Tarea 3 - Sistemas Operativos

Integrantes: Phanor Castillo, María Camila Guzmán, Carlos Orduz, Juan Diego Rojas

Tabla de contenido:

1. Máquinas Virtuales

Para el desarrollo del ejercicio, se construyeron tres máquinas virtuales (cada una con el mismo sistema operativo Linux) mediante el uso de la herramienta KVM/QEMU. A continuación se explicará la configuración de cada una, en el orden en que se sugirió que fueran realizadas.

1.1. Servidor Web Apache

```
$ qemu-system-x86_64 -enable-kvm \
    -cpu host \
    -drive file=Apache-VM.qcow2,if=virtio \
    -netdev user,id=net1 \
    -device virtio-net,netdev=net1,mac=e2:71:ce:a1:9a:07 \
    -m 1G \
    -smp 1 \
```

En la imagen anterior se muestra el comando que inicia la máquina virtual.[1]

En esencia, la primera línea indica que se va a emular una arquitectura de 64 bits y luego activa el KVM (Kernel-based Virtual Machine). La siguiente línea configura la máquina virtual para que tenga la misma configuración del procesador físico de la computadora en la que se ejecuta. Luego, se especifica el archivo de imagen del disco virtual que contiene el sistema operativo y el servidor web Apache configurado. Se utilizó QCOW2 por su soporte de snapshots y su eficiencia en el uso del disco. También se especifica que se usará la interfaz de disco “Virtio”. La siguiente línea crea un dispositivo de red sin necesidad de configuraciones complejas de host. Luego se le asigna una dirección MAC a la máquina, y finalmente se asignan la cantidad de procesadores y de RAM.

```
# pacman -Syu
# pacman -S apache git
```

```
$ git clone https://github.com/JDRV17/JDRV17.github.io.git
$ mv JDRV17.github.io/* /srv/http/
```

```
# systemctl enable httpd.service
# systemctl start httpd.service
```

Los tres pares de comandos anteriores se encargan de los siguientes pasos del proceso. En el primero par, se sincronizan y actualizan la base de datos de paquetes del sistema y se instalan el servidor web Apache y Git. El siguiente par muestra cómo se clona un repositorio de GitHub, y cómo se mueven los archivos

de la página al directorio /srv/http/, que es la ubicación predeterminada donde Apache sirve los archivos web en Arch Linux [4]. Finalmente, el último par de comandos configura, habilita e inicia Apache.

1.2. Servidor FTP usando Docker-Compose

En su gran mayoría, la parte inicial de el proceso con esta máquina virtual es muy parecida a la anterior que usó los servidores web de Apache. La máquina virtual se inicia con prácticamente el mismo comando, a excepción de la línea 3, que en su lugar, ahora cambia por la siguiente:

```
-drive file=Docker-VM.qcow2,if=virtio \
```

Adicionalmente, en las parejas de comandos que se mostraron para los siguientes pasos hubo unos cuantos cambios, pero en general también se sigue la misma estructura (con la diferencia de que esta vez no se necesita clonar el repositorio de GitHub).[2]

```
# pacman -Syu
# pacman -S docker docker-compose

    # sytsemctl enable docker
    # systemctl start docker
```

Como se puede apreciar, los comandos que usan *pacman* son prácticamente iguales a excepción claro, de que ya no se tiene que instalar Apache y Git sino Docker y Docker-Compose. La habilitación y el inicio de Docker sí funcionan de manera paralela a lo previamente enseñado en Apache. De aquí en adelante se complican más las cosas.

Una vez Docker-Compose está instalado, se crea un directorio para poder utilizarlo. Asimismo, se aprovecha para que queden creados de una vez otros dos directorios que serán necesarios para el servidor FTP. Esto se hace con los siguientes comandos:

```
$ mkdir -p ~/docker
$ mkdir -p ~/ftp_data
$ mkdir -p ~/ftp_config
```

Luego se crea el archivo passwd que contiene las cuentas de usuario FTP, para finalmente, crear el archivo .yml del Docker-Compose.

```
user:password::user:/home/user:/bin/bash
```

```
services:
    ftp_server:
        image: stilliard/pure-ftpd
        container_name: ftpServer
        ports:
            - "21:21"
            - "30000-30009:30000-30009"
        volumes:
            - "/home/arch/ftp_data:/home/arch"
            - "/home/arch/ftp_config/passd:/etc/ftpServer/
passwd
        environment:
            PUBLICHOST: "localhost"
            FTP_USER_NAME: user
            FTP_USER_PASS: password
            FTP_USER_HOME: "/home/arch"
        restart: always
```

