UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

TRABAJO DE GRADO

Evaluación de uso de un cluster oportunista basados en sistemas operativos heterogéneos y HTCondor.

Autor:
Carlos BUELVAS MONTES

Director:
Msc. Jairo SERRANO

Trabajo de grado entregado como requisito para el titulo de Ingeniero de Sistemas

Como parte del grupo de investigación

GRITAS
Facultad de Ingeniería

14 de junio de 2016

Declaración de autoría

Yo, Carlos BUELVAS MONTES, declaro que este trabajo de grado, «Evaluación de uso de un cluster oportunista basados en sistemas operativos heterogéneos y HTCondor.» y el trabajo realizado son de mi autoría. Confirmo que:

- Este trabajo fue realizado parcial o total mente como estudiante de pregado de esta universidad.
- Donde se consulta el trabajo publicado de otros, se hace la respectiva atribución.
- Reconozco las principales fuentes de ayuda para este trabajo.
- Este trabajo fue realizado bajo la supervisión del Msc Jairo Serrano, quien me guío y orientó en el proceso.

Firma:

Fecha:

«Las grandes ideas, de grandes hombres en el mundo, que pueden llevar al progreso y la evolución del mismo, por lo general son impedidas por una sola cosa...

»El resto de los hombres.»

Thomas Hamillton (Black Sails)

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

Resumen

Facultad de Ingeniería

Programa de Ingeniería de Sistemas

Ingeniero de Sistemas

Declaración de autoría

by Carlos BUELVAS MONTES

In the technological university of Bolivar, there is a high demand on machines to handle a huge amount of data, in order to process that workload, It uses a grid system, a cluster to provide an infrastructure support to handle that work. To achieve this goal implements the use of HTCondor, a software to control the jobs assigned to this cluster, allowing to perform different tasks, follow the job process, define architecture, even a operative system for the job execution.
We test the performance of HTCondor over different operative system, looking for deploy the optimal solution to each tasks, we are building a grid system that can solve any type of task, using the resources we have. To make the most of this resources, we use an opportunistic cluster with heterogeneous operative system, this allow us use a machine if is idle or if the use of that machine is lower that a specific limit. using this we can test the use and performance of tool HTCondor on the different machine and architectures available...

Agradecimientos

Doy gracias a Dios, a mi padre por tener paciencia conmigo, a mis profesores que me enseñaron tanto en todo este tiempo, y a la Universidad Tecnológica de Bolívar por brindarme tantas oportunidades.

Gracias al profesor Jairo Serrano e Isaac Zuñiga que fueron un gran apoyo todo este tiempo.

Índice general

Índice de figuras

Dedicado a mi padre, Dona:

Primero las gracias a Dios, por que es por él que eres mi padre, pero a ti te agradezco por darme la vida, por estar allí para mi en los momentos difíciles, por tus sabios regaños que me ayudaron a encaminarme en lo correcto, por tus charlas de animo, por las risas, por los llantos, es por ti que hoy soy quien soy, te agradezco desde lo mas profundo de mi corazón por siempre tenderme una mano…

Gracias.

Descripción del proyecto

1.1. Descripción del problema

Tomando en cuenta las soluciones HPC que provee la Universidad tecnológica de Bolívar a sus estudiantes y profesores que están encaminados a la investigación, haciendo uso de la infraestructura actual, que cuenta con 4 servidores de alto procesamiento, **Spider**, **Spider01**, **Spider02**, **Spider03**, pero que en algunos casos se quedan sin poder llevar a cabo solicitudes para nuevos trabajos.

Para estas nuevas solicitudes existe la necesidad de hacer uso de recursos adicionales con los que cuenta la universidad. A nivel de infraestructura se cuenta con laboratorios donde es posible hacer uso de máquinas de propósito general en estados de ocio, las cuales se pueden agregar a un cluster para poder aprovechar dichos recursos.

HTCondor, nace como una herramienta de control y despliegue de cluster, lo que permite un mayor manejo sobre las máquinas que hacen parte de este, y sobre las diferentes posibles tareas que se puedan ejecutar en el mismo. La universidad Tecnológica de Bolívar, al ser líder en la implementación de nuevas tecnologías, busca permitir a profesores como a los estudiantes un recurso (cluster) en el cual puedan llevar a cabo tareas de computación de alto desempeño, mostrando así, la capacidad, la infraestructura, y el liderazgo que posee en el campo de las nuevas tecnologías.

1.2. Objetivos

1.2.1. Objetivo General

- Desplegar un cluster oportunista anexo al laboratorio de computación de alto desempeño de la Universidad Tecnológica de Bolívar, HPClab para el envío de trabajos de cómputo a diferentes entornos de ejecución, haciendo uso de GPU/CPU disponibles en máquinas en estado de espera en los laboratorios del campus.

1.2.2. Objetivos específicos

- Crear una guía para la implementación de un cluster heterogéneo mediante el uso de HTCondor en sistemas operativos Linux, Windows y OSX. (*HTCondor*)
- Documentar los casos donde se puede hacer uso del cluster oportunista, y colocar esta información en la Wiki del laboratorio de alto desempeño para que este disponible al alcance de todos.

Marco teórico

Este trabajo se enfoca principalmente en el uso de recursos computacionales que se encuentran en un estado ocioso en el campus universitario. Para hacer uso de esos recursos se dispone de un software que controla el envío de trabajos y la ejecución de los mismos en estos equipos, HTCondor, permite el manejo de los recursos, maneja un cluster de equipos de computo, de manera oportunista, haciendo uso de estos en los momentos en que el equipo no esta siendo utilizado.

2.1. Cluster de computadores

Un cluster, es un sistema de procesamiento paralelo o distribuido, que consta de un conjunto de computadoras independientes, interconectadas entre si, de tal manera que funcionan como un solo recurso computacional.

A cada uno de los elementos del cluster se le conoce como **nodo**, estos cuentan con uno o más procesadores, memoria RAM, Interfaces de red, dispositivos E/S y un sistema operativo, todo interconectados entre si mediante una red de área local (LAN).

Comúnmente entre todas las maquinas del cluster existe una que es llamada el **nodo-maestro**, que es la encargada de administrar, controlar y monitorear todas las aplicaciones y recursos del sistema, mientras que el resto de nodos están dedicados al procesamiento de datos o a ejecutar operaciones aritméticas, a estos últimos se les conoce como **worers** o **nodos-esclavos**.

Clasificación de los clusters

Los clusters de computadoras se clasifican de acuerdo a sus características.

- **Disponibilidad**: En esta categorías se puede contar con clusters *dedicados* y *no-dedicados*, los primeros se destinan a ejecutar un solo trabajo (código, programa o aplicación), en estos los procesadores trabajan en su totalidad en realizar esta tarea. En los segundos los procesadores pueden ser utilizados al tiempo por diferentes trabajos.
- **Aplicación**: Aquí entran los clusters que ejecutan aplicaciones utilizadas en el cómputo científico, donde lo más importante es obtener un alto desempeño, optimizando el tiempo de procesamiento, es decir, evitando en lo posible demasiado tiempo de CPU en procesos de respaldo y lectura de datos. También en este grupo se encuentran

los clusters de alta disponibilidad, donde lo fundamental es que los nodos-esclavos siempre se encuentren funcionando de manera óptima.

- **Configuración**: Pueden ser homogéneos o heterogéneos, en el caso de los homogéneos todos los nodos cuentan con la misma arquitectura y el mismo sistema operativo, y en los heterogéneos los nodos poseen arquitecturas diferentes y sistemas operativos diferentes.

2.2. Metodología

Esta actividad tiene como fin obtener la mayor cantidad de información posible referente a manejo e implementación de HTCondor en los diferentes sistemas operativos que utilizaremos, también la documentación de los despliegues e instalaciones que se lleven a cabo, con el fin de analizar los pro y los contras del funcionamiento de HTCondor sobre cada OS utilizado.

Por ello, esta enfocado principalmente a el uso de un cluster oportunista basado en sistemas operativos heterogéneos y diferentes arquitecturas. Para ello se dispondrá de *nodos-esclavos* en equipos con sistema operativo en Windows y un *nodo maestro* con sistema operativo Ubuntu(Linux).

Estado del arte

3.1. HTCondor

HTCondor es un sistema de gestión de carga de trabajo especializado para pruebas de ejecución de cálculo intensivo. Al igual que otros sistemas que hacen uso del batch con todas las funciones, HTCondor ofrece un mecanismo para trabajos en cola, una política de planificación, un esquema de prioridades, el seguimiento de los recursos y la gestión de recursos. Los usuarios envían trabajos ya sea en serie o paralelos a HTCondor, HTCondor los coloca en una cola, decide cuándo y dónde ejecutar los trabajos en base a una política, monitorea cuidadosamente su progreso, y en última instancia, informa al usuario sobre la terminación.

