Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Algoritmos de planificación de tiempo real en sistemas distribuidos heterogéneos considerando la migración de tareas

Tesis
Que para optar por el grado de
Doctor en Ciencias
(Computación)

Presenta:

José Ángel Hermosillo Gómez

Director de tesis:
Dr. Héctor Benítez Pérez
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Miembros del Comité Tutor:
Dr. Alberto Contreras Cristán
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
Dr. Javier Gómez Castellanos
Facultad de Ingeniería
Dr. Jorge Luis Ortega Arjona
Facultad de Ciencias
Dr. Víctor Rangel Licea
Facultad de Ingeniería

Ciudad Universitaria, Cd. Mx., enero 2020

Dedicado a las personas más importantes: mis padres,
Isabel Gómez Pérez y Rigoberto Hermosillo Ramírez, y
mis hermanos, Joel y Rigoberto.

"A veces me da por añorar la ciencia. La felicidad es un dueño tiránico, si no se está condicionado para aceptar incuestionablemente nada, salvo la verdad [...] Pero la verdad es una amenaza y la ciencia un peligro público. Tanto peligrosa como benéfica [...] Pero no podemos permitir que la ciencia destruya su propia obra, excelente obra..." Aldous Huxley

Agradecimientos

Indiscutiblemente, quiero agradecer a la Universidad Nacional Autónoma de México por haberme permitido la oportunidad de realizar mis estudios de doctorado, en específico al *Programa de Posgrado en Ciencia e Ingeniería de la Computación* (PCIC).

Agradezco al Consejo de Ciencia y Tecnología (*CONACyT*) por el apoyo económico brindado durante mi formación (CVU: 365413) y a los proyectos *PAPIIT IN104516* y *IN100813*.

Quiero agradecer a todas las personas del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas que hicieron posible mi estancia en ese increíble lugar, en particular al *Dr. Héctor Benítez Pérez* por su dedicación y compromiso ante la formación de nuevas generaciones de investigadores.

No quiero dejar pasar la oportunidad para agradecer al *Dr. Javier Gómez Castellanos* por su ardua labor como coordinador del *PCIC* y por dar seguimiento a cada uno de los que formamos parte de dicho programa; también, a la *Dra. Beatriz Aurora Garro Licón* por sus consejos e instrucción y, como olvidar, a todos los compañeros del "cubo" de control del IIMAS: *Nora, Alberto, Gerardo, Oriol, Eduardo* y todos aquellos que de alguna u otra forma me enseñaron, aconsejaron, orientaron y brindaron parte de su tiempo para ayudarme, gracias *Adrián*.

Le agradezco a mis padres por el apoyo brindado durante todo este tiempo y a mis hermanos.

Índice general

1. Introducción **1**
- 1.1. Problema . . . 2
- 1.2. Hipótesis . . . 3
- 1.3. Contribuciones . . . 4
- 1.4. Organización de este documento . . . 4

2. Antecedentes **5**
- 2.1. Retos en el análisis de sistemas de tiempo real . . . 5
- 2.2. Taxonomía de las plataformas multiprocesador . . . 11
  - 2.2.1. Basada en la rapidez de procesamiento . . . 11
  - 2.2.2. Basada en la migración de tareas entre procesadores . . . 11

3. Trabajo relacionado **15**
- 3.1. Revisión para plataformas idénticas . . . 15
  - 3.1.1. Planificación online . . . 16
  - 3.1.2. Aumento de recursos . . . 19
  - 3.1.3. Planificación sin migración . . . 20
  - 3.1.4. Planificación con migración . . . 21
- 3.2. Revisión para plataformas heterogéneas uniformes . . . 23
- 3.3. Planificación basada en métodos heurísticos . . . 24
  - 3.3.1. Shifting bottleneck scheduling para job-shops . . . 25
  - 3.3.2. Redes Bayesianas . . . 27

4. Planificación determinista **29**
- 4.1. Trabajo relacionado . . . 31
- 4.2. Descripción del problema . . . 31
- 4.3. Transformaciones . . . 32
- 4.4. Estados de las tareas . . . 35
- 4.5. Diseño del servidor . . . 37
- 4.6. Interrupciones . . . 38
- 4.7. Teorema . . . 40
- 4.8. Conclusión . . . 41

5. Planificación no determinista **43**
- 5.1. El problema . . . 44
- 5.2. Mapeo del problema a la red bayesiana . . . 46

5.2.1. Nodos: Tareas e intervalos ociosos 46
5.2.2. Relaciones 49
5.2.3. Especificación de la plataforma 50
5.3. Descripción de la estrategia 50
5.3.1. $\mathcal{A}$-*modificado* 51
5.4. Función objetivo 54
5.5. Cotas 55
5.5.1. Número de posibles instancias 55

6. Resultados **57**
6.1. Caso de estudio: RUN-modificado 57
6.2. Experimentación 58
6.2.1. Ejemplificación 58
6.2.2. Resultados generales 61
6.3. Conclusión 64

7. Conclusiones **67**
7.1. Sobre la estrategia basada en redes bayesianas 67
7.2. Sobre la estrategia basada en el análisis del WCET 70
7.3. Trabajo a futuro 70

A. Propagación en redes probabilísticas **73**
A.1. Probabilidad condicional 73
A.1.1. Propagación sobre redes probabilísticas 74
A.2. Redes Bayesianas 75
A.2.1. Algoritmo de Pearl: propagación sobre poliárboles 76

Índice de tablas

6.1. Asignación de los jobs de las diez tareas a lo largo de la plataforma 60
6.2. Frecuencias y promedio de las frecuencias (última columna) de ejecución completa de grupos de tareas 62
6.3. Tiempo ocioso promedio de cada instancia 64
6.4. Tiempo aprovechado promedio, tiempo máximo aprovechado y tiempo mínimo aprovechado obtenido en cada instancia 64
6.5. Valor promedio de la función objetivo y tasa de aprovechamiento promedio (tiempo aprovechado promedio/tiempo ocioso promedio) 65

Índice de figuras

2.1. Tarea de tipo gráfica acíclica dirigida (*DAG task*) 8
2.2. Taxonomía de las políticas de planificación. 9
2.3. Esquema de planificación basado en migración 11

3.1. Dos posibilidades para la ejecución de τ_3 en el intervalo $[0, 2)$ 17
3.2. Planificación para tareas periódicas (imagen tomada de [1]) 19

4.1. Estados de los jobs en tiempo de ejecución 35
4.2. Diferentes subintervalos en los que se divide el intervalo $[t, t^+)$: x_1, x_2 y ϕ . 38
4.3. Correspondencia entre un job-shop (abajo) y su tarea pseudoperiódica asociada (arriba) 41

5.1. Variables aleatorias que describen al job $\tau_{i,j}$ 47
5.2. Parámetros que conforman al modelo de tareas 47
5.3. Obtención del valor de la variable aleatoria $I_{k,l}$ 48
5.4. Tres tipos de relaciones que se pueden establecer con el job $\tau_{i,j}$ 49
5.5. Configuración permitida en la red bayesiana. A la izquierda se muestra la ejecución de jobs en $I_{k,l}$; a la derecha, su representación en la red. 49
5.6. Diagrama de flujo del proceso de inferencia sobre la red 50

6.1. Procesador *vs.* tiempo: ejemplo de una posible salida producida usando inferencia mediante redes bayesianas. La zona de ocio se encuentra coloreada en negro (intervalo de tiempo dejado por RUN). Las tareas τ_2, τ_5, τ_7 y τ_9 son *job-shops* compuestos por al menos dos jobs (imagen tomada de [2]) . . 59
6.2. Grafo asociado (imagen tomada de [2]) 60
6.3. Frecuencia *vs.* número total de jobs servidos por ejecución (imagen tomada de [2]) 60
6.4. Frecuencia *vs.* número total de tareas completas (job-shops) servidos por ejecución (imagen tomada de [2]) 61
6.5. Frecuencia de ejecución de tareas completas 63
6.6. Promedio de las frecuencias del número de tareas ejecutadas 65

7.1. Posible extensión de la estrategia basada en redes bayesianas para reducir el tiempo e implementarlas a lo largo del ciclo de vida del sistema 69

A.1. Graph with four random variables 76

Capítulo 1

Introducción

Contents

1.1. Problema **2**

1.2. Hipótesis **3**

1.3. Contribuciones **4**

1.4. Organización de este documento **4**

Actualmente, existe una gran variedad de aplicaciones en las que los sistemas de tiempo real están inmersos: sistemas embebidos en celulares, sistemas de manufactura, control en centrales nucleares y sistemas en los que la seguridad es crucial como en la aviónica, por mencionar algunas. Cada una de estas aplicaciones tiene requerimientos temporales específicos y, en menor o mayor medida, el incumplimiento de las restricciones temporales en cada aplicación implicará consecuencias serias. Por ejemplo, el incumplimiento de los requerimientos temporales en centrales nucleares podría causar sobrecalentamiento de algún reactor o, incluso, causar serias consecuencias debidas a emisiones de material radioactivo al medio ambiente [3].

Los sistemas de tiempo real se caracterizan por llevar a cabo un conjunto de actividades o manejar eventos dentro de cierto intervalo de tiempo que depende de la aplicación diseñada. Las aplicaciones de tiempo real son medidas usando tiempo *"real"* de reloj (*wall-clock*) en lugar de utilizar alguna otra forma de medición interna como lo son los ciclos de reloj, esto es debido a que los sistemas de tiempo real deben mantener exactitud temporal, es decir, la salida de la aplicación debe ser generada dentro del intervalo de tiempo que le fue designado en el diseño; además, se debe mantener la exactitud *lógica* de la aplicación que consiste en generar una salida correcta.

Los sistemas de tiempo real están conformados por instancias de tareas, a las que se referirá desde ahora como jobs, y una plataforma. Un *job* es un conjunto finito de instrucciones de código que se debe ejecutar dentro de una cantidad acotada de tiempo, mientras que la plataforma es el procesador o los procesadores sobre los que se ejecutaran o llevaran a cabo los jobs.

A pesar de la heterogeneidad (dada en términos de la tasa de rapidez de cómputo) de

los sistemas actuales, los esfuerzos se han enfocado en el desarrollo de estrategias de planificación sobre plataformas conformadas por procesadores idénticos (plataformas homogéneas). Uno de los trabajos recientes más destacados es el algoritmo RUN [4], ya que éste, a diferencia de otros algoritmos óptimos, hace una asignación *offline* de tareas periódicas a una plataforma homogénea de tal forma que el número de apropiaciones[1], de ahora en adelante se referirá a este termino como preemptions, por job está acotado superiormente[2].

1.1. Problema

La planificación tiene un impacto directo sobre la eficiencia de los sistemas computacionales (e inclusive en otros ámbitos como el de producción, manufactura, logística, por mencionar algunos ejemplos), por lo que este tema se ha convertido en un foco de atención para la investigación.

Los enfoques tradicionales de planificación son deterministas, es decir, suponen que los eventos que ocurren dentro de un sistema están necesariamente definidos por sus condiciones iniciales. Dichos enfoques de análisis y modelado tienen que ver, en su mayoría, con condiciones ideales como tareas periódicas de tipo preemptive planificadas sobre plataformas homogéneas. En poco más de dos décadas atrás han sido desarrollados varios algoritmos óptimos de tiempo real, los cuales son capaces de encontrar un acomodo (*schedule*) válido de tareas (siempre que uno exista) sobre plataformas homogéneas con múltiples procesadores, capaces de ocupar la totalidad del tiempo de cómputo de la plataforma: Proportionate fairness scheduling [6] (*p-fair*), *BF* [7], *EKG*[3] [8], *LLREF* [9], *DP-Wrap* [10] y, más recientemente, RUN [4]. A partir de las dificultades exhibidas por *John Stankovik* [11] se puede argumentar que las exigencias modernas no pueden estar tan limitadas (modelar los problemas usando pocos parámetros); lo que da lugar a la implementación de estrategias poco convencionales.

Sin embargo, la mayoría de los trabajos actuales se enfocan en garantizar la planificabilidad de ciertos conjuntos de tareas haciendo uso del análisis del peor caso de ejecución (en inglés, Worst Case Execution Time, abreviando WCET). Si bien este tipo de análisis garantiza la ejecución de dichas tareas dentro de sus plazos, son análisis de suficiencia, lo que implica que siempre habrá conjuntos de tareas que serán rechazados por dichos análisis cuando, en realidad, estos si pueden ser ejecutados. Además, dependiendo del número de parámetros considerados para modelar el sistema (por ejemplo, el tipo de plataforma, tipo de tareas, etc.), puede llegar a ser altamente intratable obtener un análisis de planificabilidad.

[1] El término *preemption* es utilizado para referirse al mecanismo que suspende a una tarea en ejecución y que invoca a un algoritmo de planificación que elige a la siguiente tarea que será ejecutada [5].

[2] La cota superior es de a lo más $\lg_2 m$, donde m es el número de procesadores que constituyen a la plataforma, lo que representa un resultado bastante útil al momento de considerar aspectos como, por ejemplo, los *overheads* causados por los cambios de contexto en la migración de tareas entre procesadores. De suponer constantes a dichos overheads, al tiempo de ejecución de un job se le sumaría el valor de los costos de los cambios de contexto multiplicados por dicha cota superior.

[3] EDF with task splitting and k processors in a group.

El principal problema de modelar instancias de problemas de tiempo-real considerando escenarios pesimistas (utilizar el WCET de las tareas) es que se ha probado que, en promedio, las tareas se ejecutan en menos tiempo que el reservado [12], [13], [14]; lo que conlleva a que los recursos de un sistema computacional sean subutilizados. Una forma de lidiar con dicho problema es mediante una estrategia capaz de aprovechar los posibles intervalos ociosos de longitud desconocida que surjan durante la ejecución de un conjunto de tareas.

Una manera de aprovechar dichos intervalos ociosos es a través de la inclusión de tareas nuevas; el origen de dichas tareas se puede explicar a partir de los esquemas actuales de los sistemas, cuyas necesidades de responder ante eventos o peticiones externas están presentes.

Para este trabajo, se considera que las tareas de tipo *job-shop* son las que modelan de una forma mejor aproximada a los eventos nuevos que pueden surgir durante la planificación. Por todo lo anteriormente mencionado se propone una estrategia que trabaja bajo un enfoque no determinista; la cual consiste en la aplicación de una técnica estadística de aprendizaje (en específico, redes bayesianas).

Cuando se amplía un problema de planificación, tanto los algoritmos deterministas como los métodos de optimización existentes se enfrentan a grandes retos en términos de estabilidad computacional y de tiempo. Ahora, en el entorno de la Industria 4.0, la programación debe tratar con sistemas de fabricación inteligentes apoyados por tecnologías de fabricación novedosas y emergentes [15].

Ampliar un problema de planificación sobre una plataforma multiprocesador resulta en un problema bastante complejo, si además se considera que dicha plataforma es heterogénea el problema se dificulta aún más. Sin embargo, se propone una posible solución que, desde un enfoque determinista, intenta planificar job-shops completos con un único plazo (todos los jobs que conforman a la tarea se llevan a cabo antes de dicho plazo) sobre una plataforma heterogénea mediante la extensión del trabajo de *Baruah* [16].

1.2. Hipótesis

Las hipótesis de este trabajo se enuncian a continuación:

- Dado un acomodo de tareas válido, cuyos peores tiempos de ejecución son considerados, es posible planificar bajo cierta probabilidad nuevas tareas aperiódicas no críticas de tipo *job-shop* con comportamiento estocástico sobre los procesadores de un sistema distribuido homogéneo, mediante la inferencia basada en redes Bayesianas formadas durante la ejecución de un algoritmo factible.
- Existe un *test* de planificabilidad, bajo un enfoque determinista y considerando el peor tiempo de ejecución, para asignar nuevas tareas no críticas de tipo *job-shop* aperiódicas sobre una plataforma heterogénea de tal manera que se cumplen sus requerimientos temporales.

1.3. Contribuciones

Los principales objetivos de este trabajo son, por un lado, el desarrollo de una posible mejora para algoritmos de planificación *offline* sobre una plataforma homogénea, dicha mejora utiliza una política de planificación que se basa en el uso de una distribución de probabilidad conjunta, cuyos parámetros se aprenden mediante información previamente recabada y; por el otro, el desarrollo de una estrategia de planificación sobre una plataforma heterogénea cuyo análisis de planificabilidad se efectúe *online* y garantice la ejecución de nuevos *job-shops* previamente a su ejecución y descartar aquellas tareas que, por el contrario, no aprueben dicho análisis. Ambas cubren las siguientes contribuciones:

- Para el caso de la estrategia sobre plataformas homogéneas
    1. Abstracción y reducción del problema de planificación de tareas esporádicas con relaciones de precedencia (*job-shops*) a una representación gráfica (Red Bayesiana).
    2. Definición de un análisis de planificabilidad que permita seleccionar aquellas tareas que tienen posibilidad de ser ejecutadas dentro de sus deadlines.
- Para el caso de la estrategia sobre plataformas heterogéneas
    1. Transformación de una plataforma heterogénea a una homogénea.
    2. Definición de una estrategia de planificación que permita ejecutar tareas *non-preemptive* con relaciones de precedencia (*job-shops*) sobre una plataforma homogénea de tal forma que se garanticen sus deadlines una vez aceptadas.

1.4. Organización de este documento

El resto de esta tesis está organizado de la siguiente forma. El capítulo 3 muestra una revisión bibliográfica del estado del arte con respecto a estrategias de planificación sobre plataformas constituidas por múltiples procesadores. En el capítulo 4 se propone un análisis de planificación determinista para verificar si es posible ejecutar un nuevo job-shop sobre una plataforma heterogénea. En el capítulo 5 se propone una estrategia heurística basada en el uso de redes bayesianas para lograr un mejor aprovechamiento del uso de una plataforma homogénea. En el capítulo 6 se presenta un caso de estudio que consiste en la aplicación de la estrategia descrita en el capítulo 5 sobre el algoritmo [4]. Finalmente, el capítulo 7 muestra algunas conclusiones finales.

Capítulo 2

Antecedentes

Contents

2.1. Retos en el análisis de sistemas de tiempo real 5

2.2. Taxonomía de las plataformas multiprocesador 11

2.2.1. Basada en la rapidez de procesamiento 11

2.2.2. Basada en la migración de tareas entre procesadores 11

2.1. Retos en el análisis de sistemas de tiempo real

En [11] se vislumbran una serie de características (o dimensiones, como son nombradas originalmente) que siguen siendo todavía vigentes al momento de diseñar un sistema de tiempo real:

1. **Granularidad de los plazos y laxitud de las tareas**. La *laxitud* se refiere al tiempo que queda de la diferencia entre el intervalo definido por el tiempo de activación de una tarea hasta el instante en el que debe terminar su ejecución y su tiempo de cómputo; mientras que, la *granularidad* se refiere a la cercanía del plazo de una tarea, cuánto más cercano a cumplirse la granularidad disminuye. Lo que implica que el tiempo de reacción del sistema operativo tenga que ser corto.

2. **La severidad de los plazos**. Se refiere a la ganancia (definida por alguna función objetivo) que consigue una tarea que se ejecuta después de su plazo.

   a) Para las tareas con plazos *hard* (*hard dealines*) no hay valor alguno cuando la tarea se ejecuta después de su plazo.

   b) Una tarea con un plazo *soft* (*soft deadlines*) todavía consigue una ganancia, aunque disminuida, después de que ocurre su plazo.

3. La **confiabilidad** tiene que ver con la construcción de sistemas de tal manera que los requerimientos puedan ser garantizados desde el diseño. Esto significa que un análisis *off-line* debería demostrar la satisfacción de los requerimientos temporales,

sujetos a las suposiciones hechas en condiciones operativas previstas para el sistema. Muchos de los sistemas de tiempo real operan bajo los más severos casos de *confiabilidad*, es decir, se supone que las tareas tienen deadlines (plazos) críticos que siempre satisfarán. Bajo dicho supuesto, usualmente para satisfacer los requerimientos temporales es necesario realizar un análisis *offline* o utilizar esquemas que reservan recursos (procesadores, memoria, bus de comunicación, etc.) para las tareas, aún cuando eso significa mantener a los procesadores ociosos la mayor parte del tiempo. Un error común es el de considerar todas las tareas con plazos de tipo *hard* como tareas críticas (cuando realmente sólo algunas de ellas lo son), lo que resulta en sistemas con requerimientos erróneos y un diseño sobredimensionado y un sistema inflexible. También se suele confundir a las tareas de plazos *hard* como tareas que tienen tanto plazos críticos como tareas con importancia crítica, se debe tener una clara noción ya que estos términos no están siempre relacionados.

4. El **tamaño** del sistema y el grado de **interacción** (coordinación) entre sus componentes se refieren a la cantidad de memoria que el sistema requiere para su correcto funcionamiento y a las interacciones entre los componentes del sistema. En la mayoría de los sistemas de tiempo real, el sistema completo es cargado en memoria o, en ocasiones, sólo una parte de éste se carga en memoria (de acuerdo a como haya sido diseñado el sistema), es decir, existe un uso muy frecuente del recurso (memoria) y la complejidad aumenta según la cantidad de subsistemas de los que estén compuestos (entre más subsistemas haya, mayor será la complejidad para desarrollar políticas para mantener en memoria sólo aquellos subsistemas requeridos en un instante de tiempo). En muchas aplicaciones, muchos subsistemas que las conforman son considerados como altamente independientes uno de otro y por lo tanto hay muy poca o nula cooperación(comunicación) entre las tareas (nótese que entre mayor sea la interacción entre tareas mayor será la complejidad del algoritmo de planificación). La suposición de cargar los sistemas completos en memoria y limitar las interacciones entre tareas simplifica muchos aspectos en la construcción y el análisis de sistemas de tiempo real. Sin embargo, los requerimientos de los sistemas actuales implican alto grado de interacción y recursos limitados (lo que implica desarrollo de políticas eficientes para gestionar los recursos de un sistema).

5. Las características del **ambiente** en el que el sistema opera juegan un rol importante en el diseño de sistemas de tiempo real. Muchos ambientes están muy bien definidos (tales como los experimentos de laboratorio, el motor de un automóvil, una linea de ensamble, etc.). Los diseñadores hacen la suposición de que estos ambientes son deterministas (en caso de no ser así, los sistemas son forzados a serlo). Estos ambientes propician el surgimiento de sistemas pequeños y estáticos, dónde todos los plazos pueden ser garantizados a priori. Aún en estos ambientes simples es necesario definir restricciones en las entradas. Por ejemplo, un conjunto de tareas periódicas asignado a un procesador, las cuales tienen una utilización total de uno, si se asigna una tarea más, ésta no cumplirá con su plazo. Se puede considerar que el sistema es predecible una vez que se sabe que esperar dadas las restricciones sobre el ambiente bien definido.

   El problema surge cuando las estrategias desarrolladas para sistemas relativamente

pequeños y estáticos no escalan a otros ambientes que son más grandes, más complicados y menos controlables. Los sistemas actuales son más grandes, complejos, distribuidos y adaptables, contienen muchos tipos de restricciones temporales, necesitan trabajar bajo ambientes no deterministas y evolucionan a lo largo de un largo ciclo de vida del sistema. Por lo que, es mucho más difícil forzar este ambiente a parecer determinista y, por lo tanto, será más difícil definir la predictibilidad de estos sistemas.

Las características previamente descritas solo consideran algunos aspectos que, si bien son relevantes, no consideran del todo los aspectos requeridos en los sistemas actuales (multiprocesador, distribuidos, etc.), por lo que se han añadido las características que se enumeran a continuación (note el carácter no determinista de las definiciones dadas para las tareas periódicas, aperiódicas y esporádicas).

6. El **tipo de tareas** que conforman al sistema. La gran parte de la literatura acerca del tema se ha enfocado a desarrollar análisis para determinar la predictibilidad de sistemas relativamente simples, esto es, consideran un sólo tipo de tareas (mayormente periódicas); sin embargo, los sistemas actuales son capaces de trabajar con una diversidad amplia de tareas. En esa diversidad de tareas se encuentran:

   a) Tareas *aperiódicas* y *esporádicas*: según [17], cada tarea de este tipo puede ser vista como un "flujo" de jobs aperiódicos y esporádicos, respectivamente, tales que los jobs de cada tarea son similares en el sentido de que tienen el mismo comportamiento estadístico y el mismo requerimiento de tiempo. Sus tiempos entre llegadas (*interarrival times*) pueden modelarse con variables aleatorias idénticamente distribuidas con alguna distribución de probabilidad $A(x)$. De forma similar, los tiempos de ejecución en cada tarea aperiódica (o esporádica) son variables aleatorias idénticamente distribuidas, cada una distribuida de acuerdo a la función de distribución de probabilidad $B(x)$. Estas suposiciones significan que el comportamiento estadístico del sistema y su ambiente no cambian con el tiempo, es decir, el sistema es estacionario. Además, los deadlines de los jobs en una tarea aperiódica son de tipo *soft*, mientras que para las tareas esporádicas, los deadlines son de tipo *hard*, en cuanto a sus tiempos de liberación (o llegada) no son conocidos a priori.

   b) Periódicas: según [17], cada cálculo o transmisión de datos que es ejecutado repetidamente o en intervalos de tiempo regulares o semiregulares, con el propósito de proveer de alguna función al sistema, es modelado como una tarea periódica.

   c) Tareas DAG (del inglés *Directed Acyclic Graph-task*, también llamadas *macro-data-flow graphs*) son tareas que son representadas mediante una gráfica acíclica dirigida, $G = (V, E)$, donde los vértices, V, representan jobs, $\tau_{i,j}$, donde $i \in \mathbb{N}$, $1 \leq j \leq n_i$ y $n_i \in \mathbb{N}$ y las aristas, $(\tau_{i,u}, \tau_{i,v}) \in E$, indican las precedencias, es decir, el job $\tau_{i,u}$ debe ser completado antes que el job $\tau_{i,v}$, dos jobs se dicen independientes cuando no se cumplen $(\tau_{i,u}, \tau_{i,v})$ ni $(\tau_{i,v}, \tau_{i,u})$. Un job con predecesores está *listo* para ejecución cuando el tiempo actual está en o después de su tiempo de liberación y todos los predecesores han sido completados.

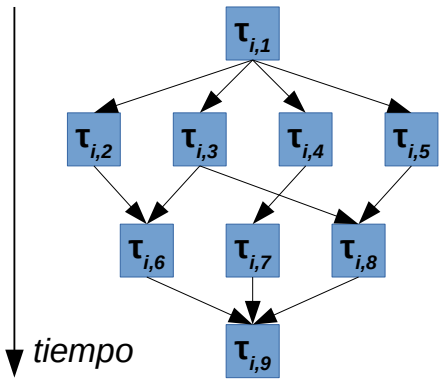


Figura 2.1: Tarea de tipo gráfica acíclica dirigida (*DAG task*)

Algunos casos particulares de este tipo de tareas se presentan a continuación. En una tarea DAG, el job sin ningún predecesor es llamado job de entrada y el job que no tiene sucesor alguno es llamado el job de salida. Note que en este tipo de tareas existen jobs que pueden ejecutarse de forma paralela; por ejemplo, vea los jobs $\tau_{i,2}$ a $\tau_{i,5}$ de la figura 2.1.

Dentro de las tareas DAG podemos encontrar a los job-shops como un tipo particular:

1) Los *job-shops* son tareas constituidas por una colección ordenada de jobs, sea así pues la tarea i, τ_i, con n_i jobs, son denotados así $\tau_{i,j}$, tal que $1 \leq j \leq n_i$ y las relaciones de precedencia entre jobs están definidas de la siguiente forma: para cuales quiera dos jobs $\tau_{i,k}$, $\tau_{i,j} \in \tau_i$, se cumple que si $j < k$, entonces el job $\tau_{i,j}$ debe concluir su ejecución antes que $\tau_{i,k}$. Cada job tiene su propio tiempo de ejecución $C_{i,j} \in \mathbb{N}$ y dicha colección sólo tiene un deadline D_i.

7. Tipo de **política** de planificación: La política o criterio de planificación elegido determina las restricciones y pautas para realizar el análisis de factibilidad de planificación (*schedulability analysis*) de un conjunto de tareas sobre una plataforma. Dicho análisis permite determinar si un sistema de tareas, con determinadas características, puede ser acomodado sobre un conjunto de procesadores de tal manera que todos los jobs cumplan con sus requerimientos temporales (*deadlines*).

