Reporte Final: Sistemas Distribuidos

Alex Palma. (Estudiante de 6to Semestre, Ingeniería de Software, PUCEM)

11 de noviembre de 2025

Resumen

El presente informe documenta la implementación de un sistema distribuido compuesto por dos partes: un sistema de procesamiento concurrente (hilos vs. procesos) y un sistema de almacenamiento distribuido (sharding en MongoDB con Docker). Los resultados demuestran la superioridad de los hilos para tareas I/O-Bound y la correcta implementación de locks y sharding para garantizar la consistencia y el balance de carga.

1. Parte 1: Procesamiento Concurrente (Hilos vs. Procesos)

1.1. Resultados Obtenidos

Se procesaron 20 tareas con dificultad variable (simulando carga de I/O con `time.sleep(difficulty * 0.3)`).

Mecanismo	Tiempo Total (s)
Hilos (Threading)	1.5035
Procesos (Multiprocessing)	1.9840

Cuadro 1: Comparación de tiempos de ejecución para 20 tareas I/O-Bound.

1.2. Análisis y Conclusiones

Ventaja de Hilos: Los **Hilos** (`1.5035s`) superaron a los **Procesos** (`1.9840s`). Esto se debe a que la tarea es **I/O-Bound**. En Python, el Global Interpreter Lock (GIL) se libera durante las operaciones de espera (`time.sleep`), permitiendo que otros hilos utilicen eficientemente el tiempo muerto. Los Procesos, al requerir un alto *overhead* de creación y comunicación, resultaron ser más lentos para este tipo de carga.

1.3. Sincronización y Condiciones de Carrera

Para evitar condiciones de carrera al actualizar el contador de tareas completadas, se aplicaron los siguientes mecanismos de sincronización:

- **Hilos:** Se utilizó `threading.Lock()` para garantizar la exclusión mutua. El hilo debe adquirir el lock antes de modificar la variable compartida en memoria y liberarlo al terminar.
- **Procesos:** Se utilizó `multiprocessing.Value('i', 0)` con su método `.get_lock()`, el cual gestiona un segmento de memoria compartida entre los procesos y asegura que las actualizaciones sean atómicas y seguras.

2. Parte 2: Sistema de Almacenamiento Distribuido

2.1. Implementación con Docker y Sharding

El sistema utiliza Docker Compose para orquestar dos nodos de MongoDB (`mongo1`, `mongo2`) y una aplicación Python (`storage_app`) que implementa la lógica de distribución.

Mecanismo de Sharding: La distribución de documentos se realiza mediante una función de **Hashing Consistente**.

$$\text{node\_index} = \text{int}(\text{hashlib.md5(id).hexdigest()}, 16) \quad (\text{mód número de nodos})$$

Este método garantiza que, para un ID dado, el documento siempre sepa a qué nodo debe dirigirse para su inserción o búsqueda.

2.2. Estadísticas de Distribución (100 Documentos)

Conclusión: El resultado de 48 %/52 % confirma que la función de hash distribuye la carga de manera efectiva y casi perfectamente balanceada, lo que optimiza el rendimiento y la escalabilidad del sistema.

Nodo	Documentos	Porcentaje
node1	48	48.00 %
node2	52	52.00 %
Total de Documentos:		100

Cuadro 2: Balance de carga después de la inserción.

2.3. Estrategia de Búsqueda

Para la búsqueda del `document_id=50`, se obtuvo el siguiente resultado:

- **Resultado:** Encontrado en `node1` y no encontrado en `node2`.

Dado que el algoritmo de *sharding* es determinístico, la aplicación **podría haber sabido** que el documento estaba en `node1` usando el mismo hash. Sin embargo, se realizó una búsqueda en ambos nodos para demostrar la **capacidad de recuperación** y el modelo de **búsqueda distribuida total** (aunque ineficiente, es robusto) del sistema.

3. Conclusiones Generales

La implementación de este proyecto práctico ha consolidado la comprensión de dos pilares fundamentales en la Ingeniería de Software Distribuida:

- **Decisión de Concurrencia:** La elección entre hilos y procesos no es trivial, sino que debe estar dictada por la naturaleza de la tarea. Para tareas que implican esperas (I/O), la liberación del GIL en Python favorece claramente a los hilos. Para tareas intensivas en cálculo (CPU-Bound), los procesos son la única vía para lograr un verdadero paralelismo.
- **Consistencia es Prioridad:** Tanto en la concurrencia como en la distribución de datos, la sincronización (usando Locks) es un requisito no negociable para mantener la integridad del estado compartido.
- **Escalabilidad Horizontal:** La implementación del *sharding* por hash, orquestada por Docker, demuestra una solución de escalabilidad horizontal eficiente. Permite añadir más nodos de almacenamiento en el futuro y garantiza que la carga de datos se mantenga equitativa (48 %/52 %), lo cual es crucial para sistemas de alto rendimiento en producción.