Programación de Sistemas Distribuidos

Curso 2018/2019

Práctica 3

Evaluación del rendimiento en entornos distribuidos utilizando la plataforma de simulación SIMCAN

1.- Introducción

En esta práctica se van a realizar estudios de rendimiento en entornos distribuidos. Para ello, se utilizará la plataforma de simulación SIMCAN, la cual permite tanto el modelado como la simulación de entornos altamente distribuidos y paralelos.

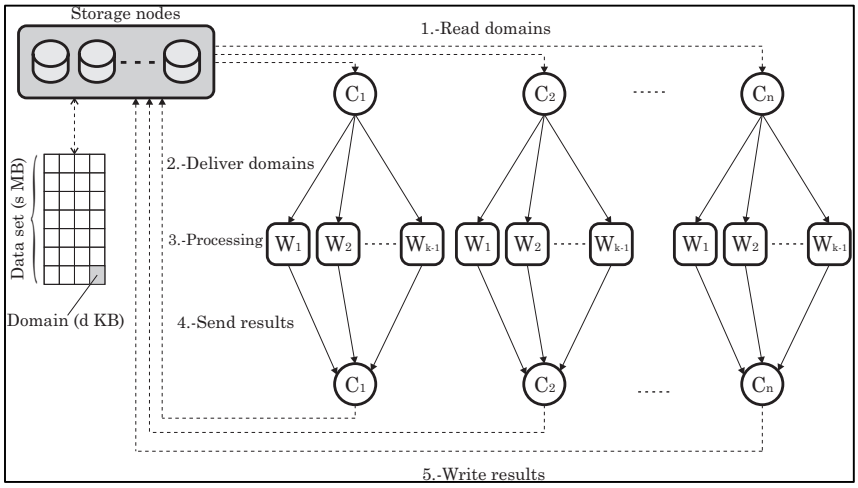
Para realizar esta práctica será necesario utilizar la máquina virtual ubicada en la siguiente dirección http://antares.sip.ucm.es/cana/Lubuntu_PSD_16.04_v1.7z. Esta máquina virtual puede cargarse con el programa Virtual Box, el cual está disponible para los sistemas operativos Windows, Linux y MacOS. En el laboratorio se utilizará el software Virtual Box en la partición de Linux Debian.

El nombre de usuario y la clave para iniciar la máquina virtual es `simulation`

Adicionalmente, Virtual Box se puede descargar de https://www.virtualbox.org

2. Modelado de aplicaciones MPI

En este apartado se va a estudiar el rendimiento de aplicaciones MPI en entornos altamente distribuidos. Para ello, se utilizará el modelo de aplicación de SIMCAN `ApplicationHPC`, la cual modela el comportamiento de una aplicación MPI. El diseño de esta aplicación se describe en la siguiente figura:



El esquema de esta aplicación es similar al estudiado en clase, donde intervenían dos tipos de procesos: un proceso *master* y varios procesos *worker*. Mientras que, generalmente, el proceso *master* se encargaba de asignar los datos a los procesos *worker*, estos últimos realizaban únicamente tareas de cómputo y devolvían los datos procesados al proceso *master*.

Sin embargo, el modelo que utilizaremos para simular aplicaciones MPI es ligeramente diferente. En este caso podemos utilizar varios procesos *master*, denominados *coordinator*, que se encargan de distribuir datos a los procesos *worker*.

A cada proceso *coordinator* le corresponde un número determinado de procesos *worker*. Este conjunto de procesos (1 *coordinator* y n *workers* asociados) se denomina *set*. De esta

forma, se puede aumentar el grado de paralelismo al leer datos de disco o escribir los resultados, aliviando en cierta medida posibles cuellos de botella.

Inicialmente, la aplicación tiene asignado un conjunto de datos, denominado *dataSet*. La aplicación finaliza cuando el *dataSet* se ha procesado completamente. Hasta que esto ocurra, cada proceso *coordinator* obtiene una porción del *dataSet*, denominado *domain*, y la reparte entre los procesos *worker* que tiene asociados. Es importante remarcar que un proceso sólo podrá comunicarse con otros procesos que pertenezcan a su mismo *set*. Una vez procesados los datos, cada proceso *worker* enviará la información al proceso *coordinator* para que, finalmente, los resultados puedan grabarse en disco.

