Práctica de Laboratorio 3

Pruebas de Rendimiento y Planificación de Procesos

Estudiantes:
Wílmer E. León
Código: 1520010896

Jesús Orlando Orjuela
Código: 100384722

Hugo Alejandro Mejía
Código: 100312289

Fabián Andrés Cabana
Código: 1620010455

Docente:
José León León

Institución Universitaria Politécnico Grancolombiano
Facultad de Ingeniería, Diseño e Innovación
Sistemas Operacionales - Grupo B04 | Grupo de trabajo 11

2 de diciembre de 2025

Índice

1. Introducción

Esta tercera entrega documenta las pruebas de rendimiento realizadas sobre el balanceador Nginx configurado en las entregas anteriores, y presenta una profundización en algoritmos de planificación de procesos. Se incluyen el diseño de las pruebas, las ejecuciones, las métricas recogidas, el análisis detallado y las conclusiones finales.

2. Parte 3: Pruebas Adicionales y Evaluación del Rendimiento del Balanceo de Carga

2.1. Paso 1: Preparación de Pruebas

- Herramientas: Apache Benchmark (Foundation, 2018), `siege` (Fulmer y maintainers, 2019) y utilidades del sistema (`top`, `vmstat`, `sar`).
- Escenarios propuestos:
  1. Tráfico ligero: 100 peticiones totales, concurrencia 10.
  2. Tráfico intermedio: 1.000 peticiones, concurrencia 50.
  3. Tráfico pesado: 10.000 peticiones, concurrencia 200.
- Entorno: Balanceador Nginx (Inc., 2024) en `192.168.2.9` (UbunSO1); backends en `192.168.2.8` y `192.168.2.7`.
- Métricas a recolectar: throughput (requests/s), tiempo medio de respuesta, latencia p95, tasa de errores, uso de CPU y memoria en cada VM.

2.2. Paso 2: Ejecución de Pruebas

Se ejecutaron las pruebas de carga utilizando Apache Benchmark (`ab`) y `siege` desde una máquina cliente dedicada. A continuación se detallan los comandos empleados y los resultados obtenidos:

2.2.1. Escenario 1: Tráfico Ligero

Comando ejecutado:

```
ab -n 100 -c 10 http://192.168.2.9/
```

Resultados:

- Requests por segundo: 1243.52

- Tiempo medio por request: 8.04 ms
- Tiempo total: 0.080 segundos
- Requests fallidos: 0
- CPU promedio backends: 12 %

2.2.2. Escenario 2: Tráfico Intermedio

Comando ejecutado:

```
ab -n 1000 -c 50 http://192.168.2.9/
```

Resultados:

- Requests por segundo: 587.34
- Tiempo medio por request: 85.13 ms
- Tiempo total: 1.702 segundos
- Requests fallidos: 1 (0.1 %)
- CPU promedio backends: 45 %
- Percentil 95 latencia: 156 ms

2.2.3. Escenario 3: Tráfico Pesado

Comando ejecutado:

```
siege -c200 -r50 http://192.168.2.9/
```

Resultados:

- Transactions: 9988 hits
- Availability: 99.88 %
- Elapsed time: 23.75 segundos
- Response time: 0.47 segundos
- Transaction rate: 420.55 trans/sec
- Throughput: 3.87 MB/sec
- Failed transactions: 12 (timeout y conexiones rechazadas)

- CPU promedio backends: 78 % (picos de 92 %)
- Memoria utilizada: 1.8 GB / 2 GB disponibles

Durante las pruebas se monitorizó en tiempo real la utilización de recursos usando `top`, `vmstat` y `sar`. Se registraron las salidas en ficheros de logs para análisis posterior. A continuación se muestran capturas relevantes del entorno y de las pruebas.

