Adaptación Computacional en Sistemas Percepto-Efectores

Daniel Hernández Sosa, Jorge Cabrera Gámez, Antonio Domínguez Brito,
Josep Isern González, Óscar Déniz Suárez, José Fernández Pérez
y Claudio Pascual Cruz
Dpto. de Informática y Sistemas, Universidad de Las Palmas de G. C.
E-mails: {dhernandez,jcabrera,acdbrito,jisern,odeniz}@dis.ulpgc.es

Resumen

Uno de los aspectos que se considera fundamental en un sistema percepto-efector es la capacidad para garantizar un comportamiento robusto. Esto implica que en condiciones del entorno cambiantes o incluso desfavorables, la integridad del sistema en primer lugar, y su eficacia en segundo, deben disponer de elementos de salvaguarda.

En este trabajo se describen un conjunto de estrategias de control que tratan de ajustar el consumo de recursos computacionales de un sistema percepto-efector basado en agentes distribuidos a los disponibles en cada momento. Se aborda tanto el caso de sistemas calibrados como no calibrados.

1. Introducción

Un objetivo prioritario de un sistema percepto-efector es alcanzar un comportamiento robusto en el desempeño de las tareas que se le encargan [9]. Una de las principales fuentes de inestabilidad para alcanzar este objetivo de robustez deriva de las relaciones de bajo nivel entre el tiempo de respuesta requerido y la capacidad de cómputo disponible [4] [8]. Lo ideal es que el sistema degrade su funcionamiento de una manera controlada de forma que, para una cantidad de recursos determinada, el resultado que se obtiene es una configuración que proporciona el "mejor rendimiento posible". Asimismo, es necesario poder recuperar ordenadamente el sistema desde una situación degradada cuando las condiciones así lo permitan. La complejidad del problema es elevada si tenemos en cuenta factores como son la dificultad para estimar los recursos disponibles, la complejidad para evaluar la calidad de un resultado, las múltiples alternativas de degradación/promoción o la diversidad de tareas en ejecución.

Nuestro interés se dirige especialmente a la adaptación computacional de bajo nivel. Los esfuerzos de investigación se concentran aquí en dos aspectos: variación de la carga y planificación de las tareas. Por un lado se buscan esquemas de cómputo para las tareas que permitan variar dinámicamente la carga que suponen para el sistema. Por otro, se intenta aprovechar ese nuevo grado de libertad en los mecanismos de planificación, a fin de permitir un reparto óptimo de los recursos disponibles.

La modificación de las demandas computacionales de una tarea puede a su vez alcanzarse mediante diferentes estrategias. Los algoritmos *anytime* [5] constituyen estrategias de procesamiento iterativas que proporcionan resultados de calidad creciente a medida que aumenta el tiempo disponible. En la computación imprecisa (*Imprecise Computation*) [10] se fraccionan las tareas en una parte obligatoria y una parte opcional, que sólo se ejecuta si existen recursos computacionales disponibles. El diseño en función del tiempo (*Design-to-time*) [6], o método de múltiples versiones, se basa en disponer de varias implementaciones alternativas para cada tarea, cada una con diferentes consumos de recursos. En la planificación deliberativa (*Deliberative scheduling*) [1] se consideran todas las tareas como algoritmos *anytime*, planificándose la ejecución de manera que se maximice alguna medida de la utilidad (calidad) global del sistema.

Todos estos métodos se basan de una u otra manera en el manejo de perfiles de rendimiento en los que se representa la evolución de la calidad que se alcanza en función de la cantidad de recurso disponible. Otros autores como Musliner et al. [11] trasladan estos resultados al contexto de una arquitectura robótica híbrida intentando eliminar la necesidad de disponer de perfiles de rendimiento. Para ello se combinan soluciones tácticas a nivel reactivo con soluciones estratégicas a nivel deliberativo. Los perfiles de rendimiento son reemplazados por una medida de la calidad global alcanzada.

Una vez identificados los elementos de ajuste disponibles, el siguiente paso es utilizar esos grados de libertad para alcanzar un comportamiento adecuado en situaciones de carga y recursos computacionales variables. La planificación previa basada en el peor caso resulta excesivamente conservadora, si se quieren evitar las

violaciones temporales [15]. Se precisan, por tanto, mecanismos adaptativos dinámicos que ajusten el consumo de recursos a los disponibles en cada momento [2] [12]. El objetivo de estas técnicas es normalmente extender los mecanismos de planificación básicos para sistemas de tiempo real duro EDF (*Earliest Deadline First*) de prioridades dinámicas o RM (*Rate Monotonic*) de prioridades fijas, de manera que puedan aprovecharse las características de la carga computacional variable. Algunos ejemplos son los trabajos que se concentran en la fijación de las frecuencias de funcionamiento [13] [3].

Otro enfoque para el problema de la adaptación consiste en hacer énfasis en la integración en sistemas tolerantes a fallos [7] [2]. En estos casos se buscan combinaciones de las técnicas de redundancia para alcanzar un comportamiento adaptativo. Dichas técnicas incluyen TMR (*Triple Modular Redundancy*), PB (*Primary Backup*) y PE (*Primary Exception*). Normalmente se aplican en entornos multiprocesador. Stewart [14] propone un mecanismo de planificación híbrido denominado MUF (*Maximum Urgency First*). En este trabajo se combinan las prioridades estáticas del RM con las prioridades dinámicas del EDF.

Pocos sistemas, sin embargo, ofrecen recursos para garantizar la estabilidad y la degradación controlada para entornos altamente dinámicos y con múltiples objetivos.

