Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática (Grupo en inglés)

Trabajo Fin de Grado

Modelo de Simulación de un Sistema de Almacenamiento de Información

Autor: Victor Moreno Arribas
Tutor(a): Juan Antonio Fernández del Pozo

Madrid, Junio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática (Grupo en inglés)

Título: Modelo de Simulación de un Sistema de Almacenamiento de Información

Junio 2024

Autor: Victor Moreno Arribas
Tutor: Juan Antonio Fernández del Pozo
Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Este trabajo se centra en el desarrollo de un modelo de simulación para un sistema de almacenamiento de información. Se analiza y describe detalladamente la evolución del almacenamiento desde las primeras formas físicas hasta las soluciones digitales actuales. Se clasifican los tipos de almacenamiento en físico, en la nube y en red, destacando ejemplos de tecnologías como HDDs, SSDs, Amazon Web Services y Microsoft Azure. Además, se estudian los elementos esenciales de un sistema de almacenamiento, incluyendo servidores web y de aplicaciones, bases de datos, almacenamiento de archivos, caché, colas de mensajes, componentes de seguridad, monitoreo y registro, y sistemas de copia de seguridad y recuperación. Finalmente, se presentan conclusiones y recomendaciones basadas en un modelo matemático que simula eventos aleatorios para evaluar el rendimiento del sistema. Este trabajo proporciona una guía comprensiva y práctica sobre sistemas de almacenamiento de información, útil para profesionales y académicos en el campo.

Abstract

This study focuses on developing a simulation model for an information storage system. The evolution of storage is thoroughly analyzed and described, from the earliest physical forms to modern digital solutions. Storage types are categorized into physical, cloud, and network storage, with examples of technologies such as HDDs, SSDs, Amazon Web Services, and Microsoft Azure highlighted. Additionally, the essential components of a storage system are examined, including web and application servers, databases, file storage, cache, message queues, security components, monitoring and logging, and backup and recovery systems. Finally, conclusions and recommendations are provided based on a mathematical model that simulates random events to assess system performance. This work offers a comprehensive and practical guide to information storage systems, beneficial for professionals and academics in the field.

Tabla de contenidos

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto y Motivación

El almacenamiento de información es una de las áreas más críticas en la tecnología de la información, ya que el volumen de datos generados y almacenados continúa creciendo a un ritmo exponencial. Desde los primeros métodos de almacenamiento, como las tablillas de arcilla y el papiro, hasta las modernas soluciones de almacenamiento en la nube, la capacidad para almacenar y acceder a grandes cantidades de datos ha sido fundamental para el avance de la civilización. Hoy en día, la eficiencia, la seguridad y la disponibilidad del almacenamiento de datos son esenciales para una amplia gama de aplicaciones, desde bases de datos empresariales hasta servicios de streaming y almacenamiento personal.

1.2. Objetivo del Trabajo

El objetivo principal de este Trabajo de Fin de Grado (TFG) es desarrollar un modelo de simulación de un sistema de almacenamiento de información. A través de la creación de varios prototipos de complejidad creciente, se pretende estudiar el comportamiento del sistema bajo diferentes condiciones de carga y demanda. Este modelo permitirá evaluar aspectos clave como la carga de trabajo, la disponibilidad, los tiempos de respuesta, la planificación de procesos y las políticas de asignación de recursos. Además, se considerarán diversas configuraciones, incluyendo servidores individuales, redes de servidores y soluciones en la nube con infraestructuras distribuidas.

1.3. Metodología y Objetivos

Para llevar a cabo este proyecto, se seguirán una serie de objetivos específicos que guiarán el desarrollo y la evaluación del modelo de simulación:

1. **Desarrollar un modelo de simulación de un sistema de almacenamiento de información**: Crear prototipos que incorporen la configuración del

sistema de ficheros, unidades operativas, red, equipamiento, respaldo, y políticas de consumo de energía.

2. **Documentar las variables y establecer objetivos de simulación**: Identificar y documentar todas las variables relevantes y establecer los objetivos que guiarán las simulaciones del sistema.

3. **Definir las estructuras de datos y patrones de tareas**: Especificar las estructuras de datos para representar tanto el sistema físico (hardware) como el lógico (software), así como los patrones de tareas incluyendo llegadas y servicio.

4. **Implementar un sistema de registro (log) y un dashboard de visualización**: Crear un sistema de ficheros de log para recopilar datos de la simulación y diseñar una interfaz esquemática que visualice el estado del sistema en un dashboard.

5. **Simular y evaluar el comportamiento del sistema bajo diversas condiciones**: Realizar simulaciones para analizar el comportamiento del sistema en términos de carga de trabajo, disponibilidad, tiempos de respuesta y políticas de asignación de recursos.

6. **Validar y analizar los resultados de la simulación**: Comparar los resultados obtenidos con datos reales o casos de estudio para validar el modelo y realizar un análisis exhaustivo de los datos para mejorar la fiabilidad, seguridad y eficiencia del sistema.

1.4. Resumen de los Capítulos

Para proporcionar una visión clara y estructurada del contenido de este Trabajo de Fin de Grado (TFG), a continuación se presenta un resumen detallado de cada uno de los capítulos que componen este documento. Cada capítulo ha sido diseñado para abordar diferentes aspectos del desarrollo del modelo de simulación de un sistema de almacenamiento de información, desde los fundamentos teóricos hasta la implementación y análisis de resultados. Este enfoque sistemático asegura que se cubran todos los elementos críticos necesarios para comprender y evaluar el sistema propuesto.

1.4.1. Capítulo 1: Introducción

El primer capítulo establece el contexto y la motivación del trabajo, explicando la importancia del almacenamiento de información en la era digital y los desafíos asociados con la gestión eficiente de grandes volúmenes de datos. Se detallan los objetivos del proyecto y se describe la metodología que se seguirá para alcanzarlos. Además, se proporciona una visión general de la estructura del documento, preparando al lector para el contenido que se desarrollará en los capítulos siguientes.

1.4.2. Capítulo 2: Marco Teórico

En este capítulo, se revisan los fundamentos teóricos del almacenamiento de información, incluyendo una descripción de los diferentes tipos de sistemas de almacenamiento, sus características y aplicaciones. También se analizan los antecedentes y trabajos similares en el campo de la simulación de sistemas de almacenamiento, destacando las contribuciones más relevantes y las tecnologías utilizadas. Esta revisión bibliográfica proporciona el contexto necesario para comprender las bases teóricas del modelo de simulación desarrollado en este TFG.

1.4.3. Capítulo 3: Desarrollo del Modelo de Simulación

Este capítulo se centra en el desarrollo del modelo de simulación. Se describe el diseño del modelo, incluyendo los parámetros y la configuración utilizados. Se detallan las estructuras de datos y los patrones de tareas definidos, así como las reglas de las colas y las medidas de rendimiento, fiabilidad, seguridad y disponibilidad. Además, se explica el proceso de documentación de las variables y la definición de los objetivos de simulación. Este capítulo proporciona una guía detallada sobre cómo se construyó el modelo y los componentes clave que lo conforman.

1.4.4. Capítulo 4: Resultados de Simulación

Este capítulo presenta los resultados obtenidos de las simulaciones realizadas. Se analizan los datos recopilados, interpretando las métricas y evaluando el comportamiento del sistema bajo diferentes condiciones de carga y demanda. Además, se valida el modelo comparando los resultados con datos reales o casos de estudio. Se discuten las implicaciones de los resultados y se identifican áreas de mejora. Este análisis exhaustivo es fundamental para entender la eficacia y la eficiencia del modelo de simulación desarrollado.

1.4.5. Capítulo 5: Mejoras del Modelo de Simulación

En este capítulo, se proponen y detallan una serie de mejoras para el modelo de simulación. Estas mejoras están orientadas a optimizar el rendimiento, mejorar la claridad y estructura del código, y añadir nuevas funcionalidades para una simulación más realista y completa. Se abordan aspectos como la optimización del rendimiento, la mejora de la estructura del código, y la inclusión de nuevas funcionalidades como la monitorización en tiempo real, la simulación de fallos y la escalabilidad.

1.4.6. Capítulo 6: Conclusiones y Trabajo Futuro

El capítulo final resume las conclusiones del trabajo, destacando los principales logros y hallazgos del TFG. Se reflexiona sobre el cumplimiento de los objetivos planteados y se discuten las posibles aplicaciones prácticas del modelo de simulación desarrollado. Además, se proponen recomendaciones para futuras

investigaciones y desarrollos en el campo del almacenamiento de información. Este capítulo cierra el documento ofreciendo una visión crítica y prospectiva del trabajo realizado.

Capítulo 2

Marco teórico

2.1. Historia del almacenamiento de la información

El deseo de registrar y preservar información es antiguo. Las sociedades cazadoras-recolectoras transmitían conocimiento oralmente, pero esto era limitado por la fragilidad de la memoria humana.

Con las primeras civilizaciones agrícolas, como los sumerios, surgió la escritura cuneiforme en tablillas de arcilla, permitiendo registrar transacciones y leyes. En Egipto, la invención del papiro facilitó la escritura de textos religiosos y administrativos, difundiendo el conocimiento.

La imprenta de Gutenberg en el siglo XV revolucionó la difusión del conocimiento, permitiendo la producción masiva de libros y promoviendo la Revolución Científica y el Renacimiento.

En el siglo XX, la era digital introdujo dispositivos de almacenamiento magnético, como cintas y discos duros, transformando el procesamiento y almacenamiento de información. La computación moderna se basa en estos avances.

Recientemente, el almacenamiento en la nube y en dispositivos móviles ha cambiado cómo accedemos a la información. Servicios como Google Drive y iCloud permiten almacenar datos en línea, accesibles desde cualquier lugar. Los teléfonos inteligentes nos permiten llevar información en nuestros bolsillos.

En resumen, el almacenamiento de información ha evolucionado desde la tradición oral hasta la era digital, con constantes innovaciones que continúan transformando nuestro acceso al conocimiento y la cultura.

2.2. Tipos de almacenamiento de la información

2.2.1. Almacenamiento físico

Los discos duros tradicionales (HDD) y las unidades de estado sólido (SSD) representan la base del almacenamiento físico. Los HDDs utilizan discos magnéticos, ofreciendo grandes capacidades a bajo costo, aunque con limitaciones en

velocidad y durabilidad. Los SSDs, por otro lado, emplean memoria flash, proporcionando accesos más rápidos y mayor resistencia, aunque a un precio más elevado. Además, se incluyen dispositivos USB y tarjetas SD por su portabilidad y facilidad de uso [1].

Discos duros HDD

Los HDDs operan mediante platos magnéticos y cabezas de lectura/escritura móviles. Son ideales para el almacenamiento de grandes volúmenes de datos y copias de seguridad a largo plazo. Ejemplos comunes incluyen Western Digital, Seagate y Toshiba [2].

Unidades de Estado Sólido SSD

Los SSDs utilizan memoria NAND flash, ofreciendo velocidades de acceso y escritura significativamente más rápidas que los HDDs. Son más resistentes a daños físicos y eficientes en consumo energético. Ejemplos comunes incluyen Samsung, Crucial, SanDisk e Intel [3].

Medios Extraíbles (USB, Tarjetas SD)

Los dispositivos USB y tarjetas SD ofrecen una forma portátil de transferir y almacenar datos. Son ideales para compartir archivos entre dispositivos sin conexión a internet. Ejemplos comunes incluyen SanDisk, Kingston y Lexar [4].

2.2.2. Almacenamiento en la nube

El almacenamiento en la nube permite almacenar información en servidores remotos, reduciendo la carga en el almacenamiento local y mejorando la colaboración y accesibilidad. Ejemplos comunes incluyen Amazon Web Services, Microsoft Azure, Google Cloud Platform, IBM Cloud y Oracle Cloud Infrastructure.

1. Amazon Web Services (AWS): Ofrece servicios como Amazon S3, Amazon EBS y Amazon Glacier, destacándose por su escalabilidad, confiabilidad y seguridad [5].
2. Microsoft Azure: Proporciona Azure Blob Storage, Azure File Storage y Azure Disk Storage, con características avanzadas como replicación geográfica e integración con Office 365 y Dynamics 365 [6].
3. Google Cloud Platform (GCP): Incluye Google Cloud Storage, Google Cloud Filestore y Google Cloud Persistent Disk, conocido por su escalabilidad, rendimiento y análisis de datos avanzados [7].
4. IBM Cloud Object Storage: Ofrece una escalabilidad infinita y alta durabilidad, junto con soluciones de almacenamiento de archivos y bloques, y herramientas avanzadas de análisis [8].

5. Oracle Cloud Infrastructure (OCI): Proporciona Oracle Cloud Object Storage, Oracle Cloud File Storage y Oracle Cloud Block Volume, destacándose por su alto rendimiento y seguridad [9].

