PROGRAMACIÓN DISTRIBUIDA Y TIEMPO REAL

TRABAJO PRÁCTICO N°2

Juan Cruz Cassera Botta 17072/7
Lucio Bianchi Pradas 19341/8

1) **Buscar y utilizar en vagrant una versión más reciente de Ubuntu LTS que la dada en la explicación de práctica. Identificar versión, origen y si tuviera diferencias, explicar brevemente.**

El archivo vagrantfile de la explicación de práctica crea dos máquinas virtuales con sistema operativo Ubuntu 18.04 LTS (Bionic64). Esto se específica en ésta línea de código:

machine.vm.box = "hashicorp/bionic64"

Como esta versión es bastante antigua (publicada en abril de 2018) y ya no tiene más soporte estándar, vamos a modificar esta línea para usar una versión más reciente de Ubuntu. En nuestro caso, elegimos Ubuntu 22.04 LTS, la cual tiene el codename Jammy Jellyfish y fue publicada en abril de 2022. Entonces creamos una copia del vagrantfile con solo la línea de código anterior modificada de esta forma:

machine.vm.box = "generic/ubuntu2204"

Versión

Como mencionamos, la versión de Ubuntu original es la 18.04, y la que vamos a usar es la 22.04.

Origen

La versión 18.04 que proveyó la cátedra proviene de la cloud de HashiCorp, mientras que la que elegimos nosotros, la 22.04, proviene de la cloud de "generic" que está administrada por roboxes.

Diferencias

Las principales diferencias entre estas dos versiones de Ubuntu son:

1. Kernel: La versión 18.04 usa el Linux Kernel versión 4.15, mientras que la 22.04 usa la versión 5.15, la cual provee mejor soporte de hardware, optimizaciones de rendimiento, seguridad mejorada, drivers de GPU mejorados, etc.

2. GNOME: La versión 22.04 usa una versión de GNOME mucho más nueva, lo cual mejora notablemente todo lo que tiene que ver con lo visual: animaciones, dark mode, escalado, mejor soporte para múltiples monitores, etc.
3. Software: La versión 18.04 usa versiones antiguas de software para desarrolladores, como por ejemplo Python 3.6, GCC 7, etc. Ubuntu 22.04 usa versiones mucho más actualizadas de todo este tipo de software.

2) **En caso de haber hecho los experimentos de tiempos de comunicaciones 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 (en este último caso, es posible que deban modificar al menos un router si ambas computadoras tienen acceso vía NAT). Pueden utilizar las computadoras del aula o sus propias computadoras. En todos los casos provean una descripción mínima de las computadoras usadas (CPU, RAM, OS).**

Para este experimento, utilizamos 2 computadoras distintas en una **misma red local**, y usamos el mismo código de Cliente y Servidor (Java) que el que usamos en el ejercicio 3 del TP anterior.

Server (Archivo ServerEjercicio3.java)**:**

El servidor, una computadora de escritorio, estaba conectado a internet a través de Ethernet.

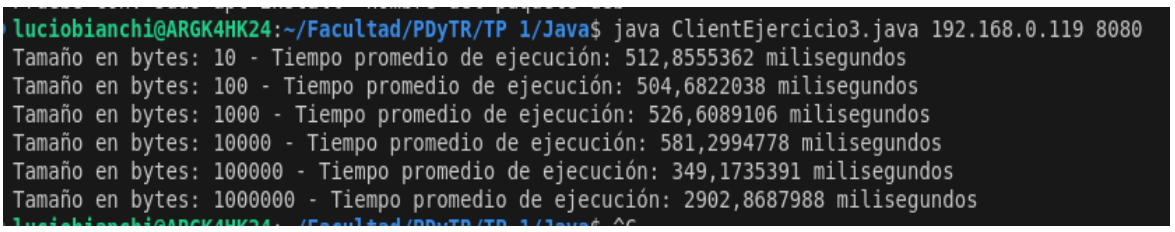
CPU: AMD Ryzen 5 5600X (6 núcleos 12 hilos)
RAM: 16 GB
OS: Windows 11 Home

Cliente (Archivo ClientEjercicio3.java):

El cliente, una laptop, estaba conectado a internet a través de Wi-Fi.

