Centro de Investigación y de Estudios
Avanzados del IPN

Departamento de Ingeniería Eléctrica
Sección de Computación

Planificador de Tareas en Tiempo Real con Restricciones de Energía en Retroalimentación

Tesis de Maestría

Ing. Julio César Cornejo Herrera
jcornejo@computacion.cs.cinvestav.mx

Asesores:

Dr. Pedro Mejía Alvarez
pmejia@cs.cinvestav.mx
Dr. Alberto Soria López
soria@ctrl.cinvestav.mx

noviembre 2002

Resumen

En esta tesis presentamos un *planificador de tareas en tiempo real retroalimentado con restricciones de energía* para sistemas con comportamiento dinámico, en donde los tiempos de ejecución de las tareas no se conocen con exactitud. Aplicamos una metodología basada en la teoría de control para el diseño del planificador retroalimentado para satisfacer las especificaciones de desempeño en estado transitorio y estado estable de un sistema en tiempo real adaptable.

Proponemos un planificador de tareas en tiempo real con una arquitectura en retroalimentación o lazo cerrado, en donde la carga del sistema se ajusta dinámicamente por medio de un controlador *PID* ampliamente utilizado en la industria del control de procesos. La variación de la carga en el sistema se obtiene cambiando las velocidades de ejecución de las tareas del sistema mediante el uso de *procesadores de velocidad variable*. Debido a que las tareas en el sistema consumen una cierta cantidad de energía, la cual depende de la velocidad seleccionada para su ejecución, podemos obtener un ahorro en el consumo de energía tratando de ejecutar el mayor número de tareas a la menor velocidad posible. Por otro lado, al ejecutar una tarea a baja velocidad su tiempo de ejecución se incrementa, por lo que es necesario mantener un equilibrio entre el ahorro en el consumo de energía y la perdida de plazos.

El proceso de selección de las velocidades de ejecución mientras maximizamos la energía ahorrada en el sistema requiere la exploración de un gran número de combinaciones, lo cual

consume demasiado tiempo en calcularse. Para ello, utilizamos una heurística la cual ejecuta un procedimiento optimizado en un tiempo de computo bajo, este problema de optimización está formulado como una aproximación al problema de la mochila con múltiples opciones *(multiple-choices knapsack problem)* con variables binarias.

En las pruebas de simulación realizadas, utilizamos las políticas de planificación más representativas de los algoritmos de planificación en tiempo real (*Rate Monotonic (RM)* para la planificación estática y *Earliest Deadline First (EDF)* en el caso de la planificación dinámica) con el fin de observar el comportamiento del sistema controlado y comparar sus efectos en los niveles de energía ahorrada.

En la configuración del sistema las variables de control utilizadas en los lazo de control, son la *utilización*, la *razón de plazos perdidos* y el control integrado, la cual utiliza las dos variables de control (*utilización + razón de perdida*).

Abstract

In this thesis we develop the algorithms of a real-time feedback scheduler with energy constraints for systems executing in a dynamic environment, where the execution time of the tasks is unknown. We use a methodology based on control theory for the design of the feedback scheduler that yield the specifications of performance in transient and steady state of an adaptive real-time system.

We propose a closed loop feedback real-time scheduling architecture, where the workload is dynamically adjusted by a *PID* controller widely used in the control process industry. The variation of the system load is obtained by adjusting the execution speed (or frequency) of all tasks using variable speed processors. The tasks in the system consume a certain amount of energy, which depend on selected speed. We can save energy consumption by executing the tasks at the minimum possible speed. However, when a low speed is selected, the task's execution time will increase possibly jeopardizing the temporal execution of the tasks. So is necessary make a trade-off between saving energy and minimizing the number of deadline misses.

The process of selecting the execution speed while minimizing the energy consumption in the system requires the exploration of a large number of combinations, which is too time consuming to be computed on-line. To solve this problem, we develop an integrated heuristic which executes an optimization procedure in a short computation time. This optimization problem is formulated as a multiple-choices knapsack problem (MCKP) with binary variables.

We execute an extensive set of tests to simulate the performance of our feedback real-time scheduler architecture. In this simulations we use two of the most employed scheduling polices in real-time scheduling systems, (*Rate Monotonic (RM)* for static scheduling and *Earliest Deadline First (EDF)* for dynamic scheduling), and observe the behavior of the controlled. In the system configuration we use the followings control variables in the control loops: *utilization*, *miss deadline ratio*, and a combination of both *utilization + miss ratio* integrated control.

Dedicatoria

Este trabajo esta dedicado a la memoria de mi abuelo Don Emilio Herrera Hernandez Q.E.P.D.

Agradecimientos

Al *CONACYT* por el apoyo económico que me brindó durante el transcurso de la maestría.

Agradecimiento especial para mis asesores, al Dr. Pedro Mejía Álvarez y al Dr. Alberto Soria López, por sus consejos y quienes sin su ayuda no podría haber terminado este trabajo. También a los doctores Dr. Guillermo Benito Morales Luna y Dr. Luis Gerardo de la Fraga por sus consejos en la escritura de este documento.

A los doctores de la *Sección de Computación del Departamento de Ingeniería Eléctrica*, por sus excelentes clases, en especial al Dr. Pedro Mejía por su enseñanzas en sistemas de tiempo real e ingeniería de software, al Dr. Gerardo de la Fraga quien con sus exigencias en las clases de graficación y procesamiento de imágenes nos demostró que si se puede, a los profesores M. en C. Ulises Juárez, Dr. José Oscar Olmedo, Dra. Xiaoou Li, Dr. Jorge Buenabad, Dr. Sergio Víctor Chapa, Dr. Adriano de Luca y al Dr. Arturo Díaz por sus consejos en clase.

A mi querida esposa Laura de Montserrat, con quien inicie una aventura, de la cual ya da los primeros frutos. Gracias por tu paciencia, por tu comprensión y sobre todo gracias por que siempre estuviste ahí.

A mis padres, por la educación que inculcaron en mi. Gracias por todo su cariño. A mis hermanos, hermanas y sobrinos.

A Sofia Reza y Anabel Díaz por toda su ayuda.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Motivación

La presente tesis, está motivada por la observación de que muchos de los sistemas de cómputo móvil (portátil) o cómputo embebido sobre los que operan los sistemas de tiempo real son altamente dinámicos, impredecibles y con requerimientos estrictos de tiempos y consumo de energía. Estos ambientes se presentan en aplicaciones como teléfonos celulares, *PDA´s (personal digital assistans)*, control de procesos, bases de datos en tiempo real, etc.

La reducción en el consumo de potencia es un reto en el diseño de sistemas portátiles. Estos dispositivos obtienen su alimentación de baterías, por lo tanto reduciendo el consumo de la potencia extendemos su tiempo de operación. El consumo de potencia es un problema que también afecta a las computadoras de escritorio o servidores, ya que un alto consumo de potencia eleva la temperatura de estos dispositivos y deteriora el desempeño y la confiabilidad del sistema.

En esta tesis, se diseñarán técnicas de manejo de procesos en sistemas operativos de tiempo real que permitan reducir el consumo de energía en sistemas de cómputo portátiles y embebidos. Dichas técnicas incluirán la capacidad de monitorizar, medir, y optimizar el

consumo de la energía consumida por los procesos además de planificar los procesos de forma que cumplan con sus tiempos de respuesta. En la actualidad, las técnicas utilizadas en la planificación de este tipo de sistemas producen resultados en donde el tiempo de diseño es alto y el desempeño y la confiabilidad es baja, además de que no se tienen garantías sobre si se cumplirán o no las restricciones temporales de las tareas y las restricciones en el consumo de energía.

Por otro lado, la configuración de los algoritmo de planificación existentes, como RM (*Rate Monotonic*) o EDF (*Earliest Deadline First*), están diseñadas para ejecutarse en ambientes estáticos [1] en donde los parámetros de sistema no cambian con el tiempo. Estas restricciones limitan la adopción de estas técnicas en ambientes de cómputo dinámicos en donde los parámetros del sistema (como son los tiempos de cómputo, tiempos de arribo de los procesos o plazos de respuesta) cambian continuamente. Por su naturaleza estática, dichas técnicas de planificación no son capaces de reaccionar o adaptarse a cambios en el ambiente, ya que se ejecutan en lazo abierto. Lazo abierto se refiere al hecho de que, una vez planificado un conjunto de tareas, los parámetros temporales de éstas no se ajustan en base al comportamiento del sistema. Este tipo de planificación es adecuado en sistemas en donde los parámetros temporales no cambian durante la ejecución, estos son conocidos en todo momento, y en donde no existen sobrecargas en el sistema. Sin embargo, este tipo de planificadores no es adecuado para sistemas en donde las tareas arriban en tiempos desconocidos y los tiempo de ejecución de las tareas no se conocen con precisión. Para este tipo de sistemas es conveniente utilizar planificadores con configuración en *lazo cerrado*.

En esta tesis, se plantea el diseño de mecanismos en el sistema operativo para la planificación de tareas en tiempo real con restricciones de tiempo y consumo de energía sobre sistemas altamente dinámicos, lo cual requiere del uso de técnicas de planificación en lazo cerrado.

Son diversos los factores que han motivado un incremento en las investigaciones referentes

[1] El término estático y dinámico en el contexto de sistemas en tiempo real, se refiere al ambiente de ejecución de las tareas, se dice que el ambiente es estático cuando los parámetros temporales de las tareas no cambian

al ahorro en el consumo de la energía en los sistemas de cómputo. Entre los factores más importantes, están, el crecimiento del número de sistemas portátiles utilizados en la actualidad, las tecnologías de baterías las cuales actualmente limitan el tiempo de operación de este tipo de sistemas de cómputo, el aumento de computadoras personales y servidores, la confiabilidad de los sistemas y también los costos ambientales que resulta de la generación de electricidad.

Sistemas portátiles

En los últimos años, se ha observado un incremento en el mercado de dispositivos electrónicos portátiles [39]. Aún y cuando estas formas tradicionales de cómputo móvil continuarán en aumento en los próximos años, los analistas pronostican un incremento exponencial en el mercado de dispositivos portátiles debido a la evolución de las nuevas tecnologías como las redes de área personal inalámbricas que proporcionarán acceso a *Internet* y contenido de multimedia en teléfonos celulares de tercera generación y en *PDA's (personal digital assistans)*. Por lo que la reducción en el consumo de energía es un punto clave en el diseño de estos dispositivos.

En la figura 1.1 se muestran los valores de la potencia consumida por los componentes de una *laptop Toshiba* 410 CDT [44]. Los resultados de la figura 1.1 están basados en estimaciones sobre un uso típico. Las medición de potencia para el sistema completo fue de 14 *Vatios*, la cual es pequeña comparada con los dispositivos eléctricos (por ejemplo, una lámpara consume 60 *Vatios*), pero es muy grande si se compara con un sistema que se energiza con baterías.

En la figura 1.1 se observa que los dispositivos que más consumen energía en un sistema de cómputo móvil o portátil son el *display* (pantalla de vídeo), el procesador/memoria y el disco duro. Por lo que una buena medida para disminuir el consumo de energía es estos dispositivos es implementando técnicas para el manejo de energía en el procesador. Además, en sistemas embebidos que no cuentan con *display*, discos ni con fuente de poder, la energía es consumida mayormente por el procesador y la memoria.

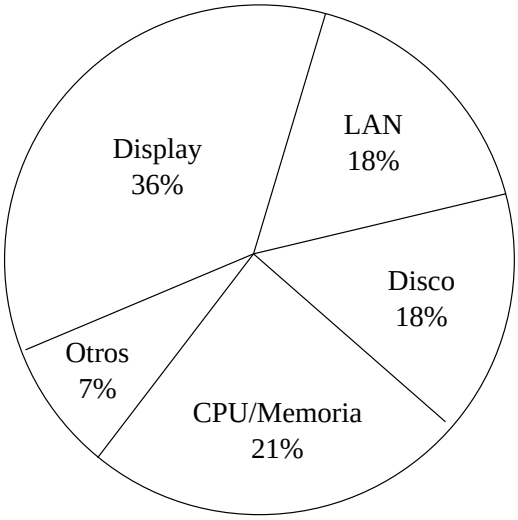


Figura 1.1: Consumo de energía de los componentes de un sistema de cómputo portátil

Ahorro de energía vs. Tecnología de baterías

Una de las restricciones más importantes en el diseño de sistemas portátiles es el bajo consumo de energía [39]. El consumo de la energía dicta el tiempo de vida de las baterías en los sistemas portátiles. Es un hecho que la velocidad de los procesadores y su consumo de la energía se ha incrementado rápidamente. Sin embargo, el mejoramiento en las tecnologías de baterías ha sido lenta en comparación con el resto de los subsistemas que componen un dispositivo de cómputo móvil. El desempeño de las baterías se caracteriza por la cantidad total de energía que éstas puedan almacenar y por sus dimensiones físicas (peso y tamaño). Hoy en día la mayor parte de las baterías son de *Litio-ion (Li-ion)*, las cuales tienen una mejor densidad de energía (energía por unidad de peso) en comparación con las antiguas baterías de *Níquel Cadmio (NiCd)* y *Hidruro de Níquel Metal (NiMH)*.

Otro factor importante en los sistemas de cómputo móvil es el peso de las baterías. Una solución impráctica al problema del tiempo de vida limitado de las baterías, sería cargar con varias baterías y simplemente reemplazarlas cuando se necesite. Existen *laptops* donde es posible instalar dos baterías, extendiendo el uso de las *laptops* (hasta por 8-12 horas de operación contínua) pero a expensas de un aumento en el peso. La capacidad de las baterías

se ha incrementado solamente en un 400% en la última década. Nuevas tecnologías ofrecen mayores capacidades, pero tienen la desventaja de ser demasiado costosas. Bajo estas circunstancias, el diseño de sistemas con manejo de energía se ha vuelto más rentable en comparación con el uso de nuevas tecnologías de baterías.

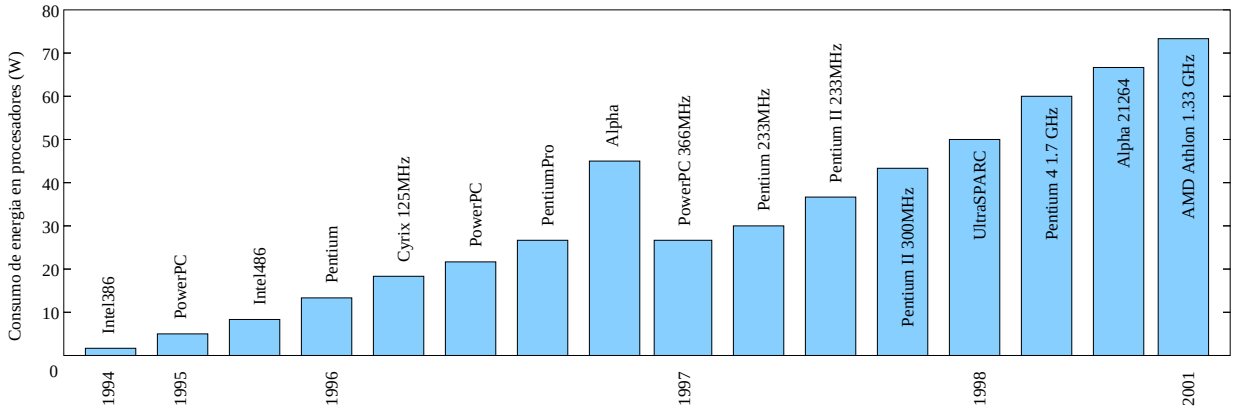


Figura 1.2: Tendencias en el consumo de energía en los microprocesadores

Computadoras personales y servidores

Un incremento en el uso de técnicas para la reducción en el consumo de energía en computadoras personales y servidores se traduce en una reducción significativa en los costos de electricidad, así como en los costos de enfriamiento [39]. Un estudio realizado por el gobierno de los Estados Unidos [16], mostró que el consumo de electricidad de todas las oficinas y equipos de redes en los Estados Unidos fue de casi 100 TWHr/año (como comparación, la electricidad residencial consumida fue cerca de 1100 TWHr/año en el 2000). En el mismo reporte [39], se ha estimado que utilizando técnicas de manejo de energía en los equipo de oficina, se podría obtener hasta un 35% de ahorro de energía. La figura 1.2 muestra el incremento en el consumo de energía (en Vatios) en procesadores de propósito general en los últimos años [39].

Confiabilidad y costos ambientales

En servidores de cómputo y equipos de red se consume una gran cantidad de energía, por lo cual la electricidad representa un costo considerablemente alto. El consumo de energía en

estos sistemas también se ve afectado por los siguientes factores[39]:

- *Disipación de calor.* La energía de los circuitos digitales es disipada en forma de calor. El calor generado tiene que ser disipado para que el dispositivo continué funcionando. Si la temperatura del circuito integrado se eleva arriba del rango especificado por el fabricante, el comportamiento del circuito podría cambiar. La temperatura de fundición de los circuitos es de 100 °C, por lo que para evitar que se alcance esta temperatura se emplean mecanismos de enfriamiento y empaquetamiento sofisticados, lo cual incrementa los costos. Por lo anterior podemos deducir que el diseño de sistemas de cómputo móvil con bajo consumo de energía se traduce en un bajo costo en mantenimiento y un incremento en la confiabilidad del sistema, además en una reducción en los costos de enfriamiento.

- *Factores ambientales.* Una universidad típica de los Estados Unidos de Norte-América puede contar con aproximadamente 10,000 computadoras. Si una computadora consume en promedio 150 *Vatios*, esto se traduce en tres millones kWHr al año de electricidad suponiendo un uso solo en horas de oficina. La emisión de gas requerida para generar esta electricidad es cerca de 2250 toneladas de bióxido de carbono y 208,000 árboles, lo cual implica un alto impacto ecológico.

1.2 Técnicas de Manejo de Energía

Las técnicas de reducción de potencia pueden ser clasificadas en estáticas y dinámicas [2]. Las técnicas estáticas son aplicadas en la etapa de diseño; como ejemplo están la síntesis y la compilación. Las técnicas dinámicas son aplicadas en tiempo de ejecución y operan sobre cargas de trabajo (conjunto de procesos en ejecución) que cambian con el tiempo.

Un sistema de cómputo está estructurado en capas como se muestra en la figura 1.3 [25]. El hardware se encuentra en la capa más baja y el sistema operativo se encuentra

[2]Los términos estático y dinámico mencionadas en esta sección, se refiere a la técnica utilizada en el manejo de energía

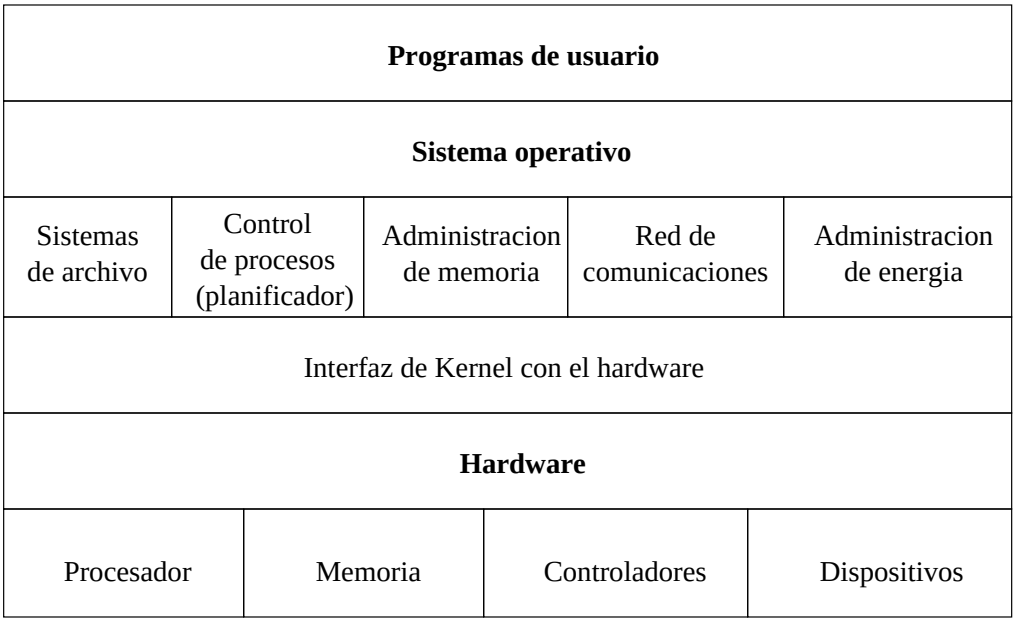


Figura 1.3: El manejo de energía puede implementarse en diferentes capas del sistema

entre el hardware y los programas de las aplicaciones. El sistema operativo se comunica con el hardware a través de llamadas al *BIOS*. El sistema operativo provee servicios para el manejo de archivos, manejo de procesos y la manejo de la memoria. Los programas de usuario accesan al hardware mediante llamadas al sistema a través del sistema operativo. La implementación de las técnicas de manejo de energía en cada capa cuenta con diversas problemáticas y tiene sus ventajas y desventajas. Un manejador de energía que se implementa en las capas inferiores, como por ejemplo en hardware, optimiza el consumo de la energía solo al nivel de algún componente de hardware individual. Sin embargo, para lograr una mayor reducción en el consumo de la energía es necesario conocer además las características de las capas superiores, en donde se encuentran el sistema operativo y los programas de usuario. Ninguna técnica aplicada a una sola capa es capaz de optimizar el consumo de la energía. Lo más recomendable, es verificar las fuentes de consumo de energía y las cantidades de consumo (como se indica en la figura 1.1), a fin de diseñar técnicas en las capas que más demanden energía del sistema.

Manejo de energía implementado en hardware

La reducción en el consumo de energía en los procesadores es posible mediante un diseño orientado al ahorro de energía [8]. Una de las formas más efectivas para lograr la reducción de energía de un circuito consiste en *reducir el voltaje de alimentación*. Esto se debe a que el consumo de energía disminuye cuadráticamente con el voltaje. El problema con esta técnica es que al reducir el voltaje se reducirá el desempeño del sistema, por lo que cualquier reducción en el voltaje debe estar balanceado a fin de contrarrestar cualquier caída en el desempeño. Para compensar y mantener el desempeño, es posible añadir hardware extra (lo cual podría implicar también consumir más energía). El problema de optimizar el consumo de la energía, mediante la reducción en el voltaje de alimentación, pero sin tener que aumentar el hardware es un problema de investigación de actualidad.

El consumo de energía en los circuitos CMOS es proporcional a la *capacitancia*. Esto se debe al hecho de que las conexiones a componentes externos consumen más energía ya que tienen una mayor capacitancia que las conexiones internas. Por ejemplo, el acceso a la memoria externa consume mucha energía, por lo que una forma de reducir la capacitancia es reducir los accesos externos y optimizar el sistema usando recursos como *caches* y registros. Una forma de reducir la capacitancia es disminuir el área del chip. Otra forma de reducir el consumo de energía en circuitos CMOS se consigue evitando actividades innecesarias del sistema. Una opción en este caso, consiste en desactivar componentes internos o periférico del sistema que no se utilicen por un periodo de tiempo. Algunos procesadores y dispositivos le desactivan el reloj a ciertas secciones del sistema con el fin de reducir el consumo de energía.

A nivel de diseño, el ahorro de energía se consigue principalmente reduciendo al mínimo el número de transistores en el sistema. Otra forma de reducir el consumo de energía se consigue aplicando metodologías de diseño asíncrono las cuales evitan actividades innecesarias. Esta es una buena técnica para los circuitos CMOS ya que las compuertas disipan energía solo cuando son activadas. Los circuitos asíncronos se activan cuando realizan trabajo útil.

Otras tecnologías pueden mejorar el ahorro de energía de los sistemas de almacenamiento. Los esquemas de memoria jerárquicas pueden ser utilizados en un sistema para reducir el

consumo de energía. La idea básica en estos esquemas, es almacenar el código o datos que se accesan con mucha frecuencia, en una pequeña parte de la memoria cercana al procesador, como la memoria cache. De está forma, la reducción en el consumo de energía se obtiene debido a que la mayor parte del tiempo sólo una pequeña memoria es leída. Otra forma de reducir el consumo de la energía, es utilizando memoria que se pueda dividir en bloques, con esta técnica, se pretende ahorrar energía mediante la desactivación de los bloques que no estén siendo utilizados por el manejador de memoria del sistema operativo.

Otra forma de ahorrar energía en sistemas de cómputo móvil se consigue implementando técnicas de manejo de energía sobre los componentes que lo conforman [22]. Generalmente, las técnicas para manejo de energía se enfocan sobre el consumo de energía en un solo componente. Los *discos duros* son uno de los componentes con mas consumo de energía en un sistema de cómputo. Actualmente, la técnica más utilizada para ahorrar energía en el disco consiste en apagar el motor después de un periodo fijo de inactividad. De la misma forma, es posible disminuir el consumo de energía en los discos, interceptando y disminuyendo los accesos innecesarios al disco, con lo cual se reduce significativamente el consumo total de energía del sistema.

Los dispositivos de comunicación inalámbrica aparecen cada vez más en los sistemas portátiles, es por eso que últimamente ha habido un gran auge en las investigaciones en el manejo de energía en estos dispositivos. Comúnmente, estos dispositivos se ponen en modo *inactivo* cuando no es necesario transferir o recibir datos. Existen varias técnicas para reducir el tiempo en el que un dispositivo necesita para transferir o recibir datos. Estas técnicas incluyen protocolos de comunicación que permiten verificar con anticipación cuando los dispositivos no están transmitiendo ningún dato. Adicionalmente, algunos dispositivos de comunicación inalámbrica recientes tienen la capacidad de cambiar dinámicamente su potencia de transmisión.

Las pantallas de vídeo consumen más energía que cualquier otro componente de la computadora, por lo que el manejo de energía en este componente es especialmente importante. Actualmente, los modos de ahorro de energía disponibles incluyen la disminución del brillo,

el apagado de la pantalla (después de un periodo de inactividad), el cambio de color a monocromático y la reducción de la frecuencia de barrido y actualización de la pantalla. Otros componentes sobre los cuales es posible el manejo de energía son el módem y el sistema de sonido.

Las técnicas de manejo de energía implementadas en hardware tienen dos ventajas, (1) son independientes del software y (2) tienen un tiempo de cálculo corto. Por otro lado, el hardware no cuenta con suficiente información acerca de las aplicaciones. Por ejemplo, es difícil para un manejador de energía implementado en hardware predecir los tiempo de ejecución de las tareas en un sistema de tiempo real. Además, si un manejador es implementado directamente en hardware, no hay flexibilidad de realizarle ajustes.

Manejo de energía implementado en software

En los sistemas de manejo de energía tradicionales, el hardware provee el manejo de energía en forma automática, buscando que sea transparente para la aplicación y para el usuario. Esto podría ocasionar algunos problemas al usuario, por ejemplo, provocando que se apague la pantalla durante presentaciones o vídeos (debido a que no se accesa al teclado) ó que se presenten retardos inesperados al accesar el disco duro. Dado que las aplicaciones tienen un conocimiento directo de la forma en que el usuario se encuentra utilizando el sistema, es posible utilizar este conocimiento en al tomar las decisiones sobre el manejo de energía en el sistema.

Unas de las técnicas de manejo de energía a nivel de software es la *transformación de código y algoritmos* [8]. En la mayoría de los compiladores, la función de costo a optimizar es la velocidad de procesamiento o el tamaño del código a ejecutar. Los compiladores convencionales tiene como función principal traducir un código fuente a código objeto. Adicionalmente el compilador intenta producir código eficiente con el menor número de instrucciones posibles implementando la mayor velocidad de procesamiento posible en el código objeto. Por lo cual, un compilador empleado en el manejo de energía tiene que hacer un compromiso entre el tamaño del código producido (su complejidad y eficiencia) y la velocidad para obtener una reducción en el consumo de la energía. Algunos métodos para reducir el número de opera-

ciones incluyen la eliminación de subexpresiones y de código muerto y la optimización de los ciclos de control. La reducción puede obtenerse también al reemplazar operaciones que consumen energía mediante una combinación de operaciones simples, por ejemplo reemplazar multiplicaciones por operaciones de desplazamiento de bits.

En [10] se presentan distintas estrategias de compilación que identifican posibles partes del código en donde es posible realizar escalamiento de frecuencia y voltaje del procesador sin un incremento significativo en el tiempo de ejecución de los programas. La disipación de potencia se estima mediante la suma del número de operaciones dentro de los algoritmos. El resultado de la suma refleja un cambio en la capacitancia, y por lo tanto, afectan el tamaño y la complejidad de los algoritmos.

En general, las técnicas para el manejo de energía en los compiladores tienen las siguientes limitaciones, (1) diferentes aplicaciones podrían requerir al mismo dispositivo en diferentes estados de energía, (2) es complicado diseñar compiladores que manejen la energía y que al mismo tiempo produzcan código eficiente y rápido, y (3) es impráctico re-escribir los programas para añadir características de manejo de energía.

Manejo de energía implementado en el sistema operativo

El mejor lugar para implementar el manejo de energía es en el sistema operativo, ya que estos tienen la flexibilidad de ajustar las políticas de atención a procesos y de manejo de memoria, y proveen de servicios a los programas de aplicación. Los sistemas operativos están estructurados en diferentes capas, como se muestra en la figura 1.3. En la capa del control de procesos o planificador, se cuenta con toda la información referente a las tareas, como son los tiempo de activación, su periodicidad y sus tiempo de ejecución estimados; lo cual ayuda a predecir el comportamiento del sistema.

En la actualidad, existen principalmente dos métodos para el manejo del consumo de energía del procesador a nivel del sistema operativo. El primero método consiste en poner al procesador en modo de *baja potencia*. El modo de baja potencia se activa generalmente cuando el sistema de cómputo se encuentra en estado inactivo después de un largo tiempo. A fin de ahorrar energía, y mientras el procesador se encuentra en estado inactivo, se desha-

bilitan ciertas partes del sistema, dejando habilitadas solo las partes esenciales (tales como el reloj, los circuitos temporizadores y la memoria). A estas técnicas se les conoce como técnicas para *manejo dinámico de energía* o *DPM* (*dynamic power management*) por sus siglas en ingles [3]. Cuando se requiere un alto desempeño, las técnicas *DPM* permiten al hardware consumir más energía, en caso contrario, el hardware entra en un estado de baja potencia. *DPM* es una metodología de diseño que reconfigura dinámicamente un sistema electrónico para proveer niveles de servicio y desempeño con un mínimo número de componentes activos o una mínima carga sobre los componentes. *DPM* utiliza un conjunto de técnicas para ejecutar cálculos de forma eficiente y con el menor consumo de energía. Esto se logra apagando o reduciendo el desempeño de algunos componentes cuando se encuentran inactivos o sin carga de trabajo. La aplicación de las técnicas *DPM* debe darse cuando los sistemas experimenten cargas de trabajo no uniformes durante su operación y cuando sea posible predecir las fluctuaciones de la carga de trabajo con cierto grado de certeza. En la implementación de este tipo de técnicas es necesario una colaboración directa con el hardware [25], ya que este tiene que soportar la programación de interfaces que permitan al sistema operativo cambiar los estados de energía. Una de estas interfaces es el estándar *ACPI (Advanced Configuration and Power Interface)* (ver apéndice A) [14]. El problema con estos métodos, es que su ejecución resulta impredecibles, y por lo tanto poco factible, como para ser utilizado en aplicaciones de tiempo real.

El segundo método para el manejo de energía a nivel de sistema operativo, es conocido como *escalamiento dinámico de voltaje o frecuencia* o DVS (*dynamic voltage scaling*) por sus siglas en inglés. Estos algoritmos reducen el consumo de energía cambiando la velocidad y el voltaje del procesador en forma dinámica (en tiempo de ejecución). Los procesadores utilizados por este método son los llamados *procesadores de velocidad variable* (sección 1.3). En general, los procesadores tienen un mejor desempeño cuando operan a una velocidad alta, sin embargo, esto produce un alto consumo de energía. Dentro de una aplicación de tiempo real, los procesadores de velocidad variable pueden disminuir su velocidad siempre y cuando el desempeño del sistema no se degrade. La reducción en la velocidad de procesamiento,

disminuye el consumo de la energía, pero incrementa los tiempos de cómputo de las tareas, causando posibles pérdidas de plazos de respuesta.

1.3 Procesadores de Velocidad Variable

Con el incremento en la demanda de dispositivos portátiles, nuevos procesadores que consumen menos energía se han introducido al mercado. Estos procesadores consumen menos energía que sus procesadores contemporáneos con las mismas características. Ejemplos de procesadores de velocidad variable son los siguiente:

- *Crusoe - Transmeta* [43]. Esta familia de procesadores ha sido específicamente diseñado para aplicaciones de baja potencia. Su tecnología permite al procesador operar a bajas frecuencias y voltajes de operación (reduciendo el consumo de energía). El modelo TM5400 puede operar desde 200 MHz@1.1V hasta 700MHz@1.6V, con cambios de velocidad de 33MHz. La arquitectura *Crusoe* es una combinación de hardware y software flexible y eficiente que reemplaza millones de transistores por software, manteniendo completa compatibilidad x86.

- *StrongArm - Intel* [12]. La familia de procesadores *StrongArm* ofrece una combinación de alto desempeño y bajo consumo de energía. El modelo SA-1100 soporta 11 niveles de frecuencia de 59 hasta 221 MHz en pasos de 14.7 MHz. La reducción de energía en estos procesadores se obtiene al eliminar bloques funcionales como la unidad de punto flotante, reducir el tamaño de la memoria *cache* y simplificar la complejidad del x86.

- *X-Scale - Intel* [13]. Construido en base a la arquitectura de 0.18 *micras* de *Intel*, la cual permite al procesador operar sobre un amplio rango de velocidades . Los procesadores X-Scale consumen menos de la mitad de energía que sus predecesores (SA-1100). Operan bajo los sistemas operativos de *Microsoft*, *Palm*, *Symbian* y *Linux*. El escalamiento dinámico de frecuencia de estos procesadores varía desde 40 mW/185 MIPS a 150 MHz, 450 mW/750 MIPS a 600 MHz hasta 900 mW/1000 MIPS a 800 MHz.