   Las políticas de planificación *online* pueden basarse en algún criterio de proporcionalidad [6] (*proportional-fairness*) o en prioridades (*priority-driven-scheduling*), las cuales se subdividen en dinámicas o estáticas, ver la figura 2.2; por ejemplo, en el caso de los algoritmos basados en prioridades, se puede hacer la asignación de prioridades a las tareas basándose en sus deadlines, tiempos de cómputo, tiempos de liberación, etc.

   ca en el análisis de sistemas de tiempo real en los que se busca planificar tareas aperiódicas no criticas de tipo job-shop *non-preemptive* con deadlines sobre plataformas

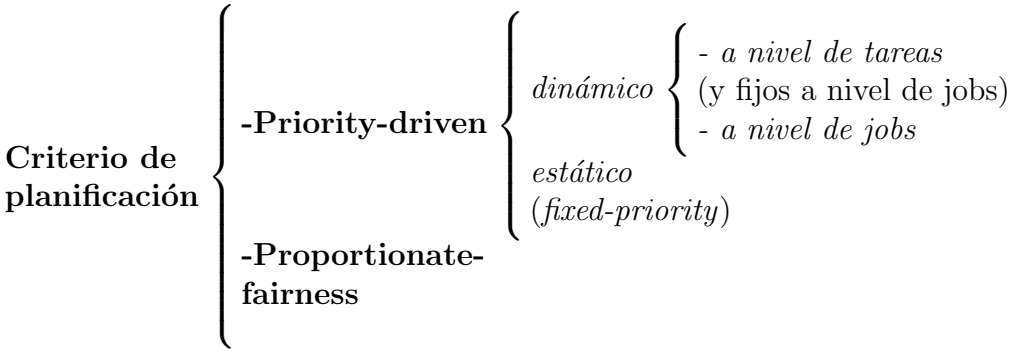


Figura 2.2: Taxonomía de las políticas de planificación.

multiprocesador desde dos perspectivas. La primera, desde un e

La planificación en tiempo de ejecución (*online* scheduling) es el *proceso* de determinar, durante la realización de una aplicación de tiempo real, que job se debería llevar a cabo sobre la plataforma a cada instante de tiempo. Los algoritmos de planificación *online* se implementan típicamente de la siguiente forma: en cada instante de tiempo, se le asigna una prioridad a cada job *activo*[1] y asigna algún procesador de la plataforma al job con la más alta prioridad.

En el caso de los algoritmos de planificación basados en prioridades (*priority-driven algorithms*) los tipos de políticas de planificación pueden clasificarse en:

a) **Dinámicas a nivel de tarea** (y estáticas a nivel de jobs): Un algoritmo se considera dinámico si la prioridad de una tarea puede cambiar durante el proceso de ejecución, sin embargo, a nivel de job las prioridades permanecen iguales. Dadas las tareas τ y τ' y ambas tareas tienen jobs activos en los instantes t_1 y t_2 (tal que $t_1 < t_2$); en el instante t_1, un job de τ tiene mayor prioridad que la del job de τ' y las prioridades se mantienen iguales para los jobs en el instante t_2. Un ejemplo de algoritmo que implementa este tipo de políticas es el algoritmo *EDF* [5] ya que es un planificador basado en deadlines (*deadline-based scheduling*).

b) **Estáticas a nivel de tarea**: Un algoritmo se considera como estático si éste especifica la prioridad de una tarea una sola vez y ésta no cambia en tiempo de ejecución. En otras palabras, los algoritmos de prioridad estática satisfacen la propiedad de que para cada par de tareas τ y τ', siempre que ambas tengan jobs activos, se dará el caso de que los jobs de las tareas tienen la misma prioridad. Un ejemplo de algoritmo que se basa en la elección de los jobs con base en prioridades fijas son *DM* [18] y *RM* [5].

c) **Dinámica a nivel de job**: Un algoritmo se considera dinámico a nivel de jobs ((fully dynamic algorithm)) si la prioridad de los jobs de una tarea cambian durante el proceso de ejecución. Es importante hacer notar que *EDF* no per-

[1]Informalmente hablando, se dice que un job llega a estar *activo* cuando éste se libera, y permanece así hasta que haya sido ejecutado por una cantidad equivalente a su tiempo de cómputo, o bien, hasta que su deadline haya expirado.

tenece a esta categoría[2]. Dadas las tareas τ y τ' y ambas tareas tienen jobs activos en los instantes t_1 y t_2; en el instante t_1, un job de τ tiene mayor prioridad que la de el job de τ'; mientras que en el instante t_2, el job de τ' tiene mayor prioridad que el de τ. Un ejemplo de algoritmo que emplea políticas de este tipo es *LLF* [19] (*laxity-based scheduling*).

8. El **tamaño de la plataforma** sobre la cual se planifican las tareas debe ser considerado para el análisis de tiempo, así como el **hardware subyacente** de la plataforma. Los procesadores modernos se basan en el uso de memorias cache, pipelines profundos (*deep pipelines*), *prefetching*, etc. para lograr un alto poder de cómputo. Todos estos rasgos complejos que implica el hardware y la complejidad inherente del software hacen de la dificultad de determinar el WCET para las tareas, de una forma eficiente y precisa, una labor mucho más complicada aún para un pequeño bloque de código [20]. Por ejemplo, predicciones erróneas menores del comportamiento del acceso a memoria cache puede ocasionar grandes desviaciones en el análisis para determinar el WCET de un programa, ya que el tiempo de ejecución de instrucciones individuales puede variar por muchos ordenes de magnitud dependiendo si el acceso a memoria cache fue un *hit* o un *miss*.

   La tendencia del hardware por desplazarse a las plataformas multicore y multiprocesador trae consigo un aumento en la complejidad. Mientras que la planificación de tareas en un procesador implica sólo una dimensión en tiempo, es decir, determinar cuándo ejecutar una cierta tarea, el problema se convierte en uno bidimensional en múltiples procesadores porque también se requiere decidir dónde (en que procesador) ejecutar la tarea. Además de los diferentes cores, las diferentes tareas contendiendo por otros recursos, tales como la memoria cache, los buses compartidos, entre otros. El uso concurrente de estos recursos compartidos crea un inmensurable espacio de estados para el comportamiento del sistema haciendo del análisis del tiempo una tarea muy difícil, si no imposible [21].

9. Cambios de contexto generados por la **migración** de tareas. En los sistemas actuales compuestos por plataformas con múltiples procesadores, trabajando sobre esquemas de planificación semiparticionados y globales (ver sección 2.2.2) es común encontrar varios componentes distribuidos e interconectados (pueden ser desde las tareas, hasta las unidades de procesamiento que conforman a una plataforma) trabajando juntos por lo que se vuelve necesario considerar las restricciones que impone la infraestructura de comunicación (por ejemplo, la velocidad a la que un mensaje se transmite, a veces, considerando la perdida de paquetes y otros aspectos implícitos en las comunicaciones). En cuanto a la migración de tareas, se deben considerar las restricciones que imponen las tecnologías o aplicaciones actuales que la permiten.

entonces, de las consideraciones previamente descritas, se puede tener diferentes diseños. Sin embargo, un aspecto común que siempre se busca cumplir es que el sistema sea *predecible* (*predictable*). Es decir, debe ser posible mostrar, demostrar o probar que los requerimientos se cumplen para cualesquiera que sean las suposiciones hechas[11], por ejemplo, si el sistema está sobrecargado o si ocurren fallas. Sin embargo, las estrategias

[2]Si bien, EDF mantiene variables las prioridades de las tareas, al nivel de jobs individuales, las prioridades se mantienen fijas.

actuales de análisis sólo logran la *predictibilidad* si consideran actividades lógicamente aisladas una de otra (*hardware partitioning*), por ejemplo, aislando a la memoria y a las comunicaciones de un sistema, es decir, suponer que las comunicaciones no dependen de la memoria y viceversa.

2.2. Taxonomía de las plataformas multiprocesador

Se presentan a continuación distintos tipos de taxonomías que se aplican a los sistemas de planificación de tareas en plataformas multiprocesador.

2.2.1. Basada en la rapidez de procesamiento

Basada en la rapidez de los procesadores, una plataforma multiprocesador puede clasificarse en alguna de las tres categorías que a continuación se listan [22].

Heterogéneas no relacionadas El término usado en inglés es *Unrelated Heterogeneous Multiprocessors*. En estas plataformas, la rapidez de procesamiento no sólo depende del procesador sino que también de la tarea que es ejecutada. Para cada tarea τ_i ($i = 1, \ldots$) asignada al procesador $Proc_k$ ($k = 1, \ldots, m$) de una plataforma heterogénea no relacionada tiene asociada una tasa de rapidez de ejecución $s_{k,i}$.

Heterogéneas uniformes En inglés se conoce como *Uniform Heterogeneous Multiprocessors*, también se referirá a este tipo de plataforma como *heterogénea* a lo largo de este trabajo. En estas plataformas, la rapidez de procesamiento de las tareas depende sólo del procesador. Específicamente, para cada procesador $Proc_k$, con $k = 1, \ldots, m$, y para cualesquiera dos procesadores $Proc_a$ y $Proc_b$ se tiene $s_a \neq s_b$. Se utilizará s_k para denotar la tasa de rapidez del procesador $Proc_k$ con $k = 1, \ldots, m$.

Procesadores idénticos En inglés se les conoce como *Identical Multiprocessors*, también se referirá a este tipo de plataforma como *homogénea* a lo largo de este trabajo. En estas plataformas, todas las tasas de rapidez de procesamiento son las mismas. Usualmente, en estos sistemas, las tasas de rapidez son normalizadas a una unidad de trabajo por unidad de tiempo (el término usado en inglés para referirse a éstos es *unit-capacity* processors).

Note que las plataformas heterogéneas no relacionadas pueden verse como una generalización de las plataformas heterogéneas uniformes.

2.2.2. Basada en la migración de tareas entre procesadores

Desde la perspectiva de la migración entre procesadores (*interprocessor migration*), las técnicas de planificación multiprocesador se pueden clasificar en los siguientes esquemas [22]:

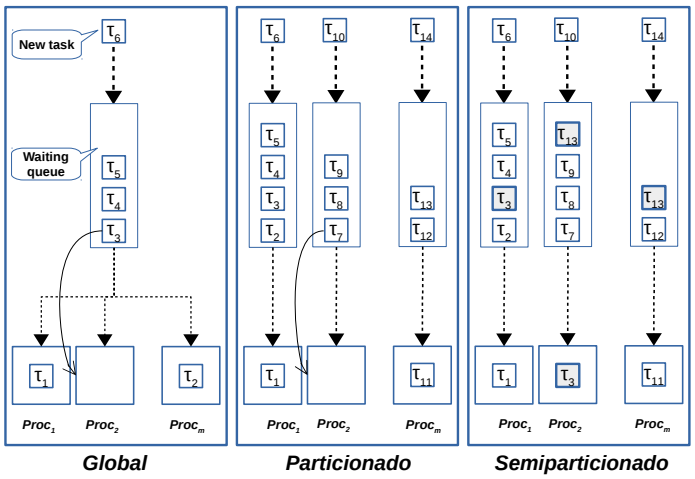


Figura 2.3: Esquema de planificación basado en migración

Esquema global En la planificación global, todas las tareas elegibles son almacenadas en una cola ordenada de acuerdo a su prioridad (asignada por algún criterio: deadlines, tiempos de llegada, prioridad dada por el usuario, etc.); luego, el planificador global elige los jobs de las tareas de más alta prioridad en dicha cola para ejecución. Ejemplos de algoritmos basados en este esquema son [6], [4]. Desafortunadamente, extender la aplicación de algoritmos óptimos para plataformas uniprocesador (RM, EDF, LLF, etc.) a plataformas multiprocesador resultaría en una utilización arbitrariamente baja [22].

Esquema particionado No hay migración, pues cada tarea es asignada a un sólo procesador, en el cual todos sus jobs se ejecutarán y no existe interacción entre los procesadores para la planificación (todos los jobs generados por una tarea se tienen que ejecutar sobre el procesador correspondiente). La principal ventaja de las estrategias basadas en este esquema es que se reduce el problema multiprocesador a varios de un sólo procesador (note que las políticas de planificación pueden ser distintas para cada procesador). Desafortunadamente, dividir las tareas es en sí mismo un problema: El *Bin-Packing problem* es reducible a encontrar una asignación óptima (en términos de la utilización de una plataforma multiprocesador) de tareas a procesadores [23], [24]; por lo que se trata de un problema $NP-hard$ en el sentido *estricto*. Por lo tanto, se puede optar por hacer esta asignación mediante métodos heurísticos. Otra desventaja surge cuando el sistema de tareas es planificable si y sólo si este no es partido.

Esquema semiparticionado En este esquema se asigna una tarea (generalmente de tipo recurrente) a cierto procesador; sin embargo, las diferentes instancias de ésta (jobs) pueden ser ejecutadas en otros procesadores disponibles. Es decir, la migración se da a nivel de jobs; es decir, el contexto de ejecución de cada job necesita ser mantenido en sólo un procesador; sin embargo, el contexto a nivel de tarea puede ser migrado. A este esquema también se le conoce como migración restringida. Un

ejemplo de algoritmo basado en una estrategia semiparticionada es [8] (ver sección 3.1.4).

Es importante resaltar que cuando los jobs son de tipo *preemptive* [5] se permite que estos sean interrumpidos por otros de mayor prioridad y que, posteriormente, retomen su ejecución en otro procesador distinto, si están en el esquema global o semiparticionado (lo que implica un aumento en los costos de migración del contexto a nivel de tarea o job, según el esquema elegido, entre procesadores) o en otro instante, si el esquema es particionado.

Anteriormente la migración había sido evitada por la comunidad de sistemas de tiempo-real (haciendo del esquema particionado el mejor acogido para su estudio) por las siguientes razones:

- En muchos sistemas, el costo asociado a transferir el contexto de un job de un procesador a otro podía ser muy elevado.
- La teoría carecía de técnicas, herramientas tecnológicas y resultados para permitir un análisis detallado de los sistemas en los que se consideraba la migración.

Los desarrollos recientes en arquitectura de computadoras y redes de comunicaciones cada vez más veloces han hecho que esta característica sea menos costosa y posible de implementar. Los beneficios de realizar la migración son variables y pueden ser comprobados en ciertas situaciones [25], [26]. El procesador desde el cual se migra el proceso se ve recompensado con una menor utilización de sus recursos, que a su vez, pueden ser utilizados por tareas con mayor prioridad o que son "más locales", es decir, tareas que son más relativas al sistema operativo en sí. De esta manera se logra tener un mejor rendimiento global que se basa en las eficiencias locales.

Capítulo 3

Trabajo relacionado

Contents

3.1. Revisión para plataformas idénticas 15

3.1.1. Planificación online 16

3.1.2. Aumento de recursos 19

3.1.3. Planificación sin migración 20

3.1.4. Planificación con migración 21

3.2. Revisión para plataformas heterogéneas uniformes 23

3.3. Planificación basada en métodos heurísticos 24

3.3.1. Shifting bottleneck scheduling para job-shops 25

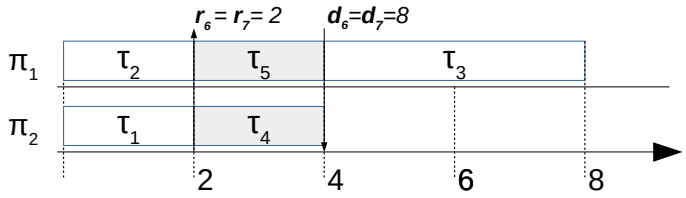
3.3.2. Redes Bayesianas 27

El surgimiento de nuevas arquitecturas con tendencias al multicore y multiprocesador y el aumento en sus capacidades: rapidez de procesamiento, rapidez de acceso a memoria y organización de la misma (niveles de cache, por ejemplo), aumento en la rapidez de los buses, etc.; así como el desarrollo de herramientas científicas y tecnológicas que ayudan a una mejor explotación de los recursos disponibles han propiciado el desarrollo de algoritmos y estrategias enfocadas a plataformas multiprocesador. Razón por la cual la implementación de sistemas de tiempo-real ha enfocado su atención en estas plataformas.

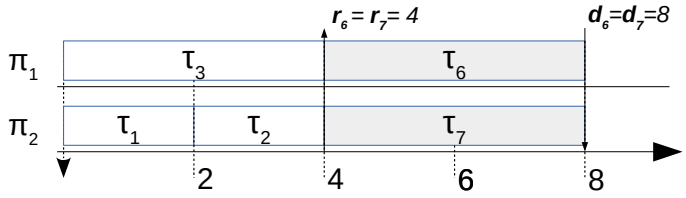
A lo largo de este capítulo se presentan algunos de los resultados más relevantes para la presente tesis, desarrollados para plataformas multiprocesador.

3.1. Revisión para plataformas idénticas

Existe una basta cantidad de trabajos en los que se muestran importantes aportaciones en el área de planificación sobre plataformas idénticas (homogéneas). A continuación sólo se reportan aquellas que dan “rumbo” al presente trabajo.



(a) τ_3 no se puede ejecutar en el intervalo $[0, 2)$



(b) τ_3 se tiene que ejecutar en el intervalo $[0, 2)$

Figura 3.1: Dos posibilidades para la ejecución de τ_3 en el intervalo $[0, 2)$

3.1.1. Planificación online

Uno de los resultados más relevantes en planificación multiprocesador en tiempo-real es el demostrado por Hong y Leung [27] y, por otro lado y de manera independiente, Mok y Dertouzos [1] que asegura lo siguiente: No puede haber algoritmo *online óptimo* para planificar tareas con requerimientos de tiempo (*real-time instances*[1] en inglés) sobre plataformas idénticas.

Teorema 1 ([27]) *No puede existir planificación óptima para instancias con dos o más deadlines distintos para cualquier plataforma idéntica de tamaño m, donde $m > 1$.*

Ejemplo 1 ([27]) *Suponga que existe un algoritmo online óptimo para dos procesadores y considere el siguiente escenario: En el instante 0, se liberan tres tareas τ_1, τ_2 y τ_3 con $d_1 = d_2 = 4$ y $d_3 = 8$, $C_1 = C_2 = 2$ y $C_3 = 4$. El algoritmo óptimo debería planificar estas tres tareas en dos procesadores, comenzando en el instante 0. Se consideran los dos siguientes casos dependiendo de si τ_3 se ejecuta o no en el intervalo $[0, 2)$.*

Caso 1: *τ_3 se ejecuta durante el intervalo $[0, 2)$.*

Para este caso se agregarán las tareas τ_4 y τ_5 con las siguientes características: Instante de liberación $r_4 = r_5 = 2$, tiempos de cómputo $C_4 = C_5 = 2$ y plazos $d_4 = d_5 = 4$ a las tareas ya existentes (τ_i con $i = 1, 2, 3$). Entonces alguna de las tareas incumplirá su deadline. En la figura 3.1 (a) se ilustra un acomodo válido de todas las tareas sobre dos procesadores idénticos. Note que los procesadores ejecutan las tareas τ_1, τ_2, τ_4 y τ_5 sin dejar intervalos ociosos durante el intervalo $[0, 4)$ (además, note que estas cuatro tareas

[1]Entiéndase por instancia a una tarea o, dado el caso, al job o jobs de una tarea.

tienen el mismo deadline en el instante 4). Por lo tanto, si τ_3 se ejecutase en cualquier momento dentro de $[0, 4)$ al menos una unidad de tiempo (porque el tiempo se considera discreto), causaría el incumplimiento del plazo de alguna de las otras tareas.

Caso 2: *τ_3 no se ejecuta durante el intervalo $[0, 2)$.*

Para este caso se agregarán las tareas τ_6 y τ_7 con las siguientes características: Instante de liberación $r_6 = r_7 = 4$, tiempos de cómputo $C_6 = C_7 = 4$ y plazos $d_6 = d_7 = 8$ a las tareas ya existentes (τ_i con $i = 1,\ 2,\ 3$). Entonces una de las tareas incumplirá su deadline. La figura 3.1 (b) muestra un acomodo válido de las tareas τ_i donde $i = 7,\ 6,\ 3,\ 2,\ 1$ sobre los dos procesadores idénticos. Note que los procesadores ejecutan las tareas $\tau_1,\ \tau_2,\ \tau_3$ dentro del intervalo $[0, 4)$ y las tres han concluido su ejecución con éxito (han terminado o antes o en el instante $t = 4$). Además, note que las tareas τ_6 y τ_7 requieren de cuatro unidades de tiempo de procesamiento dentro del intervalo $[4, 8)$. Si la tarea τ_3 no se ejecuta completamente en el intervalo $[0, 2)$, ésta no completaría su ejecución antes del instante $t = 4$. Por lo tanto, ésta requeriría utilizar tiempo de cómputo dentro del intervalo $[4, 8)$ y por lo menos alguna de las tareas τ_3, τ_6 o τ_7 incumplirá con su deadline.

Por lo tanto, las tareas τ_i con $i = 1,\ 2,\ 3$ no pueden ser planificadas de tal manera que se aseguren acomodos válidos para todas las tareas factibles sin el conocimiento previo de las tareas que se liberen después del instante $t = 2$.

En [1] también se establece que parámetros deben ser conocidos en las tareas aperiódicas[2] (tareas generales de tiempo-real) para planificarlas de manera óptima.

Teorema 2 ([1]) *Para dos o más procesadores, ningún algoritmo basado en prioridades basadas en deadlines puede ser óptimo sin un conocimiento previo de los 1) plazos, 2) tiempos de cómputo y 3) tiempos de inicio de las tareas.*

Mok y *Dertouzos* también identificaron condiciones bajo las cuales una planificación válida puede ser asegurada aún cuando una o más de estas propiedades no es conocida. Para instancias generales de tiempo-real, determinaron que si las instancias pueden ser planificadas de manera factible cuando son liberadas al mismo tiempo, entonces es posible planificarlas sin conocer sus tiempos de liberación aún cuando no se liberan simultáneamente.

Teorema 3 ([1]) *Sean $I = \tau_1,\ \tau_2,\ \ldots, \tau_n$ y $I' = \tau'_1,\ \tau'_2,\ \ldots, \tau'_n$ dos conjuntos de tareas que cumplen lo siguiente:*

- *Las tareas de I y de I' difieren solamente en sus tiempos de liberación: Los requerimientos de ejecución son iguales, $C_i = C'_i$, y tienen la misma cantidad de tiempo entre sus plazos y tiempos de liberación, $d_i - r_i = d'_i - r'_i$, para toda $i = 1,\ 2,\ \ldots,\ n$.*
- *Las tareas de I' se liberan al mismo tiempo $r_i = r_j$ para toda $i, j = 1\ , 2\ ,\ \ldots,\ n$.*
- *I' puede ser planificado de manera factible sobre m procesadores idénticos.*

Entonces I puede ser planificado para satisfacer todos los plazos aún cuando los tiempos de liberación no son conocidos de antemano. En particular, el algoritmo LLF planificará de manera exitosa a I en m procesadores idénticos.

[2]Este resultado no es aplicable a conjuntos de tareas periódicas o esporádicas.

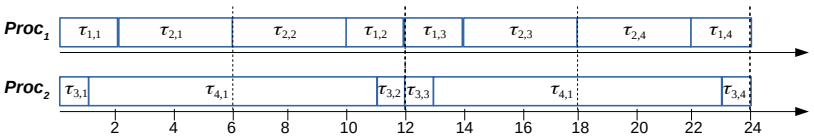


Figura 3.2: Planificación para tareas periódicas (imagen tomada de [1])

En el mismo trabajo [1] también se establece una condición suficiente para asegurar que un conjunto de tareas periódicas cumplirá todos sus plazos cuando las tareas son planificadas sobre m procesadores idénticos y las apropiaciones (preemptions) están permitidas en instantes enteros. Con el siguiente resultado 4 se establece que existe una planificación válida mediante el uso del concepto de rebanada de tiempo (*time-slicing*, en inglés). En esta estrategia, la planificación es generada en rebanadas de T unidades de tiempo. Dentro de cada rebanada, cada tarea τ_i recibe $T \cdot \frac{C_i}{T_i}$ unidades de ejecución. Además, en cada rebanada, cada tarea debe ejecutarse la misma cantidad de tiempo.

Teorema 4 ([1]) *Sea* $\tau = \{(C_1, T_1),\ (C_2, T_2),\ \ldots,\ (C_n, T_n)\}$ *un conjunto de tareas periódicas con* $u_{max} := \max_{i=1,\ldots,n}\{\frac{C_i}{T_i}\}$ *y* $U_{total} := \sum_{i=1}^{n} \frac{C_i}{T_i}$ *tales que* $u_{max} \leq 1$ *y* $U_{total} \leq m$. *A continuación, se definen* T *y* t*:*

$$T := gcd\{T_1,\ T_2,\ \ldots,\ T_n\}$$

$$t := gcd\{T,\ T \cdot \frac{C_1}{T_1},\ \ldots,\ T \cdot \frac{C_n}{T_n}\}$$

Si t *es un número entero,* $t \in \mathbb{Z}^+$*, entonces existe un acomodo válido para* τ *sobre* m *procesadores idénticos (unit-speed) con apropiaciones ocurriendo en instantes con valores enteros.*

Ejemplo 2 ([1]) *Sea el siguiente conjunto de tareas periódicas (tabla 2). Un posible acomodo de las instancias de las tareas se muestra en la figura 3.2*

Tarea	Consumo	Periodo
τ_1	2	6
τ_2	4	6
τ_3	2	12
τ_4	20	24

$$T = gcd\{6,\ 6,\ 12,\ 24\} = 6$$

$$t = gcd\{6,\ 6 \cdot \frac{2}{6},\ 6 \cdot \frac{4}{6},\ 6 \cdot \frac{2}{12},\ 6 \cdot \frac{20}{24}\}$$
$$= gcd\{6,\ 2,\ 4,\ 1,\ 5\} = 1$$

Ambos resultados [1] y [27] dan una idea muy clara de las limitaciones a las que se está sujeto al diseñar algoritmos *online* sobre plataformas multiprocesador idénticas.

En 1996, *Baruah* [6] propuso el primer algoritmo óptimo para sistemas de tareas periódicas sobre múltiples procesadores idénticos. Éste funciona migrando las tareas entre los procesadores, esta estrategia de tipo *p-fair* (forma abreviada de *proportionate progress*) puede encontrar de manera exitosa un acomodo (*schedule*) para cualquier conjunto de tareas tales que no excedan la capacidad de un procesador unitario (tareas con tasa de utilización a los más uno). Más tarde, este algoritmo fue modificado para atender tareas esporádicas [28] de tal forma que también se garantiza la optimalidad del mismo.

3.1.2. Aumento de recursos

El término *resource augmentation* (aumento de recursos) fue acuñado por *Phillips, et al.* [29] para describir a la técnica de añadir recursos para superar las limitaciones de la planificación *online*.

A pesar de la inexistencia de algoritmos *online* óptimos, en [30] (propuesto por *Kalyanasundaram, et al.*) se define una estrategia uniprocesador que puede ser extendida a plataformas idénticas, en la que por medio del incremento en la rapidez de los procesadores es posible hacer frente a los efectos del desconocimiento de la información concerniente a las tareas que son liberadas, a saber, deadline, requerimiento de ejecución y tiempo de liberación (de ellas sólo se sabe que son *preemptive*), para encontrar un acomodo factible online. La forma en la que se modera dicho incremento en la rapidez de los procesadores es a través de la tasa de competitividad (*competitive ratio*) trayendo consigo la mejora el desempeño del sistema.

La tasa de competitividad provee una medida para determinar que tan cerca está un algoritmo $\mathcal{A}$ del óptimo, $\mathcal{O}$. En [30] también se introdujo la rapidez de los procesadores en el análisis de competitividad. En lugar de comparar los algoritmos $\mathcal{A}$ y $\mathcal{O}$ sobre la misma plataforma, se permitió al algoritmo $\mathcal{A}$ ejecutarse en una plataforma compuesta por m procesadores con rapidez de s, mientras que el algoritmo $\mathcal{O}$ es ejecutado en una plataforma *unitaria* (todos los procesadores tienen la misma rapidez de procesamiento):

$$\max_{I} \frac{\mathcal{A}_s(I)}{\mathcal{O}_1(I)}$$

donde $s = 1+\varepsilon$ y $\varepsilon > 0$, $\mathcal{A}_s(I)$ denota el costo computacional de la planificación producida por el algoritmo *online* $\mathcal{A}$ sobre la entrada I y $\mathcal{O}_1(I)$ denota el costo de obtener la planificación óptima. Usando este concepto se probó que algunos algoritmos que tienen una tasa de competitividad pobre pueden mejorar cuando la rapidez es incrementada aún un poco.

La idea central en [30] es la siguiente: Si la entrada I (conjunto de tareas de tiempo-real independientes) es factiblemente planificada sobre m procesadores idénticos, entonces un algoritmo *online* satisfará todos los deadlines de las tareas en I sobre una plataforma con procesadores de rapidez s.