2.2.3. Almacenamiento en red

El NAS (Network Attached Storage) y las SAN (Storage Area Network) son soluciones de almacenamiento en red. Los NAS son ideales para pequeñas empresas y hogares, mientras que las SAN son más adecuadas para grandes empresas que requieren alto rendimiento y capacidad. Ejemplos de NAS incluyen Synology DiskStation, QNAP y NetApp FAS Series. Ejemplos de SAN incluyen Dell EMC Unity, HPE Nimble Storage y IBM FlashSystem.

1. Synology DiskStation: Reconocidos por su fiabilidad y versatilidad, ofrecen almacenamiento para hogares y empresas con aplicaciones integradas para copias de seguridad, colaboración y más [10].
2. QNAP NAS: Conocidos por su flexibilidad y personalización, ofrecen características como almacenamiento en caché SSD, virtualización de máquinas y servicios de vigilancia [11].
3. NetApp FAS Series: Soluciones empresariales escalables y de alto rendimiento, con almacenamiento híbrido y características avanzadas como instantáneas y replicación [12].
4. Dell EMC Isilon: Diseñados para grandes volúmenes de datos y archivos no estructurados, ofrecen escalabilidad lineal y protección de datos integrada [13].
5. Buffalo TeraStation: Diseñados para pequeñas y medianas empresas, ofrecen alta capacidad de almacenamiento con características como replicación remota y compatibilidad con la nube [14].

2.3. Elementos de un Sistema de Almacenamiento de la Información

Un sistema de almacenamiento de la información puede incluir varios componentes clave para asegurar la funcionalidad, eficiencia y seguridad. Aquí se describen algunos de los elementos principales y algunos ejemplos de cada caso:

1. **Web Server (Servidor Web)**:
   - **Descripción**: Maneja las solicitudes HTTP de los clientes (navegadores web) y sirve los contenidos solicitados (páginas web, imágenes, etc.).
   - **Ejemplos**: Apache, Nginx, IIS (Internet Information Services) [15].
2. **Application Server (Servidor de Aplicaciones)**:
   - **Descripción**: Procesa la lógica de la aplicación. Puede ejecutar código, interactuar con bases de datos, manejar la autenticación, etc.

- **Ejemplos**: Node.js, Tomcat, Django (para Python), Ruby on Rails [16].

3. **Database (Base de Datos)**:
   - **Descripción**: Almacena datos estructurados necesarios para la aplicación, tales como usuarios, transacciones, contenido, etc.
   - **Ejemplos**: MySQL, PostgreSQL, MongoDB, Oracle Database [17].
4. **File Storage (Almacenamiento de Archivos)**:
   - **Descripción**: Almacena archivos estáticos y grandes como imágenes, videos, documentos, etc.
   - **Ejemplos**: Amazon S3, Google Cloud Storage, sistemas de archivos distribuidos como HDFS [18].
5. **Cache**:
   - **Descripción**: Almacena temporalmente datos frecuentemente solicitados para mejorar el rendimiento y reducir la carga en las bases de datos.
   - **Ejemplos**: Redis, Memcached [19].
6. **Message Queue (Cola de Mensajes)**:
   - **Descripción**: Facilita la comunicación y coordinación entre diferentes partes del sistema mediante el envío y recepción de mensajes.
   - **Ejemplos**: RabbitMQ, Apache Kafka, AWS SQS [20].
7. **Security Components (Componentes de Seguridad)**:
   - **Descripción**: Protege el sistema contra accesos no autorizados, ataques y otras amenazas de seguridad.
   - **Ejemplos**: Firewalls, sistemas de detección de intrusos (IDS), certificados SSL/TLS, autenticación y autorización (OAuth, JWT).
8. **Monitoring and Logging (Monitoreo y Registro)**:
   - **Descripción**: Monitorea el rendimiento del sistema y registra eventos y errores para análisis y resolución de problemas.
   - **Ejemplos**: Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), Datadog.
9. **Backup and Recovery (Copia de Seguridad y Recuperación)**:
   - **Descripción**: Mantiene copias de seguridad de los datos y proporciona mecanismos para recuperar datos en caso de fallos.
   - **Ejemplos**: Herramientas de backup específicas del proveedor de la base de datos, soluciones en la nube como AWS Backup, Google Cloud Backup.

2.4. Modelo Matemático

Para la simulación de eventos aleatorios se han empleado diversas distribuciones estadísticas que simulan el comportamiento y las respuestas de los servidores web frente a diferentes solicitudes. A continuación, se detallan las distribuciones utilizadas y sus fórmulas matemáticas.

Distribución Poisson

La distribución Poisson se utiliza para modelar el número de eventos que ocurren en un intervalo de tiempo fijo. Es útil para representar la ocurrencia de eventos raros en un gran número de oportunidades. La fórmula de la distribución Poisson es:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

donde:

- $P(X = k)$ es la probabilidad de que ocurran k eventos.
- λ es la tasa media de ocurrencias en un intervalo de tiempo dado.
- k es el número de ocurrencias.

Distribución Normal

La distribución normal, también conocida como distribución de Gauss, es una de las distribuciones más importantes en estadística. Modela variables continuas que están influenciadas por muchos pequeños factores aleatorios, y tiene la forma de una campana simétrica. La función de densidad de probabilidad de la distribución normal es:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

donde:

- $f(x)$ es la función de densidad de probabilidad.
- μ es la media de la distribución.
- σ es la desviación estándar de la distribución.
- x es el valor de la variable aleatoria.

Distribución Exponencial

La distribución exponencial modela el tiempo entre eventos en un proceso de Poisson, caracterizando el tiempo de espera entre eventos independientes que

ocurren a una tasa constante. La función de densidad de probabilidad de la distribución exponencial es:

$$f(x) = \lambda e^{-\lambda x}$$

donde:

- $f(x)$ es la función de densidad de probabilidad.
- λ es la tasa de ocurrencia de eventos.
- x es el valor de la variable aleatoria.

Distribución Binomial

La distribución binomial modela el número de éxitos en una serie de ensayos independientes y con dos posibles resultados (éxito o fracaso). Es útil en escenarios donde se tienen múltiples intentos o experimentos. La fórmula de la distribución binomial es:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

donde:

- $P(X = k)$ es la probabilidad de obtener k éxitos en n ensayos.
- $\binom{n}{k}$ es el coeficiente binomial.
- n es el número de ensayos.
- k es el número de éxitos.
- p es la probabilidad de éxito en cada ensayo.
- $1 - p$ es la probabilidad de fracaso en cada ensayo.

Distribución Uniforme

La distribución uniforme modela una variable aleatoria que tiene la misma probabilidad de ocurrir en cualquier punto dentro de un intervalo específico. Es útil para representar variables aleatorias sin sesgo. La función de densidad de probabilidad de la distribución uniforme continua es:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{si } a \leq x \leq b \\ 0 & \text{en otro caso} \end{cases}$$

donde:

- $f(x)$ es la función de densidad de probabilidad.
- a y b son los límites inferior y superior del intervalo.

- x es el valor de la variable aleatoria.

Distribución Geométrica

La distribución geométrica modela el número de ensayos necesarios para obtener el primer éxito en una serie de ensayos independientes, cada uno con la misma probabilidad de éxito. Es útil para escenarios donde se quiere medir el tiempo o número de intentos hasta el primer éxito. La fórmula de la distribución geométrica es:

$$P(X = k) = (1 - p)^{k-1}p$$

donde:

- $P(X = k)$ es la probabilidad de que el primer éxito ocurra en el k-ésimo ensayo.
- p es la probabilidad de éxito en cada ensayo.
- $1 - p$ es la probabilidad de fracaso en cada ensayo.
- k es el número de ensayos hasta el primer éxito.

Capítulo 3

Desarrollo del Modelo de Simulación

3.1. Plan de Diseño

El objetivo principal de este proyecto es desarrollar un sistema de simulación para un entorno web, donde múltiples usuarios interactúan con servidores web y otros elementos de la infraestructura de red. La simulación se llevará a cabo en tres fases distintas, cada una incrementando la complejidad y realismo del modelo:

1. **Primer Modelo de Simulación**: En esta fase inicial, se simulará un único servidor web que maneja todas las solicitudes de los usuarios. Este modelo más sencillo y controlado sirve como base para comprender las interacciones básicas entre usuarios y servidores. Se establecerá una arquitectura simple para facilitar la evaluación inicial del rendimiento del servidor bajo diferentes cargas de trabajo.

2. **Segundo Modelo de Simulación**: En esta fase avanzada, se implementará una infraestructura más compleja que incluye un dominio principal con varios subdominios, cada uno gestionado por su propio servidor web, y un DNS que resuelve las direcciones IP de los subdominios. Este modelo es más realista y se aproxima a un entorno web distribuido real, permitiendo estudiar la distribución de carga y la resolución de nombres de dominio.

3. **Tercer Modelo de Simulación**: En este último modelo, se incorporan nuevos elementos como bases de datos, memorias caché y colas de mensajes. Este modelo combina todos los componentes en una arquitectura unificada, permitiendo una simulación integral y detallada del entorno web, incluyendo la gestión de datos, el almacenamiento en caché y la comunicación asíncrona entre servicios.

3.2. Tecnologías a Utilizar

Para el desarrollo del modelo de simulación, se seleccionaron varias tecnologías y bibliotecas que facilitan la implementación y gestión de datos:

- **Python**: El lenguaje principal para desarrollar la simulación debido a su simplicidad y robustez [21].
- **NumPy**: Utilizado para generar distribuciones probabilísticas que modelan el comportamiento de los servidores web [22].
- **JSON**: Para estructurar los datos de las solicitudes de los usuarios y permitir un fácil intercambio de información [23].
- **CSV**: Para registrar los logs de actividades en archivos CSV, facilitando la revisión y análisis de los datos generados [24].
- **OpenPyXL**: Para manejar la escritura y manipulación de archivos Excel, donde se registrarán las actividades del DNS y los servidores web [25].
- **Matplotlib**: Para la visualización de los resultados de la simulación a través de gráficos, permitiendo una mejor comprensión de los patrones de tráfico y los códigos de estado HTTP generados [26].

Estas tecnologías fueron seleccionadas por su capacidad para manejar grandes cantidades de datos, su compatibilidad con Python y su amplio uso en la comunidad de desarrolladores, así como por la facilidad de lectura y análisis de datos que proporcionan al usuario.

3.3. Desarrollo del Primer Modelo de Simulación

En el primer modelo de simulación, se implementó un sistema donde múltiples usuarios interactúan con un único servidor web. Este modelo inicial es fundamental para establecer una base sólida antes de avanzar hacia una infraestructura más compleja.

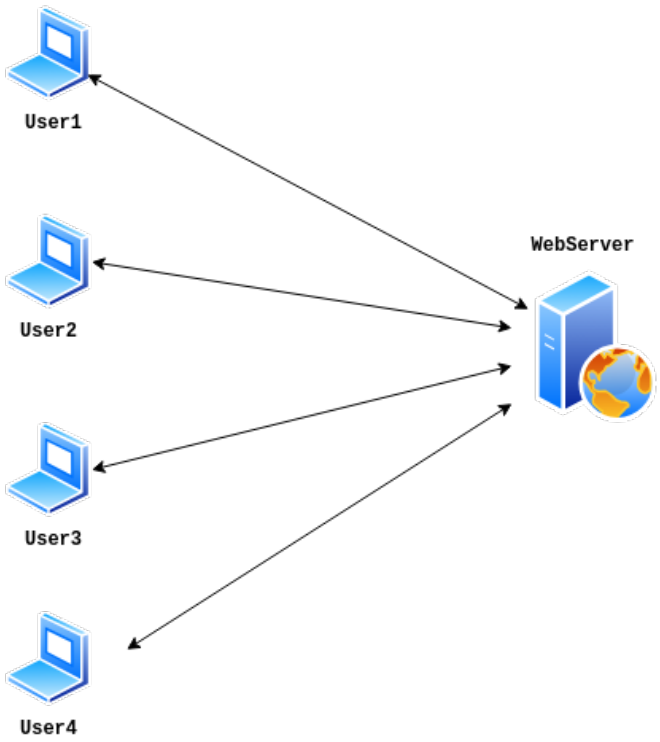


Figura 3.1: Diagrama que representa la arquitectura del primer modelo de simulación.

Vamos a crear una lista de usuarios utilizando la clase `User`, quienes realizarán peticiones a un único servidor web del tipo `WebServer`. El servidor web recibirá todas estas peticiones y las gestionará resolviéndolas con diferentes códigos HTTP. En la mayoría de los casos, estos códigos serán generados de manera aleatoria utilizando las diversas distribuciones estadísticas que se pueden consultar en la subsección 2.4.

3.3.1. Clase `User`

La clase `User` representa a un usuario que interactúa con el servidor a través de Internet. Cada usuario tiene varios atributos, como dirección IP, navegador, país, correo electrónico, nombre de usuario, estado de la cuenta y timestamps de conexión. La clase incluye métodos para generar solicitudes HTTP y actualizar el timestamp de la última conexión.