2. Adaptación de la Carga

El objetivo de este trabajo es plantear un esquema de adaptación para un contexto de tiempo real blando con múltiples tareas en ejecución concurrente. Para garantizar una mayor aplicabilidad, se busca que los requerimientos impuestos al hardware y sistema operativo de soporte sean lo menos restrictivos posible.

En el marco de trabajo definido son planteables diferentes bucles de control con el objeto de alcanzar un comportamiento adaptativo del sistema en tiempo de ejecución. Cada uno de estos bucles se corresponde con las siguientes señales de referencia básicas:

Timeouts: Verificación de la frecuencia de funcionamiento deseada para cada tarea. Supone, en principio, un bucle de control específico para cada tarea.

Nivel de carga deseado: Valor de referencia para la carga global del sistema. Constituye un bucle de control global para todo el sistema.

Homogeneidad de la carga: Se persigue una distribución temporal uniforme de la carga del sistema.

Tiempo de respuesta: Definido como el intervalo que transcurre entre la aparición de un estímulo en las entradas y la generación de la salidas correspondientes.

2.1. Descripción del sistema

Los esquemas de adaptación descritos en este trabajo se han ensayado sobre un sistema percepto-efector basado en agentes distribuidos, y organizado jerárquicamente en estados, tareas y módulos. En el nivel más bajo, existen cuatro tipos de módulos: sensores, de diagnóstico, de acción y efectores. Los módulos sensores son los encargados de capturar los datos a través de los sensores físicos del sistema. Los diagnósticos son módulos de transformación de datos que toman datos de entrada, de sensores o de otros diagnósticos, y suministran los resultados de la aplicación de diferentes algoritmos de procesamiento sobre los mismos. Las acciones son módulos de control de flujo que analizan los resultados de los diagnósticos y generan comandos como respuesta. Finalmente los efectores son módulos que encapsulan actuadores físicos, y se encargan de ejecutar los comandos que reciben desde las acciones. Todos los módulos comparten una estructura interna común, formada por una unidad de control (TD), una unidad de procesamiento (BU) y una unidad de comunicaciones (COM).

Las tareas son el resultado de la combinación de módulos para formar una cadena de percepción-acción. El flujo de datos en la tarea se inicia en los módulos sensores, continúa en los diagnósticos y las acciones, y culmina en los efectores. La acción es el módulo que supervisa la evolución de la tarea.

Finalmente, los estados están formados por agrupaciones de tareas que se ejecutan de forma simultánea en el sistema. Se definen transiciones para evolucionar a través de los diferentes estados del sistema.

2.2. Modelado de la carga

La carga a la que está sometido un sistema puede determinarse, de manera simplificada, a partir de las cargas aportadas por cada una de las tareas y módulos que se ejecutan en él. Definiremos las siguientes variables:

nm**:** Número total de módulos en ejecución.

Tt_i**:** Periodo de funcionamiento de la tarea t_i.

Tm_i**:** Periodo de funcionamiento del módulo m_i.

Tem_i**:** Tiempo de ejecución, medido como el tiempo transcurrido desde que el módulo i inició la ejecución hasta que ésta finaliza.

Tpm_i**:** Tiempo de procesamiento del módulo m_i. Coincide con Tem_i cuando la CPU está ocupada únicamente por dicho módulo.

La carga media evaluada en un intervalo Tm_i que un módulo m_i induce en la CPU puede aproximarse por

$$\widehat{Cm_i} = \frac{Tpm_i}{Tm_i}$$

donde el periodo del módulo se calcula como el mínimo de los periodos asociados a las tareas en las que está incluido. La carga media global del conjunto (C) se obtiene como resultado de la acumulación de las cargas medias aportadas por todos los módulos que se ejecutan en el sistema sobre un periodo determinado. Para que la medida sea estable debe tomarse como periodo de evaluación al menos el mayor de los periodos de las tareas del sistema, siendo lo ideal considerar ese periodo como el mínimo común múltiplo de todos los periodos T^*. Se tiene entonces

$$\widehat{Cm_i^*} = \frac{Tpm_i^*}{T^*}$$

donde Tpm_i^* es el tiempo de procesamiento consumido por el módulo m_i en el intervalo T^*. Finalmente se llega a

$$\hat{C} = \sum_{i=1}^{n} \widehat{Cm_i^*}$$

Esta aproximación es conservadora, puesto que no se tienen en cuenta los costes adicionales que surgen cuando el número de tareas crece, como son los debidos a la conmutación de contexto o la distribución de los datos a los diferentes destinos. Tampoco se tiene en cuenta la carga generada por los módulos supervisores, aunque siempre será reducida. Según esto, se tiene $C > \hat{C}$.

2.3. Medida de la carga real

La estimación de la carga real del sistema es un factor fundamental a la hora de aplicar los mecanismos de adaptación. Permite cerrar los diferentes bucles de control generando las señales de realimentación necesarias para su comparación con las correspondientes señales de referencia. En este sistema se consideran dos tipos de medidas básicas: de carácter local y de carácter global. Las medidas locales corresponden a estimaciones internas a las tareas y los módulos, mientras que las medidas globales suministran estimaciones generales para todo el sistema.

Como estimación local de la carga se toma la relación que existe entre el tiempo de ejecución de cada módulo (Tem_i) y el tiempo máximo disponible, esto es, su periodo (Tm_i).

$$\widehat{Clm_i} = \frac{Tem_i}{Tm_i}$$

Un valor cercano a la unidad indica un sistema sobrecargado, mientras que valores cercanos a cero reflejan un sistema con gran cantidad de recursos disponibles. Otra medida que puede estimarse localmente es la homogeneidad de la distribución temporal de la carga. Para ello se comparan los tiempos empleados por cada módulo en completar el procesamiento en diferentes intervalos de tiempo.

