VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PLANIFICACIÓN DE SISTEMAS DISTRIBUIDOS EN TIEMPO-REAL

T E S I S

QUE PARA OBTENER EL GRADO DE:

DOCTOR EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

ANTONIO FRANCISCO MENÉNDEZ LEONEL DE CERVANTES

DIRECTOR DE LA TESIS: DR. HÉCTOR BENÍTEZ PÉREZ

MÉXICO, D.F. **2009.**

Índice

Resumen

Este trabajo estudia el comportamiento de los sistemas distribuidos en tiempo-real y plantea dos contribuciones para realizar un análisis más acucioso de dichos sistemas: en primer lugar se plantea una métrica de disponibilidad de los nodos que participan en un sistema distribuido, esta métrica además de proponer como evaluar dicha disponibilidad, permite también evaluar el rendimiento del sistema distribuido. La segunda contribución del trabajo, plantea la necesidad y la posibilidad de tener un planificador global para los sistemas distribuidos en tiempo-real.

Se presentan dos casos de estudio en los que se pueden apreciar tanto la viabilidad como las características de ambas contribuciones; en el primer caso de estudio se implementa la métrica de disponibilidad del nodo en un sistema distribuido de alto rendimiento (Cluster), en donde la métrica permite obtener un uso equitativo del sistema distribuido al balancear la carga entre los nodos participantes con lo que se logra un buen rendimiento del sistema distribuido y un incremento en su desempeño al compararlo con otras técnicas de distribución de cargas. En el segundo caso de estudio se utiliza el planificador global propuesto para planificar un sistema distribuido de control en tiempo-real. El uso de este planificador presenta entre otras ventajas la posibilidad de encontrar las cotas de muestreo del sistema que le permitan operar dentro de los límites de desempeño aceptable así como la velocidad de transmisión que debe tener la red de comunicaciones.

1 Introducción

1.1 Objetivo

El objetivo de este trabajo es el estudiar el comportamiento de los Sistemas Distribuidos en Tiempo-Real (RTDS por las siglas en inglés de "Real-Time Distributed Systems") en particular como Sistemas de Control en Red (NCS por las siglas en inglés de "Networked Controlled Systems") y desde el punto de vista de la planificación. Se analiza la composición jerárquica de planificadores, las relaciones temporales entre los períodos de operación de las tareas, la reconfiguración dinámica y el reordenamiento de tareas.

Como parte del estudio se propone una métrica de desempeño que permita evaluar el comportamiento del RTDS y su incorporación a la toma de decisiones para la planificación.

Se presentan dos casos de estudio, en el primero se utiliza un sistema distribuido de alto rendimiento (cluster) en donde se implementa la métrica propuesta para realizar el balanceo de cargas, el segundo caso es un sistema de control por red con base en un sistema distribuido en Tiempo-Real, con una red en Tiempo-Real y planificadores de Tiempo-Real (locales a cada procesador).

1.2 Metas

En [RAVINDRAN et al., 03] se proponen diversos algoritmos para sistemas distribuidos en Tiempo-Real asíncronos y tolerantes a fallas buscando la localización activa de recursos, tomando en cuenta la migración de procesos, el balanceo de las cargas y la reconfiguración en línea. Para esto, se proponen dos tipos de algoritmos basados en la localización de recursos siguiendo el mejor esfuerzo [JENSEN, 92]; el primero desde la perspectiva del análisis de tiempos y el segundo con base en el análisis de la sobrecarga del sistema.

En este sentido, distintos procedimientos de optimización pueden ser llevados a cabo para definir plataformas básicas con ciertos rendimientos preestablecidos. Por ejemplo, el manejo adaptable

de recursos para tiempo real [AGRAWAL et al., 2003] proponen la evaluación de múltiples modelos con distintas características de rendimiento y requerimientos de recursos para la evaluación de las trayectorias candidatas en la arquitectura de control para multi-modelos activos. [BENITEZ et al., 05] han propuesto el manejo de retardos de tiempos considerando el uso de sistemas de planificación en Tiempo-Real dando como resultado la predicibilidad en la conducta del sistema de comunicación, esto permite un manejo adecuado de los recursos del sistema de cómputo aún cuando se restrinjan las capacidades del sistema.

Dado lo anterior se propone como meta el definir diversas formas de análisis de sistemas en tiempo real en donde se contemplen tanto el balanceo de cargas como la reconfiguración en línea con el objeto de proponer sistemas distribuidos adaptables.

Así mismo es necesario definir un método que permita acoplar cambios recurrentes en línea a requerimientos propios de la arquitectura de la red. Esto por medio del uso de sistemas de tipo medio (Middleware).
Una vez definida la arquitectura de cómputo con una serie de requerimientos básicos, es necesario implementar un algoritmo de planificación que permita manejar los recursos del sistema distribuido con base a dos características principales: el manejo de tiempos y el manejo de eventos.

En este sentido se definirá un algoritmo de planificación híbrido para manejar los recursos, usando como medida de desempeño la eficiencia del caso de estudio.

Por lo tanto es necesario establecer como meta el desarrollo de dicho producto en un caso de estudio real, desde el punto de vista del sistema distribuido. Para tal motivo, se pretende configurar una red de cómputo en Tiempo-Real con sistemas operativos en Tiempo-Real.

1.3 Alcances y contenido

En el capítulo 2, se presentan los antecedentes de los Sistema Distribuidos en Tiempo-Real (RTDS), posteriormente (capítulo 3) se presenta un análisis y modelado de un RTDS del tipo

NCS incluyendo el planificador propuesto. En el capítulo 4 se presenta la métrica de "Disponibilidad del nodo" propuesta para evaluar el desempeño del RTDS, así como el algoritmo HILO para su implementación y un análisis del comportamiento de la métrica. En el capítulo 5 están los dos casos de estudio, el primero consta de un sistema distribuido de alto desempeño tipo "cluster", en el que se evalúa el comportamiento de la métrica y el algoritmo para realizar el balanceo de cargas y el segundo caso de estudio es un sistema de control en red a través de un sistema distribuido en Tiempo-Real. Por último se presenta las conclusiones en el capítulo 6.

1.4 Audiencia

Este trabajo está dirigido a los investigadores en el área de sistemas distribuidos en Tiempo-Real (RTDS por sus siglas en inglés), en particular a aquellos que están interesados en la planificación y verificación global de este tipo de sistemas. En este trabajo se plantea primero el modelado de los RTDS en capas y su formalización por medio del álgebra de procesos. Lo que permite visualizar la necesidad de un planificador global para el RTDS y su evaluación por medio de una métrica de desempeño del mismo.

1.5 Contribuciones

Las contribuciones de este trabajo de investigación a la planificación de sistemas distribuidos en tiempo-real son dos. En primer lugar está la *Métrica de disponibilidad del nodo*, la cual permite entre otros aspectos el evaluar el rendimiento de un sistema distribuido, provee las especificaciones de la velocidad de la red de comunicaciones requeridas para tener un sistema balanceado y con un desempeño aceptable y por último, permite tener una visión más amplia de la carga impuesta a los nodos debido a los procesos que deben ejecutar, al considerar no solamente la utilización teórica del procesador sino además: los tiempos de las comunicaciones, las tareas inherentes a cualquier nodo participando en un sistema distribuido, el uso real del procesador y la disponibilidad de memoria. La segunda contribución es el planificador propuesto que permite dar un primer paso en la planificación global de los sistemas distribuidos en tiempo-real.

2 Antecedentes

Para el estudio de los sistemas distribuidos en Tiempo-Real se deben considerar varios elementos, siendo fundamentales: el modelado de los sistemas distribuidos, los conceptos básicos de los sistemas en Tiempo-Real haciendo hincapié en los planificadores y por último las métricas de rendimiento y planificabilidad.

2.1 Sistemas distribuidos

Tomando como base las definiciones de sistema distribuido de Colouris y Tanenbaum, "*Un sistema distribuido consiste de una colección de computadoras autónomas, unidas por una red de computo y equipadas con software específico*"[COLOURIS, 99] y *"Un sistema distribuido es una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora"* [TANENBAUM, 96]. Un Sistema Distribuido de cómputo es un conjunto de computadoras independientes conectadas por medio de una red, y que con un software específico (el cual permite que las computadoras coordinen sus actividades y compartan recursos) se muestran a los usuarios como una única computadora. La diferencia entre un Sistema Distribuido y una red de computadoras estriba precisamente en el software que constituye el sistema distribuido como un solo equipo virtual. Retomando a Tanenbaum, un sistema distribuido es: "*Un sistema de software construido sobre una red*" [TANENBAUM, 03].

El desarrollo de la tecnología y en particular la de los sistemas distribuidos de cómputo ha tenido como metas a alcanzar, el tener sistemas cada vez más rápidos, eficientes, confiables, precisos y adaptables. Algunos sistemas buscan brindar mayor calidad de servicio y rapidez al aumentar el número de servidores. En los sistemas de misión crítica, lo que se busca es la continuidad en la operación del sistema, donde lo más importante no es la rapidez sino su disponibilidad, aún a expensas de realizar inversiones redundantes en tecnología que sólo será usada en caso de falla. En otro tipo de sistemas lo que se busca es la precisión o capacidad de cómputo, pasando a un segundo plano el tiempo que el sistema tarde en entregar los resultados. Otro tipo de necesidades son aquellas que surgen cuando lo más importante es el tiempo de entrega, si el resultado es correcto pero tardío, se convierte en inválido para todo el sistema, o peor aún, ocasiona que el

sistema se colapse; este último tipo de sistemas se conocen como “Sistemas de Tiempo-Real” [GARCÍA, 03]

Un tipo de sistema distribuido es un sistema de control en red NCS (Networked Control System) en el cual, el impacto de la arquitectura de la red en el rendimiento del NCS es fundamental [FENG et al., 02], por lo que hay que considerar para el diseño de la misma dos puntos vitales para que el desempeño del NCS este en un rango aceptable [BENITEZ et al., 05]. Siendo el primero de ellos el dado por los retrasos inducidos por la propia red y los ocasionados por el procesamiento, y el segundo es el punto de saturación de la red, debido a un período de muestreo por debajo de la capacidad de esta.

En la Figura 2-1 se aprecian de forma gráfica estos conceptos, teniendo la línea punteada del centro de la gráfica como el límite entre el desempeño aceptable o inaceptable. Se observa que el “Control Digital” depende únicamente del período de muestreo para tener un desempeño aceptable y cruza la frontera en el punto A, mientras que el “Control por red” tiene límites inferior y superior para los períodos de muestreo, reflejándose en la gráfica en los punto B y C, siendo B el punto mínimo necesario de muestreo y el punto C el de saturación de la red. Por último en la parte inferior se aprecia la recta correspondiente al “Control Continuo”, el cual no depende de un período de muestreo dado que no se define en su estructura y por ende siempre presenta un desempeño aceptable.

El componente fundamental de este NCS es sin duda el protocolo de comunicaciones y el planificador para entrega de mensajes que se implemente. Desde el punto de vista de los algoritmos y protocolos de comunicaciones en Tiempo-Real existen dos tipos de planificación: por un lado están los planificadores estáticos, los cuales presentan las ventajas de garantizar la “planificabilidad” de los mensajes con un consumo adicional de tiempo y de recursos muy bajo, aunque en contraparte presentan las desventajas de una tabla de planificación potencialmente enorme y la nula flexibilidad operativa. Por otro lado existen los esquemas de planificación dinámicos, los cuales presentan una gran flexibilidad operacional y una tabla de planificación mínima, aunque en contraparte consumen muchos recursos para poder realizar la planificación

de manera dinámica y requieren de un analizador en línea para poder garantizar la planificabilidad [ALMEIDA et al., 04].

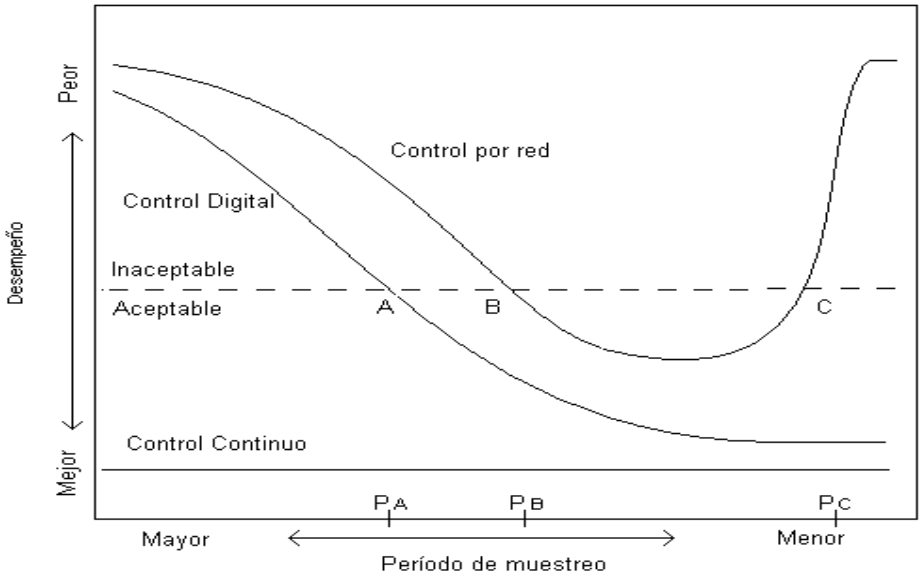


Figura 2-1. Comparación de rendimiento entre Control: Continuo, Digital y por red. [FENG et al., 02].

Independientemente del tipo de planificador (estático o dinámico), es necesario establecer una serie de requerimientos estructurales para poder definir una estrategia de planificación. Esto lleva a plantear un problema con alto grado de complejidad. El cual va desde el punto de vista de la arquitectura de la red, hasta las estrategias de comunicación de procesos y el desempeño de la herramienta que necesite Tiempo-Real.

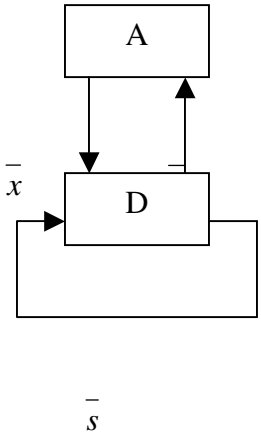
2.2 Sistemas de Tiempo-Real

En la década de los setentas se empezó a utilizar el término de sistemas de Tiempo-Real, para referirse a aquellos sistemas que podían dedicar todos sus recursos a un solo proceso en contraposición a los sistemas de tiempo-compartido, en los que el sistema dividía el tiempo de procesamiento entre todas las tareas que se tuvieran en ejecución. Una de las principales razones para utilizar los sistemas de Tiempo-Real, fue la necesidad de saber con certeza el tiempo en el que se podrían obtener los resultados del sistema [ALUR et al., 92], es decir, surgió el

requerimiento de conocer en forma determinista respecto al tiempo, el comportamiento del sistema.

Un error común cuando se habla de Tiempo-Real, es creer que se está hablando de un proceso instantáneo o sin retraso [AGRAWAL et al., 03]. La percepción generalizada es que el Tiempo-Real es una característica o virtud de algún sistema gracias a la cual, el resultado se obtiene casi instantáneamente. Nada más alejado de la realidad que el no considerar el tiempo que tardará un proceso en Tiempo-Real, ya que la principal característica para poder definir el Tiempo-Real es precisamente el considerar el instante de finalización del proceso. Apoyados en esta premisa se puede definir inicialmente que: el Tiempo-Real es un retraso conocido y determinado en el que se espera una respuesta, y está permitirá que el sistema reaccione dentro de los parámetros temporales del medio ambiente con el que interactúa [BENITEZ et al., 05]

Un Sistema de Tiempo-Real es aquel en el cual se deben cumplir dos condiciones fundamentales: Que las respuestas o salidas sean las correctas y además en un tiempo determinado; siendo esta última restricción la característica distintiva y más importante de un sistema en Tiempo-Real.



Donde: Vectores de sensores, Vector de decisión $\bar{y} \in Y$, Vector de "estado del sistema" $\bar{s} \in S$, Restricciones del medio ambiente A, Mapa de decisión D, D: S x X $\rightarrow$ S x Y, Conjunto de restricciones de tiempo T, Conjunto de restricciones de integridad I.

Figura 2-2. Modelo funcional de Sistema en Tiempo-Real [CHENG, 02].

En la Figura 2-2 se presenta un modelo funcional de un sistema de Tiempo-Real en donde: **X** es el espacio de valores de entrada de los sensores, **Y** es el espacio de los valores de decisión y **S** es el espacio de los valores de estado del sistema. Por lo tanto podemos decir que $\bar{x}(t)$ representa el valor de los sensores de entrada $\bar{x}$ en el tiempo t y así para todos los casos. Las restricciones del medio ambiente **A**, son relaciones sobre X, Y, S y son aseveraciones acerca del efecto de una decisión de control sobre el mundo exterior, las que a su vez influenciarán los valores futuros de los sensores de entrada.

El mapa de decisión D, relaciona $\bar{y}(t+1), \bar{s}(t+1)$ con $\bar{x}(t), \bar{s}(t)$, esto es: dado el estado actual del sistema y los sensores de entrada, D determina las siguientes decisiones y los valores de estado del sistema. Las decisiones especificadas por D, deben estar acordes con el conjunto de restricciones de integridad (seguridad) I. La implementación del mapa de decisiones D, está sujeta a las restricciones de tiempo T [CHENG, 02].

La descripción de este modelo funcional de un sistema en Tiempo-Real se podría resumir de la siguiente manera: Un sistema de Tiempo-Real es aquel que deberá procesar sus entradas (sensores) y estado actual, con restricciones de seguridad, para entregar las salidas en un tiempo específico. Siendo la capacidad de cumplir o no con este término perentorio, lo que determina si el sistema es o no de Tiempo-Real. Este sistema de cómputo que interactúa en Tiempo-Real con el medio ambiente a través de sensores y actuadores, puede ser representado por bloques como lo muestra la Figura 2-3.

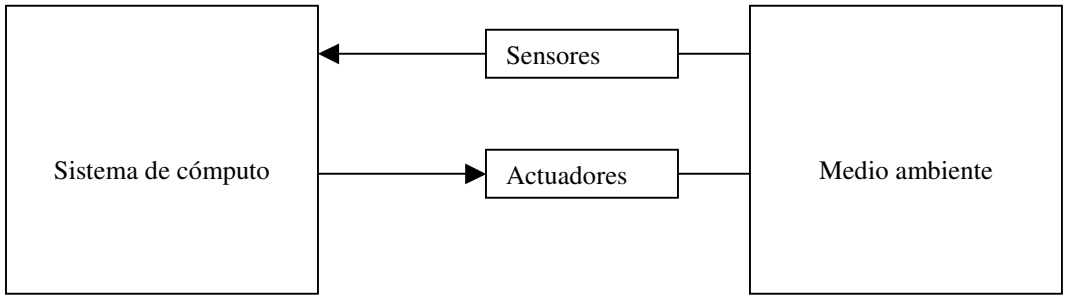


Figura 2-3. Modelo de bloques Sistema en Tiempo-Real.

En este modelo de bloques, se aprecia que los componentes de un sistema de Tiempo-Real son: El medio ambiente a controlar, los sensores de estado del medio ambiente, los actuadores sobre el medio ambiente y el sistema de cómputo encargado de recibir las lecturas de los sensores, evealuarlas y activar los actuadores, en el tiempo preciso para mantener al sistema de Tiempo-Real bajo control.

Al remarcar que es necesario que todas las acciones se lleven a cabo en el tiempo preciso, surge la necesidad de tener un componente del sistema de Tiempo-Real que garantice la factibilidad de ejecutar todas las tareas con estas restricciones de tiempo.

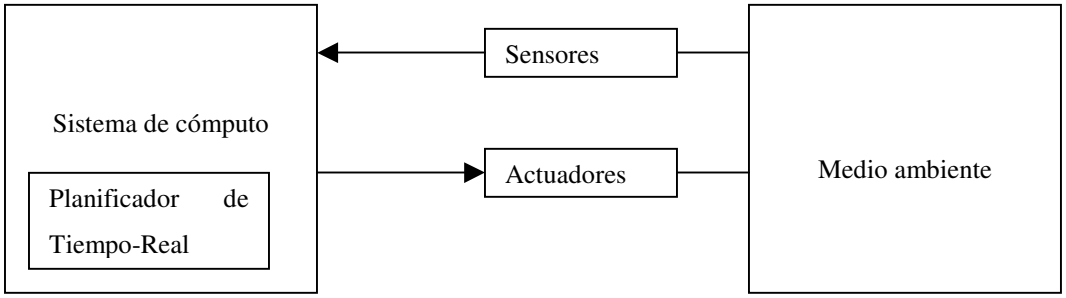


Figura 2-4. Modelo de bloques de Sistema en Tiempo-Real con Planificador.

Los sistemas de Tiempo-Real se clasifican en dos tipos dependiendo de lo estricto que deba ser el cumplir con su plazo: "inflexibles" (hard) y "flexibles" (soft). En los inflexibles, cada ejecución del proceso debe ser terminada exactamente o antes de que se llegue al plazo fijado para ello; los flexibles en cambio, tienen la tolerancia de no cumplir dicho plazo en algunas ocasiones conocidos también como "mejor esfuerzo", estos sistemas serán de Tiempo-Real si cumplen de manera estadística con la ejecución de sus procesos en la mayoría de los casos.[LIU, 00] (*p. Ej. El número de plazos no cumplidos por minuto es de dos o menos*). En general la implementación de los sistemas de Tiempo-Real usará una mezcla de ambos tipos, es decir, tendrán algunos procesos con restricciones inflexibles de tiempo (con plazos duros) y otros con restricciones flexibles (con plazos suaves).

Los sistemas de cómputo de Tiempo-Real deben ser procesados por sistemas operativos de Tiempo-Real, los cuales pueden ser considerados como sistemas monousuario ya que el concepto tradicional de usuario independiente no existe. Todo el sistema esta dedicado al proceso físico que se está controlando. *"En un sistema operativo de Tiempo-Real, el concepto tradicional de usuario independiente no existe tal y como sucede en los sistemas operativos de tiempo compartido o en los sistemas de redes de computadoras. Todo el sistema esta dedicado a realizar una misión particular, y puede ser concebido como que tiene un solo usuario, el cual consta del proceso físico que se está controlando."* [NORTHCUTT, 87]

En los sistemas de Tiempo-Real se considera a una "tarea" como la unidad elemental de proceso. Las tareas se clasifican respecto a su ejecución en "expulsivas" (Preemptive) y "no expulsivas"

(Non-Preemptive), las tareas expulsivas son aquellas que pueden ser interrumpidas en cualquier instante y continuar su ejecución en algún momento posterior. Por otro lado las tareas no expulsivas son aquellas tareas que no pueden ser interrumpidas durante su ejecución, una vez iniciadas se debe esperar a que terminen para poder asignar el recurso a alguna otra tarea.

Respecto al plazo para terminar su ejecución las tareas se clasifican como de plazo duro o inflexible (Hard-Deadline) y de plazo suave o flexible (Soft-Deadline). Las tareas de plazo duro son aquellas que en caso de no terminar en o antes de su plazo perentorio, ocasionan una falla catastrófica en el sistema. En contraste, las de plazo suave en ocasiones también llamadas de "mejor esfuerzo" son las tareas que estadísticamente pueden cumplir sus plazos, sin que el no cumplirlo ocasionalmente ocasione una falla del sistema [LIU, 00].

Respecto a sus características temporales la tareas se clasifican en periódicas, periódicas. y esporádicas. Periódicas son las tareas que se ejecutarán a intervalos constantes durante la operación del sistema (períodos). Aperiódicas son las tareas eventuales que pueden tener ya sea un plazo suave o sin plazo para terminar su ejecución. Por último las tareas esporádicas son aquellas tareas eventuales que tienen un plazo duro para concluir su ejecución.

El Consumo de una tarea es el tiempo en el que la tarea se ejecuta, considerándose el tiempo más grande o peor caso de ejecución. El motivo de considerar este peor caso es claro desde el punto de vista de la planificabilidad, ya que para poder garantizar que todas las tareas incluidas en un plan de ejecución serán ejecutadas en Tiempo-Real se tiene que considerar que la sumatoria de los tiempos máximos de ejecución de cada una de las tareas, no excede la capacidad de procesamiento temporal del sistema.

Para definir las tareas en Tiempo-Real y sus características se utiliza el modelo $\mathbf{t_i(T_i,C_i,(D_i))}$, en donde:

$\mathbf{t_i}$ = tarea i.
$\mathbf{T_i}$ = Período de ejecución de la tarea i.
$\mathbf{C_i}$ =Tiempo de cómputo de ejecución de la tarea i.

D_i = Plazo de terminación de la ejecución de la tarea i.

Cabe hacer notar que se considera que cada tarea es periódica e independiente de las otras; puede ser expulsada (preempted) sin haber terminado su ejecución; a menos que se aclare lo contrario el período T y el plazo (deadline) D serán idénticos para cada tarea (i.e. $D_i = T_i$) y se usa el modelo simplificado $\mathbf{t_i(T_i,C_i)}$.

2.3 Planificadores

Seguramente el punto medular de un sistema de Tiempo-Real es el algoritmo de planificación, el cual decidirá que tarea será ejecutada por el sistema en cada momento.

En un sistema distribuido de control en Tiempo-Real, son las tareas periódicas las que representan la mayor demanda de recursos.

Un planificador es considerado óptimo, sólo si es capaz de planificar un conjunto de tareas *planificable* y reduciendo al mínimo los tiempos ociosos del sistema.

Cuando un planificador suspende alguna tarea en ejecución, para iniciar otro se le llama expulsivo ("preemptive"), siendo los no expulsivos aquellos que una vez iniciada una tarea no la suspenden hasta que termine la ejecución de la misma.

En forma general se pueden clasificar los planificadores como estáticos o dinámicos [GARCÏA, 03] y a su vez, los dinámicos se subdividen en dos categorías, con prioridades fijas o prioridades dinámicas.

Existe un gran número de planificadores de Tiempo-Real, siendo los más usados "Rate-Monotonic" RM y "Earliest-Dealine_First" EDF. Mientras el primero es estático de prioridades fijas basadas en los períodos de las tareas, el segundo es dinámico basado en la cercanía del plazo de cada tarea, lo cual asigna la prioridad dinámica.

El algoritmo “Rate-Monotonic” asigna a cada tarea una prioridad fija dependiendo de su período de operación, es decir que la tarea con el período más corto tiene la mayor prioridad. El algoritmo EDF asigna la prioridad de manera dinámica, siendo la tarea con mayor prioridad en un momento determinado, la que tenga su plazo (*deadline*) más cercano. Ambos algoritmos consideran que las tareas pueden ser “expulsadas” del procesador en cualquier momento y son considerados óptimos en el sentido de que son capaces de planificar (estática o dinámicamente) cualquier conjunto de tareas planificable por otro algoritmo del mismo tipo (estático o dinámico).Existen un gran número de planificadores de Tiempo-Real, similares a RM y EDF, los que se consideran los más representativos para sistemas que cuentan con un procesador.

El análisis de planificabilidad para el algoritmo EDF se realiza con la siguiente ecuación de [LIU, 00] para calcular el factor de utilización del procesador U:

$$U = \sum_{i=!}^{n} \frac{C_i}{P_i}$$

(2-1)

en donde i tal que 1 < i < n es el número de tarea y n es el total de tareas.

En el caso del algoritmo RM la utilización del procesador [BUTAZZO, 2004] debe de cumplir:

$$U \leq n(2^{1/n} - 1)$$
$$\lim_{n \to \infty} n(2^{1/n} - 1) = \ln 2 \approx 0.69$$

(2-2)

En el caso de la planificación con dos o más recursos o procesadores se busca la coordinación o composición de los planificadores individuales, la investigación en el campo de composición o coordinación, de algoritmos en los sistemas de Tiempo-Real, está enfocada a los modelos de composición jerárquica [ALMEIDA et al., 04; CHAKRABORTY et al., 06; EASWARAN et al., 06; FENG et al., 02-2; MOK et al., 04; REGEHR et al., 02; SHIN, 03].

Los algoritmos de planificación jerárquica, gestionan el uso de un recurso, generalmente la CPU, de la siguiente forma: El recurso se coloca en la cima de la jerarquía y distribuye su uso hacia los niveles inferiores, por medio de un modelo de interfaz (creado por Deng y Liu [DENG et al.,

97]), $I(G_s,G_d)$ en donde G_s, representa la garantía de Tiempo-Real que el nodo padre provee al nodo hijo y G_d representa la garantía de Tiempo-Real que el nodo hijo solicita de nodo padre.

Un marco (framework) de planificación jerárquica, puede ser representado como un árbol o una jerarquía de nodos, donde cada nodo representa un modelo de planificación (ver Figura 4-1). Estos modelos de planificación constan de un algoritmo, una carga de trabajo y un recurso. Para gestionar la disponibilidad del recurso, cada nodo hijo solicita a su nodo padre la garantía de Tiempo-Real que requiere G_d, basándose el su carga de trabajo y el nodo padre contesta con la garantía provista G_s.

Cada modelo de planificación M (nodo) está definido como: $M\ (W,R,A)$, en donde W representa al modelo de carga de trabajo (conjunto de tareas) planificadas por M, R es el modelo del recurso y A es el algoritmo de planificación.

Para modelar W se utiliza el modelo de tareas periódicas $t_i(T_i,C_i)$ previamente descrito en donde cada tarea es definida por su período T y el tiempo de ejecución C. Cabe hacer notar que en este modelo de tareas, cada una de ella es independiente de las otra y pueden ser expulsadas (preempted). W es definida como el conjunto de tareas: $W=\{t_1,t_2,...,t_n\}$.