Variables de la Clase User

- **ip_address**: Dirección IP del usuario.
- **browser**: Navegador que utiliza el usuario.
- **country**: País desde donde accede el usuario.

- **email**: Correo electrónico del usuario.
- **username**: Nombre de usuario.
- **account_status**: Estado de la cuenta (por defecto .ªctive").
- **first_connection_timestamp**: Captura del tiempo de la primera conexión (por defecto, el momento de creación del objeto).
- **last_login_timestamp**: Marca de tiempo del último inicio de sesión.

Métodos de la Clase User

- **__init__**: Constructor que inicializa las variables con valores proporcionados o por defecto.
- **__repr__**: Representación en cadena de una instancia de `User`.
- **update_last_login**: Actualiza `last_login_timestamp` a la fecha y hora actuales, para representar los múltiples accesos de un mismo usuario.
- **generate_request**: Genera una solicitud HTTP simulada con detalles del usuario en formato JSON.

El código completo de la clase `User` de este primer modelo de simulación se encuentra en el Anexo .1.

3.3.2. Clase `WebServer`

La clase `WebServer` simula un servidor web que procesa las solicitudes de los usuarios. La clase incluye métodos para procesar solicitudes HTTP, agregar usuarios y registrar todas las acciones en un archivo de logging. Los códigos de estado HTTP se generan utilizando diversas distribuciones probabilísticas para modelar el comportamiento real del servidor bajo diferentes condiciones.

Variables de la Clase WebServer

- **ip_address**: Dirección IP del servidor web.
- **port**: Puerto en el cual el servidor está escuchando.
- **document_root**: Directorio raíz donde se almacenan los documentos del servidor.
- **server_name**: Nombre del servidor.
- **users**: Lista de usuarios conectados al servidor.
- **blocked_countries**: Lista de países bloqueados por el servidor.
- **start_time**: Captura del tiempo de cuando se inició el servidor.

Métodos de la Clase WebServer

- **__init__**: Constructor que inicializa las variables con valores proporcionados o por defecto.
- **__repr__**: Representación en cadena de una instancia de `WebServer`.
- **process_request**: Procesa una solicitud HTTP y devuelve un código de estado HTTP. Utiliza diferentes distribuciones estadísticas para simular errores y condiciones de red.
- **add_user**: Agrega un usuario si la solicitud tiene un código de estado exitoso (200 OK).
- **log_action**: Registra una acción incluyendo la marca de tiempo, la acción y el estado.

Además, se ha añadido la opción de indicar una semilla al servidor web para que devuelva los mismos códigos de estado a las peticiones de los usuarios, permitiendo comprobar que nuestra simulación funciona de manera consistente.

```
# Crear una instancia del servidor web con una semilla específica
server = WebServer(ip_address="192.168.1.1", port=8080, document_root="/var/www", server_name="MyServer", seed=42)
```

Listing 3.1: Ejemplo de creación de una instancia del servidor web con una semilla específica

El código completo de la clase `WebServer` se encuentra en el Anexo .2.

3.3.3. Generación de Usuarios Aleatorios

Para simular un entorno realista, se generaron usuarios de manera aleatoria. Estos usuarios tienen direcciones de correo electrónico generadas a partir de varios dominios y usan diferentes navegadores y países de origen. Las direcciones IP se generan aleatoriamente, y el nombre de usuario se define como la parte del correo electrónico antes de la arroba. Estos usuarios se utilizan en los logs de los servidores web.

```
def generate_random_email():
    prefixes = ["user", "contact", "mail", "info", "admin", "support", "sales", "service", "team", "noreply", "hello", "account", "customer", "feedback", "newsletter", "office", "marketing", "billing", "tech"]
    domains = ["example", "domain", "website", "mail", "service", "company", "business", "online", "web", "network", "enterprise", "solution", "cloud", "system", "portal", "hub", "app", "platform", "group"]
    tlds = [".com", ".net", ".org", ".io", ".biz", ".co", ".info", ".tech", ".us", ".uk", ".ai", ".app", ".dev"]
    prefix = random.choice(prefixes)
```

```
    domain = random.choice(domains)
    tld = random.choice(tlds)
    number = random.randint(1, 99999)
    random_string = ''.join(random.choices(string.
        ascii_lowercase + string.digits, k=7))
    return f"{prefix}{number}{random_string}@{domain}{tld}"
```

Listing 3.2: Función para generar cuentas de correo electrónico aleatorias

```
def generate_random_users(num_users):
    countries = ["USA", "Canada", "Mexico", "UK", "Germany", "
        France", "Japan", "China", "Spain", "Chile", "Poland", "
        Russia", "Portugal", "Argentina"]
    browsers = ["Chrome", "Firefox", "Safari", "Edge"]
    users = []

    for i in range(num_users):
        ip_address = f"{random.randint(0, 255)}.{random.randint
            (0, 255)}.{random.randint(0, 255)}.{random.randint(1,
             254)}"
        browser = random.choice(browsers)
        country = random.choice(countries)
        email = generate_random_email()
        username = email.split('@')[0]
        first_connection_timestamp = datetime.datetime.now() -
            datetime.timedelta(hours=random.randint(0, 24))
        user = User(ip_address, browser, country, email,
            username, first_connection_timestamp=
            first_connection_timestamp)
        users.append(user)

    return users
```

Listing 3.3: Función para generar usuarios aleatorios

3.3.4. Simulación de Tráfico

La simulación de tráfico implica la generación de múltiples solicitudes a un servidor web por parte de diferentes usuarios en un periodo de simulación de 24 horas. Esta función permite evaluar cómo responde el servidor bajo diferentes condiciones de carga y patrones de tráfico.

```
def simulate_traffic(server, users, num_requests):
    http_statuses = defaultdict(int)

    for _ in range(num_requests):
        user = random.choice(users)
        request_time = server.start_time - datetime.timedelta(
            hours=random.uniform(0, 24))
```

```
7            status = server.process_request(user.generate_request(),
                 request_time)
8            http_statuses[status] += 1
9
10           if status == 200:
11               server.add_user(user, request_time)
12
13       return http_statuses
```

Listing 3.4: Función para la generación de tráfico web hacia un mismo servidor

3.3.5. Uso del Ejemplo

Este ejemplo demuestra cómo se pueden generar y procesar solicitudes de usuarios, y cómo se registran las actividades en un archivo de log. Vamos a simular el funcionamiento del sistema con 1000 usuarios accediendo a un solo `WebServer`.

```
1  if __name__ == "__main__":
2      with open('primero_log.csv', mode='w', newline='') as file:
3          writer = csv.writer(file)
4          writer.writerow(['Timestamp', 'Action', 'Status'])
5
6      server = WebServer(ip_address="192.168.0.1", port=80,
           document_root="/var/www/html", server_name="MyServer")
7
8      users = generate_random_users(100)
9
10     http_statuses = simulate_traffic(server, users, 1000)
11
12     for status, count in http_statuses.items():
13         print(f"HTTP Status {status}: {count} times")
```

Listing 3.5: Ejemplo de ejecución del primer modelo de simulación

3.4. Desarrollo del Segundo Modelo de Simulación

En el segundo modelo de simulación, se ha introducido una infraestructura más compleja con múltiples subdominios y servidores web, así como un DNS para la resolución de direcciones IP. Este modelo avanza significativamente en la representación de un entorno web distribuido real.

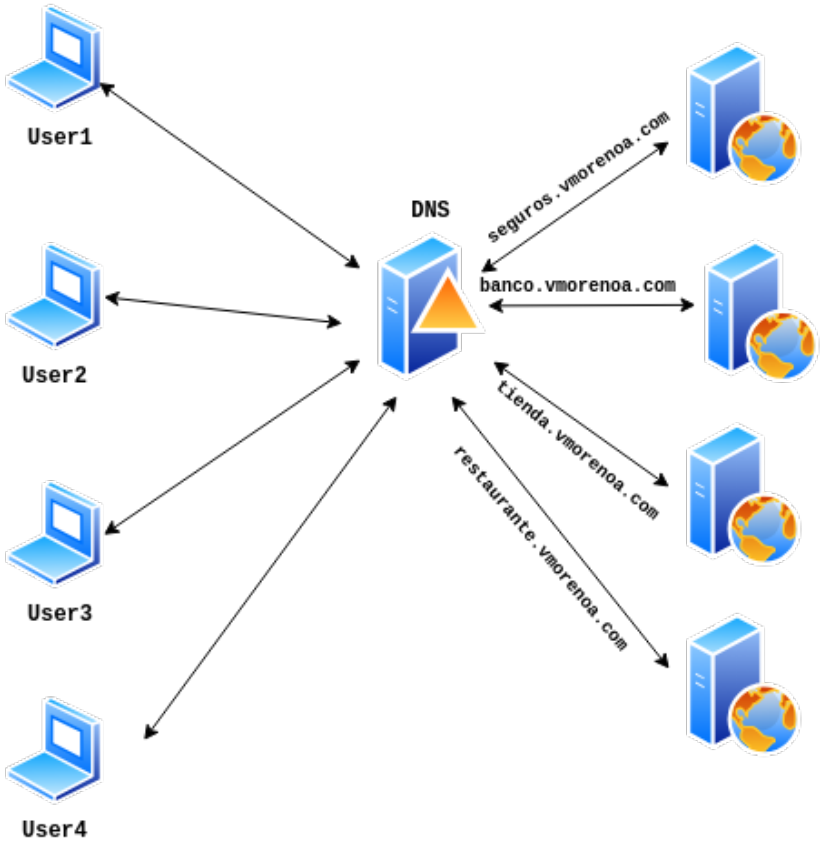


Figura 3.2: Diagrama que representa la arquitectura del segundo modelo de simulación.

3.4.1. Clase `DNS`

La clase `DNS` gestiona la resolución de subdominios a direcciones IP. Cada vez que un usuario hace una solicitud a un subdominio, el DNS verifica su tabla para devolver la dirección IP correspondiente o un error 404 si el subdominio no está registrado.

El código completo de la clase `DNS` se encuentra en el Anexo .3.

3.4.2. Simulación de Tráfico Actualizada

La función de simulación de tráfico se actualizó para incluir registros de actividad del DNS y los `WebServers` en un archivo Excel con múltiples hojas. Esta actualización permite una mejor organización y análisis de los datos generados durante la simulación.

```
1 def simulate_traffic(dns, servers, users, num_requests, workbook
    ):
2     http_statuses = defaultdict(int)
3
```

```
    dns_log = workbook.create_sheet("DNS_Log")
    dns_log.append(["Timestamp", "Action", "IP"])

    server_logs = {}
    for ip, server in servers.items():
        server_logs[ip] = workbook.create_sheet(server.
            server_name)
        server_logs[ip].append(["Timestamp", "Action", "Status"
            ])

    for _ in range(num_requests):
        user = random.choice(users)
        subdomain = random.choice(list(dns.dns_table.keys()))
        request_time = datetime.datetime.now() - datetime.
            timedelta(hours=random.uniform(0, 24))
        ip_address = dns.resolve(subdomain, request_time,
            dns_log)

        if ip_address == "404":
            http_statuses[404] += 1
            continue

        server = servers[ip_address]
        status = server.process_request(user.generate_request(),
            request_time)
        http_statuses[status] += 1

        if status == 200:
            server.add_user(user, request_time, server_logs[
                ip_address])

    return http_statuses
```

Listing 3.6: Función para simular el tráfico del segundo modelo de simulación

3.4.3. Uso del Ejemplo

En el ejemplo de uso, se inicializan el DNS y múltiples servidores web, se generan usuarios aleatorios y se simula el tráfico, registrando todas las actividades en un archivo Excel.

```
if __name__ == "__main__":
    workbook = Workbook()

    dns = DNS()

    servers = {
```

```
7           "192.168.0.1": WebServer(ip_address="192.168.0.1", port
               =80, document_root="/var/www/html", server_name="
               vmorenoa.com"),
8           "192.168.0.2": WebServer(ip_address="192.168.0.2", port
               =80, document_root="/var/www/banco", server_name="
               banco.vmorenoa.com"),
9           "192.168.0.3": WebServer(ip_address="192.168.0.3", port
               =80, document_root="/var/www/restaurante",
               server_name="restaurante.vmorenoa.com"),
10          "192.168.0.4": WebServer(ip_address="192.168.0.4", port
               =80, document_root="/var/www/seguros", server_name="
               seguros.vmorenoa.com"),
11          "192.168.0.5": WebServer(ip_address="192.168.0.5", port
               =80, document_root="/var/www/universidad",
               server_name="universidad.vmorenoa.com"),
12          "192.168.0.6": WebServer(ip_address="192.168.0.6", port
               =80, document_root="/var/www/tienda", server_name="
               tienda.vmorenoa.com"),
13          "192.168.0.7": WebServer(ip_address="192.168.0.7", port
               =80, document_root="/var/www/foro", server_name="foro
               .vmorenoa.com")
14      }
15
16      users = generate_random_users(1000)
17
18      http_statuses = simulate_traffic(dns, servers, users, 1000,
           workbook)
19
20      for status, count in http_statuses.items():
21          print(f"HTTP Status {status}: {count} times")
22
23      workbook.save("segundo_log.xlsx")
```

3.5. Desarrollo del Tercer Modelo de Simulación

El segundo modelo de simulación registraba solicitudes y respuestas básicas, pero no proporcionaba detalles sobre el rendimiento del sistema. El tercer modelo mejora esto con un monitoreo avanzado, registrando variables como la carga del servidor, el número de solicitudes recibidas y el estado de la conexión. Esto permite un seguimiento detallado del rendimiento y la identificación de cuellos de botella.