El siguiente resultado 5 ha permitido probar otros resultados importantes concernientes a algoritmos de planificación *online* en tiempo-real sobre plataformas idénticas. En particular, se probó que *cualquier instancia de tiempo-real que es factible sobre una plataforma de*

tamaño m de procesadores unitarios es ***global EDF-schedulable*** *si los m procesadores tienen una tasa de procesamiento de* $(2 - \frac{1}{m})$. Por lo tanto, el incremento de velocidad permite utilizar un algoritmo online aun cuando ninguno de este tipo es óptimo:

Teorema 5 ([29]) *Sea* π *una plataforma compuesta por m procesadores unitarios y sea* π' *otra plataforma compuesta por m procesadores de rapidez* $(2 - \frac{1}{m})$. *Sea* $\mathcal{O}$ *algún algoritmo de planificación multiprocesador y sea* $\mathcal{A}$ *algún algoritmo de planificación de tipo work-conserving. Por lo tanto, para cualquier conjunto tareas I y cualquier instante t,*

$$W(\mathcal{A}, \pi', I, t) \leq W(\mathcal{O}, \pi, I, t)$$

donde $W(\mathcal{A}, \pi, I, t)$ *es la cantidad de trabajo realizado por el algoritmo* $\mathcal{A}$ *sobre las tareas en I en el intervalo* $[0, t)$ *con* $t \geq 0$, *mientras se ejecutan en la plataforma* π.

Tiempo después se exploró en [31] la posibilidad de aumentar tanto la rapidez como el número de procesadores para encontrar una planificación factible:

Teorema 6 ([31]) *Sea I un conjunto de tareas con restricciones temporales. Si se supone que I es factiblemente planificable sobre m procesadores unitarios. Entonces, I satisfará todos sus deadlines cuando sea planificado sobre* $m + p$ *procesadores con rapidez de al menos* $2 - \frac{1+p}{m+p}$ *usando el algoritmo EDF global, donde* $p \in \mathbb{Z}^+$.

3.1.3. Planificación sin migración

Los resultados previamente presentados permiten la migración de tareas. En esta sección se presentará uno de los resultados más importantes para la planificación de tareas basada en esquemas particionados.

El trabajo propuesto por *López* [32] es uno de los más relevantes ya que explora la planificación sobre una plataforma idéntica basada en estrategias de partición de tareas y se determina una cota para la utilización con *EDF* como el algoritmo de planificación subyacente. Para hacer dicho análisis se utilizaron algoritmos heurísticos para realizar la asignación de las tareas entre los procesadores que conforman a la plataforma, a saber:

FF: Fisrt fit, que consiste en asignar una tarea al primer procesador que tenga el espacio suficiente para contenerla.

BF: Best fit, en el que una tarea es asignada al procesador con la menor cantidad de espacio disponible que exceda la utilización de la tarea.

RF: Random fit, en el que cada tarea puede ser asignada a algún procesador que tenga suficiente espacio para contenerla.

Otro resultado de [32] es que sin importar cual de estas formas de asignación se utilice, se obtiene la misma cota para la utilización. Esta cota de utilización está en función de $\lfloor \frac{1}{u_{max}(\tau)} \rfloor$, representado por β, que es el número más pequeño de tareas que pueden ser asignadas a un procesador antes de intentar asignar una tarea que sobrepase la capacidad de dicho procesador. El test ahí desarrollado considera la planificación de n tareas sobre m procesadores unitarios. Si $n \leq m \cdot \beta$, entonces cualquier asignación razonable funcionará.

Si $n > m \cdot \beta$, la cota para la utilización que garantiza la planificación utilizando *EDF particonado* es

$$U_{sum} \leq \frac{\beta \cdot m + 1}{\beta + 1}$$

Por medio de esta cota también es posible determinar el número mínimo de procesadores requeridos para planificar n tareas con valores dados de β y U_{sum}:

$$m \geq \begin{cases} 1 & \text{si } U_{sum} \leq 1 \\ \text{mín}\left\{\lceil \frac{n}{\beta} \rceil,\ \lceil \frac{(\beta+1)U_{sum}-1}{\beta} \rceil\right\} & \text{si } U_{sum} > 1 \end{cases}$$

3.1.4. Planificación con migración

Los algoritmos que a continuación se presentan resuelven el problema de planificar un conjunto de tareas con liberación periódica de jobs sobre plataformas multiprocesador con el propósito de cumplir los deadlines. Los procesadores que conforman a la plataforma son idénticos, en términos de la velocidad.

Algoritmo EKG

El algoritmo propuesto por *Björn Andersson* [8], denominado EDF con fraccionamiento de tareas y k procesadores en un grupo (EKG), es un algoritmo capaz de encontrar un acomodo óptimo de tareas periódicas sobre una plataforma homogénea. Las tareas son de tipo *preemptive* y tienen la capacidad de migrar en un subgrupo de procesadores (note que se trata de planificación sobre un esquema semiparticionado).

Este algoritmo consta de dos partes: asignación de tareas a procesadores y planificar las tareas sobre un procesador. El algoritmo asigna tareas a los procesadores de tal forma que las tareas no excedan la utilidad en cada procesador no exceda a 1. Éste cuenta con un parámetro k, tal que $1 \leq k \leq m$. El algoritmo diferencia las tareas *pesadas* y a las *ligeras* mediante un valor *separador*, *SEP*. Una tarea es pesada si $C_i/T_i > SEP$, de lo contrario se le denomina ligera. El valor de *SEP* se computa de la siguiente forma

$$SEP = \begin{cases} \frac{k}{k+1} & k < m \\ 1 & k = m \end{cases} \tag{3.1}$$

EKG presenta dos resultados que se enuncian a continuación:

Teorema 7 ([8]) *Si un conjunto de tareas periódicas* $\tau = \{\tau_i | (C_i, T_i) \forall i = 1, \ \ldots \ , \ n\}$ *satisface la condición*

$$\frac{1}{m} \cdot \sum_{i=1}^{n} \frac{C_i}{T_i} \leq SEP \tag{3.2}$$

y EKG es utilizado, entonces todos los deadlines son satisfechos y una tarea nunca se ejecuta en dos o más procesadores simultáneamente.

Teorema 8 ([8]) *Considérese el caso en el que todas las tareas son liberadas en el instante* 0 *y éstas son planificadas usando EKG. Se afirma que el número de preemptions durante* $[0, lcm)$ *(lcm es el mínimo común múltiplo de los periodos de las tareas) dividido por el número de jobs que fueron ejecutados durante* $[0, lcm)$ *es a lo más* k*.*

El algoritmo asigna primero $L \in \mathbb{N}$ tareas pesadas a L procesadores dedicados y el resto de los procesadores $(m - L)$ tiene que lidiar con las tareas ligeras. El algoritmo hace un barrido de cada tarea ligera $i = n - L + 1, \ \ldots, \ n$, al tratar de asignar la tarea i al procesador $p = m - L + 1, \ \ldots, \ m$ se verifica que la tarea i no exceda la utilidad del procesador p (≤ 1), si dicha condición es falsa, la tarea ligera se parte en dos porciones y se asignan al procesador p y $p + 1$. Con lo anterior se forman k grupos de tareas que no interactúan.

Después, las tareas se despachan a los procesadores, las tareas pesadas se ejecutan en sus respectivos procesadores dedicados y las tareas ligeras, por otro lado, son asignadas a grupos de procesadores y cuando una tarea es liberada en uno de esos grupos, los despachadores son activados en todos los procesadores de ese grupo. Los valores t_0 y t_l denotan el instante de liberación de una tarea en el grupo y el siguiente tiempo de llegada de cualquier otra tarea en ese mismo grupo, respectivamente. Denotemos a las dos subtareas de τ_i como τ_i' y τ_i'', ambas tienen el mismo periodo y se liberan al mismo tiempo. Cuando la subtarea llega a algún procesador, sea t_0 el tiempo de liberación y sea t_l el instante de la siguiente liberación de un job en algún procesador. La tarea τ_i' se ejecuta durante $(C_i/T_i) \cdot (t_l - t_0)$ unidades de tiempo de manera continua desde el instante t_0. La tarea τ_i'' es ejecutada durante $(C_i''/T_i'') \cdot (t_l - t_0)$ unidades de tiempo y termina su ejecución en t_l, las subtareas (τ_i' y τ_i'') se ejecutan en diferentes procesadores y sus ejecuciones no se traslapan en tiempo ya que $C_i/T_i \leq 1$ lo que es lo mismo que $C_i'/T_i' + C_i''/T_i'' \leq 1$. Estas subtareas se ejecutan con la más alta de las prioridades, mientras que el resto de las tareas se planifican de acuerdo a *EDF* con una menor prioridad.

Algoritmo RUN

El algoritmo denominado en inglés *Reduction to UNiprocessor* (RUN) propuesto por *Regnier*, et. al. [4] es un algoritmo óptimo para tareas periódicas e independientes sobre plataformas multiprocesador idénticas.

RUN produce un acomodo de tareas válido sobre una plataforma multiprocesador con pocas *preemptions* por job: el número de preemptions por job está acotado superiormente por $log_2(m)$, donde m es el número de procesadores que conforman a la plataforma. Dicha cota permite suponer un bajo costo por *overhead* a comparación de otros algoritmos óptimos ([6], [7], [8], [9], [9], [10]). RUN reduce el problema de planificación en tiempo-real sobre una plataforma multiprocesador a un conjunto equivalente de problemas sobre un procesador que es más fácil de resolver; dicha reducción la realiza por medio de las operaciones DUAL y PACK, las cuales son aplicadas *offline*.

La reducción de una plataforma multiprocesador a una uniprocesador se da gracias a la

agrupación de tareas a servidores. En este algoritmo los servidores son tratados como tareas conformadas por secuencias de jobs, sin embargo, éstos no son tareas dentro del sistema; un servidor es un *proxy* para una colección de tareas cliente.

Este algoritmo sólo se puede llevar a cabo sobre un conjunto de tareas periódicas cuya utilización total sea igual a m. En caso de no ser así se agregarán las tareas *dummy* (tareas de relleno o *no operation tasks*) en cada uno de los procesadores según sea necesario para lograr la utilización total del sistema.

3.2. Revisión para plataformas heterogéneas uniformes

La heterogeneidad puede definirse en términos varios de los aspectos que implican las nuevas arquitecturas de procesadores, entre los más importantes se encuentran la capacidad y rapidez de acceso a la memoria, así como sus diferentes niveles, tasa de rapidez del procesador, rapidez de transmisión de los buses o, dado el caso, la rapidez de la red de comunicaciones (en caso de tratarse de sistemas distribuidos), etc.

Para los propósitos de este trabajo, sólo se considera como plataforma heterogénea a aquella que está compuesta por procesadores cuyas tasas de rapidez son diferentes.

Uno de los trabajos que deben ser mencionados en esta sección es [33], ya que propone condiciones suficientes para determinar si es posible planificar un conjunto de tareas con restricciones temporales sobre una plataforma heterogénea utilizando *EDF* global, cabe mencionar que este trabajo toma como referencia el teorema 5 mencionado en la sección 3.1.2.

Otra característica por destacar de este análisis es que se basa enteramente en las tasas de rapidez de los procesadores y no de las características de las tareas.

Se rescatarán las siguientes definiciones de [33] para explicar el teorema 9.

Definición 1 (Plataforma uniforme heterogénea) *Sea π una plataforma uniforme heterogénea.*

- *El número de procesadores que comprende a π se denota por $m(\pi)$.*
- *Para toda i, $1 \leq i \leq m(\pi)$, la rapidez (la capacidad de cómputo) del procesador i más rápido es denotado por $s_i(\pi)$.*
- *La capacidad total de cómputo de todos los procesadores en π se denota por $S(\pi)$: $S(\pi) := \sum_{i=1}^{m(\pi)} s_i(\pi)$.*

La siguiente definición indica que tan parecidos son los procesadores de una plataforma heterogénea uniforme:

Definición 2 *El nivel de similitud de una plataforma heterogénea uniforme de tamaño*

$m(\pi)$ se medirá de la siguiente forma:

$$\lambda(\pi) := \max_{i=1,\ldots,m(\pi)} \left\{ \frac{\sum_{j=i+1}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} \tag{3.3}$$

note que si $\lambda(\pi) = m - 1$, entonces π estará conformada por m procesadores idénticos.

El teorema 9 determina las condiciones bajo las cuales *EDF* ejecutándose en la plataforma π_1 satisfará todos los deadlines de un conjunto de tareas que es factible sobre una plataforma π_0. La condición (3.4) en el teorema 9 expresa la capacidad de cómputo adicional que necesita la plataforma π_1 dada la capacidad de π_0.

Teorema 9 ([33]) *Sea I un conjunto de tareas que es factiblemente planificado sobre una plataforma uniforme π_0. Sea π_1 otra plataforma uniforme si la siguiente condición es satisfecha por las plataformas π_0 y π_1:*

$$S(\pi_1) \geq s_1(\pi_0) \cdot \lambda(\pi_1) + S(\pi_0), \tag{3.4}$$

donde s_1 denota la rapidez del procesador más rápido. Entonces I satisfará todos los deadlines cuando se planifique utilizando el algoritmo EDF ejecutándose sobre la plataforma π_1.

3.3. Planificación basada en métodos heurísticos

Cuando se abordan problemas cuyo espacio de posibles soluciones se puede conocer, la búsqueda se reduce a hallar el óptimo (un máximo o mínimo) que da solución a un determinado problema dado un conjunto de restricciones. Sin embargo, existen problemas de optimización combinatoria complejos en diversos campos de la ingeniería, la economía, las comunicaciones, el comercio, la industria, la medicina y, en particular, en los sistemas de tiempo-real, que a menudo son difíciles de resolver y cuyo modelado matemático coherente[3], no permite una solución con las herramientas analíticas al alcance. Por lo tanto, se hace necesario el desarrollo y uso de métodos que permitan el manejo de situaciones problemáticas desde diferentes puntos de vista. Debido al bajo rendimiento de los algoritmos exactos para muchos problemas, se han desarrollado algoritmos aproximados, que proporcionan soluciones de alta calidad para estos problemas combinatorios (aunque no necesariamente la solución óptima) en un tiempo computacional aceptable; estos algoritmos incluyen las denominadas técnicas heurísticas [34].

Definición 3 (Heurística [34]) *Se califica de **heurístico** a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad o su factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación. Se utiliza el calificativo heurístico en contraposición a exacto.*

[3]Se considera que un problema ha sido coherentemente formulado cuando se han definido las posibles entradas, la forma de la solución y lo que se quiere lograr (optimizar el valor de la función objetivo), aunque no haya un método de solución evidente.

Los algoritmos de planificación son por lo general problemas combinatorios *NP-hard* y mucha investigación se ha desarrollado para resolver estos problemas mediante el uso de heurísticas, algunos ejemplos de estas estrategias se pueden encontrar en [35], [36], [37] y [38]. Sin embargo, la mayoría de estas aproximaciones dependen del problema que se requiere resolver mas no son estrategias generales.

3.3.1. Shifting bottleneck scheduling para job-shops

En [39] se describe un método *offline* heurístico para resolver el problema de encontrar una planificación tal que tenga el *makespan* mínimo para un conjunto de *job-shops* sobre una plataforma idéntica, π, de tamaño m.

Para computar la solución de este algoritmo se requiere representar el problema sobre una gráfica G tal que $G = (N, A, E)$ donde N es el conjunto de nodos que representan a los jobs, A es el conjunto de aristas *conjuntivas* (aristas que definen las relaciones de precedencia entre jobs) y E es el conjunto de aristas *disyuntivas* (aristas que marcan la secuencia de ejecución de jobs en el mismo procesador.

A grandes rasgos, el algoritmo 1 muestra los pasos principales de [39].

Algorithm 1 Algoritmo Shifting Bottleneck

1: $\pi_0 \leftarrow \varnothing$
2: $E \leftarrow A$
3: Sea C_{max} la ruta crítica computada a partir de las aristas en E
4: **for all** $Proc_i \in \pi \setminus \pi_0$ **do**
5: Computar la planificación $1|r_j|L_{max}$ para $Proc_i$
6: Calcular $L_{max}(Proc_i)$
7: **end for**
8: Sea k el índice del procesador $Proc_k \in \pi \setminus \pi_0$ que maximiza $L_{max}(Proc_i)$ para $i = 1, \ldots, m$
9: $\pi_0 \leftarrow \pi_0 \cup \{Proc_k\}$
10: Computar $1|r_j|L_{max}$ para $Proc_k$
11: Actualizar E con las aristas disyuntivas activadas en 5 para $Proc_k$
12: Computar la ruta crítica, $C_{max}(E, Proc_k)$, a partir de las aristas en E
13: **for all** $Proc_i \in \pi_0 \setminus \{Proc_k\}$ **do**
14: Computar $1|r_j|L_{max}$ para $Proc_i$
15: Actualizar E con las aristas disyuntivas activadas en 5 para $Proc_i$
16: Sea $C_{max}(E, Proc_i)$ la ruta crítica computada a partir de las aristas en E
17: **end for**
18: **if** $\pi = \pi_0$ **then**
19: Terminar
20: **else**
21: Ir a la instrucción 4
22: **end if**

Para el cómputo de $1|r_j|L_{max}$ es utilizada una estrategia de *branch and bound* [40, cap.

10] con EDF_{np} (aunque también existe la variante con *preemptive EDF*) de tal forma que cada vez se obtiene la secuencia de jobs en un procesador que minimiza el *tardiness*.

3.3.2. Redes Bayesianas

En el mundo real, los ambientes de producción están sujetos a muchas fuentes de incertidumbre o aleatoriedad. Las fuentes de incertidumbre que podrían tener mayor impacto sobre estos incluyen las caídas de sistemas (*breakdowns* en inglés) y la liberación de tareas de más alta prioridad [41, cap. 9]. Otra fuente de incertidumbre recae en los tiempos de procesamiento que, con frecuencia, son desconocidos y, sin duda, los accesos a recursos compartidos como la memoria, en donde pueden ocurrir los *cache misses* en detrimento del rendimiento del sistema. Para esta tesis, sólo se considerarán como fuentes de incertidumbre para el sistema el desconocimiento de los requerimientos de ejecución y los deadlines.

El uso de redes bayesianas en el ámbito de la solución de problemas de planificación es relativamente reciente. En [35], por ejemplo, se propone una solución al problema de encontrar una planificación semanal de enfermeras en un hospital que cubran ciertas restricciones mediante el uso de redes bayesianas. En este caso, el uso de un método gráfico como las redes bayesianas resulta bastante conveniente ya que es un problema que está delimitado por muchas restricciones y cuya solución es intratable con el uso de métodos de resolución determinista (por ejemplo, programación lineal entera).

En [42] se presenta un algoritmo *offline* basado en el uso de redes bayesianas sobre una plataforma heterogénea no relacionada (ver sección 2.2.1), cuyo propósito es el de minimizar el valor del *makespan* de un conjunto de tareas que no tienen deadlines asociados y siguen relaciones de precedencia; en particular, utiliza el algoritmo de optimización bayesiana [43] (en inglés, BOA: Bayesian optimization algorithm). *BOA* funciona de la siguiente manera: Parte de la configuración inicial de una gráfica dirigida, $\mathcal{B}$, que representa la configuración inicial de una tarea *"global"* y cuyos vértices representan a las tareas; mientras que sus aristas representan los costos de comunicación entre tareas que siguen relaciones de precedencia (note que esto restringe la dirección en las comunicaciones).

A dicha gráfica, $\mathcal{B}$, se le trata de modificar su estructura mediante la agregación de nuevas aristas (que preservan las relaciones de precedencia) entre nodos hermanos para mejorar la *calidad de la estructura* de la red (grafo), mediante una función de calificación (*scoring metric*) que es utilizada como el *fitness*, en particular utiliza la métrica *Bayesian-Dirichlet* [44], esta última toma en consideración la distribución de probabilidad a priori de la estructura de red $\mathcal{B}$.

Se debe resaltar que lo propuesto en [42], a diferencia del método propuesto en el capítulo 5 de esta disertación, corresponde a la implementación de redes bayesianas para encontrar el acomodo de un conjunto de tareas de tal forma que se reduce el tiempo global de ejecución sin considerar restricciones temporales (en particular deadlines). Mientras que el trabajo aquí presentado va más allá y propone una metodología que busca el aprovechamiento de tiempos ociosos de manera *online* incorporando tantas nuevas tareas (*job-shops* con sus respectivos *deadlines*) como sea posible a un sistema ya existente de tareas, cumpliendo

siempre con sus restricciones temporales e incorporando las redes bayesianas para la toma de decisiones.

Capítulo 4

Planificación determinista

Contents

4.1. Trabajo relacionado 31
4.2. Descripción del problema 31
4.3. Transformaciones 32
4.4. Estados de las tareas 35
4.5. Diseño del servidor 37
4.6. Interrupciones 38
4.7. Teorema 40
4.8. Conclusión 41

En poco más de dos décadas, fueron desarrollados varios algoritmos óptimos que siempre encuentran un acomodo (*schedule*) válido de tareas (siempre que uno exista) en plataformas multiprocesador, esto es, los algoritmos encuentran un schedule capaz de ocupar la totalidad del tiempo de cómputo de una plataforma multiprocesador idéntica para un conjunto de tareas periódicas: *p-fair* [6], *Boundary Fair* (*BF*) [7], *EKG* [8], *LLREF* [9], *DP-Wrap* [10], *RUN* [4]. Todas, excepto [4], se basan en alguna forma de *proportional fairness* y, al igual que en el algoritmo de *McNaughton* [45], todos suponen que los deadlines son iguales. La mayoría logran esta equidad en los deadlines (cuando el sistema de tareas tiene deadlines distintos) subdividiendo los requerimientos de ejecución de los jobs de las tareas e imponiendo los deadlines de cada tarea a todas las otras tareas, que a su vez, se imponen a todas las "subdivisiones" [10]. Lo que conlleva a un número excesivo tanto de cambios de contexto (*context switching*) como de migraciones.

Todos estos algoritmos tienen la limitación de que sólo son aplicables a plataformas multiprocesador idénticas con sistemas de tareas recurrentes (periódicas) y preemptive.

En este capítulo se presenta una contribución que unifica varias características poco consideradas en la planificación: planificación de job-shops con instancias non-preemptive sobre plataformas heterogéneas; sin embargo, son de importancia para la industria y, en particular, altamente aplicables a la industria 4.0.

Tradicionalmente, planificar tareas de tipo job-shop es un problema de optimización en el cual muchos jobs (de manufactura, p.e.) son asignados a máquinas en ciertos instantes, mientras tratan de minimizar el *makespan*. Este trabajo se enfoca en la planificación de un sistema de tareas compuesto por job-shops, cuyas instancias son non-preemptive, con un sólo deadline sobre una plataforma heterogénea. Para lograr tal objetivo se propone extender de [16] para tareas non-preemptive. Además, se supondrá que los job-shops son conocidos hasta sus respectivos tiempos de llegada.

La investigación sobre planificación necesita enfocarse en este tipo de tareas (job-shops) debido a su importancia en los procesos inherentes de la industria (manufactura, costos de inventario, planificación de transportación, etc.). En el ámbito de la industria 4.0, la planificación debería hacer frente al soporte de sistemas de manufactura distribuidos e inteligentes por medio de tecnologías de manufactura emergentes tal como la personalización masiva[1], sistemas cyber-físicos, digital twins y SMAC (Social, Mobile, Analytics, and Cloud), big data, etc. [15]. Los sistemas embebidos son una plataforma para implementar proyectos en términos de los requerimientos de IoT y de la industria 4.0 considerando las diferentes capacidades de los recursos (comunicaciones, capacidad cómputo, memoria, etc.), eficiencia de energía, etc.

Los sistemas embebidos deben operar en ambientes dinámicos donde las actividades humanas ocurren en cualquier momento; entonces, algunas tareas tales como las tareas de emergencia, tareas de eventos externos, tareas de interacciones humanas, etc., llegan de manera aperiódica [46].

La planificación de tareas non-preemptive es ampliamente utilizada en la industria y ésta puede ser preferible a la planificación de tareas preemptive por ciertas razones [47]: los algoritmos para tareas non-preemptive son, relativamente, más fáciles de implementar y tienen bajo *overhead* durante el tiempo de ejecución a comparación de los algoritmos preemptive; el overhead de algoritmos preemptive es más difícil de caracterizar y predecir (por ejemplo en [48]) que para planificación non-preemptive debido a la interferencia que entre tareas causada por el manejo de la memoria cache (*caching*) y *pipelinig*. Estos beneficios que ofrece la planificación non-preemptive son de suma importancia para plataformas multiprocesador: por si mismo, el overhead provocado por la migración es mayor y más difícil de predecir [49].

Un resultado que da la posibilidad de pensar que para una parte considerable de aplicaciones para la vida-real sobre plataformas multiprocesador, la planificación non-preemptive podría ser una mejor elección con respecto al tiempo real de cumplimiento está especificado en [49], ya que en él se muestra experimentalmente que para conjuntos de tareas, en los que el rango de los requerimientos de ejecución no es tan amplio, el desempeño de EDF_{np} (forma breve de referirse a Earliest Deadline First non-preemptive) es muy cercano a EDF y DM_{np} (forma corta de escribir Deadline Monotonic non-preemptive) se desempeña mejor que DM.

[1]Mass customization o «personalización masiva» es una manera de fabricar y prestar servicios para hacer frente a las demandas del mercado. La idea principal de este sistema de manufactura es satisfacer los gustos y necesidades de los clientes de forma individual con costos similares a los de producción en masa. Requiere estrategias centradas en el cliente, la innovación, la producción, la logística y la sostenibilidad.

La propuesta aquí desarrollada es una extensión de [16] para tareas non-preemptive que están compuestas por jobs que siguen relaciones de precedencia, a saber job-shops con deadlines únicos, sobre una plataforma multiprocesador heterogénea.

4.1. Trabajo relacionado

El análisis de suficiencia propuesto por Baruah [16], de ahora en adelante llamado *BAR06*, consiste en validar la restricción (4.1) para un conjunto de n tareas periódicas non-preemptive, τ, sobre una plataforma multiprocesador idéntica con m procesadores.

$$V_{sum}(\tau) \leq m - (m-1) \cdot V_{max}(\tau) \tag{4.1}$$

donde

$$V_{sum}(\tau) := \sum_{\tau_i \in \tau} V(\tau_i, \tau) \tag{4.2}$$

$$V_{max}(\tau) := \max_{\tau_i \in \tau} \{V(\tau_i, \tau)\} \tag{4.3}$$

y $V(\tau_i, \tau)$ es la utilización de la tarea τ_i que es periódica y no preemptive:

$$V(\tau_i, \tau) := \frac{c_i}{T_i - e_{max}(\tau)} \tag{4.4}$$

$e_{max}(\tau)$ es el valor máximo que puede tener el requisito de ejecución de una tarea en τ, como puede ser apreciado en (4.4), que es el peor escenario para el tiempo de interferencia causado por el hecho de que las tareas son preemptive; $e_{max}(\tau)$ se define a continuación:

$$e_{max}(\tau) := \max_{i=1,\ldots,n} \{c_i\} \tag{4.5}$$

donde c_i y T_i son el requerimiento de ejecución y el tiempo entre llegadas de la tarea periódica, $\tau_i \in \tau$, respectivamente. Una observación obvia es que una tarea $\tau_i \in \tau$ con baja utilización puede no satisfacer (4.1) si $c_i > T_{min}$, donde $T_{min} := \min_{i=1,\ \ldots,\ n}\{T_i\}$ denota el periodo mínimo de las tareas en τ.

4.2. Descripción del problema

En los sistemas de tiempo-real, el modelo de computación es mucho más complejo ya que están conformados por diferentes componentes y, por lo tanto, estos tienen diferentes capacidades de cómputo. Por ejemplo, los sistemas embebidos deben operar en ambientes dinámicos donde las actividades humanas ocurren en cualquier momento, esto puede verse

como la llegada de tareas aperiódicas al sistema; por lo tanto, existe un dinamismo en el número de tareas que llegan a la plataforma solicitando el uso de algún recurso.

Contrario a lo anteriormente mencionado, *BAR06* que funciona para condiciones estáticas[2]; en este trabajo, la atención se centra en aquellas tareas que llegan en instantes desconocidos a la plataforma, esto es que las tareas son conocidas hasta su tiempo de llegada. Se supondrá para este modelo de tareas que cada job-shop tendrá un único deadline que, además, es crítico (otro término para referirse a los *hard deadlines*).

