Programación Distribuida
y
Tiempo Real

Práctica 2

AÑO 2022 – 2° Semestre

Referencias

https://developer.hashicorp.com/vagrant/docs

1. **En caso de haber hecho el ej. 3 de la práctica 1 en una única computadora, rehacer el experimento de manera tal que se utilicen 2 computadoras. Aclare si las computadoras están en la misma red local o en dos redes locales diferentes (es posible que deban modificar al menos un router en la mayoría de los casos). Pueden utilizar las computadoras del aula o sus propias computadoras.**

   Antes de realizar las pruebas de comunicación entre sockets en el escenario donde se comuniquen dos computadoras diferentes, se decide actualizar el código implementado que fue presentado en la entrega anterior, para posteriormente realizar el experimento sobre la nueva implementación y en base a esos resultados, compararlo con los demás experimentos solicitados en la entrega.

   Cambios realizados:

   - Se implementa en el programa del cliente una estructura “for” en donde se realiza el envío y recepción de información que realizan tanto el cliente con el servidor, donde por defecto se iterará en base a una variable “loop” seteada con el valor 100.
   - Se mejora el método de asignación de datos a cada posición en la variable buffer, donde ahora se lleva a cabo mediante la función strcat().
   - Se permite agregar el tamaño que poseerá la variable buffer mediante argumentos que serán pasados al programa tanto del lado de la implementación del cliente, como del servidor.
   - Se analiza que los datos que recibe el cliente sean idénticos a los que envió al servidor para evaluar la integridad de los datos.
   - Gracias a la iteración en la comunicación implementada, se aprovecha para llevar a cabo el cálculo del promedio y desviación estándar.
   - Se cambia la forma de medir los tiempos, en donde ahora el programa del cliente es el único que mide el tiempo, y lo cuenta desde antes de que comience la transferencia de información hasta posteriormente de recibir información enviada por el servidor. En donde se espera que el tamaño de información enviada y recibida sea la misma.

   Además de los cambios mencionados, se desarrollaron unos pequeños scripts (“client.sh” y “server.sh”) en donde se automatiza el experimento que se quiere realizar, donde se calcula tanto el promedio como desviación estándar para cada uno de los tamaños del buffer solicitados

(siendo de 10^1 a 10^6 bytes), y se persiste la información obtenida sobre un archivo “informe.txt” para su posterior análisis.
En base a todos estos nuevos cambios, se procede a realizar el experimento en el escenario donde tanto el proceso cliente como el proceso servidor serán ejecutados sobre una misma computadora.

En este caso al ejecutar los scripts, el client.sh requiere un argumento, que es la ip del servidor, por lo que se indica “localhost”.

Resultados del primer experimento:

Tiempo promedio de comunicación	
bytes	segundos
10	1.008e-05
100	1.0915e-05
1000	8.695e-06
10000	9.73e-06
100000	0.000102115
1000000	0.00149058

Desviación estándar	
bytes	segundos
10	8.40825e-06
100	2.24192e-05
1000	9.03933e-06
10000	2.9643e-06
100000	4.2265e-05
1000000	0.000710234

Se presenta la información en formato de gráfico en donde el eje x representa el tamaño de información que se envía desde un proceso al

otro y el eje y representa el tiempo empleado para completar esa transferencia.

Tiempo promedio en comunicación entre procesos de una misma máquina

Desviación estándar entre procesos de una misma máquina

Ahora se realizará el mismo experimento, pero donde cada proceso se encuentre en una computadora diferente. En este escenario al ejecutar los scripts, en el client.sh se debe pasar el ip de la computadora donde se encuentre corriendo el proceso del servidor.