Para modelar el recurso R se utilizan modelos de recursos particionados y acotados tipo $R_b\ (U_b, L_b)$, en donde U representa el uso del recurso y L el retraso, haciendo una analogía con el modelo de tareas t, U es análoga a C y L representa el retraso que se tiene para poder utilizar el recurso, no es análoga a T que representaría una utilización periódica del recurso. Por ello, Shin y Lee [SHIN, 03] proponen usar como modelo de recursos (R) a $\Gamma\ (\pi,\theta)$ en donde el modelo gama Γ es totalmente análogo al modelo t, siendo (pi) π el período P ,y (teta) θ el número de unidades de tiempo C. Definiendo entonces a través de gamma, que el recurso está garantizado por teta unidades cada pi tiempo.

El algoritmo de planificación A se define como EDF para planificación dinámica y RM para planificación por prioridades fijas. A se define como el conjunto: A={EDF,RM).

La ventaja de utilizar el modelo gamma para la definición del recurso en Tiempo-Real R del modelo genérico, es que el uso del recurso se modela cómo una tarea de Tiempo-Real $t_i(T_i,C_i)$, lo que permite que cada modelo de planificación este definido como *M (W, Γ,A).*

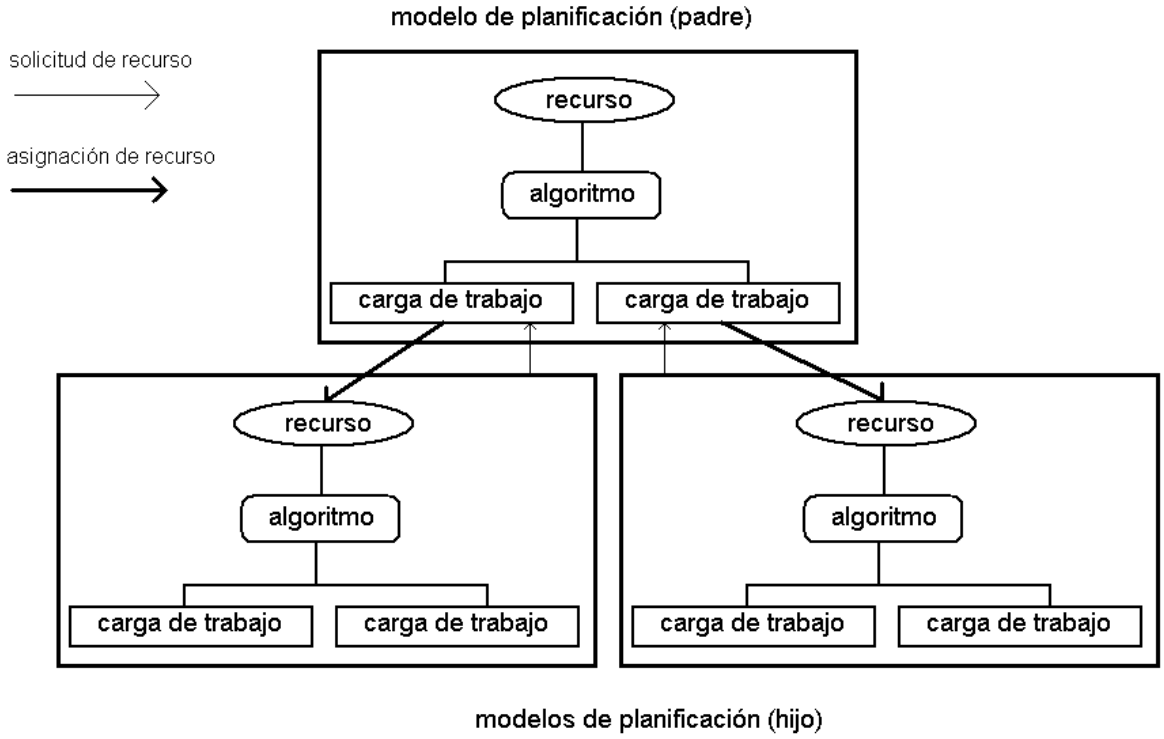


Figura 2-5. Marco de planificación jerárquica: modelo de planificación de padre e hijos[SHIN, 03].

La principal restricción que impone el modelo de planificación jerárquica, es que toda la planificación del sistema depende de un solo recurso.

El utilizar modelos de planificación jerárquica para los sistemas distribuidos en Tiempo-Real (RTDS), plantea la interrogante de cual sería el recurso "Padre" dentro de la jerarquía, ya que no existe un modelo que permita integrar en una sola entidad a los distintos componentes de un sistema distribuido (middleware, sistemas operativos y red de comunicaciones) ver Figura 3-1.

Precisamente el tema de la investigación doctoral, es el encontrar un modelo que permita ver al RTDS como una sola entidad o recurso, planteando como una alternativa el uso de planificadores jerárquicos. El planificador propuesto en el siguiente capítulo aborda este punto.

2.4 Métricas

Para evaluar los sistemas distribuidos desde el punto de vista del Tiempo-Real se propone utilizar tanto algunas métricas existentes como una métrica nueva propuesta en este trabajo y llamada "Métrica de Disponibilidad del Nodo", considerando a una métrica como una medida cuantitativa y periódica, interpretada en el contexto de una serie de mediciones previas equivalentes.

Como se describió anteriormente las tareas de Tiempo-Real pueden ser con plazo duro o suave, en el caso de las tareas con plazo duro el planificador es óptimo si puede planificar un conjunto de tareas de manera que cada una termine dentro del plazo. La métrica es entonces relativamente sencilla, si alguna tarea de plazo duro no puede terminar a tiempo, el sistema es inviable, ya que no es planificable con ese algoritmo.

Para las tareas de plazo suave o flexible, el objetivo es maximizar la calidad de servicio de cada aplicación. Esta calidad de servicio puede ser evaluada con diferentes métricas, siendo una de ellas la que pretende minimizar el error individual o el error total de un conjunto de tareas, considerando al error como el tiempo de procesador que no se asigno a la tarea con plazo suave. En general esto sistemas se evalúan con métricas basadas en la calidad de servicio, minimización del error, mejor esfuerzo o estadísticamente.

Conclusiones

Los sistemas de control por red basan su rendimiento en el período de muestreo (ver Figura 2-1) acotado en ambos extremos. Este tipo de restricción temporal permite que un NCS pueda implementarse como un sistema distribuido en Tiempo-Real.

Existen algoritmos probados para la planificación en Tiempo-Real sobre un procesador, pero en el área de sistemas dinámicos con múltiples procesadores no se cuenta con algoritmos precisos y robustos de planificabilidad que validen este tipo de sistemas distribuidos [LIU, 00].

Una alternativa para garantizar el cumplimiento de las restricciones temporales de un sistema distribuido en Tiempo-Real es el contar con un algoritmo de planificación que permita agrupar los recursos de un RTDS (nodos y red de comunicaciones) en uno solo, como el recurso "Padre" en una jerarquía (ver Figura 2-5). Este recurso único, es en sí la capacidad del RTDS para realizar la planificación de un conjunto de tareas.

Parta determinar si un sistema distribuido en Tiempo-Real cumple con todas sus restricciones temporales de manera eficiente se debe contar con una métrica que permita evaluar el rendimiento de un sistema con estas características.

3 Análisis y modelado de un RTDS del tipo NCS

Como se comentó en el capítulo anterior, un componente fundamental para el estudio de los Sistemas Distribuidos de Tiempo-Real (RTDS por sus siglas en inglés) es su modelado, en este capítulo se presenta un análisis y un modelo de capas para los RTDS, así como su formalización utilizando el álgebra de procesos para ello. Al final del capítulo se plantea el planificador propuesto para los sistemas distribuidos en Tiempo-Real.

3.1 Modelo

Para poder conocer en forma determinista el comportamiento temporal de un sistema distribuido, multimodal y en Tiempo Real, es necesario modelarlo formalmente, para ello se ha definido una arquitectura básica capaz de soportar aplicaciones en Tiempo-Real. En dicha arquitectura, se propone montar las aplicaciones sobre un sistema de tipo intermedio medio "*middleware"* en Tiempo-Real, el que a su vez esta soportado por un sistema operativo en Tiempo-Real y tiene como base una red de comunicaciones también en Tiempo-Real. Esquemáticamente el modelo es el siguiente:

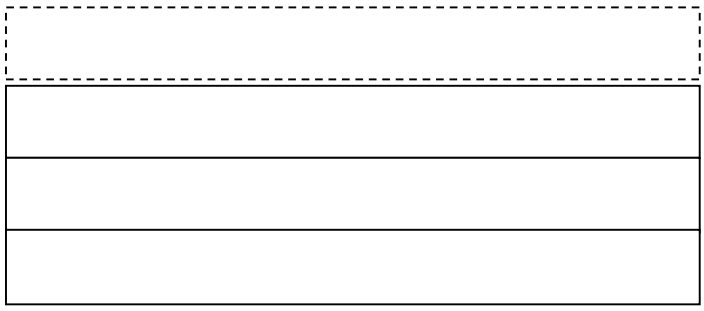


Figura 3-1. Modelo de tres capas de un sistema distribuido en Tiempo-Real

Las diferentes teorías de álgebras de procesos han sido creadas para modelar formalmente sistemas concurrentes[BATEN et al., 90], usando un lenguaje de alto nivel para describir tanto a los procesos como a los sistemas. Esto permite la descripción de la interacción y sincornización de procesos independientes como comunicación entre ellos, no como modificación de variables

compartidas. El álgebra de procesos provee de reglas algebraicas para analizar, manipular y minimizar las descripciones de los procesos.

La “bisimulación” (el estado del arte en cuanto a equivalencias de procesos se refiere) permite un razonamiento formal para establecer equivalencias entre procesos. Esto implica no solamente que dos gráficas de procesos puedan ejecutar las mismas cadenas de acciones, sino que también tengan la misma estructura desde el punto de vista de sus ramificaciones. La bisimulación [BATEN et al., 02] permite tener una equivalencia adecuada cuando se trata de razonar acerca de procesos concurrentes. Puede ser utilizada para la verificación de sistemas asíncronos y síncronos.

La notación textual por medio de ecuaciones (estilo algoritmo), es más adecuada para representar la composición jerárquica y manipular las especificaciones parametrizadas de los sistemas concurrentes, que las notaciones gráficas (p. ej. Petri nets)

El álgebra de procesos fue seleccionada en este trabajo al ser considerada como una herramienta útil como marco para poder razonar formalmente sobre los procesos y los datos, así como para el modelado formal de los sistemas concurrentes, permitiendo también realizar su verificación.

Símbolos y términos del álgebra de procesos utilizados.

BPA	Álgebra de Procesos Básica (Basic Process Algebra).
RTPA	Álgebra de Procesos de Tiempo-Real (Real-Time Process Algebra).
+	Composición alternativa.
•	Composición secuencial.
⌊⌊	Unión izquierda (Composición paralela y comunicación).

σ^{t}_{rel} Retraso relativo.

$\sum_{t}$ Sumatoria (Composición alternativa múltiple).

$\int_{[0,\infty)}$ Integral (Proceso que es capaz de realizar una acción en todos los puntos de un intervalo de tiempo).

A continuación se describen cada una de las capas del modelo y se utiliza el Álgebra de Procesos [BATEN et al., 90 y 02 y FOKKINK, 00] para modelar formalmente el sistema [WANG, 02] y sus componentes [YAO et al., 04].

Red de Comunicaciones

Las comunicaciones entre los procesadores están representadas en la capa inferior del modelo, llamada (RC) "Red de comunicaciones" (Figura 3-1), se plantea que se debe estudiar dicha capa considerando las comunicaciones en Tiempo-Real (T-R) desde el punto de vista del protocolo, independientemente de la estructura física de la red.

La RC debe estar distribuida entre dos tipos de tráfico, el periódico o síncrono y el eventual o asíncrono, y se representa en AP (álgebra de procesos) por:

$$RC = (MS + MA + to) \bullet RC \qquad (3\text{-}1)$$

Donde MS = mensajes o tráfico síncronos, MA = mensajes o tráfico asíncronos y to = tiempo ocioso o no-transmisión.

Esta ecuación solamente muestra a la RC que está desempeñando alguno de los tres procesos por los que puede optar, pero no toma en cuenta las restricciones temporales del envío de mensajes, es decir que estos cumplan con su plazo perentorio (*"deadline"*), característica indispensable para que la RC sea considerada como de T-R.

Para poder garantizar el cumplimiento de los plazos en el envío de mensajes, el ancho de banda de la RC se distribuye en dos "ventanas" temporales, una para los mensajes asíncronos y la otra para los síncronos quedando la ecuación de la RC de la siguiente forma:

$$RC = (MA + to) \bullet \sigma_{rel}^{LMA} (\underline{MS}) \bullet RC \qquad (3\text{-}2)$$

Donde *LMA* = Longitud de los Mensajes Asíncronos.

Esta ecuación indica que la RC podrá estar transmitiendo mensajes asíncronos, o esperando por ellos, durante el inicio de cada ciclo, con una duración de LMA. Una vez transcurrido el tiempo asignado para la LMA, la RC iniciará la transmisión de los mensajes síncronos.

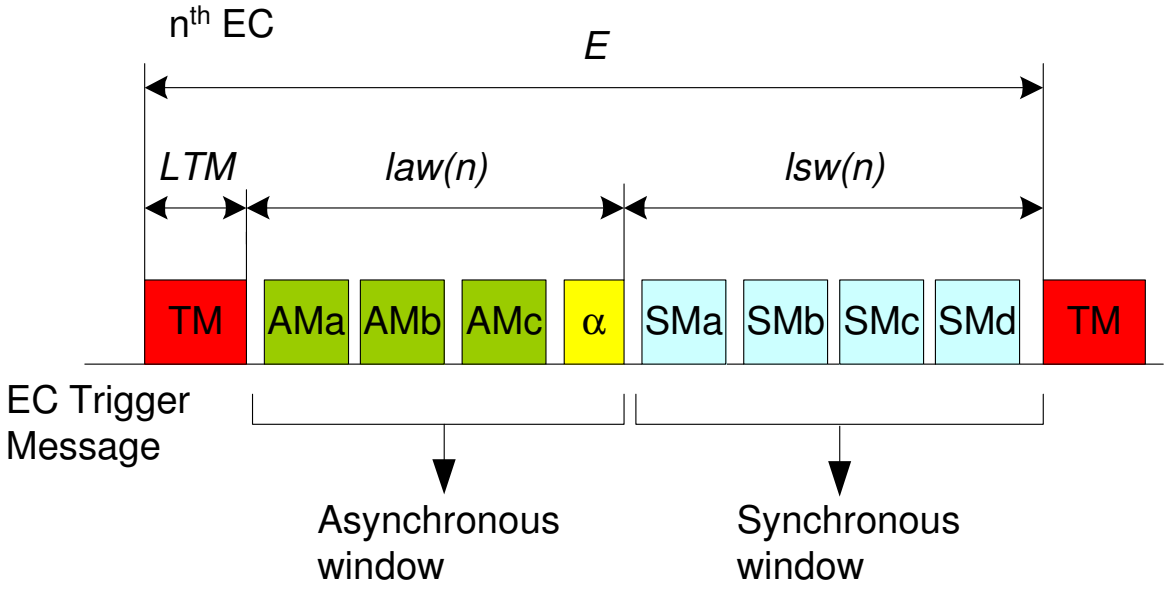


Figura 3-2. Ventana Asíncrona en FTT-CAN. [ALMEIDA et al., 04].

Sistema Operativo

La capa del Sistema operativo en T-R (**RTOS** Real-Time Operating System) actúa como interfaz entre el *middleware* y la red de comunicaciones, proporcionando todos los servicios y recursos básicos que el *middleware* requiere (Figura 3-1), su ecuación es la siguiente:

$$RTOS = Procesador \parallel RT\text{-}Scheduler \parallel Recursos \quad (3\text{-}3)$$

Dependiendo de las políticas del RTOS, el procesador será utilizado por el proceso de planeación (**RT-Scheduler**) o los recursos (**Recursos**).

$$Procesador = RT\text{-}Scheduler + Recursos \quad (3\text{-}4)$$

Donde los recursos son el manejo básico de memoria, soporte para multiproceso y multihilos, eventos e interrupciones, así como los servicios básicos de comunicación.

Middleware

Las aplicaciones usan como interfaz al middleware, el cual tiene la función primordial de "crear" un sistema distribuido transparente para ellas, permitiendo con esto que las aplicaciones no tengan la necesidad de "saber" en que sistemas operativos o equipos *("hardware")* se está procesando la información que requieren, ni que existe una red de comunicaciones y las características de esta (Figura 3-1).

El middleware tiene como propósito permitir a las aplicaciones utilizar todos los objetos de un sistema multimodal.

Al utilizar un gestor de requerimientos de objetos ORB ("*Object Request Broker"*) por su siglas en inglés, como sistema medio, se logra el propósito antes mencionado, pues el ORB implementa

la funcionalidad necesaria para que cada aplicación que requiere de un objeto, lo opere como si fuese una invocación local, es decir la aplicación desconoce si el objeto que desea operar se encuentra dentro de ella misma, en el mismo equipo en otra aplicación o en algún otro equipo dentro del sistema distribuido.

La ecuación del ORB de Tiempo-Real es la siguiente

$$\text{RT-ORB} = \text{RT-Scheduler} \parallel \text{OS} \parallel \text{RT-ObjectAdapter} \quad (3\text{-}5)$$

En donde el RT-ORB se encarga de coordinar al planificador de Tiempo-Real (**RT-Scheduler**), al sistema operativo (**OS**) y al adaptador de objetos en Tiempo-Real (**RT-ObjectAdapter**).

3.2 **Planificador propuesto**

Un sistema distribuido en Tiempo-Real (RTDS) es un sistema distribuido, en el que cumplir con las restricciones temporales es el principal objetivo. Dos son los principales problemas que hay que resolver; el primero es que no existe un concepto homogéneo del tiempo, es decir que cada nodo tiene su propio mecanismo interno de reloj y el segundo problema, ocasionado en parte por el primero, es que no existe un planificador que pueda garantizar el comportamiento temporal del sistema distribuido.

Con el propósito de garantizar el cumplimiento de los plazos de las tareas en un RTDS y para tener un mejor aprovechamiento de los recursos que componen dicho sistema, se propone tener un planificador global llamado “rt-Scheduler” ver Figura 3-3.

El análisis y la formalización del modelo del RTDS permiten plantear la necesidad de utilizar un planificador que permita garantizar el comportamiento temporal de todo el sistema (Figura 3-3), con lo que surge el modelo del RTDS con un planificador global. Este planificador global se

encarga de garantizar que una tarea cumpla su plazo en las distintas capas del modelo, añadiendo los retrasos inducido por la planificación y ejecución de las tareas a través de las distintas capas.

En la Figura 3-3 se aprecia como un conjunto de tareas, (en este caso 3) pueden aparecer como consecutivas en la capa superior (middleware), sin embargo debido a la velocidad de procesamiento de las capas inferiores del sistema operativo (O.S.) y la red (Network), estas tareas no son realizadas de manera consecutiva en el tiempo, dejando espacios temporales ociosos (los rectángulos negros) que pueden ser utilizados en la ejecución de otras tareas. El planificador global es el encargado de administrar estos tiempos ociosos de procesamiento y de validar la ejecución temporal de cada tarea a través de las tres capas del modelo.

La ecuación de álgebra de procesos que representa dicho modelo es:

$$\text{RTDS} = \text{rt-Scheduler} \parallel (\text{CN} \mid \text{rt-Scheduler}) \parallel (\text{RTOS} \mid \text{rt-Scheduler}) \parallel (\text{Middleware} \mid \text{rt-Scheduler}) \quad (3\text{-}6)$$

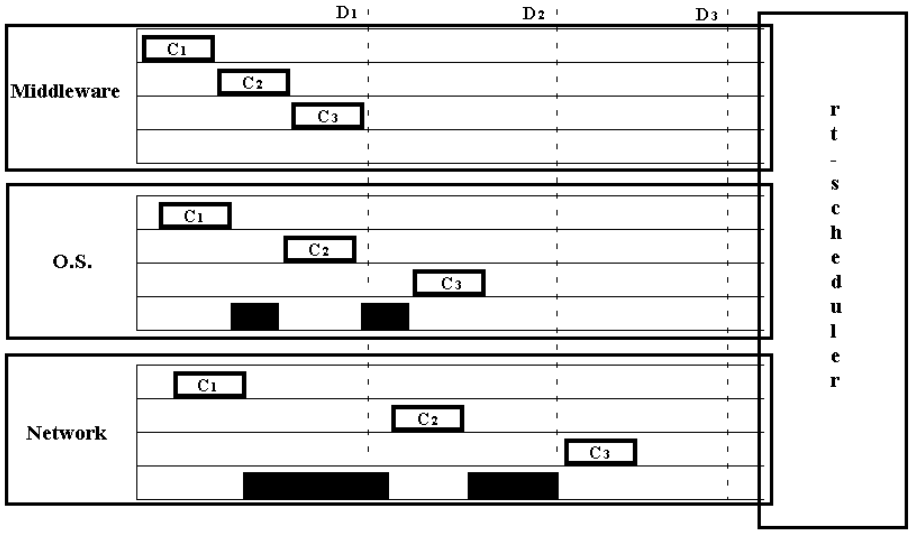


Figura 3-3. Modelo de capas de un RTDS con planificador global[MENÉNDEZ et al., 07-1].

Para su implementación, es necesario que cada nodo participante y la red de comunicaciones, reserven espacio para un proceso de memoria compartida (Shared memory), con la prioridad más alta. Este proceso permite la coordinación de los nodos y de las comunicaciones, así como la garantía de Tiempo-Real. En la Figura 3-4 se presenta un diagrama de bloques mostrando la idea. En la parte superior de la figura se aprecian dos rectángulos que representan dos nodos del RTDS, cada uno de ellos tiene una tarea de la más alta prioridad llamada rt-sched-proc y su conjunto respectivo de tareas que se ejecutarán con una prioridad menor asignada por el algoritmo de planificación local seleccionado (como ejemplo se utiliza EDF para el nodo 1 y RM para el nodo 2).

.

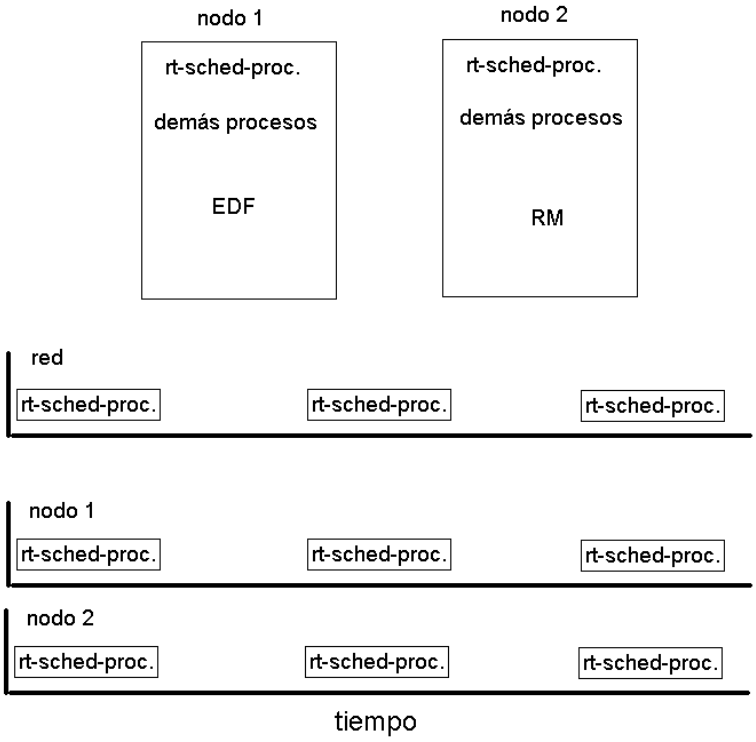


Figura 3-4. Planificador propuesto

En la parte inferior de la Figura 3-4 se aprecian las tres líneas de tiempo que corresponden a la red y los nodos de RTDS, el planteamiento general es que se defina un tiempo en el que periódicamente se actualizará la información del planificador global. Este tiempo se define como período "base" del RTDS.

Este período “base” puede ser utilizado por todos los nodos y la red para la tarea rt-sched-proc. Sin embargo y con la finalidad de tener una mayor holgura en la planificación de las tareas en cada nodo del RTDS, se plantea que los períodos operacionales puedan tener distintas duraciones en cada nodo. La forma propuesta para lograrlo es utilizar un factor de “dispersión”, que junto con el valor de “base” se utiliza para calcular el período operacional de cada nodo usando la siguiente ecuación:

$$\text{periodo}_i = \text{base} \times (1 \pm \text{dispersión}) \qquad (3\text{-}7)$$

en donde período_i = período operacional de la tarea rt-sched-proc en el nodo i, base es el período “base” y $0 \leqslant$ dispersión $\leqslant 1$ es el factor de dispersión.

El valor del factor de dispersión permite en el extremo inferior que el período de operación de los nodos sean iguales al período del “base” (cuado dispersión = 0) y en el otro extremo cuando “dispersión” aumenta de valor los períodos serán en dicha proporción más grandes (si se suma el valor de dispersión a 1) o más pequeños (cuando dispersión se resta de 1). Obviamente la “dispersión” puede ser vista como el porcentaje que “base” aumenta o disminuye en cada nodo, y se plantea que dependiendo del RTDS que se analice “dispersión” tiene un valor máximo cercano a 0.25 antes de que el RTDS no presente un desempeño aceptable. Esta estrategia de planificación se ilustra con el caso de estudio 5.2.

Conclusiones

El área de la planificación de sistemas en Tiempo-Real ha sido estudiada desde el punto de vista de un sistema de cómputo que consta por lo general de un procesador, sin embargo el área de los sistemas distribuidos en Tiempo-Real está abierta en problemas como la composición de planificadores y la medición del desempeño del sistema. Por ello se propone la utilización de un planificador global.

El modelado en capas del sistema distribuido en Tiempo-Real permite ver las relaciones existentes entre estas capas y las tareas ejecutándose en cada una de ellas por lo que la propuesta de utilizar un planificador global queda justificada.

4 Métrica de disponibilidad del nodo

Independientemente del sistema operativo con el que cuente un nodo, una vez que los procesos están en ejecución demandan dos recursos primordiales: el uso de Procesador y asignación de memoria (ram), por esta razón es que estas dos medidas son las utilizadas como base para definir la métrica de "Disponibilidad del Nodo" [MENÉNDEZ, et al., 07-2]. Considerando a un nodo como una unidad de cómputo que consta de al menos: un procesador, memoria (ram) y un dispositivo de acceso a la red de comunicaciones.

En la Figura 4-1 se presenta una gráfica que muestra el uso del procesador y la memoria de un sistema durante la ejecución de un proceso que tarda aproximadamente 100 segundos representados en el eje inferior. El eje izquierdo representa el porcentaje de disponibilidad, y se grafican la disponibilidad de la memoria y el porcentaje tiempo que el procesador está disponible.

El proceso de ejemplo sirve para ver que al inicio de su ejecución se le asigna todo el tiempo el procesador y la memoria, el procesador eventualmente realizará otras tareas del sistema operativo lo que se aprecia en los segundos iniciales y finales de la ejecución. Se aprecia también que el proceso termina su ejecución aproximadamente en el segundo 79 liberando el uso del procesador, pero no así la memoria, la cual es liberada hasta después del segundo 103.

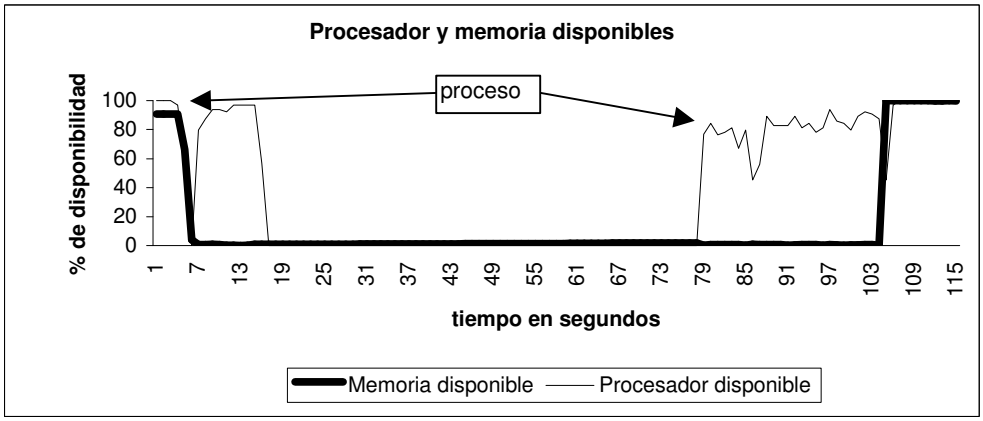


Figura 4-1. Procesador y la memoria disponibles durante la ejecución de un proceso

La gráfica de la Figura 4-1 muestra de manera contundente que el uso del procesador y el de la memoria son dos métricas indispensables para poder evaluar la disponibilidad de un nodo.

4.1 Métrica Phi

La métrica de disponibilidad del nodo **Φ** (phi), ha sido creada para permitir conocer de manera seudo dinámica la diferencia de disponibilidad entre dos nodos con la misma o similar capacidad. La primera ventaja que presenta estriba en que la información que brinda proviene de un sistema en operación, en contraposición a la métrica de utilización del procesador "U" calculada de manera teórica con la ecuación (2-1). Otro motivo adicional para utilizar **Φ** radica en el hecho de los elementos que se utilizan para su cálculo, ya que se toman en cuenta no solamente el uso del procesador (como en "U") sino también el uso de la memoria, el número de procesos en la cola de ejecución y por ello el costo en tiempo inherente al cambio de contexto normal a cualquier sistema operativo, y por último y característico de los sistemas distribuidos, el número de procesos comunicándose y los tiempos asociados a está comunicaciones ya sean internas o externas. En resumen, al considerar las características de los sistemas distribuidos y la sobrecarga de trabajo que estos imponen a los nodos que lo integran, la métrica refleja de manera cabal la disponibilidad de cada nodo de un sistema distribuido, permitiendo comparar las diferencias empíricas que existen entre dichos nodos al momento de su operación.

