Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

Sistemas Distribuidos

Entrega 1: Proyecto Sistemas Distribuidos

Autores:
Lucas Vicuña
Vicente Silva

Profesor: Nicolás Hidalgo

30 de abril de 2025

Índice

1. Descripción general del proyecto

Este proyecto consistió en la construcción de un sistema distribuido orientado a la recopilación, almacenamiento, análisis y simulación de datos de tráfico en la Región Metropolitana, utilizando técnicas de scraping, bases de datos relacionales y caché.

El objetivo principal fue demostrar el impacto positivo de un sistema de caché Redis ante consultas repetitivas, y cómo herramientas como Docker y MySQL permiten una arquitectura portable y escalable.

2. Scraping y base de datos

El proceso de scraping se implementó utilizando Selenium en modo headless, recorriendo una cuadrícula expandible centrada en la Región Metropolitana. El sistema recolectó eventos visuales del mapa de Waze, extrayendo su tipo, latitud, longitud, y descripción.

Los datos fueron almacenados en una base de datos MySQL estructurada bajo el esquema `eventos(id, tipo, descripcion, lat, lon, fecha_extraccion, cuadrante)`. Se diseñó una estrategia que permite escalar la recolección por cuadrantes hasta alcanzar los 10.000 eventos, monitoreando el progreso con barras visuales.

3. Simulador de tráfico y sistema de caché

3.1. Generador de tráfico

Se creó un generador de tráfico que simula consultas a la base de datos siguiendo dos distribuciones:

- **Poisson:** donde los tiempos entre consultas siguen una exponencial con media $1/\lambda$
- **Exponencial:** con un parámetro de espera promedio directo

Ambas simulaciones fueron configuradas para repetir 10 veces un conjunto de 10 `id_evento` únicos (100 consultas en total), simulando así múltiples accesos reales a los mismos datos.

3.2. Sistema de Caché con Redis

Redis fue configurado como caché en un contenedor independiente, enlazado con el servicio de MySQL. El sistema primero consulta Redis y solo accede a MySQL en caso de que el evento no esté almacenado (MISS).

Cada entrada en caché se guarda con un TTL de 3600 segundos. El script Python

registra y clasifica cada consulta como HIT o MISS, y grafica el rendimiento acumulado.

3.3. Resultados

- **Poisson:** 100 consultas → 90 hits, 10 misses → 90 % efectividad
- **Exponencial:** 100 consultas → 91 hits, 9 misses → 91 % efectividad