Los parámetros de esta aplicación se describen a continuación:

- `workersSet`: Número de procesos de un set, compuesto por 1 *coordinator* y *workersSet-1* workers.
- `dataSet`: Tamaño del conjunto de datos a procesar (en MiB).
- `sliceToWorkers` (en MB): Tamaño de la porción de datos asignada a un proceso *worker* por cada iteración.
- `sliceToCoordinator` (en MB): Tamaño de datos generada por un proceso *worker* en cada iteración.
- `sliceCPU`: Número de MIs que cada proceso *worker* necesita para procesar una porción de datos.
- `inputFilePrefix`: Prefijo del fichero de entrada.
- `outputFilePrefix`: Prefijo del fichero de salida.
- `workersRead`: Si se establece este parámetro a `true`, los procesos *worker* leen los datos directamente de disco. En caso contrario, reciben los datos de su proceso *coordinator*.
- `workersWrite`: Si se establece este parámetro a `true`, los procesos *worker* escriben los datos directamente a disco, en caso contrario, envían los datos a su proceso *coordinator*.

Este modelo permite 4 variantes diferentes dependiendo de qué procesos realizan las tareas de E/S. Así, la aplicación cuenta con 2 parámetros para definir este comportamiento. La siguiente tabla detalla las 4 posibles configuraciones:

`workersRead`	`workersWrite`	Descripción
true	true	Los procesos *worker* reciben del proceso *coordinator* la cantidad de datos a leer, de forma estos procesos realizan la lectura directamente de disco. Además, al terminar el procesado de datos, escriben en disco los datos generados.
true	false	Los procesos *worker* reciben del proceso *coordinator* la cantidad de datos a leer, de forma que los procesos *worker* realizan la lectura directamente de disco. Al terminar el procesado de datos, envían los datos generados a su proceso *coordinator*.
false	true	Los procesos *worker* reciben de su proceso *coordinator* los datos a procesar. Una vez procesados, los escriben en disco.
false	false	Los procesos *worker* reciben de su proceso *coordinator* los datos a procesar. Una vez procesados, los envían a su proceso *coordinator*.

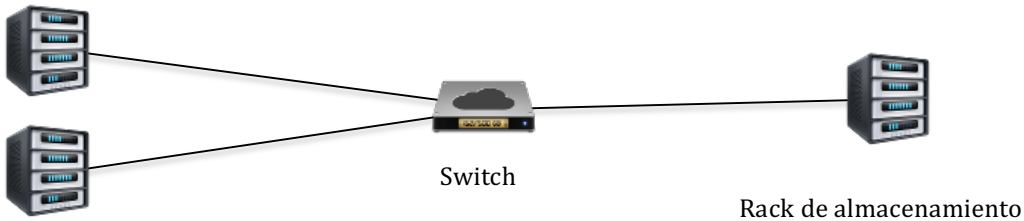
El objetivo en este apartado será estudiar aplicaciones de alto rendimiento ejecutadas en entornos distribuidos con distintas configuraciones. Para ello se modelará una aplicación de tipo `ApplicationHPC`.

Los parámetros de la aplicación son los siguientes:

- `workersSet`: 16
- `dataSet`: 1024
- `sliceToWorkers`: 1
- `sliceToCoordinator`: 1
- `sliceCPU`: 1000000
- `inputFilePrefix`: "/inputFile".
- `outputFilePrefix`: "/outputFile".
- `workersRead`: false.
- `workersWrite`: false.

Para generar correctamente los ficheros de configuración, se debe marcar el *checkBox* "Generate network Configuration".

La aplicación modelada se ejecutará en un data-center con varias configuraciones. El data-center utilizado estará formado por 2 racks de cómputo y 1 rack de almacenamiento. Los 3 racks estarán conectados al mismo switch, tal y como muestra la siguiente figura.