1.4 Consumo de Energía en Procesadores

Es bien conocido que existen tres fuentes principales en el consumo de potencia en circuitos CMOS [37], los cuales están dadas por:

$$P_{prom} = P_{cc} + P_{fuga} + P_{dinamica} \tag{1.1}$$

El primer término representa la potencia disipada por el corto circuito entre el voltaje de alimentación V y la tierra. El segundo término representa la corriente de fuga, la cual está determinada principalmente por la tecnología utilizada. Estos dos componentes son prácticamente despreciables en circuitos CMOS de baja potencia. El último término representa el consumo dinámico en la potencia, el cual es un factor dominante que representa al menos el 90% de la disipación total de potencia, la cual está dada por [37]:

$$P_{dinamica} = \alpha f C_L V^2 \tag{1.2}$$

donde α es el factor de actividad del circuito o actividad de interrupción, f es la frecuencia del reloj y C_L representa la capacitancia de carga y está dada por la suma de las capacitancias de los transistores. En la ecuación 1.2 se observa que la potencia varía linealmente con la frecuencia y en forma cuadrática con respecto al voltaje. Por lo tanto, ajustando ambos términos podríamos reducir la potencia en forma cúbica, al menos en teoría. Sin embargo, la reducción del voltaje de alimentación implica un decremento en la frecuencia del reloj. La frecuencia máxima para un voltaje de alimentación se obtiene mediante:

$$f_{max} \propto \frac{(V - V_T)^\alpha}{V} \tag{1.3}$$

donde V_T es el voltaje de umbral ($0 < V_T < V$) y α es un factor dependiente de la tecnología ($1 \leq \alpha \leq 2$). Debido a la no linealidad entre la frecuencia y el voltaje, tanto el escalamiento del voltaje como el escalamiento de la frecuencia producirá un ahorro cuadrático de potencia (al menos), y por lo tanto, también de la energía. La tabla 1.1 muestra la relación entre la frecuencia del reloj, el voltaje de alimentación y la potencia consumida en un procesador *Crusoe TM5400*.

Frecuencia (MHz)	70	100	200	300	350	400	500	600	700
Voltaje (V)	0.90	0.95	1.10	1.25	1.33	1.40	1.50	1.60	1.65
Potencia (%)	3.0	4.7	12.7	24.6	32.5	41.1	59.0	80.6	100

Tabla 1.1: Relación ente frecuencia, voltaje y potencia consumida en procesadores Crusoe

El consumo total de energía ϵ en un procesador, operando durante un periodo de tiempo T, es aproximadamente igual a:

$$\epsilon \approx P_{dinamica} * T \tag{1.4}$$

donde $P_{dinamica}$ representa la mayor fuente de disipación de potencia de la ecuación 1.2.

Finalmente, es importante mencionar que los cambios de voltaje del procesador no ocurren de forma instantánea. Por ejemplo, en el microprocesador Crusoe de Transmeta el cambio de nivel de voltage, a un nivel próximo (superior o inferior), se ejecuta con un retraso de 80-100 microsegundos [43].

1.5 Trabajo Relacionado

El presente trabajo incursiona en dos nuevos paradigmas de planificación de tareas en sistemas de tiempo real de reciente estudio. Uno de estos paradigmas es el de *los planificadores de voltaje variable*, conocidos también como algoritmos de *escalamiento dinámico de voltage (DVS)*, los cuales utilizan los procesadores de velocidad variable (sección 1.3).

El segundo paradigma es el llamado *planificador con control retroalimentado*, los cuales difieren de la planificación clásica como EDF o RM en que su configuración es en lazo cerrado, es decir, se utiliza una retroalimentación para controlar algún parámetro del sistema.

1.5.1 Planificadores de Voltaje Variable

Para el manejo de energía en sistemas de tiempo real, un nuevo paradigma en métodos de planificación, conocido como *planificación de voltaje variable* o *escalamiento dinámico de*

voltaje DVS (Dynamic Voltage Scaling), se ha desarrollado en los últimos años. En este tipo de planificación es posible ajustar dinámicamente la velocidad o la frecuencia del procesador, produciendo cambios en el consumo de la energía. El objetivo consiste en minimizar el consumo de la energía en la ejecución de un conjunto de tareas de tiempo real, teniendo en consideración el cumplimiento de las restricciones temporales de dichas tareas.

La planificación de voltaje variable es un problema que consiste en asignar frecuencias o velocidades de ejecución a un conjunto de tareas, y ajustar el voltaje de manera tal que ninguna tarea pierda su plazo de respuesta y que el consumo total de energía sea minimizado. Investigaciones recientes han propuesto diferentes formas para determinar las frecuencias apropiadas a través de algoritmos en línea y fuera de línea. La idea básica de estos algoritmos es disminuir la velocidad de ejecución de las tareas tanto como sea posible sin violar sus plazos de respuesta.

Planificación de voltaje variable fuera de línea

Este tipo de planificación se utiliza en sistemas donde los parámetros temporales de las tareas (tiempo de cómputo, periodo de activación, plazo de respuesta, etc.) no cambian durante la operación del sistema. Los cálculos necesarios para minimizar la energía se realizan antes de la ejecución del sistema, y una vez encontrada la solución, los voltajes de cada tarea nunca se modifican.

En [18] se propone un algoritmo de planificación basada en el esquema cíclico que consiste en dos etapas. La primera es una etapa de pre-procesamiento, que se basa en los tiempo de ejecución máximos del sistema. En este algoritmo se hace una búsqueda combinatoria mediante algoritmos de programación dinámica para encontrar los niveles de voltaje óptimos para el sistema, de forma que se minimice el consumo de energía en el procesador. La segunda etapa de este algoritmo, se lleva a cabo durante la ejecución del sistema y consiste en buscar tiempos ociosos de las tareas, o de la carga en general, para que estos puedan ser asignados a otras tareas y así minimizar el consumo de energía. El trabajo propuesto en [45] describe un algoritmo estático fuera de línea para tareas aperiódicas, no expulsivas y suponiendo tiempo de ejecución máximos. En este algoritmo, las tareas críticas son planificadas a la

velocidad más alta y para las tareas no críticas la asignación de velocidad se resuelve en forma recursiva. El trabajo propuesto por [15] describe un esquema de planificación en donde no existen dependencias entre las tareas (las tareas no comparten recursos), y las tareas son expulsivas. El problema de encontrar los voltajes para cada tarea tal que se minimice el consumo total de energía se resuelve mediante programación lineal. En [11] se contemplan dos fases de planificación. En la primer fase todas las tareas son planificadas a un voltaje *nominal*, en la segunda fase los voltajes de cada tarea se ajustan hasta el punto en que no sea posible obtener un mayor ahorro en el consumo de energía. En [29] se propone un algoritmo de planificación estático en el cual se minimiza el consumo de la energía para el caso en el que las tareas tengan distintos tiempo de arribo, plazos de respuesta y tiempo de ejecución. El objetivo es encontrar el plan de ejecución para un tiempo dado, junto con el voltaje de operación para cada tarea tal que se minimice el consumo de energía. Para la búsqueda se utiliza un algoritmo (de tiempo de ejecución polinomial) basado en multiplicadores de *Lagrange*.

Algunos de los trabajos recientes sobre planificadores de voltaje variable han explorado los intervalos inactivos (ociosos) en la ejecución de las tareas de tiempo real. Esos nuevos algoritmos se han desarrollado en el contexto de las políticas de planificación *Rate Monotonic* y *Earliest Deadline First* [35, 18, 2, 21].

En [33] se propone una técnica capaz de explorar los tiempo ociosos del sistema para reducir la velocidad de las tareas y así reducir el consumo de energía. La novedad de este mecanismo es que las decisiones de minimización de consumo de energía se deciden en tiempo de compilación.

La mayoría de las anteriores investigaciones que trabajan la planificación de voltaje variable, asumen que todas las tareas tienen funciones de consumo similares. En [2, 7] se usa una suposición alternativa, en donde cada tarea de tiempo real tiene diferentes características o factores de consumo de potencia. Una investigación reciente de estos métodos [34], muestra como se comportan los diferentes algoritmos bajo este marco de trabajo.

Planificación de voltaje variable en línea

La planificación en línea se lleva a cabo de forma dinámica (en tiempo de ejecución). El problema en este tipo de planificación consiste en que durante la ejecución de las tareas de tiempo real se deben determinar los tiempos en que el procesador cambiará su velocidad y en que nivel de voltaje, de forma que se minimice el consumo de la energía y no se afecten las restricciones temporales del sistema. Aún en el caso de que se conozcan los parámetros temporales del sistema, el problema consiste en encontrar la mejor combinación de voltaje para cada tarea del sistema, de tal manera que ninguna tarea pierda su plazo de respuesta y que el consumo de energía sea mínimo. La solución a este problema combinatorio es de tipo *NP-Dura (NP-Hard)*, por lo que su solución implica gastar demasiado tiempo de cómputo en enumerar todas las posibles soluciones y encontrar la mejor, o en caso contrario proponer una heurística que permita encontrar una solución aproximada en un tiempo razonable. Si a este problema se le añade el hecho de que los parámetros temporales cambian continuamente, esto implica que se deben buscar soluciones con mucha frecuencia.

El trabajo propuesto en [35] incluye la exploración de los intervalos desocupados del procesador, con el fin de reducir la velocidad del procesador. La reducción del voltaje asignado se consigue en los casos en los que al finalizar la actual tarea en ejecución exista un tiempo desocupado, y antes de que una nueva tarea se active. Al detectarse esta situación, se reduce la velocidad de la actual tarea en ejecución. El trabajo propuesto en [26] minimiza el consumo de energía en sistemas de cómputo embebido que utilizan baterías. Este trabajo estudia el comportamiento dinámico de la descarga de la batería con el fin de maximizar su tiempo de vida. Implementa la planificación de voltaje variable mediante la búsqueda de tiempos ociosos, a fin de asignar estos tiempos a otras tareas y así reducir el consumo de energía. En el trabajo propuesto en [33] se hace un análisis a nivel de código del lenguaje (flujo de código, estados condicionales, ciclos, etc.) para determinar los flujos de ejecución y las instrucciones que son posibles optimizar a fin de minimizar el consumo de la energía. En este trabajo se hace la suposición de que solo una de las tareas es la que ocupa la mayor parte del tiempo de ejecución del sistema.

El trabajo propuesto en [38], se basa en un enfoque de cómputo adaptivo que permite planificar dinámicamente el voltaje en ambientes de tareas en tiempo real con restricciones no críticas (se permite la pérdida de plazos en forma controlada). El enfoque en este trabajo consiste en la observación del comportamiento del sistema. De tal manera, la secuencia de ejecución del sistema es continuamente monitorizada con el fin de obtener distintas estadísticas (tales como los tiempos de ejecución, los tiempos ociosos, los tiempos de acceso a disco a la memoria y a otros dispositivos). Se permite configurar un patrón de ejecución del sistema para predecir su comportamiento y así poder planear en forma dinámica las variaciones de voltaje necesarias. En [27], se propone una metodología de planificación heurística para maximizar el ahorro de energía en procesadores que manejan niveles de voltaje discretos. En esta metodología se propone un servidor aperiódico para la atención de tareas con restricciones de manejo de energía. Dichas tareas arriban al sistema en instantes desconocidos, permanecen en el sistema por un número determinado de instancias, y cuentan con restricciones de tiempo crítico. El servidor está compuesto por un procedimiento de optimización y un algoritmo ambicioso (*greedy*) aproximado, y se ejecuta de forma dinamica (en cada arribo y terminación de cada tarea). El objetivo del servidor es minimizar el consumo de energía en sistemas de tiempo real. Los resultados de las simulaciones en este trabajo de investigación mostraron que esta metodología alcanza un desempeño cercano al óptimo en cuanto a la selección de los niveles de voltaje dentro de un conjunto de tareas en tiempo real.

Dependiendo de la granularidad del escalamiento de voltaje deseado existen dos tipos de planificadores de voltaje variable [34]:

- **IntraDVS:** el cual ajusta el voltaje dentro del límite de una tarea individual.
- **InterDVS:** el cual determina el voltaje tomando en cuenta a varias tareas.

La principal diferencia entre estos algoritmos está en la forma en que se distribuye el tiempo de holgura. El tiempo de holgura se define como la diferencia entre el tiempo de cómputo máximo de la tarea y el tiempo de cómputo actual. Se asume que entre cada instancia de las tareas el tiempo de cómputo actual cambia, pero se mantiene menor al tiempo de cómputo máximo.

Los algoritmos *InterDVS* distribuyen el tiempo de holgura de la tarea actual a las demás tareas del sistema. Esto produce una reducción de las velocidades de las demás tareas del sistema, y consecuentemente se reduce el consumo de la energía del sistema. En [36] se proponen métodos estáticos (fuera de línea) utilizando algoritmos InterDVS mientras que en [36, 1, 17, 31] se proponen métodos dinámicos (en línea).

Los algoritmos *IntraDVS* distribuyen el tiempo de holgura de la tarea actual sobre la misma tarea, con el fin de reducir su consumo de energía. En este algoritmo se asume que una tarea del sistema será la que ocupe la mayor parte del procesamiento del sistema. Por lo cual se tratan de optimizar las secuencias de ejecución de las tareas con el fin de reducir su consumo de energía. Estos algoritmos son de naturaleza estática [34, 33, 19, 6].

1.5.2 Planificadores con Control Retroalimentado

Es conocido que los algoritmos de planificación tradicionales como *Rate Monotonic* (RM) y *Earliest Deadline First* (EDF) están configurados para ejecutarse en *lazo abierto*. La configuración en lazo abierto se refiere al hecho de que una vez planificado el conjunto de tareas éstas no se ajustan a los cambios dinámicos del sistema. Los algoritmos de planificación en lazo abierto pueden desempeñarse adecuadamente en ambientes predecibles (nunca ocurren cambios), en los cuales, la carga de trabajo puede ser modelada con precisión como ocurre en los sistemas de control de procesos tradicionales. Sin embargo, existe un bajo desempeño cuando se ejecutan bajo ambientes impredecibles, en donde los parámetros del sistema cambian durante la ejecución de las tareas. En este tipo de sistemas la carga de trabajo es difícil de modelar o no es posible modelarla con precisión. Los planificadores en lazo abierto como RM o EDF están normalmente basados sobre la suposición de parámetros de carga de trabajo del *peor caso*. Esto significa que los tiempos de cómputo de cada tarea siempre se ejecutan al máximo de su capacidad. Sin embargo, cuando no se conocen con precisión los parámetros temporales de la carga de trabajo o estos no están disponibles, el sistema resultante presenta una baja utilización y un pobre desempeño.

En [40, 24, 23], se presenta un modelo llamado *planificación en tiempo real con control*

retroalimentado (FC-EDF *Feedback Control EDF*) para sistemas de tiempo real adaptables. En este modelo se aplica una metodología basada en la teoría de control para diseñar algoritmos de planificación con control retroalimentado que satisfacen las especificaciones de desempeño en *estado estable y estado transitorio*. También se presenta la arquitectura del modelo, la cual permite utilizar diferentes políticas de planificación y algoritmos heurísticos para la selección de la *calidad de servicio* de las tareas. En este modelo las tareas del sistema cuentan con distintos niveles de calidad de servicio (o distintos tiempos de ejecución). El objetivo de los algoritmos de calidad de servicio es seleccionar un nivel de servicio para cada tarea tal que se maximice la calidad de servicio del sistema.

1.6 Trabajo Propuesto

La figura 1.4 muestra la jerarquía de los trabajos descritos en 1.5.1 y 1.5.2. Cabe hacer notar que los planificadores en general se dividen en estáticos y dinámicos, esto por la naturaleza misma de las políticas de planificación, por ejemplo la política RM se dice que es estática ya que una vez planificado un conjunto fijo de tareas mediante esta política las prioridades asignadas no cambian durante la ejecución. Por otro lado, la política EDF se dice que es dinámica ya que las prioridades varían continuamente, debido a que los plazos de de respuesta cambian conforme transcurre el tiempo.

En la figura 1.4 se muestra la relación de los trabajos más relevantes en los últimos años en las áreas de los planificadores con manejo de energía y de los planificadores con control retroalimentado. En el recuadro con líneas punteadas de la figura 1.4, se indican las áreas en las cuales nuestro trabajo propuesto esta basado.

Por un lado tenemos los planificadores retroalimentados *FC-EDF* [40, 24, 23], de los cuales tomamos los conceptos de control de los sistemas en tiempo real, como son las variables de control y las especificaciones de desempeño. La diferencia principal de este trabajo con el nuestro, es que el planificador *FC-EDF* controla el desempeño del sistema mediante la variación de los tiempos de ejecución de las tareas, utilizando un modelo de tarea de cómputo impreciso (calidad de servicio). Mientras que en nuestro trabajo, el modelo de tareas utilizado

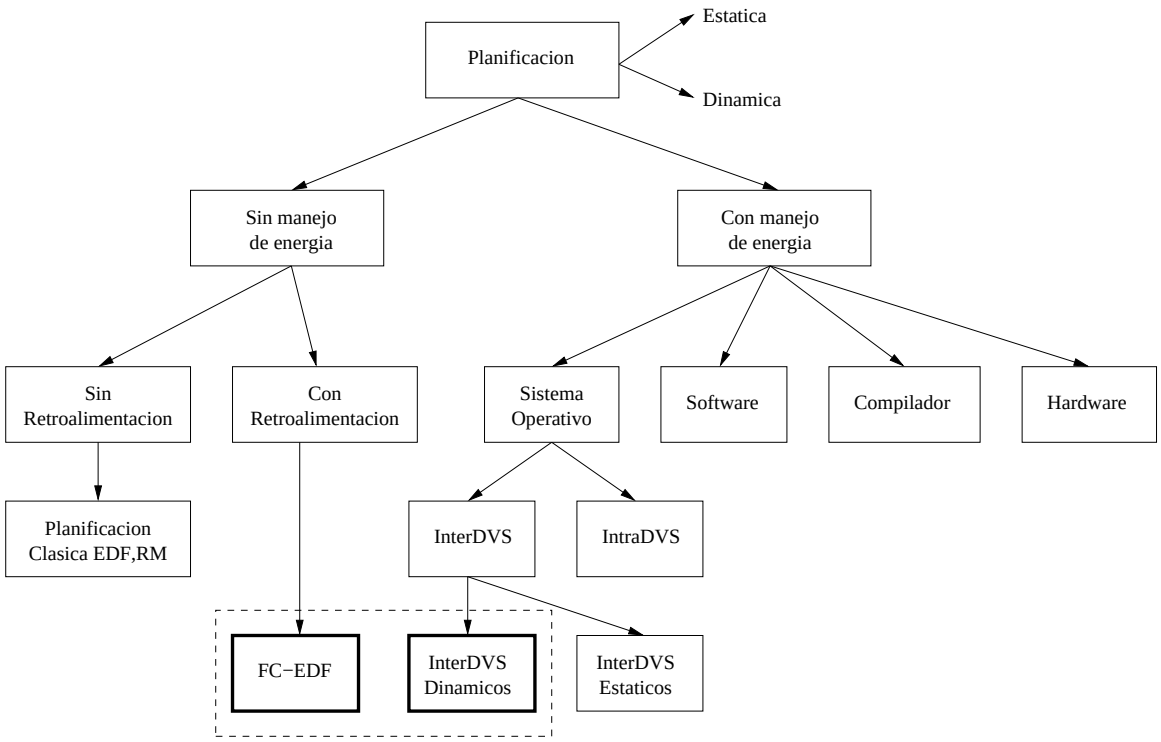


Figura 1.4: Jerarquía de los planificadores en tiempo real

contempla tareas con diferentes velocidades de ejecución y por lo tanto con diferentes tiempos de cómputo.

Por otro lado, tenemos los planificadores con manejo de energía *InterDVS*, los cuales, cambian las velocidades de ejecución de las tareas dependiendo de los tiempos de holgura de dos o más tareas. Nuestro trabajo forma parte de los algoritmos *InterDVS*, ya que cambiamos las velocidades de ejecución de todas las tareas, mediante una distribución de los tiempos de holgura de todas las tareas (a través de la utilización total del sistema), de tal manera que se maximice el ahorro de energía consumida por el procesador, y se minimice la pérdida de plazos de respuesta de las tareas.

Nuestro trabajo es una combinación de los planificadores *FC-EDF* y de los planificadores *InterDVS dinámicos*, ya que empleamos un lazo de control retroalimentado para controlar algunos de los parámetros del sistema (como es la utilización) en tiempo real y así modificar las velocidades ejecución de las tareas.

1.7 Objetivos de la Tesis

Los principales objetivos de la presente tesis son:

- **Integrar mecanismos de planificación de voltaje variable en línea en un sistema operativo de tiempo real para que soporte manejo de energía.** En la sección 1.2 vimos como la mejor técnica para implementar el manejo de energía en un sistema de cómputo es a nivel del sistema operativo. Los objetivos particulares a conseguir en este trabajo de tesis consisten en:

  - *Minimizar el consumo de energía en el procesador,* este ahorro de energía representa un aumento en el tiempo de vida de las baterías en sistemas portátiles o un aumento en el ahorro del consumo de electricidad y mantenimiento de los sistemas de cómputo personal.
  - *Minimizar la pérdida de plazos de respuesta de las tareas,* como el sistema contemplado es un sistema de tiempo real, es necesario garantizar las restricciones temporales del conjunto de tareas del sistema.

- **Integrar algoritmos de control de procesos (controladores PID) al planificador de tareas para adaptar el sistema a cambios en el ambiente.** En los sistemas de tiempo real altamente dinámicos, la carga de trabajo cambia continuamente, por lo que una manera de mantener una estabilización de la carga (utilización del sistema) se puede llevar a cabo utilizando controladores.

- **Desarrollar mecanismos para monitorizar el consumo de energía en procesadores de velocidad variable.** La estabilización de la carga la establece el controlador, para lograrlo es necesario modificar la carga del procesador. Esta estabilización se obtiene cambiando las velocidades de ejecución del conjunto de tareas del sistema. La medición del consumo de energía se obtiene individualmente por tareas, ya que la energía es un producto (multiplicación) de la velocidad y del tiempo durante el cual se ejecuto cada tarea.

- **Integrar algoritmos heurísticos combinatorios para optimizar el consumo de la energía en el procesador.** La asignación de las velocidades de ejecución a las tareas del sistema es un proceso complejo, por lo cual es necesario que los algoritmos implementados encuentren una solución aproximada, cercana a la óptima, en un tiempo de cómputo bajo.

- **Desarrollar una herramienta de simulación en lenguaje C para medir el comportamiento de los algoritmos.** El propósito principal de esta herramienta de desarrollo será implementar los algoritmos de los puntos anteriores y presentar los resultados, como el consumo de energía y plazos de respuesta perdidos, en forma gráfica. Esta herramienta de simulación incluye:

  - La implementación de las políticas de planificación *EDF* y *RM*.
  - La implementación de los algoritmos de control para regular la carga.
  - La implementación de los algoritmos heurísticos combinatorios para optimizar el consumo de energía.
  - La monitorización de los parámetros principales del sistema (consumo de energía, utilización, plazos perdidos).
  - Una interfaz gráfica que permita seleccionar la configuración a simular.
  - La presentación de resultados en archivos, los cuales se utilizan para presentar los resultados en forma gráfica mediante *gnuplot*.

1.8 Organización de la Tesis

En el capítulo 2 se introducen los conceptos básicos sobre los sistemas de tiempo real, se describen las restricciones que se pueden presentar en las tareas y cuales son los parámetros que las caracterizan. También se define el problema de planificación, así como los diferentes tipos en que se clasifican los planificadores, y se enumeran las condiciones asumidas en el

sistema. Se define también el factor de utilización y por último se explican las políticas de planificación *Rate Monotonic* y *Earliest Deadline First* junto con un ejemplo descriptivo.

El capítulo 3 explica la terminología de la teoría de control moderna, utilizada en esta tesis. Se describen los diferentes tipos de controles, sus diferencias, y se explica el objetivo de la retroalimentación y sus efectos en el sistema. Se presentan los diferentes tipos de control con retroalimentación que existen, así como los controladores mas utilizados en la industria, y por último, se describe brevemente la teoría de los sistemas de control digital.

En el capítulo 4 se describe la contribución novedosa de esta tesis que consiste en un algoritmo de planificación retroalimentado con restricciones de energía. Se describe la arquitectura de planificación propuesta, y el modelo de tareas utilizado. En el capitulo 5 se describen los experimentos de simulación llevados a cabo para medir el desempeño de la arquitectura de planificación propuesta para distintas políticas de planificación y las condiciones de la simulación y los resultados obtenidos para las diferentes configuraciones.

En el apéndice A, revisamos el estándar *ACPI* utilizado para el manejo de energía en sistemas de cómputo. Finalmente en el apéndice B, describimos el programa desarrollado para las simulaciones en forma de manual de usuario.

Capítulo 2

Sistemas de Tiempo Real

2.1 Introducción

En este capitulo se definirán conceptos de los sistemas de tiempo real que se utilizarán en esta tesis que servirán como base para los siguientes capítulos.

La entidad de software más importante de cualquier sistema operativo, es *el proceso*. Un proceso o tarea es un cálculo que es ejecutado por el procesado de una forma secuencial. Cuando un único procesador tiene que ejecutar un conjunto de tareas, el procesador tiene que ser asignado a varias tareas de acuerdo a un criterio u orden de ejecución predefinido, llamado política de planificación. El conjunto de reglas que determina el orden en el cual las tareas son ejecutados es llamado algoritmo de planificación.

Una tarea que está lista para ejecutarse, estará en ejecución si está ha sido seleccionada por el planificador, o se encontrará en espera si alguna otra tarea de mayor prioridad se encuentra en ejecución. Una tarea que puede potencialmente ejecutarse en el procesador es llamada tarea activa. Una tarea esperando por el procesador es llamada tarea lista, y todas las tareas esperando por el procesador son mantenidas en una cola de espera llamada cola de listos. El planificador elige el orden de ejecución de las tareas basándose en la política

establecida por el algoritmo de planificación elegido.

Las tareas también pueden estar en estado de espera por tiempo o espera por un recurso físico. En estos estados las tareas solo estarán listas para ejecución hasta que se termine el tiempo de espera o se desocupe el recurso físico por el que están esperando. Las tareas en estado de espera por tiempo se mantienen en una cola que se denomina cola de procesos retrasados por tiempo mientras que las tareas en estado de espera por recurso se mantienen en una cola de procesos esperando por el recurso.

En la mayoría de los sistemas operativos que permiten la activación dinámica de tareas, la tarea en ejecución puede ser interrumpido en cualquier parte de su código, con el fin de que una tarea de mayor prioridad pueda acceder al procesador cuando lo requiera. En este caso, la tarea en ejecución es interrumpida e insertada en la cola de listos, mientras que el procesador es asignado a la tarea con mayor prioridad. La operación de suspender la tarea en ejecución e insertarla dentro de la cola de listos es llamada desalojo.

2.2 Tipos de Restricciones

Las restricciones típicas que pueden ser especificadas sobre las tareas en tiempo real son las siguientes: restricciones de tiempo, restricciones de precedencia y restricciones de exclusión mutua sobre recursos compartidos.

Restricciones en tiempo

Los sistemas en tiempo real están caracterizados por actividades computacionales (tareas o procesos) con restricciones severas de tiempo que deben completarse en orden para alcanzar el comportamiento deseado.

Una restricción en tiempo que presentan las tareas es el plazo de respuesta, el cual representa el tiempo antes del cual una tarea debe completar su ejecución sin causar ningún daño del sistema. Dependiendo de las consecuencias que se presenten por la pérdida de los plazos, las tareas de tiempo real se distinguen normalmente en dos clases:

- **Duras**. Se dice que una tarea es dura si la pérdida del plazo de respuesta puede causar

consecuencias catastróficas dentro del sistema. En este caso, cualquier instancia de las tareas debe garantizar a priori el cumplimiento del plazo de las tareas. Esto se hace considerando el peor caso del tiempo de cómputo de las tareas.

- **Suaves**. Se dice que una tarea es suave si la pérdida de un plazo de respuesta no pone en peligro a personas o causa pérdidas económicas. En los sistemas de tiempo real suaves, la pérdida de plazos produce únicamente una disminución en el desempeño del sistema.

En general, una tarea de tiempo real τ_i puede ser caracterizado por los siguientes parámetros:

- **Tiempo de llegada** B_i: Es el tiempo en el cual una tarea está lista para su ejecución.
- **Tiempo de cómputo en el peor caso** C_i: Es el tiempo de cómputo necesario para ejecutar la tarea sin interrupción.
- **Plazo de respuesta máxima** D_i: Es el tiempo limite antes del cual una tarea debe completarse para evitar daño al sistema.
- **Tiempo de comienzo** s_i: Es el tiempo en el cual una tarea inicia su ejecución.
- **Tiempo de finalización** f_i: Es el tiempo en el cual una tarea termina su ejecución.
- **Criticidad**: Es un parámetro relacionado a la consecuencia de la pérdida de plazos de respuesta, o se relaciona también con la importancia de las tareas en el sistema.
- **Latencia** L_i**:** $L_i = f_i - D_i$, representa el retraso en la terminación de una tarea con respecto a su plazo de respuesta. Si $L_i \leq 0$ la tarea τ_i no pierde su plazo.

Los parámetros temporales son ilustrados en la figura 2.1. Otra característica que puede ser especificada dentro de las tareas en tiempo real concierne la regularidad de su activación. En particular, las tareas pueden ser definidas como periódicas o aperiódicas. Las tareas periódicas consisten de una secuencia infinita de actividades de cómputo, llamadas instancias, que son activadas regularmente a una frecuencia fija. Las tareas aperiódicas se caracterizan

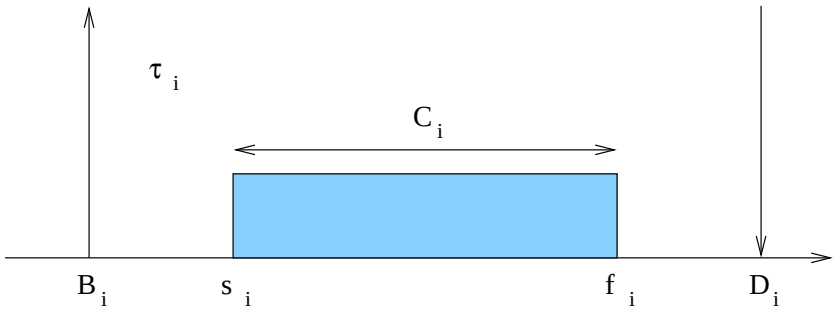


Figura 2.1: Parámetros típicos de una tarea en tiempo real

por consistir de una sola instancia y tener tiempo de arribo desconocido. El tiempo de cómputo de una tarea aperiódica se conoce solo hasta que la tarea arriba.

Restricciones de precedencia

En ciertas aplicaciones, las actividades computacionales no pueden ser ejecutadas en orden arbitrario, si no que tienen que respetar algunas relaciones de precedencia definidas en la etapa de diseño.

Restricciones sobre recursos

Desde el punto de vista de un proceso, un recurso es cualquier dispositivo de entrada/salida o estructura de software que puede ser utilizada por el proceso. Típicamente, un recurso puede ser una estructura de datos, un conjunto de variables, una área de memoria, un archivo, una porción de un programa o un dispositivo periférico. Un recurso que es dedicado exclusivamente a un proceso en particular se dice que es un recurso privado, mientras que un recurso que puede ser utilizado por dos o más tareas es llamado recurso compartido.

Para mantener la consistencia de datos en la mayoría de recursos compartidos, no se permite el acceso simultáneo por dos o más tareas a estos datos, sino que se requiere del cumplimiento de la condición de exclusión mutua entre las tareas que compiten por este recurso. Supongamos que R es un recurso exclusivo compartido por las tareas τ_a y τ_b. Si α es la operación realizada por τ_a sobre R, y β es la operación realizada sobre R por τ_b, entonces α y β nunca deben de ejecutarse al mismo tiempo. El conjunto de lineas de código que se ejecutan bajo restricciones de exclusión mutua se le conoce como sección o región crítica.

Para asegurar el acceso secuencial sobre los recursos compartidos, los sistemas operativos normalmente proveen mecanismos de sincronización (tales como semáforos) que pueden ser usados por las tareas para crear regiones críticas.

2.3 Definición del Problema de Planificación

Para definir un problema de planificación necesitamos especificar los siguientes tres conjuntos [4]: un conjunto de n tareas $\Gamma = \tau_1, \tau_2, \ldots, \tau_n$, un conjunto de m procesadores $\Lambda = p_1, p_2, \ldots, p_m$ y un conjunto recursos $R = r_1, r_2, \ldots, r_s$. Además, es necesario especificar las restricciones de tiempo y de precedencia entre tareas. En este contexto, la planificación permite asignar los procesadores del conjunto Λ y los recursos del conjunto R a tareas del conjunto Γ de acuerdo a un orden que permita completar todas las tareas bajo las restricciones impuestas. Se ha demostrado que este problema es de tipo NP-completo, lo cual implica que su solución es muy difícil de obtener.

Para reducir la complejidad en la construcción de planificadores, podemos simplificar la arquitectura de la computadora, por ejemplo, restringiendo al sistema para que utilice un solo procesador, adoptar un modelo con desalojo, utilizar prioridades fijas, eliminar precedencias o restricciones de recursos, asumir una activación simultanea de tareas y suponer que en el sistema existen conjuntos de tareas homogéneos (únicamente periódicas o únicamente aperiódicas).

2.4 Clasificación de los Algoritmos de Planificación

De entre la gran variedad de algoritmos propuestos para la planificación de tareas en tiempo real, podemos identificar las siguientes clases:

- **Planificación con desalojo**. La tarea en ejecución puede ser interrumpida en cualquier momento para asignar al procesador a otra tarea, de acuerdo a una política de planificación predefinida.

- **Planificación sin desalojo**. Una tarea una vez iniciada, es ejecutada por el procesador hasta que termine su actividad. En este caso, todas las decisiones de planificación son tomadas cuando una tarea inicia o termina su ejecución.

- **Planificación estática**. Los algoritmos estáticos son aquellos en los cuales las decisiones de planificación están basados en parámetros fijos, los cuales son asignados a las tareas antes de su activación.

- **Planificación dinámica**. Los algoritmos dinámicos son aquellos en los cuales las decisiones de planificación están basadas en parámetros dinámicos que pueden cambiar durante la ejecución del sistema.

- **Planificación fuera de línea**. Decimos que un algoritmo de planificación es usado fuera de línea si es ejecutado sobre el conjunto completo de tareas antes de la ejecución actual de las tareas. El planificador generado en esta forma, es almacenado en una tabla y después ejecutado por un despachador.

- **Planificación en línea**. Decimos que un algoritmo de planificación es usado en línea si las decisiones de planificación son tomadas a tiempo de ejecución cada vez que una nueva tarea llega o cuando una tarea termina su ejecución.