Este último bloque[3.] muestra los comandos para la configuración de un servidor FTP. En este caso, la imagen de Docker utilizada por servidores FTP, sería la de la línea 3 del bloque (stilliard/pure-ftpd). Se configuran los puertos 21 (se usa para conexiones FTP) y del 30000 al 30009 (para conexiones pasivas). Posteriormente, se monta el directorio local ftp_data en /home/arch dentro del contenedor, y también se monta ftp_config para que el archivo passwd se encuentre en /etc/ftpServer/ dentro del contenedor. Finamente, se definen las variables de entorno, en las que quedan establecidos el usuario, la contraseña y el directorio base para el usuario FTP.

```
# docker-compose up -d
```

Finalmente, el último paso consiste en el despliegue del servidor FTP, que se realiza con el comando de arriba. Una vez completado este paso, el servidor FTP queda listo para ser utilizado.

1.3. Proxy Inverso Nginx

1.4. Bridge

Un bridge[8.] actúa como un dispositivo virtual que conecta varias interfaces de red, permitiendo que operen como si estuvieran en la misma red física. En este caso, se creará un bridge llamado *br0.* El primer paso es la configuración de red del bridge por DHCP y el establecimiento del Gateway del mismo.

```
config_br0="dhcp"

routes_br0=( "default via 192.168.64.1" )
```

Posteriormente se hace la asignación de la interfaz, y se establece la dependencia entre el bridge y la interfaz.

```
bridge_br0="enp0s20f0u2"
rc_net_br0_need="net.enp0s20f0u2"
```

Lo siguiente es establecer el retraso de reenvío en cero, lo cual minimiza la latencia antes de reenviar un paquete a través del bridge.

```
bridge_forward_delay_br0=0
setfd 0
```

Luego se configura el intervalo entre los mensajes Hello del protocolo STP en 10 ms, que es parte de la configuración de STP para la detección de bucles en la red, e inmediatamente se activa este mismo protocolo justamente para prevenir dichos bucles.

```
sethello 10
stp on
```

Luego se crea un enlace simbólico para la interfaz del bridge, permitiendo manejar la interfaz del bridge de forma similar a una interfaz de red normal. Y finalmente y sin mayor dilación, se activa el bridge.

```
# ln -s /etc/init.d/net.lo /etc/init.d/net.br0
# /etc/init.d/net.br0 start
```

Adicionalmente se debe configurar el servicio Dnsmasq[9], que se encarga de proveer al sistema servicios de DHCP y DNS.

```
interface=br0
server=192.168.80.128
# dhcp-range=192.168.80.1,192.168.80.254,72h
# /etc/init.d/dnsmasq start
```

En todo ese bloque de comandos se llevó a cabo la configuración del servicio Dnsmasq. Se comenzó configurando la interfaz de escucha de Dnsmasq, para que escuche en la interfaz de nuestro bridge, lo que permite que se proporcionen servicios DHCP y DNS a esa red. Luego se configura el servidor DNS, y el rango de direcciones IP para el DHCP. Y finalmente, se termina con la activación del servicio Dnsmasq con su despliegue.

También se debe tener en cuenta otro servicio, el servicio libvirtd, que es el responsable de la administración de máquinas virtuales y sus recursos de red, lo que permite la gestión de conexiones de red y almacenamiento para máquinas virtuales. El siguiente paso es habilitar e iniciar este servicio.

```
# rc-service libvirtd start
```

Luego se agregan las máquinas al entorno de administración de virt-manager[5], y se configura cada una para que incluyan el bridge

