index.php/news-a-event/capabilities/data-visualization

fecha, tipo de crimen y distrito, este tipo de herramientas de visualización de datos interactivas son cada vez mas populares en todo tipo de organizaciones ya que facilitan el análisis efectivo de sus datos.

2.6 Computación en la nube

La computación en la nube se refiere tanto a las aplicaciones desplegadas como servicios en Internet y el hardware y los sistemas computaciones en los centros de datos que proveen dichos servicios [19], a los servicios en si mismos se les ha dado el nombre de Software como Servicio (SaaS) y al hardware y software en los centros de datos es a lo que llamamos una nube. Cuando una nube se hace disponible para el publico en general a través de alguna forma de pago, se le llama una nube publica y el servicio que se esta vendiendo es Computación como Utilidad (Utility Computing), se le llama nube privada a los centros de datos internos de negocios u otras organizaciones pero que no pueden ser usados por el publico general, por lo tanto llamamos computación en

la nube a la suma de SaaS y Computación como Utilidad sin incluir nubes privadas[19]

La computación en la nube se caracteriza por ofrecer métodos de pago flexibles que permiten pagar solo por los recursos que se están utilizando (pay-as-you-go) y con una fina granularidad de modo que es lo mismo pagar mil procesadores una hora que un procesador por mil horas, esto permite a aplicaciones manejar cargas y escalas fluctuantes o patrones de uso específicos sin necesidad de tener hardware que esté ocioso durante largos periodos de tiempo. La nube ofrece a desarrolladores la ilusión de recursos computaciones ilimitados y disponibles a petición, eliminando la necesidad de planificar y aprovisionar equipos de hardware, y minimizar los riesgos relacionados con subestimar una aplicación que explota en popularidad o sobrestimar una aplicación que no llena las expectativas, en otras palabras no es necesario tener un gran capital de inversión inicial independientemente de la escala que resulte necesario manejar a corto o mediano plazo.

2.6.1 Modelos de servicio en la nube

Existen tres modelos de servicios en la nube que forman una "arquitectura orientada a servicios", estos son:

2.6.1.1 Infraestructura como Servicio (IaaS)

Infraestructura como Servicio a veces llamado Hardware como Servicio (HaaS) ofrece funciones básicas de almacenamiento y capacidad de computo como servicio ya sea a través de equipos de hardware físicos o de forma mas común como maquinas virtuales y una larga gama de imágenes de software disponible.

Esta capa abstrae al usuario de los detalles de infraestructura de los recursos de computo físico, localización, configuración, escala, seguridad, respaldo, mantenimiento, etc.

2.6.1.2 Plataforma como Servicio (PaaS)

Plataforma como servicio ofrece como servicio una plataforma para el desarrollo, ejecución y manejo de aplicaciones web, los recursos computacionales demandados por

la aplicación son manejados automáticamente, de modo que la complejidad de manejar la infraestructura subyacente queda eliminada, esto permite reducir enormemente la complejidad necesaria para desplegar aplicaciones, que pueden pasar de la etapa de desarrollo y pruebas rápidamente a un entorno de producción con un esfuerzo mínimo.

2.6.1.3 Software como Servicio (SaaS)

En el modelo de Software como servicio, los usuarios ganan acceso y hacen uso de aplicaciones de software, generalmente este tipo de software se paga bajo una subscripcion o por uso, este tipo de aplicaciones en la nube es conveniente y atractiva para los usuarios ya que estos no necesitan instalar software adicional en sus dispositivos sino que puede acceder a la aplicación desde navegadores web, esto permite que los usuarios puedan tener la misma experiencia en cualquier plataforma y reduce los costos operacionales.

2.7 Herramientas usadas para el desarrollo del sistema

A continuación se describen las herramientas usadas para el desarrollo del sistema de monitoreo de redes orientado a la recolección masiva de datos.

2.7.1 Modelo Vista Controlador (MVC)

El Modelo Vista Controlador es un patrón de diseño que divide la lógica de los datos y la presentación de forma claramente identificable y bien definida [20].

Este patrón de diseño es especialmente popular en el marco de aplicaciones web ya que su abstracción permite escribir software altamente desacoplado y fácil de mantener y escalar.

2.7.1.1 Modelo

Según [20] *"El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar, así por ejemplo un sistema de administración*

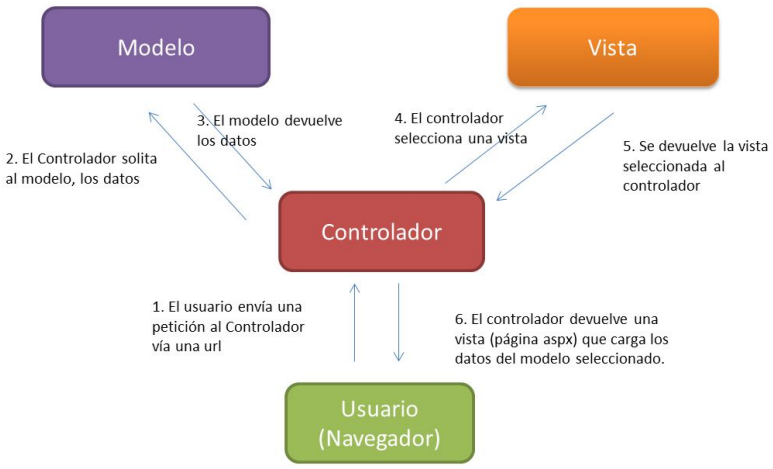

Figura 2.3: Ejemplo de la relación entre los componentes de una aplicación MVC – Imagen tomada del sitio: http://mind42.com

de datos climatologías tendrá un modelo que representará la temperatura, humedad ambiental, estado del tiempo esperado etc"

Según la implementación de MVC el modelo puede dividirse en el modelo del dominio que es el modelo propiamente dicho, es decir una colección de clases que modelan la realidad relevante a la aplicación, y opcionalmente el modelo de la aplicación; este modelo tiene conocimiento de las vistas y es capaz de enviar notificaciones cuando ocurren cambios en el modelo. El modelo de la aplicación también es llamado coordinador de la aplicación [20] .

2.7.1.2 Vista

La vista es la encargada de determinar que información contenida en el modelo mostrar al usuario y la presentación, por ejemplo si se está modelando una caldera es posible tener una vista que dibuje gráficamente el nivel de la caldera y un termómetro con su temperatura y otra vista que sencillamente muestre estas propiedades en una tabla.

La vista pudiera cambiar cuando se actualiza el modelo del dominio a partir de notificaciones emitidas por el modelo de la aplicación, siguiendo con el ejemplo anterior de esta forma sería posible monitorear en tiempo real el estado de la caldera a partir

de sensores que mantengan actualizado el modelo.

2.7.1.3 Controlador

El controlador es el encargado de dirigir el flujo de control de la aplicación a partir de mensajes externos, como datos introducidos por el usuario en el caso de una aplicación de escritorio o peticiones HTTP en el caso de aplicaciones web. A partir de estos estímulos, el controlador se encargará de invocar las vistas apropiadas, actualizar el modelo y hacer todas las acciones necesarias.

Distintas implementaciones del patrón MVC se toman la libertad de establecer la linea que separa el controlador y la vista de forma distinta, por ejemplo en algunas implementaciones el controlador se encarga tanto de actualizar el modelo y las vistas como de responder a los mensajes externos, es decir que el controlador es el encargado de ejecutar toda la lógica, y la vista meramente contiene la presentación de los datos, en otras implementaciones, la vista es la encargada tanto de seleccionar los datos que van a mostrarse así como de desplegar la presentación, esta ultima es una variación de MVC a veces llamado MTV (Modelo-Template-Vista)

2.7.2 Django

Django es un framework de desarrollo de aplicaciones web construido en python, en este trabajo usamos Django como el fundamento para Octopus Head. Django permite construir aplicaciones web rápidamente gracias a su filosofía de "baterias incluidas", es decir, que incluye una inmensa gama de características comunes a la mayoría de las aplicaciones web como validaciones de formularios, autenticación de usuarios, manejo de sesiones entre muchos otros [21]; así el desarrollador puede concentrarse en escribir la lógica que es especifica a su aplicación y dejar que Django maneje los aspectos repetitivos y muchas veces tediosos de la pila de desarrollo web.

Django esta diseñado con una arquitectura MVC es decir que separa claramente la lógica, de los datos y la forma en que dichos datos son presentados al usuario, en el caso de Django la "vista" describe que datos son presentados al usuario y el "template" representa la forma en que dichos datos son presentados.

Cuando Django recibe una petición esta pasa por un despachador de URLs (URL Dispatcher), cuya tarea es emparejar el URL con una vista y delegar a la vista el manejo de la petición. La vista contiene la lógica necesaria para atender la petición entrante, generalmente esto consiste en retirar o actualizar algunos datos del modelo, que a su vez se comunica con el manejador de base de datos, finalmente la vista combina los datos retirados de la base de datos, la petición y la sesión activa con una plantilla (template) para generar la respuesta que será devuelta, en la figura 2.4 puede verse el ciclo de petición-respuesta tal como se ha descrito.

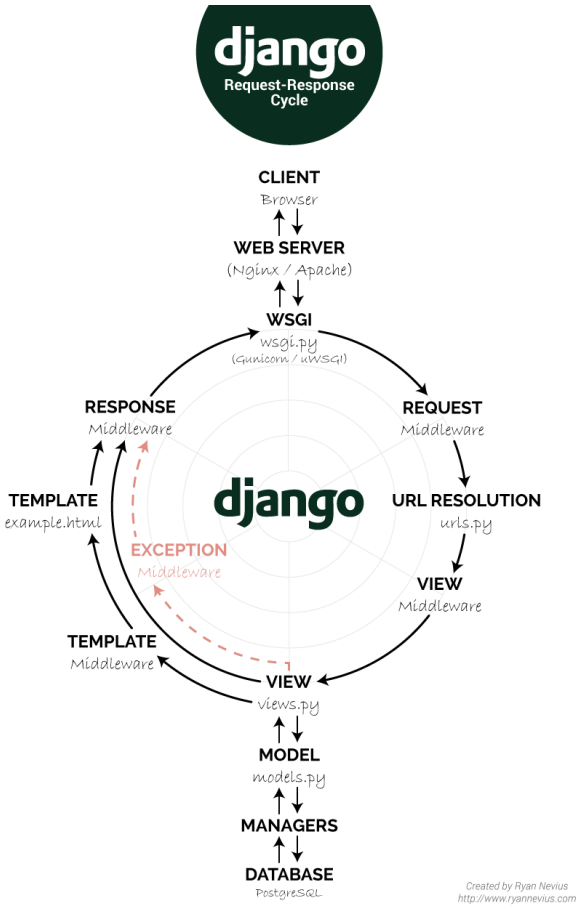

Figura 2.4: Ciclo de peticion-respuesta de Django

Las respuestas generadas por Django pueden ser tanto paginas HTML con CSS y Javascript pensadas para la interacción con el usuario así como respuestas en formato json o xml para la construcciones de APIs que pueden ser usados para la comunicación

maquina-maquina, esto es especialmente útil cuando se desea desarrollar aplicaciones en otras plataformas como Android o iOS que compartan el mismo backend.

Django ha demostrado ser escalable y flexible, se sabe de instancias de Django atendiendo ráfagas de cincuenta mil peticiones por segundo, ademas es de código abierto, gratuito y cuenta con una enorme comunidad de colaboradores y amplia documentación.

2.7.3 Celery

Celery es un sistema de procesamiento de tareas asíncrono, que permite tanto el encolamiento de tareas en tiempo real así como la planificación de tareas para ser ejecutadas mas tarde. Celery en realidad no implementa la mayoría de sus componentes, sino que define un protocolo de comunicación entre una serie de componentes (tambien llamados micro-servicios)[22]:

- Bróker de mensajería: es el encargado almacenar las tareas que deben ser ejecutadas y pasarlas entre los distintos entes, este consiste de una o mas colas de prioridad, de esta manera es posible controlar finamente que tareas deben ser ejecutadas con mayor urgencia, y por que trabajadores.

- Planificador: el planificador es el encargado de encolar tareas previamente planificadas para que sean ejecutadas en algún momento especifico.

- Workers (Ejecutores): como su nombre lo indica, estos son los encargados de ejecutar las tareas; estos se subscriben a una o mas colas y consumen los mensajes según la prioridad establecida en la cola finalmente escriben el estado y resultados obtenidos en un backend de resultados

- Backend de Resultados: sirve como un almacén donde se guarda el estado y los resultados de las tareas durante un periodo de tiempo dado, esto hace posible hacer seguimiento de las tareas y obtener sus valores de retorno.

Uno de los problemas comunes en el desarrollo de aplicaciones web es la ejecución de tareas largas o altamente bloqueantes que mantienen ocupado al servidor web durante

periodos prolongados y peor aun mantienen al usuario final esperando, para evitar este escenario el servidor web debe tener una forma de delegar este tipo de tareas para ser ejecutadas después y responder inmediatamente.

Celery es usado comúnmente junto a Django y permite a una aplicación web escalar a bajo costo ya que solo hace falta aumentar el número de trabajadores disponibles que se pueden distribuir en tantas maquinas como sea necesario.

2.7.4 Redis

Redis es un almacén de estructuras de datos en memoria que soporta una amplia gama de tipos como cadenas de caracteres, hashes, listas, conjuntos, conjuntos ordenados, mapas de bits, indices geo-espaciales, etc [28].

Redis es comúnmente usado como base de datos, cache o bróker de mensajería; y se caracteriza por su velocidad de respuesta ya que todos sus datos se mantienen en memoria principal y solo invierte recursos en asegurar ciertos niveles de persistencia, sin embargo, por omisión, los datos almacenados se pierden en caso de que ocurra cualquier falla inesperada.

2.7.5 Bases de Datos

Uno de los pilares fundamentales de casi toda aplicación moderna es tener un modo de almacenar datos de forma persistente en el tiempo así como consultarlos y actualizarlos de forma rápida, segura y resistente a fallas.

Según [23] una base de datos es una colección de datos estructurados. Puede ser cualquier cosa desde una simple lista de compras, una galería de fotos o las bastas cantidades de información en una red corporativa. Para agregar, acceder y procesar la data almacenada en una base de datos, se necesita un sistema manejador de base de datos. Ya que los computadores hacen un muy buen trabajo manejando grandes cantidades de datos, los sistemas manejadores de bases de datos juegan un papel central en la computación como utilidades independientes o partes de otras aplicaciones.

SQL por sus siglas en ingles "Structured Query Language" (lenguaje de consulta estructurado) es el lenguaje estandarizado mas común para acceder a bases de datos,

SQL está definido por el Estándar ANSI/ISO SQL y ha ido evolucionando desde 1986 para convertirse en un estándar de facto en el mundo de la computación.

2.7.5.1 Mysql

Mysql es un sistema manejador de bases de datos relacionales (RDBMS por sus siglas en ingles) de codigo abierto bajo la licencia GPL (GNU General Public License). Mysql se caracteriza por ser rápido, confiable, escalable y fácil de usar, es posible instalar Mysql tanto en una maquina junto a otras aplicaciones como servidores web o también instarlo en maquinas dedicadas para que use todo el poder de cómputo disponible. Mysql posee características para ejecutarse en clusters de maquinas junto con un motor de replicacion para obtener una alta escalabilidad [23].

2.7.5.2 Mapeo Objeto-Relacional

El Mapeo Objeto-Relacional (ORM por sus siglas en ingles) es un método para interactuar con bases de datos relaciones desde el paradigma de la programación orientada a objetos, de esta manera es posible aprovechar conceptos como herencia y polimorfismo.

Según [24] la mayoría de las aplicaciones modernas usan lenguajes orientados a objetos como Java o C# para construir aplicaciones y bases de datos estructuradas para almacenar datos, por lo tanto, es util tener una interfaz que transforme los datos entre estos tipos de incompatibles.

El uso de un ORM simplifica enormemente el manejo de la estructura de datos subyacente ya que permite al programador manejar los datos a un mayor nivel de abstracción como si fueran objetos, sin necesidad de generar manualmente las consultas SQL, ademas esta capa de abstracción permite desacoplar el código de la aplicación de los detalles específicos de cada RDBMS.

2.7.6 Json (JavaScript Object Notation)

Json (JavaScript Object Notation) o notación de objectos javascript es un formato textual de intercambio y almacenamiento de datos no estructurados, json posee un

formato que es fácil de leer para humanos y fácil de interpretar para maquinas y mas ligero que XML por lo que se ha popularizado para el desarrollo de APIs.

Json soporta dos tipos de estructuras de datos:

1. Colecciones de pares <nombre,valor> comparable a un diccionario o tablashash.
2. Listas ordenadas de valores, similar a las listas o vectores que existen en virtualmente cualquier lenguaje de programación

Un archivo en formato json puede estar formado de cualquiera de las estructuras de datos antes descritas o cualquier permutación de dichas estructuras anidadas.

Los tipos de datos soportados por json son:

- Number: numeros flotantes de doble presicion en formato Javascript
- String: cadenas de caracteres unicode encerradas en dobles comillas
- Boolean: true o false
- Arreglo: secuencia de valores ordenados
- Objeto: colección no ordenada de pares <nombre,valor>
- Null: nulo o vacío

2.7.7 Dropbox

Dropbox es una plataforma de almacenamiento de datos en la nube de tipo PaaS y SaaS que permite compartir y sincronizar archivos entre un número arbitrario de clientes.

Dropbox es usado por aproximadamente 400 millones de personas y 100.000 organizaciones[25] y posee aplicaciones en Windows, Linux, Mac OS X, iOS, Android, Blackberry y web.

Como todo servicio en la nube es atractivo para desarrolladores por ser robusto y confiable y es gratis hasta alcanzar una cierta cantidad de espacio de almacenamiento usado, a partir de ese punto incluye planes de pago que dependen de la cantidad de espacio usado.

2.7.7.1 API de Dropbox

Un API (Aplication Programming Interface) o interfaz de programación de aplicaciones, es una serie de métodos o funciones orientados a la comunicación maquina-maquina, en el caso de la computación en la nube un API conforma un servicio que permite desarrollar aplicaciones sobre una plataforma (en este caso Dropbox).

El API de Dropbox permite realizar peticiones (como subir o descargar archivos, listar directorios o crear carpetas) sobre el espacio de almacenamiento de un usuario, el usuario debe previamente dar permiso a la aplicación para que esta pueda hacer cambios a su nombre, el usuario puede elegir denegar el acceso a la aplicación en todo momento y existen distintos tipos de esquemas de acceso donde una aplicación solo tiene acceso a un conjunto limitado de directorios dentro del espacio de almacenamiento del usuario.

El API de Dropbox impone ciertos límites de peticiones por usuario para impedir que una aplicación realice una cantidad excesiva de peticiones en un periodo corto de tiempo, sin embargo el limite se considera lo suficientemente alto como para no entorpecer la inmensa mayoría de los casos de uso.

2.7.8 Diseño web adaptable

Debido a la inmensa diversidad de dispositivos desplegados en el mercado y sus distintos tamaños y formas es imposible realizar manualmente diseños que puedan ajustarse a cada uno de ellos, anteriormente una solución popular a este problema era tener varias versiones con distintas resoluciones y elegir que versión mostrar a cada cliente, sin embargo esto ya no es necesario gracias a las nuevas herramientas disponibles en HTML5 CSS y Javascript que están ampliamente implementadas en navegadores modernos.

El diseño web adaptable o "Responsive Web Design" es la tendencia en el diseño de paginas web que se ajusten elásticamente a cualquier resolución, adaptando la forma en que se presentan sus elementos de forma "inteligente".

Las técnicas mas comunes para lograr esto es tener elementos que ocupen el mayor espacio horizontal posible en pantallas grandes y mientras el tamaño horizontal se

reduce estos pasan a ocupar el espacio verticalmente; siempre ocupando el máximo del ancho disponible, evitando crear barras de desplazamiento horizontal que desorientan e incomodan a los usuarios.

Otra heuristica en la creación de sitios web adaptables es escalar o esconder elementos gráficos decorativos o menús de navegación laterales, reducir el tamaño de margenes o incluso cambiar tipos de letras para que sean mas legibles en dispositivos móviles, mientras el tamaño del dispositivo es menor, cada pixel se vuelve mas precioso.

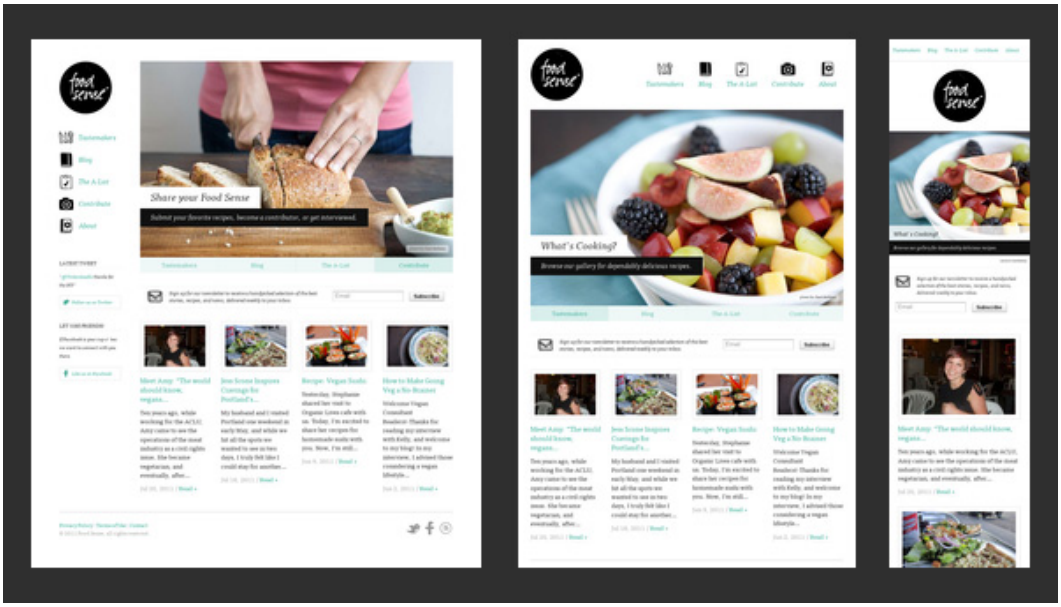

Figura 2.5: Ejemplo de diseño adaptable, se puede ver como los elementos se reagrupan, escalan o esconden para ajustarse al tamaño de la pantalla.

El diseño web adaptable no solo facilita el desarrollo de sitios web ahorrando a los desarrolladores y diseñadores el costo de construir múltiples versiones de un mismo sitio web sino que ademas el sitio web ofrece una experiencia similar independientemente del dispositivo con el que se esté visitando.

2.7.9 Highcharts

Highcharts es una biblioteca Javascript para dibujar gráficas en entornos HTML es gratis para proyectos no comerciales y de código abierto. Las gráficas generadas por highcharts aprovechan las características de HTML5 por lo que pueden soportar

enormes conjuntos de datos sin afectar el rendimiento incluso en dispositivos móviles, son totalmente interactivas de manera que el usuario puede inspeccionar detalladamente el conjunto de datos y son dinámicas permitiendo actualizar la gráfica en tiempo real cuando se reciben nuevos datos.