- **Planificación óptima**. Se dice que un algoritmo es óptimo si minimiza algunas funciones de costo definidas sobre el conjunto de tareas. Cuando ninguna función de costo es definida y lo único concerniente es alcanzar una planificación factible, entonces se dice que un algoritmo de planificación es óptimo si este encuentra una solución de planificación que no puede ser contradecida por ningún otro algoritmo de planificación. La solución de planificación indica si un conjunto de tareas es factible o no factible (cumple o no con sus plazos de respuesta.

- **Planificación heurística**. Un algoritmo heurístico encuentra una solución aproximada que intenta estar cerca de la solución óptima.

Algoritmos con garantías

En aplicaciones en tiempo real críticas que requieren un comportamiento altamente predecible, el cumplimiento de plazos de respuesta de todas las tareas del sistema debe ser garantizado en forma anticipada, es decir, antes de la ejecución de las tareas. De esta forma, si una tarea crítica no puede ser planificada dentro de su plazo de respuesta, el diseñador del sistema puede cambiar los parámetros temporales de las tareas o optimizar el software a fin de lograr una planificación factible. Para comprobar la viabilidad de la planificación antes de la ejecución de las tareas, el sistema tiene que planear sus acciones asumiendo escenarios en el peor caso.

En sistemas de tiempo real estáticos, donde el conjunto de tareas es fijo y conocido, todas las activaciones pueden ser precalculadas fuera de línea, y la planificación completa puede ser almacenada en una tabla que contiene todas las tareas ordenadas y garantizadas. En este caso, durante la ejecución, el despachador (el cual forma parte del planificador) simplemente sigue el orden de ejecución de las tareas definido en la tabla. La ventaja principal del método estático es que el tiempo de ejecución del algoritmo de planificación no se toma en cuenta. Esto permite la utilización de algoritmos muy complejos para encontrar secuencias de planificación óptimas. Por otro lado, una desventaja de este tipo de planificación es que produce un sistema inflexible a cambios en el ambiente. En sistemas de tiempo real dinámicos, donde nuevas tareas pueden activarse en tiempo de ejecución, la garantía de planificabilidad debe realizarse en línea cada vez que una tarea entra o sale del sistema.

Debido a que los mecanismos de garantía están basados sobre suposiciones en el peor caso, muchas tareas podrían no ejecutarse. Esto implica que la garantía de tareas duras se alcanza con un costo de reducción del desempeño promedio del sistema. Por otro lado, el beneficio de tener un mecanismo de garantía es que las situaciones de sobrecarga pueden ser detectadas anticipadamente y evitar efectos negativos en el sistema. En resumen, una prueba de garantía asegura que una vez que una tarea es aceptada para ejecución en el sistema, ésta completará su ejecución dentro del plazo de respuesta y también, su ejecución no arriesgará la factibilidad de las tareas que previamente habían sido garantizadas.

Algoritmos de mejor esfuerzo

En ciertas aplicaciones en tiempo real, las actividades computacionales tienen restricciones de tiempo suaves que deben de cumplirse siempre a fin de satisfacer los requerimientos del sistema. Sin embargo, en este caso ningún evento catastrófico ocurrirá si una o más tareas pierden sus plazos de respuesta. La única consecuencia asociada con la pérdida de plazos, es la degradación en el desempeño del sistema.

Los algoritmos de planificación de mejor esfuerzo intentan alcanzar los plazos de respuesta, sin embargo no proveen garantía de planificación. Estos algoritmos obtienen un mejor desempeño que los esquemas con garantías. Debido a que la suposiciones pesimistas hechas en la planificación pueden causar el rechazo de tareas, en algoritmos de mejor esfuerzo una tarea es abortada únicamente bajo condiciones reales de sobrecarga.

Algoritmos de cómputo impreciso

Existen aplicaciones en donde se permite una degradación del sistema a cambio del cumplimiento de plazos de todas las tareas. Un modelo de planificación que permite degradar a las tareas a cambio del cumplimiento de plazos es el modelo de cómputo impreciso.

En este tipo de sistemas, cada tarea τ_i se divide en una subtarea obligatoria M_i y una subtarea opcional O_i. La parte obligatoria es la porción del cálculo que debe realizarse para producir un resultado de calidad aceptable, mientras que la parte opcional refina el resultado. En este modelo, una planificación es factible si cada parte obligatoria M_i termina su ejecución dentro del intervalo $[B_i, D_i]$. En una planificación precisa, tanto la parte obligatoria como la opcional concluyen dentro del intervalo $[B_i, Di]$.

2.5 Planificación de Tareas Periódicas

En la mayoría de las aplicaciones de control en tiempo real, las actividades periódicas representan la mayor demanda computacional del sistema. Las tareas periódicas típicamente realizan acciones como la adquisición de datos, lazos de control y monitoreo del sistema. Tales actividades necesitan ejecutarse en forma cíclica a frecuencias fijas, estas frecuencias pueden deducirse de los requerimientos del sistema.

Cuando una aplicación de control consiste de varias tareas periódicas concurrentes con restricciones de tiempo, es necesario que el sistema garantice que cada instancia periódica se active regularmente a su propia frecuencia y se finalice antes de su plazo de respuesta. En general, los plazos de respuesta de las tareas puede ser diferente que a su periodo. Para facilitar la descripción de los métodos de planificación presentados en este capitulo, la siguiente tabla muestra la notación será utilizada:

Parámetro	**Notación**
Γ	Denota un conjunto de tarea periódicas.
τ_i	Denota una tarea periódica i.
τ_i^x	Denota la instancia x de la tarea τ_i.
ϕ_i	Denota la fase de la tarea τ_i, el tiempo de activación de la primera instancia.
D_i	Denota el plazo de respuesta relativo de la tarea τ_i.
P_i	Denota el periodo de activación de la tarea τ_i.
d_i^x	Denota el plazo de respuesta absoluto de la instancia x de la tarea τ_i ($d_i^x = \phi_i + (x-1)P_i + D_i$).
s_i^x	Denota el tiempo de inicio de la instancia x de la tarea τ_i, esto es, el tiempo en el cual inicia su ejecución.
f_i^x	Denota el tiempo de finalización de la instancia x de la tarea τ_i, esto es, el tiempo en el cual completa su ejecución.

Tabla 2.1: Notación utilizada

Para simplificar los análisis de planificabilidad, las siguientes hipótesis son asumidas sobre las tareas:

A1 . Las instancias de una tarea periódica τ_i son activadas regularmente a una frecuencia constante. El intervalo P_i entre dos activaciones consecutivas es llamado periodo de la tarea.

A2 . Todas las instancias de una tarea periódica τ_i tienen el mismo tiempo de ejecución C_i en el peor caso.

A3 . Todas las instancias de una tarea periódica τ_i tienen el mismo plazo de respuesta D_i, el cual es igual al periodo P_i.

A4 . Todas las tareas en Γ son independientes, esto es, no tiene relaciones de precedencia ni restricciones de recursos.

A5 . Ninguna tarea puede suspenderse a si mismo, por ejemplo en operaciones de E/S.

A6 . El tiempo consumida por las operaciones del kernel son despreciables.

Con respecto a las suposiciones A1, A2, A3 y A4; se dice que una tarea periódica τ_i puede ser completamente caracterizado por los siguientes tres parámetros: su fase ϕ_i, su periodo P_i y su tiempo de cómputo en el peor caso C_i. De está manera, un conjunto de tareas periódicas puede denotarse por
$\Gamma = \tau_i(\phi_i, P_i, C_i), i = 1, \ldots, n.$
Una tarea τ_i se dice que es factible si todas sus instancias finalizan dentro de sus plazos de respuesta. Un conjunto de tarea Γ se dice que es planificable si todas las tareas en Γ son factibles.

2.6 Factor de Utilización

Dado un conjunto Γ de n tarea periódicas, el factor de utilización del procesador U es la fracción de tiempo consumido por el procesador en la ejecución del conjunto de tareas. Dado que $\frac{C_i}{P_i}$ es la fracción de tiempo que consume el procesador en ejecutar la tarea τ_i, el factor de utilización para n tareas está dado por:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \tag{2.1}$$

El factor de utilización provee una medición de la carga de trabajo ejecutándose en la UCP (Unidad Central de Procesamiento). A pesar de que la utilización del procesador puede

mejorarse incrementando el tiempo de cómputo de la tarea o decrementando sus periodos, existe un valor máximo de U debajo del cual Γ es planificable y encima del cual Γ no lo es. Tal límite depende del conjunto de tareas y del algoritmo usado para planificar las tareas. $U_{ls}(\Gamma, A)$ es el *límite superior* del factor de utilización del procesador para el conjunto de tareas Γ bajo un algoritmo dado A.

Cuando $U = U_{ls}(\Gamma, A)$, el conjunto Γ se dice que utiliza completamente al procesador. En está situación, Γ es planificable por A, pero un incremento en el tiempo de cómputo en cualquiera de las tareas harán el conjunto no factible. Para un algoritmo A dado, el *límite superior mínimo* $U_{lsm}(A)$ del factor de utilización del procesador es el mínimo factor de utilización sobre todo el conjunto de tareas que utilizan completamente al procesador:

$$U_{lsm}(A) = min_{\Gamma} U_{ls}(\Gamma, A) \tag{2.2}$$

U_{lsm} define una característica importante de un algoritmo de planificación porque permite fácilmente verificar la planificabilidad del conjunto de tareas. De hecho, cualquier conjunto de tareas cuyo factor de utilización este por debajo de esté límite es planificable por el algoritmo. Si el factor de utilización de un conjunto de tareas es mayor que uno, el conjunto de tareas no puede ser planificable por ningún algoritmo.

2.7 Planificación Rate Monotonic

El algoritmo de planificación Rate Monotonic (RM) se basa en una simple regla que asigna las prioridades de las tareas de acuerdo a su frecuencia. Las tareas con periodos más cortos tendrán prioridades más altas. Debido a que los periodos son constantes, RM es una asignación de prioridades fijas o estáticas. Las prioridades son asignadas a las tareas antes de su ejecución y no cambian con el tiempo. Por otra parte, RM es una política de planificación desalojable ya que permite que las tareas en ejecución puedan ser desalojadas por tareas con más alta prioridad.

Liu y Layland [20] demostraron en 1973 que RM es óptimo entre todos los métodos de asignación de prioridades fijas en el sentido que ningún otro algoritmo de prioridades fijas

puede planificar un conjunto de tareas que no puedan ser planificadas por RM.

El límite superior mínimo del factor de utilización bajo el algoritmo de planificación RM para un conjunto de tareas n es:

$$U_{lsm} = n(2^{\frac{1}{n}} - 1) \tag{2.3}$$

Este valor decrece con n, para valores altos de n, el limite superior converge en:

$$U_{lsm} = \ln 2 \approx 0.69 \tag{2.4}$$

2.8 Planificación Earliest Deadline First

El algoritmo de planificación (EDF) es un planificador dinámico que selecciona tareas de acuerdo a sus plazos de respuesta absolutos. Las tareas con plazos más cercanos se ejecutarán a prioridades altas. Debido a que los plazos absolutos de una tarea periódica depende de la instancia actual x dada por:

$$d_i^x = \phi_i + (x-1)P_i + D_i \tag{2.5}$$

EDF es una política de planificación dinámica. La tarea en ejecución es desalojada en el momento en que otra tarea con plazo más cercano se llegará a activar. La planificación EDF no hace una suposición especifica sobre la periodicidad de las tareas, por lo que puede ser usado para planificación de tareas periódicas como aperiódicas.

La planificabilidad de un conjunto de tareas manejadas por EDF puede verificarse a través del factor de utilización del procesador. En este caso, el límite superior mínimo es uno. Un conjunto de tareas periódicas es planificable con EDF si y solo si:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{2.6}$$

2.9 Ejemplo

Considerar el conjunto de tareas $\tau_1(2,5)$ y $\tau_2(4,7)$, mostrado en la figura 2.2. El factor de utilización de estas dos tareas es:

$$U = \frac{C_1}{P_1} + \frac{C_2}{P_2} = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \approx 0.97 \tag{2.7}$$

Esto significa que el 97% del tiempo del procesador es utilizado para ejecutar las tareas aperiódicas, mientras que el procesador permanece ocioso el 3% restante. Como $U > ln2$, la planificación del conjunto de tareas no puede garantizarse bajo RM, mientras que sí está garantizado bajo EDF.

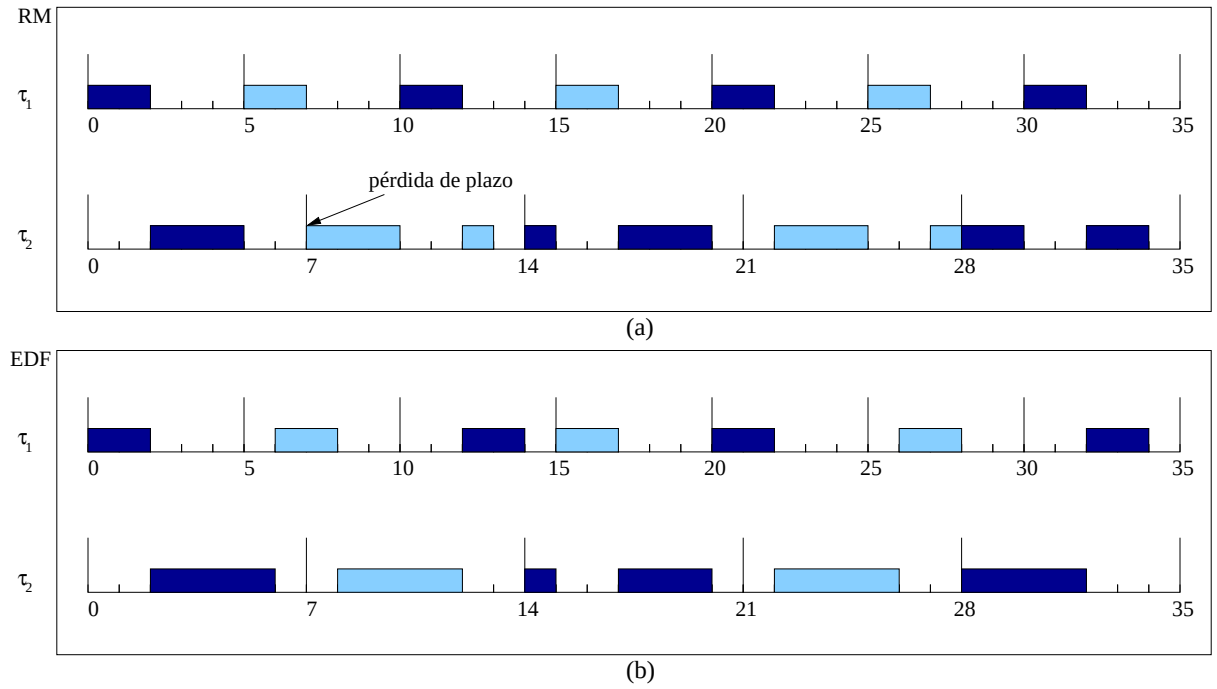


Figura 2.2: Planificación producida por RM (a) y EDF (b) sobre un conjunto de tareas

La figura 2.2 muestra la planificación del conjunto de tareas τ_1 y τ_2 bajo las políticas RM y EDF. Los colores en las tareas indican diferentes instancias de ejecución. En la figura 2.2(a) se observa como como RM produce una pérdida del plazo de la tarea τ_2, en el instante de tiempo $t = 7$, ya que en su primera instancia de ejecución solo pudo ejecutarse por tres unidades de tiempo. Siendo que su tiempo de cómputo es de $C_2 = 4$

Por otro lado, es posible observar en la figura 2.2(b) que utilizando EDF, como política de planificación, permite terminar todas las instancias de las tareas dentro de sus plazos de respuesta, es decir, sin ningún plazo perdido.

Otra diferencia importante entre RM y EDF concierne al número de desalojos que ocurren en la planificación. Bajo RM cada instancia de la tarea τ_2 es desalojada 5 veces, en los tiempos

7,10,15,25, y 30. Bajo EDF, la misma tarea es desalojada únicamente una vez, en el tiempo 15. El pequeño número de desalojos en EDF es una consecuencia directa de la asignación dinámica de prioridades.

2.10 Manejo de Sobrecargas

En aplicaciones del mundo real, aún y cuando el sistema este bien diseñado, una sobrecarga transitoria puede ocurrir por diferentes razones, tales como cambios en el ambiente, llegada simultánea de eventos asíncronos, fallas en los dispositivos periféricos o excepciones en el sistema. El mayor riesgo que puede ocurrir en estas situaciones es que alguna tarea crítica pierda su plazo de respuesta, exponiendo el correcto funcionamiento del sistema.

EDF puede degradar rápidamente su desempeño durante intervalos de sobrecarga. Esto es debido al hecho que EDF da la más alta prioridad a aquellas tareas que estén más cerca de perder sus plazos. Existen casos en los cuales la llegada de una nueva tarea puede causar que todas las tareas pierdan sus plazos, esté es un fenómeno indeseable llamado *efecto Domino*.

Para evitar el efecto domino, el sistema operativo y el algoritmo de planificación deben estar diseñados para manejar sobrecargas transitorias en una forma controlada, de manera que el daño debido a la pérdida de plazos pueda minimizarse.

En un ambiente de tiempo real duro, un sistema está sobrecargado cuando, basado en suposiciones del peor caso, no existe planificación factible para el conjunto total de tareas, por lo cual una o más tareas perderán sus plazos de respuesta.

2.11 Resumen

En este capitulo se describen los conceptos más importantes de los sistemas de tiempo real, los cuales son utilizados en el resto de la tesis. En este capitulo se enumeraron los diferentes tipo de restricciones que pueden especificarse sobre las tareas de tiempo real, de las cuales la más importante para este estudio son las restricciones de tiempo , esto es, si las tareas son duras o suaves.

Se definió también el problema de planificación, así como la clasificación de los algoritmos de planificación. Se describió el concepto de factor de utilización ampliamente utilizado en esta tesis, y se describieron los algoritmos de planificación mas utilizados en los sistemas de tiempo real, el EDF (*Earliest Deadline First*) y el RM (*Rate Monotonic*), con un ejemplo donde se explica la planificación de estos dos algoritmos sobre un conjunto de dos tareas.

El sistema desarrollado en esta tesis, contempla la implementación de las políticas de planificación EDF y RM. Mediante simulación observaremos el comportamiento del sistema propuesto con las dos políticas de planificación implementadas, y bajo diferentes configuraciones. En el capitulo 4 describiremos la arquitectura del sistema propuesto.

Capítulo 3

Sistemas Automáticos de Control

3.1 Introducción

En este capitulo introduciremos los conceptos de control utilizados en el desarrollo del trabajo. Vimos en el capitulo 1, que el sistema de tiempo real propuesto tiene como objetivos principales el ahorro en el consumo de la energía en el procesador mientras minimizamos la pérdida de plazos perdidos de las tareas, para ello proponemos una arquitectura del sistema en lazo cerrado en donde haremos uso de un controlador ampliamente utilizado en la industria del control de procesos. El sistema propuesto en lazo cerrado, es retroalimentado en forma muestreada por variables de control, en este trabajo, las variables de control son la utilización, la razón de plazos perdidos o ambos, estas variables tienen una relación directa con el sistema de tiempo real. El medio de actuación se logra utilizando *procesadores de velocidad variable* los cuales varían la carga del procesador mediante el cambio en las velocidades de ejecución de cada una de las tareas del sistema, una vez que el controlador realice su trabajo el planificador calendarizará la ejecución de las tareas, dependiendo de la política de planificación utilizada (RM o EDF). El uso del controlador está justificado debido a que los tiempo de ejecución de las tareas varían de instancia a instancia y la carga de trabajo o

número de tareas no es constante.

En este capitulo describiremos los sistemas de control en lazo abierto y lazo cerrado, también se mostrará porque es útil el uso de la *retroalimentación* en los sistemas de control y cuales son sus efectos sobre el sistema controlado. Se describen algunos conceptos básicos de la teoría de control y por último revisaremos los diferentes tipos de retroalimentación así como sus características, las ecuaciones que lo describen tanto en el espacio continuo como en el discreto y veremos la importancia de la selección del periodo de muestreo en el desempeño de un controlador digital.

3.2 Introducción al Control

Los sistemas de control automático son sistemas dinámicos y un conocimiento de la teoría de control proporcionará una base para entender el comportamiento de tales sistemas. Estos sistemas emplean frecuentemente componentes de diferentes tipos, por ejemplo, componentes mecánicos, eléctricos, hidráulicos, neumáticos y combinaciones de estos; por lo tanto, al trabajar con controladores es necesario estar familiarizado con las leyes fundamentales que rigen a estos componentes. En este capítulo comprenderemos los siguientes temas:

1. ¿Que es un sistema de control?
2. ¿Por qué son importantes los sistema de control?
3. ¿Cuáles son los componentes básicos de un sistema de control?
4. ¿Por qué incorporar retroalimentación en los sistemas de control?
5. ¿Cuáles son los diferentes tipos de retroalientación?
6. ¿Como se implementa un controlador en forma digital?

Con respecto a la dos primeras preguntas, podemos citar al ser humano, que es quizás el sistema de control más sofisticado y completo que existe. Un ser humano promedio es capaz de llevar a cabo una gran diversidad de tareas. Algunas de estas tareas, como la recolección

de objetos o una caminata, suelen ser labores rutinarias. Bajo ciertas circunstancias, éstas tareas deben llevarse a cabo en forma óptima. Por ejemplo, un atleta que corre los 100 m planos tiene por objetivo recorrer esta distancia en el menor tiempo posible. Un corredor de maratón no sólo debe recorrer la distancia con la mayor rapidez posible, sino que además, para lograrlo, debe controlar el consumo de energía y obtener un resultado óptimo. Por consiguiente, se puede decir en forma general que la vida impone el logro de muchos objetivos, y los medios para alcanzarlos casi siempre dependen de algún sistema de control.

En años recientes, los sistemas de control han venido adquiriendo un papel muy importante en el desarrollo y avance de la civilización y tecnología modernas. Casi todos los aspectos de nuestras actividades cotidianas son afectados por algún tipo de sistema de control. Por ejemplo, en el campo doméstico, los controles automáticos para calefacción y aire acondicionado regula la temperatura y la humedad de los hogares y edificios para lograr una vida cómoda. Para alcanzar una eficiencia máxima en el consumo de energía, mucho sistemas modernos de calefacción y de aire acondicionado están computarizados, en especial en los grandes edificios y las fábricas.

Los sistema de control son muy comunes en todos los sectores industriales, desde el control de calidad de productos industriales, líneas de ensamble automático, control de máquinas herramientas, tecnología espacial, armamento, control por computadora, sistema de transportación, robótica y muchos otros. Incluso problemas como el control de inventarios y los sistemas de control sociales y económicos, pueden resolverse con enfoques de la teoría del control.

Terminología Utilizada

Para comprender los sistemas de control automáticos, es necesario definir los siguientes términos los sistemas de control[30]:

Planta. Una planta es un equipo o quizás simplemente un juego de piezas de una máquina funcionando juntas, cuyo objetivo es realizar una operación determinada.

Sistema. Un sistema es una combinación de componentes que actúan conjuntamente y cumplen determinado objetivo. Un sistema no está limitado a los objetos físico. El

concepto de sistema puede ser aplicado a fenómenos abstractos y dinámicos, como son los sistemas de tiempo real.

Perturbaciones. Una perturbación es una señal que tiende a afectar adversamente el valor de la salida de un sistema. Un ejemplo de perturbaciones en los sistema de tiempo real es la variación en los tiempo de ejecución de las tareas con respecto a los tiempo estimados o del peor caso.

Sistema de control retroalimentado. Es aquel que tiende a mantener una relación preestablecida entre la salida y la entrada de referencia, comparando ambas y utilizando la diferencia como parámetro de control.

Sistema de control automático. Es un sistema de control retroalimentado en el que la entrada de referencia o la salida deseada son constantes o varían lentamente en el tiempo, y donde la tarea fundamental consiste en mantener la salida en el valor deseado a pesar de las perturbaciones presentes.

Cualquiera que sea el tipo de sistema de control considerado, los ingredientes básicos del sistema pueden describirse en término de:

1. Objetivos del control.
2. Componentes del sistema de control.
3. Resultados.

En la figura 3.1(a) se ilustra la relación entre estos tres ingredientes básicos en forma de diagrama de bloques. Estos tres ingredientes básicos pueden identificarse como entradas (referencias), componentes del sistema (controlador y planta) y resultados (salidas), respectivamente, como se muestra en la figura 3.1(b).

En general, el objetivo del sistema de control consiste en controlar las salidas y de una manera predeterminada, la cual está dada por la referencia r y el controlador. A la entradas del sistema se le llama también consigna de operación y a las salidas variables controladas.

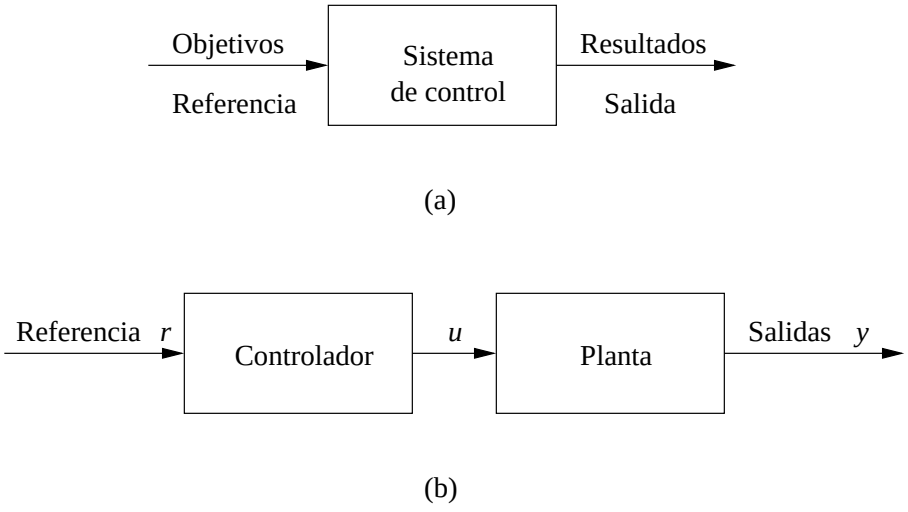


Figura 3.1: Componentes básicos de un sistema de control

Como ejemplo simple del sistema de control descrito en la figura 3.1, consideremos el sistema direccional de un automóvil. La dirección de las dos ruedas frontales puede considerarse como la variable controlada y o salida; la dirección del volante es la señal de control u o entrada a la planta. La planta o proceso en este caso está constituido por los mecanismos de la dirección y la dinámica de la totalidad del automóvil. Sin embargo, si el objetivo consiste en controlar la velocidad del vehículo, entonces, el grado de precisión ejercida sobre el pedal del acelerador es la señal de control u y la velocidad lograda es la velocidad controlada y. En su conjunto, podemos considerar al sistema de control del automóvil como constituido por dos entradas (volante y acelerador) y dos salidas (dirección y velocidad). En este caso, los dos controles y salidas son independientes entre sí; pero en general, existen sistemas en los que los controles están acoplados. A los sistemas con más de una entrada y una salida se le llama sistemas multivariables.

Otro ejemplo de un sistema de control, es el control del motor de un automóvil con un régimen de marcha en reposo. El objetivo de este tipo de sistema de control consiste en mantener la marcha en reposo del motor a un valor relativamente bajo (para economía del combustible) cualquiera que sea la carga aplicada al motor (por ejemplo, transmisión, dirección hidráulica, aire acondicionado, etc.). Sin contar con el control de marchan en reposo,

cualquier aplicación repentina de una carga al motor causaría una caída de la velocidad del mismo y podría provocar que se parara. De esta manera, los objetivos principales del sistema de control con marchan reposo son (1) eliminar o reducir al mínimo la caída de velocidad del motor cuando se le aplica una carga y (2) mantener la marcha en reposo en el valor deseado. La figura 3.2 muestra el diagrama de bloques del sistema de control de marcha en reposo desde el punto de vista de entradas-sistema-salida. En este caso, el ángulo del obturador de la gasolina α es la entrada, el par de carga T_L representa una perturbación externa debido a la aplicación del aire acondicionado, dirección hidráulica, trasmisión, frenos, etc. Las revoluciones del motor ω son la salida. y el motor es el proceso o sistema controlado.

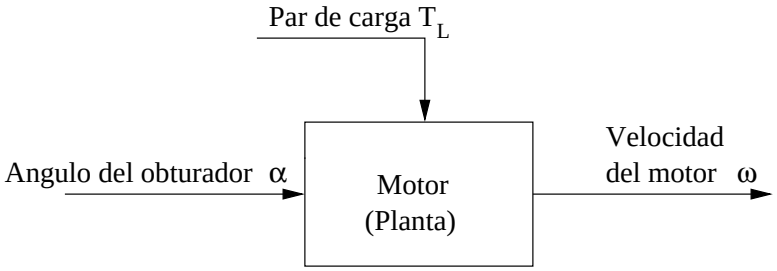


Figura 3.2: Sistema de control de marcha en reposo

3.2.1 Sistemas de Control en Lazo Abierto

El sistema de control de marchan reposo que se ilustra en la figura 3.2 es poco sofisticado y se clasifica como *sistema de control en lazo abierto*. Resulta fácil apreciar que dicho sistema, no cumpliría en forma satisfactoria con los requerimientos de desempeño deseado. Por ejemplo, si el ángulo del obturador α se fija un cierto valor inicial que corresponda a una determinada velocidad del motor, al aplicar una carga de par T_L, no hay manera de evitar una caída de la velocidad del motor. La única forma en que podrá operar el sistema será contando con los medios para ajustar α en respuesta a un cambio de T_L, para mantener ω en el nivel deseado. Debido a la simplicidad y economía de los sistema de control de lazo abierto, estos se usan en la práctica en muchas situaciones. De hecho, casi todos los automóviles fabricados antes

de 1981 no contaban con un sistema de control de marcha en reposo.

Otro ejemplo de sistemas en lazo abierto son las lavadoras eléctricas, pues en su diseño típico el ciclo de lavado queda determinado en su totalidad por la estimación y el criterio del operador humano. En una lavadora eléctrica verdaderamente automática contaría con los medios para comprobar el grado de limpieza de la ropa en forma continua y suspendería la operación por sí misma al alcanzar el grado de lavado deseado. Los elementos del sistema

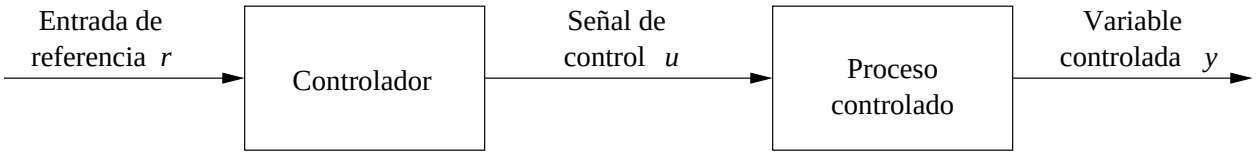


Figura 3.3: Elementos de un sistema de control en lazo abierto

de control en lazo abierto casi siempre pueden dividirse en dos partes: el controlador y del proceso controlado, tal como lo ilustra el diagrama de bloques de la figura 3.3. Se aplica una señal de entrada o comando r al controlador, cuya salida actúa como señal de control u; la señal actuante controla el proceso controlado, de tal manera que la variable controlada y se comporte de acuerdo con estándares predeterminados.

3.2.2 Sistema de Control en Lazo Cerrado

En los sistema de control en lazo abierto, el elemento faltante para lograr un control más preciso es un enlace o retroalimentación de la salida a la entrada del sistema. Para obtener un control más preciso, la señal controlada $y(t)$ debe retroalimentarse y compararse con la entrada de referencia, tras lo cual se envía a través del sistema una señal de control proporcional a la diferencia entre la entrada y la salida, con el objetivo de corregir el error. A los sistemas con uno o más lazo de retroalimentación de este tipo se le llama *sistema en lazo cerrado*. En la figura 3.4 se muestra el diagrama de bloques de un sistema de control en marcha en reposo en lazo cerrado. La entrada de referencia ω_r fija la velocidad deseada. Comúnmente, cuando el par de carga es cero, la velocidad del motor en reposo debe concordar

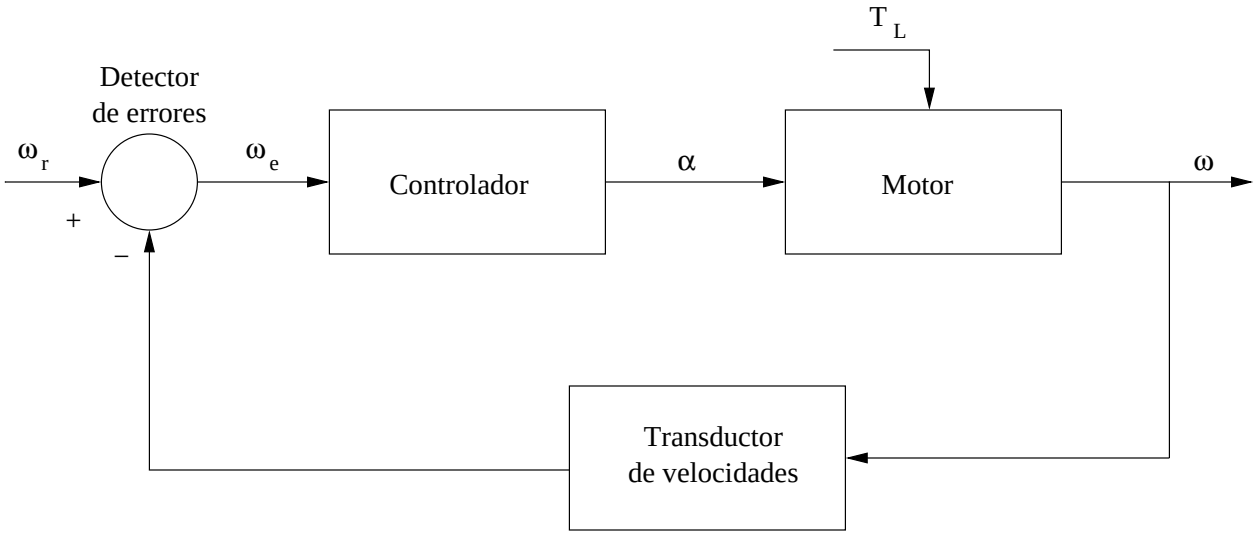


Figura 3.4: Sistema de control de marcha en reposo en lazo cerrado

con el valor de referencia ω_r, y cualquier diferencia entre la velocidad real y el valor deseado, causado por cualquier perturbación del par de carga T_L, es detectada por el transductor de velocidad y el detector de errores, con lo que el controlador operará sobre esta diferencia y proporcionará una señal para ajustar el ángulo del obturador α que corrija el error.