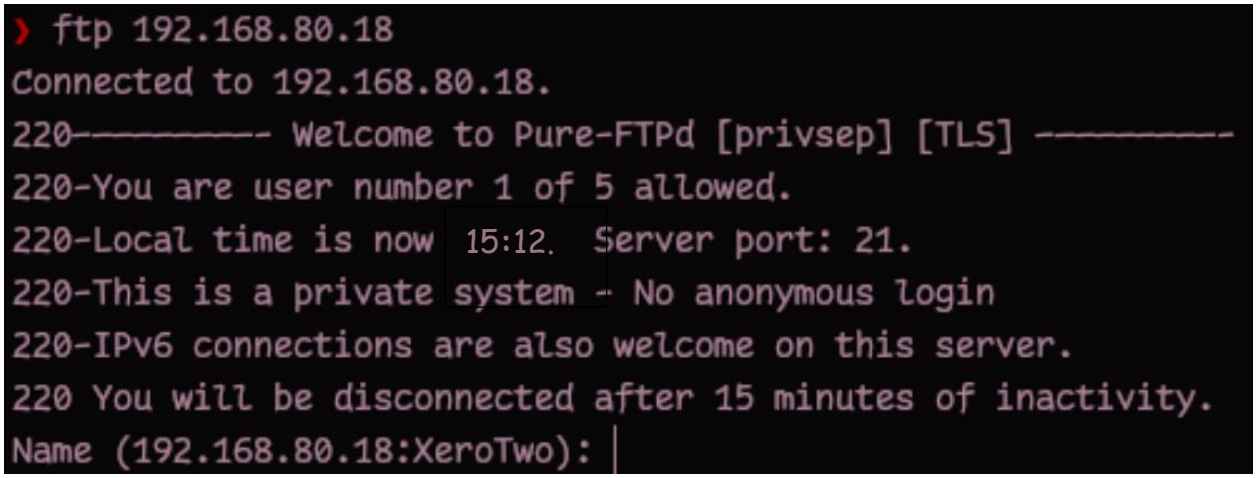
Con esto, finalmente quedaron conectadas las máquinas virtuales al bridge, y por tanto, se puede acceder a los servicios.

En esa imagen se ve resaltada en rojo la dirección IP 192.168.80.18, que es la dirección en la que se encuentra funcionando el proxy inverso. El servidor Apache yace escuchando en el puerto 443 y el FTP en el puerto 21.

Hello World

I'm hosted with GitHub Pages.

Esta imagen muestra el acceso a la página del servidor web de Apache, utilizando la dirección IP del proxy, accediendo al puerto 433. La siguiente mostrará lo propio, pero en el puerto 21, mostrando el servidor FTP.



2. Cluster Computacional

Nuevamente, al igual que en el ejercicio anterior, para la resolución de este se requirieron tres máquinas virtuales. En este caso, se implementó el framework Ray[11,12,13.], que tiene la capacidad de computar en procesamiento paralelo.

El proceso de conexión de las tres máquinas virtuales es prácticamente idéntico al del ejercicio anterior, por lo que no ahondaremos mucho en el tema. Nuevamente, se debe acceder al virt-manager, construir un bridge y configurar el servicio Dnsmasq de la misma manera que en la primera parte de este informe. Cuando ya se hubo tenido acceso a las máquinas virtuales, se pudo proceder[14.].

El framework Ray fue instalado en cada una de las máquinas, asegurándose además de que en las tres máquinas se cumpliera que la versión de Python y de Ray fueran la misma.

```
# pip install ray
```

Para el ejercicio, se definió que una de las máquinas sería la maestra y las demás serían esclavas. Con esto en mente, el siguiente paso era diseñar un código en Python en la máquina maestra con el programa a ejecutar, asegurándose de incluir la librería de Ray. Una vez hecho esto, desde la máquina maestra se ejecuta el siguiente comando:

```
$ ray start --head
```

Esto nos permite obtener la IP y puerto a asignar en las máquinas esclavas. El ejecutar ese comando nos retorna un output que contiene la siguiente línea:

```
$ ray start --address='192.168.80.24:6379'
```

Esta línea debe ejecutarse desde las máquinas esclavas para que se conecten con la maestra y pueda simularse el paralelismo que ofrece Ray.

Para probarlo entonces, se tomó un código público del usuario *idigitopia* en GitHub[10.] (en las referencias irá el link al repositorio exacto del que se tomó la aplicación).

El código implementa un algoritmo de iteración de valor (Value Iteration) distribuido para resolver problemas de toma de decisiones en entornos donde se tiene un conjunto de estados y acciones. La iteración de valor es un método de programación dinámica utilizado para resolver problemas en los que un agente intenta maximizar su recompensa en el tiempo mediante la toma de decisiones óptimas en cada estado.

```python
import numpy as np
import ray

ray.shutdown()
ray.init(address='auto')


# A : Action Space
# S : State Space

@ray.remote
class VI_worker(object):
    def __init__(self, list_of_actions, tran_dict, reward_dict, beta, backup_states, true_action_prob=0.8,
                 unknown_value=0):
        self.backup_states = backup_states
        self.list_of_actions = list_of_actions
        self.tran_dict = tran_dict
        self.reward_dict = reward_dict
        self.beta = beta
        self.unknown_value = unknown_value  # Default Value for any states that do not have transitions defined.

        self.true_action_prob = true_action_prob
        self.slip_prob = 1 - self.true_action_prob
        self.slip_action_prob = self.slip_prob / len(self.list_of_actions)
```