Highcharts incluye una amplia gama de gráficas predefinidas que la hace ideal para todo tipo de visualización de datos como mapas de calor, splines, gráficas de área, barras, pie, mapas, entre muchos otros; cada una de ellas ofrece un gran control sobre la forma en que son presentadas de modo que es posible lograr casi cualquier resultado deseado.

Desde el punto de vista del programador es muy fácil de usar ya que solo requiere especificar un "objeto de configuración" y uno o mas arreglos conteniendo los datos a desplegar, es posible obtener distintas representaciones de los mismos datos solo cambiando el objeto de configuración.

2.7.10 Google Maps

Google Maps es el servicio de mapas web de Google, ofrece distintos tipos de mapas, como mapas viales, mapas de relieve e incluso imágenes satelitales e imágenes de calles en tercera dimensión (llamado Google Street View)

Los mapas de Google Maps son totalmente dinámicos, permitiendo al usuario desplazarse, hacer zoom y cambiar el tipo de mapa a voluntad, Google Maps funciona dividendo el espacio mostrado en sectores que son descargados individualmente, de manera que cuando el usuario desplaza el mapa solo es necesario descargar los nuevos sectores desde los servidores de Google.

Google Maps es una de las herramientas mas populares para dibujar mapas web ya que ofrece un API gratuito y fácil de usar que permite enmarcar mapas en cualquier sitio web con solo algunas lineas, ademas los mapas de Google son de altisima calidad y se mantienen constantemente actualizados.

2.7.11 Geo-localizacion IP

La geo-localizacion IP consiste en asignar a un IP la localizacion geográfica de la maquina anfitriona correspondiente[26].

Existen dos paradigmas principales para aproximar la localización geográfica de una dirección IP: activo y pasivo; las técnicas activas de localización se basan en mediciones de retraso y en muchos casos proveen resultados precisos[26], el paradigma pasivo consiste del uso de bases de datos que contienen rangos de direcciones ips a los que se les llaman bloques o prefijos relacionados a una localización geográfica especifica, sin embargo su precisión puede estar sujeta a errores substanciales. En ambos casos es imposible conocer la localización exacta asociada a una dirección IP sin la colaboración activa de los anfitriones finales, sin embargo es posible hacer buenas aproximaciones en algunos a nivel de ciudades o países.

Ya que conocer la localización de sus clientes a partir de su dirección ip es útil para muchos servicios (por ejemplo, para conducir anuncios localizados) existe una gran variedad de soluciones de geolocalizacion tanto gratuitas como de pago.

2.7.12 Ajax

AJAX (Asynchronous JavaScript And XML) es una técnica para construir sitios web interactivos a través de peticiones asíncronas con Javascript que mantienen comunicación con el servidor web para mantener el estado del cliente actualizado sin necesidad de que el usuario tenga que refrescar la pagina o realizar consultas adicionales, de esta manera el usuario tiene una experiencia similar a la que tendría con aplicaciones de escritorio.

A pesar de que el nombre AJAX sugiera el uso XML como lenguaje para la transferencia de datos entre el cliente y el servidor, se puede usar cualquier formato como texto plano, HTML y JSON

2.7.13 APScheduler

APScheduler (Advanced Python Scheduler) es una biblioteca python para retrasar la ejecución de rutinas a un instante dado en el futuro ya sea como eventos de una sola

vez o de forma recurrente[27], esta biblioteca provee las herramientas para construir cualquier esquema de planificación que se desee su arquitectura consta de componentes altamente desacoplados que pueden ser mezclados, combinados o incluso extendidos para ofrecer nuevas funcionalides.

2.7.13.1 Triggers (gatillos)

Los triggers (gatillos) son los encargados de determinar el momento de la próxima ejecución de una tarea, APScheduler incluye tres gatillos predefinidos: DateTrigger que ejecuta tareas dada una fecha y hora, IntervalTrigger que ejecuta tareas de forma recurrente dado un intervalo fijo y CronTrigger que ejecuta tareas de forma similar a crontab del sistema UNIX.

2.7.13.2 Job Stores (almacenes de tareas)

Job Stores o almacenes de tareas determinan la forma en que las tareas serán alojadas por el planificador, por defecto las tareas se guardan en memoria, sin embargo también es posible guardar las tareas en almacenes persistentes como bases de datos SQL o mongo.

2.7.13.3 Executors (ejecutores)

Los ejecutores son los encargados de ejecutar tareas y posteriormente informar al planificador del estado de la tarea. El ejecutor por defecto consta de un arreglo de hilos (thread pool) pre-instanciados listos para aceptar tareas, sin embargo si se tienen tareas de uso intensivo de CPU, esta disponible un ejecutor basado en procesos que puede aprovechar mejor las características de procesadores multinúcleos [27].

2.7.13.4 Shedulers (Planificadores)

Los planificadores son los que unen todos los elementos mencionados anteriores y son los encargados de gestionar las tareas, es decir que delegan a los ejecutores la ejecución de las tareas en el tiempo adecuado y duerme esperando la ocurrencia de eventos que requieran su atención.

APScheduler incluye dos planificadores predefinidos: BackgroundScheduler que se ejecuta instanciando un hilo en segundo plano y permite que el flujo del programa continúe, útil cuando se desea realizar otras funciones o como agregar o modificar tareas después de que el planificador se ha inicializado y BlockingScheduler que se ejecuta en el mismo hilo de ejecución bloqueando el flujo del programa indefinidamente.

2.7.14 Ping

Ping es un programa utilitario incluido en todos los sistemas basados en UNIX y Windows que comprueba la presencia y tiempo de respuesta de un host en una red IP. Ping utiliza el Protocolo de Mensajes de Control de Internet (ICMP) para enviar un paquete de solicitud ICMP (ICMP Echo Request) y espera el mensaje de respuesta del host remoto (ICMP Echo Reply); calculando la diferencia de tiempo entre el envío y la recepción se puede calcular la latencia de la red, ping también incluye funciones para enviar paquetes en ráfaga útil cuando se desea medir la perdida de paquetes.

Ya que históricamente ping se ha usado por atacantes para determinar la presencia de equipos en una red o realizar ataques de denegación de servicio (ping flood) muchos enrutadores y firewalls bloquean estos mensajes como medida de seguridad, aunado a esto ya que ping utiliza ICMP que es un protocolo de capa de red, este no es capaz de alcanzar equipos detrás de un NAT; a pesar de esto, ping ha demostrado ser una herramienta vital en el diagnostico y monitoreo de redes IP.

2.7.15 Traceroute

Traceroute (tambien llamado Tracert en sistemas Windows) es un programa utilitario de diagnostico que permite conocer los hosts que visita un paquete durante su transito por una red.

Al igual que Ping, Traceroute utiliza el protocolo ICMP pero envía paquetes con un valor de "Time to Live" (TTL) incremental, cada vez que un nodo de la red recibe un paquete decrementa su valor de TTL y si este llega a cero lo descarta y envía de vuelta al host emisor un mensaje de control indicando que el TTL llegó a 0, de esta manera Traceroute puede generar una lista de los nodos visitados y el valor de RTT

para cada uno de ellos.

Un análisis cuidadoso de la salida de Traceroute puede ayudar a diagnosticar numerosos problemas en una red como ineficiencias en el enrutamiento, presencia de enrutadores congestionados, cuellos de botella, comportamientos inesperados, etc.

2.7.16 Iperf

Iperf es una herramienta que permite medir el rendimiento (throughput) entre un par de nodos en un red. Al igual que muchas otras pruebas para medir velocidad de transferencia, Iperf funciona con una arquitectura cliente-servidor, donde el cliente genera un flujo de datos hacia el servidor y mide la velocidad obtenida, también es posible ejecutar una prueba "en reversa" donde es el servidor el que genera el flujo de datos.

Iperf puede ejecutar pruebas utilizando TCP o UDP, sin embargo existen diferencias entre ellos:

- Ya que UDP a diferencia de TCP no implementa ningún algoritmo de control de congestión se podría obtener un rendimiento ligeramente superior con este protocolo.
- Con UDP se puede obtener una estadística de la perdida de datagramas durante la prueba.
- Con UDP es el servidor el que totaliza los resultados, ya que el cliente no tiene manera de saber que datagramas se han recibido.

Para obtener una buena medición del rendimiento del enlace hay que ajustar los parámetros de la prueba cuidadosamente, por ejemplo, es posible ajustar la cantidad de datos que se van a transferir durante la prueba, elegir una cantidad muy pequeña podría resultar en que no sea suficiente para saturar el enlace, mientras tanto elegir una cantidad demasiado grande podría resultar en una prueba innecesariamente larga, en ambos casos el valor del rendimiento obtenido no reflejará la realidad, también hay que tomar en cuenta que es difícil obtener una lectura exacta del rendimiento del enlace ya que podrían existir otros flujos en la red que afecten el resultado de la prueba.

Capítulo 3

Aplicación Web "Optopus Head"

El sistema de monitoreo de redes "Octopus Monitor" consiste de cuatro elementos principales: (1) La aplicación web "Octopus Head" que funciona como cerebro y coordinador del monitoreo, (2) un conjunto de agentes monitores de red "Tentacle Probe Source" que realizan acciones de monitoreo a partir de las instrucciones de Octopus Head, (3) Nodos a monitorear "Tentacle Probe Destination" y (4) La nube, que ofrece funcionalidades de paso de mensajes y alojamiento de datos compartido entre los distintos entes.

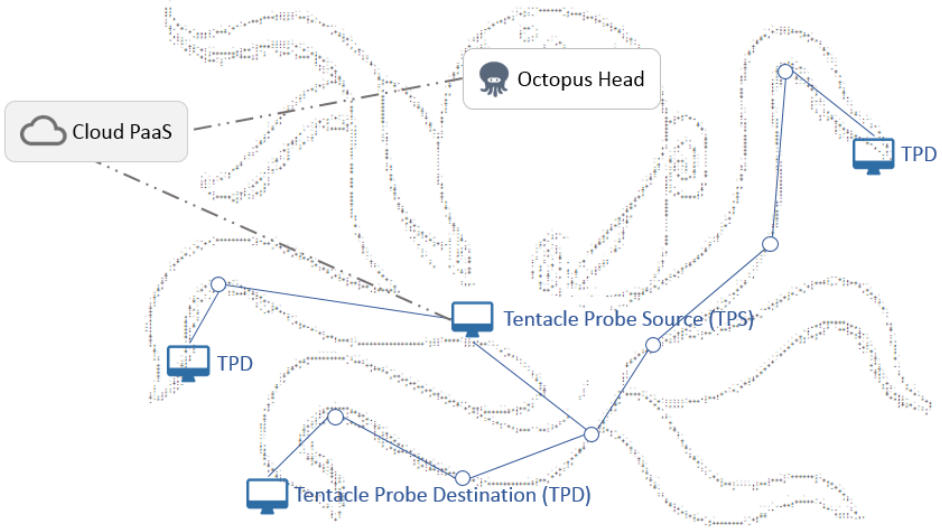

Figura 3.1: Sistema de Monitoreo "Octopus Monitor"

En la Figura 3.1 se puede ver como hemos usado una analogía sencilla para identificar todos los entes que participan en el sistema, la cabeza del pulpo representa a la aplicacion web, los tentáculos conformados por monitores Tentacle Probe Sorce y nodos Tentacle Probe Destination en sus extremos, representan los enlaces a monitorear, se desea que usuarios no-técnicos puedan entender fácilmente el esquema del sistema y así puedan monitorear sus redes y servicios.

Los monitores de red remotos funcionan bajo una planificación dada, a la que llamaremos plan de monitoreo, es en la aplicacion web que los usuarios pueden definir (y re-definir) las políticas de monitoreo y sincronización de los datos, toda la comunicación entre la aplicacion web y los monitores de red ocurre a través de la nube, la aplicacion web publica mensajes destinados a un monitor específico y los monitores a su vez son notificados por el servicio de la nube cuando hay nuevos mensajes para ellos, el monitor de red se mantiene constantemente actualizado sobre los cambios realizados al plan de monitoreo y regularmente sube los datos que obtiene a partir de las pruebas, la frecuencia de actualización también puede estar sujeta a ciertas políticas y restricciones de ancho de banda.

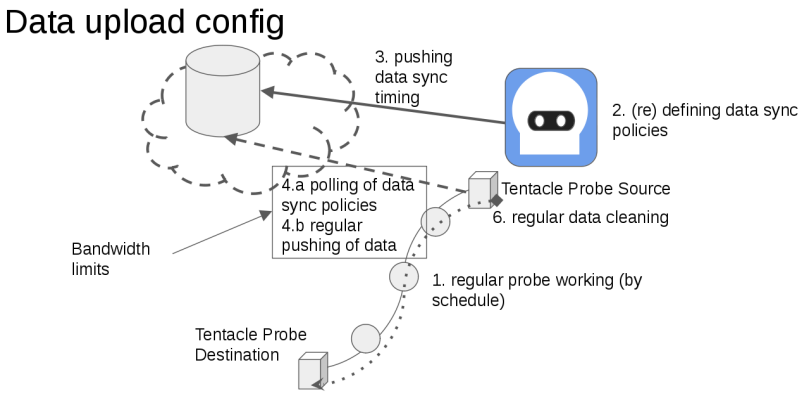

Figura 3.2: Planificación del monitoreo

Este esquema asegura que la aplicacion web sirva como interfaz entre el usuario y los monitores, esto es especialmente útil ya que los monitores de red podrían potencialmente ser abandonados en sitios de difícil acceso, se hablará a fondo de los monitores de red en el Capítulo 4.

La aplicacion web es ademas la responsable de desplegar visualizaciones a partir de los datos obtenidos del monitoreo, para esto, los datos subidos por los monitores son recolectados regularmente y procesados en un formato que facilite el procesamiento de las visualizaciones, es necesario tener una arquitectura que permita manejar los recursos disponibles de forma eficiente y así poder tanto ofrecer un servicio web rápido como planificar y ejecutar tareas computacionalmente intensivas.

3.1 Diseño de la aplicacion web

Para el desarrollo de esta aplicación se usó el Framework de desarrollo web Django que implementa un patrón MVC, ya que nuestra aplicación web se encargará de manejar tareas computacionalmente intensas, o de largo tiempo de ejecución, Django se integró con el sistema de procesamiento de tareas distribuido Celery, esto no solo con la finalidad de que el servidor web pueda delegar estas tareas y responder rápidamente al usuario, sino también para permitir una mejor escalabilidad del sistema, que entonces podrá responder a un mayor número de peticiones por unidad de tiempo.

3.1.1 Arquitectura

La arquitectura de Octopus Head consiste en cuatro capas, la capa superior o capa de presentación se ejecuta en el navegador del cliente monitor, administrador o quien sea que vea los resultados obtenidos a partir del monitoreo, esta capa se comunica con la capa dos o capa de servicio que es ejecutada por el servidor web, este responde a las peticiones de los usuarios, delega tareas a la capa tres y retira y actualiza datos de la capa cuatro, la capa tres es la encargada de ejecutar tareas largas o computacionalmente intensas así como de planificar e iniciar tareas periódicas, la capa cuatro o capa de datos aloja los datos de la aplicación como usuarios, monitores, planes de monitoreo, historiales, datos recolectados de las redes, reportes, cache, datos en la nube, etc.

NOTA: Agregar "carriles" para que queden claras las "capas", actualizar la capa de servicio web para incluir el servidor estatico y el proxy, agregar el cache.

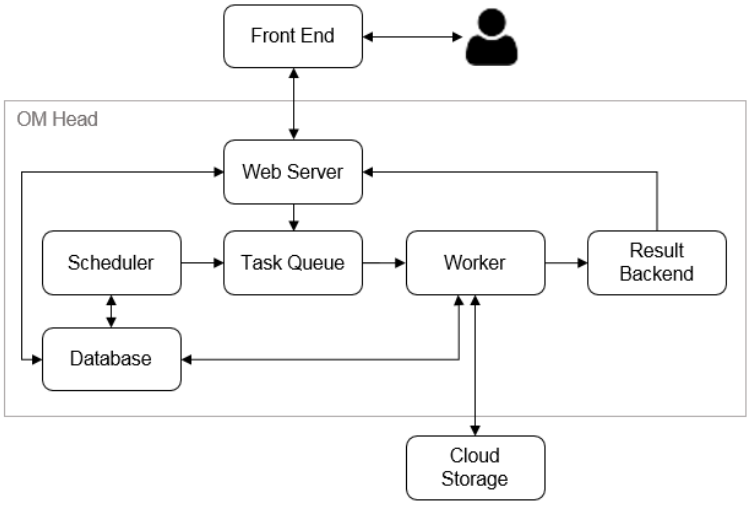

Figura 3.3: Arquitectura de la aplicacion web.

3.1.2 Capa 1: Presentación

La capa de presentación o front-end es la interfaz gráfica que permite la interacción del usuario con el sistema de monitoreo, esta se despliega a través de un navegador en cualquier dispositivo conectado a Internet y su propósito es ofrecer al usuario un flujo de trabajo claro para ingresar al sistema, configurar y manejar monitores TPS, planificar pruebas, observar los resultados obtenidos, entre otros.

La capa de presentación puede ser extendida para permitir la implementación de distintos front-ends como aplicaciones nativas para sistemas operativos móviles o de escritorio a través de APIs programáticas.

3.1.3 Capa 2: Servidor Web

El servidor web es el punto de entrada a la aplicacion, este responde a las peticiones realizadas desde el front-end a través del protocolo HTTP, el enfoque del servidor web es manejar tareas ligeras de la forma mas rápida posible y retornar respuestas para ofrecer al usuario baja latencia en su interacción con el sistema.

La ejecucion de tareas largas y pesadas es delegada a la capa de procesamiento asíncrono de tareas, de esta manera un solo usuario no ocupa los recursos del servidor

	Acción
/static/	static serve
/media/	static serve
/	proxy to web app

Tabla 3.1: Tabla de enrutamiento del servidor web

web durante mucho tiempo independientemente de la petición que realice; esto hace posible manejar un mayor numero de usuarios que visiten concurrentemente el sitio, igualmente el servidor web no se sobrecarga aunque exista una alta carga de trabajos de fondo como computo de visualizaciones y sincronizaciones.

El servidor web debe manejar dos tipos de peticiones: aquellas que requieren respuestas dinámicas ajustadas a cada usuario particular que visite el sistema y peticiones de archivos estáticos como css, javascript, imágenes o archivos html predefinidos que cambien poco o rara vez. A pesar de que la aplicacion web es capaz de manejar ambos tipos de peticiones, cada una de ellas tendrá que pasar por todo el pipeline de Django (middlewares, resolución de URLs, procesamiento de la vista, etc), esta complejidad es innecesaria cuando se trata de archivos estáticos y va a sobrecargar severamente el servidor web en entornos de producción.

Para manejar eficientemente todo tipo de peticiones, debemos implementar un mecanismo que sea capaz de servir los archivos estáticos rápidamente y que sirva como proxy entre el front-end y la aplicacion web, para esto existe una gran variedad de servidores ligeros como ngnix o lighthttpd, está tendrá una sencilla tabla de enrutamiento que decidirá como manejar las peticiones. la aplicacion web podrá correr entonces en un proceso separado sobre un servidor web y solo atenderá peticiones dinámicas y se asegurará la eficiencia en el uso de los recursos computaciones disponibles.

3.1.4 Capa 3: Procesamiento asíncrono de tareas

Mantener el servidor web ocupado con tareas largas puede degradar su calidad de servicio y ocupar rápidamente sus recursos disponibles, mas aún muchas veces no es posible dejar al usuario esperando indefinidamente, por este motivo es menester tener

un sistema de procesamiento asíncrono de tareas, de manera que las tareas se puedan mandar a ejecutar y ofrecer una respuesta inmediata al usuario.

Ya que es imposible conocer de antemano el número de tareas que se van a estar ejecutando concurrentemente necesitamos una manera de encolar las tareas para que puedan ser ejecutadas cuando alguno de los trabajadores se desocupe, de igual forma, para la implementación de de sincronización periódicas de los monitores, necesitamos tener un planificar que inicie las tareas según el horario definido.

La inmensa mayoría de las peticiones realizadas por los usuarios pueden ser manejadas directamente por el servidor web, las tareas delegadas a este capa son las siguientes:

- **Conexiones con la nube:** A pesar de que la mayoría de las tareas que incluyen conexiones con el API de la nube son bastante sencillas, estas tareas pueden considerarse "de largo tiempo de ejecucion" ya que es necesario obtener recursos presentes en Internet.

- **Computo de visualizaciones:** el tiempo y poder de computo necesario para generar visualizaciones depende altamente de la visualización y la cantidad de datos involucrados, algunas de ellas como mapas de calor, periodos de actividad continua o días y horas activos pueden tardar decenas de segundos en calcularse y requerir una cantidad extraordinaria de computo especialmente cuando se están visualizando largos periodos de tiempo.

- **Sincronizaciones:** las sincronizaciones combinan varias subtareas altamente costosas: la primera de ellas es descargar los archivos de la nube, esto depende altamente de la cantidad de archivos por descargar y el tamaño de los archivos, la segunda es procesar los archivos para convertirlos a un formato adecuado para pasarlos a la base de datos, la tercera es la inserción a la base de datos, la cual depende de los indices de la tabla y la cantidad de datos en la base de datos.

El sistema de procesamiento asíncrono de tareas está construido usando Celery, Celery no es solo cada uno de los micro-servicios necesarios para el procesamiento distribuido de las tareas, sino también el protocolo de comunicación entre ellos. Hay que tener en mente que el esquema de Celery **no incluye ninguna autoridad central**

que lleve la cuenta de todos los entes participantes. A continuación se explica cada uno de los micro-servicios y su implementación en el marco de Octopus Head.

3.1.4.1 Planificador

El planificador retrasa la ejecucion de tareas según un horario dado, Celery incluye un planificador llamado Celery Beat que se puede configurar fácilmente para usar la base de datos de la aplicacion web a modo de almacén para sus tareas a ejecutar. Celery Beat consulta constantemente una tabla en la base de datos buscando cambios en las tareas planificadas, la aplicacion web solo modifica esa tabla y deja que Celery Beat se encargue de inciar las tareas en el momento adecuado.

El planificador Celery Beat no funciona muy distinto al planificador usado por el monitor, a diferencia de que Celery Beat no manda a ejecutar las tareas directamente por un trabajador sino que las inserta a la cola de tareas, es por esto que no es posible asegurar que las tareas se ejecuten justo en el momento planificado, sino que puede existir un retraso variable en la ejecucion basado en el tamaño de las colas y la carga de trabajo de los trabajadores.

3.1.4.2 Broker de Mensajería

El broker de mensajería es un micro-servicio responsable de pasar mensajes entre los entes que encolan tareas (aplicacion web y planificador) y los trabajadores que las ejecutan usando una o mas colas de prioridad. Es posible enrutar tareas a trabajadores específicos a través del uso de múltiples colas, por ejemplo sería posible tener una cola para las tareas de sincronización, y otra cola para el computo de visualizaciones y subscribir un distinto número de trabajadores a cada una de ellas. Este sistema asegura una gran granularidad para ejercer control en el orden de ejecucion de las tareas, ya que ademas es posible establecer prioridades, mandando ciertas tareas rápidamente al tope la cola.