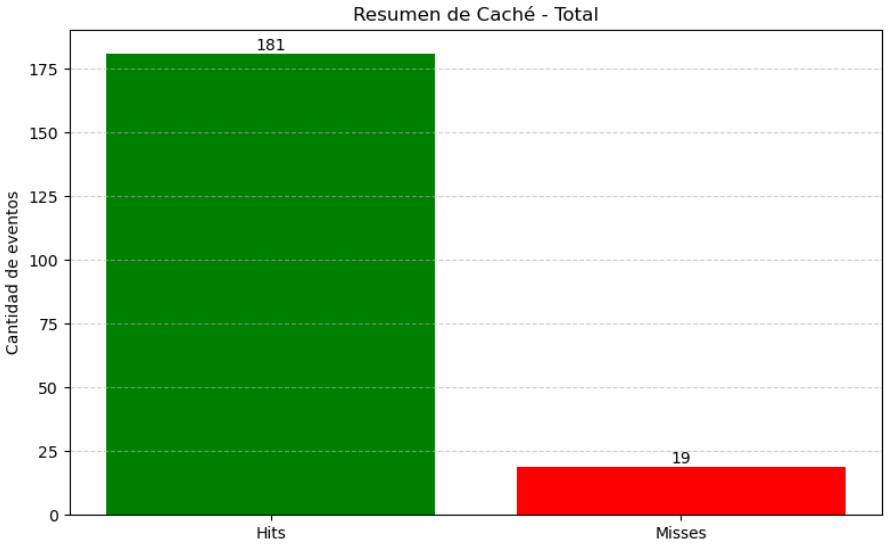

Figura 1: Resumen total de hits y misses

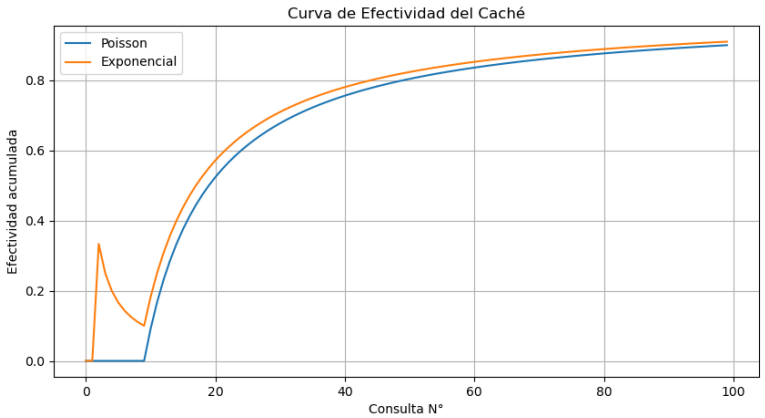

Figura 2: Curva de efectividad acumulada del sistema de caché

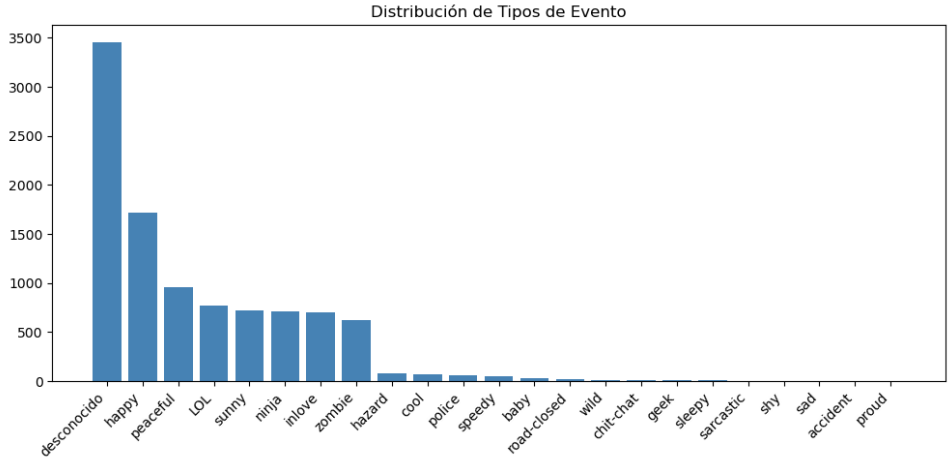

Figura 3: Distribución de tipos de evento almacenados

4. Extras y funcionalidades adicionales

4.1. CRUD Interno Simulado

Durante el scraping, se implementaron operaciones equivalentes a un CRUD:

- **Create:** inserción de eventos en tiempo real
- **Read:** consulta desde Redis o MySQL
- **Update:** actualización del TTL de la caché
- **Delete:** limpieza manual del caché o limpieza por expiración automática

4.2. Dockerización completa del sistema

Todos los servicios fueron dockerizados utilizando un archivo `docker-compose.yml`, el cual define dos contenedores:

- `mysql_eventos`: Base de datos MySQL
- `redis_cache`: Servicio Redis

El entorno no incluye un contenedor de aplicación, ya que el código Python es ejecutado desde el notebook `Tarea1.ipynb` o el script `funciones.py` directamente en el host.

Para asegurar la instalación de todas las dependencias, se utilizó un archivo `requirements.txt` con los paquetes necesarios:

- `mysql-connector-python`

- `redis`
- `pandas, numpy, matplotlib`
- `openpyxl`

5. Análisis y Discusión

5.1. Generador de Tráfico

Se utilizaron las distribuciones Poisson y Exponencial debido a que son modelos ampliamente usados para simular tiempos entre llegadas en sistemas de colas, redes de telecomunicaciones y tráfico. Estas distribuciones son realistas, ya que el tráfico de usuarios no es constante ni determinista, sino que sigue patrones probabilísticos.

5.2. Almacenamiento

Se utilizó **MySQL** como sistema de almacenamiento debido a su madurez, eficiencia y compatibilidad con múltiples herramientas. La principal limitación es su escalabilidad vertical, pero su uso está justificado por su integración sencilla con Python y Docker.

5.3. Métricas y Políticas de Cache

Se utilizaron como métricas: porcentaje de HITs, MISSes y efectividad acumulada. La política utilizada fue TTL (Time To Live) fija de 1 hora. Redis mostró una efectividad del 90–91 %, validando su uso en este contexto.

5.4. Pruebas de Rendimiento y Escalabilidad

Se probaron 100 consultas en patrones realistas, validando el impacto positivo del cache. La arquitectura dockerizada permite escalar fácilmente los servicios de base de datos y caché en futuras entregas.

6. Conclusiones

El desarrollo de esta entrega permitió consolidar una arquitectura funcional distribuida que resuelve de manera efectiva el desafío de extracción y análisis de eventos de tráfico a gran escala. A través de la implementación modular de scraping, almacenamiento, generación de tráfico sintético y sistema de caché, se logró cumplir con todos los requisitos establecidos en la pauta del proyecto.

Uno de los aprendizajes más relevantes fue la validación empírica del uso de Redis como sistema de cacheo. Los resultados obtenidos mostraron una mejora significativa

en la eficiencia de las consultas repetidas, con un **90-91 % de efectividad en cache** en escenarios simulados de 100 consultas sobre 10 eventos únicos. Esta alta tasa de hits demostró el impacto directo que puede tener una buena estrategia de caching sobre el rendimiento general del sistema.

El uso de distribuciones estadísticas (Poisson y Exponencial) para el modelado de tráfico de consultas fue crucial para simular condiciones realistas de acceso concurrente. Estas distribuciones representan correctamente contextos donde la llegada de eventos puede ser variable, como sucede en sistemas urbanos dinámicos. Además, su integración permitió contrastar el comportamiento del caché en distintos patrones de carga.

Desde el punto de vista de infraestructura, la dockerización de los servicios MySQL y Redis permitió garantizar portabilidad y replicabilidad del entorno, facilitando su uso en distintos equipos y futuras entregas. El uso de Jupyter como entorno de experimentación también potenció la visibilidad y trazabilidad del proceso, en paralelo al script Python que automatiza tareas.

Finalmente, el proyecto cumplió con la meta de construir una base robusta para las futuras etapas del sistema distribuido. La información recolectada, la arquitectura implementada y los mecanismos de prueba sentaron las bases para incorporar nuevas funcionalidades como procesamiento de datos, visualización avanzada y escalabilidad horizontal.

7. Referencias

- Docker Compose Documentation
- Redis Documentation
- MySQL Reference Manual
- NumPy
- Matplotlib

8. Entrega del Proyecto

A continuación, se indican los enlaces correspondientes a la entrega oficial del proyecto:

- **Project ID:** `69384236`
- **Repositorio general del curso (GitLab):**
  `https://gitlab.com/proyectos_u/sistemas_distribuidos`
- **Repositorio del grupo con commit específico:**
  `629a4e1f1053e4978d748cdd4e3b2c96d7af5ff3`