Una de las tantas característica de HTCondor es que puede ser usado para administrar clusters conformados por nodos de computo dedicado. Aun así, también tiene la capacidad de configurarse para utilizar solamente nodos de computo no dedicados, de tal forma que emplea mecanismos para aprovechar los tiempos ociosos de la CPU siempre y cuando estos no presenten eventos en el mouse o teclado, por ejemplo, presionar alguna tecla.

En el caso que un nodo no dedicado (previamente se encuentre disponible u ocioso) esté ejecutando alguna tarea enviada por HTCondor y algún usuario de dicho nodo realice algún evento en el teclado o mouse, HTCondor está en la capacidad de detectar este evento y parar la tarea en dicho nodo y reenviarla a otro nodo que sí se encuentre en estado ocioso o disponible (checkpoint). Otra característica muy importante y que lo convierte en una herramienta con una gran ventaja en comparación con otros gestores de trabajos, es que tiene un diseño limpio que simplifica el envío de trabajo de los usuarios por medio del mecanismo ClassAds (siglas en inglés de Classified Advertisements), el cual permite a los usuarios pedir recursos necesarios y deseados para ejecutar los trabajos.

Para hacer uso de esta herramienta, se debe contar con tres elementos claves, que son un nodo maestro (**Master**), uno o varios nodos donde ejecutar los trabajos (**Workers**) y un **universo de ejecución**.

- **Nodo maestro (Master)**: es el encargado de administrar y coordinar las tareas que van a ser ejecutadas en el cluster. Esto quiere decir, que es capaz de determinar cuáles nodos esclavos se encuentran disponibles o cuales son ideales para ejecutar una tarea. Por otra parte, este nodo también permite al usuario el acceso al cluster, por consiguiente, por cada grupo o entorno HTCondor(pool) debe existir únicamente

un solo nodo master. Es de importancia resaltar, que este nodo también puede hacer el rol de worker y submit. A este nodo también se le conoce como Central Manager.

- **Nodo trabajador (Worker)**:este tipo de nodo es el que está en capacidad de recibir y ejecutar las tareas enviadas por el nodo submit.
- **universo de ejecución**: en HTCondor un universo es un entorno de ejecución el cual depende del tipo de aplicación que se desea ejecutar. Los universos son: standard, el cual provee confiabilidad y migración de las tareas por medio del mecanismo de checkpoint, para que un programa pueda ser enviado por medio de este universo, ha debido ser compilado con las librerías de HTCondor. vanilla, es el universo por defecto si no se especifica, aunque el administrador puede configurar HTCondor para que otro universo lo sea, es el menos restrictivo con los programas que se puedan enviar, ya que acepta cualquier programa escrito en cualquier lenguaje de programación, su desventaja es que no permite el mecanismo de checkpoint. El universo grid permite a los usuarios enviar trabajos usando la interfaz de HTCondor, estos trabajos son enviados para ser ejecutados sobre recursos que se encuentren en una grid. El universo java permite enviar trabajos escritos en lenguaje, el universo parallel es para enviar programas que requieran múltiples máquinas(nodos esclavos) para ejecutar un solo trabajo o aplicación paralela, para utilizarlo se debe hacer uso del estándar MPI (siglas en ingles de Message Passing Interface).

3.2. Planta Física Universidad Tecnológica de Bolívar

Distribución de la estructura física de los servidores con los que cuenta actualmente la Universidad Tecnológica de Bolívar

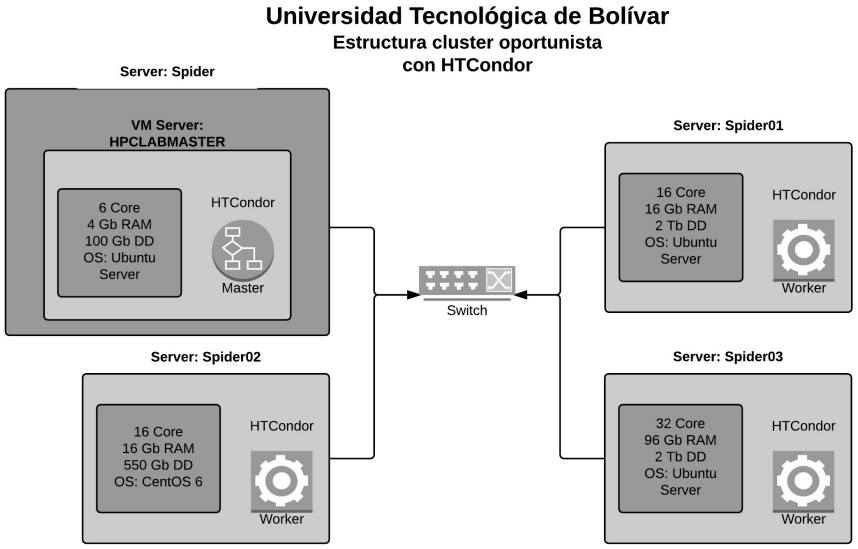


FIGURA 3.1: Estructura HTCondor en la UTB

El nodo maestro es una máquina virtual instalada en el servidor físico **Spider**, esta máquina virtual cuenta con 6 procesadores, 4Gb RAM y 100Gb de espacio en disco para almacenamiento.

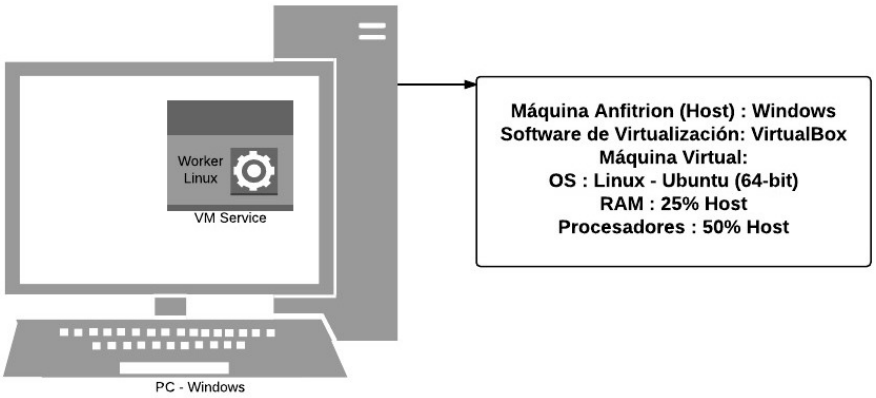


FIGURA 3.2: Estructura HTCondor en Nodo Windows

Ademas de ello cuenta con 6 salones laboratorios que tienen 30 estaciones de trabajo con sistema operativo Windows que son el foco donde se implementará este trabajo.

Implementación: Despliegue

4.1. Creación máquina base

Para poder desplegar las máquinas como servicio en los nodos de trabajo con Windows, primero se crea una máquina virtual para usarla como base.

El gestor de maquinas virtuales utilizado es **VirtualBox v 5.0.18** que se puede encontrar en la página principal o el enlace en la wiki de la universidad (*HPCLab*).

Sistema Operativo Base

Como sistema operativo base para el nodo de trabajo, se instala **Ubuntu 14.04** en su versión de escritorio mínima, el disco de instalación no pesa mas de 50 megas y permite instalar solo los componentes necesarios para la ejecución de y trabajos en segundo plano con la herramienta HTCondor.

Una vez instalado el gestor de máquinas virtuales, se ejecuta y se crea una nueva máquina virtual.

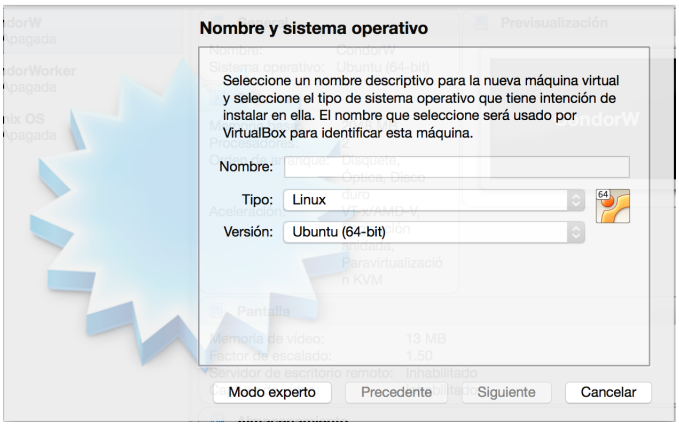


FIGURA 4.1: Creación máquina virtual

Luego de seleccionar un nombre para la máquina y escoger el tipo de sistema operativo y la versión, se selecciona la memoria RAM para la máquina virtual.