El rack de almacenamiento utilizará un número variable de *blades*, siendo éstos 2, 4, 8 y 16. Todos los *blades* estarán en un único *board*, de forma que cada uno estará configurado con un disco `Disk500_D`, una CPU con 2 cores y 4GB de memoria RAM. Los ficheros de entrada deben tener un tamaño de 4000000 KB.

Los racks de cómputo estarán compuestos por 4 *boards* de 8 *blades* (nodos) cada uno. El modelo de nodo utilizado en estos racks utilizará 4GB de RAM y un disco `Disk500`. Sin embargo, el modelo de CPU, así como el número de aplicaciones será variable. En los experimentos se utilizarán 4 tipos de CPU, con 1, 2, 4 y 8 cores. Estos modelos de CPU deberán modelarse previamente a la configuración del nodo. Para realizar el modelado de las CPUs con varios cores, pueden obtenerse los datos de la siguiente dirección https://en.wikipedia.org/wiki/Instructions_per_second

Es importante remarcar que será **necesario modificar los parámetros de las CPUs utilizadas antes de realizar las simulaciones**. Para ello, se deberán modificar los parámetros de las velocidades de procesamiento un valor arbitrario no superior a ±5% con respecto a los valores preestablecidos. El objetivo de esta modificación es evitar que varios grupos de prácticas obtengan, exactamente, los mismos resultados.

El número de aplicaciones ejecutadas en un nodo será siempre igual al número de cores que tenga la CPU. De esta forma, al utilizar una CPU *singleCore*, se ejecutará una aplicación. Sin embargo, si utilizamos una CPU *quadCore* (4-cores), se ejecutarán 4 aplicaciones en el nodo. El número y tipo de aplicaciones ejecutadas en el nodo se configura en el panel *Applications* del panel *Nodes*.

En total se realizarán 16 experimentos, ya que contamos con 2 parámetros variables, cada uno con 4 valores distintos: 4(tipo de CPU) x 4(tipo rack almacenamiento)=16. Con el fin de agilizar, en medida de lo posible, la configuración de los entornos, se aconseja seguir los pasos siguientes:

- Modelar 4 CPUs nuevas.
- Modelar la aplicación `parallelApp` con los parámetros correspondientes.
- Modelar un nodo de cómputo utilizando:
    - Nueva CPU de 1-core
    - Ejecución de 1 aplicación.
- Modelar un rack de cómputo:
    - 4 *boards* de 8 *blades* cada uno
- Modelar el rack de almacenamiento:
    - Inicialmente con 2 *blades*.
- Modelar el data-center.
- Modelar un escenario con la configuración descrita en los pasos anteriores.
- Generar ficheros de configuración.

Para guardar el escenario en el repositorio, se puede emplear un nombre similar al siguiente: `scenario_2servers_1cores`. En este punto disponemos de un escenario con 2 servidores de almacenamiento y nodos de cómputo con 1 core. El objetivo es generar el resto de los escenarios utilizando esta configuración. Para ello, se realizarán los siguientes pasos, modificando el número de *blades* del rack de almacenamiento.

- Modificar el número de *blades* del rack de almacenamiento:
    - En esta configuración, serán 4 *blades*.
- Grabar el modelo de rack.
- Cargar el escenario anterior (`scenario_2servers_1cores`).
- Grabar el escenario
    - Utilizar otro nombre, como `scenario_4servers_1cores`.

El nuevo escenario contiene un rack con 4 *blades*. La idea consiste en modificar el rack, luego cargar el escenario anterior, que ya tendrá el rack nuevo, y así poder grabar la nueva configuración. Estos pasos se deben repetir para 8 y 16 *blades* en el rack servidor.

De forma similar, se procede a la generación de escenarios modificando las CPUs de los nodos en los racks de cómputo.

- Modificar el nodo de los racks de cómputo
    - Utilizar CPU con 2 cores.
    - Modificar la tabla *Applications* para que se ejecuten 2 aplicaciones.
- Grabar el nodo.
- Cargar un escenario con 2 servidores de almacenamiento.
    - Por ejemplo, `scenario_2servers_1cores`
- Grabar el nuevo escenario
    - Utilizar otro nombre, como `scenario_2servers_2cores`
    - Ya contiene la configuración de 2 CPUs en los racks de cómputo.

