uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Sistemas Paralelos y Distribuidos

Félix García Carballeira

Alejandro Calderón Matos

Tema 5. Simulación de sistemas distribuidos y paralelos

Objetivo

- Comprender las ventajas que aportan las técnicas de simulación cuando se quiere realizar investigación y experimentar sobre sistemas paralelos y distribuidos de gran escala y las ventajas que aporta el entorno de simulación SimGrid

Contenido

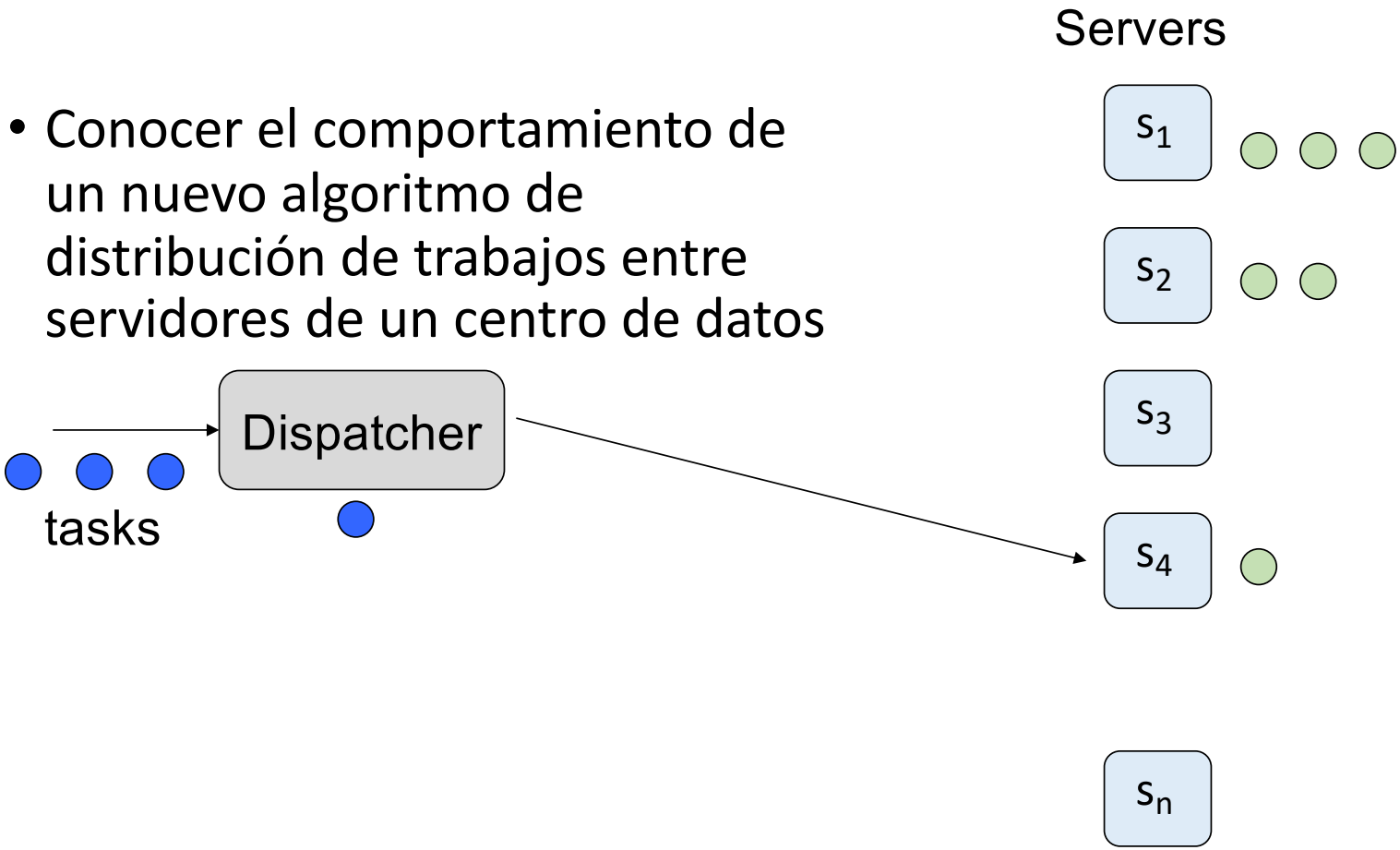
- Conceptos básicos sobre simulación
- Introducción a Simgrid
- Métricas de rendimiento
- Experimentos de simulación
- Estimación de errores
- Comparación de dos alternativas
- Trabajo práctico

¿Por qué es importante la simulación?

- Conocer el comportamiento de un sistema cuando no se dispone de él bajo una determinada carga de trabajo
- Ejemplos:
  - Conocer el comportamiento de un determinado algoritmo de planificación
  - Comparar el funcionamiento de dos algoritmos de distribución de carga
  - Conocer el comportamiento de un sistema de ficheros distribuido

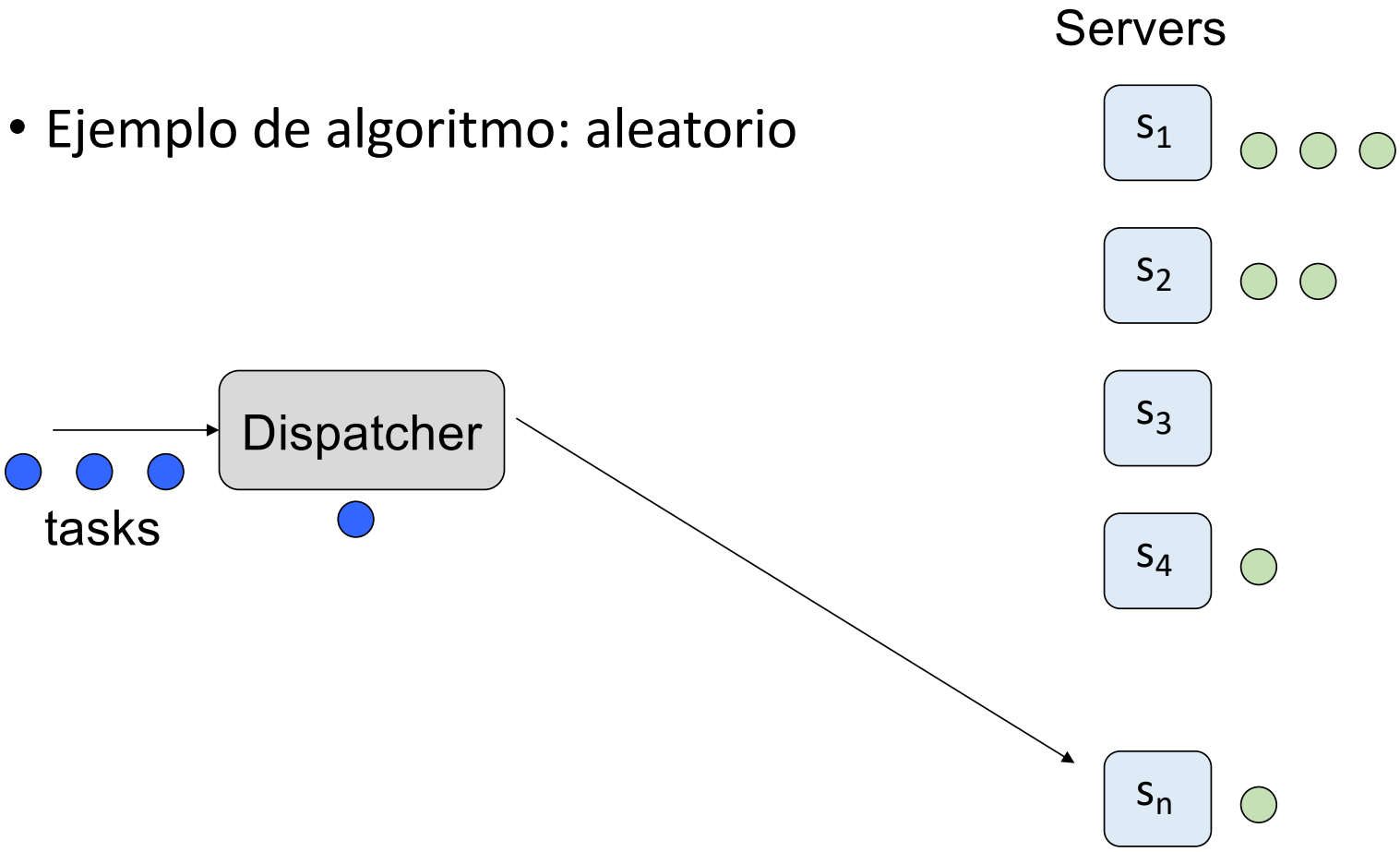
Motivación

- Conocer el comportamiento de un nuevo algoritmo de distribución de trabajos entre servidores de un centro de datos



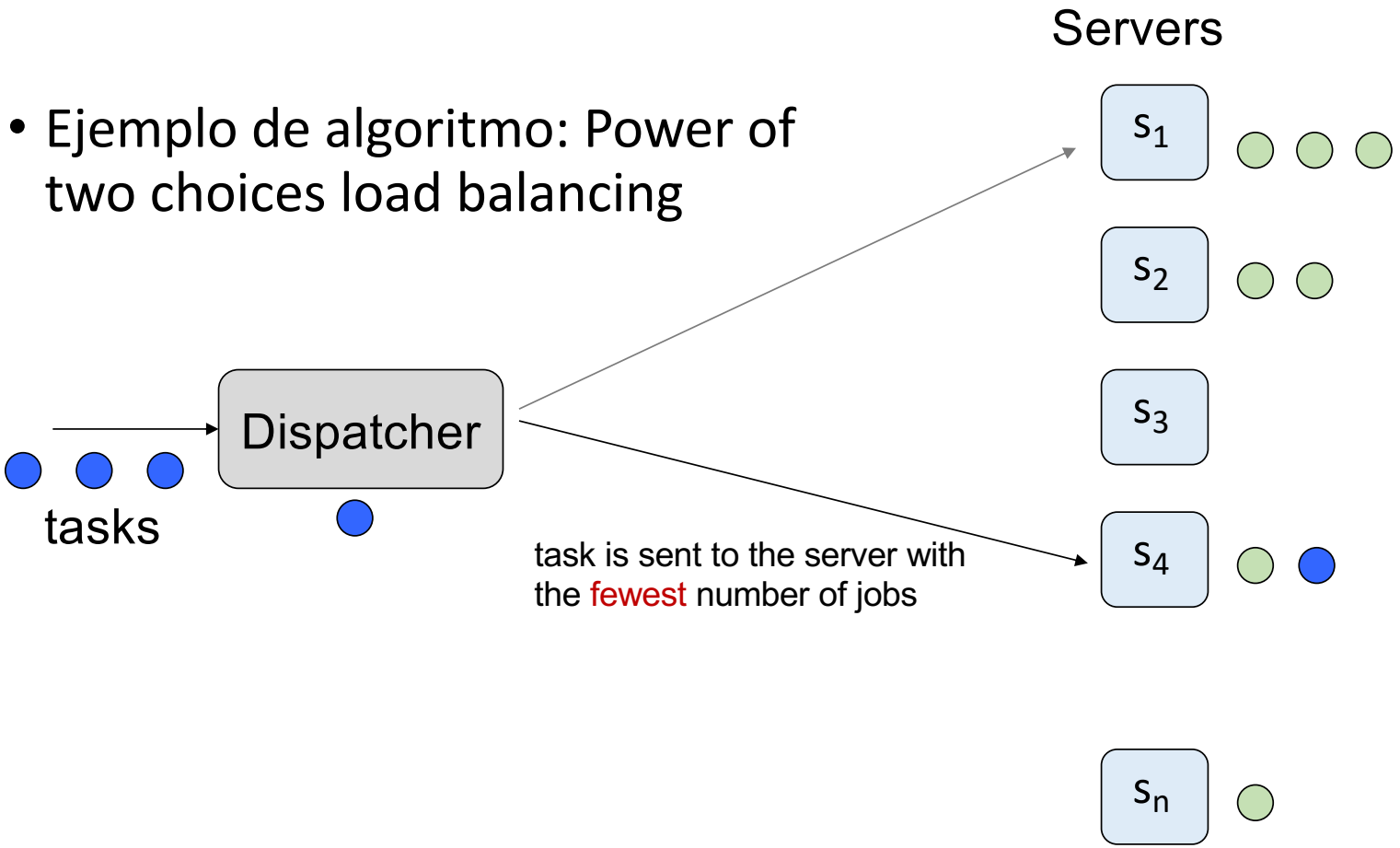
Motivación

- Ejemplo de algoritmo: aleatorio



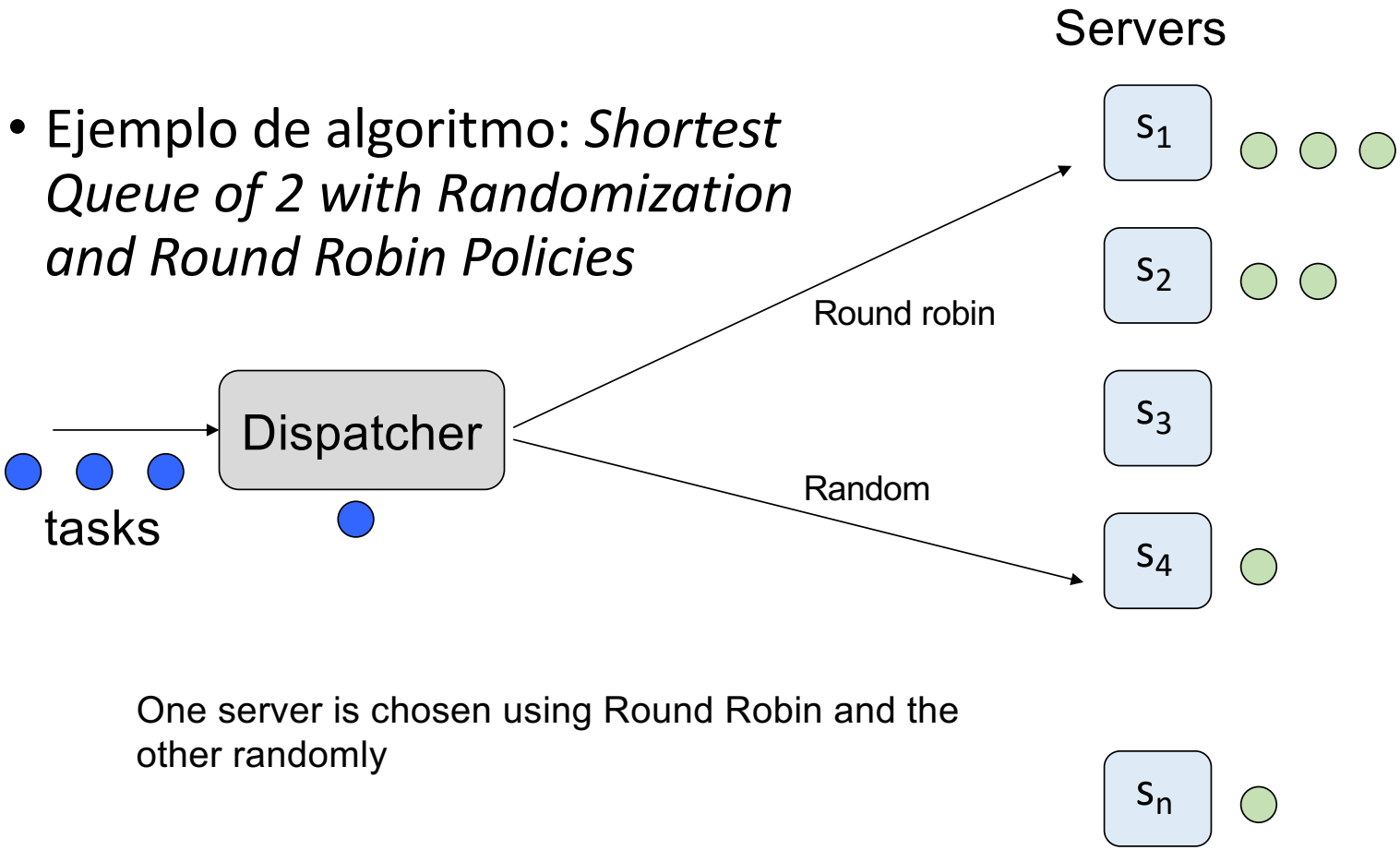
Motivación

- Ejemplo de algoritmo: Power of two choices load balancing



Motivación

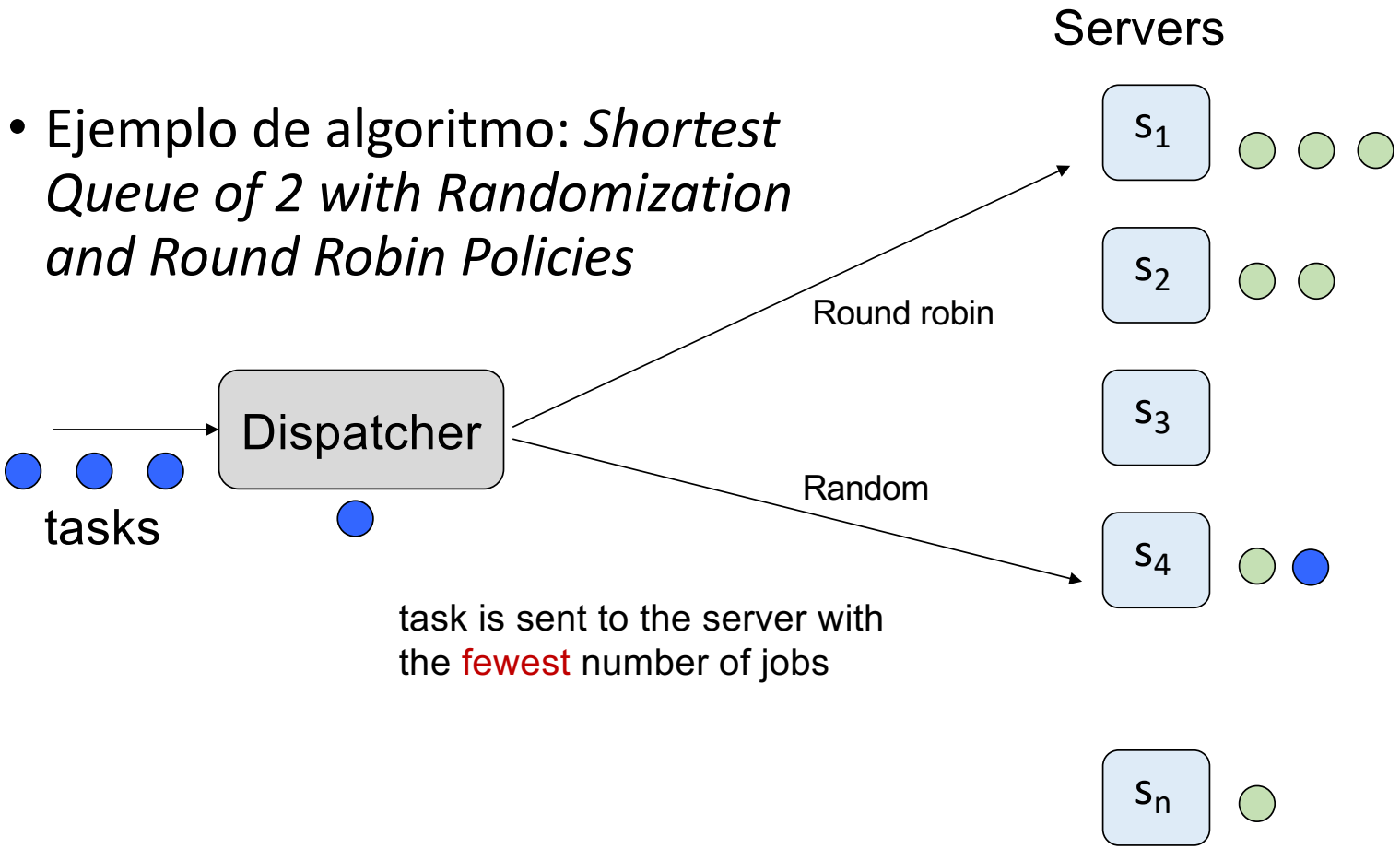
- Ejemplo de algoritmo: *Shortest Queue of 2 with Randomization and Round Robin Policies*