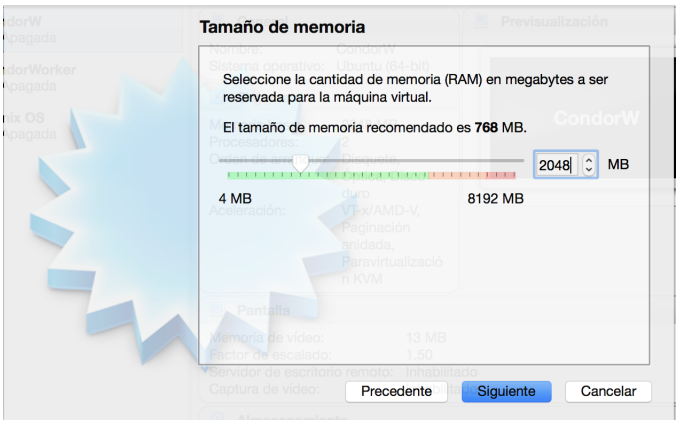


FIGURA 4.2: Memoria máquina virtual

Se crea un disco virtual que contendrá toda la información de nuestro nodo de trabajo.

FIGURA 4.3: Creación disco virtual

El tipo de disco que se crea, en este caso es un Virtual Hard Disck (VHD) con capacidad de 10 GB.

Este espacio sera asignado de manera dinámica, así el tamaño total de la imagen del nodo de trabajo sera mas pequeña y fácil de compartir por la red de la universidad.

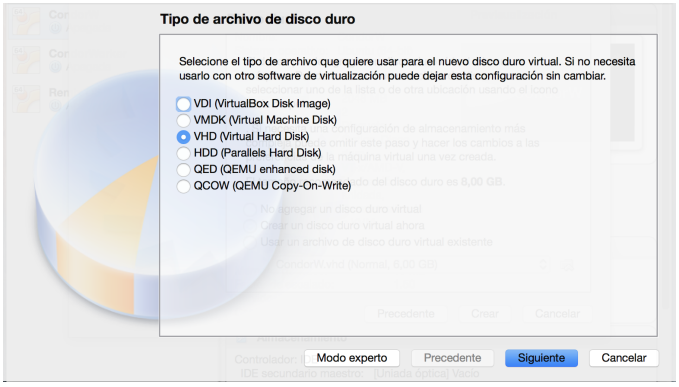


FIGURA 4.4: Tipo de disco virtual

El nombre del disco virtual, es por defecto igual al nombre de la máquina virtual.

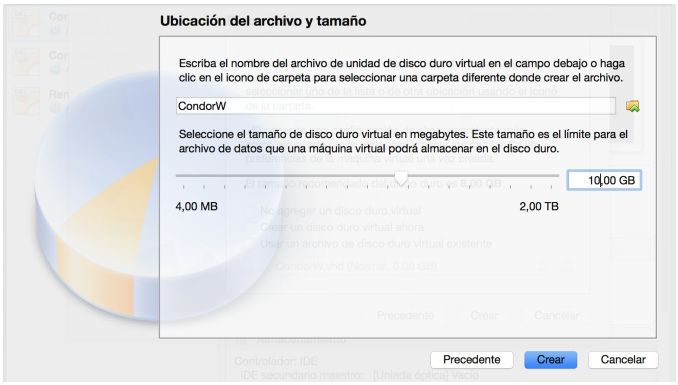


FIGURA 4.5: Tamaño disco virtual

Se le da un tamaño de 10Gb al disco virtual, por lo que se espera trabajar con grandes volúmenes de datos, pero como este disco esta asignado de manera dinámica, el tamaño final es menor al 10Gb, próximo a 3Gb.

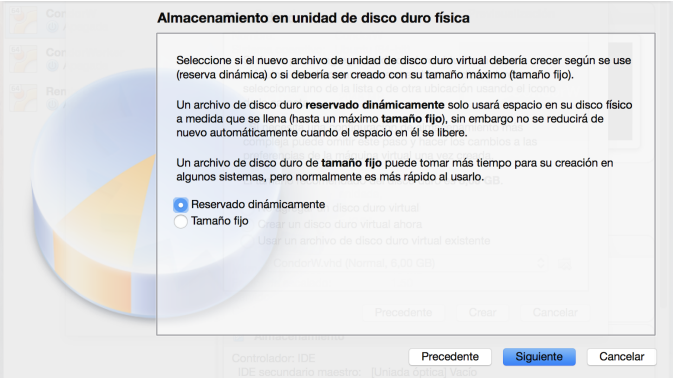


FIGURA 4.6: Tamaño asignado dinámico

Una vez creado el disco, se procede a montar la imagen del sistema operativo en una unidad virtual para la instalación en nuestra maquina.

Se realiza la instalación de Ubuntu 14.04 Desktop Minimal.

Una vez instalado el sistema operativo, accedemos a la máquina y procedemos a actualizar e instalar los paquetes necesarios para el correcto funcionamiento de HTCondor, El universo de Java, y los scripts necesario para automatizar el nombre de la máquina en el cluster.

4.2. Instalación de HTCondor

El proceso de instalación de HTCondor es realmente sencillo, se hace uso de los repositorios propios de Ubuntu para esto, una vez actualizado el sistema.

4.2.1. Paso a paso

Primero acedemos como usuario **root** e instalamos HTCondor usando el comando:

apt-get install htcondor

Nos mostrará una lista de dependencias que se deben instalar para el correcto funcionamiento de la herramienta.

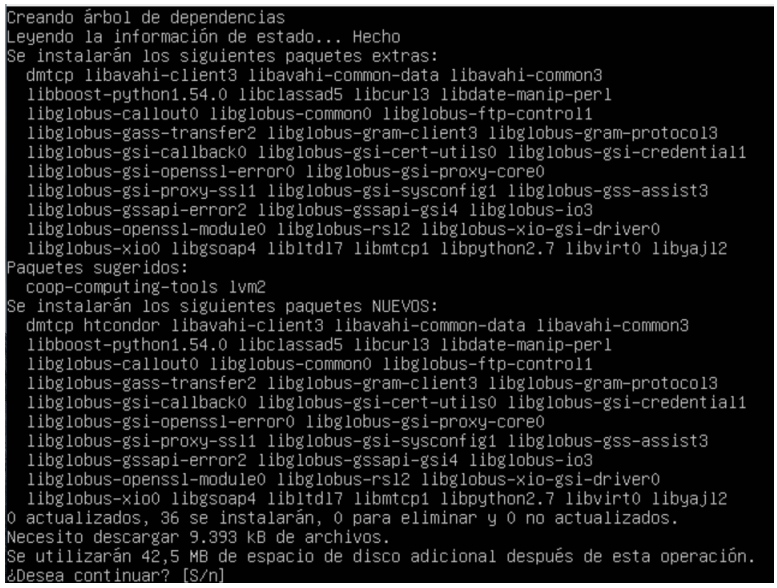


FIGURA 4.7: Dependencias necesarias para HTCondor

Luego nos pedirá si deseamos modificar los parámetros de configuración inicial de condor, le decimos que no, ya que el archivo de configuración lo editaremos mas adelante.