Además, se han añadido funcionalidades para la escalabilidad y resiliencia del sistema. Se implementa el uso de memoria caché y colas de mensajes para mejorar los tiempos de respuesta y gestionar mejor las solicitudes entrantes, aumentando la capacidad del sistema para manejar grandes volúmenes de tráfico.

También se ha priorizado la integración con bases de datos y sistemas de copias de seguridad. Ahora, la base de datos maneja consultas y registra operaciones exitosas y fallidas, y se incluye un sistema de copias de seguridad con una mayor capacidad, asegurando la integridad y disponibilidad de los datos.

Finalmente, se ha mejorado el monitoreo de la base de datos y el sistema de copias de seguridad, registrando actividades y capacidades restantes, permitiendo un análisis exhaustivo del rendimiento y la utilización de recursos.

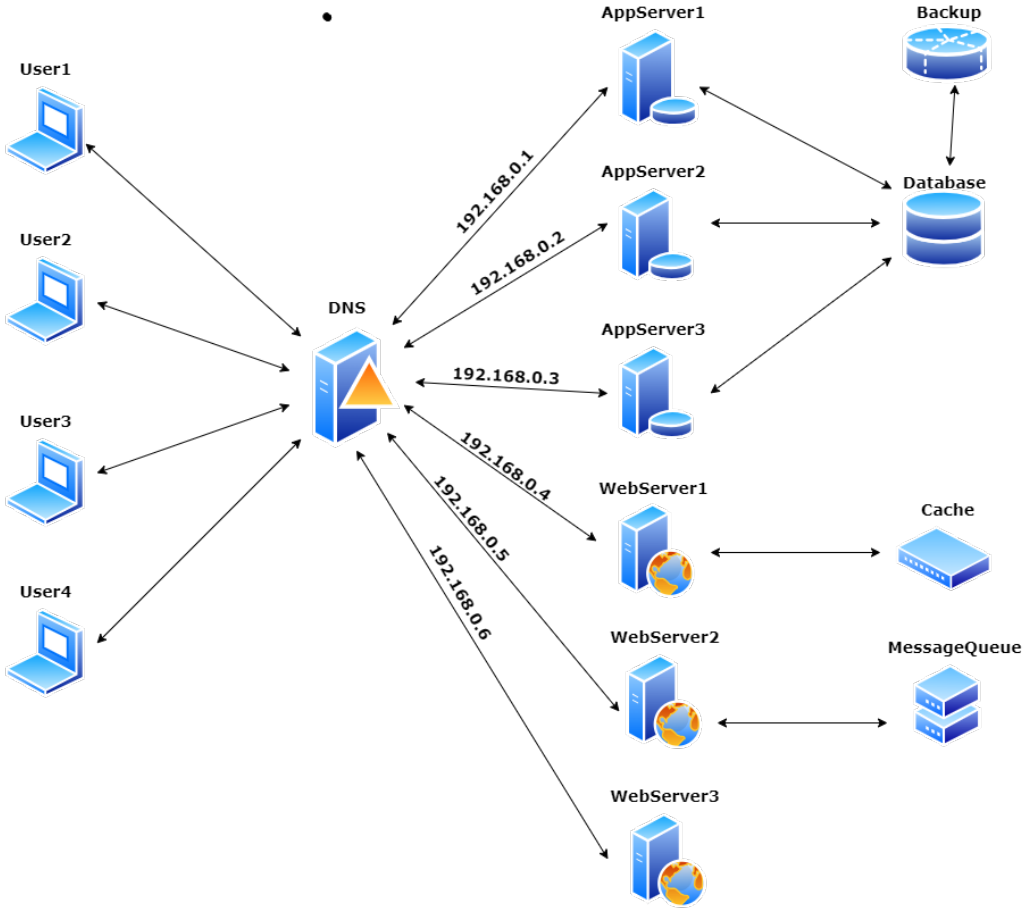


Figura 3.3: Diagrama que representa la arquitectura del tercer modelo de simulación.

3.5.1. Clase `AppServer`

- Esta clase hereda de `WebServer`.

- **database**: Objeto de base de datos asociado al servidor de aplicaciones.

Métodos:

- `app_process_request`: Procesa una solicitud HTTP y realiza una consulta a la base de datos.

El código completo de la clase `AppServer` se encuentra en el Anexo .5.

3.5.2. Clase `Cache`

- **capacity**: Capacidad máxima de la caché.
- **data**: Diccionario para almacenar los datos en caché.

Métodos:

- `get`: Recupera un elemento de la caché.
- `set`: Añade un elemento a la caché.
- `log_action`: Registra una acción en el log con su correspondiente marca de tiempo y estado.

El código completo de la clase `Cache` se encuentra en el Anexo .6.

3.5.3. Clase `MessageQueue`

- **capacity**: Capacidad máxima de la cola de mensajes.
- **queue**: Lista de mensajes en la cola.

Métodos:

- `enqueue`: Añade un mensaje a la cola.
- `dequeue`: Recupera un mensaje de la cola.
- `log_action`: Registra una acción en el log con su correspondiente marca de tiempo y estado.

El código completo de la clase `MessageQueue` se encuentra en el Anexo .7.

3.5.4. Clase `CachedWebServer`

- Esta clase hereda de `WebServer`.
- **cache**: Objeto de caché asociado al servidor.

Métodos:

- `cache_process_request`: Procesa una solicitud HTTP utilizando la caché.

El código completo de la clase `CachedWebServer` se encuentra en el Anexo .8.

3.5.5. Clase `QueuedWebServer`

- Esta clase hereda de `WebServer`.
- **message_queue**: Objeto de cola de mensajes asociado al servidor.

Métodos:

- `queue_process_request`: Procesa una solicitud HTTP utilizando la cola de mensajes.

El código completo de la clase `QueuedWebServer` se encuentra en el Anexo .9.

3.5.6. Clase `DatabaseBackup`

- **capacity**: Capacidad máxima del backup de la base de datos.
- **data**: Diccionario para almacenar los datos del backup.

Métodos:

- `backup`: Realiza un backup de los datos.
- `log_action`: Registra una acción en el log con su correspondiente marca de tiempo y estado.

El código completo de la clase `DatabaseBackup` se encuentra en el Anexo .10.

3.5.7. Clase `Database`

- **data**: Diccionario para almacenar los datos de la base de datos.
- **capacity**: Capacidad máxima de la base de datos.
- **failure_rate**: Tasa de fallos de la base de datos.
- **backup**: Objeto de backup asociado a la base de datos.

Métodos:

- `query`: Realiza una consulta a la base de datos.
- `log_action`: Registra una acción en el log con su correspondiente marca de tiempo y estado.
- `reset_capacity`: Reinicia la capacidad de la base de datos.

El código completo de la clase `Database` se encuentra en el Anexo .11.

3.5.8. Clase `DNS`

- **dns_table**: Diccionario que mapea nombres de dominio a direcciones IP.

Métodos:

- `resolve`: Resuelve un nombre de dominio a una dirección IP.

El código completo de la clase `DNS` se encuentra en el Anexo .12.

3.5.9. Simulación de Tráfico Actualizada

```
def simulate_traffic(dns, servers, users, num_requests, workbook):
    http_statuses = defaultdict(int)
    dns_log = workbook.create_sheet("DNS_Log")
    dns_log.append(["Timestamp", "Action", "IP"])

    db_log = workbook.create_sheet("DB_Log")
    db_log.append(["Timestamp", "Action", "Status", "Capacity"])

    backup_log = workbook.create_sheet("Backup_Log")
    backup_log.append(["Timestamp", "Action", "Status", "Backup Size"])

    cache_log = workbook.create_sheet("Cache_Log")
    cache_log.append(["Timestamp", "Action", "Status", "Cache Size"])

    queue_log = workbook.create_sheet("Queue_Log")
    queue_log.append(["Timestamp", "Action", "Status", "Queue Size"])

    server_logs = {}
    for ip, server in servers.items():
        server_logs[ip] = workbook.create_sheet(server.server_name)
        server_logs[ip].append(["Timestamp", "Action", "Status", "Load", "Requests Received", "Connection Status"])

    for _ in range(num_requests):
        user = random.choice(users)
        subdomain = random.choice(list(dns.dns_table.keys()))
        request_time = datetime.datetime.now() - datetime.timedelta(hours=random.uniform(0, 24))
        ip_address = dns.resolve(subdomain, request_time, dns_log)

        if ip_address == "404":
            http_statuses[404] += 1
            continue

        server = servers[ip_address]
        if isinstance(server, AppServer):
            status = server.app_process_request(user.generate_request(), request_time, db_log, backup_log)
        elif isinstance(server, CachedWebServer):
```

```
37              status = server.cache_process_request(user.
                    generate_request(), request_time, cache_log)
38          elif isinstance(server, QueuedWebServer):
39              status = server.queue_process_request(user.
                    generate_request(), request_time, queue_log)
40          else:
41              status = server.process_request(user.
                    generate_request(), request_time)
42          http_statuses[status] += 1
43
44          if status == 200:
45              server.add_user(user, request_time, server_logs[
                    ip_address])
46
47          log_server_state(request_time, server, server_logs[
                ip_address])
48
49      return http_statuses
```

3.5.10. Uso del Ejemplo

En este ejemplo de uso, se inicializan el DNS, múltiples servidores web, usuarios aleatorios y se simula el tráfico, registrando todas las actividades en un archivo Excel.

```
1  if __name__ == "__main__":
2      workbook = Workbook()
3
4      dns = DNS()
5
6      servers = {
7          "192.168.0.1": WebServer(ip_address="192.168.0.1", port
               =80, document_root="/var/www/html", server_name="
               vmorenoa.com"),
8          "192.168.0.2": WebServer(ip_address="192.168.0.2", port
               =80, document_root="/var/www/banco", server_name="
               banco.vmorenoa.com"),
9          "192.168.0.3": WebServer(ip_address="192.168.0.3", port
               =80, document_root="/var/www/restaurante",
               server_name="restaurante.vmorenoa.com"),
10         "192.168.0.4": WebServer(ip_address="192.168.0.4", port
               =80, document_root="/var/www/seguros", server_name="
               seguros.vmorenoa.com"),
11         "192.168.0.5": WebServer(ip_address="192.168.0.5", port
               =80, document_root="/var/www/universidad",
               server_name="universidad.vmorenoa.com"),
12         "192.168.0.6": WebServer(ip_address="192.168.0.6", port
```

```
            =80, document_root="/var/www/tienda", server_name="
            tienda.vmorenoa.com"),
13          "192.168.0.7": WebServer(ip_address="192.168.0.7", port
            =80, document_root="/var/www/foro", server_name="foro
            .vmorenoa.com")
14      }
15
16      users = generate_random_users(1000)
17
18      http_statuses = simulate_traffic(dns, servers, users, 1000,
            workbook)
19
20      for status, count in http_statuses.items():
21          print(f"HTTP Status {status}: {count} times")
22
23      workbook.save("tercero_log.xlsx")
```

Capítulo 4

Resultados de la Simulación

4.1. Resultados

El análisis de los resultados obtenidos del tercer modelo de simulación revela varias mejoras significativas respecto al segundo modelo, centradas en el monitoreo avanzado, la escalabilidad, la resiliencia y la integración con bases de datos y sistemas de copias de seguridad.

En definitiva, podemos concluir que la simulación realizada ha sido un éxito rotundo. La figura 4.1 proporciona una clara evidencia de que el `AppServer` ha gestionado eficazmente una carga de trabajo variable a lo largo del tiempo durante un periodo de ejecución de 24 horas. Este gráfico muestra cómo la carga del servidor fluctuó, reflejando el comportamiento típico de un entorno de producción real, donde la demanda de recursos y las solicitudes de usuarios pueden variar considerablemente en diferentes momentos del día.

