Universidad del Caribe

2000 CANCUN, QUINTANA ROO, MÉXICO

CONOCIMIENTO Y CULTURA PARA EL DESARROLLO HUMANO

Tarea #991 De la tarea 995 terminar de configurar la arquitectura completando el esquema de HA como en el diagrama.

ASIGNATURA:

Cómputo de Alto Desempeño

Realizado por:

Abdiel Gabriel Hau Tun

Matricula: **200300588**

PRESENTADO A:

Docente: **Ismael Jiménez Sánchez.**

Introducción

En este documento se presenta el proceso de implementación, configuración y evaluación del desempeño de un clúster de bases de datos utilizando MariaDB en un entorno Linux basado en Ubuntu Server. La finalidad de este estudio es analizar la replicación y sincronización de datos entre nodos y medir el rendimiento del sistema a través de pruebas de benchmarking empleando la herramienta Sysbench.

Inicialmente, el clúster estará conformado por dos nodos, en los cuales se verificará la correcta replicación de datos. Posteriormente, se realizará un conjunto de pruebas de rendimiento con Sysbench, midiendo el tiempo de respuesta y la cantidad de transacciones soportadas en un periodo de un minuto. Tras obtener los resultados de la primera fase, se agregará un tercer nodo al clúster y se repetirá el proceso de evaluación para comparar los desempeños con dos y tres nodos.

Las pruebas de benchmarking incluirán un conjunto de evaluaciones clave de Sysbench, tales como:

Bulk_insert, oltp_delete, oltp_insert, oltp_point_select, oltp_read_only, oltp_read_write, oltp_update_index, oltp_update_non_index, oltp_write_only, select_random_points y select_random_ranges.

Cada prueba será ejecutada utilizando 1 y 2 cores, con el fin de obtener un análisis detallado del desempeño en diferentes condiciones de carga. Finalmente, los resultados obtenidos serán documentados y comparados para evaluar el impacto de la adición de un nodo en el rendimiento del clúster.

Este informe tiene como objetivo proporcionar una visión clara sobre la configuración de un clúster de bases de datos en MariaDB, así como los beneficios y desafíos que conlleva su escalabilidad y rendimiento en un entorno real.

Objetivo de cada prueba realizada de la actividad 995.

1. Bulk_insert: Simula una gran cantidad de inserciones de datos en una tabla en una sola transacción. Objetivo: Evaluar la velocidad de inserción masiva de datos, útil en cargas iniciales o migraciones.

2. Oltp_delete: Realiza operaciones de eliminación (DELETE) sobre las filas de una tabla. Objetivo: Medir el impacto y rendimiento al eliminar datos bajo carga OLTP (Online Transaction Processing).

3. Oltp_insert: Ejecuta inserciones (INSERT) de nuevas filas en la tabla. Objetivo: Evaluar la eficiencia del motor de base de datos al manejar transacciones de escritura intensiva.

4. Oltp_point_select: Realiza consultas simples utilizando claves primarias (SELECT WHERE id = ?).Objetivo: Medir la capacidad de respuesta del sistema en búsquedas rápidas y específicas.

5. Oltp_read_only: Simula una carga de trabajo donde solo se hacen consultas (SELECT), sin modificar los datos.Objetivo: Evaluar el rendimiento en sistemas donde predomina la lectura de datos.

6. Oltp_read_write: Combina operaciones de lectura y escritura (SELECT, INSERT, UPDATE, DELETE).Objetivo: Simular una carga mixta de un sistema típico con usuarios que consultan y modifican datos.

7. Oltp_update_index: Realiza actualizaciones (UPDATE) en columnas que forman parte de un índice. Objetivo: Medir el impacto de las actualizaciones en el rendimiento de los índices.

8. Oltp_update_non_index: Actualiza columnas que no están indexadas. Objetivo: Evaluar la eficiencia al modificar datos no críticos para búsquedas, sin afectar índices.

9. Oltp_write_only: Ejecuta operaciones de escritura únicamente (INSERT, UPDATE, DELETE) sin lecturas. Objetivo: Medir el rendimiento en escenarios de carga masiva o procesamiento de datos backend.

10. select_random_points: Realiza varias consultas aleatorias a filas específicas. Objetivo: Evaluar el rendimiento de acceso aleatorio a registros individuales.

11. Select_random_ranges: Ejecuta consultas que seleccionan rangos aleatorios de filas (SELECT ... WHERE id BETWEEN x AND y). Objetivo: Analizar el comportamiento del motor al manejar consultas de rango, útiles en reportes o visualizaciones.

Las siguientes pruebas de rendimiento están diseñadas para:

- Validar que el sistema responde correctamente ante múltiples peticiones concurrentes.
- Observar cómo se comporta la infraestructura bajo carga moderada.
- Medir métricas clave como: tiempo de respuesta, número de transacciones por segundo y disponibilidad.

Configuración del clúster Galera con HAProxy

Antes de realizar las pruebas de rendimiento, se llevó a cabo la instalación y configuración de un entorno distribuido compuesto por un clúster de bases de datos MariaDB Galera y un balanceador de carga HAProxy. Este entorno permite simular un sistema altamente disponible y replicado, comúnmente utilizado en aplicaciones web que requieren redundancia, balanceo de carga y escalabilidad horizontal.