Este proceso se debe repetir para los 4 tipos de CPUs. Al finalizar, se debe tener 16 directorios, cada uno con una configuración correspondiente a la indicada en el enunciado.

Para ejecutar los experimentos, se puede hacer uso de un script que ejecute las simulaciones de forma secuencial. El siguiente fichero muestra el contenido de una porción del script para lanzar las 16 simulaciones.

```
cd scenarioAp3_2servers_1cores
./run -u Cmdenv > 2servers_1cores_1G.txt
cd ..

cd scenarioAp3_2servers_2cores
./run -u Cmdenv > 2servers_2cores_1G.txt
cd ..

cd scenarioAp3_2servers_4cores
./run -u Cmdenv > 2servers_4cores_1G.txt
cd ..

cd scenarioAp3_2servers_8cores
./run -u Cmdenv > 2servers_8cores_1G.txt
cd ..

cd scenarioAp3_4servers_1cores
./run -u Cmdenv > 2servers_8cores_1G.txt
cd ..

. . .
```

Este script deberá estar ubicado en el directorio `$SIMCAN_HOME/simulations`. Como puede apreciarse, los resultados de cada simulación se guardarán en un fichero distinto. De esta forma, una vez finalicen las simulaciones, resultará sencillo obtener los datos de cada simulación.

Una vez ejecutados los experimentos, se deberá generar tanto la tabla con los resultados como la gráfica correspondiente. Para generar la gráfica, se puede hacer uso del script `performance3d.gnu`. El contenido de este script se muestra a continuación:

```
1.  ### Set enhanced fonts
2.  set termopt enhanced

3.  ### output for PDF
4.  set terminal pdf
5.  set output 'performance.pdf'

6.  ### Set View
7.  #set view 120, 30, 1, 1

8.  ### Key
9.  unset key

10. ### Tile
11. set title "Performance of CPU/Data intensive app" font 'Times-Roman,8'

12. ### Ticks
13. set ticslevel 0
14. set xtics rotate font 'Times-Roman,5'
15. set xtics offset 0,-0.5,0 font 'Times-Roman,5'
16. set ytics offset 4,0,0 font 'Times-Roman,5'
17. set xrange [0:*]
18. set yrange [0:*]
19. set zrange [0:*]
20. set hidden
```

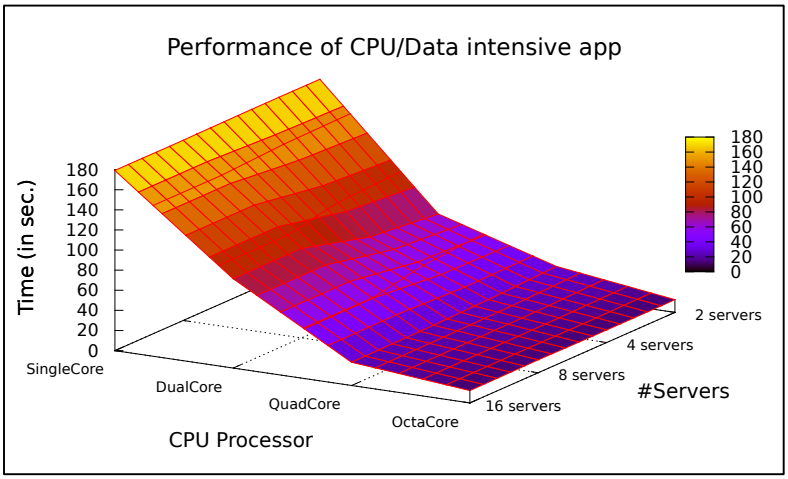
```
21. ### Labels
22. set xlabel "CPU Processor" offset 0, -2, 0 font 'Times-Roman,7'
23. set ylabel "#Servers" offset 2,-1,0 font 'Times-Roman,7'
24. set zlabel "Time (in sec.)" rotate left font 'Times-Roman,7'
25. unset clabel

26. ### Set 3D
27. set grid
28. set hidden3d
29. set contour surface
30. set dgrid3d 4 4 qnorm 2
31. set pm3d at s hidden3d 100 interpolate 5,5
32. set style line 10 linecolor rgb "black"
33. unset hidden3d
34. unset surf

35. ### Plot
36. sp "performance.txt" using 1:2:3:xtic(4):ytic(5) with pm3d
```

La línea 5 indica el nombre del fichero donde se guardará la gráfica. La línea 11 contiene el título de la gráfica. La línea 36 contiene el nombre del fichero con los datos de entrada.