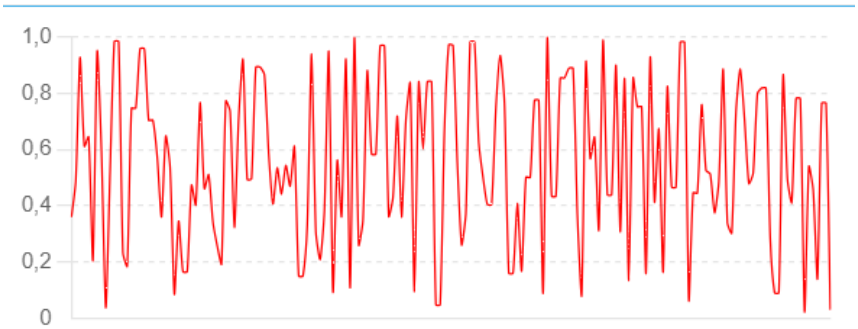


Figura 4.1: Gráfica sobre la carga de trabajo del AppServer principal del Tercer Modelo de Simulación

El monitoreo continuo y el registro detallado de la carga del `AppServer` indican que, a pesar de las variaciones en la carga, el servidor mantuvo su operatividad sin interrupciones. No se registraron tiempos de inactividad, lo que demuestra la robustez y la resiliencia del sistema frente a fluctuaciones en la demanda. Esta capacidad de mantenerse operativo bajo condiciones variables es crucial para

cualquier sistema de producción, ya que asegura la disponibilidad constante del servicio para los usuarios.

Además, la simulación ha replicado con precisión las condiciones de un entorno real, incluyendo la gestión de solicitudes concurrentes, la utilización de recursos y la respuesta a diferentes patrones de tráfico. La capacidad del `AppServer` para manejar estos factores sin fallos es un indicador claro de que el modelo de simulación desarrollado es fiable y puede ser utilizado para predecir el comportamiento del sistema en escenarios reales.

Este nivel de detalle y realismo en la simulación no solo valida la eficacia del `AppServer` en términos de rendimiento y estabilidad, sino que también proporciona valiosos insights para futuras optimizaciones y mejoras del sistema. La representación gráfica de la carga del servidor, como se observa en la figura 4.1, confirma que la simulación ha capturado adecuadamente las dinámicas de un entorno de producción, ofreciendo una herramienta poderosa para el análisis y la planificación.

Asimismo, podemos afirmar que el rendimiento del `AppServer` en términos de las peticiones realizadas a la base de datos ha sido notablemente exitoso. La figura 4.2 muestra el número de consultas a la base de datos realizadas por el `AppServer` durante el mismo periodo de 24 horas. Este gráfico evidencia cómo el servidor ha manejado un volumen considerable de consultas, reflejando un comportamiento consistente y eficiente en la gestión de datos.

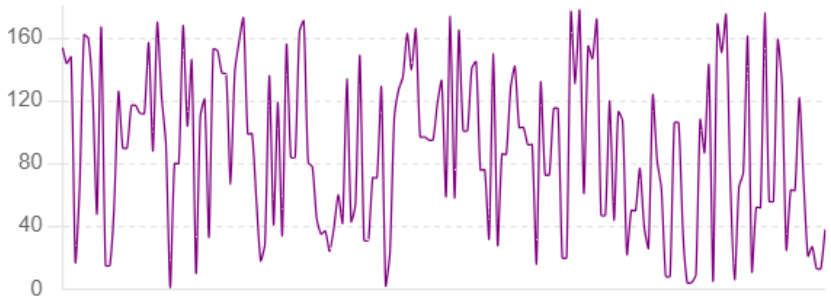


Figura 4.2: Gráfica sobre las peticiones realizadas por el AppServer principal del Tercer Modelo de Simulación

A lo largo del periodo de simulación, el `AppServer` no solo mantuvo una tasa constante de consultas exitosas, sino que también demostró su capacidad para adaptarse a picos en la demanda sin comprometer el rendimiento ni la integridad de los datos. Cada consulta fue procesada correctamente, lo que sugiere que el sistema de base de datos estaba bien configurado y optimizado para manejar cargas variables de trabajo.

La ausencia de errores o fallos críticos en las consultas a la base de datos durante el periodo de simulación indica que tanto el `AppServer` como el sistema de base de datos operaron de manera coordinada y eficaz. Esta robustez es esencial para garantizar que las aplicaciones basadas en datos puedan ofrecer un

servicio continuo y fiable a los usuarios finales.

La figura 4.2 proporciona una visión clara y detallada del volumen de consultas y su distribución a lo largo del tiempo, validando la eficacia del `AppServer` en términos de acceso y manipulación de datos. Esta representación gráfica confirma que el sistema es capaz de mantener un rendimiento constante y fiable bajo diversas condiciones de carga, lo cual es un indicador de éxito significativo para el proyecto.

En resumen, la simulación ha demostrado que el `AppServer` puede manejar de manera eficiente y eficaz un alto volumen de peticiones a la base de datos, manteniendo la integridad y la disponibilidad de los datos en todo momento. Este éxito no solo valida el diseño y la implementación del sistema, sino que también proporciona una base sólida para futuras optimizaciones y mejoras, asegurando que el sistema puede escalar y adaptarse a las necesidades cambiantes de los usuarios.

4.1.1. Monitoreo Avanzado

En el segundo modelo, el monitoreo se limitaba al registro básico de solicitudes y respuestas, con una atención principal en los códigos de estado HTTP y la cantidad de solicitudes procesadas. Este enfoque no proporcionaba una visión completa del rendimiento del sistema ni del estado de los servidores.

El tercer modelo introduce un sistema de monitoreo más avanzado, registrando variables adicionales como la carga del servidor, el número de solicitudes recibidas y el estado de la conexión. Esta mejora permite un seguimiento más detallado del rendimiento del sistema, facilitando la identificación de cuellos de botella y áreas de mejora.

4.1.2. Escalabilidad y Resiliencia

El segundo modelo carecía de mecanismos específicos para mejorar la escalabilidad y la resiliencia del sistema. No se incluían técnicas de balanceo de carga ni de gestión avanzada de la resiliencia.

En contraste, el tercer modelo incorpora funcionalidades para simular el uso de memoria caché y colas de mensajes. Un servidor web está conectado a una memoria caché, lo que reduce la carga en los servidores principales y mejora los tiempos de respuesta al almacenar temporalmente datos solicitados con frecuencia. Otro servidor web utiliza una cola de mensajes para gestionar mejor las solicitudes entrantes y mantener un rendimiento óptimo bajo alta demanda. Estas mejoras aumentan la capacidad del sistema para manejar mayores volúmenes de tráfico y mejorar su resiliencia frente a fallos.

4.1.3. Integración con Bases de Datos y Sistemas de Copias de Seguridad

En el segundo modelo, la integración con bases de datos era limitada y no se implementaban mecanismos de respaldo.

El tercer modelo mejora significativamente en este aspecto, integrando una base de datos que maneja consultas y registra tanto operaciones exitosas como fallidas. Además, se ha añadido un sistema de copias de seguridad con una capacidad mayor, permitiendo almacenar datos críticos de manera segura y recuperarlos en caso de fallo de la base de datos principal. Esta integración asegura la integridad y disponibilidad de los datos, incluso en escenarios de fallo.

4.1.4. Memoria Caché y Cola de Mensajes

La memoria caché y la cola de mensajes son nuevas adiciones en el tercer modelo. La memoria caché reduce la carga en los servidores principales al almacenar temporalmente datos solicitados frecuentemente, mejorando los tiempos de respuesta. La cola de mensajes ayuda a gestionar mejor las solicitudes entrantes, especialmente en condiciones de alta demanda, evitando que el servidor principal se sature con solicitudes. Este componente de nuestra arquitectura es candidato a ser mejorado debido a que en nuestro modelo de simulación, unicamente recibe mensajes y los almacena adecuadamente de uno en uno, pero no es capaz de liberar mensajes. Esta funcionalidad necesria en este componente.

4.1.5. Impacto de las Distribuciones Estadísticas

Las distribuciones estadísticas aplicadas en el modelo de simulación han tenido un impacto significativo en los resultados obtenidos. Se utilizaron diferentes distribuciones para simular diversos aspectos del sistema, como la carga del servidor, las solicitudes recibidas y la capacidad de la base de datos. Estas distribuciones permitieron una representación realista del comportamiento del sistema bajo diferentes condiciones.

4.1.6. Conclusión

En resumen, el tercer modelo de simulación implementa mejoras significativas en el monitoreo avanzado, la escalabilidad, la resiliencia y la integración con bases de datos y sistemas de copias de seguridad. Estas mejoras no solo permiten un seguimiento más detallado del rendimiento del sistema, sino que también aseguran una mayor capacidad para manejar grandes volúmenes de tráfico y mantener la integridad y disponibilidad de los datos. Las distribuciones estadísticas utilizadas han proporcionado una representación realista del comportamiento del sistema bajo diferentes condiciones, permitiendo un análisis exhaustivo y preciso del rendimiento y la resiliencia del sistema.

Capítulo 5

Mejoras del modelo de simulación

5.1. Introducción

En este documento se presentan una serie de mejoras propuestas para el tercer modelo de simulación de servidores. Estas mejoras están orientadas a optimizar el rendimiento, mejorar la claridad y estructura del código, y añadir nuevas funcionalidades para una simulación más realista y completa.

5.2. Optimización del Rendimiento

5.2.1. Estructuras de Datos Eficientes

Reemplazar las listas con estructuras de datos más eficientes puede mejorar significativamente el rendimiento. Por ejemplo, utilizar `deque` para operaciones de cola y `set` para búsquedas rápidas.

5.2.2. Mejoras en la Caché

Implementar una política de reemplazo de caché más avanzada como LRU (Least Recently Used) en lugar de una simple FIFO. Esto puede mejorar el rendimiento al mantener los datos más relevantes en la caché.

5.2.3. Paralelización

Para manejar múltiples solicitudes de manera concurrente, se puede utilizar `threading` o `multiprocessing`. Esto permitirá que el sistema procese más solicitudes en menos tiempo, aprovechando mejor los recursos del hardware.

5.3. Mejora de la Estructura y Claridad del Código

5.3.1. Modularización

Dividir el código en módulos más pequeños y manejables. Por ejemplo, crear módulos separados para `User`, `Server`, `Cache`, `Queue`, `Database` y `DNS`. Esto hará que el código sea más fácil de mantener y entender.

5.3.2. Documentación

Añadir `docstrings` y comentarios detallados para cada clase y método. Esto ayudará a otros desarrolladores (y a ti mismo en el futuro) a entender el propósito y funcionamiento de cada parte del código.

5.3.3. Tipado Estático

Utilizar anotaciones de tipo en Python (`type hints`) para mejorar la legibilidad y facilitar la detección de errores durante el desarrollo. Esto es particularmente útil en proyectos grandes y colaborativos.

5.4. Nuevas Funcionalidades

5.4.1. Monitorización en Tiempo Real

Implementar una interfaz para monitorizar el estado del sistema en tiempo real. Esto podría incluir el uso de herramientas de visualización o paneles de control (dashboards) para mostrar métricas clave como la carga del servidor, el número de solicitudes procesadas y el estado de la caché y las colas de mensajes.

5.4.2. Simulación de Fallos

Añadir la capacidad de simular fallos en los servidores y la base de datos para probar la robustez del sistema. Esto puede incluir fallos de hardware, interrupciones de red y errores en el software.

5.4.3. Escalabilidad

Implementar la capacidad de escalar dinámicamente el número de servidores en función de la carga. Esto puede lograrse mediante la implementación de servidores adicionales en respuesta a aumentos en la demanda, y desactivación de servidores cuando la demanda disminuye.

5.4.4. Análisis de Logs

Añadir funcionalidad para analizar los logs y generar reportes de rendimiento y errores. Esto puede incluir el uso de herramientas de análisis de logs y generación de informes para identificar patrones y áreas de mejora.

5.5. Conclusión

Las mejoras propuestas están diseñadas para hacer que el modelo de simulación de servidores sea más eficiente, claro y funcional. Implementar estas mejoras no solo optimizará el rendimiento del sistema, sino que también mejorará la mantenibilidad del código y la capacidad del sistema para manejar situaciones del mundo real.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Objetivos del Trabajo Fin de Grado

Desde el inicio de este proyecto, se establecieron varios objetivos clave que debían ser alcanzados para cumplir con los requisitos del Trabajo Fin de Grado:

1. **Desarrollar un modelo de simulación de un sistema de almacenamiento de información**: Se desarrollaron prototipos iniciales que incorporaron configuraciones básicas de sistemas de ficheros, unidades operativas y redes. Aunque no se logró un modelo completamente detallado, se establecieron las bases para futuras mejoras.

2. **Documentar las variables y establecer objetivos de simulación**: Se identificaron y documentaron las variables más críticas, como la carga del servidor y el número de solicitudes, estableciendo objetivos claros para las simulaciones. Sin embargo, algunas áreas quedaron pendientes para futuras iteraciones.

3. **Definir las estructuras de datos y patrones de tareas**: Se especificaron estructuras de datos básicas y patrones de tareas, como la llegada de solicitudes y el servicio de las mismas. Esta especificación inicial proporcionó una base para simular escenarios realistas.