Establecer un esquema apropiado de prioridades y encolamiento es esencial para asegurar un tiempo de espera óptimo, las tareas de cómputo de visualizaciones deben tener una muy baja latencia ya que generalmente el usuario está esperando por una respuesta lo más rápida posible, la subida de mensajes a la carpeta del monitor puede

tener una latencia ligeramente mayor, sin embargo no mayor a unos 10 segundos, las sincronizaciones planificadas por otra parte pueden tener un mayor tiempo de espera en cola donde 10-60 segundos es aceptable.

Existen distintos sistemas que pueden hacer el papel de broker de mensajería, algunas opciones populares incluyen RabbitMQ, un sistema de mensajería robusto y persistente, Redis del que se habló en la Sección 2.7.4 o algún SMBD, la elección del message broker debe depender de volumen de tareas que se va a manejar, el uso de base de datos puede ser aceptable cuando se maneja una cantidad reducida de tareas, pero sistemas más rápidos como RabbitMQ y Redis son preferibles en entornos de producción.

3.1.4.3 Trabajadores

Los trabajadores son los encargados de ejecutar las tareas encoladas, estos se subscriben a una o mas colas y toman las tareas en el tope, determinar la cantidad necesaria de trabajadores para cada cola es esencial para evitar que las colas crezcan indefinidamente (y por lo tanto tambien, el tiempo de espera de las tareas).

Tener múltiples trabajadores permite tener un gran control sobre los recursos computacionales usados por el sistema, es posible instanciar nuevos trabajadores ejecutándose en distintas maquinas físicas durante los momentos de alta carga del sistema, y liberar estos recursos cuando no se estén usando, esto hace posible manejar una escala variable y minimiza los costos de operación.

Ya que los trabajadores se pueden estar ejecutando en múltiples máquinas físicas es necesario tener alguna manera de controlarlos remotamente, para esto, los trabajadores se subscriben a una cola de 'broadcasting' de alta prioridad en la que se pueden dirigir comandos a todos o a un grupo específico de trabajadores. Existe un conjunto de comandos predefinidos para realizar acciones básicas como revocar tareas, apagar trabajadores o hacer "ping", sin embargo, también es posible implementar nuevos comandos, por ejemplo sería posible implementar un comando para que los trabajadores puedan actualizar su propio código y reiniciar, de este modo podrían aceptar nuevas tareas implementadas.

3.1.4.4 Backend de Resultados

Ya que Celery es un sistema distribuido sin una autoridad central que conozca el estado de todo el sistema, es necesario tener algún micro-servicio donde los distintos entes puedan publicar y consultar el estado de las tareas y su valor de retorno. Tener un backend de resultados permite hacer seguimiento del estado de las tareas e implementar mecanismos como barras de carga que indiquen el progreso de alguna tarea particularmente costosa, es posible crear estados personalizados para estos fines, los estados incorporados por omisión en el sistema se pueden ver en la Tabla 3.1.4.4

Estado	Descripcion	Metadata
PENDING	La tarea está esperando ser ejecutada	-
STARTED	La tarea ha sido iniciada	pid y hostname del trabajador
SUCESS	La tarea se ejecuto con exito	valor de retorno de la tarea
FAILURE	La tarea no se ejecuto con exito	traza de la exepcion
RETRY	Se está reintando ejecutar la tarea	traza de la última excepcion
REVOKED	La tarea se ha cancelado	-

Tabla 3.2: Estados de las tareas

Para este sistema hemos elegido Redis como backend, ya que guarda el estado de las tares en memoria ofreciendo consultas rápidas y poco costosas a costo, el comportamiento por defecto del backend es poner un tiempo de vencimiento a sus registros, pasado el cual son eliminados y no pueden volver a ser consultados.

3.1.5 Capa 4: Datos

La capa de datos incluye todos los sistemas encargados de almacenar o alojar datos de forma persistente ya sea a través de bases de datos, estructuras de archivos o la nube.

3.1.5.1 Base de Datos

La base de datos es el almacén principal de los datos estructurados de la aplicacion web, almacena todo tipo de información como preferencias del usuario, datos de los

monitores, horario de sincronizaciones, parametros de conexion a Dropbox, y sirve como backend del Framework de Integracion de Pruebas del que se hablará en el capitulo 5.

El diseño de la base de datos es uno de los pilares fundamentales del diseño del sistema, el correcto diseño e indexación de las tablas de la base de datos tiene repercusiones importantes en la escalabilidad y tiempo de respuesta del sistema, el diseño de la base de datos se explica a fondo en la Sección 3.1.6

3.1.5.2 Caché de archivos

El caché de archivos es un almacén de archivos con el objetivo principal de almacenar visualizaciones pre-computadas y así evitar la repetición de procesamiento ya realizado.

Los archivos se guardan en distintos directorios dependiendo del tipo de visualización, cada visualización tiene un código hash asociado que se genera a partir de sus parámetros, por lo tanto el acceso a los archivos en el caché es directo, la velocidad de búsqueda en el caché depende de la implementación del sistema de archivos (por ejemplo ext3, ext4, XFS), los sistemas de archivos modernos pueden manejar cientos de miles de archivos en el mismo directorio sin ningún problema de rendimiento. Usar un caché de archivos ciertamente es mas lento que usar un caché en memoria pero es capaz de almacenar una cantidad mucho mayor de datos a un menor costo.

3.1.5.3 Alojamiento en la nube

El alojamiento en la nube sirve como intermediario entre la aplicacion web OH y los monitores remotos TPS, la nube ofrece muchas ventajas con respecto a la escalabilidad y facilidad de implementación del sistema. Usamos el alojamiento en la nube para guardar los archivos de trazas generados por los monitores en las redes remotas y también a modo de "buzón" para enviar mensajes a los monitores remotos.

3.1.6 Diseño de la base de datos

La base de datos de Octopus Head esencialmente modela un conjunto de monitores junto con sus enlaces monitoreados (tentáculos), los resultados obtenidos para cada tentáculo (o monitor) de cada una de sus pruebas, las pruebas y sus parámetros, el

historial de cambios en el plan de monitoreo, entre otros. Ya que Django usa un ORM para manejar la base de datos, desde el punto de vista de la aplicación cada entidad es una clase por lo que es posible aprovechar todas las características de la programación orientada a objetos, como herencia de clases, clases abstractas implementación de métodos específicos a un modelos y sobrecarga de métodos de los padres, a continuación se describen las entidades que se desa modelar:

Cada usuario registrado del sistema está representado por el modelo "User" que viene incluido como parte del framework de Django, este guarda la información mínima necesaria para autenticar al usuario, así como sus grupos y permisos, un usuario autenticado puede estar en uno o mas grupos y puede tener uno o mas permisos, por defecto se tienen tres tipos de usuarios: el usuario normal, el usuario "staff" que puede ingresar la panel de administración del sistema y el superusuario, el superusuario tiene todos los privilegios del sistema, por lo tanto puede manejar otros usuarios (agregando grupos, permisos, cambiando el tipo de otros usuarios, etc) y hacer cambios a cualquier otro modelo del sistema.

Los usuarios finales del sistema (es decir aquellos que no son staff o superuser) tienen un modelo adicional llamado "UserProfile" (perfil de usuario), para implementar esta funcionalidad podría parecer conveniente sencillamente extender el modelo "User" sin embargo esto interfiere con el mecanismo de autenticacion (que no espera que exista otra tabla de usuarios) de modo que implementar una relación uno-a-uno entre el modelo "UserProfile" y el modelo "User" es preferido. El perfil de usuario aloja los detalles adicionales del usuario monitor como credenciales de Dropbox y códigos de confirmación.

El modelo central de la base de datos es el monitor, que representa un Tentacle Probe Source haciendo pruebas en la red remota. Cada usuario posee uno o mas monitores, el monitor está compuesto por un número de enlaces monitoreados y un plan de monitoreo asi como su horario de sincronizacion y sus detalles específicos como zona horaria, posición geográfica, dirección ip entre otros.

Los tentaculos son un concepto fundamental en el enfoque de monitoreo de este trabajo, cada tentáculo (o enlace monitoreado) consiste de un Tentacle Probe Source (el monitor) y un Tentacle Probe Destination (un nodo monitoreado) y cualquier número

de nodos intermedios entre ellos (como enrutadores, proxys, etc), para representar esto usamos el modelo "Link" que guarda el ip del Tentacle Probe Source y otros detalles como posición geográfica, nombre y descripción.

Las pruebas son procedimientos que el usuario puede planificar para que sus monitores ejecuten según una planificación, es a esto a lo que llamamos el 'plan de monitoreo', una prueba consiste en cualquier código python ejecutable, generalmente con el objetivo de obtener algún dato de la red monitoreada, desde el punto de vista de la aplicacion web una prueba se modela como una agregación de parámetros configurables, estos son usados para construir un formulario que permite al usuario editar el plan de monitoreo.

Los resultados de las pruebas también llamados trazas se guardan según las características especificas de cada prueba, no existe ningún limitante a la hora de diseñar modelos para los resultados de las pruebas, sin embargo las pruebas implementadas hasta ahora tienen en común un timestamp (el tiempo de ejecucion de la prueba) y el tentáculo relacionado a la prueba, sin embargo sería posible guardar trazas que no estén relacionadas a ningún tentáculo (por ejemplo, resultados del sondeo de el número de errores en una interfaz del TPS)

Un "MonitorTest" es una modelo intermedio que representa una prueba planificada para un monitor y sus parámetros, en otras palabras el conjunto de "MonitorTest" para un monitor conforman el plan de monitoreo, este modelo aloja el tiempo inicial de ejecucion de una prueba, el tiempo final, intervalo entre pruebas, y su estado (activa o inactiva).

Ya que el usuario tiene la libertad de modificar el plan de ejecucion en cualquier momento dado, se guarda un historial de los cambios hechos a los "MonitorTest" en una tabla a modo de historial, de esta manera es posible reconstruir con que parámetros estaba siendo ejecutada una prueba en cualquier momento especifico, esto no solo con fines informativos para el usuario, sino también puede ser útil para algoritmos de análisis de los datos que necesiten conocer los parámetros de ejecucion en algún momento especifico.

A los distintos métodos de análisis y visualización los datos que permiten al usuario explorar y obtener información relevante a partir de ellos, las llamamos visualizaciones,

computar visualizaciones consiste en retirar los datos seleccionados de la base de datos, pasar estos datos por algún algoritmo de análisis y prepararlos para ser desplegados en alguna forma elegida por el usuario, como mapas de calor, gráficas de barras, mapas, etc.

Las sincronizaciones consisten en recoger los datos dejados por los monitores remotos en la nube y insertarlos a la base de datos, generalmente haciendo algún preprocesamiento o parsing, se guarda un historial de sincronizaciones no sólo como forma de informar al usuario de la cantidad de archivos recolectados y si ocurrieron errores, el mecanismo de sincronizacion depende un cursor que indica al sistema cuales son los archivos nuevos o modificados que deben ser insertados, el mecanismo de sincronizacion se explicará a fondo en la subsección 3.2.3.

Como se vió en la figura 3.3 el planificador depende de la base de datos para determinar el horario de sincronizaciones, por lo que es necesario un modelo para guardar el horario de sincronizaciones, ya sea este por intervalos o a una hora especifica del día.

Los reportes son una forma de mostrar y compartir conjuntos de visualizaciones en una sola vista, de esta manera es posible hacer comparaciones de gráficas de distintos tentáculos, o incluso de distintos monitores y ademas compartirlas con usuarios no autenticados haciendo los resultados públicos.

3.1.6.1 Diagrama de entidad-relación

La figura 3.4 muestra las entidades primordiales de la base de datos: usuarios, monitores, tentáculos y resultados de las pruebas implementadas.

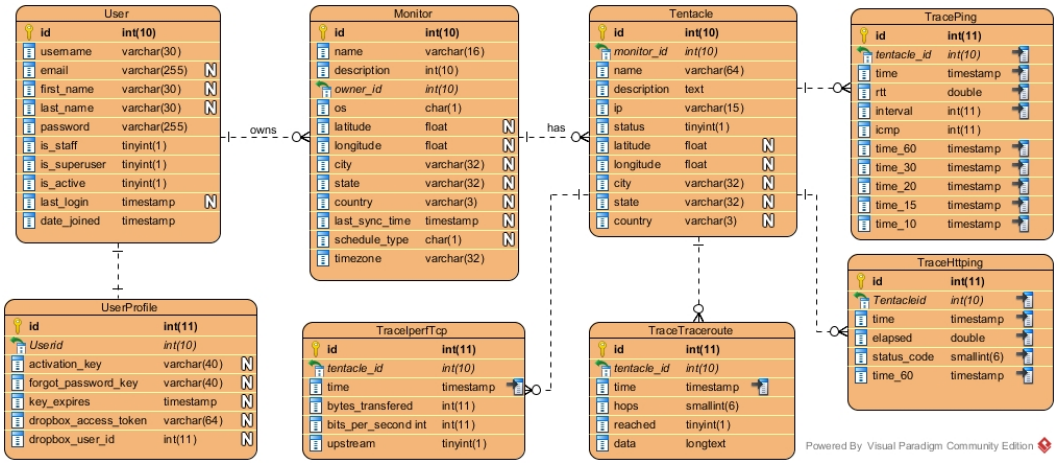

Figura 3.4:

La figura 3.5 muestra las entidades relacionadas al plan de monitoreo, así como su relación con la entidad monitor.

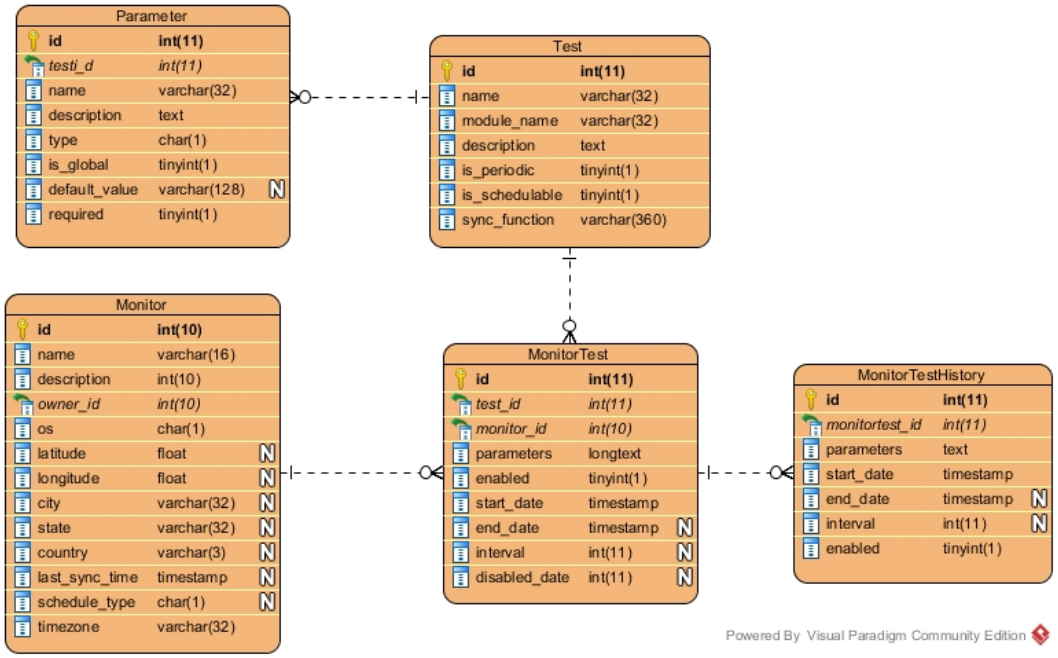

Figura 3.5:

La figura 3.6 muestra las entidades relacionadas a mecanismo de sincronizacion y horario de sincronizaciones del monitor.

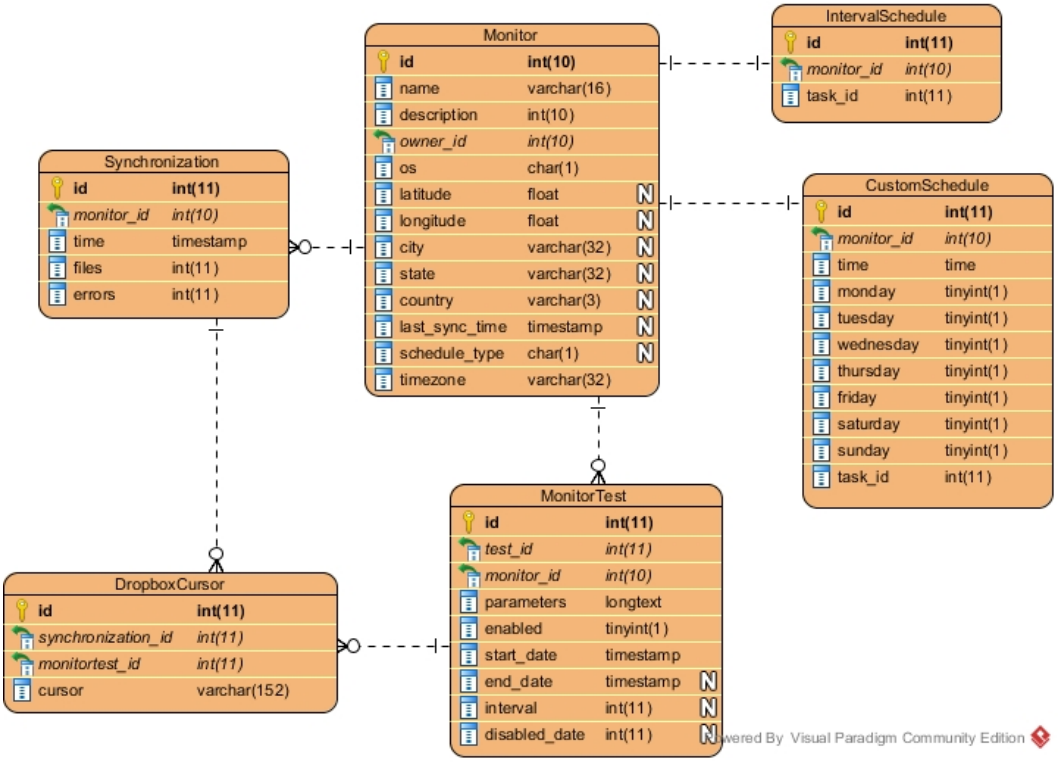

Figura 3.6:

La figura 3.7 muestra las entidades relacionadas a las vistas de resultados y reportes.

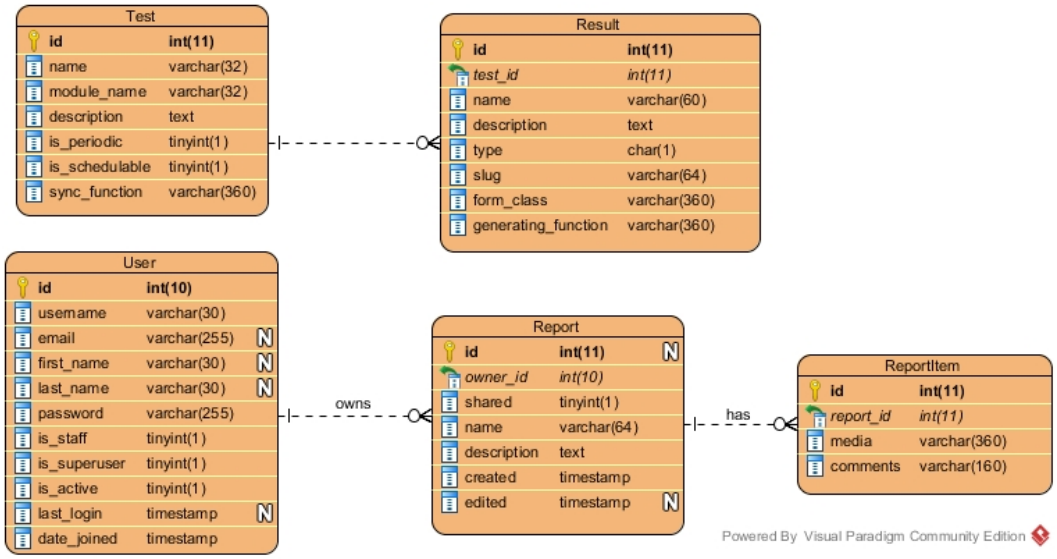

Figura 3.7:

3.1.6.2 Entidades

Las siguientes tablas resumen las entidades y la función de las entidades que forman parte de la aplicacion web.

Tabla 3.3: Entidades relacionadas al manejo de usuarios

Nombre de la entidad	**Función**
User	Almacena credenciales e información básica del usuario.
UserProfile	Almacena credenciales de dropbox y códigos de confirmación del usuario .

Tabla 3.4: Entidades relacionadas a los monitores

Nombre de la entidad	**Función**
Monitor	Almacena detalles del monitor (TPS) como dirección ip, posición geográfica y zona horaria.
Link	Almacena detalles del Tentacle Probe Destination como dirección ip, posición geográfica y si está activo para el monitoreo.
TracePing	Almacena el resultado de una prueba de ping como marca de tiempo, rtt y TPD relacionado.
TraceHttping	Almacena el resultado de una prueba de httping como marca de tiempo, tiempo de respuesta, código de estado y TPD relacionado.
TraceTraceroute	Almacena el resultado de una prueba de traceroute como marca de tiempo, número de saltos, alcanzabilidad y TPD relacionado.
TraceIperf	Almacena el resultado de una prueba de iperf como marca de tiempo, dirección del flujo, rendimiento obtenido en bytes por segundo y TPD relacionado.
Test	Almacena el nombre, descripción y tipo de una prueba.
Parameter	Almacena el nombre, descripción tipo de dato y otros detalles sobre un parámetro relacionado a una prueba.
MonitorTest	Tabla intermedia entre monitor y test que almacena una prueba planificada para un monitor, así como su tiempo inicial, final, su estado y parámetros.
MonitorTestHistory	Almacena los cambios realizados a una prueba planificada así como sus parámetros y el periodo en que estos fueron efectivos.

Tabla 3.5: Entidades relacionadas al planificador

Nombre de la entidad	**Función**
Syncronization	Almacena el resultado de una planificación y algunos detalles como número de archivos recolectados y número de errores.
DropboxCursor	Almacena un cursor obtenido de la última sincronizacion de un monitor.
IntervalSchedule	Almacena el intervalo entre sincronizaciones de un monitor.
CustomSchedule	Almacena la hora de y los días de la semana en que debe sincronizarse un monitor.

Tabla 3.6: Entidades relacionadas a las visualizaciones y reportes

Nombre de la entidad	**Función**
ResultView	Almacena el nombre y descripción de una visualización así como su tipo y slug.
Report	Almacena el nombre y descripción de un reporte, su dueño y tipo (publico o privado).
ResultItem	Almacena el url relacionado a una visualización que es parte de un reporte y comentarios.

3.1.6.3 Relaciones

- **users - userprofiles (1-1)**: Un usuario solo puede tener un perfil de usuario.
- **users - monitors (n-1):** Un usuario puede ser dueño de cero o mas monitores, pero un monitor solo puede tener un dueño.
- **monitors - tentacles (n-1):** Un monitor puede tener cero o mas tentacles, pero un tentacle solo puede estar en un monitor.