Como medida global de la carga (C_{tt}) se emplea una tarea de test a la que se le asigna la prioridad mínima y una frecuencia de operación elevada (F_{tt}). Comparando el número de ejecuciones (NEx_{tt}) de dicha tarea en un intervalo de tiempo de análisis dado ($IntAn$) con el máximo esperado ($IntAn \times F_{tt}$) se obtiene una aproximación al nivel de carga del sistema. En este caso:

$$C_{tt} = 1 - \frac{NEx_{tt}}{IntAn \times F_{tt}}$$

2.4. Acciones de control

Como resultado de los diferentes bucles de control, pueden detectarse errores que, una vez analizados por los controladores, se traducen en acciones de control. Las acciones de control disponibles para modificar la carga computacional de las tareas que se ejecutan en el sistema son las siguientes:

- Degradación/Promoción en calidad.
- Degradación/Promoción en frecuencia.
- Desplazamiento temporal.
- Suspensión/Reactivación.

El sistema operativo sobre el que se ejecutan las tareas debe suministrar un planificador que utilice Round-Robin con colas de prioridad (tipo Linux, Windows NT/2000), dado que las estrategias de control planteadas se apoyan en este supuesto para su correcto funcionamiento.

Dentro de las acciones de control que afectan a la calidad de los resultados producidos tenemos la variación del tamaño o la resolución de los datos de entrada y los cambios del nivel de precisión en el cómputo de los resultados. La primera de las acciones se aplica principalmente sobre los sensores que suministran volúmenes de datos elevados, como es el caso de las cámaras. Los algoritmos de procesamiento, a su vez, pueden proporcionar resultados con distinto nivel de precisión, lo que también afecta a los recursos computacionales demandados. Por tanto, la aplicación de las acciones de control en calidad está condicionada a que las diferentes implementaciones de sensores, diagnósticos, etc., ofrezcan la posibilidad de trabajar con distintas resoluciones y niveles de precisión. Formalmente, estaríamos considerando un esquema de funcionamiento del tipo procesamiento *anytime* [16]. Nuestra propuesta de sistema computacional adaptativo se basa en algoritmos *anytime* tipo contrato, donde el tiempo disponible es aquél que permita a la tarea verificar su frecuencia de funcionamiento.

Las acciones de control que afectan a la frecuencia de funcionamiento consisten en la alteración del periodo de las tareas, lo que implica una variación en la carga computacional demandada por las mismas. En general, esta acción debe ser usada con precaución, pues puede comprometer la integridad del sistema; por ejemplo, en aplicaciones de detección de obstáculos o en lazos de control que impliquen movimiento.

Otra posibilidad estudiada ha sido el análisis de la distribución temporal de la carga en el sistema. Se ha comprobado como durante la ejecución pueden aparecer zonas de congestión, cuando coinciden en el tiempo múltiples tareas, acompañadas de intervalos de baja ocupación de la CPU. La solución al problema consiste en la detección de dichas zonas de congestión y en intentar reducirlas.

Finalmente, y como recurso extremo, el sistema puede decidir que la configuración actual de tareas no es sostenible, con lo que la única solución posible es la suspensión o sustitución de algunas de las tareas demandadas al sistema. En caso de ser suspendidas, dichas tareas podrán ser retomadas cuando las condiciones lo permitan.

2.5. Caracterización de la adaptación computacional

La evaluación de la calidad de la adaptación de los sistemas puede realizarse a través de diferentes medidas como son su flexibilidad y su progresividad. La primera de las medidas hace referencia a la variación de carga que sobre el sistema supone ir desde el nivel de máxima degradación hasta el de funcionamiento a pleno rendimiento con calidad máxima. La medida de la flexibilidad tendría la forma siguiente:

$$F = \frac{C_{max} - C_{min}}{100}$$

donde C_{max} es la carga para la calidad máxima y C_{min} la carga correspondiente a la calidad mínima.

Los efectos de la cuantización en los niveles de calidad se traducen en la imposibilidad de alcanzar todos los niveles posibles de carga en el sistema. Asumiendo que el sistema de control es tal que se genera siempre una carga menor o igual a la deseada, se obtiene una gráfica escalonada como la del ejemplo mostrado en la figura 1. Si se define el grado de utilización de recursos como el cociente entre la carga/tiempo asignado y la carga/tiempo disponible puede trazarse una curva sobre todo el recorrido del sistema. La figura 2 muestra esta curva para un sistema sencillo con 5 niveles de calidad distribuidos desde el 10 hasta el 90 % de carga.

La progresividad mide la relación que existe entre el tiempo disponible y el tiempo asignado, evaluada sobre un intervalo de carga determinado. Una posible medida consiste en evaluar el área bajo la curva de utilización y compararla con el valor máximo posible. Un sistema con perfiles continuos poseerá una progresividad con valor próximo a la unidad, mientras que un sistema con pocos niveles de calidad disponibles tomará valores cercanos a cero.