Recapitulando, el modelo de tareas que aquí se maneja consiste de job-shops, se utilizará τ_i para denotar al job-shop con identificador i ($i \in \mathbb{N}$). Una tarea de este tipo está compuesta por n_i ($n_i \in \mathbb{N}$ y $n_i < \infty$) jobs; un *job* se define como una secuencia de instrucciones que se efectúan sin interrupción. Al job número j de la *i-ésima* tarea se le representa como $\tau_{i,j}$, donde $j = 1, \ldots, n_i$ y $i = 1, \ldots$.

Los job-shops son caracterizados por restricciones temporales tales como un único tiempo de llegada (o liberación), denotado por r_i, un único deadline, d_i y n_i requerimientos de ejecución (uno por cada job) denotado como $c_{i,j}$ (donde $j = 1, \ldots, n_i$). Dichos valores serán conocidos hasta r_i.

Un job-shop, τ_i, es una secuencia ordenada de jobs o, definido de manera más precisa, sea $\tau_i = \{\tau_{i,j} \colon i = 1, \ldots,\ j = 1, \ldots, n_i\}$ un job-shop donde $n_i \in \mathbb{N}$ denota su número de jobs. Estos jobs siguen *relaciones de precedencia*, esto significa, el job $\tau_{i,j}$ no puede comenzar su ejecución si el job previo, $\tau_{i,j-1}$, no ha finalizado todavía.

Definición 4 (Relación de precedencia) *Para cualesquiera dos jobs consecutivos $\tau_{i,j-1}$, $\tau_{i,j} \in \tau_i$, el job $\tau_{i,j-1}$ precede a $\tau_{i,j}$ si su tiempo de finalización $f_{i,j-1}$ satisface que $f_{i,j-1} \leq b_{i,j}$, donde $b_{i,j}$ denota al tiempo de comienzo de la ejecución $\tau_{i,j}$, $i = 1, \ldots$ y $j = 2, \ldots, n_i$.*

La plataforma sobre la que se planificaran estas tareas es uniforme (heterogénea). $\pi = (s_1,\ s_2,\ \ldots, s_m)$ denota a dicha plataforma multiprocesador de tamaño m en donde el procesador $Proc_i$ tiene tasa de rapidez s_i, donde $s_i \geq s_{i+1}$ para $i = 1, \ldots, m-1$, entre mayor sea el valor de s_i más rápido será $Proc_i$.

El problema abordado en este trabajo es el de determinar si se asigna o no un nuevo job-shop liberado en el instante t sobre una plataforma heterogénea. Por lo que el test de planificabilidad propuesto está definido en función de las tareas activas en el instante t.

4.3. Transformaciones

Con el propósito de aplicar el análisis de planificabilidad propuesto por *Baruah* en [16] sobre job-shops en una plataforma heterogénea, la siguiente serie de transformaciones *virtuales* es llevada a cabo:

1. Transformación de una plataforma heterogénea en una homogénea.

[2]No sólo el número de tareas permanece igual sino que también los requerimientos de ejecución permanecen iguales durante la ejecución.

2. Los job-shops aperiódicos non-preemptives que son liberados en tiempo de ejecución son transformados en tareas "periódicas".

Lema 1 (Plataforma homogénea asociada) *Una plataforma uniformemente heterogénea tiene asociada una plataforma idéntica (homogénea).*

Demostración 1 *La prueba es por construcción. Se denotará con* $\pi = (s_1, s_2, \ldots, s_m)$ *a las tasas de rapidez de una plataforma uniforme heterogénea de tamaño* m *donde el* $k-$*ésimo procesador tiene una tasa de* s_k *donde* $0 < s_k \leq 1$.

Para convertir la plataforma heterogénea actual en una homogénea es necesario satisfacer que las velocidades sean las mismas. Para ello, es necesario encontrar un factor para cada tasa de rapidez de la plataforma heterogénea, sea α_k $(k = 1, \ldots, m)$ *dicho factor, tal que* $\alpha_k \cdot s_k = s_{max}$ *donde* s_{max} *es la tasa del procesador más lento (note que la tasa* $s_{max} = 1$ *indicaría al procesador más lento posible)*

$$s_{max} := \max_{k=1,\ldots,m} \{s_k\}$$

entonces

$$\alpha_k = \frac{s_{max}}{s_k}$$

Mediante los factores de rapidez de una plataforma es posible encontrar a la plataforma asociada más lenta posible.

Esta plataforma homogénea es una nueva plataforma virtual sobre la cual el test de planificabilidad será aplicado para las nuevas tareas que vayan siendo liberadas. Note que el factor α_k, para toda $k = 1, \ldots, m$, indica que tanto el procesador $Proc_k$ debe alentarse para formar parte de la nueva plataforma más lenta posible. Note también que tanto más grande sea α_k, más lento se volverá el k-ésimo procesador.

Para el lema 2, los requerimientos de las tareas *pseudoperiódicas* son calculados bajo la suposición de que se cuenta con una plataforma homogénea como la construida a partir del lema 1. Se asume que para cada job liberado $\tau_{i,j} \in \tau_i$ existe un requerimiento de ejecución conocido $c_{i,j}$ $(i = 1, \ldots, n)$ y está dado en términos del procesador más lento en π. Esto significa que un job, $\tau_{i,j}$, ejecutado sobre un procesador de la plataforma heterogénea tardará menos tiempo que en la homogénea subyacente: $c_{i,j} \geq c_{i,j} \cdot s_k$ para toda $k = 1, \ldots, m$.

Lema 2 (Tarea pseudoperiódica) *Un job-shop,* τ_i*, tiene su tarea periódica asociada, la cual es llamada la tarea pseudoperiódica, se denotará como* τ_i'.

Demostración 2 *La prueba es por construcción. Una tarea pseudoperiódica,* τ_i'*, puede ser representada como una tupla* $(r_i, C_i, T_i(t), d_i)$*, donde* r_i *es el mismo tiempo de liberación que tiene el job-shop* τ_i*;* C_i*, es la abreviación para* $C(\tau_i)$ *que es el requerimiento de ejecución virtual y se computa de la siguiente forma:*

$$C(\tau_i) := \max_{\tau_{i,j} \in \tau_i} \{c_{i,j}\} \tag{4.6}$$

por lo tanto, para cada $\tau_{i,j} \in \tau_i$ con requerimiento de ejecución $c_{i,j}$, es necesario añadir $C_i - c_{i,j}$ unidades de tiempo ocioso, tal que $j = 1, \ldots, n_i$, donde n_i es el número de instancias no ejecutadas de τ_i; $T_i(t)$, es la forma breve de escribir $T(\tau_i, t)$ que es el tiempo de liberación virtual entre instancias de una tarea (interarrival time), note que este atributo es impuesto a τ_i para darle forma de tarea periódica y depende del instante t en el que se calcula:

$$T(\tau_i, t) := \left\lfloor \frac{d_i - t}{n_i} \right\rfloor \tag{4.7}$$

donde d_i es el deadline absoluto de τ_i y t es el instante en el que se detona una interrupción. Se deben satisfacer las siguientes restricciones $t + T_i(t) \cdot n_i \leq d_i$ y $C_i \leq T_i(t)$.

Note que cuanto mayor sea el valor de t, con respecto de r_i, el tiempo de liberación entre tareas para τ_i, $T_i(t)$, se reduce y por lo tanto su utilización aumenta.

En este escenario de estudio, los jobs de $\tau_i \in \tau$, para toda $i = 1, 2, \ldots$, son finitos; sus requerimientos de ejecución no son necesariamente iguales; los nuevos job-shops son conocidos hasta sus respectivos tiempos de llegada y son planificados online. De ahí que sea necesario modificar el análisis *BAR06*: Se debe considerar la evolución en el tiempo del sistema de tareas con job-shops "aceptados".

Cuando el job $\tau_{i,j} \in \tau_i'$ es consumido, es descartado (*discarded*). Con ello, las características del job-shop son otras, a diferencia de 4.4 que mantiene sus características estáticas a lo largo de la ejecución, por lo tanto se propone la siguiente forma para el cálculo de la utilización para tareas pseudoperiódicas non-preemptive, $v(t, \tau_i, \tau)$, este valor se calcula a partir de (4.6) y (4.7) (ver lema 2) y luego se sustituyen en (4.4) cada vez que la tarea servidor se encuentre activa en el instante t.

$$v(t, \tau_i, \tau) := \frac{C_i}{T_i(t) - e_{max}(\tau)} \tag{4.8}$$

donde $e_{max}(\tau)$ es el valor máximo que pueden tomar los requerimientos de ejecución de los jobs de las tareas (pseudo)periódicas en el conjunto τ:

$$e_{max}(\tau) := \max_{\tau_i \in \tau} \{C_i\} \tag{4.9}$$

si se considera todo este "dinamismo" en el instante t para el nuevo modelo, la desigualdad (4.1) se modifica como se muestra a continuación:

$$v_{sum}(t, \tau) \leq m - (m - 1) \cdot v_{max}(t, \tau) \tag{4.10}$$

donde

$$v_{max}(t, \tau) := \begin{cases} \max_{\tau_i \in \tau} \{v(t, \tau_i, \tau)\} & \text{if } \tau \neq \phi \\ 0 & \text{if } \tau = \phi \end{cases} \tag{4.11}$$

y

$$v_{sum}(t,\tau) := \begin{cases} \sum_{\tau_i \in \tau} v(t,\tau_i,\tau) & \text{if } \tau \neq \phi \\ 0 & \text{if } \tau = \phi \end{cases} \tag{4.12}$$

note la dependencia que ahora se tiene del tiempo para este nuevo análisis de aceptación.

Hasta aqui, se conoce como ajustar las restricciones temporales de un conjunto fijo de job-shops a un conjunto de tareas periódicas, pero no se ha definido la forma de proceder con los nuevos job-shops liberados en el sistema. Por lo tanto, en las secciones subsecuentes se describe una manera sistemática para convertir y verificar las tareas liberadas.

4.4. Estados de las tareas

En este trabajo, se propone una posible solución utilizando una serie de *interrupciones* y un *servidor* (τ^s), estos son adaptados a una plataforma multiprocesador para asignar los recursos a las tareas liberadas.

Existen cuatro posible estados por los que una tarea puede pasar, a saber *released, wait, active* y *dead*; cada estado dispone de una estructura de datos global que le auxilia para mantener la referencia a las tareas con las que debe trabajar; por ejemplo, como es mostrado en la figura 4.1, el estado *released* tiene asociada la estructura de datos $\tau_{released}$, que recibe todas las nuevas solicitudes de las tareas que van llegando en cualquier momento al sistema.

Cuando una interrupción ocurre en el instante t y se tiene el *presupuesto* (budget) suficiente en la tarea servidor (C^s denota dicha capacidad destinada al presupuesto del servidor), las tareas referenciadas por $\tau_{released}$ comienzan a ser transformadas (ver lema 2) y después éstas cambian su estatus a *wait*, esto significa que el servidor las movió a la estructura τ_{wait}. Formalmente, $\tau_{released}$ se define a continuación

$$\tau_{released} := \{\tau_i : 0 \leq r_i \leq t\}$$

donde t es el instante actual y r_i es el tiempo de liberación de la tarea τ_i para $i = 1,\ 2,\ \ldots$.

Cuando EDF_{np} elige a τ^s como la tarea de mayor prioridad y todas las tareas de $\tau_{released}$ ya han sido transformadas, τ^s valida cuales tareas de τ_{wait} siguen siendo planificables.

Definición 5 (Tarea planificable) *En el instante t, dado el tiempo total de utilización de las tareas pseudoperiódicas que actualmente están siendo atendidas, $v_{sum}(t, \tau_{active})$, se dice que una tarea $\tau_i \in \tau_{wait}$ es planificable en el instante t si ésta puede ser ejecutada antes de su deadline, d_i:*

$$t + n_i \cdot C_i + v_{sum}(t, \tau_{active}) \cdot \frac{(d_i - t)}{m} \leq d_i \tag{4.13}$$

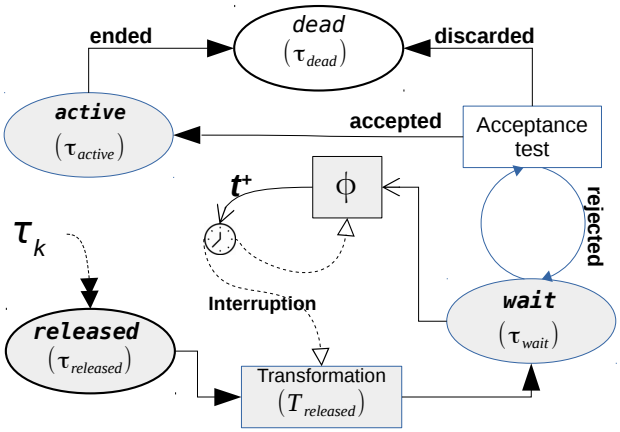


Figura 4.1: Estados de los jobs en tiempo de ejecución

donde n_i es el número de jobs en τ_i que todavía no han sido atendidos y m es el número de procesadores. Note que este análisis supone una carga de trabajo "balanceada" entre los procesadores de la plataforma.

Si una tarea pseudoperiódica planificable τ'_k aprueba el análisis *BAR06* en el instante t, entonces su estatus cambia a *active* y es puesta en τ_{active} siempre y cuando haya tiempo disponible en la plataforma o, bien, se colocan en τ_{wait} donde sus correspondientes jobs esperan por ser atendidos, es decir, las tareas son rechazadas (*rejected*); de lo contrario, las tareas son descartadas (trasladadas a τ_{dead})

$$\tau_{wait} := \{\tau_i : t + n_i \cdot C_i + v_{sum}(t, \tau_{active}) \cdot \frac{(d_i - t)}{m} \leq d_i \text{ and } v_{sum}(t, \tau) > m - (m-1) \cdot v_{max}(t, \tau)\} \tag{4.14}$$

donde $\tau = \tau_{active} \cup \{\tau_i\}$.

Como es posible observar en la figura 4.1, las tareas rechazadas (*rejected*) siguen siendo planificables, mientras que las descartadas (*discarded*) son desechadas pues éstas ya no pueden satisfacer la restricción (4.13); además, de acuerdo con [16], se debe satisfacer la siguiente restricción antes de verificar la desigualdad (4.10): El máximo de los requerimientos de ejecución en cualquier conjunto de tareas τ, $e_{max}(\tau)$, debe ser menor o igual que el mínimo de los periodos de τ: $e_{max}(\tau) \leq T_{min}$.

Cuando una tarea satisface (4.10) es añadida a τ_{active} la cual es una estructura global que conserva las referencias de las tareas pseudoperiódicas siendo ejecutadas por EDF_{np}.

Como se muestra en el algoritmo 2, cada vez que una interrupción es detonada en el instante t, cada $\tau_k \in \tau_{released}$ es almacenada en una estructura de datos auxiliar, denotada como $T_{released}$, antes de ser transformada a su tarea periódica asociada, τ'_k, y formar parte de τ_{wait} (ver linea 1); después se verifican las restricciones (4.13) y (4.10) para cada tarea $\tau'_k \in \tau_{wait}$ (ver lineas 3-10).

Algorithm 2 ACCEPTANCE TEST

1: $T_{released} \longleftarrow convert2Pseudoperiodic(\tau_{released})$
2: $\tau_{wait} \longleftarrow \tau_{wait} \cup T_{released}$
3: **for all** $\tau'_k \in \tau_{wait}$ or c^s is not used up **do**
4: $\quad \tau \longleftarrow \tau_{active} \cup \{\tau'_k\}$
5: $\quad$ **if** $t + n_k \cdot C_k + v_{sum}(t, \tau_{active}) \cdot \frac{(d_k - t)}{m} \leq d_k$ **and** $C_k \leq min_{\tau_i \in \tau_{active}}\{T_i\}$ **and** $T_k \geq$ máx$_{\tau_i \in \tau_{active}}\{C_i\}$ **and** $v_{sum}(t, \tau) \leq m - (m-1) \cdot v_{max}(t, \tau)$ **then**
6: $\quad\quad \tau_{active} \longleftarrow \tau$
7: $\quad$ **else if** $t + n_k \cdot C_k + v_{sum}(t, \tau_{active}) \cdot \frac{(d_k - t)}{m} > d_k$ **or** $C_k > min_{\tau_i \in \tau_{active}}\{T_i\}$ **or** $T_k <$ máx$_{\tau_i \in \tau_{active}}\{C_i\}$ **then**
8: $\quad\quad$ discard(τ'_k)
9: $\quad$ **end if**
10: **end for**

4.5. Diseño del servidor

Generalmente, la tarea servidor (o simplemente servidor), denotada por τ^s, es un caso especial de tarea periódica cuyo propósito es el de ocuparse de peticiones aperiódicas tan pronto como sea posible. En el caso del presente trabajo son consideradas dos tipos de peticiones. Primera, aquellas tareas que están en espera hasta el instante t en $\tau_{released}$ para ser transformadas. Segunda, aquellas tareas referenciadas por τ_{wait} y necesitan ser validadas, en caso de ser aceptadas, éstas son transferidas a τ_{active}. τ^s se adapta a si misma de acuerdo a las condiciones actuales de la plataforma, esto significa que la tarea servidor es capaz de cambiar la capacidad de su budget en la detonación de cada interrupción (ver sección 4.6). La ejecución del servidor no se restringe a ningún procesador, es decir, es global.

En la mayoría de los casos, desde el punto de vista de un sistema de tareas preemptive, la tarea servidor se suele diseñar de acuerdo a la utilización disponible de la plataforma, se denotará por U^s a dicho valor. La utilización para τ^s está acotada de acuerdo a la capacidad dejada por por las tareas en τ_{active}. En el instante t, cuando una interrupción ha sido lanzada, U^s es calculada como se indica a continuación:

$$U^s(t) = m - v_{sum}(t, \tau_{active}) \tag{4.15}$$

ya que el problema abordado hace frente a tareas non-preemptive, U^s es tratado como la cota superior para el valor de la utilización de τ^s, se denota por u^s, esto es $0 < u^s \leq U^s$.

Para el análisis de planificabilidad, es necesario considerar también a τ^s como una tarea extra, además de las ya existentes en τ_{active}, por lo que se debe satisfacer la restricción (4.10):

$$v_{sum}(t, \tau_{active}) + u^s \leq m - (m-1) \cdot v_{max}(t, \tau_{active})$$

por lo tanto, para el sistema de tareas non-preemptive, u^s debe satisfacer

$$u^s \leq m - (m-1) \cdot v_{max}(t, \tau_{active}) - v_{sum}(t, \tau_{active}) \tag{4.16}$$

El valor mínimo que puede tomar u^s se supondrá como parámetro definido por el usuario, sea u^s_{min} dicho valor, tal que $0 < u^s_{min} \leq u^s$. De todo lo dicho se concluye que u^s se puede elegir a partir de (4.16) y u^s_{min} de la siguiente forma:

$$u^s := \text{máx}\{u^s_{min},\ m - (m-1) \cdot v_{max}(t, \tau_{active}) - v_{sum}(t, \tau_{active})\} \tag{4.17}$$

con esto, se pretende asignar el valor más grande posible de utilización para u^s.

La capacidad para el presupuesto de τ^s, denotada por C^s, debe estar delimitada superiormente por $e_{max}(\tau_{active})$, (i.e. $0 < C^s \leq e_{max}(\tau_{active})$), esto es para garantizar que no habrá necesidad de calcular nuevamente el valor de utilización de todas las tareas en $\tau_{acepted}$. Para garantizar previsibilidad, es necesario que también se satisfaga que siempre haya suficiente budget para ejecutar, al menos, una petición en el sistema, entonces C^s es acotada de la siguiente manera:

$$max\{c_1, c_2\} \leq C^s \leq e_{max}(\tau_{active}) \tag{4.18}$$

Una vez que C^s fue asignado, es necesario definir el cálculo del deadline absoluto de τ^s, se denota al deadline con d^s. Éste es calculado de tal forma que la interferencia de la tarea con el mayor de los requerimientos de ejecución, $e_{max}(\tau_{active})$, sea considerado, y todo el *budget* asignado sea agotado antes de dicho deadline:

$$d^s(t) = t + \left\lceil \frac{C^s}{u^s_{min}} + e_{max}(\tau_{active}) \right\rceil \tag{4.19}$$

Aunque el sistema de tareas consiste de jobs con relaciones de precedencia, el análisis es llevado a cabo a nivel de job. Bajo las condiciones establecidas y las transformaciones propuestas es posible garantizar la ejecución de una tarea a través de sus jobs.

Las reglas para la reposición del budget se enuncian a continuación:

- Inicialmente, en $t = 0$, d^s es puesto en cero, u^s es inicializado de acuerdo a (4.17) y τ_{active} está vacía ($\tau_{active} = \phi$) por lo tanto $u^s = m$ (ver definiciones (4.11) y (4.12)), esto significa que, al comienzo, la tarea τ^s dispone totalmente de la plataforma.
- Cada vez que haya sido alcanzado el deadline de τ^s, a C^s se le asigna el valor mínimo que le es posible tomar: $C^s = max\{c_1, c_2\}$ (ver (4.18)) y el siguiente deadline d^s es calculado nuevamente de acuerdo a la ecuación (4.19).

4.6. Interrupciones

Las interrupciones son importantes para este trabajo por dos razones. La primera, porque en la ocurrencia de cada interrupción, en el instante t, el budget de τ^s se ajusta de acuerdo

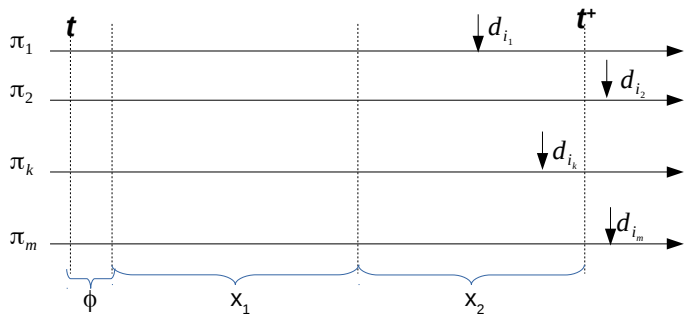


Figura 4.2: Diferentes subintervalos en los que se divide el intervalo $[t, t^+)$: x_1, x_2 y ϕ

al número de tareas en τ_{active} (más tareas en τ_{active} implicarán menor budget para τ^s). La segunda, la siguiente interrupción es reprogramada, se le denotará t^+, para darle atención a las tareas liberadas (tareas referenciadas por $\tau_{released}$).

Se asume que la primera interrupción ocurre en el instante $t = 0$ y hay también un valor mínimo de utilización, u^s_{min}, definido por el usuario para la tarea τ^s, entonces t^+ es calculada como sigue

$$t^+ = t + \left\lceil \frac{\phi + x_1 + x_2}{u^s_{min}} \right\rceil \tag{4.20}$$

donde ϕ es el intervalo que le tomará a la plataforma calcular la siguiente interrupción, t^+, y el budget para el servidor; mientras que x_1 y x_2 son los intervalos de tiempo que tomaran las operaciones de transformación y validación (por medio de la condición (4.10)), respectivamente, ver figura 4.2.

Para saber cuanto tiempo dura calcular t^+ las siguientes suposiciones son hechas:

S0 Se considera que τ^s siempre trabaja bajo el peor escenario, es decir, se cumple que $u^s = u^s_{min}$ (ver sección 4.5).

S1 El tiempo requerido para calcular ϕ se considerará constante. En la implementación, es posible dotar a las estructuras de datos (τ_{active}, τ_{wait}, etc.) de contadores que permitan conocer en cada instante el número total de jobs o tareas, según sea requerido, sean $c^{\tau_x}_{jobs}$ y c^{τ_x} los contadores que almacenan el número de jobs y tareas, respectivamente, para la estructura τ_x.

S2 Los tiempos de cómputo que le toman a ambas operaciones, transformación y validación, dependen del número de jobs y tareas, respectivamente.

S3 Ni la operación de trasformación ni la de validación toman tiempo cero. La proporción de tiempo utilizada por dichas operaciones es indicada por las constantes c_1 y c_2 tal que $c_1, c_2 > 0$, para las operaciones de transformación y validación, respectivamente.

De **S2** y **S3**, es fácil deducir que al complejidad de calcular t^+ sólo depende del número de jobs en $\tau_{released}$ y del número de tareas en τ_{wait}.

Lema 3 (Tiempo de transformación de tareas) *El tiempo necesario para transformar las tareas en $\tau_{released}$ depende del número de jobs que ésta tenga*

$$x_1 = c_1 \cdot \ c_{jobs}^{\tau_{released}}$$

que puede ser reescrita como

$$x_1 = c_1 \cdot \sum_{\tau_i \in \tau_{released}} n_i$$

donde n_i es el número de jobs (aún sin ejecutar) en la tarea $\tau_i \in \tau_{released}$.

Demostración 3 *La prueba sigue de las suposiciones **S2** and **S3**.*

Después de llevar a cabo la transformación de las tareas en $T_{released}$, éstas son añadidas a τ_{wait} donde esperan hasta ser aceptadas o descartadas. Note también que es posible conocer en un cierto intervalo si una tarea puede ser ejecutada antes de su deadline, a saber $[t, t + \phi + x_1)$, lo que da la posibilidad al usuario de cargar la tarea en otro sistema.

Lema 4 (Tiempo de validación de tareas) *Una vez que las tareas han sido transformadas, el tiempo requerido para validar a las tareas en τ_{wait} por medio de BAR06 es*

$$x_2 = c_2(2 \cdot c^{\tau_{active}} + c^{\tau_{wait}})$$

donde $c^{\tau_{active}}$ y $c^{\tau_{wait}}$ representan el número de tareas referenciadas por las estructuras de datos τ_{active} y τ_{wait}, respectivamente.

Demostración 4 *Sea τ_{wait}, la estructura de datos que referencia a aquellas tareas hasta su ejecución o remoción (discard).*

Para aplicar BAR06 es necesario calcular dos valores de las tareas que están actualmente asignadas a la plataforma (en ejecución), τ_{active}; primero, el requerimiento de ejecución máximo $v_{max}(t, \tau_{active})$, computar esto toma tiempo proporcional al número de tareas en τ_{active}, o sea, $c^{\tau_{active}}$; segundo, $v_{sum}(t, \tau_{active})$ que también toma tiempo proporcional a $c^{\tau_{active}}$.

Las tareas en τ_{wait} son validadas de manera secuencial, por lo tanto toda tarea que trate de formar parte de $\tau_{acvtive}$ no incrementará el tiempo de complejidad. Éste seguirá teniendo complejidad lineal, de acuerdo al número de tareas que necesiten ser validadas, entonces el tiempo que toma la validación de todas las tareas en τ_{wait} es añadido.

Como se mencionó antes, en la suposición **S1**, a través del uso de contadores en cada estructura de datos, es posible obtener la información del número de tareas y jobs de $\tau_{released}$, τ_{wait}, and τ_{active} en tiempo constante, por lo tanto el cálculo de x_1 y x_2 puede considerarse como nulo y así también para ϕ. Por lo tanto, la ecuación (4.20) puede ser reducida a

$$t^+ = t + \left\lceil \frac{x_1 + x_2}{u_{min}^s} \right\rceil \tag{4.21}$$

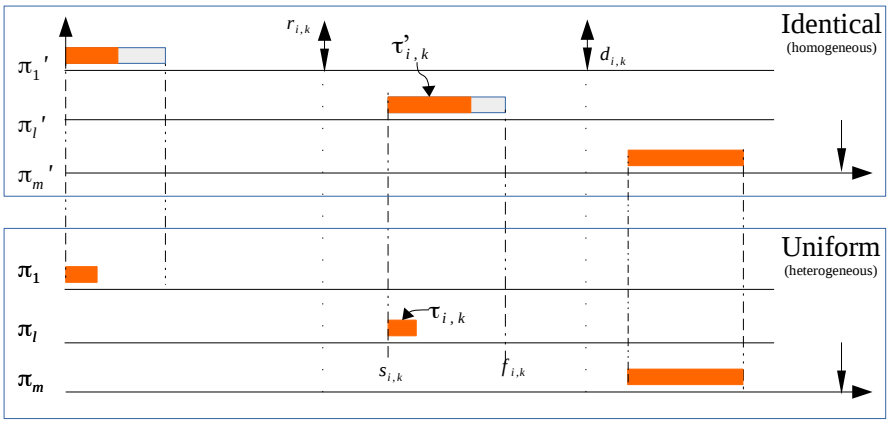


Figura 4.3: Correspondencia entre un job-shop (abajo) y su tarea pseudoperiódica asociada (arriba)

4.7. Teorema

Después de las transformaciones aplicadas tanto a la plataforma como al conjunto de tareas, la validación de tareas e interrupciones, se enuncia el siguiente resultado.