4. **Implementar un sistema de registro (log) y un dashboard de visualización**: Se desarrolló un sistema de logs para recopilar datos de las simulaciones. Aunque no se implementó un dashboard interactivo completo, se avanzó en la visualización de datos mediante gráficos con Matplotlib.

5. **Simular y evaluar el comportamiento del sistema bajo diversas condiciones**: Se realizaron simulaciones para analizar el comportamiento del sistema bajo diferentes condiciones de carga y disponibilidad, proporcionando una comprensión básica del rendimiento y las limitaciones del sistema.

6. **Validar y analizar los resultados de la simulación**: Los resultados de las simulaciones se compararon con expectativas teóricas y se realizó un análisis preliminar. No se validaron exhaustivamente con datos reales, pero

se obtuvieron insights valiosos para futuras optimizaciones.

A lo largo del desarrollo del proyecto, se han logrado avances significativos en cada uno de estos objetivos, proporcionando una base sólida para futuras mejoras y ampliaciones.

6.2. Líneas Futuras

A pesar de las limitaciones de tiempo, el proyecto ha sentado una base sólida para futuras mejoras. Algunas de las posibles líneas de desarrollo incluyen:

- **Escalabilidad:** Mejorar la capacidad del sistema para manejar un mayor número de usuarios y peticiones mediante la implementación de técnicas de balanceo de carga y la optimización del rendimiento del servidor. Esto podría incluir la incorporación de más servidores y la distribución de la carga de manera más eficiente.
- **Copias de Seguridad:** Ampliar los mecanismos de respaldo mediante la implementación de diferentes niveles de RAID (Redundant Array of Independent Disks) para aumentar la redundancia y la fiabilidad de los datos almacenados. Esto proporcionaría una mayor protección contra fallos de hardware y pérdida de datos.
- **Consumo Energético:** Optimizar el sistema para reducir el consumo energético, implementando técnicas de eficiencia energética tanto en el hardware como en el software. Esto no solo reduciría los costos operativos sino que también contribuiría a la sostenibilidad ambiental.
- **Automatización y DevOps:** Implementar prácticas de DevOps para automatizar los procesos de despliegue y gestión del sistema. Esto mejoraría la eficiencia operativa y reduciría el tiempo de inactividad, facilitando una respuesta más rápida a los cambios y problemas.

6.3. Evaluación del Trabajo

Durante el desarrollo del proyecto, se siguió una metodología establecida por la universidad, la cual consistia en entregar un Plan de Trabajo y una Memoria de Seguimiento a lo largo del semestre. La documentación y aprendizaje se realizaron mediante:

- **Revisión Bibliográfica:** Consulta de artículos académicos, libros y recursos en línea sobre simulación de sistemas, programación en Python y arquitecturas web.
- **Implementación Práctica:** Desarrollo de código y pruebas iterativas para validar las funcionalidades del sistema, utilizando herramientas como NumPy, Matplotlib, y OpenPyXL.
- **Aprendizaje Continuo:** Adquisición de habilidades en programación y comprensión de sistemas complejos, enfocándose en cómo los diferentes com-

ponentes interactúan y afectan el rendimiento general del sistema.

Personalmente, ya tenia conocimientos de programción con Python, sobretodo en el ámbito de creación de APIs. He mejorado mis conocimientos de Python, he acomprendido mejor las arquitecturas de almacenamiento de información y a manejar grandes volúmenes de datos de manera eficiente, dentro del perimetro de las herramientas sencillas de las que disponemos. La metodología empleada permitió un seguimiento detallado del progreso del proyecto y la identificación de áreas de mejora a lo largo del desarrollo.

6.4. Objetivos de Desarrollo Sostenible

La Universidad Politécnica de Madrid (véase en [27]) esta comprometida con la divulgación sobre la estrecha unión entre la ingeniería y los Objetivos de Desarrollo Sostenible (ODS) [28] de la ONU, especialmente en proyectos como este, que en caso de aplicarse en gran escala, son las pequeñas optimizaciones las que marcan los grandes cambios, por ejemplo:

- **ODS 9 - Industria, Innovación e Infraestructura:** La simulación de sistemas complejos y el desarrollo de soluciones eficientes para la gestión de información pueden impulsar la innovación y mejorar las infraestructuras digitales, fundamentales para el desarrollo sostenible de la industria.
- **ODS 12 - Producción y Consumo Responsables:** Al optimizar el consumo energético y mejorar la eficiencia de los sistemas de almacenamiento, este trabajo contribuye a la reducción del impacto ambiental y promueve prácticas de producción y consumo más sostenibles.
- **ODS 17 - Alianzas para Lograr los Objetivos:** La creación de sistemas de almacenamiento de información robustos y eficientes puede facilitar la colaboración y el intercambio de información entre diversas organizaciones y sectores, apoyando las alianzas necesarias para alcanzar los ODS.

En conclusión, aunque no se han alcanzado todos los objetivos iniciales debido a limitaciones de tiempo, este proyecto ha proporcionado una experiencia de aprendizaje valiosa y ha sentado las bases para futuras mejoras y exploraciones en el campo de las arquitecturas de almacenamiento de información. Agradezco la oportunidad de haber trabajado en este proyecto y confío en que las lecciones aprendidas serán de gran utilidad en mis futuras iniciativas profesionales.

Bibliografía

[1] J. Smith, «HDD Technology and Applications», *Journal of Data Storage*, 2022.

[2] B. Williams, «Comparative Analysis of HDD Brands», *Data Management Review*, 2021.

[3] C. Brown, «SSDs in Modern Computing», *Journal of Advanced Computing*, 2022.

[4] E. Martinez, «Portable Storage Devices: USB and SD Cards», *Electronics Weekly*, 2023.

[5] Amazon Web Services, *Amazon Web Services Overview*, AWS Whitepapers, 2023.

[6] Microsoft, *Microsoft Azure Storage Solutions*, Microsoft Docs, 2022.

[7] Google, *Google Cloud Storage*, Google Cloud Documentation, 2023.

[8] IBM, *IBM Cloud Object Storage*, IBM Whitepapers, 2023.

[9] Oracle, *Oracle Cloud Infrastructure*, Oracle Whitepapers, 2023.

[10] Synology, *Synology DiskStation Features*, Synology Documentation, 2022.

[11] TechRadar, *QNAP NAS Review*, 2023.

[12] NetApp, *NetApp FAS Series Overview*, NetApp Whitepapers, 2022.

[13] Dell, *Dell EMC Isilon Specifications*, Dell Documentation, 2022.

[14] Buffalo Technology, *Buffalo TeraStation Capabilities*, 2023.

[15] J. Cui, Z. Li, P. He, Z. Gong y J. Dong, «Electromechanical transient modeling of energy storage based on virtual synchronous machine technology», *Archives of Electrical Engineering*, vol. 71, n.º 3, págs. 581-599, 2022.

[16] C. Hemme y W. van Berk, «Hydrogeochemical Modeling to Identify Potential Risks of Underground Hydrogen Storage in Depleted Gas Fields», *Applied Sciences*, vol. 8, n.º 11, pág. 2282, 2018.

[17] X. Chen, L. Shen, Z. Sha et al., «A Survey of Multi-Space Techniques in Spatio-Temporal Simulation Data Visualization», *Visual Informatics*, vol. 3, n.º 3, págs. 129-139, 2019.

[18] K. Vimal, A. Singh y H. Bhasin, «An Overview of Apache Hadoop Ecosystem for Big Data Analytics», *International Journal of Information Technology and Computer Science*, vol. 9, n.º 6, págs. 57-69, 2017.

[19] M. Li y J. Doe, «In-Memory Data Management for Big Data Processing: Redis vs. Memcached», *Journal of Data Science and Engineering*, vol. 5, n.º 4, págs. 256-268, 2020.

[20] L. White y E. Harris, «Messaging Systems in Modern Application Architectures: RabbitMQ vs. Apache Kafka», *Journal of Software Engineering and Applications*, vol. 13, n.º 4, págs. 198-205, 2022.

[21] Python Software Foundation, *Python*, Versión 3.10.4, 2023. dirección: https://www.python.org/ (visitado 26-05-2024).

[22] T. E. O. et al., *NumPy*, Versión 1.22.4, 2023. dirección: https://numpy.org/ (visitado 26-05-2024).

[23] D. Crockford, *JSON*, Versión estándar, 2023. dirección: https://www.json.org/ (visitado 26-05-2024).

[24] Python Software Foundation, *CSV Module*, Módulo estándar de Python, 2023. dirección: https://docs.python.org/3/library/csv.html (visitado 26-05-2024).

[25] E. Gazoni y C. Clark, *OpenPyXL*, Versión 3.0.9, 2023. dirección: https://openpyxl.readthedocs.io/ (visitado 26-05-2024).

[26] J. D. H. et al., *Matplotlib*, Versión 3.5.1, 2023. dirección: https://matplotlib.org/ (visitado 26-05-2024).

[27] Universidad Politécnica de Madrid, *La UPM y los ODS*, 2024. dirección: https://sostenibilidad.upm.es/inicio/la-upm-y-los-ods/.

[28] Universidad Politécnica de Madrid, *Conoce los Objetivos de Desarrollo Sostenible*, 2024. dirección: https://sostenibilidad.upm.es/conoce-los-objetivos-de-desarrollo-sostenible/.

Anexos

.1. Clase User del Primer Modelo de Simulación

```python
class User:
    def __init__(self, ip_address, browser, country, email, username, account_status="active",
                 first_connection_timestamp=None, last_login_timestamp=None):
        self.ip_address = ip_address
        self.browser = browser
        self.country = country
        self.email = email
        self.username = username
        self.account_status = account_status
        self.first_connection_timestamp = first_connection_timestamp or datetime.datetime.now()
        self.last_login_timestamp = last_login_timestamp

    def __repr__(self):
        return (
            f"User(ip_address='{self.ip_address}', browser='{self.browser}', "
            f"country='{self.country}', email='{self.email}', "
            f"username='{self.username}', account_status='{self.account_status}', "
            f"first_connection_timestamp='{self.first_connection_timestamp}', "
            f"last_login_timestamp='{self.last_login_timestamp}')"
            )

    def update_last_login(self):
        self.last_login_timestamp = datetime.datetime.now()

    def generate_request(self):
        # Generate a random HTTP request.
        timestamp = datetime.datetime.now().isoformat()
        request_body = {
            "ip_address": self.ip_address,
            "browser": self.browser,
            "country": self.country,
            "email": self.email,
            "username": self.username,
            "timestamp": timestamp
        }
        return json.dumps(request_body)
```

.2. Clase `WebServer` del Primer Modelo de Simulación

```
class WebServer:
    def __init__(self, ip_address, port, document_root,
        server_name, seed=None):
        self.ip_address = ip_address
        self.port = port
        self.document_root = document_root
        self.server_name = server_name
        self.users = []
        self.blocked_countries = ["Russia", "Chile", "Germany",
            "UK"]
        self.start_time = datetime.datetime.now()

        # Set the seed for reproducibility
        if seed is not None:
            np.random.seed(seed)
            random.seed(seed)

    def __repr__(self):
        return (f"WebServer(ip_address='{self.ip_address}', port
            ={self.port}, document_root='{self.document_root}', "
                f"server_name='{self.server_name}')")

    def process_request(self, request, request_time):
        """
        Process an HTTP request and return an HTTP status code.
        """
        request_data = json.loads(request)
        country = request_data.get("country")
        browser = request_data.get("browser")
        username = request_data.get("username")

        # 403 Forbidden for blocked countries
        if country in self.blocked_countries:
            return 403  # Forbidden

        # 408 Request Timeout if browser is outdated or blocked
        outdated_browsers = ["Internet Explorer", "Netscape", "
            Opera", "Firefox"]
        if browser in outdated_browsers:
            return 408  # Request Timeout

        # 429 Too Many Requests using a Poisson distribution
        if np.random.poisson(1) > 2:
            return 429  # Too Many Requests

```

```
42          # 500 Internal Server Error using a Normal distribution
43          if np.random.normal(0, 1) > 2:
44              return 500  # Internal Server Error
45
46          # 503 Service Unavailable using an Exponential
                distribution
47          if np.random.exponential(1) > 3:
48              return 503  # Service Unavailable
49
50          # 400 Bad Request using a Binomial distribution
51          if np.random.binomial(1, 0.1) == 1:
52              return 400  # Bad Request
53
54          # 401 Unauthorized using a Uniform distribution
55          if np.random.uniform(0, 1) < 0.05:
56              return 401  # Unauthorized
57
58          # 404 Not Found using a Geometric distribution
59          if np.random.geometric(0.1) == 1:
60              return 404  # Not Found
61
62          # 200 OK for successful requests
63          return 200  # OK
64
65      def add_user(self, user, request_time):
66          """
67          Add a user if the user has a successful status code.
68          """
69          request = user.generate_request()
70          status = self.process_request(request, request_time)
71          if status == 200:
72              self.users.append(user)
73              action = f"User {user.username} added."
74          else:
75              action = f"User {user.username} rejected with status
                    {status}."
76
77          self.log_action(request_time, action, status)
78
79      def log_action(self, timestamp, action, status):
80          """
81          Log an action to a CSV file.
82          """
83          with open('webserver_log.csv', mode='a', newline='') as
                file:
84              writer = csv.writer(file)
85              writer.writerow([timestamp, action, status])
```