- **tentacles - traces (n-1):** Un tentaculo puede tener cero o mas trazas relacionadas, pero una traza solo puede estar relacionada a un tentacle.
- **test - parameter (n-1):** Una prueba puede tener uno o mas parámetros, pero un parámetro solo puede ser parte de una prueba.
- **test - monitor (n-n):** Una prueba puede ser planificada cero o mas veces en cero o mas monitores.
- **monitortest - monitortesthistory (n-1):** Una instancia de monitortest puede tener una o mas entradas en el historial, pero una entrada en el historial solo se refiere a un monitortest.
- **monitor - synchronization(n-1):** Un monitor puede ser sincronizado cero o mas veces pero una sincronizacion solo puede estar relacionada a un monitor.
- **synchronization - dropboxcursor (n-1):** Una sincronizacion puede tener uno o mas cursores, pero un cursor solo pertenece a una sincronizacion.
- **monitor - intervalschedule (1-1):** Un monitor solo puede tener un horario de sincronizacion en intervalo y este solo puede pertenecer a un monitor.
- **monitor - customschedule (1-1):** Un monitor solo puede tener un horario de sincronizacion personalizado y este solo puede pertenecer a un monitor.
- **test - resultview (n-1):** Una prueba puede tener una o mas vistas de resultados, pero una vista de resultados solo puede pertenecer a una prueba.
- **user - report (n-1):** Un usuario puede ser dueño de cero o mas reportes y un reporte solo puede pertenecer a un usuario.
- **report - reportitem (1-1):** Un reporte puede tener uno o mas items y un item solo puede pertenecer a un reporte.

3.1.7 Diseño de Pantallas

Todas las pantallas del sistema fueron creadas bajo el paradigma del diseño web adaptable, es decir que pueden ajustarse a cualquier resolución de pantalla sin reducir la usabilidad o sacrificar la experiencia del usuario, el servidor no tiene que decidir entre un conjunto de plantillas para distintas resoluciones sino que envía al cliente un solo documento HTML y este combina las reglas de presentación de archivos css y la lógica en archivos javascript para desplegar una pagina web adaptada al dispositivo del usuario.

Ya que una aplicacion web está compuesta por decenas de vistas que comparten componentes como barras de navegación, cabeceras, y pies de pagina, Django incluye un micro-lenguaje de plantillas con funcionalidades de herencia e inclusión de plantillas y estructuras de repetición y decisión, esto hace que sea posible tener una taxonomía de plantillas, de manera que solo es necesario escribir los elementos comunes a un conjunto de plantillas una vez y crear nuevas pantallas en el tope de otras. Al proceso de combinar los datos de la base de datos y la petición del usuario, junto con una plantilla predefinida en la aplicacion para generar páginas web personalizadas se le llama 'rendering'.

En la Figura **??** se puede ver como todas las plantillas heredan de la plantilla base, esta incluye la declaración del archivo html, la cabecera y define bloques que pueden ser sobreescritos por las plantillas "hijas", los bloques que define la plantilla base son: **(1) bloque de título** permite extender el titulo de la pagina **(2) bloque de estilo** permite reemplazar los archivos de estilo **(3) bloque de modales** permite introducir código antes del cuerpo del documento como modales **(4) bloque de contenido** bloque principal para el contenido de la pagina **(5) bloque de javascript** bloque para incluir archivos y métodos javascript, los archivos javascript se cargan al final para acelerar la carga de las paginas. La plantilla base es a su vez heredada por la plantilla "dashboard" que incluye el menú de navegación lateral y a su vez esta es heredada por la plantilla "monitor" que agrega el submenú del monitor.

A continuación se muestra el diseño de las principales pantallas de la interfaz de usuario del servicio web.

- **Pantalla Inicial del sistema,** esta es la pantalla principal del servicio web disponible en linea y sirve como punto de inicio donde los usuarios no-autenticados pueden registrarse, iniciar sesión y obtener información básica sobre el sistema.

- **Pantalla de registro de usuario,** en esta pantalla un usuario que desee utilizar el sistema de monitoreo introduce su nombre de usuario, contraseña y correo electrónico para registrarse

- **Pantalla de inicio de sesión,** en esta pantalla un usuario autenticado puede introducir su nombre de usuario y contraseña para iniciar sesión, también se incluyen enlaces para recuperar contraseña o crear una nueva cuenta en caso de no tener una.

- **Pantalla "Home",** esta es la pantalla de bienvenida al sistema para un usuario autenticado, esta muestra algunas estadísticas rápidas del uso del sistema (como número de monitores, tentáculos, pruebas, sincronizaciones ejecutada, etc), así como enlaces a las principales subsecciones de cada monitor. Todas las pantallas a partir de ahora comparten un menú de navegación superior con enlaces al home, y funcionalidades para ver el perfil de usuario y hacer logout y un menú de navegación lateral con enlaces a las principales secciones del sistema

- **Pantalla de selección de monitores,** esta pantalla muestra los monitores del usuario y le permite seleccionar alguno de ellos para dirigirlo a la pantalla del monitor, ademas incluye un botón para crear nuevos monitores.

- Pantalla de creación de monitores, en esta pantalla se introducen del detalles del monitor a crear como nombre, descripción, dirección ip, sistema operativo del dispositivo host y zona horaria del monitor, todas las pantallas

- **Pantalla del monitor,** esta es la pantalla principal del monitor donde se muestran los detalles del monitor, un mapa de la red mostrando la localizacion de los distintos dispositivos de red (TPS y TPS) generado a partir de la información suministrada por el usuario y una lista de verificación para ayudar al usuario a seguir los pasos para monitorear una red. Todas las vistas relacionadas a

un monitor específico despliegan un submenú de navegación para visitar las distintas subsecciones de manejo del monitor como tentáculos, resultados y sincronizaciones.

- **Pantalla de tentáculos,** esta pantalla muestra los detalles de los distintos tentáculos de un monitor, y incluye enlaces para registrar nuevos tentáculos y editar los detalles de los existentes.

- **Pantalla de Plan de monitoreo,** esta pantalla permite ver el plan de monitoreo por tipo de prueba, incluye enlaces para editar pruebas planificadas y para planificar nuevas pruebas ya sean periodicas o de una sola vez. En caso de que hayan muchas pruebas planificadas separa el contenido en páginas numeradas

- **Pantalla de planificación de pruebas,** esta pantalla permite agregar pruebas al plan de monitoreo, si se trata de una prueba de una sola vez permite elegir la hora y fecha de ocurrencia, en el caso de pruebas periódicas permite elegir fecha inicial, fecha final e intervalo entre pruebas, en todos los casos se incluye el formulario para fijar los parámetros de ejecucion de la prueba

- **Pantalla de detalles de una prueba,** esta pantalla muestra los parámetros activos de una prueba planificada en un formulario editable, así como opciones para habilitar o deshabilitar. En el panel lateral se muestra el historial de la prueba, mostrando los periodos en que estuvo inactiva y sus distintos parámetros en periodos de inactividad, así como enlaces para volver a activar parámetros anteriores en cualquier punto de la historia.

- **Pantalla de resultados,** esta pantalla muestra todas las visualizaciones disponibles para un monitor (a partir de las pruebas activas o planificadas en algún momento) y una barra de búsqueda para filtrar los resultados.

- **Pantalla de visualización de resultados,** esta pantalla muestra un formulario para introducir los datos a visualizar y una barra de herramientas, con opciones para re-computar la visualizaciones, obtener enlaces directos para compartir o agregar la visualización a un reporte dado. Esta pantalla muestra un animación

de carga para indicar que una gráfica se está computando o retirando del servidor y mensajes de error en caso de que el servicio no esté disponible o un problema haya aparecido

- **Pantalla de sincronizaciones,** esta pantalla muestra el tiempo desde la última sincronizacion y el tiempo para la próxima así como un botón para forzar una sincronizacion inmediata, también muestra un formulario para definir el horario de sincronizacion y en un panel lateral muestra el historial de sincronizaciones, este ofrece información sobre la cantidad de archivos recolectados y el numero de errores encontrados durante las últimas sincronizaciones

- **Pantalla de reportes,** esta pantalla muestra una tabla de los reportes del usuario con enlaces para ver cada uno de ellos y su última fecha de modificación así como un botón para ir al formulario de creación de reportes

- **Pantalla de visualización y edición reporte,** esta pantalla puede desplegarse de distintas maneras según el usuario que la observe, si se trata del usuario al que pertenece el reporte entonces este verá una barra de herramientas para agregar visualizaciones al reporte, cambiar la privacidad del reporte (publico o privado) o editar y borrar, del mismo modo podrá editar los comentarios de cada una de las visualizaciones o quitarlas del reporte. Un usuario que no sea dueño del reporte solo podrá leer su contenido y observar las visualizaciones

- **Pantalla de cuenta del usuario,** esta pantalla muestra los detalles de la cuenta del usuario, así como enlaces para cambiar la contraseña y renovar el token de acceso a dropbox en caso de que en algún momento este haya sido revocado

- **Pantalla de descargas,** en esta pantalla se muestra una lista de las descargas disponibles, como distintas versiones del montitor de red o publicaciones relacionadas

- **Pantalla de ayuda,** esta pantalla muestra tópicos de ayuda al usuario como conceptos básicas, manual de instalación básico, especificaciones de las pruebas, etc.

3.1.8 Diseño de URLs

TODO

3.2 Componentes

En esta sección se explican los distintos componentes de software que son parte de la aplicacion web y su funcionalidad.

3.2.1 Enlazador de cuentas de Dropbox

Para realizar todas las tareas relacionadas con el almacenamiento remoto en la nube, la aplicacion web hace peticiones a nombre de un usuario de Dropbox, sin embargo primero es necesario realizar un proceso de autenticación que dará permisos a la aplicacion web de realizar estas peticiones. El proceso de autenticacion es un algoritmo a dos pasos que se puede observar a continuación:

La estructura de esta función es similar a muchas otras que manejan datos suministrados por el usuario: si la petición es de tipo POST pasamos a validar el formulario si es la válida entonces hacemos alguna operación (casi siempre actualizar la base de datos) y redireccionamos al usuario a una vista indicando el éxito de la operación, de modo contrario el formulario es inválido y se dibuja la plantilla mostrando los mensajes de error correspondientes, si el método es GET entonces mostramos una formulario en blanco.

En este caso cuando el usuario visita la ruta '/dropbox/link-account/' con el método GET la aplicacion web crea un URL hacia el sitio de Dropbox y muestra al usuario un formulario con una caja para introducir un código y un enlace con el URL generado previamente, cuando el usuario hace click al URL este lo lleva al sitio de Dropbox que a su vez despliega un dialogo preguntando al usuario si dará permiso a la aplicacion de Octopus Monitor para hacer cambios en su carpeta personal, en caso positivo se genera un token que el usuario debe copiar en el formulario, cuando el usuario hace click en submit una peticion HTTP POST se envía al servidor, entonces este valida los datos introducidos y llama al método "finish" con el token, este método retorna el

Algorithm 1 Enlazamiento con Dropbox

```
1: procedure LINK_DROPBOX_ACCOUNT(request)
2:     if request.method = 'POST' then
3:         form ← DropboxCodeForm
4:         if form is valid then
5:             flow ← DropboxOAuth2FlowNoRedirect
6:             access_token , user_id ←flow.finish(form.code)
7:             profile ← get(UserProfile, id = request.user.id)        ▷ Retira un objeto
   UserProfile por su id de la base de datos
8:             profile.dropbox_access_token ← access_token
9:             profile.dropbox_user_id ← user_id
10:            profile.save()                    ▷ Guarda los cambios en la base de datos
       return HttpResponseRedirect("/dropbox/link-account/done")
11:    else
12:        flow ← DropboxOAuth2FlowNoRedirect
13:        url ← flow.start
14:        form ← DropboxCodeForm
       return render(request,"link_dropbox.html",form)
```

código de acceso al Dropbox del usuario para nuestra aplicacion y el identificador del usuario, guardamos los cambios en la base de datos y llevamos al usuario a una página indicando que el proceso de enlazamiento fue exitoso.

A partir de este momento tenemos los dos elementos necesarios para hacer peticiones al API de Dropbox: los permisos sobre la carpeta en la nube del usuario y el token de acceso a su dropbox.

3.2.2 Subidor de mensajes

Una de las tareas fundamentales de la aplicacion web es mantener a los monitores remotos informados de los cambios hechos al plan de monitoreo por parte de los usuarios, para esto usamos un concepto similar al de un buzón de correo electrónico, la aplicacion web sube archivos a modo de mensajes a una carpeta en la nube que hace las veces de buzón, hay dos tipos mensajes: aquellos relacionados a los tentáculos (nuevo tentáculo registrado o cambios a un tentáculo existente) y los relacionados a las pruebas (nueva prueba planificada, cambios a los parámetros, cambios al intervalo entre pruebas, prueba deshabilitada, etc.)

Los mensajes que se suben a la nube se escriben en formato json, este formato ofrece grandes ventajas para el desarrollo de aplicaciones distribuidas ya que es fácil de codificar del lado del servidor y fácil de decodificar del lado del monitor es mucho mas ligero que otros formatos como XML y al mismo tiempo es legible para seres humanos, todos las funciones que suben archivos a la nube funcionan de forma simular (1) retiran de la base de datos el token de acceso a Dropbox del usuario (2) determinan el nombre del archivo a subir (3) codifican el archivo json a enviar (4) crean un objetivo tipo archivo o buffer en memoria (5) suben el archivo a la nube, el procedimiento se puede ver a continuación:

3.2.3 Recolector de datos

El objetivo de la realización de pruebas en las redes a monitorear es recolectar una cantidad de datos suficientes para obtener información del estado de la red, si el monitor tiene un plan de monitoreo definido y está realizando pruebas, este dejará en la nube

Algorithm 2 Subir prueba a Dropbox

```
1: procedure UPLOAD_TEST_FILE(monitor_test,data)
2:     user ← monitor_test.monitor.owner                ▷ Retira el usuario relacionado
3:     profile ← GET(UserProfile, id = user.id)         ▷ Retira el perfil de usuario
4:     client ← DROPBOXCLIENT(profile.dropbox_access_token)
5:     directory ← MONITOR_FOLDER_NAME+monitor_test.monitor.id+'mailbox'
6:     filename ← directory + 'test_' + monitor_test.id + '.json'
7:     buffer ← STRINGIO                                ▷ Crea un objeto tipo archivo en memoria
8:     json ← DUMPS(data)  ▷ Codifica una cadena de caracteres json a partir de un
   objeto python
9:     buffer.WRITE(json)
10:    client.PUT_FILE(filename, buffer, overwrite = True)
       return
```

archivos con resultados de forma constante; es trabajo de la aplicacion web a su vez recolectar estos archivos y introducirlos a la base de datos en un formato tal que permita generar visualizaciones de la forma mas conveniente posible para el usuario, en el marco de este sistema, a este proceso se le da el nombre de "sincronizacion".

El algoritmo general de sincronizacion (Algoritmo 3), retira de la base de datos el monitor a sincronizar, el token de acceso a Dropbox del usuario y las pruebas activas para el monitor, luego para cada prueba llama a una subrutina que descarga los archivos, los procesa y introduce los resultados obtenidos en la base de datos finalmente totaliza el número de archivos recolectados, el número de errores (ya sea errores de parsing de los archivos o problemas de conexión a Dropbox), guarda en la base de datos el registro de la sincronizacion, los cursores (de los cuales se hablará próximamente) y actualiza el tiempo de ultima sincronizacion del monitor.

Debemos recordar que los resultados de las pruebas se guardan en una carpeta por prueba, donde constantemente se están guardando nuevos archivos, por lo tanto, la subrutina de sincronizacion debe determinar cuales son los archivos nuevos desde la ultima sincronizacion, para esto, el API de Dropbox provee el método "delta", que determina los cambios realizados a una carpeta a partir de un momento específico, a continuación puede verse un ejemplo de la respuesta en formato json del obtenida tras

Algorithm 3 Sincronizar monitor

```
1:  procedure SYNCRONIZE(monitor_id)
2:      monitor ← GET(Monitor, id = monitor_id)        ▷ Retira el monitor por su id
3:      profile ← GET(UserProfile, id = monitor.owner.id) ▷ Retira el perfil de usuario
4:      client ← DROPBOXCLIENT(profile.dropbox_access_token)
5:      tests ← Get all active tests for this monitor
6:      cursors ← empty dictionary
7:      files ← 0
8:      errors ← 0
9:      for test in tests do
10:         cursor,_files,_errors ← _SYNCRONIZE(test, monitor, client)
11:         Insert cursor on cursors using test as key
12:         files ← files + _files
13:         errors ← errors + _errors
14:     sync ← SYNCRONIZATION(monitor, files, errores)
15:     sync.SAVE
16:     for key,value in cursors do
17:         cursor ← DROPBOXCURSOR(value, sync, monitor_test)
18:         cursor.SAVE
19:     monitor.last_sync_time ← get current time
20:     monitor.SAVE
        return
```

llamar el método delta para una carpeta:

```
{
        "reset":false,
        "cursor":"AAHskXVmJSRVG_bgh4Oq2VPQqG79nWM26Tl8jSmRyiGM3M0EGL..." ,
        "has_more":false,
        "entries":[
                [
                        "/monitor1/ping/link_2_2015_11_02_15.txt",
                        {
                                "revision":3,
                                "rev":"30ec9923d",
                                "thumb_exists":false,
                                "bytes":0,"modified":
                                "Wed, 20 Mar 2013 05:58:43 +0000",
                                "path":"/proj1",
                                "is_dir":true,
                                "icon":"folder_app",
                                "root":"app_folder",
                                "size":"0 bytes"
                        }
                ],
        ]
}
```

Como puede observarse es un diccionario con cuatro entradas:

- reset: determina si se debe limpiar el estado local antes de procesar las entradas delta, es verdadero solo durante la primera llamada a delta o en raras ocasiones.
- cursor: el cursor es una cadena de caracteres de longitud 64, y representa el estado de la carpeta en un momento dado, se puede usar en llamadas sucesivas de "delta" para obtener los cambios a partir de ese momento.
- has_more: determina si hay mas entradas delta en la carpeta, en tal caso es

necesario volver a llamar el método delta inmediatamente de nuevo, esto solo es necesario cuando hay cantidades considerables de archivos en una carpeta

- entries: es una lista de "entradas delta" cada entrada representa un archivo o carpeta que cambió desde la ultima llamada a delta, cada entrada es un par <path,metadata> donde "path" es el nombre del archivo y "metadata" es un diccionario conteniendo algunos datos relevantes al archivo, si "metadata" es "null" indica que el archivo fue eliminado.

El algoritmo de sincronización entonces depende del cursor para determinar los cambios desde la última sincronización, llamar al método delta y obtener la lista de "entradas delta" luego recorre aquellas entradas delta cuya metada no es nula (no fueron borrados) y le pasa el contenido el archivo a un algoritmo de parsing, que lo lee y inserta su contenido en el formato apropiado en la base de datos, debemos recordar que cada archivo de traza puede tener un formato distinto dependiendo de a que tipo de prueba pertenezca, por lo que también es trabajo de del algoritmo de sincronización elegir el método correcto al cual pasará el archivo.

3.2.4 Computo de visualizaciones

El computo de las visualizaciones tiene tres partes que son comunes a todas las visualizaciones, primero, los datos necesarios son retirados de la base de datos, después los datos obtenidos son transformados de alguna manera, por ejemplo, los timestamps se transforman en horas localizadas según la zona horaria del monitor, o se retiran valores atípicos de la muestra y finalmente estos datos se combinan con una plantilla para generar una página HTML.

Muchas de las visualizaciones obtenidas en el sistema se pueden generar trivialmente pasando a Highcharts un arreglo de pares <x,y> por ejemplo rtt vs tiempo, sin embargo otras visualizaciones requieren de algoritmos de análisis que hagan algún pre-procesamiento de los datos o saquen algún tipo de conclusión para mostrar al usuario, a continuación se explica el calculo de las visualizaciones no-triviales del sistema.

3.2.4.1 Cálculo de Mapas de Calor

En este trabajo usamos mapas de calor para representar el valor del RTT tiempo de respuesta promedio en relación a la fecha y la hora del día, los valores "altos" están representados por colores cálidos y los valores "bajos" están representados por colores fríos; cabe destacar que un valor de "alto" para un red de fibra óptica puede ser "bajo" para una red satelital, de modo que estos valores dependen unicamente de los datos en la muestra.

Los mapas de calor permiten evaluar rápidamente la fluctuación del RTT durante un día especifico o notar patrones a largo plazo, otra ventaja de los mapas de calor es que es posible observar periodos de inactividad de la red como "espacios en blanco" en la gráfica ya que para estos periodos no hay datos.

Los parámetros de los mapas de calor son los siguientes:

- **Enlace:** el enlace o tentáculo seleccionado
- **Intervalo de agrupación:** la granularidad del mapa de calor, a mayor intervalo de agrupación mas muestras se van a promediar por intervalo, el valor máximo es 60 minutos, lo cual resultará en un mapa de calor con 24 puntos por día, el menor es 10 minutos, lo cual resultará en un mapa de calor con 240 puntos por día
- **Rango de tiempo:** ajusta el periodo de tiempo seleccionado para mostrar, se puede seleccionar por meses, años o "todo el tiempo".

Para calcular el mapa de calor debemos primero filtrar las muestras por enlace y periodo de tiempo, luego, estas se deben agrupar según el intervalo de agrupación y sacar el promedio de cada grupo, hay muchas formas de realizar esta operación sin embargo cuando se trata de grandes conjuntos de datos esta puede ser una operación costosa por lo que preferimos dejar todos los filtros, ordenamientos, agrupaciones, cálculo de promedios, sumas, etc a la base de datos, si la tabla está apropiadamente indexada esta puede realizar dichos cálculos ordenes de magnitud mas rápido que retirando todos los datos y haciendolos directamente en la aplicacion.

timezone	time	time_60	time_10
-4:30	2015-11-17 22:29:42	2015-11-17 17:00:00	2015-11-17 17:50:00
UTC	2015-11-17 22:29:42	2015-11-17 22:00:00	2015-11-17 22:20:00

Tabla 3.7: Ejemplo de los valores de time_60 y time_10 para la misma traza en UTC(+0:00) y Caracas (-4:30)

Como se vio previamente la tabla "TracePing" tiene los campos "time_60", "time_30", "time_15", "time_20" y "time_10" estos campos tienen un valor pre-calculado de tiempo que aloja el periodo de agrupación al que pertenece la muestra, estas columnas facilitan al SMBD a agrupar las muestras en periodos de 60, 30, 20, 15 o 10 minutos respectivamente, mas aún, son guardados en la base de datos tomando en cuenta la zona horaria del monitor, de modo que no es necesario volver a localizar las fechas cuando se está generando el mapa de calor; un ejemplo de esto puede ser visto en la tabla 3.2.4.1.

Para acelerar aún mas el computo, evitamos usar el ORM para hacer consultas a la base de datos, esto se debe a que, a pesar de que el ORM en el fondo haga las mismas consultas, este desperdicia tiempo y recursos en convertir estos datos crudos en los objetos correspondientes según el modelo de la base de datos, por lo tanto, es preferible construir nuestras propias consultas SQL.

La consulta realizada para retirar los valores de la base se puede ver a continuacion:

```
1 SELECT time_n,AVG(rtt)
2 FROM traceping
3 WHERE link_id=x AND time >= s AND time <= f AND rtt>0
4 GROUP BY time_n
```

Donde:

- n: es el periodo de agrupación
- x: es el id del enlace
- s: es la fecha inicial
- f: es la fecha final

Esta consulta retorna entonces el promedio del valor del RTT de las trazas agrupadas según su intervalo de agrupamiento, ya teniendo estos datos es trivial

desplegar el mapa de calor usando highcharts, sencillamente se prepara un arreglo de tuplas de tipo <fecha,hora,rtt promedio> donde fecha es el día mes y año del grupo, hora es un flotante entre 0 y 24 que resulta de sumar la hora (0-23) al minuto dividido entre 60.