CPU: Intel Core i7 1370p (14 núcleos 20 hilos)
RAM: 32 GB
OS: Ubuntu 22.04.4 LTS

Los resultados de tiempo de ejecución fueron muchísimo más lentos que en el experimento usando una única PC:



Debajo comparamos los tiempos de ejecución obtenidos en el experimento del TP anterior vs los obtenidos ahora. El método para calcular el tiempo fue exactamente el mismo que el explicado en el TP anterior.

Tamaño en bytes	**Tiempo de ejecución usando una única PC**	**Tiempo de ejecución usando 2 PCs (red local)**
10	3.19 ms	512.8 ms
100	2.37 ms	504.59 ms
1000	2.04 ms	526.60 ms
10000	2.26 ms	581.29 ms
100000	2.90 ms	349.17 ms
1000000	13.36 ms	2902.86 ms

3) **Teniendo en cuenta los experimentos de tiempos realizados, 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 comunicación.**

Para realizar los experimentos de tiempos en dos máquinas virtuales, cada una con un proceso de comunicación, desarrollamos el siguiente script para conectarse por ssh a vm1 y vm2 y ejecutar los respectivos .java del servidor y del cliente:

```
$vm1 = "vm1"
$vm2 = "vm2"
$server_ip = "192.168.0.187"
$server_port = "8080"

# Empezar el servidor vm1 en background
Write-Output "Comenzando servidor en $vm1..."
Start-Process -NoNewWindow -FilePath "vagrant" -ArgumentList
"ssh $vm1 -c 'cd ../../vagrant && java ServerEjercicio3.java
$server_port'" -PassThru

# Ejecutar el cliente en vm2
Write-Output "Comenzando cliente en $vm2..."
vagrant ssh $vm2 -c "cd ../../vagrant && java
ClientEjercicio3.java $server_ip $server_port"

Write-Output "Fin."
```

En cuanto a los tiempos de comunicación, obtuvimos los siguientes resultados:

```
Tamaño en bytes: 10 - Tiempo promedio de ejecución: 435.5220180 milisegundos
Tamaño en bytes: 100 - Tiempo promedio de ejecución: 439.9390582 milisegundos
Tamaño en bytes: 1000 - Tiempo promedio de ejecución: 439.6788921 milisegundos
Tamaño en bytes: 10000 - Tiempo promedio de ejecución: 15.8314566 milisegundos
Tamaño en bytes: 100000 - Tiempo promedio de ejecución: 18.7019359 milisegundos
Tamaño en bytes: 1000000 - Tiempo promedio de ejecución: 73.7127712 milisegundos
```

Tamaño en bytes	Tiempo de ejecución usando 2 VMs
10	435.52 ms
100	439.93 ms
1000	439.67ms
10000	15.83 ms
100000	18.70 ms
1000000	73.71 ms

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.

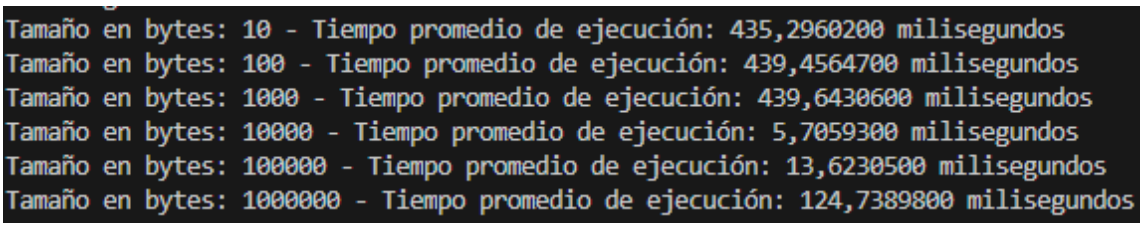
```
$vm1 = "vm1"
$server_ip = "192.168.0.187"
$server_port = "8080"

# Empezar el servidor vm1 en background
Write-Output "Comenzando servidor en $vm1..."
Start-Process -NoNewWindow -FilePath "vagrant" -ArgumentList
"ssh $vm1 -c 'cd ../../vagrant && java ServerEjercicio3.java
$server_port'" -PassThru

# Ejecutar el cliente en el Host
Write-Output "Comenzando cliente en el Host"
java ClientEjercicio3.java $server_ip $server_port

Write-Output "Fin."
```

