Asignación de Tareas en Sistemas Distribuidos de Tiempo Real

Norma Perez, Mario Berón, Enrique Miranda, Hernán Bernardis, Mariano Luzza, Matias Truglio
Facultad de Ciencias Físico Matemáticas y Naturales - Universidad Nacional de San Luis
Ejército de los Andes 950 - San Luis- Argentina
email: {nbperez,mberon,eamiranda,hbernardis,mluzza,mitrugli}@unsl.edu.ar

Resumen

El crecimiento excesivo de la tecnología en las computadoras y el surgimiento de bases de datos con grandes volúmenes de información, han hecho que en los últimos años, el uso de sistemas distribuidos de tiempo real sea tenido en cuenta para el desarrollo de aplicaciones en diversas áreas tales como: sistemas de aviación, orientación de misiles, navegación, sistemas robóticos para manipulación de materiales peligrosos, entre otros. El objetivo de tales sistemas es principalmente proporcionar mejores tiempos de respuesta. Por esta razón, en el contexto de los sistemas distribuidos, despierta interés el problema de *asignación de tareas*; que es un problema NP-completo.

En este artículo se presenta una línea de investigación que consiste en el estudio de algorítmos eficientes para la asignación de tareas.

Palabras clave: Sistemas Distribuidos de Tiempo Real, Asignación de Tareas, Optimización, Ford-Furkelson.

1. Contexto

La línea de investigación descripta en este artículo se encuentra enmarcada en el contexto del proyecto: *'Ingeniería del Software: Conceptos, Métodos, Técnicas y Herramientas en un Contexto de Ingeniería de Software"* de la Universidad Nacional de San Luis. Dicho proyecto, es reconocido por el programa de incentivos, y es la continuación de diferentes proyectos de investigación de gran éxito a nivel nacional e internacional.

2. Introducción

Los Sistemas Distribuidos de Tiempo Real (SDTR) han sido cada vez más utilizados en la resolución de problemas que demandan grandes cantidades de tiempo de procesamiento, ya que permiten la utilización simultánea de varios recursos computacionales. Para garantizar el buen funcionamiento de los sistemas de tiempo real, no es suficiente que las acciones necesarias para ejecutar las tareas estén correctamente implementadas, las mismas deben ser ejecutadas en un intervalo de tiempo especificado. Si el intervalo de tiempo asignado para la ejecución de tareas no es suficiente, se pueden producir fallas eventualmente catastróficas. Como ejemplos, se pueden considerar: sistemas de aviación, robótica, reproducción de imágenes de TV, frenado ABS, etc., en los cuales una acción tardía (o el no cumplimiento de restricciones) puede conducir a situaciones catastróficas. Un ejemplo es presentado en la Figura 1, donde se distinguen tres situaciones:

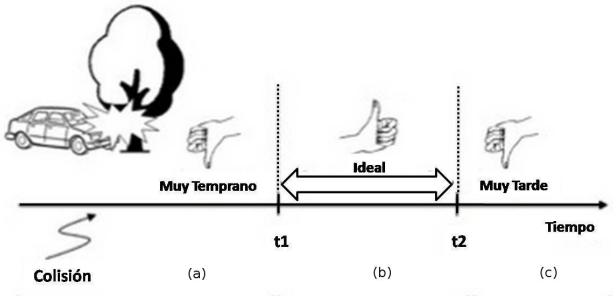

Figura 1: Sistema *airbag* de un automóvil en una colisión.

(a) El *airbag* se infla demasiado pronto. El sistema produce que el *airbag* sea desinflado antes de proteger al automivilista del impacto. (b) El *airbag* se infla en el instante adecuado entre los tiempos t_1 y t_2. (c) El *airbag* se infla demasido tarde. Esta situación lleva a que el automovilista choque con el volante cuando ocurre la colisión.

El desempeño de los SDTR's será afectado si no se realiza un buen planeamiento de la distribución de tareas [5, 6] en los procesadores que participan en el

sistema. Esta no es una tarea fácil ya que existen m^n maneras posibles de asignación [7] en un sistema con m tareas y n procesadores. En un intento de obtener solución para este problema, los investigadores usan diferentes abordajes: *algoritmos basados en teoría de grafos* [2, 12], *algoritmos heurísticos* [1, 2, 4, 7], entre otros métodos.

Este artículo está organizado de la siguiente manera. La sección 3 describe la línea de investigación. La sección 4 presenta los resultados obtenidos. Finalmente, la sección 5 describe brevemente las tareas realizadas en el contexto de los recursos humanos.

3. Líneas de investigación y desarrollo

El desempeño de los sistemas puede ser mejorado por medio de la utilización de varios procesadores [13].