3.2.4.2 Cálculo de horas activas

En el marco de este trabajo llamaremos a una hora "activa" cuando se pudieron recoger al menos el 50% de las muestras durante esa hora, análogamente una hora "inactiva" es aquella donde no se pudieron recoger el 50% de las muestras, este concepto es útil para determinar la disponibilidad y calidad de un servicio, si tenemos una prueba tal que se esté recogiendo una muestra cada 5 segundos, entonces la hora se consideraría activa si se obtienen al menos 720*0.5 muestras durante esa hora.

Al igual que el algoritmo anterior, debemos agrupar las trazas y contar el número de trazas por hora, afortunadamente podemos aprovechar la columna "time_60" para hacer este agrupamiento, luego debemos comparar con el número de trazas esperado para esa hora específica, determinar el número de trazas esperado puede hacerse de varias maneras, la primera de ellas es buscar en el plan de monitoreo los parámetros que estuvieron activos para esa hora, sin embargo la tabla TracePing tiene un campo 'interval' que podemos usar para este propósito.

Sabiendo esto, la consulta a la base de datos queda así:

```
1 SELECT time_60, COUNT(*), AVG(`interval`)
2 FROM traceping
3 WHERE link_id=x AND time_60 >= s AND time_60 < f AND rtt>0
4 GROUP BY time_60
```

Donde:

- x: es el id del enlace
- s: es la fecha inicial
- f: es la fecha final

La consulta anterior retorna: un timestamp que representa la hora de agrupación, el número de muestras recogidas esa hora, y el intervalo entre pruebas promedio; calculamos el promedio ya que el usuario puede cambiar cualquier parámetro en

cualquier momento, de modo que una misma hora puede tener muestras tomadas con distinto tiempo entre pruebas, si calculamos el promedio obtendremos un valor que es al menos "suficientemente bueno" para determinar la cantidad de muestras esperada.

El algoritmo completo de cálculo de horas activas se puede ver a continuación:

Algorithm 4 Calculo de horas activas

```
1:  procedure ACTIVE_HOURS(parameters)
2:      link ← GET(Link, id = parameters.link_id)        ▷ Retira el enlace por su id
3:      start_date ← parameters.start_date
4:      end_date ← parameters.end_date
5:      array ← get data from sql query        ▷ Ejecuta la consulta a la base de datos
6:      days ← (start_date − end_date).DAYS                ▷ Número de dias total
7:      total_hours ← days ∗ 24
8:      hours ← array[24]                  ▷ Arreglo tamaño 24 inicializado en 0
9:      if array lenght > 0 then
10:         j ← 0
11:         for i in RANGE(0,total_hours) do
12:             current_hour ← start_date + i              ▷ Suma a la hora inicial i horas
13:             if array[j][0] = current_hour then
14:                 if j= length of  array then BREAK
15:                 if array[j][1] ⩾ 1800/array[j][2] then ▷ Condicion para que la hora
    sea activa
16:                     hours[i%24] ← hours[i%24] + 1      ▷ le suma a las horas activas
    para esa hora
17:             j ← j + 1
        return hours
```

El algoritmo totaliza el número de horas activas por hora del día, es decir que si se elige un periodo de n días el algoritmo retorna un arreglo de tamaño 24, donde cada posición del arreglo tendrá un entero entre 0 y n. El algoritmo recorre el arreglo retornado por la consulta SQL verificando la condición para que la hora sea activa, si es activa, suma uno al total para esa hora del día, luego es trivial calcular el porcentaje de horas activas por hora del día dividiendo entre el número total de días.

3.2.4.3 Cálculo de periodos de actividad continúa

Como se dijo en la Subsección 2.3.7 llamaremos "actividad continua" a un evento durante el cual un servicio está continuamente disponible para un cliente, en este trabajo un servicio está disponible en un instante de tiempo cuando se ha capturado exitosamente una muestra durante ese instante, en caso contrario se guarda un valor de rtt de "-1" indicando que el servicio no estaba disponible, las pruebas que usamos para determinar periodos de actividad continua son las de ping y httping ya que ambas realizan un trabajo similar, pero usando protocolos distintos.

El algoritmo de cálculo de periodos de actividad continúa recorre las trazas para el intervalo de tiempo seleccionado extendiendo periodos de actividad cuando encuentra trazas **consecutivas** para las cuales el valor de RTT es positivo (hubo respuesta del TPS y por lo tanto el servicio está disponible) y de forma análoga extiende periodos de inactividad continua cuando encuentra trazas **consecutivas** para las cuales el valor de RTT es -1 (no hubo respuesta). La consulta a la base de datos es muy sencilla ya que solo hace falta consultar el rtt y marca de tiempo de las trazas ordenadas por tiempo:

```
1 SELECT time,rtt
2 FROM traceping
3 WHERE link_id=x AND time >= s AND time <= f
4 ORDER BY time
```

Donde:

- x: es el id del enlace
- s: es la fecha inicial
- f: es la fecha final

El algoritmo comienza ejecutando la consulta a la base y verifica que se tenga al menos una muestra, en caso de no tener muestras, a falta de información el periodo se considera totalmente inactivo. En caso contrario, se revisa la primera traza, si es positiva entonces se crea un periodo activo y se marca el inicio del periodo como el tiempo inicial, luego se pasa a revisar la siguiente traza hasta que se encuentre una traza no positiva y se marca como el tiempo final del periodo activo al tiempo de la traza negativa, este proceso se repite análogamente para el periodo inactivo así hasta

llegar a la última traza, cada vez que se determina un periodo este se añade a una lista correspondiente. Todo el proceso se puede observar a detalle en el Algoritmo 5.

3.2.4.4 Cálculo de días activos

A diferencia de las horas activas, definimos a un día como "activo" cuando podemos recoger un periodo de actividad continua de al menos una hora durante ese día, el algoritmo para determinar días activos es en realidad muy similar al de cálculo de periodos de actividad continua (Algoritmo 5), con la excepción de que este funciona separando las trazas por días, y luego ejecutando el algoritmo de termina cuando encuentra un periodo de al menos una hora de actividad continua para un día especifico.

Ya que el algoritmo de días activos es usado generalmente para dibujar un gráfico de barras indicando el porcentaje de días activos, también es necesario determinar la cantidad de días totales en el periodo seleccionado por el usuario, luego cuando se tenga el total de días activos por día de la semana se puede dividir entre los días totales, un ejemplo del resultado deseado como salida del algoritmo de días activos puede ser visto en la Tabla 3.2.4.4

	L	M	W	J	V	S	D
Activos	4	5	3	4	4	3	4
Porcentaje	100	100	60	100	100	75	100
Totales	4	5	5	4	4	4	4

Tabla 3.8: Tabla de días activos para el mes de septiembre del servidor de RESIDE

Para facilitar la comprensión del algoritmo este se va a separar en tres subrutinas, el Algoritmo 6 corresponde al cálculo de días totales en el periodo, el Algoritmo 7 corresponde a la separación de trazas por día y el Algoritmo 8 corresponde al computo de días activos.

Algorithm 5 Calculo de periodos de actividad continua

```
1: procedure CONTINUOUS_ACTIVITY(parameters)
2:     link ← GET(Link, id = parameters.link_id)        ▷ Retira el enlace por su id
3:     start_date ← parameters.start_date
4:     end_date ← parameters.end_date
5:     traces ← get data from sql query        ▷ Ejecuta la consulta a la base de datos
6:     num_traces ← length of  traces
7:     active_periods ← new array
8:     inactive_periods ← new array
9:     if num_traces = 0 then
10:        append to inactive_periodos (start_date, end_date)
        return active_periods, inactive_periods
11:    period_start ← start_date
12:    if traces[0][1] > 0 then
13:        period_active ← True
14:    else
15:        period_active ← False
16:    for i in RANGE(1,num_traces-1) do
17:        if period_active and traces[i][1] <0 then
18:            append to active_periods (duration, period_start, traces[i][0])
19:            period_active ← False
20:            period_start ← traces[i][0]
21:        else if !period_active and traces[i][1] >0 then
22:            append to inactive_periods (duration, period_start, traces[i][0])
23:            period_active ← True
24:            period_start ← traces[i][0]
25:    if period_active then
26:        append to active_periods (duration, period_start, end_time)
27:    else
28:        append to inactive_periods (duration, period_start, end_time)
        return active_periods, inactive_periods
```

Algorithm 6 Calculo de días totales

```
1: procedure TOTAL_DAYS(start_date,end_date)
2:     number_of_days ← (end_date − start_date).DAYS
3:     total_days ← array[7]                    ▷ Arreglo de tamaño 7 inicializado en 0
4:     for i in RANGE(0,number_of_days) do
5:         current_day ← start_date + 1         ▷ Suma a la fecha inicial un día
6:         weekday ← current_day.WEEKDAY        ▷ Toma el día de la semana de la fecha
7:         total_days[weekday] ← total_days[weekday] + 1
       return total_days
```

Algorithm 7 Separación de trazas por día

```
1: procedure SPLIT_TRACES(traces)
2:     days ← empty dictionary
3:     for trace in traces do
4:         ordinal ← trace[0].TOORDINAL    ▷ Obtiene el ordinal proléptico gregoriano
   de la fecha
5:         if ordinal in days then
6:             append trace todays[ordinal]
7:         else
8:             days[ordinal] ← empty list
9:             append trace todays[ordinal]
       return days
```

Algorithm 8 Cálculo de días activos

```
1:  procedure ACTIVE_DAYS(parameters)
2:      link ← GET(Link, id = parameters.link_id)
3:      start_date ← parameters.start_date
4:      end_date ← parameters.end_date
5:      total_days ← TOTAL_DAYS(start_date, end_date)
6:      active_days ← array[7]                    ▷ Arreglo tamaño 7 inicializado en 0
7:      traces ← get data from sql query          ▷ ejecuta la consulta a la base de datos
8:      days ← SPLIT_TRACES(traces)
9:      for key in days.KEYS do                   ▷ itera el diccionario por claves
10:         active_period ← 0
11:         traces ← days[key]
12:         length ← length of traces
13:         for i in RANGE(1,length) do
14:             if traces[i][1]¿0 then
15:                 delta_seconds ← traces[i][0] − traces[i − 1][0]
16:                 active_period ← active_period + delta_seconds
17:                 if active_period > 3600 then          ▷ El día es activo
18:                     weekday ← traces[i][0].WEEKDAY
19:                     active_days[weekday] ← active_days[weekday] + 1
20:                     break
21:             else
22:                 active_period ← 0
        return total_days,active_days
```

3.3 Casos de Uso

En esta sección se explica el flujo de trabajo y casos de uso del sistema desde el punto de vista de los distintos actores del sistema: usuario no autenticado, usuario y administrador.

3.3.1 Usuario no autenticado

Tabla 3.9: Caso de uso – Ver Página Principal

UN-01	*Ver página principal*	
Descripción	El usuario desea visitar la página principal del sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario escribe el URL del sitio y presional enter.
	2	El servidor retorna la página principal.
Excepciones	**Paso**	**Acción**
	3	Si el servicio no está disponible el usuario ve un mensaje de error 503.
Postcondicion	Pantalla principal del sistema.	

Tabla 3.10: Caso de uso – Inicio de sesión

UN-02	*Iniciar sesión*	
Descripción	El usuario desea ingresar al sistema.	
Precondicion	Pantalla principal del sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el enlace de login.
	2	El servidor muestra el formulario de login.
	3	El usuario ingresa sus credenciales.
	4	El usuario hace click en el botón login.
Excepciones	**Paso**	**Acción**
	3	Si las credenciales son incorrectas el servidor informa del error.
Postcondicion	Usuario autenticado y Pantalla "Home".	

Tabla 3.11: Caso de uso – Registro de usuario

UN-03	*Registro de usuario*	
Descripción	El usuario desea registrarse para ingresar al sistema.	
Precondicion	pantalla principal del sistema.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el enlace de registro.
	2	El servidor muestra el formulario de registro.
	3	El usuario ingresa su nombre de usuario
	4	El usuario ingresa su contraseña
	5	El usuario ingresa su correo electrónico
	6	El usuario hace click en el botón submit
	7	El servidor envía un correo electrónico de confirmación
Excepciones	**Paso**	**Acción**
	7	Si el nombre de usuario o correo electronico son incorrectos el servidor informa del error.
	8	Si las contraseñas no coinciden el servidor informa del error.
Postcondicion	usuario registrado y correo enviado.	

Tabla 3.12: Caso de uso – Confirmación de usuario

UN-04	*Confirmación de usuario*	
Descripción	El usuario desea confirmar su cuenta.	
Precondicion	Correo de confirmación enviado (UN-03).	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el enlace de confirmación
	2	El servidor activa la cuenta del usuario
Excepciones	**Paso**	**Acción**
	3	Si el código de confirmación está vencido la activación falla.
Postcondicion	Cuenta de usuario activa y Pantalla de Login.	

Tabla 3.13: Caso de uso – Recuperar contraseña

UN-05	*Recuperar contraseña*
Descripción	El usuario ha olvidado su contraseña y desea recuperarla.
Precondicion	Pantalla de login.
Secuencia normal	**Paso** / **Acción** 1 / El usuario hace click en "forgot password" 2 / El usuario ingresa su correo electrónico 3 / El servidor envía un correo electrónico de recuperación de contraseña 4 / El usuario hace click en el enlace de recuperación 5 / El usuario ingresa una nueva contraseña
Excepciones	**Paso** / **Acción** 6 / Si las contraseñas no coinciden el servidor informa del error.
Postcondicion	Contraseña restablecida.

3.3.2 Usuario

Tabla 3.14: Caso de uso – Cambiar contraseña

UA-01	*Cambiar contraseña*	
Descripción	El usuario desea cambiar su contraseña.	
Precondicion	Pantalla de cuenta de usuario.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en "cambiar contraseña"
	2	El usuario ingresa su nueva contraseña
	3	El usuario confirma su nueva contraseña
	4	El servidor actualiza la contraseña
Excepciones	**Paso**	**Acción**
	6	Si las contraseñas no coinciden el servidor informa del error.
Postcondicion	Contraseña actualizada.	

Tabla 3.15: Caso de uso – Vincular cuenta de Dropbox

UA-02	*Vincular cuenta de Dropbox*
Descripción	El usuario desea vincular su cuenta de Dropbox con Octopus Monitor.
Precondicion	Pantalla de cuenta de usuario.
Secuencia normal	**Paso** / **Acción** 1 / El usuario hace click en "vincular cuenta de Dropbox" 2 / El servidor retorna un enlace a Dropbox y un formulario 3 / El usuario hace click en el enlace 4 / El usuario ingresa a Dropbox 5 / El usuario da permisos a la aplicacion de Octopus Monitor 6 / Dropbox muestra un código al usuario 7 / El usuario ingresa el código en el formulario 8 / El servidor envía el codigo a Dropbox 9 / Dropbox retorna el código de acceso del usuario 10 / El servidor actualiza el perfil del usuario
Excepciones	**Paso** / **Acción** 5 / Si el usuario no da permisos a la aplicacion el flujo termina.
Postcondicion	Cuenta de Dropbox vinculada.

Tabla 3.16: Caso de uso – Ver monitores

UA-03	*Ver monitores*	
Descripción	El usuario desea ver la lista de sus monitores.	
Precondicion	Pantalla "Home".	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en "monitores" en el menú lateral
	2	El servidor retorna la lista de los monitores
Postcondicion	Pantalla de monitores.	

Tabla 3.17: Caso de uso – Crear monitor

UA-04	*Crear monitor*	
Descripción	El usuario desea crear un monitor.	
Precondicion	Pantalla de monitores.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el boton de agregar monitor
	2	El servidor envía el formulario de datos básicos
	3	El usuario ingresa los datos básicos del monitor
	3	El usuario hace click en "next"
	4	El servidor envía el formulario de localización
	5	El usuario ingresa los datos de localización
	6	El usuario hace click en "next"
Excepciones	**Paso**	**Acción**
	2	Si los datos no son validos el servidor informa del error.
Postcondicion	Pantalla de instrucciones de instalación del monitor.	

Tabla 3.18: Caso de uso – Ver monitor

UA-05	*Ver monitor*
Descripción	El usuario desea ver los detalles de un monitor.
Precondicion	Pantalla de monitores.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el enlace "view" de un monitor
Postcondicion	Pantalla del monitor.

Tabla 3.19: Caso de uso – Editar monitor

UA-05	*Editar monitor*
Descripción	El usuario desea editar los detalles de un monitor.
Precondicion	Pantalla del monitor.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el boton "edit" del monitor 2 El servidor envía el formulario de edición 3 El usuario ingresa los datos 4 El usuario hace click en el botón "done"
Excepciones	**Paso** **Acción** 3 Si los datos no son validos el servidor informa del error.
Postcondicion	Pantalla del monitor.

Tabla 3.20: Caso de uso – Eliminar monitor

UA-06	*Eliminar monitor*	
Descripción	El usuario desea eliminar un monitor.	
Precondicion	Pantalla del monitor.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el boton ”delete” del monitor
	2	El servidor envía un dialogo de confirmación
	3	El usuario confirma que desea continuar
	4	El servidor elimina el monitor y todos las entradas relacionadas en el base de datos
Excepciones	**Paso**	**Acción**
	3	Si el usuario no confirma el flujo termina.
Postcondicion	Pantalla de monitores.	

Tabla 3.21: Caso de uso – Ver Tentáculos

UA-07	*Ver Tentáculos*	
Descripción	El usuario desea ver la lista de tentáculos para un monitor.	
Precondicion	Pantalla del monitor.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el enlace ”tentaculos” del sub-menu lateral del monitor
Postcondicion	Pantalla de Tentáculos.	

Tabla 3.22: Caso de uso – Registrar tentáculo

UA-08	*Registrar tentáculo*	
Descripción	El usuario desea registrar un tentáculo.	
Precondicion	Pantalla de tentáculos.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el botón "register tentacle"
	2	El servidor envía el formulario de registro de tentáculo
	3	El usuario ingresa los datos del tentáculo
	4	El usuario hace click en "submit"
Excepciones	**Paso**	**Acción**
	3	Si los datos no son válidos el servidor informa del error.
Postcondicion	Pantalla de tentáculos.	

Tabla 3.23: Caso de uso – Editar tentáculo

UA-09	*Editar tentáculo*	
Descripción	El usuario desea editar un tentáculo.	
Precondicion	Pantalla de tentáculos.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el ícono de lapiz del tentáculo en la lista.
	2	El servidor envía el formulario de edición de tentáculo
	3	El usuario ingresa los datos del tentáculo
	4	El usuario hace click en "submit"
Excepciones	**Paso**	**Acción**
	3	Si los datos no son válidos el servidor informa del error.
Postcondicion	Pantalla de tentáculos.	

Tabla 3.24: Caso de uso – Editar localización del tentáculo

UA-10	*Editar tentáculo*	
Descripción	El usuario desea editar la localización un tentáculo.	
Precondicion	Pantalla de tentáculos.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el ícono del marcador de mapa del tentáculo en la lista.
	2	El servidor envía el formulario de localizacion de tentáculo
	3	El usuario ingresa los datos de localización tentáculo
	4	El usuario hace click en "submit"
Excepciones	**Paso**	**Acción**
	3	Si los datos no son válidos el servidor informa del error.
Postcondicion	Pantalla de tentáculos.	

Tabla 3.25: Caso de uso – Ver plan de monitoreo

UA-11	*Ver plan de monitoreo*	
Descripción	El usuario desea ver el plan de plan de monitoreo.	
Precondicion	Pantalla del monitor.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en "tests" en el menú lateral de navegación.
	2	El servidor envía un menú para seleccionar el tipo de prueba a observar
	3	El usuario selecciona un tipo de prueba y hace click en "view test plan"
Postcondicion	Pantalla de Plan de Monitoreo para una prueba.	

Tabla 3.26: Caso de uso – Planificar prueba periódica

UA-12	*Planificar prueba periódica*
Descripción	El usuario desea planificar una prueba de tipo periódico.
Precondicion	Pantalla de plan de Monitoreo.
Secuencia normal	**Paso** **Acción** 1 El usuario hace click en el botón "schedule periodic tests". 2 El servidor envía una lista de pruebas disponibles para el monitor 3 El usuario selecciona una prueba de la lista. 4 El servidor envía el formulario de planificacion y configuracion de prueba 5 El usuario ingresa la fecha inicial, final y el intervalo entre pruebas 6 El usuario ingresa los parámetros de la prueba 7 El usuario hace click en el boton "submit" 8 El servidor guarda la prueba y sube un mensaje al buzón del monitor
Excepciones	**Paso** **Acción** 5 Si la fechas son inválidas o el intervalo no es un número el servidor informa del error. 6 Si alguno de los parámetros es invalido el servidor informa del error
Postcondicion	Pantalla de detalles de la prueba.

Tabla 3.27: Caso de uso – Planificar prueba periódica

UA-13	*Planificar prueba de una sola vez*	
Descripción	El usuario desea planificar una prueba de tipo periódico.	
Precondicion	Pantalla de plan de Monitoreo.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en el botón "schedule periodic tests".
	2	El servidor envía una lista de pruebas disponibles para el monitor
	3	El usuario selecciona una prueba de la lista.
	4	El servidor envía el formulario de planificacion y configuracion de prueba
	5	El usuario ingresa la fecha inicial, final y el intervalo entre pruebas
	6	El usuario ingresa los parámetros de la prueba
	7	El usuario hace click en el boton "submit"
	8	El servidor guarda la prueba y sube un mensaje al buzón del monitor
Excepciones	**Paso**	**Acción**
	5	Si los datos no son válidos el servidor informa del error.
Postcondicion	Pantalla de detalles de la prueba.	

ver pagina de ayuda

¿enlaces

¿pruebas agregar pruebas configurar prueba ver historial de cambios de la prueba habilitar/deshabilitar pruebas ¿visualizaciones ver lista de visualizaciones ver visualizacion generar visualizacion regenerar visualizacion obtener enlace directo ver en nueva pestaña ¿sincronizaciones ver ultimas sincronizaciones solicitar sincronizacion configurar planificacion de sincronizaciones ¿cuenta cambiar password dar acceso a dropbox renovar acceso a dropbox ¿ reportes crear reporte editar reporte hacer reporte publico/privado agregar grafica a reporte editar grafico en reporte eliminar grafica de reporte eliminar reporte —— admin ———- crear prueba editar prueba crear parametro de prueba editar parametro de prueba eliminar parametro de prueba crear visualizacion editar visualizacion se deberian incluir meta-casos de uso o solo los que son parte de la interfaz web en si???? actualizar base de datos reiniciar servidor reiniciar celery cluster

3.4 Pruebas de Rendimiento

Capítulo 4

Monitor de Red "Tentacle Monitor"

Tentacle Monitor es un monitor de red ligero, diseñado para ajustarse a las limitaciones de sistemas de bajo costo y con la intensión de ser desplegado en las redes que se desea monitorear y ser dejado desatendido durante periodos arbitrarios de tiempo.

