uc3m | Universidad Carlos III de Madrid

OpenCourseWare

Sistemas Paralelos y Distribuidos

Félix García Carballeira

Alejandro Calderón Matos

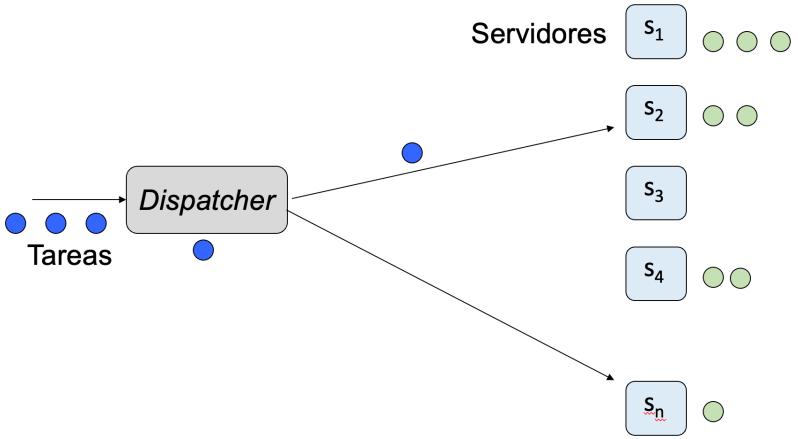
Tema 3. Planificación y distribución de carga en sistemas paralelos y distribuidos

Planificación y distribución de carga (*Scheduling and load balancing*)

- Conceptos muy relacionados entre sí
- En ambos casos se parte:
  - Un conjunto de procesadores (CE) y unidades de almacenamiento (SE) distribuidos
  - Un conjunto aplicaciones (trabajos) con requisitos de CPU y de almacenamiento.
    - Trabajos secuenciales
    - Trabajos paralelos

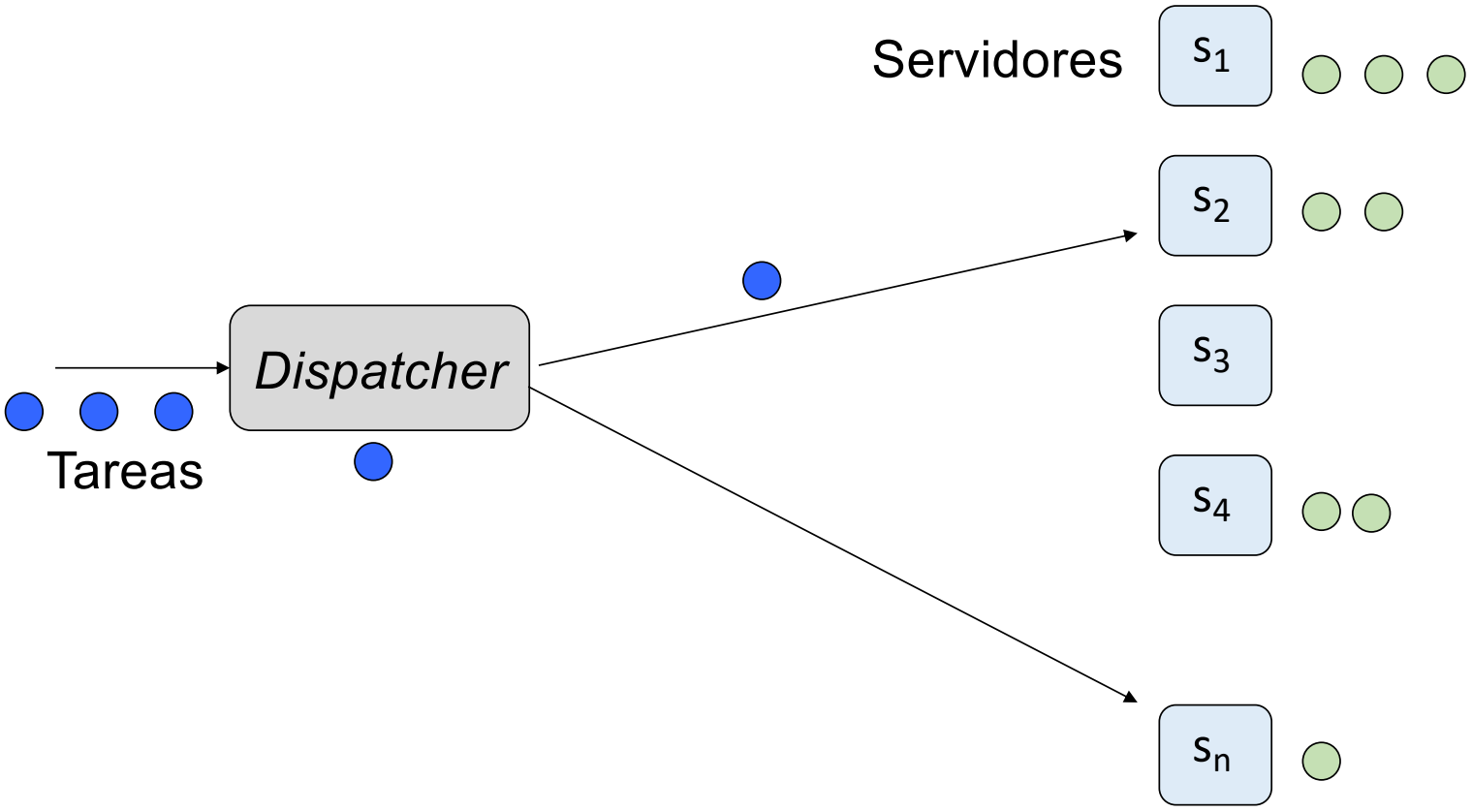
Planificación y distribución de carga

- El algoritmo de distribución de carga es responsable distribuir los trabajos según van llegando al sistema.



- El algoritmo de planificación es responsable determinar el orden en el que los trabajos que hay en un sistema se ejecutan en los procesadores

Planificación y distribución de carga



Planificación

- El algoritmo de planificación es responsable determinar el orden en el que los trabajos se ejecutan en los procesadores
- Un trabajo (*job*) está compuesto por un conjunto de tareas (*tasks*)
- Objetivo: buscar una estrategia que
    - Maximice la productividad (*throuhgput*)
    - Minimice el tiempo de respuesta
    - Maximice el uso de los recursos en global

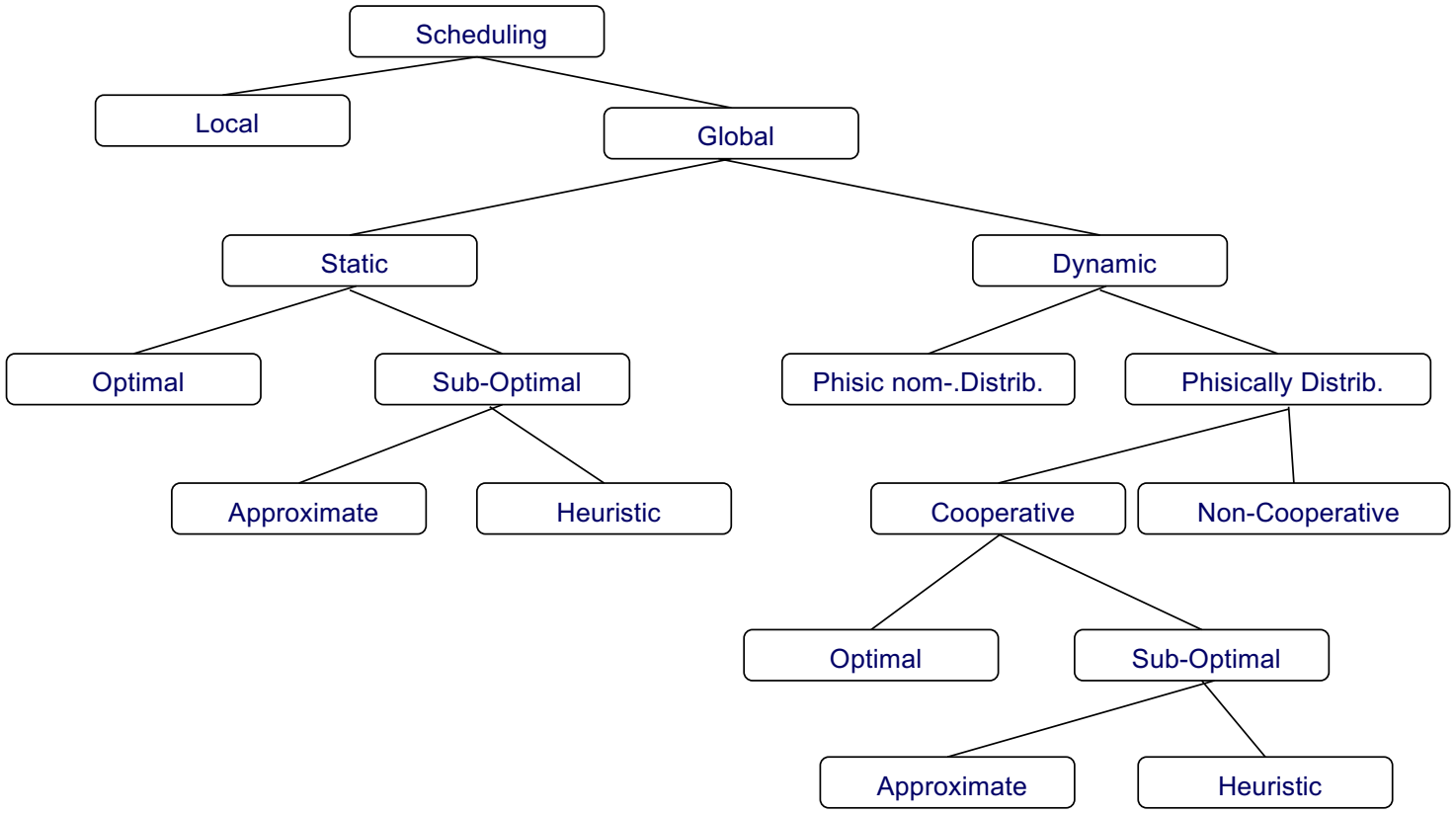
Diferentes modelos

- Un trabajo (*job*) está compuesto por un conjunto de tareas (*tasks*)
- Diferentes modelos:
  - Trabajos secuenciales (una tarea por trabajo)
  - Trabajos paralelos (varias tareas por trabajo):
    - Tareas independientes
    - Tareas que se comunican entre sí y pueden ejecutar en paralelo (planificación Gang)
    - Trabajo formado por tareas con restricciones de precedencia (planificación DAG)
    - Trabajos con restricciones de tiempo (*Real time scheduling*)

Aspectos a considerar en la planificación

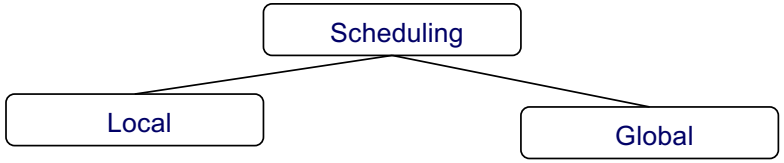
- Coste de las tareas:
  - ¿Tienen todas el mismo coste computacional?
  - ¿Se conocen los costes, cuando?
- Dependencia entre las tareas
  - ¿Pueden ejecutar en paralelo, en cualquier orden?
  - ¿Existen dependencias?
- Localidad:
  - ¿Es importante que algunas tareas se ejecuten en el mismo procesador para reducir las comunicaciones?
- Prioridad

Clasificación de los esquemas de planificación



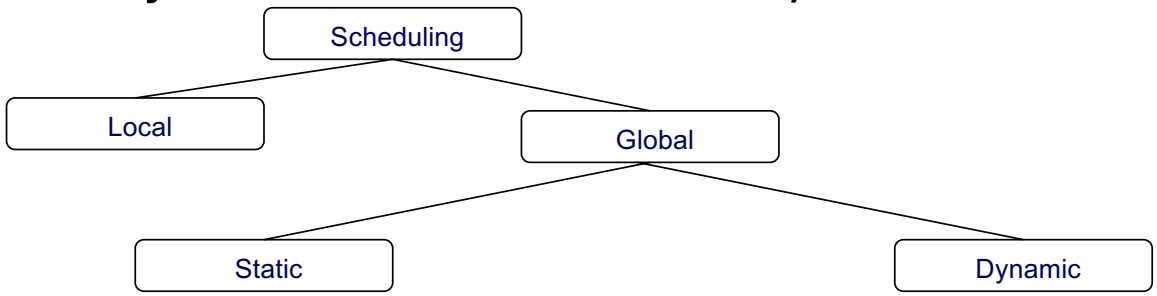
Clasificación

- Planificación local: planificación en un único procesador
- Planificación global: planificación en varios procesadores



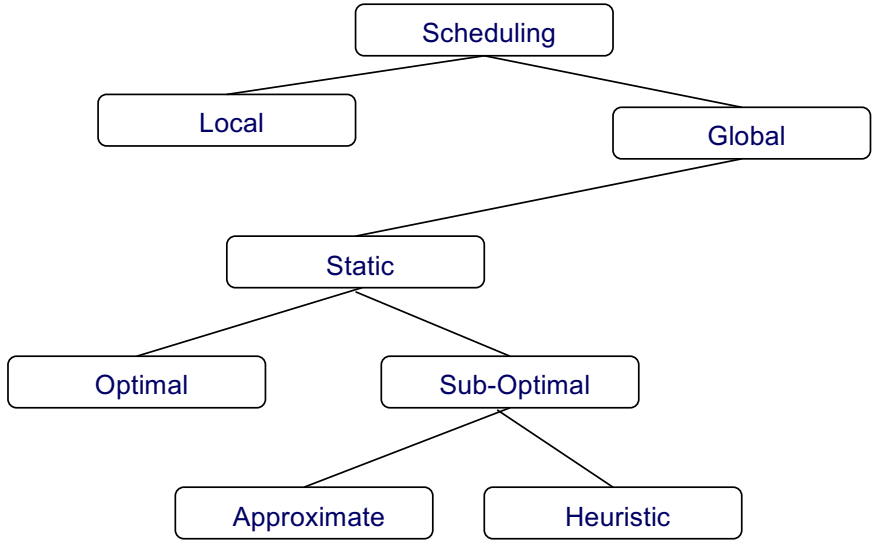
Planificación global

- Planificación estática: decisión tomada antes de la ejecución
- Planificación dinámica: planificación en tiempo de ejecución (adecuada para ejecución no determinista)



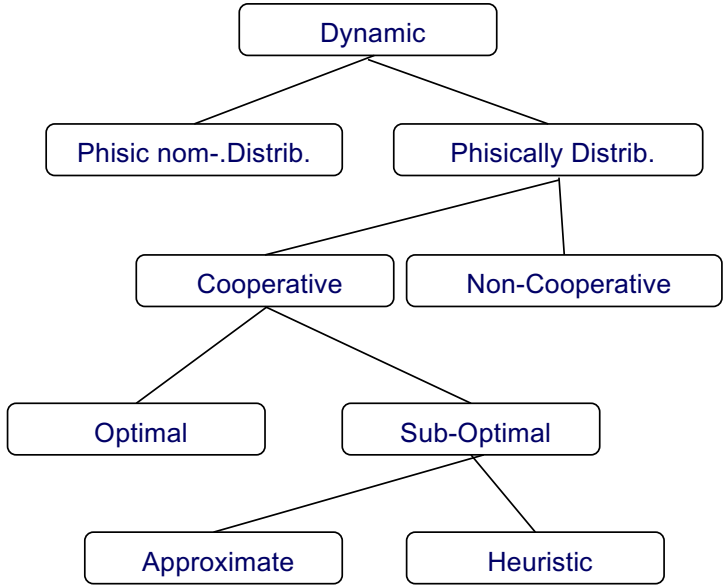
Planificación estática

- Solución óptima: Problema NP-Completo
- Soluciones sub-óptimas: basadas en heurísticas o métodos aproximados