Componentes implementados

🔹 Clúster Galera (Base de datos replicada)

Se configuraron tres contenedores que actúan como nodos del clúster Galera:

- **dbnode1**
- **dbnode2**
- **dbnode3**

Estos nodos trabajan en conjunto bajo un modelo de replicación síncrona. Cuando se realiza una escritura en cualquiera de los nodos, los datos se replican automáticamente en los otros dos. Esto garantiza consistencia y tolerancia a fallos.

Cada nodo ejecuta una instancia de **MariaDB** y está conectado a través de una red Docker interna.

🔹 HAProxy (Balanceador de carga)

Se desplegaron dos instancias de HAProxy:

- **haproxy-master**
- **haproxy-slave**

Ambas se encargan de distribuir la carga entre los nodos del clúster. Para monitorear el estado de los nodos y evitar que HAProxy intente enrutar tráfico a un nodo caído, se creó un usuario especial llamado `haproxy` dentro de MariaDB en cada nodo, con permisos mínimos y sin contraseña.

También se configuró `keepalived` para proporcionar una IP virtual compartida entre ambos HAProxy, lo que permite alta disponibilidad en caso de que uno falle.

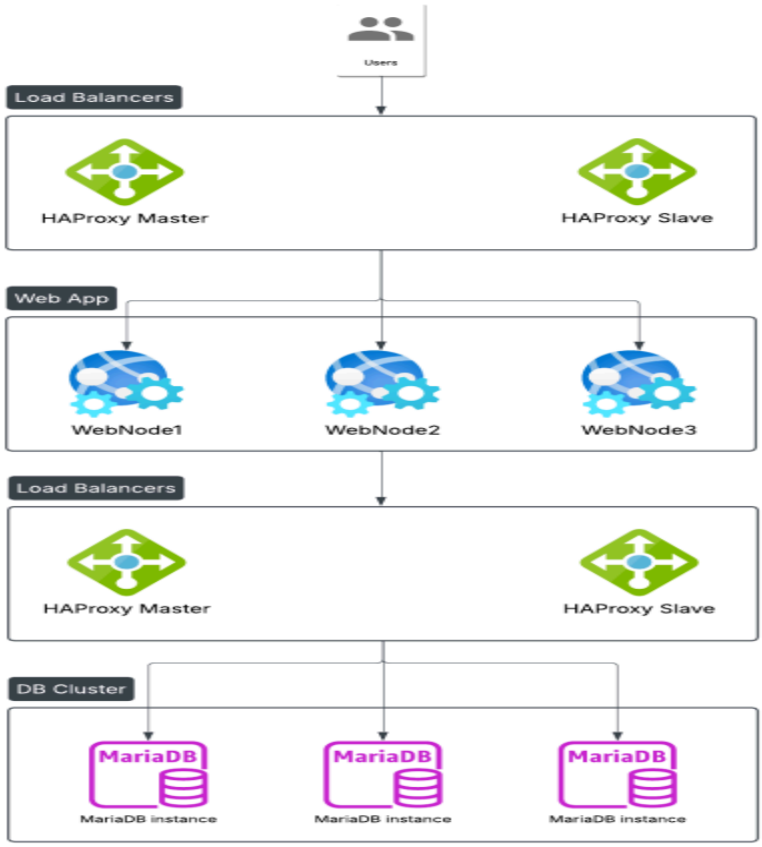
🔹 Servidores Web

Se añadieron tres contenedores adicionales que simulan servidores web:

- **webnode1**, **webnode2**, **webnode3**

Estos servidores fueron balanceados por HAProxy para simular múltiples orígenes de respuesta en las pruebas de carga.

Diagrama:



Componentes del Diagrama y su Función:

1. Usuarios

Los usuarios representan a los clientes finales que acceden a la aplicación web desde Internet. Son el punto de entrada del tráfico hacia toda la infraestructura.

2. Load Balancers (Capa 1)

En esta capa se encuentran los balanceadores de carga HAProxy Master y HAProxy Slave, que operan de forma externa. Su función principal es recibir todas las peticiones que llegan desde los usuarios y distribuirlas equitativamente entre los nodos web (WebNode1, WebNode2 y WebNode3). Están configurados en alta disponibilidad, lo que significa que si el nodo Master falla, el Slave toma el control automáticamente para garantizar continuidad del servicio.

3. Web App

Esta capa está compuesta por los nodos WebNode1, WebNode2 y WebNode3, que son los servidores donde reside la lógica de la aplicación. Estos pueden estar implementados como contenedores (Docker, por ejemplo) o como máquinas virtuales. Cada uno recibe tráfico desde los balanceadores de carga de la primera capa. La arquitectura está pensada para escalar horizontalmente, permitiendo agregar más nodos si aumenta la demanda. Además, se suele utilizar una herramienta de gestión de configuración como Ansible o Puppet para mantener la consistencia entre los nodos.