3.3 Retroalimentación y sus Efectos

En el ejemplo presentado en la sección anterior, el uso de la retroalimentación tiene el propósito de reducir el error entre la entrada de referencia y la salida del sistema. Sin embargo, la importancia de los efectos de la retroalimentación en los sistemas de control es mucho más importante. La reducción del error del sistema es solamente uno de los diferentes efectos importantes que la retroalimentación tiene en un sistema. En las siguientes secciones se mostrará que la retroalimentación también tiene efectos en las características de desempeño del sistema tales como estabilidad, ganancia total, sensibilidad y reducción de ruido.

En general, se puede afirmar que, cuando las variables de un sistema exhiben una secuencia cerrada de *relaciones de causa y efecto*, el sistema cuenta con una retroalimentación. Pode-

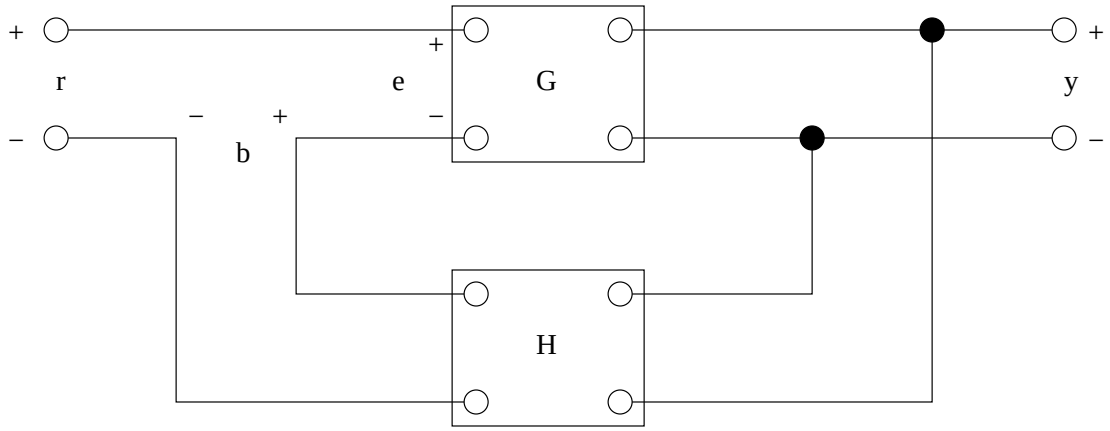


Figura 3.5: Sistema de retroalimentación

mos investigar los efectos de la retroalimentación sobre diversos aspectos del desempeño de un sistema. En este punto, sin contar con los conocimientos generales necesarios y sin haber estudiado las bases matemáticas de la teoría de los sistemas lineales, sólo podemos aplicar una notación estática simple para la discusión. Consideremos la configuración del sistema de retroalimentación simple que se muestra en la figura 3.5, donde r es la señal de entrada, y es la señal de salida, e es el error y b es la señal de retroalimentación. Los parámetros G y H pueden considerarse como ganancias constantes. Mediante operaciones algebraicas simples, se puede demostrar que la relación entrada-salida de un sistema es:

$$M = \frac{y}{r} = \frac{G}{1 + GH} \tag{3.1}$$

Esta relación básica de la estructura del sistema retroalimentado nos permitirá conocer más a fondo los efectos importantes de la retroalimentación.

Efecto de la retroalimentación sobre la ganancia total

Como puede apreciarse en la ecuación 3.1, la retroalimentación afecta a la ganancia G de un sistema sin retroalimentación por un factor de $1 + GH$. La referencia de la retroalimentación en el sistema de la figura 3.5 es negativa, pues a la señal de retroalimentación se le asigna un signo menos. La cantidad GH puede incluir en sí misma un signo negativo, por lo que el efecto general de la retroalimentación es que puede incrementar o reducir la ganancia. En un sistema de control práctico, G y H son funciones de frecuencia, por lo que la magnitud de

$1 + GH$ puede ser mayor que 1 en un intervalo de frecuencia pero inferior a 1 en otros. Por consiguiente, la retroalimentación puede aumentar la ganancia del sistema en un intervalo de frecuencia y disminuirlo en otro.

Efecto de la retroalimentación sobre la estabilidad

La estabilidad es un concepto que describe si un sistema será capaz de seguir una entrada. Se dice que un sistema es inestable cuando su salida está fuera de control o aumenta sin límites. Para comprender los efectos de la retroalimentación sobre la estabilidad, podemos referirnos nuevamente a la expresión de la ecuación 3.1. Cuando $GH = -1$, la salida del sistema es infinita para cualquier entrada finita. Se puede decir que la retroalimentación puede causar inestabilidad en un sistema originalmente estable.

Efecto de la retroalimentación sobre la sensibilidad

Las consideraciones de sensibilidad suelen tener un papel importante en el diseño de sistemas de control. Puesto que todos los elementos tienen propiedades que varían con el medio ambiente y su tiempo de uso, no siempre es posible considerar que los parámetros de un sistema pueden ser totalmente estacionarios en el intervalo total de la vida operacional del sistema. En general, un buen sistema de control debe ser insensible a estas variaciones de los parámetros y seguir siendo capaz de producir una respuesta adecuada.

Efecto de la retroalimentación sobre el ruido

Todos los sistemas de control están sometidos a señales extrañas o ruidos durante su operación. Ejemplos de estas señales son las variaciones en los tiempo de ejecución de las tareas en un sistema de tiempo real, el voltaje de ruido térmico en los amplificadores electrónicos y el ruido de escobillas o conmutadores en los motores eléctricos. El efecto de la retroalimentación sobre el ruido depende en gran parte del punto de introducción del ruido al sistema, en muchas situaciones, la retroalimentación puede reducir el efecto del ruido sobre el desempeño del sistema.

3.4 Tipos de Retroalimentación

En los controles automáticos industriales son muy comunes los seis tipos de acción básica de control [30], [5]: de dos posiciones (todo o nada), proporcional, integral, proporcional integral, proporcional derivativo y proporcional integral derivativo. Es importante comprender las características básicas de las diversas acciones para elegir la más adecuada para determinada aplicación, en este caso vamos a descartar el control de dos posiciones ya que no lo utilizaremos en nuestro estudio.

Retroalimentación proporcional

Para un control de acción proporcional, la relación entre la salida del controlador $u(t)$ y la señal de error $e(t)$ es:

$$u(t) = K_p e(t) \tag{3.2}$$

o en magnitudes de transformada de Laplace,

$$\frac{U(s)}{E(s)} = K_p \tag{3.3}$$

donde K_p se conoce como sensibilidad proporcional o ganancia. El control proporcional esencialmente es un amplificador con ganancia ajustable. Valores muy grandes de K_p a menudo pueden llevar a la inestabilidad. Para la mayoría de los sistemas hay un límite superior en la ganancia proporcional para lograr una respuesta bien amortiguada y estable.

Retroalimentación integral

Es un control con acción integral, el valor de la salida del controlador $u(t)$ varía proporcionalmente a la señal de error actuante $e(t)$. Esto es:

$$\frac{du(t)}{dt} = K_i e(t) \tag{3.4}$$

o

$$u(t) = K_i \int_0^t e(t)dt \tag{3.5}$$

donde K_i es una constante regulable. La función de transferencia del control integral es:

$$\frac{U(s)}{E(s)} = \frac{K_i}{s} \tag{3.6}$$

Si se duplica el valor de $e(t)$, el valor de $u(t)$ varía dos veces más rápido. Para un error actuante igual a cero, el valor de $u(t)$ se mantiene estacionario. La acción de control integral recibe a veces el nombre de control de reposición.

Retroalimentación proporcional e integral PI

La acción de control proporcional e integral queda definida por la siguiente ecuación:

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(t)dt \tag{3.7}$$

o la función de transferencia del control es:

$$\frac{M(s)}{E(s)} = K_p \left(1 + \frac{1}{T_i s}\right) \tag{3.8}$$

donde K_p representa la sensibilidad proporcional o ganancia y T_i el tiempo integral o tiempo de restablecimiento. Tanto K_p como T_i son regulables. El tiempo integral regula la acción de control integral, una modificación en K_p afecta tanto la parte integral como la proporcional de la acción de control. A la inversa del tiempo integral T_i se le llama velocidad de restablecimiento. Esta retroalimentación tiene la virtud de poder proporcionar un valor finito de $u(t)$ sin señal de error de entrada $e(t)$. Esto ocurre porque $u(t)$ es una función de los valores pasados de $e(t)$ más que del valor actual, esto es, los errores pasados del integrador actuaran aunque el error se haga cero. Esta característica significa que las perturbaciones se pueden acomodar con error cero porque ya no es necesario que $e(t)$ sea finito para producir un control que cancele la perturbación. La motivación principal para añadir la acción integral es el reducir o eliminar los errores en estado estable, pero esta ventaja se consigue a costa de una estabilidad reducida.

Retroalimentación proporcional y derivativo PD

La acción de control proporcional y derivativo queda definida por la siguiente ecuación:

$$u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt} \tag{3.9}$$

y la función de transferencia es:

$$\frac{U(s)}{E(s)} = K_p \left(1 + T_d s\right) \tag{3.10}$$

donde K_p es la sensibilidad proporcional y T_d es el tiempo derivativo, los dos parámetros son regulables. La acción de control derivativa es conocida también como control de velocidad debido a que el valor del control es proporcional a la velocidad de variación de la señal de error actuante. El tiempo derivativo T_d es el intervalo de tiempo en el que la acción de velocidad se adelanta al efecto de acción proporcional. La acción de control derivativa tiene la ventaja de ser anticipatoria, pero tiene la desventaja de que amplifica las señales de ruido y puede producir efecto de saturación en el actuador, se utiliza junto a la proporcional para aumentar la amortiguación y para incrementar la estabilidad del sistema. La acción derivativa por sí misma no lleva el error a cero.

Retroalimentación proporcional integral derivativa PID

Representa la combinación de las acciones proporcional, integral y derivativa y tiene las ventajas de cada una de las tres acciones de control individuales. Normalmente se le conoce por sus siglas *PID*. La ecuación de un control con esta acción de control combinada está dada por:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \tag{3.11}$$

y la función de transferencia es:

$$\frac{U(s)}{E(s)} = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \tag{3.12}$$

donde K_p representa la sensibilidad proporcional, T_d el tiempo derivativo y T_i el tiempo integral. Esta combinación es a menudo utilizada para proveer un grado aceptable de reducción del error simultáneamente a una estabilidad y amortiguación aceptables. Normalmente, los controladores disponibles comercialmente tienen está forma, y el ingeniero de control únicamente tiene que ajustar o sintonizar las tres contantes de la ecuación 3.11 para obtener un comportamiento aceptable.

3.5 Implementación Digital del PID

Los controladores fueron implementados originalmente usando técnicas analógicas. En la actualidad es una practica común implementar controladores *PID* mediante microprocesado-

res, para lo cual deben tomarse en cuenta algunas consideraciones al implementarlo en forma digital, las consideraciones más importantes tienen que ver con el muestreo, la discretización y la selección del periodo de muestreo.

Muestreo

Cuando una computadora digital es utilizada para implementar una ley de control, todo el procesamiento de señales es realizada en instancias discretas en tiempo. La secuencia de operaciones es:

1. Esperar la interrupción del reloj
2. Leer la entrada analógica
3. Calcular la entrada de control
4. Escribir la salida analógica
5. Actualizar las variables del controlador
6. Ir al paso 1

Las acciones de control están basadas sobre los valores de la salida en tiempo discretos. Este proceso es llamado *muestreo*. El caso normal es muestrear la señal periódicamente con periodo T. Un punto importante en la implementación digital es la selección de este periodo de muestreo (sección 3.5).

Discretización

Para implementar una ley de control continua en el tiempo como el controlador *PID* (ecuaciones 3.11 y 3.12) sobre una computadora digital, es necesario aproximar las partes derivativas e integral que aparecen en la ecuación 3.11. Existen diferentes maneras de hacer esto [41]:
El término proporcional u_p está dado por la ecuación 3.2 donde $e(t)$ es la diferencia entre la referencia r y la variable controlada y, por lo que la implementación digital se obtiene reemplazando las variables continuas con sus respectivas versiones muestreadas:

$$u_p(kT) = K_p[r(kT) - y(kT)] \tag{3.13}$$

donde kT es el instante de muestreo y $r(kT) - y(kT) = e(kT)$ es el error.
El término integral está dado por la ecuación 3.4, aproximando la derivada por una diferencia obtenemos

$$\frac{u_i[(k+1)T] - u_i(kT)}{h} = K_i e(kT) \tag{3.14}$$

lo que nos resulta

$$u_i[(k+1)T] = u_i(kT) + K_i h e(kT) \tag{3.15}$$

Existen diferentes métodos de aproximar la acción derivativa, como *forward differences*, *backward differences* y *Tustin's approximation*, pero todas dan como resultado la misma forma:

$$u_d(kT) = a_i u_d[(k-1)T] + b_i[y(kT) - y[(k-1)T]] \tag{3.16}$$

solo que con diferentes valores de a_i y b_i. Estos algoritmos son conocidos como algoritmos de posición porque dan la salida del controlador directamente. En implementaciones digitales se utiliza una forma incremental. Esta forma se obtiene calculando las diferencias de la salida del regulador u y añadiendo los incrementos de cada una de las acciones de control:

$$\Delta u(kT) = u(kT) - u[(k-1)T] = \Delta u_p(kT) + \Delta u_i(kT) + \Delta u_d(kT) \tag{3.17}$$

los incrementos de la parte proporcional e integral son fácilmente calculadas de las ecuaciones 3.13 y 3.15

$$\Delta u_p(kT) = u_p(kT) - u_p[(k-1)T] = K_p[r(kT) - y(kT) - r[(k-1)T] + y[(k-1)T]] \tag{3.18}$$

$$\Delta u_i(kT) = u_i(kT) - u_i[(k-1)T] = K_i h e[(k-1)T] \tag{3.19}$$

vimos como la parte derivativa puede calcularse de varias maneras utilizando diferentes métodos de aproximación, utilizaremos la versión unificada 3.16 para calcular la parte derivativa incremental

$$\Delta u_d(kT) = u_d(kT) - u_d[(k-1)T] = \frac{b_i}{1 - a_i}[y(kT) - 2y[(k-1)T] + y[(k-2)T]] \tag{3.20}$$

Frecuentemente la acción derivativa no es utilizada. Una observación interesante es que muchos controladores industriales únicamente cuentan con acción *PI*. Este tipo de control es

adecuado para los procesos donde las dinámicas del proceso son esencialmente de primer orden, es fácil determinar si este es el caso, por ejemplo si la respuesta escalón se comporta como un sistema de primer orden, entonces es suficiente un controlador *PI*.

Selección del periodo de muestreo

La selección del periodo de muestreo T es un problema fundamental en los sistemas muestreados [42]. El periodo de muestreo seleccionado depende de las propiedades de la señal, el método de reconstrucción y el propósito del sistema. En un problema de procesamiento de señales, el propósito es simplemente obtener una señal digital y después recuperarla desde sus muestras, por lo tanto, un criterio razonable para la selección del periodo de muestreo puede ser el tamaño del error entre la señal original y la señal reconstruida.

La fundamentación principal para la selección de la frecuencia de muestreo w_s o el periodo de muestreo T es el teorema de muestreo de *Shannon* [9]. Este teorema especifica que una frecuencia de muestreo debe ser al menos el doble de la frecuencia más alta de la señal. El ancho de banda de las computadoras afecta la precisión de los valores obtenidos debido a los convertidores A/D y D/A, los coeficientes del controlador y las operaciones aritméticas de la computadora. Otras consideraciones que afectan la precisión incluyen los compensadores, las señales de perturbaciones, las mediciones del ruido, el tiempo de retardo inherente en el sistema, etc. En esencia *la influencia principal sobre el periodo de muestreo es al ancho de banda característica de estas señales.* Tomando la frecuencia más alta del sistema, aplicamos el teorema de Shannon para determinar la frecuencia de muestreo. En la mayoría de las implementaciones de controladores digitales ésta frecuencia se incrementa por un factor de 5 a 10, o tal vez hasta 20 veces. Esta heurística ayuda a resolver las dificultades implícitas en el análisis no lineal de las operaciones de cómputo y las restricciones del ancho de palabra finito. Las simulaciones pueden utilizarse para intentar disminuir la frecuencia de muestreo para una aplicación especifica. *La determinación de una frecuencia de muestreo óptima es un compromiso entre varios factores*, por ejemplo, el costo, la flexibilidad y la precisión. Factores mas específicos que dictan la frecuencia de muestreo son:

- La respuesta deseada del sistema a una entrada de control dada

- La respuesta deseada a perturbaciones externas
- La sensibilidad de los parámetros sobre el modelo de la planta o sistema
- La aplicación del control (sistema de tiempo real)
- Las técnicas de diseño empleadas en el control
- Las capacidades de la computadora (memoria, velocidad de procesamiento, ancho de palabra, etc.)

3.6 Resumen

En este capitulo se presentaron los conceptos de control que se utilizaran en el desarrollo del sistema de tiempo real retroalimentado con restricciones de energía. Se describieron los diferentes tipos de retroalimentación (proporcional, integral, proporcional + integral PI, proporcional + derivativa PD, proporcional + integral + derivativa PID).

La implementación de un controlador digital requiere de la selección de la ley de control discretizada y la correcta selección del periodo de muestreo. Esta última depende de varios factores pero principalmente depende de las capacidades de la computadora, la aplicación a controlar, la respuesta deseada del sistema así como a la respuesta a perturbaciones externas.

Un sistema de tiempo real es complejo, de segundo o mayor orden, para propósitos de control podemos suponer a los sistemas de tiempo real como sistemas de primer orden sin pérdida de generalidad, para este tipo de sistemas vimos como un controlador PI es suficiente para obtener buenas respuestas y una estabilización aceptable.

Para el sistema propuesto utilizaremos la implementación discreta de la ley de control dada por la ecuación 3.17 en el lazo de control retroalimentado.

Capítulo 4

Planificador de Voltaje Variable con Retroalimentación

4.1 Introducción y Motivación

La administración de energía se ha convertido en un factor importante en el diseño de sistemas portátiles. El ahorro en el consumo de la energía es importante en dispositivos como *laptops*, teléfonos celulares, *PDA's* y en sistemas de cómputo embebido en general; debido a que se extiende el tiempo de uso de las baterías sobre las que operan. La necesidad de diseñar sistemas con bajo consumo de energía no es exclusivo de los sistemas de cómputo portátiles, ya que el ahorro de energía se ha vuelto una restricción en el diseño de prácticamente cualquier sistema de cómputo, incluyendo las computadoras de escritorio, servidores, ruteadores, sistemas de entretenimiento, etc.

El problema de la reducción en el consumo de la energía ha sido estudiado en la últimos diez años desde diferentes enfoques, por ejemplo mediante la introducción de componentes y dispositivos que consumen menos potencia, nuevas técnicas en el diseño *VLSI/IC* para la operación de dispositivos y nuevas técnicas de compilación (sección 1.2).

Recientemente, los *planificadores de voltaje variable* han emergido como una alternativa para la administración de la energía en los sistemas operativos en tiempo real. En este tipo de planificadores, el voltaje y la frecuencia se ajustan dinámicamente mediante el uso de *procesadores de velocidad variable*[43, 12, 13]. Como se discutió anteriormente, el consumo de energía en una tarea depende en forma lineal del tiempo de cómputo de la tarea y en forma cuadrática del voltaje/velocidad asignado a la tarea (ecuaciones 1.2 y 1.4), por lo que reduciendo el voltaje/velocidad asignado a las tareas, el sistema consumirá menos energía, con la consecuencia de que las tareas tomarán más tiempo para completar su ejecución.

Los algoritmos de planificación de tareas en tiempo real como *RM* y *EDF*, vistos en la sección 2.7 y 2.8, son capaces de manejar conjuntos de tareas con característica sofisticadas, tales como, arribo de tareas aperiódicas, restricciones de precedencia, recursos compartidos, o ejecución en ambientes distribuidos. Sin embargo, se conoce que la ejecución de estos algoritmos de planificación se realiza en *lazo abierto*. El término lazo abierto se refiere al hecho de que una vez planificado el conjunto de tareas, los parámetros de éstas no pueden ser "ajustados" al presentarse variaciones en la carga. Los algoritmos de planificación en lazo abierto se desempeñan eficientemente en ambientes predecibles, en los cuales la carga de trabajo no cambia y puede ser modelada con precisión (por ejemplo los sistemas estáticos). Sin embargo, estos sistemas observan un desempeño pobre cuando se ejecutan en ambientes impredecibles en donde la carga de trabajo cambia frecuentemente y no puede ser modelada con precisión. Además, los planificadores en lazo abierto son diseñados en base a parámetros de carga del *peor caso* (sin considerar que una tarea puede variar sus tiempo de cómputo entre las distintas instancias de su ejecución), como resultado se obtiene un sistema con bajo desempeño y baja utilización, debido a las estimaciones pesimistas hechas sobre la carga de trabajo.

El objetivo principal de este trabajo es el desarrollo de un mecanismo en línea para la planificación de voltaje variable en un sistema operativo de tiempo real. Para lograr este objetivo, se desarrolló un planificador configurado en lazo cerrado, que incorpora un lazo de retroalimentación y una acción de control que permitirán al sistema regular (ajustar) la

carga del procesador en forma dinámica. Este planificador es capaz de manejar cargas de trabajo dinámicas y de aceptar tareas (periódicas y aperiódicas) que arriben al sistema en forma impredecible y que cuenten con tiempos de cómputo variables.

4.2 Modelo del Sistema

Consideremos un conjunto $\Gamma = \{\tau_1, \ldots, \tau_n\}$ de n tareas periódicas de tiempo real ejecutándose en un solo procesador. Las tareas son independientes (no comparten recursos) y no tienen restricciones de precedencia. El tiempo de arribo B_i de la tarea τ_i es desconocido. El tiempo de vida de la tarea τ_i consiste de un número fijo de instancias de ejecución r_i. Después de la ejecución de la instancia r_i, la tarea termina su ejecución.

Supondremos que la velocidad del procesador puede cambiar en niveles discretos entre una velocidad mínima V_{min}, la cual corresponde al voltaje mínimo necesario para mantener al sistema funcionando, y una velocidad máxima V_{max}. La velocidad V_{ij} corresponde a la velocidad de ejecución de una instancia de la tarea τ_i cuando se ejecuta al nivel de velocidad j, donde $V_{min} \leq V_{ij} \leq V_{max}$. El consumo de potencia de una tarea τ_i está dada por la función de velocidad $g_i(V_{ij})$. Si la tarea τ_i ocupa al procesador durante el intervalo de tiempo $[t_1, t_2]$, entonces la energía consumida durante este intervalo es $\epsilon(t_1, t_2) = \int_{t_1}^{t_2} g_i(V_{ij})dt$ (ecuaciones 1.2 y 1.4).

Adicionalmente haremos las siguientes suposiciones:

- El tiempo asociado con los cambios de voltaje es despreciable.
- El cambio en los niveles de voltaje se realizan en forma discreta.
- La velocidad permanece constante durante la ejecución de cada instancia de las tareas, hasta que se ejecute el próximo ciclo de retroalimentación.
- Las tareas pueden tener diferentes consumos de potencia.

Se define a N_i como el conjunto de velocidades de ejecución para cada tarea τ_i. El tamaño del conjunto N_i depende del número de velocidades discretas que soporta el procesador, (ver

sección 1.4). Cada nivel de velocidad $j \in N_i$ por cada tarea τ_i, representa un *ahorro de energía*, el cual se calcula mediante la siguiente ecuación:

$$S_{ij} = (\epsilon_{i1} - \epsilon_{ij}) \tag{4.1}$$

donde ϵ_{i1} representa la energía consumida por la tarea τ_i ejecutándose a su máxima velocidad y ϵ_{ij} representa la energía consumida por τ_i ejecutándose a velocidad j. Además, cada tarea ejecutándose a una velocidad V_{ij}, tendrá una utilización:

$$Ue_{ij} = \frac{C_{ij}}{P_i} \tag{4.2}$$

en donde C_{ij} es el tiempo de ejecución estimado de una sola instancia y P_i es el periodo de activación de τ_i. La utilización Ue_{ij} indica la fracción de tiempo de procesador ocupado por la tarea τ_i cuando se ejecuta a la velocidad j. Suponemos que los elementos $j \in N_i$ para todas las tareas están definidas en orden creciente $\{S_{i1} < S_{i2} < \ldots < S_{iv_{min}}\}$ o $\{Ue_{i1} < Ue_{i2} < \ldots < Ue_{iv_{min}}\}$, donde v_{min} representa la velocidad mínima a la cual se puede ejecutar la tarea τ_i, S_{i1} y Ue_{i1} son los elementos con valores más pequeños en N_i (con menor valor de ahorro de energía). Cada tarea τ_i en el sistema acumula un ahorro de energía S_i^k cuando ejecuta un número de instancias durante el intervalo de tiempo entre $[(k-1)W, kW]$, donde k es el instante de tiempo muestreado y W es el periodo de muestreo. El término S^k representa la cantidad de ahorro de energía acumulada por todas las tareas dentro del sistema durante el intervalo de tiempo $[(k-1)W, kW]$, esto es, $S^k = \sum_{i=1}^{n} S_i^k$.

La tabla 4.1 muestra los parámetros de las tareas del sistema utilizados en este trabajo. De acuerdo a la notación descrita en la tabla 4.1, la utilización estimada Ue se calcula mediante los tiempos de ejecución estimados y los periodos de activación dados por el diseñador del sistema antes de la ejecución. Estos parámetros son estimados fuera de línea en base a mediciones del *peor caso* de los tiempos de ejecución.

La utilización medida $Um(k)$ representa la suma de los tiempos de cómputo actuales de las instancias ejecutadas sobre un intervalo de tiempo especifico $[(k-1)W,\ kW]$, esto es $Um(k) = \frac{\sum_{i=1}^{n} A_{ij}}{W}$, donde W es el *periodo de muestreo* y k representa el instante de muestreo $(k = 1, 2, 3, \ldots)$.

Parámetro	**Notación**
P_i	Periodo de invocación de τ_i
D_i	Plazo de respuesta máximo de τ_i, en este caso $D_i = P_i$
C_{ij}	Tiempo de ejecución estimado de τ_i ejecutándose a la velocidad j
A_{ij}	Tiempo de ejecución actual de τ_i ejecutándose a la velocidad j, A_{ij} varía de instancia a instancia y es desconocida por el planificador
B_i	Tiempo de arribo de tareas periódicas
W	Periodo de muestreo
Ue_{ij}	Utilización estimada de τ_i ejecutándose a la velocidad j, $Ue_{ij} = \frac{C_{ij}}{P_i}$
Ue	Utilización estimada del procesador, $Ue = \sum_{i=1}^{n} Ue_{ij}$
Um	Utilización medida del procesador, $\frac{\sum_{i=1}^{n} A_{ij}}{W}$ sobre una ventana de tiempo W, incluye solamente algunas instancias por tarea

Tabla 4.1: Parámetros de las tareas del sistema

Se considera que durante el intervalo de tiempo W, las tareas pueden ejecutarse por varias instancias. La utilización Um, es una medida real del tiempo ocupado por el procesador ya que considera las variaciones de los tiempos de cómputo de las tareas entre distintas instancias, durante los periodos de muestreo.

La figura 4.1 muestra los parámetros temporales de dos tareas τ_1 y τ_2 sobre un periodo de muestreo W, los tiempo de arribo están representados por $B_1 = 40ms$ y $B_2 = 80ms$. Se observa como los tiempos de ejecución estimados $C_1 = 50ms$ y $C_2 = 60ms$ no cambian entre instancias de ejecución, mientras que los tiempo de ejecución actual A_1 y A_2 si cambian entre instancias. Estos cambios se deben a la impredecibilidad que se obtiene al ejecutar diferentes partes del programa (if,switch-case,for) en cada instancia de ejecución. Tenemos por ejemplo, para la tarea τ_1 sus tiempo para la primera y segunda instancia son de $30ms$ y $40ms$, y para la tarea τ_2, los tiempo varían desde $30ms$ hasta $50ms$ en las dos primeras instancias.

Los periodos de activación de las tareas τ_1 y τ_2 son $P_1 = 70ms$ y $P_2 = 80ms$, los plazos respuesta son iguales a su periodo de activación ($D_i = P_i$).

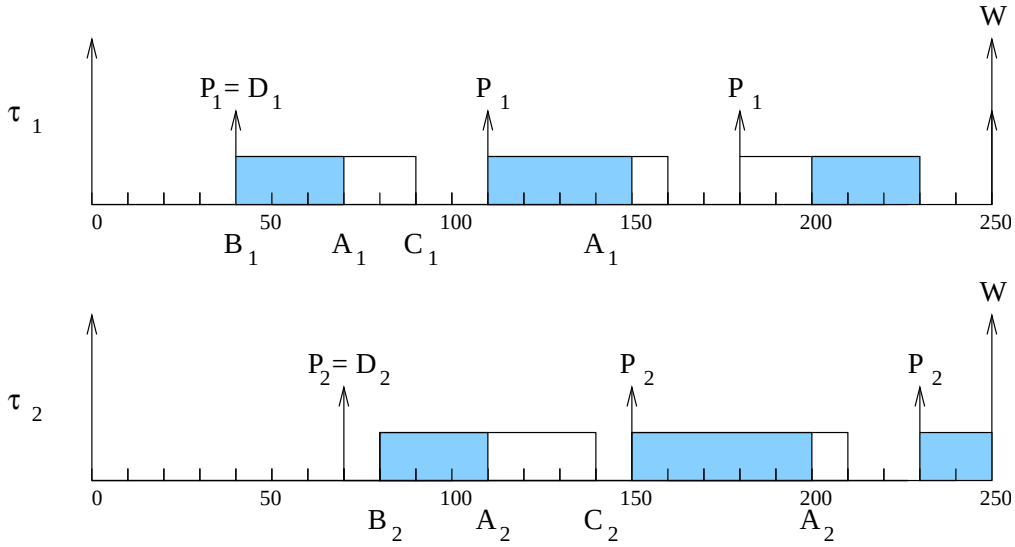


Figura 4.1: Parámetros temporales de las tareas

En nuestra implementación, el planificador será capaz de monitorizar el sistema y adaptarse a cambios en la carga de trabajo de las tareas, cada cierto tiempo, el cual está representado mediante el periodo de muestreo W, en el caso de la figura 4.1, $W = 250ms$.

Un punto clave en este modelo, es que caracteriza sistemas con ambientes impredecibles, en donde la utilización actual de cada tarea varía con el tiempo y es desconocida al planificador. En tales sistemas, una alternativa es el uso de lazos de control retroalimentados en la planificación.

4.3 Descripción del Problema

El problema a solucionar consiste en planificar un conjunto de tareas de tiempo real con restricciones en el consumo de energía del procesador, con las siguientes características:

- Las tareas del sistema arriban al sistema el el tiempo B_i en forma impredecible.
- Las tareas permanecen en el sistema por un tiempo especifico y después de este tiempo terminan su ejecución.

- Durante la ejecución, las tareas del sistema varían su tiempo de cómputo A_i en cada instancia de su ejecución. Solo se conoce por anticipado su tiempo de ejecución estimado C_i (peor caso).

Por lo tanto, la carga de trabajo del sistema estará variando debido a los nuevos arribos (y terminación) de las tareas y a los cambios continuos de los tiempos de cómputo de las tareas. Esta impredecibilidad podría introducir sobrecargas en el sistema, y por lo tanto comprometer la planificabilidad de las tareas de tiempo real.

El planificador diseñado para manejar este tipo de sistemas de tiempo real, debe ser capaz de adaptarse a la carga del sistema de tal forma que se cumplan los objetivos de *maximizar el consumo de energía y minimizar el número de plazos perdidos en el sistema.* Adicionalmente, se espera que el planificador resuelva el problema de optimización planteado, mediante algoritmos de bajo costo computacional, ya que el problema deberá ser solucionado con una alta frecuencia.

Específicamente, el objetivo del problema de optimización es encontrar un nivel de velocidad $j \in N_i$ para cada tarea τ_i tal que la suma del ahorro de energía para todas las tareas sea maximizada y sin que la utilización medida total del conjunto de tareas exceda la capacidad del sistema cs. Esto es:

Maximizar :

$$Z = \sum_{i=1}^{n} \sum_{j \in N_i} S_{ij} x_{ij} \tag{4.3}$$

Sujeto a:

$$\sum_{i=1}^{n} \sum_{j \in N_i} Ue_{ij} x_{ij} \leq cs \tag{4.4}$$

$$\sum_{j \in N_i} x_{ij} = 1,\ i = 1, \ldots, n$$

$$x_{ij} = \begin{cases} 1 & \text{Si se selecciona una velocidad } V_{ij}\ (j \in N_i) \text{ para la tarea } \tau_i \\ 0 & \text{En caso contrario,} \end{cases}$$

donde cs representa la utilización mínima bajo la cual la planificación del conjunto de n tareas es factible, esto es, la utilización bajo la cual las tareas no pierden ningún plazo de respuesta. El valor de cs depende de la política de planificación utilizada, por lo que toma el

valor de 1 para la política EDF y toma el valor de $cs = n(2^{\frac{1}{n}} - 1)$ ($cs \approx 0.69$ cuando $n \to \infty$) para la política RM [20].

La condición del objetivo del problema de optimización (4.4), indica que el sistema no presenta pérdida de plazos de respuesta en ninguna de las tareas en tiempo real. Sin embargo, en nuestro modelo de tareas del sistema, se asume que la carga de trabajo constantemente sobrecarga al procesador, con lo cual, no será siempre posible garantizar que no se pierdan plazos de respuesta de las tareas. Esta situación se debe al arribo constante de nuevas tareas al sistema, y a que los parámetros temporales de las tareas varían entre cada instancia de ejecución.

Por lo anterior, el objetivo del problema de optimización tiene que replantearse a fin de que se maximice el ahorro de energía del sistema y que se minimice el número de plazos perdidos de las tareas (ver sección 4.7). La solución a este problema de optimización se calculará cada vez que termine el periodo de muestreo W, por lo cual, se pretende que los algoritmos que permitan solucionar este problema se ejecuten en un bajo tiempo de cómputo.

A continuación, describiremos la solución de planificación desarrollada, para el problema planteado.