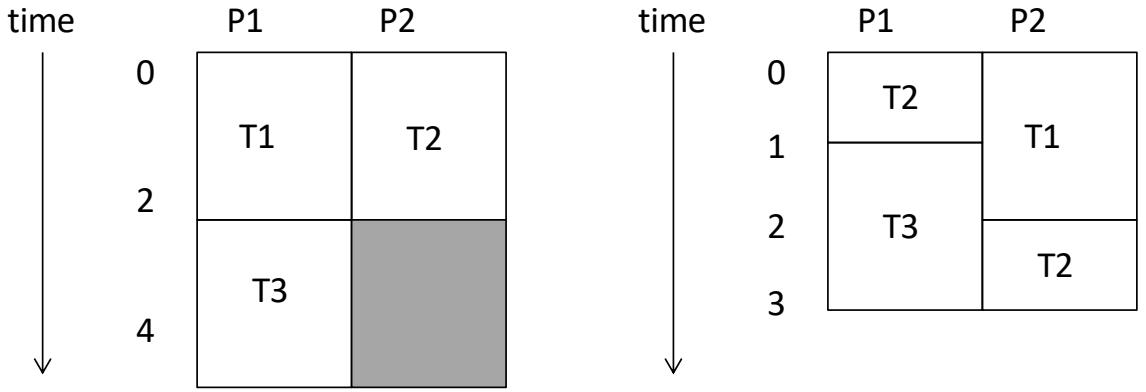
Planificación dinámica

- Físicamente no-distribuida: decisión realizada en un único procesador
- Físicamente distribuida: decisiones tomadas entre diferentes procesadores
  - Cooperativa entre diferentes planificadores locales
  - No Cooperativa



Esquemas expulsivos-No expulsivos

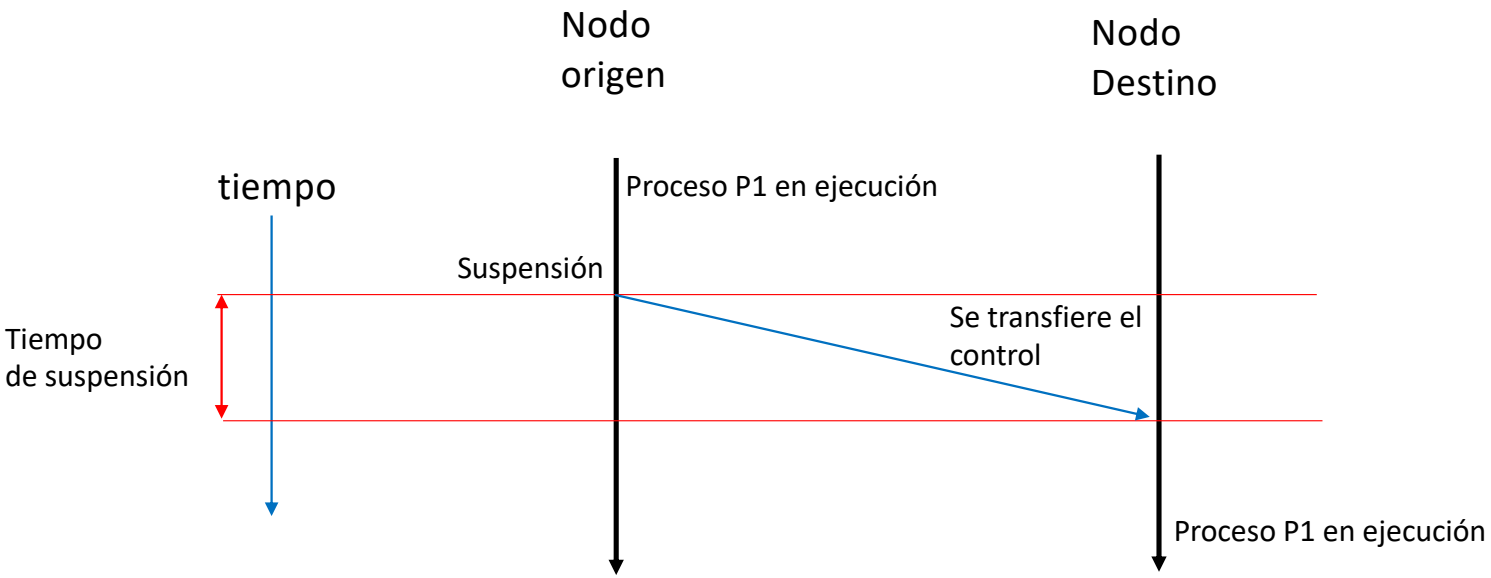
- Planificación no expulsiva (*nonpreemptive*): una tarea no puede ser interrumpida una vez comenzada su ejecución
- Planificación expulsiva (*preemptive*): una tarea puede ser expulsada de un procesador



Migración de procesos

- Migrar la ejecución de un proceso de un nodo a otro
  - Suspender la ejecución, almacenar estado (BCP)
  - Transferir el espacio de direcciones al nodo destino
  - Transferir estado
  - Reenviar los mensajes recibidos al proceso migrado

Mecanismo de migración



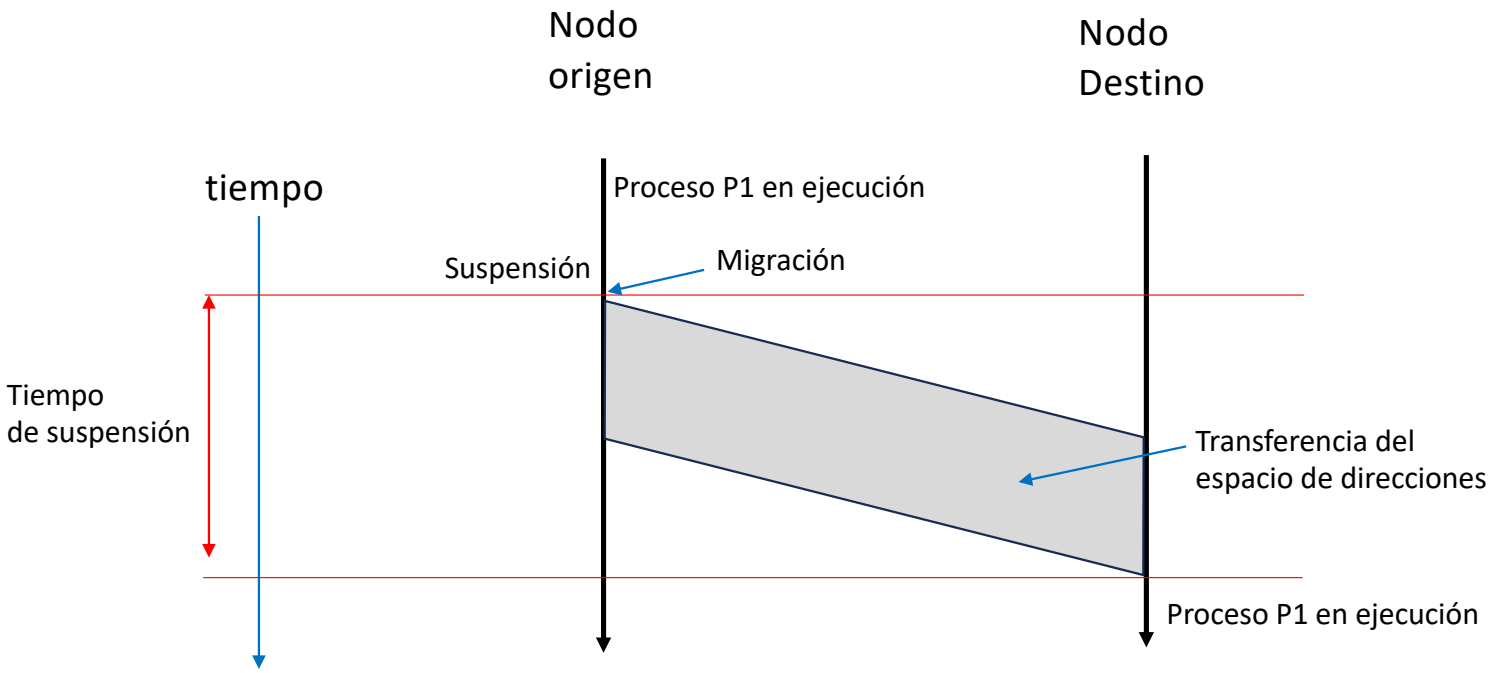
Suspensión de la ejecución

- Esperar la terminación de las operaciones de E/S pendientes
- Suspender la ejecución del proceso
- Registrar información sobre ficheros abiertos
- Crear un proceso vacío en el nodo destino
- Transferir el espacio de direcciones al nodo destino
- Transferir información del proceso al nodo destino
- Continuar la ejecución en el nodo destino

Mecanismos de suspensión y transferencia

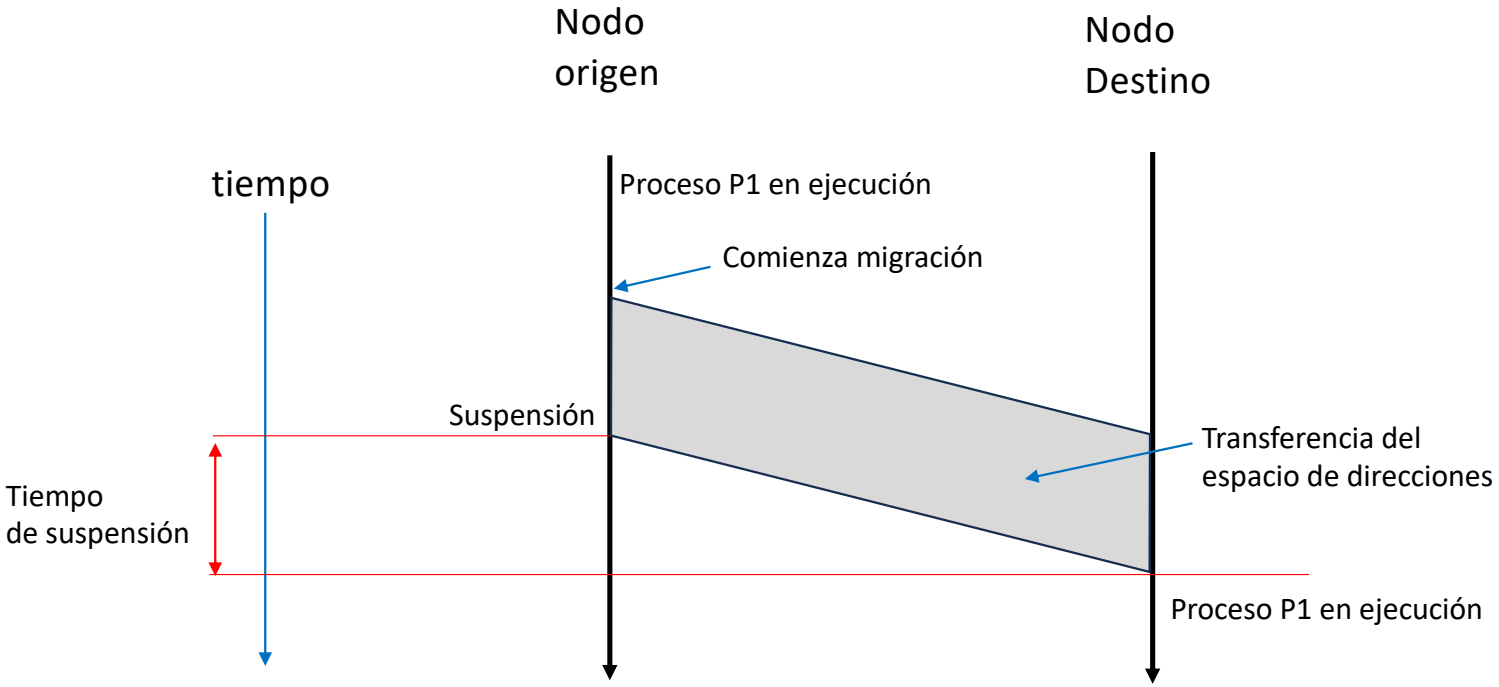
- Suspensión total del proceso
- Pre transferencia del proceso
- Transferencia bajo demanda

Suspensión total



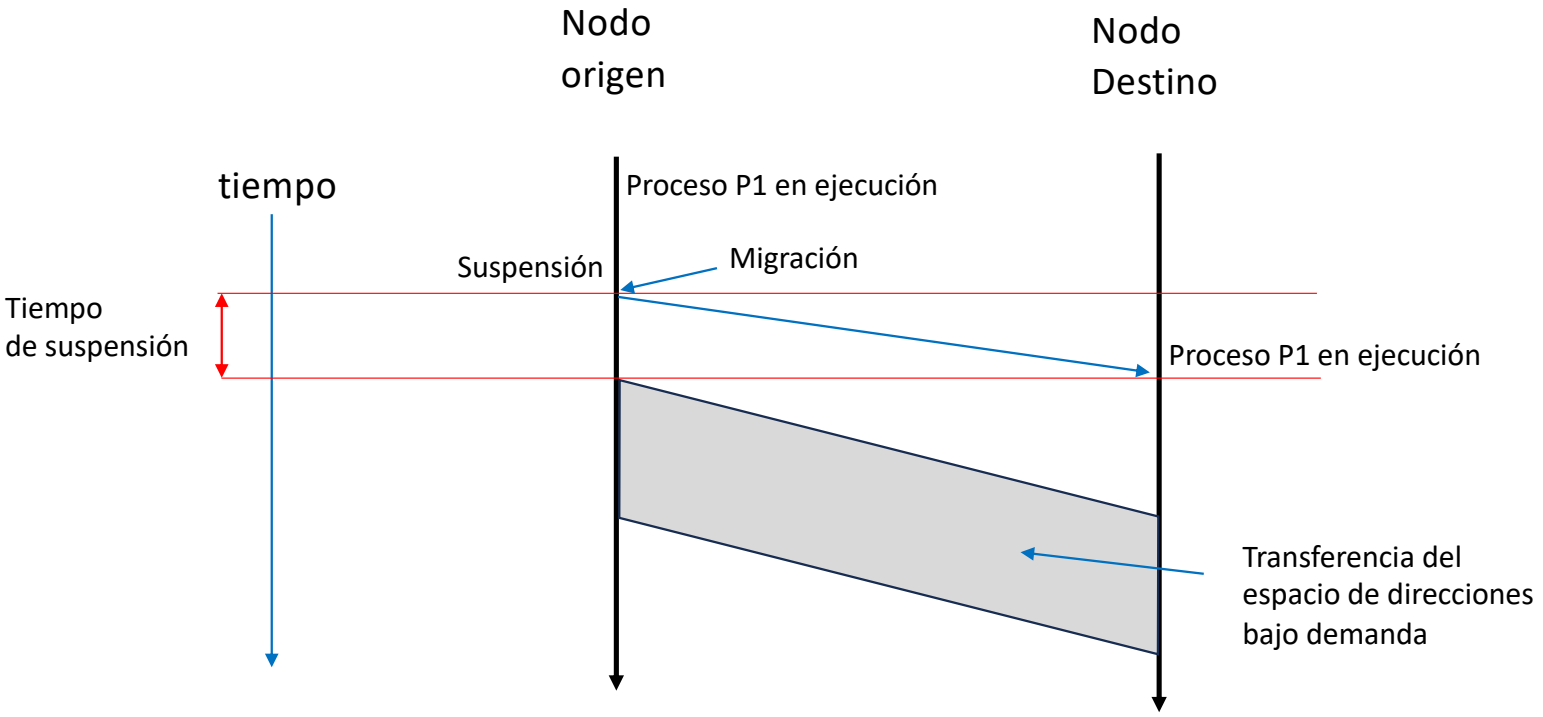
Pre-transferencia

- El espacio de direcciones es transferido mientras el proceso continúa ejecutando en el nodo origen



Transferencia bajo demanda

- El estado del proceso se transfiere inmediatamente y el espacio de direcciones se transfiere bajo demanda



Planificación de procesos en sistemas distribuidos y paralelos

- Definición del problema:
  - Dados un conjunto de procesadores (CE) y unidades de almacenamiento (SE) distribuidos
  - Dados trabajos con requisitos de CPU y de almacenamiento.
- El algoritmo de planificación es responsable de asignar trabajos a procesadores y determinar el orden en el que los trabajos se ejecutan en los procesadores
- Objetivo: buscar una estrategia que:
  - Maximice la productividad (*throuhgput*)
  - Minimice el tiempo de respuesta
  - Maximice el uso de los recursos en global

Planificación de tareas

- Trabajos que constan de un conjunto de n >= 1 tareas que pueden ejecutar en paralelo.
  - Número de tareas = grado de paralelismo
  - Cada tarea se asigna a un procesador
  - Se puede asignar más de una tarea del mismo trabajo al mismo procesador
  - Puede requerir sincronización de tareas al final de su ejecución
- En general el objetivo es equilibrar la carga y maximizar la productividad