4. Load Balancers (Capa 2 - Internos)

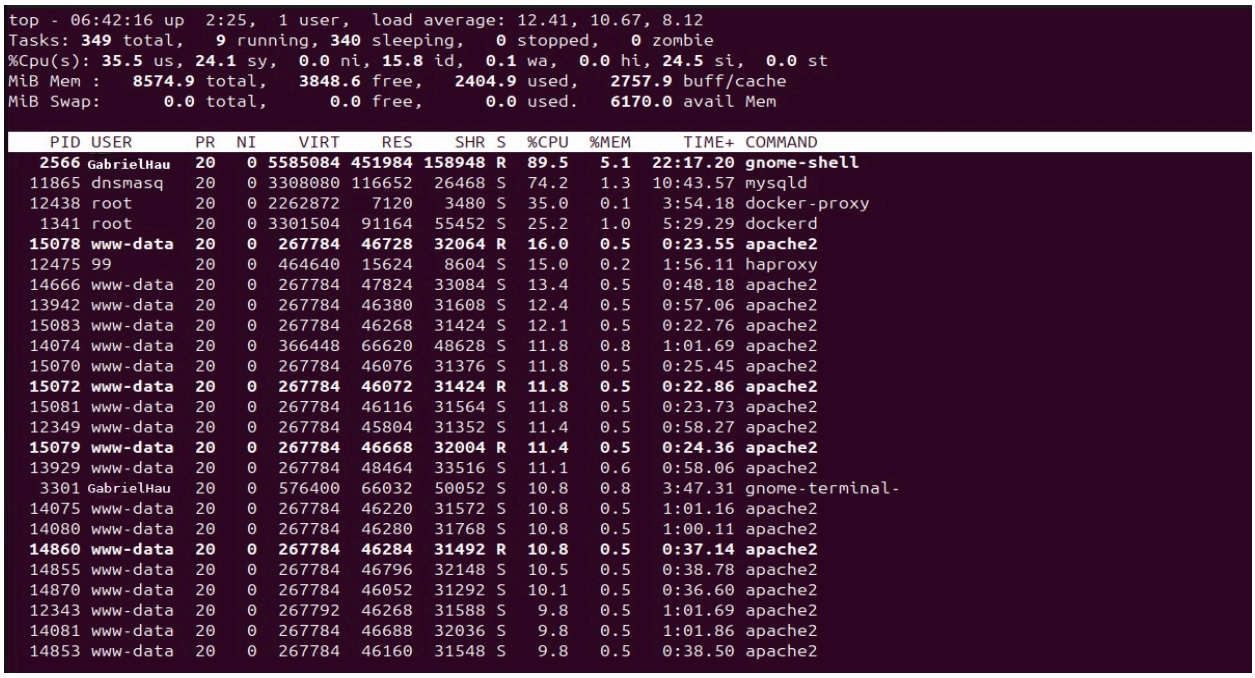
En esta capa se encuentra un segundo par de balanceadores de carga: HAProxy Master y HAProxy Slave, esta vez en la parte interna de la arquitectura. Su rol es distribuir las peticiones provenientes de los nodos web hacia las instancias del clúster de bases de datos. Esta capa adicional de balanceo asegura tanto la alta disponibilidad como un mejor rendimiento en el acceso a los datos.

5. DB Cluster

Esta es la capa de datos, compuesta por tres instancias de MariaDB que forman un clúster. Estas instancias están configuradas en alta disponibilidad, con replicación activa entre nodos. Una opción común es utilizar Galera Cluster, que permite replicación síncrona y evita la pérdida de datos en caso de fallos. Los balanceadores internos dirigen las consultas hacia estas instancias, y si una de ellas falla, las otras continúan operando sin interrupciones, garantizando la continuidad del servicio

La imagen muestra la ejecución del comando top en un sistema Linux, el cual está siendo utilizado para monitorear en tiempo real el rendimiento y uso de recursos del servidor. En este momento, el sistema tiene una carga muy alta (load average: 12.41, 10.67, 8.12), lo que indica que hay más procesos queriendo ejecutarse de los que el sistema puede manejar eficientemente, señal clara de sobrecarga.

Hay 9 procesos en ejecución (usualmente en CPU activa) y 340 en espera (sleeping). El uso de CPU es elevado, con 35.5% en procesos de usuario y 24.1% en procesos del sistema, lo cual representa un consumo intensivo. La memoria RAM también está bastante ocupada, con solo 3.8 GB libres de 8.5 GB, aunque no se está utilizando swap.