One server is chosen using Round Robin and the other randomly

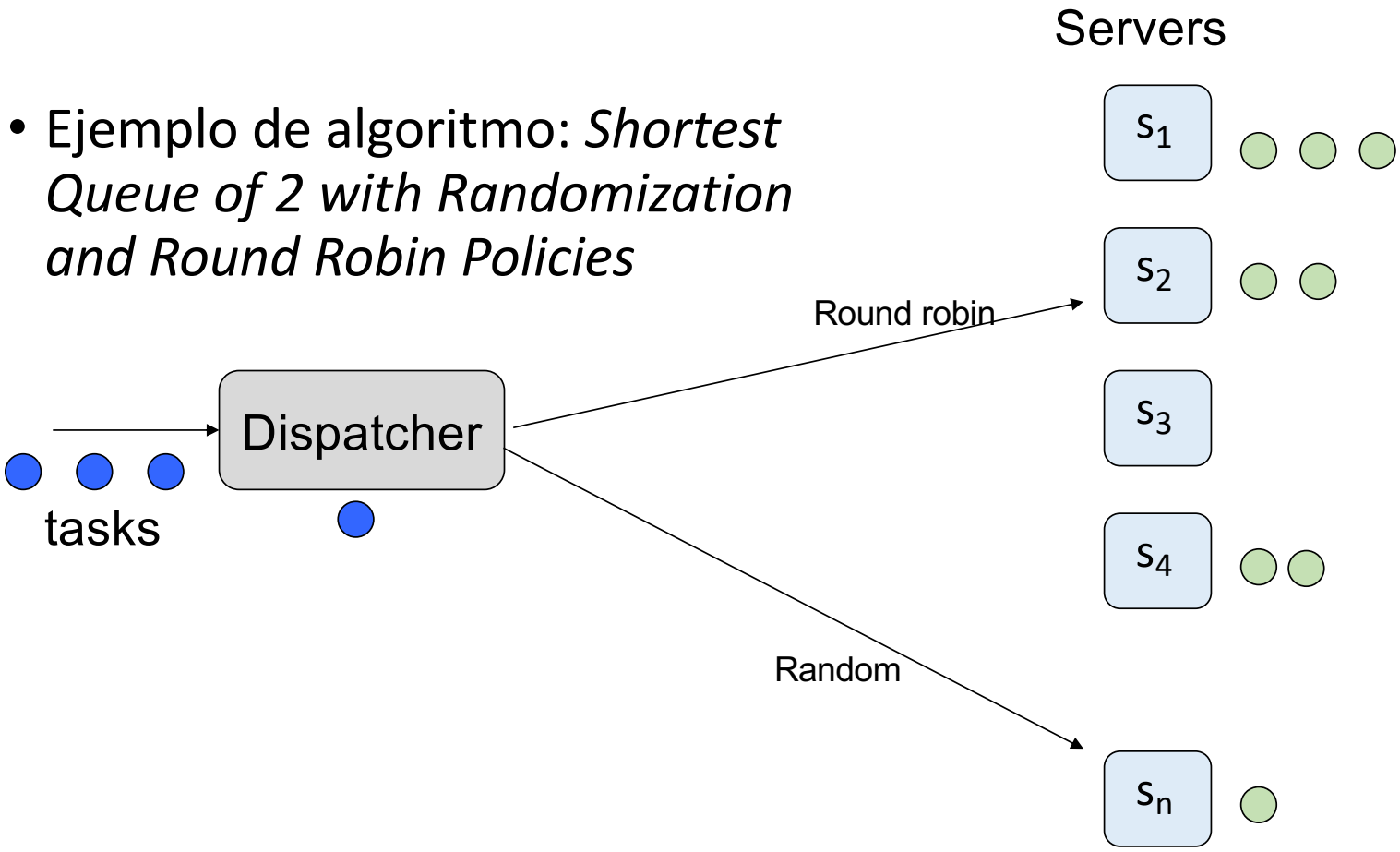
Motivación

- Ejemplo de algoritmo: *Shortest Queue of 2 with Randomization and Round Robin Policies*



Motivación

- Ejemplo de algoritmo: *Shortest Queue of 2 with Randomization and Round Robin Policies*



Ejemplo de análisis

- Queremos saber el comportamiento del sistema:
  - Tiempo medio de ejecución de las tareas
  - Número medio de tareas en las colas de los servidores esperando para su ejecución

Aspectos a considerar en el análisis

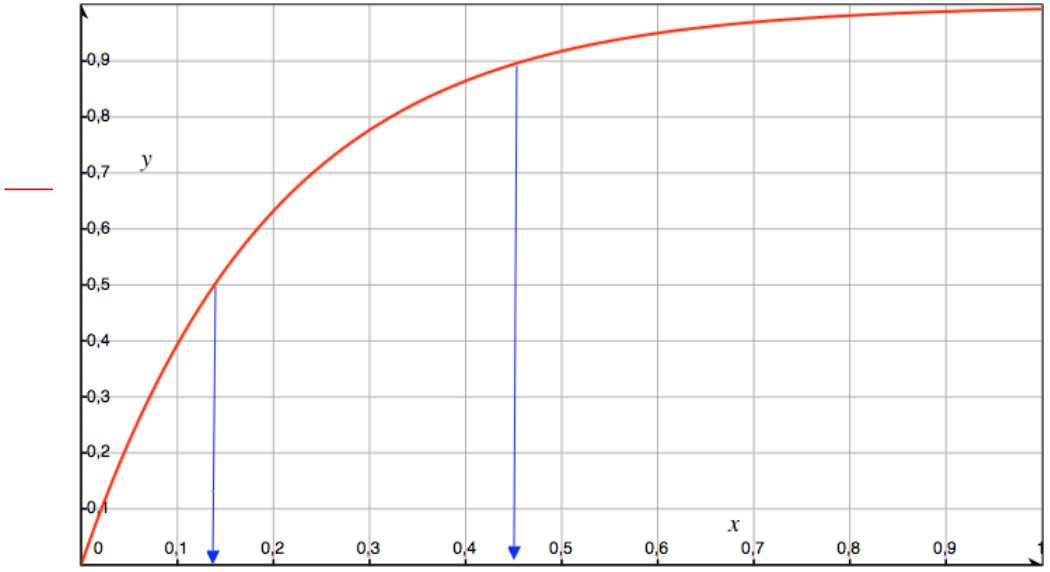
- Tiempo de llegada de las peticiones al dispatcher
- Tiempo de servicio de las peticiones en el servidor
- Orden de ejecución dentro del servidor (FCFS, FSJ, …)
- Estos tiempos se suelen modelizar utilizando una variable de distribución exponencial:
  - En el tiempo de llegada:
    - Distribución exponencial de parámetro $\lambda=5$
    - Tiempo medio entre llegadas = $1/\lambda = 0.2$ h = 12 minutos
  - En el tiempo de salida:
    - Tiempo medio = $1/\mu = 0.1666$ h = 10 minutos
    - Distribución exponencial de parámetro $\mu=6$ (6 trabajos por hora)

Distribución exponencial

$$F(x) = \begin{cases} 1 - e^{-\mu x} & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

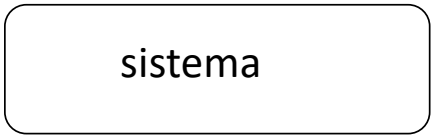
$para \quad \mu = 5$

Tiempo medio = 1/μ = 0.2 h = 12 minutos

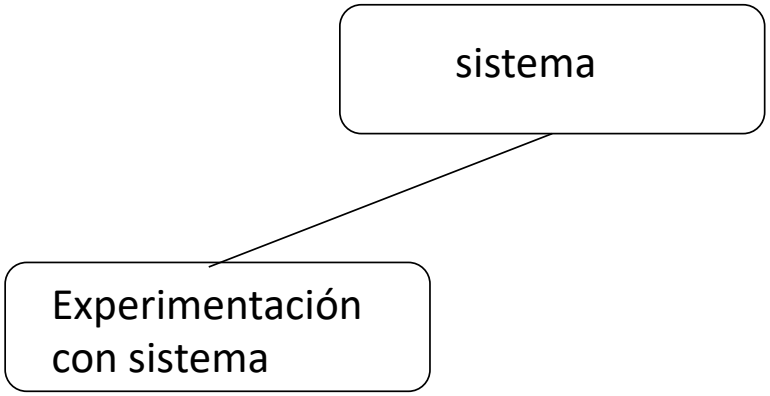


minutos

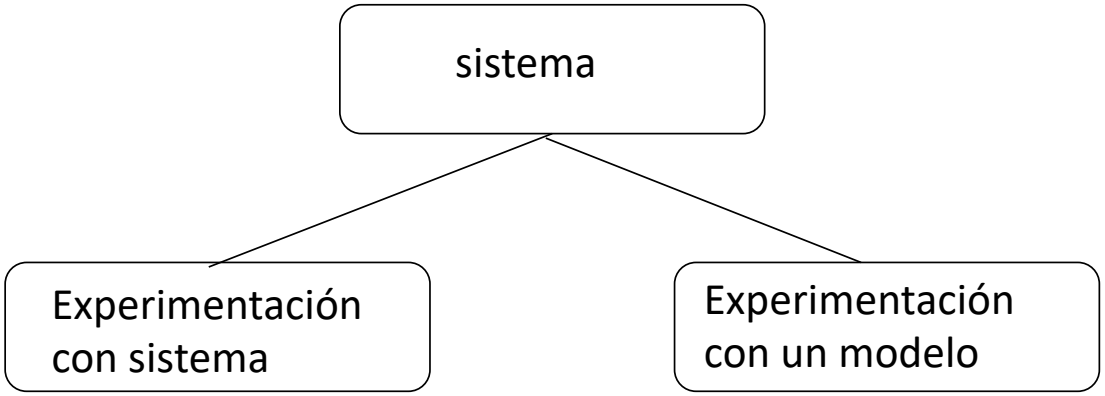
Enfoques de estudio



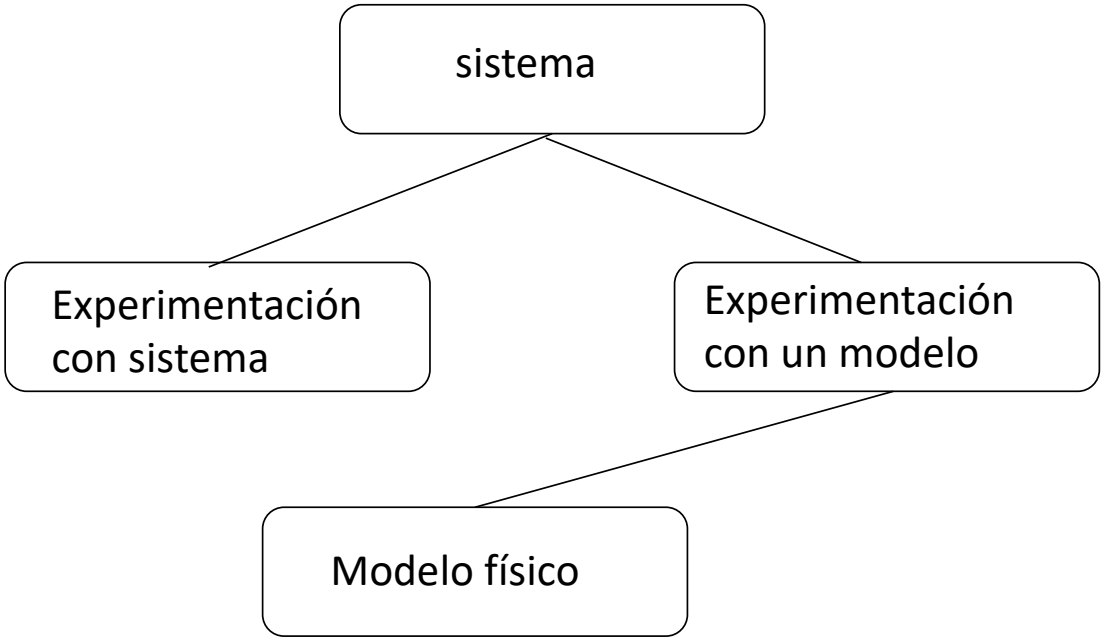
Enfoques de estudio



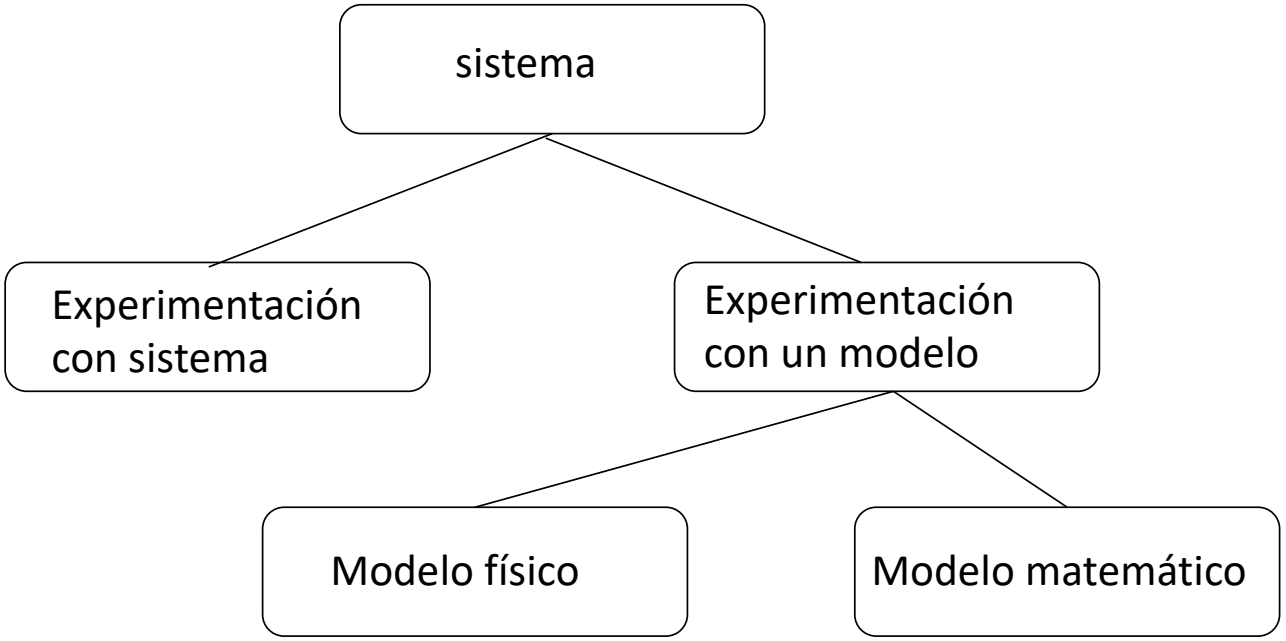
Enfoques de estudio



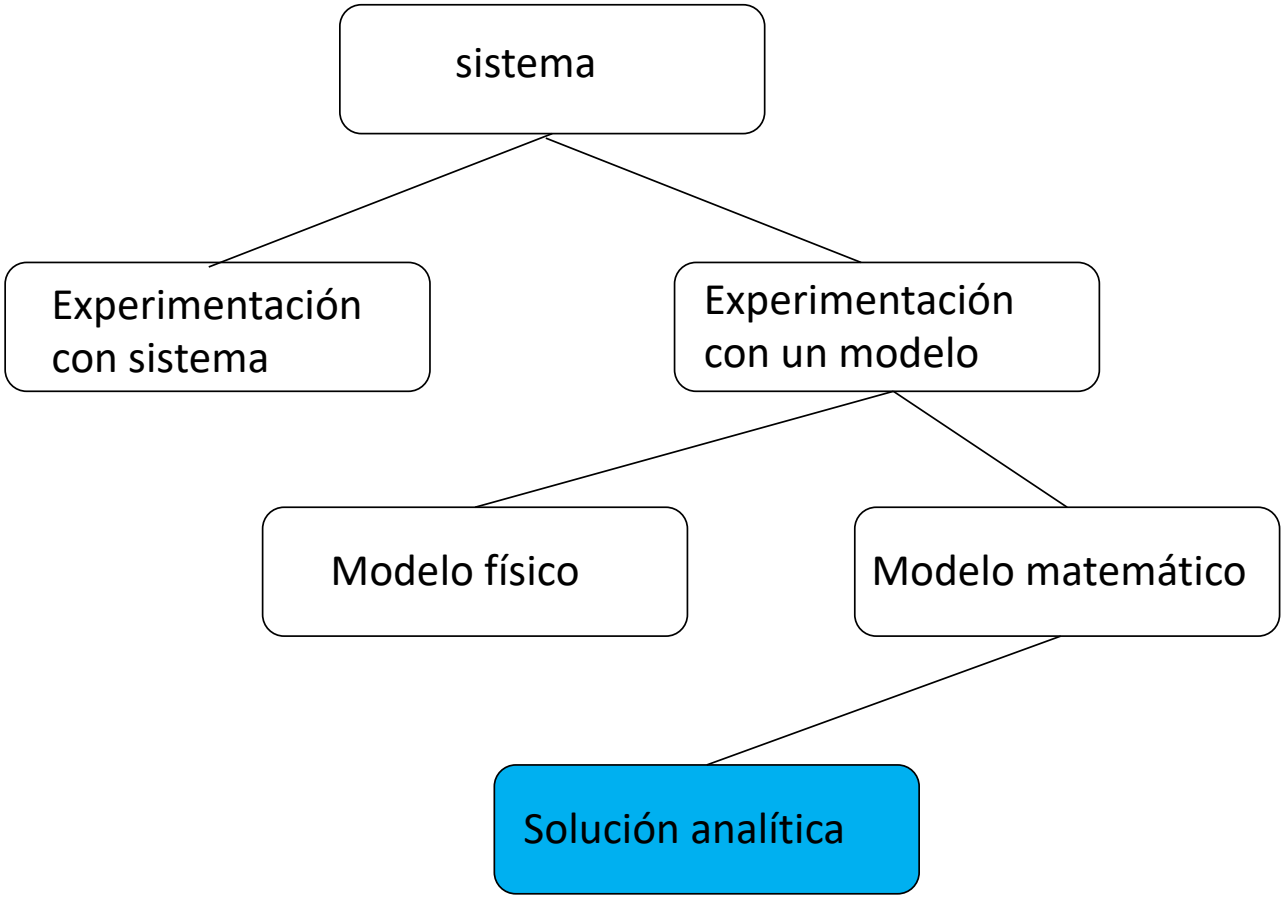
Enfoques de estudio



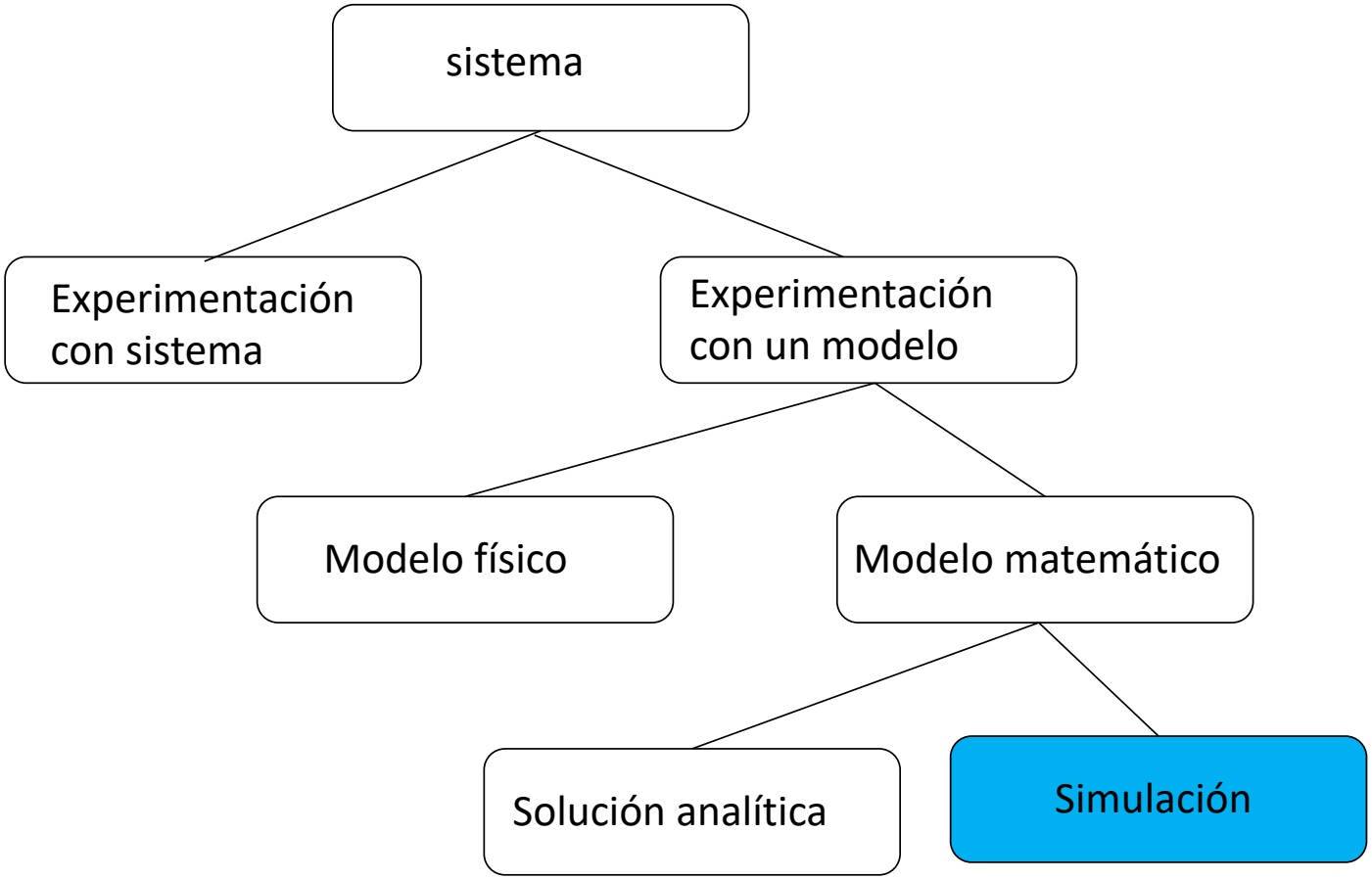
Enfoques de estudio



Enfoques de estudio



Enfoques de estudio



Tipos de sistemas

- Discretos: las variables de estado cambian de valor en instantes separados en el tiempo
  - El número de paquetes recibidos cambia cuando llega un paquete
  - El número de procesos en una cola de ejecución cambia cuando llega un nuevo proceso para ejecutar
- Continuos: las variables de estado cambian de forma continua con respecto al tiempo
  - La distancia recorrida por un avión cambia en cada instante del tiempo