Algoritmos de planificación

- FIFO (*fisrt in first out*).
  - Más justo para trabajos individuales, sin *overhead*, rendimiento subóptimo
- STF (*shortest time first*)
  - Necesita conocimiento previo de los tiempos de servicio
  - Algoritmo no expulsivo
  - Intenta minimizar el retardo medio y aumentar la productividad
  - Puede haber inanición
  - Requiere reordenación de las colas cada vez que llega un trabajo nuevo

Algoritmos de planificación

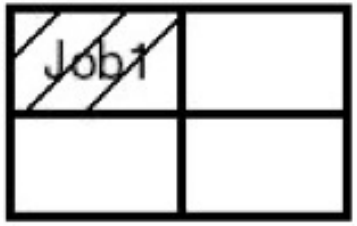
- *Epoch STF* (ESTF)
  - Las colas de planificación solo se reordenan en ciertos periodos de tiempo
- *Largest Job First Served (LJFS)*
  - Las tareas se ordenan por tamaño. Las tareas que pertenecen a trabajos más grandes se planifican primero
  - Este método mejora el rendimiento de grandes trabajos paralelos, pero esto es deseable en muchos casos (centros de supercomputación)

Algoritmos de planificación

- Backfilling:
  - Con FIFO, cuando el job que está en la cabeza intenta ejecutar y no tiene recursos disponibles, toda la cola se ve bloqueada, por lo que la finalización de las tareas se ve retrasada
  - Con backfilling se adelanta la ejecución de otros procesos cuando:
    - El job necesita los mismos o menos recursos que el job bloqueado
    - Su tiempo de ejecución esperado es inferior al tiempo calculado en que el job bloqueado tendrá sus recursos disponibles

Algoritmos de planificación

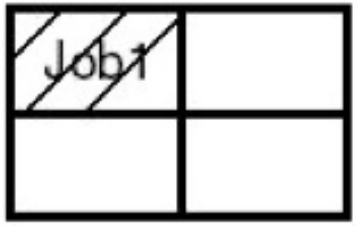
- Algoritmo Backfilling



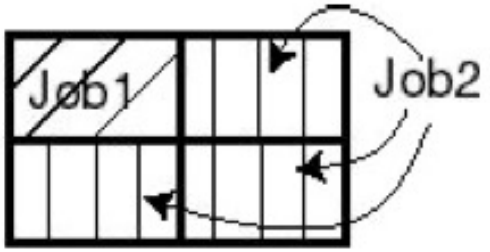
Trabajo 1 comienza a 7:00
y finaliza a las 9:00

Algoritmos de planificación

- Algoritmo Backfilling



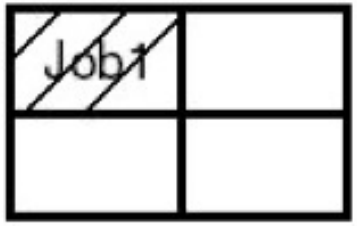
Trabajo 1 comienza a 7:00
y finaliza a las 9:00



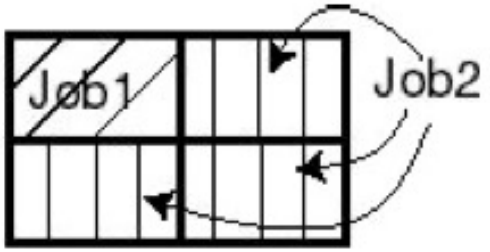
El job2 necesita 4 cores → bloquea

Algoritmos de planificación

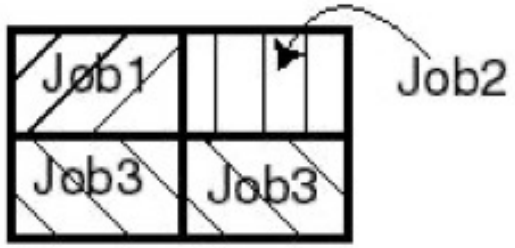
- Algoritmo Backfilling



Trabajo 1 comienza a 7:00
y finaliza a las 9:00



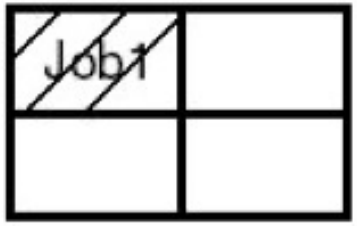
El job2 necesita 4 cores → bloquea



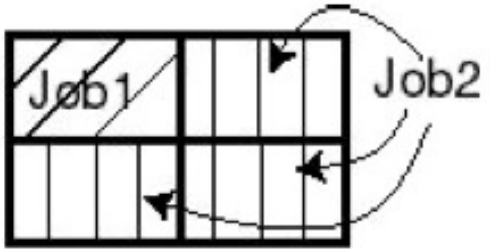
Trabajo 3 llega a 7:30
con tiempo planificado de 1:00 h

Algoritmos de planificación

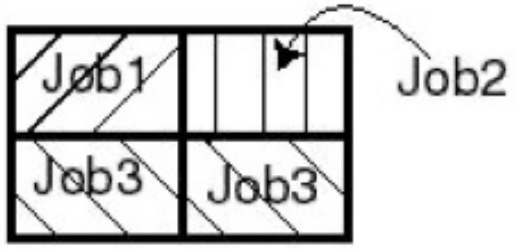
- Algoritmo Backfilling



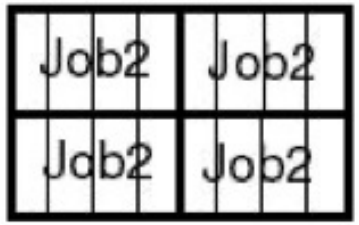
Trabajo 1 comienza a 7:00
y finaliza a las 9:00



El job2 necesita 4 cores → bloquea



Trabajo 3 llega a 7:30
con tiempo planificado de 1:00 h



A las 9:00 se planifica el Job2

Ejemplo de gestor de trabajos: SLURM

- Gestiona la asignación de recursos a trabajos en clusters
  - Nodos y cores
- Ofrece un entorno para arrancar, ejecutar y monitorizar la ejecución de trabajos
- Características:
  - Open Source
  - Tolerancia a fallos
  - Altamente escalable

Comandos básicos

- sbatch {job_script} : lanza un trabajo
- squeue: muestra los trabajos enviados
- scancel {job_id}: cancela un trabajo
- salloc: asignación de recursos:
  - Ejemplo: asignación de una sesión interactiva:
    - $> salloc --partition=interactive
- Ejemplo: Iniciar sesión interactiva reservando un nodo de cómputo en exclusiva:
  - $> salloc --exclusive

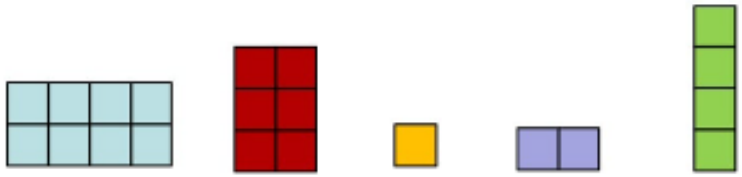
Ejemplo de trabajo secuencial

```
#!/bin/bash

#SBATCH --job-name="test_serial"
#SBATCH --output=serial_%j.out
#SBATCH --error=serial_%j.err
#SBATCH --ntasks=1
#SBATCH --time=00:02:00

./serial_binary> serial.out
```

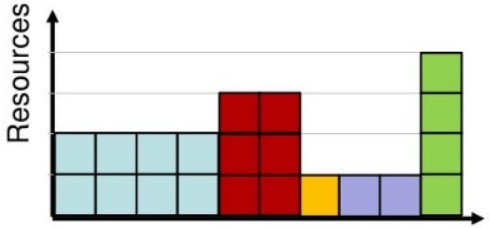
Ejemplo de trabajo paralelo con MPI

```
#!/bin/bash
#SBATCH --job-name=mpi
#SBATCH --output=mpi_%j.out
#SBATCH --error=mpi_%j.err
#SBATCH --nodes=16
#SBATCH --exclusive
#SBATCH --time=00:30:00

srun –n 16 ./mpi_binary
```

Planificación de trabajos en SLURM

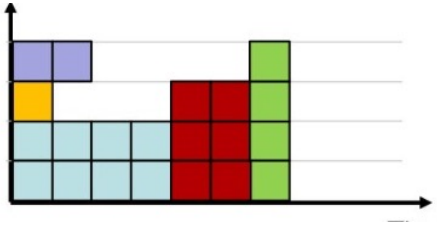


FIFO Scheduling

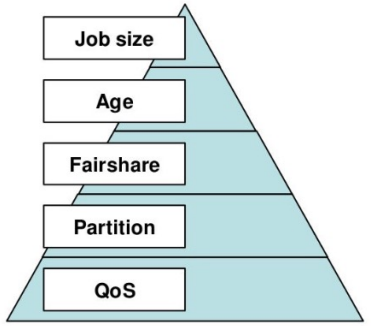


Backfill Scheduling

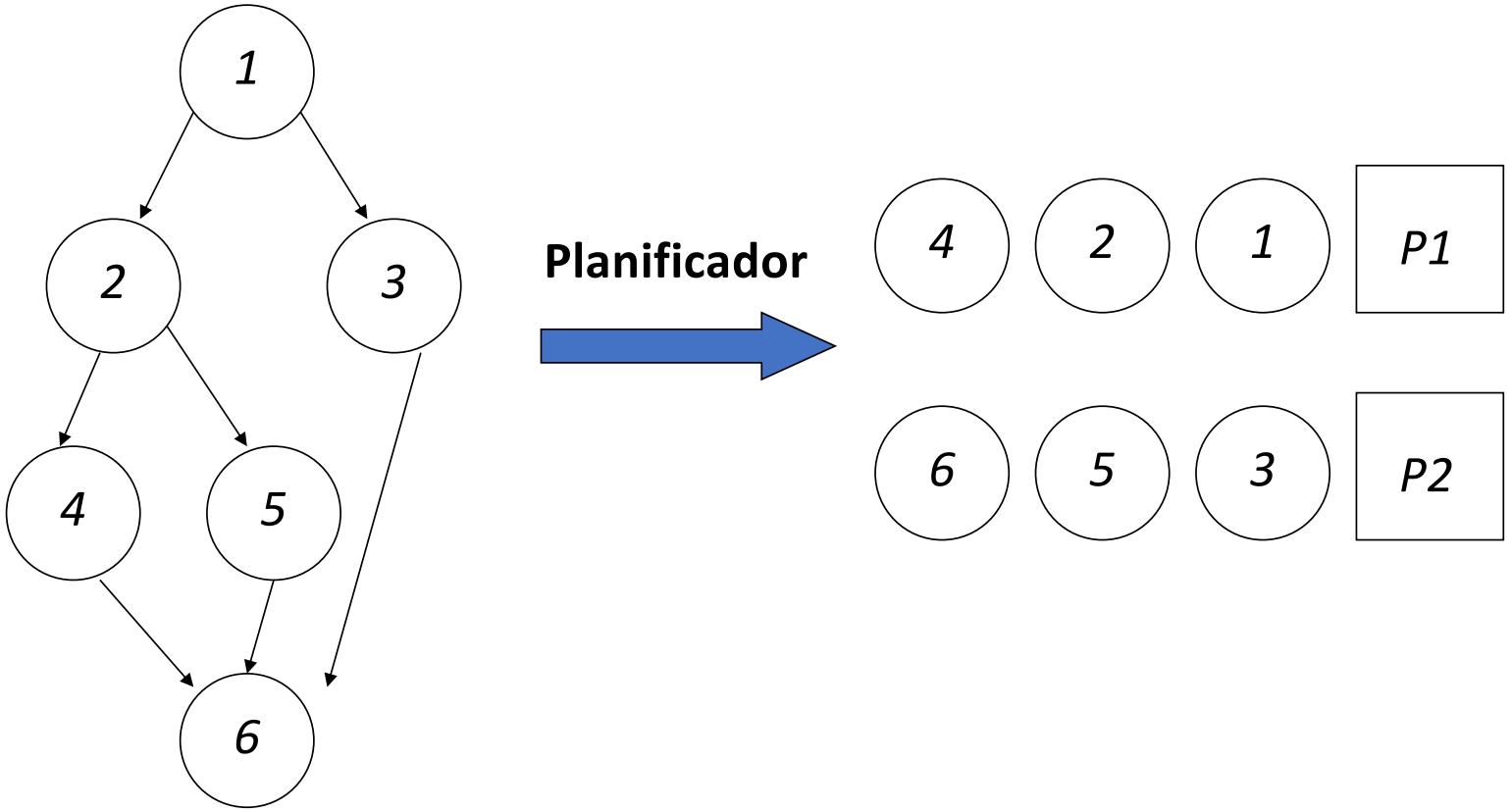
- Job priority
- Time limit **(Important!)**



- Priority factor:



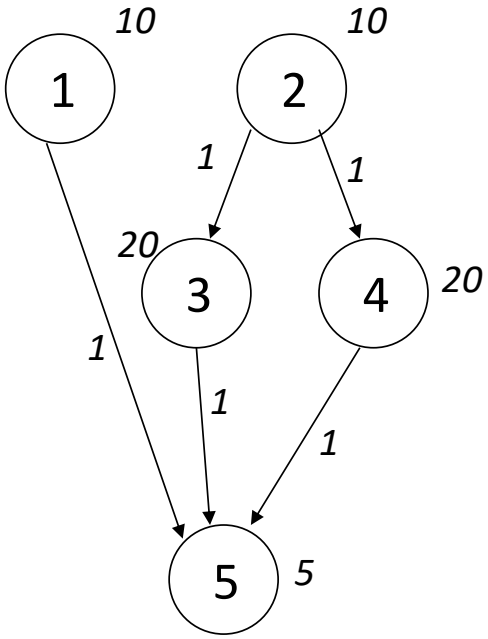
Planificación DAG



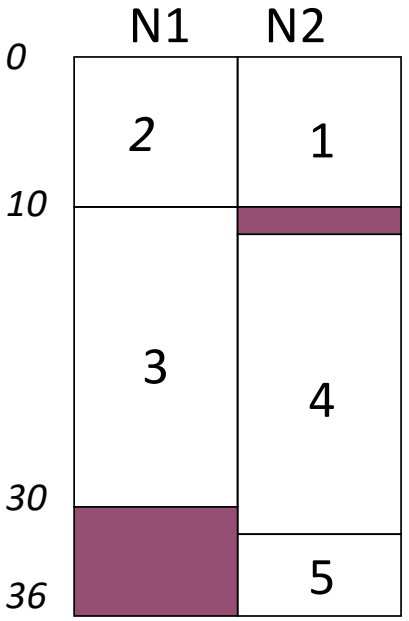
Complejidad del problema

- El problema en su forma general es NP-completo
- Algoritmos con complejidad polinomial:
  - Cuando sólo hay dos procesadores.
- En el caso general se utilizan heurísticas.
- Los planificadores se eligen por 2 métricas:
  - El rendimiento del plan generado.
  - La eficacia del planificador: tiempo tomado por el planificador para generar un plan.