La métrica de disponibilidad del nodo **Φ** (phi) esta dada por la ecuación:

$$\Phi = \frac{\alpha\beta}{e^{(\varepsilon+\rho\kappa+\tau\upsilon)}} \qquad (4\text{-}1)$$

en donde:

Tabla 4-1. Variables para el cálculo de Φ

α	Porcentaje de tiempo que el procesador del nodo está ocioso durante la muestra.
β	Porcentaje de memoria (ram) disponible en el nodo durante la muestra.

ε	Número de procesos en la cola de ejecución del nodo durante la muestra.
κ	Número de procesos en ejecución por el nodo durante la muestra, con comunicaciones internas.
ν	Número de procesos en ejecución por el nodo durante la muestra, con comunicaciones externas.
ρ	Tiempo que toma una comunicación interna.
τ	Tiempo que toma una comunicación externa.

La razón para multiplicar los porcentajes de tiempo disponible del procesador y memoria disponible en la ecuación (4-1), es encontrar el nodo más disponible y balanceado considerando ambos criterios. No se estima conveniente el sumar los porcentajes ya que nodos evidentemente más desbalanceados obtendrían el mismo factor parcial de disponibilidad $\alpha\beta$ (Ej. Si se suman se obtiene: 9+1 = 5+5 = 2+8 por otro lado multiplicándolos se obtiene que: 9*1 < 2*8 < 5*5).

4.2 Algoritmo HILO

Para realizar la distribución y el balanceo de cargas en un sistema distribuido se presenta la adaptación de la métrica de disponibilidad del nodo a un conocido algoritmo para balanceo de cargas llamado HILO.

HILO necesita conocer la disponibilidad de cada nodo y con ella se define al nodo $\mathbf{\Phi_{min}}$ como al nodo que presenta la menor disponibilidad y el que tiene la mayor disponibilidad es $\mathbf{\Phi_{max}}$ esto es:

$$\Phi = \{\Phi_1, \Phi_2, ..., \Phi_n\}$$
$$\Phi_{max} = \max(\Phi)$$
$$\Phi_{min} = \min(\Phi)$$

(4-2)

El sistema distribuido se considera balanceado cuando la disponibilidad de todos sus nodos es igual o similar. La "desigualdad" del sistema se calcula con los dos nodos que tienen respectivamente la disponibilidad menor y la mayor. Con estos valores (mínimo y máximo de disponibilidad), se calcula el factor de balanceo (μ) del sistema distribuido con la siguiente ecuación:

$$\mu = \frac{\Phi_{min}}{\Phi_{max}} \qquad (4\text{-}3)$$

donde Φ_{min} es el valor correspondiente a la disponibilidad del nodo con menor disponibilidad y Φ_{max} es el valor correspondiente al nodo con mayor disponibilidad del DS.

Se aprecia que cuando los valores de Φ_{min} y Φ_{max} son similares μ tiende a uno, indicando con ello que el sistema esta balanceado, es decir que los nodos tienen disponibilidades similares o iguales. Por otro lado cuando el nodo con mayor disponibilidad Φ_{max} tiene un valor cercano a la unidad y el nodo menos disponible tiene un valor cercano a cero el valor de μ tiende también a cero. Queda claro que un sistema balanceado tiene un valor de μ cercano a la unidad y conforme este valor sea menor, el sistema esta menos balanceado.

Cuando llega una tarea nueva HILO la asigna al nodo más disponible (Φ_{max}). Posteriormente cada δ muestras utilizando Φ_{min} y Φ_{max}, se calcula el nivel de balanceo del sistema (μ) usando la ecuación (4-3). Si el factor de balanceo calculado es menor que el factor de balanceo deseado, entonces HILO toma la última tarea en la cola de ejecución del nodo Φ_{min} y la migra al nodo Φ_{max}.

El seudo código del algoritmo HILO y los distintos métodos que utiliza es el siguiente:

El método de asignación de un proceso nuevo se llama "Scheduler", e invoca al método de calcular la carga de los nodos (Node_Loa") y asigna el proceso nuevo (NewProc) al nodo más disponible (Φ_{max}):

Scheduler

Node_Load

Add_Queue(Φ_{max}) NewProc

El seudo código del algoritmo HILO que se ejecuta cada δ muestras es el siguiente:

HILO

sleep (δ)

Node_Load

if $\Phi_{min} / \Phi_{max} < \mu$ then *Balance*

HILO

El método "Node_load" calcula la disponibilidad de cada nodo i, en la muestra j tomado δ muestras. También asigna el identificador del nodo más disponible a Φ_{max} y el del menos disponible a Φ_{max}:

Node_Load

For i=1 to n

Evaluate Φ (i)

next i

Φ_{max} = MAX (Φ)

Φ_{min} = MIN (Φ)

El método para calcular la carga de los nodos (Node_Load) se ejecuta cada ocasión en que un proceso nuevo llegua o cada balanceo del DS, el cual es efectuado de forma periódica cada δ muestras.

El método "Balance" selecciona al último proceso en la cola de Φ_{min} que se denomina "LastProc", lo elimina de la cola y lo asigna al nodo Φ_{max}.

Balance

LastProc = Find(Last_Proc(Queue(Φ_{min})))
Remove_Queue(Φ_{min})
Add_Queue(Φ_{max}) LastProc

4.3 Análisis de la métrica

A continuación se presenta el análisis de un sistema distribuido en Tiempo-Real (RTDS) basándose en la métrica de disponibilidad del nodo [MENÉNDEZ et al., 07-2]. El objetivo de este análisis es el utilizar la métrica de disponibilidad del nodo Φ (phi) para poder apreciar el impacto que tienen el uso de la memoria y las comunicaciones además del uso del procesador en un sistema distribuido en Tiempo-Real.

La métrica de disponibilidad del nodo es propuesta con el propósito de analizar el desempeño de los nodos de un sistema distribuido y en particular de un sistema distribuido en Tiempo-Real (RTDS). Considerando a un nodo como una unidad de cómputo que consta de al menos: un procesador, memoria (ram) y un dispositivo de acceso a la red de comunicaciones.

El análisis teórico de planificabilidad de Liu-Layland (U), plantea que un conjunto de tareas de Tiempo-Real, es planificable en un procesador siempre y cuando el uso de este sea menor o igual al 100%, este análisis no toma en consideración el uso integral del nodo para realizar las tareas inherentes a un sistema distribuido. La métrica de disponibilidad del nodo (Φ) es diseñada precisamente para tomar en consideración los elementos adicionales al uso del procesador que intervienen en el desempeño de cada nodo dentro de un sistema distribuido. Estos elementos adicionales son: el porcentaje de memoria (ram) disponible, el número de procesos en la cola de ejecución del procesador, el número procesos sosteniendo comunicaciones con otros procesos

del nodo y el número de procesos que se comunican con procesos que están siendo ejecutados en otros nodos del sistema distribuido.

Las razones para utilizar Φ como medida de disponibilidad para un sistema distribuido de tipo NCS son las siguientes:

- Los procesos que se ejecutan en un DS generalmente son de consumo intensivo de procesador, de memoria o ambas, no son procesos que se caractericen por tener un alto grado de entradas y salidas (I/O).

- El comportamiento general de los sistemas operativos al momento de ejecutar un proceso, es el de "reservar" la memoria que el proceso en ejecución requiere tratando de usar toda la memoria disponible y por lo tanto disminuyendo la disponibilidad de esta (Figura 4-1.). Posteriormente el procesador será utilizado conforme el proceso lo demanda. De forma análoga, lo primero que libera un proceso terminado es el procesador y posteriormente la memoria. En la gráfica de la Figura 4-1 se muestra como la disminución de disponibilidad de la memoria es más constante que la disminución en la disponibilidad del procesador. También se aprecia como el procesador está disponible prácticamente al instante de terminar la ejecución de algún proceso, mientras la liberación de la memoria toma más tiempo.

- No se conocen los tiempos globales de ejecución de cada proceso, ya que de saberlos, el balanceo y la planificación de la ejecución del conjunto de procesos puede ser realizado por medio de diferentes algoritmos ya probados [CHIASSON et al., 05; GROSU, 04].

La métrica Φ se calcula para cada uno de los nodos del RTDS, tomándose muestras periódicas en ellos de: α,β,ε,κ y υ. Las cuales constituyen las variables de muestreo, mientras que los factores de comunicación (ρ y τ) son valores calculados en cada muestra. El valor de Φ va desde 0 que significa un nodo totalmente saturado y 1 que significa que el nodo está disponible. Este valor es análogo al factor de utilización U (calculado con la ecuación (2-2) que si bien tiene el mismo rango de valores (entre cero y uno), el significado es inverso. Un procesador con

utilización U = 1, está totalmente ocupado mientras que un nodo con disponibilidad Φ = 1 está totalmente desocupado y viceversa.

Para calcular los valores de los factores de comunicación (ρ y τ) de la métrica Φ, se plantea que estos valores se ven afectados por el grado de balanceo del RTDS, ya que en un nodo con menor disponibilidad estos factores serán mayores que en un nodo con mayor disponibilidad.

Se estipula que un RTDS tiene un uso uniforme o eficiente de los recursos cuando las cargas impuestas a cada nodo son equivalentes, es decir que existe un balanceo de cargas. De esta forma, podemos definir al error del sistema Ω (omega) como la magnitud de desbalanceo existente en el RTDS, es decir:

$$\Omega = \frac{(1-\mu)^2}{2}$$

(4-4)

Para poder minimizar el error (Ω) del DS, se requiere que el sistema este balanceado, es decir que μ tienda a uno, esto es:

$$\text{si} \quad \mu \to 1$$
$$\text{entonces} \quad \Omega \to 0$$

(4-5)

Para lograr disminuir el error del sistema y por ende el valor de Ω se plantea encontrar el valor mínimo del mismo utilizando el método de gradiente descendiente por lo que hay que calcular las derivadas parciales de Ω respecto a los factores de comunicación (ρ y τ) de los nodos con máxima y mínima disponibilidad respectivamente:

$$\frac{\partial \Omega}{\partial \rho_{\max}}, \frac{\partial \Omega}{\partial \rho_{\min}}, \frac{\partial \Omega}{\partial \tau_{\max}}, \frac{\partial \Omega}{\partial \tau_{\min}}$$

(4-6)

La ecuación de omega que calcula el error del sistema en la muestra j, utilizando al nodo con Φ_{min} y al nodo con la Φ_{max} es:

$$\Omega^j = \frac{1}{2}\left(\frac{\Phi^j_{\min}}{\Phi^j_{\max}}\right)^2 = \frac{1}{2}\left(1 - \frac{\alpha^j_{\min}\beta^j_{\min}}{\alpha^j_{\max}\beta^j_{\max}} e^{\left(\varepsilon^j_{\max} - \varepsilon^j_{\min} + \rho^j_{\max}\kappa^j_{\max} - \rho^j_{\min}\kappa^j_{\min} + \tau^j_{\max}\upsilon^j_{\max} - \tau^j_{\min}\upsilon^j_{\min}\right)}\right)^2$$

(4-7)

agrupando la parte derecha de la ecuación anterior en λ_1 y λ_2 de la siguiente manera:

$$\lambda_1^j = \frac{\alpha^j_{\min}\beta^j_{\min}}{\alpha^j_{\max}\beta^j_{\max}}$$
$$\lambda_2^j = \left(\varepsilon^j_{\max} - \varepsilon^j_{\min} + \rho^j_{\max}\kappa^j_{\max} - \rho^j_{\min}\kappa^j_{\min} + \tau^j_{\max}\upsilon^j_{\max} - \tau^j_{\min}\upsilon^j_{\min}\right)$$

(4-8)

permite expresar a Ω como:

$$\Omega^j = \frac{1}{2}\left(1 - \lambda_1^j e^{\left(\lambda_2^j\right)}\right)^2$$

(4-9)

Las derivadas parciales (en la j ésima muestra) son las siguientes:

$$\frac{\partial\Omega^j}{\partial\rho^j_{\min}} = (1-\lambda^j)(\lambda_1^j)(-\kappa^j_{\min})e^{\lambda_2^j} \qquad \frac{\partial\Omega^j}{\partial\rho^j_{\max}} = (1-\lambda^j)(\lambda_1^j)(\kappa^j_{\max})e^{\lambda_2^j}$$
$$\frac{\partial\Omega^j}{\partial\tau^j_{\min}} = (1-\lambda^j)(\lambda_1^j)(-\nu^j_{\min})e^{\lambda_2^j} \qquad \frac{\partial\Omega^j}{\partial\tau^j_{\max}} = (1-\lambda^j)(\lambda_1^j)(\nu^j_{\max})e^{\lambda_2^j}$$

(4-10)

donde: $\lambda = \lambda_1\lambda_2$

lo que permite obtener los valores de (ρ y τ) que se utilizarán en j+1, con las siguientes ecuaciones:

$$\rho^{j+1}_{\min} = \rho^j_{\min} + \eta\frac{\partial\Omega^j}{\partial\rho^j_{\min}} \qquad \rho^{j+1}_{\max} = \rho^j_{\max} + \eta\frac{\partial\Omega^j}{\partial\rho^j_{\max}}$$
$$\tau^{j+1}_{\min} = \tau^j_{\min} + \eta\frac{\partial\Omega^j}{\partial\tau^j_{\min}} \qquad \tau^{j+1}_{\max} = \tau^j_{\max} + \eta\frac{\partial\Omega^j}{\partial\rho^j_{\max}}$$

(4-11)

donde: η es un factor de diseño o velocidad de aprendizaje que permite junto con las derivadas parciales que Ω converja a cero y obtener los tiempos de las comunicaciones internas (ρ) y externas(τ).

Para demostrar el uso de la métrica Φ como herramienta de análisis para un sistema distribuido en Tiempo-Real (RTDS) se realizó una simulación que consta de dos casos en los que se compara la utilización U de los procesadores contra la métrica Φ.

La simulación contempla la ejecución de varias tareas periódicas de duración variable en los nodos que lo conforman, estas tareas son independientes entre sí y por lo tanto pueden ser migradas entre los nodos del RTDS.

Caso 1. En un RTDS que consta de 10 nodos, se asignan a cada uno hasta 5 tareas periódicas de consumos y períodos aleatorios. Durante 100 épocas o muestras se evalúa para cada nodo el valor de Φ y el de U. Ambos valores están comprendidos entre cero y uno y se calculan respectivamente con las ecuaciones (4-1) y (2-1) Ya que la métrica de disponibilidad del nodo (Φ) representa de manera muestral el comportamiento del mismo, su valor debe ser complementario al de la utilización del procesador (U), de forma que teniendo un número suficiente de muestras converja en: $\Phi = 1 - U$.

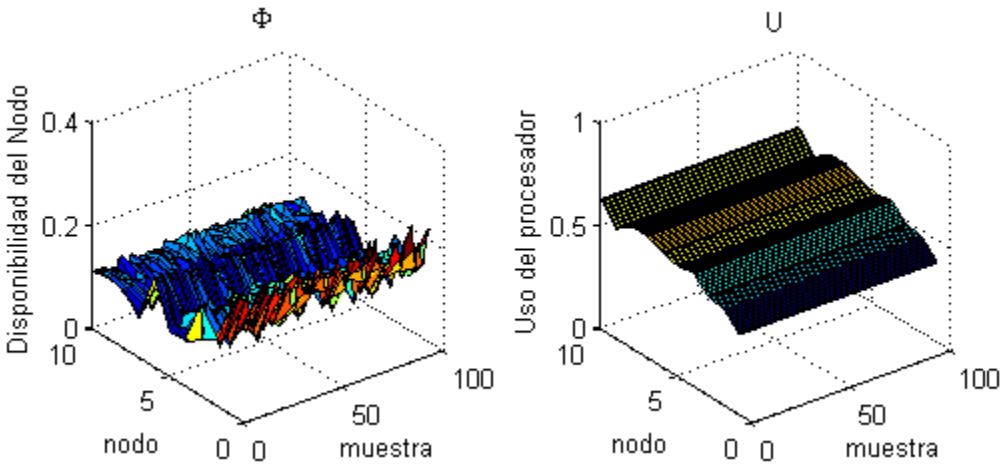


Figura 4-2. Comportamiento de Φ y de U (Caso1)

En la Figura 4-2 se observa el comportamiento del RTDS durante las 100 muestras, el valor de U se mantiene fijo en cada nodo ya que no cambia la asignación de tareas, sin embargo el valor de Φ sí lo hace en cada muestra ya que el uso del procesador y la memoria varían en el tiempo y esto no es reflejado por U.

Caso 2. El RTDS consta de 10 nodos sin asignación de tareas periódicas al inicio de la prueba, las tareas se asignan de manera aleatoria a partir de la muestra 2 y hasta la muestra 600. En cada muestra se genera una tarea con duración y período aleatorios, esta tarea es añadida a la cola de ejecución del nodo más disponible, es decir el que tiene el valor de Φ_{max}, siempre y cuando el factor de utilización (U) de dicho nodo no exceda 1.

A partir de la muestra 2 se realiza el balanceo de cargas, dicho balanceo se realiza migrando la tarea con menor factor de utilización (U) del nodo con menor disponibilidad Φ_{min} al nodo con mayor disponibilidad Φ_{max}, cuando el error (Ω) del sistema es mayor a 0.15. En este caso se realizaron 681 migraciones o balanceos. En la gráfica de la izquierda se muestra el comportamiento de Φ y en la derecha el de U, apreciándose como el valor de Φ es más sensible y por ende presenta una mayor variación que U.

Como muestran las siguientes gráficas (Figura 4-3), a partir de la muestra 600 aproximadamente, el sistema tiende a ser estable y estar prácticamente balanceado, lo que permite que los valores de Φ y U tengan un comportamiento similar y análogo, ya que el valor de Φ debe converger con el de U -1.

Al utilizar la métrica de disponibilidad del nodo (Φ) como un criterio para realizar una distribución de tareas entre los nodos del RTDS, se logra en algunos casos obtener un RTDS con grado de utilización (U) homogéneo o similar en cada uno de los nodos que lo integran.

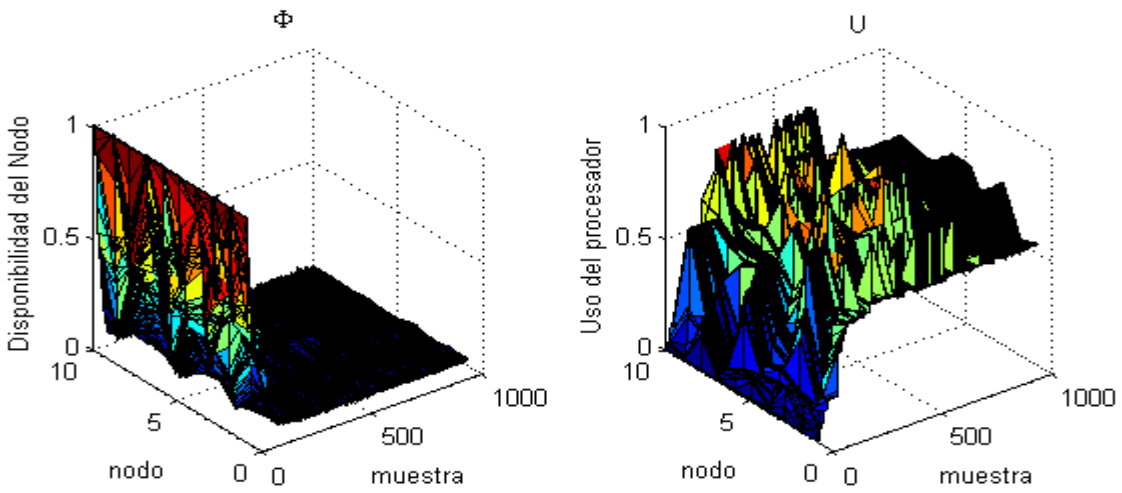


Figura 4-3. Comportamiento de Φ y de U (Caso 2)

El valor de las derivadas crece hacia +- infinito, por lo que los valores de Φ de todos los nodos convergen en cero, con una rapidez que depende del valor del factor de aprendizaje "eta" (ecuación (4-11). Cabe recalcar que al encontrar el valor de "tau", es decir el tiempo que debe tomar una comunicación externa, se encuentra la velocidad que debe tener la red de comunicaciones de este RTDS.

4.4 Comportamiento de Phi

La siguiente ecuación (análoga a la ecuación (4-1) estima la disponibilidad (Φ) del i-ésimo nodo, en la j-ésima muestra, calculando el promedio aritmético de las δ muestras. En este caso las muestras corresponden a las δ-1 muestras anteriores mas la propia muestra j:

$$\Phi_i^j = \left(\sum_{h=j-\delta+1}^{j} \frac{(\alpha_i^h)(\beta_i^h)}{e^{(\varepsilon_i^h + \rho_i^h \kappa_i^h + \tau_i^h \nu_i^h)}} \right) 1/\delta$$

$$\forall j \geq \delta$$

(4-12)

en donde:

i = {1,2,...,n}; n = número de nodos; j = {1,2...,m}; m = número de muestras; δ = tamaño de la ventana o período de operación (número de muestras para realizar el cálculo); h = número de muestra en la ventana y las demás variables como en la Tabla 4-1.

La disponibilidad del nodo Φ en conjunto con el algoritmo HILO para realizar el balanceo de cargas y distribución de procesos presenta un comportamiento diferente dependiendo de las distintas condiciones con las que se evalúe, a continuación se presentan una serie de casos en los que se describen los distintos valores utilizados para cada simulación y los resultados obtenidos.

El grado de balance del DS está dado por el factor de balanceo (μ), el cual se calcula mediante la ecuación (4-3)cada δ muestras, de forma que cuando el nodo más disponible (Φ_{max}) y el nodo menos disponible (Φ_{min}) tienen el mismo valor, el sistema está completamente balanceado (μ=1).

Cabe recalcar que el tamaño de la ventana (δ) tiene un impacto en el cálculo de la disponibilidad de los nodos y del factor de balanceo del sistema, ya que no solamente indica el número de muestras a considerarse para el cálculo, sino también la periodicidad del mismo, es decir que cada δ muestras se calculan los valores de Φ para todos los nodos, así como el factor de balanceo μ. Por ello el valor de δ representa tanto el tamaño de la ventana como el período de cálculo de la métrica.

En las siguientes simulaciones, se realiza un balanceo de cargas cada vez que el factor de balanceo (μ) del sistema excede el factor de balanceo esperado. Este balanceo (utilizando el algoritmo HILO) se realiza básicamente de la siguiente manera: Se elimina el último proceso de la cola de ejecución del nodo menos disponible (con el valor de Φ_{min}) y se añade a la cola del nodo más disponible (correspondiente a Φ_{max}).

En cada caso se presentan los valores utilizados para la simulación y las gráficas mostrando el comportamiento del sistema y los valores finales del factor de balanceo (μ) y el número de balanceos efectuados. Los valores utilizados en cada caso son: “Número de procesos listos para su ejecución” se estima el número de procesos que están listos para ser ejecutados por el sistema

en cada muestra (independientemente del valor de δ), el número aleatorio de procesos listos para su ejecución tiene una distribución de Poisson con la media λ indicada en cada caso. El "Tiempo de ejecución por proceso" es representado como el número de muestras que tarda la ejecución de cada proceso, este tiempo aleatorio presenta una distribución Exponencial y se indica la media m en cada caso. El "Total de muestras" es el tiempo que dura la simulación. La "Generación de procesos" indica cuando (en que muestra) están listos los procesos para ser ejecutados por el sistema. Los dos valores restantes son el "Tamaño de la ventana" (δ), que representa tanto el tamaño de la ventana como el período de cálculo de la métrica y el "Factor de balanceo esperado" (μ).

Valores Caso1:

Número de procesos listos para su ejecución.	Distribución de Poisson con λ=1
Tiempo de ejecución por proceso.	Distribución Exponencial con m=15
Total de muestras	50
Generación de procesos	1 cada muestra
Tamaño de la ventana	δ={1,2,4,20}
Factor de balanceo esperado	μ > 0.66

En las gráficas del comportamiento de Φ como la Figura 4-4, se observa en la parte superior el valor de δ (tamaño de la ventana) utilizado, el valor final de μ (factor de balanceo alcanzado) y "b" que representa el número de balanceos efectuados. El eje de las abscisa tiene el valor de Φ (de 0 a 1) correspondiente a cada nodo (eje horizontal izquierdo) durante cada muestra (eje horizontal derecho).

El tamaño de la ventana (δ) influye en el comportamiento general del DS ya que dependiendo de este, se obtienen valores diferentes de μ. En las gráficas de la Figura 4-4 se muestra el comportamiento de Φ durante 50 muestras, cabe resaltar que entre más grande es delta, más suave se ve la curva del comportamiento de Φ en los distintos nodos. Obsérvese que las primeras delta muestras de cada ejemplo no proveen información.

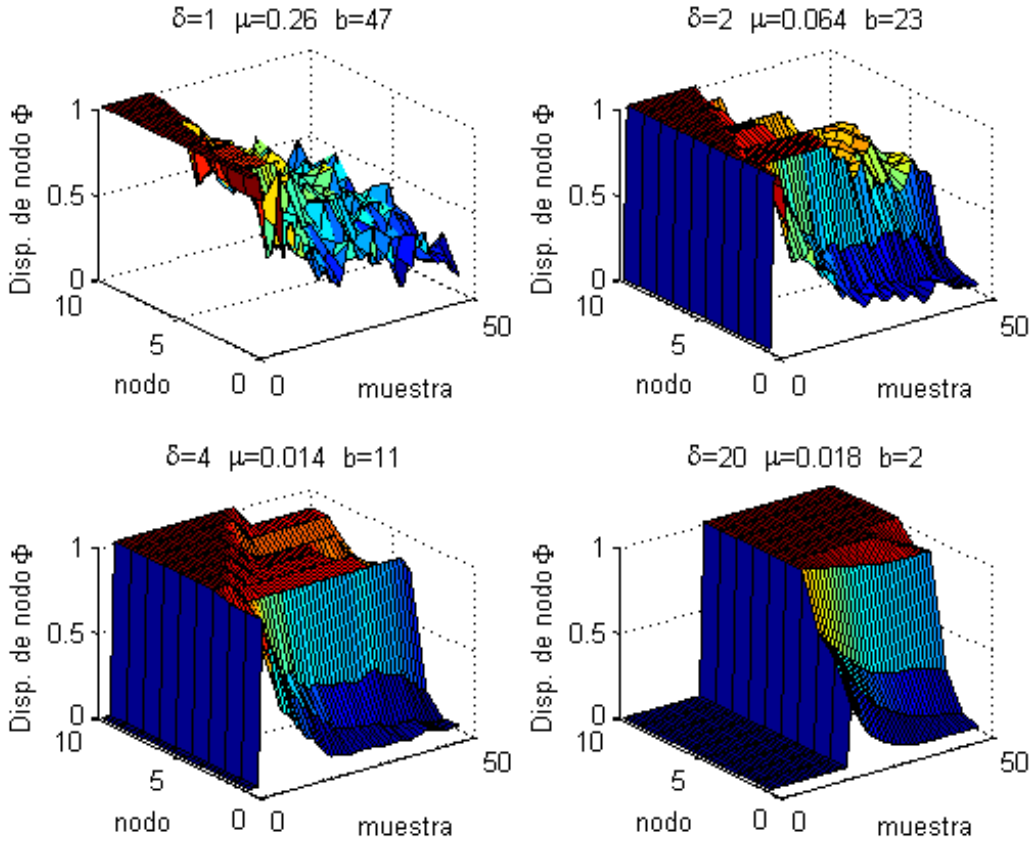


Figura 4-4. Comportamiento de Φ Caso 1

En este primer caso (Figura 4-4) es notorio ver que el sistema presenta un mejor nivel de balanceo mientras más pequeño es el valor de δ y que el número de balanceos "b" es inversamente proporcional a delta. En la primera gráfica cuando δ=1 y b=47 significa que HILO realiza una operación de balanceo en prácticamente cada una de las 50 muestras y si bien se alcanza un factor de balanceo más alto que en las otras tres gráficas (μ=0.26) también se aprecia que la disponibilidad de todos los nodos está por debajo del 0.5. En la segunda gráfica con δ=2, se aprecia en primera instancia que el algoritmo HILO no realiza nada sino hasta la segunda muestra (análogamente cuando δ=4 ó δ=20, HILO empieza a trabajar después de las δ muestras). El factor de balanceo presenta un valor muy bajo comparado con la primera gráfica, ya que desciende hasta μ=0.064 y en este caso se realiza un balanceo cada período (δ) de cálculo de la métrica dando b=23. De igual manera, en las dos gráficas inferiores, cuando δ=4 y δ=20 se realizan balanceo en cada período de cálculo obteniéndose respectivamente b=11 y b=2; y al

igual que con delta=2 el factor de balanceo obtenido presenta un valor muy por debajo del esperado (μ=0.014 y 0.018 respectivamente). Como primera aproximación se podría asumir que es mejor tener una delta pequeña; pero esto no es necesariamente correcto ya que se realiza un alto número de balanceos ocasionando con ello una sobrecarga del sistema, lo cual se aprecia gráficamente ya que conforme δ disminuye la disponibilidad general del sistema también disminuye y de manera más rápida.