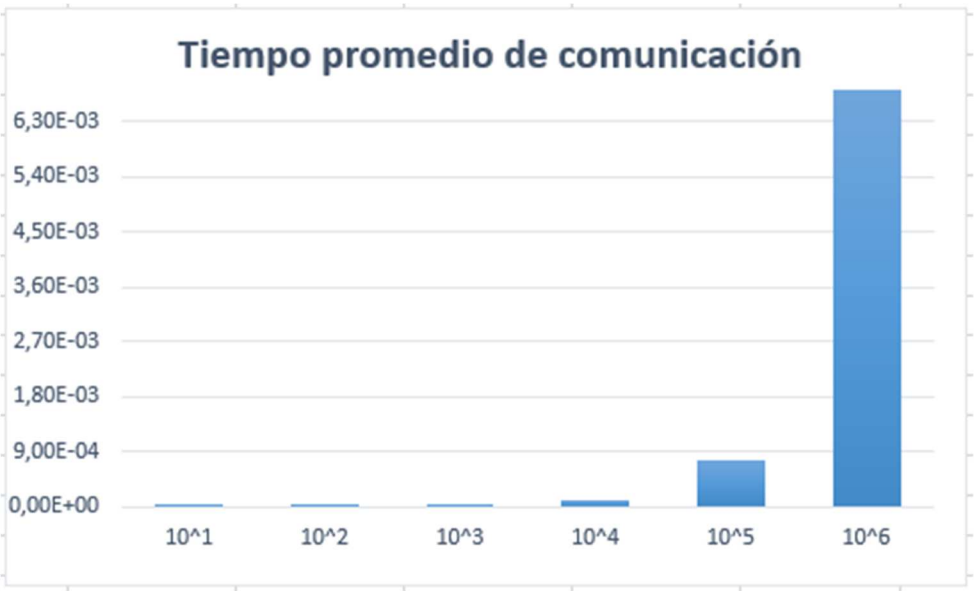
La dirección IP de una computadora puede identificarse mediante el comando “ifconfig” ejecutado sobre una terminal. Y para verificar que dos computadoras se comunican, se puede ejecutar desde una computadora externa el comando “ping ‘IP’”, donde IP es la obtenida en el primer comando indicado.

El experimento fue realizado sobre las computadoras brindadas por la biblioteca de la facultad.
Resultados del segundo experimento:

Tiempo promedio de comunicación	
bytes	segundos
10	2.6325e-05
100	2.6765e-05
1000	2.885e-05
10000	9.9385e-05
100000	0.000745605
1000000	0.0068158

Desviación estándar	
bytes	segundos
10	2.92179e-06
100	3.07771e-06
1000	3.15e-06
10000	2.01385e-05
100000	4.11402e-05
1000000	0.000353738

Se vuelve a presentar la información en formato de gráfico para observar de mejor forma las diferencias en tiempos entre las distintas comunicaciones de información. Recordando que el eje x representa el tamaño de la información transferida y el eje y representa el tiempo empleado en completar dicha transferencia.



Tiempo promedio en comunicación entre procesos de diferentes máquinas

Desviación estándar entre procesos de diferentes máquinas

La conclusión que se puede obtener en base a los experimentos es:

- Por un lado, el evidente aumento exponencial de tiempos que requieren las transferencias cuando estos deben de transportar datos de tamaños considerables, debido a la imposibilidad de poder transportar la información completa. Esto también explica el por qué en las transferencias que involucran datos de menor tamaño, estos pueden transportarse en una única transferencia, por lo que los tiempos no difieren demasiado en esos casos.
- Por el otro lado, se observa que se obtuvieron tiempos más extensos en el experimento donde se empleaban dos

computadoras en comparación del experimento donde solo se requería una única computadora.

2. **Teniendo en cuenta el ej.3 de la práctica 1, desarrollar scripts para desplegar un ambiente de experimentación de comunicaciones en una computadora con Vagrant para los siguientes escenarios:**
   a. **Dos máquinas virtuales, cada una con un proceso de comunicaciones.**
   b. **Una máquina virtual con uno de los procesos de comunicaciones y el otro proceso de comunicaciones en el host.**

   En todos los casos deberían dejar los resultados disponibles para su posterior análisis.