Ejemplo



Planificador

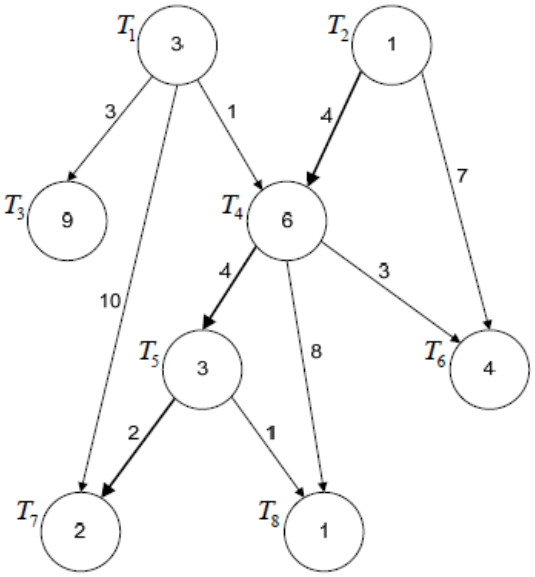


Criterios para asignar tareas

- Si dos tareas que se comunican se ejecutan en el mismo nodo, el tiempo de comunicación es cero y se reduce el tiempo de respuesta.
- Asignar tareas a diferentes nodos, incrementa el paralelismo y puede reducir el tiempo total de ejecución

Planificación DAG

- Una tarea sin predecesores se denomina tarea de entrada
- Una tarea sin sucesores se denomina tarea terminal
- Los predecesores de una tarea se denominan tareas padres
- Los sucesores de una tarea se denominan hijos
- Cada vértice en un DAG representa una tarea y el arco representa un mensaje que debe ser enviado de una tarea a otra.
- Cada arco tiene asignado un coste de comunicación y cada nodo el coste de ejecución
- Una tarea hijo puede comenzar su ejecución si ha recibido los datos de entrada de todos sus padres



A Directed Acyclic Graph (DAG)

Modelo

- Conjunto de procesadores $P = \{P_1, .. . P_m\}$
  - S_i es la velocidad del procesador P_i

- Una trabajo T consta de:
  - Un conjunto de tareas $T = \{T_1, .. T_n\}$
  - Un orden $<$; $T_i < T_j \rightarrow T_i$ debe terminar antes de que T_j pueda empezar
  - (D_{ij}) es la matriz de comunicaciones D_{ij} representa el coste de comunicación entre las tareas T_i y T_j
  - (A_i) es el vector de cómputo, A_i representa el coste de ejecución de la tarea T_i

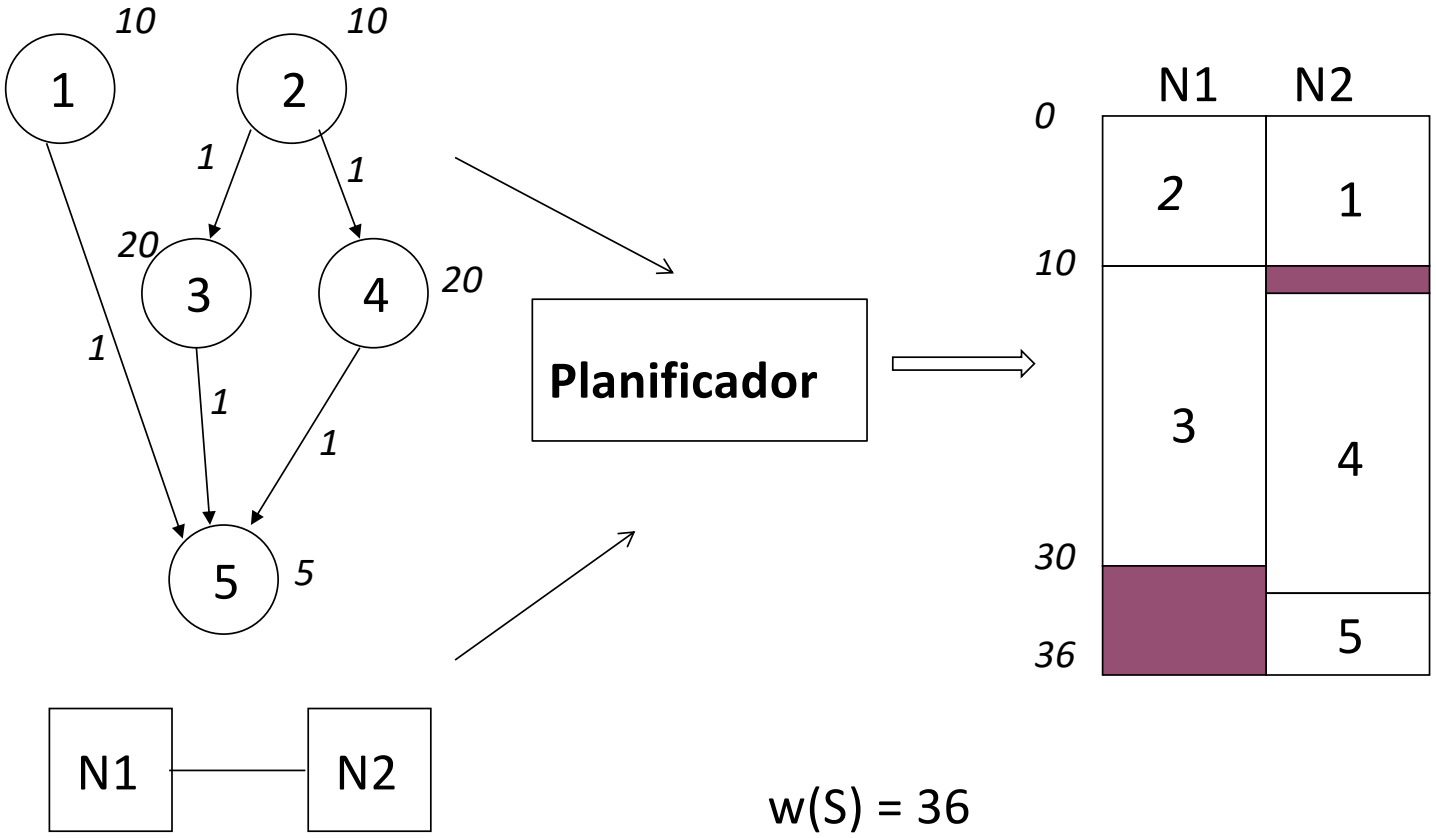
Medida de rendimiento

- s_i denota el tiempo de comienzo de la tarea T_i
- f_i denota el tiempo de terminación de la tarea T_i

- Longitud de planificación o tiempo máximo de terminación

  $w(S) = \max \{ f_i(S) \}$

Ejemplo


10
10
1
2
1
1
20
3
4
20
1
1
1
5
5
Planificador
N1
N2
0
10
30
36
2
1
3
4
5
N1
N2
w(S) = 36

Planificación estática óptima con dos procesadores

- No hay ningún algoritmo polinomial de planificación para tareas DAG

- Algoritmo de *Coffman-Graham* para *m = 2* (procesadores) $O(n^2)$
  - *Todas las tareas tienen el mismo tiempo de ejecución*
  - *Objetivo:* asignar una etiqueta j a cada tarea

Algoritmo de *Coffman-Graham*

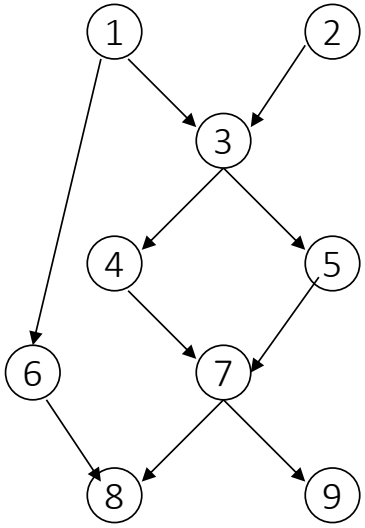
1) Asignar (etiquetas) 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por las etiquetas de sus sucesores
   - Sea x un elemento de S tal que $\forall$ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Algoritmo de *Coffman-Graham*

3) Cuando todas las tareas tienen etiqueta, se usa la lista $(T_n, T_{n-1}, \ldots T_1)$ donde $L(T_i) = i$ para planificar las tareas

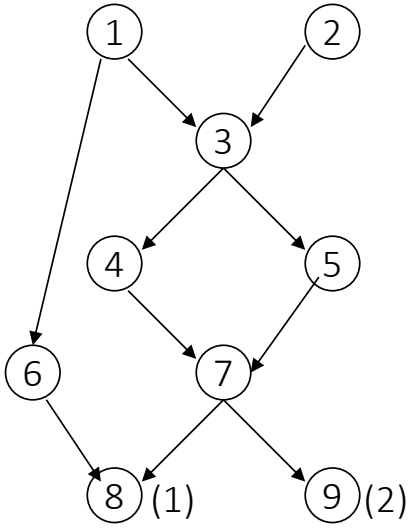
4) Cuando un procesador se hace disponible se asigna la siguiente tarea de la lista.

Ejemplo



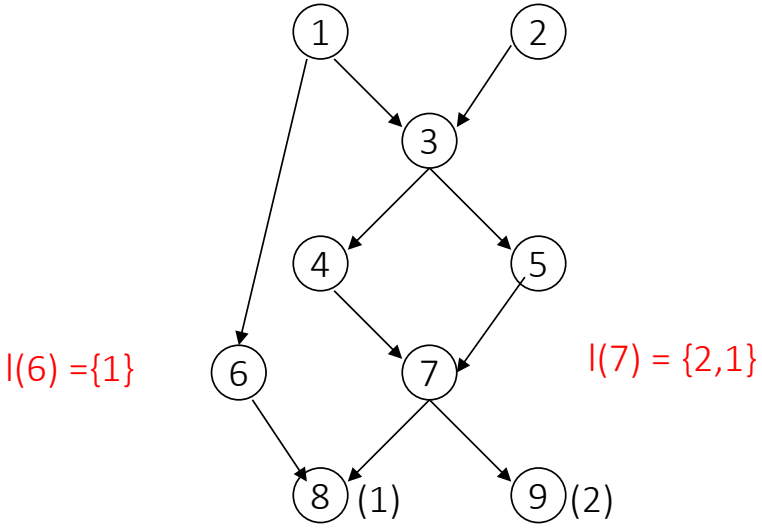
1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que $\forall$ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Ejemplo



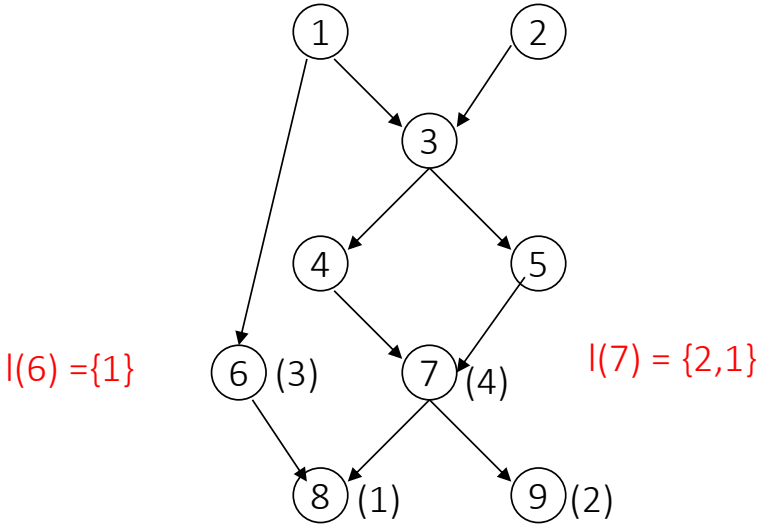
1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que $\forall$ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Ejemplo



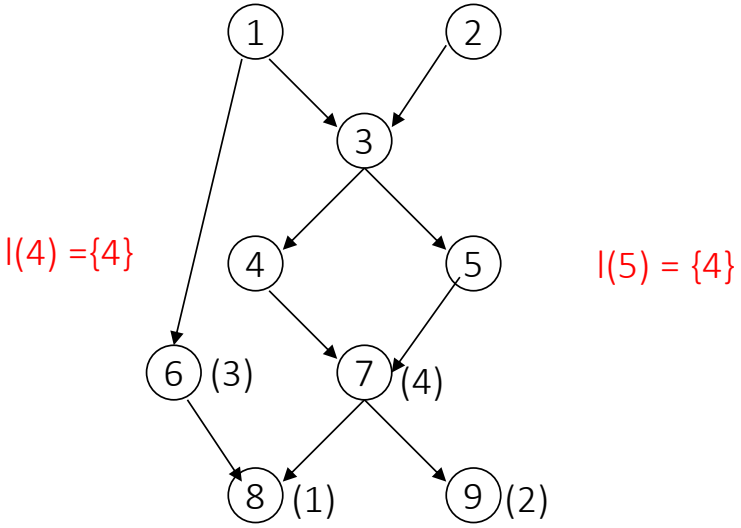
1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que ∀ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Ejemplo



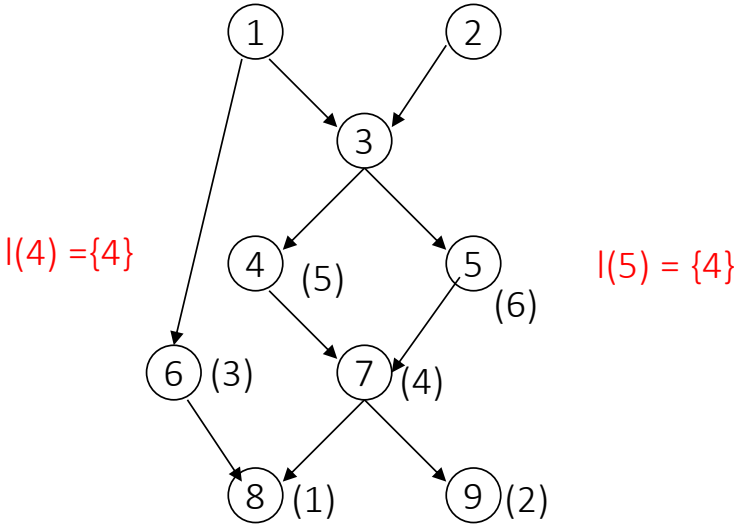
1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que $\forall$ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Ejemplo



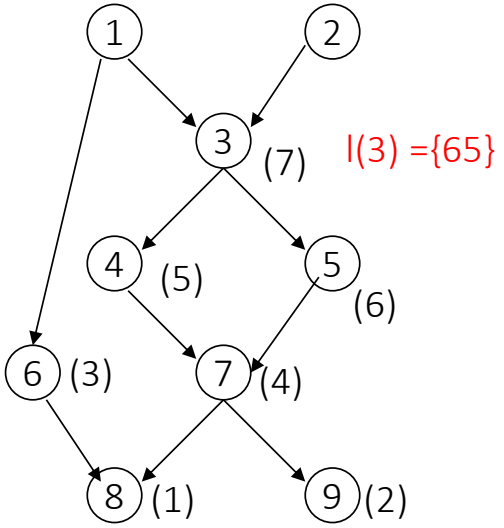
1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que $\forall$ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Ejemplo



1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que $\forall$ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

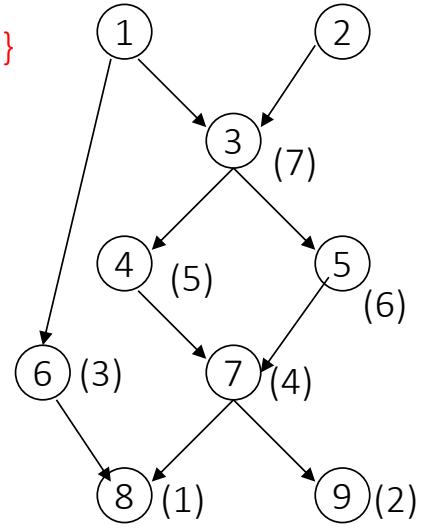
Ejemplo



1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que $\forall$ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Ejemplo

l(1)={73}

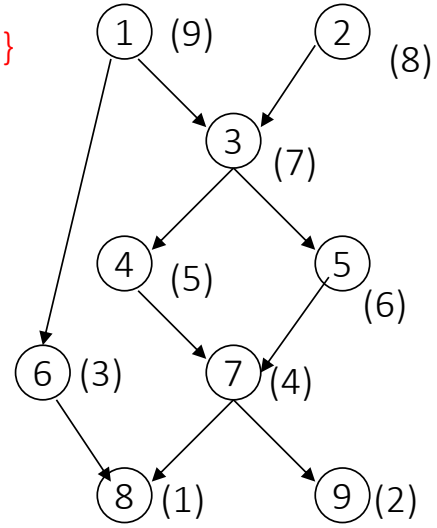


l(2) = {7}

1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que ∀ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

Ejemplo

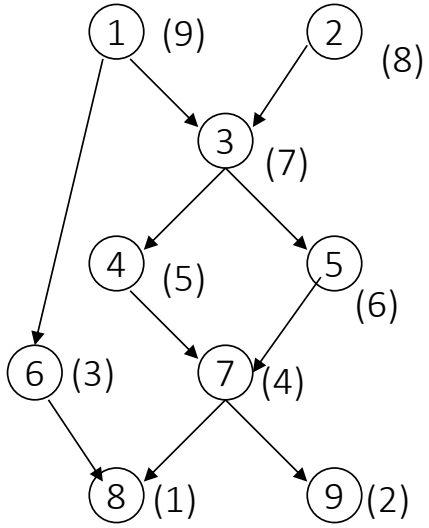
l(1)={73}



L(2) = {7}

1) Asignar 1,2, 3,.. a una de las tareas terminales
2) Sea 1, 2, … *j*-1, el conjunto de etiquetas ya asignadas. Sea S el conjunto de tareas no asignadas con sucesores etiquetados. Asignamos la siguiente etiqueta j a un elemento de S:
   - Para cada nodo x en S, $l(x) = \{y_1, y_2, \ldots y_k\}$
     - y_i : etiquetas de sus sucesores
   - l(x) es la secuencia decreciente de enteros formados por la etiquetas de sus sucesores
   - Sea x un elemento de S tal que ∀ x' en S, $l(x) \leq l(x')$ (lexicográficamente)
     - L(x) = j (se asigna la etiqueta j al nodo x)

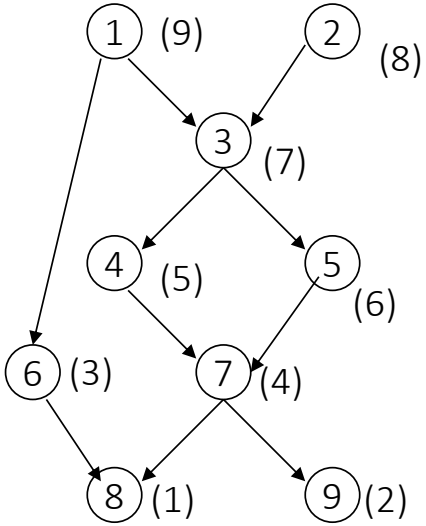
Ejemplo



3) Cuando todas las tareas tienen etiqueta, se usa la lista $(T_n, T_{n-1}, \ldots T_1)$ donde $L(T_i) = i$ para planificar las tareas

4) Cuando un procesador se hace disponible se asigna la siguiente tarea de la lista.