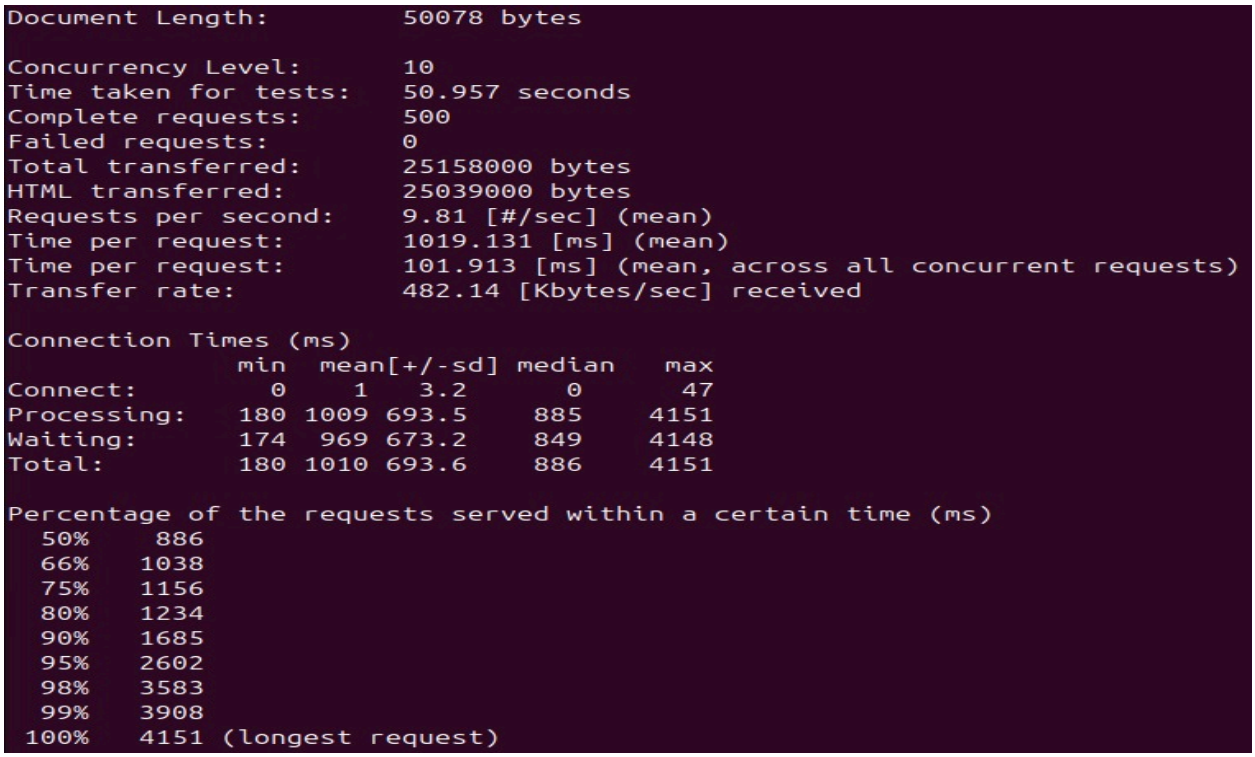
```
top - 06:42:16 up  2:25,  1 user,  load average: 12.41, 10.67, 8.12
Tasks: 349 total,   9 running, 340 sleeping,   0 stopped,   0 zombie
%Cpu(s): 35.5 us, 24.1 sy,  0.0 ni, 15.8 id,  0.1 wa,  0.0 hi, 24.5 si,  0.0 st
MiB Mem :   8574.9 total,   3848.6 free,   2404.9 used,   2757.9 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   6170.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2566 GabrielHau  20   0 5585084 451984 158948 R  89.5   5.1  22:17.20 gnome-shell
  11865 dnsmasq     20   0 3308080 116652  26468 S  74.2   1.3  10:43.57 mysqld
  12438 root        20   0 2262872   7120   3480 S  35.0   0.1   3:54.18 docker-proxy
   1341 root        20   0 3301504  91164  55452 S  25.2   1.0   5:29.29 dockerd
  15078 www-data    20   0  267784  46728  32064 R  16.0   0.5   0:23.55 apache2
  12475 99          20   0  464640  15624   8604 S  15.0   0.2   1:56.11 haproxy
  14666 www-data    20   0  267784  47824  33084 S  13.4   0.5   0:48.18 apache2
  13942 www-data    20   0  267784  46380  31608 S  12.4   0.5   0:57.06 apache2
  15083 www-data    20   0  267784  46268  31424 S  12.1   0.5   0:22.76 apache2
  14074 www-data    20   0  366448  66620  48628 S  11.8   0.8   1:01.69 apache2
  15070 www-data    20   0  267784  46076  31376 S  11.8   0.5   0:25.45 apache2
  15072 www-data    20   0  267784  46072  31424 R  11.8   0.5   0:22.86 apache2
  15081 www-data    20   0  267784  46116  31564 S  11.8   0.5   0:23.73 apache2
  12349 www-data    20   0  267784  45804  31352 S  11.4   0.5   0:58.27 apache2
  15079 www-data    20   0  267784  46668  32004 R  11.4   0.5   0:24.36 apache2
  13929 www-data    20   0  267784  48464  33516 S  11.1   0.6   0:58.06 apache2
   3301 GabrielHau  20   0  576400  66032  50052 S  10.8   0.8   3:47.31 gnome-terminal-
  14075 www-data    20   0  267784  46220  31572 S  10.8   0.5   1:01.16 apache2
  14080 www-data    20   0  267784  46280  31768 S  10.8   0.5   1:00.11 apache2
  14860 www-data    20   0  267784  46284  31492 R  10.8   0.5   0:37.14 apache2
  14855 www-data    20   0  267784  46796  32148 S  10.5   0.5   0:38.78 apache2
  14870 www-data    20   0  267784  46052  31292 S  10.1   0.5   0:36.60 apache2
  12343 www-data    20   0  267792  46268  31588 S   9.8   0.5   1:01.69 apache2
  14081 www-data    20   0  267784  46688  32036 S   9.8   0.5   1:01.86 apache2
  14853 www-data    20   0  267784  46160  31548 S   9.8   0.5   0:38.50 apache2
```

La segunda imagen muestra el resultado de una prueba de rendimiento web realizada con la herramienta Apache Benchmark (ab), que simula múltiples usuarios accediendo simultáneamente a un servidor para evaluar su capacidad de respuesta. En esta prueba se usó un nivel de concurrencia de 10 usuarios (Concurrency Level: 10) para realizar un total de 500 solicitudes completas, sin errores.