Considere un sistema compuesto por m tareas y n procesadores, donde $\mathcal{T} = \{t_1, ..., t_m\}$ representa el conjunto de tareas, y $\mathcal{P} = \{p_1, ..., p_n\}$ los procesadores, y considere que se tiene un mecanismo de asignación que atribuye cada tarea t_i, $1 \leq i \leq m$ a uno de los p_j procesadores, $1 \leq j \leq n$. En esta línea de investigación se estudia el diseño e implementación de mecanismos de asignación de tareas para un SDTR con las siguientes características:

- Se utiliza asignación estática, donde las especificaciones de cada tarea implicada en el sistema son conocidas a priori.
- El sistema está compuesto por procesadores idénticos, interconectados a través de una red de comunicación.
- El sistema está compuesto por tareas periódicas.
- Cada tarea es una tupla (t_i, C_i, T_i), donde t_i es el número de la tarea, C_i su tiempo de ejecución y T_i el plazo para su ejecución.
- Cada tarea es independiente en el sentido que no necesita de la ejecución de otra tarea para completar su ejecución.
- Cada tarea se ejecuta completamente en el procesador al cual fue asignada.
- Las tareas asignadas a cada procesador tienen prioridades fijas.
- Las tareas no son interrumpibles.

3.1. Complejidad del problema

El problema de asignación pertenece a la clases de problemas NP-Hard [8], ya que:

- La verificación de cada tarea está asociada a un procesador y se puede realizar en tiempo lineal.
- La verificación de que la capacidad de cada procesador se respeta, puede ser realizada en tiempo lineal.

Para probar que este problema es NP-Hard, se debe reducir un problema NP-Completo al mismo. Con esta finalidad se seleccionó el problema de suma de los conjuntos ya que facilita el proceso de reducción. El mismo se define a continuación.

3.2. Problema de Suma de Conjuntos

Una instancia del problema de suma de subconjuntos es un par (L,k) donde L es un conjunto= $\{ e_1, e_2, ..., e_n \}$ *de enteros positivos y k es un entero positivo.*
Este problema de decisión pregunta si existe un subconjunto de L cuya suma sea exactamente el valor de destino k. Una demostración de que este problema de suma de subconjuntos es NP-completo se presenta en [8]. Se necesita ahora transformar una instancia arbitraria del *problema de suma de subconjuntos* en una instancia del *problema de distribución de tareas*, de manera tal que una solución al problema de las particiones de subconjuntos sea positiva sí y solo sí una solución del correspondiente problema de distribución de tareas fuese positiva.
Suponga que $L = \{e_1, e_2, ..., e_n\}$ es una lista de elementos del problema de suma de subconjuntos. Se usa un problema de asignación con dos procesadores con capacidad $\frac{\sum_{e_k \in L}(e_k)}{2}$ cada uno y con un número de tareas $n = |L|$. Cada tarea del problema de asignación consume, en cualquiera de los dos procesadores, la misma cantidad de recursos e_k y tiene costo unitario en cualquiera de ellos.

El problema es verificar sí existe una distribución de costo menor o igual a $|L|$ en el problema de distribución de tareas.

La respuesta será positiva sí y solo sí fuese posible asignar todas las tareas. Para conseguir tal objetivo, se deben dividir las tareas en dos procesadores de igual capacidad. Como la suma de estas es igual a la sumatoria de los elementos de L, esto es solo posible cuando la capacidad total de los dos procesadores es utilizada. De este modo, el problema de suma de subconjuntos fue reducido al problema de distribución de tareas. Esta reducción puede ser realizada en tiempo polinomial.

3.3. Algoritmos

Existen distintos algoritmos que resuelven el problema planteado en la sección anterior, a continuación se describen algunos de ellos:

Simulated Annealing (SA)
Es una función metaheurística que pertenece a las clases de algoritmos de búsqueda local, que hace uso de una técnica de búsqueda local y probabilística, y se fundamenta en una analogía con la termodinámica [9].

Este algoritmo es viable para el problema de asignación de tareas donde los resultados obtenidos están próximos del ideal. El objetivo del mismo es la obtención de un óptimo local cuyo valor es cercano al valor del óptimo global, teniendo un buen comportamiento cuando el tamaño de los datos de entrada aumenta. El Apéndice 1 del artículo [15] presenta el problema de asignación a través de un ejemplo que muestra como el algoritmo encuentra una buena solución en un problema artificial; los detalles de estas pruebas son presentados en [14].