Valores Caso 2:

Número de procesos listos para su ejecución.	Distribución de Poisson con λ=1
Tiempo de ejecución por proceso.	Distribución Exponencial con m=1
Total de muestras	50
Generación de procesos	1 cada muestra
Tamaño de la ventana	δ={1,2,4,20}
Factor de balanceo esperado	$\mu > 0.66$

La diferencia entre este "Caso 2" con el "Caso 1" anterior, radica en que los procesos tienen un tiempo de ejecución con promedio de 1 en lugar de 15 lo que permite tener un sistema con menor carga de trabajo reflejándose en una mayor disponibilidad en cada nodo. Al ser los procesos de una menor duración en promedio, el factor de balanceo alcanzado es mayor como se aprecia en los resultados de la Figura 4-5 y pese a que las gráficas de ambos casos ("Caso 1" Figura 4-4 y "Caso 2" Figura 4-5) parecen similares puede apreciarse que los factores de balanceo crecen significativamente. En el "Caso 2", cuando δ=1 el número de balanceos es igual que la misma delta del "Caso 1" (b=47), pero el factor de balanceo se duplica (μ=0.42). Este incremento en el nivel de balanceo del sistema es más drástico en la segunda gráfica, es decir cuando δ=2, ya que si bien el número de balanceos es igual al del caso anterior (b=23), el factor aumenta de μ=0.064 para el "Caso 1" a μ=0.14. En las gráficas inferiores el aumento en el factor de balanceo es casi despreciable ya que con δ=4 pasa de μ =0.014 del "Caso 1" a μ =0.019 "Caso 2", sin alterar el números de balanceos b=11. Con δ=20 se mejora un poco más que con δ=4, ya que el sistema pasa de μ =0.018 "Caso 1" a μ =0.044 "Caso 2", nuevamente sin alterar el números de balanceos b=2.

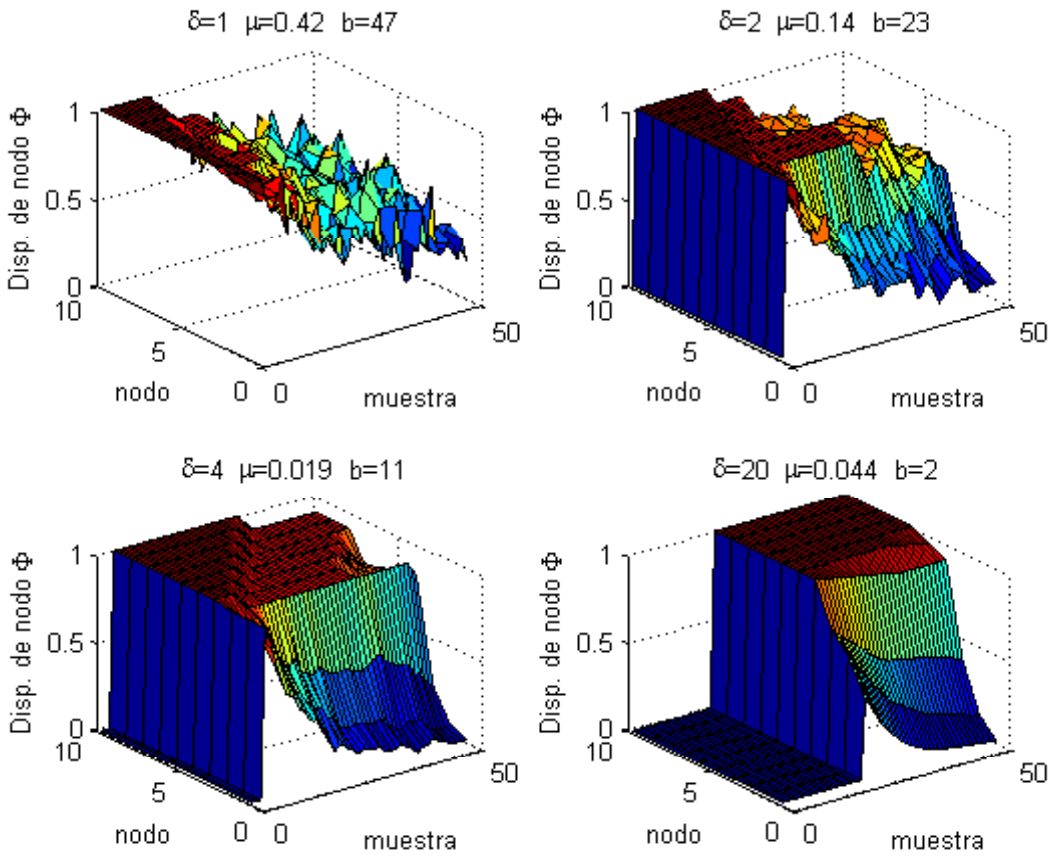


Figura 4-5. Comportamiento de Φ Caso 2

A partir del "Caso 3" y hasta el "Caso 5" el tiempo que duran las simulaciones se incrementa a 500 muestras.

Valores Caso 3:

Número de procesos listos para su ejecución.	Distribución de Poisson con $\lambda=1$
Tiempo de ejecución por proceso.	Distribución Exponencial con m=15
Total de muestras	500
Generación de procesos	1 cada muestra
Tamaño de la ventana	$\delta=\{1,2,4,20\}$
Factor de balanceo esperado	$\mu > 0.66$

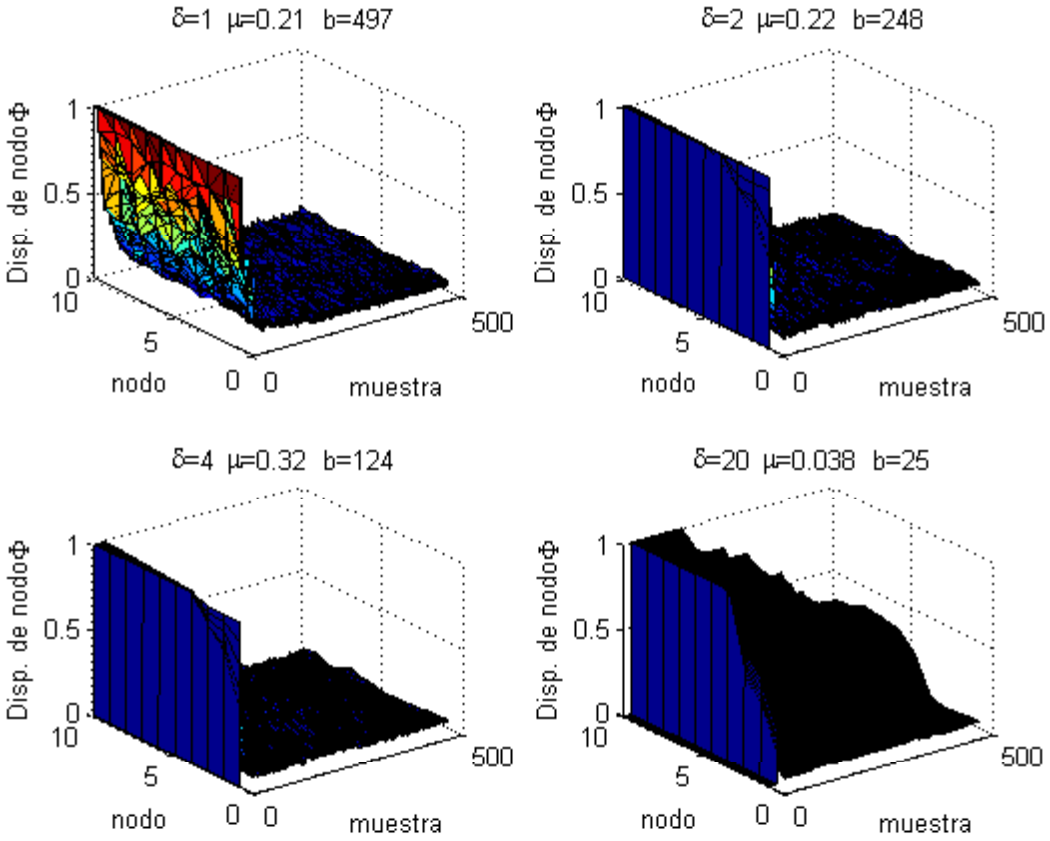


Figura 4-6. Comportamiento de Φ Caso 3

En el "Caso 3" el tiempo que dura la simulación se incrementa hasta 500 muestras y el tiempo promedio de ejecución de cada proceso es de 15 muestras como en el "Caso 1", resalta que el período de operación (δ) relativamente pequeño (1 ó 2) no influye grandemente en el comportamiento general del DS ya que se obtienen valores similares de μ (ver gráficas superiores de la Figura 4-6). El sistema presenta una disponibilidad baja ya que el promedio de duración de los procesos está nuevamente en 15 como en el "Caso 1" y pese al alto número de balanceos efectuados (prácticamente un balanceo en cada ocasión que se calcula Φ) el valor de μ es relativamente bajo. Obsérvese que el período de operación δ=4, permite alcanzar el mayor factor de balanceo μ =0.32 por arriba de los otros en donde μ =0.21 para δ=1, μ =0.22 para δ=2, y el más bajo de todos, μ =0.038 para δ=20. Para los cuatro valores de delta el un número de balanceos es equivalente a 500 entre la delta respectiva, es decir que para δ=1 b=497, para δ=2 b=248, para δ=4 b=124 y para δ=20 b=25.

Para el "Caso 4" los procesos están listos para ser ejecutados desde la primera muestra hasta la muestra número 50, esto permite tener un sistema con menos carga que el "Caso 3" (en el cual los procesos están listos para su ejecución desde la muestra 1 hasta la 500) lo cual se refleja en las gráficas correspondientes (Figura 4-7).

Valores Caso 4:

Número de procesos listos para su ejecución.	Distribución de Poisson con $\lambda=1$
Tiempo de ejecución por proceso.	Distribución Exponencial con m=15
Total de muestras	500
Generación de procesos	1 cada muestra hasta la muestra 50
Tamaño de la ventana	$\delta=\{1,2,4,20\}$
Factor de balanceo esperado	$\mu > 0.66$

El nivel de balanceo del "Caso 4" solo cumple con el valor esperado cuando $\delta=4$, con un valor final de $\mu=0.74$ y realizando solamente 47 balanceos durante todo el proceso (gráfica inferior izquierda de la Figura 4-7). Estos datos contrastan con los de la primera gráfica, en donde el valor de delta es $\delta=1$, lo que ocasiona que el algoritmo realice un balanceo prácticamente en cada muestra, sin que esto mejore el nivel de balanceo del sistema durante las 500 muestras ya que termina con el menor factor de balanceo de las cuatro gráficas con $\mu=0.4$, mientras que con $\delta=2$ aumenta el balanceo a $\mu=0.56$. En la última gráfica, con $\delta=20$ el factor de balanceo aumenta considerablemente al compararlo con el obtenido en el "Caso 3" de $\mu=0.038$ con en del "Caso 4" $\mu=0.51$, si bien en ambos casos el número de balanceos es igual b=25.

Claramente se puede ver que al no tener generación o arribo de procesos, el sistema tiende a estar mejor balanceado con una delta con valor de cuatro, mientras que con delta en la unidad el sistema presenta el menor factor de balanceo de las cuatro gráficas.

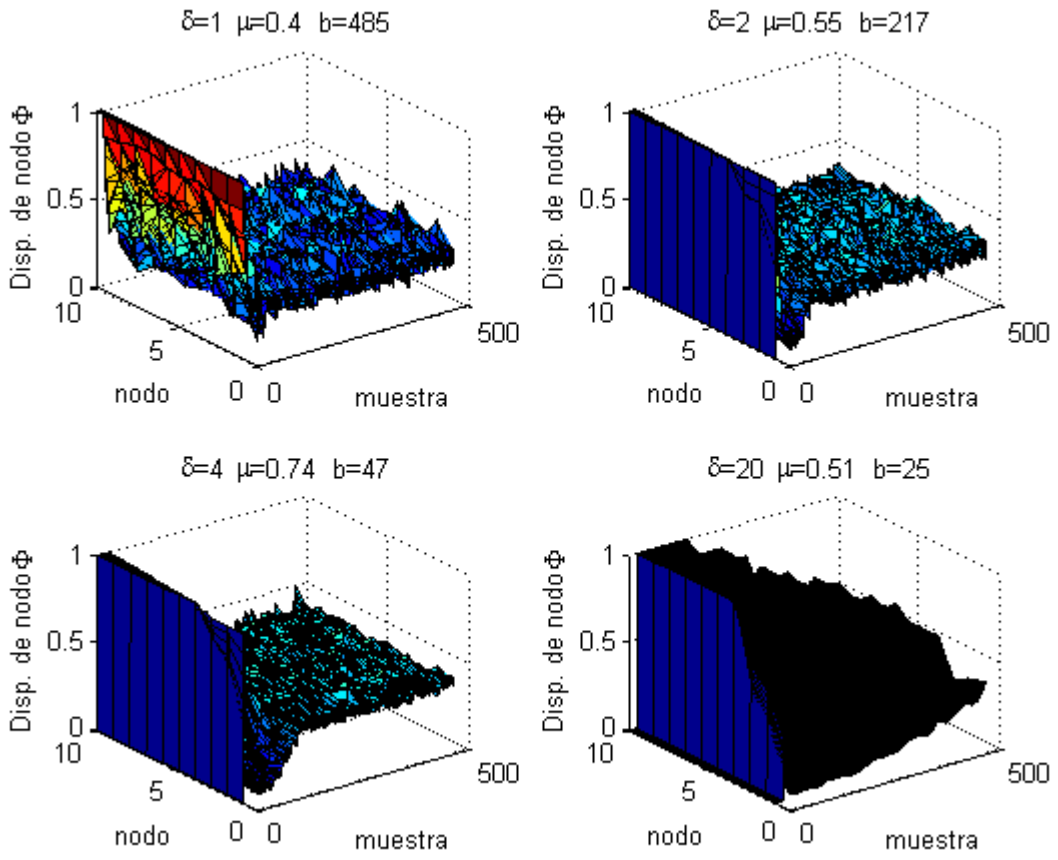


Figura 4-7. Comportamiento de Φ Caso 4

Para el siguiente "Caso 5" se espera tener un factor de balanceo igual o mayor a 0.9 sin cambiar ningún otro parámetro contra el "Caso 4".

Valores Caso 5:

Número de procesos listos para su ejecución	Distribución de Poisson con λ=1
Tiempo de ejecución por proceso.	Distribución Exponencial con m=15
Total de muestras	500
Generación de procesos	1 cada muestra hasta la muestra 50
Tamaño de la ventana	δ={1,2,4,20}
Factor de balanceo esperado	μ > 0.9

Al desear tener un factor de balanceo alto ($\mu > 0.9$) se obliga al algoritmo HILO a realizar un balanceo en prácticamente cada período de operación como sucedió en el "Caso 3" y como se aprecia en las gráficas de la Figura 4-8, lo que no necesariamente redunda en un mejor factor de balanceo del sistema.

Comparando los resultados del "Caso 4" con los del "Caso 5" se observa que para $\delta=1$ el número de balanceos aumenta en 12 y el factor de balanceo final lo hace en una décima de $\mu=0.4$ a $\mu=0.5$. Cuando $\delta=2$ el número de balanceos aumenta en casi 30, mientras que el factor de balanceo no sufre cambios y permanece en $\mu=0.56$. Cuando $\delta=20$ no existe ningún cambio, sin embargo cuando $\delta=4$ el desempeño general del sistema tiene una caída notable, ya que el número de balanceos aumenta de b=47 para el "Caso 4" a b=124 para el "Caso 5" y análogamente el factor de balanceo desciende de $\mu=0.74$ a $\mu=0.6$.

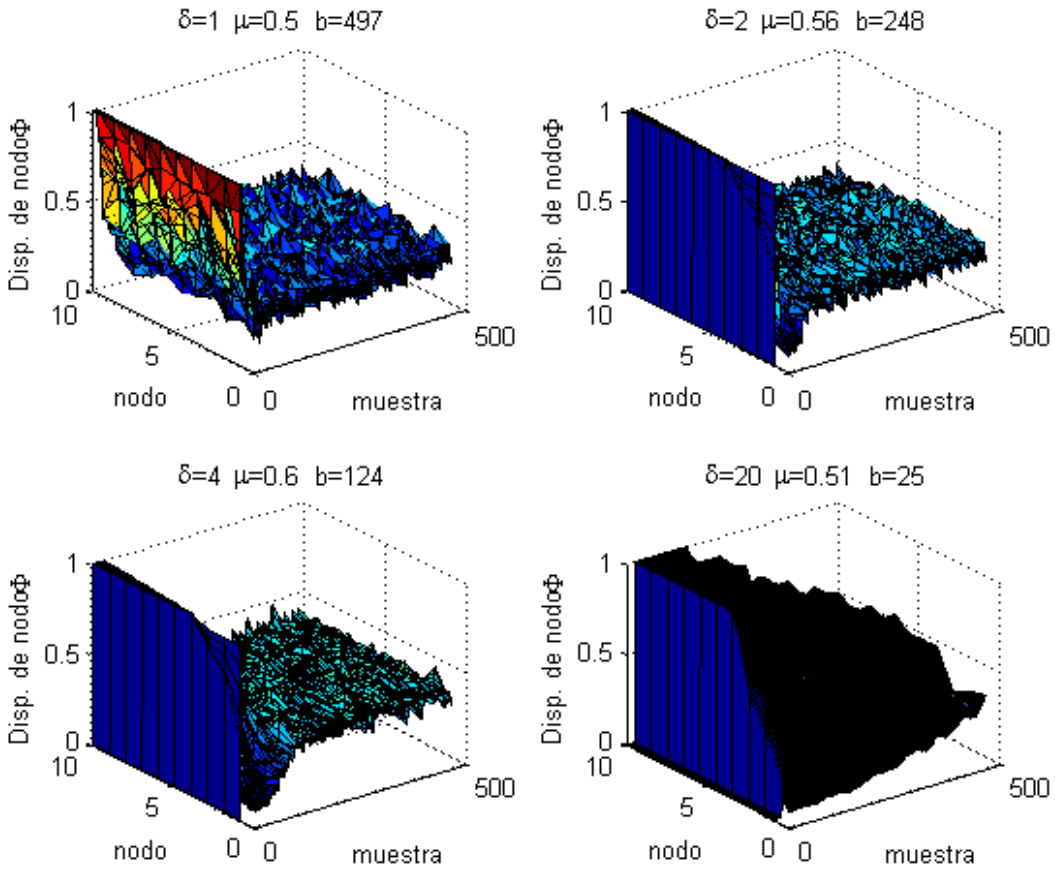


Figura 4-8. Comportamiento de Φ Caso 5

Conclusiones

La métrica de disponibilidad del nodo (Φ) como un criterio para realizar una distribución de tareas entre los nodos del RTDS, permite tener un RTDS con grado de utilización (U) similar en cada uno de los nodos que lo integran. La métrica provee información complementaria como el tiempo para realizar las comunicaciones externas, lo cual se puede interpretar como la especificación de la velocidad de la red.

La métrica de disponibilidad del nodo permite tener una visión más amplia de la carga impuesta a los nodos por las tareas que deben ejecutar, al considerar no solamente la utilización teórica del procesador sino además: los tiempos de las comunicaciones, las tareas inherentes a cualquier nodo participando en un sistema distribuido, el uso real del procesador y la disponibilidad de memoria.

Dependiendo del tiempo promedio de ejecución de los procesos y el nivel de balanceo esperado del sistema se debe encontrar el tamaño de ventana más adecuado para el cálculo de Φ como se demostró en los cinco casos presentados en el análisis del comportamiento de Φ.

5 Casos de estudio

Un sistema distribuido en Tiempo-Real consta de una red de comunicaciones en Tiempo-Real como puede ser FTT-CAN [ALMEIDA, et al. 04], a través de la cual se comunican los nodos, los cuales a su vez tienen un sistema operativo en Tiempo-Real. Como se ha estipulado en capítulos anteriores, se plantea utilizar un planificador global (rt-Scheduler) el cual puede ser implementado como una jerarquía de planificadores que permita agrupar los nodos para la planificación de la ejecución de las tareas del RTDS.

En la Figura 5-1 se presenta un modelo jerárquico de este RTDS, en donde se distinguen tres tipos de componentes: en la parte superior derecha está el Nodo maestro, el segundo es la red en Tiempo-Real, y el último tipo es el de los nodos restantes del RTDS.

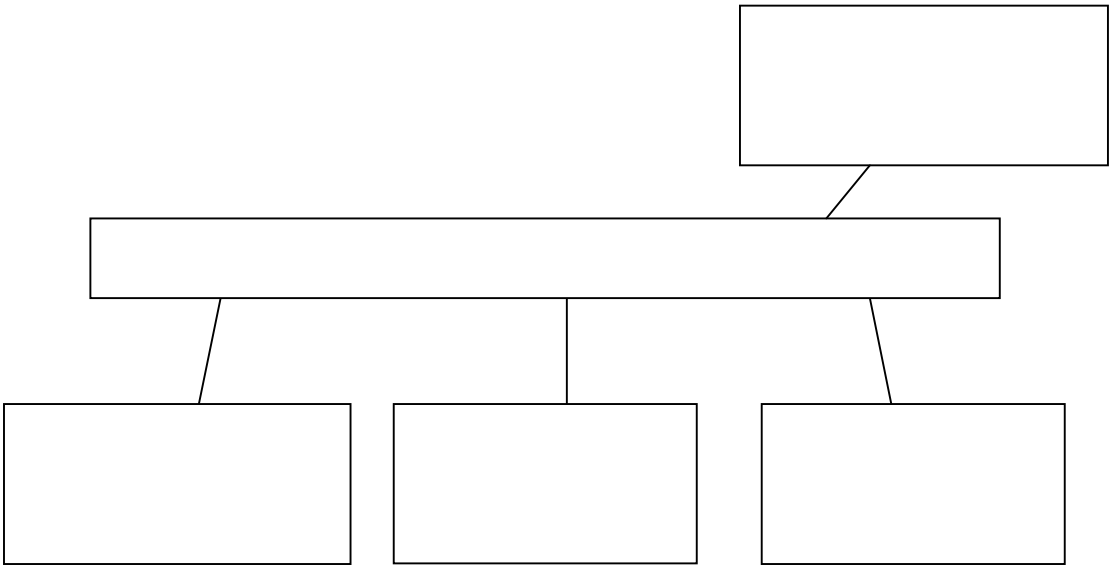


Figura 5-1. Modelo jerárquico de un Sistema Distribuido en Tiempo-Real

El nodo “Maestro” es el encargado de la definir el tiempo y período del proceso (rt-sched-proc) de comunicación de los miembros del RTDS (ver Figura 3-4). Además este nodo es el responsable de realizar el balanceo de cargas, al igual que la planificación del tráfico de la red.

Teniendo en cuenta este modelo Figura 5-1 y el comportamiento de los sistemas de control por red Figura 2-1 se plantea analizar las restricciones de Tiempo-Real a través de la red de comunicaciones ya que generalmente es el recurso restringente. Para dicho análisis se presentan dos casos de estudio, el primero es un "Cluster" y el segundo el controlar un helicóptero a través del RTDS.

Cada vez que un nodo es dado de alta o baja en la red, el nodo maestro lo incluye en la planificación del tráfico de Tiempo-Real y balancea la carga considerando el cambio en el número de nodos. El proceso de planificación de cada tarea en el RTDS se realiza en primera instancia en el planificador de envío de mensajes en la red. La planificación de la red se realiza utilizando un algoritmo de planificación que permita el tráfico de mensajes síncronos y asíncronos. Una vez aceptado el tamaño y período de la transmisión que realiza una tarea, está se asigna al nodo más conveniente para ejecutar dicha tarea.
La elección del nodo más conveniente, es tomada con base en la métrica de disponibilidad del nodo "Phi". Y por supuesto que el nodo en cuestión no tenga una utilización "U" mayor que la unidad. En cada nodo se existe una jerarquía de planificación y las tareas son planificadas de acuerdo a está.

El planificador global o "rt-scheduler" obliga a plantear la composición o coordinación, de algoritmos de planificación. Área abierta de investigación en los sistemas de Tiempo-Real y en particular en al área de sistemas distribuidos.

Por un lado están los algoritmos enfocados a la capa de la red, los cuales no toman en cuenta las características de los procesadores o nodos que conforman dicha red.

5.1 Sistema distribuido de alto rendimiento (Cluster)

En este caso de estudio se presenta la adaptación de un algoritmo periódico para lograr un balanceo de cargas en un sistema distribuido de alto desempeño del tipo "Cluster", partiendo del hecho de que en la mayoría de los casos no se conocen a priori los tiempos reales de ejecución de los procesos que son ejecutados. Para poder realizar el balanceo de cargas sin conocer el tiempo

que tarda la ejecución de cada proceso se utiliza la métrica de “Disponibilidad del Nodo”. El caso de estudio muestra que el algoritmo presentado y la métrica de disponibilidad del nodo, permiten una fácil implementación y un rendimiento aceptable al compararlo con otros algoritmos.

Al realizar el balanceo de cargas (LB) en un sistema distribuido (DS) se espera que los recursos (en particular los procesadores) se utilicen de manera óptima y se obtenga un beneficio substancial en el desempeño general de sistema [FERNANDES et al., 06]. Considerando que el desempeño general del DS puede ser medido como el tiempo total en el que se ejecutan un conjunto de procesos [LEE et al., 04], el balanceo de cargas es una estrategia útil para mejorar el desempeño de un DS ya que reduce el tiempo que se requiere para concluir la ejecución de un conjunto de procesos.

El tiempo total de ejecución de un conjunto de procesos en un DS esta en función de la cantidad y complejidad de los procesos que el DS debe ejecutar, así como del uso de los recursos de procesamiento disponibles. En este caso de estudio se presenta el uso de un algoritmo periódico para realizar el balanceo de cargas (HILO) utilizando la métrica de disponibilidad del nodo para ello y así lograr un desempeño óptimo del DS.

El caso de estudio se implementó en un “cluster” que consta de 16 nodos con la siguiente configuración:

- Un nodo maestro con 2 procesadores Xeon de 2.6 GHz, 1.5 GB de memoria y Linux Kernel 2.6.8.
- 15 nodos con procesador Pentium IV de 2.6 GHz, 512 MB de memoria y Linux Kernel 2.6.12.

El nodo utilizado como maestro realiza las funciones de distribución y planificación, distribuyendo la carga a los 15 nodos restantes.

El cluster de manera dedicada ejecuta conjuntos de procesos independientes entre sí, los cuales tienen 100, 200, 300, 400, 500, 1000 y 1500 procesos respectivamente.

Se utiliza una distribución de Poisson para simular el tiempo de llegada de los procesos al cluster y una distribución exponencial para el tiempo de ejecución de los mismos [CHIASSON et al., 05; ZENG et al., 04].

Cada proceso realiza sumas consecutivas y copia cadenas de caracteres, siendo distinto el número de operaciones de suma o copia en cada caso y por lo tanto, el uso de procesador o memoria.

Se seleccionaron tres algoritmos para realizar el balanceo de cargas, estos son: Random [SIT et al., 04; TAKEMOTO et al., 04] en el cual cada vez que llega un proceso al cluster se selecciona de manera aleatoria (con distribución uniforme) el nodo que lo va a ejecutar y una vez seleccionado el nodo, este ejecuta el proceso hasta su terminación sin la posibilidad de migrarlo hacia otro nodo. Round-Robin [TAKEMOTO et al., 04; TORQUE, 06] el algoritmo tiene una lista circular de los nodos y un apuntador a uno de ellos (el siguiente en la lista para recibir un proceso), cuando llega un proceso lo asigna al nodo que indica el apuntador y actualiza dicho apuntador hacia el siguiente nodo de la lista. Una vez asignado un proceso a un nodo, el comportamiento del algoritmo Round-Robin es similar al algoritmo Random en lo referente a que una vez seleccionado el nodo este debe ejecutar el proceso sin la posibilidad de migrarlo. El tercer algoritmo es HILO (descrito en el capítulo anterior), el cual por su naturaleza evalúa al sistema cada determinado número de muestras y si es el caso realiza la migración de procesos a diferencia de los otros dos algoritmos.

Cada conjunto de procesos se ejecuta utilizando los tres algoritmos (Random, Round-Robin e HILO) previamente descritos.

El algoritmo HILO se ejecuta usando los siguientes valores, seleccionados de manera heurística, $\delta = 4$, $j = 2$, $n = 15$ y μ esperado de 0.6.

Resultados:

Se analizaron los tiempos totales de ejecución de cada conjunto de procesos usando los algoritmos antes mencionados. En la Figura 5-2 se aprecia que el algoritmo propuesto supera a los otros dos. El tiempo de ejecución del algoritmo HILO se considera como el 100%, los otros dos algoritmos (Round-Robin y Random) tardan del 110% al 165% de tiempo al ejecutar el mismo conjunto de procesos.