Tipos de simulación

- Emulación
- Simulación estática/dinámica
- Simulación determinista/estocástica
- Simulación de sistemas continuos/discretos

Emulación

- Un emulador es una combinación de HW/SW que el recrea el funcionamiento/comportamiento de otro sistema, es decir, ofrece el servicio que ofrece el sistema que está emulando

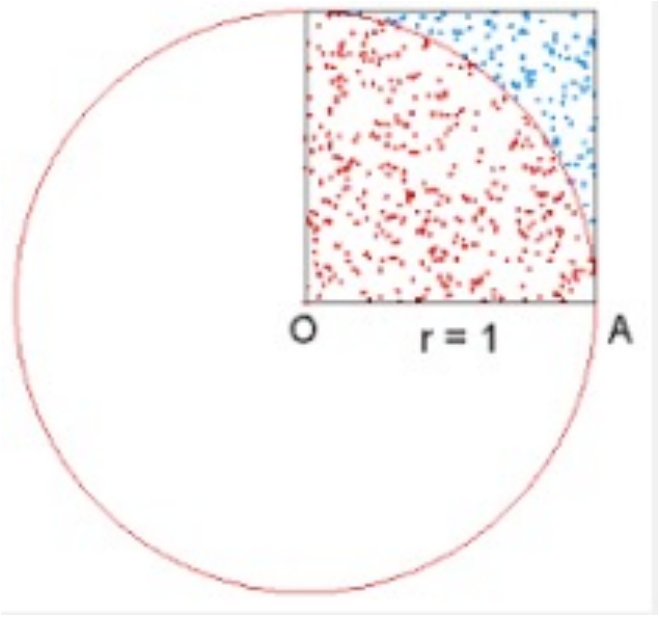
- Enfoque experimental que permite la ejecución de aplicaciones reales sobre un entorno virtual

Simulación estática/dinámica

- Simulación estática
  - No utiliza como parámetro el tiempo. La simulación se ejecuta hasta alcanzar un determinado equilibrio
  - Ejemplo: simulaciones de Monte Carlo
    - Simulación estática está dirigida por secuencias de números aleatorios
- Simulación dinámica
  - El estado del sistema varía con el tiempo

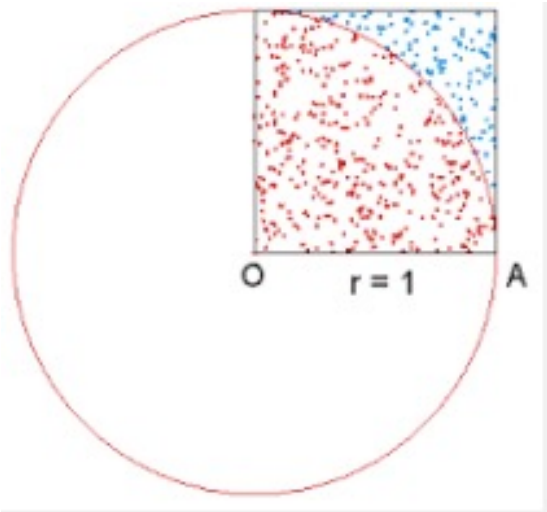
Ejemplo de simulación de Monte Carlo

- Estimar el valor del número π
- Se parte de un circulo con radio 1



- El área del cuadrado mostrado en la figura es 1
- El área de la superficie inscrita en este cuadrado es $A=\pi r^2/4= \pi/4$
  - El valor de $\pi=4A$
- Objetivo: determinar A

Ejemplo de simulación de Monte Carlo



- Se generan N pares de valores aleatorios (x,y)
- Se contabilizan cuantos caen dentro de arco Nc
- La probabilidad de que el punto esté dentro del arco es proporcional al área $A=(\pi \times 1^2) / 4$
- $\pi/4 = Nc/N \Rightarrow \pi = 4*Nc/N$
- Ejemplo:
  - N=1000000
  - Nc=785581
  - π=3,1423

Simulación determinista/estocástica

- Determinista
  - No interviene ninguna variable aleatoria
  - Salida=F(entrada, estado)
  - Ejemplo: ecuación diferencial que describe una reacción química
- Estocástica
  - Entradas y/o relaciones se modelan como variables aleatorias
  - Las salidas de la simulación deben tratarse como variables aleatorias
  - Ejemplo: la llegada de peticiones a un servidor Web no es determinista, sigue una determinada función de distribución

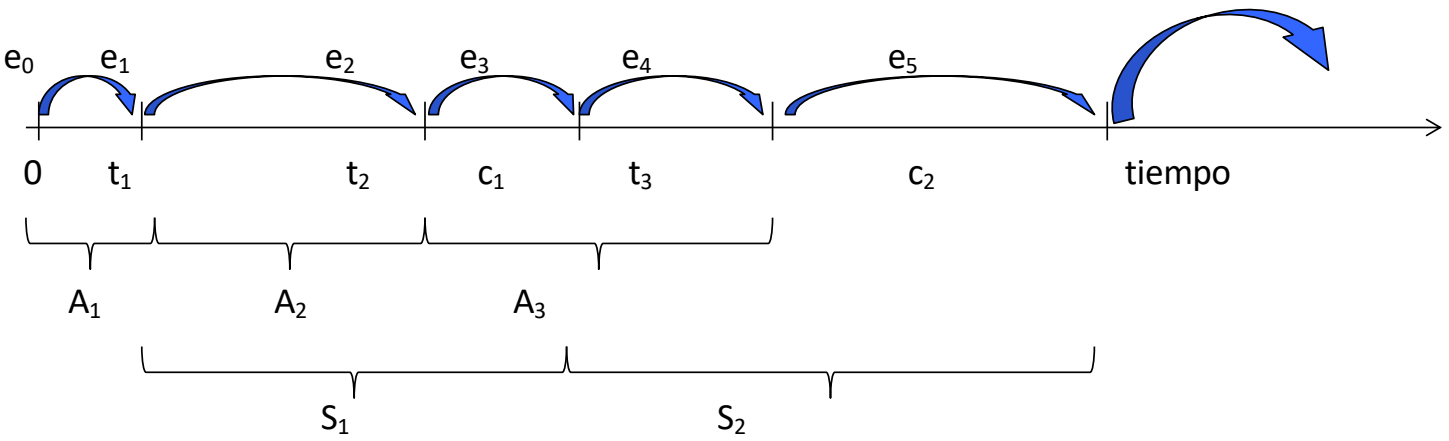
Simulación de sistemas continuos/discretos

- Simulación de sistemas continuos
  - Las variables de estado evolucionan de forma continua con respecto al tiempo
  - Simulación de sistemas físicos
- Simulación de sistemas discretos
  - Las variables de estado cambian de valor en momentos instantáneos de tiempo
  - Ejemplo: el número de procesos en una cola cambia en el momento en el que llega un nuevo trabajo

Simulación de eventos discretos

- Modela un sistema cuyo estado global cambia en función del tiempo
- El estado global se actualiza cada vez que ocurre un evento
- Características
  - Dinámica
  - Estocástica
  - Discreta

Eventos típicos



- e_i= evento del sistema
  - t_i = tiempo de llegada de una petición, tarea, trabajo, etc.
  - c_i= t_i+D_i+S_i= tiempo de servicio de una petición, tarea, etc
- A_i = t_j-t_i = tiempo entre llegadas
- S_i= tiempo de servicio para la petición, tarea, trabajo, etc.
- D_i= tiempo en la cola del servidor que atiende peticiones, etc.

Algoritmo de simulación

```
Inicializar las variables de estado
Tiempo = 0
Obtener el primer evento
while ((no haya mas eventos) AND (tiempo < máximo)) {
        avanzar el tiempo
        Obtener/eliminar el siguiente evento de la lista
        Procesar el evento {
                Actualizar el estado global
                Actualizar las estadísticas de la simulación
                Generar nuevos eventos y añadirlos a la lista
                de eventos
        }
}
Imprimir resultados
```

Generación de eventos

- Dirigidas por traza
  - Se registran las trazas de eventos que ocurren en un sistema real y se alimenta el simulador con las mismas
- Dirigidas por una distribución aleatoria
  - Los eventos y entradas son generadas a partir de una determina función de distribución
  - Ejemplo: la llegada de trabajos para su ejecución se puede modelar como la cantidad de tiempo que transcurre entre dos llegadas consecutivas. Esta cantidad se puede modelar con una función de distribución exponencial

Generación de valores aleatorios

- Un generador de números pseudoaleatorios genera una secuencia de valores z_i, que se aproximan a la distribución U(0,1)
- Los generadores de números pseudoaleatorios parten de una semilla inicial z0
  - Distintos valores de la semilla generan distintas secuencias de números

Requisitos de un generador de números pseudoaleatorios

- Aleatoriedad
  - Muestras aproximadas a U(0,1)
  - Sin correlación entre las muestras
- Eficiencia
  - Mide el tiempo y memoria necesaria para la generación
- Periodo máximo
  - Número de muestras que se pueden obtener antes de repetir la secuencia
  - El generador *Mersenne twister* tiene un periodo de $2^{19937}-1$
- Secuencia reproducible
  - Para poder repetir exactamente la misma simulación

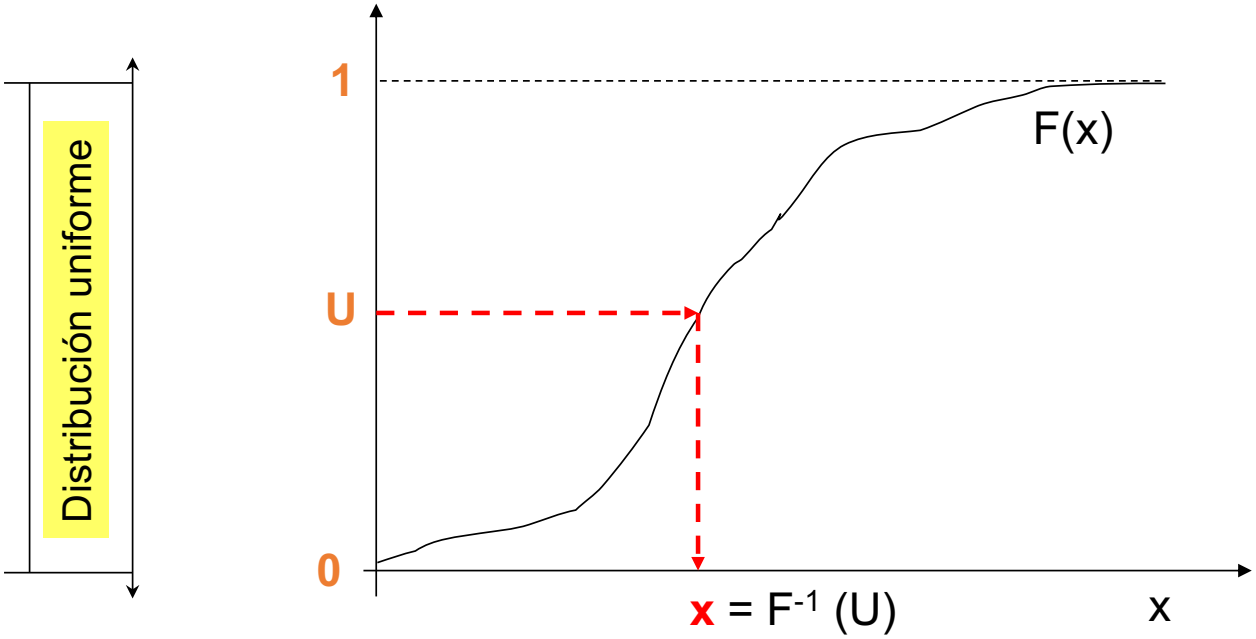
Generación de variables aleatorias

- ¿Cómo generar variables aleatorias de otras funciones de distribución a partir de una muestra de variables aleatorias con distribución U(0,1)?
- Métodos
  - Método de la transformada inversa
  - Método de la composición
  - Método de la convolución
  - Método de aceptación-rechazo

Método de la transformada inversa

- Sea X una variable aleatoria con una función de distribución F_x
- Para generar una variable aleatoria X ~ F_X
  - Se genera u ~ U(0,1)
  - Se devuelve x = $F^{-1}(u)$
    - x es una muestra de una variable aleatoria con función de distribución F_x
- $F^{-1}(U)$ siempre está definida porque $0 \leq U \leq 1$ y F está comprendida en [0,1]

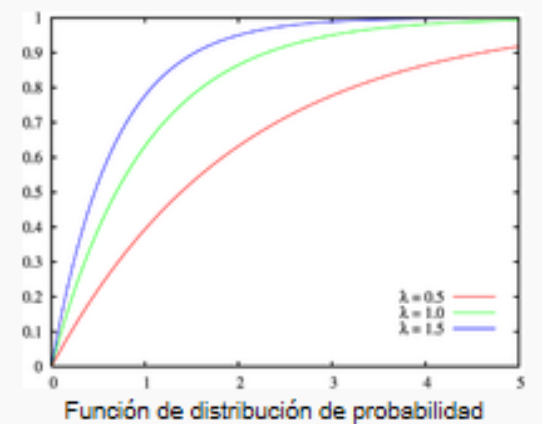
Método de la transformada inversa



Ejemplo: función de distribución exponencial

- Si X es una variable aleatoria exponencial con parámetro λ:
- Función de distribución:

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$



Función de distribución de probabilidad

- Media y varianza $E[X] = \frac{1}{\lambda}$ $V(X) = \frac{1}{\lambda^2}$

Ejemplo: función de distribución exponencial

- Si *u* es una muestra de una variable aleatoria U(0,1)
- Entonces:

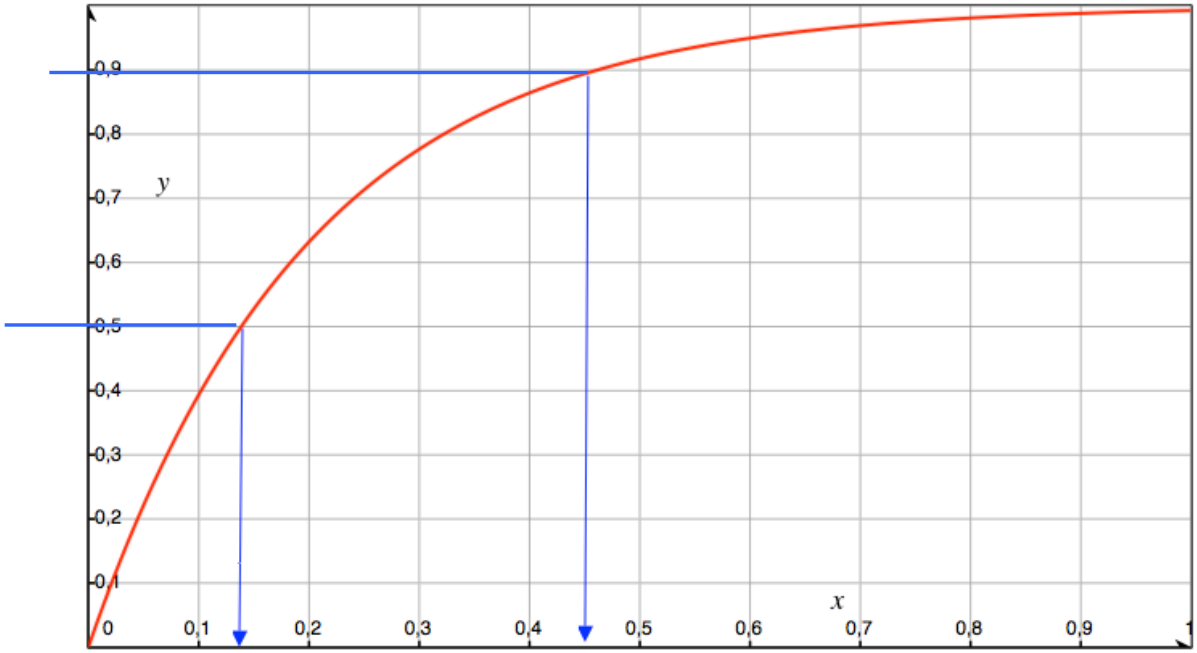
$$x = -\frac{1}{\lambda}\ln(1-u)$$

- *x* es una muestra de una variable aleatoria exponencial con parámetro λ

Ejemplo

$$F(x) = \begin{cases} 1 - e^{-\mu x} & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

$$para \quad \mu = 5$$



Entornos y lenguajes de simulación

- OMNet++
- PARSEC
- Desmo-J
- JavaSim
- Adves
- OPNET
- Arena
- Simio
- Simscript
- Simulink
- Simgrid

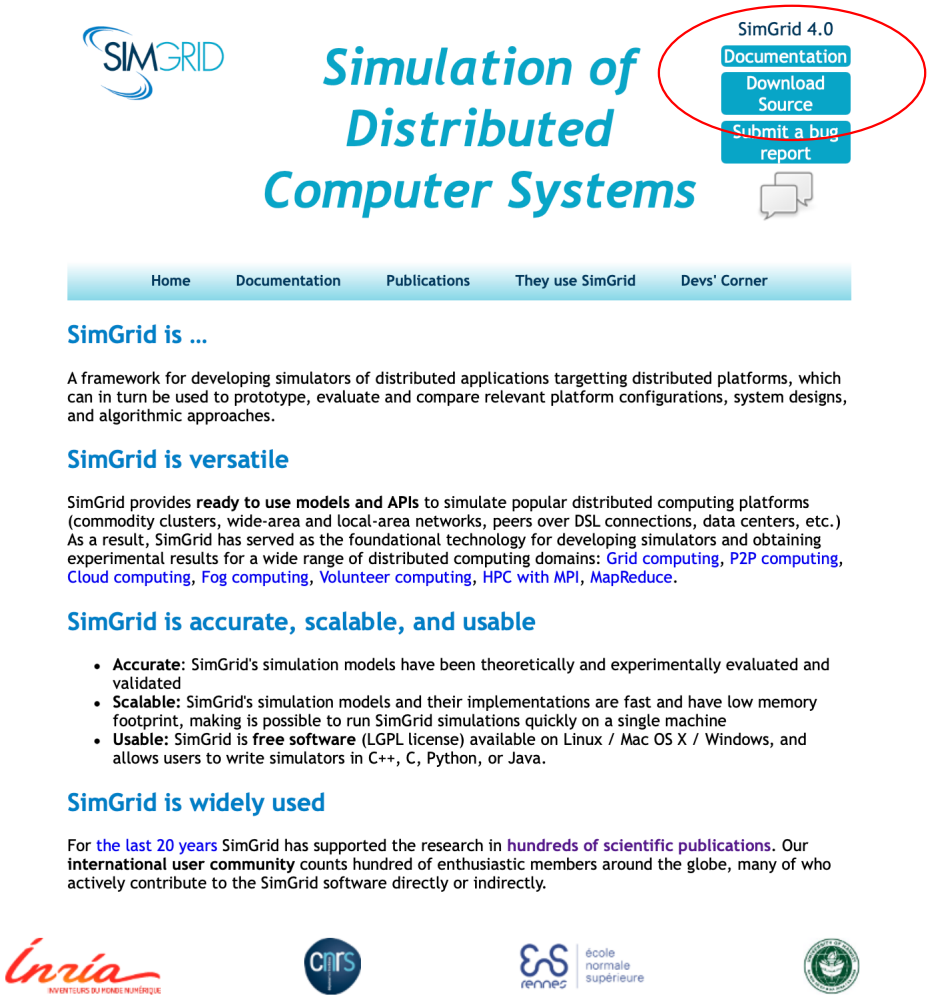
Introducción a SIMGRID

- SIMGRID: *Versatile Simulation of Distributed Systems*
- *https://simgrid.org/*
- Permite la simulación de sistemas distribuidos de gran escala
  - Grids, Clouds, HPC, P2P, aplicaciones MPI, …