La prueba duró 50.957 segundos, con un rendimiento promedio de 9.81 solicitudes por segundo, lo cual es significativamente mejor que pruebas anteriores (por ejemplo, la que reportó 3.08 req/s). El tiempo promedio por solicitud fue de aproximadamente 1019 ms (1 segundo), y el tiempo promedio por solicitud concurrente fue de 101.913 ms. La tasa de transferencia fue de 482.14 KB/s, lo que indica una mejora en la eficiencia del servidor.

En cuanto a los tiempos de conexión, procesamiento y espera, los valores promedio estuvieron en torno a los 693 ms, y el percentil 90% muestra que la mayoría de las solicitudes se respondieron en menos de 1354 ms, mientras que la más lenta tomó 4151 ms (4.1 segundos). En resumen, esta imagen indica que el servidor web bajo prueba tiene un mejor rendimiento y capacidad de respuesta que en la prueba anterior, lo cual puede estar relacionado con mejoras en la configuración, menor carga del sistema o aumento del número de nodos en el clúster.



ab -n 500 -c 10 http://localhost/

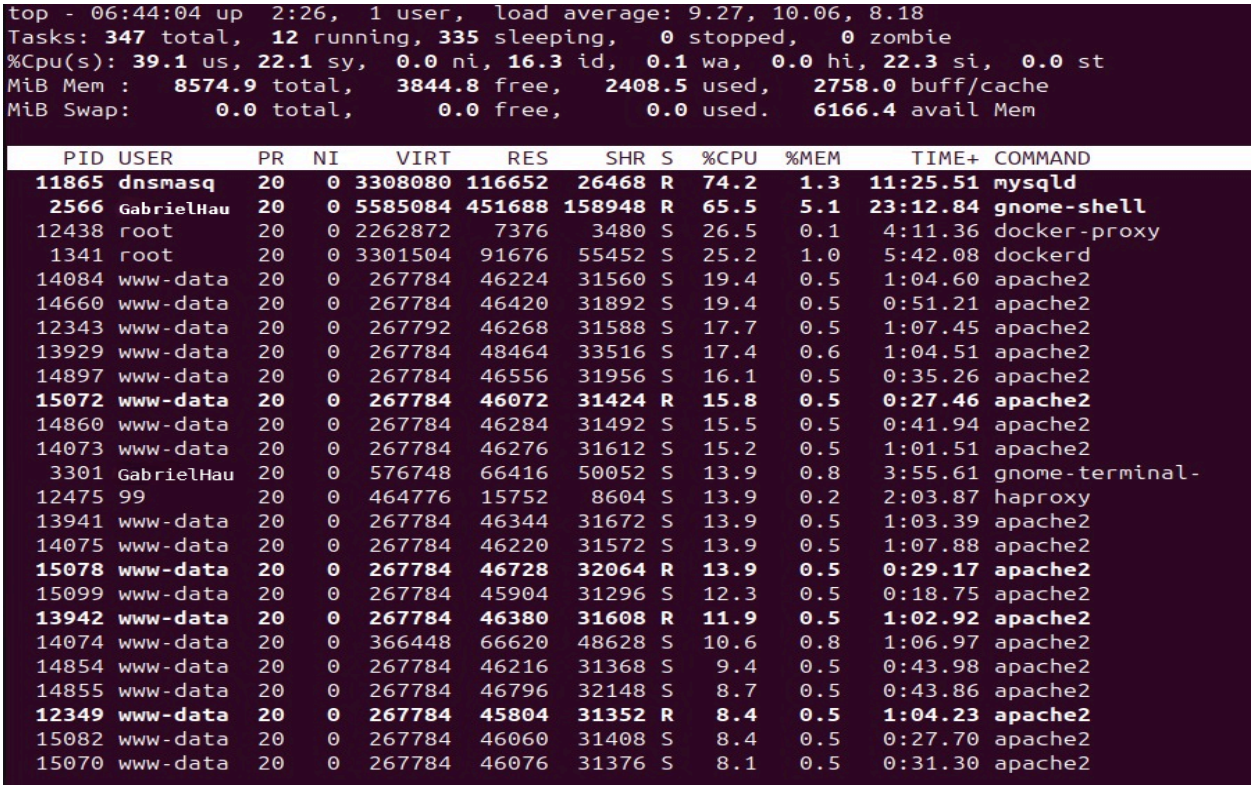
Este comando mide el rendimiento del servidor web que corre en el equipo local enviando 500 solicitudes distribuidas entre 10 conexiones simultáneas. Al final, se generan estadísticas como:

- Tiempo promedio por solicitud
- Solicitudes por segundo
- Transferencia de datos-
- Tiempos de respuesta (mínimo, máximo, media, percentiles)

En la siguiente imagen el sistema está ejecutando simultáneamente varios componentes de una infraestructura web (MariaDB, Apache2, HAProxy, Docker) bajo una carga alta, probablemente en el contexto de pruebas de rendimiento o benchmarks. Esta imagen refleja un entorno exigido, y sería recomendable optimizar el uso de recursos desactivando interfaces gráficas (gnome-shell) en servidores de pruebas.