La clase VI_worker es un trabajador remoto que calcula el valor esperado para un subconjunto de los estados del espacio de estados. En el constructor, se definen los parámetros importantes para el problema, como la lista de acciones posibles, el diccionario de transiciones, el diccionario de recompensas, el factor de descuento, y otros valores que se usarán para el cálculo.

```python
def compute(self, V_t, backup_states=None):
    """

    :param V_t: Value Vector at t
    :return:
    """
    backup_states = backup_states or self.backup_states

    V_tplus1 = {s: 0 for s in backup_states}
    max_vals = {s: float("-inf") for s in backup_states}

    max_error = 0

    for s in backup_states:
        for a in self.tran_dict[s]:
            expected_ns_val = 0
            for ns in self.tran_dict[s][a]:
                try:
                    expected_ns_val += self.tran_dict[s][a][ns] * V_t[ns]
                except:
                    expected_ns_val += self.tran_dict[s][a][ns] * self.unknown_value

            expect_s_val = self.reward_dict[s][a] + self.beta * expected_ns_val
            max_vals[s] = max(max_vals[s], expect_s_val)
            V_tplus1[s] += self.slip_action_prob * expect_s_val
        V_tplus1[s] += (self.true_action_prob - self.slip_action_prob) * max_vals[s]

        max_error = max(max_error, abs(V_tplus1[s] - V_t[s]))

    return V_tplus1, max_error
```

El método compute es el método principal de la clase, que calcula los valores actualizados de los estados para la iteración actual.

Para cada estado en backup_states, el trabajador calcula el valor esperado tomando en cuenta las posibles acciones y las transiciones probables a otros estados. Se actualiza el valor del estado considerando la recompensa esperada de cada acción y el valor esperado del siguiente estado. Finalmente, devuelve los valores actualizados y el error máximo para ver si el algoritmo ha convergido (o sea, si los valores actuales están suficientemente cerca de los valores anteriores).

```python
def distributed_value_iteration(S, A, reward_dict, tran_dict, seed_value=None, unknown_value=0, true_action_prob=0.8,
                                beta=0.99, epsilon=0.01, workers_num=4, verbose=True):
    # Split the state space evenly to be distributed to VI workers
    state_chunks = [a.tolist() for a in np.array_split(np.array(S), workers_num)]
    V_t = {s: 0 for s in S} if seed_value is None else seed_value

    # Make VI workers
    workers_list = [VI_worker.remote(list_of_actions=A,
                                     tran_dict=tran_dict,
                                     reward_dict=reward_dict,
                                     beta=beta,
                                     backup_states=state_chunk,
                                     unknown_value=unknown_value,
                                     true_action_prob=true_action_prob)
                    for state_chunk in state_chunks]

    # Do VI computation
    error = float('inf')
    while error > epsilon:
        object_list = [workers_list[i].compute.remote(V_t) for i in range(workers_num)]
        error_list = []
        for i in range(workers_num):
            finish_id = ray.wait(object_list, num_returns=1, timeout=None)[0][0]
            object_list.remove(finish_id)
            V_tplus1, error = ray.get(finish_id)

            V_t.update(V_tplus1)
            error_list.append(error)

            if (verbose):
                print("Error:", error)

        error = max(error_list)

    pi = get_pi_from_value(V_t, A, tran_dict, reward_dict, beta)
```

La función distributed_value_iterations muestra el proceso de iteración de valor de manera distribuida. El conjunto de estados S se divide en partes, asignando cada una a un VI_worker. Entonces, se crean los trabajadores de iteración de valor cada uno de los cuales procesará una parte de los estados. Se ejecuta un bucle hasta que el error sea menor que el umbral epsilon. En cada iteración, los trabajadores calculan los valores actualizados de los estados y el error asociado a cada uno. Los resultados se recopilan, y los valores se actualizan en V_t (el valor de los estados en la iteración actual). Si el error máximo de los trabajadores es menor que epsilon, el bucle termina. Finalmente, se utiliza la función get_pi_from_value para obtener la política óptima pi basada en los valores V_t.

```python
def simple_value_iteration(S, A, reward_dict, tran_dict, seed_value=None, unknown_value=0, true_action_prob=0.8,
                           beta=0.99, epsilon=0.01, workers_num=4, verbose=True):
    slip_prob = 1 - true_action_prob
    slip_action_prob = slip_prob / len(A)

    V_t = {s: 0 for s in S} if seed_value is None else seed_value
    error = float("inf")

    while error > epsilon:
        V_tplus1 = {s: 0 for s in S}
        max_vals = {s: float("-inf") for s in S}
        max_error = 0

        for s in S:
            for a in tran_dict[s]:
                expected_ns_val = 0
                for ns in tran_dict[s][a]:
                    try:
                        expected_ns_val += tran_dict[s][a][ns] * V_t[ns]
                    except:
                        expected_ns_val += tran_dict[s][a][ns] * unknown_value
                expect_s_val = reward_dict[s][a] + beta * expected_ns_val
                max_vals[s] = max(max_vals[s], expect_s_val)
                V_tplus1[s] += slip_action_prob * expect_s_val
            V_tplus1[s] += (true_action_prob - slip_action_prob) * max_vals[s]
            max_error = max(max_error, abs(V_tplus1[s] - V_t[s]))

        V_t.update(V_tplus1)
        error = max_error

        if (verbose):
            print("Error:", error)

    pi = get_pi_from_value(V_t, A, tran_dict, reward_dict, beta)
    return V_t, pi
```