Ejemplo



La lista de ejecución es:

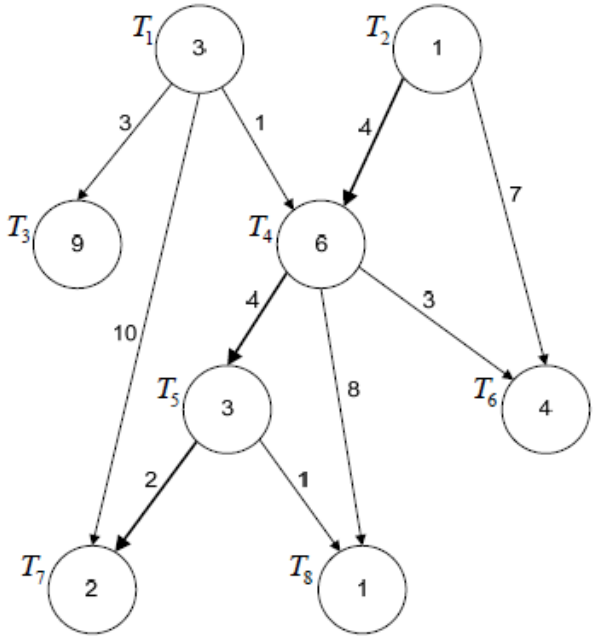
L={1,2,3,5,4,7,6,9,8}

0	1	2	3	4	5	6	7
1	3	5	7	9			
2		4	6	8			

Planificación basada en caminos críticos

- El nivel L de una tarea es la longitud del camino más largo de esa tarea a una tarea terminal del grafo, teniendo en cuenta los tiempos de ejecución de las tareas y los costes de comunicación
  - El nivel L de una tarea terminal es igual a su tiempo de ejecución.
- El nivel SL (*static level*) de una tarea es la longitud del camino más largo de una tarea a una terminal sin tener en cuenta el coste de comunicación
  - El nivel SL de una tarea terminal es igual a su tiempo de ejecución.
- La longitud del camino crítico de un DAG es la longitud del camino más largo de una tarea de entrada a una tarea terminal

Ejemplo



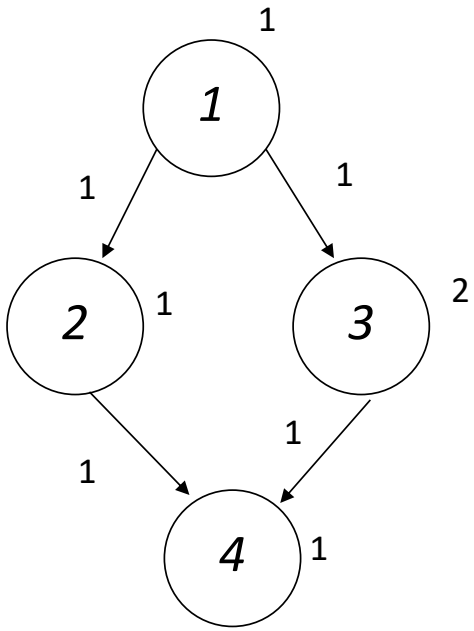
Task	Level L	Static Level SL
T_1	21	14
T_2	22	12
T_3	9	9
T_4	17	11
T_5	7	5
T_6	4	4
T_7	2	2
T_8	1	1

El camino crítico es 22

Algoritmo *Earliest-Task-First*

1. Prioridad de la tarea j = Nivel de la tarea
2. Se calcula el número de predecesores (NP) inmediatos de cada tarea
3. Se inicializa una cola con las tareas listas que no tienen predecesores inmediatos (NP = 0)
4. Mientras la cola no esté vacía
    1. Obtener una tarea de la cola
    2. Seleccionar un procesador para ejecutar la tarea. Se selecciona un procesador de forma que la tarea no pueda terminar antes en otro procesador
    3. Cuando una tarea termina se resta 1 al valor NP de todos sus sucesores
    4. Cuando NP se hace 0 para una tarea, se inserta en la cola

Ejemplo



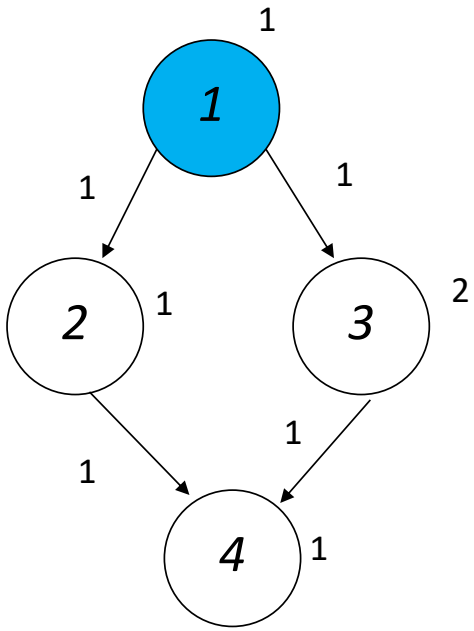
número de predecesores (NP) inmediatos de cada tarea

Tarea	NP	Nivel L
1	0	6
2	1	3
3	1	4
4	2	1

longitud del camino más largo de esa tarea a una tarea terminal del grafo

Ejemplo

1. Se inicializa una cola con las tareas listas que no tienen predecesores inmediatos (NP = 0) y se ejecutan

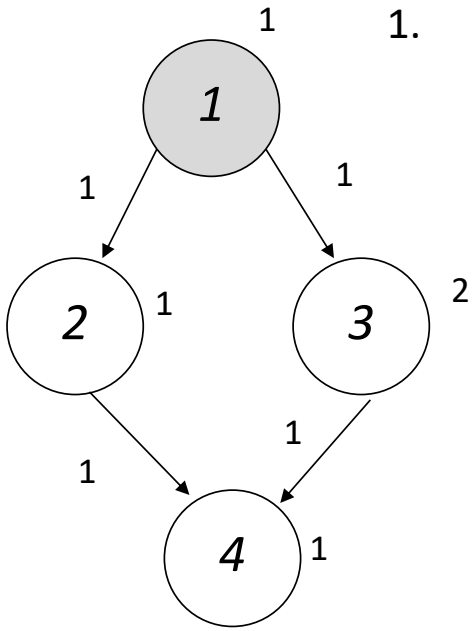


Tarea	NP	Nivel L
1	0	6
2	1	3
3	1	4
4	2	1

Ejemplo

1. Cuando una tarea termina se resta 1 al valor NP de todos sus sucesores y se
2. Insertan en la lista de nodos con NP=0

1.

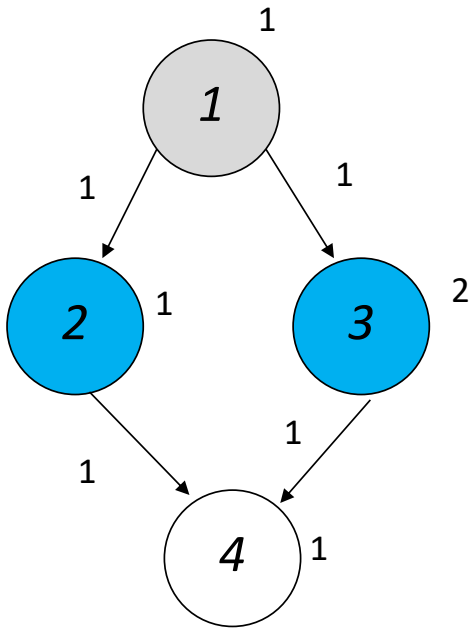


Tarea	NP	Nivel L
1	0	6
2	0	3
3	0	4
4	2	1

Ejemplo

1. Se ejecutan las tareas con NP = 0

1.

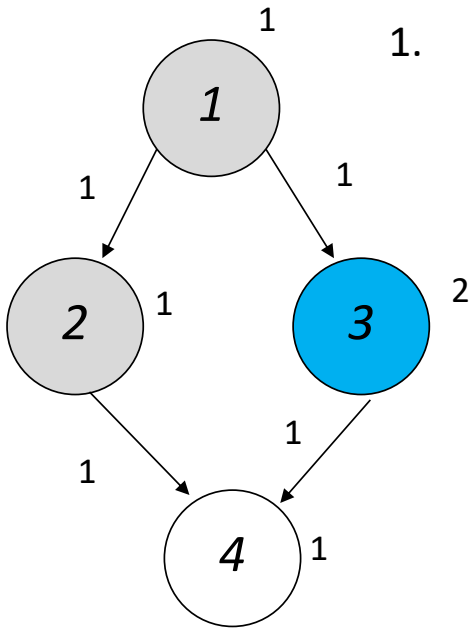


Tarea	NP	Nivel L
1	0	6
2	0	3
3	0	4
4	2	1

Ejemplo

1. Termina 2
2. Se resta 1 al valor NP de 4

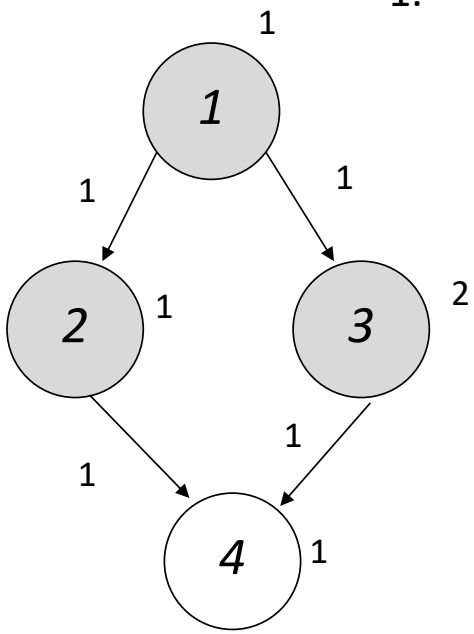
1.



Tarea	NP	Nivel L
1	0	6
2	0	3
3	0	4
4	1	1

Ejemplo

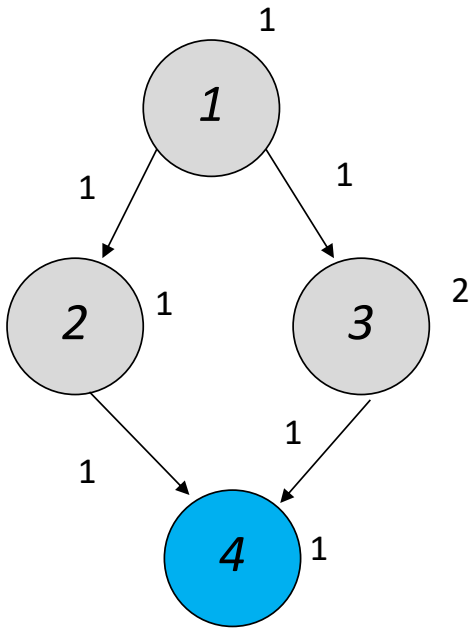
1. Termina 3
2. Se resta 1 al valor NP de 4 se inserta en la lista de tareas con NP = 0
1.



Tarea	NP	Nivel L
1	0	6
2	0	3
3	0	4
4	0	1

Ejemplo

1. 4 se inserta en la lista de tareas con NP = 0 y se ejecuta
1.

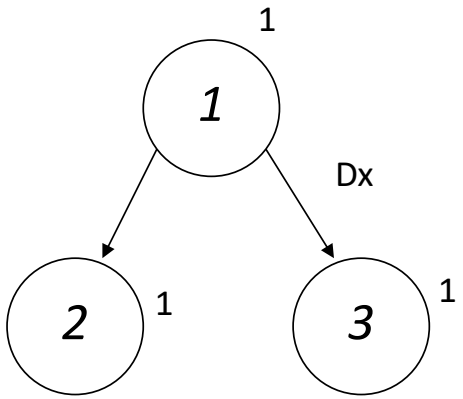


Tarea	NP	Nivel L
1	0	6
2	0	3
3	0	4
4	0	1

Heurísticas de planificación

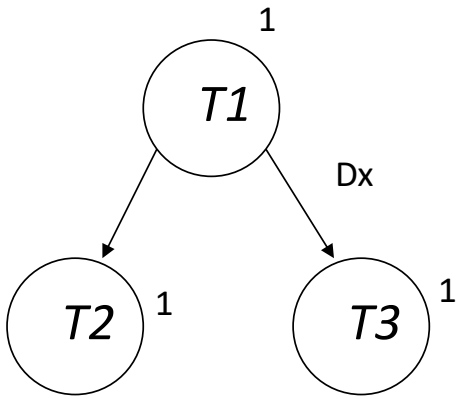
- Factores que incrementan la complejidad de la planificación:
  - Tareas con tiempos de ejecución diferentes
  - Enlaces entre nodos con velocidades diferentes
  - Enlaces compartidos o dedicados
  - Topología de la red
  - Heterogeneidad de los nodos
  - Estructura del programa paralelo
- La planificación heurística intenta aplicar heurísticas que reduzcan la complejidad del problema

Paralelismo y retardo en las comunicaciones

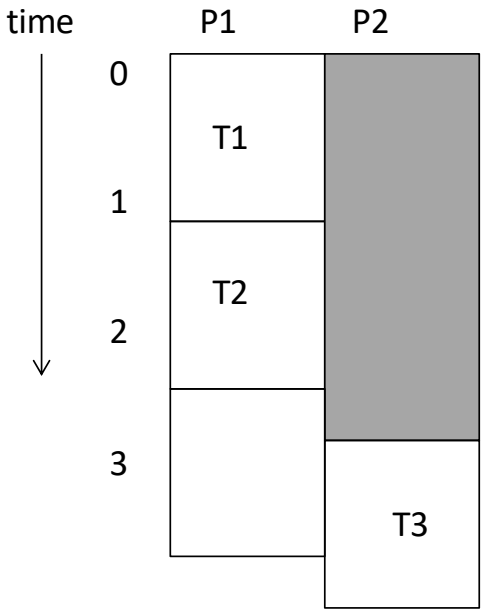


¿Cómo planificar en dos procesadores?