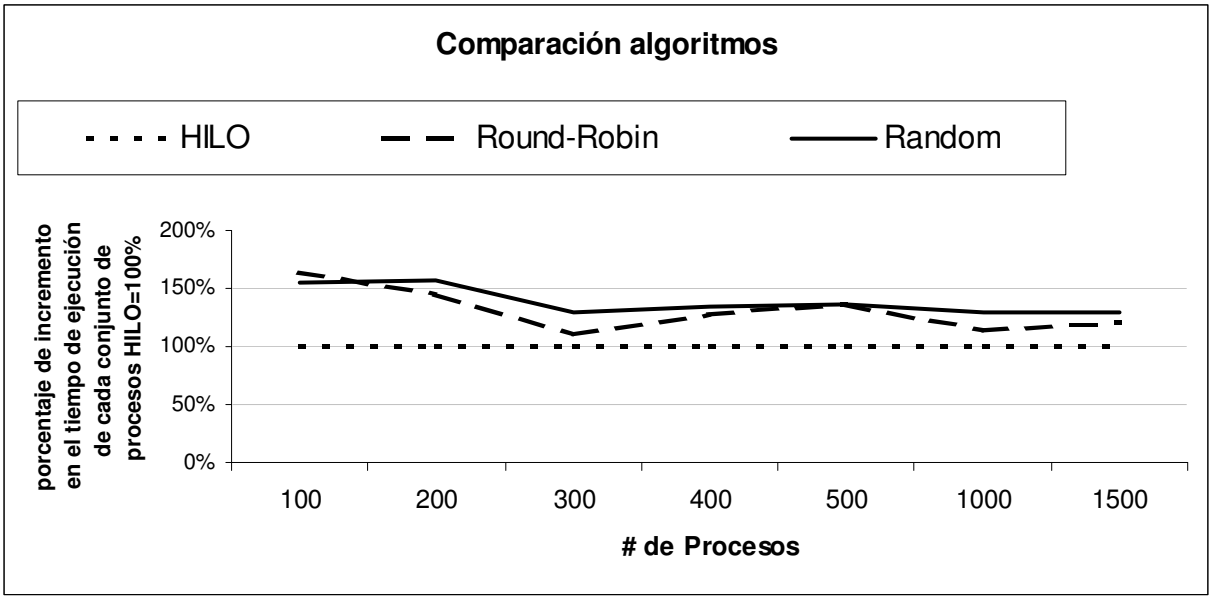


Figura 5-2. Eficiencia comparada del algoritmo HILO.

En cada ocasión que el factor μ deseado fue menor que el calculado se realizaba un balanceo. El número de balanceos que HILO realizó para cada conjunto de procesos se muestra en la Tabla 2, el tiempo total esta expresado en minutos:segundos.

Tabla 5-1 Número de balanceos por conjunto de procesos

# Procesos	**# Balanceos**	**Tiempo total**
100	31	01:32.0
200	86	03:17.8
300	137	04:50.0

400	186	06:27.3
500	226	07:52.0
1,000	456	15:36.0
1,500	676	22:59.9

.

En las gráficas de las Figura 5-3 y Figura 5-5 se aprecia el comportamiento del sistema con los distintos conjuntos de procesos. En términos generales se aprecia que la disponibilidad de los nodos desciende y asciende de manera más o menos uniforme al inicio y fin de cada ejemplo, indicando con ello que el sistema tuvo un uso balanceado de los recursos.

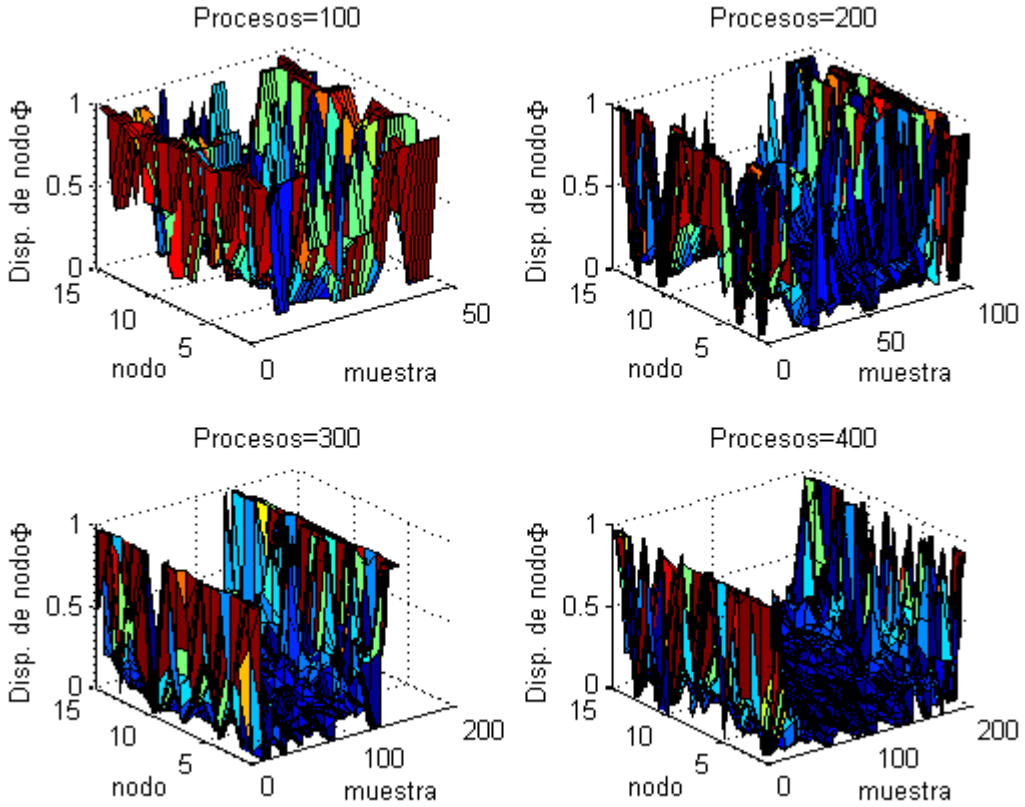


Figura 5-3. Comportamiento de Phi con HILO.

En la gráfica superior izquierda de la Figura 5-3 se puede apreciar que el conjunto de los cien procesos que se ejecutan toma 50 muestra, y se realizan 31 balanceos (Tabla 5-1) lo que aparentemente indica un alto número de balanceos, pero al comparar los resultados contra los

que se obtienen al no realizar ningún balaceo (usando los algoritmos de Round-Robin o Random) se obtiene una disminución en el tiempo de ejecución total de al menos un 50% (ver Figura 5-2). En el caso de 200 procesos (gráfica superior derecha Figura 5-3) el sistema tiene muchas variaciones, al igual que en el caso anterior, aunque se empieza a presentar una tendencia a balance en el centro de la misma y se realizan más del doble de balanceos (81). En la gráfica correspondiente a los 300 procesos se observa claramente el valle formado en la parte intermedia de la gráfica, mostrando un sistema más balanceado que el presentado en las dos gráficas anteriores, se resalta también el hecho de que la finalizar la ejecución el sistema presenta disponibilidad absoluta en todos los nodos. En la última gráfica de la Figura 5-3 se aprecia que el sistema no termina totalmente balanceado como en el caso anterior de los 300 procesos.

En la Figura 5-4 se muestra que el que algoritmo HILO es más eficiente al momento de ejecutar los distintos conjuntos de procesos, ya que en todos los casos el tiempo total de ejecución de estos es menor al utilizar HILO que al usar cualquiera de los otros dos. Cabe resaltar que el caso de 300 procesos, la eficiencia de HILO está más cercana a la de los otros dos algoritmos y es el caso en el que el sistema termina mejor balanceado, pero con la mayoría de los nodos presentando disponibilidad de uno durante varias muestras (ver Figura 5-3 gráfica inferior derecha).

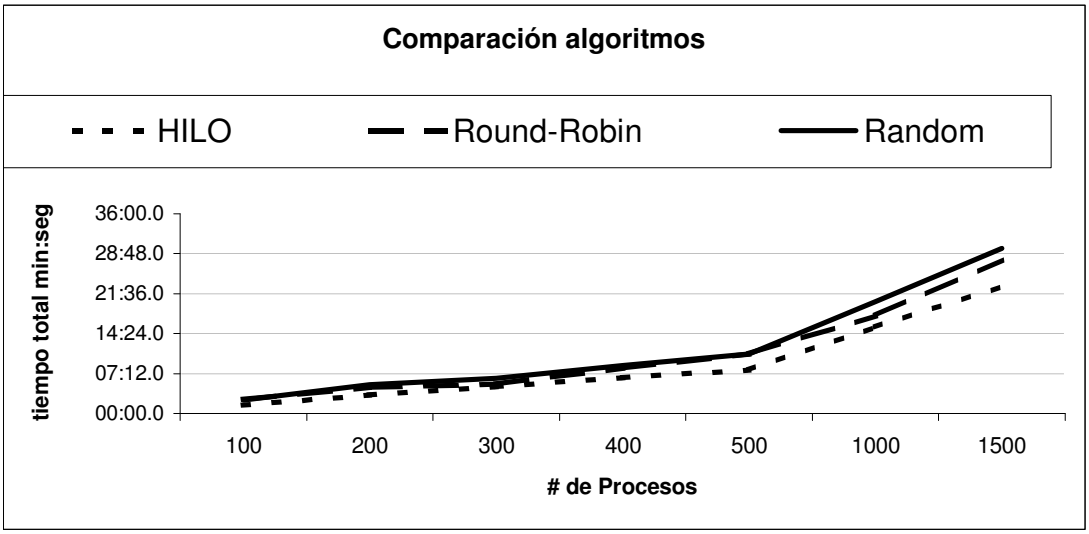


Figura 5-4. Comparación de algoritmos, con base al tiempo de ejecución de un conjunto de procesos.

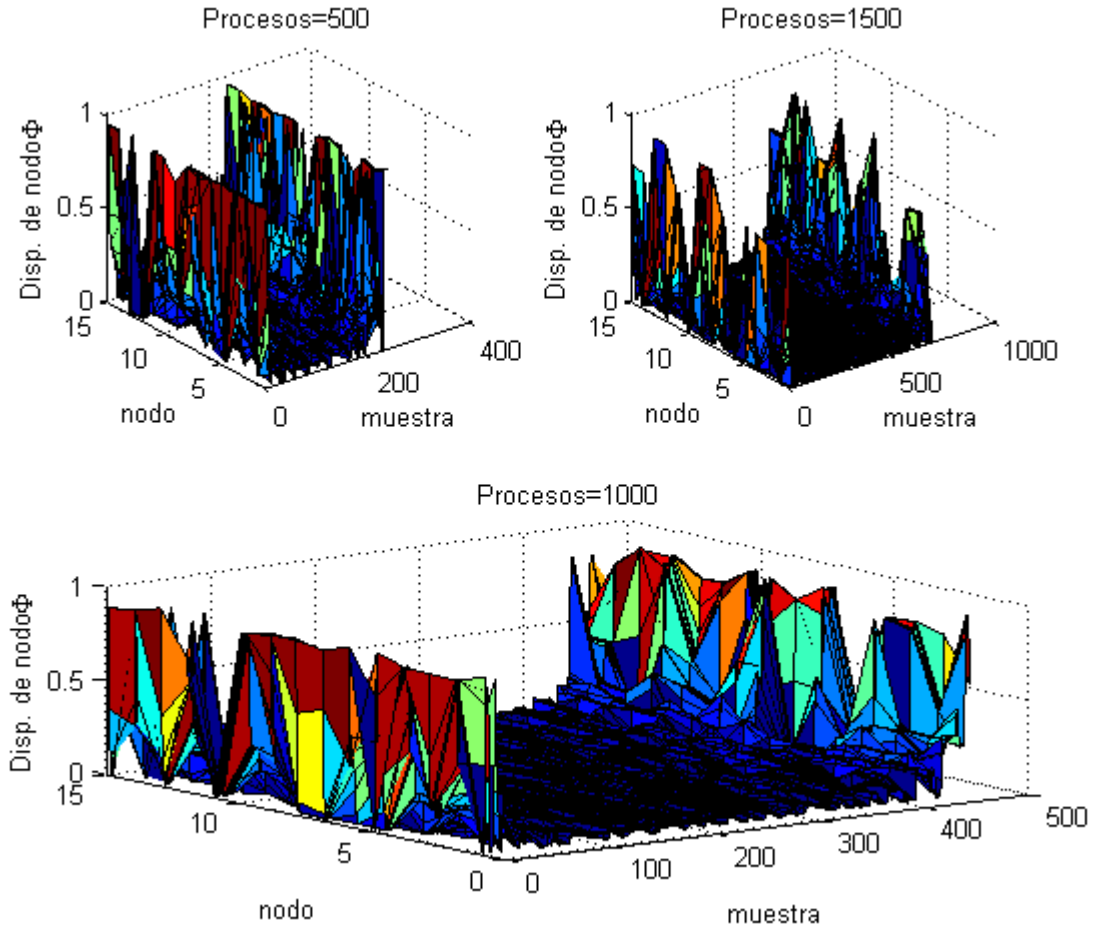


Figura 5-5. Comportamiento de Phi con HILO.

Las tres gráficas de la Figura 5-5 muestran el comportamiento del sistema durante la ejecución de los respectivos conjuntos de procesos (500, 1000 y 1500), en el caso de 1000 procesos al igual que en el caso de los 300 procesos (ver Figura 5-3) se presenta que el mayor factor de balanceo final del sistema se obtiene con el caso de menor diferencia en la eficiencia de HILO contra los otros dos algoritmos (ver Figura 5-2). En la gráfica superior derecha de la Figura 5-5 que corresponde a la ejecución de 1500 procesos, se aprecia que tanto al inicio como al final los nodos se desbalancean rápidamente debido a la alta carga del sistema, pero durante prácticamente toda la ejecución el sistema está balanceado. En la misma figura, en el caso de los 500 procesos se aprecia que el sistema inicia y termina mejor balanceado que en el caso de los 1500 procesos, pero el nivel de balanceo durante la ejecución está evidentemente más desbalanceada que en el caso de los 1000 procesos.

Se presentaron dos aportaciones para lograr disminuir el tiempo de ejecución de un conjunto de procesos en un cluster. La primera es el utilizar un algoritmo periódico (HILO) para realizar el balanceo de cargas en un sistema distribuido. La segunda es la creación de la medida de disponibilidad del nodo (Φ), la cual permite realizar un LB eficiente sin conocer a priori los tiempos de ejecución de los procesos, con la ventaja adicional de cierta tolerancia a fallas, ya que si algún nodo esta teniendo problemas de disponibilidad no se le asignarán más procesos y se eliminarán de su cola de ejecución los procesos asignados, permitiendo que algún otro nodo los ejecute. Al aumentar el valor de balanceo (μ) de manera significativa (i.e 4 o 5), el número de balanceos disminuye hasta llegar a cero. En el otro extremo al tender el valor de μ a 1, el sistema está con carga balanceada pero con un rendimiento ineficiente (i.e. el tiempo total de ejecución de cada conjunto de procesos aumenta), ya que cada lectura de las medidas de disponibilidad se realiza un balanceo.

5.2 Sistema distribuido de control en Tiempo-Real (Helicóptero)

Para la simulación del comportamiento temporal de un sistema en Tiempo-Real existe una herramienta de software acoplada a Matlab llamada “True-Time”. Al ser capaz de ejecutar las funciones de un controlador o de una red en Tiempo-Real también se utiliza como plataforma experimental para investigar el comportamiento de sistemas de control dinámicos en Tiempo-Real [CERVIN et al., 03]. Con True-Time es posible estudiar los esquemas de compensación que ajusten el algoritmo de control basándose en la medición de las variaciones reales del tiempo (p. Ej. Tratando la incertidumbre temporal como una perturbación y manejándola con planificación retroalimentada o con ganancia) [CERVIN et al., 07]..

Con la finalidad de mostrar una estrategia de planificación global para un RTDS, se presenta el caso de estudio de el sistema de control por red (NCS) de un helicóptero de dos rotores, demostrando que el desempeño de un RTDS depende no solamente de los períodos de muestreo (ver Figura 2-1) de los componentes individuales, sino también de las relaciones que existen entre estos períodos. También se demuestra que es posible tener un RTDS planificable pero inestable desde el punto de vista de control, es decir como NCS.

En general se acepta que el desempeño de un NCS depende de los períodos de muestreo de cada uno de sus componentes individuales y aunque esto en lo general es cierto, para los RTDS el área de rendimiento aceptable tiende a ser más pequeña (ver Figura 2-1 área aceptable entre los puntos B y C) y no necesariamente continua. Una de las razones para este comportamiento es que las relaciones entre los períodos de los componentes del RTDS (nodos y red) tienen un alto grado de impacto en el desempeño global del sistema.

Es posible tener un RTDS en el que las restricciones de Tiempo-Real de cada tarea se cumplen, es decir que cada tarea es planificada de forma que cumpla con sus plazos. En este tipo de casos el trabajo de planificación es enorme, el sistema es poco flexible y no existe ninguna garantía de que el sistema permanezca estable en el largo plazo.

El objetivo de este caso de estudio es el mostrar una estrategia de planificación basada en el análisis del comportamiento y las limitantes de un sistema distribuido en Tiempo-Real desempeñando las funciones de un sistema de control por red.

Para probar esta aseveración se implementó un caso de estudio que consiste de un helicóptero Quanser 2 DOF de dos rotores y el software desarrollado con Matlab, Simulink y True-Time.

El caso de estudio consta de dos componentes principales, el modelo (físico y matemático) del helicóptero y el RTDS (simulado e implementado en Matlab con True-Time).

Helicóptero

Si bien la descripción del helicóptero está más allá de los alcances de este trabajo, se presenta una breve descripción del mismo, para mayor detalle consultar [QUANSER, 06]. El helicóptero Quanser 2 DOF de dos rotores está montado en una base fija. Ambos rotores son movidos por motores de corriente directa, el rotor delantero controla la inclinación de la nariz del helicóptero sobre el eje horizontal “pitch” y el rotor trasero controla el movimiento de viraje en el eje lateral

o “yaw”. Ambos ángulos inclinación (pitch) y viraje (yaw) son medidos usando decodificadores de alta precisión.

El helicóptero tiene dos grados de libertad, y rota en un ángulo θ sobre el eje horizontal y en un ángulo ψ sobre el eje lateral. El ángulo θ se define como positivo cuando la nariz del helicóptero va hacia arriba y el ángulo ψ se define positivo si el giro es en el sentido de las manecillas del reloj. La siguiente Tabla 5-2 lista varias longitudes, masas y momentos de inercia asociados al modelo del helicóptero.

Tabla 5-2 Especificaciones del Helicóptero y parámetros del modelo

Variable	*Descripción*	*Valor*	*Unidad*
$B_{eq,p}$	Fricción equivalente en el eje de elevación.	0.800	N/V
$B_{eq,y}$	Fricción equivalente en el eje de viraje.	0.318	N/V
$J_{eq,p}$	Momento total de inercia en el pivote de elevación.	0.0384	$kg \cdot m^2$
$J_{eq,y}$	Momento total de inercia en el pivote de viraje.	0.0384	$kg \cdot m^2$
K_{pp}	Torque de empuje actuando en el eje de elevación del motor de elevación.	0.204	N·m/V
K_{py}	Torque de empuje actuando en el eje de elevación del motor de viraje.	0.0068	N·m/V
K_{yp}	Torque de empuje actuando en el eje de viraje del motor de elevación.	0.0219	N·m/V
K_{yy}	Torque de empuje actuando en el eje de viraje del motor de viraje.	0.072	N·m/V
l_{cm}	Longitud del Centro de masa desde el eje de elevación..	0.186	Cm
m_{heli}	Total de masa en movimiento del helicóptero.	1.3872	kg

Un controlador del tipo FF+LQR+I por sus siglas en inglés (Feed Forward + Linear Quadratic Regulator + Integrator) implementado en Matlab y Simulink se utiliza en el modelo del helicóptero para regular los ángulos de inclinación y viraje usando un integrador en el ciclo de retroalimentación para mejorar el rendimiento al disminuir el error en estado estacionario “sse” (por las siglas del inglés: steady-state error). Utiliza algoritmos de pro-alimentación (feed-forward) positiva y velocidad proporcional integral (PIV) para regular la inclinación y sólo PIV para el ángulo de viraje.

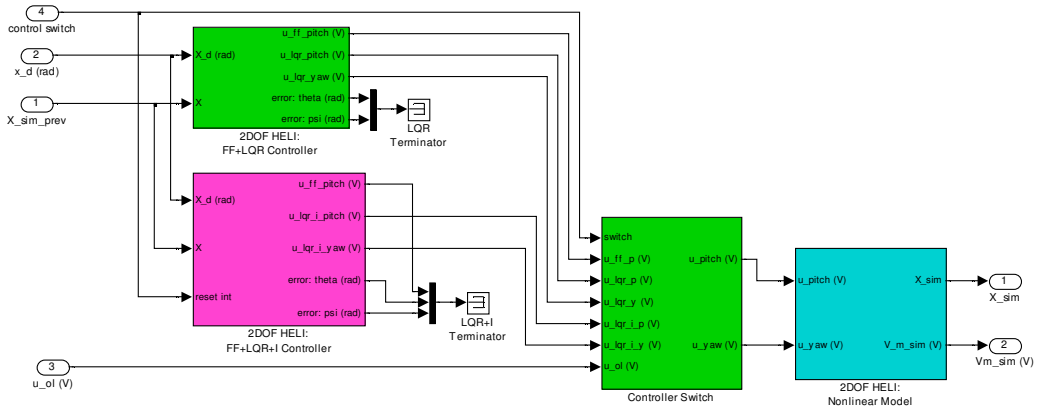


Figura 5-6. Modelo de Simulación de sistema de ciclo-cerrado.

El modelo lineal del espacio de estados esta dado por la siguiente ecuación (5-1), en donde se resuelve para $\dot{x}$ en el punto de reposo ($\theta_0 = 0, \Psi_0 = 0, \dot{\theta}_0 = 0, \dot{\Psi}_0 = 0$), substituyendo el estado $X=[\theta_0, \Psi_0, \dot{\theta}_0, \dot{\Psi}_0]$ equivalente a las coordenadas X_1, X_2, X_3 y X_4 y aumentando los estados X_5 y X_6 para minimizar el error en estado estable o "sse" (para mayor detalle de como se llega a esta ecuación consultar la documentación del fabricante [QUANSER, 06]).

$$\dot{X} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\dfrac{B_p}{J_{eq,p} + m_{heli} l_{cm}^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\dfrac{B_y}{J_{eq,y} + m_{heli} l_{cm}^2} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \dfrac{K_{pp}}{J_{eq,p} + m_{heli} l_{cm}^2} & \dfrac{K_{py}}{J_{eq,p} + m_{heli} l_{cm}^2} \\ \dfrac{K_{yp}}{J_{eq,y} + m_{heli} l_{cm}^2} & \dfrac{K_{yy}}{J_{eq,y} + m_{heli} l_{cm}^2} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x$$

(5-1)

EL controlador FF+LQR+I que hace converger $(\theta_0, \Psi_0, \dot{\theta}_0, \dot{\Psi}_0 \rightarrow \theta_0, \Psi_0, 0, 0)$ está definido en la siguiente ecuación:

$$\begin{bmatrix} u_p \\ u_y \end{bmatrix} = \begin{bmatrix} k_{ff} \dfrac{m_{heli} g l_{cm} \cos\theta_d}{k_{pp}} \\ 0 \end{bmatrix} - \begin{bmatrix} k_{11} k_{12} k_{13} k_{14} \\ k_{21} k_{22} k_{23} k_{24} \end{bmatrix} \times \begin{bmatrix} \theta - \theta_d \\ \psi - \psi_d \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} - \begin{bmatrix} \int_{t_0}^{t_f} k_{15}(\theta - \theta_d) + \int_{t_o}^{t_f} k_{16}(\psi - \psi_d) \\ \int_{t_0}^{t_f} k_{25}(\theta - \theta_d) + \int_{t_o}^{t_f} k_{26}(\psi - \psi_d) \end{bmatrix}$$

(5-2)

en donde u_p y u_y son los voltajes que corresponden a los motores de elevación y viraje respectivamente, θd es el ángulo deseado de elevación, ψd es el ángulo deseado de viraje, k_{ff} es la ganancia del controlador de proacción que tiene un valor de 1 V/V. y de k_{11} a k_{24} son los valores de ganancia (ecuación (5-3) que permiten mejorar el rendimiento (basándose en el "sse") de acuerdo al manual [QUANSER, 06] y al modelo implementado en Matlab.

$$\begin{bmatrix} k_{11} \ldots k_{16} \\ k_{21} \ldots k_{26} \end{bmatrix} = \begin{bmatrix} 18.9 & 1.98 & 7.48 & 1.53 & 7.03 & 0.770 \\ -2.22 & 19.4 & -0.450 & 11.9 & -0.770 & 7.03 \end{bmatrix}$$

(5-3)

Implementación del RTDS.

El sistema de control por red (NCS) consta de ocho nodos, cada uno con "kernel" independiente en Tiempo-Real (ver Figura 5-7) de estos 8 nodos 4 son sensores, 2 nodos son actuadores, uno es el controlador y el octavo es el nodo planificador. Los nodos se comunican entre ellos por medio de una red en Tiempo-Real de tipo "CAN". El modelo está implementado en Matlab utilizando la herramienta True-Time de la universidad de Lund Suecia [CERVIN et al., 07] para simular el "kernel" en Tiempo-Real de cada nodo y la red de comunicaciones. Las salidas del sistema

pueden ser conectadas tanto al modelo simulado en Matlab del helicóptero como al helicóptero físico, lo que permite evaluar el desempeño del RTDS en ambos escenarios.

En primer nodo en el modelo (extremo izquierdo en la Figura 5-7 es el controlador, cuya función principal es tomar los valores provistos por los 4 nodos sensores y calcular los voltajes para los motores de elevación y viraje (u_p y u_y) con la ecuación (5-2). Los nodos sensores, localizados a la derecha del nodo controlador en la corresponden respectivamente a los valores del ángulo de elevación (θ), el ángulo de viraje (ψ), la derivada de θ (θ^o), y la derivada de ψ (ψ^o), también utilizados en el cálculo de los voltajes de salida realizado con la ecuación (5-2) y los cuales se envían a los nodos actuadores (extremo inferior derecho en la Figura 5-7). El octavo nodo del modelo (extremo superior derecho de la Figura 5-7) es el nodo planificador, el cual es responsable de asignar la red de comunicaciones de forma periódica y planificar las actividades del RTDS enviando a los demás nodos un período operacional llamado “base”.

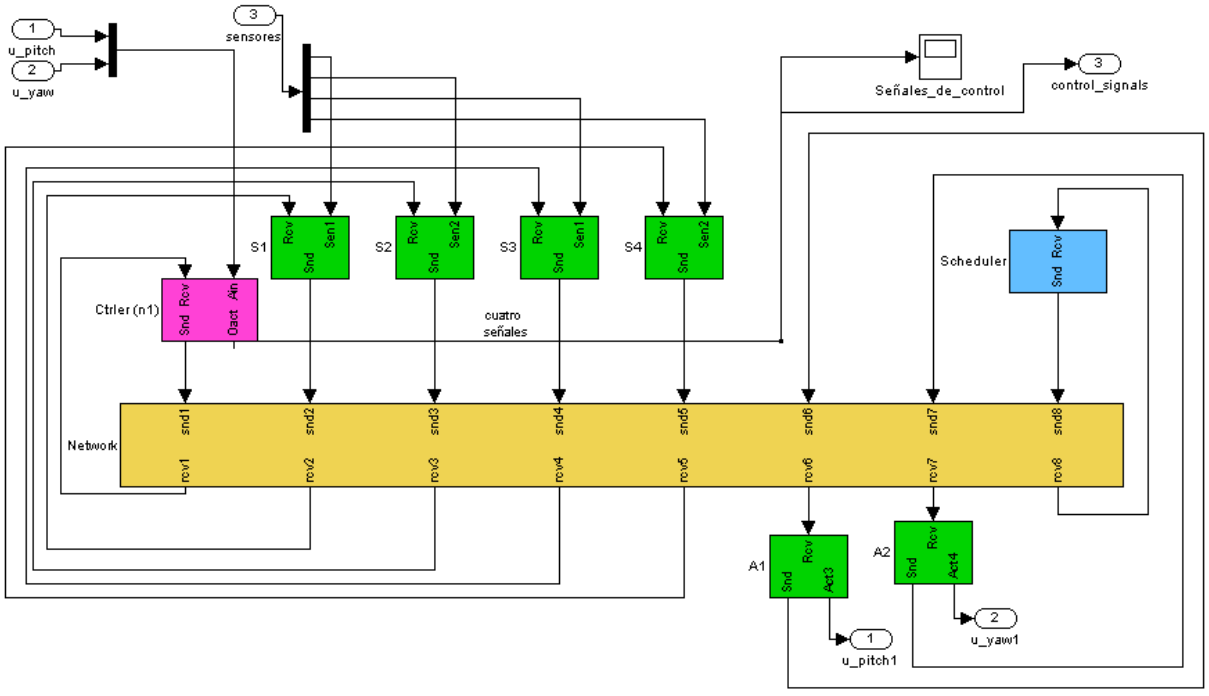


Figura 5-7. Implementación del caso de estudio del RTDS

Este período “base” transmitido por el nodo planificador, es el período de la tarea controladora en el correspondiente nodo controlador. En los 4 nodos sensores, el valor de “base” y un factor

de dispersión se utilizan para calcular el período operacional de cada nodo sensor usando las siguientes ecuaciones con base en la ecuación (3-7):

$$\begin{aligned} \text{sensor1} &= \text{base} \times (1 + \text{dispersión}) \\ \text{sensor2} &= \text{base} \times (1 - \text{dispersión}) \\ \text{sensor3} &= \text{base} \times (1 + (\text{dispersión} \times 1.1)) \\ \text{sensor4} &= \text{base} \times (1 - (\text{dispersión} \times 1.1)) \end{aligned}$$

(5-4)

en donde sensorN = período operacional del sensor N, base es el período "base" y $0 \leqslant$ dispersión $\leqslant 0.2$ es el factor de dispersión.

El valor del factor de dispersión permite en el extremo inferior, que los períodos de operación de los nodos sensores sean iguales al período del controlador (cuado dispersión = 0) y en el otro extremo cuando dispersión = 0.2 los sensores tendrán períodos de operación de hasta 22% más grandes o chicos que el período base. Una dispersión mayor al 25% ocasiona que el sistema tenga un desempeño inaceptable.

Para evaluar el desempeño de la métrica, se calcula el error de DRTS como la sumatoria de la diferencia cuadrada sobre dos entre el ángulo real (θ) obtenido y el deseado (θd) usando la ecuación (5-5), el error es sumado durante toda la simulación.

$$error = \sum \frac{(\theta_t - \theta d_t)^2}{2}$$

(5-5)

En la Figura 5-8 se aprecia la diferencia entre el ángulo real θ y el ángulo deseado θd en un caso en el que el RTDS tiene un desempeño aceptable durante los 50 segundos de la simulación. El eje de las abscisas representa el los grados del ángulo (negativo indica que la nariz del helicóptero apunta hacia abajo) y el de las ordenadas es el tiempo en segundos.