.3. Clase DNS

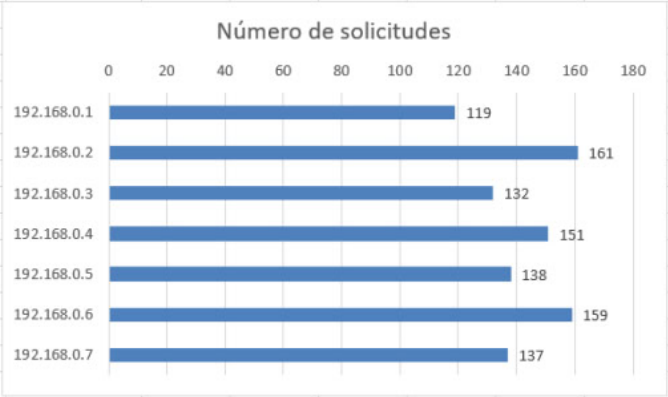
```
class DNS:
    def __init__(self):
        self.dns_table = {
            "vmorenoa.com": "192.168.0.1",
            "banco.vmorenoa.com": "192.168.0.2",
            "restaurante.vmorenoa.com": "192.168.0.3",
            "seguros.vmorenoa.com": "192.168.0.4",
            "universidad.vmorenoa.com": "192.168.0.5",
            "tienda.vmorenoa.com": "192.168.0.6",
            "foro.vmorenoa.com": "192.168.0.7"
        }

    def resolve(self, subdomain, request_time, log_sheet):
        ip = self.dns_table.get(subdomain, "404")
        action = f"Resolved {subdomain} to {ip}" if ip != "404"
           else f"Failed to resolve {subdomain}"
        log_sheet.append([request_time, action, ip])
        return ip
        return ip
```

.4. Logs del Segundo Modelo de Simulación

.4.1. Log del DNS

Timestamp	Action	IP
2024-06-02 6:56:18	Resolved tienda.vmorenoa.com to 192.168.0.6	192.168.0.6
2024-06-02 6:58:14	Resolved restaurante.vmorenoa.com to 192.168.0.3	192.168.0.3
2024-06-02 7:00:36	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
2024-06-02 7:01:47	Resolved restaurante.vmorenoa.com to 192.168.0.3	192.168.0.3
2024-06-02 7:04:47	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
2024-06-02 7:06:14	Resolved universidad.vmorenoa.com to 192.168.0.5	192.168.0.5
2024-06-02 7:07:06	Resolved restaurante.vmorenoa.com to 192.168.0.3	192.168.0.3
2024-06-02 7:07:17	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
2024-06-02 7:09:54	Resolved banco.vmorenoa.com to 192.168.0.2	192.168.0.2
2024-06-02 7:13:34	Resolved universidad.vmorenoa.com to 192.168.0.5	192.168.0.5
2024-06-02 7:15:39	Resolved vmorenoa.com to 192.168.0.1	192.168.0.1
2024-06-02 7:15:48	Resolved vmorenoa.com to 192.168.0.1	192.168.0.1
2024-06-02 7:16:06	Resolved banco.vmorenoa.com to 192.168.0.2	192.168.0.2
2024-06-02 7:16:54	Resolved tienda.vmorenoa.com to 192.168.0.6	192.168.0.6
2024-06-02 7:18:06	Resolved foro.vmorenoa.com to 192.168.0.7	192.168.0.7
2024-06-02 7:18:48	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
2024-06-02 7:19:55	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
2024-06-02 7:21:13	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
2024-06-02 7:22:44	Resolved banco.vmorenoa.com to 192.168.0.2	192.168.0.2
2024-06-02 7:23:25	Resolved banco.vmorenoa.com to 192.168.0.2	192.168.0.2
2024-06-02 7:23:59	Resolved tienda.vmorenoa.com to 192.168.0.6	192.168.0.6
2024-06-02 7:24:27	Resolved banco.vmorenoa.com to 192.168.0.2	192.168.0.2
2024-06-02 7:30:42	Resolved vmorenoa.com to 192.168.0.1	192.168.0.1
2024-06-02 7:34:38	Resolved universidad.vmorenoa.com to 192.168.0.5	192.168.0.5
2024-06-02 7:35:46	Resolved tienda.vmorenoa.com to 192.168.0.6	192.168.0.6
2024-06-02 7:36:06	Resolved banco.vmorenoa.com to 192.168.0.2	192.168.0.2

IP	# solicitudes
192.168.0.1	119
192.168.0.2	161
192.168.0.3	132
192.168.0.4	151
192.168.0.5	138
192.168.0.6	159
192.168.0.7	137



DNS_Log | vmorenoa.com | banco.vmorenoa.com | restaurante.vmorenoa.com | seguros.vmoreno ...

Figura 1: Log del segundo modelo de simulación de tráfico web

.4.2. Log del WebServer principal

	Timestamp	Action	Status
2	2024-06-02 7:15:39	User marketing79991ul2ok0y added.	200
3	2024-06-02 7:15:48	User office19734boyxmf8 rejected with status 503.	503
4	2024-06-02 8:57:19	User mail94342dxxw349 rejected with status 429.	429
5	2024-06-02 9:00:21	User tech48915ge2lru4 added.	200
6	2024-06-02 9:03:32	User mail561614o2vpiy rejected with status 503.	503
7	2024-06-02 9:09:12	User service19046hrmn3yb added.	200
8	2024-06-02 9:11:20	User contact17243xav2tz4 rejected with status 404.	404
9	2024-06-02 9:40:10	User account61810okwcilg added.	200
10	2024-06-02 10:07:46	User contact78384dpkmj1c added.	200
11	2024-06-02 10:31:59	User contact57449ceapfqg rejected with status 400.	400
12	2024-06-02 12:12:27	User tech29526ehtg81b added.	200
13	2024-06-02 12:27:43	User tech966654izx6ph added.	200
14	2024-06-02 12:46:19	User newsletter155510wcs6b7 rejected with status 404.	404
15	2024-06-02 13:00:49	User newsletter90218b93khqr added.	200
16	2024-06-02 13:17:42	User admin74551imd5s4s rejected with status 429.	429

DNS_Log | vmorenoa.com | banco.vmorenoa.com | restaurante.vmorenoa.com | seguros.vmoreno ...

Figura 2: Log de actividad del WebServer principal en el segundo modelo de simulación.

.4.3. Log del WebServer banco

	Timestamp	Action	Status
2	2024-06-02 7:16:06	User sales3838247jq9fc added.	200
3	2024-06-02 7:22:44	User service2713flgb7gw rejected with status 503.	503
4	2024-06-02 7:23:25	User mail84602j9vv8hr added.	200
5	2024-06-02 7:24:27	User billing92405oh6ro22 added.	200
6	2024-06-02 7:38:30	User feedback47245ockk5dd added.	200
7	2024-06-02 7:51:24	User newsletter42421lof922p added.	200
8	2024-06-02 8:53:24	User contact33496ayjgu28 added.	200
9	2024-06-02 9:23:18	User hello5479on89xy0 added.	200
10	2024-06-02 9:55:11	User feedback10916zz8w4m0 rejected with status 400.	400
11	2024-06-02 10:59:10	User feedback45669s6d9nrn added.	200
12	2024-06-02 11:53:40	User contact67129ks4he2i rejected with status 400.	400
13	2024-06-02 11:56:24	User sales7014kjduqk rejected with status 429.	429
14	2024-06-02 12:01:22	User sales91582tdn0k60 added.	200
15	2024-06-02 12:20:25	User hello58421opnmgud rejected with status 503.	503

DNS_Log | vmorenoa.com | banco.vmorenoa.com | restaurante.vmorenoa.com | seguros.vmoreno ...

Figura 3: Log de actividad del WebServer del subdominio banco en el segundo modelo de simulación.

.4.4. Log del WebServer restaurante

	Timestamp	Action	Status
2	2024-06-02 6:58:14	User support62544jbvr3zq rejected with status 404.	404
3	2024-06-02 7:01:47	User marketing3876c87aknk added.	200
4	2024-06-02 7:07:06	User team27924gnz5num added.	200
5	2024-06-02 7:40:52	User user63205664t0go rejected with status 404.	404
6	2024-06-02 7:59:01	User account46475amebcj0 rejected with status 500.	500
7	2024-06-02 8:04:46	User mail7873msv8ess added.	200
8	2024-06-02 8:40:53	User sales20585j8t4egk rejected with status 404.	404
9	2024-06-02 10:12:57	User feedback3718w8pdyko added.	200
10	2024-06-02 10:25:24	User tech54336wh2i5hy added.	200
11	2024-06-02 10:48:18	User marketing714562axii21 added.	200
12	2024-06-02 12:31:24	User team25116zmirg10 added.	200
13	2024-06-02 13:38:33	User tech54336wh2i5hy rejected with status 503.	503
14	2024-06-02 14:20:49	User admin23535b4mbck6 added.	200
15	2024-06-02 14:51:04	User account99742lvtmt6g added.	200

DNS_Log | vmorenoa.com | banco.vmorenoa.com | restaurante.vmorenoa.com | seguros.vmoreno ...

Figura 4: Log de actividad del WebServer del subdominio restaurante en el segundo modelo de simulación.

.5. Clase AppServer del Tercer Modelo de Simulación

```python
class AppServer(WebServer):
    def __init__(self, ip_address, port, document_root,
        server_name, database):
        super().__init__(ip_address, port, document_root,
            server_name)
        self.database = database

    def app_process_request(self, request, request_time,
        db_log_sheet):
        status = super().process_request(request, request_time)
        if status != 200:
            return status

        # Simulación de una consulta a la base de datos
        request_data = json.loads(request)
        db_status = self.database.query(request_data,
            request_time, db_log_sheet)
        return db_status
```

.6. Clase Cache del Tercer Modelo de Simulación

```python
class Cache:
    def __init__(self, capacity):
        # Inicializa una nueva instancia de la clase Cache con
            una capacidad dada.
```

```
        self.capacity = capacity
        self.data = {}

    def get(self, key):
        # Obtiene el valor asociado a una clave en el caché.
        # Devuelve None si la clave no está presente.
        return self.data.get(key, None)

    def set(self, key, value):
        # Establece el valor para una clave en el caché.
        # Si el caché alcanza su capacidad, elimina el elemento
           más antiguo.
        if len(self.data) >= self.capacity:
            self.data.pop(next(iter(self.data)))
        self.data[key] = value

    def log_action(self, timestamp, action, status, log_sheet):
        # Registra una acción realizada en el caché.
        # Anade una entrada en log_sheet con el timestamp, acció
           n, estado y tamano actual del caché.
        log_sheet.append([timestamp, action, status, len(self.
           data)])
```

.7. Clase `MessageQueue` del Tercer Modelo de Simulación

```
class MessageQueue:
    def __init__(self, capacity):
        # Inicializa una nueva instancia de la clase
           MessageQueue con una capacidad dada.
        self.capacity = capacity
        self.queue = []

    def enqueue(self, message):
        # Anade un mensaje a la cola si no se ha alcanzado la
           capacidad máxima.
        # Devuelve True si el mensaje se anadió correctamente,
           False en caso contrario.
        if len(self.queue) < self.capacity:
            self.queue.append(message)
            return True
        return False

    def dequeue(self):
        # Elimina y devuelve el primer mensaje de la cola.
        # Devuelve None si la cola está vacía.
        if self.queue:
```

```
            return self.queue.pop(0)
        return None

    def log_action(self, timestamp, action, status, log_sheet):
        # Registra una acción realizada en la cola de mensajes.
        # Anade una entrada en log_sheet con el timestamp, acció
            n, estado y tamano actual de la cola.
        log_sheet.append([timestamp, action, status, len(self.
            queue)])
```

.8. Clase CachedWebServer del Tercer Modelo de Simulación

```
class CachedWebServer(WebServer):
    def __init__(self, ip_address, port, document_root,
        server_name, cache):
        # Inicializa una nueva instancia de CachedWebServer con
            la información del servidor y la caché proporcionada.
        super().__init__(ip_address, port, document_root,
            server_name)
        self.cache = cache

    def cache_process_request(self, request, request_time,
        cache_log_sheet):
        # Procesa una solicitud utilizando la caché.
        # Si la información está en la caché, registra un "cache
            hit" y devuelve el estado 200.
        # Si la información no está en la caché, procesa la
            solicitud normalmente,
        # almacena los datos en la caché y registra un "cache
            miss" si la solicitud fue exitosa.
        request_data = json.loads(request)
        username = request_data.get("username")
        cached_data = self.cache.get(username)

        if cached_data:
            self.cache.log_action(request_time, f"Cache hit for
                {username}", 200, cache_log_sheet)
            return 200

        status = super().process_request(request, request_time)
        if status == 200:
            self.cache.set(username, request_data)
            self.cache.log_action(request_time, f"Cache miss for
                {username}", 200, cache_log_sheet)