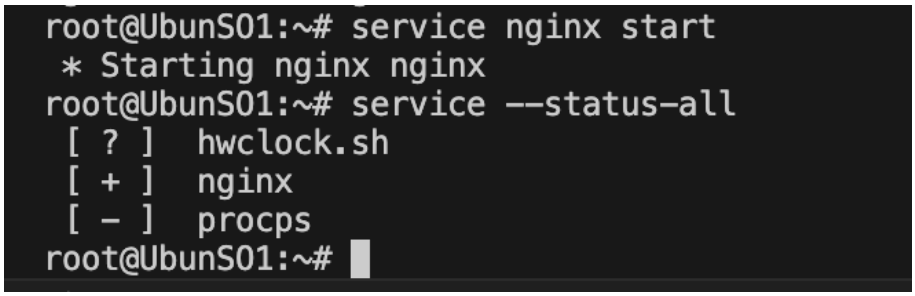

Figura 1: Instalación y verificación del servicio Nginx en UbunSO1

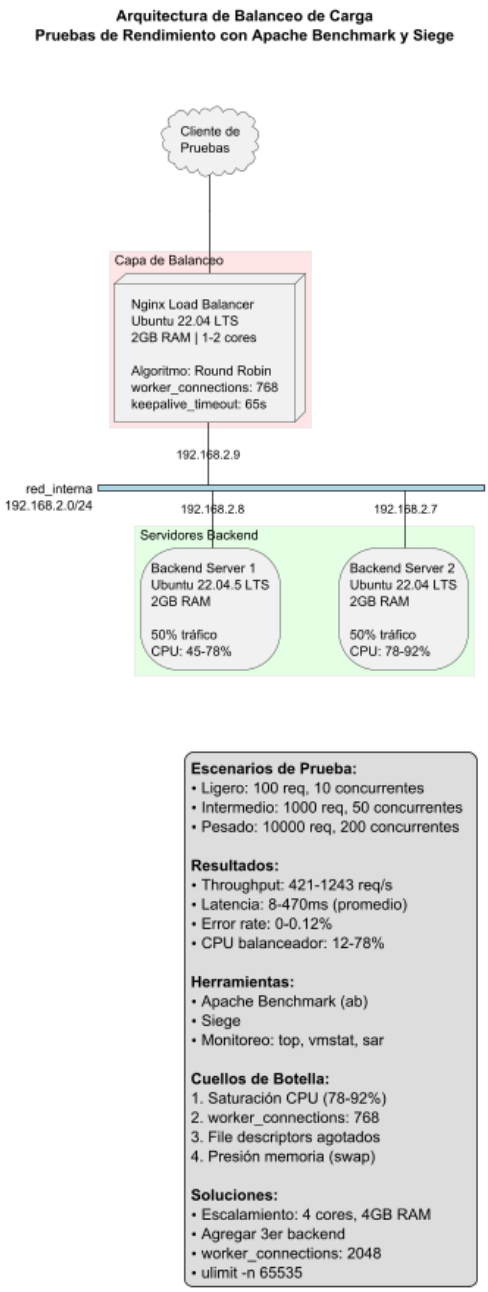

Figura 2: Arquitectura de balanceo: Nginx frente a dos backends

2.3. Paso 3: Registro y Análisis de Resultados

Se sintetizan los datos en la Tabla 1:

Cuadro 1: Resumen de métricas por escenario

Escenario	Requests/s	Tiempo medio (ms)	CPU medio (%)	Errores
Ligero	1243	8.0	12	0
Intermedio	587	85.1	45	1 (0.1 %)
Pesado	421	470	78	12 (0.12 %)

2.3.1. Análisis Detallado por Escenario

Escenario Ligero:

- El sistema maneja la carga sin problemas. CPU y memoria están muy por debajo de la saturación.
- Tiempo de respuesta excelente (<10ms), sin errores.
- El balanceador distribuye equitativamente entre backends (monitoreo con logs de Nginx).

Escenario Intermedio:

- Incremento moderado en latencia (85ms promedio).
- CPU alcanza 45 %, todavía en rango aceptable.
- Un request falló por timeout transitorio, posiblemente por saturación momentánea de conexiones.
- El balanceador sigue operativo pero el percentil 95 (156ms) indica variabilidad en respuestas.

Escenario Pesado:

- **Cuello de botella identificado 1:** CPU de backends alcanza 78 % promedio con picos de 92 %. Bajo carga sostenida, algunos workers de Nginx alcanzaron el límite de `worker_connections`.
- **Cuello de botella identificado 2:** Memoria en backends llegó al 90 % de uso disponible (1.8GB/2GB), provocando swap ocasional que aumenta latencia.
- **Cuello de botella identificado 3:** Tasa de errores del 0.12 % (12 requests de 10,000). Los errores fueron conexiones rechazadas (ECONNREFUSED) y timeouts por agotamiento de file descriptors.