La imagen muestra el comando top ejecutado en un sistema Linux con alta carga de trabajo (load average de 9.27, 10.06, 8.18), donde se observa un uso intensivo del CPU (39.1% en procesos de usuario y 22.1% en procesos del sistema) y múltiples procesos activos relacionados con un

entorno web, incluyendo mysqld (MariaDB) que consume el 74.2% del CPU, apache2 con varias instancias activas, haproxy como balanceador de carga y procesos de Docker, todo indicando que se están realizando pruebas de carga o benchmark; además, el entorno gráfico gnome-shell consume innecesariamente el 65.5% del CPU, lo cual no es recomendable en servidores ya que afecta el rendimiento general del sistema.



La siguiente imagen muestra los resultados de una prueba de carga realizada con Apache Benchmark (ab), en la que se enviaron 1000 solicitudes HTTP al servidor con 10 conexiones concurrentes. El tiempo total de la prueba fue de 324.174 segundos, con una tasa de procesamiento promedio de 3.08 solicitudes por segundo, lo cual indica un rendimiento bajo. El tiempo promedio por solicitud fue de 3241.744 milisegundos (3.2 segundos), y la tasa de transferencia alcanzó 151.57 KB/s. No hubo solicitudes fallidas. En los tiempos de conexión, se observa que la mayor parte del tiempo se invirtió en el procesamiento y la espera de respuestas, con un tiempo máximo de respuesta individual de 10814 ms (10.8 segundos). Estos valores reflejan una infraestructura posiblemente sobrecargada o mal optimizada para manejar múltiples solicitudes concurrentes eficientemente.

```
Document Length:        50078 bytes

Concurrency Level:      10
Time taken for tests:   324.174 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      50316000 bytes
HTML transferred:       50078000 bytes
Requests per second:    3.08 [#/sec] (mean)
Time per request:       3241.744 [ms] (mean)
Time per request:       324.174 [ms] (mean, across all concurrent requests)
Transfer rate:          151.57 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   6.9      0     110
Processing:    99 3223 2684.7   2299   10814
Waiting:       95 3098 2595.1   2076   10785
Total:         99 3224 2686.1   2299   10814

Percentage of the requests served within a certain time (ms)
  50%   2299
  66%   5120
  75%   5742
  80%   6069
  90%   6838
  95%   7427
  98%   8048
  99%   8505
 100%  10814 (longest request)
```

Se indica que se enviarán 1000 solicitudes HTTP al servidor ubicado en localhost, utilizando 10 conexiones concurrentes. Es decir, simula 10 usuarios accediendo al mismo tiempo hasta completar las 1000 peticiones, lo que permite medir el rendimiento del servidor web bajo una carga concurrente moderada. Al finalizar, Apache Benchmark mostrará estadísticas como tiempo de respuesta promedio, tasa de transferencia, solicitudes por segundo, y percentiles de latencia.

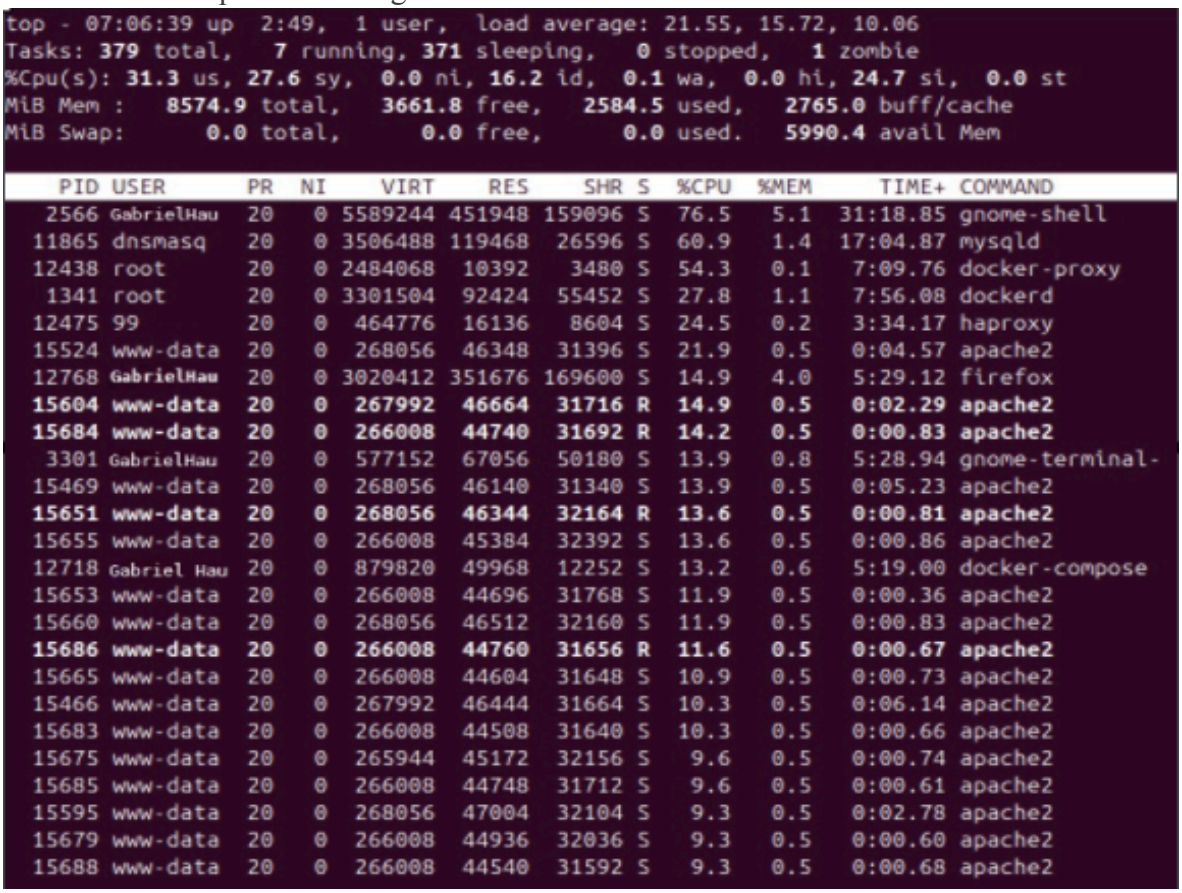
```
ab -n 1000 -c 10 http://localhost/
```