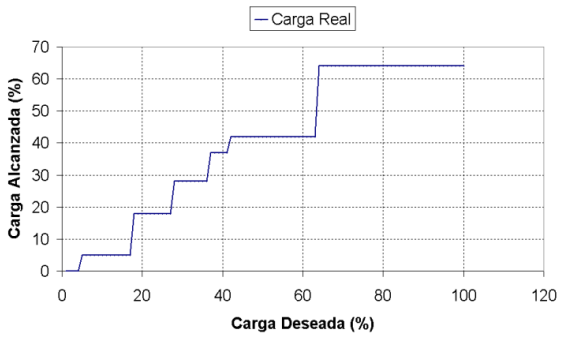

Figura 1: Relación entre la carga deseada y la carga correspondiente a los diferentes niveles de calidad del sistema

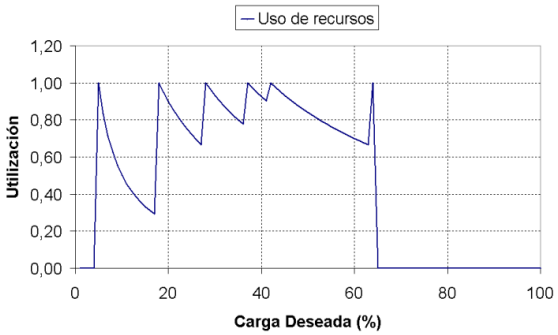

Figura 2: Efecto de la cuantización en el uso de los recursos disponibles

3. Adaptación Computacional en Sistemas no Calibrados

En un sistema no calibrado, no se dispone de información detallada acerca de la relación que existe entre la carga y el nivel de calidad de cada módulo, simplemente se conocen para cada módulo sus posibilidades de degradación. Para este caso se ha diseñado un esquema de organización de los niveles de degradación/promoción de cada módulo en el sistema en base a umbrales de homogeneidad utilizados tanto a nivel local como global. Dichos umbrales consisten en dos límites, inferior y superior, que indican cuáles deben ser los niveles de degradación mínimos y máximos que pueden presentar en un momento determinado los módulos que se encuentran en ejecución en el sistema.

3.1. Políticas de control

Partiendo de la base de las medidas de carga explicadas anteriormente, el sistema debe hacer operativas políticas de control que coordinen las respuestas del sistema ante las diferentes señales de entrada posibles. Describiremos a continuación la implementación de una política de control que combina las acciones de regulación disponibles en el sistema.

La eficacia de la adaptación depende en gran medida de los criterios que se apliquen a la hora de seleccionar alguna tarea o módulo candidato para la promoción/degradación. Dentro de las acciones de control en calidad, la reducción del tamaño de los datos tiene un efecto más rápido, aunque también más violento, sobre la carga del sistema, mientras que con el uso de niveles de precisión inferiores se produce una acción de control más paulatina. La topología afecta igualmente a la magnitud de las correcciones, de modo que un sensor cuyos datos de salida se suministran a módulos de cómputo de diagnósticos tiene tanta más relevancia como elemento de descarga del sistema cuanto mayor es el número de dichas conexiones.

La frecuencia de operación de cada tarea también condiciona el resultado de la acción de control aplicada, siendo lógicamente las respuestas más rápidas e intensas las que se obtienen al degradar las tareas con frecuencias más elevadas. Otra característica a considerar es el hecho de que no todas las tareas cargan igualmente al sistema, por lo que conviene buscar tareas con un elevado tiempo de procesamiento en relación con su tiempo de ciclo si se desea conseguir un mayor impacto.

3.2. Control de las violaciones temporales

El algoritmo de control de violaciones temporales se activa de forma local cuando un timeout es detectado por algún TD dentro del sistema. Como ya se ha indicado, la "sensibilidad" de esa detección puede ajustarse para evitar que el sistema de control reaccione ante situaciones de sobrecarga puntual o medidas ruidosas. La solución al problema puede alcanzarse bien dentro de un ámbito local o bien a nivel global. Puesto que se trata de un sistema de tiempo real blando, no se precisan mecanismos de detección más exigentes, como serían los demandados en un entorno de tiempo real duro [15].

En primera instancia, se aplican las acciones de control locales, cuyo margen de actuación se encuentra delimitado por los umbrales de homogeneidad. De esta forma, si el controlador de la tarea encuentra algún

Algoritmo 1 Algoritmos de control de timeouts local (tarea t_i) y global

CONTROL LOCAL

$ControlActivado = falso$
if Timeouts detectados en t_i **then**
  for $m_j \in$ { Módulos en tarea t_i} **do**
    if Degradación m_j dentro de límites **then**
      Añadir m_j a $Lista\_Candidatos$.
    end if
  end for
  if $Lista\_Candidatos$ no vacía **then**
    Seleccionar candidato a degradar m_d de $Lista\_Candidatos$.
    Enviar señal degradación a m_d.
    $ControlActivado = verdadero$
  else
    Notificar nivel superior.
  end if
end if
if $ControlActivado == verdadero$ **then**
  Deshabilitar verificación de timeouts en intervalo Tt_i.
end if

CONTROL GLOBAL

for $M_i \in$ { Módulos en nivel de prioridad $Nivel\_P$} **do**
  if Degradación M_i dentro de límites **then**
    Añadir M_i a $Lista\_Candidatos$.
  end if
end for
if $Lista\_Candidatos$ no vacía **then**
  Seleccionar candidato a degradar M_d de $Lista\_Candidatos$.
  Enviar señal degradación M_d.
else
  if Umbrales de homogeneidad en máxima degradación **then**
    Suspender tareas en $Nivel\_P$.
  else
    Desplazar umbrales.
  end if
end if

módulo degradable dentro de los límites establecidos, lo añade a la lista de candidatos. Una vez completado el recorrido por los módulos pertenecientes a la tarea, se selecciona uno de ellos para la degradación. Si esto no es posible, se agotan los recursos a nivel local, por lo que genera una señal para notificar el evento al nivel superior. El algoritmo 1 forma parte del código del bucle de control en las unidades TD, y resume la secuencia seguida en el control local de las violaciones temporales.