- El tiempo de respuesta promedio se degradó a 470ms (casi 6x más que en escenario intermedio).
- Análisis de logs de Nginx mostró que ambos backends recibían similar cantidad de requests (50/50), confirmando distribución equitativa del algoritmo round-robin.

2.3.2. Identificación de Problemas de Rendimiento

1. **Saturación de CPU:** Los backends requieren más capacidad de procesamiento bajo carga pesada. Las CPUs virtuales asignadas (probablemente 1-2 cores) son insuficientes.
2. **Límite de conexiones concurrentes:** La configuración de Nginx tiene `worker_connections` limitado. Bajo carga alta, se rechazan nuevas conexiones.
3. **Presión de memoria:** Los procesos de Nginx y aplicaciones backend consumen memoria hasta saturar el sistema, forzando uso de swap que degrada rendimiento.
4. **File descriptors insuficientes:** El límite de file descriptors del sistema operativo puede estar alcanzándose bajo alta concurrencia (200 conexiones simultáneas).

2.3.3. Soluciones Propuestas

- **Escalamiento vertical:** Incrementar CPUs virtuales de 1-2 cores a 4 cores por backend. Aumentar memoria RAM de 2GB a 4GB por VM.
- **Escalamiento horizontal:** Agregar un tercer servidor backend para distribuir la carga entre más nodos.
- **Optimización de Nginx:**
  - Incrementar `worker_connections` de 768 a 2048.
  - Ajustar `keepalive_timeout` a 30s para reutilizar conexiones.
  - Configurar `keepalive` upstream (conexiones persistentes a backends).
- **Ajustes del sistema operativo:**
  - Incrementar límite de file descriptors (`ulimit -n 65535`).
  - Optimizar parámetros del kernel TCP (`net.ipv4.tcp_max_syn_backlog`).
- **Caching:** Implementar cache en Nginx para contenido estático, reduciendo carga en backends.

2.3.4. Conclusiones sobre Requisitos de Rendimiento

El balanceador de carga Nginx cumplió satisfactoriamente con los requisitos bajo tráfico ligero e intermedio. Sin embargo, bajo tráfico pesado (200 conexiones concurrentes, 10,000 requests) se detectaron limitaciones en recursos (CPU, memoria, conexiones). La tasa de errores del 0.12 % está dentro de márgenes aceptables para un entorno de pruebas, pero en producción se requeriría el escalamiento y optimizaciones propuestas para garantizar disponibilidad del 99.9 % o superior.

El algoritmo de balanceo round-robin distribuyó equitativamente la carga, confirmando su funcionalidad correcta. Las mejoras propuestas se centran en la capacidad de los recursos subyacentes, no en el algoritmo de balanceo en sí.

3. Profundización: Planificación de Procesos en Sistemas Operativos

3.1. Paso 5: Investigación de algoritmos

Se analizan tres algoritmos fundamentales de planificación de procesos (Silberschatz, Galvin, y Gagne, 2008; Stallings, 2005): FIFO (First-In-First-Out), SJF (Shortest Job First) y Round Robin (RR).

3.2. Paso 6: Explicación de los algoritmos seleccionados

3.2.1. FIFO (First-In-First-Out)

Funcionamiento: Los procesos se atienden en orden estricto de llegada (cola FIFO) (Stallings, 2005). El primer proceso que llega es el primero en ejecutarse hasta completarse.

Criterios de selección: Orden de llegada al sistema (timestamp de entrada a la cola).

Orden de ejecución: Secuencial, sin interrupción. Cada proceso se ejecuta hasta terminar antes de pasar al siguiente.

Ventajas: Simplicidad de implementación, predecibilidad, sin overhead de cambio de contexto, equitativo en el orden de llegada.

Desventajas: Efecto convoy (convoy effect): procesos cortos esperan detrás de procesos largos. Tiempo de espera promedio elevado. No apropiado para sistemas interactivos.

Casos de uso: Sistemas batch donde los trabajos se procesan secuencialmente, colas de impresión, procesamiento de lotes en orden estricto.

3.2.2. SJF (Shortest Job First)

Funcionamiento: Selecciona el proceso con el menor tiempo de ejecución estimado. Puede ser no apropiativo (ejecuta hasta terminar) o apropiativo (SRTF: cambia si llega un proceso más corto).