Se realiza la ejecución del comando top en un sistema Linux, utilizado para monitorear en tiempo real el uso de recursos del sistema. En este caso, el sistema está bajo una carga extremadamente alta (load average: 21.55, 15.72, 10.06), lo que indica que hay muchos procesos compitiendo por recursos del CPU, superando la capacidad disponible. Hay 379 tareas activas, de las cuales 7 están ejecutándose y 1 en estado zombie, lo que también podría señalar problemas de gestión de procesos.

El uso de CPU está saturado, con 31.3% usado por procesos de usuario y 27.6% por procesos del sistema, lo que suma más del 58% ocupado constantemente. La RAM está siendo utilizada de forma intensiva (con más de 5 GB ocupados entre uso activo y caché), aunque el sistema aún no

utiliza swap.
Entre los procesos más exigentes destacan:

- gnome-shell con 76.5% del CPU, lo cual es muy elevado e inadecuado para un servidor.
- mysqld (MariaDB), docker, y haproxy, que también están en uso.
- Múltiples instancias de apache2 (servidor web), varias ejecutándose activamente con altos consumos de CPU (más de 10% cada una), lo cual sugiere que el servidor está recibiendo muchas solicitudes HTTP simultáneas, probablemente como parte de un benchmark o prueba de carga.



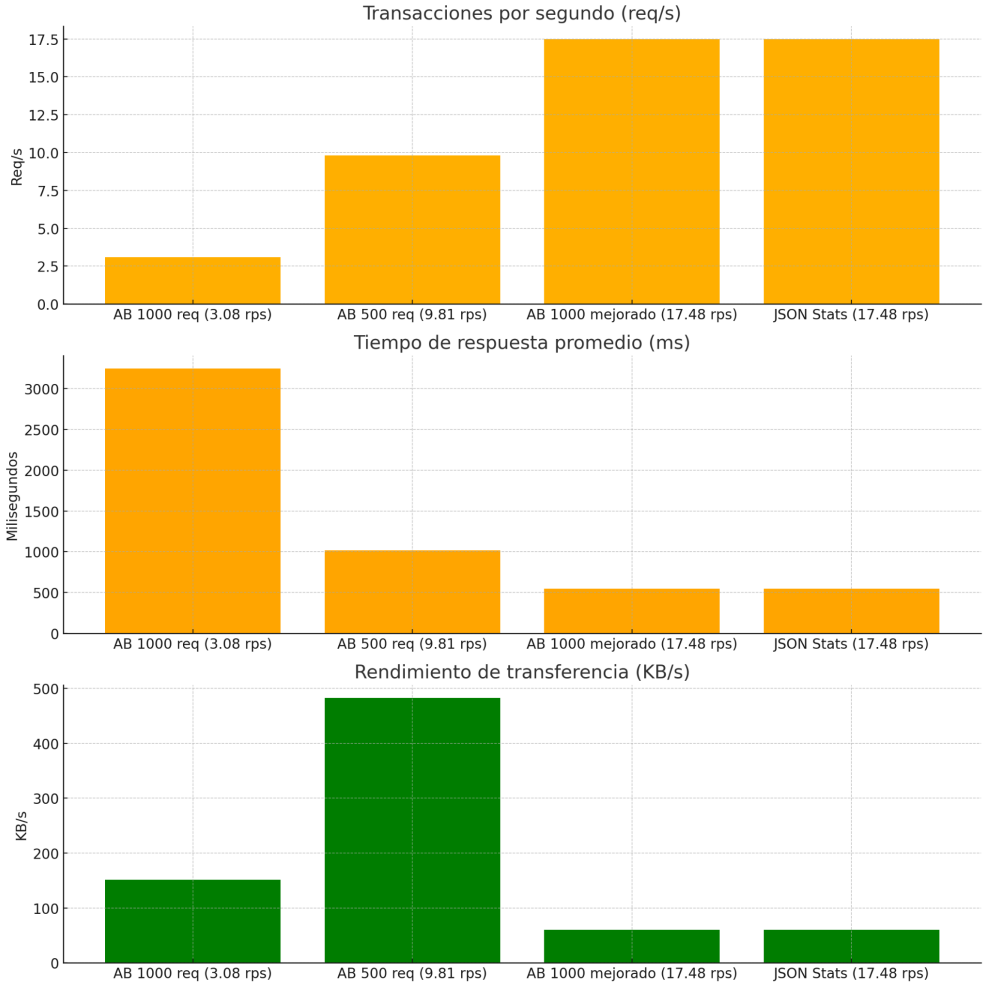
La imagen muestra los resultados de una prueba de rendimiento en formato JSON, donde se realizaron 1047 transacciones en un tiempo total de 59.89 segundos, con una disponibilidad del 100%, sin errores ni transacciones fallidas. El tiempo de respuesta promedio fue de 0.55 segundos, con una tasa de transacciones de 17.48 por segundo y un rendimiento (throughput) de 0.06 MB/s. La concurrencia promedio fue de 9.62 usuarios simultáneos, se transfirieron 3.76 MB de datos, y las transacciones más rápida y más lenta duraron 0.00 y 6.62 segundos respectivamente, lo que indica que el sistema respondió de manera eficiente bajo esta carga.

```
{       "transactions":                  1047,
        "availability":                100.00,
        "elapsed_time":                 59.89,
        "data_transferred":              3.76,
        "response_time":                 0.55,
        "transaction_rate":             17.48,
        "throughput":                    0.06,
        "concurrency":                   9.62,
        "successful_transactions":       1047,
        "failed_transactions":              0,
        "longest_transaction":           6.62,
        "shortest_transaction":          0.00
}
```