4.4 Arquitectura del Sistema

A fin de cumplir con los objetivos de optimización antes planteados, para un sistema de tiempo real con restricciones de energía, proponemos un planificador que sea capaz de ejecutarse en lazo cerrado. La figura 4.2 muestra la arquitectura general del sistema en lazo cerrado propuesto para resolver el problema de optimización. A continuación se presentan las distintas etapas de procesamiento que experimentan las tareas en el sistema bajo la arquitectura del sistema propuesto.

1. **Arribo de Tareas y Mecanismo de Aceptación.**

   El sistema propuesto, presenta como entrada, tareas de tiempo real que arriban al sistema en tiempos desconocidos. Una vez que las tareas arriban al sistema, el mecanismo

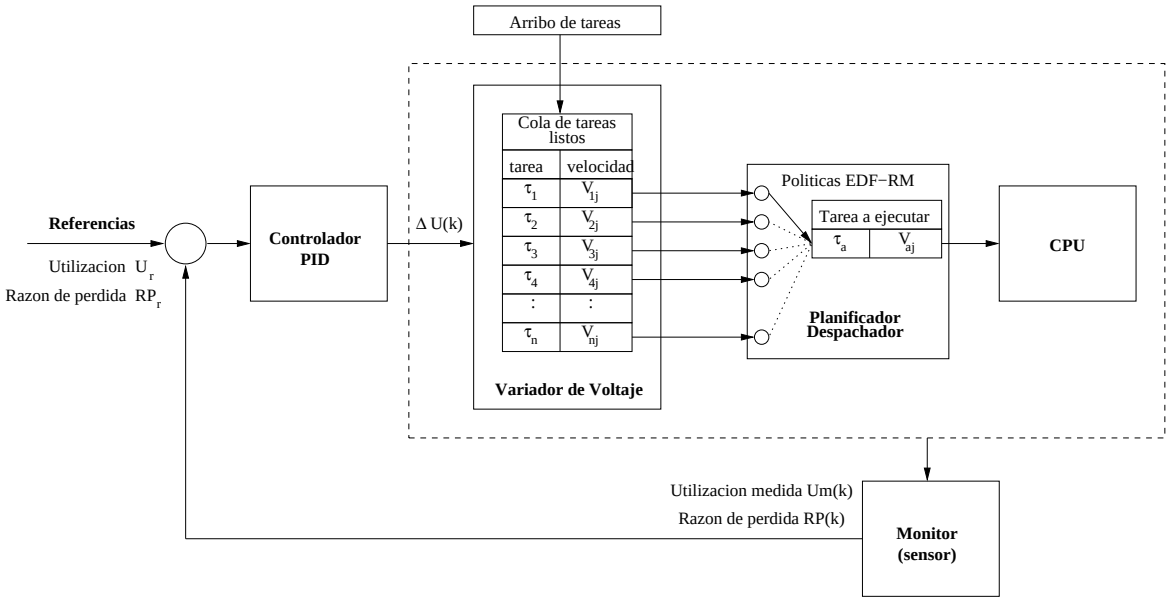


Figura 4.2: Arquitectura general del sistema

de aceptación se encarga de verificar si éstas pueden ser aceptadas, tal que no perjudiquen a los plazos de respuesta de las tareas ya aceptadas. El mecanismo de aceptación está basado en la siguiente condición:

Si las tareas en el sistema ya se encuentran ejecutando a la máxima velocidad del procesador, y la utilización medida del sistema Um se encuentra en su máximo nivel ($Um = 1$), no será posible aceptar nuevas tareas. En caso de no cumplirse esta condición, las tareas serán aceptadas en el sistema.

Cabe hacer notar que la utilización Um cambia constantemente debido a que los parámetros del sistema varían durante la ejecución. Además, debido a que las tareas del sistema se ejecutan solo por un número fijo de instancias, la carga del sistema se decrementara al ocurrir una terminación de alguna tarea. De cualquier forma, dada la alta variabilidad en la carga del sistema y a que el planificador solo es capaz de adaptarse a estos cambios solo hasta que ocurra el periodo de muestreo W, es posible que existan sobrecargas y que algunas tareas pierdan plazos de respuesta durante este periodo de tiempo.

2. **Cola de Tareas Listos.**

   Las tareas que han sido aceptadas para ejecución serán enviadas a una cola de tareas listas para ejecución. El planificador del sistema operativo selecciona a la tarea con mayor prioridad de esta cola para ponerla en ejecución, siguiendo una política de planificación (EDF ó RM).

3. **Lazo de Control Retroalimentado**

   La arquitectura del *planificador retroalimentado con restricciones de energía* (figura 4.2) utiliza un lazo de control, el cual es invocado a cada instante de muestreo k, y está compuesto de un *monitor* (sensor), un *controlador PID* y un *actuador* representado por el *variador de voltaje*.

   3.1. **Monitor de Carga.**

   El *monitor de carga* se encarga de verificar los datos referentes a la carga del procesador como son la utilización medida $Um(k)$ y la razón de pérdida de plazos $RP(k)$ en el instante de muestreo k, las cuales son utilizadas por el controlador PID para determinar el error. Este error representa la diferencia entre la *referencia de desempeño* y la *variable a controlar*.

   La utilización $Um(k)$ representa la utilización medida del sistema descrita en la tabla 4.1, en el instante de muestro k, y la razón de pérdida de plazos $RP(k)$ representa el número de plazos perdidos por periodo de muestreo. El monitor de carga transfiere estos datos al *Controlador PID*.

   3.2. **Controlador PID.**

   El controlador PID recibe los datos del monitor de carga, y se encarga de *calcular* el monto de la carga de trabajo del sistema que es necesario ajustar para que se cumplan los objetivos de optimización. El Controlador PID compara las referencias con las variables a controlar para obtener el error, y calcula la cantidad necesaria a cambiar en la utilización total estimada Ue; esta cantidad es llamada *utilización requerida* y está representada por $\Delta U(k)$. El cambio en la utilización total estimada es: $Ue(k+1) =$

$Ue(k) + \Delta U(k)$. El controlador utiliza una función de control PID, la cual calcula el valor de la variable manipulada Ue y compensa las variaciones en la carga, manteniendo las variables a controlar ($Um(k)$ ó $RP(k)$) cercanas a sus referencias de desempeño.

El *controlador PID* le transfiere al *variador de voltaje* el ajuste de carga $\Delta U(k)$ necesario para obtener el desempeño deseado en el sistema.

El controlador PID utilizado en nuestro sistema, soporta las siguientes variables de control:

- **Variables a Controlar**. Son las variables controladas por el planificador-controlador para obtener el desempeño deseado del sistema. Las variables a controlar en un sistema de tiempo real durante el instante de muestreo k son la *razón o promedio de plazos perdidos* $RP(k)$ y la *utilización del procesador medida* $Um(k)$. Ambas variables están definidas sobre una ventana de tiempo $[(k-1)W, kW]$, donde k es el *instante de muestreo* y W es el ancho de la ventana de muestreo.
  - La razón de pérdida $RP(k)$ en el instante de muestreo k está definida como el número de plazos perdidos dividido por el número total de instancias que hayan terminado o abortado su ejecución, dentro de la ventana de muestreo $[(k-1)W, kW]$
  - La utilización $Um(k)$ en el instante de muestreo k es el porcentaje del tiempo ocupado por el procesador dentro de la ventana de muestreo $[(k-1)W, kW]$, esto es $\frac{\sum_{i=1}^{n} A_{ij}}{W}$.
- **Referencias de desempeño**. Representan el desempeño deseado del sistema en términos de la variable a controlar. La *razón de pérdida deseada* está representada por RP_r y la *utilización deseada* está representada por U_r. Como ejemplo, un sistema en particular pudiera requerir una razón de pérdida $RP_r = 1\%$ y una utilización del $U_r = 90\%$. La diferencia entre una referencia y el valor actual de la correspondiente variable a controlar es conocida como *error*. El error de la razón de pérdida sería $E_{RP} = RP_r - RP(k)$ y el error de la utilización $E_U = U_r - Um(k)$

.

- **Variable Manipulada**. Son parámetros del sistema que el controlador puede cambiar dinámicamente y los cuales afectan los valores de las variables a controlar. En este sistema, la variable manipulada es la *utilización total estimada* $Ue(k)$. Al Comparar $Ue(k)$ con la variable de control $Um(k)$, $Um(k)$ normalmente se incrementa al incrementarse $Ue(k)$, pero $Um(k)$ es diferente a $Ue(k)$ debido a los errores en la estimación de los tiempo de ejecución de la tareas, ya que la carga de trabajo es impredecible y variante con el tiempo. Otra diferencia entre $Um(k)$ y $Ue(k)$ es que la primera no puede exceder más del 100% mientras que $Ue(k)$ no tiene este límite.

Dependiendo de la variable de control utilizada en el lazo de control retroalimentado, podemos tener tres diferentes configuraciones del sistema a controlar:

- Controlar la utilización $Um(k)$
- Controlar la razón de pérdida $RP(k)$
- Controlar simultáneamente la razón de pérdida $RP(k)$ y la utilización $Um(k)$.

3.3. **Variador de Voltaje.**

El *variador de voltaje* (o actuador) se encarga de ajustar las velocidades de ejecución V_{ij} de cada tarea τ_i de acuerdo al ajuste de carga calculado por el controlador PID. Consecuentemente, el *variador de voltaje* cambia la utilización total estimada $Ue(k)$ en cada instante de muestreo k, dependiendo de la utilización requerida $\Delta U(k)$, obtenida del controlador PID.

El cambio de la utilización total estimada $Ue(k)$ se realiza al cambiar las velocidades de ejecución V_{ij} de las tareas τ_i. Por lo tanto la utilización estimada de cada tarea Ue_{ij} cambia al modificarse las velocidades de ejecución.

El objetivo del variador de voltaje es cambiar la nueva utilización total estimada a $Ue(k+1) = Ue(k)+\Delta U(k)$. El variador de voltaje ejecuta un algoritmo de optimización

para minimizar el consumo de la energía, estos algoritmos serán descritos con mayor detalle en la sección 4.7.

4. **El Planificador de Tareas en Tiempo Real**

   La arquitectura del planificador retroalimentado con restricciones de energía cuenta con un planificador básico y un despachador. El planificador se encarga de dar un orden de ejecución a las tareas que se encuentran en la cola de tareas aceptadas. El orden de ejecución se da de acuerdo a la política de planificación seleccionada (RM o EDF). El despachador se encarga de posicionar la tarea τ_a, la cual es seleccionada por el planificador, en el procesador para su ejecución.

5. **Ejecución de las Tareas.**

   Una vez que la tarea τ_a es seleccionada para ejecución junto con su correspondiente velocidad de ejecución V_{aj}, la tarea τ_a se ejecutará en el procesador a la velocidad V_{aj} seleccionada por el variador de voltaje. De esta forma, mientras la tarea τ_a se encuentre en ejecución, no podrá cambiar su velocidad de procesamiento sino hasta el próximo periodo de muestreo.

   Durante la ejecución, las tareas pueden variar sus tiempos de cómputo o ejecución A_{ij} por dos razones:

   - Debido a que entre cada instancia se pueden ejecutar por distintos tiempos de cómputo.
   - Debido a que en cada periodo de muestreo, se seleccionan diferentes velocidades de ejecución para las tareas.

   En si, el planificador se encuentra embebido dentro del lazo de control, y comprende todas las etapas descritas de la arquitectura del sistema.

   Una diferencia clave entre la teoría de la planificación clásica y el planificador con retroalimentación planteado en esta tesis, es que la primera asume que los parámetros temporales de cada tarea son conocidos *a priori*, y nuestro trabajo se enfoca a sistemas

en donde el ambiente es impredecible y variante en el tiempo. Por esta razón se da la necesidad del lazo de control que dinámicamente ajusta la carga de trabajo en tiempo de ejecución.

4.5 Metodología de Diseño del Controlador

Basado en la arquitectura del planificador y las especificaciones de desempeño, establecemos una metodología de diseño basada en la teoría de control con retroalimentación. Usando esta metodología, un diseñador de sistemas puede diseñar un planificador adaptable y satisfacer las especificaciones de desempeño del sistema mediante métodos analíticos. Esta metodología funciona contrariamente a aproximaciones existentes que dependen de iteraciones laboriosas de diseño, sintonización y pruebas. Los pasos de esta metodología son:

1. El diseñador especifica el comportamiento dinámico deseado mediante métricas de desempeño en *estado transitorio y estado estable* (ver capítulo 3).

2. El diseñador establece un modelo del sistema en tiempo real, este modelo describe la relación matemática entre la entrada de control y la variable a controlar de un sistema mediante ecuaciones diferenciales o de diferencia. El modelo matemático es importante porque proporciona las bases en el diseño del controlador.

3. Basándose en las especificaciones y el modelo obtenido en los pasos 1 y 2, el diseñador aplica técnicas matemáticas existentes en la teoría de control para diseñar el algoritmo de planificación retroalimentado con garantías analíticas sobre los comportamientos en estado transitorio y estado estable. Este paso es similar al proceso que un ingeniero de control utiliza al diseñar un controlador para un sistema de control retroalimentado.

Para demostrar la eficiencia de esta metodología, está será aplicada en las siguientes secciones en el diseño del algoritmo de planificación con la finalidad de garantizar las restricciones de un conjunto de especificaciones de desempeño enfrentando variaciones en la carga de trabajo.

4.5.1 Especificación y Métricas de Desempeño

El comportamiento dinámico de los sistemas en tiempo real adaptables a cambios en la carga o en los recursos ha recibido especial atención en los últimos años. El comportamiento transitorio de un sistema adaptable representa la respuesta y la eficiencia de los cambios en las velocidades de ejecución de las tareas en respuesta a los cambios en los tiempo de cómputo de las tareas. El comportamiento en estado estable describe el desempeño del sistema después de su respuesta transitoria. Las especificaciones de desempeño consisten de un conjunto de perfiles de desempeño en términos de las variables controladas, las cuales están representadas mediante la utilización medida $Um(k)$ y razón de pérdida de plazos $RP(k)$.

Perfiles de desempeño

Los perfiles de desempeño describen propiedades importantes en estado transitorio y estado estable de un sistema en término de su variable controlada. Desde el punto de vista de la teoría de control, un sistema de tiempo real cambia de *estado estable* a *estado transitorio* cuando el controlador cambia alguna de sus variables controladas. Después de transcurrido un intervalo de tiempo, el sistema alcanza un nuevo estado estable. Para los sistemas en tiempo real, el estado estable puede ser definido como un estado cuando $RP(k)$ se encuentra a $\gamma\%$ de la referencia (por ejemplo $\gamma\% = 2\%$). Los perfiles de desempeño describen el desempeño del sistema tanto en estado transitorio como en estado estable. Algunos perfiles de desempeño utilizados en la teoría de control y que pueden ser utilizados en los sistemas de tiempo real son:

- **Estabilidad:** Un sistema es estable si sus variables a controlar (razón de pérdida de plazos $RP(k)$ ó utilización $Um(k)$), están siempre limitadas para referencias limitadas.
- **Respuesta en estado transitorio:** Representa la respuesta y la eficiencia de los cambios en las velocidades de ejecución de las tareas en respuesta a los cambios en los tiempos de ejecución.
  - **Sobrepaso máximo RP_s y U_s:** Es el máximo valor de una variable controlada en estado transitorio. Representa el desempeño transitorio en el peor caso de un

sistema en respuesta a un perfil de carga. El sobrepaso está definida como la máxima cantidad que el sistema sobrepasa la referencia dividida por la referencia $RP_s = (RP_{max} - RP_r)/RP_r$, $U_s = (U_{max} - U_r)/U_r$. Esta es una métrica importante debido a que una respuesta alta en la razón de pérdida o utilización puede causar la falla del sistema.

- **Tiempo de asentamiento T_e:** Es el tiempo que le toma la variable controlada en entrar a un estado estable después de activar el controlador. Representa que tan rápido puede alcanzar el estado estable.

- **Error en estado estable E_{ERP} y E_{EU}:** Es la diferencia entre los valores promedio de una variable controlada dentro del estado estable y su referencia. El error en estado estable describe cuan preciso el sistema puede alcanzar la referencia en estado estable.

4.5.2 Modelo del Sistema en Tiempo Real

Antes de aplicar los métodos analíticos en el diseño del controlador, necesitamos primero establecer el modelo matemático de la arquitectura mostrada en la figura 4.2. El sistema controlado incluye el variador de niveles de voltaje, el sistema de tiempo real, el planificador y el monitor. La entrada al sistema a controlar es la entrada de control, representada por la utilización requerida $\Delta U(k)$. La salida del sistema incluye las variables de control: razón de pérdida $RP(k)$ y utilización medida $Um(k)$. A pesar de que es difícil modelar un sistema no lineal y variante en el tiempo como son los sistemas en tiempo real, normalmente estos sistemas se consideran lineales para el propósito del diseño, debido a la robustez del control retroalimentado con respecto a las variaciones nominales del sistema. El diagrama a bloques del modelo del sistema controlado se muestra en la figura 4.3. La meta es deducir la función de transferencia del sistema en tiempo real, inspeccionando el modelo desde la entrada de control $\Delta U(z)$, a través de cada bloque, hasta las variables controladas $Um(z)$ y $RP(z)$.

Comenzando desde la entrada de control $\Delta U(k)$, la utilización total estimada $Ue(k)$ es la

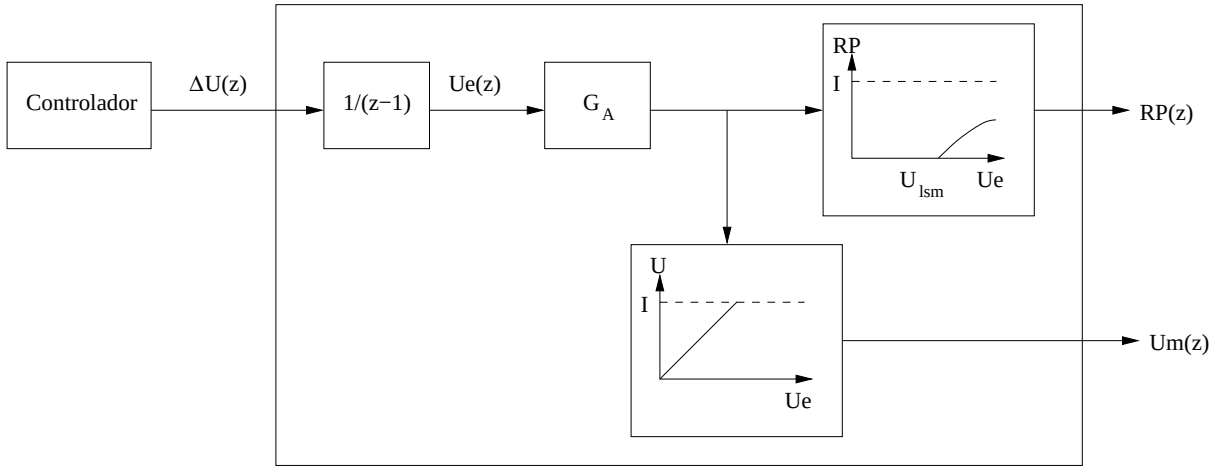


Figura 4.3: Modelo del sistema a controlar

integración de la entrada de control:

$$Ue(k+1) = Ue(k) + \Delta U(k) \tag{4.5}$$

Como los tiempo de ejecución de cada tarea son desconocidos y variantes con el tiempo, la utilización estimada $Ue(k)$ se ve afectada por una función $G_A(k)$. Esta función representa la variación en la carga de trabajo en el procesador. $G_A(k)$ es una función compleja y difícil de modelar, la cual depende de muchos factores, de los cuales los más importantes son:

- La impredecibilidad en los tiempos de ejecución de las tareas, ya que en cada instancia se ejecutan diferentes partes de la tarea. Esto se debe a que en la estructura del código del programa (if,case,for,etc) se presenten diferentes caminos de ejecución en distintas instancias.

- La impredecibilidad inherente en las velocidades de ejecución V_{ij}. Estas velocidades cambian dependiendo de las utilización requerida $\Delta U(k)$, ya que con esta utilización los algoritmos del variador de voltaje (aleatorio y EGA) cambian las velocidades de ejecución de las tareas en cada periodo de muestreo W.

- La variación de la carga de trabajo o número de tareas n en el sistema, ya que la utilización total estimada $Ue(k)$ depende del conjunto de tareas (ver tabla 4.1).

Como $G_A(k)$ es variante con el tiempo, podemos utilizar el máximo valor posible $G_A = max\{G_a(k)\}$. Este valor es conocido como la *razón de utilización* en el *peor caso*, y es utilizado en el diseño de control para garantizar estabilidad en todos los casos. En este sistema suponemos que $G_A \leq 1$, lo cual significa que la utilización real es menor que la utilización total estimada $Ue(k)$. Para obtener el modelo del sistema utilizaremos el término $(G_A * Ue(k))$ para referirnos a la utilización tomando en cuenta las variaciones en los tiempos de ejecución de las tareas del sistema.

La relación entre la utilización total estimada $Ue(k)$ y las variables controladas $Um(k)$ y $RP(k)$ no son lineares debido a la saturación. La saturación complica el diseño de control debido a que las variables controladas se vuelven insensible al control en su zona de saturación. Cuando el procesador se encuentra con baja utilización, $Um(k)$ está fuera de su zona de saturación y permanece cerca de $G_A * Ue(k)$, esto es:

$$Um(k) \approx G_A * Ue(k) \qquad cuando \qquad ((G_A * Ue(k)) \leq 1) \tag{4.6}$$

Sin embargo, como la utilización nunca debe exceder el 100%, $Um(k)$ se satura cuando el procesador se encuentra sobrecargado:

$$Um(k) = 1 \qquad cuando \qquad ((G_A * Ue(k)) > 1) \tag{4.7}$$

Por otro lado, la razón de pérdida $RP(k)$ se satura en cero cuando el procesador se encuentra con baja utilización, por ejemplo cuando la utilización total estimada $(G_A * Ue(k))$ se encuentra por debajo del *límite superior mínimo* $U_{lsm}(k)$ o umbral (ver sección 2.6):

$$RP(k) = 0 \qquad cuando \qquad ((G_A * Ue(k)) \leq U_{lsm}(k)) \tag{4.8}$$

Cuando $(G_A * Ue(k)) > U_{lsm}(k)$, $RP(k)$ normalmente se incrementa en forma no lineal. La relación entre $RP(k)$ y $(G_A * Ue(k))$ necesita ser linealizada tomando la derivada en la vecindad del punto de operación $((G_A * Ue(k)) = U_{lsm}(k))$ como:

$$G_M = \frac{dRP(k)}{d(G_A * Ue(k))} \qquad cuando \qquad ((G_A * Ue(k)) = U_{lsm}(k)) \tag{4.9}$$

G_M es el *factor de razón de pérdida* y puede ser estimada experimentalmente graficando $RP(k)$ en función de $(G_A * Ue(k))$ basado en datos experimentales y medir la pendiente en la

vecindad del punto donde $RP(k)$ empieza a incrementarse después de cero. En la vecindad donde $G_A * Ue(k) = U_{lsm}(k)$, obtenemos la siguiente formula lineal:

$$RP(k) = RP(k-1)+G_M(G_A*Ue(k)-G_A*Ue(k-1)) \qquad cuando \qquad ((G_A*Ue(k)) > U_{lsm}(k)) \tag{4.10}$$

Con las ecuaciones 4.5 a 4.10, podemos deducir la función de transferencia para cada controlador cuando se encuentre fuera de su zona de saturación:

- **Control de utilización:** Bajo la condición que $(G_A * Ue(k)) < 1$, existe una función de transferencia $H_U(z)$ desde la entrada de control $\Delta U(z)$ hasta la variable controlada $Um(z)$. $Um(z) = P_U(z)\Delta U(z)$ y:

$$P_U(z) = \frac{G_A}{(z-1)} \qquad cuando \qquad ((G_A * Ue(k)) < 1) \tag{4.11}$$

- **Control de razón de pérdida:** Bajo la condición de que $(G_A * Ue(k)) > U_{lsm}(k)$, existe una función de transferencia $H_{RP}(z)$ desde la entrada de control $\Delta U(z)$ hasta la variable controlada $RP(z)$. $RP(z) = P_{RP}(z)\Delta U(z)$ y:

$$P_{RP}(z) = \frac{G_A G_M}{(z-1)} \qquad cuando \qquad ((G_A * Ue(k)) > U_{lsm}) \tag{4.12}$$

4.5.3 Diseño del Algoritmo de Control

En esta sección, aplicamos metodologías de la teoría de control en el análisis y diseño del *controlador*. El controlador es el elemento clave del algoritmo de planificación retroalimentado. Primero, definimos las especificaciones de desempeño para un sistema en tiempo real, después presentamos el algoritmo de control y el modelo del lazo para cada variable de control. Basado en el modelo del sistema, aplicamos el método del *lugar geométrico de las raíces* para sintonizar el controlador.

Especificación de desempeño de un sistema en tiempo real

Suponemos un sistema de tiempo real que tenga las especificaciones de desempeño mostrados en la tabla 4.2 con un periodo de muestreo de $W = 0.5$ segundos [23]. Los requerimientos de desempeño en estado transitorio y estable son (1) La razón de pérdida del sistema deberá

permanecer estable ante una carga escalón de 200%, (2) El sistema deberá alcanzar el estado estable dentro de los 15 segundos después de aplicada la carga (tiempo de asentamiento), (3) la máxima razón de pérdida durante el estado transitorio deberá ser menor del 15%. (4) El sistema deberá tener un error en estado estable de razón de pérdida E_{ERP} de menos del 1%, esta razón debe lograrse independientemente de las variaciones en el tiempo de ejecución de las tareas.

Perfiles de carga	**Carga escalón(0,200%)**
T_e	< 30 seg.
RP_{max}	$< 15\%$
E_{ERP}	$< 1\%$
W	0.5 seg.

Tabla 4.2: Especificaciones de desempeño

Diseño del controlador

En cada periodo de muestreo, el controlador calcula la entrada de control $\Delta U(k)$ o cambio en la utilización total estimada, basada en el error de razón de pérdida $E_{RP}(k) = RP_r - RP(k)$ y/o en el error de la utilización $E_U(k) = U_r - Um(k)$. La meta del controlador es satisfacer las especificaciones de desempeño de la tabla 4.2.

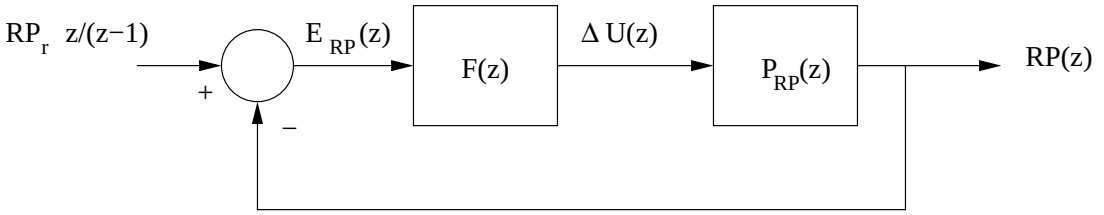
El controlador utiliza una función de control PI (proporcional-integral) para calcular la entrada de control (sección 3.5). Implementaciones digitales de la función de control PI están representadas en las ecuaciones 4.13 y 4.14. La ecuación 4.13 y 4.14 son equivalentes pero la ecuación 4.14 es más eficiente en tiempo de ejecución. La función de transferencia $F(z)$ en el dominio de z del controlador PI está dada por la ecuación 4.15.

$$\Delta U(k) = K_p(E(k) + K_i \sum_{j=0}^{k} E(k)) \tag{4.13}$$

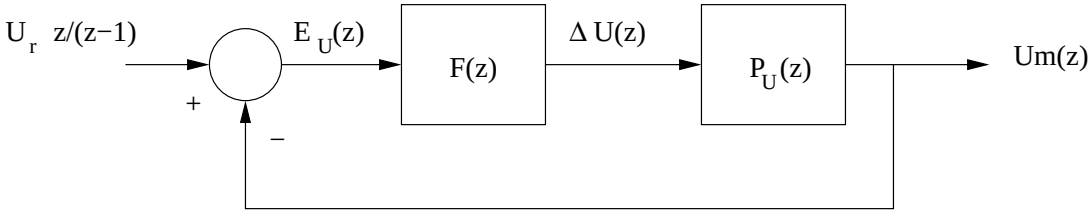
$$\Delta U(k) = \Delta U(k-1) + K_p((K_i + 1)E(k) - E(k-1)) \tag{4.14}$$

$$F(z) = \frac{q(z-t)}{z-1} \qquad donde \qquad q = K_p(K_i+1) \quad y \quad t = \frac{1}{K_i+1} \tag{4.15}$$

El control PI es una función de control ampliamente utilizada y que obtiene buenos desempeños en sistemas de primer y segundo orden. El desempeño de un sistema en tiempo real dependerá de los parámetros del controlador, por lo que para sintonizar estos parámetros aplicaremos métodos de la teoría de control con el fin de garantizar las especificaciones de desempeño.



(a) Lazo de control por razon de perdida



(b) Lazo de control por utilizacion

Figura 4.4: Modelo de los lazos de control

Sintonización del sistema

Para propósitos de diseño, la entrada de un lazo de control con un solo controlador (utilización o razón de pérdida) es la referencia RP_r o U_r, y la salida es la variable controlada razón de pérdida $RP(k)$ o utilización $Um(k)$. Dado el modelo de los sistemas controlados por las ecuaciones 4.11 y 4.12 y el controlador $F(z)$ de la ecuación 4.15, la función de transferencia de cada lazo de control sería:

$$H_{RP}(z) = \frac{F(z)P_{RP}(z)}{1+F(z)P_{RP}(z)} \qquad H_U(z) = \frac{F(z)P_U(z)}{1+F(z)P_U(z)} \tag{4.16}$$

$$RP(z) = \frac{RP_r z}{z-1} H_{RP}(z) \qquad Um(z) = \frac{U_r z}{z-1} H_U(z) \tag{4.17}$$

Basado en estos modelos, podemos aplicar la teoría de control para sintonizar los parámetros del controlador y analizar las propiedades del desempeño del sistema.

Según la teoría de control, el desempeño de un sistema depende de los polos en su función de transferencia en lazo cerrado. Entonces este problema puede verse como un problema de localización de polos. El método del lugar geométrico de las raíces es una técnica gráfica que traza los polos de un sistema en lazo cerrado dentro del plano z. Para aplicar este método podemos utiliza algunas herramientas como *MATLAB* para sintonizar los parámetros del controlador y satisfacer las especificaciones de desempeño. Consideremos el lazo de control de utilización y supongamos también que la carga de trabajo tiene una razón de utilización en el peor caso de $G_A = 1$, obtenemos que la localización de los polos en lazo cerrado son $p_0 = 0.89$ y $p_1 = -0.19$ teniendo los siguientes parámetros en el controlador:

$$K_p = 0.5, \qquad K_i = 0.1 \tag{4.18}$$

La tabla 4.3 resume la simbología utilizada en el diseño del controlador, así como las variables utilizadas en las diferentes configuraciones de control propuestas.

4.6 Algoritmos de Planificación Retroalimentados

En esta sección presentamos el diseño de los algoritmos de planificación retroalimentados utilizando control de utilización y/o razón de pérdida de plazos. La meta principal es alcanzar las especificaciones de desempeño de la tabla 4.2.

Control de utilización $Um(k)$

Este algoritmo de planificación utiliza un lazo de control de utilización (figura 4.4(b)), para controlar la utilización $Um(k)$. Este algoritmo puede garantizar que el sistema tenga cero plazos perdidos en estado estable si la referencia $U_r \leq U_{lsm}$ donde U_{lsm} es la utilización superior mínima o utilización de umbral, bajo la cual el sistema es planificable.

Parámetro	Notación
k	Instante de muestreo $k = 1, 2, 3, \ldots$
W	Periodo de muestreo
$Ue(k)$	Variable manipulada, utilización estimada en el instante k
$Um(k)$	Variable a controlar, utilización medida en el instante k
$RP(k)$	Variable a controlar, razón de plazos perdidos en el instante k
$\Delta U(k)$	Utilización requerida, salida del controlador en el instante k
U_r	Referencia de desempeño de utilización
RP_r	Referencia de desempeño de razón de plazos perdidos
E_U	Error de utilización $E_U = U_r - Um(k)$
E_U	Error de razón de plazos perdidos $E_{RP} = RP_r - RP(k)$
E_{ERP}	Error en estado estable de razón de plazos perdidos
E_{EU}	Error en estado estable de utilización
RP_s	Valor máximo de $RP(k)$ en estado transitorio
U_s	Valor máximo de $Um(k)$ en estado transitorio
T_e	Tiempo de asentamiento o estabilización de la variable de control
G_A	Factor de utilización en el peor caso
G_M	Factor de razón de plazos perdidos en el peor caso
$P_U(z)$	Modelo del sistema del lazo de control de Um
$P_{RP}(z)$	Modelo del sistema del lazo de control de RP
$H_U(z)$	Función de transferencia del lazo de control de Um
$H_{RP}(z)$	Función de transferencia del lazo de control de RP
$F(z)$	Función de transferencia del controlador PI
K_p y K_i	Parámetros del controlador PI

Tabla 4.3: Simbología utilizada en el diseño del controlador

Este algoritmo no puede detectar cuan severo el sistema se encuentra sobrecargado cuando $Um(k)$ permanece al 100%, debido a que la utilización del procesador medida $Um(k)$ se satura al 100%. Como consecuencia este algoritmo puede tener un mayor tiempo de asentamiento que el obtenido en los análisis en condiciones severas de sobrecarga. Por ejemplo, supongamos que la utilización estimada tomando en cuenta sus variaciones este en $G_A * Ue(k) = 200\%$ y la referencia sea $U_r = 99\%$, el error en la utilización calculado por el controlador sería $E_U = U_r - Um(k) = 0.99 - 1 = -0.01$, sin embargo, el error debería ser $E_U = U_r - G_A * Ue(k) = 0.99 - 2 = -1.01$ conforme al modelo lineal. En el caso extremo cuando $U_r = 100\%$, el sistema puede permanecer en sobrecarga (un tiempo de asentamiento infinito) debido a que el error es cero ($E_U = 0$) incluso cuando el sistema este sobrecargado. Por lo tanto la

referencia U_r deberá estar por debajo del 100% (por ejemplo 95% o 90%), para evitar el impacto de la saturación en el desempeño del control.