Tentacle Monitor ejecuta pruebas periódicas siguiendo un plan de monitoreo definido por el usuario con el objetivo recoger datos sobre las métricas relevantes a la red a monitorear, los resultados de las pruebas son guardados temporalmente en la memoria del dispositivo y se suben a la nube regularmente, la frecuencia en que se suben los datos a la nube depende de la cantidad de memoria disponible en el dispositivo, la cantidad de datos generados por el monitoreo y el ancho de banda disponible.

Uno de los objetivos principales de Tentacle Monitor es ejecutar las pruebas en el momento preciso dado por el plan de monitoreo, ya que el posterior análisis de los datos por parte de Octopus Head exige que las muestras se tomen con un patrón regular y conocido, para esto, un planificador que asegura la ejecución precisa de las pruebas es central en la arquitectura del monitor. Como ya se mencionó, el comportamiento del monitor viene dado por un plan de monitoreo, este es definido por el usuario en Octopus Head, toda la comunicación entre Tentacle Monitor y Octopus Head se realiza a través de la nube, Octopus Head transfiere el plan de monitoreo a través de archivos a modo de mensajes y los guarda en una carpeta a modo de buzón; Tentacle Monitor

mantiene una conexión con la nube para ser notificado de cambios en dicha carpeta con baja latencia de esta manera se pueden leer los mensajes rápidamente y tomar las acciones necesarias, como incluir nuevas pruebas, replanificar pruebas, cambiar los parámetros de ejecución, o incluir nuevos enlaces a monitorear.

4.1 Diseño del Monitor

El diseño del monitor esta sujeto a los siguientes baremos:

- **Estabilidad** ya que el monitor podría dejarse desatendido es esencial que sea estable, si el monitor no se está ejecutando el usuario no obtendrá los resultados esperados en la aplicación web.
- **Flexibilidad** el monitor debe ser capaz de cargar nuevas pruebas en tiempo de ejecución, esto es especialmente importante ya que no desea interrumpir otras pruebas que se estén ejecutando.
- **Planificación precisa** es importante que las pruebas se ejecuten en el momento adecuado, siguiendo de manera fiel el plan de monitoreo.
- **Ejecutable en equipos de bajo costo** el monitor debe incluir el código mínimo para su funcionamiento, tener un uso eficiente de memoria y evitar desbordar la memoria secundaria del host.
- **Configuración remota** el monitor debe incluir algún protocolo para actualizar su plan de monitoreo a partir de cambios hechos en la aplicación web.
- **Bitácora** ya que el monitor se ejecuta como un proceso daemon, se desea tener una bitácora donde se puedan leer mensajes sobre los eventos relevantes durante la ejecución.

4.1.1 Arquitectura

Como se puede observar en la figura 4.1, Tentacle Monitor tiene una arquitectura basada en componentes altamente desacoplados con responsabilidades bien delimitadas

para facilitar el desarrollo de la aplicación; cada uno de estos componentes puede ser reemplazado fácilmente siempre y cuando la interfaz entre ellos se mantenga intacta.

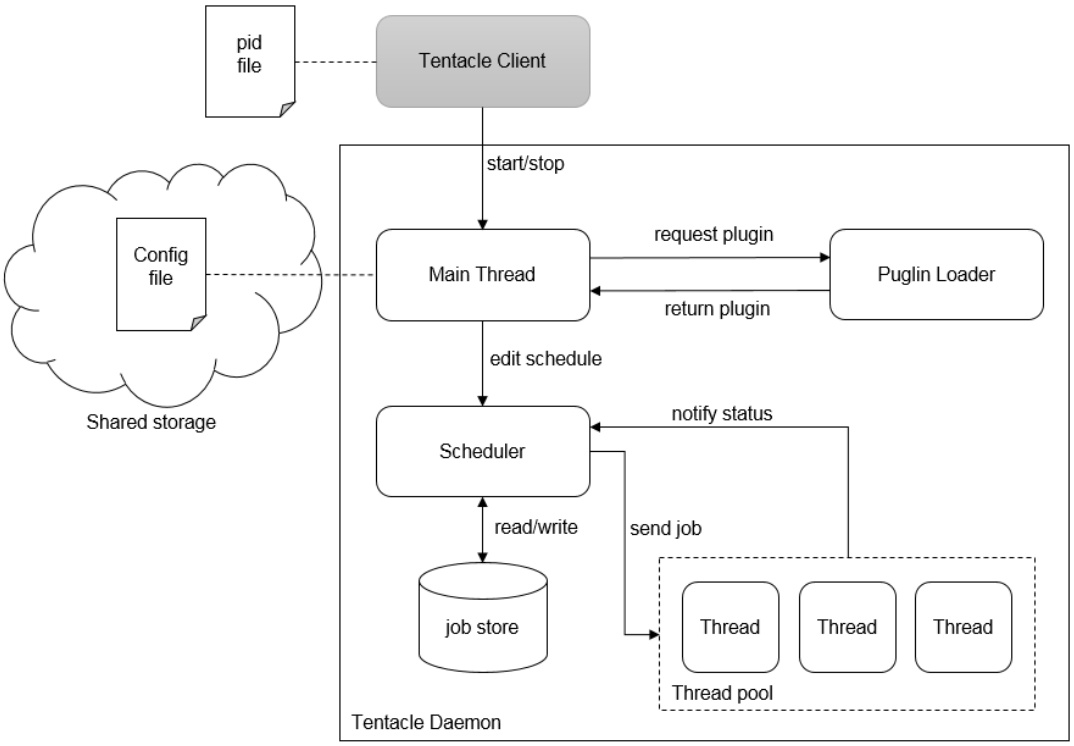

Figura 4.1: Diagrama de Componentes Octopus Tentacle

4.1.2 Diagrama de actividades

La ejecución del monitor comienza en el cliente, el cliente es el encargado de instanciar el proceso daemon y lo hace a través del método bien conocido de doble fork. Para asegurarse de no ejecutar múltiples instancias del monitor se revisa un archivo .pid, si el archivo existe significa que el monitor se está ejecutando y el cliente informa del error al usuario.

Después de que el proceso esta daemonizado comienza la fase de inicializacion, para esto se crea el planificador y se cargan las tareas del jobstore, importando el codigo de las pruebas a ejecutar. A este punto las pruebas están planificadas tentativamente y cargadas en memoria pero solo serán ejecutadas después de que se inicie el planificador.

Al iniciar el planificador este pasa a ejecutarse como un subproceso y se encarga de iniciar las tareas en el momento preciso. Cuando el planificador inicia una tarea se la pasa al ejecutor en este caso un grupo de subprocesos previamente inicializados, cuando un subproceso termina de ejecutar una tarea informa al planificador, esto con la finalidad de implementar políticas de concurrencia de tareas.

Mientras tanto, el hilo principal se conecta a la nube usando el API de Dropbox que posee un método para notificar a los clientes sobre cambios a una carpeta en tiempo real y con baja latencia, el método consiste en abrir una conexión HTTP con un alto valor de timeout (entre 30 y 120 segundos), si ocurre un cambio en la carpeta, el servidor de Dropbox responde de inmediato indicando que ocurrieron cambios y el monitor procederá a manejar este evento (descargando los archivos nuevos y aplicando los cambios al plan de monitoreo), en caso contrario, Dropbox responde indicando que no ocurrieron cambios y el monitor reinicia la conexión.

Los eventos relevantes al monitor tentacle son los siguientes: (1) una prueba se ha agrega al plan de monitoreo y debe cargarse a memoria y planificarse (2) una prueba se ha eliminado y debe ser eliminada del plan de monitoreo (3) el intervalo entre pruebas de una prueba ha cambiado y debe replanificarse la próxima ejecución (4) los parámetros de una prueba han cambiado (5) se ha agregado un nuevo enlace a monitorear (6) se ha eliminado un enlace monitoreado.

El monitor solo puede detenerse a través de una señal del sistema operativo (SIGTERM), el cliente utiliza el archivo pid para determinar el id del proceso y enviar la señal. El manejador de excepciones del monitor entonces inicia una secuencia de apagado, deteniendo las pruebas, apagando el planificador y desbloqueado el archivo .pid.

Esta secuencia de actividades puede verse en la figura 4.2.

4.2 Componentes

4.2.1 Cliente

El cliente de Octopus Tentacle es el programa encargado de iniciar o detener la ejecución del monitor en modo daemon y funge como interfaz entre el usuario y monitor.

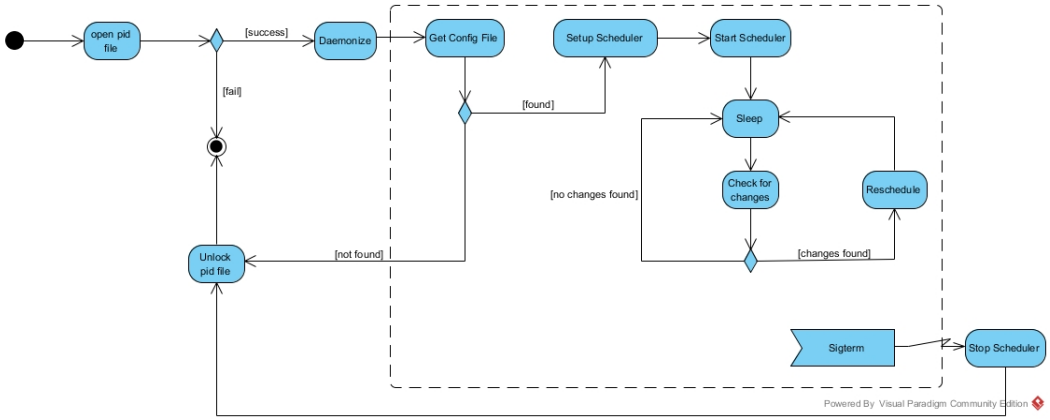

Figura 4.2: Diagram de actividades de Tentacle

Se puede invocar el cliente con los siguientes comandos:

Comando	Descripción
start	Inicia el monitor como un proceso daemon, falla si el archivo .pid ya existe (el monitor se esta ejecutando)
stop	Detiene el monitor enviando la señal SIGTERM al proceso daemon, falla si el archivo pid no existe (el monitor no se esta ejecutando)
restart	Detiene e inicia el monitor
jobs	Muestra las tareas pendientes en el planificador (plan de monitoreo)

Tabla 4.1: Comandos del cliente Octopus Tentacle

Los comandos start y restart se pueden invocar con la opción –no-daemon para iniciar el monitor en modo consola (sin daemonizar)

4.2.2 Hilo Principal

El hilo principal de ejecución es el punto de partida desde el momento en que el proceso ya se ha convertido en un daemon, se encarga de inicializar el planificador el cual a su vez pasa a ejecutarse en segundo plano, luego, su tarea consiste en mantener el plan de monitoreo al día a partir de los cambios que se hagan en Octopus Head.

El hilo principal mantiene una conexión constante con el servidor de Dropbox para ser notificado de cambios en el plan de monitoreo, si hay cambios, este llama los métodos

para leer los archivos y a partir de ellos hacer cambios en plan de monitoreo, como eliminar pruebas o agregar nuevos enlaces monitoreados.

4.2.3 Planificador

Para la implementación del planificador se hizo uso de APScheduler, una biblioteca que permite retrasar la ejecución de código python; el planificador se puede ejecutar como un subproceso de modo que es posible agregar, eliminar y re-planificar tareas en cualquier momento desde el hilo principal de la aplicación.

APScheduler consiste de un conjunto de componentes configurables que se pueden extender o re-usar para obtener cualquier comportamiento deseado, a continuación se explica su funcionamiento y como se usaron en el marco de esta aplicación.

4.2.3.1 Triggers (Gatillos)

Los triggers contienen la lógica para determinar en que momento se debe ejecutar una tarea, la biblioteca incluye varios triggers predefinidos, de los cuales dos se han usado para el desarrollo de esta aplicación:

- Interval Trigger: ejecuta pruebas en intervalos regulares, opcionalmente se pueden suministrar fechas finales e iniciales de modo que las pruebas solo se ejecuten durante un periodo especifico
- Date Trigger: ejecuta la prueba en una fecha especifica dada, una sola vez, útil para desplegar pruebas que usen muchos recursos de red como benchmarks de ancho de banda o capturas de tráfico de la red.

4.2.3.2 Executors (Ejecutores)

El ejecutor es el ente encargado de llevar a cabo la ejecución de las tareas, en nuestro caso se ha hecho uso de un grupo de subprocesos (thread pool), la cantidad de hilos en el grupo esta dado por el numero de pruebas que estaremos ejecutando de modo que siempre se tenga al menos un hilo disponible cuando se inicia una prueba.

4.2.3.3 Almacén de tareas (Job store)

El almacén de tareas guarda las tareas planificadas, el comportamiento por defecto es guardar las tareas en memoria, pero existen distintos tipos de almacenes como redis (ver sección 2.7.4) y bases de datos; hemos usado el SQLAlchemy job store que permite guardar tareas en una base de datos ligera como sqlite y así asegurar la persistencia de los datos de la aplicación.

4.2.4 Almacenamiento Compartido

El almacenamiento compartido se encarga de alojar los resultados de las pruebas en archivos de trazas, por cada prueba que se esté ejecutando se crea una carpeta donde se alojan sus respectivos archivos.

Para mantener el numero de archivos en el almacenamiento compartido razonablemente pequeño se crea para cada enlace un archivo por hora y todas las trazas que se generen en ese periodo se anexan al archivo correspondiente; el costo de alojar archivos en la nube no depende del numero de archivos sino del espacio total de disco en uso, ya que existe un limite de peticiones diarias por usuario debemos intentar minimizar la cantidad de peticiones que realizamos a Dropbox. Este tema se explicará a profundidad en la sección **??**.

4.3 Pruebas Implementadas

Gracias a la arquitectura de Tentacle, implementar una prueba es tan sencillo como crear un paquete e implementar la función "run" en el archivo init.py, todas las pruebas implementadas hasta ahora comparten una flujo de ejecución similar:

1. Se lee el archivo de configuración
2. Se obtienen los enlaces a monitorear y los parámetros globales
3. Se ejecuta la prueba por cada enlace monitoreado (ya sea en paralelo o en secuencia).

4. Se ejecuta un comando externo externo como ping o traceroute o se usa una biblioteca python para evaluar alguna métrica del enlace.

5. (opcional) si se ejecuta un comando externo se hace un parsing para extraer los resultados relevantes de la salida del programa

6. Se guardan los resultados en un archivo de trazas; generalmente el resultado de una prueba está representado por una linea en archivo de trazas, sin embargo el desarrollador tiene libertad total sobre el formato que utilice para generar archivos de trazas

Durante el desarrollo de este proyecto se han implementado las siguientes pruebas:

4.3.1 Ping

Esta prueba hace uso del comando ping para obtener datos de la latencia en un enlace, como ya se menciono en la sección 2.7.14 ping viene incluido en todas las distribuciones de linux por lo que no es necesario instalar ninguna dependencia o programa externo.

La prueba consiste en ejecutar el comando ping para cada uno de los enlaces monitoreados, ejecutamos el comando con la opción -D para que ping imprima cada resultado de latencia con una marca de tiempo entre corchetes, un ejemplo de la salida de ping se puede ver en la figura 4.3.

Es muy sencillo extraer los datos relevantes de la salida de ping, para esto recorremos la salida descartando las lineas que no comiencen con el carácter '[', separamos la salida en palabras, la palabra en la posición 0 corresponde a la marca de tiempo, luego buscamos las palabras que comiencen por "icmp_seq o icmp_req y time" para obtener numero de secuencia icmp y tiempo de ida y vuelta respectivamente.

Independientemente del número de sondas que se envíen elegiremos solo un resultado de latencia por prueba (la mediana), si para una prueba no se obtiene ninguna respuesta entonces guardamos una traza con rtt=-1, indicando que el enlace está inactivo o el nodo está rechazando el protocolo.

```
jesus@jesus-pc:~$ ping 150.185.138.59 -c 10 -i 1 -D
PING 150.185.138.59 (150.185.138.59) 56(84) bytes of data.
[1443758089.876135] 64 bytes from 150.185.138.59: icmp_seq=1 ttl=47 time=8485 ms
[1443758090.547867] 64 bytes from 150.185.138.59: icmp_seq=2 ttl=47 time=8148 ms
[1443758091.448875] 64 bytes from 150.185.138.59: icmp_seq=3 ttl=47 time=8041 ms
[1443758092.293731] 64 bytes from 150.185.138.59: icmp_seq=4 ttl=47 time=7878 ms
[1443758093.116296] 64 bytes from 150.185.138.59: icmp_seq=5 ttl=47 time=7693 ms
[1443758093.928141] 64 bytes from 150.185.138.59: icmp_seq=6 ttl=47 time=7497 ms
[1443758094.578914] 64 bytes from 150.185.138.59: icmp_seq=7 ttl=47 time=7140 ms
[1443758095.072853] 64 bytes from 150.185.138.59: icmp_seq=8 ttl=47 time=6625 ms
[1443758095.625384] 64 bytes from 150.185.138.59: icmp_seq=9 ttl=47 time=6170 ms
[1443758095.701993] 64 bytes from 150.185.138.59: icmp_seq=10 ttl=47 time=5246 m
s

--- 150.185.138.59 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9065ms
rtt min/avg/max/mdev = 5246.154/7292.862/8485.394/958.056 ms, pipe 9
```

Figura 4.3: Ping output

Parametro	Tipo	Descripcion
Numero de sondas	Entero	Número de sondas icmp a enviar.
Timeout	Float	Tiempo a esperar por una respuesta antes de asumir una sonda como perdida.
Intervalo entre sondas	Float	Tiempo entre el envío de cada sonda individual.

Tabla 4.2: Parametros de la prueba ping

4.3.2 Httping

Esta prueba usa el protocolo HTTP para hacer un HEAD Request y obtener el tiempo de respuesta y código de estatus HTTP, a diferencia de la prueba con ping esta no ejecuta un comando externo sino que llama a una función de la biblioteca 'requests' que hace la petición directamente por lo que no es necesario ningún tipo de parsing.

La tabla 4.3.2 muestra los parámetros de la prueba httping,

Esta prueba es útil para monitorear todo tipo de servidores web y posee la ventaja sobre ping de utilizar un protocolo de capa aplicación por lo que da una idea mas precisa de la experiencia del usuario al visitar dicho servicio; a diferencia de ping, el resultado obtenido no solo es la latencia de la red, sino la suma de la latencia de la red y el tiempo de respuesta del servidor, que puede estar sujeto, por ejemplo, al nivel de carga que esté manejando dicho servicio, o a la petición especifica que se esté realizando.

Parametro	Tipo	Descripcion
Timeout	Float	Tiempo a esperar por una respuesta
Path	String	Cadena de caracteres que se adjunta al ip o nombre de dominio del enlace, especialmente útil para probar servicios específicos de una aplicación o servicio web..
Port	Entero	Especificia el puerto al que se envía la peticion HTTP.

Tabla 4.3: Parametros de la prueba httping

4.3.3 Traceroute

Esta prueba ejecuta el comando traceroute para obtener la ruta entre un par de nodos a través de una red ip, llamamos ruta a una secuencia de saltos (hops) que hace un paquete al atravesar un enrutador.

El comando traceroute imprime la ruta como una lista ordenada donde cada linea representa un salto, con su dirección ip, nombre de dominio y latencia, dependiendo del numero de sondas que se estén enviando por salto pueden existir casos en que se obtenga respuesta de mas de una dirección ip, este comportamiento se puede ver en la figura 4.4 en el salto 9 se observa que obtenemos una respuesta de la dirección ip 154.54.31.230 y dos de 154.54.47.154

```
jesus@jesus-pc:~$ traceroute 150.185.138.59
traceroute to 150.185.138.59 (150.185.138.59), 30 hops max, 60 byte packets
 1  192.168.1.1 (192.168.1.1)  0.296 ms  0.399 ms  0.504 ms
 2  186.14.23.1 (186.14.23.1)  463.401 ms  463.450 ms  464.369 ms
 3  200.82.134.67 (200.82.134.67)  460.462 ms  461.378 ms  462.339 ms
 4  10.1.232.145 (10.1.232.145)  492.168 ms  493.138 ms  495.226 ms
 5  10.1.230.98 (10.1.230.98)  582.430 ms *  583.399 ms
 6  ro-ccs-bdr-02-TenGig4-1-0.ln.inter.com.ve (10.1.230.62)  496.203 ms  458.475 ms  503.555 ms
 7  tengigabitethernet4-1.asr1.ccs1.gblx.net (64.215.248.93)  505.796 ms  506.888 ms  507.880 ms
 8  te0-0-0-34.ccr21.mia03.atlas.cogentco.com (154.54.12.69)  593.042 ms  591.014 ms  592.035 ms
 9  te4-1.mag01.mia03.atlas.cogentco.com (154.54.31.230)  590.031 ms te7-1.mag01.mia03.atlas.cogentco.com (154.54.47.154)  596.323 ms  593.998 ms
10  * 38.104.95.186 (38.104.95.186)  594.960 ms  570.787 ms
11  pa-us.redclara.net (200.0.204.6)  634.901 ms  637.250 ms *
12  reacciun-pa.redclara.net (200.0.204.150)  761.604 ms  758.751 ms  756.794 ms
13  150.185.255.86 (150.185.255.86)  680.235 ms  677.786 ms  680.704 ms
14  190.168.0.5 (190.168.0.5)  709.765 ms  709.753 ms  709.954 ms
15  150.185.163.248 (150.185.163.248)  710.052 ms  687.120 ms *
16  * 150.185.138.59 (150.185.138.59)  654.827 ms  654.779 ms
```

Figura 4.4: Salida del comando traceroute

A partir de la salida de traceroute se debe obtener una estructura de datos que

facilite el análisis de la ruta, para esto se usó el modulo traceroute parser.py[1], que extrae la información de la cabecera (ip destino y nombre de dominio), asi como una lista de hops (saltos), cada hop es a su vez una lista de probes (sondas), cada sonda tiene dirección ip, nombre de dominio, rtt y anotaciones; ya que el ip destino es conocido, solo se guarda en el archivo de trazas la lista de saltos en formato json.

El formato esta compuesto de la siguiente manera:

```
1  [
2  [
3  {
4  "anno": anno_1,
5  "rtt": rtt_1,
6  "ipaddr": ipaddr_1,
7  "name": name_1
8  },
9  {
10 "anno":  anno_2,
11 "rtt": rtt_2,
12 "ipaddr": ipaddr_2,
13 "name": name_2
14 },
15 ...
16 ],
17 ...
18 ]
```

Cada vez que se realiza una prueba con traceroute se anexa al archivo de trazas una entrada con la marca de tiempo de inicio de la prueba, un carácter de separación y luego una cadena en el formato json antes mostrado.

