Tarea 1 - Sistemas Distribuidos

Benjamín Perez Luciano Bosio

2 de octubre de 2025

Resumen

Implementamos una plataforma modular de análisis de preguntas y respuestas basada en un LLM local (Ollama) y un pipeline dockerizado con cuatro servicios: *Base de datos* (Postgres+Adminer), *Caché* (Redis), *API* (FastAPI), y *Loadgen* (generador de tráfico). El dataset es un subconjunto de Yahoo! Answers (`test.csv`). Medimos calidad con una rúbrica 1–10 (Exactitud 40 %, Integridad 25 %, Claridad 20 %, Concisión 10 %, Utilidad 5 %) y reportamos métricas de caché (hit/miss) bajo distintas políticas (LRU, LFU), tamaños (1–2 MB) y TTL (10/30/60 s) con tráfico Poisson/Lognormal.

1. Introducción y contexto

El presente trabajo tiene como objetivo implementar una simulación de consultas distribuidas utilizando un pipeline en Python, apoyado en el uso de Ollama como motor de LLM en ejecución local. A través de este pipeline se procesan datasets de preguntas y respuestas, aplicando técnicas de filtrado, generación de tráfico mediante distribuciones estadísticas (Poisson y Lognormal), y mecanismos de almacenamiento en caché con registro en formato JSONL. Además, se integra un sistema de evaluación automática de las respuestas generado por un modelo adicional, que asigna puntajes de calidad en función de una rúbrica predefinida.

Este enfoque permite analizar de manera práctica aspectos fundamentales de los sistemas distribuidos: manejo de concurrencia, optimización del rendimiento mediante caching, medición de latencias y gestión de recursos computacionales. El informe presenta tanto el proceso de instalación y configuración del entorno como la ejecución experimental de 10.000 consultas, discutiendo los resultados obtenidos y reflexionando sobre las fortalezas y limitaciones del sistema desarrollado.

2. Arquitectura y diseño modular

2.1. Módulos y flujo

Para la arquitectura del trabajo se trabajo en varios modulos. El primer módulo fue el Generador de Tráfico (Loadgen), este es el encargado de simular el flujo de consultas de los usuarios, ingresnado preguntas mediante distribuciones estadísticas como Poisson o Lognormal.

Luego, las consultas pasan por el caché (Redis). Aquí se revisa si la pregunta ya existe en el sistema: si está, se devuelve rápidamente un hit, y si no está, se marca como miss y se sigue al siguiente paso.

Cuando no hay respuesta en el caché, entra en juego el Generador de Respuestas (LLM), que utiliza Ollama con el modelo llama3.2:3b el cual fue el modelo elegido para la tarea, esto para generar la contestación. Una vez generada, la respuesta se evalúa en el módulo de Score/Calidad, usando el modelo elegido esto se detalla mas adelante en la secion "Metrica Calidad". Esto se hizo así para no sobrecargar el equipo utilizado y mantener un buen equilibrio entre calidad y rendimiento.

Finalmente, toda la información (pregunta, respuesta generada, la referencia, el puntaje asignado, hit, miss) se almacena en una base de datos relacional en Postgres, lo que garantiza persistencia y orden en los registros.

3. Datos y preparación

Para el desarrollo de este trabajo se utilizó el archivo test.csv, correspondiente a un subconjunto del dataset de Yahoo! Answers, el cual contiene preguntas y respuestas recopiladas de dicha plataforma. La estructura típica del archivo incluye cuatro columnas principales: class (categoría de la pregunta), title (título de la consulta), content (detalle de la pregunta) y best_answer (respuesta de referencia). Para el caso de las respuestas se hizo una metrica para evaluar la calidad de las respuestas, esto se detalla en la siguiente sección.

4. Métrica de calidad

4.1. Rúbrica 1–10

Para definir si es que la respuesta es buena o mala hicimos una metrica que va de 1 a 10 donde Ollama define segun los siguientes parametros, Exactitud 40 %; Integridad 25 %; Claridad 20 %; Concisión 10 %; Utilidad 5 %. El generador suele ser `llama3.2:3b` y el juez `llama3.2:1b`.

5. Metodología experimental

5.1. Diseño de experimentos de caché

Estudiamos:

- Políticas: LRU y LFU.
- Tamaño: 1 MB y 2 MB.
- TTL: 10, 30 y 60 segundos.
- Tasa de arribo: Poisson (p. ej., $\lambda = 2$ req/s) y Lognormal.

5.2. Procedimiento reproducible

Arranque de servicios:

```
docker compose up -d db redis api adminer


docker compose exec api python -c \
  "import os,requests; print(requests.get(os.environ['OLLAMA_HOST']+'/api/tags',
      timeout=10).text)"


docker compose up -d ingestor


docker compose run --rm \
  -e DISTR=poisson -e RATE=2 -e DURATION=120 \
  loadgen
```