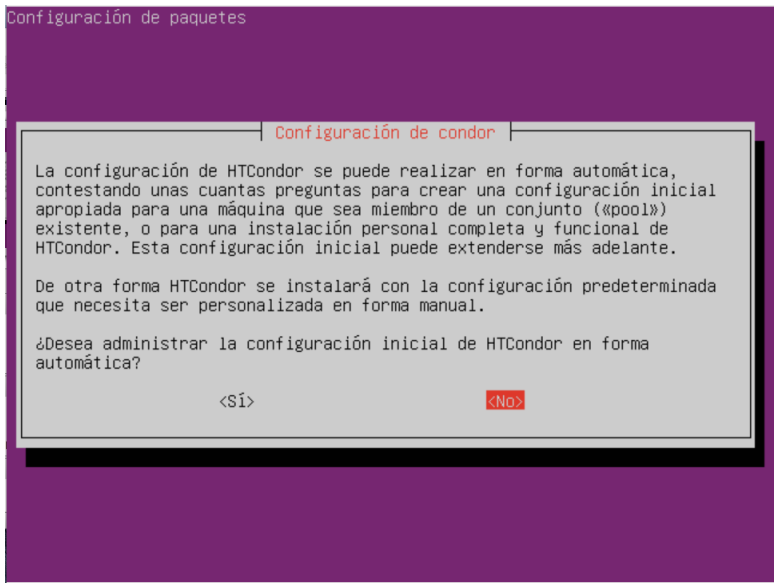


FIGURA 4.8: Configuración inicial

Verificamos que HTcondor esta correctamente instalado y ejecutándose usando el comando:

ps aux | grep condor

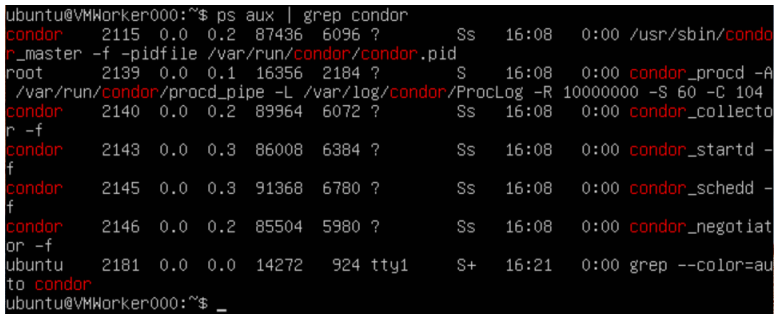


FIGURA 4.9: HTCondor activo en el sistema

4.3. Instalación de Java

Luego de la instalación de HTCondor, se procede con la instalación de Java en el sistema, para ello se usa un PPA con la información reciente de Java versión 8.

para ello primero instalamos la herramienta de manejo de PPA

apt-get install software-properties-common

y luego agregamos el PPA correspondiente a Java.

sudo add-apt-repository ppa:webupd8team/java

FIGURA 4.10: PPA Java

FIGURA 4.11: PPA Java instalado

Luego se actualiza la lista de paquetes del sistema usando el comando:

sudo apt-get update

Y se procede a instalar Java 8.

sudo apt-get install oracle-java8-installer

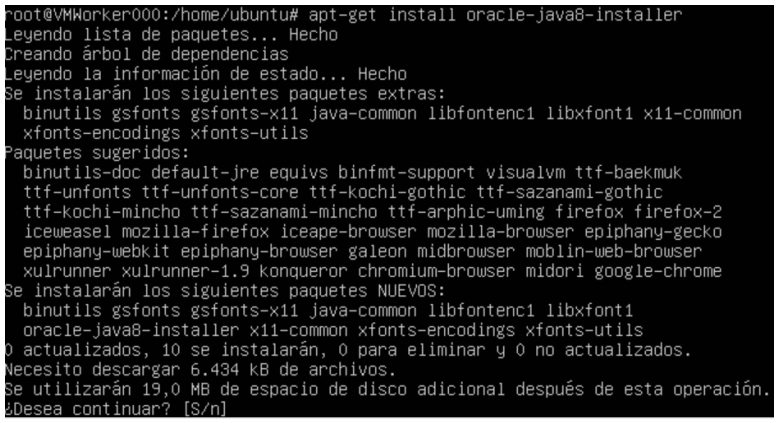


FIGURA 4.12: Comando instalación de Java

Se aceptan luego los términos y condiciones de Java 8.

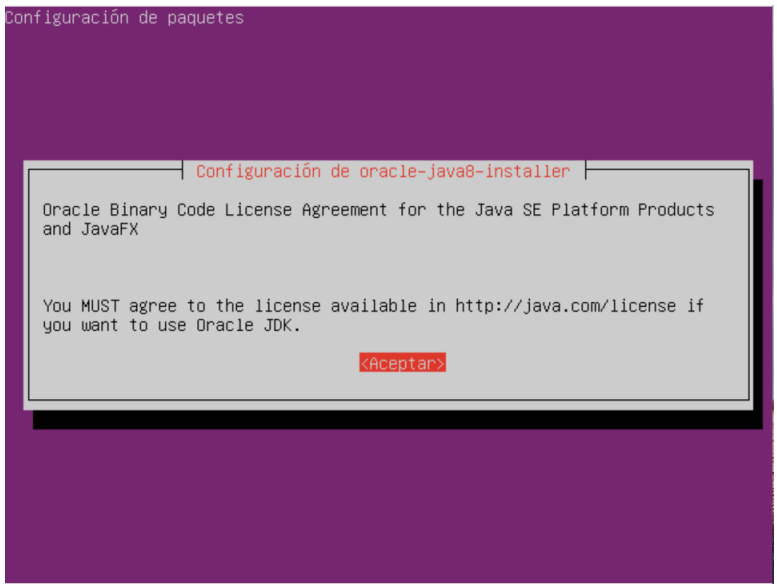


FIGURA 4.13: Aceptación de instalación

Confirmamos que java este instalado correctamente y vemos la versión del mismo con el comando:

java -version

```
root@VMWorker000:/home/ubuntu# java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)
```

FIGURA 4.14: Instalación finalizada

4.4. Instalación de GCC

Una vez instalado Java, procedemos a la instalación de las librerías y compiladores de C y C++ (**gcc**).

apt-get install gcc

```
root@VMWorker000:/home/ubuntu# apt-get install gcc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  cpp cpp-4.8 gcc-4.8 libasan0 libatomic1 libc-dev-bin libc6-dev libcloog-isl4
  libgcc-4.8-dev libgmp10 libgomp1 libisl10 libitm1 libmpc3 libmpfr4
  libquadmath0 libtsan0 linux-libc-dev manpages-dev
Paquetes sugeridos:
  cpp-doc gcc-4.8-locales gcc-multilib make autoconf automake1.9 libtool flex
  bison gdb gcc-doc gcc-4.8-multilib gcc-4.8-doc libgcc1-dbg libgomp1-dbg
  libitm1-dbg libatomic1-dbg libasan0-dbg libtsan0-dbg libquadmath0-dbg
  glibc-doc
Se instalarán los siguientes paquetes NUEVOS:
  cpp cpp-4.8 gcc gcc-4.8 libasan0 libatomic1 libc-dev-bin libc6-dev
  libcloog-isl4 libgcc-4.8-dev libgmp10 libgomp1 libisl10 libitm1 libmpc3
  libmpfr4 libquadmath0 libtsan0 linux-libc-dev manpages-dev
0 actualizados, 20 se instalarán, 0 para eliminar y 0 no actualizados.
Necesito descargar 17,2 MB de archivos.
Se utilizarán 58,9 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] _
```

FIGURA 4.15: Instalación de gcc

Verificamos la instalación de gcc con el comando:

which gcc

```
root@VMWorker000:/home/ubuntu# which gcc
/usr/bin/gcc
```

FIGURA 4.16: gcc instalado

En la carpeta del usuario, hay un ejemplo un programa en c que corre el famoso Hola mundo! (Hola, UTB!) que muestra el funcionamiento de las librerías de c y c++

FIGURA 4.17: Código de prueba gcc

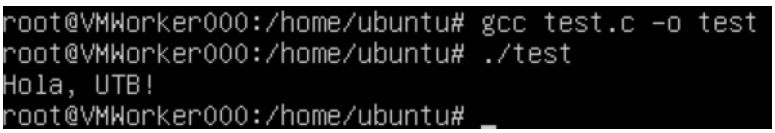


FIGURA 4.18: gcc prueba de ejecución

Uva vez ya configuradas las herramientas se procede a configurar HTCondor para la ejecución de tareas.

Implementación: Configuración

5.1. Configuración de HTCondor

Se deben tener en claro dos cosa a la hora de configurar una máquina para el uso de HTCondor, si esta sera un nodo de trabajo, o sera el nodo maestro que se encarga de enviar las tareas que se llevaran a cabo.

Dado que este trabajo esta basado en un cluster oportunista, se centra en la configuración de los nodos que realizan los trabajos ("workers").

5.1.1. Configuración inicial

Una vez terminada la instalación de las dependencias necesarias, se procede con la creación y modificación del archivo que contendrá la configuración con la cual iniciará el nodo de trabajo.

Para tener en cuenta: Los cambios que se realizan a continuación se hacen como usuario **root** que es quien tiene los permisos para modificar estos archivos.