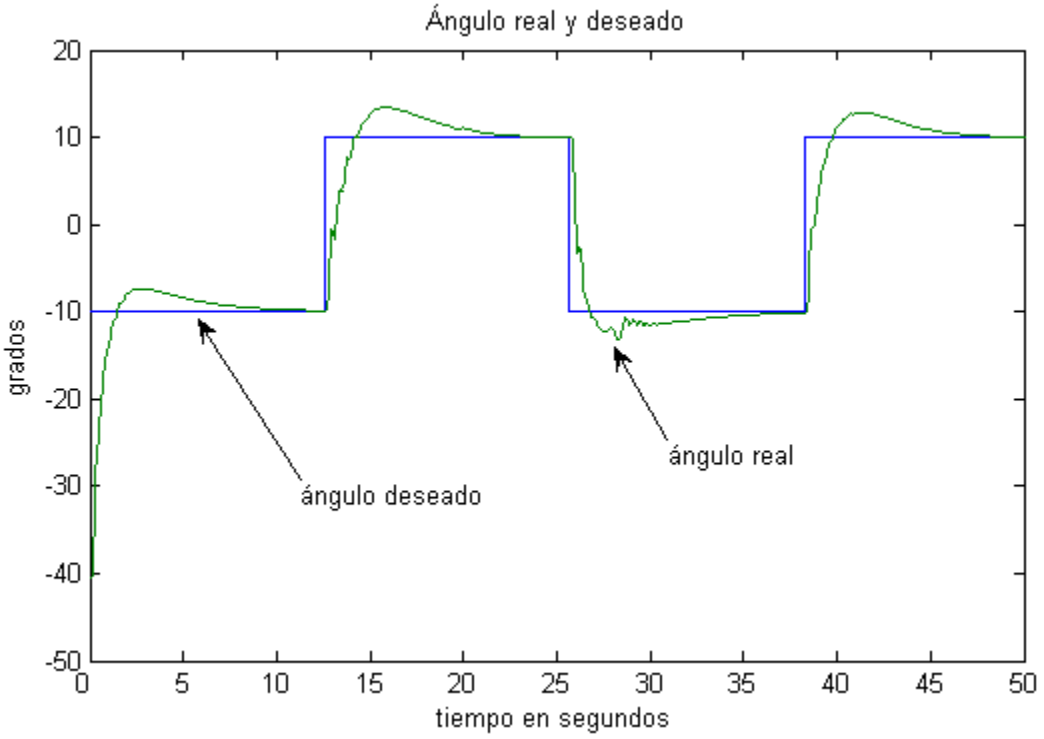


Figura 5-8. Ángulo real θ ángulo deseado θd.

La gráfica logarítmica (Figura 5-9) muestra el desempeño del sistema al graficar el error acumulado (eje de las abscisas) contra el período "base" de operación (eje de las ordenadas). La línea punteada horizontal indica si el nivel del desempeño es aceptable (abajo de la línea o inaceptable (arriba de la línea) en cada período "base", dependiendo del error acumulado durante 50 segundos. Comparando esta gráfica con la presentada en la Figura 2-1, se aprecia que los puntos B y C de esa gráfica corresponden a los puntos de bajo-muestreo y sobre-muestreo (0.027 y 0.001 segundos respectivamente) en la Figura 5-9, sin embargo en está última, el área de desempeño aceptable del sistema no es continua, ya que en al menos tres puntos el error crece por encima de la frontera de aceptabilidad del desempeño del sistema, como cuando "base" es igual a 15 milisegundos en donde claramente se ve que el error está muy por encima del correspondiente a un desempeño aceptable.

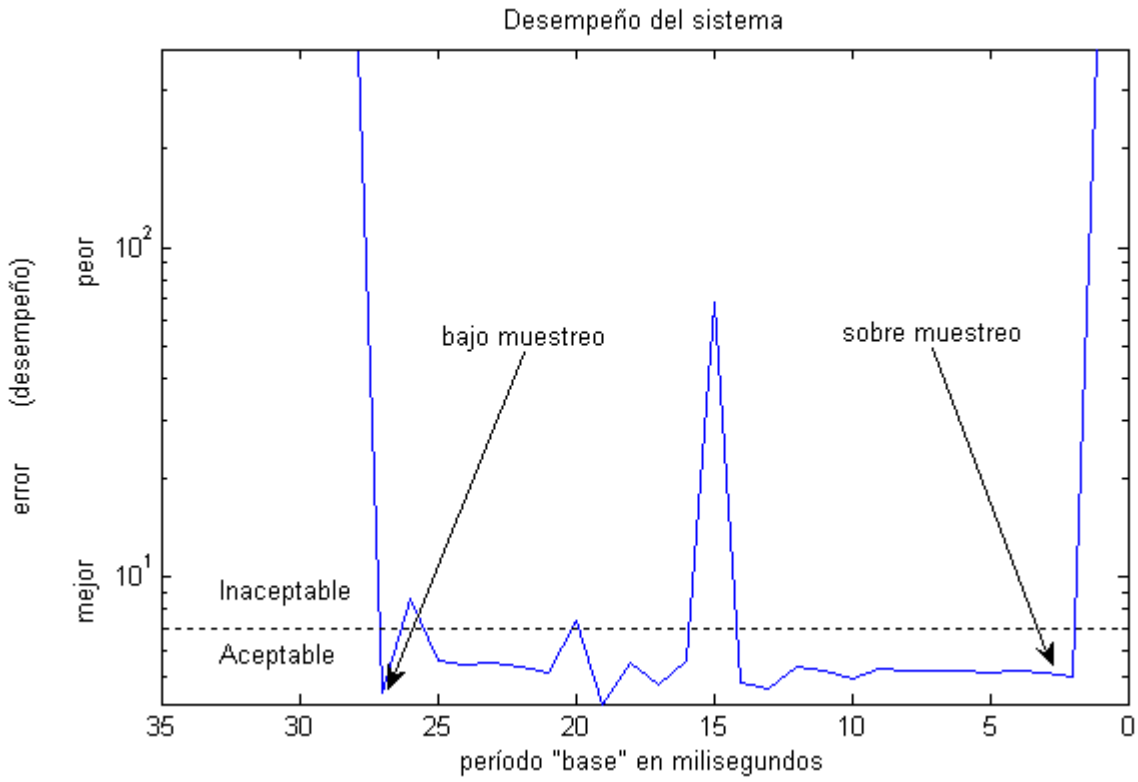


Figura 5-9. Desempeño del sistema. Período de muestreo en 10^{-3} segundos contra el error logarítmico.

La siguiente consideración es el ancho de banda asignado a cada nodo del DTRS. Cuando todos los nodos compiten por obtener acceso a la red y lo obtienen por medio del algoritmo de control de acceso a la red (por ejemplo el protocolo de prioridades fijas CAN), el sistema tiende a un estado inestable como puede ser inferido en la Figura 5-10, en la que la diferencia entre el ángulo real θ y el deseado θd se incrementa conforme pasa el tiempo. En este caso de estudio se elimina este problema al tener un nodo planificador que asigna a cada nodo una ventana de tiempo en la que puede transmitir..

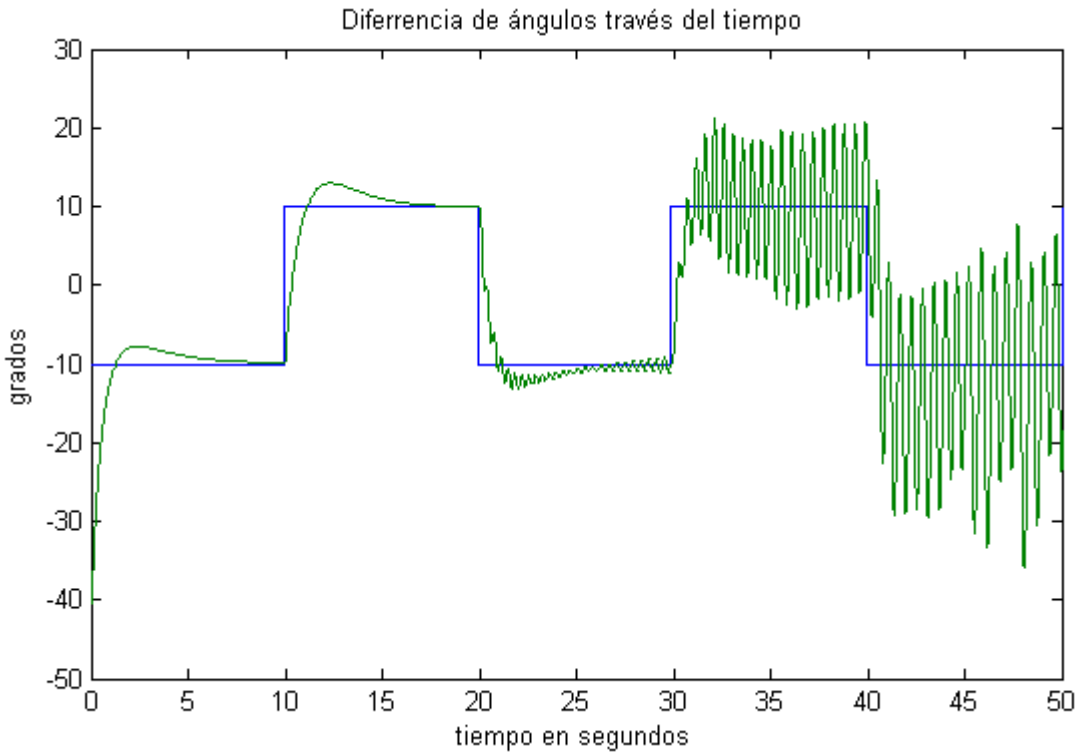


Figura 5-10. Error a través del tiempo (Diferencia entre el ángulo real θ el deseado θd).

Resultados

Una serie de pruebas fueron ejecutadas con las siguientes características. el período "base" con valores de 0.001 segundos y a 0.027 segundos (puntos de sobre y bajo-muestreo en la Figura 5-9, con incrementos de 0.001 segundos. El factor de dispersión en cada caso va de 0 a 0.2 en incrementos de 0.05. El tiempo total de ejecución de cada prueba es de 25 segundos.

La gráfica de la Figura 5-11 muestra el número total de transmisiones hechas por todos los nodos durante la pruebas graficadas contra los diferentes períodos "base" y factores de dispersión. Se aprecia que el número total de transmisiones depende del período "base" , distribuido uniformemente entre los factores de dispersión.

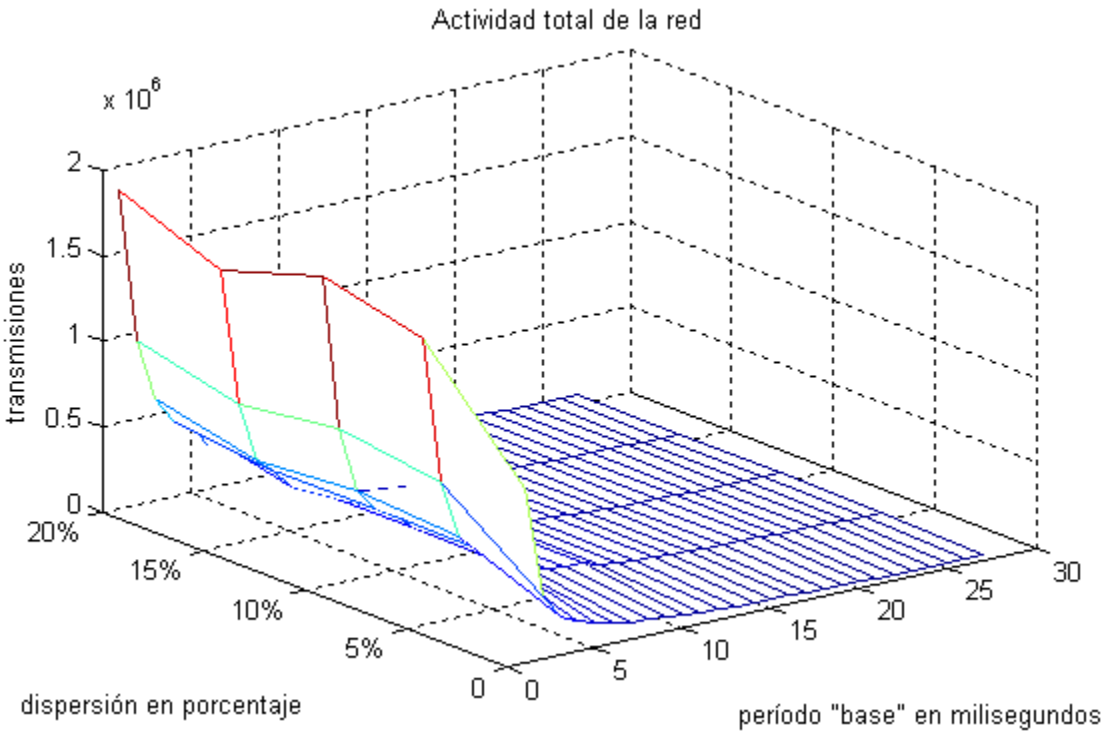


Figura 5-11. Número total de transmisiones variando el período “base” y el factor de dispersión..

Al igual que el número total de transmisiones, el número de transmisiones por nodo depende básicamente del período “base” y como se aprecia en la Figura 5-12 el número d e transmisiones es inversamente proporcional al período “base”. Ya que el factor de dispersión tiene un impacto prácticamente nulo, no se incluye en la gráfica.

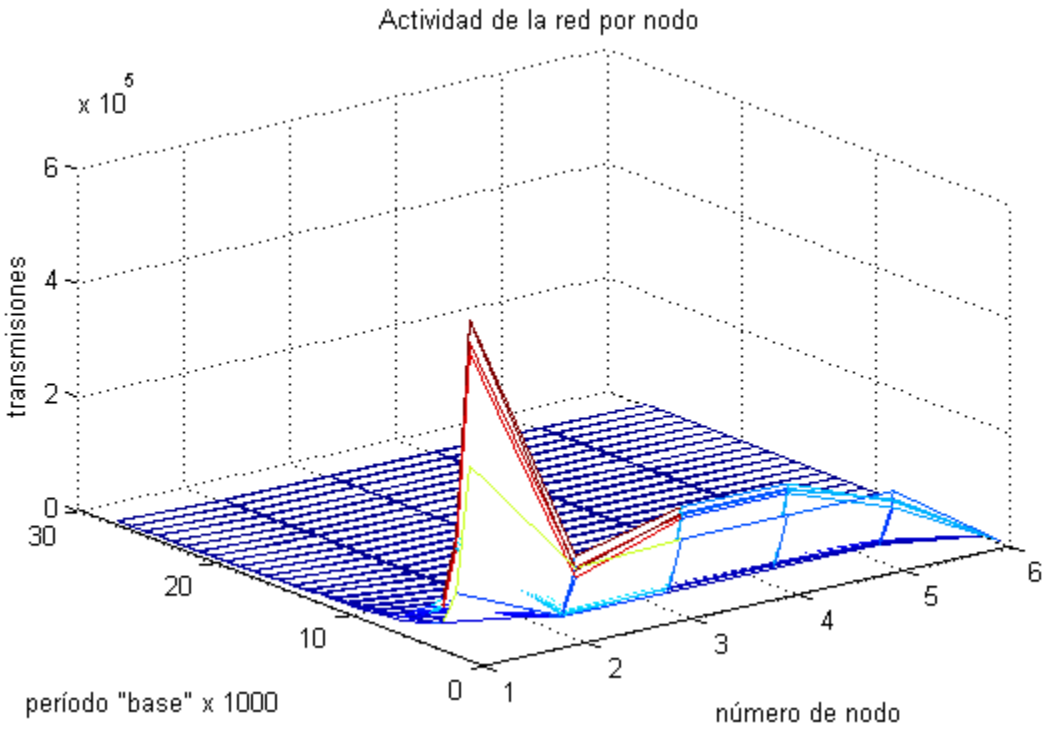


Figura 5-12.Relación entre el número de transmisiones por nodo y el período “base”..

La siguiente gráfica muestra el error acumulado en la serie de pruebas, mostrando que el desempeño aceptable del RTDS depende no solo del período “base” sino también de las dependencias temporales entre sus componentes.

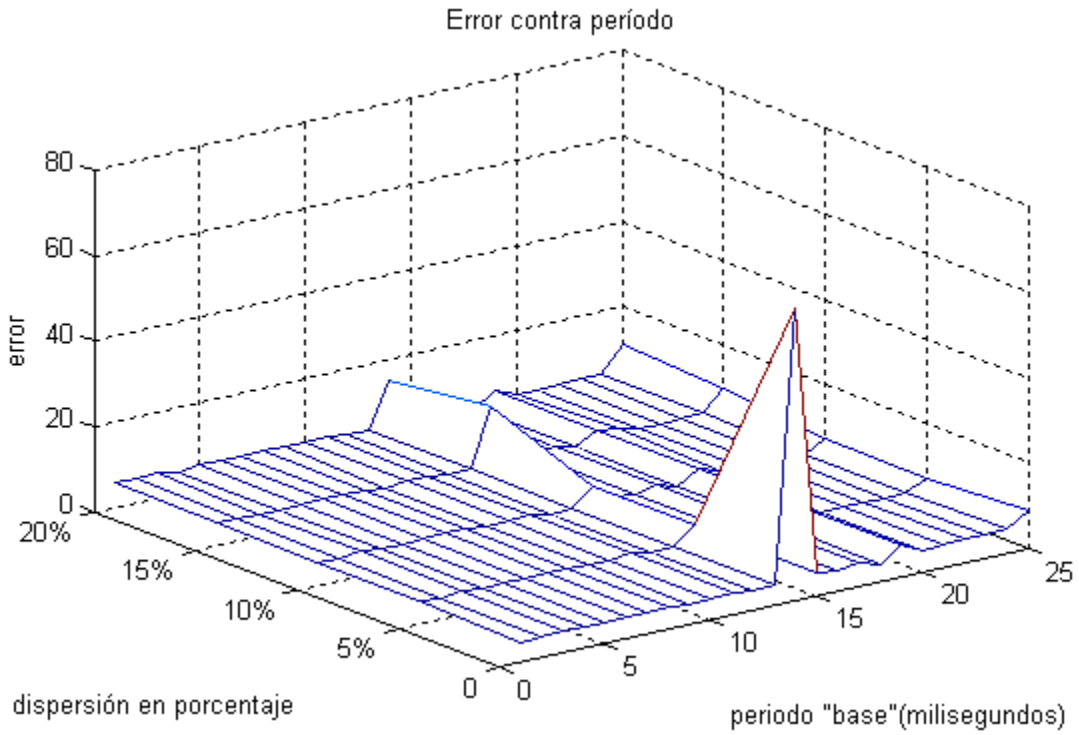


Figura 5-13. Error del sistema

En la gráfica del Error del sistema se aprecia que algunos períodos de operación tienen un error más grande de lo aceptable, por ejemplo cuando el período “base” es de 15 milisegundos y el factor de dispersión es cero, el error acumulado después de 25 segundos es de 80. Tanto el error del sistema (Figura 5-13) como el desempeño del mismo (Figura 5-9) muestran que el desempeño aceptable del RTDS y el período “base” no tienen una relación lineal.

En la Figura 5-13 se muestra que la curva del error del sistema no es convexa como se espera de un NCS, en un RTDS los límites de un desempeño aceptable no necesariamente lo garantizan en todos los puntos intermedios, probando con ello que el un RTDS estable depende no solo del período de muestreo, sino también en las relaciones entre dichos períodos y los período de operación de los demás nodos que conforman el RTDS.

En la Figura 5-14 se aprecia como el comportamiento del sistema (utilizando un período “base” de 0.015 segundos) se vuelve inestable conforme pasa el tiempo.

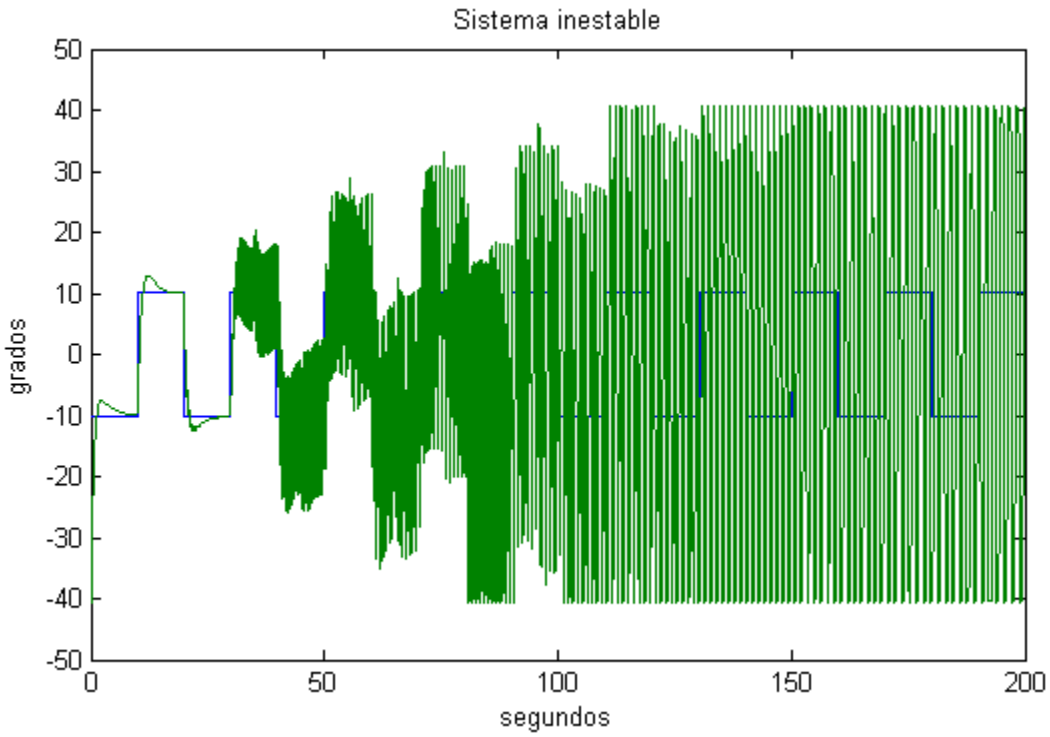


Figura 5-14 Sistema inestable con un período "base" de 0.015 segundos.

El nodo planificador asigna a cada nodo ancho de banda de la red de comunicaciones, al definir una ventana temporal en la que los nodos pueden transmitir, como una solución heurística el controlador recibe 1/3 del ancho de banda y los 2/3s restantes se reparten equitativamente a los nodos sensores. La Figura 5-15 muestra claramente que la capacidad de la red no es el recurso restringente del RTDS ya que claramente se aprecia que tiene capacidad sobrada. Nuevamente se hace énfasis en las relaciones temporales entre los componentes del sistema.

La gráfica en la Figura 5-15 tiene cinco líneas horizontales que representan las transmisiones durante un segundo del nodo controlador (nodo 1) y los cuatro sensores (nodos 2 al 5). Cada una de las líneas representa tres posibles estados del nodo (ver Figura 5-16) que son: sin transmitir, esperando disponibilidad de la red para transmitir y transmitiendo (sumando al número de nodo los valores de 0, 0.25 y 0.5 respectivamente). Es fácil apreciar que la mayoría del tiempo los nodos están sin trasmitir ya que sus respectivas líneas están en el nivel bajo.

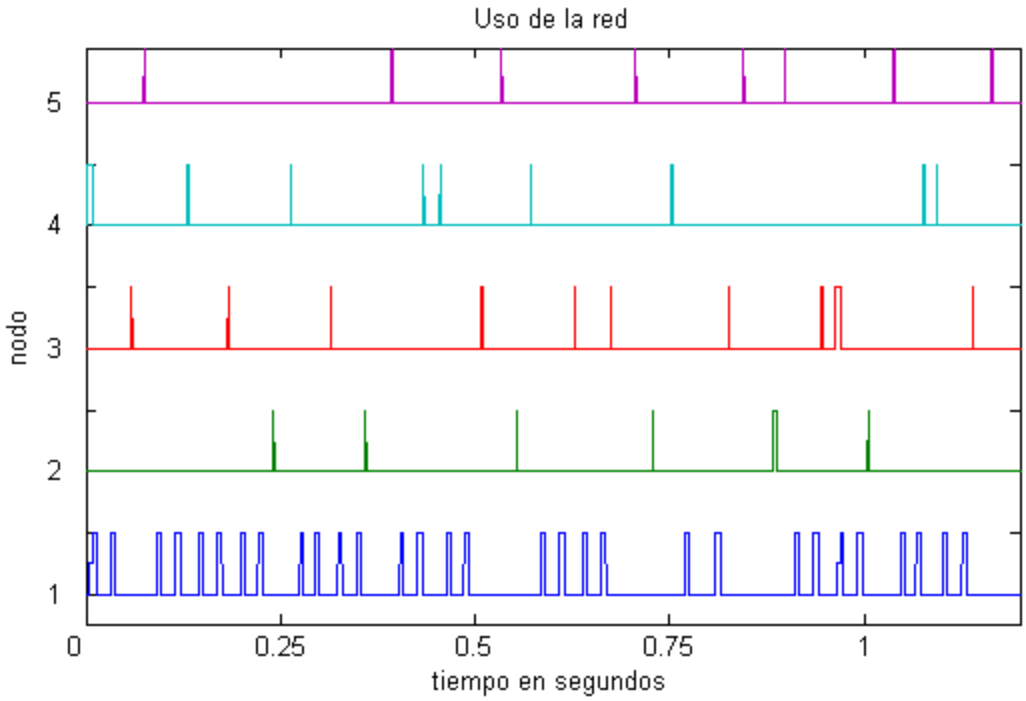


Figura 5-15 Uso de la red.

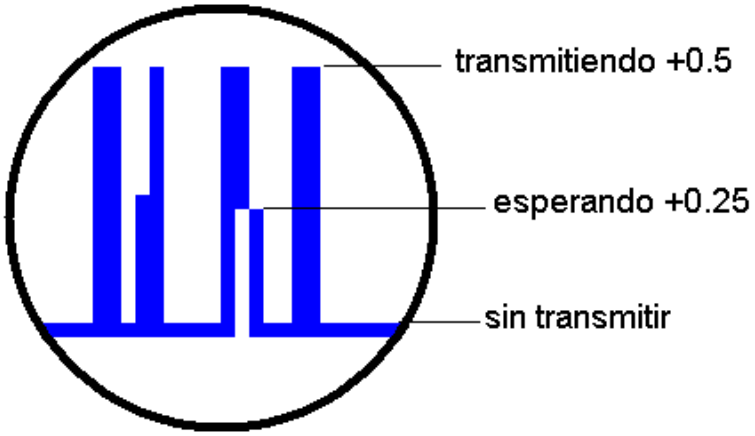


Figura 5-16 Acercamiento a la utilización de la red, los tres estados de un nodo.

En el caso de sobre-muestreo la red de comunicaciones está ocupada el 100% del tiempo y en varias ocasiones los nodos están esperando para poder transmitir (ver Figura 5-17). Está falta de disponibilidad de la red resulta en pérdidas de plazos de las tareas en todos los nodos, ocasionando que si llegan a tener acceso a la red, la información que el controlador y los

sensores transmiten es en la mayoría de los casos obsoleta, por lo que el sistema al operar prácticamente sin datos válidos se vuelve inestable casi de inmediato.

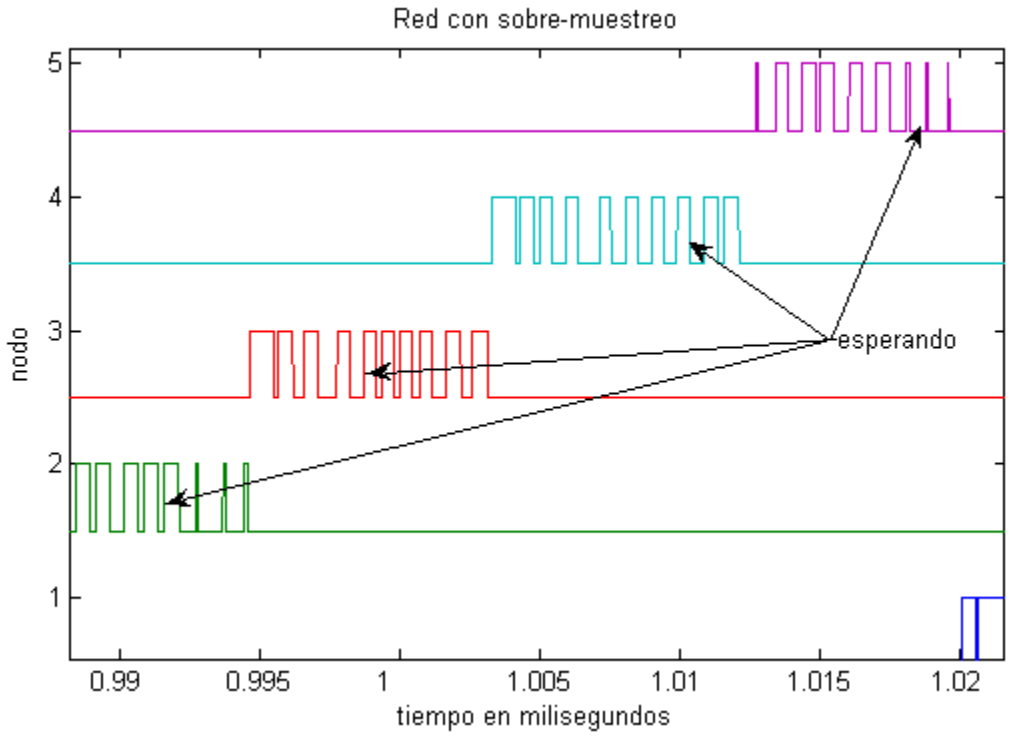


Figura 5-17 Utilización de la red con sobre-muestreo.