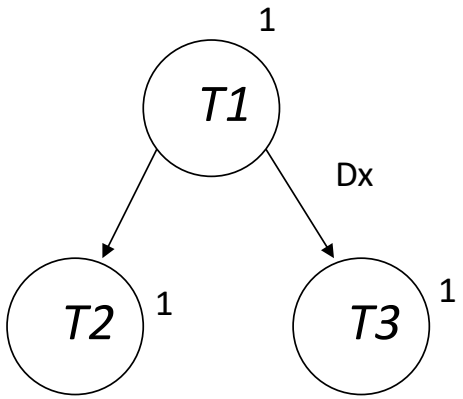
Paralelismo y retardo en las comunicaciones



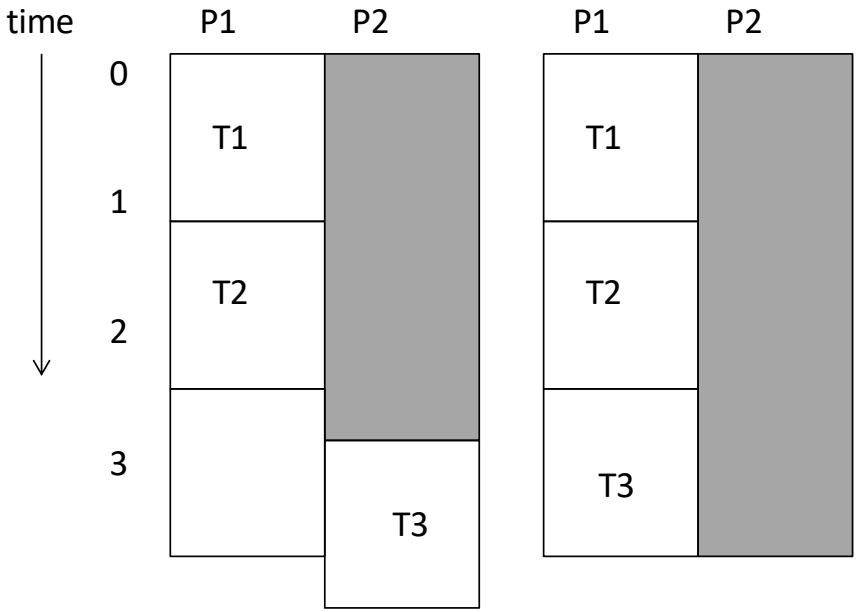
Si Dx > tiempo de ejecución de T2



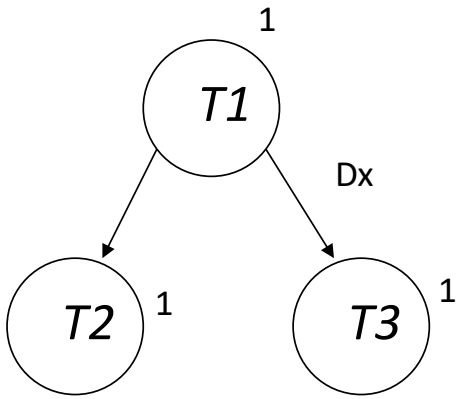
Paralelismo y retardo en las comunicaciones



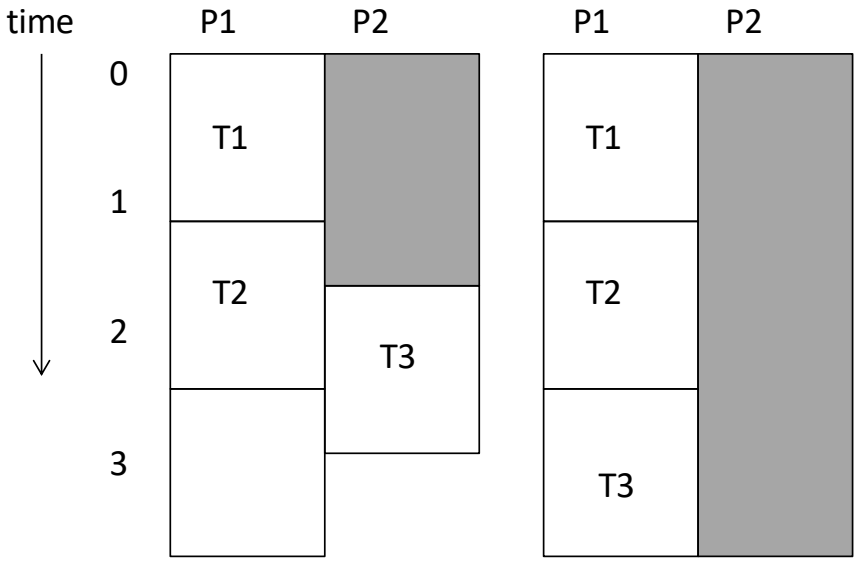
Si Dx > tiempo de ejecución de T2



Paralelismo y retardo en las comunicaciones

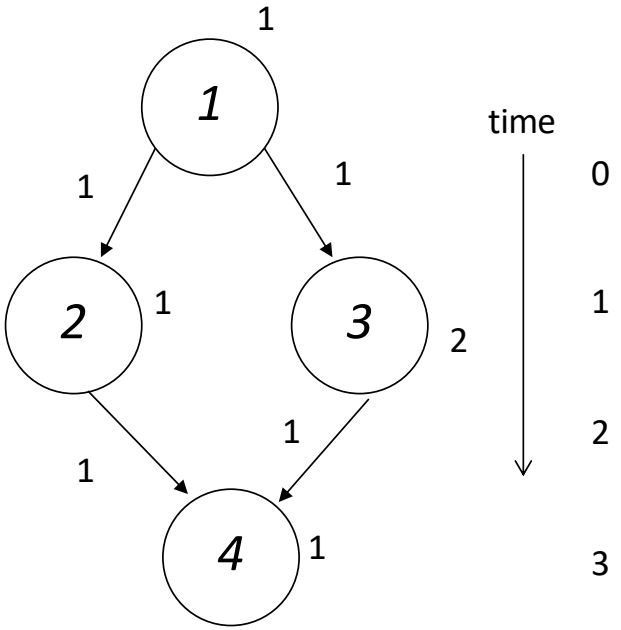


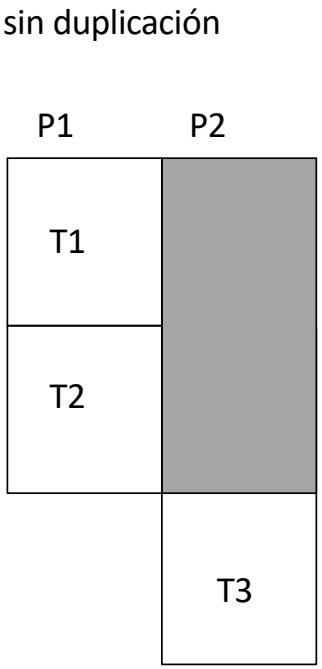
Si Dx < tiempo de ejecución de T2



Duplicación de tareas

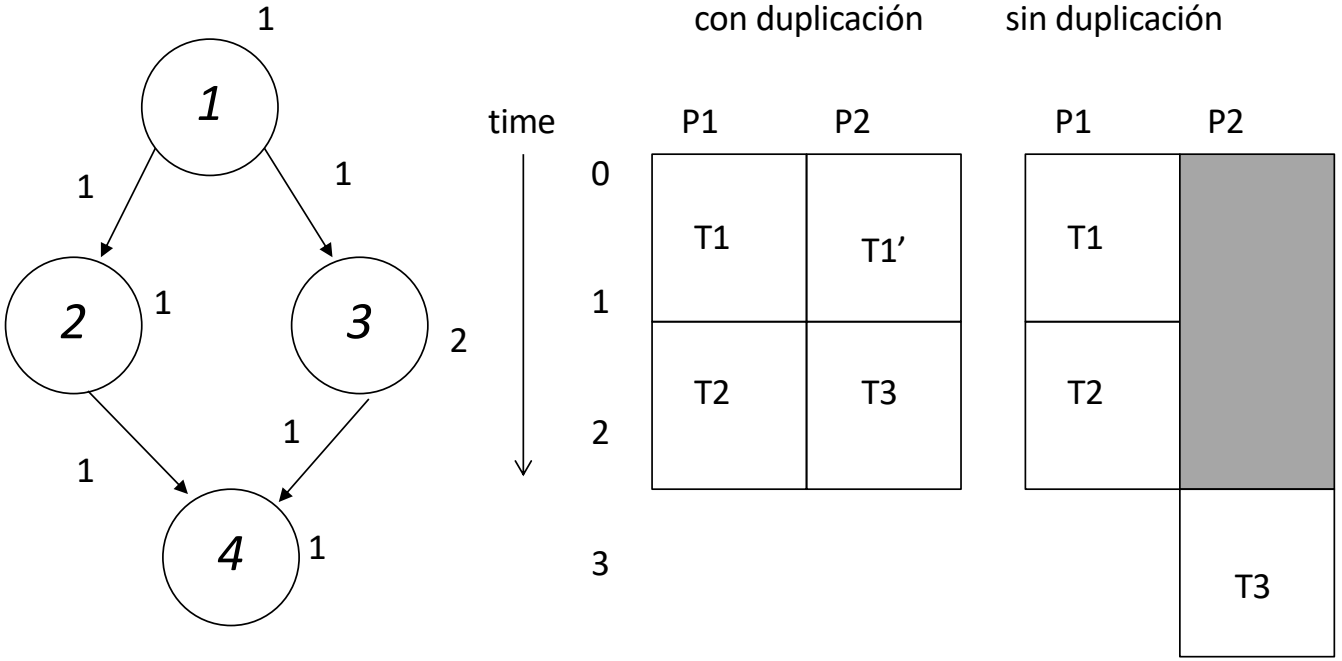
- Heurística que duplica la ejecución de las tareas para reducir los retardos de comunicación





Duplicación de tareas

- Heurística que duplica la ejecución de las tareas para reducir los retardos de comunicación



Algoritmos de distribución de la carga

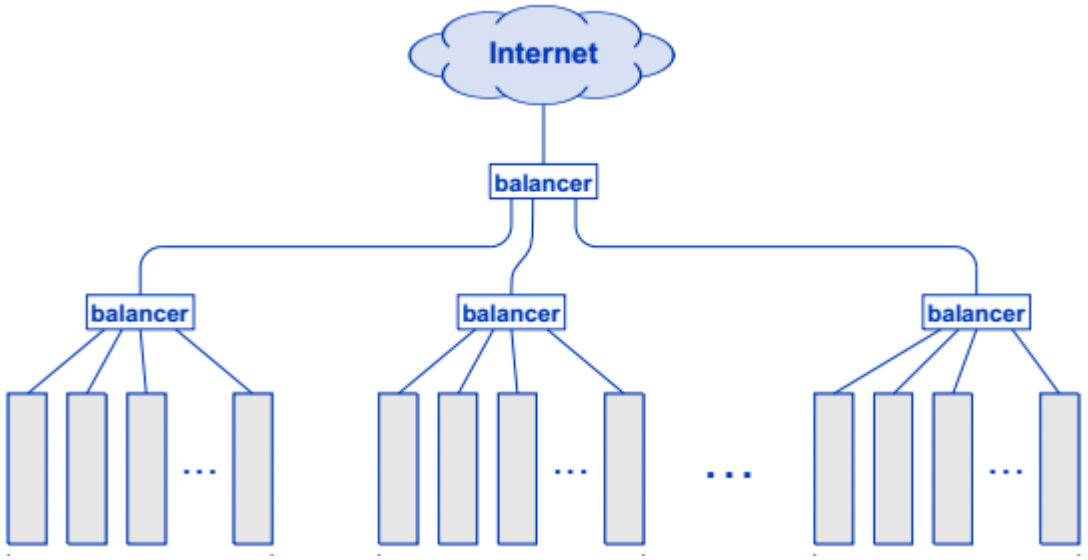
- ***Objetivo***: decidir en qué procesador se debería ejecutar un nuevo trabajo para equilibrar la carga y optimizar el rendimiento.
  - Evitar que un nodo esté inactivo mientras hay procesos esperando a ejecutar.
- ***Suposiciones***:
  - Todos los procesadores son compatibles en el código.
  - La velocidad de los procesadores puede ser distinta.
  - Conectividad total: cualquier procesador puede comunicarse con cualquier otro.

Algoritmos clásicos

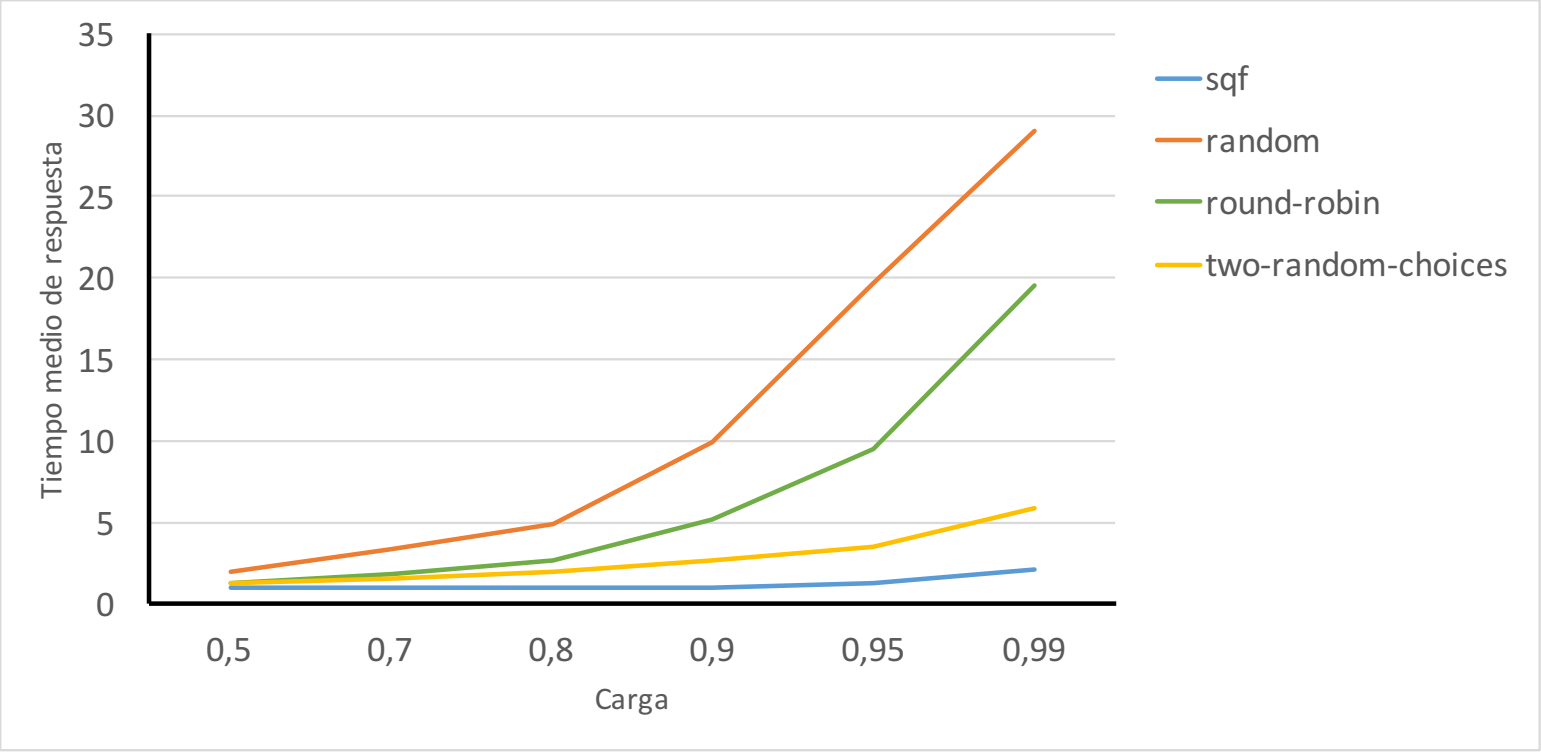
- Random: cada trabajo nuevo se envía a un procesador/servidor elegido aleatoriamente
- Cíclico (Round Robin): los trabajos se van asignando de forma cíclica
- *Shortest-queue-first*: cada trabajo nuevo se asigna al procesador/servidor con el menor número de trabajos
- *Two-random-choices (SQ-2)*: cada vez que llega un proceso nuevo se eligen dos procesadores/servidores de forma aleatoria y el trabajo se envía al que menor número de trabajos tiene
- Two-random choices with round-robin (SQ-RR(2)): cada vez que llega un proceso nuevo se eligen dos procesadores/servidores, uno de forma aleatoria y otro cíclico y el trabajo se envía al que menor número de trabajos tiene

Felix García, and Alejandro Calderon. "Reducing Randomization in the Power of Two Choices Load Balancing Algorithm." *High Performance Computing & Simulation (HPCS), 2017 International Conference on*. IEEE, 2017.