Restricciones de precedencia en SDTR
Oroscos y su grupo de investigación [11] proponen un método para resolver el problema de *jitter* a través de un dispositivo conectado al planificador con el objetivo de detectar y medir con anticipación el dato de llegada de su predecesor. La liberación de la tarea puede ser pospuesta hasta llegar a un intervalo correcto para su liberación. El dispositivo utilizado es un contador de liberación. Además los autores presentan sus principales conclusiones obtenidas al evaluar, en un SDTR, la asignación de tareas a lo largo de un conjunto de procesadores teniendo en cuenta las restricciones impuestas sobre las tareas involucradas en el sistema.

Ambos trabajos descriptos anteriormente utilizan para sus experimentos, la misma cantidad de tareas y la misma tabla de valores correspondiente a cada tarea involucrada en la simulación.

4. Resultados

Los resultados obtenidos hasta el momento se relacionan con la elaboración de un algoritmo que se aplica a SDTR's, y garantiza el cumplimiento de las restricciones que fueron impuestas sobre las tareas. A continuación se describen los conceptos fundamentales en los que se basa la idea.

4.1. Algoritmo Basado en Flujos en Grafos de Redes de dos Terminales (AFR)

En un grafo G de dos terminales se supone que existen dos nodos especiales: Un nodo de origen y un nodo de destino. En ese tipo de grafo se puede determinar un *Conjunto de Corte*(CC). Un CC es un conjunto de aristas tales que, cuando son eliminadas, desconectan el grafo y lo particionan en dos componentes. Una componente contiene al nodo origen y se denomina *Conjunto Origen* y la otra contiene el nodo destino que se conoce con el nombre de *Conjunto Destino*. Cada corte, de un grafo posee un peso asociado, denotado por *W(C)*, que es igual a la suma de los pesos de todas las aristas en C. Un conjunto de corte mínimo (u óptimo) es un conjunto de corte cuyo peso es menor que el peso de cualquier otro conjunto de corte del grafo.

AFR supone la restricción de que existen dos procesadores en el sistema distribuido. El algoritmo transforma el grafo de interacción de tareas en un grafo de red de dos terminales. Además, coloca uno de los procesadores como el nodo origen y el otro como el nodo destino y encuentra el conjunto de corte mínimo en el grafo. Las tareas que quedan en el conjunto origen resultante deberán ser asignadas al procesador correspondiente al nodo origen y las tareas que quedan son asignadas al otro procesador. El algoritmo de AFR es descripto en [2, 12].

4.2. Método Ford-Fulkerson

Este método propone buscar caminos que aumenten el flujo hasta alcanzar un flujo máximo. La idea es encontrar una ruta de penetración desde el nodo origen al nodo destino con el flujo positivo adecuado. Este algoritmo fue utilizado para obtener el conjunto de corte mínimo C. Un corte define un conjunto

de aristas cuya eliminación de la red provoca una interrupción completa del flujo entre origen y destino.

La capacidad del corte es igual a la suma de las capacidades de las aristas asociadas a él. Entre todos los posibles cortes en la red, el corte con menor capacidad provee el flujo máximo en la red. Este método está completamente descripto en [3]. Las siguientes, suposiciones fueron consideradas para la implementación del algoritmo.

- Los identificadores de los nodos en el grafo son números positivos.
- Las capacidades en un sentido son positivas y en la dirección opuesta tienen valor 0 (cero).

4.3. Rate Monotonic

El modelo inicial propuesto por Liu y Layland en 1972 [10], fue la base de la teoría para la planificación con prioridades fijas. Rate Monotonic (RM) era bastante limitado porque los autores impusieron una larga lista de restricciones sobre el modelo computacional. A lo largo de los años surgieron estudios que eliminan parte de las restricciones originales, haciendo que el modelo de planificación con prioridades pase a ser muy completo, poderoso y atractivo. La asignación de prioridades se realiza de acuerdo con los períodos de tareas, o sea, son asignadas prioridades más altas a las tareas con plazo más corto. *Rate Monotonic Scheduling* (RMS) es un método basado en RM que es óptimo, en el sentido de que un conjunto de tareas es escalable con cualquier asignación de prioridades. Liu et. al en [10] presenta una medida de utilización de RM para un conjunto de m tareas usadas en el sistema. La solución propuesta se basa en la extensión de un algoritmo de asignación de tareas para *dos* procesadores basado en el algoritmo de Ford-Fulkerson a otro de n procesadores, desarrollado por los autores de este artículo [10]. El algoritmo funciona como sigue. En el primer paso el algoritmo asigna i tareas P_1 y j tareas a P_2. Luego, si existen más procesadores, se aplica el mismo razonamiento para el subgrafo determinado por las i tareas que fueron asignadas al procesador P_1, y ahora se hace una nueva asignación de tareas al procesador adicional P_3, incorporando una partición diferente. Esto significa que el procesador P_1 ahora va a tener un conjunto de k tareas y P_3 otro de l tareas, donde $i = k + l$ y así siguiendo. Los mismos pasos son realizados en el procesador destino.