Introducción a SIMGRID

- Ofrece un conjunto de funcionalidades para simular aplicaciones distribuidas en entornos distribuidos heterogéneos
- Interfaces:
  - S4U
  - SMPI para código MPI

Instalación



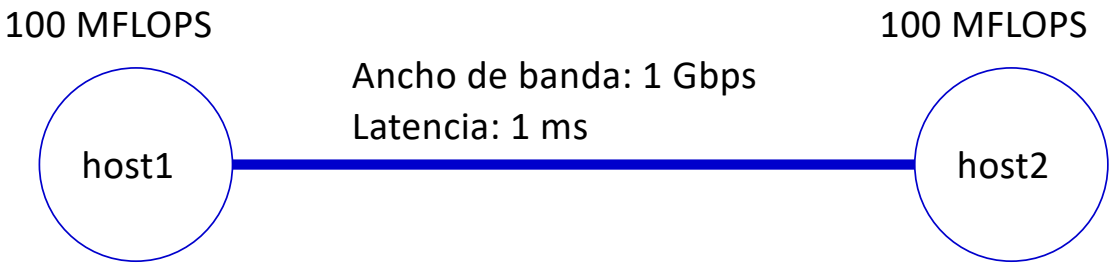
Instalación

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

cd simgrid

mkdir /directorio_install

cmake -DCMAKE_INSTALL_PREFIX=/ directorio_install

make

make install
```

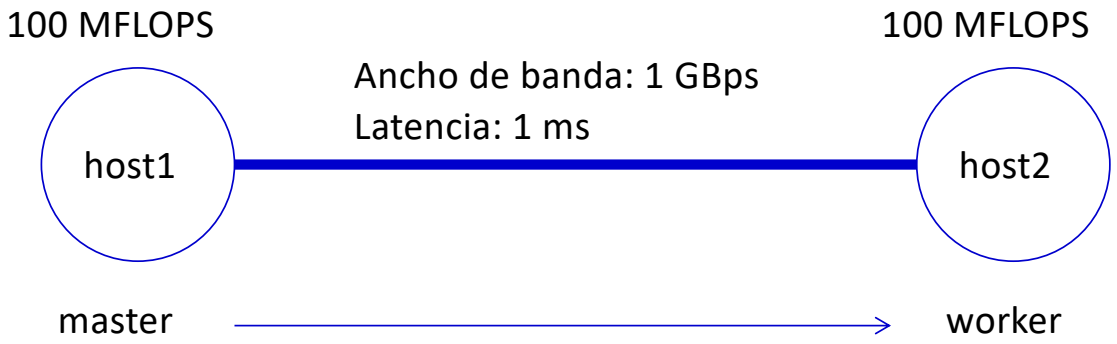
Interfaz S4U

- Permite describir algoritmos para entornos distribuidos, cloud, P2P, HPC, IoT y similares
- Principales abstracciones:
  - Actor: algún código, con datos, que ejecuta en un host
  - Host: localización en la que ejecutan los actores
  - Mailbox: lugar donde se envían los mensajes y reciben los mensajes
    - Independientes de la posición de la red
    - Los mensajes se envían a un *mailbox* y se reciben de un *mailbox*
    - Se identifican como strings
      - Funcionamiento síncrono o asíncrono

Ejemplo


100 MFLOPS
100 MFLOPS
Ancho de banda: 1 Gbps
Latencia: 1 ms
host1
host2

Ejemplo

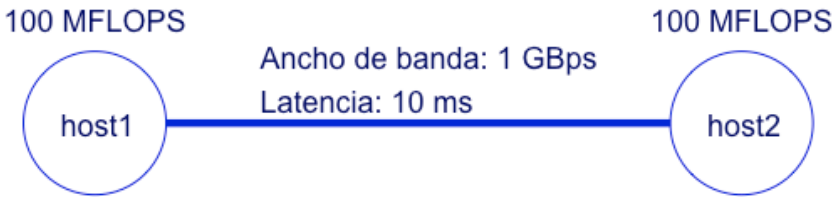


100 tareas

- Tamaño: 100000 bytes
- Duración del cálculo: 50 MFLOPS
  (cantidad de procesamiento en flops)

Descripción de la plataforma platform.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM
"http://simgrid.org/simgrid.dtd">
<platform version="4.1">

<zone id="AS0" routing="Full">
        <host id="host1" speed="1E8f"/>
        <host id="host2" speed="1E8f"/>

        <link id="link1" bandwidth="1GBps" latency="1ms"/>

        <route src="host1" dst="host2">
                <link_ctn id="link1"/>
        </route>

</zone>
</platform>
```



Descripción del despliegue deployment.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <!-- The master actor (with some arguments) -->
  <actor host="host1" function="master">
    <argument value="20"/>       <!-- Number of tasks -->
    <argument value="50000000"/>  <!-- Computation size of tasks -->
    <argument value="1000000"/>   <!-- Communication size of tasks -->
    <argument value=»1"/>         <!-- Number of workers -->
  </actor>

  <!-- The worker actors (with mailbox to listen on as argument) -->
  <actor host="host2" function="worker" >
    <argument value="0"/>
  </actor>
value="1"/>
  </actor>
</platform>
```

Definición del master

```
#define FINALIZE 221297

void master(int argc, char* argv[])
{
  long number_of_tasks = xbt_str_parse_int(argv[1], "Invalid amount of tasks");       /* - Number of tasks  */
  double comp_size     = xbt_str_parse_double(argv[2], "Invalid computational size"); /* - Compute cost    */
  long comm_size       = xbt_str_parse_int(argv[3], "Invalid communication size");    /* - Communication size */
  long workers_count   = xbt_str_parse_int(argv[4], "Invalid amount of workers");     /* - Number of workers  */

  XBT_INFO("Got %ld workers and %ld tasks to process", workers_count, number_of_tasks);

  for (int i = 0; i < number_of_tasks; i++) { /* For each task to be executed: */
    char mailbox_name[100];
    char task_name[100];

    double* payload = xbt_malloc(sizeof(double));

    sprintf(mailbox_name,  "worker-%ld", i % workers_count); /* - Select a @ref worker in a round-robin way */
    sprintf(task_name,  "Task_%d", i);

    sg_mailbox_t mailbox = sg_mailbox_by_name(mailbox_name);
    *payload             = comp_size;


    sg_mailbox_put(mailbox, payload, comm_size); /* - Send the amount of flops to compute to the @ref worker */

  }
```

Definición del master (cont)

```
/* - Eventually tell all the workers to stop by sending a "finalize" task */

    for (int i = 0; i < workers_count; i++) {
        char mailbox_name[100];
        sprintf(mailbox_name,  "worker-%ld", i % workers_count);

        double* payload      = xbt_malloc(sizeof(double));
        sg_mailbox_t mailbox = sg_mailbox_by_name(mailbox_name);

        *payload             = FINALIZE;
        sg_mailbox_put(mailbox, payload, 0);
    }
}
```

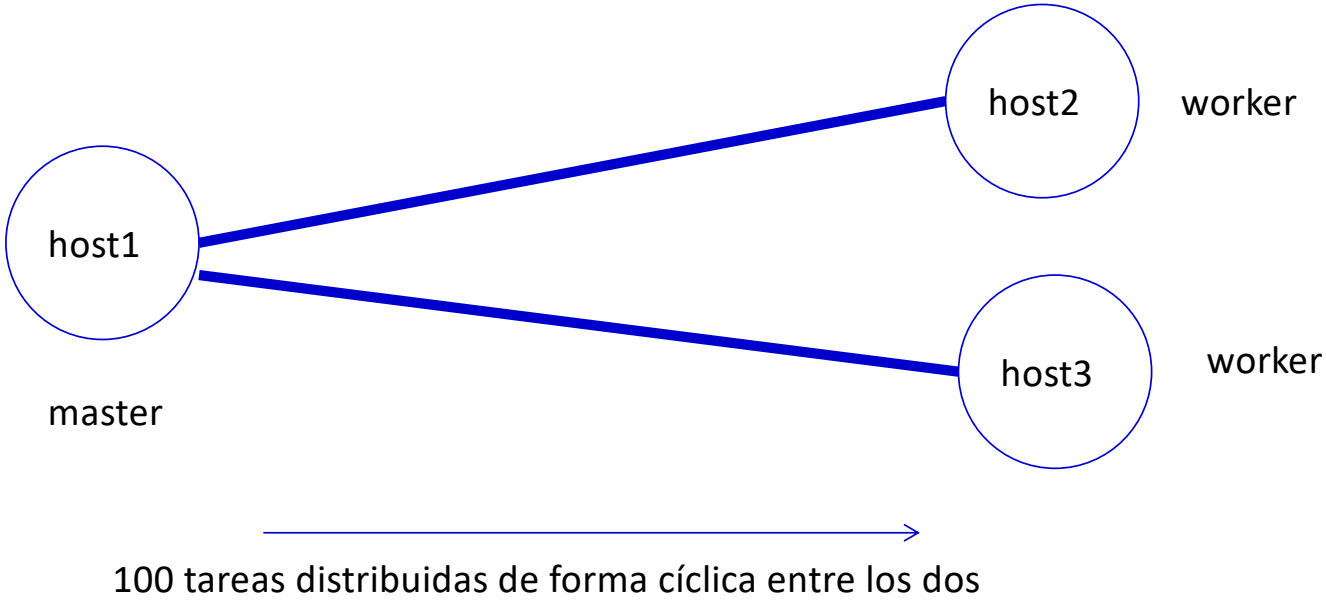
Definición del worker

```
void worker(int argc, char* argv[])
{
  char mailbox_name[100];

  long id = xbt_str_parse_int(argv[1], "Invalid argument");

  sprintf(mailbox_name,  "worker-%ld", id);
  sg_mailbox_t mailbox = sg_mailbox_by_name(mailbox_name);

  while (1) { /* The worker wait in an infinite loop for tasks sent by the @ref master */
    double* payload = (double*)sg_mailbox_get(mailbox);

    if (*payload == FINALIZE) {
      xbt_free(payload); /* - Exit if 'finalize' is received */
      break;
    }
    sg_actor_execute(*payload); /*  - Otherwise, process the received number of flops*/
    xbt_free(payload);
  }
  XBT_INFO("I'm done. See you!");
}
```

Función `main()`

```
int main(int argc, char* argv[])
{
  simgrid_init(&argc, argv);
  xbt_assert(argc > 2,
             "Usage: %s platform_file deployment_file\n"
             "\tExample: %s platform.xml deployment.xml\n",
             argv[0], argv[0]);

  simgrid_load_platform(argv[1]); /* - Load the platform description */

  simgrid_register_function("master", master); /* - Register the function to be executed by the actors */
  simgrid_register_function("worker", worker);

  simgrid_load_deployment(argv[2]); /* - Deploy the application */

  simgrid_run(); /* - Run the simulation */

  XBT_INFO("Simulation time %g", simgrid_get_clock());

  return 0;
}
```

Modificación. Añadir un host adicional



Nueva plataforma

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <zone  id="AS0"  routing="Full">
    <host id="host1" speed="1E8f"/>
    <host id="host2" speed="1E8f"/>
    <host id="host3" speed="1E8f"/>


    <link id="link1"  bandwidth="1GBps" latency="1ms"/>
    <link id="link2"  bandwidth="1GBps" latency="1mss"/>

    <route src="host1" dst="host2">
            <link_ctn id="link1"/>
    </route>

    <route src="host1" dst="host3">
      <link_ctn id="link2"/>
    </route>
  </zone>
</platform>
```

Nuevo despliegue

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM
"http://simgrid.gforge.inria.fr/simgrid/simgrid.dtd">
<platform version="4.1">

<!-- The master process -->
        <process host="host1" function="master">
                <argument value="100"/><!--argv[1]:#tasks-->
                <argument value="2"/><!--argv[2]:#workers-->
        </process>

<!-- The workers -->
        <process host="host2" function="worker">
                <argument value="0"/>
        </process>
        <process host="host3" function="worker">
                <argument value="1"/>
        </process>
</platform>
```

Descripción de plataformas en SimGrid

- Plataforma en SimGrid: archivo XML que describe:
  - Hosts con su potencia de cómputo
  - Enlaces con sus anchos de banda y latencia
  - Rutas entre hosts

```xml
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid/simgrid.dtd">
<platform version="4.1">
  <zone id="first zone" routing="Full">
    <!-- the resources -->
    <host id="host1" speed="1Mf"/>
    <host id="host2" speed="2Mf"/>
    <link id="link1" bandwidth="125MBps" latency="100us"/>
    <!-- the routing: specify how the hosts are interconnected -->
    <route src="host1" dst="host2">
      <link_ctn id="link1"/>
    </route>
  </zone>
</platform>
```

Elementos que se pueden describir

- *Host*
- *Link*
- *Router*
- *Cluster*: hosts interconectados por una red dedicada

Especificación de un host

```
<host id="host1”
      speed="1000000000f”
      [state=“ON”]
      [availability_file=“host.trace”]
      [core=“4”]
```

- `id`: identificador del host
- `speed`: potencia en Flops
- `availability_file`: ficher de traza asociado
- `state`: estado inicial del host (ON/OFF)
- `core`: número de cores

Fichero de disponibilidad

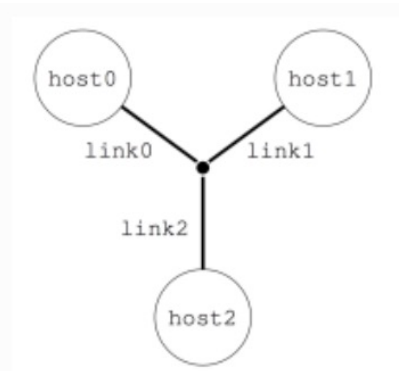
```
<platform version="4">
  <host id="bob" speed="500Gf" availability_file="bob.trace" />
</platform>
```

Example of "bob.trace" file

```
PERIODICITY 1.0
  0.0 1.0
  11.0 0.5
  20.0 0.8
```

- En el instante 0: el host proporciona 500 Mflops
- En el instante 11.0: entrega 250 Mflops
- En el instante 20.0: entrega el 80%, 400 Mflps
- El proceso se repite

Ejemplo




```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <zone id="AS0" routing="Full">
    <host id="host0" speed="1Gf"/>
    <host id="host1" speed="2Gf"/>
    <host id="host2" speed="40Gf"/>
    <link id="link0" bandwidth="125MBps" latency="100us"/>
    <link id="link1" bandwidth="50MBps" latency="150us"/>
    <link id="link2" bandwidth="250MBps" latency="50us"/>
    <route src="host0" dst="host1"><link_ctn id="link0"/><link_ctn id="link1"/></route>
    <route src="host1" dst="host2"><link_ctn id="link1"/><link_ctn id="link2"/></route>
    <route src="host0" dst="host2"><link_ctn id="link0"/><link_ctn id="link2"/></route>
  </zone>
</platform>
```

Enlaces de red

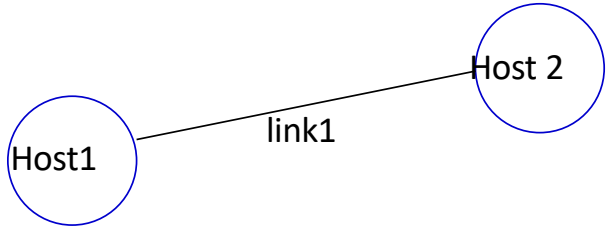
```
<link id="enlace_1"
        bandwidth="125MBps"
        latency="0.01"
       [sharing_policy="SHARED"]  />
```

Unit	Meaning	Duration in seconds
ps	picosecond	10^{-12} = 0.000000000001
ns	nanosecond	10^{-9} = 0.000000001
us	microsecond	10^{-6} = 0.000001
ms	millisecond	10^{-3} = 0.001
s	second	1
m	minute	60
h	hour	60 * 60
d	day	60 * 60 * 24
w	week	60 * 60 * 24 * 7

- `id`: identificador del enlace
- `bandwidth`: ancho de banda en bytes/s
- `latency`: laencia del enlace
- `sharing_policy_`
  - `SHARED` (por defecto): Los flujos comparten el ancho de banda disponible
  - `FATPIPE`: Cada flujo obtiene todo el ancho de banda del enlace

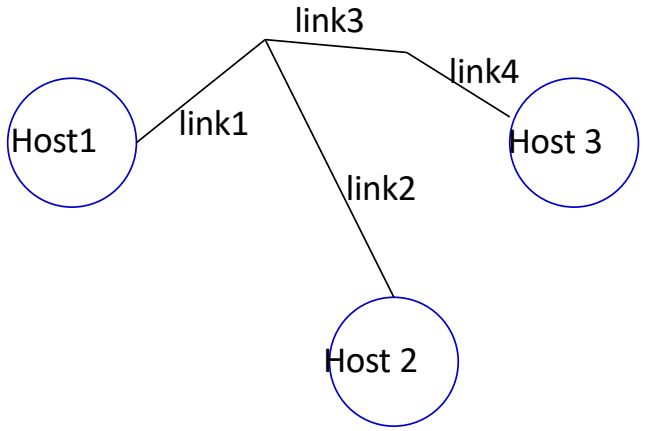
Rutas

```
<host id="host1" speed="1E8"/>
<host id="host2" speed="1E8"/>
<link id="link1" bandwidth="1Gbps" latency="100us"/>

<route src="host1" dst="host2">
    <link_ctn id="link1"/>
</route>
```



Rutas con múltiples saltos

```
<host id="host1" speed="1E8"/>
<host id="host2" speed="1E8"/>
<host id="host2" speed="1E8"/>

<link id="link1" bandwidth="1E6" latency=”100ms"/>
<link id="link2" bandwidth="1E6" latency=”500us"/>
<link id="link3" bandwidth=“2E6" latency=”150ms"/>
<link id="link4" bandwidth=“2E6" latency=”100ms"/>

<route src="host1" dst="host3">
    <link_ctn id="link1"/>
    <link_ctn id="link3"/>
    <link_ctn id="link4"/>
</route>

<route src="host2" dst="host3">
    <link_ctn id="link2"/>
    <link_ctn id="link3"/>
    <link_ctn id="link4"/>
</route>
```

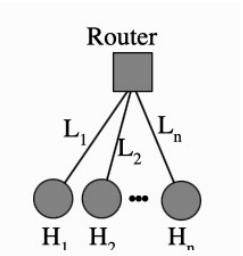


Especificaciones de routers

```
<router id=”R1”>
<router id=”R2”>

<route src=“A” dest=”R1”>
   <link_ctn id=“link1”/>
</route>

<route src=“R1” dest=”B”>
   <link_ctn id=“link2”/>
</route>

<route src=“R1” dest=”C”>
   <link_ctn id=“link3”/>
</route>
```

Definición de clusters

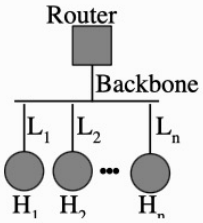
Cluster with a Crossbar