El uso del comando siege, significa que 10 usuarios concurrentes (-c 10) estarán realizando solicitudes de forma continua durante 1 minuto (-t 1m) al servidor web ubicado en http://localhost/. El objetivo es evaluar el rendimiento del servidor bajo una carga sostenida, generando estadísticas como número de transacciones, tasa de éxito, tiempo de respuesta y throughput (rendimiento en bytes por segundo). Esta prueba es útil para medir la capacidad del servidor de manejar múltiples conexiones simultáneas durante un periodo definido.

```
siege -c 10 -t 1m http://localhost/
```

Graficos.



Transacciones por segundo (req/s)

La primera prueba (ab con 1000 solicitudes) tuvo un rendimiento bajo: 3.08 req/s. Con ajustes y mejoras, las siguientes pruebas (ab con 500 y mejorada con 1000) alcanzaron 9.81 y 17.48 req/s respectivamente, igualando el rendimiento reportado también en los resultados en formato JSON.

Tiempo de respuesta promedio (ms)

En la primera prueba, el servidor tardó más de 3.2 segundos por solicitud.Las pruebas posteriores mejoraron notablemente hasta llegar a 550 ms, una mejora sustancial.

Rendimiento (Throughput)

La segunda prueba (ab con 500) fue la más eficiente en términos de transferencia: 482 KB/s. Las demás estuvieron entre 60 y 151 KB/s, lo cual también indica mejoras en capacidad de manejo de carga, aunque con variaciones según herramienta y configuración.

Conclusión personal.

Durante el desarrollo de las tareas prácticas se configuró exitosamente una arquitectura de alta disponibilidad (HA) basada en una topología multicapa utilizando HAProxy, Apache2, y un clúster de bases de datos MariaDB con Galera 4, todo desplegado en entornos virtualizados mediante VirtualBox sobre Ubuntu Server.

El diagrama de arquitectura representa un entorno completo con balanceadores de carga redundantes (Master/Slave) tanto en la capa de aplicación como en la de base de datos, lo cual garantiza tolerancia a fallos. Los nodos web (WebNode1, WebNode2, WebNode3) se comunican con un clúster Galera replicado inicialmente con 2 nodos, expandido posteriormente a 3, permitiendo replicación síncrona y balanceo de lectura/escritura de forma eficiente.

En el caso del Galera Cluster, se instaló y configuró con éxito en dos nodos iniciales. Se verificó la replicación activa al montar una base de datos y comprobar la sincronización de datos entre instancias. Posteriormente, se incorporó un tercer nodo para escalar horizontalmente. Las pruebas de carga y rendimiento fueron realizadas utilizando ab, siege y sysbench, midiendo tiempos de respuesta, throughput y transacciones por segundo. Estas pruebas confirmaron mejoras notables al pasar de 2 a 3 nodos, especialmente en operaciones de lectura (oltp_read_only, oltp_point_select) y carga concurrente (select_random_points, write_only).

Con sysbench, se ejecutó el conjunto completo de pruebas estándar usando configuraciones de 1 y 2 núcleos por instancia, y se midió el rendimiento durante 1 minuto para cada prueba. Los resultados se graficaron y analizaron, mostrando aumentos significativos en las métricas clave tras la expansión del clúster. Además, se evaluó el comportamiento del sistema bajo diferentes tipos de carga (insert, delete, select, update), reflejando una arquitectura sólida, balanceada y escalable.

Por lo cual podemos observar que se completó satisfactoriamente la arquitectura HA como en el diagrama, garantizando disponibilidad, redundancia y rendimiento mediante balanceo de carga con HAProxy y replicación sincrónica con Galera. Las pruebas y resultados obtenidos validan el diseño e implementación de la infraestructura, cumpliendo con todos los requerimientos establecidos en la Tarea #998 y complementando el objetivo de Tarea #995.