El algoritmo de control global (algoritmo 1) para los timeouts toma como dato de entrada la violación temporal detectada por los niveles inferiores e intenta seleccionar candidatos para la degradación que se encuentren en el mismo nivel de prioridad en el que ha sido detectado el error. Si la búsqueda no tiene éxito, se desplazan los umbrales de homogeneidad hacia niveles de mayor degradación y se repite el proceso. Finalmente puede ocurrir que ni en el nivel máximo de degradación sea posible respetar las frecuencias de funcionamiento deseadas, con lo que la única solución es la reconfiguración del sistema o la suspensión de algunas de las tareas actualmente en ejecución.

La anchura de la banda de homogeneidad debe reflejar un compromiso entre rapidez de respuesta y estabilidad. Una banda ancha ofrece un margen de maniobra mayor para las acciones locales a costa de incrementar los desequilibrios dentro del sistema, mientras que una banda estrecha conduce a un sistema muy homogéneo pero incrementa el tiempo de respuesta.

3.3. Control del nivel de carga

Otra de las situaciones en la que se activan los mecanismos de acomodación es el establecimiento de un nivel de carga de referencia para el sistema, con el fin de liberar recursos para admitir nuevas tareas o, simplemente, garantizar un margen de seguridad. En este caso, las políticas de control aplicadas son exclusivamente globales.

El control se inicia con la comparación entre el nivel de referencia externo especificado (carga deseada) y la carga medida para el sistema. En el caso de la degradación, la estrategia de control se inicia intentando seleccionar alguna tarea de la más baja prioridad degradable en calidad, si esto no es posible se desplazan los límites de homogeneidad hacia niveles más altos de degradación y se repite la búsqueda. Alcanzados los límites máximos de degradación en calidad, se intenta realizar una degradación en frecuencia, comenzando por las tareas menos degradadas. Una vez agotados todos los recursos en este nivel de prioridad, y si aún es necesario seguir degradando el sistema, se pasa al nivel de prioridad superior y se repite el proceso. Cuando todas las posibilidades de degradación han sido activadas, el único mecanismo posible para conseguir reducciones adicionales de carga es la suspensión de las tareas, comenzando nuevamente por las de menor prioridad. El algoritmo 2 muestra, de forma resumida, el código utilizado para la degradación en el control del nivel de carga.

Cuando el nivel de referencia de carga se encuentra por encima del nivel actual el mecanismo de control

Algoritmo 2 Algoritmo de control de nivel de carga (degradación)

```
Deg = falso
Nivel_P = Min_P
while (Deg == falso)&(Nivel_P < Max_P) do
  Degradación en calidad:
  while (Deg == falso)&(Lim_Hom < Max_DegL) do
    if Algún módulo degradable en calidad en Nivel_P then
      Enviar señal degradación.
      Deg = verdadero
    else
      Incrementar Lim_Hom.
    end if
  end while
  if NoDeg then
    Degradación en frecuencia:
    if Alguna tarea degradable en frecuencia en Nivel_P then
      Enviar señal degradación.
      Deg = verdadero
    else
      Incrementar Nivel_P.
    end if
  end if
end while
if Deg == falso then
  Suspensión de tareas:
  Elegir tarea menos prioritaria T_i.
  Suspender T_i.
end if
```

que se dispara es el inverso, esto es, la promoción. Ésta se inicia con la recuperación de posibles tareas suspendidas, comenzando por los niveles más altos de prioridad. Cuando todas las tareas se encuentran en ejecución, se intenta restablecer los periodos de funcionamiento fijados inicialmente para las mismas, empezando nuevamente por las más prioritarias. Si la promoción en frecuencia finaliza y el nivel de carga especificado lo sigue permitiendo, se pasa a la promoción en calidad en orden decreciente de prioridad. En el caso extremo, el sistema llegará a funcionar a pleno rendimiento en todas las tareas. El algoritmo es similar al empleado en la degradación, aunque reordenando las acciones.

4. Adaptación Computacional en Sistemas Calibrados

El comportamiento de los diferentes algoritmos de procesamiento dentro del sistema puede caracterizarse mediante perfiles de rendimiento. Disponer de esta información a priori permite plantear esquemas de configuración o compilación del procesamiento solicitado con antelación a su ejecución en el sistema. El objetivo final es obtener una distribución del tiempo de procesamiento entre los diferentes módulos en ejecución que cumpla con las restricciones de la carga máxima admisible proporcionando un nivel de calidad aceptable.

4.1. Tiempo de procesamiento y calidad

En un contexto de procesamiento *anytime* la calidad de un resultado es una cantidad, normalmente expresada en el intervalo [0,1], que mide la bondad del producto de un determinado algoritmo. La relación que existe entre tiempo de procesamiento y calidad queda determinada por los denominados perfiles de rendimiento. Esta representación muestra cómo evoluciona la calidad del resultado de un algoritmo *anytime* a medida que el tiempo de procesamiento disponible aumenta (figura 3). Se trata siempre de funciones monótonas crecientes, puesto que la calidad siempre debe aumentar a medida que el algoritmo dispone de más tiempo de procesamiento.

Los perfiles ideales son continuos. En la práctica, sin embargo, nos encontraremos con funciones escalonadas, puesto que no será posible aprovechar cada unidad de tiempo adicional para incrementar la calidad. Serán las

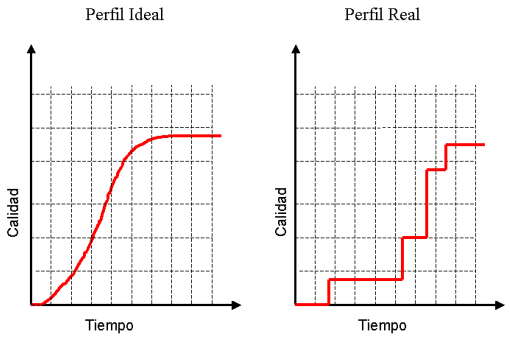

Figura 3: Perfiles de rendimiento

características de cada implementación las que determinen el número de niveles o saltos de calidad disponibles.