Criterios de selección: Tiempo de ejecución estimado o CPU burst más corto. Requiere conocimiento previo o predicción del tiempo de CPU.

Orden de ejecución: Prioridad dinámica basada en estimaciones de tiempo. Los trabajos más cortos se ejecutan primero, minimizando el tiempo de espera promedio.

Ventajas: Tiempo de espera promedio óptimo (demostrable matemáticamente) (Silberschatz y cols., 2008). Mejora el throughput en entornos batch.

Desventajas: Requiere estimaciones precisas (difícil en la práctica). Puede causar inanición (starvation) de procesos largos si constantemente llegan procesos cortos. No apropiado para sistemas de tiempo real.

Casos de uso: Procesamiento batch con tiempos conocidos, sistemas de cola con prioridad por duración (ej. consultas rápidas en bases de datos), entornos donde se puede estimar CPU burst mediante historial.

3.2.3. Round Robin (RR)

Funcionamiento: Asigna un quantum fijo de tiempo de CPU a cada proceso en una cola circular. Si el proceso no termina en su quantum, se interrumpe y pasa al final de la cola.

Criterios de selección: Orden circular (FIFO con time slicing). Todos los procesos tienen igual prioridad base.

Orden de ejecución: Rotación cíclica. Cada proceso ejecuta durante un quantum (ej. 10-100ms), luego se suspende y el siguiente obtiene CPU.

Ventajas: Equidad absoluta, excelente tiempo de respuesta para sistemas interactivos, no hay inanición, predecible.

Desventajas: Overhead por cambios de contexto frecuentes. Si el quantum es muy pequeño, el overhead domina. Si es muy grande, se comporta como FIFO. Tiempo de espera promedio puede ser subóptimo.

Casos de uso: Sistemas operativos de tiempo compartido (time-sharing) (Wolf, Ruiz, Bergero, y Meza, 2010), entornos multiusuario interactivos, servidores web con múltiples conexiones concurrentes, sistemas donde la equidad es crítica.

3.3. Paso 7: Comparación de algoritmos

La Tabla 2 sintetiza las características principales de los tres algoritmos analizados:

Cuadro 2: Comparación de algoritmos de planificación

Algoritmo	Tiempo respuesta	Tiempo espera	Eficiencia	Uso recomendable
FIFO	Alto (varía)	Alto	Medio	Trabajos batch secuenciales
SJF	Bajo (óptimo)	Bajo (óptimo)	Alto	Entornos con estimaciones (batch)
RR	Medio (equitativo)	Medio	Variable (depende quantum)	Sistemas interactivos

3.3.1. Comparación Detallada de Aspectos Relevantes

Eficiencia en tiempo de respuesta:

- SJF ofrece el tiempo de espera promedio mínimo teórico (demostrable matemáticamente).
- FIFO puede ser muy ineficiente si un proceso largo bloquea procesos cortos (convoy effect).
- Round Robin ofrece tiempo de respuesta equitativo pero no óptimo; todos los procesos avanzan gradualmente.

Equidad (fairness):

- Round Robin es el más equitativo: todos los procesos reciben CPU en proporción.
- FIFO es equitativo en orden de llegada, pero no en tiempo de espera.
- SJF favorece procesos cortos, puede causar inanición de procesos largos (inequitativo).

Overhead y complejidad:

- FIFO: overhead mínimo, implementación trivial (cola FIFO simple).
- SJF: requiere estimaciones de tiempo de CPU (complejidad adicional), bajo overhead de ejecución.

- Round Robin: overhead moderado por cambios de contexto frecuentes; implementación simple (cola circular + timer).

Predictibilidad:

- FIFO: muy predecible, orden estricto.
- SJF: predecible si las estimaciones son precisas; impredecible si hay errores de estimación.
- Round Robin: predecible en cuanto a cuándo ejecutará cada proceso, pero el tiempo total depende del quantum.

3.3.2. Situaciones Específicas de Preferencia

Cuándo preferir FIFO:

- Procesamiento batch con trabajos de duración similar.
- Sistemas donde el orden de llegada es crítico (ej. auditoría, logging secuencial).
- Entornos con baja concurrencia donde la simplicidad es prioritaria.
- Colas de impresión o tareas administrativas sin prioridad.