Conclusiones

Al no tener un mecanismo para validar el comportamiento temporal del RTDS de manera global, es posible tener un sistema planificable a nivel componente pero inestable como NCS.

Un RTDS como NCS depende no solamente de los períodos de operación y muestreo en cada nodo, sino también de las relaciones entre ellos.

La capacidad de la red no es necesariamente el recurso restringente del RTDS. Nuevamente se hace énfasis en las relaciones temporales entre los componentes del sistema ya que se puede presentar un desempeño inaceptable a cuando un alto porcentaje del abcho de banda no se utilice.

Un planificador global para el RTDS, aún tan simple como el propuesto, permite incrementar el rango de períodos que presenta un desempeño aceptable del NCS.

6 Conclusiones y trabajo futuro

La planificación y el análisis de panificabilidad para un procesador ejecutando tareas en Tiempo-Real es un área de estudio muy desarrollada, no así cuando se trata de múltiples procesadores. De manera similar, las redes de comunicaciones en Tiempo-Real han tenido un desarrollo notable. Existen varios algoritmos que permiten planificar un conjunto de tareas con características de Tiempo-Real en un procesador (p. Ej. RM y EDF), también hay redes y protocolos de comunicaciones que cumplen con las restricciones temporales de los sistemas en Tiempo-Real (p. Ej. CAN y FTT).

Para poder conocer en forma determinista el comportamiento temporal de un sistema distribuido en Tiempo Real es necesario modelarlo formalmente, para ello se ha definido una arquitectura básica capaz de soportar aplicaciones en Tiempo-Real y su modelo con álgebra de procesos. El modelado en capas del sistema distribuido en Tiempo-Real permite ver las relaciones existentes entre estas capas y las tareas ejecutándose en cada una de ellas por lo que surge la propuesta de utilizar un planificador global.

Una alternativa para garantizar el cumplimiento de las restricciones temporales de un sistema distribuido en Tiempo-Real es el contar con un algoritmo de planificación que permita agrupar los recursos de un RTDS (nodos y red de comunicaciones) en uno solo, como el recurso “Padre” en una jerarquía de planificadores (ver Figura 2-5).

Parta determinar si un sistema distribuido en Tiempo-Real cumple con todas sus restricciones temporales de manera eficiente se debe contar con una métrica que permita evaluar el rendimiento de un sistema con estas características.

Es deseable el contar con algún planificador que permita determinar si un sistema distribuido en Tiempo-Real cumplirá con todas sus restricciones temporales, más aún que exista alguna métrica que permita evaluar el rendimiento de un sistema con estas características. La estrategia de planificación propuesta en el capítulo 2, fundamentada en un período “base” global, es una alternativa viable para estos dos puntos.

Al no tener un mecanismo para validar el comportamiento temporal de un RTDS de manera global, es posible tener un sistema planificable a nivel componente pero no predecible a como sistema.

Un RTDS como NCS depende no solamente de los períodos de operación y muestreo en cada nodo, sino también de las relaciones entre ellos, por lo que se requiere realizar un mayor análisis de las relaciones existentes entre los períodos de operación de los nodos de un RTDS.

La capacidad de la red no es necesariamente el recurso restringente del RTDS. Nuevamente se hace énfasis en las relaciones temporales entre los componentes del sistema ya que se puede presentar un desempeño inaceptable a cuando un alto porcentaje del ancho de banda no se utilice.

La métrica de disponibilidad del nodo permite tener una visión más amplia de la carga impuesta a los nodos por las tareas que deben ejecutar, al considerar no solamente la utilización teórica del procesador sino además: los tiempos de las comunicaciones, las tareas inherentes a cualquier nodo participando en un sistema distribuido, el uso real del procesador y la disponibilidad de memoria.

La métrica de disponibilidad del nodo (Φ) como un criterio para realizar una distribución de tareas entre los nodos del RTDS, permite tener un RTDS con grado de utilización (U) similar en cada uno de los nodos que lo integran. La métrica provee información complementaria como el tiempo para realizar las comunicaciones externas, lo cual se puede interpretar como la especificación de la velocidad de la red.

Un planificador global para el RTDS, aún uno tan simple como el propuesto, permite incrementar el rango de períodos que presenta un desempeño aceptable del NCS

No existe ningún algoritmo robusto y preciso que permita realizar un análisis de planificabilidad a los sistemas distribuidos en Tiempo-Real duro para validarlos como sistemas multiprocesador dinámicos.

El uso de composición de planificadores o técnicas relacionadas debe ser investigado para poder tener un planificador global de RTDS que permita realizar análisis formales de la planificabilidad de los sistemas distribuidos en tiempo-real

Este trabajo presenta un planificador global para sistemas distribuidos en tiempo-real, que si bien tiene muchas ventajas también presenta áreas de oportunidad y se seguirá trabajando con el tanto en investigaciones abiertas como en tesis doctorales.

7 Lista de figuras

8 Anexos

8.1 Código Sistema distribuido de alto rendimiento (Cluster)

Balanceo de cargas entre el nodo más disponible (PUN) al menos disponible (PON):

```
function b = Balance(PUN,PON)
global n muestra
apun=size(n(PUN).queue,1)+1;
apon=size(n(PON).queue,1);
acp=0;
b=0;
for i=1:apon
   if n(PON).queue(i,3)==0
      acp=acp+1;
   end
end
if acp>1
   while apon > 1 && n(PON).queue(apon,3) > 0
      apon=apon-1;
   end
   if apon > 2

n(PUN).queue(apun,1)=n(PON).queue(apon,1);
      n(PUN).queue(apun,2)=muestra;
      n(PUN).queue(apun,3)=0;
      n(PUN).queue(apun,4)=PON;
      n(PON).queue(apon,3)=muestra;
      proc_in(PUN);
      proc_out(PON);
   end
   b=1;
end
```

Evaluación del costo o tiempo de las comunicaciones:

```
function cc = commcost(PON,PUN)
global rho tao n
cc =  2* rho + 2* tao;
n(PON).alfa=n(PON).alfa -rho-tao;%*rand-
tao*rand;
n(PUN).alfa=n(PUN).alfa -rho-tao;%*rand-
tao*rand;
n(PON).niu=n(PON).niu + tao;
n(PUN).niu=n(PUN).niu + tao;

if n(PON).alfa<=0
   n(PON).alfa=0.001;
end
if n(PUN).alfa<=0
   n(PUN).alfa=0.001;
end
```

Calculo del promedio de phi las delta muestras:

```
function delphi = deltaphi(i)
global delta phin muestra deltamov
delphi=0;
if muestra>=delta
   if deltamov==0
      for j=muestra:-1:muestra-delta+1
         delphi=delphi+phin(i,j);
      end
      delphi=delphi/delta;
   else
      x=floor(muestra/delta)-1;
         for j=x*delta+1:(x+1)*delta
            delphi=delphi+phin(i,j);
         end
         delphi=delphi/delta;
   end
end
```

Código del ejecutable HILO

```
clear all
global jrho n nn delta phin muestra deltamov
phinb uf omega
%Hilo periodic tasks
rand('state',0)
nn=10;
n=nodos(nn);
tasks;
bt=0;
delta=4;
deltamov=0;
mu=0;
for j=1:1000
   muestra=j;
   for i=1:nn
      phin(i,j)=phi(i);
   end
    if mod(j,delta)==0
      [UN,ON,PUN,PON]= Node_Load;
      mu=ON/UN;

      omega(j)=((1-mu)^2)/2;  %1-mu;
      if j>=delta && omega(j)> 0.01%0.15
         bt=bt+migratetask(PUN,PON);
         [dj(PUN,j), di(PON,j)]=rho(PUN,PON);
      end
   end
```

```
 for i=1:nn
      phinb(i,j)=deltaphi(i);
      uf(i,j)=u(i);
      jrho(i,j)=n(i).ro;
      hp(i,j)=hyperperiod(i);
      tt(i,j)=n(i).epsilon;total tasks
 end

   genera tareas random
    if j< 100 randtasks ; end
matar tareas
   if j>250 && mod(j,50)==0
       for i=1:nn
          deletetask(i);
       end
    end
end
bt
mu
```

Gráficas

```
figure
surf(phinb)
xlabel('Time')
ylabel('Node')
zlabel('Availability')
title('Phi');

figure
surf(uf)
xlabel('Time')
ylabel('Node')
zlabel('Processor-Utilization')
title('U');

figure
plot(omega)
xlabel('Time')
ylabel('Error')
title('Omega-Error');

 figure
 surf(hp)
 title('Hyper-Period');

 figure
 surf(jrho)
 title('rho');

 figure
 surf(dj)
 title('dj');
```

Función para genera un proceso nuevo:

```
function [np,varargout] = new_proc
np=poissrnd(.5);
for i=1:np
   varargout(i)=exprnd(10);
end
function [UN,ON,PUN,PON]=Node_Load
global nn
for i=1:nn
    philb(i)=deltaphi(i);
end
[UN,PUN]=max(philb);
[ON,PON]=min(philb);
function n = nodos(x)
for i=1:x
   n(i).alfa=1;%0.8+0.2*rand;
   n(i).beta=1;%0.8+0.2*rand;
   n(i).epsilon=0;
   n(i).kapa=0;%rand*0.1;
   n(i).ro=0.1;
   n(i).nu=0;%rand*0.2;
   n(i).tao=0.1;
   n(i).queue=[0,0];
end
```

Cálculo de phi

```
function valphi = phi(i)
global n tt muestra
e=0;
for j=1:size(n(i).queue,1)
   if n(i).queue(j,1)>0;e=e+1;end
end
n(i).epsilon=e;
tt(i,muestra)=e;
n(i).kapa =floor(.7 * e);
n(i).nu =floor(.7 * e);
prob=1-(u(i)/2);
n(i).alfa=prob;
n(i).beta=prob;
%valphi=(n(i).alfa * n(i).beta* exp(-
(n(i).epsilon*.1 + n(i).ro * n(i).kapa + n(i).tao*
n(i).nu)));
valphi=(n(i).alfa * n(i).beta* exp(-(n(i).epsilon*.1
+ 0.1 * n(i).kapa + 0.1* n(i).nu)));
if isfinite(valphi)==0||valphi==0;valphi=0.01;end
if valphi>1;valphi=1;end
%if valphi<0;valphi=0;end
endfunction pi = proc_in(i)
global n
n(i).alfa=n(i).alfa - 0.1;
n(i).beta=n(i).beta - 0.1;
if n(i).alfa<=0
   n(i).alfa=0.0001;
end
if n(i).beta<=0
   n(i).beta=0.0001;
```

```
end
```

Eliminación de un proceso de la cola de ejecución:

```
function po = proc_out(i)
global n
n(i).alfa=n(i).alfa + 0.1;
n(i).beta=n(i).beta + 0.1;
if n(i).alfa>1
   n(i).alfa=1;
end
if n(i).beta>1
   n(i).beta=1;
end
```

Valores de la "semilla" para la generación de variables aleatorias:

```
function nula=s
s=[0.169994338896023;0.487587196726568;0.
985471938827335;0.455806170631933;0.7386
44201418533;0.179479177871024;0.66477492
8625908;0.215279723440565;0.588346019826
048;0.984903184828427;0.132820929852715;0
.023224036207863;0.447773646232944;0.5224
64706039011;0.759509923894544;0.21083015
7845802;0.945919196102758;0.963060572415
565;0.061073655352522;0.890463844547086;0
.349473516767047;0.152362125352475;0.2442
07246024703;0.068647719958759;0.00502795
6712605;0.763008920811015;0.018340907499
209;0.529460531635131;0.314939184473262;0
.97758863358823;0.605267484720588;0.15181
9643128554;0;0;0.000000471645426;];
rand('state',s);
```

Cálculo de la utilización del procesador U por cada nodo (i):

```
function ut = u(i)
global n
ut=0;
l=size(n(i).queue,1);
for k=1:l
   if n(i).queue(k,1)==0
      ut=ut+0;
   else
      ut=ut+(n(i).queue(k,2)/n(i).queue(k,1));
   end
end
```

Alta de tareas periódicas con consumo (c) y período (p) en la cola de ejecución del nodo (i):

```
function addt = addtask(i,p,c)
global n
addt=0;

   if u(i)+(c/p)<1
      addt=1;
      l=size(n(i).queue,1);
      for j=1:l
         if n(i).queue(j,1)==0, break, end
      end
      if n(i).queue(j,1)>0; j=j+1; end
      n(i).queue(j,1) = p;
      n(i).queue(j,2) = c;
      n(i).queue = sortqueue(n(i).queue);
   end
end
```

Creación de tareas periódicas:

```
%create tasks
global n
for k=1:5
   for i=1:size(n,2);
      p=int32(rand*10000)+1000;
      c=(rand*10)+1;
      addtask(i,p,c);
   end
end
```

Cálculo de epsilon:

```
function nepsilon = epsi(i)
global n
nepsilon=0;
for k=2:size(n(i).queue,1)
   if n(i).queue(k,3)==0
      nepsilon = nepsilon + 1;
   end
end
```

Cálculo de la utilización promedio del procesador U en delta muestras:

```
function ut = utilization(i)
global n delta muestra deltamov
ut=0;
if muestra>=delta
   if deltamov==0
      for j=muestra:-1:muestra-delta+1
         ut=ut+epsi(i);
      end
   else
      x=floor(muestra/delta)-1;
         for j=x*delta+1:(x+1)*delta
            ut=ut+epsi(i);
         end
   end
```

```
   ut=ut/delta;
end
```

Migración de tareas:

```
function migt = migratetask(j,i)%i=PUN,j=PON
global n
   migt=0;
   l=size(n(i).queue,1);
   while l>1 && n(i).queue(l,1)==0
      l=l-1;
   end
   p=n(i).queue(l,1);
   c=n(i).queue(l,2);
   if p>0 && c>0
   if addtask(j,p,c)==1
      n(i).queue(l,1)=0;
      n(i).queue(l,2)=0;
      migt=1;
   end
   end
end
```

Cálculo de las derivadas parciales del error respecto a rho:

```
function zet= zeta(i)
zet=(u(i)-phi(i))*phi(i)*-1*rho(i);%parcial respecto
a rho
end
function [der_eroj,der_eroi]=rho(PUN,PON)
global muestra n
eta=0.125;
% r=false;
   if muestra>1
%       r=true;
      nmin=n(PUN);
      nmax=n(PON);
      a=muestra-1;

la1=(nmax.alfa*nmax.beta)/(nmin.alfa*nmin.beta
);
      la2=nmin.epsilon-
nmax.epsilon+(nmin.tao*nmin.nu)-
(nmax.tao*nmax.nu)+(nmin.ro*nmin.kapa);
      la3=-nmax.ro*nmax.kapa;
      la4=la2+la3;
      la=la1*exp(la2+la3);
      der_eroj=(1-la)*la1*(-
1*nmax.kapa)*exp(la4);
      der_eroi=(1-la)*la1*nmin.kapa*exp(la4);
      der_etaj=(1-la)*la1*(-1*nmax.nu)*exp(la4);
      der_etai=(1-la)*la1*nmin.nu*exp(la4);

      if isfinite(der_eroj)==0 ||abs(der_eroj)>10;
der_eroj=0;end
      if isfinite(der_eroi)==0 ||abs(der_eroi)>10;
der_eroi=0;end
      if isfinite(der_etaj)==0 ||abs(der_etaj)>10;
der_etaj=0;end
      if isfinite(der_etai)==0 ||abs(der_etai)>10;
der_etai=0;end


n(PUN).ro=nmax.ro+eta*der_eroj;%abs(nmax.ro
+eta*der_eroj);
      n(PON).ro=nmin.ro-
eta*der_eroi;%abs(nmin.ro-eta*der_eroi);
      n(PUN).tao=nmax.ro+eta*der_etaj;
      n(PON).tao=nmin.ro-eta*der_etai;
%       for i=1:10
%          n(i).ro=0;
%          n(i).tao=0;
%       end

      %r=(jrho(i,a)-0.5)*2*((uf(i,a)-
phinb(i,a))*zetha(i,a));% rho i+1 etha=0.5
   end
end
```

Clasificación de la cola de procesos en el nodo, ordenados por consumo:

```
function q = sortqueue(queue)
flag=0;
      l=size(queue,1);
      for j=2:l
         if queue(j-1,1)==0
            if queue(j,1)~=0
               flag=1;
               queue(j-1,1)=queue(j,1);
               queue(j-1,2)=queue(j,2);
            end
         else
            if queue(j,1)>0 && queue(j-1,1)>0
               if  (queue(j,2)/queue(j,1)>queue(j-
1,2)/queue(j-1,1))
                  flag=1;
                  sw=queue(j-1,1:2);
                  queue(j-1,1)=queue(j,1);
                  queue(j-1,2)=queue(j,2);
                  queue(j,1)=sw(1);
                  queue(j,2)=sw(2);
               end
            end
         end
      end
   if flag==1; sortqueue(queue); end
q=queue;
```

Cálculo del hiperperíodo:

```
function hp=hyperperiod(i)
global n
hp=1;

   l=size(n(i).queue,1);
   for j=1:l
      if n(i).queue(j,1)~=0
         hp=lcm(hp,n(i).queue(j,1));
      end
   end
```

Creación de tareas aleatorias:

```
%create random tasks
global n

   for i=1:size(n,2);
      if poissrnd(1)>2
         p=int16(exprnd(50))+1;
         c=int16(exprnd(5))+1;
         addtask(i,p,c);
      end
   end
```

Borrado de tarea:

```
function delt = deletetask(i)
global n
   delt=0;
   l=size(n(i).queue,1);
   while l>1 && n(i).queue(l,1)==0
      l=l-1;
   end
   p=n(i).queue(l,1);
   c=n(i).queue(l,2);
   if p>0 && c>0
      n(i).queue(l,1)=0;
      n(i).queue(l,2)=0;
      delt=1;
   end
end
```

Cálculo de demanda mínima ildb:

```
function ild = ildb(i,t)
global n
dbf=0;
for j=1:size(n(i).queue,1)
   pi=n(i).queue(j,1);
   if pi>0
      ci=n(i).queue(j,2);
      dbf=dbf+floor(t/pi)*ci;
   end
end
   ldb=u(i)*t;
   ild=ldb-dbf;
end
```

8.2 Código Sistema distribuido de control en Tiempo-Real (Helicóptero)

Código de los actuadores:

```
function [exectime, data] = actcode(seg, data)
global actuator_code
switch seg,
   case 1,
      exectime=actuator_code(data);
   case 2,
      exectime = -1;
end

function actuator_init(arg)
global actuator_init_period extra_task;
ttInitKernel(0, 1, 'prioFP'); % nbrOfInputs,
nbrOfOutputs, fixed priority
prio = 2;
offset=0;
period=actuator_init_period(arg);
   ttCreatePeriodicTask(['pid_task',arg], offset,
period, prio, 'actcode',arg);

% Initialize network
prio=1;
ttCreateInterruptHandler('nw_handler1', prio,
'act_rcv',arg);
ttInitNetwork(arg, 'nw_handler1'); % node(# arg)
in the network
% extra tasks
prio=2;
for i=1:4
   n=arg-1;
   period=extra_task(n,i,1);
   c=extra_task(n,i,2);
   tn=arg*10+i;
   ttCreatePeriodicTask(['pid_task',tn], 0, period,
prio, 'extrataskcode',c);
end

function [exectime, data] = act_rcv(seg, data)
global actuator_receive_time
switch seg,
   case 1,
      d = ttGetMsg;
      et=actuator_receive_time(data);
      if d.o ~= 8
         ttAnalogOut(1, d.v);
      else
         et=d.v;
      end
      exectime=et;
   case 2,
      exectime = -1;
end
```

Código del controlador:

```
function [exectime, data] = controlercode(seg,
data)
global controler_time sched_w

switch seg,
   case 1,
      v.o=1;
      v.v=ttAnalogIn(1);
      if sched_w==1; ttSendMsg(6, v, 4); end
      exectime=controler_time;
    case 2,
      v.o=1;
      v.v=ttAnalogIn(2);
      if sched_w==1; ttSendMsg(7,v,4); end
      exectime=controler_time;
   case 3,
      exectime = -1;
end

function controler_init(arg)
global controler_period sched_ctrl
ttInitKernel(2, 4, sched_ctrl); % nbrOfInputs,
nbrOfOutputs, fixed priority
prio = 1;
offset=0;
period=controler_period;
ttCreatePeriodicTask('pid_task1', offset, period,
prio, 'controlercode',arg);

% Initialize network
prio=2;
ttCreateInterruptHandler('nw_handler1', prio,
'controler_rcv',arg);
ttInitNetwork(arg, 'nw_handler1'); % node #(arg)
in the network

% Initialize network activity if node 1 replaced
with block
% if arg==1
%    data=arg;
%    deadline = 2;
%    prio = 1;
%    ttCreateTask('pid_task9', deadline, prio,
'initnet', data);
%    ttCreateJob('pid_task9');
% end
%missed deadline
%prio=4;
```

```
%ttAttachDLHandler(taskname, handlername)
% ttCreateInterruptHandler('dl_handler1', prio,
'deadline1');
% ttAttachDLHandler('pid_task2', 'dl_handler1');

function [exectime, data] = controler_rcv(seg,
data)

switch seg,
   case 1,
      d = ttGetMsg;
      port=d.n - 1;
      v=d.val;
      ttAnalogOut(port,v);
      exectime=0.0001;
   case 2,
      exectime = -1;
end
```

Código del controlador provisto por el fqabricante:

```
% D_HELI_2D_LQR

% Control Lab: Design of a LQR Controller for
the Quanser 2 DOF Helicopter.

% D_HELI_2D_LQR designs a LQR controller
for the 2D HELI system,
% and returns the corresponding gain vector: K

% Copyright (C) 2006 Quanser Consulting Inc.
% Quanser Consulting Inc.


function [ K ] = d_heli_2d_lqr( A, B, C, D, Q, R )
%PLOT_RESPONSE = 'YES';
PLOT_RESPONSE = 'NO';
%SYS_ANALYSIS = 'YES';
SYS_ANALYSIS = 'NO';
%
% Open Loop system
HELI_2D_OL_SYS = ss( A, B, C, D, 'statename',
{ '\theta' '\psi' '\theta_dot' '\psi_dot' }, 'inputname',
{'u_p' 'u_y'} , 'outputname', { '\theta' '\psi'
'\theta_dot' '\psi_dot' } );
%
% calculate the LQR gain vector, K
[ K, S, EIG_CL ] = lqr( A, B, Q, R );
%
% Closed-Loop State-Space Model
A_CL = A - B * K;
B_CL = B;
C_CL = C;
D_CL = D;
%
% Closed-Loop System
HELI_2D_CL_SYS = ss( A_CL, B_CL, C_CL,
D_CL, 'statename', { '\theta' '\psi' '\theta_dot'
'\psi_dot' }, 'inputname', {'\theta_d' '\psi_d'} ,
'outputname', { '\theta' '\psi' '\theta_dot' '\psi_dot'
} );
%
if strcmp( PLOT_RESPONSE, 'YES' )
   % initialization
   close all
   fig_h = 1; % figure handle number
   %
   % let's look at the step response of the
closed-loop system
   % unit step closed-loop response of all 4
states
   figure( fig_h )
   % plotting of a step response
   step( HELI_2D_CL_SYS )
   set( fig_h, 'name', strcat( 'Closed-Loop
System: 2 DOF HELI + LQR' ) )
   grid on
   orient tall
   %
   %
   % unit step closed-loop response
   [ yss, tss, xss ] = step( HELI_2D_CL_SYS );
   fig_h = fig_h + 1;
   figure( fig_h )
   subplot( 2, 1, 1 )
   plot( tss, ( yss( :, 1, 1 ) * 180 / pi ) )
   grid on
   title( 'Unit Step Response on \theta' )
   xlabel( 'Time (s)' )
   ylabel( '\theta (deg)' )
   subplot( 2, 1, 2 )
   plot( tss, ( yss( :, 2, 1 ) * 180 / pi ) )
   grid on
   xlabel( 'Time (s)' )
   ylabel( '\psi (deg)' )
   set( fig_h, 'name', strcat( 'Closed-Loop
System: 2 DOF HELI + LQR' ) )
   %
   fig_h = fig_h + 1;
   figure( fig_h )
   subplot( 2, 1, 1 )
   plot( tss, ( yss( :, 1, 2 ) * 180 / pi ) )
   grid on
   title( 'Unit Step Response on \psi' )
   xlabel( 'Time (s)' )
   ylabel( '\theta (deg)' )
   subplot( 2, 1, 2 )
   plot( tss, ( yss( :, 2, 2 ) * 180 / pi ) )
   grid on
   xlabel( 'Time (s)' )
   ylabel( '\psi (deg)' )
```

```
   set( fig_h, 'name', strcat( 'Closed-Loop
System: 2 DOF HELI + LQR' ) )
   fig_h = fig_h + 1;
end

% carry out some additional system analysis
if strcmp( SYS_ANALYSIS, 'YES' )
   ULABELS = [ 'u_p u_y' ];
   XLABELS = [ '\theta \psi \theta_dot \psi_dot' ];
   YLABELS = XLABELS;
   % print the Open-Loop State-Space Matrices
   disp( 'Open-Loop System' )
   printsys( A, B, C, D, ULABELS, YLABELS,
XLABELS )
   % one unstable open-loop pole
   OL_poles = eig( A )
   % print the Closed-Loop State-Space
Matrices
   disp( 'Closed-Loop System' )
   printsys( A_CL, B_CL, C_CL, D_CL,
ULABELS, YLABELS, XLABELS )
   % Closed-Loop poles, damping, and natural
frequency
   damp( HELI_2D_CL_SYS )
   % or: Closed-Loop eigenvalues
   CL_poles = eig( A_CL ); % = EIG_CL
end
% end of function 'd_heli_2d_lqr( )'
% D_HELI_2D_LQR_I
%
% Control Lab: Design of a LQR+I Controller for
the Quanser 2 DOF Helicopter.
%
% D_HELI_2D_LQR designs a LQR+I controller
for the 2D HELI system,
% and returns the corresponding gain vector: K
%
% Copyright (C) 2005 Quanser Consulting Inc.
% Quanser Consulting Inc.


function [ K ] = d_heli_2d_lqr( A, B, C, D, Q, R )

% Augment the state.
A = [
   A(1,:) 0 0;
   A(2,:) 0 0;
   A(3,:) 0 0;
   A(4,:) 0 0;
   1 0 0 0 0 0;
   0 1 0 0 0 0 ];
   %
B = [
   B;
   0 0;
   0 0];
%
C = [
   C(1,:) 0 0;
   C(2,:) 0 0;
   C(3,:) 0 0;
   C(4,:) 0 0;
   0 0 0 0 0 0;
   0 0 0 0 0 0 ];
%
D = zeros(6,2);
%
%PLOT_RESPONSE = 'YES';
PLOT_RESPONSE = 'NO';
%SYS_ANALYSIS = 'YES';
SYS_ANALYSIS = 'NO';
%
% Open Loop system
HELI_2D_OL_SYS = ss( A, B, C, D, 'statename',
{ '\theta' '\psi' '\theta_dot' '\psi_dot' '\theta_int'
'\psi_int'}, 'inputname', {'u_p' 'u_y'} ,
'outputname', { '\theta' '\psi' '\theta_dot' '\psi_dot'
'\theta_int' '\psi_int' } );
%
% calculate the LQR gain vector, K
[ K, S, EIG_CL ] = lqr( A, B, Q, R );
%
% Closed-Loop State-Space Model
A_CL = A - B * K;
B_CL = B;
C_CL = C;
D_CL = D;
%
% Closed-Loop System
HELI_2D_CL_SYS = ss( A_CL, B_CL, C_CL,
D_CL, 'statename', { '\theta' '\psi' '\theta_dot'
'\psi_dot' '\theta_int' '\psi_int' }, 'inputname',
{'\theta_d' '\psi_d'} , 'outputname', { '\theta' '\psi'
'\theta_dot' '\psi_dot' '\theta_int' '\psi_int' } );
%
if strcmp( PLOT_RESPONSE, 'YES' )
   % initialization
   close all
   fig_h = 1; % figure handle number
   %
   % let's look at the step response of the
closed-loop system
   % unit step closed-loop response of all 4
states
   figure( fig_h )
   % plotting of a step response
   step( HELI_2D_CL_SYS )
   set( fig_h, 'name', strcat( 'Closed-Loop
System: 2 DOF HELI + LQR + I' ) )
   grid on
   orient tall
   %
```

```
    %
    % unit step closed-loop response
    [ yss, tss, xss ] = step( HELI_2D_CL_SYS );
    fig_h = fig_h + 1;
    figure( fig_h )
    subplot( 2, 1, 1 )
    plot( tss, ( yss( :, 1, 1 ) * 180 / pi ) )
    grid on
    title( 'Unit Step Response on \theta' )
    xlabel( 'Time (s)' )
    ylabel( '\theta (deg)' )
    subplot( 2, 1, 2 )
    plot( tss, ( yss( :, 2, 1 ) * 180 / pi ) )
    grid on
    xlabel( 'Time (s)' )
    ylabel( '\psi (deg)' )
    set( fig_h, 'name', strcat( 'Closed-Loop
System: 2 DOF HELI + LQR' ) )
    %
    fig_h = fig_h + 1;
    figure( fig_h )
    subplot( 2, 1, 1 )
    plot( tss, ( yss( :, 1, 2 ) * 180 / pi ) )
    grid on
    title( 'Unit Step Response on \psi' )
    xlabel( 'Time (s)' )
    ylabel( '\theta (deg)' )
    subplot( 2, 1, 2 )
    plot( tss, ( yss( :, 2, 2 ) * 180 / pi ) )
    grid on
    xlabel( 'Time (s)' )
    ylabel( '\psi (deg)' )
    set( fig_h, 'name', strcat( 'Closed-Loop
System: 2 DOF HELI + LQR' ) )
    fig_h = fig_h + 1;
end

% carry out some additional system analysis
if strcmp( SYS_ANALYSIS, 'YES' )
    ULABELS = [ 'u_p u_y' ];
    XLABELS = [ '\theta \psi \theta_dot \psi_dot
\theta_int \psi_int' ];
    YLABELS = XLABELS;
    % print the Open-Loop State-Space Matrices
    disp( 'Open-Loop System' )
    printsys( A, B, C, D, ULABELS, YLABELS,
XLABELS )
    % one unstable open-loop pole
    OL_poles = eig( A )
    % print the Closed-Loop State-Space
Matrices
    disp( 'Closed-Loop System' )
    printsys( A_CL, B_CL, C_CL, D_CL,
ULABELS, YLABELS, XLABELS )
    % Closed-Loop poles, damping, and natural
frequency
    damp( HELI_2D_CL_SYS )
    % or: Closed-Loop eigenvalues
    CL_poles = eig( A_CL ); % = EIG_CL
end
% end of function 'd_heli_2d_lqr_i( )'
function e = errcuad (scope,b)
e=0;
l=length(scope);
if nargin > 1
    for i=1:l
        if scope(i,1)>=b, break, end
    end
    ini=i-1;
else
    ini=1;
end
for i=ini:l
    e = e + ((scope(i,2)-scope(i,3))^2)/2;
end
e=e/(i-1);function [exectime, data] =
extrataskcode(seg, data)
switch seg,
    case 1,
        exectime=data;
    case 2,
        exectime = -1;
end
% Matlab equation file:
"HELI_2D_ABCD_eqns.m"
% Open-Loop State-Space Matrices: A, B, C,
and D
% for the Quanser 2 DOF Helicopter
Experiment.

A( 1, 1 ) = 0;
A( 1, 2 ) = 0;
A( 1, 3 ) = 1;
A( 1, 4 ) = 0;
A( 2, 1 ) = 0;
A( 2, 2 ) = 0;
A( 2, 3 ) = 0;
A( 2, 4 ) = 1;
A( 3, 1 ) = 0;
A( 3, 2 ) = 0;
A( 3, 3 ) = -B_p/(J_eq_p+m_heli*l_cm^2);
A( 3, 4 ) = 0;
A( 4, 1 ) = 0;
A( 4, 2 ) = 0;
A( 4, 3 ) = 0;
A( 4, 4 ) = -B_y/(J_eq_y+m_heli*l_cm^2);

B( 1, 1 ) = 0;
B( 1, 2 ) = 0;
B( 2, 1 ) = 0;
B( 2, 2 ) = 0;
B( 3, 1 ) = K_pp/(J_eq_p+m_heli*l_cm^2);
```

```
B( 3, 2 ) = K_py/(J_eq_p+m_heli*l_cm^2);
B( 4, 1 ) = K_yp/(J_eq_y+m_heli*l_cm^2);
B( 4, 2 ) = K_yy/(J_eq_y+m_heli*l_cm^2);

C( 1, 1 ) = 1;
C( 1, 2 ) = 0;
C( 1, 3 ) = 0;
C( 1, 4 ) = 0;
C( 2, 1 ) = 0;
C( 2, 2 ) = 1;
C( 2, 3 ) = 0;
C( 2, 4 ) = 0;
C( 3, 1 ) = 0;
C( 3, 2 ) = 0;
C( 3, 3 ) = 1;
C( 3, 4 ) = 0;
C( 4, 1 ) = 0;
C( 4, 2 ) = 0;
C( 4, 3 ) = 0;
C( 4, 4 ) = 1;

D( 1, 1 ) = 0;
D( 1, 2 ) = 0;
D( 2, 1 ) = 0;
D( 2, 2 ) = 0;
D( 3, 1 ) = 0;
D( 3, 2 ) = 0;
D( 4, 1 ) = 0;
D( 4, 2 ) = 0;
```