Control de razón de pérdida $RP(k)$

Este algoritmo utiliza un lazo de control de razón de pérdida (figura 4.4(a)), para controlar directamente la razón de pérdida del sistema $RP(k)$. Comparado con el control de utilización la ventaja de este algoritmo es que no depende de ningún conocimiento acerca del límite de utilización, y por lo tanto, es aplicable en la mayoría de los sistemas. El lazo de control de razón de pérdida cambia la utilización medida $Um(k)$ a la vecindad del umbral de utilización U_{lsm}. Una ventaja adicional de este algoritmo es que puede alcanzar una mayor utilización que el control de utilización debido a que la utilización de umbral es mayor al límite de utilización.

Este algoritmo tiene restricciones en la referencia de la razón de pérdida RP_r debido a la saturación, esto es porque la razón de pérdida $RP(k)$ se satura en cero, por lo que no puede detectar cuan baja pueda estar la utilización del sistema. Al igual que el control de utilización, entre más pequeña sea la referencia, mayor será el tiempo de asentamiento, esto es porque el control de razón de pérdida mide un error de una pequeña magnitud y genera una pequeña entrada de control. Por ejemplo, supongamos que la utilización estimada se encuentre en $G_A * Ue(k) = 10\%$ y que la referencia de razón de pérdida sea 1%, el error medido por el controlador debería ser $E_{RP} = RP_r - RP(k) = 0.01 - 0 = 0.01$. Sin embargo, el error debería ser mucho mayor de acuerdo al modelo lineal. En el caso extremo, cuando $RP_r = 0$ el procesador puede tener una baja utilización debido a que el error $E_{RP} = 0$ siempre que el sistema se encuentre con una baja utilización. Por lo tanto, la referencia de razón de pérdida deberá estar a alguna distancia del limite de saturación 0 (por ejemplo $RP_r = 1\%$) y así evitar el impacto de la saturación sobre el desempeño de control. Desafortunadamente, una referencia positiva significa que el sistema no puede conseguir una razón de pérdida de cero.

Control integrado de utilización $Um(k)$ y razón de pérdida $RP(k)$

Este algoritmo integra el control de utilización y el control de razón de pérdida, con el propósito de combinar las ventajas de ambos controles. En este esquema, tanto $Um(k)$ como $RP(k)$ son monitorizados en cada periodo de muestreo y alimentados a dos controladores independientes. La entrada del control de utilización $\Delta U_U(k)$ es comparada con la entrada de control del control de razón de pérdida $\Delta U_{RP}(k)$ y el más pequeño de los dos $\Delta U(k) = min(\Delta U_U(k), \Delta U_{RP}(k))$ se envía al actuador o variador de voltaje. La estructura del control integrado puede conseguir las ventajas de ambos controles. Si el control de utilización se emplea solo, este intentará cambiar la utilización $Um(k)$ a su referencia U_r en estado estable, y el lazo de control de razón de pérdida intentará cambiar $Um(k)$ a la vecindad del límite mínimo de utilización $U_{lsm}(k)$ en estado estable.

Debido al operador $min(U_{lsm}(k), U_r)$ sobre las dos entradas de control, el lazo integrado de control deberá cambiar la utilización estimada al valor más pequeño de los dos. En un sistema planificado por este algoritmo, si $U_r \leq U_{lsm}(k)$ el control de utilización domina en estado estable y garantiza que la utilización medida $Um(k)$ permanezca cerca de la referencia U_r provocando que una razón de pérdida en estado estable de $RP(k) = 0$. Por otro lado, si $U_r > U_{lsm}(k)$, el control de razón de pérdida domina en estado estable y garantiza que la utilización medida $Um(k)$ permanezca cerca de su límite mínimo de utilización $U_{lsm}(k)$ provocando una razón de pérdida $RP(k) = RP_r$ en estado estable.

4.7 Algoritmos del Variador de Voltaje

El variador de voltaje (o actuador) se encarga de ajustar las velocidades de ejecución de cada tarea τ_i de acuerdo al ajuste de carga $\Delta U(k)$ calculado por el controlador PID. Consecuentemente, el variador de voltaje cambia la utilización estimada $Ue(k)$ en cada instante de muestreo k, dependiendo de la utilización requerida $\Delta U(k)$.

El objetivo del variador de voltaje consiste en maximizar el ahorro de energía del sistema, con la restricción de que la utilización total estimada $Ue(k+1)$ no sobrepase la nueva utilización $Ue(k) + \Delta U(k)$. En consecuencia, el problema de optimización de la sección 4.3,

se replantea de la siguiente forma:

Maximizar :

$$Z = \sum_{i=1}^{n} \sum_{j \in N_i} S_{ij} x_{ij} \tag{4.19}$$

Sujeto a:

$$Ue(k+1) \leq Ue(k) + \Delta U(k) \tag{4.20}$$

$$\sum_{j \in N_i} x_{ij} = 1,\ i = 1, \ldots, n$$

$$x_{ij} = \begin{cases} 1 & \text{Si se selecciona una velocidad } V_{ij}\ (j \in N_i) \text{ para la tarea } \tau_i \\ 0 & \text{En caso contrario,} \end{cases}$$

La condición 4.20 nos indica que la utilización total estimada $Ue(k+1)$ en el siguiente instante de muestreo k, deberá ser menor o igual a la utilización estimada del instante anterior $Ue(k)$ más la utilización requerida $\Delta U(k)$, con el propósito de maximizar el ahorro de energía consumida por el procesador.

Cada tarea en el sistema consume una cierta cantidad de energía, la cual depende de la velocidad V_{ij} seleccionada para su ejecución. Por lo cual, el problema de optimización, consiste en encontrar las velocidades de ejecución de cada tarea tal que se maximice el ahorro de energía del sistema y que se ajuste la utilización estimada $Ue(k+1)$ de acuerdo a la restricción descrita en la ecuación 4.20. La solución a este problema, requiere la exploración de un gran número de combinaciones (debido a que se trata de un problema de tipo NP-Duro), lo cual demanda un alto tiempo de cómputo.

Con el fin de proporcionar una solución de bajo costo computacional a este problema para un ambiente dinámico, hemos diseñado dos algoritmos de manejo dinámico de voltaje que se ejecutan dentro de la arquitectura de planificación con retroalimentación planteada en la figura 4.2 de la sección 4.4.

El primer algoritmo selecciona las velocidades de las tareas para su ejecución en forma aleatoria, mientras que el segundo algoritmo utiliza un procedimiento ambicioso mejorado (*enhanced greedy algorithm EGA* [28]) en la selección de las velocidades de ejecución de las tareas.

4.7.1 Algoritmo Aleatorio

Este algoritmo cambia la utilización requerida $\Delta U(k)$ del sistema mediante el ajuste de las velocidades de ejecución de las tareas en el procesador. Por ejemplo, si se cambia la velocidad de ejecución de la tarea τ_i desde V_{im} hasta V_{il}, esto ajusta la utilización requerida por $\Delta U(k) = Ue_{il}(k) - Ue_{im}(k)$, donde $Ue_{il}(k)$ y $Ue_{im}(k)$ representan la utilización estimada de la tarea τ_i ejecutándose a los niveles de velocidad l y m respectivamente. En la figura 4.5 se presenta el pseudocódigo de este algoritmo.

```
1:  Algoritmo Aleatorio
2:  entrada Utilización requerida ΔU(k)
3:  salida
4.  begin
5:      Util_ini = Ue(k);
6:      if(ΔU(k) < 0)
7:          while(Ue() >= (Util_ini+ ΔU(k))) && (existe_tarea_rápida() == TRUE)
8:              índice = sel_tarea_rápida();
9:              cambiar_niveles_voltaje(índice,lenta);
10:     else
11:         while(Ue() <= (Util_ini+ΔU(k))) && (existe_tarea_lenta() == TRUE)
12:             índice = sel_tarea_lenta();
13:             cambiar_niveles_voltaje(índice,rápida);
14: end
```

Figura 4.5: Pseudocódigo del algoritmo aleatorio

Básicamente, dependiendo del signo de la utilización requerida $\Delta U(k)$, el algoritmo aumenta o disminuye la utilización estimada. Por ejemplo, si se requiere una utilización negativa, esto significa que es necesario disminuir la utilización estimada $Ue(k)$ cambiando la velocidad de algunas de las tareas a una velocidad más rápida, la selección de las tareas se rea-

liza de forma aleatoria mediante la función *sel_tarea_rápida()* o *sel_tarea_lenta()* dependiendo del signo de la utilización requerida $\Delta U(k)$. Una vez seleccionada la tarea, se le cambia la velocidad o voltaje de ejecución de esa tarea mediante la función *cambiar_niveles_voltaje()*. El parámetro *lenta* en la función *cambiar_niveles_voltaje()* significa que se le cambiará la velocidad a la tarea *índice* un nivel de velocidad más lento que el actual. El parámetro *rápida* indica lo mismo, solo que cambia un nivel de velocidad más rápida con la que se encontraba, para la tarea *índice*.

Este procedimiento se repite hasta que el cambio de la utilización requerida se haya alcanzado o hasta que ninguna tarea se le pueda cambiar su velocidad de ejecución. El objetivo es obtener la nueva utilización estimada total $Ue(k+1) = Ue(k) + \Delta U(k)$.

4.7.2 Algoritmo EGA

En el segundo algoritmo implementado en el variador de voltaje utilizamos un procedimiento de optimización formulado como el problema de la mochila con múltiples opciones (*multiple-choices knapsack problem MCKP*) con variables binarias [28]. En este procedimiento la capacidad de la *mochila* c cambia en cada periodo de muestreo debido a la variación de la utilización estimada total $Ue(k)$ dado por los cambios en la utilización requerida $\Delta U(k)$. Entonces, la capacidad de la *mochila* está dada por:

$$c = Ue(k) + \Delta U(k) \tag{4.21}$$

El algoritmo del variador de voltaje consiste de tres parte [28]:

1. Un *algoritmo de reducción*, el cual convierte el problema *MCKP* original en un problema (*knapsack problem KP*) estándar.

2. Un algoritmo de aproximación ambicioso mejorado (*enhanced greedy algorithm EGA*), el cual encuentra una solución aproximada al problema *KP*.

3. Un *algoritmo de restauración*, el cual reconstruye la solución del *MCKP* de la solución del *KP* estándar.

Este algoritmo se basa en la reducción del *MCKP* a *KP* mediante el concepto de las gráficas convexas (*convex hull*). Para reducir el problema P_0 del *MCKP* utilizaremos los siguiente problemas auxiliares:

P_1: El problema del *MCKP* Truncado

El problema P_1 se construye a partir del problema de optimización replanteado en las ecuaciones 4.19 y 4.20. En P_1 se extraen los elementos más ligeros de cada tarea (elementos con la menor Ue_{ij} o mayor velocidad de ejecución), y todos estos elementos son insertados en la mochila. La sumatoria de los elementos más ligeros está representado por $S_{min} = \sum_{i=1}^{n} S_{i1}$ y $U_{min} = \sum_{i=1}^{n} Ue_{i1}$, donde n es el número de tareas en el sistema, S_{min} es el mínimo ahorro que se puede obtener y U_{min} es la utilización de las tareas ejecutándose a la máxima velocidad. Ahora, debemos de considerar la nueva capacidad de la mochila $c = c - U_{min}$ y la condición $\sum_{j \in N_i} x_{ij} = 1$ cambia por $\sum_{j \in N_i} x_{ij} \leq 1$, debido a que los elementos ligeros se encuentran en la solución del problema.

P_2: El problema del *MCKP* Truncado con relajación

El problema P_2 se obtiene del problema P_1 relajando la condición de integridad: $0 \leq x_{ij} \leq 1$, en otras palabras, los valores de x_{ij} no son necesariamente 0 o 1.

P_3: El problema del *MCKP* sobre la gráfica convexa

Dado el problema P_2, encontramos los elementos de la gráfica convexa (*convex hull*) por cada tarea. Los elementos que constituyen la gráfica convexa son llamados *P-no-dominados* y están representados en una gráfica de (R_{ij} *ahorro de energía*, H_{ij} *utilización*). El ahorro de energía y la utilización (sin tomar en cuenta los elementos más ligeros de P_2) se definen como: $s_{ij} = S_{ij} - S_{i1}$ y $u_{ij} = Ue_{ij} - Ue_{i1}$.

Para reducir el conjunto de elementos de la gráfica convexa, se aplican los siguientes *criterios de dominación*:

Criterio 1. Si dos elementos r y q de la misma tarea, satisfacen las condiciones $s_{ir} \leq s_{iq}$ y $u_{ir} \geq u_{iq}$, entonces se dice que r es dominado por q. En una solución óptima al problema P_3, $x_{ir} = 0$, es decir, los elementos dominados no entran en la solución óptima.

Criterio 2. Si tenemos algunos elementos de la misma tarea r, q y t, tal que $s_{ir} \leq s_{iq} \leq s_{it}$,

$u_{ir} \leq u_{iq} \leq u_{it}$ y

$$\frac{(s_{iq} - s_{ir})}{(u_{iq} - u_{ir})} \leq \frac{(s_{it} - s_{iq})}{(u_{it} - u_{iq})} \tag{4.22}$$

entonces $x_{iq} = 0$ para cualquier solución óptima de P_2, se dice que el elemento q es *P-dominado*. Para obtener la solución de P_3, excluimos los elementos *P-dominados*. Los elementos restantes después de excluir todos los elementos *P-dominados* son llamados elementos *P-no-dominados*. Todos los elementos no dominados de la misma tarea, si se representan como puntos en un espacio de dos dimensiones, formarían la parte superior de la gráfica convexa del conjunto N_i y denotan el nuevo conjunto de elementos *P-no-dominados* $\{(R_{ij},H_{ij})\}$.

El conjunto de elementos *P-no dominados* se encuentra examinando todos los elementos de cada tarea en orden incremental, en base a la ecuación 4.22.

P_4 Problema *KP* equivalente

El problema P_4 se construye a partir de P_3. Por cada clase se definen incrementos definidos como:

$$P_{ij} = (R_{ij} - R_{i(j-1)});\ i = 1, \ldots, n;\ j = 2, \ldots, CH_i \tag{4.23}$$

$$W_{ij} = (H_{ij} - H_{i(j-1)});\ i = 1, \ldots, n;\ j = 2, \ldots, CH_i \tag{4.24}$$

donde CH_i es el número de elementos *P-no dominados* de la gráfica convexa de la clase N_i. Cuando se resuelve el problema P_3, podemos descartar la condición $\sum_{j \in N_i} \ x_{ij} \leq 1,\ i = 1, \ldots, n$ y resolver el problema seleccionando incrementos en cada clase. Para resolver el problema P_4 es necesario seguir las siguientes propiedades: [1]

Propiedad 1. La combinación de todos los elementos (incrementos) de la clase N_i $(i = 1, \ldots, n)$ que resulten en una solución óptima al problema P_4, corresponde a un elemento especifico seleccionado de la clase N_i dentro del problema P_3. Si m denota este elemento especifico, entonces $\sum_{j \in N_i}(P_{ij}x_{ij}) = R_{im}$ y $\sum_{j \in N_i}(W_{ij}x_{ij}) = H_{im}$. En cada clase todas los incrementos están ordenadas en orden decreciente dada la relación $\frac{P_{ij}}{W_{ij}}$.

Propiedad 2. No debe existir espacios vacíos dentro de un conjunto de incrementos correspondiente a la solución de cualquier clase.

[1] No confundir los términos P_{ij} y W_{ij} con el periodo de las tareas P_i y el periodo de muestreo W

Para resolver el problema P_4 debemos recolectar todas los incrementos de todas las clases (siguiendo un orden decreciente de $\frac{P_{ij}}{W_{ij}}$) como candidatas para incluirlas dentro de una sola clase PW, ahora el problema se convierte en un problema de la mochila estándar. La idea principal del algoritmo ambicioso mejorado (EGA) para resolver el problema de la mochila consiste en insertar los incrementos p_i, w_i dentro de la capacidad de la mochila ($c - U_{min}$) en orden decreciente (dependiendo del factor $\frac{p_i}{w_i}$) hasta que la capacidad de la mochila se llene por completo, o hasta que no se puedan incluir más incrementos. Si la mochila se completa exactamente a su capacidad, entonces la solución es óptima. Una solución aproximada al problema P_4 se obtiene mediante:

$$Z_4 = max\{p_{max}, (p_1 + p_2 + \ldots + p_{m-1} + \alpha)\} \tag{4.25}$$

el término α es un incremento, que se consigue después de encontrar un incremento de quiebre, el cual es llenado por un incremento de otra clase que no ha sido agregado.La solución aproximada al problema P_0 está definida como $Z_4 + S_{min}$. Donde S_{min} son los elementos truncados en el problema P_1. El pseudocódigo del algoritmo EGA se presenta en la figura 4.6.

4.8 Resumen

En este capitulo se presenta el modelo del sistema y la arquitectura propuesta del planificador de tareas en tiempo real con retroalimentación. Se definieron las variables de control $Um(k)$ y $RP(k)$, las cuales son utilizadas por el sistema para alcanzar las especificaciones de desempeño. Se presenta el diseño del controlador mediante técnicas tomadas de la teoría de control. El sistema a controlar puede se modeló como una función de transferencia de primer orden (ecuaciones 4.11 y 4.12) con una zona de saturación (ecuaciones 4.7 y 4.8) para cada variable de control, utilización $Um(k)$ y razón de pérdida $RP(k)$.

Se formuló el diseño de los tres diferentes tipos de control; control de utilización, control de razón de pérdida y la integración de estos dos. En el diseño del controlador se utilizó la teoría de control basada en el modelo matemático de un sistema en tiempo real. Y por

1: **Algoritmo EGA (Enhanced Greedy Algorithm)**

2: **entradas:** Capacidad del *Knapsack* $c = Ue(k) + \Delta U(k)$

S_{ij},Ue_{ij} *//Ahorro de energía y utilización del problema* P_0

$S_{min} = \sum_{i=1}^{n} S_{i1}$; $U_{min} = \sum_{i=1}^{n} Ue_{i1}$ *//Ahorro y utilización a la máxima velocidad*

3: **salidas:** x_i, (p^*, w^*): *//Conjunto solución*

4: **begin**

5: **for** (j = 1; j < n, j++) **begin**

6: $s_{ij} = S_{ij} - S_{i1}$; $u_{ij} = Ue_{ij} - Ue_{i1}$; $\hat{c} = (c - U_{min})$ *//Problema* P_1

7: **for** (todos los elementos en N_i) *//Problema* P_3

8: **if** ($s_{ir} \leq s_{iq}$ y $u_{ir} \geq u_{iq}$)

9: **Elimina** *elemento r de* N_i *(Criterio de dominancia 1)*

10: **if** $\left(\frac{s_{iq}-s_{ir}}{u_{iq}-u_{ir}} \leq \frac{s_{it}-s_{iq}}{u_{it}-u_{iq}}\right)$

11: **Elimina** *elemento q de* N_i *(Criterio de dominancia 2)*

12: **for** (todos los elementos P-no dominados en N_i)

13: $R_{ij} = s_{ij}$; $H_{ij} = u_{ij}$ *//Gráfica convexa con elementos P-no dominados*

14: $P_{ij} = (R_{ij} - R_{i(j-1)})$; $W_{ij} = (H_{ij} - H_{i(j-1)})$, $j = 2, \ldots, CH_i$ *//Incrementos* P_4

15: **endfor**

16: **Inserta** *todos los elementos* (P, W) *ordenados por* $\frac{P}{W}$ *en el arreglo PW*

17: **Ejecuta** *el algoritmo EGA con PW como entrada, la cual de como salida* x_{ij}, p^*, w^*

18: $P = S_{min} + p^*$; $W = U_{min} + w^*$; //**Soluciones** *de ahorro de energía y utilización*

19: **end**

Figura 4.6: Pseudocódigo del algoritmo EGA

último, se presentaron los dos algoritmos implementados en el *variador de voltaje*. El primero algoritmo utiliza una función aleatoria para la selección de las velocidades de ejecución. El segundo algoritmo implementado en el variador de voltaje, formula el problema como el problema de la mochila con múltiples opciones (*MCKP*) con variables binarias. La solución se obtiene mediante un algoritmo ambicioso mejorado (*enhanced greedy algorithm EGA*).

En el siguiente capitulo se presentará un conjunto de simulaciones de la arquitectura de planificación propuesto. Estas simulaciones tienen como objetivo verificar el comportamiento de la arquitectura de planificación, ante distintos conjuntos de tareas de tiempo real con parámetros dinámicos.

Capítulo 5

Simulación y Evaluación de Resultados

5.1 Introducción

En este capitulo introduciremos los principales componentes del software diseñados para la simulación de la arquitectura, así como los resultados de las simulaciones de las diferentes configuraciones del sistema revisadas en el capitulo anterior. Básicamente, el software contempla la simulación del funcionamiento del procesador de velocidad variable, la ejecución de las tareas en tiempo real sobre el procesador, la monitorización de las variables de control $Um(k)$ y $RP(k)$, la operación del controlador y el cambio de las velocidades de ejecución de las tareas mediante los algoritmos implementados en el variador de voltaje.

El software de simulación está desarrollado en lenguaje C (*gcc*) sobre plataforma *Linux Red-Hat 7.1* en una computadora Pentium III a 550 MHz con 256 en memoria RAM. Se diseño una interface gráfica en donde el usuario podrá seleccionar la configuración deseada del sistema y probar su desempeño. Para el desarrollo de esta interface gráfica se utilizó la librería *Qt* [32] para Linux.

El simulador está desarrollado de forma modular, de modo que modificaciones en el software como son los cambios en la configuración de los lazos de control, la implementación de nuevas políticas de planificación, cambios de los parámetros en la generación de las tareas, se puedan realizar con facilidad.

Este programa simula la ejecución de las tareas en tiempo real sobre procesadores de velocidad variable (sección 1.3). El tiempo total simulado es de 60 segundos. El simulador está compuesto de cinco módulos principales:

1. *Generador de tareas*, el cual genera los parámetros de las tareas (periodos P_i , plazos de respuesta D_i, tiempo de ejecución estimados C_i y tiempo de ejecución actuales A_i) en forma aleatoria.

2. *Planificador ejecutor*, el cual emula el funcionamiento del reloj en un sistema de tiempo real.

3. *Planificador*, el cual simula la planificación o calendarización del conjunto de tareas del sistema. Este procedimiento se encarga de dar un orden de ejecución al conjunto de tareas. Este orden depende de la política de planificación seleccionada (EDF o RM).

4. *Controlador PID*, el cual calcula la utilización requerida $\Delta U(k)$ dependiendo de las referencias de desempeño y las variables de control obtenidas por el *monitor*.

5. *El variador de niveles de voltaje*, el cual calcula las velocidades de ejecución V_{ij} para cada una de las tareas τ_i mediante la implementación de los algoritmo combinatorios descritos en la sección 4.7.

5.2 Carga de Trabajo

Para probar el desempeño del *planificador de tareas en tiempo real con restricciones de energía en retroalimentación*, es necesario sobrecargar al sistema mediante un aumento gradual en la carga de trabajo. Inicialmente, la carga de trabajo está compuesta de 30 tareas periódicas, los parámetros temporales de éstas tareas (tiempos de cómputo, periodos, etc.) se generan

en forma aleatoria al inicio de la simulación. Estas tareas se activan en el tiempo cero de la simulación

Durante el transcurso de la simulación, se genera otro conjunto de 30 tareas periódicas, pero con tiempos de arribo desconocidos. El tiempo de arribo de éstas tareas se genera siguiendo una distribución uniforme entre [10, tiempo total], donde el tiempo total es de 60 segundos.

Se simulará la utilización de procesadores *Crusoe TM5400*, los cuales cuentan con la capacidad de varía su velocidad y voltaje de operación desde 200MHz@1.1V hasta 700MHz@1.6V en pasos discretos de 33.33MHz. Los niveles de velocidad-voltaje normalizados entre 0-1 están representado por el $V_{ij} = \{$*1.0, 0.952, 0.904, 0.857, 0.809, 0.761, 0.714, 0.666, 0.619, 0.571, 0.523, 0.476, 0.428, 0.380, 0.333, 0.285*$\}$, por lo que una tarea τ_i ejecutándose a la velocidad 1.0 ($j = 1$) terminará su ejecución más rápido que si se ejecutara a la velocidad 0.285, solo que el procesador al operar a al velocidad 1.0 (máxima velocidad) consumirá más energía que ejecutándose a la velocidad 0.285.

La generación de tareas se efectúa al inicio de cada simulación, y el tiempo de ejecución actual se genera en cada instancia de ejecución. En este sistema supondremos una unidad de tiempo de 0.1 ms para la generación de tareas. En la generación de los parámetros para cada una de las tareas en tiempo real τ_i, se utilizaron las siguientes distribuciones estadísticas:

C_{i1}: Tiempo de ejecución estimado de la tarea τ_i ejecutándose a la velocidad 1, el cual sigue una distribución uniforme en el rango de [0.2,0.8] ms. Para el resto de los niveles de velocidades j, el tiempo de ejecución se obtiene mediante: $C_{ij} = \frac{C_{i1}}{V_{ij}}$.

A_{ij}: Tiempo de ejecución actual de la tareas τ_i ejecutándose a la velocidad j, el cual sigue una distribución uniforme en el rango de [C_{ij}-raíz, C_{ij}], donde $raiz$ se obtiene mediante $raiz = sqrt(C_{ij})$.

P_i: Periodo de activación de la tarea τ_i, cada una de las versiones de la tarea τ_i (que se ejecutan a diferentes velocidades j) tienen el mismo periodo. P_i sigue una distribución uniforme en el rango de [40,50] ms.

D_i: Plazo de respuesta máximo de la tarea τ_i, igual al periodo $D_i = P_i$.

B_i: Tiempo de arribo de las tarea τ_i con comportamiento aperiódico, se obtiene siguiendo una distribución uniforme en el rango de [10,*tiempo total*] donde *tiempo total* es el tiempo que tarda en completarse una simulación.

5.3 Saturación de las Variables de Control

En esta sección, se realizaron un conjunto de experimentos para verificar las propiedades de saturación de las variables de control utilizando las políticas de planificación *EDF* y *RM*. En estos experimentos se utilizaron diferentes cargas para forzar al sistema ($Ue = 70, 80, 90, \ldots, 200$), mientras que el controlador y el actuador permanecen inactivos por 60 segundos. Las gráficas de la utilización promedio Um y la razón de pérdida RP con respecto a la utilización total estimada Ue tanto para las políticas *EDF* como *RM* se muestran en la figuras 5.1 y 5.2 respectivamente. Cada punto en la figura representa los valores promedio de 5 ejecuciones.

Podemos ver en la figura 5.1, que aplicando la política de planificación *EDF*, la utilización Um se satura al 100% mientras que la utilización total estimada Ue excede el 100%. La razón de pérdida se mantiene en cero cuando la utilización actual medida Um se encuentra por debajo del 100% de la utilización. La pérdida de plazos empieza a ocurrir cuando la utilización Um se satura al 100%. En la figura 5.2 observamos la saturación de las variables de control aplicando la política de *RM*, vemos como la utilización Um se satura al 100% al igual que la planificación *EDF*. La razón de pérdida se satura en cero como se muestra en la figura 5.2, cuando la utilización total estimada Ue se encuentra por debajo del 80%, a diferencia de la planificación *EDF* en donde el límite es 100%. La pérdida de plazos ocurre cuando la utilización actual total alcanza el 80%. Las gráficas muestran que la saturación de la utilización Um se obtiene aproximadamente al 100% bajo las dos políticas, la diferencia principal es que bajo *EDF* la razón de pérdida se presenta alrededor del 100% de la utilización total estimada, mientras que con *RM* la pérdida de plazos de respuesta de las tareas comienza

un poco antes entre el 80% y 90%. Una vez que el sistema se encuentra sobrecargado ($Um > 100\%$), la planificación *RM* tiene un mejor comportamiento (menor pérdida de plazos de repuesta en las tareas) en comparación con la política *EDF*.

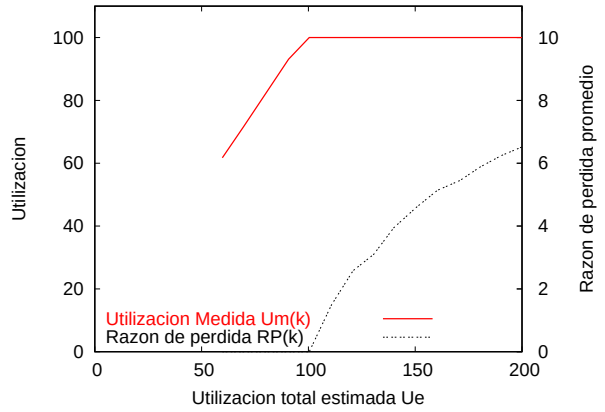


Figura 5.1: Saturación en planificación EDF

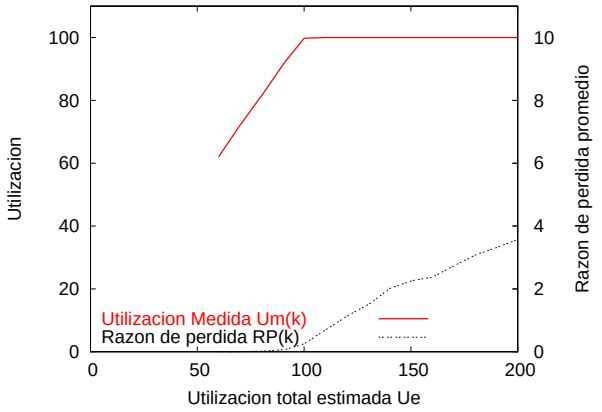


Figura 5.2: Saturación en planificación RM

5.4 Evaluación de Resultados

En esta sección presentamos los resultados de los tres tipos de controles del planificador de tareas en tiempo real con restricciones de energía en retroalimentación: *control de utilización, control de razón de pérdida y control integrado de utilización + razón de pérdida.* Los resultados de las simulaciones muestran el desempeño del sistema (ahorro de energía y pérdida de plazos), utilizando las políticas de planificación *EDF* y *RM* y el algoritmo aleatorio y EGA implementados en el variador de voltaje (sección 4.7).

La tabla 5.1 muestra la configuración del sistema utilizada en las simulaciones, en donde las ganancias del controlador K_p y K_i son iguales para ambos controladores (utilización y razón de pérdida). La referencia de control U_r se fija al 95% para el control de utilización, debido a que se desea maximizar la utilización del procesador sin sobrepasar el límite máximo del 100%. La referencia para el control de razón de pérdida RP_r utilizado es de 0.01%. El objetivo de esta referencia es minimizar el número de plazos de respuesta perdidos. Durante

	Control Utilización Um	**Control Razón de Pérdida** RP	**Control integrado** Um+RP
K_p	0.5	0.5	0.5
K_i	0.1	0.1	0.5
Referencias	$U_r = 95\%$	$RP_r = 0.01\%$	$U_r = 95\%$ y $RP_r = 0.01\%$
Periodo de muestreo	0.5 seg.	0.5 seg.	0.5 seg.

Tabla 5.1: Configuración de los algoritmos

cada periodo de muestreo W se activa el lazo de control retroalimentado, con un periodo de muestreo de $W = 0.5$ segundos.

Por cada simulación se presentan los resultados mediante dos gráficas. La primer gráfica muestra el desempeño del control, mediante la monitorización de la utilización medida $Um(k)$ (utilización obtenida en base al tiempo real ocupado por el procesador sobre la ventana de tiempo W) y la utilización estimada Ue (utilización calculada con los tiempos de ejecución estimados de las tareas). También se muestra la variación del número de tareas que se encuentran en el sistema (carga). En estas gráficas se presenta el porcentaje de utilización del procesador (Um y Ue) y el número de tareas con respecto al tiempo.

En la segunda gráfica se muestra la señal de control, que en este caso es la utilización requerida $\Delta U(k)$, y la variable controlada $RP(k)$ (porcentaje de plazos de respuesta perdidos), esta última es utilizada por el control de razón de pérdida y por el control de utilización-razón de pérdida, el eje y de esta gráfica representa la utilización del procesador (normalizada entre 0-1) y la razón de plazos de respuesta perdidos.

En las siguientes secciones se describirán los resultados obtenidas de las simulaciones, probando las diferentes configuraciones de control planteadas en la sección 4.6 y los dos algoritmos implementados en el variador de voltaje descritos en la sección 4.7. En la figura 5.3 se muestra la distribución de los resultados de las simulaciones realizadas. En esta figura, se indican las secciones y las figuras en las cuales se describen los resultados con más detalle.

Planificacion EDF	Algoritmo Aleatorio seccion 5.4.1	Control Um Figura 5.3(a) y 5.3(b)
		Control RP Figura 5.3(c) y 5.3(d)
		Control Um/RP Figura 5.3(e) y 5.3(f)
	Algoritmo EGA seccion 5.4.2	Control Um Figura 5.4(a) y 5.4(b)
		Control RP Figura 5.4(c) y 5.4(d)
		Control Um/RP Figura 5.4(e) y 5.4(f)
Planificacion RM	Algoritmo Aleatorio seccion 5.4.3	Control Um Figura 5.5(a) y 5.5(b)
		Control RP Figura 5.5(c) y 5.5(d)
		Control Um/RP Figura 5.5(e) y 5.5(f)
	Algoritmo EGA seccion 5.4.4	Control Um Figura 5.6(a) y 5.6(b)
		Control RP Figura 5.6(c) y 5.6(d)
		Control Um/RP Figura 5.6(e) y 5.6(f)

Figura 5.3: Distribución de los resultados

5.4.1 Controladores con Planificación EDF y Algoritmo Aleatorio

En esta sección veremos el desempeño del sistema utilizando la política de planificación EDF y las tres diferentes configuraciones de control: control de utilización, control de razón de pérdida y el control combinado de utilización y razón de pérdida. Los resultados que se presentan en esta sección muestran el comportamiento del sistema en tiempo real, con el algoritmo aleatorio implementado en el variador de voltaje (sección 4.7.1).

Control de utilización

El desempeño del sistema con control de utilización bajo la política *EDF* y con el algoritmo aleatorio se ilustra en las figuras 5.4(a) y 5.4(b). En el inciso (a) de la figura 5.4 se muestra la utilización total estimada $Ue(k)$, la variable de control $Um(k)$ y la variación del número de tareas n. Se observa como aproximadamente a los 5 segundos de iniciada la simulación se

presenta una pequeña saturación al 100% causado por una sobrecarga debida a las variaciones de los tiempo de ejecución y al arribo de nuevas tareas. La utilización estimada $Ue(k)$ indica la suma de las utilizaciones de las tareas en base a la estimación en sus tiempo de ejecución y la utilización medida $Um(k)$ representa la utilización total tomando en cuenta las variaciones en los tiempos de ejecución de las tareas.