Las parámetros de esta prueba son los siguientes:

[1] traceroute parser.py es cortesia del proyecto Netalyzr: http://netalyzr.icsi.berkeley.edu

Parametro	Tipo	Descripcion
Numero de sondas	Entero	Número de sondas a enviar por cada valor de TTL.
TTL Maximo	Entero	Numero maximo de saltos antes de asumir que el nodo no es alcanzal

Tabla 4.4: Parametros de la prueba con traceroute

4.3.4 Throughput con Iperf

La prueba de throughput con iperf mide el ancho de banda entre un par de nodos usando el programa Iperf, como se pudo ver en las pruebas anteriores, siempre es necesario que el equipo destino (Tentacle Probe Destination) ofrezca algún tipo de respuesta ya sea en forma de ICMP Echo Reply o HTTP Response, en estos casos no necesariamente hace falta instalar o configurar el equipo destino pues estos ya implementan dichos servicios, sin embargo en el caso de Iperf el usuario que realiza el monitoreo debe asegurarse de mantener una instancia de Iperf en modo servidor para realizar las pruebas, en otras palabras el usuario debe tener acceso o contar con la colaboración explicita de los puntos finales a monitorear.

Parametro	Tipo	Descripcion
Probar este enlace	Booleano	Determina si se debe o no ejecutar la prueba para un enlace e
Tiempo de transmisión	Entero	Numero de segundos para transmitir datos durante la prueba.
Bytes a enviar	String	Cantidad de bytes a enviar durante la prueba, se puede especi
Numero de flujos	Entero	Numero de flujos TCP simultáneos a usar durante la prueba.
Puerto	Entero	Especifica el puerto en que el servidor está escuchando en el T

Tabla 4.5: Parametros de la prueba con iperf

4.4 Casos de uso

Ya que el monitor de red Tentacle funciona de forma automatizada y todas las acciones de configuración y visualización de los datos recolectados por el se realizan en la aplicación web, solo se tienen cuatro casos de uso para el monitor.

Tabla 4.6: Caso de uso – Iniciar monitor

TM-01	*Iniciar monitor*	
Descripción	El usuario desea iniciar el monitor de red tentacle.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario invoca el cliente con el comando "start".
	2	El cliente crea el proceso daemon y retorna.
Excepciones	**Paso**	**Acción**
	3	Si el archivo pid existe entonces el cliente muestra un mensaje de error indicando que el monitor ya se esta ejecutando.

Tabla 4.7: Caso de uso – Detener monitor

TM-02	*Detener monitor*	
Descripción	El usuario desea detener el monitor que se esta ejecutando en modo daemon.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario invoca el cliente con el comando "stop".
	2	El cliente abre el archivo pid y envía la señal SIGTERM al proceso daemon.
	3	El monitor maneja la excepción deteniendo su ejecución.
Excepciones	**Paso**	**Acción**
	4	Si el archivo pid no existe, el cliente informa al usuario que el monitor no se esta ejecutando.

Tabla 4.8: Caso de uso – Reiniciar monitor

TM-03	*Reiniciar monitor*	
Descripción	El usuario desea reiniciar el monitor que se esta ejecutando en modo daemon.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario invoca el cliente con el comando "restart".
	2	El cliente abre el archivo pid y envía la señal SIGTERM al proceso daemon.
	3	El monitor maneja la excepción deteniendo su ejecución.
	4	El cliente crea el proceso daemon y retorna.
Excepciones	**Paso**	**Acción**
	5	Si el archivo pid no existe, el cliente informa al usuario que el monitor no se esta ejecutando.

Tabla 4.9: Caso de uso – Ver Plan de Monitoreo

TM-04	*Ver Plan de Monitoreo*	
Descripción	El usuario desea ver el plan de ejecución del monitor.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario invoca el cliente con el comando "jobs".
	2	El cliente abre la base de datos y retira la información.
	3	El cliente muestra por pantalla las tareas planificadas y los enlaces monitoreados y retorna.
Excepciones	**Paso**	**Acción**
	5	Si la base de datos no se ha creado muestra una lista vacía.

Capítulo 5

Framework de integración de pruebas

A pesar de que Octopus Monitor incluye un conjunto de pruebas como ping, httping, traceroute y iperf que proveen datos sobre latencia, rendimiento, disponibilidad y alcanzabilidad de los enlaces y servicios monitoreados, una de sus características mas importantes es la facilidad de implementación de nuevas pruebas.

El framework de integración de pruebas le ahorra al desarrollador la repetición de las tareas comunes a todas las pruebas como definición de URLs, formularios de parámetros, lógica de las vistas, escritura de plantillas,etc. Gracias a esto, el desarrollador se puede concentrar en escribir solo el código especifico a cada prueba particular facilitando y acelerando la expansión de la aplicacion.

Para entender los requisitos que resultarán en los distintos componentes que serán parte del framework, debemos entender el flujo de trabajo del monitoreo a través del cual se obtienen datos de las redes monitoreadas:

1. El usuario selecciona una prueba de la lista de pruebas disponibles para el monitor.

2. La aplicación web despliega un formulario donde el usuario puede planificar la ejecución de la prueba seleccionada y establecer sus parámetros.

3. La aplicacion web sube un mensaje al buzón del monitor de red con los detalles

de la prueba planificada.

4. El monitor es notificado del nuevo mensaje, lee su contenido y prepara la prueba para su ejecucion

5. El monitor ejecuta la prueba y guarda los resultados en un archivo de trazas temporal

6. El monitor sube los archivos generados a la nube para su almacenamiento a largo plazo

7. La aplicacion web inicia una rutina de sincronización la cual descarga los archivos, los procesa según el formato del archivo y los guarda en la base de datos.

8. El usuario selecciona una visualización dada de la lista de visualizaciones disponibles y selecciona los datos y las opciones a través de un formulario.

9. El servidor busca la visualización solicitada en el caché, si no la encuentra, llama a la función correspondiente para computarla, la introduce en el cache para futuras consultas y la muestra al usuario.

A partir de este flujo podemos inferir los siguientes requisitos:

1. Un módulo de planificación genérico de pruebas que incluya los siguientes componentes:

   - Definiciones de las pruebas disponibles
   - Una lista de parámetros para cada una de las pruebas
   - Un formulario para permitir la selección de los parámetros cada vez que el usuario planifica una prueba

2. Un módulo de sincronización genérico que pueda procesar los archivos de trazas y almacenar los datos de las pruebas.

3. Un módulo de análisis y visualización de los datos recolectados que incluya los siguientes componentes:

- Definiciones de las visualizaciones disponibles para los datos de cada prueba.
- Un formulario de configuración para cada visualización.
- Una función para computar cada visualización.
- Un mecanismo de caching genérico.
- Una plantilla para dibujar cada visualización.

5.1 Planificación genérica de pruebas

La planificación genérica de pruebas provee una interfaz única que permite a los usuarios agregar pruebas a ejecutar para sus monitores remotos. Como ya se explicó en la Sección 3.1.6 mantenemos en la base de datos dos tablas que describen las pruebas que son parte del sistema, la tabla "Test" que incluye los detalles básicos de cada prueba como su nombre, descripción y *nombre de módulo* y la tabla "Parameter" que incluye el nombre, tipo, valor por omisión, ámbito y otros detalles de los parámetros, una prueba puede ser vista como una agregación de sus parámetros. El sistema soporta parámetros de tipo bolean, entero, flotante y cadena de caracteres, sin embargo nuevos tipos podrían ser implementados para ser parte del sistema.

Las pruebas planificadas para un monitor se guardan en la tabla intermedia *MonitorTest*, ademas de guardar las claves del monitor y la prueba relacionadas, esta tabla guarda los parámetros de la prueba en formato json y la información de la planificación (intervalo entre prueba, fecha inicial (o fecha de ejecucion), fecha final, estado, etc.)

Es posible ejecutar pruebas según dos esquemas de planificación: pruebas periódicas que son ejecutadas en intervalos regulares a partir de una fecha inicial y hasta una fecha final y pruebas de una sola ejecucion que pueden ser ejecutadas en un tiempo específico. Algunas pruebas como las de ping y httping están pensadas para ser ejecutadas de forma periódica y así obtener datos constantemente, otras pruebas como iperf que usan un gran número de recursos de red pueden ser planificadas para ejecutarse en un momento específico o con un intervalo entre pruebas mucho mas largo en caso de planificarse de forma periódica.

Cuando un usuario desea planificar una prueba, el sistema filtra entre el conjunto de pruebas para determinar aquellas que están disponibles para su monitor, de esta manera, por ejemplo, un dispositivo de bajo poder de computo como un raspberry pi no tendría disponible una prueba computacionalmente intensiva de troughput multi-flujos, esto se podría extender también para tomar en cuenta el sistema operativo del dispositivo host y así se podrían implementar pruebas específicas para cada sistema operativo. Por ahora, el sistema filtra las pruebas según el esquema de planificación: algunas pruebas solo pueden ser planificadas en intervalos periódicos, otras solo pueden ser planificadas en un tiempo específicos y otras pueden ser planificadas con ambos esquemas.

Cuando el usuario selecciona una prueba el sistema crea un formulario partir de la lista de parámetros, ya se mencionó que los parámetros pueden ser de tipo bolean, entero, flotante y cadena de caracteres, pero también pueden ser "globales" de forma tal que se apliquen a todos los enlaces monitoreados o "específicos" de forma tal que se puedan establecer por enlace.

Los formularios en Django se modelan a través de una clase "Form" que provee todas las funcionalidades necesarias para el despliegue de formularios y validación de datos introducidos por el usuario, cuando deseamos crear un nuevo formulario debemos crear una subclase de "Form" e incluir todos los campos que serán parte del formulario y cualquier validación especial de dichos campos.

5.1.1 Formulario de Parámetros de la Prueba

Es posible crear clases de forma dinámica en tiempo de ejecucion usando las características de meta-programación de python; el Algoritmo 9 muestra la estructura básica para agregar campos al formulario según el tipo de datos de los parámetros, sin embargo cada parámetro también incluye datos para determinar si el campo es obligatorio, texto de ayuda y valor por omisión.

La clase generada luego es pasada a una plantilla genérica capaz de dibujar cualquier formulario dado, cuando el usuario introduce los datos estos son validados por la misma forma, el formulario generado incluye validaciones básicas de tipos de datos, un trabajo futuro podría incluir validaciones mas complejas como intervalos máximos y mínimos

Algorithm 9 Generación dinámica de formularios

```
1: procedure MAKE_PARAMETER_FORM(test,monitor)
2:     parameters ← get all parameters for test
3:     form_fields ← empty ordered dictionary
4:     for field in parameters do
5:         if field is global then
6:             form_fields[field.name] ← GET_FIELD(field.type)
7:         else
8:             links ← all monitor links
9:             for link in links do
10:                key ← concatenate field.name '_' link.id
11:                form_fields[key] ← GET_FIELD(field.type)
       return A class inheriting FORM base class containing form_fields
12: procedure GET_FIELD(type)
13:    if type is boolean then return boolean field
14:    else if type is integer then return integer field
15:    else if type is float then return float field
16:    else if type is string then return char field
17:    else
18:        throw exception "The type is unsuported"
```

para los capos numéricos o validaciones personalizables para campos de tipo "string".

5.1.2 Formulario de Planificación

Junto con el formulario de parámetros debemos incluir un formulario para introducir las fechas iniciales, finales y intervalo entre pruebas en caso de ser una prueba periódica o el tiempo de ejecucion en caso de ser una prueba de una sola vez.

La validación de este formulario es usada para asegurar las reglas de negocio del sistema:

1. La fecha inicial y la fecha final no pueden ser previas al tiempo actual

2. la fecha final no puede ser previa a la fecha inicial

3. La fecha inicial no puede ser modificada después de que la prueba ya ha comenzado

4. El intervalo entre pruebas no puede ser menor o igual que 0.

5. Solo puede ser planificada una prueba periódica simultánea del mismo tipo por monitor, es posible deshabilitar una prueba y luego planificar otra del mismo tipo para el mismo periodo

Todas las fechas se validan usando la zona horaria del monitor, para validar la condición 5. se retiran de la base de datos todos los otros *MonitorTest* pertenecientes a dicho monitor y cuya fecha final sea mayor al tiempo actual (es decir, aquellos que se estén ejecutando o se van a ejecutar en el futuro) y se verifica que la prueba a insertar en el plan de monitoreo no se intercepte con ninguna otra.

La pantalla de planificación genérica de pruebas puede ser vista en la figura **??**.

5.2 Sincronización genérica de datos

El mecanismo general de recolección de datos se ha explicado detalladamente en la Sección 3.2.3, se ha dicho que por cada prueba activa para el monitor se llama a una subrutina de sincronización que descarga los archivos de trazas e introduce los resultados en la base de datos, sin embargo no se ha entrado en detalle sobre el método de procesamiento de los archivos y la forma en que los datos de cada prueba se almacenan en la base de datos.

5.2.1 Almacén de datos

Los resultados de las pruebas generalmente consisten de una marca de tiempo, un enlace relacionado y uno o mas valores conteniendo los datos obtenidos de las pruebas, desafortunadamente no hay forma de guardar datos de esta manera "semi-estructurada" en bases de datos estructuradas por lo que sería necesario definir dos tablas, una para guardar la marca de tiempo de la traza y el tipo de prueba y

otra para guardar la lista de datos relacionados a esa prueba, sin embargo esto sería prohibitivamente ineficiente ya que habría que hacer varios joins en tablas de potencialmente cientos de millones de entradas solo para obtener los datos relacionados a una traza, igualmente se podría estar tentado a usar una tabla con el timestamp y un campo text para guardar datos en cualquier formato (como json), pero de nuevo esto sería ineficiente en términos de espacio de almacenamiento y no se podrían aprovechar las características del manejador de bases de datos para calcular promedios, máximos, mínimos o filtrar trazas por columnas específicas.

Estos argumentos evidencian la necesidad de definir modelos específicos diseñados para ajustarse a las necesidades de cada prueba y optimizados para facilitar y acelerar el cálculo de las visualizaciones relacionadas a las pruebas, como ya se vio con el calculo de mapas de calor para las datos de ping y httping.

Al diseñar el almacén de datos el desarrollador debe tomar en cuenta las características de los resultados de las pruebas, por ejemplo, muchas pruebas podrían ajustarse fácilmente al formato de una marca de tiempo seguida por un conjunto definido de valores obtenidos para ese instante, sin embargo, los resultados de las pruebas podrían tener cualquier otra estructura o estar compuestas de datos no-esturcturados; por ejemplo, traceroute tiene como resultado una lista de n saltos para alcanzar el host destino y cada salto a su vez está compuesto de m sondas.

Django incluye un poderoso ORM que puede ayudar a construir y mantener cualquier modelo de base de datos estructurada, los cambios hechos al modelo son registrados por archivos de migración que pueden ser creados en el entorno de desarrollo y luego aplicados fácilmente en el entorno de producción con un comando de migración, de esta manera crear y agregar nuevas columnas e indices a los almacenes de datos es sencillo y rápido.

5.2.2 Procesamiento de archivos de trazas

Ya que los requisitos y características de cada prueba son distintas, el desarrollador de las pruebas tiene la libertad de establecer cualquier formato al guardar sus archivos de trazas, es por esto que el sincronizador no tiene una forma "generica" de procesador dichos archivos; depende del desarrollador escribir la función o *parser* que tome como

entrada un archivo de trazas en el formato establecido para el tipo de prueba y lo transforme en objetos a introducir en la base de datos.

Para elegir la función correcta para procesar un archivo de trazas específico el mecanismo de sincronización busca en la base de datos la dirección absoluta del método y lo importa, python incluye funcionalidades para importar módulos en tiempo de ejecucion y pasar métodos a modo de variables, esto puede verse en al Algoritmo 10.

Algorithm 10 Getter de la función de sincronización

```
1: procedure GET_PARSER(test)
2:     function_path ← test.sync_function        ▷ Se obtiene la ruta absoluta del parser
3:     module_name,function_name ← split function_path        ▷ Se separa la ruta
   separando el nombre de la funcíon del nombre del módulo
4:     module ← IMPORT(module_name)
5:     function ← GET(module, function_name)  ▷ Retorna la funcion buscándola en
   el modulo
       return function
```

Es responsabilidad del desarrollador tanto escribir la ruta correctamente en la base de datos como implementar la función en sí, la función puede vivir en cualquier parte del código pero por convención se colocan en el módulo *parsing.py* dentro del paquete *management*.

El parser debe manejar archivos corruptos o mal formados evitando propagar excepciones al mecanismo general de sincronización y debe imponer las reglas de negocio específicas a la prueba, por ejemplo, no tiene sentido introducir a la base de datos valores negativos para el número de saltos para alcanzar un host en o valores imposiblemente altos, introducir valores espurios a la base de datos resultará en visualizaciones inesperadas o erróneas.

En un trabajo futuro se podría incluir un esquema de archivos de trazas genéricos en los cuales cada línea del archivo representa una traza y cada traza viene dada por un diccionario en formato json, luego el parser puede leer el archivo y mapear los diccionarios a objetos e introducirlos en la tabla correspondiente a la prueba en la base de datos, eso se podría ajustar perfectamente a pruebas ya implementadas como las de

ping, httping y iperf.

5.3 Visualización genérica de datos

Después de haber recolectado y almacenado los datos de las pruebas el ultimo paso lógico y el objetivo final del sistema de monitoreo es la visualización de los datos; las visualizaciones son cargadas y desplegadas a través de una interfaz genérica que incluye un formulario para seleccionar las opciones y los datos a visualizar, herramientas como re-computo de visualizaciones (para incluir los datos mas recientes en visualizaciones que estén obsoletas) , creación de enlaces directos para compartir y opciones para ver en una nueva pestaña.

Las visualizaciones al igual que las pruebas se guardan en su propia tabla en la base de datos; como ya se dijo en la Sección 3.1.6, esta tabla lleva el nombre de *ResultView*, y sirve para ayudar al cargador genérico de visualizaciones tanto a computar gráficas como crear urls legibles y desplegar el formulario apropiado cuando se genera la gráfica. Los tipos de visualizaciones incluidos en el sistema se pueden ver en la Tabla 5.1

5.3.1 Construcción de URLs genéricos

Debemos recordar que estamos desarrollando un framework en el tope de una aplicacion web, por lo tanto, debemos tomar en cuenta que el servidor debe tener una manera de identificar las peticiones y poder relacionarlas a una visualización específica, en la mayoría de los casos sencillamente incluimos el identificador del objeto, por ejemplo, la dirección *"/monitor/2/"* identifica de forma única la página del monitor con id 2, podríamos identificar las visualizaciones de la misma manera, sin embargo, preferimos usar cadenas de caracteres legibles por humanos que identifiquen la visualización por su nombre, a este tipo de cadenas de caracteres se les da el nombre de *slug* y se construyen reemplazando todos los espacios en el nombre por guiones (-) y los caracteres especiales por sus equivalentes en ASCII. Cuando deseamos ver una visualización específica en vez de visitar un url como "/monitor/2/view/1" se visita un URL legible como "/monitor/2/view/average-rtt-heatmap".

Para generar el slug correspondiente a la visualización, sobrescribimos el método

save de la clase *ResultView*, ya que solo deseamos generar el slug la primera vez que se guarda el registro, revisamos si el id es nulo (solo se le asigna el id a un objeto después de que es guardado por primera vez). Django incluye un método llamado *slugify* que transforma la cadena de caracteres en un slug válido, los slugs pueden ser de hasta 50 caracteres, así que el si el slug es muy largo debe ser cortado, finalmente debemos asegurarnos de que sea único, para esto buscamos en la base de datos otro *ResultView* con el mismo slug y si se encuentra anexamos al final un guión y un número, esto se repite hasta hallar un slug único de hasta 50 caracteres.

5.3.2 Interfaz de visualización

Cuando un usuario selecciona una visualización de la lista de visualizaciones disponibles para su monitor, el servidor elije el formulario correspondiente, cada visualización tiene distintos parámetros de configuración, por ejemplo algunas pueden pedir al usuario que introduzca un tiempo inicial y un tiempo final para filtrar los datos, otras podrían tener opciones para seleccionar un mes específico, o un periodo de tiempo desde el momento actual como "últimas 24 horas" o "la semana pasada", algunas visualizaciones toman en cuenta los datos de todos los tentáculos de un monitor, otras piden seleccionar primero un tentáculo específico. Como hemos visto en varias oportunidades antes, los formularios no son mas que clases que podemos importar con el método definido en el Algoritmo 10, para esto la tabla *ResultView* guarda una referencia a la clase del formulario en la columna *form_class*.

Ya que los formularios de las visualizaciones generalmente incluyen los mismos campos, el framework incluye algunos formularios que pueden ser reutilizados o extendidos para lograr cualquier comportamiento deseado; todos los formularios de visualizaciones heredan de un formulario base llamado *BaseResultForm*, este recibe el id del monitor que luego podría ser usado por algún formulario que tome en cuenta los enlaces del monitor. Por ejemplo, la clase *LinkSelectionForm* crea un menú para seleccionar de entre la lista de enlaces del monitor, todos los formularios predefinidos pueden ser vistos en la tabla 5.3.2

Una característica muy poderosa de los formularios es que se les puede "adjuntar" archivos javascript y hojas de estilo, luego la plantilla que dibuja el formulario

puede vincular estos archivos y así obtener cualquier comportamiento deseado en el formulario, por ejemplo, el formulario de los mapas de calor de RTT permite seleccionar entre periodos de "todo el tiempo", "por año" y "por mes"; si se selecciona "año" se muestra un campo para introducir el año y si se selecciona "mes" se muestra un dropdown para seleccionar el mes. Los archivos css y javascript van dentro del paquete "management" en la carpeta "static/css" y "static/js" respectivamente.

Cuando el usuario hace click en el botón "PLOT" comienza el proceso de validación del formulario, una rutina javascript toma el control, extrae los valores seleccionados del formulario y los envía al servidor a través de una petición POST, el servidor valida el formulario y genera una respuesta json:

- Si el formulario es válido, el servidor codifica un url de la visualización con los parámetros en formato GET y un "url directo".
- Si el formulario es inválido el archivo json contiene un código HTML que luego se incrusta en el tope del formulario e indica los errores de validación.
- Si no se recibe respuesta del servidor o el código de estado de la respuesta es distinto de "200 OK", entonces la rutina javascript muestra un mensaje indicando al usuario que ocurrió un problema.

Para desplegar las visualizaciones se usa un elemento HTML llamado *iframe*, este elemento se usa para incrustar páginas html dentro de otras a partir de un URL dado, la rutina javascript le pasa al iframe el URL retornado en el archivo json y el iframe a su vez lo solicita al servidor, del lado del servidor el cargador genérico devuelve una visualización en formato HTML.

Algunas veces las visualizaciones pueden tardar largos periodos de tiempo en computarse, para ofrecer al usuario retroalimentación es recomendable mostrar algún tipo de animación de carga o indicador de progreso de la operación. Para implementar esta funcionalidad, se muestra una animación de carga inmediatamente después de que el usuario hace click en el botón "PLOT" o "Re-compute", luego es posible aprovechar una señal emitida por el iframe cuando obtiene respuesta del servidor para ocultar esta animación.

La característica de re-computar visualizaciones funciona de manera análoga al flujo normal de visualización, con la única diferencia de que se solicita al cargador genérico que ignore el caché y force el computo de la visualización, el proceso completo explicado en esta subsección puede verse en la Figura **??**

TODO: insertar diagrama de actividades de la interfaz de visualizacion

5.3.3 Cargador genérico de visualizaciones

El cargador genérico de visualizaciones maneja la tarea de retornar visualizaciones listas en formato HTML, el cargador recibe el slug correspondiente a la visualización y una cadena de parámetros GET, el cargador no computa las visualizaciones inmediatamente, sino que primero se asegura de que la visualización exista en el caché.