   Para poder desplegar uno o varios ambientes de desarrollo mediante la herramienta de Vagrant, se implementa un Vagrantfile tomando como base el brindado por la cátedra, y se trabaja en base a ese ejemplo, donde en este caso se decide, entre otras modificaciones, por unas máquinas virtuales “ubuntu/xenial64” ya brindada por la comunidad que utiliza la herramienta, y se asigna manualmente las IPs que poseerán cada máquina, para facilitar posteriormente las comunicaciones requeridas para los experimentos.

   Para las pruebas de este enunciado se emplearán los scripts y programas desarrollados para el ejercicio anterior. Es decir, los cambios implementados en “client.c” y “server.c” y la integración de sus respectivos scripts “client.sh” y “server.sh”

   Inciso a.

   Para el primer escenario solicitado, se desarrolló un script llamado “vagantV1.sh” el cual se encargará de levantar las dos máquinas virtuales solicitadas mediante el comando “vagrant up” posteriormente de ejecutar el comando “vagrant halt” para apagar las máquinas, por si estas se encontraban en actividad.

   Secuencia de actividades que realiza “vagrantV1.sh”:
   - Ingresa a la consola de la primera máquina virtual llamada vm1 mediante el comando “vagrant ssh vm1 -c” y ejecuta el script “server.sh” para que se quede en espera de peticiones para atender.

- Ejecuta un “sleep 10” para darle el tiempo suficiente a la máquina vm1 de ejecutarse y quedarse en espera de peticiones antes de proseguir con las demás instrucciones.
- Ingresa a la consola de la segunda máquina virtual llamada vm2 y ejecuta el script “client.sh” donde le pasará como argumento la IP "192.168.0.201". Esto funciona gracias a que esta IP fue declarada en el vagrantfile.
- Al finalizar todas las pruebas de transferencias, ingresa a la consola de la vm2 y copia el archivo “informe.txt” generado sobre la carpeta compartida con el nombre “informe_dos_maquinas_virtuales.txt” para su posterior estudio.

Resultados del experimento:

Tiempo promedio de comunicación	
bytes	segundos
10	5.178e-05
100	3.3835e-05
1000	4.1805-e05
10000	9.201e-05
100000	0.000424925
1000000	0.00463872

Desviación estándar	
bytes	segundos
10	3.62668e-05
100	2.30855e-05
1000	1.48667e-05
10000	2.74496e-05
100000	0.000104647
1000000	0.00102529

Al igual que en el ejercicio anterior, se presenta la información en formato de gráfico para poder comparar los tiempos empleados de mejor manera.

- Eje X: tamaño en bytes.
- Eje Y: tiempo empleado.

Tiempo promedio en comunicación entre procesos de diferentes máquinas virtuales

Desviación estándar entre procesos de diferentes máquinas virtuales

Inciso b.

Para el segundo escenario solicitado, se desarrolló un script llamado “vagrantV2.sh” el cual trabaja muy similarmente al “vagrantV1.sh” anteriormente explicado, con la diferencia de que solo levanta una única

máquina virtual, en este caso vm1, y en vez de utilizar una segunda máquina virtual, las instrucciones las ejecuta el host. Por lo que la secuencia de ejecución del script es idéntica a la anterior.
El archivo que generará al finalizar el script se llama "informe_maquina_virtual_y_host.txt".

Resultados del experimento:

Tiempo promedio de comunicación	
bytes	segundos
10	1.2665e-05
100	1.3355e-05
1000	1.343e-05
10000	6.6395e-05
100000	0.00016136
1000000	0.00296228

Desviación estándar	
bytes	segundos
10	5.30474e-06
100	7.62669e-06
1000	3.57563e-06
10000	3.07783e-05
100000	4.68334e-05
1000000	0.000785349

Se procede a mostrar los datos obtenidos en forma de gráfico.

Eje X: tamaño en bytes.
Eje Y: tiempo empleado.

Tiempo promedio en comunicación entre procesos de máquina virtual y host

Desviación estándar entre procesos de máquina virtual y host

Como indica en el enunciado, al finalizar cada Script, se genera en la carpeta compartida brindada por Vagrant un informe para su posterior análisis.