```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <zone id="world" routing="Full">
    <cluster id="cluster-crossbar"
             prefix="node-" radical="0-65535" suffix=".simgrid.org"
             speed="1Gf" bw="125MBps" lat="50us"/>
  </zone>
</platform>
```

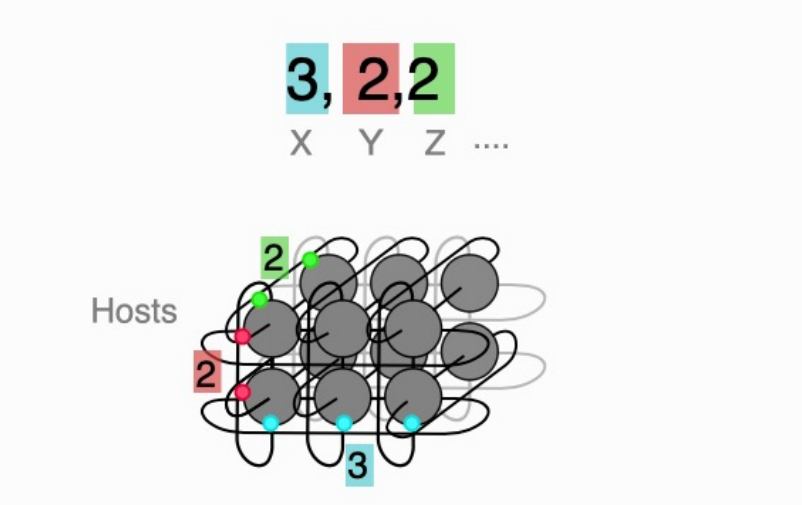
Definición de clusters

Cluster with a Shared Backbone



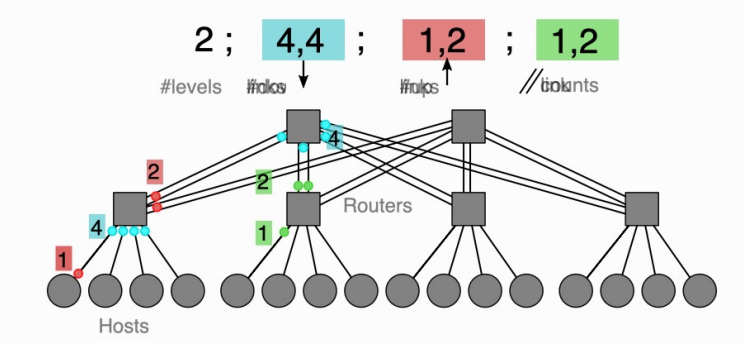

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <cluster id="cluster0" prefix="node-" radical="0-99" suffix=".simgrid.org"
           speed="1Gf" bw="125MBps" lat="50us"
           bb_bw="2.25GBps"  bb_lat="500us"/>
</platform>
```

Torus Cluster




```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <zone id="world" routing="Full">
    <cluster id="bob_cluster" topology="TORUS" topo_parameters="3,2,2"
             prefix="node-" radical="0-11" suffix=".simgrid.org"
             speed="1Gf" bw="125MBps" lat="50us"
             loopback_bw="100MBps" loopback_lat="0"/>
  </zone>
</platform>
```

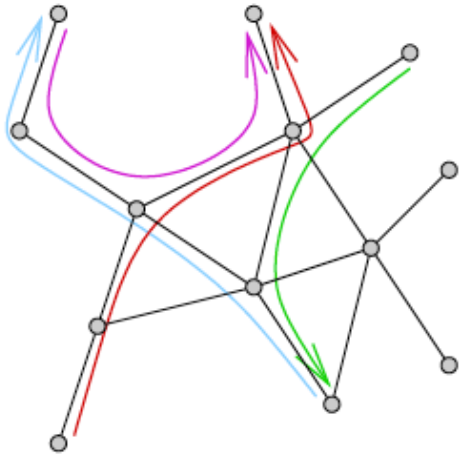
Fat-Tree Cluster




```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <zone id="world" routing="Full">
    <cluster id="bob_cluster"
             prefix="node-" radical="0-15" suffix=".simgrid.org"
             speed="1Gf" bw="125MBps" lat="50us"
             topology="FAT_TREE" topo_parameters="2;4,4;1,2;1,2"
             loopback_bw="100MBps" loopback_lat="0" />
  </zone>
</platform>
```

Modelización básica de la red

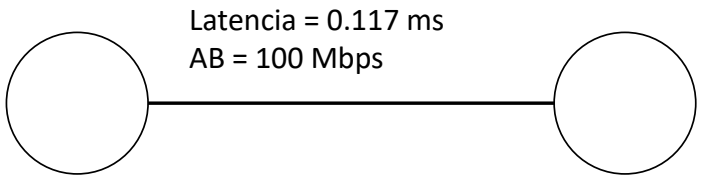
$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

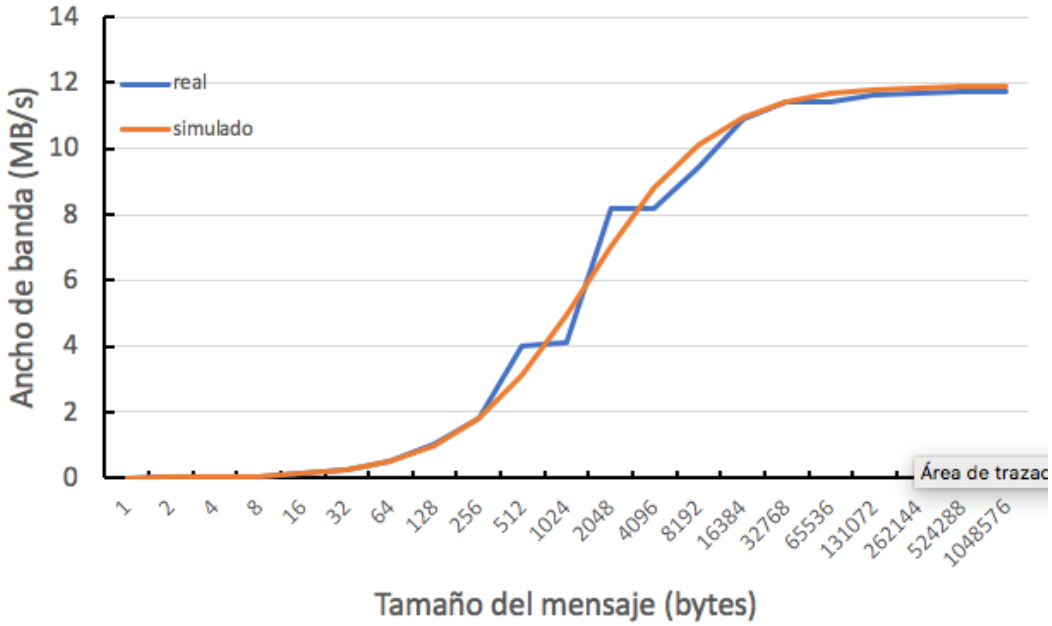


Uso de modelos analíticos
para una rápida simulación

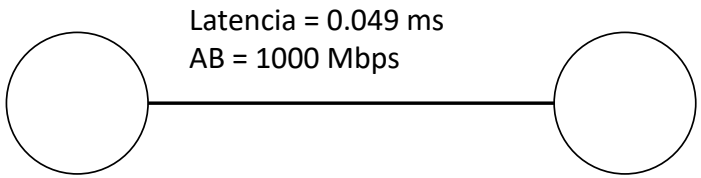
Velho, Pedro, Schnorr, Lucas Mello, Casanova, Henri, Legrand, Arnaud.
On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations.
ACM Transactions on Modeling and Computer Simulation (TOMACS), 2013.
WWW doi:10.1145/2517448

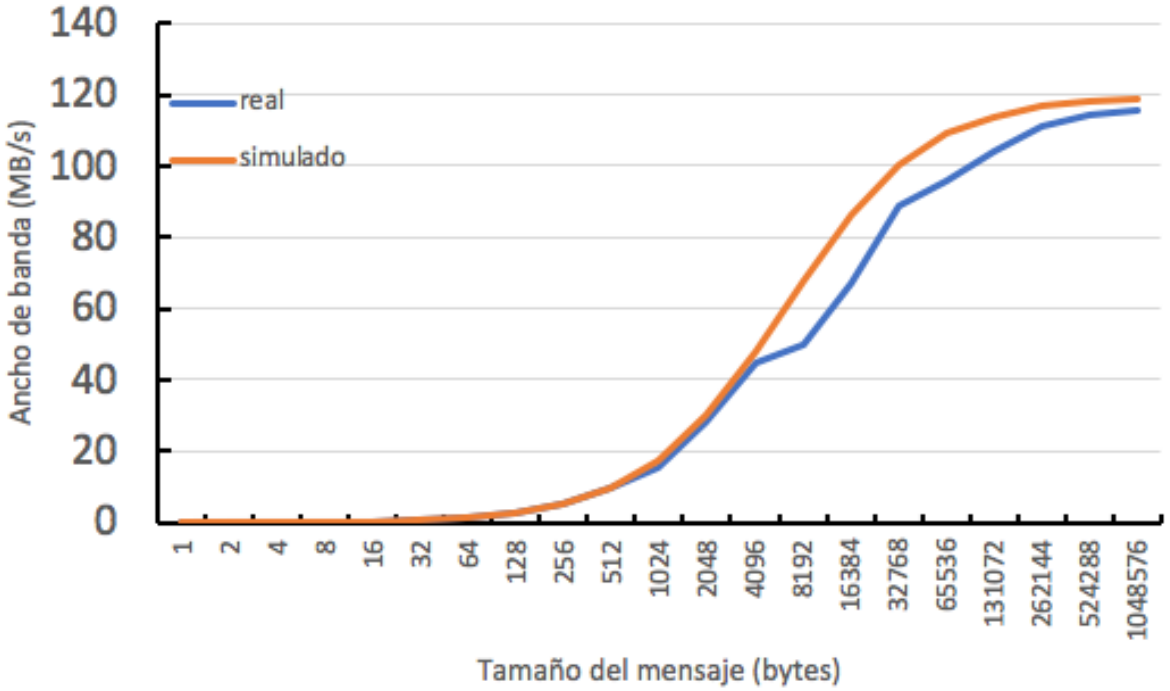
Precisión del modelo de red





Precisión del modelo de red





Tipos de comunicación

- Comunicaciones síncrona:
  - void sg_mailbox_put (sg_mailbox_t mailbox, void *payload, long simulated_size_in_bytes)
  - void sg_mailbox_get (sg_mailbox_t mailbox);

- Comunicación asíncrona
  - sg_comm_t sg_mailbox_put_async(sg_mailbox_t mailbox, void *payload, long simulated_size_in_bytes)
  - sg_comm_t sg_mailbox_get_async (sg_mailbox_t mailbox, void **data)

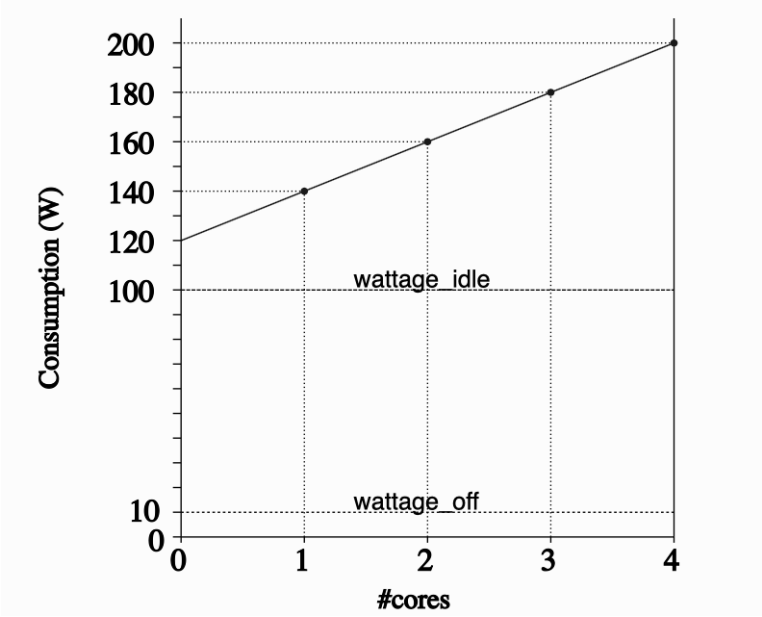
Modelización de la CPU

- Una CPU tiene una potencia en flop/sec
- Una tarea requiere *size* flops
- El tiempo para ejecutar la tarea será:

$$\frac{size}{pow}\,\mathrm{sec}$$

Energía

```
<host id="HostA" speed="100.0Mf" core="4">
    <prop id="wattage_per_state" value="100.0:120.0:200.0" />
    <prop id="wattage_off" value="10" />
</host>
```



Ejemplo

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <cluster bb_bw="2.25GBps" bb_lat="500us" bw="125MBps" core="4" id="cluster" lat="50us" prefix="MyHost" radical="1-2"
 speed="100.0Mf,50.0Mf,20.0Mf" suffix="">
    <!--  List of idle_power:epsilon_power:max_power pairs (in Watts) -->
    <!--  The list must contain one speed tuple for each previously defined pstate-->
    <prop id="wattage_per_state" value="100.0:93.0:200.0, 93.0:90.0:170.0, 90.0:90.0:150.0" />
    <prop id="wattage_off" value="10" />
  </cluster>
</platform>
~
```

Ejemplo (exec-energy)

```
static void dvfs(int argc, char* argv[])
{
  sg_host_t host = sg_host_by_name("MyHost1");

  XBT_INFO("Energetic profile: %s", sg_host_get_property_value(host, "wattage_per_state"));
  XBT_INFO("Initial peak speed=%.0E flop/s; Energy dissipated =%.0E J", sg_host_get_speed(host),
           sg_host_get_consumed_energy(host));

  double start = simgrid_get_clock();
  XBT_INFO("Sleep for 10 seconds");
  sg_actor_sleep_for(10);
  XBT_INFO("Done sleeping (duration: %.2f s). Current peak speed=%.0E; Energy dissipated=%.2f J",
           simgrid_get_clock() - start, sg_host_get_speed(host), sg_host_get_consumed_energy(host));

  // Run a task
  start = simgrid_get_clock();
  XBT_INFO("Run a task of %.0E flops", 100E6);
  sg_actor_execute(100E6);
  XBT_INFO("Task done (duration: %.2f s). Current peak speed=%.0E flop/s; Current consumption: from %.0fW to %.0fW"
           " depending on load; Energy dissipated=%.0f J",
           simgrid_get_clock() - start, sg_host_get_speed(host), sg_host_get_wattmin_at(host, sg_host_get_pstate(host)),
           sg_host_get_wattmax_at(host, sg_host_get_pstate(host)), sg_host_get_consumed_energy(host));
```

Ejemplo (exec-energy)

```
// ========= Change power peak =========
int pstate = 2;
sg_host_set_pstate(host, pstate);
XBT_INFO("========= Requesting pstate %d (speed should be of %.0E flop/s and is of %.0E flop/s)", pstate,
         sg_host_get_pstate_speed(host, pstate), sg_host_get_speed(host));

// Run a second task
start = simgrid_get_clock();
XBT_INFO("Run a task of %.0E flops", 100E6);
sg_actor_execute(100E6);
XBT_INFO("Task done (duration: %.2f s). Current peak speed=%.0E flop/s; Energy dissipated=%.0f J",
         simgrid_get_clock() - start, sg_host_get_speed(host), sg_host_get_consumed_energy(host));

start = simgrid_get_clock();
XBT_INFO("Sleep for 4 seconds");
sg_actor_sleep_for(4);
XBT_INFO("Done sleeping (duration: %.2f s). Current peak speed=%.0E flop/s; Energy dissipated=%.0f J",
         simgrid_get_clock() - start, sg_host_get_speed(host), sg_host_get_consumed_energy(host));
```

Ejemplo (exec-energy)

```
  // =========== Turn the other host off ==========
  XBT_INFO("Turning MyHost2 off, and sleeping another 10 seconds. MyHost2 dissipated %.0f J so far.",
           sg_host_get_consumed_energy(sg_host_by_name("MyHost2")));
  sg_host_turn_off(sg_host_by_name("MyHost2"));
  start = simgrid_get_clock();
  sg_actor_sleep_for(10);
  XBT_INFO("Done sleeping (duration: %.2f s). Current peak speed=%.0E flop/s; Energy dissipated=%.0f J",
           simgrid_get_clock() - start, sg_host_get_speed(host), sg_host_get_consumed_energy(host));
}

int main(int argc, char* argv[])
{
  simgrid_init(&argc, argv);
  sg_cfg_set_string("plugin", "host_energy");

  xbt_assert(argc == 2, "Usage: %s platform_file\n\tExample: %s platform.xml\n", argv[0], argv[0]);

  simgrid_load_platform(argv[1]);
  sg_actor_create("dvfs_test", sg_host_by_name("MyHost1"), &dvfs, 0, NULL);

  simgrid_run();

  XBT_INFO("End of simulation");

  return 0;
}
```

Salida

```
[MyHost1:dvfs_test:(1) 0.000000] [energy_exec/INFO] Energetic profile: 100.0:93.0:200.0, 93.0:90.0:170.0, 90.0:90.0:150.0
[MyHost1:dvfs_test:(1) 0.000000] [energy_exec/INFO] Initial peak speed=1E+08 flop/s; Energy dissipated =0E+00 J
[MyHost1:dvfs_test:(1) 0.000000] [energy_exec/INFO] Sleep for 10 seconds
[MyHost1:dvfs_test:(1) 10.000000] [energy_exec/INFO] Done sleeping (duration: 10.00 s). Current peak speed=1E+08; Energy dissipated=1000.00 J
[MyHost1:dvfs_test:(1) 10.000000] [energy_exec/INFO] Run a task of 1E+08 flops
[MyHost1:dvfs_test:(1) 11.000000] [energy_exec/INFO] Task done (duration: 1.00 s). Current peak speed=1E+08 flop/s; Current consumption: from 93W to 2
00W depending on load; Energy dissipated=1120 J
[MyHost1:dvfs_test:(1) 11.000000] [energy_exec/INFO] ========= Requesting pstate 2 (speed should be of 2E+07 flop/s and is of 2E+07 flop/s)
[MyHost1:dvfs_test:(1) 11.000000] [energy_exec/INFO] Run a task of 1E+08 flops
[MyHost1:dvfs_test:(1) 16.000000] [energy_exec/INFO] Task done (duration: 5.00 s). Current peak speed=2E+07 flop/s; Energy dissipated=1645 J
[MyHost1:dvfs_test:(1) 16.000000] [energy_exec/INFO] Sleep for 4 seconds
[MyHost1:dvfs_test:(1) 20.000000] [energy_exec/INFO] Done sleeping (duration: 4.00 s). Current peak speed=2E+07 flop/s; Energy dissipated=2005 J
[MyHost1:dvfs_test:(1) 20.000000] [energy_exec/INFO] Turning MyHost2 off, and sleeping another 10 seconds. MyHost2 dissipated 2000 J so far.
[MyHost1:dvfs_test:(1) 30.000000] [energy_exec/INFO] Done sleeping (duration: 10.00 s). Current peak speed=2E+07 flop/s; Energy dissipated=2905 J
[30.000000] [host_energy/INFO] Total energy consumption: 5004.750000 Joules (used hosts: 2904.750000 Joules; unused/idle hosts: 2100.000000)
[30.000000] [energy_exec/INFO] End of simulation
[30.000000] [host_energy/INFO] Energy consumption of host MyHost1: 2904.750000 Joules
[30.000000] [host_energy/INFO] Energy consumption of host MyHost2: 2100.000000 Joules
```

SMPI

- Permite la ejecución de aplicaciones MPI en SimGrid
  - Permite la experimentación reproducible de código MPI
  - Probar código MPI sobre una plataforma simulada

Ejemplo

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

#define N (1024 * 1024 * 1)

int main(int argc, char* argv[])
{
  int size, rank;
  struct timeval start, end;
  char hostname[256];
  int hostname_len;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Get_processor_name(hostname, &hostname_len);

  // Allocate a 1 MiB buffer
  char* buffer = malloc(sizeof(char) * N);
```

Ejemplo

```
  char* buffer = malloc(sizeof(char) * N); // Allocate a 1 MiB buffer

  // Communicate along the ring
  if (rank == 0) {
    gettimeofday(&start, NULL);
    printf("Rank %d (running on '%s'): sending rank %d\n", rank, hostname, 1);
    MPI_Send(buffer, N, MPI_BYTE, 1, 1, MPI_COMM_WORLD);
    MPI_Recv(buffer, N, MPI_BYTE, size - 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Rank %d (running on '%s'): received from rank %d\n", rank, hostname, size - 1);
    gettimeofday(&end, NULL);
    printf("%f\n", (end.tv_sec * 1000000.0 + end.tv_usec - start.tv_sec * 1000000.0 -
                      start.tv_usec) / 1000000.0);

  } else {
    MPI_Recv(buffer, N, MPI_BYTE, rank - 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Rank %d (running on '%s'): receive the message and sending it to rank %d\n",
               rank, hostname, (rank + 1) % size);
    MPI_Send(buffer, N, MPI_BYTE, (rank + 1) % size, 1, MPI_COMM_WORLD);
  }
  MPI_Finalize();
  return 0;
}
```

Compilando y ejecutando el ejemplo

```
smpicc  roundtrip.c -o roundtrip

smpirun -np 16 -platform cluster_crossbar.xml
                       -hostfile  cluster_hostfile.txt
                           ./roundtrip
```

Hostfile y plataforma

cluster_hostfile.txt

```
node-0.simgrid.org
node-1.simgrid.org
node-2.simgrid.org
node-3.simgrid.org
```

cluster_crossbar.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">
  <!--              _________
                   |         |
                   | crossbar |
                   |_________|
                      / | \
                     /  |  \
                  l0 / l1|   \l2
                    /    |    \
                   /     |     \
            node-0   node-1   node-2 ...