La utilización $Um(k)$ permanece estable cercano a la referencia U_r=95% durante el transcurso de la simulación presentando ligeros picos al arribo de nuevas tareas, por ello las pérdidas de plazos que pudieran ocurrir se deben a estos sobrepasos, los cuales se presentan durante los intervalos entre periodos de muestreo, en este caso se observa como no se sobrecarga el sistema durante la simulación ($Um(k) < 100\%$).

La figura 5.4(b) muestra la señal de control $\Delta U(k)$ y la razón de pérdida de plazos $RP(k)$. En esta gráfica vemos como la razón de pérdida permanece sin cambio $RP(k) = 0$ durante la simulación debido a que no se presenta sobrecarga alguna en el sistema ($Um(k) < 100\%$), por lo que no se presentan pérdida de plazos de respuesta de 12798 instancias posibles, según los resultados de la simulación.

La señal de control $\Delta U(k)$ representa la salida del controlador *PID* conocida también como señal de corrección, esta señal nos muestra la reacción del controlador a cargas variables y ruidos que se presenten en el sistema.

Control de razón de pérdida

Al igual que el control de utilización, el desempeño del sistema controlado por razón de pérdida bajo la política *EDF* y el algoritmo aleatorio implementado en el variador de voltaje se presenta en las figuras 5.4(c) y 5.4(d). La primer gráfica muestra la utilización actual medida $Um(k)$ y la utilización total estimada $Ue(k)$, la segunda gráfica presenta el comportamiento de la señal de control $RP(k)$ y de la utilización requerida $\Delta U(k)$.

La variable a controlar en esta configuración es la razón de pérdida $RP(k)$, la referencia se fija en $RP_r = 0.01\%$ como se indica en la tabla 5.1. En la figura 5.4(d) vemos como el controlador intenta estabilizar $RP(k)$ para alcanzar la referencia RP_r durante el transcurso de la simulación. En la figura 5.4(c) se observa que aproximadamente a la mitad de la simulación

(30 segundos) la variable $Um(k)$ se satura al 100% debido a que el sistema requiere de una mayor utilización $\Delta U(k)$ y presenta un incremento en la razón de pérdida $RP(k)$ como se muestra en la figura 5.4(d).

En esta configuración se presentan más plazos de respuesta perdidos, comparado con el control de utilización (90 plazos perdidos de 15172 instancias ejecutadas). Como la variable a controlar es precisamente la razón de pérdida $RP(k)$, es normal que se pierdan plazos de respuesta, ya que la variable a control RP intenta estabilizarse en la referencia RP_r.

Control de utilización-razón de pérdida

Esta configuración implementa los dos tipos de control, control de utilización y control de razón de pérdida, para más detalles ver sección 4.6. Al contar con dos controladores la salida de control $\Delta U(k)$ se selecciona tomando el mínimo valor de los dos controladores, de esta forma las características más importantes de los dos se presentan en este tipo de control.

El comportamiento de las señales de control se muestran en las figuras 5.4(e) y 5.4(f). En el inciso (e) se observa como durante los primeros 30 segundos el comportamiento del sistema es parecido al comportamiento utilizando el control de razón de pérdida $RP(k)$, durante los últimos 30 segundos el sistema intenta estabilizar la utilización $Um(k)$ a la referencia 95% ya que el control de utilización entra en acción durante los últimos 30 segundos. En esta configuración se presentan 52 plazos de respuesta perdidos de un total de 15785 instancias ejecutadas.

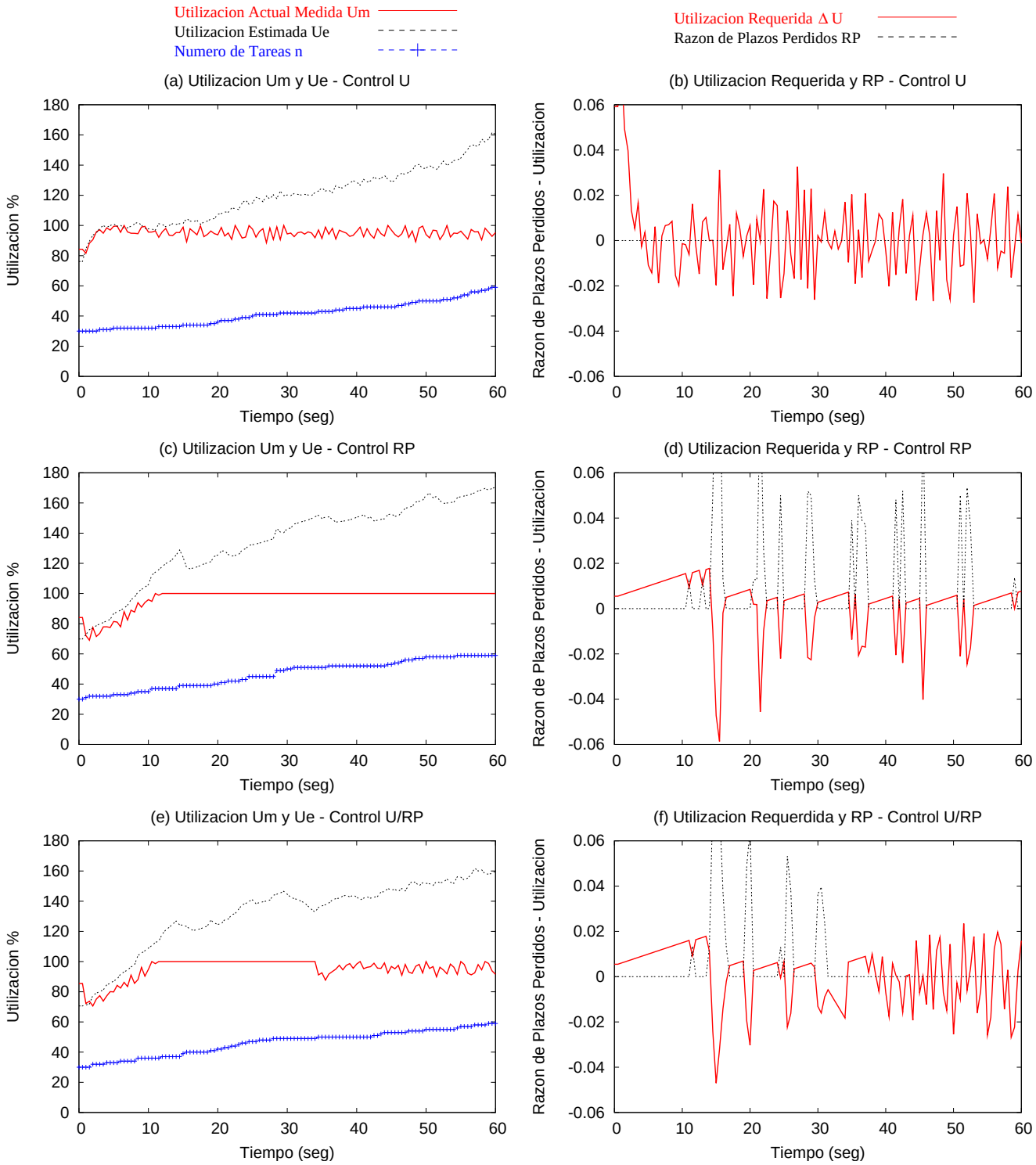


Figura 5.4: Control de carga y señales de control, configuración EDF y Algoritmo aleatorio

5.4.2 Controladores con Planificación EDF y Algoritmo EGA

En esta sección al igual que la anterior, veremos el desempeño del sistema utilizando las tres configuraciones de control y con el algoritmo *EGA* (sección 4.7.2) implementado en el variador de voltaje.

Control de utilización

Los incisos (a) y (b) de la figura 5.5 muestran el desempeño del sistema utilizando el control de utilización bajo la planificación EDF. El inciso (a) de esta figura muestra un comportamiento de $Um(k)$ similar al control de utilización con el algoritmo aleatorio, la diferencia principal, es que con el algoritmo *EGA* se presenta una estabilización mayor entre arribo de tareas en comparación con la estabilización del sistema del algoritmo aleatorio. El inconveniente con este algoritmo es que al arribo de una nueva tarea se presenta una pequeña sobrecarga en el procesador ($Um(k) > 100\%$), provocando una ligera pérdida de plazos (69 - 12314), como se muestra en el inciso (b) de la figura 5.5, en donde se observan pequeños sobresaltos en la razón de plazos perdidos $RP(k)$.

Control de razón de pérdida

El desempeño del sistema utilizando control de razón de pérdida bajo la política EDF y el algoritmo EGA se presenta en los incisos (c) y (d) de la figura 5.5. Vemos como en el inciso (c), la señal de control $Um(k)$ se satura aproximadamente a los 30 segundos de haber iniciado la simulación, tiempo mayor a los 10 segundos que tomó el mismo control con el algoritmo aleatorio, esto demuestra que la acción de control utilizando el algoritmo EGA es más lenta que la acción de control utilizando el algoritmo aleatorio.

En el inciso (d) de la figura 5.5, observamos como la razón de plazos perdidos $RP(k)$ intenta estabilizarse en 0.01 después de que ocurre la saturación de $Um(k)$, aproximadamente a los 30 segundos. Con esta configuración se presentaron 90 plazos de respuesta perdidos de un total de 15404 posibles en la simulación.

Control de utilización-razón de pérdida

Los incisos (e) y (f) de la figura 5.5 muestran las señales de control $Um(k)$ y $RP(k)$ para el sistema controlado por utilización y razón de pérdida, las cuales el sistema configurado de esta manera intenta mantener lo más cercana posible a las referencias U_r y RP_r, descritas en la tabla 5.1.

En el inciso (e), vemos como el control por utilización toma acción durante los primeros 30 segundos de la simulación y durante un pequeño lapso alrededor de los 50 segundos; mientras que el control de razón de pérdida toma acción sobre el sistema en el tiempo restante de la simulación. El inciso (f) muestra como la señal $RP(k)$ sobrepasa más la referencia 0.01 que el mismo tipo de control pero con el algoritmo aleatorio. Con esta configuración se presentaron 94 plazos perdidos de 14652 instancias de tareas ejecutadas durante la simulación.

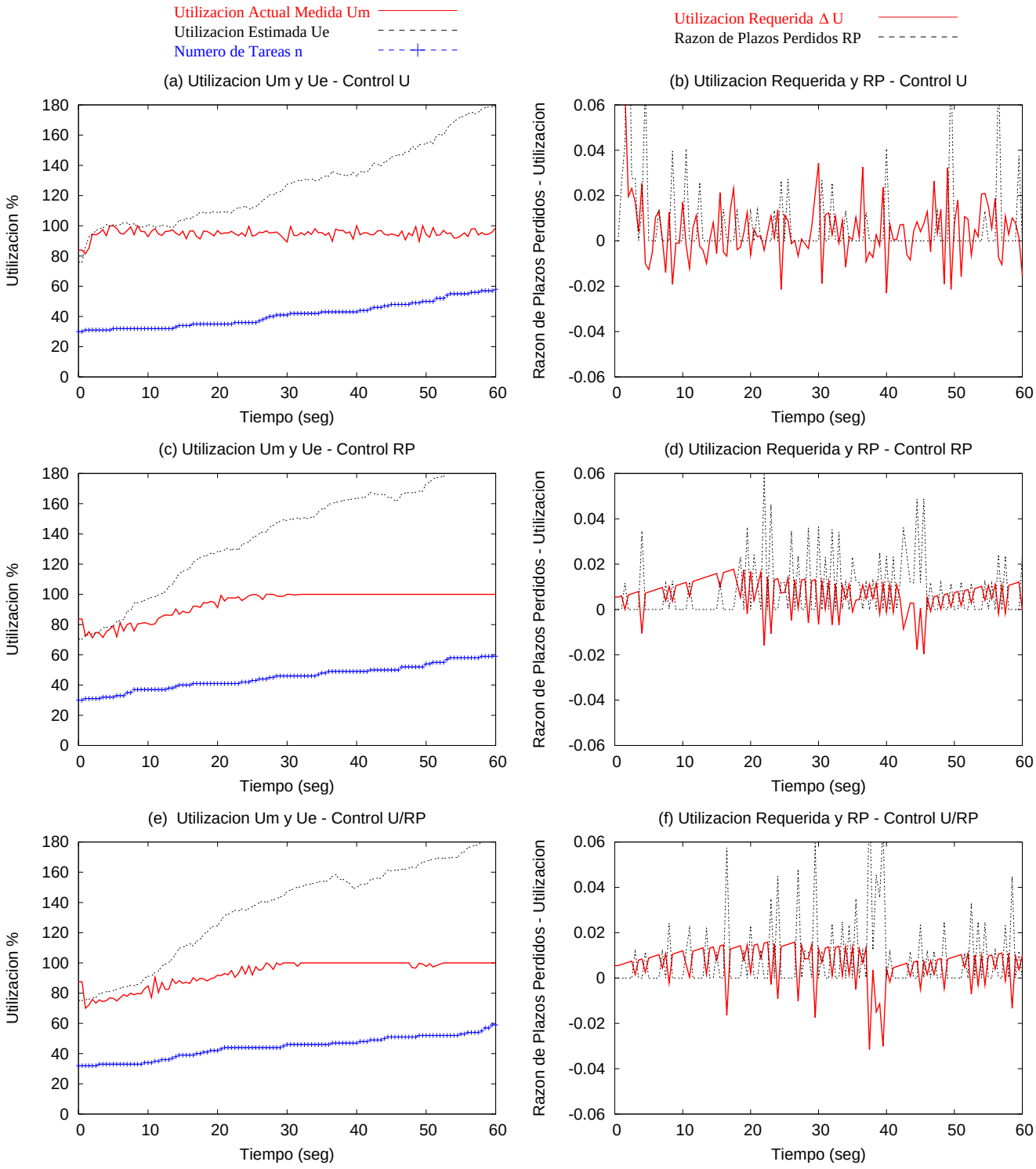


Figura 5.5: Control de carga y señales de control, configuración EDF y Algoritmo EGA

5.4.3 Controladores con Planificación RM y Algoritmo Aleatorio

En esta sección, al igual que en la anterior, comparamos el desempeño del sistema bajo las tres diferentes configuraciones de control y utilizando el algoritmo aleatorio en el variador de voltaje. Esta vez la planificación se realiza mediante la política *Rate Monotonic RM*.

Control de utilización

La figura 5.6 incisos (a) y (b), muestran el desempeño del sistema cuando se utiliza el control de utilización bajo la política RM.

En el inciso (a) de la figura, vemos que la variable de control $Um(k)$ tiene un comportamiento aceptable, ya que nunca sobrepasa el 100% de utilización. Por otro lado, en el inciso (b), vemos que la señal razón de pérdida $RP(k)$ presenta varios sobresaltos durante el transcurso de la simulación, esto debido a que el control intenta regular la utilización $Um(k)$ al 95% (tabla 4.2). Sin embargo, como la $Um(k)$ se encuentra por encima de la U_{lsm} (límite superior máximo de utilización) (sección 2.7), es posible que se presenten pérdida de plazos. Con esta configuración, se presentaron 33 plazos de respuesta perdidos de un total de 13496 posibles.

Control de razón de pérdida

La utilización medida $Um(k)$ y la estimada $Ue(k)$ del sistema, bajo la configuración de control por $RP(k)$ se muestra en los inciso (c) y (d) de la figura 5.6. Comparado con las gráficas de control por $RP(k)$ con EDF, la utilización $Um(k)$ se mantiene al borde de la saturación, casi al 100%, esto debido a que el sistema intenta regular $RP(k)$ al 0.01% (figura 5.6(d)), es decir, que al registrarse una cierta cantidad de plazos perdidos, el sistema, intenta regular $RP(k)$, mientras que $Um(k)$ se mantiene cerca del U_{lsm} , el cual para RM se encuentra por debajo del 100%. Se presentaron 106 - 16256 plazos perdidos con esta configuración.

Control de utilización-razón de pérdida

El desempeño de esta configuración se muestra en la figura 5.6 incisos (e) y (f). Observamos como el comportamiento es muy similar al control de $RP(k)$, pero con menor razón de pérdida (inciso (f)). Se presentaron 77 plazos perdidos de un total de 15953 posibles.

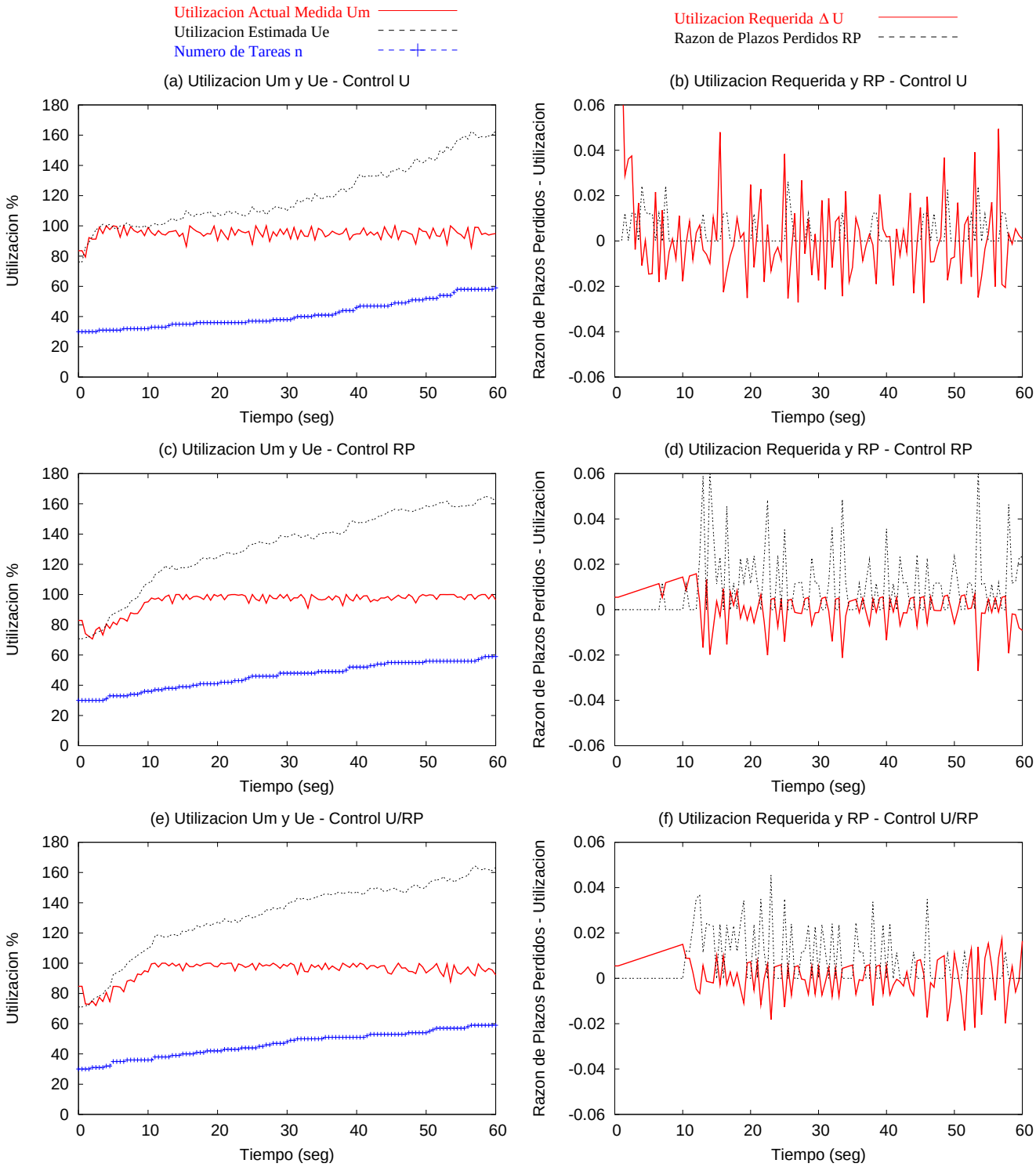


Figura 5.6: Control de carga y señales de control, configuración RM y Algoritmo aleatorio

5.4.4 Controladores con Planificación RM y Algoritmo EGA

Por último, se describirá el comportamiento del sistema cuando se utiliza el algoritmo EGA en el variador de voltaje y la política de planificación RM, utilizando las tres configuraciones de control propuestas.

Control de utilización

La figuras 5.7(a) y 5.7(b), muestran el desempeño del sistema controlado mediante la utilización $Um(k)$, haciendo uso de la política de planificación RM y del algoritmo EGA.

Prácticamente, el algoritmo EGA presenta el mismo comportamiento de la variable de control $Um(k)$ que el algoritmo aleatorio 5.6(a), la diferencia se presenta en la razón de pérdida $RP(k)$ (inciso (b)), ya que se presentan 155 plazos de respuesta perdidos de 14958 posibles. La ventaja del algoritmo EGA se obtiene en los porcentajes de ahorro de energía consumida por el procesador. Los porcentajes de ahorro para todos los algoritmos y configuraciones se presentarán en la sección 5.4.6.

Control de razón de pérdida

En los incisos (c) y (d) de la figura 5.7, se muestra el comportamiento de las variables $Um(k)$ y $RP(k)$ con respecto a la variación de la carga de trabajo. El inciso (c) muestra como la variable $Um(k)$ se mantiene al margen de la saturación 100%, mostrando similitud con el comportamiento de $Um(k)$ del algoritmo aleatorio, la diferencia estriba en que la señal $Um(k)$ tiene un mayor tiempo de asentamiento con este algoritmo que con el aleatorio. Se presentaron 81 plazos de respuesta perdidos de un total de 14722 instancias ejecutadas.

Control de utilización-razón de pérdida

Por ultimo, presentamos el desempeño obtenido de las simulaciones con la configuración RM y control unificado $Um(k)$ y $RP(k)$. Al igual que la respuesta de las señales $Um(k)$ y $RP(k)$ obtenidos con el algoritmo aleatorio implementado en el variador de voltaje, el comportamiento con el control unificado es similar al control de razón de pérdida $RP(k)$. Se presentaron 88 plazos de respuesta perdidos de 15113 totales posibles.

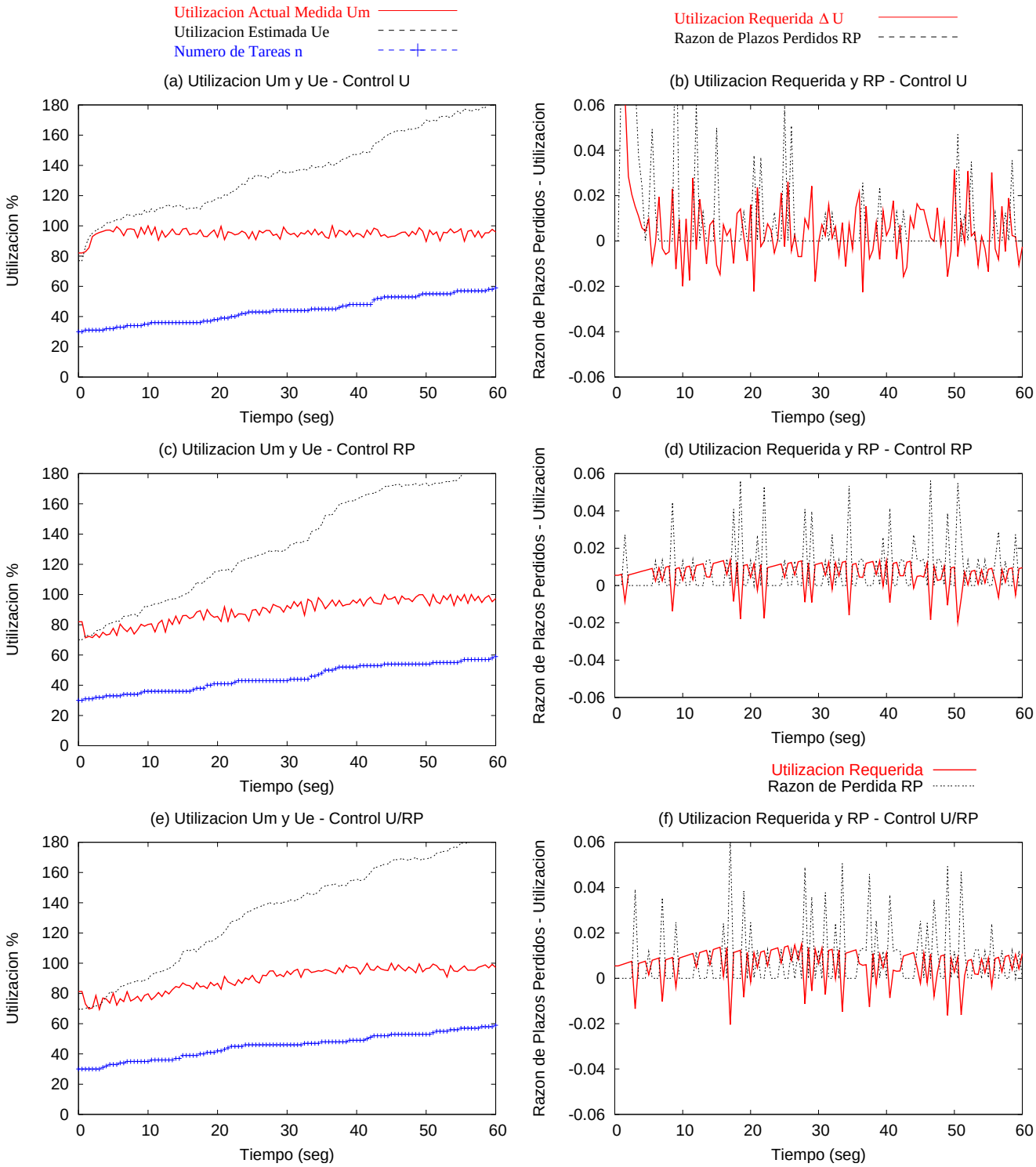


Figura 5.7: Control de carga y señales de control, configuración RM y Algoritmo EGA

5.4.5 Sin Control o Lazo Abierto

En comparación con los algoritmos retroalimentados descritos en las secciones anteriores, el sistema en lazo abierto planifica el conjunto de tareas sin cambiar la velocidad de ejecución de esas tareas, es decir no modifica la utilización total $Ue(k)$ ya que no cuenta con la retroalimentación que intente controlar algún parámetro del sistema ($Um(k)$ o $RP(k)$).

Para comprobar la ineficiencia del algoritmo en lazo abierto, probamos el sistema utilizando el algoritmo EGA en el variador de voltaje con una capacidad del *Knapsack* constante del 100%, el controlador y el monitor se mantienen desactivados durante la simulación. Esta configuración satura a $Um(k)$ casi inmediatamente después de iniciada la simulación, como se muestra en las figuras 5.8(a) y 5.8(c) utilizando las políticas EDF y RM respectivamente.

En los incisos (b) y (d) de la figura 5.8 vemos como se presentan altos niveles de plazos de respuesta perdidos $RP(k)$ tanto en EDF como en RM. Estas pérdidas son causadas principalmente por el aumento en la carga del sistema (número de tareas).

Estos resultados demuestran que un sistema de tiempo real en lazo abierto es incapaz de mantener un desempeño satisfactorio enfrentando severas variaciones en la carga.

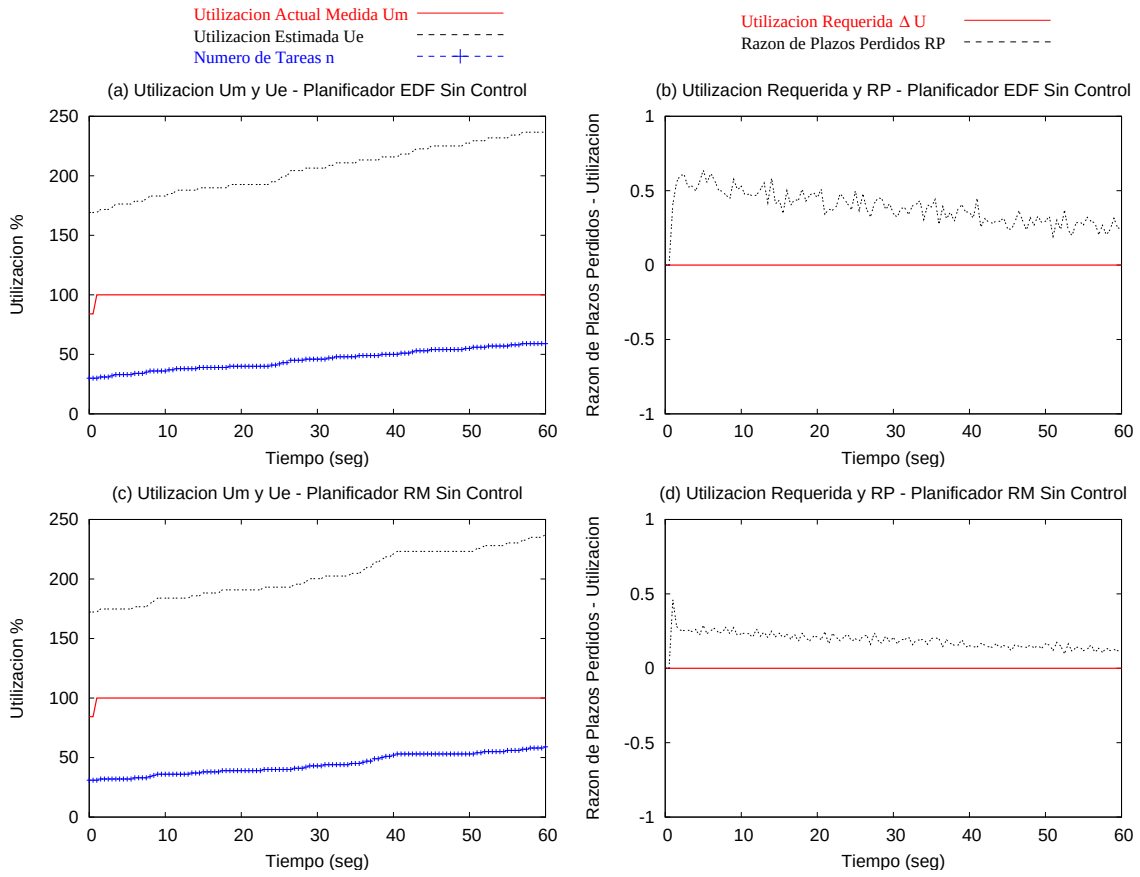


Figura 5.8: Control de carga y señales de control, configuración en lazo abierto

5.4.6 Medición de Energía y Pérdida de Plazos

En la sección 1.4 del capitulo 1, vimos que los objetivos principales del sistema propuesto son: (1) maximizar el ahorro de energía consumida por el procesador, (2) minimizar el número de plazos de respuesta que se puedan perder. Por lo tanto, es necesario tener unas métricas que nos permitan comparar el desempeño de las distintas configuraciones del sistema propuesto.

Una de estas métricas es el porcentaje de ahorro de energía del procesador $\%\epsilon$. Vimos en el capitulo 1 que el consumo de energía ϵ para cada tarea τ_i está dada por la ecuación 1.4. Esta ecuación nos indica que la energía consumida por una tarea es proporcional a su tiempo de ejecución actual multiplicado por la velocidad V_{ij} a la que se ejecuta al cuadrado.

Para obtener los porcentajes de ahorro de energía, obtenemos la energía consumida ϵ_{ij}

por cada una de las tareas τ_i al nivel de velocidad j con la que se ejecuta en el procesador: $\epsilon_c = \sum_{i=1}^{n} \epsilon_{ij}$. Y calculamos cuanta energía podría haberse consumido si esa misma tarea se ejecutara a la máxima velocidad: $\epsilon_m = \sum_{i=1}^{n} \epsilon_{i1}$, con estos dos valores de energía obtenemos el porcentaje de ahorro de energía $\%\epsilon$, dado por:

$$\%\epsilon = \frac{1 - \epsilon_c}{\epsilon_m} * 100 \tag{5.1}$$

Por otro lado, la razón de pérdida es una forma de medir la pérdida de plazos en razón al número de instancias ejecutadas durante una ventana de tiempo, otra forma de demostrar el desempeño de un sistema en tiempo real es conocer el número total de plazos de respuesta perdidos en comparación con el número total de posibles instancias durante la simulación.

La siguientes tablas muestra los niveles de ahorro de energía $\%\epsilon$ y las pérdidas de plazos totales de las diferentes configuraciones (control de Um, control de RP y control de Um y RP), utilizando las políticas *EDF* y *RM*.

	Algoritmo Aleatorio		**Algoritmo EGA**	
Tipo de Control	**Plazos perdidos / Plazos totales**	**Ahorro de energía** $\%\epsilon$	**Plazos perdidos / Plazos totales**	**Ahorro de energía** $\%\epsilon$
Um	0 / 12798	32.83%	69 / 12314	42.70%
RP	90 / 15172	36.66%	90 / 15404	41.48%
Um + RP	52 / 15785	33.49%	94 / 14652	40.09%
Sin control	ND	ND	3894 / 15259	73.38%

Tabla 5.2: Ahorro de energía y plazos perdidos totales bajo EDF

En la tabla 5.2 se observa que el algoritmo aleatorio obtiene resultados ligeramente mejores en cuanto al número de plazos perdidos, en comparación con el algoritmo EGA. En cambio, el algoritmo EGA presenta mejores resultados en lo correspondiente al porcentaje de ahorro de energía. Con el algoritmo EGA obtenemos casi más del 10% en el control de utilización Um, aproximadamente 5% más de ahorro con el control de razón de pérdida RP y 6.5% más con el control combinado Um y RP.

	Algoritmo Aleatorio		**Algoritmo EGA**	
Tipo de Control	**Plazos perdidos / Plazos totales**	**Ahorro de energía** $\%\epsilon$	**Plazos perdidos / Plazos totales**	**Ahorro de energía** $\%\epsilon$
Um	33 / 13496	33.32%	115 / 14958	43.08%
RP	106 / 16256	34.37%	81 / 14733	35.58%
Um + RP	77 / 15953	32.85%	88 / 15113	36.22%
Sin control	ND	ND	1897 / 14852	74.03%

Tabla 5.3: Ahorro de energía y plazos perdidos totales bajo RM

La tabla 5.3, muestra los resultados de los porcentajes de ahorro y el número de plazos perdidos para las diferentes configuraciones bajo la política RM. Al igual que con la política EDF, estos resultados muestran que al utilizarse el algoritmo EGA en el variador de voltaje se presentan mejores resultados en los porcentajes de ahorro de energía, en comparación con el algoritmo aleatorio.