Teorema 10 (Planificabilidad en plataformas heterogéneas) *Si un conjunto de tareas pseudoperiódicas es planificada en la plataforma homogénea asociada, entones dicho conjunto es planificable también sobre la plataforma uniforme heterogénea subyacente.*

Demostración 5 *Los job-shops son tratados como tareas periódicas debido a que se les ha aplicado la transformación del lema 2 y sus respectivos jobs serán elegidos por EDF_{np} global durante el tiempo de ejecución. Cuando EDF_{np} selecciona un job $\tau'_{i,j} \in \tau'_i$ de τ_{active} para asignarlo a la plataforma homogénea, éste, de hecho, asigna al job $\tau_{i,j}$ correspondiente al job-shop τ_i, el cual es asignado a la plataforma heterogénea.*

Los tiempos de comienzo $s_{i,k}$ y liberación $r_{i,j}$ de cada job serán los mismos en ambas plataformas. A pesar de que los jobs asignados a la plataforma heterogénea posiblemente finalizan su procesamiento en menor tiempo, estos siguen la forma de proceder de la plataforma homogénea (ver figura 4.3).

Cuando la planificabilidad de una tarea pseudoperiódica $\tau'_i \in \tau_{active}$ es garantizada por BAR06, la cual está dada en términos de la tasa de rapidez del procesador más lento, entonces en tiempo de ejecución cuando $\tau'_{i,j}$ es asignado a la plataforma homogénea, con un requerimiento de ejecución C_i, en el instante $s_{i,j}$, entonces el job asociado del job-shop, $\tau_{i,j}$, ejecutado sobre el procesador π_l de la plataforma heterogénea no excederá el límite impuesto por la plataforma homogénea, esto es

$$s_{i,j} + C_i \cdot \frac{s_l}{s_{max}} \leq s_{i,j} + C_i$$

como $0 < s_l \leq s_{max} \leq 1$ entonces $\frac{s_l}{s_{max}} \leq 1$, donde s_l es la tasa de rapidez de π_l.

En general, como las tasas de rapidez en la plataforma heterogénea son menores que la tasa de rapidez de la plataforma homogénea (see lemma 1), el requerimiento de cada $\tau_{i,j}$, $c_{i,j}$, satisfará que $c_{i,j} \leq C_i$, donde C_i es el requerimiento de ejecución de la tarea pseudoperiódica asociada τ'_i.

4.8. Conclusión

Esta extensión al análisis de planificabilidad *BAR06* permite determinar de manera *online* si un nuevo job-shop que se libera en el sistema es capaz de cumplir con su deadline. Ya que este análisis de planificabilidad tarda tiempo lineal, de acuerdo al número de tareas, el resultado presente tiene el propósito de ser implementado en sistemas con recursos limitados debido al bajo costo computacional que este análisis representa.

Capítulo 5

Planificación no determinista

Contents

5.1. El problema 44
5.2. Mapeo del problema a la red bayesiana 46
5.2.1. Nodos: Tareas e intervalos ociosos 46
5.2.2. Relaciones 49
5.2.3. Especificación de la plataforma 50
5.3. Descripción de la estrategia 50
5.3.1. *$\mathcal{A}$-modificado* 51
5.4. Función objetivo 54
5.5. Cotas 55
5.5.1. Número de posibles instancias 55

El análisis tradicional para garantizar el cumplimiento de tareas con plazos de tipo "estricto" (*hard deadlines*, como se les conoce en inglés) se basa en considerar los peores tiempos de ejecución (WCET, por sus siglas en inglés), dicho análisis resulta ser una visión pesimista del fenómeno modelado; ya que los tiempos de cómputo típicamente presentan una alta variabilidad, y las situaciones del peor escenario son inusuales. La consecuencia inmediata a este tipo de análisis es la subutilización de los recursos de los que dispone la plataforma multiprocesador sobre la que se realiza la asignación de tareas (o sus instancias) la mayor parte del tiempo; lo que a su vez conlleva a un impacto significativo en el costo total del desarrollo de un sistema que emplee este tipo de análisis [14].

En promedio, las tareas no siempre hacen uso total de sus peores tiempos de cómputo [14], lo que hace suponer la existencia de intervalos de tiempo ocioso la mayor parte del tiempo, cuando el análisis se basa en el peor caso de ejecución. En particular, el trabajo presentado en este capítulo se enfoca en tareas que concluyen su ejecución en menos tiempo del establecido en su WCET, para ello se propone un análisis de planificabilidad que se basa en el uso de la esperanza matemática de la utilización de una plataforma homogénea, con esto logra un incremento en la eficiencia del uso de los recursos de una plataforma (en términos del número de tareas ejecutadas: si éstas aumentan, aumenta la

utilización de la plataforma); sin embargo, el análisis propuesto no ofrece una garantía para cumplir los plazos.

Para los fines del trabajo presentado en este capítulo, se consideran algoritmos *offline* que se realizan en dos etapas: En la primera se genera un acomodo factible de tareas (se genera un registro con los tiempos de comienzo, fin y *preemptions* de cada tarea cumpliendo con los deadlines) y, en la segunda se asignan las tareas a los procesadores en tiempo de ejecución (esto es *online*).

En este capítulo se presenta una estrategia para el aprovechamiento de intervalos de tiempo en los que un procesador no tiene trabajo que realizar, de ahora en adelante denominados intervalos *ociosos*, dichos intervalos son causados por algún algoritmo de planificación en tiempo de ejecución. Ya que los WCETs son los valores máximos de cómputo y que las tareas pueden terminar antes de lo previsto, entonces se puede tener un amplio rango de posibles intervalos ociosos. Es decir, no es posible determinar exactamente el tamaño de los intervalos de tiempo ocioso que podrían aparecer, lo que implica un comportamiento aleatorio para estos.

En tiempo de ejecución (la segunda etapa), cuando alguna tarea termina antes de su WCET, los algoritmos *offline* son incapaces de reacomodar las tareas para lograr ocupar los intervalos ociosos. Por lo tanto, se aborda el problema de aprovechar los intervalos ociosos en tiempo de ejecución mediante la inclusión de un nuevo conjunto de tareas. Las tareas nuevas, con las que se ocuparan los intervalos ociosos, son *job-shops* cuyos tiempos de cómputo también tienen un comportamiento aleatorio y las instancias de tareas (*jobs*) que los conforman siguen relaciones de precedencia. Asimismo, estas nuevas tareas tienen plazos que siguen un comportamiento aleatorio, que se supone sigue una distribución *uniforme*, los cuales deben ser satisfechos. Para lidiar con este problema se propone un método heurístico novedoso basado en el uso de redes bayesianas.

En este capítulo se presentan dos contribuciones principales. La primera es la descripción de como hacer el mapeo del problema de planificación hacia una red bayesiana. La segunda es la definición de la metodología de selección del mejor job, del nuevo conjunto de tareas, que ocupará el intervalo ocioso dejado durante el proceso de ejecución.

Los objetivos que se pretenden alcanzar con la estrategia propuesta son la reducción del tiempo ocioso total global dentro de un intervalo, así como encontrar un acomodo factible de jobs (de por lo menos un job-shop), del nuevo conjunto de tareas, tal que cumplan sus plazos.

5.1. El problema

Los análisis de planificabilidad para algoritmos deterministas (p.e. RUN, P-fair, RM, EDF, DM, BF, etc.) con plazos *hard* se basan en la consideración de condiciones pesimistas de acuerdo al algoritmo, por ejemplo, especifican si se puede llevar a cabo la planificación considerando los peores tiempos de cómputo de las tareas (WCET), el peor offset de las tareas periódicas de un conjunto, entre otras.

En promedio, en el caso de los peores tiempos de cómputo (WCET), durante la ejecución, no siempre se utiliza todo el tiempo reservado [12], [13], [14]. Este trabajo se inspira en el aprovechamiento de los intervalos ociosos dejados por los algoritmos deterministas offline durante su etapa de ejecución.

El trabajo propuesto está enfocado en el mejor uso de los intervalos ociosos dejados en tiempo de ejecución por algoritmos de planificación deterministas *offline* que consideran WCET, que producen acomodos factibles para un conjunto de n tareas periódicas sobre una plataforma multiprocesador con m procesadores idénticos, donde $n,\ m \in \mathbb{N}$ y $1 \leq m \leq n$, se llamará $\tau^{\mathcal{A}}$ al conjunto de tareas periódicas planificadas (offline) por el algoritmo $\mathcal{A}$.

El problema aparece cuando se hace la suposición de que el tiempo de cómputo que requiere cada tarea $\tau_i' \in \tau^{\mathcal{A}}$ puede ser menor a su peor tiempo de ejecución, de ahora en adelante se referirá a éste como WCET (Worst Case Execution Time). De manera más precisa, durante la etapa de ejecución (online) del algoritmo $\mathcal{A}$, cada tarea $\tau_i' \in \tau^{\mathcal{A}}$ podría terminar su ejecución con un valor c_i' tal que $0 < c_i' \leq C_i'$ donde C_i' es el peor caso de ejecución de τ_i'. Otra observación importante es que el valor que toma c_i' es conocido en tiempo de ejecución. Nótese que, bajo dicho supuesto, la plataforma puede ser infrautilizada. Ya que el tiempo de ejecución real de cada tarea en $\tau^{\mathcal{A}}$ es desconocido, entonces los intervalos ociosos tendrán un comportamiento estocástico. La aparición de los mencionados intervalos ociosos da la pauta para introducir un nuevo conjunto de tareas, τ, que se intentará planificar online dentro de los intervalos ociosos. Se asumirá que $\tau \neq \tau^{\mathcal{A}}$.

τ consta de tareas con comportamiento estocástico: tanto sus plazos como sus requerimientos de ejecución son desconocidos (ver sección 5.2.1 para más detalles sobre τ) hasta su ejecución. De antemano se conocen los posibles tiempos de ejecución de cada job $\tau_{i,j} \in \tau_i$; se supondrá que los valores de los requerimientos de ejecución se obtuvieron por la ejecución aislada de cada tau_i. Por lo anteriormente dicho, las restricciones temporales son manejadas como variables aleatorias discretas. Nótese que el problema abordado es completamente estocástico (fully stochastic) [50], por ello se utilizaran métodos de inferencia estadística para aprovechar los intervalos ociosos disponibles.

Este trabajo propone una estrategia para acomodar la carga adicional de trabajo representada por τ. Las tareas en τ sólo tienen un plazo [51] y restricciones temporales estocásticas, a saber, sus requerimientos de ejecución y sus plazos. Note que, en el proceso offline, no es posible planificar de manera conjunta a $\tau^{\mathcal{A}}$ y τ ya que el algoritmo $\mathcal{A}$ esta diseñado para trabajar solamente con tareas periódicas. Los tiempos de liberación de las tareas τ_i son conocidos durante el tiempo de ejecución.

Como fue mencionado antes, la planificación se llevará a cabo sobre una plataforma idéntica; también se utilizaran redes bayesianas para hacer inferencia durante el proceso online de $\mathcal{A}$. El seguimiento de la ejecución de las tareas en τ se hará el el intervalo $[t_{ini}, t_{end})$, donde t_{ini} es el instante en el que comienza el algoritmo y

$$t_{end} := \text{máx}\{\max_{i=1,\ldots,n}\{R_{D_i}\}\} \tag{5.1}$$

R_X es el conjunto de valores que puede tomar la variable aleatoria X.

5.2. Mapeo del problema a la red bayesiana

En esta sección se presentan las diferentes partes que modelan el problema y como éstas se interconectan para formar una red bayesiana que ayudará a la detección del "mejor" job cada vez que sea necesario. Este modelo consta de un grafo acíclico dirigido (DAG), el cual está formado por nodos que son variables aleatorias y sus respectivos vínculos.

Una red bayesiana es una tupla *(D,P)* donde D es un DAG y

$$P := \{P(X_i|\pi(X_i))|\ i = 1,\ \ldots,\ n\}$$

que es el conjunto de n funciones de probabilidad condicional, para toda variable aleatoria en D, y $\pi(X_i)$ es el conjunto de predecesores de X_i, donde $X_i \in D$. El conjunto P define una distribución de probabilidad conjunta según se observa a continuación (regla de la cadena)

$$P(X) := \Pi_{i=1}^{n} P(X_i|\pi(X_i)) \tag{5.2}$$

El algoritmo utilizado para hacer inferencia es el algoritmo de *Pearl* [50]. El proceso de inferencia se realiza sobre un poliárbol[1] (polytree, en inglés) que es un tipo de DAG. Por lo tanto, es necesario identificar las dos partes que conforman al modelo: la primera es un conjunto de vértices que está conformado por los intervalos ociosos, a si como por un conjunto de jobs provenientes de las tareas en τ; y la segunda, un conjunto de aristas que son las relaciones entre los intervalos ociosos y los jobs. En las siguientes secciones se describe cada componente y sus interacciones.

5.2.1. Nodos: Tareas e intervalos ociosos

Para este trabajo, dos aspectos con comportamiento aleatorio fueron elegidos para ser modelados: El primero representa a los jobs en τ y el segundo representa a los intervalos ociosos.

El nuevo conjunto, τ, que es diferente de $\tau^{\mathcal{A}}$, está compuesto por n tareas. Las tareas pueden ser de dos tipos: aquellas que están compuestas por un sólo job y aquellas tareas compuestas por un conjunto finito de jobs ordenados, *job-shops*. El número de jobs en la tarea τ_i se denota como n_i, donde $n_i \in \mathbb{N}$. Cada tarea tiene sus propias restricciones temporales tal como un plazo y múltiples requerimientos de ejecución (uno por cada $\tau_{i,j} \in \tau_i$ con $j = 1,\ \ldots,\ n_i$).

Un job es una secuencia de instrucciones que es ejecutada sin interrupción, es decir, son *non-preemptive*. El j^{esimo} job de la i^{esima} tarea se representa como $\tau_{i,j}$, donde $i = 1, \ldots,\ n$ y $j = 1, \ldots,\ n_i$.

Una tarea de tipo job-shop, τ_i, es una secuencia ordenada de jobs, o, de manera más precisa,

$$\tau_i := \{\tau_{i,j} : 1 \leq i \leq n,\ 2 \leq j \leq n_i\}$$

[1] Grafos simplemente conectados en los que un nodo puede tener más de un padre.

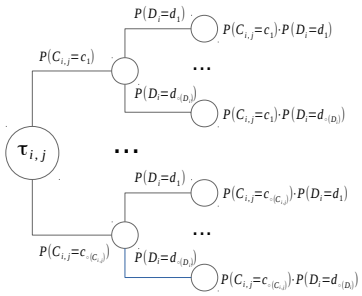


Figura 5.1: Variables aleatorias que describen al job $\tau_{i,j}$

donde n_i es un número pequeño que indica la longitud de un job-shop. Dichos jobs siguen *relaciones de precedencia*, en otras palabras, $\tau_{i,j}$ no puede comenzar su ejecución si el job previo, $\tau_{i,j-1}$, no ha finalizado previamente.

Cada tarea se comporta de manera aleatoria debido a sus restricciones temporales: tiempo de cómputo de cada uno de los jobs que la conforman y plazo. La variación en el tiempo de cómputo se explica a partir de la diferencia en las cantidades de datos entrantes en diferentes instantes. En cuanto a los plazos, la variación se debe a que se considera que la ejecución de una tarea se realiza en tiempo si ésta concluye dentro de un rango de tiempo permitido.

Para mantener la coherencia del modelo, las tareas deben cumplir con la siguiente restricción. Para toda $\tau_i \in \tau$ se cumple que

$$\sum_{j=1}^{n_i} \text{máx}\{R_{C_{i,j}}\} \leq \text{mín}\{R_{D_i}\}$$

donde R_X denota el rango[2] de la variable aleatoria X.

Las restricciones temporales de cada job $\tau_{i,j} \in \tau_i$ son representadas por las variables aleatorias discretas $C_{i,j}$ y D_i las cuales representan a su posible consumo (requerimiento de ejecución) y plazo con $j = 1, \ldots, n_i$, respectivamente. Note que los jobs pertenecientes a la misma tarea comparten el mismo plazo.

Como se muestra en la figura 5.1, cada job $\tau_{i,j}$ es modelado a través de la tupla $(C_{i,j}, D_i)$ cuyas v.a. son independientes. Por la regla del producto se tienen $|R_{C_{i,j}}| \cdot |R_{D_i}|$ posibles valores para la tupla correspondiente a $\tau_{i,j}$.

Otra restricción temporal considerada para cada $\tau_{i,j}$ es el tiempo de inicio que es representado por $s_{i,j}$ tal que $s_{i,j} \in \mathbb{N} \cup \{0\}$. Se asumirá que las tareas aparecerán durante el intervalo $[t_{ini}, t_{end})$, es decir, la tarea τ_i liberada en el instante r_i debe satisfacer la restricción $0 \leq t_{ini} \leq r_i < t_{end}$.

[2]También conocida como la imagen de una variable aleatoria, es el conjunto de los valores reales que una variable puede tomar

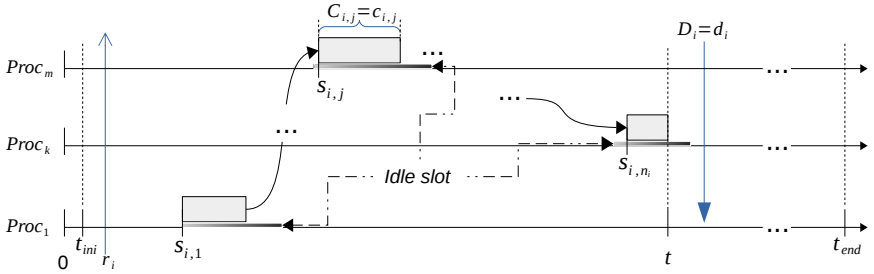


Figura 5.2: Parámetros que conforman al modelo de tareas

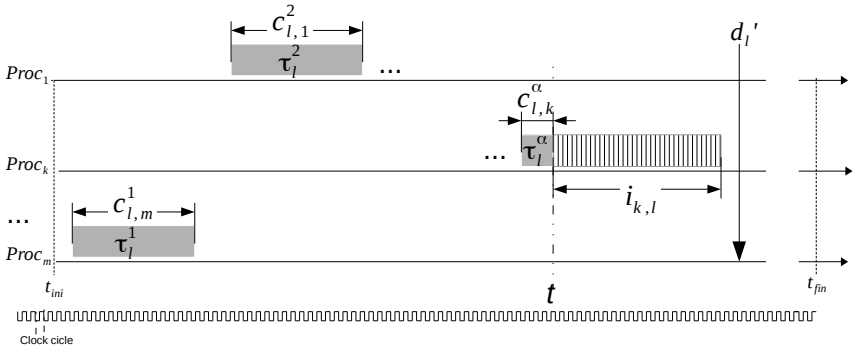


Figura 5.3: Obtención del valor de la variable aleatoria $I_{k,l}$

Para terminar la formalización del modelo de tareas, el orden de ejecución de los jobs de un job-shop tiene que satisfacer la siguiente restricción: Dados dos jobs $\tau_{i,j},\ \tau_{i,j+1} \in \tau_i$, entonces

$$s_{i,j} + c_{i,j} \leq s_{i,j+1}$$

donde τ_i es un job-shop; $j = 1,\ \ldots, n_i$; y $c_{i,j}$ es una realización en $R_{C_{i,j}}$.

Esta estrategia trabaja sobre un *esquema particionado*. Cuando una tarea en τ ha sido liberada, sus respectivos jobs constitutivos son asignados de manera arbitraria a los diferentes procesadores de la plataforma. Cada procesador tiene una cola en donde los jobs esperan por su ejecución, a la cola del procesador k ($Proc_k$) se le denota como $\tau_{wait}(Proc_k)$.

Un *intervalo ocioso* es un subintervalo $[a, b) \in [t_{ini},\ t_{end})$ en el procesador k, denotado por $Proc_k$, el cual está reservado para alguna tarea en $\tau_l' \in \tau^{\mathcal{A}}$, en el que τ_l' no se ejecuta más debido a su finalización. Se utilizará $I_{k,l}$ para denotar a un posible intervalo ocioso, dejado por la tarea τ_l' en $Proc_k$, donde $l = 1,\ 2,\ \ldots$, el índice l indica el número del posible intervalo ocioso ocurrido en $Proc_k$, donde $k = 1,\ \ldots,\ m$. La variable aleatoria $I_{k,l}$ se convierte en un nodo observado en el instante en el que la tarea $\tau_l' \in \tau^{\mathcal{A}}$ termina su ejecución (vea la figura 5.3). Esto significa que en ese mismo instante la variable $I_{k,l}$ toma el valor $i_{k,l}$ ($I_{k,l} = i_{k,l}$), obteniendo evidencia fuerte que será propagada a través del poliárbol formado hasta ese instante para la elección del "mejor" job en $\tau_{wait}(Proc_k)$.

Cuando una tarea τ_l' termina su ejecución antes de su WCET; el algoritmo a cargo, $\mathcal{A}$, debe lanzar una interrupción debido a la finalización prematura de τ_l'.

El l^{esimo} intervalo de longitud aleatoria, $I_{k,l}$, puede tener tantas realizaciones como rea-

lizaciones tenga el requerimiento de ejecución de la tarea $\tau'_l \in \tau^{\mathcal{A}}$. De lo previamente mencionado se deriva la siguiente definición.

Definición 6 (Realizaciones del intervalo $I_{k,l}$) *El rango de la variable aleatoria $I_{k,l}$ que describe al intervalo ocioso l en el procesador k, denotado por $R_{I_{k,l}}$, depende del rango de la variable aleatoria C'_l de τ'_l, denotado por $R_{C'_i}$. Entonces $R_{I_{k,l}}$ se computa así*

$$R_{I_{k,l}} := \{i_{k,l} \in \mathbb{N} \cup \{0\} | \forall c_r \in R_{C'_i} \ : \ i_{k,l} = \max_{c'_i \in R_{C'_i}} \{c'_i\} - c_r\} \tag{5.3}$$

donde C'_i es la variable aleatoria que describe los posibles valores que toma el requerimiento de ejecución de $\tau'_i \in \tau^{\mathcal{A}}$.

Durante la etapa de ejecución de $\mathcal{A}$, la red bayesiana, sobre la que se realiza el proceso de inferencia, no permanece estática ya que se modifica tanto con la aparición de un intervalo ocioso como con la anexión de cada nuevo mejor job seleccionado. Por lo tanto, se añadirán más y más relaciones conforme el sistema entero evoluciona con el tiempo.

Definición 7 (Tarea fiable) *Una tarea de tipo job-shop se calificará como fiable cuando ésta satisface la siguiente restricción en el instante t:*

$$t + \sum_{j=1}^{n_i^{(t)}} E[C_{i,j}] < E[D_i] \tag{5.4}$$

donde $n_i^{(t)}$ es el número de jobs en $tau_i \in \tau$ que todavía no han sido planificados y, por ende, todavía siguen esperando su ejecución en el instante t. Sea $I_{k,l}$ la variable aleatoria que hace referencia al l^{esimo} intervalo ocioso. Además, τ_i debe satisfacer la siguiente condición

$$\sum_{j=1}^{n_i^{(t)}} E[C_{i,j}] < i_{k,l} + \sum_{I_k}^{N^{(t)}} E[I_{k,l}] \tag{5.5}$$

donde $N_{I_k}^{(t)}$ es el número estimado de posibles intervalos ociosos que pueden ocurrir después del instante t en $Proc_k$.

5.2.2. Relaciones

Para hacer inferencia es necesario definir como se establecen las relaciones entre los nodos que conforman al poliárbol. El aprendizaje de una red bayesiana comienza con las distribuciones de probabilidad condicional a priori de cada uno de los nodos en el poliárbol, a las cuales nos referiremos como *tablas de probabilidad condicional*, CPT. El tamaño de la CPT crece exponencialmente con el número de padres de un nodo y el tamaño del rango de los padres (vistos como variables aleatorias), por lo que puede crecer demasiado. Sin embargo, para evitar un crecimiento desmedido de la tabla, se ha limitado este modelo a

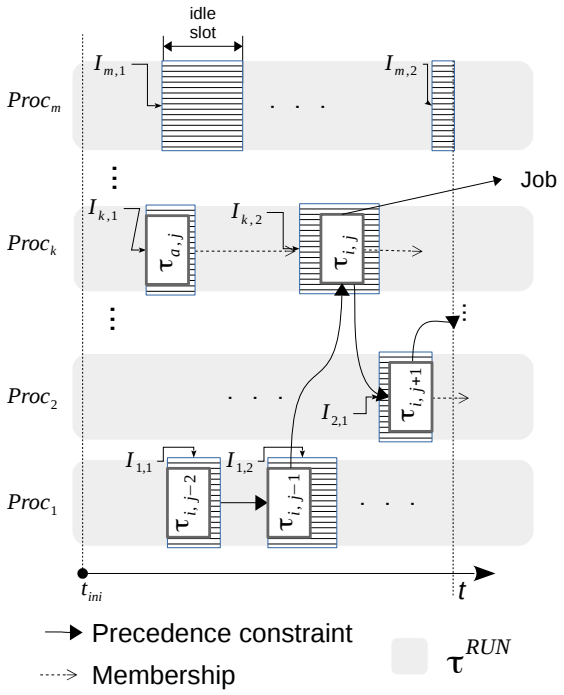


Figura 5.4: Tres tipos de relaciones que se pueden establecer con el job $\tau_{i,j}$

que un nodo pueda tener un máximo de tres padres. La figura 5.4 muestra los tres tipos de relaciones que se pueden llegar a formar.

A continuación se describen los tres tipos de relaciones y las condiciones bajo las cuales se establecen: El primer tipo, representado por una línea discontinua en la figura 5.4, se establece de acuerdo a la ubicación de ejecución de los jobs. Este tipo se establece cuando dos jobs se ejecutan uno detrás del otro en el mismo procesador y se establece durante la etapa de ejecución del algoritmo $\mathcal{A}$. El segundo tipo, representado por una línea continua en la figura 5.4, se establece de acuerdo a las relaciones de precedencia de cada job-shop. El tercer tipo surge de la observación de que la existencia de un job en $\tau_i \in \tau$ en la red es debida a que hay un intervalo ocioso, por lo tanto, los jobs dependen de un intervalo ocioso para existir. Como una primera aproximación, la relación entre un intervalo ocioso y un job se establecerá sólo entre el intervalo ocioso y el primer job que se ejecute dentro de dicho intervalo, véase como en la figura 5.5 sólo se establece la relación entre $I_{k,l}$ y $\tau_{i,j}$. Ya que es necesario mantener la estructura de poliárbol para la ejecución del algoritmo de inferencia, esta manera de establecer la tercera relación evita que haya ciclos en la red (formada hasta cierto instante t).

La red, que se va formando agregando los diferentes nodos durante el tiempo de ejecución, no es estática. Por lo tanto, conforme transcurre el tiempo, aparezcan intervalos ociosos y los job-shops se vayan liberando, la red se volverá más y más grande.

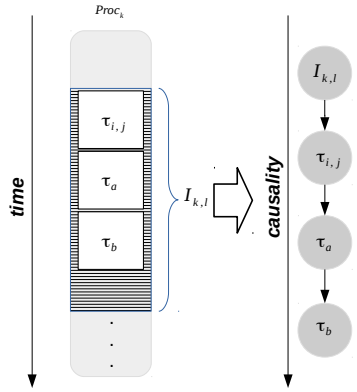


Figura 5.5: Configuración permitida en la red bayesiana. A la izquierda se muestra la ejecución de jobs en $I_{k,l}$; a la derecha, su representación en la red.

5.2.3. Especificación de la plataforma

Para este trabajo, la plataforma, representada como *Proc*, está compuesta por m procesadores idénticos, es decir, los procesadores tienen la misma velocidad, donde $2 \leq m \leq \sum_{i=1}^{n} n_i$ y n es el número de tareas que serán liberadas durante el intervalo $[t_{ini}, t_{end})$.

Los procesadores en *Proc* están completamente sincronizados y cada uno tiene su propio planificador local, que es el encargado de elegir el siguiente "mejor" job por medio de inferencia bayesiana sobre un poliárbol.

Decimos que una plataforma es idéntica cuando para cualesquiera dos procesadores $Proc_p$ y $Proc_q \in Proc$ con tasas de rapidez μ_p y μ_q se satisface que $\mu_p = \mu_q$; además, los procesadores tienen la misma arquitectura subyacente.

Para el manejo de los jobs, cada procesador $Proc_K$ tiene dos listas, a las que se referirá como $\tau_{wait}(Proc_k)$ y $\tau_{dead}(\tau_{wait})$. Éstas están compuestas por los conjuntos de jobs en espera de ser ejecutados y el de jobs finalizados, respectivamente. La lista $\tau_{wait}(Proc_k)$ contiene jobs que ya han sido liberados y que son fiables (ver definición en sección 5.2.1).