La obtención del nivel de calidad de una tarea pasa por la combinación o composición de los perfiles de calidad de los módulos que la integran. Las posibles funciones de combinación de perfiles de calidad son múltiples. El objetivo es derivar el perfil a la salida de un módulo a partir de los perfiles de las entradas y el propio del módulo.

El problema de la compilación [17] consiste en el reparto de un tiempo máximo de ejecución para una tarea entre los diferentes módulos que la forman. En general, el reparto está condicionado por la maximización o minimización de alguna función objetivo. Tanto la composición como la compilación son los elementos sobre los que se basan los esquemas de control planteados para los sistemas calibrados.

4.2. La compilación de tareas

El problema de la compilación consiste en distribuir el tiempo de procesamiento disponible entre el conjunto de módulos que integran una tarea determinada de modo que la calidad resultante sea máxima. Para un conjunto de $NumM$ módulos, se trata de encontrar la configuración de tiempos de procesamiento para cada uno de ellos ($T = Tpm_1, ..., Tpm_{NumM-1}$) de forma que deben cumplirse las siguientes condiciones:

$$\begin{cases} T_{disp} >= \sum_{i=1}^{NumM} Tpm_i \\ \\ Q(T) \text{ es máxima} \end{cases}$$

Donde T_{disp} es el tiempo total disponible para procesamiento, y Q la calidad final resultante para la tarea.

Este planteamiento supone un problema de optimización del tipo NP-Completo, por lo que una solución mediante prueba exhaustiva de todas las posibles combinaciones no es viable cuando la dimensión del problema crece. En vez de ello, pueden plantearse algoritmos de compilación local, en los que el ámbito de la optimización se reduce, haciendo que el problema global sea computacionalmente tratable. En los trabajos de Zilberstein ([16], [17]), se presentan algunos algoritmos de compilación local cuya complejidad es polinómica. El autor demuestra además que el resultado de la compilación local (al menos para el caso en que no existan expresiones repetidas) proporciona una solución óptima. Se realizará una traslación de los resultados al contexto del sistema propuesto.

El algoritmo 3, inspirado en el trabajo de Zilberstein [17], presenta un esquema que realiza una compilación local limitada a un módulo fuente y sus destinos directos, de forma que se obtenga una calidad máxima. El proceso parte de una distribución inicial de tiempo lineal (T_0) y aplica intercambio de unidades temporales (s) de tamaño decreciente en cada iteración hasta alcanzar un valor mínimo (s_{min}).

Hay que destacar aquí que, aunque la compilación local resuelve el problema de la complejidad exponencial asociada al esquema global, no por ello deja de ser un proceso computacionalmente costoso. Su aplicación, pues, debe considerarse detenidamente, especialmente en el caso de sistemas muy sobrecargados.

4.3. Algoritmos de distribución de tiempo a las tareas

A partir de la información de calibración, se plantean para el sistema diferentes estrategias de distribución del tiempo de procesamiento disponible entre todas las tareas y los módulos en ejecución con el fin de que la

Algoritmo 3 Algoritmo de compilación local de tarea

```
Asignar tiempos iniciales T_0 a partir del tiempo disponible T_disp.
Calcular calidad inicial Q_0 y tiempo de intercambio s.
while s > s_min do
  for Cada par de nodos i,j do
    t'_i = t_i − s
    t'_j = t_j + s
    Q'_0 = Calidad(T')
    if Q'_0 > Q_0 then
      t_i = t'_i
      t_j = t'_j
      Q_0 = Q'_0
    end if
  end for
  s = s/2
end while
```

carga del sistema no supere un valor máximo determinado. Se va a asumir que el tiempo de procesamiento asignado a un módulo determinado se consume como ciclos activos de CPU, sin incluir tiempos de espera. De esta forma, la carga varía linealmente con el tiempo de procesamiento y se puede establecer una relación directa con el ciclo de trabajo.

Para extender el proceso de compilación a múltiples tareas es necesaria una medida de la calidad global de todo el conjunto. Para ello basta con hacer que todas las tareas confluyan en un módulo de salida común ficticio con perfil de carga constante y unitario, de forma que su calidad sea la combinación de las calidades de todas las tareas. Otro aspecto por resolver, es la influencia de la prioridad en la asignación de carga. De alguna forma, un sistema con pocos recursos debería reflejar una relación directa entre calidad y prioridad. Una posibilidad que se ha probado es afectar la medida de la calidad final de cada tarea por un factor que dependa de su prioridad. Si ese factor es menor a medida que la prioridad es mayor se consigue el efecto deseado, puesto que el sistema asigna más recursos a las tareas más prioritarias y una menor cantidad a las menos prioritarias.

La distribución de tiempos basada en la optimización de la calidad, aunque constituya el esquema más adecuado desde el punto de vista de la calidad final alcanzada, presenta el inconveniente de que una modificación en cualquiera de los parámetros del problema puede llevar a una reconfiguración del reparto de tiempos para todo el sistema, lo que puede representar un coste y unos retardos considerables. En estos casos, un esquema como el visto en el apartado anterior permite una reconfiguración localizada, aunque no proporcione la máxima calidad posible.