Se puede decir que una visualización existe en el caché cuando los parámetros de dicha visualización son los mismos que los parámetros GET pasados al cargador, sin embargo, revisar esta condición traería una complejidad innecesaria ya que habría que guardar una arreglo de la lista de parámetros y compararlos cada vez que se solicita una visualización, esto sería lento y contra-propósito ya que habría que mantener tablas en memoria y hacer búsquedas sobre ellas.

Para evitar esta complejidad, las visualizaciones se identifican a través de un código hash que se genera a partir de la lista de parámetros, luego es tan simple como buscar el archivo por nombre en la carpeta correspondiente, ya que la búsqueda en un directorio depende de la cantidad de archivos en el, dividimos el caché por carpetas para acelerar las consultas al caché, las carpetas tienen como nombre el slug de la visualización a la que pertenecen, por lo tanto un archivo en el caché se identifica como: $CACHE\_ROOT/ < slug > / < hash > .html$

Cuando una búsqueda en el caché no obtiene resultados, entonces el sistema pasa a computar la gráfica, para esto, se guarda una referencia a la función en la base de datos, y esta se carga dinámicamente como ya se vió en el Algoritmo 10, las funciones de cómputo de visualizaciones se guardan dentro del paquete *visual*, en un módulo con el nombre $< test > \_visual.py$ donde test es el nombre de módulo de la prueba a la pertenece dicha visualización.

El único requisito para las funciones de computo es retornar una cadena de

caracteres válida en formato HTML (sin embargo los formatos CSV y JSON también son soportados), en caso de retornar algún otro tipo objeto (o retornar nulo) el usuario verá la representación en unicode de dicho objeto en la Interfaz de Visualización.

Se recomienda manejar apropiadamente las excepciones dentro de esta función, en caso contrario, estas se propagarán al cargador de visualizaciones que a su vez responderá con una página de error 500 que se cargará en el marco de la Interfaz de visualización.

5.3.3.1 Plantillas genéricas

Las plantillas son archivos que se usan para introducir el contenido variable de una página web dinámica a través del uso de un micro-lenguaje, el desarrollador tiene la libertad que escribir cualquier código HTML que desee, así como vincular cualquier dependencia como hojas de estilo, imágenes o archivos javascript. Para este trabajo se ha usado principalmente HighCharts y Google Maps, para dibujar la mayoría de las visualizaciones.

El desarrollador debe seguir una serie de pautas al escribir plantillas para visualizaciones, ninguna de ellas es obligatoria pero son recomendables para asegurar que el usuario tenga una experiencia consistente:

- La plantilla debe ajustarse responsivamente al ancho de la pantalla.
- Siempre se deben usar HighCharts o Google Maps sobre otras bibliotecas que tengas las mismas funcionalidades o similares.
- Las visualizaciones deben estar preferiblemente sobre fondo blanco.
- En caso de que no se encontraran datos para el periodo seleccionado, se debe mostrar un mensaje de alerta en vez de una visualización en blanco.
- se deben usar las hojas de estilo de Bootstrap al dibujar visualizaciones que estén compuestas de elementos HTML.

Para simplificar la tarea de escribir código repetitivo, se ha incluido un conjunto plantillas predefinidas, algunas de ellas pueden ser extendidas para escribir nuevas

visualizaciones, otras son plantillas listas para usar que aceptan alguna estructura de datos y resultan en una visualización, las plantillas predefinidas se pueden ver en la Tabla 5.3.3.1.

5.4 Workflow de integración de pruebas

Habiendo entendido todos los componentes que son parte del Framework de Integración de pruebas, solo queda explicar como usarlos para integrar rápidamente una prueba en un entorno de producción. El workflow de integración de pruebas consiste a de dos etapas: la etapa de desarrollo y la etapa de despliegue, esto puede ser visto en la figura 5.1.

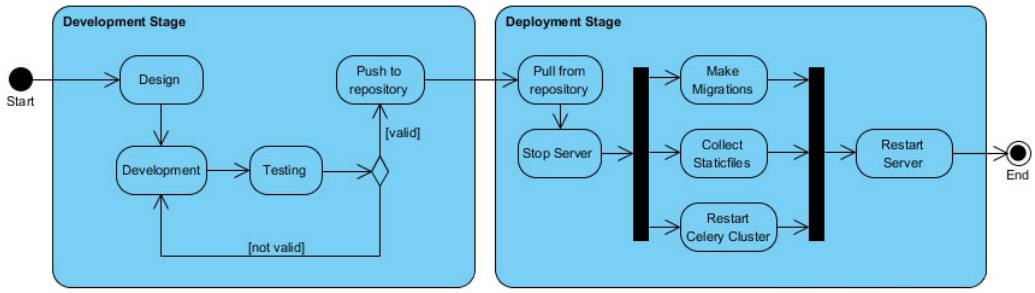

Figura 5.1: El workflow de integración de pruebas

5.4.1 Etapa de Desarrollo

Como su nombre lo indica, durante la etapa de desarrollo se diseñan, implementan y validan las pruebas y visualizaciones que serán parte del sistema en el futuro próximo; la etapa de desarrollo ocurre en un "entorno de desarrollo" que se configura para ser lo más parecido posible al entorno de producción y así evitar potenciales inconsistencias que devengan en errores críticos.

Para desarrollar código fácilmente en distintos entornos se usa un repositorio de código *git*, los cambios hechos en el entorno de desarrollo pueden ser descargados en el entorno de producción y aplicados al servidor web, los trabajadores celery y la base de datos; todos los componentes deben compartir la misma versión del código antes de ir a producción para evitar comportamientos inesperados.

Cuando se desea desarrollar una prueba, el primer paso es entender y describir los requisitos de la prueba, para facilitar el entendimiento de esta sección usaremos como ejemplo el diseño una prueba que determine la pérdida de paquetes usando ping en modo flood, es decir, que ping generará y enviará paquetes tan rápido como pueda hasta pasar una cierta cantidad de paquetes; esta prueba funcionará de forma similar a ping, en la figura **??** se puede ver el módulo de prueba, este escribirá archivos en formato csv donde cada linea del archivo tiene la forma <marca de tiempo,paquetes enviados, paquetes recibidos,porcentaje de pérdidas>

Tras tener definido el módulo de prueba y el formato del archivo de trazas, pasamos a incluir en el modelo de la base de datos la tabla que alojará las trazas de esta prueba, para eso escribimos la clase *TracePkgLoss*, en la Figura **??** se puede observar que esta consiste de un marca de tiempo (time), el identificador de enlace (link), el número de paquetes enviados (sent_packets), el número de paquetes recibidos (recieved_packets) y un índice que incluye las columnas *time* y *link*.

Cuando hacemos cambios al modelo de la base de datos debemos crear archivos llamados *migrations* que describen los cambios que deben realizarse sobre el modelo de la base de datos, para crear estos archivos, Django crea un archivo de migración inicial que luego compara con el modelo de la base de datos y así descubre los cambios que se han hecho, para realizar esta operación Django incluye el comando *makemigrations*, las migraciones son guardadas en un paquete y conforman una especie de "historial" que puede ser aplicado a la base de datos en el entorno de producción.

El siguiente paso es crear el parser del archivo de trazas, ya que el archivo de trazas viene en formato csv es muy sencillo leer el archivo linea por linea, separar cada linea por el carácter ',', convertir las cadenas de caracteres en sus tipos de dato correspondiente y guardar cada traza en la base de datos, este proceso verse en la figura **??**

Ahora que tenemos los elementos necesarios para alojar los datos obtenidos de las pruebas solo nos queda definir las visualizaciones que harán uso de esas pruebas, por motivos de simplicidad solo definiremos una visualización que muestre una gráfica de stock de la pérdida de paquetes desde el momento actual hasta una cantidad dada de tiempo en el pasado como 24 horas.

En la figura **??** se puede observar el código de la visualización, primero tomamos

del diccionario de parámetros en número de horas seleccionadas, luego hacemos una consulta a la base de datos extrayendo todas las trazas para el periodo, luego recorremos el arreglo para ajustar los timestamp a la zona horaria del monitor, finalmente llamamos al método *render_to_string* que dibuja una plantilla, en este caso usamos la plantilla genérica *generic-stock.html*.

Habiendo definido todas las clases y funciones necesarias, solo falta introducir en la base de datos los registros necesarios para que el sistema sepa de la nueva prueba y donde buscar sus funciones correspondientes. Esto puede hacerse directamente desde el CRUD del sistema, sin embargo, es recomendable crear un archivo de migración e introducir estos registros programaticamente, de modo que se introduzcan a la base de datos cuando se ejecutan las migraciones, se pueden crear archivos de migraciones vacíos llamando al comando *makemigrations* con la opción –empty, en la figura **??** puede verse que las migraciones consiste de una lista de operaciones, es posible definir operaciones que corran código python arbitrario, definimos el método *create_entries*, que introducirá en la base de datos la prueba, los parámetros y la visualización.

Antes de pasar a la siguiente etapa es recomendable probar exhaustivamente todo el código usando distintos conjuntos de datos, parámetros, y escribiendo pruebas unitarias.

5.4.2 Etapa de Despliegue

En la etapa de despliegue se aplicarán en el entorno de producción todos los cambios realizados en la etapa de desarrollo; esta etapa consiste en una serie de pasos intencionalmente simples y rápidos para evitar largas interrupciones de servicio mientras que estos son aplicados, después de que finaliza la etapa de despliegue la nueva prueba estará disponible para que los usuarios la incluyan a sus monitores.

La etapa de despliegue comienza descargando la última versión del código del repositorio y parando el servidor web para evitar cualquier petición durante este periodo, es buena idea planificar los despliegues para momentos de baja carga del servidor, y notificar a los usuarios con previo avisto.

Para actualizar el servidor web y los trabajadores basta con reemplazar el código que estos ejecutan, sin embargo, para actualizar la base de datos se hace uso de archivos de

migraciones que se crear en la etapa de desarrollo; las migraciones pueden ser aplicadas con el comando *migrate*, para saber cual fue la última migración aplicada Django mantiene una tabla para llevar cuenta del estado de la base de datos, así es posible saber cuales de las migraciones deben ser aplicadas. Se debe tener especial ciudado al ejecutar migraciones, mientras algunas veces estas pueden ser rápidas y seguras, algunas migraciones incluyen operaciones costosas como agregar indices a tablas existentes o actualizar todas las entradas de una tabla, también es importante tomar en cuenta que las migraciones no incurran en conflictos con datos contenidos en la base de datos, por ejemplo, aplicar una restricción de unicidad sobre una columna con valores repetidos.

En la Sección 3.1.3 se dijo que la aplicacion web Django no se encarga de servir los archivos estáticos al cliente, es por esto que generalmente un entidad desacoplada ya sea el mismo servidor web configurado para servir ciertas rutas sin pasarlas a Django o otro servidor web aparte deben proveer este servicio, sin embargo los archivos estáticos viven junto al resto del código de la aplicacion, para que el servidor desacoplado pueda encontrar estos archivos, estos deben ser copiados al directorio apuntado por la variable *STATIC_ROOT*, para esto Django incluye el comando *collectstatic* que busca en cada paquete los archivos estáticos y los copia a dicho directorio.

El paso final de despliegue es reiniciar los componentes que lo requieran, en otras palabras el servidor web y los trabajadores de modo que estos puedan importar el nuevo código, los demás micro-servicios como broker de mensajería, planificador y backend de resultados no requieren ser reiniciados durante los despliegues.

Reiniciar el servidor web es trivial usando los comandos *start, stop, restart*, sin embargo es buena idea borrar todos los archivos .pyc del código de la aplicacion para forzar a apache a releer todo el código antes de levantar la aplicacion.

Reiniciar los trabajadores puede hacerse de distintas maneras dependiendo de la configuración elegida, por ejemplo, es posible usar un programa como *supervisord*[1] para que monitoree los trabajadores y se asegure de que se mantengan en linea todo el tiempo, sin embargo, también es posible enviarles mensajes a todos los trabajadores en modo broadcast para que se reinicien a sí mismos, esto puede facilitar despliegues

[1] *Supervisord es un programa que permite monitorear y controlar conjuntos de procesos en sistemas UNIX*

en que se tienen trabajadores distribuidos en distintas máquinas, siempre y cuando se descargue la última versión del código en la maquina host del trabajador previamente, en futuras versiones el trabajador podría ser capaz de aceptar un comando para actualizarse y reiniciarse a si mismo.

tipo	descripción
heatmap	Mapa de calor, útil para mostrar mostrar la densidad o concentración de una variable sobre un plano 2D.
bar chart	Gráfica de barras, útil para mostrar las proporciones entre un conjunto de valores.
line chart	Gráfica lineal, útil para mostrar la evolución de una variable con respecto a otra.
scatter chart	Gráfica de puntos, generalmente usada para mostrar los valores de un conjunto de muestras sobre un plano 2D
map	Cualquier tipo de mapa útil para mostrar información geográfica.
network	Gráfica de red, útil para mostrar la relación entre distintas entidades.
table	Tabla, útil para mostrar todo tipo de información.
pie chart	Gráfica de pie, útil para mostrar la proporción entre dos variables
stock chart	Gráfica de stock, útil para mostrar la evolución de una variable y conjuntos especialmente grandes de datos.
csv	*Comma separated values* un tipo de archivo popularmente usado para analizar conjuntos de datos estructurados, podría ser usado para permitir análisis de datos externos a octopus
json	Formato json, podría usarse en versiones futuras para permitir implementar visualizaciones en otras plataformas o con otras herramientas.
other	Cualquier tipo de visualización que no quepa en las categorías anteriores.

Tabla 5.1: Tipos de visualizaciones soportadas por el sistema

Clase	Clase Base	Descripción
BaseResultForm	Form	Formulario base para todas las visualizaciones
LinkSelectionForm	BaseResultForm	Formulario para seleccionar enlace de un monitor
SampleForm	LinkSelectionForm	Formulario para seleccionar un número de muestras a graficar
DateRangeForm	BaseResultForm	Formulario para seleccionar un periodo de tiempo a graficar
ElapsedForm	BaseResultForm	Formulario para seleccionar un periodo de tiempo a partir del momento actual
LinkMultiselectForm	BaseResultForm	Formulario para seleccionar un conjunto de enlaces de un monitor

Tabla 5.2: Formularios predefinidos

Plantilla	Descripcion
base-visual.html	Plantilla base para todas las visualizaciones
base-highcharts.html	Plantilla base para todas las visualizaciones con HighCharts
base-highstock.html	Plantilla base para todas las visualizaciones con HighStock (submódulo de Hihcharts para dibujar stockcharts)
base-googlemaps.html	Plantilla base para visualizaciones con Google Maps.
generic-heatmap.html	Plantilla base para mapas de calor de tipo <hora,tiempo,valor>
generic-stock.html	Plantilla base para Stockcharts de tipo $tiempo, valor$
generic-stock-multi.html	Plantilla base para Stockcharts de tipo <tiempo,valor>con múltiples series.
generic-line.html	Plantilla base para gráficas de linea de tipo <time,value>
generic-line-multi.html	Plantilla base para gráficas de linea de tipo <time,value>con múltiples series,

Tabla 5.3: Plantillas Predefinidas

Capítulo 6

Conclusiones y Recomendaciones

A través del desarrollo de este trabajo, construimos un agente monitor de redes ligero y a la vez flexible, capaz de monitorear enlaces en despliegues arbitrarios de red tales como redes comunitarias; y una plataforma web para manejar, recolectar y analizar la data generada por los monitores remotos.

The most relevant accomplishments made from a computational standpoint are listed below: We built a monitor with the minimal amount of code to schedule periodic monitoring tasks according to a configuration file shared with the web app, the monitor, built in python is lightweight and perfect to deploy on embedded and low cost devices. We established a zero-cost file synchronization scheme between the web server and the remote monitors through the use of the free quotas that cloud storage services like Dropbox offer. A framework to easily integrate new tests and visualizations was designed and implemented, the framework was tested with three test types: Ping test: uses the ping command to determine round trip time between the monitor and a certain node, with this test it is possible to infer network congestion as well as activity and inactivity periods. HTTP test: sends an HTTP head request to determine the response time as well as status code, useful to determine availability of web services, activity and inactivity periods and infer about server congestion based on response time. Traceroute test: uses the traceroute command to determine the route that a package follows through a network to reach a certain node, this test can be used to find routing inefficiencies, congestion related to certain hosts and of course reachability.

Several data visualization methods were conceptualized and proved to be a great way to infer facts about the network behavior, a prime example of this are latency heatmaps which are useful not only to get an idea about the latency at certain moments but clearly display patterns and inactivity periods as white spaces. An architecture able to handle the data gathering and visualization generation tasks efficiently with the available resources was defined, with the appropriate delegation of long-running and I/O blocking tasks a single web server is able to quickly handle more requests, that would otherwise be queued (or worse, dropped) increasing response times. We found a multi-layered modular architecture that can be scaled indefinitely, each part of the stack can be scaled independently to avoid possible bottlenecks, the challenges and considerations to scale certain components were analyzed and possible solutions were proposed.

Through the development of this work we performed tests to determine the performance and stability of both the web server stack and the network monitor on real deployment environments, including real life use cases.

A test monitor was deployed to test critical services of ULA's network like ula.ve home page and saber.ula.ve knowledge repository, this test monitor has been running uninterrumptly since august 6th 2015 to the current date. The monitor proved to be stable and showed no apparent memory leaks that would slow its execution even after prolonged periods of time. The web app was deployed at a test server machine, constant integration and testing of new features in a production enviroment was crucial in the development of the test integration workflow, the web app is available at the following URL: http://150.185.138.59/octopusmonitor. The shared datastore synchronization along with the Dropbox API integration at the web app showed to be a reliable way to transfer the data from the monitor to the web app, although there is a limit for the API requests that can be made for each authenticated user, this limit has proven to be high enough even with the highest synchronization frequency. At the moment of the writing of this paragraph more than 1161533 traces has been synchronized though this mechanism. The web app task scheduler proved to be reliable at launching tasks at the proper time as well as quick to update its internal state after changes to the schedule on the database were made. The test integration workflow was successfully executed

several times during the development, allowing downtimes of merely a few minutes during each update deployment. Stress tests were performance against different server setups Although making analysis over the collected data it is out of the scope of this work, we made useful discoveries about the latency, availability, uptime, and reachability of the monitored links, for example, we found a clear pattern of lower and stable latency between the first hour of the day and the early hours of the morning that matches the usual pattern of network congestion, we found constant downtimes for the ula.ve home page as well as higher response time compared to saber.ula.ve, we found that both saber.ula.ve and ula.ve rejected ICMP echo requets and could only be monitored through the use HTTP, just to name a few relevant results.

Future Work

One of the key features that might make a network monitoring system attractive for potential users is having the ability to automatically detect when critical conditions are met and alert the user proactively, emitting push notifications through a mobile app or sending sms or email. Alerts could be attached to tests and conditions could be checked after every synchronization is performed. Examples of critical conditions that could trigger alerts are: a link goes down during a prolonged period of time, the latency is above a set threshold, a server is returning http status 500, etc.

Another obvious feature is the inclusion of other monitor types, currently we provide support only for one kind of monitor thought for low-cost devices that could be left unattended and are only able to perform tests periodically. New monitor types running in more powerful devices open up various possibilities where users are able to interact through the web app with their monitors, executing tests on-demand or even streaming data directly to the client, either by using the web app as a relay between the monitor and the client, or establishing a direct connection between the client and the monitor.

Bibliografía

[1] J. F. Kurose and K. W. Ross, *Computer Networking. A Top-Down Approach.* Pearson, 2013.

[2] M. Zennaro, E. Pietrosemoli, J. Mlatho, M. Thodi, and C. Mikeka, "An assessment study on white spaces in malawi using affordable tools," in *Global Humanitarian Technology Conference (GHTC), 2013 IEEE*, (San Jose, CA), pp. 265 – 269, IEEE, 2013.

[3] J. Gomez, M. Porcar, M. Hernandez, and E. Velasquez, "Malawinet network monitor.," tech. rep., Universidad de los Andes, 2015.

[4] B. Vahl, T. Luque, F.H.and Huhn, and C. Sengul, "Network monitoring and debugging through measurement visualization," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium*, (San Francisco, CA), pp. 1 – 6, IEEE, 2012.

[5] S. Cloud, "Pingdom - website monitoring." [Página web en línea] Disponible en https://www.pingdom.com/, 2015.

[6] U. R. B. A.S., "Uptime robot." [Página web en línea]. Disponible en : https://uptimerobot.com/, 2015.

[7] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 4, pp. 277–288, 1984.

[8] I. Grigorik, *High Performance Browser Networking.* O'Reilly, 2013.

[9] S. D. Strowes, "Passively measuring tcp round-trip times," *Communications of the ACM*, vol. 56, no. 10, pp. 57–64, 2013.

[10] C. Demichelis and P. Chimento, "RFC3393: IP Packet Delay Variation Metric for IP Performance Metrics," RFC 3393, RFC Editor, November 2002.

[11] Ookla, "Speedtest.net by ookla." [Página web en línea]. Disponible en : http://www.speedtest.net/, 2014.

[12] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate, "End-to-end wan service availability," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 2, pp. 300–313, 2003.

[13] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, p. 40, 2011.

[14] S. M. LLC, "Dslreports home: Broadband isp reviews news tools and forums." [Página web en línea]. Disponible en : http://www.dslreports.com/speedtest/, 2015.

[15] N. Ltd, "Thinkbroadband :: Uk broadband speed test." [Página web en línea]. Disponible en : http://www.thinkbroadband.com/speedtest.html, 2015.

[16] R. B. Fragoso, "¿que es big data?," *IBM developerWoks*, 2012.

[17] T. G. Flu and D. T. Team, "Google flu trends." [Página web en línea]. Disponible en : https://www.google.org/flutrends/about/, 2012.

[18] Pervasifm, "Data visualization." [Página web en línea]. Disponible en : http://www.pervasif.com/index.php/news-a-event/capabilities/data-visualization, 2012.

[19] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[20] E. B. Pantoja, "El patrón de diseño modelo-vista-controlador (mvc) y su implementación en java swing," *Acta Nova*, vol. 2, no. 4, p. 493, 2004.

[21] D. S. Foundation, "The web framework for perfectionists with deadlines." [Página web en línea]. Disponible en : https://www.djangoproject.com/, 2015.

[22] A. Solem, "Celery: Distributed task queue." [Página web en línea]. Disponible en : http://www.celeryproject.org/, 2015.

[23] Oracle, "Mysql 5.7 :: Reference manual 1.3.1 what is mysql?." [Página web en línea]. Disponible en : http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html, 2015.

[24] S. W. Ambler, "Mapping objects to relational databases: O/r mapping in detail." [Página web en línea]. Disponible en : http://www.agiledata.org/essays/mappingObjects.html, 2013.

[25] I. Dropbox, "Acerca de dropbox." [Página web en línea]. Disponible en : https://www.dropbox.com/about, 2015.

[26] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye, "Ip geolocation databases: Unreliable?," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 53–56, 2011.

[27] A. Grönholm, "Advanced python scheduler - apscheduler 3.1.0.dev1 documentation." [Página web en línea]. Disponible en : https://apscheduler.readthedocs.org/en/latest/, 2015.

[28] P. N. Salvatore Sanfilippo, "Redis." [Página web en línea]. Disponible en : http://redis.io/, 2015.