5.3. Descripción de la estrategia

Se supondrá que el algoritmo $\mathcal{A}$ consta de dos etapas. La primera, que se lleva a cabo offline, consiste en la generación de una tabla con los registros de la información correspondiente a los tiempos de inicio, final, suspensión (preemption) y reanudación de cada tarea $\tau_l' \in \tau^{\mathcal{A}}$, dichos registros son obtenidos antes de llevar a cabo la segunda etapa. Una vez creada, la mencionada tabla se utiliza como entrada de la segunda etapa, que consiste en la lectura de los registros para ejecutar, de manera online, las tareas correspondientes en los instantes indicados.

El objetivo de esta estrategia es el de aprovechar los intervalos ociosos que dejan las tareas en $\tau^{\mathcal{A}}$ durante la etapa de ejecución; lo que implica que la estrategia propuesta sea *online*

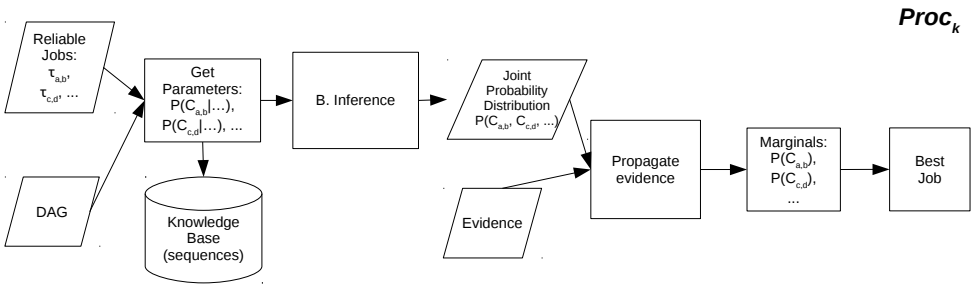


Figura 5.6: Diagrama de flujo del proceso de inferencia sobre la red

necesariamente.

Es durante la segunda etapa de $\mathcal{A}$ que los intervalos ociosos aparecen; para lograr su aprovechamiento es necesario modificar $\mathcal{A}$. Cada vez que una tarea $\tau_l' \in \tau^{\mathcal{A}}$ termina en menos tiempo que su WCET, entonces $\mathcal{A}$ debe lanzar una interrupción para comenzar un procedimiento basado en redes bayesianas; de ahora en adelante se le nombrará $\mathcal{A}$-*modificado*.

$\mathcal{A}$-*modificado* consiste en añadir un procedimiento extra a $\mathcal{A}$: Un procedimiento de inferencia bayesiana, a saber, el algoritmo de Pearl [50]. Este procedimiento se efectúa en el procesador que lanza la interrupción debida a $\tau_l' \in \tau^{\mathcal{A}}$ que ha terminado antes de lo estimado (antes de su WCET); sea $Proc_k$ dicho procesador.

5.3.1. $\mathcal{A}$-*modificado*

Hasta ahora se sabe que la estrategia de planificación propuesta utiliza un esquema particionado, es decir, para todo $Proc_k$, con $k = 1, \ldots, m$, existe un conjunto $\tau_{wait}(Proc_k)$ que contiene a los jobs que se encuentran en espera para su ejecución.

Para comprender la siguiente explicación es necesario definir el termino *secuencia*.

Definición 8 (Secuencia) *Una* ***secuencia*** *es un arreglo de eventos que ocurren en el mismo procesador. Estos eventos se encuentran ordenados de acuerdo a su ocurrencia. Se representa como* $(X_1^{(t)}, X_2^{(t+1)}, \ldots, X_{i-1}^{t+i}, \ldots)$*, donde* t *indica el instante en el que ocurrió el primer evento,* X_1*, que siempre es un intervalo ocioso.*

La tupla realmente indica la dependencia entre eventos: Si $i, j \in \mathbb{N}$ y $j > i$, entonces X_i tuvo que existir primero para que X_j existiera después; sin embargo, se prescinde del tiempo en el que ocurrieron los eventos y sólo se consideran las relaciones causales para el proceso de inferencia (ver figura 5.5). Los eventos pueden ser, ya sea, intervalos ociosos o requerimientos de ejecución de los jobs.

Regla 1 (Sobre el establecimiento de relaciones en la red bayesiana) *Las relaciones sólo se pueden establecer entre las variables aleatorias de los consumos (requerimientos*

de ejecución) de los jobs; las variables aleatorias de los intervalos ociosos y variables aleatorias de los consumos de los jobs, pero nunca entre variables aleatorias de los intervalos ociosos.

Las secuencias se obtienen después de la ejecución de una instancia (entrada de $\mathcal{A}$) en el intervalo $[t_{ini}, t_{end})$ y son almacenadas en una base de conocimiento. De secuencia se deriva la siguiente definición.

Definición 9 (Subsecuencia) *Una **subsecuencia** es un subconjunto de eventos que siguen el mismo orden que existe en una secuencia almacenada en la base de conocimiento.*

Si el conjunto local de jobs en espera de $Proc_k$, $\tau_{wait}(Proc_k)$, aún contiene jobs fiables (por ejemplo, los jobs $\tau_{a,b}$ y $\tau_{c,d}$ de la figura 5.6) y se lanza una interrupción en cierto instante, entonces se calculan los parámetros correspondientes (conditional probability tables), uno por variable aleatoria (según se observa en la figura 5.6, los parámetros correspondientes son $P(C_{a,b}|\pi(C_{a,b}))$, $P(C_{c,d}|\pi(C_{d,c}))$, etc.).

Una red bayesiana es un modelo que consiste en un grafo DAG y parámetros que son las distribuciones de probabilidad condicional que definen la distribución de probabilidad de un nodo dados sus padres. Estos parámetros se obtienen de la base de conocimiento, por medio del conteo de las frecuencias relativas (a la muestra almacenada en la base) de las subsecuencias.

A pesar de que el tamaño de los parámetros es exponencial de acuerdo al número de padres, el tamaño de estos es controlado, en este trabajo, limitando el número de vínculos en, a lo más, tres (i.e. a lo más, puede haber tres padres), explicados en la sección 5.2.2.

Una vez que los parámetros han sido calculados, se construye una función de distribución de probabilidad conjunta, abreviado como f.p.c. (siguiendo el ejemplo de la figura 5.6, la f.p.c. es por la regla de la cadena $P(C_{a,b}, C_{c,d}, \ldots) = P(C_{a,b}|\pi(C_{a,b})) \cdot P(C_{c,d}|\pi(C_{d,c})) \cdot \ldots)$. Después la evidencia (*hard evidence*) es propagada y la f.p.c. se ajusta. Finalmente, para cada job fiable $\tau_{i,j} \in \tau_{wait}(Proc_k)$, se calcula su correspondiente función de distribución de probabilidad *marginal* (ver figura 5.6: $P(C_{a,b}),\ P(C_{c,d}),\ \ldots$).

De aquellas distribuciones de probabilidad marginal se elige el **mejor** job según se describe a continuación.

Definición 10 (El mejor job) *Para todo job $\tau_{i,j} \in \tau_{wait}(Proc_k)$, con requerimiento de ejecución (consumo) $C_{i,j}$, un job fiable es considerado el **mejor** cuando la siguiente condición se satisfaga localmente en $Prock_k$*

$$\max_{\forall \tau_{i,j} \in \tau_{wait}(Prodc_k)} \left\{ P\left(C_{i,j} \leq I_{k,l} - \sum_{\tau_{a,b} \in \tau_{dead}(Proc_k)} C_{a,b} \right) \right\} \tag{5.6}$$

donde $k = 1,\ \ldots,\ m$ y $C_{a,b}$ son las realizaciones de los consumos de $\tau_{a,b}$

Después del proceso de inferencia bayesiana, para cada job fiable $\tau_{i,j} \in \tau_{wait}(Proc_k)$, una distribución de probabilidad marginal $P(C_{i,j})$ es calculada y, luego, utilizada para elegir

el job que ofrece la mayor probabilidad de ser ejecutado en el mismo o menor tiempo que tomó el valor del intervalo $I_{k,l}$.

Una vez que una ejecución haya ocurrido dentro de todo el intervalo $[t_{ini}, t_{end})$, se obtienen nuevas secuencias (una por procesador) y éstas son almacenadas en la base de conocimiento. Mayor número de secuencias en la base de conocimiento ayudan a mejorar el entrenamiento de la red.

En este trabajo, se asume que el proceso de inferencia se efectúa de manera instantánea. Por ende, el tiempo de ejecución para este proceso en particular es nulo; aunque, es bien sabido que el algoritmo de Pearl para poliárboles toma tiempo pseudopolinomial [52], [50], [53]. El algoritmo de propagación de Pearl se efectúa sobre grafos acíclicos dirigidos sin dependencias redundantes y, dado que las tareas siguen un esquema particionado, es altamente importante poner atención en como son asignados los jobs entre los procesadores.

Mientras haya jobs en espera en los procesadores que conforman a la plataforma y una interrupción sea lanzada, la función SELECT_NEXT_JOB_$Proc_k$(), descrita en el Algoritmo 3, comienza inicializando el conjunto *setReliables* (linea 1). Esta estructura de datos almacena los resultados del proceso de inferencia bayesiana de los jobs fiables. En la linea 1 se obtienen todos los jobs fiables del procesador $Proc_k$ del conjunto $\tau_{wait}(Proc_k)$ en el instante en el que la interrupción fue lanzada. Luego, estos jobs son almacenados en la estructura de datos *reliables* (linea 2). Para cada job en el conjunto *reliables*, el proceso de inferencia bayesiana será ejecutado con la red formada hasta ese momento (vea linea 5). Esto es, el proceso de inferencia bayesiana será levado a cabo tantas veces como jobs en *reliables* haya. Una vez que el proceso de inferencia bayesiana haya finalizado; con cada job fiable, se obtiene una distribución de probabilidad conjunta distinta. De estas distribuciones de probabilidad conjunta, se elige el job que ofrece la mayor de las probabilidades de ejecución con el conocimiento de eventos previos (realizaciones de los consumos de otros jobs ya acomodados y realizaciones de los intervalos ociosos que ya aparecieron), dentro de un intervalo ocioso dejado por una tarea en el conjunto $\tau^{\mathcal{A}}$ (vea linea 7). Al final, el mejor job es agregado a la red bayesiana.

En resumen, cada vez que un conjunto de jobs ha sido seleccionado como fiable en un procesador en el instante t, es momento de inferir por medio de inferencia bayesiana cual de los jobs fiables, dentro de dicho procesador, es el mejor por medio de inferencia bayesiana. Para ser más específico, el proceso de inferencia utilizado es el algoritmo de Pearl sobre poliárboles [50].

Note que, para cuando el algoritmo SELECT_NEXT_JOB_$Proc_k$() termina su ejecución en el procesador $Proc_k$, se obtiene una red de mayor tamaño (con un job más). Entonces si el proceso se lanzara en todos los procesadores en el mismo instante de tiempo, la red terminaría con a lo más m nuevos jobs.

El algoritmo SELECT_BEST_JOB() (Algoritmo 4) muestra la forma en la que la función de distribución de probabilidad respectiva a cada job fiable $\tau_{i,j}$ en $Proc_k$, denotada por $pdf_{i,j}$, es calculada en la linea 2.

La función *getReliableJobs*() (vea linea 2 del Algoritmo SELECT_NEXT_JOB_$Proc_k$()) refleja el criterio anteriormente mencionado para determinar que una tarea $\tau_i \in \tau$ es fiable.

Algorithm 3 SELECT_NEXT_JOB_$Proc_k$ ($bayesianNetwork$)

Require: $bayesianNetwork$ // Bayesian network
1: $distReliables \leftarrow [\,]$
2: $reliables \leftarrow getReliableJobs(\tau_{wait}(Proc_k))$
3: **if** $length(reliables) > 1$ **then**
4: **for all** $\tau_{i,j} \in reliables$ **do**
5: $distReliables \leftarrow distReliables \cup inference(\tau_{i,j}, bayesianNetwork)$
6: **end for**
7: $job \leftarrow SELECT\_BEST\_JOB(distReliables)$
8: $bayesianNetwork \leftarrow bayesianNetwork \cup job$
9: **end if**

Algorithm 4 SELECT_BEST_JOB($reliableDensities$)

Require: $reliableDensities$ // List of probability density functions
1: **for all** $pdf_{i,j} \in reliableDensities$ **do**
2: $probDistFunc \leftarrow probDistFunc \cup P(pdf_{i,j} < I_{k,l})$
3: **end for**
4: **return** $getJob(\text{máx}(probDistFunc))$

Como ya se mencionó, esta propuesta trata de encontrar una solución para un problema "completamente estocástico"; por lo tanto, existen un gran número de posibles escenarios que pueden ser construidos a partir de la misma instancia. Esta solución, por si misma, no garantiza la totalidad de la ejecución de nuevas tareas entrantes ya que es de naturaleza heurística; sin embargo, puede reducir el riesgo de elegir un job que no podrá ser ejecutado más tarde. La precisión para determinar al mejor job depende de la calidad de la información almacenada en la base de conocimiento, dicha información (secuencias) se obtiene conforme al número de experimentos que se llevan a cabo.

5.4. Función objetivo

El fin último de esta estrategia heurística es minimizar el tiempo ocioso *global* en el intervalo $[t_{ini}, t_{end})$, para ello se utiliza la ecuación (5.7).

$$J = \sum_{k=1}^{m} \left(\sum_{l=1}^{n_{I_k}} i_{k,l} - \sum_{\forall \tau_{i,j} \in \tau_{dead}(Proc_k)} c_{i,j} \right) \tag{5.7}$$

En la que $i_{k,l}$ y $c_{i,j}$ son las realizaciones de las variables aleatorias $I_{k,l}$ y $C_{i,j}$, respectivamente. Los valores de todas las variables aleatorias que conforman a una instancia se conocen al final de un experimento.

Para todo procesador $Proc_k$, donde $k = 1, \ldots, m$, se contabiliza el tiempo ocioso total (primer termino dentro del paréntesis) y, además, se contabiliza el tiempo aprovechado por nuevas tareas en τ (solo aquellas tareas cuyos jobs fueron ejecutados en su totalidad).

La función objetivo está condicionada a las siguientes restricciones:

$$\sum_{i=1}^{n_i} c_i < d_i, \ \forall \tau_i \in \tau \tag{5.8}$$

donde D_i es la variable aleatoria con la que se modela el comportamiento del deadline del job $\tau_{i,j}$, $\tau_{dead}(Proc_k)$ es el subconjunto de tareas en τ que fueron ejecutadas en $Proc_k$ y n_{I_k} es el número de intervalos ociosos contabilizados en $Proc_k$ al final de un experimento.

5.5. Cotas

A continuación se establecen cotas para delimitar el número de posibles configuraciones que se pueden obtener de una instancia con N_I variables aleatorias con las cuales se representa a los intervalos ociosos.

Es importante considerar la dificultad inducida por las relaciones de precedencia ya que éstas pueden tener una influencia importante sobre cuándo son asignados los nuevos jobs a cualquiera de los intervalos de tiempo ocioso, los cuales pueden ser dejados por alguna tarea en $\tau^{\mathcal{A}}$.

5.5.1. Número de posibles instancias

El siguiente límite superior para el número de configuraciones de una instancia se calcula a partir de los siguientes supuestos:

1. Para cada procesador $Proc_k$ con $k = 1, \ \ldots, \ m$, existe un número de intervalos ociosos mayor o igual al número de jobs en $\tau_{wait}(Proc_k)$, es decir, se cumple $N_{I_k} > |\tau_{wait}(Proc_k)|$.
2. Para cada job $\tau_{i,j} \in \tau_{wait}(Proc_k)$, se cumple que máx$\{R_{C_{i,j}}\} \leq$ mín$\{R_{I_{k,l}}\}$, es decir, un job siempre se puede insertar en un intervalo ocioso.
3. Los jobs no tienen relaciones de precedencia.
4. Para cada job $\tau'_l \in \tau_{\mathcal{A}}$, se tiene un límite en el número de suspensiones (preemptions), dicho límite se denota con N_p (note que los únicos trabajos que ofrecen una cota para este parámetro son [4] y [8]).

Primero se acotará el número de posibles intervalos ociosos que podrían aparecer y para ello se considera la suposición 4:

$$N_I = \sum_{k=1}^{m} N_{I_k} \leq N_p \cdot \sum_{i=1}^{|\tau^{\mathcal{A}}|} \left\lceil \frac{t_{end} - t_{ini}}{T_i} \right\rceil$$

De la suposiciones 1, 2 y 3 se deduce que, para un solo procesador, la posibilidad de elegir $|\tau_{wait}(Proc_k)|$ intervalos ociosos de entre N_{I_k} para acomodar los jobs $\tau_{i,j} \in \tau_{wait}(Proc_k)$ es

$$\binom{N_{I_k}}{|\tau_{wait}(Proc_k)|}$$

Dado que la plataforma es homogénea, para los m procesadores, el número total de posibilidades de acomodar los jobs en los intervalos ociosos es

$$\prod_{k=1}^{m} \binom{N_{I_k}}{|\tau_{wait}(Proc_k)|} \tag{5.9}$$

Nótese que la cota superior dada es una cota muy holgada debido a que se considera el caso en el que más posibilidades de ejecución podría haber.

Capítulo 6

Resultados

Contents

6.1. Caso de estudio: RUN-modificado 57

6.2. Experimentación 58

6.2.1. Ejemplificación 58

6.2.2. Resultados generales 61

6.3. Conclusión 64

En este capítulo se proveen los resultados obtenidos de la implementación de la estrategia basada en redes bayesianas que se describe en el capítulo 5. El algoritmo sobre el que se implementará el aprovechamiento mediante redes bayesianas es RUN [4]. Como se mencionó previamente, el objetivo de este trabajo es el de reducir el tiempo ocioso global promedio (función (5.7)) en un intervalo de tiempo: $[t_{ini}, t_{end})$. En [2] se presenta completa la estrategia empleada de la cual se obtienen los resultados que a continuación se presentan.

6.1. Caso de estudio: RUN-modificado

Como se estableció en el capítulo 5 para aplicar la mejora con redes bayesianas se requiere de un algoritmo que realice la planificación de un conjunto de tareas en dos etapas: la primera, se lleva a cabo offline y su salida es una tabla o registro de los instantes de liberación, comienzo, preemption, resume y finalización de cada tarea; para que, posteriormente, en la segunda etapa, se ejecuten las acciones indicadas en dicho registro.

El algoritmo RUN [4] cumple con dichas características ya que se lleva a cabo en dos etapas. RUN no crea como tal una tabla, mas bien, durante la etapa offline genera un árbol de ejecución (a través de las operaciones *dual* y *packing*) que precisa los posibles procesadores en donde se puede ejecutar un nuevo job y el intervalo de tiempo durante el que se ejecuta. Una ventaja que presenta RUN contra otros algoritmos óptimos [6], [7], [9], [10] es que éste acota el número de preemptions de un job a, lo más, $log_2 m$ (cuando

la utilización de las tareas es muy cercano a 0.5) que, incluso, puede llegar a ser una cota menor a la planteada en [8]; por lo tanto, cuanto menor número de preemptions tenga un job en el conjunto τ^{RUN}, se tendrán secciones de mayor tamaño repartidas a lo largo de la plataforma para ejecutar dicho nuevo job.

6.2. Experimentación

6.2.1. Ejemplificación

La figura 6.1 muestra un ejemplo de una salida producida por la metodología propuesta. La entrada es una instancia pseudoaleatoriamente generada compuesta por 26 jobs que están agrupados en diez tareas: siete de ellas son job-shops y sólo tres están compuestas de un job. Cada job tiene asociada una variable aleatoria discreta que caracteriza sus requerimientos de ejecución. Los valores que pueden tomar los requerimientos de ejecución de cada uno de los jobs se eligieron dentro de un rango de valores posibles: estos oscilaban entre 1 y 27 unidades de tiempo; los valores de cada variable aleatoria se eligieron a partir de una distribución de probabilidad uniforme. Aunque en principio, los tamaños de los rangos de las variables aleatorias discretas pueden tomar valores arbitrarios; por simplicidad, se limitó el tamaño del rango de las variables aleatorias, $|R_{C_i}|$, a que fuese 4.

En cuanto a la plataforma, se estableció que ésta constara de cuatro procesadores ($m = 4$) idénticos.

Por simplicidad, las diez tareas fueron liberadas en el mismo instante de tiempo ($t_{ini} = 0$), luego, los jobs fueron repartidos entre los cuatro procesadores de la plataforma. Los jobs fueron asignados a los procesadores de manera arbitraria, por lo que, para esta instancia, se obtuvo la repartición de jobs que se muestra en la tabla 6.1. Esta asignación se utilizó para un total de 605 experimentos.

De estos 605 experimentos 456 fueron ejecutados exitosamente, esto es, al menos se pudo ejecutar un job-shop completo (todos los jobs de una tarea se ejecutaron dentro de su plazo). Por lo tanto, la razón obtenida para esta instancia indica una *persistencia* de aproximadamente *0.75* de éxito para planificar, por lo menos, una tarea.

Para los propósitos de esta ejemplificación, la simulación se realizó en el intervalo de tiempo delimitado por $t_{ini} = 0$ y $t_{end} = 1200$; dichos limites fueron elegidos de esta manera debido a que el máximo de los deadlines, del nuevo set de job-shops, tiene el valor de 1176, note que no tiene sentido analizar más allá del mencionado valor, ya que después de éste no habrá más tareas en espera, a menos que se liberen más.

En total, 605 ejecuciones fueron llevadas a cabo con la misma instancia (compuesta por 26 jobs que, a su vez, están agrupados en diez tareas, repartidos sobre una plataforma de 4 procesadores idénticos). Note que este número de experimentos conforma a una muestra muy pequeña con respecto al tamaño del espacio muestral de esta instancia. Porque, aunque haya pocas variables aleatorias, cuyos tamaños de rangos son pequeños, existe una gran número de posibles escenarios. Este problema tiene un comportamiento *superexponencial* con respecto al tamaño del conjunto de tareas, el número de intervalos

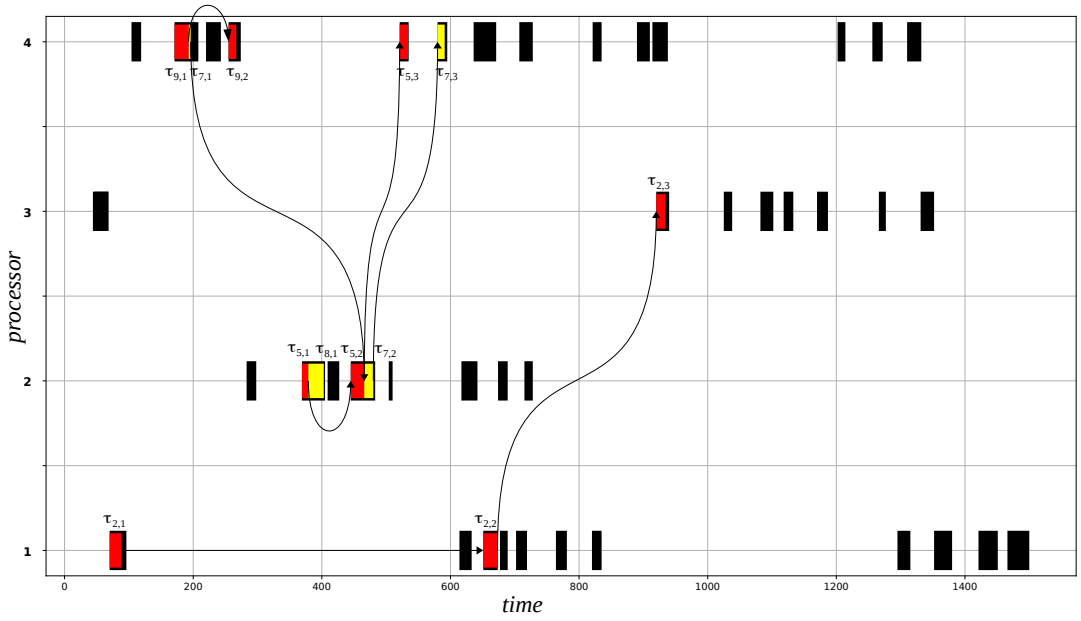


Figura 6.1: Procesador *vs.* tiempo: ejemplo de una posible salida producida usando inferencia mediante redes bayesianas. La zona de ocio se encuentra coloreada en negro (intervalo de tiempo dejado por RUN). Las tareas τ_2, τ_5, τ_7 y τ_9 son *job-shops* compuestos por al menos dos jobs (imagen tomada de [2])

Tabla 6.1: Asignación de los jobs de las diez tareas a lo largo de la plataforma

i^{th} procesador	$\tau_{wait}(Proc_i)$
$Proc_1$	$\{\tau_{2,1}, \tau_{2,2}, \tau_{6,1}, \tau_{4,1}, \tau_{3,3}, \tau_{3,4}, \tau_{3,5}\}$
$Proc_2$	$\{\tau_{1,3}, \tau_{5,1}, \tau_{5,2}, \tau_{7,2}, \tau_{8,1}, \tau_{10,3}\}$
$Proc_3$	$\{\tau_{1,2}, \tau_{2,3}, \tau_{3,1}, \tau_{3,2}, \tau_{10,2}\}$
$Proc_4$	$\{\tau_{1,1}, \tau_{1,4}, \tau_{5,3}, \tau_{7,1}, \tau_{7,3}, \tau_{9,1}, \tau_{10,1}, \tau_{9,2}\}$

ociosos y el tamaño de los rangos de las variables que describen a los intervalos ociosos y a los requerimientos de ejecución de los jobs (note que no se restringe el tamaño del rango de las variables aleatorias en ningún momento); sin embargo, para esta ejemplificación, los tamaños de los rangos de las variables aleatorias fueron elegidos iguales por simplicidad.

Al final de la ejecución de cada experimento es posible obtener un grafo como el mostrado en la figura 6.2, este grafo corresponde a un sólo escenario de la instancia antes descrita y muestra las variables aleatorias involucradas con sus correspondientes relaciones (ver 5.2.2).

Como es posible apreciar de la Figura 6.3, no fue posible completar la totalidad de los jobs; lo que es un comportamiento esperado debido a que las condiciones iniciales de la instancia no tienen suficiente tiempo ocioso para acomodar todos los jobs.

Como se aprecia en la figura 6.4, hubo al menos una tarea servida por ejecución. Es importante aclarar que las tareas servidas de cada ejecución no fueron siempre necesariamente

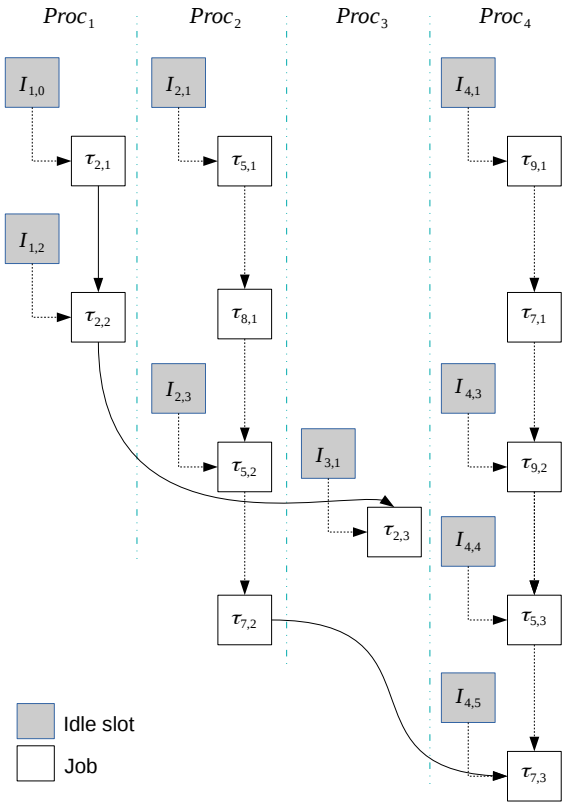


Figura 6.2: Grafo asociado (imagen tomada de [2])

las mismas.

De los resultados experimentales obtenidos de los 605 experimentos, es posible deducir que hay una prevalencia de ejecución de 0,75.

En promedio, los resultados de los 456 experimentos de éxito muestran un valor de explotación promedio de 103.36 unidades de tiempo de las 312.34 unidades de ocio. Esto da un valor promedio para la función objetivo de $\bar{J} = 208{,}98$. Estas 103.36 unidades de tiempo representan una tasa de explotación de 0.33 (una tercera parte del tiempo ocioso total promedio en la plataforma fue utilizada), este resultado pudo verse influenciado por las relaciones de precedencia entre jobs.