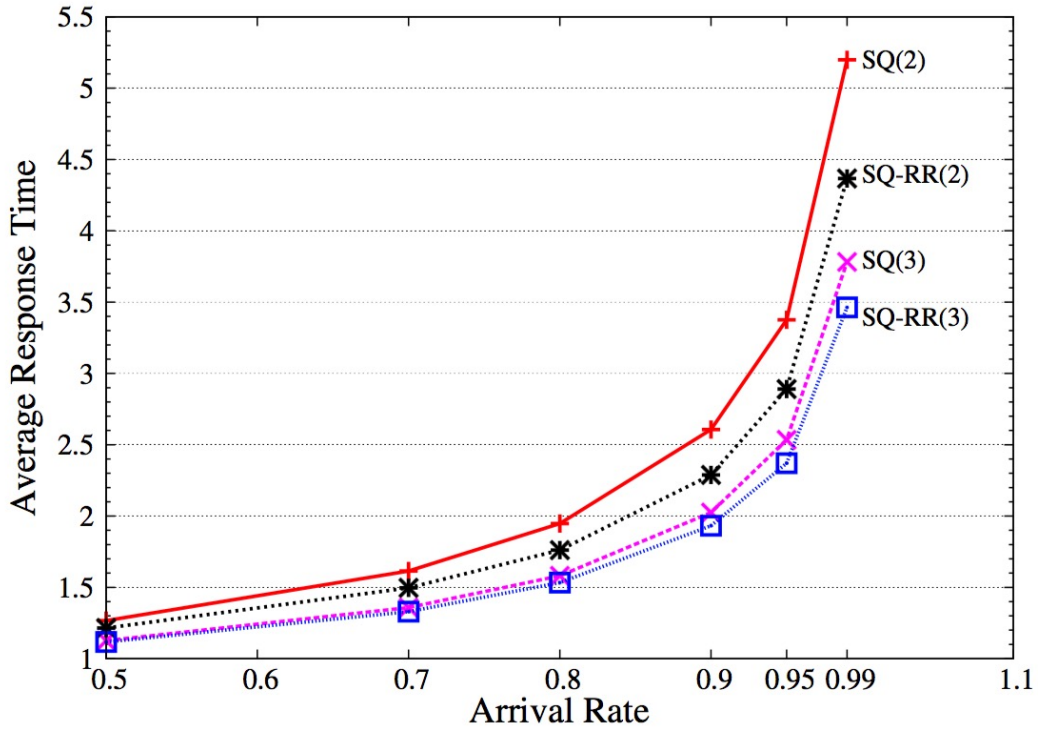
Diferentes niveles de equilibrio de carga



Evaluación


35
30
25
20
15
10
5
0
Tiempo medio de respuesta
0,5
0,7
0,8
0,9
0,95
0,99
Carga
sqf
random
round-robin
two-random-choices

SQ(2) vs SQ-RR(2)
1000 servidores



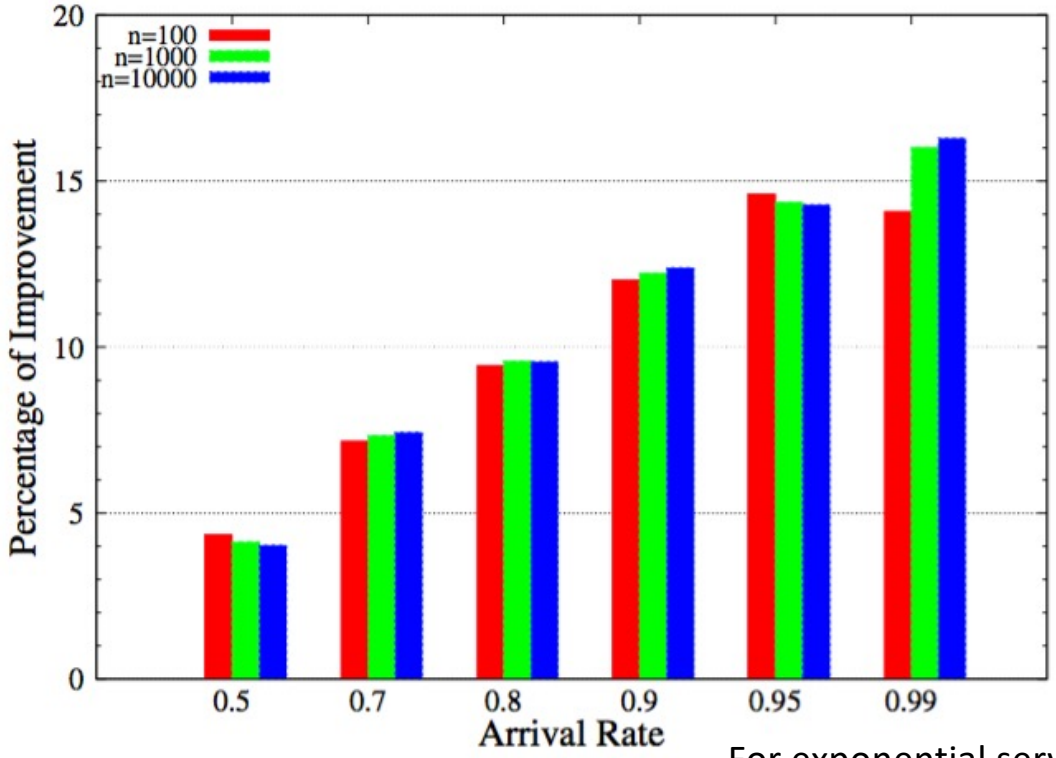
For exponential services time

Probabilidad de elegir un servidor vacío (1000 servidores y d = 2)

λ	SQ(d) Simul.	SQ-RR(d) Simul.
0.50	0.748	0.834
0.70	0.503	0.601
0.80	0.357	0.449
0.90	0.196	0.253
0.95	0.085	0.136
0.99	0.024	0.039

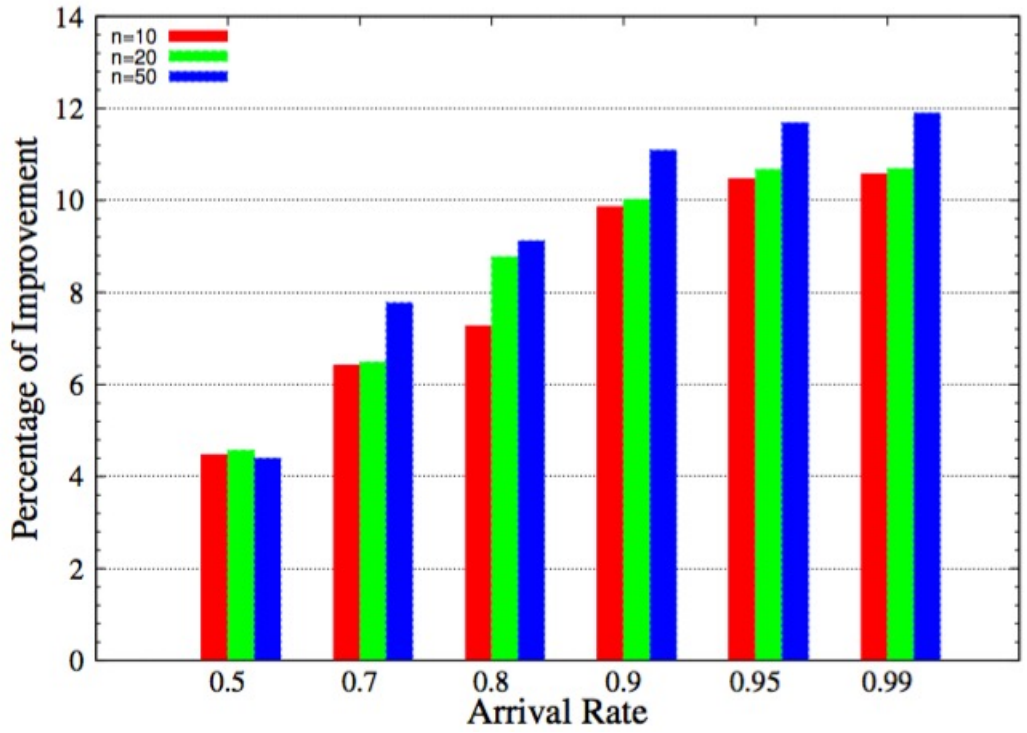
For exponential services time

Porcentaje de mejora de SQ-RR sobre SQ (n=100.n=1000. n=10000 servidores)



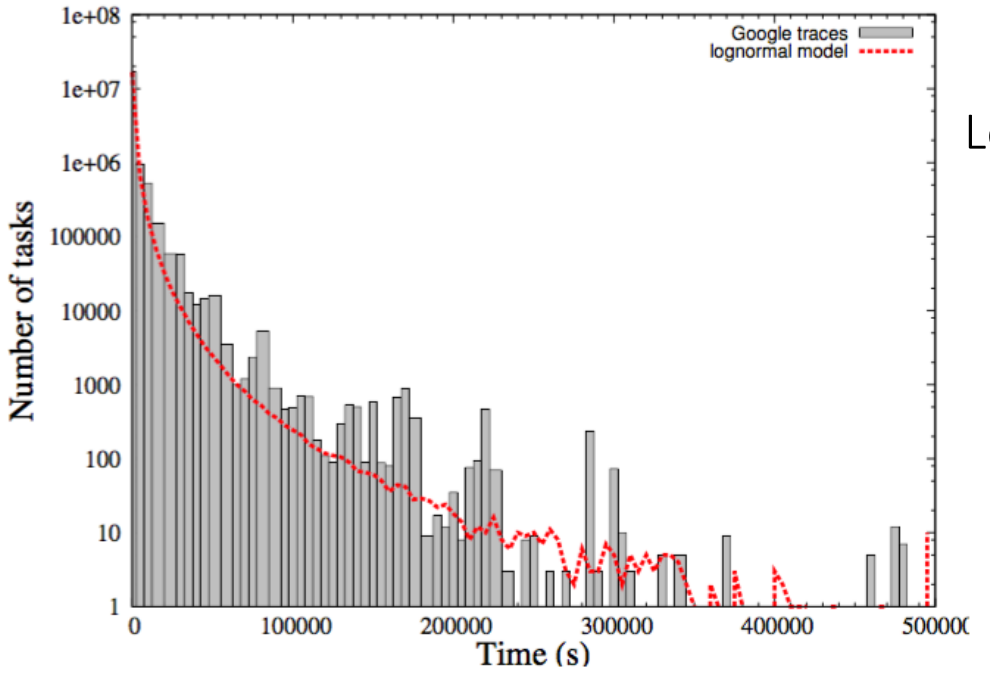
For exponential services time

Porcentaje de mejora de SQ-RR sobre SQ (n=10, 20 y 50 servidores)



For exponential services time

Resultados para cargas de trabajo reales



Log-normal aprox:

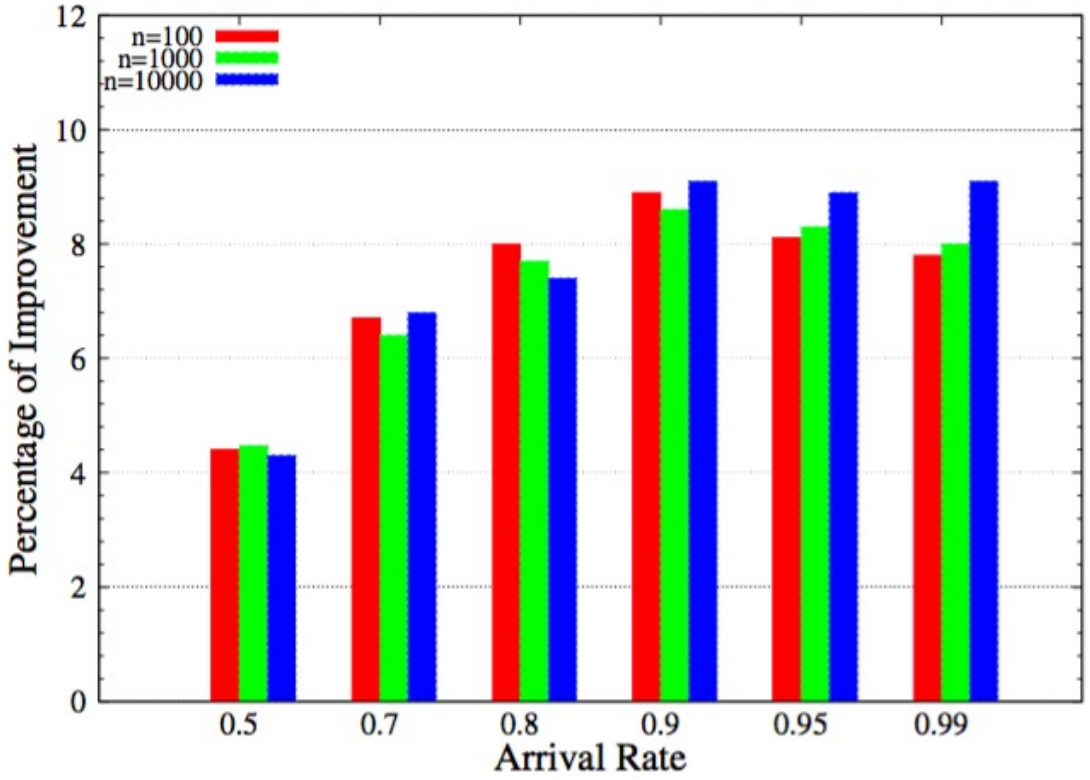
σ=1.42

μ=6.25

- Distribution of tasks execution time (Google Datacenter traces vs lognormal characterization) [7]

[7] G. Da Costa, L. Grange, I. De Courchelle *Modeling and Generating large-scale Google-like Workload*. In International Workshop on Resilience and/or Energy-aware techniques for High-Performance Computing. Nov 2016, Hangzhou, China. 2016

Porcentaje de mejora de SQ-RR sobre SQ para cargas de trabajo Google



Asignación de procesadores

- ***Estrategias***:
  - *No migratorias*: una vez arrancado un proceso éste no cambia de procesador.
  - *Migratorias*: los procesos pueden cambiar de procesador durante su ejecución.
    - Mejor equilibrio de la carga pero más complejas.
- ***Criterios de optimización***:
  - Maximizar la utilización total de los procesadores
  - Minimizar el tiempo de respuesta medio
  - Minimizar la tasa de respuesta
  - Maximizar la productividad

Algoritmos de asignación y distribución

- **Política de transferencia**: determina cuándo transferir.
- **Política de selección**: selección del proceso a transferir.
- **Política de ubicación**: selecciona el nodo al que transferir.
- **Política de información**: decide cuándo, desde dónde y qué información sobre otros nodos recoger.

Política de transferencia

(determina cuándo transferir)

- Basadas en ***umbral***:
  - Si la carga excede de *T* unidades de carga en el nodo S, éste se convierte en emisor de un proceso.
  - Si la carga cae por debajo de *T* unidades, entonces S se convierte en receptor de procesos remotos.
- Tipos de transferencias:
  - ***Expulsivas***: se pueden transferir los procesos ejecutados parcialmente.
    - Supone transferir el estado del proceso (*migración*)
  - ***No expulsivas***: los procesos en ejecución no pueden ser transferidos.

Políticas de selección

(selección del proceso a transferir)

- Elegir los procesos nuevos.
- Elegir procesos en ejecución
  - Seleccionar los procesos con un tiempo de transferencia mínimo (poco estado, mínimo uso de los recursos locales).
  - Seleccionar un proceso si su tiempo de respuesta estimado en un nodo remoto es menor que el tiempo de respuesta local.

Política de ubicación

(selecciona el nodo al que transferir)

- Muestreo: se muestrean otros nodos para encontrar uno adecuado.
  - Los nodos pueden ser muestreados secuencialmente o en paralelo.
  - Selección aleatoria.
  - Nodos más próximos.
  - Basada en información recogida anteriormente.
- Enviar un mensaje al resto de nodos (*broadcast*).
- Dos tipos de políticas:
  - Iniciadas por el emisor.
  - Iniciadas por el receptor.

Política de información

(información sobre otros nodos)

- ***Bajo demanda***: la información se recoge sólo cuando un nodo se convierte en un emisor o receptor de procesos.
  - ***Iniciada por el emisor***: el emisor busca receptores.
  - ***Iniciada por el receptor***: el receptor solicita procesos a otros nodos.
  - ***Combinada***: iniciada por el emisor o por el receptor.
- ***Periódicas***: los nodos intercambian información periódicamente.
  - Sobrecarga constante en el sistema.
  - No se adapta a las necesidades:
    - Alta periodicidad: mucha sobrecarga.
    - Baja periodicidad: reparto de carga ineficaz.