El fichero de datos deberá rellenarse con los datos obtenidos de las simulaciones. La plantilla de este fichero, llamado `performance.txt`, se muestra a continuación. Para rellenarla, deberán modificarse las XX por el valor correspondiente. Cada línea representa una configuración, la cual está indicada por el quinto y sexto campo, tipo de CPU y número de servidores, respectivamente.

```
0       0       XX      "SingleCore"    "16 servers"
0       1       XX      "SingleCore"    "8 servers"
0       2       XX      "SingleCore"    "4 servers"
0       3       XX      "SingleCore"    "2 servers"
1       0       XX      "DualCore"      "16 servers"
1       1       XX      "DualCore"      "8 servers"
1       2       XX      "DualCore"      "4 servers"
1       3       XX      "DualCore"      "2 servers"
2       0       XX      "QuadCore"      "16 servers"
2       1       XX      "QuadCore"      "8 servers"
2       2       XX      "QuadCore"      "4 servers"
2       3       XX      "QuadCore"      "2 servers"
3       0       XX      "OctaCore"      "16 servers"
3       1       XX      "OctaCore"      "8 servers"
3       2       xx      "OctaCore"      "4 servers"
3       3       XX      "OctaCore"      "2 servers"
```

Un ejemplo de gráfica resultante (con datos ficticios) sería el siguiente:



La **parte obligatoria de este apartado** consiste en realizar los experimentos descritos utilizando una red Ethernet 1Gbps, tanto en los racks, como en las conexiones de los racks con el switch.

Para realizar la **parte opcional de este apartado**, se propone modelar nuevos sistemas para estudiar la escalabilidad de la aplicación. Por ejemplo, se pueden modelar nuevos tipos de CPU, discos o emplear diferentes redes en el sistema. De esta forma, se podría utilizar una red para los nodos de cómputo y una diferente para los racks de almacenamiento.

En este apartado se pide generar una tabla con los resultados obtenidos y su correspondiente gráfica de rendimiento para cada tipo de escenario. Además, se deberá responder de forma justificada a las siguientes preguntas:

¿Cómo afecta incluir más cores en la CPU de los nodos de cómputo? ¿Cómo afecta el número de servidores de almacenamiento?

Discutir la escalabilidad de cada aplicación.

¿Sería recomendable utilizar, para estos dos tipos de aplicaciones, siempre los mejores componentes disponibles? Justificar la respuesta.

En caso de haber realizado la parte opcional, justificar la escalabilidad del sistema con las diferentes configuraciones utilizadas.

3.- Entrega de la práctica

La parte **obligatoria** de la práctica se evaluará sobre el **70% de la nota máxima**. **El 30%** restante se asignará, de **forma proporcional**, a las **partes opcionales** realizadas en cada apartado.

No serán válidas las prácticas que utilicen, exactamente, los mismos valores en los parámetros de las CPUs proporcionadas en el repositorio. No se permitirá la entrega de prácticas fuera del plazo establecido.

Para entregar esta práctica se habilitará un entregador en la página de la asignatura del Campus Virtual. La fecha límite para entregarla será el **día 19 de Diciembre de 2018 a las 23:55**.

La entrega se realizará mediante un **único fichero con extensión .zip**. el cual debe contener los ficheros `memoria.pdf` y `simulations.zip`.

El fichero memoria.pdf deberá contener las tablas, gráficas y las respuestas a las preguntas expuestas en cada apartado de este enunciado. El fichero simulations.zip, contendrá todos los entornos modelados para realizar la práctica. Es importante destacar que el **número máximo** de páginas de el documento entregado, incluyendo la página de la portada, será de **5 páginas**.

Se van a perseguir las copias y plagios de prácticas, aplicando con rigor la normativa vigente.