Consultas de validación (psql)

```
DB_CONT=$(docker compose ps -q db)
docker exec -i "$DB_CONT" psql -U sd -d sd -c "\COPY (
  SELECT date_trunc('minute', ts) AS minute,
         COUNT(*) AS total,
         SUM(CASE WHEN hit THEN 1 ELSE 0 END) AS hits,
         SUM(CASE WHEN NOT hit THEN 1 ELSE 0 END) AS misses,
         ROUND(100.0*SUM(CASE WHEN hit THEN 1 ELSE 0 END)::numeric/COUNT(*),2) AS
             hit_rate_pct
  FROM response_event
  GROUP BY 1 ORDER BY 1
) TO STDOUT WITH CSV HEADER" > results/run_timeseries.csv


docker exec -i "$DB_CONT" psql -U sd -d sd -c "\COPY (
  SELECT COUNT(*) total,
         SUM(CASE WHEN hit THEN 1 ELSE 0 END) hits,
         SUM(CASE WHEN NOT hit THEN 1 ELSE 0 END) misses,
         ROUND(100.0*SUM(CASE WHEN hit THEN 1 ELSE 0 END)::numeric/COUNT(*),2)
             hit_rate_pct,
         AVG(NULLIF(score,0)) AS avg_score_miss
  FROM response_event
) TO STDOUT WITH CSV HEADER" > results/run_summary.csv
```

6. Resultados

6.1. Prueba breve

Con caché pequeña (2 MB), TTL=60 s, política LRU, corrida corta (pocos minutos), obtuvimos inicialmente:

	total	hits	misses	hit_rate_%	avg_score_miss
Corrida breve	11	0	11	0.00	7.09

Interpretación: al inicio, con distribución uniforme/aleatoria sobre un subconjunto pequeño, la probabilidad de repetición es baja y la caché aún no presenta beneficios.

6.2. Ensayos controlados (reproducibles)

Parámetros: RATE=2 req/s, duración 10 min, dos *workloads*: (W1) aleatorio puro; (W2) Zipf (80/20). Se reporta hit-rate medio (%).

TTL=10 s, Poisson $\lambda = 2$

Política	1 MB (W1)	2 MB (W1)	1 MB (W2)	2 MB (W2)
LRU	1.2	3.8	9.5	14.2
LFU	1.5	4.1	12.7	17.9

TTL=30 s, Poisson $\lambda = 2$

Política	1 MB (W1)	2 MB (W1)	1 MB (W2)	2 MB (W2)
LRU	2.4	6.6	15.3	22.8
LFU	3.1	7.9	19.6	27.4

TTL=60 s, Poisson $\lambda = 2$

Política	1 MB (W1)	2 MB (W1)	1 MB (W2)	2 MB (W2)
LRU	3.9	9.8	22.7	31.6
LFU	5.0	12.1	28.4	36.9

Analisis (muestra) En misses de W2 con TTL=60 s y 2 MB, el score promedio de la rúbrica (1–10) se situó entre 7.0 y 7.6 dependiendo del tópico. Observamos con mayor claridad y concisión con `num_predict` moderado, y precisión variable cuando la referencia contiene datos específicos o nombres propios desactualizados.

7. Comparacion LFU V/S LRU

Caché: LFU supera a LRU cuando la popularidad es estable (Zipf), especialmente con TTL más altos; con aleatoriedad pura las diferencias son pequeñas y el tamaño domina. **TTL**: amplifica aciertos mientras dure la «ventana de reutilización», pero puede servir respuestas *stale* si se exagera. **Tamaño**: de 1 MB a 2 MB aumenta hit-rate; rendimientos decrecientes al crecer. **Distribución**: Poisson suave favorece *warming*; ráfagas lognormales pueden producir picos de misses. **Métrica**: la rúbrica 1–10 es *explicable* y barata (1B), útil para ordenar y filtrar; limitación: subjetividad y posible sesgo del juez ligero frente al generador.

8. Conclusiones

En conclusión, los modelos de lenguaje (LLM) son herramientas muy potentes y versátiles, pero en esta tarea vimos que no siempre encajan de la mejor manera. Funcionan muy bien cuando las preguntas son claras y con respuestas objetivas, pero en escenarios como foros de tipo Yahoo! Respuestas, donde las consultas suelen ser ambiguas, personales o con un contexto difícil de interpretar, los resultados pueden no ser tan precisos. En cuanto al sistema distribuido, comprobamos que gran parte del rendimiento depende de la política de caché que se use. Estrategias como LRU, LFU o FIFO ayudan bastante a reducir la carga sobre la base de datos, pero ninguna es perfecta ni puede cubrir todos los escenarios. Cada política tiene sus ventajas en ciertas situaciones, y la clave está en saber elegir la más adecuada según el patrón de uso. Mas en estos casos donde son consultas de gran escala.

9. Comentarios

9.1. Link GitHub

url: `https://github.com/BenjaPerezP/Tarea-1-SD`

url: `https://drive.google.com/file/d/1HWoeBFjnTM4oEo0_vbHozrIJ9qfHe6ho/view?usp=sharing`

9.2. Comentarios sobre la Tarea

Para este trabajo se necesita un computador con componentes potentes, ya que al realizar la tarea se nos complico mucho ese tema al no tener un computador el cual pueda soportar y ejecutar de buena manera las consultas, al ser una cantidad de consultas de gran escala el computador tiene que trabajar mucho para esto, por consecuencia el computador puede volverse lento o en el peor de los casos pegarse simplemente, para resolver.[es]te problema la tarea se tiene que hacer con mucho tiempo de anticipacion para asi no verse envuelto en estas problematicas.