Para esta instancia en particular, los jobs en τ representan una carga esperada de trabajo adicional de 349.25 unidades de tiempo, mientras que el tiempo ocioso global promedio es de 312.34 unidades de tiempo. Esta carga de trabajo adicional representa una tasa de 1.12, la cual indica que la carga de trabajo adicional es mayor al tiempo disponible.

Notar que la instancia del ejemplo está en condiciones de desventaja ya que no parece que habrá suficientes intervalos ociosos durante el tiempo de ejecución. Además, hay relaciones de precedencia entre los jobs y es posible que los tamaños de los tiempo de ejecución de los jobs sean de mayor tamaño que los intervalos ociosos. Como consecuencia, no es posible ejecutar el conjunto completo de tareas. También debe notarse que *la presencia de intervalos ociosos no es siempre la misma en cada experimento*, a pesar de que la instancia es la misma. Debido a que tanto el conjunto de tareas como los intervalos

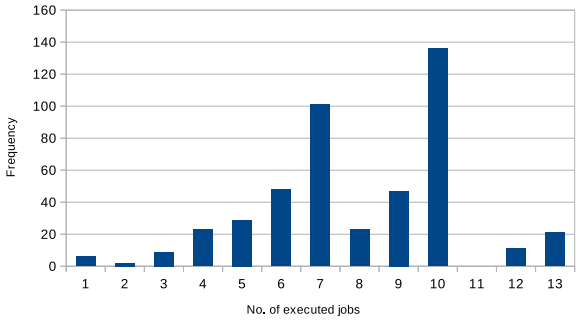


Figura 6.3: Frecuencia *vs.* número total de jobs servidos por ejecución (imagen tomada de [2])

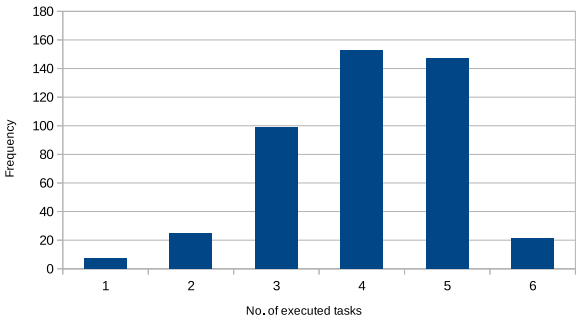


Figura 6.4: Frecuencia *vs.* número total de tareas completas (job-shops) servidos por ejecución (imagen tomada de [2])

ociosos se comportan de manera estocástica, es posible obtener una gran diversidad de salidas para la misma instancia.

6.2.2. Resultados generales

Para conseguir un mejor entendimiento del fenómeno estudiado se realizó experimentación con otras 9 instancias totalmente diferentes. Cada instancia consta de una plataforma homogénea de tamaño $m = 4$ y de un conjunto de tareas de tamaño $n = 10$. Cada job-shop, τ_i, esta conformado por un conjunto de jobs; el número de estos n_i en cada tarea fue asignado de manera pseudoaleatoria: el valor de n_i se eligió del conjunto $\{x|x = 1, \ldots, \lfloor \frac{n}{2} \rfloor\}$.

Las variables aleatorias en las tuplas que representan a cada job $(C_{i,j}, D_i)$ ($C_{i,j}$ y D_i son el consumo y deadline del job $\tau_{i,j}$, respectivamente) fueron generadas de manera aleatoria: los valores de la variable $C_{i,j}$ podían oscilar entre los números enteros positivos 1 a 26 y el tamaño del rango se fijo en $|R_{C_{i,j}}| = 4$ para toda $i = 1, \ldots, 10$; en cuanto a la variable D_i, sus valores oscilaban entre 82 y 1118 e igualmente $|R_{D_i}| = 4$.

Por simplicidad, se supuso que las diez tareas fueron liberadas en el mismo instante de tiempo ($t_{ini} = 0$), la ejecución de las instancias (experimentos) se llevo a cabo dentro del intervalo de tiempo $[0, 1200)$. Para cada experimentó, se simuló un acomodo de tareas mediante el algoritmo *RUN* sobre una plataforma conformada por 4 procesadores idénticos. Luego se intentó planificar un conjunto con $n = 10$ nuevas tareas de tipo job-shop en cada experimento. Por cada instancia se llevaron a cabo 200 ejecuciones dando un total de 1800 experimentos.

Los resultados obtenidos se muestran en la tabla 6.2 junto con sus gráficas correspondientes 6.4 y 6.6 que muestran la frecuencia con que se repitió la ejecución de un mismo número de job-shops completos, mas no de jobs ya que las tareas no necesariamente tienen el mismo número de jobs. En la figura 6.4 se muestra el detalle del conteo de tareas en cada instancia por grupo. La tabla 6.2 muestra las nueve instancias (columnas cuya etiqueta va del 1 al 9) cada columna muestra el conteo de la incidencia de los grupos de tareas ejecutadas completamente.

La persistencia promedio de ejecución exitosa (cuando por lo menos una tarea es ejecutada en su totalidad) obtenida fue de 0,66, es decir, existe una probabilidad de 0,66 de que una tarea se ejecute en su totalidad mediante el uso de esta estrategia basada en *redes bayesianas*. El resumen de todos los conteos por número de tareas exitosamente llevadas a cabo (que se ejecutaron antes de su deadline) se muestra en la figura 6.6.

Para el cálculo del valor promedio de la función objetivo en cada instancia se consideran los resultados mostrados en la tablas 6.3 y 6.4, que contienen los tiempos ociosos promedio de cada instancia, así como los tiempos promedio de aprovechamiento (unidades de tiempo utilizadas por el nuevo conjunto de tareas, τ). En la tabla 6.4 se muestran también los valores máximos y mínimos de unidades de tiempo aprovechado de cada instancia.

Los valores promedio de la función objetivo de cada una de las instancias se muestran en la tabla 6.5.

No. de tareas	Instancia									Promedios
	1	2	3	4	5	6	7	8	9	
0	4	0	155	69	144	137	92	11	0	68
1	85	1	0	0	1	0	0	0	0	9.67
2	87	10	1	1	3	5	0	2	1	12.22
3	22	37	8	10	7	9	2	0	19	12.67
4	2	53	11	19	9	13	7	13	52	19.89
5	0	65	11	42	5	13	22	11	86	28.33
6	0	34	10	59	15	16	34	25	38	25.67
7	0	0	3	0	10	7	25	24	4	8.11
8	0	0	1	0	5	0	14	113	0	14.78
9	0	0	0	0	1	0	4	1	0	0.67
10	0	0	0	0	0	0	0	0	0	0
Total	196	200	45	131	56	63	108	189	200	132

Tabla 6.2: Frecuencias y promedio de las frecuencias (última columna) de ejecución completa de grupos de tareas

Instancia	Tiempo promedio ocioso
1	517.72
2	1196.42
3	1346.48
4	941.42
5	1356.52
6	1562.29
7	1194.889
8	1329.06
9	746.92

Tabla 6.3: Tiempo ocioso promedio de cada instancia

Instancia	Promedio	Mínimo	Máximo
1	36.32	6	117
2	159.52	22	280
3	220.28	89	431
4	152.62	60	244
5	224.34	36	363
6	231.16	41	406
7	160.36	63	324
8	218.44	33	298
9	107.37	26	222

Tabla 6.4: Tiempo aprovechado promedio, tiempo máximo aprovechado y tiempo mínimo aprovechado obtenido en cada instancia

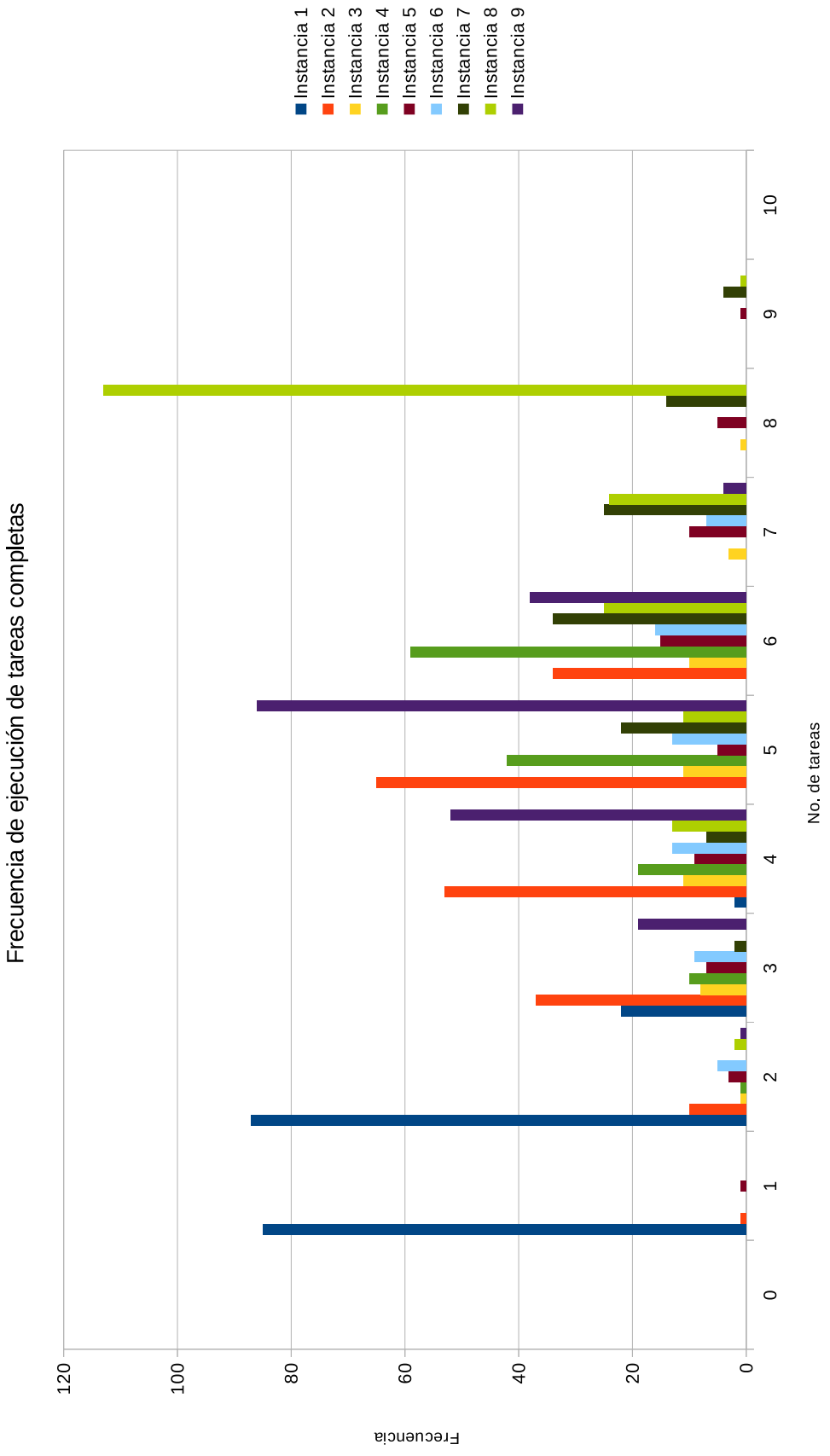


Figura 6.5: Frecuencia de ejecución de tareas completas

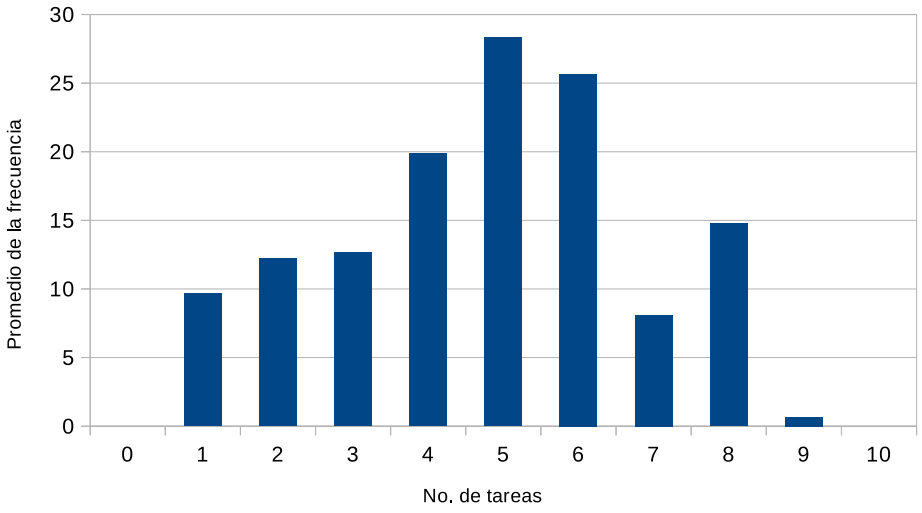


Figura 6.6: Promedio de las frecuencias del número de tareas ejecutadas

Instancia	Función objetivo	Tasa de aprovechamiento
1	481.40	0.07
2	1036.9	0.13
3	1126.20	0.16
4	788.80	0.16
5	1132.18	0.17
6	1331.13	0.15
7	1034.52	0.13
8	1110.63	0.16
9	639.55	0.14

Tabla 6.5: Valor promedio de la función objetivo y tasa de aprovechamiento promedio (tiempo aprovechado promedio/tiempo ocioso promedio)

6.3. Conclusión

Este trabajo propone una estrategia que utiliza una heurística basada en técnicas estadísticas, la cual elige cada vez al mejor job por medio del uso de redes bayesianas, siempre que el algoritmo factible usado (en este caso *RUN-modificado*) detone una interrupción desde el procesador en el que aparece un intervalo de tiempo ocioso con el propósito de utilizarlo y con ello minimizar el tiempo ocioso global. Esta heurística siempre elige al mejor job basándose en el historial de ejecuciones previas. Esta puede ser una estrategia prometedora ya que se emplea información previa para la elección del mejor job, a diferencia de las heurísticas que utilizan la información disponible al momento de tomar la decisión. Por lo tanto, con la estrategia propuesta se pueden obtener secuencias para cada procesador que aprovechan mejor el tiempo ocioso local (en un procesador) y al mismo tiempo mejorar el aprovechamiento global (en la plataforma) de tiempo de cómputo.

Tanto en [42] como en el trabajo propuesto en esta disertación (capítulo 5) tienen en común la aplicación de redes bayesianas sobre dos problemas distintos de planificación. Lo que deja ver que estas redes pueden llegar a ser un instrumento de modelación con grandes oportunidades frente a otras estrategias de *machine learning*.

La estrategia propuesta en este trabajo tiene el propósito de minimizar el tiempo global ocioso de la plataforma en un intervalo, a saber el definido por $[t_{ini}, t_{end})$. La estrategia de planificación se basa en el uso de una heurística greedy que emplea redes Bayesianas para elegir el mejor job localmente, es decir que la selección de un job se hará con los jobs pertenecientes al mismo procesador, siempre que *RUN-modificado* lance la interrupción en el procesador en el que aparece el intervalo ocioso.

En principio, conocer la información de ejecuciones previas debería permitir un mejor uso de los recursos de procesamiento en la plataforma y, como consecuencia inmediata, minimizar el tiempo total global en el sistema.

Ventajas de utilizar esta metodología.

- Al tratarse de una estrategia que no se basa en el peor tiempo de ejecución, la mejora basada en el uso de redes bayesianas da flexibilidad para considerar tareas que de otra forma serían rechazadas por análisis de planificabilidad tradicionales (deterministas).
- La estrategia propuesta tiene una complejidad pseudopolinomial con respecto al número de padres que una variable aleatoria tiene, sin embargo al limitar el número de posibles relaciones a tres (ver sección 5.2.2) y limitando el rango de las variables aleatorias a valores pequeños es posible realizar la propagación de evidencia en tiempo lineal para computar la distribución de probabilidad conjunta de la red formada hasta un cierto instante acotado por $[t_{ini}, t_{fin})$.

Algunas desventajas de la metodología basada en redes bayesianas se listan a continuación.

- Es una estrategia costosa en cuanto a memoria se refiere, ya que es necesario almacenar un historial de ejecuciones previas para cada posible instancia.
- Derivado del punto anterior, la estrategia que se apoya en la mejora basada en redes

bayesianas no puede ser implementada en sistemas embebidos.

Capítulo 7

Conclusiones

7.1. Sobre la estrategia basada en redes bayesianas

En esta tesis se han presentado dos contribuciones cada una desde un punto de vista distinto: *determinista*, utilizando el enfoque del peor tiempo de ejecución (WCET) y *no determinista*, empleando una heurística basada en estadísticas.

La primer estrategia mostrada utiliza una heurística basada en estadísticas, la cual elige al mejor job por medio del uso de redes bayesianas. Esta heurística siempre elige al mejor job basándose en un historial de ejecuciones previas, lo que significa que el modelo utiliza información ya existente para "aprender" y poder tomar decisiones. Esta estrategia puede aplicarse para mejorar el aprovechamiento del tiempo de cómputo. Desde el punto de vista del análisis de algoritmos basados en el *WCET*, las tareas son asignadas, en una primera etapa, de manera offline a una plataforma multiprocesador y luego, durante una segunda etapa, se lleva a cabo la ejecución de las mismas.

Sin embargo, se ha observado que asumir estos WCET puede representar una perdida de tiempo de en términos de la capacidad de procesamiento[54], ya que los costos de los WCET pueden estar varios ordenes de magnitud por encima de los valores promedio de ejecución de las tareas.

Una ventaja de la estrategia propuesta es su versatilidad para trabajar en conjunto con cualquier estrategia offline factible para mejorar el uso del tiempo de cómputo. Siempre que el algoritmo factible usado (como el caso de estudio descrito en el capítulo 6: *RUN-modificado*) detone una interrupción, cuando aparece un intervalo de tiempo ocioso, para que el algoritmo de inferencia basado en redes bayesianas se lleve a cabo y determine la mejor forma de utilizar dicho intervalo y lograr minimizar el tiempo ocioso de la plataforma.

Esta heurística siempre elige al mejor job basándose en el historial de ejecuciones previas. Ésta puede ser una estrategia prometedora ya que se emplea información previa para la elección del mejor job. Por lo tanto, con la estrategia propuesta se pueden obtener secuencias para cada procesador que aprovechan mejor el tiempo ocioso local (en un

procesador) y, al mismo tiempo, mejorar el aprovechamiento global (en la plataforma) de tiempo de cómputo.

Además del trabajo propuesto en esta disertación, capítulo 5 (ver [2]), ha habido otros esfuerzos por aplicar redes bayesianas a diversos problemas de planificación como [35] y [42] todos modelan problemas distintos. Lo que deja ver que estas redes pueden llegar a ser un instrumento de modelación con grandes oportunidades frente a otras estrategias de *machine learning*.

Es importante notar que la estrategia propuesta en este trabajo tiene el propósito de minimizar el tiempo global ocioso de la plataforma en un intervalo, a saber, el definido por $[t_{ini}, t_{end})$ (ver Figura 7.1 *(a)*). Sin embargo, es posible extender esta estrategia para que en lugar de realizar inferencia sobre un intervalo se realice sobre una ventana ($window_i$ en la figura 7.1 *(b)*) que se defina no a partir de un intervalo de tiempo fijo sino, más bien a partir de un determinado número de tareas previamente planificadas. El uso de esta ventana puede significar una reducción del tiempo en la toma de decisión del mejor job, ya que no se consideran todas las variables aleatorias desde el instante t_{ini} hasta el instante actual t, sino de un subconjunto de éstas; empero, al mismo tiempo esto afectaría a la precisión del cálculo de las probabilidades condicionales de cada nodo en la red, por lo que se requiere investigación al respecto.

En principio, conocer la información de ejecuciones previas debería permitir un mejor uso de los recursos de procesamiento en la plataforma y, como consecuencia inmediata, minimizar el tiempo ocioso total global en el sistema.

Algunas ventajas de utilizar esta metodología son listadas a continuación:

- Al tratarse de una estrategia que no se basa en el peor tiempo de ejecución, la mejora basada en el uso de redes bayesianas da flexibilidad para considerar tareas que de otra forma serían rechazadas por análisis de suficiencia de planificabilidad tradicionales deterministas.

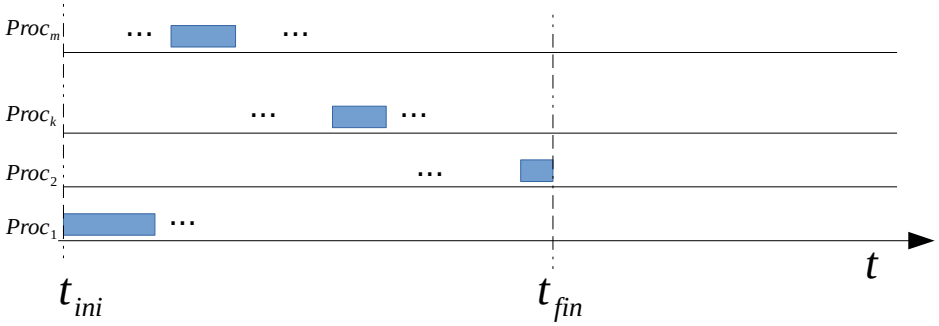
- La estrategia propuesta tiene una complejidad pseudopolinomial con respecto al número de padres que una variable aleatoria (job o intervalo ocioso) tiene, sin embargo al limitar el número de posibles relaciones a tres (ver sección 5.2.2) y limitando el rango de las variables aleatorias a valores pequeños es posible realizar la propagación de evidencia en tiempo casi lineal para computar la distribución de probabilidad conjunta de la red formada hasta un cierto instante acotado por $[t_{ini}, t_{fin})$.

- Como se mencionó antes, esta estrategia se puede extender a más allá de un intervalo para considerar nuevas tareas que pueden estar llegando mediante el uso de ventanas.

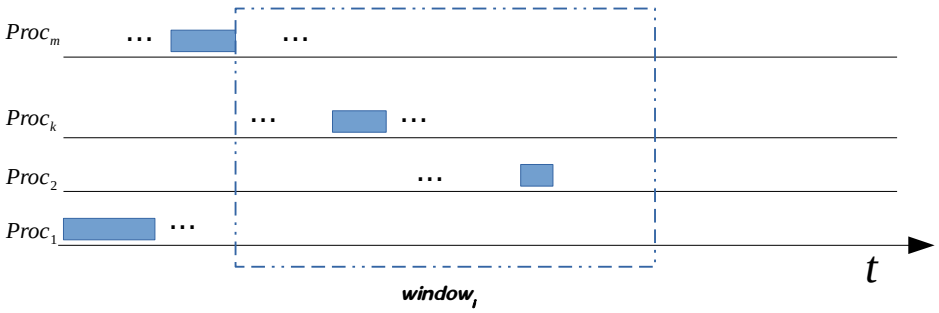
Algunas desventajas de la metodología basada en redes bayesianas se listan a continuación.

- Es una estrategia costosa en cuanto a memoria se refiere, ya que es necesario almacenar un historial de ejecuciones previas para cada posible instancia.

- Derivado del punto anterior, esta estrategia basada en redes bayesianas difícilmente puede ser implementada en sistemas embebidos.



(a) Estado del trabajo actual: planificación en el intervalo $[t_{ini}, t_{fin})$



(b) Uso de ventana, $window_i$, para extender el trabajo actual.

Figura 7.1: Posible extensión de la estrategia basada en redes bayesianas para reducir el tiempo e implementarlas a lo largo del ciclo de vida del sistema

7.2. Sobre la estrategia basada en el análisis del WCET

Otro de los trabajos pertenecientes a esta disertación consistió en desarrollar una estrategia de planificación que permite incluir un tipo de tareas de importancia especialmente para la industria, *job-shops*. Para ello se realizó una extensión al análisis de planificabilidad desarrollado en 2006 por Sanjoy Baaruah [16]. Dicha estrategia se basa en el WCET, por tanto, resulta ser una estrategia bastante restrictiva en el aspecto de que no se podrán ejecutar ciertas tareas aún cuando si sean planificables; sin embargo, el análisis de planificabilidad podrá ser utilizado para tareas críticas, es decir, aquellas tareas cuyos deadlines no sean cumplidos representan algún riesgo.

Un aspecto importante es que el análisis aquí propuesto se realiza para plataformas heterogéneas bajo un esquema global, lo que quiere decir que ocurre la migración; no obstante se considera que el tiempo que implica la migración y el cambio de contexto de un job es nulo. Actualmente, la oferta de algoritmos y análisis de planificabilidad para plataformas heterogéneas es escasa, uno de estos trabajos es el desarrollado por Shelby Funk en 2005 [55], el cual es aplicado a tareas periódicas y trabaja sobre un esquema particionado (ver sección 2.2.2).

La extensión de [16], aquí propuesta, permite determinar de forma *online* si un nuevo *job-shop* que se libera en el sistema es capaz de cumplir con su deadline. Ya que este análisis de planificabilidad tarda tiempo lineal, de acuerdo al número de tareas, el resultado presente tiene el propósito de ser implementado en sistemas con recursos limitados debido al bajo costo computacional que este análisis representa.

7.3. Trabajo a futuro

El efecto (retardo) que tiene la migración sobre la ejecución de una tarea es sin duda uno de los puntos neurálgicos en el estudio de estrategias de planificación multiprocesador, así como el efecto de los accesos a memoria.

Suponga que se tiene una tarea preemptive, τ_i, sea C_i su tiempo de cómputo requerido; se desea acotar el tiempo máximo de completado de τ_i, además suponga que esta tarea se ha dividido en dos partes: $\tau_{i,1}$ y $\tau_{i,2}$ en las siguientes proporciones α_1 y α_2 para toda $\alpha_i > 0$, donde $i = 1, 2$, tales que $\alpha_1 + \alpha_2 = 1$. Para dar una idea de como modelar el tiempo requerido por una sola tarea hasta su término, se debe saber a que procesadores han sido asignadas $\tau_{i,1}$ y $\tau_{i,2}$ de tal forma que se conozca exactamente el tiempo de término para τ_i; para ello suponga que las subtareas $\tau_{i,1}$ y $\tau_{i,2}$ han sido asignadas a los procesadores $Proc_l$ y $Proc_{l+1}$, respectivamente.

Sea s_l la tasa de rapidez de ejecución del procesador donde se ejecuta $\tau_{i,1}$ y s_{l+1} la tasa de rapidez de un segundo procesador en el que se ejecuta $\tau_{i,2}$. Por lo tanto, el tiempo requerido para llevar a cabo τ_i es

$$\sum_{l=1}^{2} \alpha_i \cdot C_{i,l} \cdot s_l \tag{7.1}$$

para cuales quiera $\alpha_1, \alpha_2 > 0$.

Si se considera el retardo que implica el tiempo de migración (debido al cambio de contexto) de la sección $\tau_{i,2}$ al segundo procesador, sea δ dicho tiempo que implica el cambio de contexto, se tendrá una mejor aproximación del tiempo de término, entonces la ecuación (7.1) cambia a

$$\sum_{l=1}^{2} \alpha_i \cdot C_{i,l} \cdot s_l + \delta \tag{7.2}$$

La ecuación (7.2) muestra un escenario claro si δ es un valor constante; sin embargo, en sistemas distribuidos se lidia con dos problemas importantes: la sincronización y la incertidumbre introducida por las comunicaciones. Ambas variables terminan generando un retardo en la ejecución de τ_i. Para modelar dicho retardo se podría utilizar un enfoque estocástico como el descrito en [54], o bien recurrir al análisis mediante teoría de colas [56] para proveer de cotas que ayuden a inferir el tiempo posible de termino para τ_i.

Apéndice A

Propagación en redes probabilísticas

Cualquier modelo probabilístico se tiene que representar (de manera explicita o implícita) por medio de una distribución de probabilidad conjunta, en otras palabras, el modelo tiene que ser capaz de describir cada evento posible definido por la combinación de los valores de todas las variables aleatorias.

Una vez definida una base de conocimiento coherente, una de las tareas más importantes de un sistema experto consiste en obtener conclusiones a medida que se va conociendo nueva información, o *evidencia*. El mecanismo para obtener conclusiones a partir de la evidencia se conoce como propagación de evidencia. Esta tarea consiste en actualizar las probabilidades de las variables en función de la evidencia. Existen tres tipos distintos de algoritmos de propagación [57]: exactos, aproximados y simbólicos. Un algoritmo de propagación se denomina *exacto* si calcula las probabilidades de los nodos sin otro error que el resultante del redondeo producido por las limitaciones de cálculo del ordenador. Los algoritmos de propagación *aproximada* utilizan distintas técnicas de simulación para obtener valores aproximados de las probabilidades. Estos algoritmos se utilizan en aquellos casos en los que los algoritmos exactos no son aplicables, o son computacionalmente costosos. Finalmente, un algoritmo de propagación *simbólica* puede operar no sólo con parámetros numéricos, sino también con parámetros simbólicos, obteniendo las probabilidades en función de los parámetros.