Se observa en los resultados de los experimentos en ambos escenarios, que las comunicaciones resultan ser ligeramente inferiores en donde se comunica el host con una máquina virtual.
Por lo que se cree que, en el primer escenario, demanda más tiempo cuando el host debe funcionar como intermediario entre ambas máquinas virtuales para que estas puedan realizar sus comunicaciones.

3. **Explique si los resultados de tiempo de comunicaciones del experimento del ej. anterior se ven afectados por el orden de ejecución y las diferencias de tiempo de ejecución iniciales de los programas utilizados. ¿Qué sucede si por alguna razón hay una diferencia de 10 segundos entre el inicio de la ejecución entre un programa y otro? Sugerencia: incluya un "sleep" de 10 segundos entre la ejecución de diferentes partes de los programas y comente.**

Sincronización en la comunicación.

Como se observa en la comunicación de sockets, estos requieren tanto de un proceso que reciba peticiones de parte de otros procesos que realizan las peticiones según las vayan requiriendo, por lo que se los llama proceso servidor y cliente respectivamente.

Para una correcta comunicación, se requiere de mantener una sincronización para, de esta forma, obtener los resultados esperados. Por lo que, al no respetar dicha sincronización esperada, tanto los procesos cliente y servidor variarán en su resultado dependiendo de quien ejecute primero ciertas instrucciones.

Un ejemplo de una mala sincronización se presenta cuando se pierden peticiones cuando el proceso Cliente intenta realizar una conexión mediante el comando connect(), cuando el procesos Servidor aun no ejecuto el comando listen() para encontrarse disponible para la espera y atención de posibles peticiones, por lo que se produce un error en la comunicación por parte del proceso cliente.

En la solución dada en los incisos anteriores, durante la implementación de los scripts que levantan las máquinas virtuales de acuerdo al inciso dado, "vagrantV1.sh" y "vagrantV2.sh", se presenciaron errores de sincronización en las comunicaciones cuando se llevan a cabo la ejecución de los scripts "client.sh" y "server.sh", lo que provoca obtener distintos resultados:

- Termina la ejecución correctamente, pero se llegan a perder ciertas comunicaciones de las seis que deben realizarse.
- El programa no termina su ejecución debido a que el servidor queda esperando una petición de parte del cliente, cuando este ya realizó las peticiones establecidas.
- Los procesos se sincronizan correctamente y finaliza la ejecución de acuerdo con los resultados esperados.

En este caso, se solucionó el problema de sincronización agregando a ambos scripts el comando “sleep 10” para que el proceso se durmiera durante ese periodo y así permitirle al proceso servidor que compile y ejecute hasta estar listo para recibir peticiones. Y que, por lo tanto, al despertarse y ejecutar al proceso cliente, el servidor ya se encontrará listo para recibir las comunicaciones.

Variación de tiempos según el escenario

Independientemente de si existen errores de sincronización que produzcan pérdidas de peticiones, o se logran llevar a cabo todas las comunicaciones correctamente, el tiempo empleado en dichas comunicaciones no se verán afectados, y estos serán los mismos, ya que empiezan a calcularse una vez se haya establecido la comunicación entre los sockets correspondientes.

En base a esa forma de tomar los tiempos establecidos, se puede observar variación dependiendo del entorno donde se esté realizando las pruebas. Por ejemplo, va a influenciar la velocidad de la red utilizada o también el sistema operativo o hardware en donde se estén ejecutando los procesos y el manejo en base a la organización de procesos que realiza la misma, en donde el proceso que se encuentra ejecutando, puede ser expulsado de la CPU y demorar indefinidamente para volver a tomarla y finalizar su ejecución.

También se observa, que durante la primera prueba el sistema requiera levantar a memoria información necesaria y realizar cómputo adicional que ralentiza las comunicaciones tomadas en una primera evaluación, y que posteriormente en unas posteriores repeticiones del experimento el tiempo obtenido como respuesta de las comunicaciones sean ligeramente inferiores.