La diferencia en los niveles de ahorro entre las configuraciones control de Um y control de RP, se debe a que la primera intenta mantener una utilización medida al 95%, es decir, intenta ocupar el mayor tiempo posible al procesador. Lo cual nos indica que el sistema alarga las tareas (disminuyendo las velocidades de ejecución), resultando en un mayor ahorro de energía. Mientras que el control de RP intenta mantener la razón de plazos perdidos al 0.01%, esto provoca que la utilización se mantenga al margen de la utilización U_{lsm}, la cual depende de la política de planificación (sección 2.6). Es por esto, que el ahorro de energía es mayor en el control de Um que en el control de RP, bajo las políticas (EDF, RM) y utilizando el algoritmo EGA.

Las tablas 5.2 y 5.3 también muestran que los sistemas en lazo abierto se comportan ineficientemente cuando la carga de trabajo varía constantemente. En la tabla 5.2 se observa como en el renglón de *sin control* se obtienen 3894 plazos de respuesta perdidos de un total de 15259 instancias ejecutadas. En la tabla 5.3 se observa que con RM en configuración en lazo abierto se obtienen 1897 plazos de respuesta perdidos de un total de 14852 instancias

totales. Los cuales podrían causar pérdidas económicas o inclusive humanas en un sistema de tiempo real crítico.

Estos resultados nos indican que cuando el sistema se encuentra saturado bajo EDF, el desempeño decae rápidamente, presentando 3894 plazos perdidos en comparación con los 1897 plazos de respuesta perdidos bajo RM.

5.5 Resumen

En este capitulo se presentó la estructura del código desarrollado para el simulador, así como los parámetros generados para las tareas de tiempo real (tiempo de cómputo estimado y actual, periodo de activación y tiempos de arribos). Se observo el comportamiento de saturación de las variables de control $Um(k)$ y $RP(k)$.

También se presentaron los resultados gráficos de las simulaciones utilizando las diferentes configuraciones de control: control de utilización Um, control de razón de pérdida RP, control combinado de Um-RP y sin control. Estos resultados muestran el desempeño del control en el sistema en tiempo real utilizando las dos políticas de planificación EDF y RM, además de los algoritmos aleatorio y EGA implementados en el variador de voltaje.

Por último, se presentaron los resultados obtenidos de las simulaciones con respecto al porcentaje de ahorro de energía $\%\epsilon$ consumido por el procesador y a la pérdida de plazos de respuesta totales, en donde claramente se observa que se obtienen mejores resultados en el consumo de energía cuando se utiliza el algoritmo EGA en el variador de voltaje, en comparación con el algoritmo aleatorio.

Estos resultados no muestran también, que el control por utilización Um presenta mejores resultados en cuanto al ahorro de energía, comparados con el control de RP y $U + RP$, mientras que los plazos de respuesta perdidos se mantiene aproximadamente igual en las tres configuraciones.

Capítulo 6

Conclusiones y Trabajo a Futuro

6.1 Conclusiones

La reducción en el consumo de la energía en los sistemas de cómputo ha sido un tema que recientemente ha atraído la atención de un numerosos grupos de investigación en el mundo. Esto debido a la gran proliferación de equipos de cómputo embebido, portátiles e inalámbricos basados en baterías. La reducción de consumo de energía en este tipo de sistemas permite reducir la disipación de calor, reducir el tamaño de las baterías y lo más importante, extender el tiempo de vida de las baterías..

En este trabajo de tesis, se presentó una arquitectura de planificación de sistemas de tiempo real con retroalimentación, para un sistema operativo en tiempo real, que permite reducir el consumo de la energía en sistemas dinámicos, en donde la carga de trabajo del procesador varía constantemente.

La arquitectura de planificación propuesta, permite ajustar la carga del sistema de forma que se cumplan los objetivos de minimizar el consumo de la energía en el sistema y minimizar el número de plazos de respuesta pérdidos. El ajuste de la carga de trabajo se logra mediante la introducción de un lazo de control con retroalimentación en la arquitectura.

Para lograr nuestros objetivos, aplicamos una metodología de control para el diseño de sistemas de tiempo real. Esta metodología nos permitió diseñar un planificador adaptable, establecer un primer modelo matemático simplificado del sistema de tiempo real a controlar, para satisfacer las especificaciones de desempeño del sistema mediante métodos analíticos. En el planificador adaptable, se diseñaron distintos algoritmos de planificación con retroalimentación capaces de alcanzar kas especificaciones de desempeño planteadas. En estos algoritmos se utilizó un lazo de retroalimentación para controlar la utilización, la razón de plazos pérdidos, y la combinación de la utilización y la razón de plazos perdidos.

Los resultados alcanzados utilizando nuestra arquitectura de planificación con retroalimentación nos permite concluir lo siguiente:

- **Se estableció una metodología de diseño para sistemas de control de tiempo real.** En este trabajo aplicamos la teoría de control para desarrollar la planificación para sistemas de tiempo real adaptables. En contraste con los algoritmos de planificación existentes que utilizan un control en lazo abierto, aplicamos una planificación con retroalimentación para sistemas de tiempo real que operan sobre ambientes impredecibles.

- **Se cumplieron los objetivos de optimización planteados dentro del sistema de control.** Los objetivos de optimización (maximizar el ahorro de la energía y minimizar el número de plazos perdidos) se cumplieron enfrentando cargas de trabajo de tiempo real altamente dinámicas. Este objetivo se cumplió con la introducción en la arquitectura de un mecanismo para el ajuste del voltaje (variador de voltaje). Este mecanismo recibe como entrada la salida del controlador, y permite ajustar la carga del procesador mediante un incremento o decremento de las velocidades de ejecución de las tareas (con el consecuente aumento o decremento de la utilización del sistema)

- **Se mejoraron los mecanismo de planificación existentes de sistemas de tiempo real para sistemas dinámicos.** La arquitectura planteada, mejora el comportamiento de los algoritmos anteriormente desarrollados, los cuales están basados en

planificación en lazo abierto. El algoritmo de control que mejor desempeño alcanzó, fue el algoritmo de control de utilización. La arquitectura planteada permite incluir distintos tipos de controladores y ajustar las referencias de desempeño para alcanzar mejores niveles de desempeño. Así mismo, nos permite incluir distintas políticas de planificación de tareas en tiempo real.

6.2 Trabajo futuro

El trabajo a futuro necesariamente tiene que considerar la implementación de los algoritmos planteados en esta tesis dentro de una plataforma de cómputo real. Esta plataforma de cómputo debe contener procesadores de velocidad variable, tales como el *Crusoe* de *Transmeta*, y la ejecución de la arquitectura sobre un sistema operativo de tiempo real. Es claro que los resultados de las simulaciones realizadas en este trabajo de tesis, entrega información parcial acerca del desempeño, ya que no se cuentan con datos de entrada reales. Además, existen otros aspectos de la simulación que no se tomaron en cuenta para el modelado, como los tiempo entre cambios de voltaje, los tiempos de cambio de contexto, etc. Es necesario mejorar más el modelo del sistema para tomar en cuenta estos aspectos no modelados, así como para proponer un análisis de estabilidad del sistema de tiempo real con retroalimentación.

Una vez implementado los mecanismo desarrollados en esta tesis sobre una plataforma de hardware con un sistema operativo de tiempo real, es conveniente también incluir al sistema lo siguiente:

- **Mecanismos de monitorización y visualización de los parámetros del sistema.** No es posible medir la eficiencia de los mecanismos de control si no se cuenta con herramientas que permitan medir el consumo de la energía, el desempeño del sistema, y la ejecución del sistema en tiempo real. Si el objetivo del sistema es maximizar el ahorro de energía debe ser posible entonces medir con precisión el consumo, a fin de medir que tan eficiente son los algoritmos diseñados. Por otro lado, debe ser posible medir en el sistema la utilización y el número de plazos perdidos. Todo esto debe ser

posible de realizarse sobre la plataforma real antes discutida.

- **Implementación de leyes de control adaptivas.** El sistema que estamos controlando es un sistema dinámico, en donde los parámetros temporales de las tareas cambian continuamente y no nos es posible predecir su comportamiento. En este trabajo de tesis, utilizamos técnicas de control convencionales que incluyen algoritmos de control PID. Otras técnicas de control que pueden ser implementadas en esta arquitectura, son los controladores adaptivos. Estos controladores permiten modificar las ganancias de los controladores a fin de hacer un ajuste a la medida para diferentes aplicaciones y ambientes de trabajo.

Apéndice A

Estándar para Manejo de Energía ACPI

El *ACPI* es un estándar ampliamente utilizado para el manejo de energía en sistemas de cómputo como son las laptops, las computadoras de escritorio y los servidores de cómputo. Este estándar fue desarrollado por los principales desarrolladores de software y hardware como *Intel*, *Microsoft*, *Compaq*, *Phoenix* y *Toshiba*. El *ACPI* permite que el sistema operativo configure directamente el manejo de la energía. Las especificaciones del estándar *ACPI* definen la mayoría de las interfaces entre el sistema operativo y el hardware. Los componentes de software y hardware del estándar *ACPI* se muestran en la figura A.1. Las aplicaciones interactúan con el kernel del sistema operativo a través de interfaces de programación conocidas como *API's (application programming interfaces)*. Un módulo del sistema operativo implementa las políticas de manejo de energía. Este módulo interactúa con el hardware a través de llamadas al sistema. El kernel a su vez, interactúa con el hardware mediante manejadores o *drivers* de dispositivos. La interfaz principal es el *driver ACPI*, el cual convierte las peticiones del kernel a comandos *ACPI*. En el *ACPI*, los componentes del sistema cuentan con cinco estados de potencia:

- **S0/Trabajando** - El procesador se encuentra funcionando, los dispositivos se encienden y se apagan según se necesite.

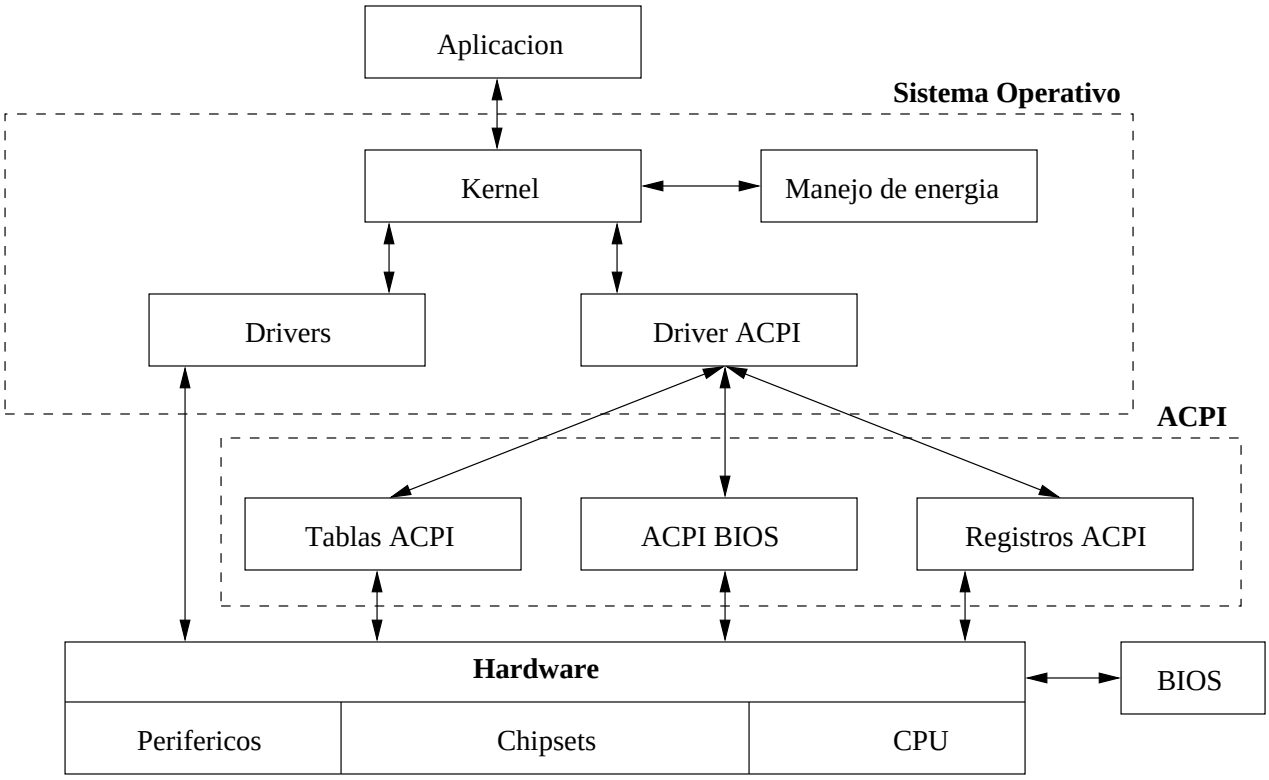


Figura A.1: Interfaz ACPI en un sistema de cómputo

- **S1** - El procesador se encuentra detenido, la memoria RAM se refresca; el sistema esta en un modo de baja potencia.
- **S2** - El procesador no cuenta con energía, la memoria RAM se refresca; el sistema esta en un modo de baja potencia menor que S1.
- **S3** - El procesador no cuenta con energía, la memoria RAM tiene una baja velocidad de refresco; la fuente de alimentación se encuentra en un modo de potencia reducida.
- **S4** - El hardware se encuentra completamente apagado, la memoria ha sido guardada en disco.
- **S5/Apagado** - El hardware se encuentra completamente apagado al igual que el sistema operativo, nada ha sido guardado.

Apéndice B

Programa de Simulación

B.1 Introducción

Este programa simula la arquitectura del *planificador de tareas en tiempo real con restricciones de energía en retroalimentación*. Los propósitos principales del simulador son calcular el consumo de energía de la arquitectura de planificación planteada en el capitulo 4, mostrar la perdida de plazos de respuesta que pudieran presentarse y generar los archivos necesarios para estudiar el comportamiento del sistema. Estos archivos tienen el formato de columnas separadas por comas y pueden presentarse en forma gráfica utilizando *gnuplot*.

El simulador esta desarrollado en lenguaje *C++* bajo ambiente Linux. Su estructura es modular para un fácil mantenimiento. Es posible realizar con facilidad modificaciones como agregar nuevas políticas de planificación, agregar nuevos algoritmos de control y cambiar los algoritmos de selección de tareas. En esta sección presentamos la instalación de simulador, la interfaz gráfica y su forma de uso y por último presentamos la estructura del código.

B.2 Instalación

El ambiente de programación del programa de simulación es *Linux*. Antes de la instalación es necesario obtener los archivos fuentes que se encuentran comprimidos en el archivo *simu-*

lador.tar.gz para descomprimir el archivo es necesario utilizar *GNU tar*:
tar -xvzf simulador.tar.gz

Una vez descomprimido el archivo, encontraremos 4 archivos en el mismo directorio donde se encuentra el archivo tar, estos archivos son:
Makefile
simulador.h
simulador.cpp
main.cpp

El archivo *Makefile* contiene las banderas y el nombre de las librerías necesarias para compilar los archivos fuentes. Para obtener el ejecutable simplemente invocamos el comando:
make

En el archivo *simulador.h* se encuentran declarados las constantes, estructuras y los prototipos de las funciones utilizadas en la clase principal *sim*. En el archivo *simulador.cpp* se encuentran la implementación de las funciones utilizadas en la clase principal, así como las declaraciones de las variables y su inicialización. En el archivo *main.cpp* se instancia un objeto de la clase *sim*. Aquí se encuentra el código que ejecuta la aplicación en la ventana principal.

Una vez compilado, obtenemos el programa ejecutable llamado *sim*. Para ejecutar la aplicación simplemente invocamos el comando
./sim

lo cual presenta la interfaz gráfica del simulador.

B.3 Interfaz Gráfica

La interfaz de usuario fue desarrollada utilizando *Q*t bajo ambiente *Linux Red-Hat 7.1.* Qt es una librería para el lenguaje C++ fácil de usar y bien documentada, actualmente la mayor parte de las aplicaciones gráficas que se ejecutan bajo Linux están desarrolladas con Qt, debido principalmente a que el ambiente gráfico *KDE* esta construido bajo Qt.

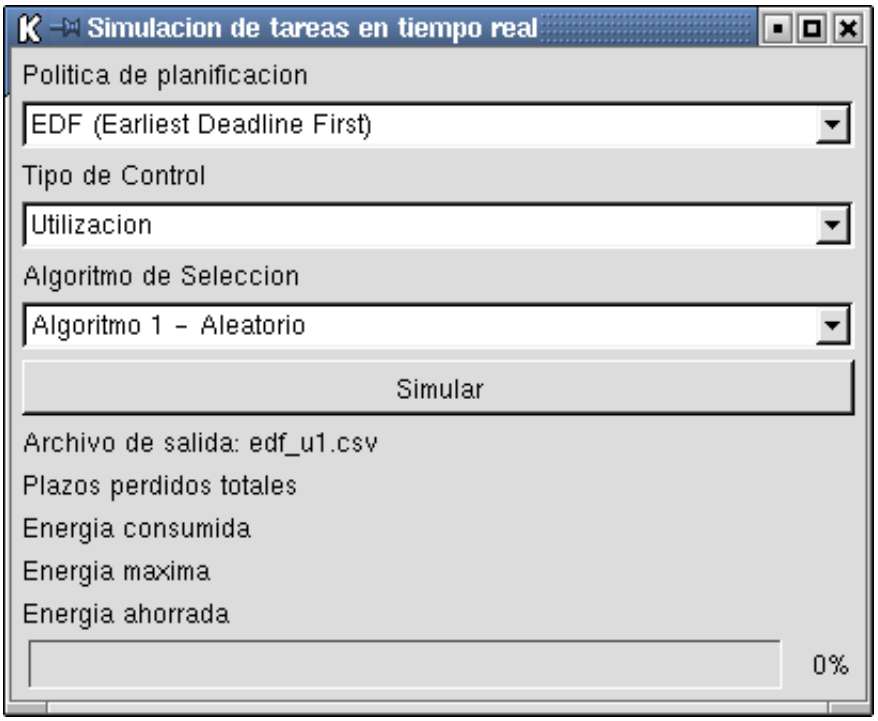


Figura B.1: Interfaz de usuario

La figura B.1 muestra la pantalla de la aplicación, en ella se puede observar los controles. En la parte superior se localiza un cuadro combinado donde seleccionamos la política de planificación que queremos utilizar para la simulación. En este sistema las opciones a seleccionar son EDF *(Earliest Deadline First)* o RM *(Rate Monotonic).*

Debajo del cuadro combinado de selección de las políticas de planificación, se encuentra el cuadro combinado donde seleccionamos el tipo de control que utilizaremos en el lazo de

control, este indica la variable de control que utilizaremos en el sistema. Las opciones para este cuadro combinado son:

- Control por *utilización*,
- Control por *razón de perdida*,
- Control por *utilización + razón de perdida sin control*.

Con el último cuadro combinado podemos escoger el algoritmo de selección de las velocidades de ejecución del conjunto de tareas, las opciones son:

- *algoritmo 1 - aleatorio*
- *algoritmo 2 - aproximación óptimo EGA*

Debajo de los cuadros combinados donde seleccionan la configuración del sistema se localiza el botón que inicia la simulación, marcado como *Simular*. Una vez iniciada la simulación el botón cambia su etiqueta por *parar*, presionando de nuevo el botón se detiene la simulación.

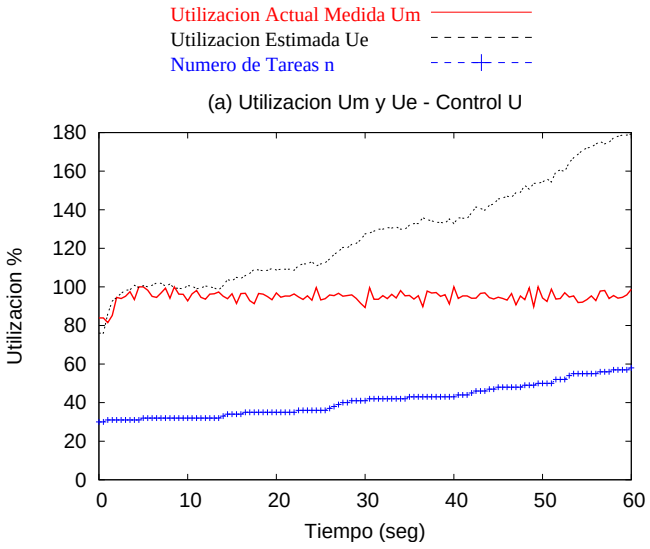


Figura B.2: Respuesta de la variable de control

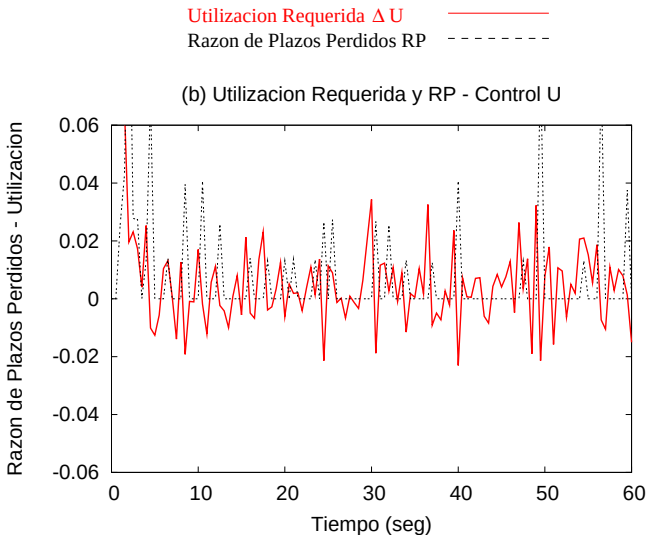


Figura B.3: Respuesta en la señal de control

En la parte inferior de la pantalla se localizan una serie de etiquetas, la primera indica el nombre del archivo de salida con extensión csv (columnas separado por comas), el nombre del archivo depende de las combinaciones de las opciones seleccionadas. Las etiquetas restantes muestran información relacionada al consumo de la energía al finalizar una simulación, como son la energía consumida, la energía máxima y la energía ahorrada. Por ultimo en la parte inferior se localiza la barra de progreso, que le indica al usuario el porcentaje de avance de la simulación.

Además de los resultados visuales presentados en la interfaz de usuario, se cuenta con los archivos de salida generados en forma de texto separados por comas. Para visualizar estos archivos en forma gráfica utilizamos *gnuplot*. el cual es un programa ampliamente utilizado en la generación de gráficas en *Linux*. La figura B.2 muestra un ejemplo de la respuesta en el tiempo de las señales de control, como son la utilización estimada (utilización basada en parámetros), la utilización medida (tiempo ocupado por el procesador) y la variación de la carga o número de tareas. La figura B.3 muestra un ejemplo de la señal de control o salida del controlador conocida como utilización requerida y la señal de razón de plazos perdidos.

B.4 Estructura del Código

Una parte importante de los sistemas de tiempo real es la configuración del reloj, debido a que en el podemos obtener la resolución del tiempo de muestreo (en micro segundos). En este caso al tratarse de una simulación utilizaremos un *timer* o temporizador para simular los ciclos del reloj, la resolución del temporizador esta dada en micro segundos, así que podemos esperar resultados cercanos a los reales.

Básicamente el programa gira en torno al evento del temporizador. A continuación se presenta el pseudocódigo del evento del temporizador el cual simula una duración de 1 minuto mediante 600,000 ciclos.

```
void Sim::timerEvent( ){
      if(ini == 1){
           //Planificador obtiene la próxima tarea a ejecución
           Planificador(ut);

           //Cada periodo de muestreo aplicar el control PID
           if((ut % periodo_muestreo)==0){
               monitor();
               PID(ut);
           }
           //Incrementa el tiempo simulado
           ut++;
      }
}
```

La variable *ini* se activa una vez que la simulación ha sido iniciada. La estructura del código del temporizador esta compuesta por dos partes principales. La primera parte es la encargada

de obtener la próxima tarea a ejecución, este orden depende de los parámetros generados por cada uno de las tareas y por la política de planificación utilizada para la simulación (EDF o RM). A continuación se presenta el pseudocódigo del planificador:

```
void Sim::Planificador(int tiempo){
     //Recorre cada una de las tareas
     for(i=1; i<=numero_tareas; i++){

     //Comprueba la perdida del plazo por cada tarea i
     perdida_plazo(i,tiempo);

     //Si se termina la ejecución de una instancia de la tarea i
     if(cola_listos[i].periodo_abs == tiempo){

          //Calcula el nuevo período absoluto
          //Reiniciar datos para la tarea i
          //Generar nuevos valores aleatorios en los tiempo de ejecución
     }

     //Checa la llegada de tareas aperiódicas

     //Obtiene la siguiente tarea a ejecutar y simula la ejecución
     index = obtener_tarea(cola_listos,tiempo);
     cola_listos[index].ejecución++;
    }
}
```

En cada periodo de muestreo k en el ciclo principal, ejecutamos el monitor y el controlador PID. El monitor mide las variables de control $RP(k)$ o $Um(k)$ que ocurren durante los periodos de muestreo. El controlador PID calcula la utilización requerida $\Delta U(k)$ dependiendo

de las variables de control y las referencias de desempeño RP_r y U_r. Una vez calculado $\Delta U(k)$, se cambian las frecuencias de operación del conjunto de tareas mediante el variador de voltaje o actuador de nuestro lazo de control. A continuación se presenta el pseudocódigo del controlador PID.

```
void PID(int tiempo){
     //Cálculos para el control por utilización. Error y Util. requerida
     util_req_util = formula PID

     //Cálculos para el control por razón de perdida. Error y Util. requerida
     util_req_rp = formula PID

     switch(control){
         //Control por utilización
         case U: variador_niveles_voltaje(util_req_util);

         //Control por razón de perdida
         case RP: variador_niveles_voltaje(util_req_rp);

         //Control integrado utilización + razón de perdida
         case RPU: if(util_requerida_rp<util_requerida_util)
                      variador_niveles_voltaje(util_requerida_rp);
                   else
                      variador_niveles_voltaje(util_requerida_util);
     }
}
```

El *variador_niveles_voltaje(Util_req)* es el encargado de cambiar las velocidades de ejecución del conjunto de tareas dependiendo de la utilización requerida $\Delta U(k)$ (si la utilización requerida es negativa selecciona versiones mas rápidas, si es positiva selecciona versiones menos

rápidas) y la utilización estimada $Ue(k)$ en el sistema. De esta forma podemos obtener un ahorro en la energía del procesador debido a la variación de las velocidades de las tareas. A continuación se presenta el pseudocódigo del variador de voltaje optimizado implementando mediante el problema del *Knapsack*.

```
void Sim::variador_niveles_voltaje(float util_req){
     //Fija el tamaño del Knapsack
     tam_knapsack = Utilizacion()*1000 + util_req*1000;
     //Acomoda las tareas en forma de matrices dependiendo de su utilización
     //dada su velocidad
     genera_matrices();
     //Acomoda las matrices de acuerdo a su peso
     arreglo();
     //Los ordena de acuerdo al tamaño de la tarea utilizando Quicksort
     ordenar();
     //Selecciona las velocidades de las tareas, solo hasta el tope del
     //tamaño del knapsack
     greedy();
}
```

Bibliografía

[1] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez, editors. *Dynamic and Agressive Scheduling Techniques for Power Aware Real-Time Systems*, December 2001.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejía Alvárez. Dynamic and aggresive scheduling techniques for power-aware real-time systems. *IEEE Real-Time Systems Symposium*, 2001.

[3] L. Benini and G. De Micheli. *Dynamic power management: Design techniques and CAD Tools*. Kluwer, 1997.

[4] Giorgio C. Butazzo. *Hard Real-Time Computing Systems. Predictable scheduling algorithms and applications*. Kluwer Academic, 1997.

[5] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Control de Sistemas Dinamicos con Retroalimentación*. Addison-Wesley Iberoamerica, 1991.

[6] F. Gruian. Hard real-time scheduling using stochastic data and dvs processors. *In Proceedings of the International Symposium on Low Power Electronics and Design*, pages 46–51, August 2001.

[7] F. Gruian and K. Kuchcinski. Lenes: Task scheduling for low energy systems using variable supply voltage processors. *In Proc. Asia South Pacific - DAC Conference*, 2001.

[8] Paul J. M. Havinga and Gerard J. m Smit. Design techiques for low-power systems. *Journal of Systems Architecture*, pages 1–21, 2000.

[9] Constantine H. Houpis and Gary B. Lamont. *Digital Control Systems, Theory, Hardware, Software*. McGraw-Hill, 1992.

[10] Chung-Hsing Hsu, Ulrich Kremer, and Michael Hsiao. Compiler-directed dynamic frequency and voltage scheduling. *Workshop on Power-Aware Computer Systems (PACS'00)*, 2000.

[11] D. Kirovski I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Power optimizarion of variable voltage core-based systems. *In Design Automation Conference*, 1998.

[12] Intel. Procesadores strongarm www.intel.com.

[13] Intel. Procesadores x-scale www.intel.com/design/intelxscale.

[14] Intel, Microsoft, Compaq, Phoenix, and Toshiba. Estándar acpi www.acpi.info.

[15] T. Ishira and H. Yasuura. Voltage scheduling problem for dynamically varying voltage processors. *In Proc. Int'l Symposium on Low Power Electronic and Design*, 1998.

[16] K. Kawamoto. Electricity used by office equipment and network equipment in the u.s.: Detailed report and appendices. Technical report, School of Electrical Engineering, March 2000.

[17] W. Kim, J. Kim, and S. L. Min, editors. *A Dynamic Valtage Scaling Algorithm for Dynamic-Priority Hard Real Time Systems Using Slack Time Analysis*, March 2002.

[18] C. M. Krishna and Y. H. Lee. Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems. *In Proc. of the IEEE Real-Time Technology and Applications Symposium*, 2000.

[19] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. *In Proceedings of the 37th Design Automation Conference*, pages 806–809, June 2000.

[20] C. L. Liu and J. Layland. Scheduling algorithm for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20:46–61, 1973.

[21] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with pace. *In Proc. of ACM SIGMETRICS Conference Cambridge*, 2001.

[22] Jacob Rubin Lorch. *Operating Systems Techiques for reducing Processor Energy Consumptio*. PhD thesis, University of California, Berkeley, 2001.

[23] Chenyang Lu, John A. Stankovic, Tarek F. Abdelzaher, Gang Tao, Sang H. Son, and Michael Marley. Performance specification and metrics for adaptive real-time systems. *Real-Time Systems Symposium*, 2000.

[24] Chenyang Lu, John A. Stankovic, Gang Tao, and Sang H. Son. Design and evaluation of a feedback control edf scheduling algorithm. *Real-Time Systems Symposium*, 1999.

[25] Yung Hsiang Lu. *Power-Aware Operating Systems for Interactive Systems*. PhD thesis, Stanford university, December 2001.

[26] J. Luo and N. K. Jha, editors. *Battery-aware static Scheduling for Distributed Real-Time Embedded Systems*, June 2001.

[27] Pedro Mejía Álvarez, Eugene Levner, and Daniel Moose. An integrated heuristic approach to power-aware real-time scheduling. *Springer Verlag's Lecture Notes on Computer Science Series (LNCS)*, 2002.

[28] Pedro Mejía Álvarez, Eugene Levner, and Daniel Moose. Power-optimized scheduling server for real-time tasks. *Proceeding of the 8th IEEE Real-Time and Embedded technology and Applications Symposium*, 2002.

[29] A. Manzak and C. Chakrabarti, editors. *Variable Voltaje Task Scheduling for Minimizing Energy or Minimizing Power*, June 2000.

[30] Katsuhiko Ogata. *Ingeniería de Control Moderna*. Prentice Hall, 1980.

[31] P. Pillai and K. G. Shin, editors. *Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems*, October 2001.

[32] Qt. Librería para el diseño de interfaces gráficas multiplataforma www.trolltech.com.

[33] D. Shin, J. Kim, and S. Lee. Intra-task voltaje scheduling for low energy hard real-time applications. *IEEE Design and Test of Computers*, pages 20–30, March 2001.

[34] D. Shin, W. Kim, J. Jeon, J. Kim, and S. L. Min. Simdvs: An integrated simulation environment for performance evaluation of dynamic voltage scaling algorithms. *In Proc. of the Workshop on Power-Aware Computer Systems (PACS'02), Cambridge Ma*, 2002.

[35] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. *In Proc. of the 36th Design Automation Conference*, pages 134–139, 1999.

[36] Y. Shin, K. Choi, and T: Sakurai, editors. *Power Optimization of Real-Time Embedded Systems on Variable Speed Processors*, November 2000.

[37] Youngsoo Shin and Kiyoung Choi. System-level power optimization of embedded systems. Technical report, School of Electrical Engineering, seoul National University, March 2000.

[38] A. Sihna and A. Chandrakasan, editors. *Dynamic Voltaje Scheduling Using Adaptive Filtering of Workload Traces*, January 2001.

[39] Amit Sinha. *Energy Eficient Operating Systems and Software*. PhD thesis, Massachusetts Institute of Technology, Agost 2001.

[40] John A. Stankovic, Chenyang Lu, Sang H. Son, and Gang Tao. The case for feedback control real-time scheduling. *In Proceedings of the IEEE EuroMicro Conference on Real-Time Systems*, 1998.

[41] Karl J. Åström and T. Hagglund. *Automatic Tuning of PID Controlers*. Instrument Society of America, 1988.

[42] Karl J. Åström and Björn Wittenmark. *Computer-controlled systems: theory and design (2nd ed.)*. Prentice-Hall, Inc., 1990.

[43] Transmeta. Procesadores crusoe www.transmeta.com.

[44] Sanjay Udani and Jonathan Smith. Power management in mobile computing. *Journal of Systems Architecture*, Agust 1996.

[45] F. Yao, A. Demers, and S. Shenker, editors. *Scheduling for Reducing CPU energy*, Octuber 1995.

Los abajo firmantes, integrantes del jurado para el examen de grado que sustentará el **Sr. Julio César Cornejo Herrera**, declaramos que hemos revisado la tesis titulada:

Planificador de Tareas en Tiempo Real con
Restricciones de Energía en Retroalimentación

Y consideramos que cumple con los requisitos para obtener el Grado de Maestría en Ciencias en la especialidad de Ingeniería Eléctrica opción Computación.

Atentamente,

Dr. Pedro Mejía Álvarez _____

Dr. Alberto Soria López _____

Dr. Guillermo Benito Morales Luna _____

Dr. Luis Gerardo de la Fraga _____