    All hosts can communicate at full speed with no interference on
    the crossbar. Only the links of each hosts are limiting.
  -->
  <zone id="world" routing="Full">
    <cluster id="cluster-crossbar"
             prefix="node-" radical="0-65535" suffix=".simgrid.org"
             speed="1Gf" bw="125MBps" lat="50us"/>
  </zone>
</platform>
```

Multiplicación de matrices

https://gitlab.com/PRACE-4IP/CodeVault/raw/master/hpc_kernel_samples/dense_linear_algebra/gemm/mpi/src/gemm_mpi.cpp

```
#include "stdio.h"
#include "mpi.h"

const int size = 1000;

float a[size][size];
float b[size][size];
float c[size][size];

void multiply(int istart, int iend)
{
    for (int i = istart; i <= iend; ++i) {
        for (int j = 0; j < size; ++j) {
            for (int k = 0; k < size; ++k) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

Multiplicación de matrices

```
int main(int argc, char* argv[])
{
    int rank, nproc;
    int istart, iend;
    double start, end;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Barrier(MPI_COMM_WORLD);
    start = MPI_Wtime();

    if (rank == 0) {// Initialize buffers.
        for (int i = 0; i < size; ++i) {
            for (int j = 0; j < size; ++j) {
                a[i][j] = (float)i + j;
                b[i][j] = (float)i - j;
                c[i][j] = 0.0f;
            }
        }
    }
```

Multiplicación de matrices

```
int main(int argc, char* argv[])
{
    int rank, nproc;
    int istart, iend;
    double start, end;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Barrier(MPI_COMM_WORLD);
    start = MPI_Wtime();

    if (rank == 0) {// Initialize buffers.
        for (int i = 0; i < size; ++i) {
            for (int j = 0; j < size; ++j) {
                a[i][j] = (float)i + j;
                b[i][j] = (float)i - j;
                c[i][j] = 0.0f;
            }
        }
    }
```

Multiplicación de matrices

```
// Broadcast matrices to all workers.
   MPI_Bcast(a, size*size, MPI_FLOAT, 0,MPI_COMM_WORLD);
   MPI_Bcast(b, size*size, MPI_FLOAT, 0,MPI_COMM_WORLD);
   MPI_Bcast(c, size*size, MPI_FLOAT, 0,MPI_COMM_WORLD);

   // Partition work by i-for-loop.
   istart = (size / nproc) * rank;
   iend = (size / nproc) * (rank + 1) - 1;

   // Compute matrix multiplication in [istart,iend]
   // C <- C + A x B
   multiply(istart, iend);

   // Gather computed results.
   MPI_Gather(c + (size/nproc*rank),   size*size/nproc, MPI_FLOAT,
              c + (size/nproc*rank),   size*size/nproc, MPI_FLOAT,
              0, MPI_COMM_WORLD);
```

Multiplicación de matrices

```
if (rank == 0) {
      // Compute remaining multiplications   // when size % nproc > 0.
      if (size % nproc > 0) {
          multiply((size/nproc)*nproc, size-1);
      }
  }

  MPI_Barrier(MPI_COMM_WORLD);
  end = MPI_Wtime();

  MPI_Finalize();
```

Multiplicación de matrices

```
    if (rank == 0) { /* use time on master node */
    float msec_total = 0.0f;

    // Compute and print the performance
    float msec_per_matrix_mul = end-start;
    double flops_per_matrix_mul = 2.0 * (double)size * (double)size * (double)size;
    double giga_flops = (flops_per_matrix_mul * 1.0e-9f) / (msec_per_matrix_mul /
1000.0f);
    printf(
            "Performance= %.2f GFlop/s, Time= %.3f msec, Size= %.0f Ops\n",
            giga_flops,
             msec_per_matrix_mul,
             flops_per_matrix_mul);
     }

    return 0;
}
```

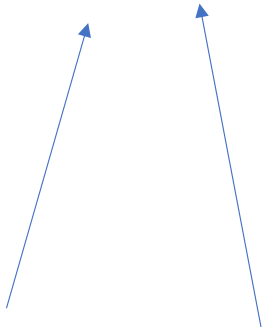
Multiplicación de matrices. Ejecución

simgrid-4/bin/smpirun -np 2 -hostfile ./cluster_hostfile.txt -platform ./cluster_crossbar.xml --cfg=smpi/display-timing:1 --cfg=smpi/host-speed:3167152404.785f ./mul

```
Performance= 2756.59 GFlop/s, Time= 5.804 msec, Size= 16000000000 Ops
[5.804758] [smpi_utils/INFO] Simulated time: 5.80476 seconds.

The simulation took 6.49218 seconds (after parsing and platform setup)
6.12215 seconds were actual computation of the application
[5.804758] [smpi_utils/INFO] More than 75% of the time was spent inside the application code.
You may want to use sampling functions or trace replay to reduce this.
```

Macros para acelerar la simulación

```
void multiply(int istart, int iend)
{
     //for (int i = istart; i <= iend; ++i) {
     SMPI_SAMPLE_GLOBAL(int i = istart, i <= iend, ++i, 10, 0.005) {
        for (int j = 0; j < size; ++j) {
            for (int k = 0; k < size; ++k) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```



se realizarán como máximo 10 iteraciones y se podrá salir de la fase de muestreo antes si se alcanza la estabilidad esperada tras un número menor de muestras.

Nueva ejecución

```
Performance= 2482.52 GFlop/s, Time= 6.445 msec, Size= 16000000000 Ops
[6.445544] [smpi_utils/INFO] Simulated time: 6.44554 seconds.

The simulation took 0.421239 seconds (after parsing and platform setup)
0.0154627 seconds were actual computation of the application
```

```
Performance= 2756.59 GFlop/s, Time= 5.804 msec, Size= 16000000000 Ops
[5.804758] [smpi_utils/INFO] Simulated time: 5.80476 seconds.

The simulation took 6.49218 seconds (after parsing and platform setup)
6.12215 seconds were actual computation of the application
[5.804758] [smpi_utils/INFO] More than 75% of the time was spent inside the application code.
You may want to use sampling functions or trace replay to reduce this.
```

Gestión de array dinámicos en SimGrid

- *Dynars* son vectores de tamaño dinámico que pueden contener cualquier tipo de datos
  - Tipo de datos: `xbt dynar t`

Algunas funciones

- `my dynar = xbt dynar new (elm size, free method)`
- `xbt dynar free container (&my dynar)`
  - Libera el array pero no el contenido
- `xbt dynar push (my dynar, &element)`
- `xbt dynar pop(my dynar, &element)`
  - Extrae el último elemento
- `void xbt_dynar_shift(my dynar, &element)`
  - Extrae el primer elemento
- `xbt dynar foreach(my dynar, ctr, element)`
  - Permite iterar sobre los elementos
- `xbt dynar length(my dynar)`
- `xbt dynar is empty(my dynar)`

Uso de algunas funciones

- Dada una petición:

```
struct ClientRequest {
        int    n_task;
        double t_arrival;   /* momento en el que llega */
        double t_service;   /* tiempo de servicio asignado */
        int    priority;
  . . .
};
```

- Una cola de peticiones vacía se crea:
  - `xbt_dynar_t client_requests ;   // cola de peticiones`
  - `client_requests = xbt_dynar_new(sizeof(struct ClientRequest), NULL);`
- Una nueva petición se inserta:
  - `xbt_dynar_push(client_requests, (char *)&req);`
- El primer elemento se extrae:
  - `xbt_dynar_shift(client_requests, (char *)&req);`
- El último elemento se extrae:
  - `xbt_dynar_pop(client_requests, (char *)&req);`

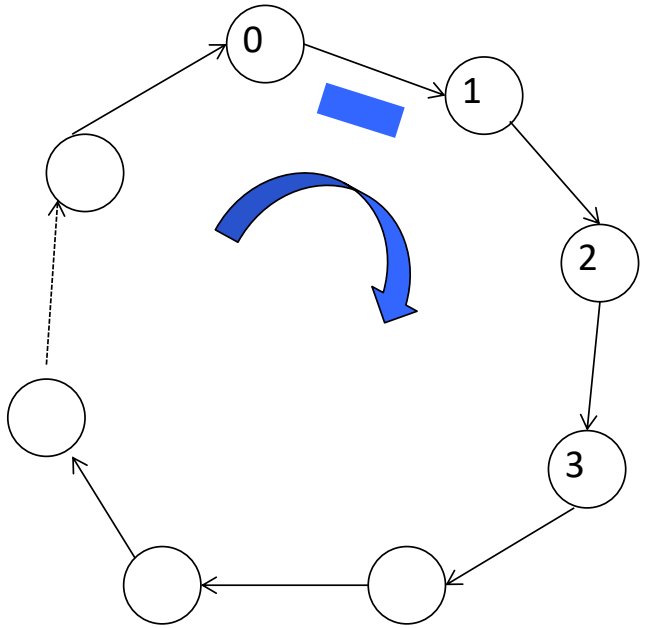
Uso de algunas funciones

- Ordenar la cola por tiempo de servicio
  - **xbt_dynar_sort**(client_requests, &sort_function);
- Donde:

```
static int sort_function(const void *e1, const void *e2)
{

        struct ClientRequest *c1 = *(void **) e1;
        struct ClientRequest *c2 = *(void **) e2;

        if (c1->t_service == c2->t_service)
                return 0;

        else    if (c1->t_service < c2->t_service)
                        return -1;
                else
                        return 1;
}
```

Escalabilidad

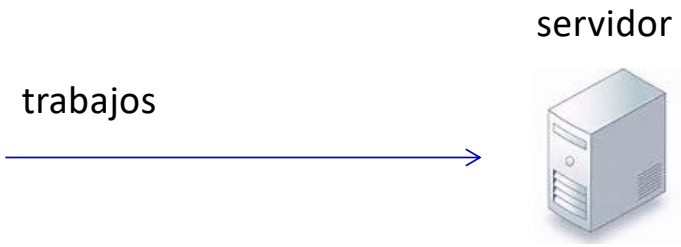
- 500 000 procesos conectados por una red de 1 Gbps
  - Paquete de 10000 bytes
  - Tiempo simulado: 2h 20 min.
  - Tiempo de simulación: 66 s.
  - Memoria: 8 GB
- 1 000 000 de procesos:
  - Tiempo simulado: 3h
  - Tiempo de simulación: 95 s
  - Memoria: 12 GB



Ejemplos de uso de SimGrid

- S. Alonso-Monsalve, F. Garcia-Carballeira, and A. Calderon, “ComBos: A complete simulator of Volunteer Computing and Desktop Grids,” *Simulation Modelling Practice and Theory*, vol. 77, pp. 197-211, 201
- S. Alonso-Monsalve, F. Garcia-Carballeira, and A. Calderon, “Analyzing the Performance of Volunteer Computing for Data Intensive Applications,” in The 2016 International Conference on High Performance Computing & Simulation (HPCS 2016). The 14th Annual Meeting, 2016.
- Alonso-Monsalve, F. Garca-Carballeira, and A. Calderón, “A heterogeneous mobile cloud computing model for hybrid clouds,” Future generation computer systems, 2018
- S. Alonso-Monsalve, F. Garcia-Carballeira, and A. Calderon, “Fog Computing Through Public-Resource Computing and Storage,” in 2nd International Conference on Fog and Edge Mobile Computing (FMEC2017), 2017
- Elias Del-Pozo, Felix Garcia-Carballeir. ”A Proposal of Mobility Support for the SimGridToolkit: Application to IoT simulations”. PDP 2022. 30th Euromicro International Conference on Pararallel, distributed, and Network based Processing. 9-1 marzo, 2022,
- Elias Del-Pozo, Felix Garcia-Carballeira. “A generic simulator for Edge Computing platforms”. The 2016 International Conference on High Performance Computing & Simulation (HPCS 2020), 22 - 27 March 2021

Ejemplo de análisis



- Al servidor llegan 5 tareas por hora. El tiempo entre llegadas sigue una distribución exponencial
  - Distribución exponencial de parámetro $\lambda=5$
  - Tiempo medio entre llegadas = $1/\lambda = 0.2$ h = 12 minutos
- El tiempo medio de ejecución de cada trabajo sigue una distribución exponencial
  - Tiempo medio = $1/\mu = 0.1666$ h = 10 minutos
  - Distribución exponencial de parámetro $\mu=6$ (6 trabajos por hora)
  - El servidor ejecuta las tareas según política FCFS sin expulsión
- ¿Cuál es el tiempo medio de ejecución de las tareas?

Simulación en Simgrid

```
int client(int argc, char *argv[]);
int server(int argc, char *argv[]);
#define N_TASKS  10000

double uniform(void)   {
        return drand48();
}
double uniform_pos(void) {
        double g;

        g = uniform();
        while (g == 0.0)
                g = uniform();

        return g;
}
```

Simulación en Simgrid

```
double exponential(double landa) {
   /* función de distribución con media 1/landa. */

    double u = uniform_pos();
    double mean = 1.0 / landa;

    return -mean * log(1-u);
}
```

$$x = -\frac{1}{\lambda}\ln(1-u)$$

Código cliente (genera las peticiones)

```
int client(int argc, char *argv[])  {
        int number_of_tasks = N_TASKS;
        struct request *req;
        double t_arrival;

        for (int i = 0; i < number_of_tasks; i++) {
                /* espera la llegada de una peticion */
                /* ARRIVAL_RATE peticiones por segundo, lamda = ARRIVAL_RATE */
                t_arrival = exponential((double) ARRIVAL_RATE);
                sg_actor_sleep_for(t_arrival);

                /* crea la peticion */
                req = (struct request *) malloc(sizeof(struct request));
                req->t_arrival = simgrid_get_clock();                  // tiempo de llegada
                req->t_service = exponential((double) SERVICE_RATE);   // tiempo de servicio
                req->n_req = i;

                /* envia la tarea a la cola */
                sg_mutex_lock(mutex);
                Nqueue++;
                sg_mutex_unlock(mutex);

                sg_mailbox_t mailbox = sg_mailbox_by_name("ServerHost");
                sg_mailbox_put_async(mailbox, req, 0);
        }
```

Código cliente (genera las peticiones)

```
        /* finalizar */
        req = (struct request *) malloc(sizeof(struct request));
        req->n_req = -1;
        sg_mailbox_t mailbox = sg_mailbox_by_name("ServerHost");
        sg_mailbox_put(mailbox, req, 0);

        return 0;
}       /* end_of_client */
```

Código servidor (atiende peticiones)

```
int server(int argc, char *argv[]) {
        int res;
        int NqueueAvg = 0;
        int Ntasks = 0;
        double timeServiceAvg = 0.0;
        double c;
        struct request *req;


        sg_mailbox_t mailbox = sg_mailbox_by_name("ServerHost");

        while (1) {
                req = (struct ClientRequest *) sg_mailbox_get(mailbox);

                if (req->n_req == -1){
                        free(req);
                        break;
                }
```

Código servidor (atiende peticiones)

```
        Ntasks++;

        // elimina un elemento de la cola
        sg_mutex_lock(mutex);
        Nqueue--;
        NqueueAvg = NqueueAvg + Nqueue;
        sg_mutex_unlock(mutex);

        // ejecuta la tara, tiempo de servicio de la cola


        sg_actor_sleep_for(req->t_service);

        c = simgrid_get_clock();

        timeServiceAvg = timeServiceAvg + (c - (req->t_arrival));


        free(req);
}
```

Código servidor (atiende peticiones)

```
        printf("-------------------------------------- \n");
        printf("Tareas ejecutadas  = %d\n",    Ntasks);
        printf("Tamanio medio de la cola = %g\n",   (double) NqueueAvg / Ntasks);
        printf("Tiempo medio de servicio %g\n", (double) timeServiceAvg / Ntasks);
        printf("-------------------------------------- \n");

        printf("I'm done. See you!");

        return 0;
}                               /* end_of_server */
```

main()

```
int main(int argc, char *argv[])
{

        srand48((int) time(NULL));

        if (argc < 3) {
                printf("Usage: %s platform_file deployment_file\n", argv[0]);
                exit(1);
        }

        simgrid_init(&argc, argv);

        mutex = sg_mutex_init();

        simgrid_load_platform(argv[1]);

        simgrid_register_function("client", client);
        simgrid_register_function("server", server);

        simgrid_load_deployment(argv[2]);

        simgrid_run();

        printf("Simulation time %g\n", simgrid_get_clock());

        return 0;
}
```

Platform.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">

 <zone  id="AS0"  routing="Full">

   <host id="ClientHost" speed="1000000f"/>
   <host id="ServerHost" speed="1000000f"/>

   <link id="1" bandwidth="100000Gbps" latency="0.0us"/>

   <route src="ClientHost" dst="ServerHost">

     <link_ctn id="1"/>
   </route>
 </zone>
 </platform>
```

Deployment.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "https://simgrid.org/simgrid.dtd">
<platform version="4.1">