Primero crearemos el archivo que contendrá la configuración en la ruta:

/etc/condor/config.d/

y le colocaremos como nombre **spider.conf**

cd /etc/condor/config.d/

touch spider.conf

Abrimos el archivo **spider.conf** y procedemos a editarlo

```
UID_DOMAIN = unitecnologica.edu.co
##################################################################
# Nombre legible para el Condor pool
COLLECTOR_NAME = "Mi cluster en $(UID_DOMAIN)"
# Permisos de escritura
ALLOW_WRITE = *.$(UID_DOMAIN)
CONDOR_ADMIN = root@$(FULL_HOSTNAME)
##################################################################
# The following should be the full name of the head node
CONDOR_HOST = 172.16.9.100
##################################################################
# Rango de puertos usados por condor
# se deben abrir estos puertos en el firewall de ser necesario
```

```
IN_HIGHPORT = 9999
IN_LOWPORT = 9000
##################################################################
# This is to enforce password authentication
SEC_DAEMON_AUTHENTICATION = required
SEC_DAEMON_AUTHENTICATION_METHODS = password
SEC_CLIENT_AUTHENTICATION_METHODS = password,fs,gsi,kerberos
SEC_PASSWORD_FILE = /etc/condor/password
ALLOW_DAEMON = condor_pool@*
##  Sets how often the condor_negotiator starts a negotiation cycle
##  for negotiator and schedd).
#  Defined in seconds and defaults to 60 (1 minute), default is 300.
NEGOTIATOR_INTERVAL = 20
##  Scheduling parameters for the startd
TRUST_UID_DOMAIN = TRUE
##################################################################
# start as available and do not suspend, preempt or kill
START = True
SUSPEND = False
CONTINUE = False
PREEMPT = False
KILL = False
WANT_SUSPEND = True
WANT_VACATE = True
##################################################################
#Como es worker, solo debe iniciar dos demonios
DAEMON_LIST = MASTER, STARTD
SOFT_UID_DOMAIN = TRUE
```

5.1.2. Seguridad y autenticación

Para el manejo de la seguridad en el envío de trabajos y como se comunican las maquinas se colocan los parámetros, y se agrega la clave del nodo maestro, que es la clave del pool de máquinas.

Para ello, se reinicia el servicio de condor, y luego se ejecuta el siguiente comando

```
condor_store_cred -c add
```

Para ingresar la clave previamente compartida del nodo maestro.

```
root@VMWorker000:/etc/condor/config.d# condor_store_cred -c add
Account: condor_pool@unitecnologica.edu.co

Enter password:

Operation succeeded.
root@VMWorker000:/etc/condor/config.d# _
```

FIGURA 5.1: Contraseña compartida aceptada

Vemos que la maquina ya lista sus recursos en el pool de condor, la captura fue tomada desde uno de los servidores físicos del plantel, que ya hace parte del cluster.

el comando para listar los recursos disponibles en el pool es:

```
condor_status
```

```
hpctest@spider01:~$ condor_status
Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@VMWorker000  LINUX      X86_64 Unclaimed Benchmar  1.000 1000  0+00:00:04
slot2@VMWorker000  LINUX      X86_64 Unclaimed Idle      0.350 1000  0+00:00:05
slot1@hpclabmaster LINUX      X86_64 Claimed   Busy      0.000  658  0+00:00:04
slot2@hpclabmaster LINUX      X86_64 Unclaimed Idle      0.000  658  0+01:10:06
slot3@hpclabmaster LINUX      X86_64 Unclaimed Idle      0.000  658  0+01:10:07
```

FIGURA 5.2: Recursos de nuestra máquina en el pool

5.1.3. Configuración para Java

Para poder usar el universo **Java**, no basta con tener el entorno de ejecución instalado, se debe descargar y configurar varios archivos, para que el nodo maestro ***hpclabmaster*** pueda reconocer las máquinas que disponen de este entorno.

editamos el archivo: sources.list

nano /etc/apt/sources.list

y al final del mismo colocamos la linea siguiente:

```
deb [arch=amd64] http://research.cs.wisc.edu/htcondor/debian/stable/ wheezy contrib
```

FIGURA 5.3: Repositorio fuente

Luego de esto descargamos la llave publica para HTCondor con el siguiente comando.

wget -qO - http://research.cs.wisc.edu/htcondor/debian/HTCondor-Release.gpg.key | sudo apt-key add -

Luego se descarga el archivo necesario para que htcondor reconozca y admita la máquina de trabajo como una que tiene el universo de Java disponible y habilitado. El archivo necesario para esto es : ***scimark2lib.jar***

nos movemos a la ruta: *cd /usr/lib/condor/*

y allí ejecutamos el siguiente comando:

wget http://math.nist.gov/scimark2/scimark2lib.jar

Una vez descargado el archivo, se ejecutan los siguientes comandos, en orden para que condor reconozca los cambios realizados.

apt update

apt dist-upgrade

apt install condor (Este ultimo reinstala la herramienta para tomar los cambios realizados)

Reiniciamos el servicio HTCondor, y para confirmar que el universo java esta habilitado ejecutamos el comando :

```
condor_status -java
```

```
root@VMWorker000:/usr/lib/condor/libexec# condor_status -java
Name               JavaVendor Ver    State     Activity LoadAv Mem   ActvtyTime

slot1@VMWorker000  Oracle Cor 1.8.0_ Unclaimed Idle      0.400 1000  0+00:00:04
slot2@VMWorker000  Oracle Cor 1.8.0_ Unclaimed Idle      0.000 1000  0+00:00:35
```

FIGURA 5.4: Universo Java Disponible

NOTA: Los pasos necesarios para el correcto funcionamiento del universo Java se realizan ya que en las nuevas distribuciones (versión 8.X en adelante) de la herramienta HTCondor no contienen el archivo de java necesario (***scimark2lib.jar***) para que la herramienta reconozca las máquinas habilitadas con este universo.

Ahora la máquina esta lista para ser usada como nodo de trabajo en el cluster, en el apartado de pruebas se llevará a cabo una para mostrar la ejecución de trabajos en Java.

5.2. Script adicional

Como serán máquinas diferentes que tendrán los nodos de ejecución, es necesario diferenciarlas, para ello, se crea un script para bash para que cambie el nombre de la maquina en la primera ejecución de la misma, luego el script se elimina a si mismo evitando futuros cambios de nombre.

Para el nombre se usa el nombre seguido numero aleatorio , quedando de esta manera ejemplo *VMWorker123456*

```
 GNU nano 2.2.6                File: selfrename.sh

#!/bin/bash
#Assign existing hostname to $hostn
hostn=$(cat /etc/hostname)
newhost="VMWorker$RANDOM"
#change hostname in /etc/hosts & /etc/hostname
sudo sed -i "s/127.0.1.1    $hostn/127.0.1.1    $newhost/g" /etc/hosts
sudo sed -i "s/$hostn/$newhost/g" /etc/hostname
#remove script from /
sudo rm /selfrename.sh
sudo reboot
```

FIGURA 5.5: Script para renombrar la máquina

Este script modifica los archivos ***hostname*** y ***hosts*** que están en las rutas respectivas:

/etc/hostname y ***/etc/hosts***

una vez creado el archivo, le damos permisos de ejecución con el comando:

chmod a+x selfrename.sh

luevo editamos el archivo *rc.local* y agregamos el script para que se ejecute al iniciar la máquina.

nano /etc/rc.local

y al final colocamos la linea respectiva al script.

```
  GNU nano 2.2.6                File: /etc/rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

/selfrename.sh
exit 0
```

FIGURA 5.6: Script para renombrar la máquina

Despliegue en Windows

Para realizar el despliegue sobre las maquinas con sistema operativo Windows se necesitan dos herramientas básicas, adicional a el disco que contiene la imagen base de los nodos de trabajo.

Estas herramientas, son el gestos de máquinas virtuales **VirtualBox** y el aplicativo que ejecuta los nodos como servicios **VBoxVMService**.

Estas herramientas se pueden descargar de los enlaces en la Wiki del laboratorio de computación de alto desempeño de la Universidad Tecnológica de Bolívar o ingresando en el navegador la siguiente dirección dentro del campus:

http://172.16.9.80/ubuntu/spider/workers/

6.1. Pasos para el despliegue en Windows

6.1.1. Instalación de VirtualBox

Es recomendado que se use la versión 5.0.18 de VirtualBox ya que es la actual al momento de realizar las pruebas. Pueden encontrar el instalador en el enlace de la Wiki.

Se des-instala cualquier versión anterior de VirtualBox y se realiza una instalación fresca de la versión 5.0.18, como administrador.

Una vez finalizada la instalación se procede a crear la máquina que sera el nodo de trabajo.

6.1.2. Nodo de trabajo

Creación máquina virtual

Primero crearemos una máquina virtual con los parámetros de configuración óptimos de acuerdo a la máquina anfitrión (máquina Windows).

La creación de esta maquina es similar a la creación de la máquina base, ver el capitulo 4 de este mismo documento, el cambio esta al usar el Disco Virtual de la máquina base descargada del enlace de la wiki.