A.1. Probabilidad condicional

El cálculo de las probabilidades condicionales de cada uno de los nodos que conforman a la red bayesiana se obtiene de la información de instancias previamente ejecutadas que se almacenan en una base de conocimiento (base de datos). De esta manera se obtienen las frecuencias absolutas y relativas. Estas frecuencias son usadas para calcular las tablas de probabilidad condicional para cada nodo mediante la conocida fórmula (A.1) de probabilidad condicional.

$$p(A|B) = \frac{p(A,B)}{p(B)} \tag{A.1}$$

En este caso el evento B es visto como una partición de x eventos independientes, esto es $B = B_1 \cup B_2 \cup \ldots \cup B_x$. Así A.1 cambia a

$$p(A = a|B_1 = b_1, \ldots, B_x = b_x) = \frac{p(A = a, B_1 = b_1, \ldots, B_x = b_x)}{p(B_1 = b_1, \ldots, B_x = b_x)} \tag{A.2}$$

Desde el punto de vista de la estadística frecuentista la ecuación (A.2) puede reescribirse de la siguiente forma

$$p(A = a|B_1 = b_1, \ldots, B_x = b_x) = \frac{card(a,\ b_1,\ \ldots,\ b_x)}{card(b_1,\ \ldots,\ b_x)} \tag{A.3}$$

donde $card(a,\ b_1,\ b_2,\ \ldots,\ b_x)$ es el número de veces que los valores indicados $a,\ b_1,\ b_2,\ \ldots,\ b_x$ para las variables aleatorias $(A, B_1, B_2, \ldots, B_x)$ aparecen en el conjunto de datos.

El teorema de Bayes también es empleado para invertir la dirección del condicionamiento. Supóngase que se pide calcular $p(A|B)$, pero sólo es conocida $p(B|A)$ entonces es posible escribir A.1 como

$$p(A|B) = \frac{p(A|B)p(A)}{p(B)} \tag{A.4}$$

A.1.1. Propagación sobre redes probabilísticas

La propagación es un mecanismo que consiste en obtener conclusiones como nueva información, conocida como evidencia, según se vaya conociendo ésta. Este mecanismo actualiza las probabilidades de las variables aleatorias en función de la evidencia. Supóngase un conjunto de variables aleatorias discretas $X = \{X_1, \ldots, X_n\}$ y una función de probabilidad $p(x)$ sobre la v.a. X. Cuando no se dispone de evidencia, el proceso de propagación consiste en calcular las probabilidades marginales $p(X_i = x_i)$, denotadas como $p(x_i)$, para cada $X_i \in X$. Estas probabilidades dan información a *priori* sobre los valores que una variable aleatoria puede tomar.

Cuando se dispone de cierta evidencia, es decir que se conocen los valores de un subconjunto de variables aleatorias, E, tal que $E \subset X$, el cual tiene valores asociados $X_i = e_i$ para cada $X_i \in E$, el proceso de propagación debe tomar en cuenta dichos valores para calcular las nuevas probabilidades para las variables aleatorias.

Definición 11 (Evidencia) ***Evidencia*** *es un subconjunto de variables aleatorias $E \subset X$ cuyos valores son conocidos.*

Bajo esta situación, el cálculo de la propagación de evidencia se obtiene por medio de computar las funciones de probabilidad condicional $p(x_i|e)$ para cada variable $X_i \notin E$, dada la evidencia $E = e$.

Estas funciones de probabilidad condicional miden el efecto producido por la evidencia en cada variable aleatoria. Cuando no hay evidencia, es decir $E = \varnothing$, las funciones de probabilidad condicional son sólo las funciones de probabilidad marginal

$$p(x_i|e) = p(x_i)$$

Una forma de calcular cada $p(x_i|e)$ consiste en utilizar (A.1):

$$p(x_i|e) = \frac{p(x_i, e)}{p(e)} \propto p(x_i, e)$$

$\frac{1}{p(e)}$ es una constante de proporcionalidad. Por lo tanto, se puede obtener $p(x_i|e)$ calculando y normalizando las probabilidades marginales $p(x_i, e)$:

$$p(x_i, e) = \sum_{x \setminus \{x_i, e\}} p_e(x_1, \ \ldots, \ x_n) \tag{A.5}$$

donde $p_e(x_1, \ \ldots, \ x_n)$ es la función de probabilidad obtenida sustituyendo en $p(x_1, \ \ldots, \ x_n)$ las variables con evidencia, E, por sus valores e. Por lo tanto, para calcular $p(x_i, e)$, debe sumarse $p_e(x_1, \ \ldots, \ x_n)$ para todas las posibles combinaciones de valores de las variables que no estén contenidas en E, excepto la variable X_i. Cuando no se dispone de evidencia, la ecuación () queda reducida a

$$p(x_i) = \sum_{x \setminus \{x_i\}} p(x_1, \ \ldots, \ x_n) \tag{A.6}$$

Debido al elevado número de combinaciones que involucra (A.6), este método de *fuerza bruta* resulta altamente ineficiente, incluso en redes con un número reducido de variables. Por ejemplo, en el caso de tener n variables aleatorias binarias, la ecuación (A.6) requiere de la suma de 2^{n-1} valores de probabilidad.

A.2. Redes Bayesianas

Una red bayesiana (BN) es un método gráfico exacto que se representa mediante la tupla (D, P), donde D es una gráfica acíclica dirigida, y $P = \{p(x_1|\pi_1), \ \ldots, p(x_n|\pi_n)\}$ es un conjunto de n funciones de probabilidad condicional, una por cada variable aleatoria, y Π_i es el conjunto de predecesores de X_i donde $X_i \in D$. El conjunto P define una función de probabilidad conjunta asociada como se indica a continuación:

$$p(x) := \prod_{i=1}^{n} p(x_i|\Pi_i) \tag{A.7}$$

Como ejemplo, considere el conjunto de variables aleatorias $\{X_1, X_2, \ldots, X_4\}$ de la gráfica mostrada en la figura A.1, su función de probabilidad correspondiente es calculada a partir de la definición A.7 como sigue

$$p(x_1, x_2, \ldots, x_4) = p(x_4|x_3)p(x_3|x_1, x_2)p(x_1)p(x_2)$$

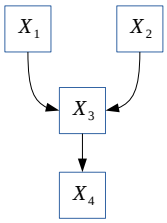


Figura A.1: Graph with four random variables

A.2.1. Algoritmo de Pearl: propagación sobre poliárboles

Un *poliárbol* (el termino en inglés es *polytree*) es uno de los modelos gráficos más simples de abstraer un problema a redes bayesianas.

A continuación, se presenta un algoritmo de propagación para este tipo de estructuras, este algoritmo fue presentado en 1983 por Kim y Pearl [53], [50]. En un poliárbol, cualesquiera dos nodos (variables aleatorias) están conectados por una única ruta, ésto implica que un nodo divide a un poliárbol en otros dos poliárboles: uno de ellos conteniendo todos los predecesores y el otro conteniendo a todos sus sucesores.

El proceso de propagación puede realizarse en este tipo de grafos de un modo eficiente combinando la información procedente de los distintos subgrafos mediante el envío de mensajes (cálculos locales) de un subgrafo a otro.

Supóngase que se conoce cierta evidencia $E = e$ y que se requiere calcular las probabilidades $p(x_i|e)$ para todos los valores x_i de un nodo X_i tal que $X_i \notin E$. Para el cálculo de estas probabilidades E se descompone en:

1. E_i^+: subconjunto de E que es accesible desde X_i a través de sus padres.
2. E_i^-: subconjunto de E que es accesible desde X_i a través de sus hijos.

Por lo tanto $E = E_i^+ \cup E_i^-$. Usando A.4 A.2 se obtiene

$$p(x_i|e) = p(x_i|e_i^-, e_i^+) = \frac{p(e_i^-, e_i^+|x_i)p(x_i)}{p(e_i^-, e_i^+)}$$

Como fue mencionado, X_i separa a E_i^+ de E_i^- en el poliárbol, es decir, E_i^+ es independiente

de E_i^- dado X_i, denotado como $E_i^+ \perp_{X_i} E_i^-$, por lo tanto se tiene

$$\begin{aligned} p(x_i|e) &= \frac{p(e_i^+|x_i)p(e_i^-|x_i)p(x_i)}{p(e_i^-, e_i^+)} \\ &= \frac{p(e_i^-|x_i)p(e_i^+, x_i)}{p(e_i^-, e_i^+)} \\ &= \kappa p(e_i^-|x_i)p(e_i^+, x_i) \\ &= \kappa \lambda_i(x_i)\rho_i(x_i) \end{aligned} \tag{A.8}$$

donde κ es una constante de normalización $\kappa := \frac{1}{p(e_i^-, e_i^+)}$, $\lambda_i(x_i)$ es un resumen de la evidencia procedente de los hijos de X_i:

$$\lambda_i(x_i) = p(e_i^-|x_i) \tag{A.9}$$

y $\rho_i(x_i)$ es un resumen de la evidencia procedente de los padres de X_i:

$$\rho_i(x_i) = p(e_i^+, x_i) \tag{A.10}$$

Por lo tanto, la función de probabilidad sin normalizar esta dada por

$$\beta_i(x_i) = \lambda_i(x_i)\rho_i(x_i) \tag{A.11}$$

entonces

$$p(x_i|e) = \kappa\beta_i(x_i) \propto \beta_i(x_i) \tag{A.12}$$

Para calcular las funciones $\lambda_i(x_i)$, $\rho_i(x_i)$, y $\beta_i(x_i)$ se considera la situación siguiente, en la que un nodo arbitrario X_i tiene p padres y c hijos. Los padres se representan mediante $U = \{U_1, \ldots, U_p\}$ y los hijos con $Y = \{Y_1, \ldots, Y_c\}$.

Teniendo en cuenta la estructura de poliárbol, el conjunto de evidencia E_i^+ es descompuesto en p subconjuntos, uno para cada padre de X_i:

$$E_i^+ = \{E_{U_1X_i}^+, \ \ldots, \ E_{U_pX_i}^+\} \tag{A.13}$$

donde $E_{U_jX_i}^+$ $(j = 1, \ \ldots, \ p)$ es el subconjunto de E_i^+ contenido en la subgráfica asociada al nodo U_j cuando se elimina la arista $U_j \to X_i$. De forma similar, la evidencia E_i^- es dividida en c subconjuntos disjuntos, uno por cada hijo de X_i:

$$E_i^- = \{E_{X_iY_1}^-, \ \ldots, \ E_{X_iY_c}^-\} \tag{A.14}$$

donde $E^-_{X_iY_j}$ $(j = 1, \ldots, c)$ es el subconjunto de E_i^- contenido en la subgráfica asociada al nodo Y_j cuando se elimina la arista $X_i \to Y_j$. Sea $u = \{u_1, \ldots, u_p\}$ una realización de los padres del nodo X_i. Entonces, la función $\rho_i(x_i)$ se calcula de la siguiente forma:

$$\begin{aligned}
\rho_i(x_i) = p(x_i, e_i^+) &= \sum_u p(x_i, u \cup e_i^+) \\
&= \sum_u p(x_i|u \cup e_i^+) p(u \cup e_i^+) \\
&= \sum_u p(x_i|u \cup e_i^+) \prod_{j=1}^{p} p(u_j \cup e^+_{U_jX_i}) \\
&= \sum_u p(x_i|u \cup e_i^+) \prod_{j=1}^{p} \rho_{U_jX_i}(u_j)
\end{aligned}$$

Dado que $\{U_j, E^+_{U_jX_i}\}$ es incondicionalmente independiente de $\{U_k, E^+_{U_kX_i}\}$ para $j \neq k$.

$$\begin{aligned}
\rho_i(x_i) &= \sum_u p(x_i|u \cup e_i^+) \prod_{j=1}^{p} p(u_j \cup e^+_{U_jX_i}) \\
&= \sum_u p(x_i|u \cup e_i^+) \prod_{j=1}^{p} \rho_{U_jX_i}(u_j)
\end{aligned} \tag{A.15}$$

donde $\rho_{U_jX_i}(u_j)$ es el mensaje ρ que el nodo U_j envía a su hijo X_i. Este mensaje sólo depende de la información contenida en la subgráfica asociada al nodo U_j cuando la arista $U_j \to X_i$ es eliminada. Note que si U_j es una variable con evidencia, $u_j = e_j$, entonces

$$\rho_{U_jX_i}(u_j) = \begin{cases} 1 & \text{si } u_j = e_j \\ 0 & \text{si } u_j \neq e_j \end{cases} \tag{A.16}$$

La función $\lambda_i(x_i)$ es calculada de forma similar: $\lambda_i(x_i) = p(e_i^-|x_i) = p(e^-_{X_iY_1}, \ldots, e^-_{X_iY_c}|x_i)$. Dado que X_i *D-separa* $E^-_{X_iY_j}$ de $E^-_{X_iY_k}$, para $k \neq j$, entonces se tiene

$$\lambda_i(x_i) = \prod_{j=1}^{c} \lambda_{Y_jX_i}(x_i) \tag{A.17}$$

donde

$$\lambda_{Y_jX_i}(x_i) = p(e^-_{X_iY_j}|x_i) \tag{A.18}$$

λ es el mensaje que el nodo Y_j envía a su padre X_i.

A partir de (A.15) puede verse que un nodo X_i puede calcular su función ρ, $\rho_i(x_i)$, una vez que haya recibido los mensajes ρ de todos sus padres. De forma similar, a partir de (A.17) puede observarse que la función $\lambda_i(x_i)$ puede ser calculada una vez que X_i haya recibido los mensajes λ de todos sus hijos.

Sustituyendo (A.15) y (A.17) en (A.12) se obtiene

$$p(x_i|e) \propto \beta_i(x_i) = \left(\sum_u p(x_i|u \cup e_i^+) \prod_{j=1}^{p} \rho_{U_j X_i}(u_j) \right) \left(\prod_{j=1}^{c} \lambda_{Y_j X_i}(x_i) \right) \tag{A.19}$$

A continuación se calculan los distintos mensajes que aparecen en la fórmula anterior. Considere un nodo arbitrario X_i y uno de sus hijos, Y_j, y utilizando la igualdad

$$E^+_{X_i Y_j} = E_i^+ \bigcup_{k \neq j} E^-_{X_i Y_k}$$

$$\begin{aligned} \rho_{X_i Y_j}(x_i) &= p(x_i \cup e^+_{X_i Y_k}) \\ &= p\left(x_i \cup e_i^+ \cdot \cup_{k \neq j} e^-_{X_i Y_k}\right) \\ &\propto \rho_i(x_i) \prod_{k \neq j} \lambda_{Y_k X_i(x_i)} \end{aligned} \tag{A.20}$$

En el caso del mensaje λ en la formula A.9, $\lambda_{X_i Y_j}(x_i)$, es considerado el conjunto de todos los padres de Y_j los cuales son diferentes de X_i, $V = \{V_1, \ldots, V_q\}$. Por lo tanto, el nodo Y_j tiene $q+1$ padres, esto significa que

$$e^-_{X_i Y_j} = e^-_{Y_j} \cup e^+_{V Y_j}$$

donde $e^+_{V Y_j}$ representa la evidencia obtenida de los padres de Y_j, excepto de X_i.

$$\begin{aligned} \lambda_{Y_j X_i}(x_i) &= \sum_{y_j} p(e^-_{Y_j}|y_j) \sum_v p(y_j|v, x_i) p(v, e^+_{V Y_j}) \\ &= \sum_{y_j} \lambda_{Y_j}(y_j) \sum_{v_1, \ldots, v_q} p(y_j|\pi_{Y_i}) \prod_{k=1}^{q} \rho_{V_k Y_j}(v_k) \end{aligned} \tag{A.21}$$

El algoritmo que calcula $p(x_i|e)$ para cada nodo $X_i \notin E$ es descrito a continuación:

Inicialización

1. Para todos los nodos con evidencia, $X_i \in E$, asignar las funciones:
   - $\rho(x_i) = 1$ si $x_i = e_i$, o $\rho(x_i) = 0$ si $x_i \neq e_i$
   - $\lambda(x_i) = 1$ si $x_i = e_i$, o $\lambda(x_i) = 0$ si $x_i \neq e_i$
2. For all the nodes such that $X_i \notin E$ and do not have parents, initialise their function $\rho_i(x_i) = p(x_i)$

3. For all the nodes such that $X_i \notin E$ and do not have children, initialise their function $\lambda_i(x_i) = 1$

Iterative phase

1. For every node $X_i \notin E$:

   a) If X_i has received all the messages ρ from its parents, then calculate $\rho_i(x_i)$ using (A.15).

   b) If X_i has received all the messages from its children, then calculate $\lambda_i(x_i)$ using (A.17).

   c) When $\rho_i(x_i)$ has been already calculated, then message $\rho_{X_iY_j}(x_i)$ must be calculated and sent to every children Y_j of X_i such that X_i has received the messages λ from the rest of its children using A.20. That is to say, the corresponding ρ messages must be sent upon X_i has received the λ messages from all its children.

   d) When $\lambda_i(x_i)$ has been already calculated, and every parent U_j of X_i such that X_i has received the ρ messages from the rest of its parents, then message $\lambda_{X_iU_j}(u_i)$ must be calculated using (A.21). That is to say, X_i can send all the corresponding λ messages upon X_i has received all the ρ messages from all its parents.

2. Repeat the previous step until ρ and λ functions had been computed for every node $X_i \notin E$ (no new message has been computed in an iteration).

3. Compute $\beta_i(x_i)$ using (A.11) for every node $X_i \notin E$. They are the non-normalized corresponding to $p(x_i|e)$.

4. Normalize function $\beta_i(x_i)$ to compute $p(x_i|e)$: $p(x_i|e) = \beta_i(x_i)/k$, where $k = \sum_{x_i} \beta_i(x_i)$.

Observe que las funciones ρ y λ son calculadas en diferentes iteraciones mientras el proceso de propagación está en ejecución. Podría ocurrir que sólo pocos nodos de la red computan sus λ o ρ mensajes en una iteración.

Una ventaja de las BN es que a pesar de que existe un número exponencial de eventos; las redes bayesianas permiten "compactar" la distribución de probabilidad conjunta en cálculos locales (se computan las probabilidades condicionales de cada variable dados sus padres).

Bibliografía

[1] A. Mok and M. Dertouzos, "Multiprocessor online scheduling of hard-real-time tasks," *IEEE Transactions on Software Engineering*, vol. 15, pp. 1497–1506, dec 1989. (Citado en páginas XI, 16, 18, and 19.)

[2] J. A. Hermosillo-Gomez and H. Benitez-Perez, "Run enhancement through bayesian networks," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 0, no. 0, pp. 1–15, 2019. (Citado en páginas XI, XI, XI, XI, 57, 59, 60, 61, and 68.)

[3] J. H. Lee, A. Yilmaz, R. Denning, and T. Aldemir, "Use of dynamic event trees and deep learning for real-time emergency planning in power plant operation," *Nuclear Technology*, vol. 205, no. 8, pp. 1035–1042, 2019. (Citado en página 1.)

[4] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *2011 IEEE 32nd Real-Time Systems Symposium*, pp. 104–115, Nov 2011. (Citado en páginas 2, 4, 12, 23, 29, 55, and 57.)

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, Jan. 1973. (Citado en páginas 2, 9, and 12.)

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600–625, Jun 1996. (Citado en páginas 2, 8, 12, 19, 23, 29, and 57.)

[7] Dakai Zhu, D. Mosse, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pp. 142–151, Dec 2003. (Citado en páginas 2, 23, 29, and 57.)

[8] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, pp. 322–334, Aug 2006. (Citado en páginas 2, 12, 21, 22, 23, 29, 55, and 58.)

[9] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pp. 101–110, Dec 2006. (Citado en páginas 2, 23, 29, and 57.)

[10] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *2010 22nd Euromicro Confe-*

rence on Real-Time Systems, pp. 3–13, July 2010. (Citado en páginas 2, 23, 29, and 57.)

[11] J. A. Stankovic and K. Ramamritham, "What is predictability for real-time systems?," *Real-Time Systems*, vol. 2, pp. 247–254, Nov 1990. (Citado en páginas 2, 5, and 10.)

[12] T. . Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. . Wu, and J. W. . Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proceedings Real-Time Technology and Applications Symposium*, pp. 164–173, May 1995. (Citado en páginas 3 and 45.)

[13] M. K. Gardner and J. W.-S. Liu, "Analyzing stochastic fixed-priority real-time systems," in *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, (Berlin, Heidelberg), pp. 44–58, Springer-Verlag, 1999. (Citado en páginas 3 and 45.)

[14] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft real-time systems: Predictability vs. efficiency*. Springer US, 01 2005. (Citado en páginas 3, 43, and 45.)

[15] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under industry 4.0," *Journal of Intelligent Manufacturing*, vol. 30, pp. 1809–1830, Apr 2019. (Citado en páginas 3 and 30.)

[16] S. K. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Systems*, vol. 32, pp. 9–20, Feb 2006. (Citado en páginas 3, 30, 31, 32, 36, and 70.)

[17] J. Liu, *Real-Time Systems*. Prentice Hall, 2000. (Citado en página 7.)

[18] N. Audsley, A. Burns, and A. Wellings, "Deadline monotonic scheduling theory and application," *Control Engineering Practice*, vol. 1, no. 1, pp. 71 – 78, 1993. (Citado en página 9.)

[19] A. K.-L. Mok, *Fundamental design problems of distributed systems for the hard-real-time environment*. PhD thesis, Massachusetts Institute of Technology, 1983. (Citado en página 9.)

[20] N. Guan, *Techniques for Building Timing-Predictable Embedded Systems*. Springer International Publishing, 2016. (Citado en página 10.)

[21] V. Suhendra and T. Mitra, "Exploring locking partitioning for predictable shared caches on multi-cores," in *2008 45th ACM/IEEE Design Automation Conference*, pp. 300–303, June 2008. (Citado en página 10.)

[22] J. Carpenter, S. Funk, P. Holman, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," in *Handbook of Scheduling - Algorithms, Models, and Performance Analysis* (J. Y. Leung, ed.), ch. 30, pp. 1–19, Chapman and Hall/CRC, 2004. (Citado en páginas 11 and 12.)

[23] D. S. Johnson, *Near optimal bin packing algorithms*. PhD thesis, Departamnt of Mathematics, Massachusetts Institute of Technology, August 1973. (Citado en página 12.)

[24] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237 – 250, 1982. (Citado en página 12.)

[25] B. Andersson and J. Jonsson, "Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition," in *Proceedings Seventh International Conference on Real-Time Computing Systems and Applications*, pp. 337–346, Dec 2000. (Citado en página 13.)

[26] G. Koren, A. Amir, and E. Dar, "The power of migration in multi-processor scheduling of real-time systems.," *SIAM Journal on Computing*, vol. 30, pp. 226–235, 08 2000. (Citado en página 13.)

[27] K. S. Hong and J. Y. . Leung, "On-line scheduling of real-time tasks," in *Proceedings. Real-Time Systems Symposium*, pp. 244–250, Dec 1988. (Citado en páginas 16 and 19.)

[28] A. Srinivasan and J. Anderson, "Fair scheduling of dynamic task systems on multiprocessors," *Journal of Systems and Software*, vol. 77, pp. 67–80, 07 2005. (Citado en página 19.)

[29] Phillips, Stein, Torng, and Wein, "Optimal time-critical scheduling via resource augmentation," *Algorithmica*, vol. 32, pp. 163–200, Feb 2002. (Citado en páginas 19 and 20.)

[30] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," *J. ACM*, vol. 47, pp. 617–643, july 2000. (Citado en páginas 19 and 20.)

[31] T. W. Lam and K. K. To, "Trade-offs between speed and processor in hard-deadline scheduling," in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, (Philadelphia, PA, USA), pp. 623–632, Society for Industrial and Applied Mathematics, 1999. (Citado en página 20.)

[32] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia, "Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems," in *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*, pp. 25–33, June 2000. (Citado en página 21.)

[33] S. Baruah, S. Funk, and J. Goossens, "Robustness results concerning EDF scheduling upon uniform multiprocessors," *IEEE Transactions on Computers*, vol. 52, pp. 1185–1195, Sep. 2003. (Citado en páginas 23 and 24.)

[34] O. de Antonio Suárez *INGE @ UAN-Tendencias en la Ingeniería*, vol. 1, no. 2, 2013. (Citado en página 25.)

[35] Jingpeng Li and U. Aickelin, "A bayesian optimization algorithm for the nurse scheduling problem," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 3, pp. 2149–2156 Vol.3, Dec 2003. (Citado en páginas 25, 27, and 68.)

[36] F. Grenouilleau, A. Legrain, N. Lahrichi, and L.-M. Rousseau, "A set partitioning heuristic for the home health care routing and scheduling problem," *European Journal of Operational Research*, vol. 275, no. 1, pp. 295 – 303, 2019. (Citado en página 25.)

[37] J. Marynissen and E. Demeulemeester, "Literature review on multi-appointment scheduling problems in hospitals," *European Journal of Operational Research*, vol. 272, no. 2, pp. 407 – 419, 2019. (Citado en página 25.)

[38] R. Ruiz, Q.-K. Pan, and B. Naderi, "Iterated greedy methods for the distributed permutation flowshop scheduling problem," *Omega*, vol. 83, pp. 213 – 222, 2019. (Citado en página 25.)

[39] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, vol. 34, no. 3, pp. 391–401, 1988. (Citado en página 25.)

[40] A. V. Aho, J. E. Hopcroft, and J. Ullman, *Data Structures and Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1983. (Citado en página 25.)

[41] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd ed., 2008. (Citado en página 27.)

[42] J. Yang, H. Xu, L. Pan, P. Jia, F. Long, and M. Jie, "Task scheduling using bayesian optimization algorithm for heterogeneous computing environments," *Applied Soft Computing*, vol. 11, no. 4, pp. 3297 – 3310, 2011. (Citado en páginas 27, 66, and 68.)

[43] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Boa: The bayesian optimization algorithm," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, GECCO'99, (San Francisco, CA, USA), pp. 525–532, Morgan Kaufmann Publishers Inc., 1999. (Citado en página 27.)

[44] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, pp. 309–347, Oct 1992. (Citado en página 27.)

[45] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Science*, vol. 6, pp. 1–12, October 1959. (Citado en página 29.)

[46] S. Kato and N. Yamasaki, "Scheduling aperiodic tasks using total bandwidth server on multiprocessors," in *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, vol. 1, pp. 82–89, Dec 2008. (Citado en página 30.)

[47] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *[1991] Proceedings Twelfth Real-Time Systems Symposium*, pp. 129–139, Dec 1991. (Citado en página 30.)

[48] T. P. Baker, "An analysis of EDF schedulability on a multiprocessor," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, pp. 760–768, August 2005. (Citado en página 30.)

[49] N. Guan, W. Yi, Z. Gu, Q. Deng, and G. Yu, "New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms," in *2008 Real-Time Systems Symposium*, pp. 137–146, Nov 2008. (Citado en página 30.)

[50] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intelligence*, vol. 29, no. 3, pp. 241 – 288, 1986. (Citado en páginas 45, 46, 51, 53, and 76.)

[51] M. Garey, D. Johnson, and M. S. M. Collection, *Computers and Intractability: A Guide to the Theory of NP-completeness*. Books in mathematical series, W. H. Freeman, 1979. (Citado en página 45.)

[52] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988. (Citado en página 53.)

[53] J. H. Kim and J. Pearl, "A computational model for causal and diagnostic reasoning in inference systems," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'83, (San Francisco, CA, USA), pp. 190–193, Morgan Kaufmann Publishers Inc., 1983. (Citado en páginas 53 and 76.)

[54] A. F. Mills and J. H. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 311–320, April 2010. (Citado en páginas 67 and 71.)

[55] S. Funk and Sanjoy Baruah, "Task assignment on uniform heterogeneous multiprocessors," in *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pp. 219–226, July 2005. (Citado en página 70.)

[56] J. Van Mieghem, "Due-date scheduling: Asymptotic optimality of generalized longest queue and generalized largest delay rules," *Operations Research*, vol. 51, pp. 113–122, 02 2003. (Citado en página 71.)

[57] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Sistemas expertos y modelos de redes probabilísticas*. Madrid : Academia de Ingeniería, 1996. (Citado en página 73.)