  <process host="ClientHost" function="client"/>
  <process host="ServerHost" function="server"/>
</platform>
```

Resultados del ejemplo para diferentes semillas

Tiempo medio de respuesta (horas)	Nº medio de tareas en cola
0,8960	3,6596
1,0300	4,4126
0,8780	3,5004
1,1649	5,0218
0,9697	4,0181
1,0380	4,3599
1,0219	4,1866
1,0173	4,2237

- En cada realización de la simulación se obtiene un valor
- ¿Cómo estimar el valor?

Métricas de rendimiento

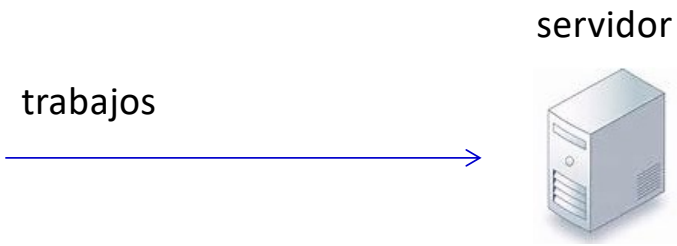
- Contador, número de veces que ocurre un determinado evento
- Duración de un determinado intervalo
- Tamaño de algún parámetro
- Un valor derivado de las anteriores medidas

Métricas más habituales

- Tiempo de respuesta:
  - Tiempo necesario para completar una operación
- *Throughput*:
  - Cantidad de trabajo completado por unidad de tiempo
    - Tareas ejecutadas por segundo, *frames* de video procesados por segundo, …
- Ancho de banda:
  - Cantidad de información transmitida por unidad de tiempo
    - Bits por segundo

Métrica de interés para el ejemplo

- Tiempo medio de ejecución de los trabajos en el servidor
- Número medio de procesos en la cola del servidor esperando para su ejecución
  - Tamaño medio de la cola del servidor



Resultados del ejemplo para diferentes semillas

Tiempo medio de respuesta (horas)	Nº medio de tareas en cola
0,8960	3,6596
1,0300	4,4126
0,8780	3,5004
1,1649	5,0218
0,9697	4,0181
1,0380	4,3599
1,0219	4,1866
1,0173	4,2237

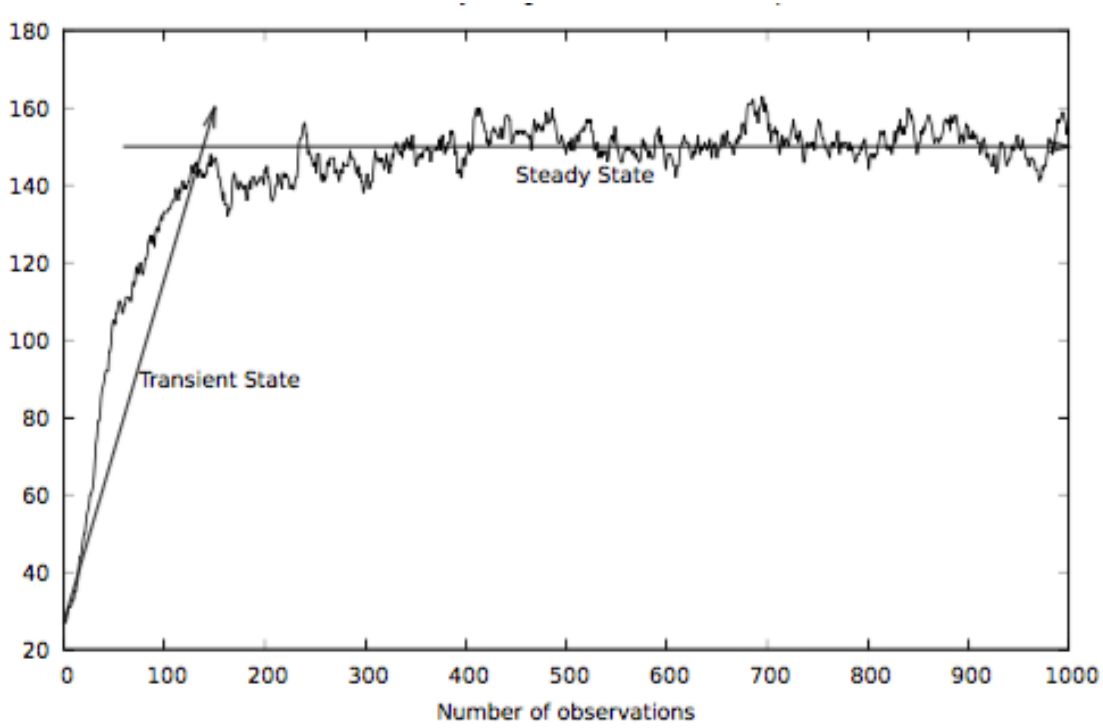
- En cada realización de la simulación se obtiene un valor
- ¿Cómo estimar el valor?

Experimentos de simulación

- Un experimento de simulación debe implicar la ejecución de la simulación varias veces con distintas semillas de números aleatorios
  - En cada realización se obtiene el valor de un parámetro de interés en distintos instantes de tiempo
  - Por ejemplo: T_i podría ser el tiempo medio de ejecución obtenido en la i-ésima realización del experimento
    - Cuando se utilizan variables aleatorias como entrada, las salidas T_1, T_2, … T_n son también variables aleatorias
      - T_1, T_2, … T_n son observaciones independientes
  - Se puede coger como valor de salida la media de las variables de salida de la simulación

$$\bar{T} = \frac{1}{n}\sum_{i=1}^{n} Ti$$

Determinar el periodo estable



Muestras obtenidas

- Si $\{x_1, x_2, \ldots x_n\}$ son los valores de salida que se obtienen en cada experimento (con semillas diferentes)
  - $\{x_1, x_2, \ldots x_n\}$ es un conjunto de variables aleatorias, independientes e idénticamente distribuidas de una función de distribución con media μ y desviación estándar s desconocidas
- La media y la desviación de la función de distribución de la variable de salida se pueden estimar como:

$$\bar{x} = \sum_{i=1}^{n} x_i \qquad s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}$$

- Según el teorema central del límite la variable aleatoria x sigue una distribución normal con media μ y desviación estándar $s/\sqrt{n}$

Resultados del ejemplo para diferentes semillas

Tiempo medio de respuesta (horas)	Nº medio de tareas en cola
0,8960	3,6596
1,0300	4,4126
0,8780	3,5004
1,1649	5,0218
0,9697	4,0181
1,0380	4,3599
1,0219	4,1866
1,0173	4,2237

- En cada realización de la simulación se obtiene un valor
- ¿Cómo estimar el valor?

Resultados del ejemplo para diferentes semillas

Tiempo medio de respuesta (horas)	Nº medio de tareas en cola
0,8960	3,6596
1,0300	4,4126
0,8780	3,5004
1,1649	5,0218
0,9697	4,0181
1,0380	4,3599
1,0219	4,1866
1,0173	4,2237

- En cada realización de la simulación se obtiene un valor
- ¿Cómo estimar el error cometido con un nivel de confianza del 90?

Error cometido en la obtención del valor medio

- ¿Cuál es el error que se comete al estimar el valor medio?
  - ¿Cuál es la probabilidad de que el error cometido en la estimación del valor medio sea menor de un determinado valor?
  - ¿Cuántas repeticiones del experimento debemos realizar para que el error cometido sea menor que un cierto valor con una determinada probabilidad?

Estimación del error

- Si$\{x_1, x_2, \ldots, x_n\}$ son las muestras obtenidas para calcular $\bar{x}$
- Si todas las muestras son independientes y proceden de una determinada distribución con media μ y desviación estándar s
  - La media $\bar{x}$ sigue una distribución normal con media μ y desviación estándar $\sigma/\sqrt{n}$

$$n = \text{número de medidas}$$

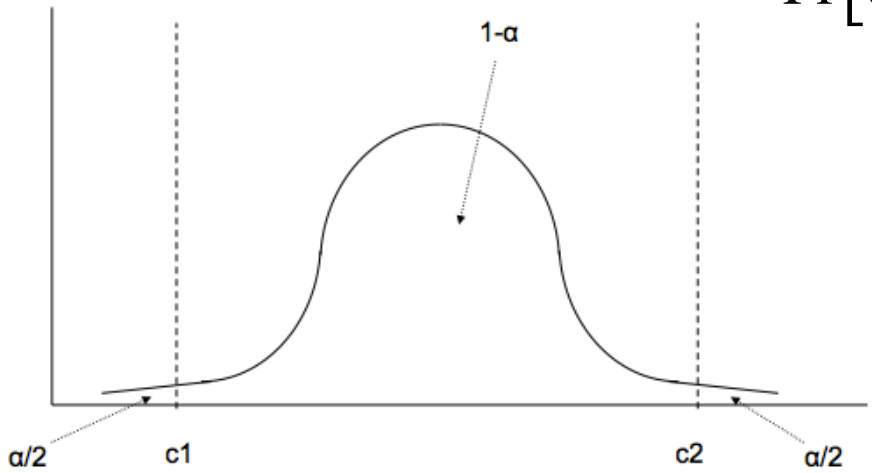
$$\bar{x} = \text{media} = \sum_{i=1}^{n} x_i$$

$$s = \text{desviación estándar} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

Modelo de errores

- Se emplean *intervalos de confianza del* α %para encontrar un rango de valores que, con una cierta probabilidad incluyen a valor real
  - Con una probabilidad de (1-α) el valor real se encuentra entre [c1,c2] encuentra

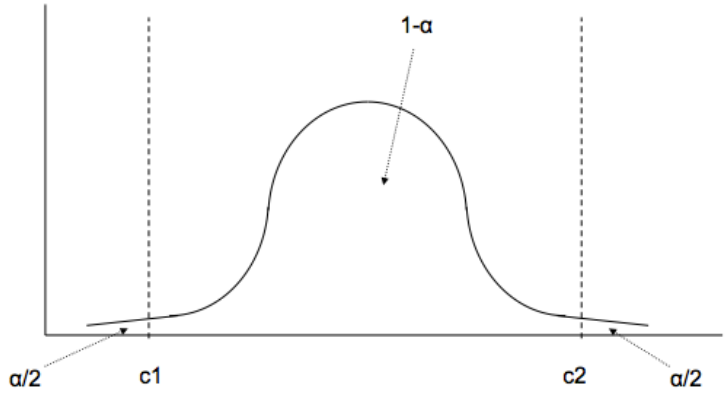
$$\Pr\left[c1 \le x \le c2\right] = 1 - \alpha$$



Intervalo de confianza
número de medidas es pequeño (n <30)

- La distribución $z = \dfrac{\overline{x} - x}{s / \sqrt{n}}$ sigue una distribución t de Student

  - Con (n-1) grados de libertad
  - Valores tabulados para t

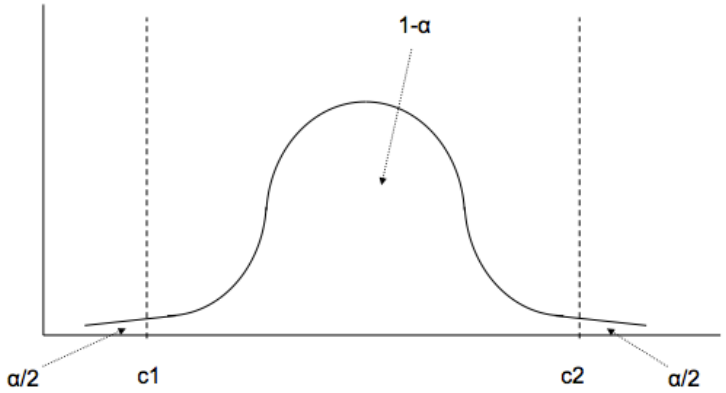


Intervalo de confianza (n < 30)

$$c_1 = \bar{x} - t_{\alpha/2;n-1}\frac{s}{\sqrt{n}}$$

$$c_2 = \bar{x} + t_{\alpha/2;n-1}\frac{s}{\sqrt{n}}$$

y $$\Pr(c_1 \le x \le c_2) = 1 - \alpha$$



	α/2		
GL	0,1	0,05	0,025
…	…	…	…
5	1.476	2.015	2.571
6	1.440	1.943	2.447
7	1.415	1.895	2.365
…	…	…	…
∞	1.282	1.645	1.960

Ejemplo

Tiempo medio de respuesta (horas)	Nº medio de tareas en cola
0,8960	3,6596
1,0300	4,4126
0,8780	3,5004
1,1649	5,0218
0,9697	4,0181
1,0380	4,3599
1,0219	4,1866
1,0173	4,2237

Ejemplo (para el tiempo medio de respuesta)

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} = 1{,}001975$$

$$s = \text{desviación estándar} = 0{,}0901$$

$$c_1 = \bar{x} - t_{\alpha/2;n-1}\frac{s}{\sqrt{n}}$$

$$c_2 = \bar{x} + t_{\alpha/2;n-1}\frac{s}{\sqrt{n}}$$

y $\Pr(c_1 \le x \le c_2) = 1 - \alpha$

- Para un intervalo de confianza del 90%, el valor real se encuentra dentro del intervalo de confianza con una probabilidad del 90%
- Para un intervalo de confianza del 90 %, 1-α=0,9
  - α/2 = 0.05
  - n=8, por tanto la distribución t de student tiene 7 grados de libertad

Intervalo de confianza del 90%

$$c_1 = \bar{x} - t_{\alpha/2;n-1}\frac{s}{\sqrt{n}}$$

$$c_2 = \bar{x} + t_{\alpha/2;n-1}\frac{s}{\sqrt{n}}$$

y $\Pr(c_1 \le x \le c_2) = 1-\alpha$

$$\alpha/2 = 0.95$$

$$t_{\alpha/2;n-1} = t_{0.05;7} = 1.895$$

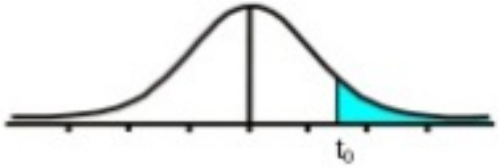
$$c_1 = 1.001975 - \frac{1.895 \cdot 0.09}{\sqrt{8}} = 0.9416$$

$$c_2 = 1.001975 + \frac{1.895 \cdot 0.09}{\sqrt{8}} = 1.0622$$

	α/2		
GL	0,1	0,05	0,025
…	…	…	…
5	1.476	2.015	2.571
6	1.440	1.943	2.447
7	1.415	1.895	2.365
…	…	…	…
∞	1.282	1.645	1.960

Tabla t de Student

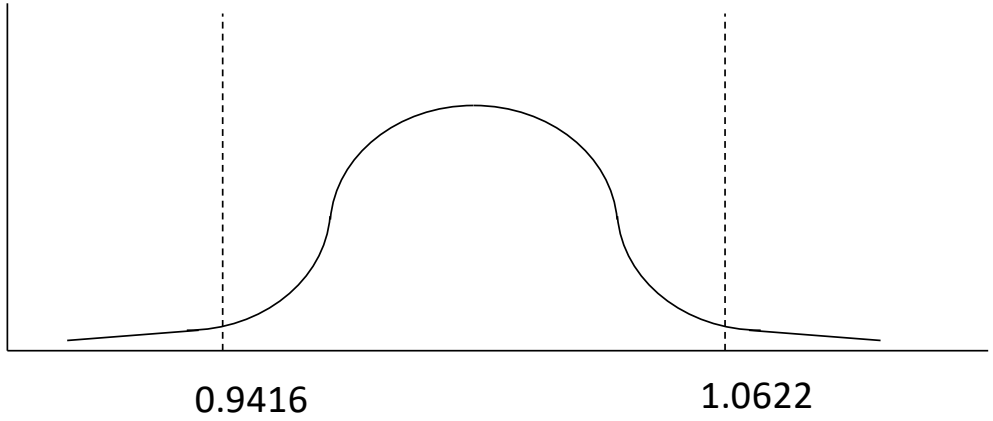
Tabla t-Student



Grados de libertad	0.25	0.1	0.05	0.025	0.01	0.005
1	1.0000	3.0777	6.3137	12.7062	31.8210	63.6559
2	0.8165	1.8856	2.9200	4.3027	6.9645	9.9250
3	0.7649	1.6377	2.3534	3.1824	4.5407	5.8408
4	0.7407	1.5332	2.1318	2.7765	3.7469	4.6041
5	0.7267	1.4759	2.0150	2.5706	3.3649	4.0321
6	0.7176	1.4398	1.9432	2.4469	3.1427	3.7074
7	0.7111	1.4149	1.8946	2.3646	2.9979	3.4995
8	0.7064	1.3968	1.8595	2.3060	2.8965	3.3554
9	0.7027	1.3830	1.8331	2.2622	2.8214	3.2498
10	0.6998	1.3722	1.8125	2.2281	2.7638	3.1693
11	0.6974	1.3634	1.7959	2.2010	2.7181	3.1058
12	0.6955	1.3562	1.7823	2.1788	2.6810	3.0545

Interpretación

- Intervalo de confianza del 90%= [0.9416, 1.0622]
  - El 90% de las veces el valor real se encuentra entre 0.9416 y 1.0622



Ejemplo (número medio de tareas en la cola)

- El número medio de tareas:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} = 4{,}17$$

$$s = \text{desviación estándar} = 0{,}471$$

- Intervalo de confianza del 90%= [3.854, 4.485]