Movemos la imagen del disco virtual a la ruta:

`C:\Users\"usuario"\VirtualBox VMs`

Donde üsuario"sera remplazado por el nombre del usuario de la máquina windows, y descomprimimos allí nuestra imagen.

Luego al seleccionar la opción “usar disco virtual existente” y colocamos la ruta a nuestro disco virtual base y le damos “crea”.

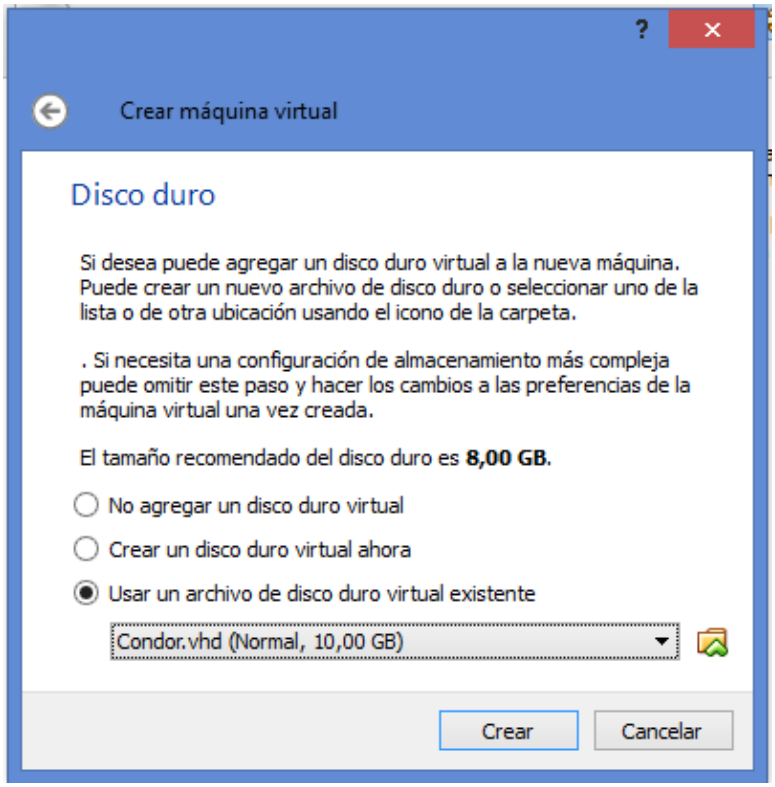


FIGURA 6.1: Disco Virtual Base

Nuestra máquina es creada y procedemos a editar la configuración de esta.

Configuración máquina virtual

Acedemos a la configuración mediante el icono del engranaje en VirtualBox. Accedemos a la pestaña de sistema donde configuraremos la cantidad de memoria RAM a utilizar y el numero de procesadores.

Es recomendado utilizar el 25 % de la cantidad total de RAM y el 50 % de los procesadores físicos (1 o 2 máximo).

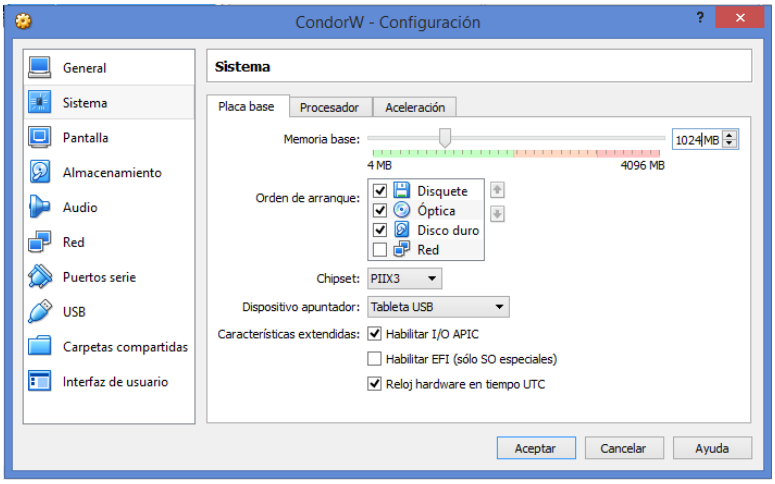


FIGURA 6.2: Opciones de RAM y procesadores

Luego vamos a la pestaña de Red y colocamos el adaptador de red en modo puente (Bridge Mode), damos aceptar y la maquina esta lista para su primera ejecución.

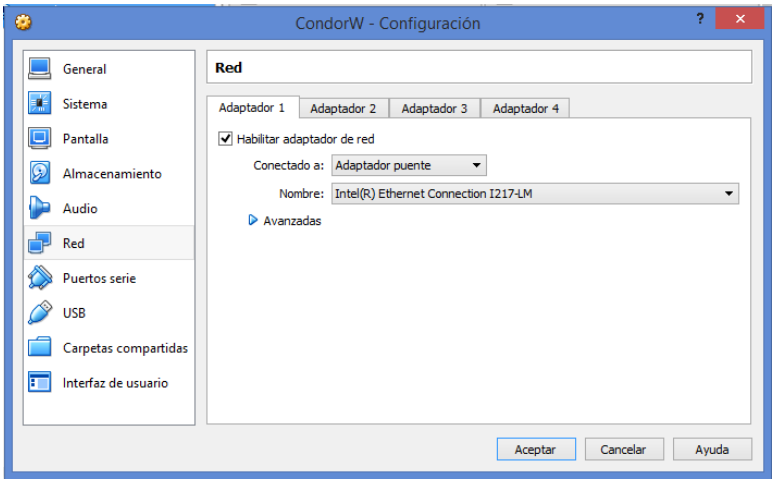


FIGURA 6.3: Red Adaptador Puente

Ejecutamos la máquina y esperamos que termine la primera ejecución, puede ser algo demorado ya que se reinicia luego de cambiar su nombre.

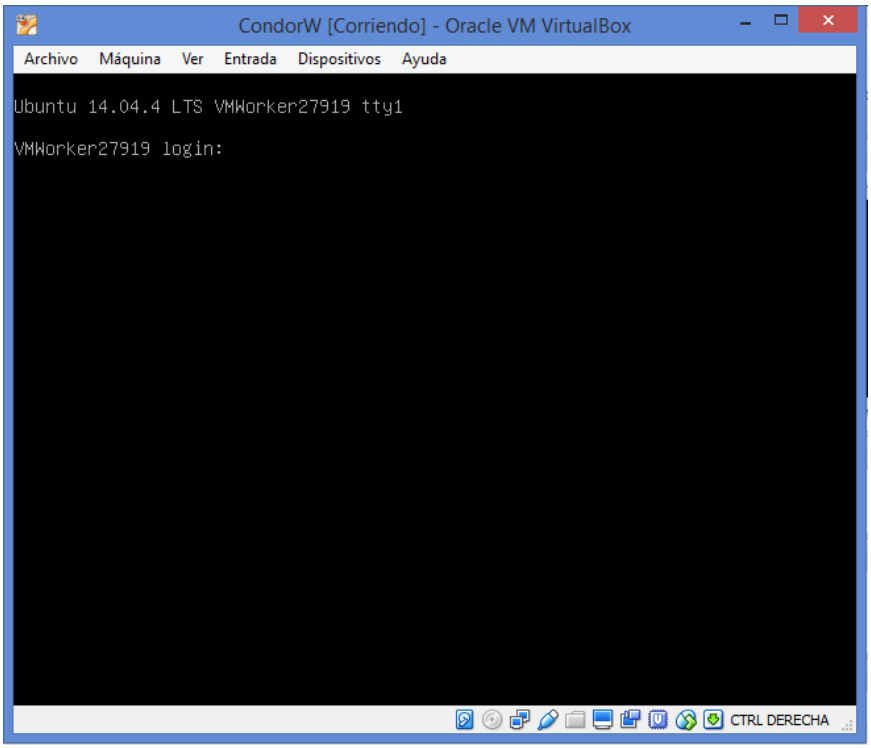


FIGURA 6.4: Máquina virtual en funcionamiento

Ahora se procede a configurar la máquina para ejecución en segundo plano, como servicio.

Máquina Virtual como servicio

Para ejecutar la máquina virtual en background (no se creará ni se ejecutará la consola de administración de VirtualBox, ni la consola de ejecución de la máquina virtual) se instala la herramienta VBoxVMService.

Se cierra por completo VirtualBox y ejecutamos como administrador el instalador de VBoxVMService. Se procede con la instalación sin cambiar la ubicación de instalación, una vez completada la instalación, finalizamos y vamos a la carpeta conde fue instalado.