4.4. Políticas de control

Una vez establecida la distribución temporal para las tareas de la aplicación, el sistema debería comportarse en base al modelo teórico. Sin embargo, puede ocurrir que durante la ejecución aparezcan variaciones en la carga no esperadas. Las causas de estas diferencias pueden tener distintos orígenes, tales como las imperfecciones en el modelado, los errores en la calibración o las cargas extras no anticipadas.

Los diferentes bucles de control a emplear coinciden con los ya vistos para el caso de los sistemas no calibrados. La diferencia fundamental radica en que los umbrales de homogeneidad se plantean en este caso de forma directa sobre las medidas de calidad de las tareas, y no sobre el número de niveles de degradación disponibles. De nuevo el coste de la compilación de tareas juega un papel importante, en especial cuando el número de módulos activados en el sistema es elevado. Hay que tener en cuenta la demanda computacional que representa este proceso para prever sus consecuencias. Suponiendo que no existen máquinas externas sobre las que lanzar la compilación en paralelo, puede ensayarse una ejecución en mínima prioridad, o bien reservar recursos anticipadamente.

El comportamiento del bucle de control depende de la selección de candidatos. Pueden adoptarse dos posturas: primar la homogeneidad o primar la rapidez de respuesta.

4.5. Transición de sistemas no calibrados a sistemas calibrados

En muchos casos nos encontraremos en un estado transitorio entre el inicio de la ejecución y la caracterización completa del sistema. Un enfoque conservador intentará, siempre que sea posible, concentrar las acciones

de control en módulos que ya hayan sido probados, lo que aumenta el margen de seguridad a costa, probablemente, de descartar acciones más efectivas. Por el contrario, un esquema más arriesgado tratará de desviar las acciones hacia módulos pobremente caracterizados, lo que permitirá mejorar la calidad de la calibración del sistema. La figura 4 muestra las relaciones que existen entre los elementos de calibración y control.

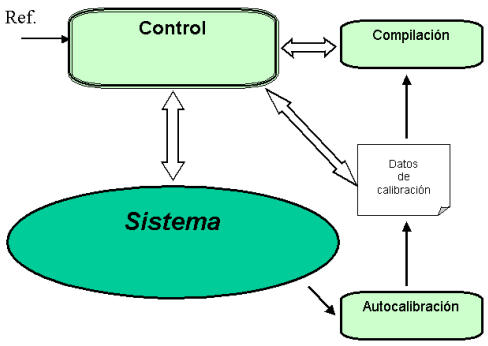

Figura 4: Estructura del control

Los escenarios de trabajo planteables pueden ser, de una parte, utilizar la compilación como configuración inicial a partir de la cual el sistema evoluciona, o de otra, emplear la compilación siempre que se requiera generar una nueva señal de control. En el primer caso, el impacto en el rendimiento producido por el coste computacional de la compilación es menor, mientras que en el segundo puede ser crítico. Los factores que deben ponderarse a la hora de seleccionar un control basado en la calibración o un control de tipo exploratorio son, entre otros, la calidad de las salidas, el nivel de calibración, el error de ajuste o el nivel de carga del sistema.

5. Experimentos

A continuación se muestran algunos ensayos realizados sobre el prototipo actualmente desarrollado. En la gráfica 5 se presenta la evolución del sistema a medida que se van indicando diferentes niveles de referencia para el nivel de carga deseado, con algunos puntos de interés etiquetados. Se incluyen dos medidas de carga: para un intervalo de análisis corto (puntual) y para un intervalo mayor (media).

Al sistema se le fija un nivel de carga inicial del 50 % y, posteriormente (etiqueta A), se le comanda como objetivo el nivel del 10 %. Los recursos de degradación en calidad se agotan en (B), aplicándose entonces control en frecuencia. Sobre el segundo 125 de ejecución (C) se han agotado todos los recursos de degradación sin que se haya alcanzado el objetivo, por lo que simplemente se devuelven señales de degradación fallida al supervisor y el sistema se estabiliza. A partir de (D) se cambia la referencia de nivel de carga para el sistema al 30 %,

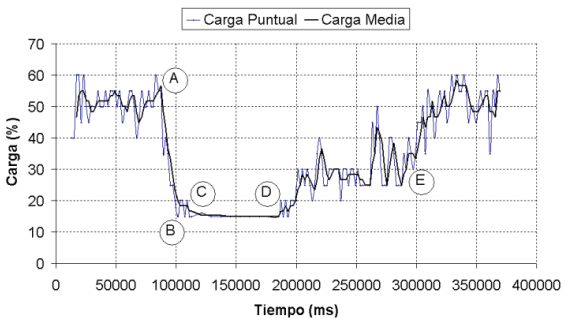

Figura 5: Evolución de la carga del sistema

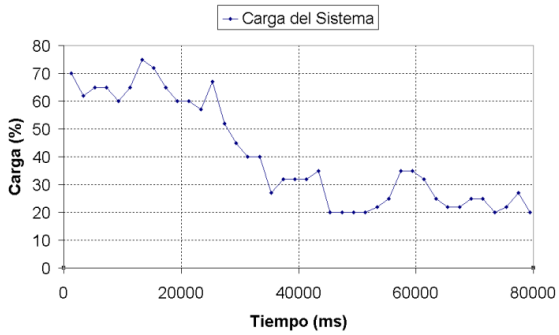

Figura 6: Evolución de la carga del sistema con tareas independientes

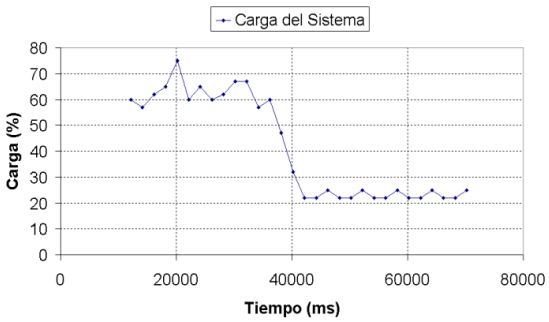

Figura 7: Evolución de la carga del sistema con tareas que comparten algunos módulos