```

```
25          return status
```

.9. Clase `QueuedWebServer` del Tercer Modelo de Simulación

```
class QueuedWebServer(WebServer):
    def __init__(self, ip_address, port, document_root,
        server_name, message_queue):
        # Inicializa una nueva instancia de QueuedWebServer con
            la información del servidor y la cola de mensajes
            proporcionada.
        super().__init__(ip_address, port, document_root,
            server_name)
        self.message_queue = message_queue

    def queue_process_request(self, request, request_time,
        queue_log_sheet):
        # Procesa una solicitud utilizando la cola de mensajes.
        # Intenta encolar un mensaje que indica la solicitud de
            un usuario.
        # Si la encolación es exitosa, registra la acción y
            devuelve el estado 200.
        # Si la cola está llena, registra que el mensaje fue
            descartado y devuelve el estado 503.
        request_data = json.loads(request)
        message = f"Request from {request_data.get('username')}"

        if self.message_queue.enqueue(message):
            self.message_queue.log_action(request_time, f"
                Message enqueued: {message}", 200,
                queue_log_sheet)
            return 200

        self.message_queue.log_action(request_time, "Queue full,
            message dropped", 503, queue_log_sheet)
        return 503
```

.10. Clase `Database Backup` del Tercer Modelo de Simulación

```
class DatabaseBackup:
    def __init__(self, capacity):
        # Inicializa una nueva instancia de DatabaseBackup con
            una capacidad dada.
```

```
        self.capacity = capacity
        self.data = {}

    def backup(self, request_data):
        # Realiza una copia de seguridad de los datos de la
          solicitud.
        # Si la capacidad máxima se ha alcanzado, elimina el
          elemento más antiguo.
        # Luego, guarda los datos de la solicitud en el
          almacenamiento.
        if len(self.data) >= self.capacity:
            self.data.pop(next(iter(self.data)))
        self.data[request_data['username']] = request_data

    def log_action(self, timestamp, action, status, log_sheet):
        # Registra una acción realizada en la copia de seguridad
          .
        # Anade una entrada en log_sheet con el timestamp, acció
          n, estado y tamano actual del almacenamiento de datos
          .
        log_sheet.append([timestamp, action, status, len(self.
          data)])
```

.11. Clase Database del Tercer Modelo de Simulación

```
import numpy as np

class Database:
    def __init__(self, capacity, backup):
        # Inicializa una nueva instancia de Database con una
          capacidad dada y una instancia de backup.
        self.data = {}
        self.capacity = capacity
        self.failure_rate = 0.1  # Probabilidad de fallo en la
          base de datos
        self.backup = backup

    def query(self, request_data, request_time, db_log_sheet):
        # Realiza una consulta en la base de datos.
        # Simula un fallo en la base de datos con una
          probabilidad definida.
        # Si la base de datos falla, registra el fallo y
          devuelve el estado 500.
        # Si la capacidad está llena, registra esta situación y
          devuelve el estado 503.
```

```
        # Si la consulta es exitosa, reduce la capacidad disponible, realiza una copia de seguridad,
        # registra la acción y devuelve el estado 200.
        if np.random.uniform(0, 1) < self.failure_rate:
            self.log_action(request_time, "DB Query Failed", 500, db_log_sheet)
            return 500  # Internal Server Error

        if self.capacity <= 0:
            self.log_action(request_time, "DB Capacity Full", 503, db_log_sheet)
            return 503  # Service Unavailable

        # Simulación de una consulta exitosa
        self.capacity -= 1
        self.backup.backup(request_data)
        self.log_action(request_time, "DB Query Successful", 200, db_log_sheet, self.capacity)
        return 200

    def log_action(self, timestamp, action, status, log_sheet, capacity=None):
        # Registra una acción realizada en la base de datos.
        # Anade una entrada en log_sheet con el timestamp, acción, estado y capacidad actual si se proporciona.
        if capacity is not None:
            log_sheet.append([timestamp, action, status, capacity])
        else:
            log_sheet.append([timestamp, action, status])

    def reset_capacity(self, new_capacity):
        # Restablece la capacidad de la base de datos a un nuevo valor.
        self.capacity = new_capacity
```

.12. Clase DNS del Tercer Modelo de Simulación

```
class DNS:
    def __init__(self):
        # Inicializa una nueva instancia de DNS con una tabla DNS predefinida.
        self.dns_table = {
            "vmorenoa.com": "192.168.0.1",
            "banco.vmorenoa.com": "192.168.0.2",
            "restaurante.vmorenoa.com": "192.168.0.3",
```

```
            "seguros.vmorenoa.com": "192.168.0.4",
            "universidad.vmorenoa.com": "192.168.0.5",
            "tienda.vmorenoa.com": "192.168.0.6"
        }

    def resolve(self, subdomain, request_time, log_sheet):
        # Resuelve un subdominio a su dirección IP correspondiente.
        # Si el subdominio no se encuentra en la tabla DNS, devuelve "404".
        # Registra la acción de resolución en log_sheet con el timestamp, la acción y la IP resultante.
        ip = self.dns_table.get(subdomain, "404")
        action = f"Resolved {subdomain} to {ip}" if ip != "404" else f"Failed to resolve {subdomain}"
        log_sheet.append([request_time, action, ip])
        return ip
```

.13. Logs del Tercer Modelo de Simulación

.13.1. Log del DNS

	Timestamp	Action	IP
2	2024-06-02 14:39:07	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
3	2024-06-02 14:42:47	Resolved universidad.vmorenoa.com to 192.168.0.5	192.168.0.5
4	2024-06-02 14:43:22	Resolved vmorenoa.com to 192.168.0.1	192.168.0.1
5	2024-06-02 14:44:00	Resolved restaurante.vmorenoa.com to 192.168.0.3	192.168.0.3
6	2024-06-02 14:45:13	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
7	2024-06-02 14:45:31	Resolved seguros.vmorenoa.com to 192.168.0.4	192.168.0.4
8	2024-06-02 14:46:33	Resolved vmorenoa.com to 192.168.0.1	192.168.0.1
9	2024-06-02 14:46:50	Resolved universidad.vmorenoa.com to 192.168.0.5	192.168.0.5
10	2024-06-02 14:47:42	Resolved banco.vmorenoa.com to 192.168.0.2	192.168.0.2

Figura 5: Log del DNS del tercer modelo de simulación.

.13.2. Log de la Database

1	Timestamp	Action	Status	Capacity
4	2024-06-02 14:47:42	DB Query Successful	200	60
6	2024-06-02 15:00:06	DB Query Successful	200	73
9	2024-06-02 15:32:08	DB Query Successful	200	35
10	2024-06-02 15:36:04	DB Query Successful	200	6
13	2024-06-02 15:54:41	DB Query Successful	200	80
15	2024-06-02 15:58:48	DB Query Successful	200	94
19	2024-06-02 16:21:42	DB Query Successful	200	87
21	2024-06-02 16:27:24	DB Query Successful	200	45
22	2024-06-02 16:28:43	DB Query Successful	200	83
24	2024-06-02 16:46:39	DB Query Successful	200	39
25	2024-06-02 17:01:40	DB Query Successful	200	23

Figura 6: Log de la Database del tercer modelo de simulación.

.13.3. Log del Backup

1	Timestamp	Action	Status	Backup Size
2	2024-06-02 14:47:42	Backup for user121686bbrqx6	200	40
3	2024-06-02 15:00:06	Backup for tech58443ohcjsta	200	27
4	2024-06-02 15:32:08	Backup for info83361gl78kzd	200	60
5	2024-06-02 15:36:04	Backup for billing541801mza14v	200	88
6	2024-06-02 15:54:41	Backup for mail10174jbpkrtg	200	20
7	2024-06-02 15:58:48	Backup for newsletter93625hqan7qv	200	6
8	2024-06-02 16:21:42	Backup for noreply18074u1fxjzn	200	13
9	2024-06-02 16:27:24	Backup for newsletter386325p41wea	200	52
10	2024-06-02 16:28:43	Backup for team35450yjzs9j	200	17

Figura 7: Log del Backup del tercer modelo de simulación.

.13.4. Log de la Cache

1	Timestamp	Action	Status	Cache Size
2	2024-06-02 16:15:24	Cache miss for hello563805m7g57u	200	1
3	2024-06-03 13:36:29	Cache miss for customer29380ul5i45z	200	2
4	2024-06-02 23:22:37	Cache miss for hello76652pdm50y7	200	3
5	2024-06-03 8:07:44	Cache miss for office63003bitwfev	200	4
6	2024-06-02 23:14:47	Cache miss for user36003kwbu25s	200	5
7	2024-06-03 5:14:30	Cache miss for team719108ta2tuo	200	6
8	2024-06-03 12:49:39	Cache miss for info6273440pxqo1	200	7
9	2024-06-02 19:40:12	Cache miss for mail19684w04mj8m	200	8
10	2024-06-03 1:20:57	Cache miss for billing45375g44w9jf	200	9
11	2024-06-02 16:41:48	Cache miss for user20263ab987tn	200	10
12	2024-06-03 7:45:23	Cache miss for newsletter78192j0h654c	200	10

Figura 8: Log de la Cache del tercer modelo de simulación.

.13.5. Log de la Message Queue

	Timestamp	Action	Status	Queue Size
2	2024-06-03 1:38:08	Message enqueued: Request from noreply472	200	1
3	2024-06-02 16:27:15	Message enqueued: Request from noreply995	200	2
4	2024-06-03 10:56:01	Message enqueued: Request from mail49371	200	3
5	2024-06-02 18:09:53	Message enqueued: Request from billing2526	200	4
6	2024-06-03 10:28:15	Message enqueued: Request from contact554	200	5
7	2024-06-03 14:38:19	Message enqueued: Request from sales34894	200	6
8	2024-06-03 11:37:17	Message enqueued: Request from info590992	200	7
9	2024-06-02 16:58:58	Message enqueued: Request from support900	200	8
10	2024-06-03 1:00:49	Message enqueued: Request from office9478	200	9
11	2024-06-02 22:34:44	Message enqueued: Request from customer7	200	10
12	2024-06-02 22:33:01	Queue full, message dropped	503	10
13	2024-06-03 13:33:13	Queue full, message dropped	503	10
14	2024-06-02 20:41:22	Queue full, message dropped	503	10

Figura 9: Log de la Message Queue del tercer modelo de simulación.

.13.6. Log del AppServer

	Timestamp	Action	Status	Load	Requests Received	Connection Status
2	2024-06-02 14:43:22	Server State - vmorenoa.com	Active	0,359506345	154	Active
3	2024-06-02 14:46:33	Server State - vmorenoa.com	Active	0,486545953	144	Active
4	2024-06-02 14:54:29	Server State - vmorenoa.com	Active	0,927282853	148	Active
5	2024-06-02 14:54:43	Server State - vmorenoa.com	Active	0,61109114	17	Active
6	2024-06-02 15:41:39	Server State - vmorenoa.com	Active	0,64547304	66	Active
7	2024-06-02 15:49:56	Server State - vmorenoa.com	Active	0,205045898	162	Active
8	2024-06-02 15:53:44	Server State - vmorenoa.com	Active	0,950745356	160	Active
9	2024-06-02 16:03:45	Server State - vmorenoa.com	Active	0,595485135	125	Active
10	2024-06-02 16:07:31	Server State - vmorenoa.com	Active	0,037075627	48	Active

Figura 10: Log del AppServer del tercer modelo de simulación.

.13.7. Log del WebServer

	Timestamp	Action	Status	Load	Requests Received	Connection Status
2	2024-06-02 14:59:08	Server State - tienda.vmorenoa.com	Active	0,320685289	45	Active
3	2024-06-02 15:13:00	Server State - tienda.vmorenoa.com	Active	0,659308883	67	Active
4	2024-06-02 15:15:45	User customer84630q7ualyn added.	200	0,381263905	9	Active
5	2024-06-02 15:15:45	Server State - tienda.vmorenoa.com	Active	0,381263905	9	Active
6	2024-06-02 15:43:33	Server State - tienda.vmorenoa.com	Active	0,942196182	24	Active
7	2024-06-02 15:56:40	User admin8430ocwjaxj added.	200	0,422034621	85	Active
8	2024-06-02 15:56:40	Server State - tienda.vmorenoa.com	Active	0,422034621	85	Active
9	2024-06-02 16:29:01	Server State - tienda.vmorenoa.com	Active	0,801272563	139	Active
10	2024-06-02 16:32:55	Server State - tienda.vmorenoa.com	Active	0,484382702	14	Active

Figura 11: Log del WebServer del tercer modelo de simulación.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Mon Jun 03 23:45:01 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)