C:\vms

Allí se modifica el archivo ***VBoxVmService*** que es el archivo de configuración.

```
[Settings]
VBOX_USER_HOME=C:\Users\Condor\.VirtualBox
RunWebService=no
PauseShutdown=5000

[Vm0]
VmName=ubuntu
ShutdownMethod=savestate
AutoStart=yes

[Vm1]
VmName=winxp
ShutdownMethod=savestate
AutoStart=no
```

FIGURA 6.5: Archivo de configuración

Editamos los parámetros para que nuestra máquina virtual ***CondorW*** inicie como servicio, y el método de apagado sea total, no guarde el estado en que se encontraba.

```
[Settings]
VBOX_USER_HOME=C:\Users\Condor\.VirtualBox
RunWebService=no
PauseShutdown=5000

[Vm0]
VmName=CondorW
ShutdownMethod=acpipowerbutton
AutoStart=yes
```

FIGURA 6.6: Configuración final CondorW

Reiniciamos nuestra máquina anfitrión y nos aseguramos que al iniciar nuevamente nuestro nodo de trabajo se esta ejecutando en background.

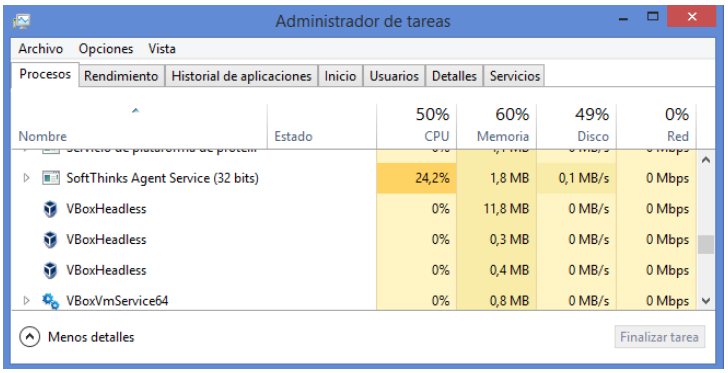


FIGURA 6.7: Procesos en segundo plano en Windows

Se puede apreciar que esta en ejecución VBoxVmService y tres procesos Headless que son los que indican que nuestra maquina esta en ejecución en background.

Evidencias

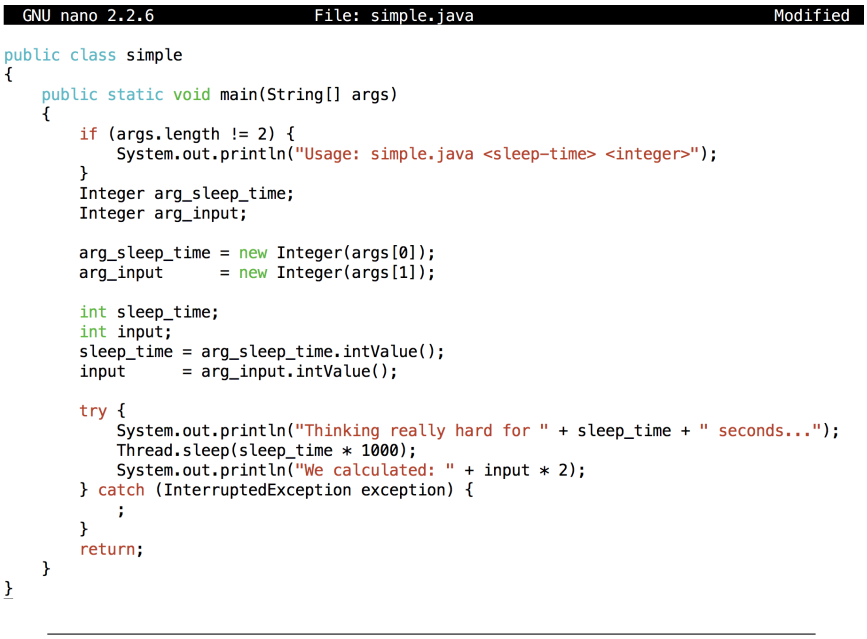
7.1. Pruebas de ejecución en universo Java

Para hacer una prueba sencilla de ejecución en el universo Java, primero debemos crear un programa en java, compilarlo y luego enviarlo a la cola de trabajos de HTCondor.

Para esta prueba usaremos un programa sencillo que hace un calculo y da una respuesta por terminal y luego veremos como se ejecuta este mismo trabajo en HTCondor.

7.1.1. Creación y ejecución de la prueba

primero creamos un archivo con nombre simple.java y lo editamos, y en el colocamos el contenido:



```
 GNU nano 2.2.6                File: simple.java                              Modified

public class simple
{
    public static void main(String[] args)
    {
        if (args.length != 2) {
            System.out.println("Usage: simple.java <sleep-time> <integer>");
        }
        Integer arg_sleep_time;
        Integer arg_input;

        arg_sleep_time = new Integer(args[0]);
        arg_input      = new Integer(args[1]);

        int sleep_time;
        int input;
        sleep_time = arg_sleep_time.intValue();
        input      = arg_input.intValue();

        try {
            System.out.println("Thinking really hard for " + sleep_time + " seconds...");
            Thread.sleep(sleep_time * 1000);
            System.out.println("We calculated: " + input * 2);
        } catch (InterruptedException exception) {
            ;
        }
        return;
    }
}
```

FIGURA 7.1: Código prueba Java

Luego compilamos el programa:

javac simple.java

Y lo ejecutamos:

java simple 4 10

nos arroja el resultado por terminal:

```
iMac-de-LabRedes05:~ cbuelvas$ java simple 4 10
Thinking really hard for 4 seconds...
We calculated: 20
iMac-de-LabRedes05:~ cbuelvas$
```

FIGURA 7.2: Resultado ejecución local

7.1.2. Configuración de archivo ClassAd

ClassAd

HTCondor funciona mediante un mecanismo llamado **ClassAd**, este es un mecanismo flexible para la representación de las características y limitaciones de las máquinas y los trabajos (tareas) en HTCondor. Dentro del sistema, los **ClassAd** pueden representar trabajos, recursos, nodos, máquinas remitentes y otros procesos de HTCondor.

Un ClassAd es un conjunto de expresiones de nombre exclusivo. Cada expresión nombrada se llama un atributo.

Enviamos a la maquina de envíos la prueba de java, y allí creamos el archivo de tarea que llamaremos ***simple.job***.

Nota: La extensión del archivo de tarea (submit file) puede ser cualquiera, luego que la configuración este realizada adecuadamente, utilizamos **.job** para mantener una idea clara de cual es su función.

El archivo de tarea quedara de la siguiente manera.

```
1  Universe   = java
2  Executable = simple.class
3  Arguments  = simple 4 10
4  Log        = simple.log
5  Output     = simple.out.$(PROCESS)
6  Error      = simple.error
7  Rank       = (machine == "VMWorker24657")
8
9  should_transfer_files = YES
10 when_to_transfer_output = ON_EXIT
11
12 Queue
```

Este archivo contiene toda la información necesaria para que HTCondor sepa donde ejecutar este trabajo.

Los parámetros mostrados describen la configuración necesaria para la tarea.

Universe: Define el entorno de ejecución.

Executable: El archivo compilado que realizará la tarea

Arguments: Los parámetros que recibe el archivo ejecutable.

Log : Crea un archivo con la información de la ejecución.

Output: Crea un archivo que contiene la salida de la ejecución.

Error: Crea un archivo de error con información en caso de fallas.

Rank: En esta prueba definimos que máquina del nodo queremos que ejecute la tarea.

should_transfer_files : define si se deben o no enviar archivos en el trabajo.

when_to_transfer_output: especifica cuando transferir los resultados de la ejecución.