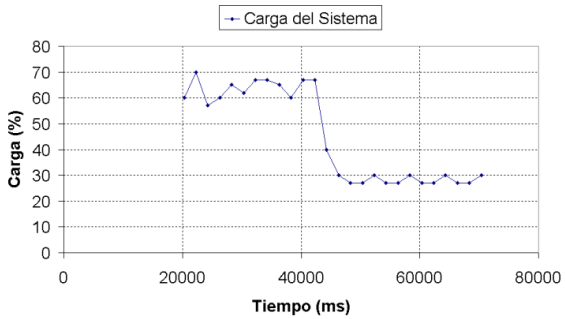

Figura 8: Evolución de la carga del sistema con tareas altamente dependientes

provocando la activación de la promoción en frecuencia y posteriormente la promoción en calidad. Finalmente, en (E) se vuelve a la referencia inicial de 50 % de carga.

La configuración de los módulos dentro del sistema determina la magnitud de las acciones de control que pueden activarse. En el siguiente ejemplo se ensayan tres configuraciones distintas sobre cuatro tareas en ejecución, comandando un nivel de referencia de carga bajo para activar las políticas de control.

Con tareas independientes, que no comparten ninguno de sus módulos productores (gráfica 6), se muestra el comportamiento de la carga del sistema desde los valores iniciales, que rondan el 65 %, hasta el 25 % que se comanda a partir del segundo 60 y se alcanza sobre los 85 segundos de tiempo de ejecución.

En la segunda prueba se considera una configuración con un sensor compartido por cada dos tareas. El resultado de la ejecución se ilustra en la figura 7, en la que se aprecia un retardo entre la fijación del nivel de referencia (en torno al segundo 32) y la finalización de la secuencia de control (hacia el segundo 42) de unos 10 segundos, frente a los 25 del caso anterior.

Por último, en la tercera prueba se considera que existe un único sensor compartido por todas las tareas. El efecto sobre el comportamiento del sistema se traduce en una transición entre los dos niveles de carga que tiene ahora una duración de unos 5 segundos aproximadamente, tal y como se muestra en la figura 8.

6. Conclusiones

Se presenta un esquema de adaptación en un sistema percepto-efector que permite ajustar los recursos demandados a las capacidades computacionales disponibles. Se analizan los problemas de control planteables, las acciones de control disponibles y su integración en políticas de control que las coordinen. Se aborda tanto el caso de los sistemas no calibrados como los calibrados. Asimismo, se incluyen modelos para la estimación de la carga del sistema y medidas para la caracterización de las capacidades de adaptación.

Dentro de la estrategia de control, se arbitran políticas tanto locales como globales. Las primeras permiten mejorar los tiempos de respuesta y reducir los overheads, en tanto que las políticas globales, ofrecen una mejor selección de los objetivos del control.

La integración de las políticas de control de bajo nivel en el propio sistema percepto-efector se ofrecen como una utilidad al diseñador, que éste puede parametrizar para conseguir un comportamiento más robusto de su aplicación.

Referencias

[1] M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–286, 1994.

[2] A. Bondavalli, J. Stankovic, and L. Strigini. Adaptable fault tolerance for real-time systems. In *Proc. of Predictably Dependable Computing Systems*, pages–, San Francisco, CA, September 1993.

[3] Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium (RTSS'98)*, pages 286–295, Madrid, Spain, 1998.

[4] B. D'Ambrosio. Resource bounded-agents in an uncertain world. In *Proceedings of the Workshop on Real-Time Artificial Intelligence Problems*, Detroit, MI, Aug. 1989.

[5] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the 7th National Conference on Artificial Intelligence, AAAI*, pages 49–54, St. Paul, MN, 1988.

[6] Alan J. Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491–1502, November/December 1993.

[7] O. González, H. Shrikumar, J. Stankovic, and K. Ramamritham. Adaptive fault-tolerance and graceful degradation under dynamic hard real-time scheduling. In *Proc. IEEE Real-Time Systems Symposium*, 1997.

[8] E. J. Horvitz, G. F. Cooper, and D. E. Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 1121–1127, Detroit, MI, 1989.

[9] Stephen D. Jones. *Robust Task Achievement*. PhD thesis, Institut National Polytechnique de Grenoble, 1997.

[10] J. Liu, K. Lin, R. Bettati, D. Hull, and A. Yu. Use of imprecise computation to enhance dependability of real-time systems, 1994.

[11] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. A framework for analyzing resource/quality tradeoffs in real-time ai, 1995.

[12] K. Ramamritham and J.A. Stankovic. Scheduling strategies adopted in spring: A overview. In A.M. van Tilborg and G.M. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Allocation*, pages 277–307. 1991.

[13] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control system. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 13–21, December 1997.

[14] David B. Stewart. *Real-Time Software Design and Analysis of Reconfigurable Multi-Sensor Based Systems*. PhD thesis, ECE Dept., Carnegie Mellon University, 1994.

[15] David B. Stewart and Pradeep K. Khosla. Mechanisms for detecting and handling timing errors. *Communications of the ACM*, 40(1):87–93, 1997.

[16] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

[17] Shlomo Zilberstein and Stuart J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.