La función simple_value_iteration realiza el mismo cálculo de iteración de valor pero en una sola máquina, sin distribuir la carga entre varios trabajadores. Esto sirve para comparar el rendimiento de la versión distribuida (con Ray) y la versión no distribuida.

```python
def get_pi_from_value(V, list_of_actions, tran_dict, reward_dict, beta):
    v_max = {s: float('-inf') for s in V}
    pi = {}

    for s in V:
        for a in tran_dict[s]:
            expected_val = 0
            for ns in tran_dict[s][a]:
                try:
                    expected_val += tran_dict[s][a][ns] * V[ns]
                except:
                    expected_val += tran_dict[s][a][ns] * 0
            expect_s_val = reward_dict[s][a] + beta * expected_val
            if expect_s_val > v_max[s]:
                v_max[s] = expect_s_val
                pi[s] = a

    return pi
```

La función get_pi_from_value calcula la **política óptima** (pi) a partir de los valores finales V_t. Para cada estado s, selecciona la acción que maximiza la recompensa esperada y la asigna a la política pi para ese estado. Así, pi representa el conjunto de acciones óptimas que el agente debe seguir para maximizar sus recompensas.

3. Dificultades en ambos ejercicios

3.1. Máquinas Virtuales

Al principio nos costó bastante a todos los miembros del equipo entender bien cómo debían hacerse todas las configuraciones entre las máquinas, pero luego de buscar en más lugares se pudieron solucionar los inconvenientes y el resto del ejercicio fluyó sin mayor complicación.

Adicionalmente, tuvimos problemas para encontrar páginas web estáticas con la cual hacer la parte del servidor web de Apache, por lo que nos resolvimos a crear una nosotros mismos. Para no complicar más el trabajo de lo que ya lo era, decidimos dejar la plantilla de las páginas de GitHub básicamente sin decoración, pues lo importante no era el contenido de la página sino que se pudiera realizar el ejercicio de las máquinas virtuales.

3.2. Cluster computacional

Como tal en el desarrollo del ejercicio no se presentaron mayores problemas. Quizás lo más complicado fue encontrar un código que sirviera para hacer las pruebas, ya que no pudimos tampoco ingeniarnos un código propio. Sin embargo, el código que terminamos encontrando fue especialmente útil ya que en sí mismo contenía una función que permitía comparar su rendimiento utilizando el cluster versus utilizando únicamente un solo computador.

4. Referencias

1. Servidor web Apache: https://wiki.archlinux.org/title/Apache_HTTP_Server
2. Docker: https://wiki.archlinux.org/title/Docker
3. Docker-Compose:https://github.com/stilliard/docker-pure-ftpd/blob/master/docker-compose.yml
4. Arch Linux: https://github.com/archlinux/arch-boxes
5. Virt-Manager: https://wiki.archlinux.org/title/Virt-manager
6. Nginx: https://nginx.org/en/docs/beginners_guide.html
7. Nginx: https://nginx.org/en/docs/stream/ngx_stream_core_module.html
8. Bridges:https://devtechs.readthedocs.io/en/latest/topics/virtualization/network-virtualization/linux-bridges.html
9. Dnsmasq: https://wiki.archlinux.org/title/Dnsmasq

10. Fuente del código utilizado para el cluster: https://github.com/idigitopia/Distributed-VI.git
11. Ray Framework: https://www.btelligent.com/en/blog/how-to-install-ray-under-windows-1/#:~:text=Issue%20the%20command%20"pipenv%20--,install%20ray"%20to%20install%20ray.
12. Ray Framework: https://docs.ray.io/en/latest/ray-overview/installation.html
13. Ray Framework: https://docs.ray.io/en/latest/ray-overview/index.html
14. Clusters: https://docs.ray.io/en/latest/cluster/getting-started.html