  - El 90% de las veces el valor real se encuentra entre real value 3.854 y 4.485

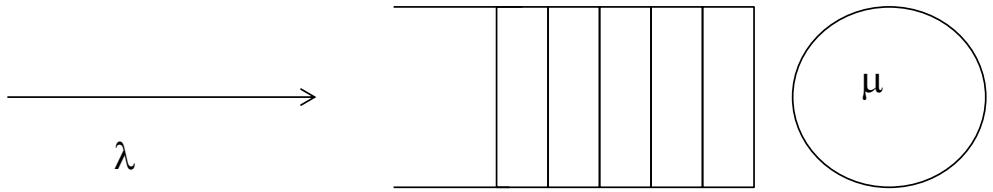
Error cometido

$$[c_1, c_2] = \left[\bar{x} - z_{\alpha/2}\frac{s}{\sqrt{n}}, \bar{x} + z_{\alpha/2}\frac{s}{\sqrt{n}}\right]$$
$$= \left[(1-e)\bar{x}, (1+e)\bar{x}\right]$$

- Intervalo de confianza del 90%= [3.854, 4.485]
  - Con una probabilidad de 0.9 el error cometido será de e = 0.076, es decir, del 7.6%

Solución analítica

- El sistema descrito se modela como una cola M/M/1

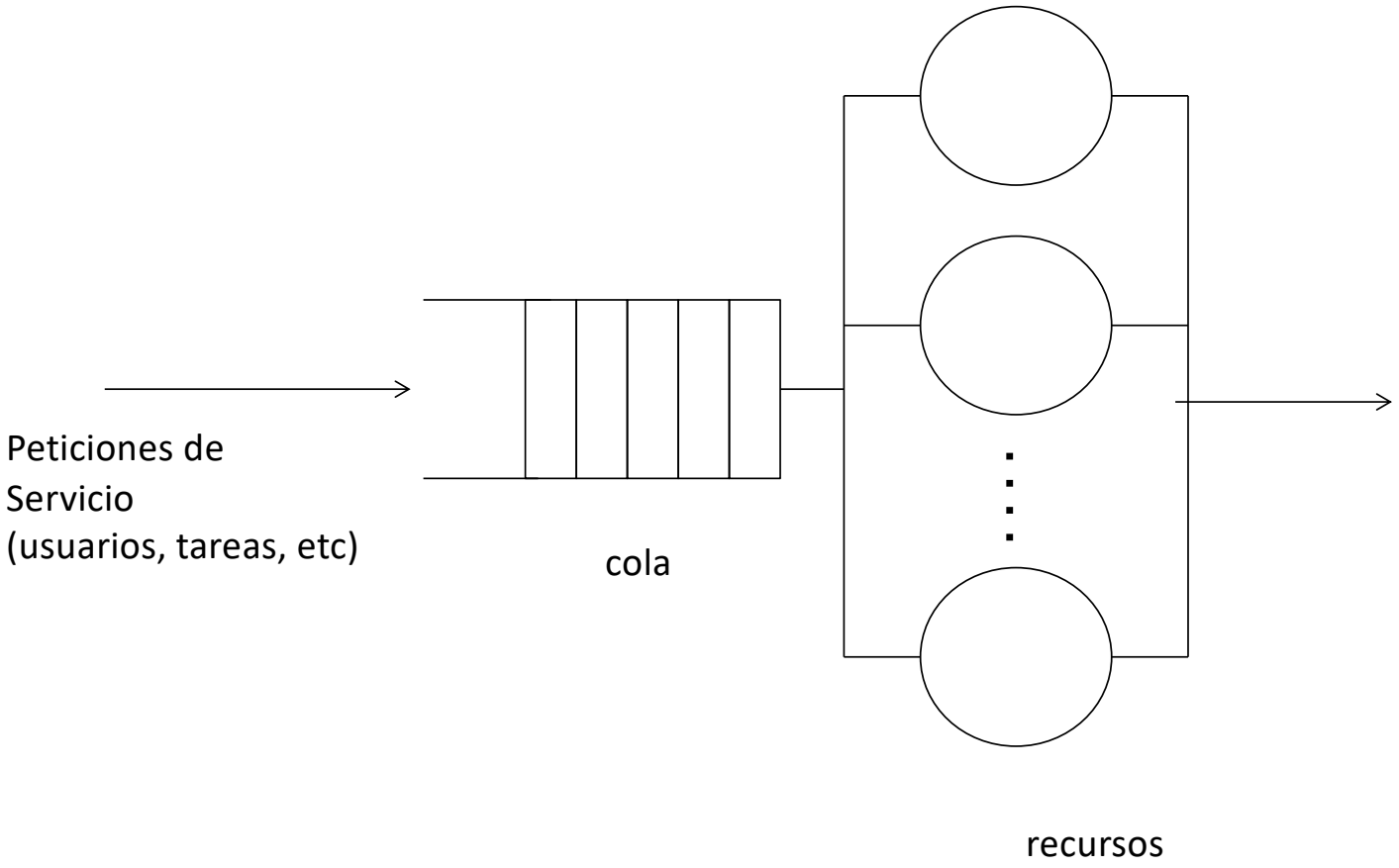


- Tiempo entre llegadas con distribución exponencial (proceso de Poisson) de media 1/ λ
- Tiempo de servicio con distribución exponencial de media 1/μ

Teoría de colas

- Estudia el comportamiento de sistemas donde existe un conjunto limitado de recursos para atender peticiones
- Requiere:
  - Modelar el sistema
  - Modelar el comportamiento aleatorio de las peticiones y tiempos de servicio
- Existen modelos de colas con solución analítica conocida
- Los modelos sin solución analítica conocida han de resolverse mediante técnicas de simulación

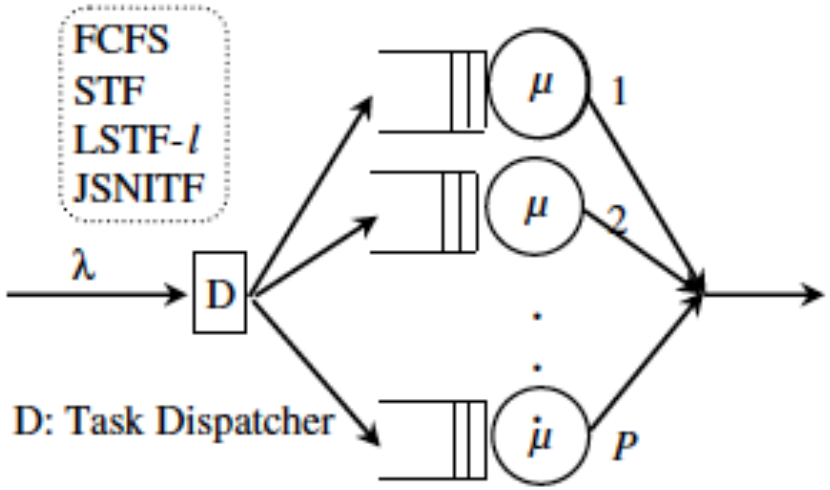
Modelo de un sistema de colas



Especificación de un sistema de colas

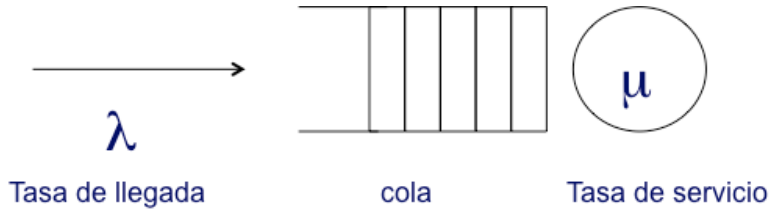
- **A / B / m / K / N / Z**
  - A: distribución de la variable aleatoria tiempo entre llegadas
  - B: distribución de la variable aleatoria tiempo de servicio
    - M: exponencial, U: uniforme, G: general (arbitraria)
  - m: número de recursos del sistema
  - K: capacidad del sistema (máximo número de tareas que pueden estar en el sistema al mismo tiempo)
  - N: tamaño de la población
  - Z: disciplina de la cola
    - FCFS, SJF, LCFS, RR, con expulsión, sin expulsión,…

Ejemplo



Solución analítica de una cola M/M/1

- Cola MM1

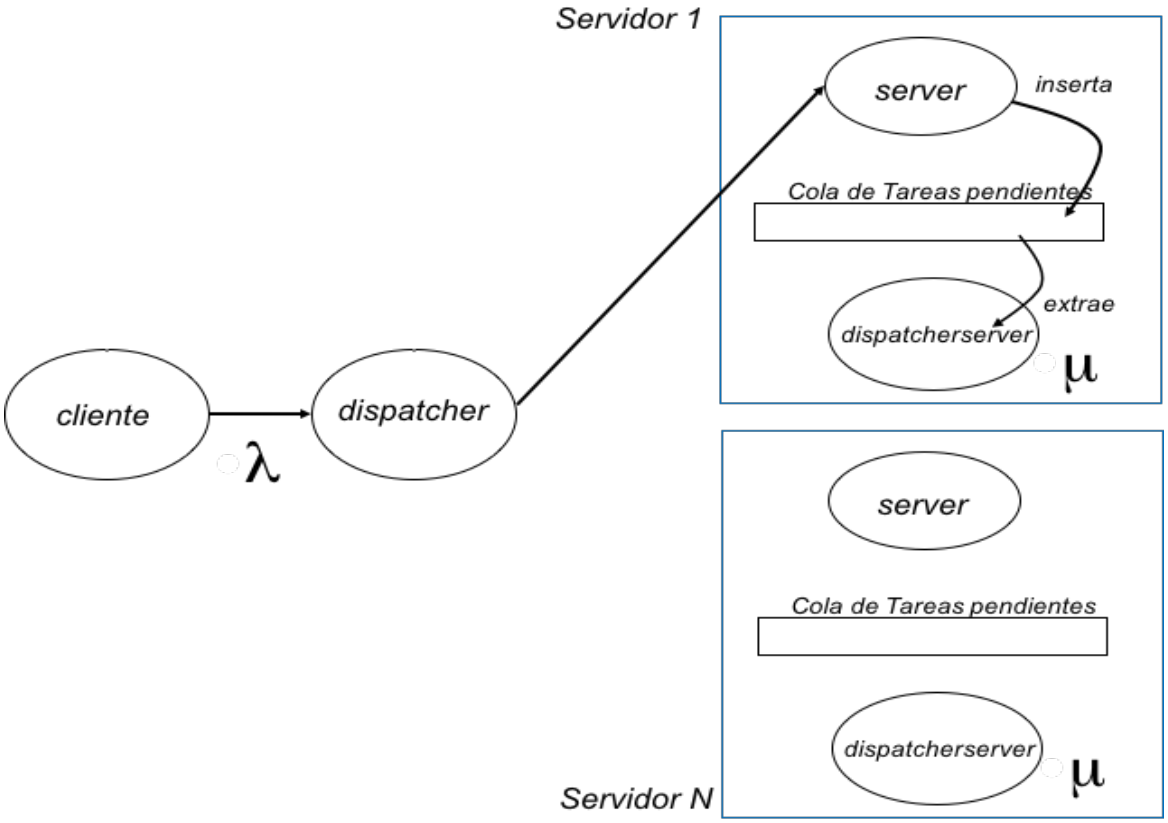


- Tiempo medio de servicio $\overline{T} = \frac{1}{\mu - \lambda}$
- Tamaño medio de la cola $\overline{Q} = \frac{\rho^2}{1-\rho} \quad con \quad \rho = \frac{\lambda}{\mu}$
- Tiempo medio de espera en la cola $\overline{W} = \frac{\rho}{\mu \cdot (1-\rho)} \quad con \quad \rho = \frac{\lambda}{\mu}$
- Número medio de tareas en el sistema $\overline{N} = \frac{\rho}{1-\rho} \quad con \quad \rho = \frac{\lambda}{\mu}$

Ejemplo

- Solución analítica:
  - Tiempo medio de servicio = $\bar{T} = \frac{1}{6-5} = 1\,h$
  - Tamaño medio de la cola = $\bar{Q} = \frac{(5/6)^2}{1-5/6} = 4{,}16$
- Solución obtenida mediante simulación:
  - Tiempo medio de servicio = 1.001975 h
  - Tamaño medio de la cola = 4,17

Trabajo de simulación



Objetivo 1

- 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.

Objetivo 2

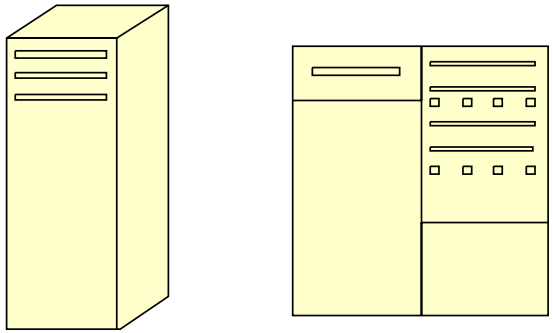
- Para cada uno los modelos anteriores se implementarán distintas políticas de planificación en cada servidor:
  - FCFS (First-Come, First-Sserver): las tareas a ejecutar en cada servidor se eligen en el orden en el que llegan.
  - STF (Shortest Job First): se elige para ejecución la tarea con menor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

Análisis

- 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%.

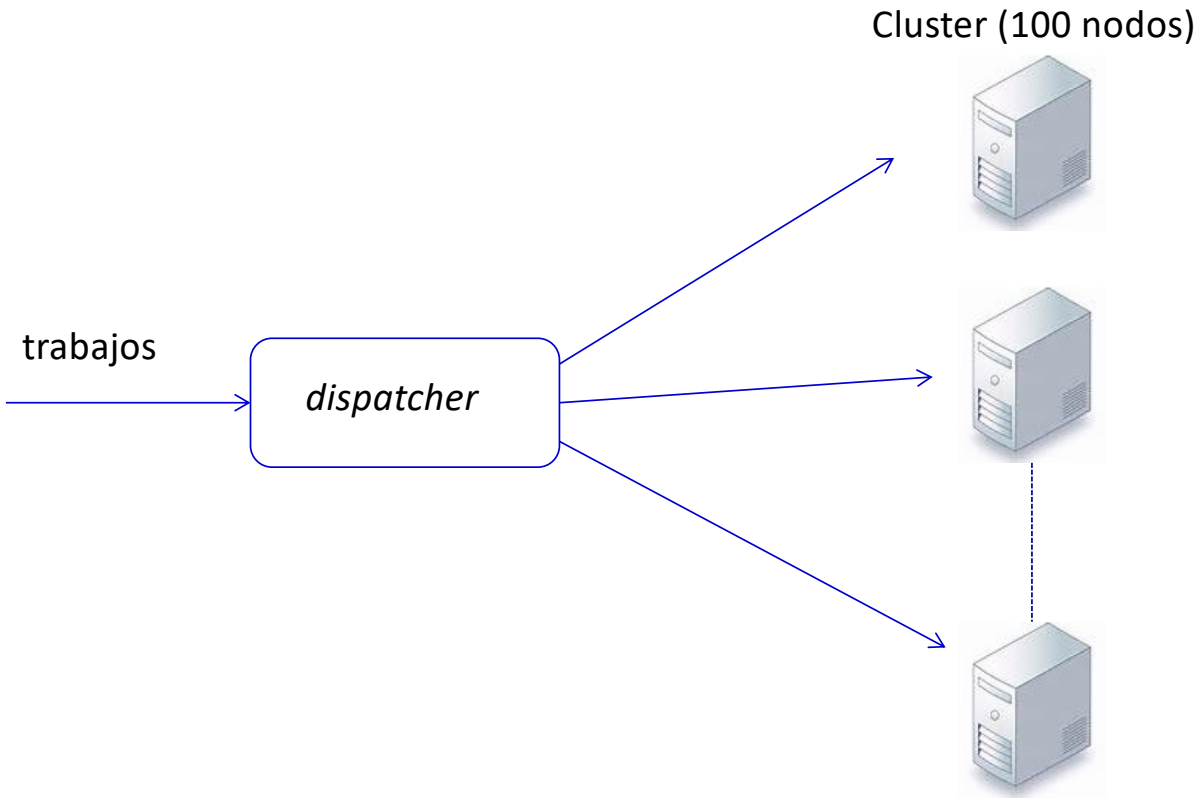
Comparación entre dos alternativas

- ¿Hay medidas significativas entre los valores obtenidos de dos sistemas diferentes?



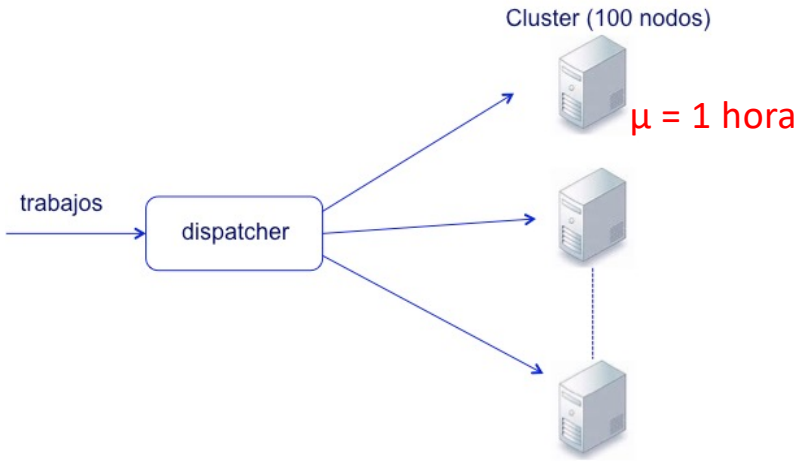
  - n_1 medidas del sistema 1
  - n_2 medidas del sistema 2

Ejemplo



Ejemplo

- Los trabajos a ejecutar tienen un tiempo de ejecución medio de 1 hora
- El tiempo de ejecución sigue una distribución exponencial de media $\mu = 1$ hora



Ejemplo

- Asumimos que cada nodo tiene una cola de procesos para ejecutar. Los procesos se ejecutan con política FCFS
- Estamos interesados en evaluar dos técnicas distintas de distribución de los trabajos:
  - *Shortest queue first*: cuando se recibe un trabajo se envía al nodo con la cola de procesos más pequeña
  - *The power of two random choice*: cuando se recibe un nuevo trabajo se eligen dos nodos de forma aleatoria y se envía al que tiene la cola de procesos más pequeña

Ejemplo

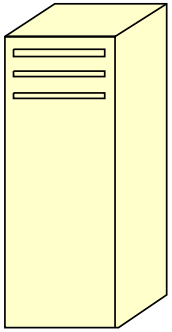
- Queremos saber el comportamiento de los dos algoritmos anteriores para una determinada tasa de llegada de trabajos
- Tasa de llegada de trabajos: sigue una distribución exponencial de tasa $\lambda = 90$ trabajos por hora
- Utilización de los servidores

$$\rho = \frac{\lambda}{n \cdot \mu} = 0.9 < 1$$

Ejemplo

- *Shortest queue first*
  - Tiempo medio: 1,067 h
  - Desviación: 0,267
- *The power of two random choice*
  - Tiempo medio: 2,6754 h
  - Desviación: 0,066
- ¿Hay diferencias significativas?

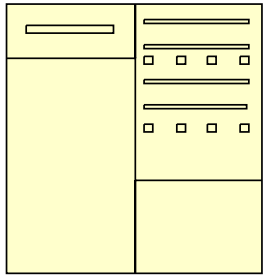
No siempre es obvio



$n_1 = 8$ medidas

$\overline{x}_1 = 1046,25$ s

$s_1 = 49,25$



$n_2 = 5$ medidas

$\overline{x}_2 = 996,6$ s

$s_2 = 78,41$

Intervalo de confianza para la diferencia de medias

- Se calculas las medias de las dos distribuciones
- Se calcula la diferencia de las medias
- Se calcula la desviación estándar de las diferencia de las medias
- Se determina el intervalo de confianza para la diferencia
- Si el intervalo incluye al 0: no hay diferencias significativas entre los dos sistemas
- Si el intervalo no incluye al 0: sí hay diferencias significativas entre los dos sistemas

Intervalo de confianza para la diferencia de medias

diferencia de las medias:

$$\overline{x} = \overline{x}_1 - \overline{x}_2$$

desviación estándar la las diferencias

$$s_x = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Si $n_1 > 30$ y $n_2 > 30$

- Se obtiene el intervalo de confianza para la diferencia de las medias

$$c_1 = \overline{x} - z_{1-\alpha/2} s_x$$

$$c_2 = \overline{x} + z_{1-\alpha/2} s_x$$

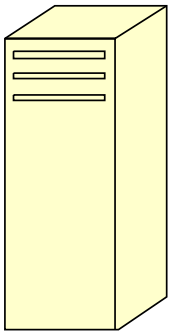
- Si el intervalo incluye al 0: no hay diferencias significativas entre los dos sistemas
- Si el intervalo no incluye al 0: sí hay diferencias significativas entre los dos sistemas

Si $n_1 < 30$ y $n_2 < 30$

- Se utiliza una t de Student con n_{df} grados de libertad

$$n_{df} = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(s_1^2/n_1\right)^2}{n_1 - 1} + \frac{\left(s_2^2/n_2\right)^2}{n_2 - 1}}$$

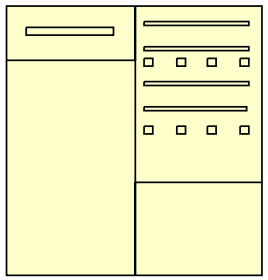
Para el ejemplo



$n_1 = 8$ medidas

$\overline{x}_1 = 1046,25$ s

$s_1 = 49,25$



$n_2 = 5$ medidas

$\overline{x}_2 = 996,6$ s

$s_2 = 78,41$

Ejemplo

$$\overline{x} = \overline{x}_1 - \overline{x}_2 = 1046,25 - 996 = 49,65$$

$$s_x = \sqrt{\frac{49,25^2}{8} + \frac{78,41^2}{5}} = 39,15$$

$$n_{df} = \frac{\left(\frac{49,25^2}{8} + \frac{78,41^2}{5}\right)^2}{\frac{\left(49,25^2/8\right)^2}{8-1} + \frac{\left(78,41^2/5\right)^2}{5-1}} = 6,01$$

Ejemplo

- Para un intervalo de confianza del 90%

$$c_1 = \overline{x} - t_{\alpha/2;n_{df}} s_x$$

$$c_2 = \overline{x} + t_{\alpha/2;n_{df}} s_x$$

$$t_{\alpha/2;n_{df}} = t_{0,05;6} = 1.943$$

$$c_1 = 49,65 - 1.943(39,15) = -26,42$$

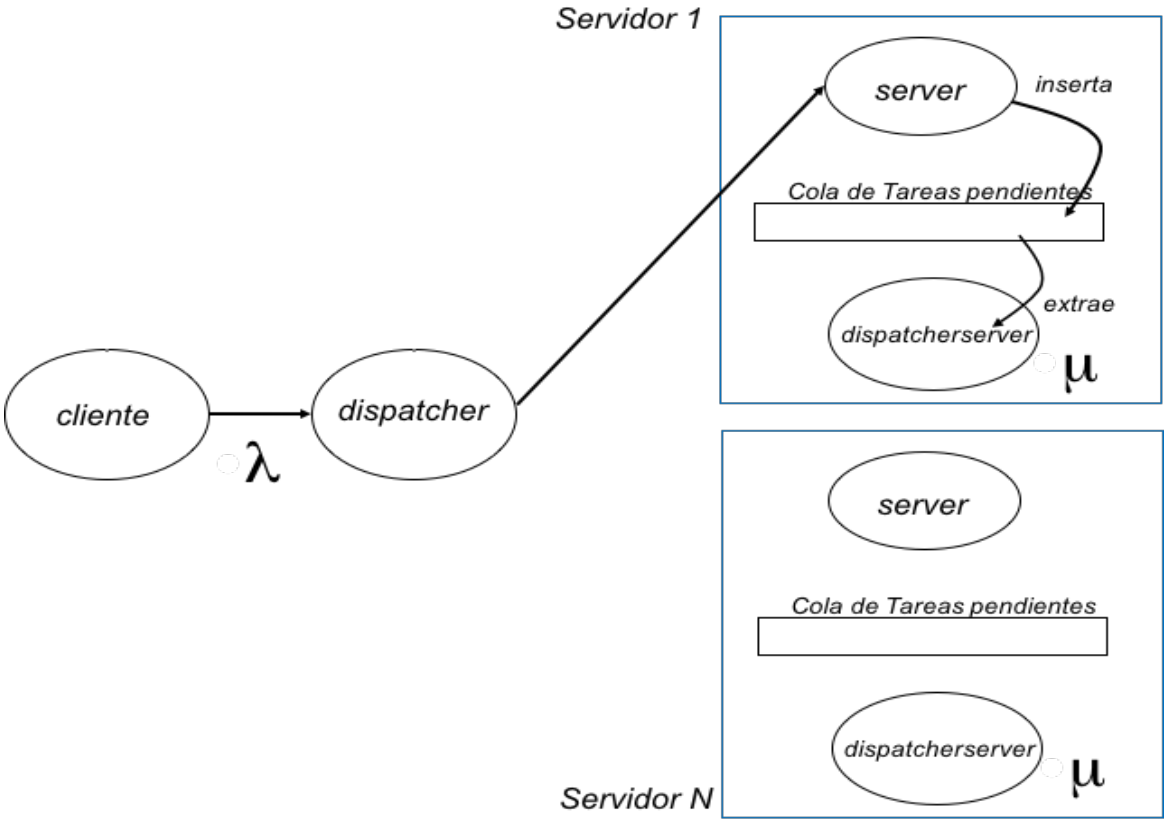
$$c_2 = 49,65 + 1.943(39,15) = 125,7$$

- Como el intervalo incluye al 0, estadísticamente, no hay diferencias significativas entre los dos sistemas

Aspectos a considerar en el análisis

- Tiempo de llegada de las peticiones al dispatcher
- Tiempo de servicio de las peticiones en el servidor
- Orden de ejecución dentro del servidor (FCFS, FSJ, …)
- Estos tiempo se suelen modelizar utilizando una variable de distribución exponencial:
  - En el tiempo de llegada:
    - Distribución exponencial de parámetro $\lambda=5$
    - Tiempo medio entre llegadas = $1/\lambda = 0.2$ h = 12 minutos
  - En el tiempo de salida:
    - Tiempo medio = $1/\mu = 0.1666$ h = 10 minutos
    - Distribución exponencial de parámetro $\mu=6$ (6 trabajos por hora)

Trabajo de simulación



Objetivo 1

- 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.

Objetivo 2

- Para cada uno los modelos anteriores 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.

Análisis

- 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%.