Código de inicialización del simulador, incluye períodos de muestreo y de operación, algoritmos de planificación, períodos y consumos de tareas y valores devariables globales:

```
global sensor_init_period sensor_code
actuator_init_period actuator_code
actuator_receive_time ...
   controler_period controler_time
scheduler_period extra_task sched_ctrl
sched_sensor

sched_ctrl='prioRM';
sched_sensor='prioRM';
red=0.025; %sobre-muestreo 0.0010 o menor,
bajo-muetreo 0.0270 o mayor;
sensor_init_period(2)=red*1.1;
sensor_code(2)=.001;
sensor_init_period(3)=red*0.9;
sensor_code(3)=0.001;
sensor_init_period(4)=red*0.89;
sensor_code(4)=0.001;
sensor_init_period(5)=red*1.11;
sensor_code(5)=0.001;
actuator_init_period(6)=10;
actuator_code(6)=0.0001;
actuator_receive_time(6)=0.001;
actuator_init_period(7)=10;
actuator_code(7)=0.0001;
actuator_receive_time(7)=0.001;
controler_period=red;
controler_time=0.00001;
scheduler_period=red;%0.01;
%tareas adicionales taskpcn (node period
consumtion)
c=0.0065; % consumo .005 RM<>EDF
extra_task(1,1,1)=red; % node 1 task 1 period
(1)= red
extra_task(1,1,2)=c; % node 1 task 1
consumption(2)= c
extra_task(1,2,1)=red;
extra_task(1,2,2)=c;
extra_task(1,3,1)=red;
extra_task(1,3,2)=c;
extra_task(1,4,1)=red;
extra_task(1,4,2)=c;

extra_task(2,1,1)=red;
extra_task(2,1,2)=c;
extra_task(2,2,1)=red;
extra_task(2,2,2)=c;
extra_task(2,3,1)=red;
extra_task(2,3,2)=c;
extra_task(2,4,1)=red;
extra_task(2,4,2)=c;

extra_task(3,1,1)=5;
extra_task(3,1,2)=c;
extra_task(3,2,1)=8;
extra_task(3,2,2)=c;
extra_task(3,3,1)=6;
extra_task(3,3,2)=c;
extra_task(3,4,1)=7;
extra_task(3,4,2)=c;

extra_task(4,1,1)=5;
extra_task(4,1,2)=c;
extra_task(4,2,1)=8;
extra_task(4,2,2)=c;
extra_task(4,3,1)=6;
extra_task(4,3,2)=c;
extra_task(4,4,1)=7;
extra_task(4,4,2)=c;

extra_task(5,1,1)=5;
extra_task(5,1,2)=c;
extra_task(5,2,1)=8;
extra_task(5,2,2)=c;
extra_task(5,3,1)=6;
extra_task(5,3,2)=c;
extra_task(5,4,1)=7;
```

```
extra_task(5,4,2)=c;

extra_task(6,1,1)=5;
extra_task(6,1,2)=1;
extra_task(6,2,1)=8;
extra_task(6,2,2)=1;
extra_task(6,3,1)=6;
extra_task(6,3,2)=1;
extra_task(6,4,1)=7;
extra_task(6,4,2)=1;
```

Código del nodo planificador:

```
function [exectime, data] = schedulercode(seg,
data)
global sched_w con
switch seg,
   case 1,
      ain=ttAnalogIn(1);
      con=int32(ain*100);
      node=mod(con,6);%12 todos
      if  node < 2 || node > 5 %7 con actuadores
         sched_w=1;
      else
         sched_w=node;
      end
      exectime=0;
   case 2,
      exectime = -1;
end

function [exectime, data] =
schedulercode_msg(seg, data)

switch seg,
   case 1,
      %synchronization execution times
      d.v=0.02;
      d.o=8;
      ttSendMsg(2, d, 10);
      ttSendMsg(3, d, 10);
      ttSendMsg(4, d, 10);
      ttSendMsg(5, d, 10);
      ttSendMsg(6, d, 10);
      ttSendMsg(7, d, 10);
      exectime=0.01;
   case 2,
      exectime = -1;
end

function scheduler_init(arg)
global scheduler_period con
ttInitKernel(1, 0, 'prioFP'); % nbrOfInputs,
nbrOfOutputs, fixed priority
con=0;
prio = 1;
offset=0;
period=scheduler_period;
ttCreatePeriodicTask('pid_task1', offset, period,
prio, 'schedulercode',arg);

% Initialize network
prio=2;
ttCreateInterruptHandler('nw_handler1', prio,
'scheduler_rcv',arg);
ttInitNetwork(arg, 'nw_handler1'); % node #(arg)
in the network

% send synchronization messages to nodes
prio = 3;
offset=0;
period=5; %scheduler_period;
%ttCreatePeriodicTask('pid_task2', offset,
period, prio, 'schedulercode_msg',arg);function
[exectime, data] = scheduler_rcv(seg, data)

switch seg,
   case 1,
     data = ttGetMsg;
     exectime = 1;
   case 3,
     exectime = -1;
end
function [exectime, data] = sensorcode(seg,
data)
global sensor_code sched_w

switch seg,
   case 1,
      v=ttAnalogIn(1);
      port=1;
      d.n=data;
      d.val=v;
      if sched_w==data; ttSendMsg(port, d,
10);end
      exectime=sensor_code(data);   %0.001;
   case 2,
      exectime = -1;
end

function sensor_init(arg)
global sensor_init_period extra_task
sched_sensor
ttInitKernel(1, 0, sched_sensor); % nbrOfInputs,
nbrOfOutputs, fixed priority
prio = 1;
offset=0;
period = sensor_init_period(arg);
ttCreatePeriodicTask(['pid_task',num2str(arg)],
offset, period, prio, 'sensorcode',arg);
% Initialize network
prio=9;
```

```
ttCreateInterruptHandler('nw_handler1', prio,
'sens_rcv',arg);
ttInitNetwork(arg, 'nw_handler1'); % node(# arg)
in the network
% extra tasks
prio=2;
for i=1:4
   n=arg-1;
%     period = sensor_init_period(arg);
   period=extra_task(n,i,1);
   c=extra_task(n,i,2);
   tn=arg*10+i;
   ttCreatePeriodicTask(['pid_task',num2str(tn)],
0, period, prio, 'extrataskcode',c);
endfunction [exectime, data] = sens_rcv(seg,
data)

switch seg,
   case 1,
      d=ttGetMsg;
      exectime=d.v;%0.001;
   case 2,
      exectime = -1;
end
```

Código del fabricante del helicóptero Quanser para configurar la simulación:

```
% SETUP_HELI_2D_CONFIGURATION
%
% SETUP_HELI_2D_CONFIGURATION sets
and returns the model model parameters
% of the Quanser 2 DOF Helicopter plant.
%
%
% Copyright (C) 2006 Quanser Consulting Inc.
% Quanser Consulting Inc.
%
%
function [ K_pp, K_yy, K_yp, K_py, J_eq_p,
J_eq_y, B_eq_p, B_eq_y, m_heli, l_cm, g ] =
setup_heli_2d_configuration( )
%
% Gravitational Constant (m/s^2)
g = 9.81;
% Pitch and Yaw Motor Armature Resistance
(Ohm)
R_m_p = 0.83;
R_m_y = 1.60;
% Pitch and Yaw Motor Current-Torque
Constant (N.m/A)
K_t_p = 0.0182;
K_t_y = 0.0109;
% Pitch and Yaw Motor Voltage-Torque
Constant (N.m/V)
K_yp = K_t_p / R_m_p;
K_py = K_t_y / R_m_y;
% Pitch and Yaw Viscous Damping Constant
(N.m.s/rad)
B_eq_p = 0.8;  % tuned while running simulation
and experiment in parallel
B_eq_y = 0.318; % identified as described in
manual
% Mass of the Helicopter (kg)
m_heli = 1.3872;
% Mass of the Helicopter with Yoke (kg)
m_heli_case = 1.421;
% Distance from Pitch Pivot to Pitch/Front Motor
and Yaw/Back Motor(m)
r_p = 7.75*0.0254;
r_y = (6+5/8)*0.0254;
% Pitch and Yaw Propeller Force-Thrust
Constant found Experimentally (N/V)
K_f_p = 5*0.2074;
K_f_y = 3*0.1426;
% Pitch and Yaw Propeller Torque-Thrust
Constant found Experimentally (N.m/V)
K_pp = K_f_p * r_p;
K_yy = K_f_y * r_y;
% Mass of Pitch/Front Motor and Yaw/Back
Motor (kg)
m_motor_p = 0.292;
m_motor_y = 0.128;
% Mass of Pitch/Front and Yaw/Back Guard +
Propeller(kg)
m_shield = 0.143 + 0.024;
% Total Mass of Pitch and Yaw Motor/Shield
Assemblies (kg)
m_props = ( m_motor_p + m_motor_y + 2 *
m_shield );
% Mass of Helicopter Body moving about Pitch
Axis (kg)
m_body_p = m_heli - m_props;
% Mass of Helicopter Body moving about Yaw
Axis (kg)
m_body_y = m_heli_case - m_props;
% Mass of Metal Shaft Moving about Yaw Axis
(kg)
m_shaft = 0.151;
% Total Length of Helicopter Body (m)
L_body = 19*0.0254;
% Helicopter Center of Mass from Pivot along
Pitch Axis (m)
l_cm = ( (m_motor_p + m_shield ) * r_p +
(m_motor_y + m_shield ) * r_y ) / ( m_props ) ;
% Length of Metal Shaft Moving about Yaw Axis
through slip ring (m)
L_shaft = 11 * 0.0254;
% Pitch and Yaw Motor Rotor Moment of Inertia
(kg.m^2)
J_m_p = 1.91e-6; %1.7070e-5;
J_m_y = 1.374e-4; %1.4471e-5;
```

```
% Moment of Inertia of Helicopter Body about its
CM (kg.m^2)
J_body_p = m_body_p * L_body^2 / 12;
J_body_y = m_body_y * L_body^2 / 12;
% Moment of Inertia of Metal Shaft about Yaw
Axis (kg.m^2)
J_shaft = m_shaft * L_shaft^2 / 3; %;0.0851;
% Moment of Inertia of Pitch Motor + Guard
Assembly about Pivot (kg.m^2)
J_p = ( m_motor_p + m_shield ) * r_p^2;
% Moment of Inertia of Yaw Motor + Guard
Assembly about Pivot (kg.m^2)
J_y = ( m_motor_y + m_shield ) * r_y^2;
% Equivalent Moment of Inertia about Pitch and
Yaw Axis (kg.m^2)
J_eq_p = J_m_p + J_body_p + J_p + J_y;
J_eq_y = J_m_y + J_body_y + J_p + J_y +
J_shaft;
%
% end of setup_heli_2d_configuration()% 
SETUP_LAB_HELI_2D
%
% 2 DOF Helicopter (2DHELI) Control Lab:
% Design of a FF+LQR+I position controller
%
% SETUP_LAB_HELI_2D sets the model
parameters and set the controller
% parameters for the Quanser 2DOF Helicopter
system.
%
% Copyright (C) 2005 Quanser Consulting Inc.
% Quanser Consulting Inc.
%
clear all;
%define periods & consumptions of sensors,
actuators & ctrls tasks

Initialize_PC;
%
% ################ USER-DEFINED 2DOF
HELI CONFIGURATION ################
% Cable Gain used for yaw and pitch axes.
K_CABLE_P = 5;
K_CABLE_Y = 3;
% UPM Maximum Output Voltage (V): YAW has
UPM-15-03 and PITCH has UPM-24-05
VMAX_UPM_P = 24;
VMAX_UPM_Y = 15;
% Digital-to-Analog Maximum Voltage (V): set to
10 for Q4/Q8 cards
VMAX_DAC = 10;
% Pitch and Yaw Axis Encoder Resolution
(rad/count)
K_EC_P = - 2 * pi / ( 4 * 1024 );
K_EC_Y = 2 * pi / ( 8 * 1024 );
% Initial Angle of Pitch (rad)
theta_0 = -40.5*pi/180;
%
% ################ END OF USER-DEFINED
DOF HELI CONFIGURATION
################
%
%
% ################ USER-DEFINED
CONTROLLER/FILTER DESIGN
################
% Anti-windup: integrator saturation (V)
SAT_INT_ERR_PITCH = 5;
SAT_INT_ERR_YAW = 5;
% Anti-windup: integrator reset time (s)
Tr_p = 1;
Tr_y = 1;
%
% Type of Controller: set it to 'LQR_AUTO' or
'MANUAL'
CONTROLLER_TYPE = 'LQR_AUTO';    % LQR
controller design: automatic mode
%CONTROLLER_TYPE = 'MANUAL';    %
controller design: manual mode
%
% Specifications of a second-order low-pass
filter
wcf = 2 * pi * 20; % filter cutting frequency
zetaf = 0.85;        % filter damping ratio
% ################ END OF USER-DEFINED
CONTROLLER DESIGN ################
%
% ################ USER-DEFINED
JOYSTICK SETTINGS ################
% Joystick input X sensitivity used for yaw
(deg/s/V)
K_JOYSTICK_X = 85;
% Joystick input Y sensitivity used for pitch
(deg/s/V)
K_JOYSTICK_Y = 85;
% Joystick input X sensitivity used for yaw
(V/s/V)
K_JOYSTICK_V_X = 10;
% Joystick input Y sensitivity used for pitch
(V/s/V)
K_JOYSTICK_V_Y = 5;
% Pitch integrator saturation of joystick (deg)
INT_JOYSTICK_SAT_LOWER = theta_0 * 180 /
pi;
INT_JOYSTICK_SAT_UPPER = abs(theta_0)  *
180 / pi;
% Deadzone of joystick: set input ranging from -
DZ to +DZ to 0 (V)
JOYSTICK_X_DZ = 0.25;
JOYSTICK_Y_DZ = 0.25;
% ################ END OF USER-DEFINED
JOYSTICK SETTINGS ################
```

```
%
% Set the model parameters of the 2DOF HELI.
% These parameters are used for model
representation and controller design.
[ K_pp, K_yy, K_yp, K_py, J_eq_p, J_eq_y, B_p,
B_y, m_heli, l_cm, g] =
setup_heli_2d_configuration();
%
% For the following state vector: X = [ theta; psi;
theta_dot; psi_dot]
% Initialization the state-Space representation of
the open-loop System
HELI_2D_ABCD_eqns;
%
if strcmp ( CONTROLLER_TYPE, 'LQR_AUTO'
)
   % Feed-forward gain adjustment (V/V)
   K_ff = 1;
   % LQR Controller Design Specifications
   % Q = diag([1 1 0.5 0.5]);
   % R = 0.005*eye(2,2);
   Q = diag([200 200 100 100]);
   R = eye(2,2);
   % Automatically calculate the LQR controller
gain
   [ K ] = d_heli_2d_lqr( A, B, C, D, Q, R );
   % Display the calculated gains
   disp( ' ' )
   disp( 'Calculated LQR controller gain
elements: ' )
   disp( [ 'K = [' num2str( K(1,1),3 ) ' V/rad  '
num2str( K(1,2),3 ) ' V/rad  ' num2str( K(1,3),3 ) '
V.s/rad  '  num2str( K(1,4),3 ) ' V.s/rad]'] )
   disp( [ '    [' num2str( K(2,1),3 ) ' V/rad  '
num2str( K(2,2),3 ) ' V/rad  ' num2str( K(2,3),3 ) '
V.s/rad  '  num2str( K(2,4),3 ) ' V.s/rad]'] )
   %
   % LQR+I Controller Design Specifications
   Qi = diag([Q(1,1) 0.75*Q(2,2) Q(3,3) 2*Q(4,4)
50 50 ]);
   % Automatically calculate the LQR controller
gain
   [ Ki ] = d_heli_2d_lqr_i( A, B, C, D, Qi, R );
   % Display the calculated gains
   disp( ' ' )
   disp( 'Calculated LQR+I controller gain
elements: ' )
   disp( [ 'Ki = [' num2str( Ki(1,1),3 ) ' V/rad  '
num2str( Ki(1,2),3 ) ' V/rad  ' num2str( Ki(1,3),3 )
' V.s/rad  '  num2str( Ki(1,4),3 ) ' V.s/rad '
num2str( Ki(1,5),3 ) ' V/(rad.s) ' num2str(
Ki(1,6),3 ) ' V/(rad.s)]'] )
   disp( [ '    [' num2str( Ki(2,1),3 ) ' V/rad  '
num2str( Ki(2,2),3 ) ' V/rad  ' num2str( Ki(2,3),3 )
' V.s/rad  '  num2str( Ki(2,4),3 ) ' V.s/rad '
num2str( Ki(2,5),3 ) ' V/(rad.s) ' num2str(
Ki(2,6),3 ) ' V/(rad.s)]' ] )
elseif strcmp ( CONTROLLER_TYPE,
'MANUAL' )
   disp( [ 'K = [' 0 ' V/rad  '  0 ' V/rad  ' 0 ' V.s/rad
'  0 ' V.s/rad]'] )
   disp( ' ' )
   disp( 'STATUS: manual mode' )
   disp( 'The model parameters of your Single
Pendulum and IP01 or IP02 system have been
set.' )
   disp( 'You can now design your state-
feedback position controller.' )
   disp( ' ' )
else
   error( 'Error: Please set the type of controller
that you wish to implement.' )
end
function u = uso(n)
global sensor_code sensor_init_period
extra_task
if n>1 && n<6
   s=0;
   u=sensor_code(n)/sensor_init_period(n);
   n=n-1;
   for i=1:4
      s= s + extra_task(n,i,2)/extra_task(n,i,1);
   end
   u=u+s;
end
```

8.3 Publicaciones

"Distributed Real-Time Systems representation using Process Algebra to define Composition Scheduling Needs"
Work-In-Progress Session of the 19th Euromicro Conference on Real-Time Systems (ECRTS-2007); Pisa, Italy 2007;4-6 July 2007; IEEE Computer Society.

"Node Availability dor Distributed Systems considering processor and RAM utilization"
Eighth Mexican International Conference on Computer Science (ENC07); Morelia, Michioacán, México; 24-28 September 2007;IEEE Computer Society.

"Hierarchical Scheduling for Real-Time Distributed Systems Integrating a validation resource an a performance metric"
The 2008 International Conference on Convergence an hybrid Information Technology (ICCIT 2008); Busan, Korea; 11-13 November 2008; IEEE Computer Society.

"Node Availability for Distributed Systems considering processor and RAM utilization"
Aceptado para su publicación por: International Journal of Computers Communications & Control; 2009.

9 Referencias

Agrawal M., Cofer D., Samad T.; “Real-Time Adaptative Resource Management for Advanced Avionics”; IEEE Control Systems Magazine páf. 76 a 86, USA, Febrero 2003.

Almeida L., Pedreiras P., “Scheduling within Temporal Partitions: Response-time Analysis and Server Design”, EMSOFT’04, September 27--29, ACM 1-58113-860-1/04/0009, 2004.

Alur Rajeev, Henzinger Thomas; “Back to the Future: Towards a Theory of Timed Regular Languages”; Bell Laboratories NJ USA; Cornell University NY USA; 1992.

Baeten J., Weijland W.; “Process Algebra”; Cambridge University Press; Great Britain; 1990.

Baeten J., Middelburg C.; “Process Algebra with Timing”; Springer; Germany; 2002.

Benítez Pérez H., García-Nocetti F. ; “Reconfigurable Distributed Control”; Springer, USA, 2005.

Bertsekas D. ; “Constrained Optimization an Lagrange Multiplie Methods” ; Academic Press Inc.; USA 1992

Butazzo G., “Hard Real-Time Computing Systems”; Kluwer academic publishers, USA, 2004.

Cervin A., Henriksson D., Lincoln B., Eker J and Årzén K.E.; “How Does Control Timing Affect Performance?”; IEEE Control Systems Magazine, 23:3, pp. 16–30; June 2003.

Cervin A., Henriksson D., Ohlin M.;“True-Time 1.5 Reference Manual”; Dep. Of Atomatic Control, Lund University; Sweden; 2007.

Chakraborty S., Priya S.; “A Framework for Compositional and Hierarchical Real-Time Scheduling”; Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06); 2006.

Chiasson J., Tang Z., Ghanem J., Chaouki T., Abdallah, J., Birdwell D., Hayat M.M., Jérez H.; “The Effect of Time Delays on the Stability of Load Balancing Algorithms for Parallel Computations”; IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, VOL. 13, NO. 6, pp. 932-942; NOVEMBER 2005.

Cheng Albert M; “Real-Time Systems”; Wiley-Interscience; New Jersey, USA; 2002.

Colouris G., “DISTRIBUTED SYSTEMS: CONCEPTS AND DESIGN”,Addison-Wesley, 1999.

Deng Z. and Liu J. W.-S.; “Scheduling real-time applications in an open environment”; In Proc. of IEEE, Real-Time Systems Symposium, pages 308–319; December 1997.

Easwaran A., Shin I., Sokolsky O., Lee I.; “Incremental Schedulability Analysis of Hierarchical Real-Time Components”; Proceedings of the 6th Annual ACM Conference on Embedded Software (EMSOFT 2006), pages 272-281; October 2006

Feng-Li Lian, James Moyne, and Dawn Tilbury; “Network Design Considerations for Distributed Control Systems”; IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, VOL. 10, NO. 2, págs. 297 a 307,USA; Marzo 2002.

Feng M., Mok A.; “A model of hierarchical realtime virtual resources”; Proc. of IEEE Real-TimeSystems Symposium, pages 26–35; December 2002.

Fernandes de Mello R., Senger L. J., Yang L.T. ; “A Routing Load Balancing Policy for Grid Computing Environments”; Proceedings of the 20th International Conference on Advanced Information Networking and Applications, 1550-445X/06 IEEE 2006.

Fokkink Wan; “Introduction to Process Algebra”; Springer; Germany; 2000.

García Zavala Angel; “Propuesta de un método de planificación para la reconfiguración en línea en un sistema de Tiempo-Real”; Universidad Nacional Autónoma de México, México, 2003.

Grosu D., Chronopoulos A.; “Algorithmic Mechanism Design for Load Balancing in Distributed Systems” ; IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 34, NO. 1, pp. 77-84; FEBRUARY 2004.

Jensen E.D.; “Asynchronous decentralized real-time computer systems”; Springer, Berlin, 1992.

Lee O., Anshel M., Chung I.; “Design of an efficient load balancing algorithm on distributed networks by employing symmetric balanced incomplete block design” IEE Proc.-Commun., Vol. 151, No. 6, December 2004.

Liu Jane W. S.; “Real-Time Systems”; Prentice-Hall; New Jersey, USA; 2000.

Menéndez L. de C. Antonio, Benítez-Pérez Héctor; “Distributed Real-Time Systems representation using Process Algebra to define Composition Scheduling Needs”; Euromicro 2007, IEEE; Pisa Italy; 2007.

Menéndez L. De C. Antonio, Benitez-Perez Héctor, Palomera-Perez M.A; "Node Availability for Distributed Systems considering Process Communications"; Encuentro Nacional de Computación, ENC 2007, IEEE; Morelia Mich. México; 2007.

Mok A., Wang W. ; "On the Composition of Real-Time Schedulers"; RTCSA 2003, LNCS 2968, pp. 18–37, Springer-Verlag Berlin Heidelberg; 2004.

Northcutt, J. Duane; "Mechanisms for Reliable Distributed Real-Time Operating Systems"; Academic Press Inc.; Orlando Fla., USA; 1987.

"Quanser 2 DOD Helicopter User and Control Manual"; Quanser Inc. rev 1.0; February 10, 2006.

Ravindran B., Li P., Hegazy T.; "Proactive resource allocation for asynchronous real-time distributed systems in the presence of processor failures"; Elsevier Inc, USA, 2003.

Regehr J., Stankovic J.; "HLS: A framework for composing soft real-time schedulers"; Proc. of IEEE Real-Time Systems Symposium, pages 3–14; December 2001

Sit H., Ho K., Va Leong H, Robert, Luk W. P., Ho L.; "An Adaptive Clustering Approach to Dynamic Load Balancing"; Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04) 1087-4089 IEEE 2004.

Shin I., Lee I.; "Periodic Resource Model for Compositional Real-Time Guarantees"; Proc. of the 24th IEEE International Real-Time Systems Symposium (RTSS'03), 0-7695-2044-8/03. USA 2003.

Tanenbaum A.; "SISTEMAS OPERATIVOS DISTRIBUIDOS"; Prentice Hall; México; 1996.

Tanenbaum A.; "Computer Networks"; Prentice Hall; USA; 2003.

Takemoto D., Tagashira S., Fujita S.; "Partitioning in Content-Addressable Networks Distributed Algorithms for Balanced Zone"; Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04) 1521-9097 IEEE 2004.

Torque Resource Manager http://www.clusterresources.com /pages /products /torque-resource-manager.php 2006.

Wang Yingxu; "RTPA: A NEW APPROACH TO REAL-TIME SYSTEM SPECIFICATION"; Proceedings of the 2002 IEEE Canadian Conferences on Electrical Computer Engineering, 0-7803-7514-9/02, IEEE; 2002.

Yao Yizheng and Wang Yingxu; “A New Approach to Test Case Generation based on Real-Time Process Algebra (RTPA)”; Dept. of Electrical & Computer Engineering University of Calgary Alberta, Canada; T2N 1N4 CCECE 2004- CCGEI 2004, Niagara Falls; May/mai 2004 0-7803-8253-6/04; 2004.

Zeng Z., Veeravalli B.; “Rate-Based and Queue-Based Dynamic Load Balancing Algorithms in Distributed Systems”; Proceedings of the Tenth International Conference on Parallel and Distributed Systems, 1521-9097/04 IEEE 2004.