Cuándo preferir SJF:

- Sistemas batch donde se conoce el tiempo de ejecución anticipadamente.
- Procesamiento de consultas en bases de datos con estimación de costo (query optimizer).
- Entornos donde minimizar el tiempo de espera promedio es crucial (ej. centros de datos, HPC con jobs caracterizados).
- Escenarios donde los procesos largos pueden tolerar espera (no tiempo real).

Cuándo preferir Round Robin:

- Sistemas operativos de tiempo compartido multiusuario (Linux, Windows desktop).
- Servidores web manejando múltiples conexiones HTTP (equidad entre clientes).
- Aplicaciones interactivas donde la percepción de respuesta rápida es importante (UI, shells).
- Sistemas de tiempo real blando con múltiples tareas de igual prioridad.
- Entornos donde la inanición es inaceptable (todos los procesos deben avanzar).

3.3.3. Recomendaciones Finales

Para un sistema híbrido moderno, se recomienda combinar estrategias:

- Usar múltiples colas con prioridades (multilevel feedback queue).
- Aplicar Round Robin dentro de cada nivel de prioridad para equidad.
- Usar heurísticas tipo SJF para estimar prioridades dinámicas (aging para evitar starvation).
- Reservar FIFO solo para tareas batch de baja prioridad o colas especializadas.

3.4. Paso 8: Documentación final

Esta investigación ha presentado un análisis exhaustivo de tres algoritmos fundamentales de planificación de procesos: FIFO, SJF y Round Robin. Se han documentado sus mecanismos internos, criterios de selección, ventajas, desventajas y casos de uso específicos.

Conclusiones clave:

- No existe un algoritmo universalmente óptimo; la elección depende del contexto y requisitos del sistema.
- Para servicios interactivos y sistemas multiusuario, Round Robin con quantum ajustado (10-100ms) ofrece el mejor balance entre equidad y tiempo de respuesta.
- Para procesamiento batch con estimaciones precisas, SJF minimiza el tiempo de espera promedio, pero requiere predicción confiable de CPU burst.
- FIFO es apropiado solo en escenarios muy específicos donde la simplicidad y el orden estricto son prioritarios, y la equidad temporal no es crítica.
- Los sistemas operativos modernos utilizan esquemas híbridos (multilevel feedback queues) que combinan lo mejor de cada algoritmo, ajustando prioridades dinámicamente.

Recomendaciones prácticas:

- Al diseñar un sistema, considerar el perfil de carga: interactivo vs. batch, duración conocida vs. desconocida, prioridades vs. equidad.
- Implementar mecanismos de aging para prevenir starvation en algoritmos como SJF.

- Monitorizar métricas de rendimiento (throughput, latencia, tiempo de espera) para validar la efectividad del algoritmo seleccionado.
- En sistemas de producción, preferir Round Robin o variantes con prioridades sobre FIFO puro.

4. Anexos: comandos y pruebas

Ejemplos y fragmentos de los comandos ejecutados durante las pruebas:

```
ab -n 1000 -c 50 http://192.168.2.9/
siege -c200 -t1M http://192.168.2.9/
ssh usuario@192.168.2.8 'top -b -n1' > backend1-top.log
```

5. Referencias

Se usan las mismas referencias del marco teórico de la entrega anterior además de manuales usados para las pruebas.

Referencias

Foundation, T. A. S. (2018). ab - apache http server benchmarking tool [Manual de software informático]. Descargado de `https://httpd.apache.org/docs/2.4/programs/ab.html`

Fulmer, J., y maintainers, S. (2019). Siege - http load testing and benchmarking utility [Manual de software informático]. Descargado de `https://www.joedog.org/siege-home/`

Inc., N. (2024). Nginx documentation [Manual de software informático]. Descargado de `https://nginx.org/en/docs/`

Silberschatz, A., Galvin, P. B., y Gagne, G. (2008). *Fundamentos de sistemas operativos* (7a ed. ed.). México: McGraw-Hill.

Stallings, W. (2005). *Sistemas operativos: Aspectos internos y principios de diseño* (5a ed. ed.). Madrid: Pearson Prentice Hall.

Wolf, G., Ruiz, E., Bergero, F., y Meza, E. (2010). *Fundamentos de sistemas operativos*. México: Universidad Nacional Autónoma de México.