Para enviar el trabajo al cluster se usa el comando: ***condor_submit nombrearchivo.job***

```
cbuelvas@openstack:~$ condor_submit simple.job
Submitting job(s).
1 job(s) submitted to cluster 311.
cbuelvas@openstack:~$
```

FIGURA 7.3: Envío de trabajo al cluster

Para verificar que el trabajo este en la cola se usa el comando : ***condor_q***

```
cbuelvas@openstack:~$ condor_q


-- Schedd: openstack : <172.16.9.77:9266?...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 311.0   cbuelvas        5/25 14:26   0+00:00:00 I  0   0.0  java simple 4 10

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
cbuelvas@openstack:~$
```

FIGURA 7.4: Cola de trabajo en el cluster

Verificación en el archivo simple.log del resultado de la ejecución.

```
cbuelvas@openstack:~$ cat simple.log
000 (311.000.000) 05/25 14:26:45 Job submitted from host: <172.16.9.77:9266?addrs=
172.16.9.77-9266>
...
001 (311.000.000) 05/25 14:27:26 Job executing on host: <172.16.5.28:9837?addrs=17
2.16.5.28-9837>
...
006 (311.000.000) 05/25 14:27:30 Image size of job updated: 0
        0  -  MemoryUsage of job (MB)
        0  -  ResidentSetSize of job (KB)
...
005 (311.000.000) 05/25 14:27:30 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
        56  -  Run Bytes Sent By Job
        1082  -  Run Bytes Received By Job
        56  -  Total Bytes Sent By Job
        1082  -  Total Bytes Received By Job
        Partitionable Resources :    Usage  Request Allocated
           Cpus                 :                1         1
           Disk (KB)            :       10       2   2875002
           Memory (MB)          :        0       0       496
...
cbuelvas@openstack:~$
```

FIGURA 7.5: Log de envío

Por ultimo verificamos el archivo de salida, que contiene los resultados de la ejecución.

```
cbuelvas@openstack:~$ cat simple.out
Thinking really hard for 4 seconds...
We calculated: 20
cbuelvas@openstack:~$
```

FIGURA 7.6: Resultado prueba Java

7.2. Prueba de ejecución con Python

Para esta prueba de ejecución, se calculara el tiempo de respuesta que le lleva a una maquina retornar el calculo y suma de los números primos en un rango dado. La prueba se ejecuta primero haciendo uso de multithreads y luego en serial, y arrojara el resultado en un archivos el cual se puede examinar una vez completado el trabajo.

Prueba en local

El script que ejecutara el calculo es el siguiente:

```
1 #!/usr/bin/python
2 import multiprocessing as mp
3 from Queue import Empty
```

```
4  import math
5  import os
6  import time
7  def primeQ(n):
8      """return a boolean, is the input integer a prime?"""
9      if n == 2 :
10         return True
11     max =  int( math.ceil( math.sqrt(n) ) )
12     i = 2
13     while i <= max :
14         if n  % i == 0 :
15             return False
16         i += 1
17     return True
18
19 def sumPrimes(n):
20     """return sum of all primes less than n """
21     return sum( [ x for x in range(2,n) if primeQ(x) ] )
22
23 def worker(q):
24     while True:
25         try:
26             x = q.get(block=False)
27             print sumPrimes(x)
28         except Empty:
29             break
30 if __name__ == "__main__":
31     start = time.time()
32 ncpus = 4
33 my_q  = mp.Queue()
34 for i in range(100000, 2000000, 100000):
35     my_q.put(i)
36 procs=[mp.Process(target=worker, args=(my_q,))  for i in range(ncpus)]
37 for ps in procs:
38     ps.start()
39 for ps in procs:
40     ps.join()
41 mid  = time.time()
42 print "now␣the␣serial␣part␣"
43 for i in range(100000, 2000000, 100000):
44     print sumPrimes(i)
45 ending = time.time()
46 print "multiprocessing␣takes␣", mid - start,      "␣seconds"
47 print "single␣thread␣takes␣",      ending - mid,  "␣seconds"
```

Luego de creado el y editado el archivo se guarda el archivo como **primes** y se le da permisos de ejecución con el comando : **chmod a+x primes**.

Lo ejecutamos de manera local y vemos el resultado en la terminal:

```
iMac-de-LabRedes05:~ cbuelvas$ ./primes
454396537
1709600813
3709507114
6458901531
9914236195
14071826345
18910286312
24465663438
30689332265
37550402023
45125753695
53433406131
62287995772
71881256647
82074443256
92878592188
104450958704
116581137847
129451433482
now the serial part
454396537
1709600813
3709507114
6458901531
9914236195
14071826345
18910286312
24465663438
30689332265
37550402023
45125753695
53433406131
62287995772
71881256647
82074443256
92878592188
104450958704
116581137847
129451433482
multiprocessing takes  41.9075229168  seconds
single thread takes  128.193656206  seconds
iMac-de-LabRedes05:~ cbuelvas$
```

FIGURA 7.7: Ejecución de la prueba local

Luego llevamos este script a una máquina de envíos de trabajo y editamos el archivo ClassAd de envío y el script con la tarea.

Nuestro script es el mismo, se muestra como se configura el script para enviar las tarea:

```
##################################################
# Tarea de condor con python                     #
# por Carlos Buelvas                             #
# Calculo de numeros primos, en multihilo y serial#
##################################################

Universe         = vanilla
Rank             =(machine == "VMWorker17967")||(machine == "VMWorker24657")||(machine == "VMWorker2778")||(machine == "VMWorker27930")
Executable       = primes

notify_user     = cbuelvas@gmail.com
notification    = Complete

log             = primes.log
output          = primes.out
error           = primes.err

should_transfer_files   = YES
when_to_transfer_output = ON_EXIT

Queue
```

FIGURA 7.8: ClassAd para el envío al cluster

Enviamos el trabajo al cluster y verificamos que este en cola de trabajo.

```
cbuelvas@openstack:~/python$ condor_submit c_primes.job
Submitting job(s).
1 job(s) submitted to cluster 317.
cbuelvas@openstack:~/python$ condor_q


-- Schedd: openstack : <172.16.9.77:9266?...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 317.0   cbuelvas        5/26 17:03   0+00:00:00 I  0   0.0  primes

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
cbuelvas@openstack:~/python$
```

FIGURA 7.9: Envío del trabajo y verificación de la cola de trabajos

Luego podemos ver la información de la ejecución en el archivo de ***log***.

```
cbuelvas@openstack:~/python$ cat primes.log
000 (318.000.000) 05/26 17:08:12 Job submitted from host: <172.16.9.77:9266?addrs=172.16.9.77-9266>
...
001 (318.000.000) 05/26 17:08:29 Job executing on host: <172.16.9.100:9433>
...
006 (318.000.000) 05/26 17:08:38 Image size of job updated: 2
        2  -  MemoryUsage of job (MB)
        1672  -  ResidentSetSize of job (KB)
...
006 (318.000.000) 05/26 17:10:52 Image size of job updated: 621508
        2  -  MemoryUsage of job (MB)
        1672  -  ResidentSetSize of job (KB)
...
005 (318.000.000) 05/26 17:10:53 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:03:04, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
                Usr 0 00:03:04, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
        561  -  Run Bytes Sent By Job
        1170  -  Run Bytes Received By Job
        561  -  Total Bytes Sent By Job
        1170  -  Total Bytes Received By Job
        Partitionable Resources :    Usage  Request Allocated
           Cpus                 :                1         1
           Disk (KB)            :       10       2  15075477
           Memory (MB)          :        2       1       658
...
cbuelvas@openstack:~/python$
```

FIGURA 7.10: Log de ejecución de la prueba

Y el resultado de la ejecución lo vemos en el archivo de salida ***.out***.

```
cbuelvas@openstack:~/python$ cat primes.out
454396537
6458901531
18910286312
45125753695
82074443256
104450958704
3709507114
14071826345
30689332265
53433406131
71881256647
116581137847
1709600813
9914236195
24465663438
37550402023
62287995772
92878592188
129451433482
now the serial part
454396537
1709600813
3709507114
6458901531
9914236195
14071826345
18910286312
24465663438
30689332265
37550402023
45125753695
53433406131
62287995772
71881256647
82074443256
92878592188
104450958704
116581137847
129451433482
multiprocessing takes  40.5052947998  seconds
single thread takes  103.260048151  seconds
cbuelvas@openstack:~/python$
```

FIGURA 7.11: Resultado de la ejecución

con esto vemos que los entornos de ejecución para los cuales se diseño esta solución, están activos y son funcionales.

7.3. Trabajos Futuros

Al desarrollar este trabajo, se encontraron diferentes obstáculos a la hora de usar la herramienta HTCondor, entre ellas hay configuraciones de seguridad que se deben tener en cuenta.

- Para lograr correcta comunicación entre los nodos de ejecución y el nodo maestro, todos deben pertenecer al mismo dominio (Ej, dominio.local.edu), para ello se deben configurar a la hora de instalar los equipos este dominio.
- Para aumentar la cantidad de nodos de trabajo, este trabajo se debe replicar en todos los salones de computo del campus.
- El nodo maestro, actualmente por pruebas esta aceptando los trabajos de cualquier dominio, esta es una falla de seguridad.
- Para hacer parte, y hacer uso de clusters de otras universidades, se debe configurar Flokking entre clusters.

Bibliografía

Bolívar, Universidad Tecnológica de. *HPCLab*. http://hpclab.unitecnologica.edu.co/.

Wisconsin–Madison, University of. *HTCondor*. https://research.cs.wisc.edu/htcondor/.