uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Sistemas Paralelos y Distribuidos

Félix García Carballeira

Alejandro Calderón Mateos

Práctica de simulación

Tema 5

Trabajo: Introducción a la simulación de sistemas paralelos y distribuidos

Objetivo: El objetivo de este trabajo práctico es realizar una primera aproximación a la simulación de sistemas distribuidos con **SimGrid**. El material de apoyo que se entrega está compilado y probado para la **versión 4 de SimGrid** (última estable con soporte para el modelo MSG).

SimGrid es un *toolkit* que permite la simulación de sistemas y aplicaciones distribuidas en entornos heterogéneos. La versión disponible se encuentra disponible en la siguiente página web:

https://framagit.org/simgrid/simgrid/-/releases

Para compilar e instalar SimGrid se pueden utilizar los siguientes pasos:

```
git clone https://framagit.org/simgrid/simgrid.git

cd simgrid

mkdir /directorio_install

cmake -DCMAKE_INSTALL_PREFIX=/ directorio_install

make

make install
```

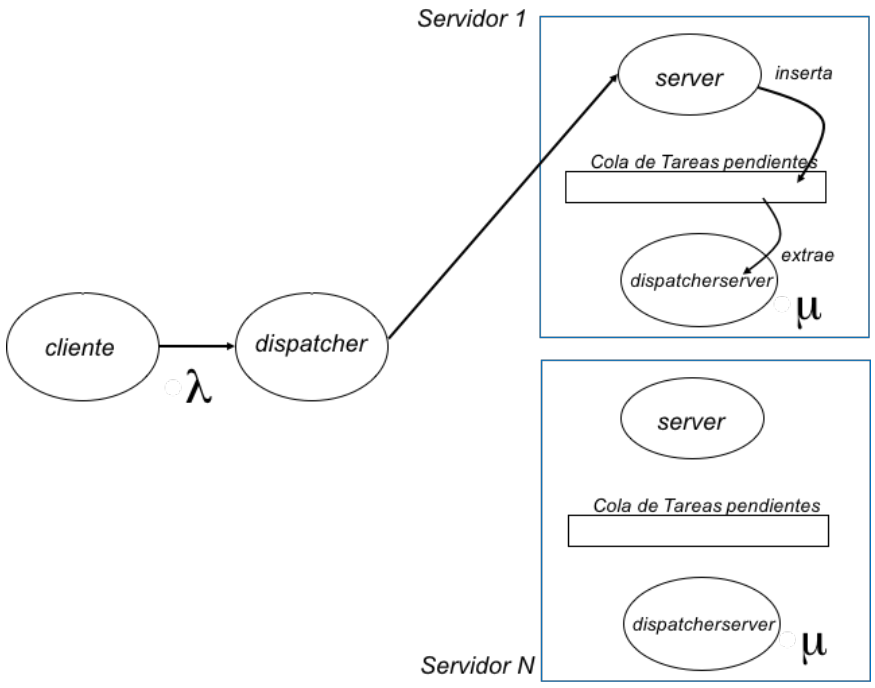
Material de apoyo: Se proporciona como material de apoyo en aula global el archivo **practica_simulacion.zip**. Una vez descomprimido en el directorio `practica_simulacion` se tendrá acceso a los siguientes ficheros:

- Makefile: que permite compilar el programa de practica: practica.c
- practica.c: descrito más abajo.
- Ficheros con la definición de la plataforma: platform-cluster.xml
- rand.c y rand.h: permiten obtener variables aleatorias de diversas funciones de distribución.

El programa `practia.c` simula un entorno como el que se muestra en la siguiente figura. Este modelo simula los siguientes elementos:

- *Cliente*: es el proceso que genera las tareas. Se asume que las tareas llegan de acuerdo a una distribución exponencial y se sirven en cada servidor con un tiempo medio que sigue también una distribución exponencial. Se asume que el tiempo medio de servicio es 1 s, lo que implica la ejecución de 1 GFlop en una máquina con 1 Gflops. La tasa de llegadas de las tareas se proporciona como parámetro al programa. Esta tasa será siempre menor que 1 y su valor λ representa que la tasa de llegadas está al (K*100)% de la capacidad del sistema.

- *Dispatcher*: proceso que recibe las tareas del cliente y aplica el algoritmo de distribución de tareas a los servidores.
- *Server*: proceso que recibe las tareas del *dispatcher*. Este proceso inserta las tareas en una cola para que el proceso *dispathcerServer* las extraiga y las ejecute finalmente.
- *DispatcherServer*: proceso que ejecuta finalmente las tareas. Este proceso junto con el anterior ejecuta en el mismo servidor (mismo host).



La implementación realizada no tiene en cuenta los tamaños de las tareas a enviar por la red y utilizará una simplificación para conocer el servidor con el menor número de tareas (basado en un array global compartido, `Nsystem`).

El programa se ejecuta de la siguiente forma (para un valor de λ igual a 0.9):

```
practica platform_cluster.xml   0.9
```

El objetivo del trabajo es modificar el código de partida para añadir otros algoritmos de distribución de trabajos:

- *Aleatorio*: cada vez que llega una tarea se asigna a un servidor escogido de forma aleatoria.
- *Cíclico*: las tareas se van asignado a los servidores en orden cíclico.
- *Shortest-queue-first* (sqf): la tarea se envía al servidor con menor número de tareas en el sistema (cola y ejecutando).
- *Two-random-choices*: cada vez que llega una tarea se eligen dos servidores aleatorios y se envía el trabajo al servidor con menor número de tareas en él.

- *Two-RR-ramdon-choices*: cada vez que llega una tarea se eligen dos servidores, uno de forma cíclica y el otro de forma aleatoria y la tarea se envía al servidor con menor número de tareas.

Para cada uno de estos modelos se implementarán distintas políticas de planificación en cada servidor:

- FCFS (First-Come, First-Sserver): las tareas a ejecutar en cada servidor se eligen en el orden en el que llegan.
- STF (Shortest Job First): se elige para ejecución la tarea con menor tiempo de ejecución.
- LJF (Longest Job First): se elige para ejecución la tarea con mayor tiempo de ejecución.

La cola de tareas en cada servidor se implementa con una estructura de datos de SimGrid que implementa un array dinámico. En el código se tiene un vector de estructuras:

```
xbt_dynar_t client_requests[NUM_SERVERS] ;
```

Cada componente del vector representa la cola de tareas de cada servidor. Para ordenar los elementos de la cola del servidor K se puede utilizar el siguiente fragmento de código:

```
xbt_dynar_sort(client_requests[K], &sort_function_short);
```

La función `sort_function_short` se proporciona en el archivo `practica.c`

Una vez que se tienen todos los modelos implementados, se realizarán diferentes experimentos de simulación para comparar el rendimiento de los diferentes algoritmos. Para ello:

- Se probará con una infraestructura con 100 servidores y valores de λ de 0,2, 0.5, 0.7 y 0.9.
- Habrá de indicarse el error cometido para un nivel de confianza del 95%.