4.4. Experimentos Realizados

Con la finalidad de estudiar el comportamiento del sistema previamente descripto, se planificaron experimentos considerando:

- diferentes grafos de tareas, donde se varía el número de ellas y los valores de las aristas. A continuación se describen los experimentos.
- **EXPERIMENTO 1.** Consiste en mostrar la asignación de las tareas a los procesadores. Para esto se realizaron diferentes simulaciones, variando el número de procesadores. El algoritmo utilizado se basa en el método de Ford-Fulkerson para distribuir las tareas en los diferentes procesadores implicados en la simulación. Se sabe que el algoritmo de Ford-Fulkerson es sensible a los valores usados para rotular las aristas del grafo analizado.

  El análisis de los resultados se centra en mostrar que el algoritmo consigue asignar las tareas de manera más uniforme entre los procesadores que los algoritmos en la sección 3.3.

  Es importante notar que en el caso de existir mayor cantidad de tareas que de procesadores, se debe realizar una planificación adecuada para la asignación de las tareas, con el fin de evitar sobrecargar algunos procesadores y evitar que otros queden ociosos.
- **EXPERIMENTO 2:** El objetivo de este experimento es mostrar la utilización del procesador en un sistema de prioridad fija, descripto en [10]. Se espera obtener un valor próximo al teórico que es del 70 %, a fin de conseguir una mayor utilización de los procesadores involucrados en el sistema.
- **EXPERIMENTO 3:** La idea es mostrar la cantidad de tareas que fueron ejecutadas y aquellas que no por el algoritmo propuesto. Este experimento es importante porque deja en evidencia cuántas tareas no fueron ejecutadas en el plazo esperado. Se sabe que esta situación se presenta cuando el esquema de asignación escogido es *apropiativo* para las tareas involucradas a la simulación. Otro factor influyente es el tiempo de ejecución que les fue atribuido. En

este experimento se emplearan diferentes test's donde la atribución de valores para cada elemento que la componen será variada.

– **EXPERIMENTO 4:** El objetivo es obtener el tiempo de respuesta producido por el sistema. A través de este experimento se pretende mostrar cómo se comporta el algoritmo propuesto a medida que se incorporan más procesadores. Se espera que más tareas sean ejecutadas por estos procesadores, disminuyendo así la cantidad de tareas que no consiguen ser ejecutadas en los tiempos establecidos.

Finalmente, se propone relajar las restricciones impuestas al algoritmo. Esta motivación se debe a que los sistemas de tiempo real tienen que ser seguros y precisan de un buen tiempo de respuesta.

5. Formación de Recursos Humanos

Las tareas realizadas en presente línea de investigación están siendo realizadas como parte del desarrollo de tesis para optar por el grado de Licenciado en Ciencias de la Computación. Se espera a corto plazo poder definir, a partir de los resultados obtenidos en las tesis de licenciatura en curso, tesis de maestría o bien de doctorado, como así también trabajos de Especialización en Ingeniería de Software.

Referencias

[1] Nicholas S. Bowen, Christos N. Nikolaou, and Arif Ghafoor. On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *IEEE Trans. Comput.*, 41:257–273, March 1992.

[2] W. W. Chu, L. J. Holloway, Min-Tsung Lan, and K. Efe. Task allocation in distributed data processing. *Computer*, 13:57–69, November 1980.

[3] T.H. Cormen. *Introduction to algorithms*. MIT electrical engineering and computer science series.

[4] Kermal. Efe. Heuristic models of task assignment scheduling in distributed systems. *Computer*, 15:50–56, June 1982.

[5] Hesham El-Rewini and Ted G. Lewis. *Distributed and parallel computing*. Manning Publications Co., Greenwich, CT, USA, 1998.

[6] Hesham El-Rewini, Theodore G. Lewis, and Hesham H. Ali. *Task scheduling in parallel and distributed systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[7] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. Softw. Eng.*, 15:1427–1436, November 1989.

[8] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[10] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20:46–61, January 1973.

[11] J. Orozco, R. Cayssials, J. Santos, and E. Ferro. Precedence constraints in hard real-time distributed systems. In *Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems*, pages 33–, Washington, DC, USA, 1997. IEEE Computer Society.

[12] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Softw. Eng.*, 3:85–93, January 1977.

[13] Andrew. S. Tanembaum. *Sistemas Operacionais Modernos*. Pearson Prentice Hall do Brasil, 2003.

[14] K. W. Tindell. Allocating real-time tasks an np-hard problem made easy. December 1990.

[15] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: an np-hard problem made easy. *Real-Time Syst.*, 4:145–165, May 1992.