Política de información

(información sobre otros nodos)

- ***Dirigida por el cambio de estado***: los nodos envían su información sólo cuando cambia su estado.
  - Centralizado: la información se envía a un nodo central.
  - Distribuido: la información se envía a todos los nodos.
  - La recogida de información depende de la carga del sistema.

Estabilidad y efectividad

- Un algoritmo es ***inestable*** si la tasa de trabajo que llega a un nodo (externa + transferencias) excede de la capacidad de ese nodo.
- Un algoritmo es ***inestable*** si existe la probabilidad de que los procesos se muevan de un nodo a otro sin haber ejecutado.
- Un algoritmo es ***efectivo*** si mejora el rendimiento del sistema.

Clasificación de los algoritmos

- ***Dinámicos***: Utilizan la carga local para tomar decisiones.
  - Tienen en cuenta las posibles fluctuaciones.
  - Sobrecarga en la recogida, almacenamiento y análisis de la información.
- ***Deterministas***: Se toma información a priori para la toma de decisiones.
- ***Adaptativos***: similar a los dinámicos pero adaptando el algoritmo a la carga del sistema.
- Algoritmos ***óptimos*** o ***subóptimos***.
- Algoritmos ***locales*** o ***globales***.
  - Con locales se transfiere un proceso cuando la máquina local está muy cargada.
  - Con globales se tiene en cuenta la carga global del sistema.

Algoritmos iniciado por el emisor

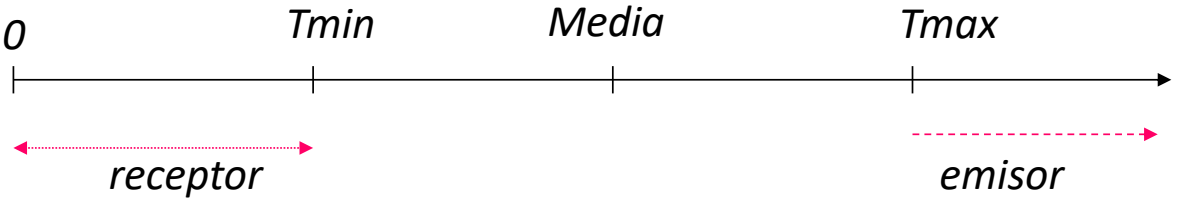
- ***Política de transferencia***: umbral basado en la longitud de la cola de CPU.
- ***Política de selección***: procesos nuevos.
- ***Política de ubicación***:
  - Elegir un nodo al azar.
  - Probar con un nº de nodos para encontrar un receptor. Si no lo hay, ejecutarlo localmente.
  - Probar en un nº de nodos y elegir aquel con la cola de planificación más corta.
- ***Política de información***: con las dos últimas la información se recoge cuando un nodo se convierte en emisor.
- ***Estabilidad***: inestable con alta carga ya que será difícil encontrar receptores y los muestreos consumen ciclos de CPU innecesarios.

Algoritmos iniciados por el receptor

- ***Política de transferencia***: umbral basado en la longitud de la cola de CPU.
- ***Política de selección***: cualquier proceso.
- ***Política de ubicación***:
  - Muestrear aleatoriamente un nº limitado de nodos hasta encontrar uno con un nivel de carga mayor que T+1.
  - Si la búsqueda falla, esperar hasta que otro proceso termine antes de intentar o esperar un periodo predeterminado.
- ***Política de información***: comienza cuando un nodo se convierte en receptor.
- ***Estabilidad***: estable. A altas cargas, es probable que los receptores encuentren emisores.

Algoritmos combinados

- ***Política de transferencia***



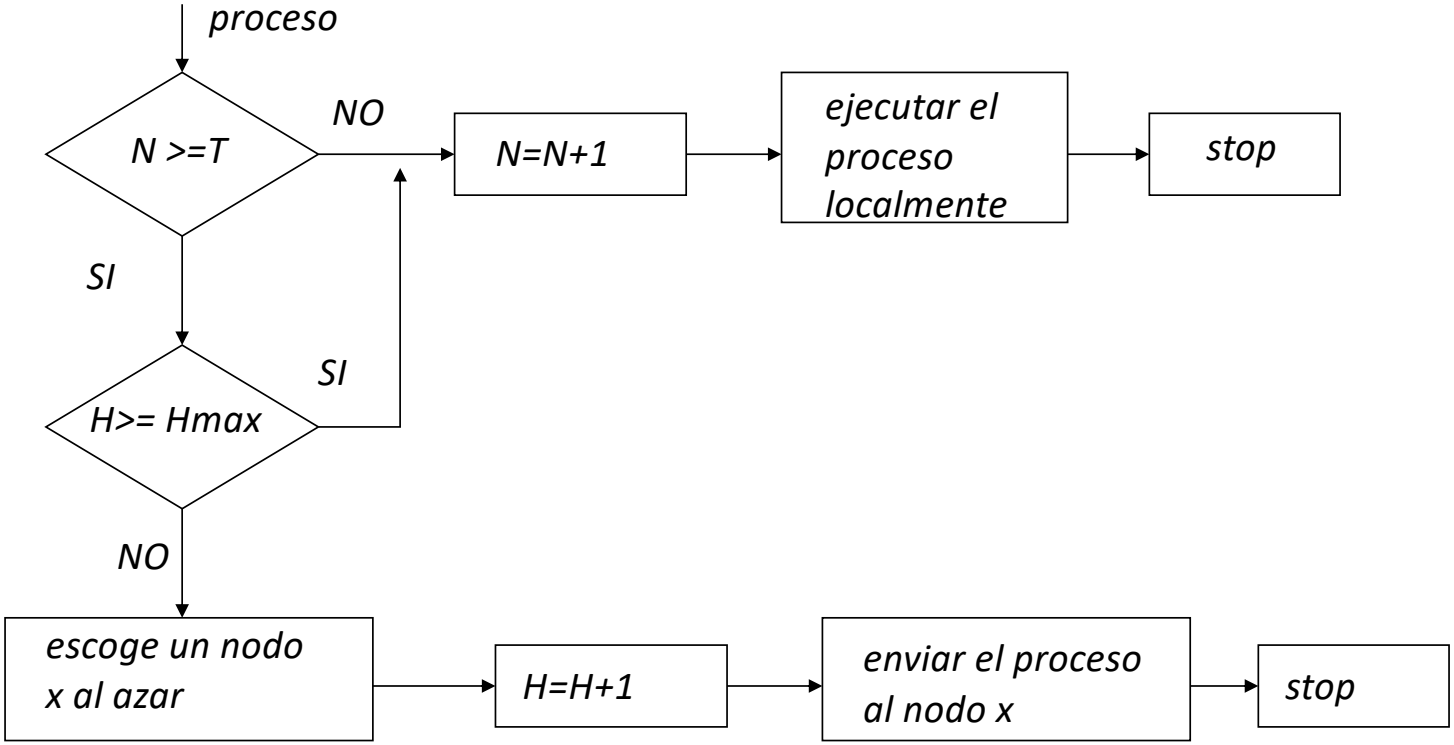
Política de ubicación dirigida por el emisor:

- El emisor difunde un mensaje *EMISOR* y espera *ACEPTAR*.
- Un receptor envía *ACEPTAR*.
- Cuando llega *ACEPTAR*: si el nodo es emisor, transfiere el proceso más adecuado.
- Si no llega ningún mensaje *ACEPTAR*, difundir un mensaje *CAMBIO-MEDIA* para incrementar la carga media estimada.

Algoritmos simétricos

- ***Política de ubicación iniciada por el receptor***:
  - Un receptor difunde un mensaje *RECEPTOR* y espera por mensajes *EMISOR*.
  - Si llega un mensaje *EMISOR*, se envía un mensaje *ACEPTAR*.
  - Si no llega ningún mensaje *EMISOR*, difundir un mensaje *CAMBIO-MEDIA* para decrementar la carga media estimada en el resto de nodos.
- ***Política de selección***: cualquier proceso.
- ***Política de información***:
  - Dirigida por demanda.
  - La carga media del sistema se determina localmente.

Ejemplo 1



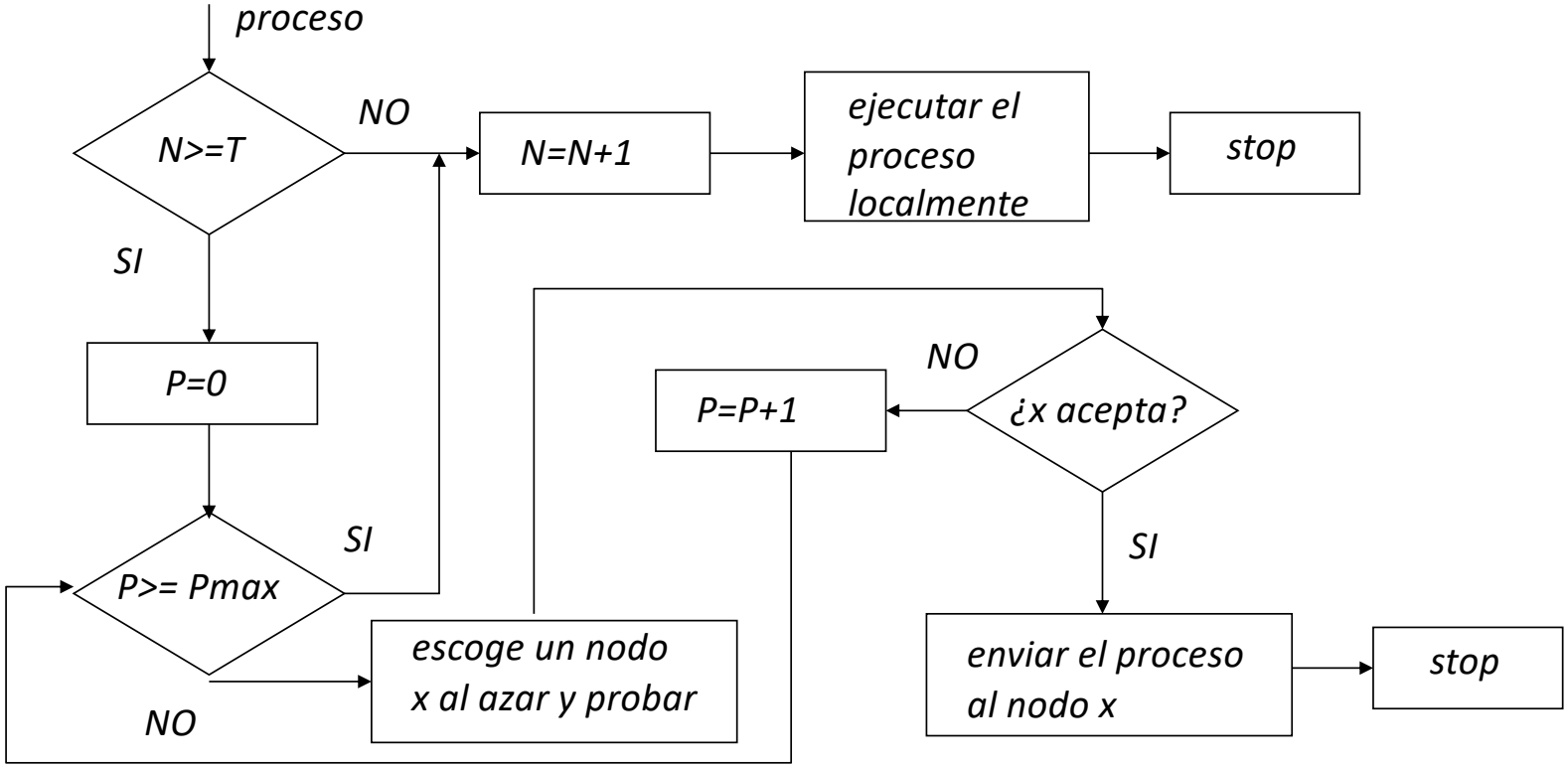
N: tamaño de la cola local; H: contador de saltos

T: tamaño umbral de la cola; Hmax: valor máximo del contador de saltos

Políticas del ejemplo 1

- ***Política de transferencia***: umbral de carga
- ***Política de selección***: cualquier proceso.
- ***Política de ubicación***: aleatoria.
- ***Política de información***: comienza cuando un nodo se convierte en emisor.

Ejemplo 2



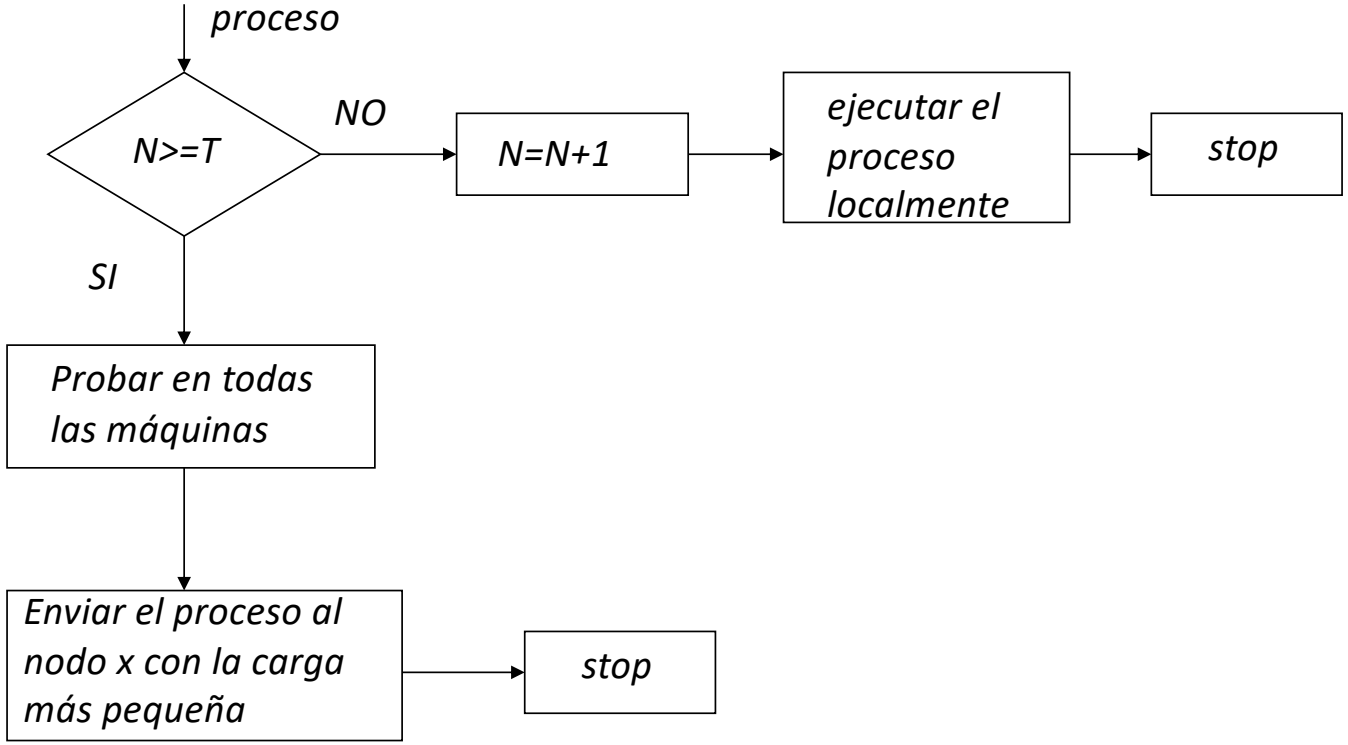
N: tamaño de la cola local; P: contador de prueba

T: tamaño umbral de la cola; Pmax: máximo valor de prueba

Políticas del ejemplo 2

- ***Política de transferencia***: umbral de carga.
- ***Política de selección***: cualquier proceso.
- ***Política de ubicación***: prueba aleatoria limitada.
- ***Política de información***: comienza cuando un nodo se convierte en emisor.

Ejemplo 3



N: tamaño de la cola local;
T: tamaño umbral de la cola;

Políticas del algoritmo 3

- ***Política de transferencia***: basada en umbral de carga.
- ***Política de selección***: cualquier proceso.
- ***Política de ubicación***: nodo con la carga más pequeña.
- ***Política de información***: comienza cuando un nodo se convierte en emisor.