En cuanto a los tiempos de comunicación, obtuvimos los siguientes resultados:



Tamaño en bytes	**Tiempo de ejecución usando una VM y un host**
10	435.29 ms
100	439.45 ms
1000	439.64ms
10000	5.70 ms
100000	13.62 ms
1000000	124.73 ms

Resumiendo, estos son los tiempos comparados entre los 3 casos:

Caso 1) Usando 2 PCs físicas (bare-metal) en una misma red local

Caso 2) Usando 2 máquinas virtuales con Vagrant

Caso 3) Usando una máquina virtual (servidor) y una PC física (cliente)

Tamaño en bytes	**Tiempo de ejecución usando 2 PCs físicas**	**Tiempo de ejecución usando 2 VMs**	**Tiempo de ejecución usando una VM y una PC física**
10	512.8 ms	435.52 ms	435.29 ms
100	504.59 ms	439.93 ms	439.45 ms
1000	526.60 ms	439.67ms	439.64ms
10000	581.29 ms	15.83 ms	5.70 ms
100000	349.17 ms	18.70 ms	13.62 ms
1000000	2902.86 ms	73.71 ms	124.73 ms

4) **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 en 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.

   Si el cliente se ejecuta mucho después que el servidor, no afecta a los tiempos de comunicaciones, ya que estos tiempos (según nuestra metodología de t1 - t0 definida en el TP anterior) empiezan a contabilizarse **después de que el cliente se conecta con el servidor**: el tiempo empieza a correr cuando el cliente le empieza a enviar datos al servidor.

   Además, en nuestro programa si se quiere ejecutar el cliente antes de que el servidor esté corriendo, ocurre un fallo al crear el Socket. En el caso de querer correr en 2 procesos distintos, paralelamente al servidor y al cliente deberíamos asegurarnos que el servidor esté corriendo y pueda responder las request antes de que el cliente se conecte. Para ello modificamos los scripts anteriores para que ejecuten 2 procesos distintos, y agregamos un sleep de 10 segundos para asegurarnos que el servidor esté corriendo antes de que el cliente se intente conectar:

```powershell
$vm1 = "vm1"

$vm2 = "vm2"

$server_ip = "192.168.0.187"

$server_port = "8080"

# Empezar el servidor vm1 en background

Write-Output "Comenzando servidor en $vm1..."

$serverProcess = Start-Process -NoNewWindow -FilePath "vagrant" -ArgumentList "ssh $vm1 -c 'cd ../../vagrant && java ServerEjercicio3.java $server_port'" -PassThru



# Asegura que comienza el servidor antes del cliente

Start-Sleep -Seconds 10



# Ejecutar el cliente en vm2 en background

Write-Output "Comenzando cliente en $vm2..."

$clientProcess = Start-Process -NoNewWindow -FilePath "vagrant" -ArgumentList "ssh $vm2 -c 'cd ../../vagrant && java ClientEjercicio3.java $server_ip $server_port'" -PassThru



# Esperar a que ambos procesos terminen

Write-Output "Esperando a que los procesos terminen..."

$serverProcess.WaitForExit()

$clientProcess.WaitForExit()



Write-Output "Fin."
```