Análisis de Sistemas de Tiempo Real: k-diagramabilidad y tolerancia a las fallas

Rodrigo M. Santos
Universidad Nacional del Sur, Dep. Ingeniería Eléctrica y Computadoras
Bahía Blanca, Argentina, 8000
ierms@criba.edu.ar

and

Jorge Santos
Universidad Nacional del Sur, Dep. Ingeniería Eléctrica y Computadoras
Bahía Blanca, Argentina, 8000
iesantos@criba.edu.ar

and

Javier D. Orozco
Universidad Nacional del Sur, Dep. Ingeniería Eléctrica y Computadoras
Bahía Blanca, Argentina, 8000
ieorozco@criba.edu.ar

Abstract

This paper presents a method to deal with the re-execution of tasks in a hard real-time system of *n* tasks subject to transient faults. It is based on the concepts of singularities and *k*-schedulability and it can determine all the possible combinations of faults that the system can admit while meeting the time constraints. The combinations are contained in a surface generated in an *n*-dimensional space. Extensive simulations carried out for different mean times between failures allow the evaluation of the proposed method. The metric used is the percentage of faulty tasks not recovered on time, for failure combinations above the surface, computed as a function of the utilization factor for randomly generated sets of tasks.

Keywords: real-time systems, fault-tolerance, schedulability.

Resumen

En el trabajo se presenta un método para la re-ejecución de tareas de un sistema de tiempo real duro de *n* tareas, sujetas a fallas transitorias de cálculo. El mismo se basa en las nociones de *k*–diagramabilidad y singularidad y permite determinar todas las combinaciones de fallas que el sistema tolera cumpliendo con todas las constricciones temporales del sistema de tiempo real duro. En un espacio *n*-dimensional, se genera una superficie en la que se encuentran los puntos que definen la tolerancia a las fallas del sistema. Mediante simulaciones con conjuntos de tareas generados aleatoriamente, se determina, para los puntos situados por encima de esa superficie, el porcentaje de tareas no recuperables en función del factor de utilización del sistema para distintos tiempos medios entre fallas.

Palabras claves*:* sistemas de tiempo real, tolerancia a las fallas.

1. Introducción

Los sistemas de tiempo real (STR) son aquellos en los cuales los resultados no sólo deben ser correctos desde un punto de vista aritmético-lógico sino que además deben ser producidos antes de un determinado instante denominado *vencimiento*. Cuando no puede perderse ningún vencimiento el sistema se denomina *duro*. Estos son, por ejemplo, los sistemas de control de centrales nucleares y los de control de vuelo de aviones y naves espaciales, en los que no cumplir con la constricción de tiempo puede acarrear consecuencias catastróficas, incluida la pérdida de vidas humanas. Un sistema que cumple todas las constricciones de tiempo se denomina *diagramable*.

Como en toda obra de ingeniería, en los STR pueden ocurrir fallas de funcionamiento. Estas pueden ser de tres clases: permanentes, intermitentes y transitorias. Las permanentes se producen, en general, por fallas irrecuperables en el "hardware". Las transitorias son malfuncionamientos temporarios que producen resultados incorrectos. Las intermitentes son transitorias repetidas [2].

En general, la recuperación del sistema luego de una falla se efectúa mediante el uso de algún tipo de redundancia. Esta puede ser *espacial* (utilización de recursos de respaldo, por ejemplo procesadores) o *temporal* (ejecución de un procedimiento de recuperación, por repetición del cálculo, luego de la detección de una falla) [6].

En las aplicaciones más usuales, los STR son del tipo multitarea-monoprocesador. Son además determinísticos y las tareas son independientes (el comienzo de la ejecución de una no depende de la finalización de la ejecución de otra), apropiables (su ejecución puede ser interrumpida por una tarea de mayor prioridad) y periódicas (reclaman ejecución a intervalos regulares de tiempo). El mínimo común múltiplo de todos los períodos se denomina hiperperíodo y dado que cada tarea tiene además un tiempo definido de ejecución, es posible determinar exactamente el tiempo necesario para ejecutar todas ellas en el hiperperíodo. El tiempo sobrante es la redundancia temporal que se puede utilizar en la recuperación de fallas. Dado que al cabo de cada hiperperíodo se repiten las condiciones iniciales, basta estudiar el sistema en ese lapso.

Todo método que utilice la redundancia temporal para hacer el sistema tolerante a las fallas, se reduce, en esencia, a redistribuir el tiempo redundante de modo tal que, ante la detección de una falla en la ejecución de una tarea, se pueda repetir la misma antes del vencimiento asegurando además que todas las otras tareas siguen cumpliendo sus constricciones de tiempo [1, 2, 5, 11, 12].

En el presente trabajo se expone un método de recuperación de fallas basado en el concepto de *k*-diagramabilidad para el caso de sistemas multitareas-monoprocesador. Las conclusiones, sin embargo, son inmediatamente extensibles para el caso multitarea-multiprocesador en el que la asignación de tareas a procesadores es estática [13]. El resto del trabajo se organiza de la siguiente manera: En la Sección 2 se presentan las nociones de *k* - PMC diagramabilidad y singularidad; en la sección 3 se aplican esos conceptos para determinar la tolerancia a las fallas de un sistema dado; en la Sección 4 se presentan los resultados de simulaciones y, finalmente, en la Sección 5 se extraen conclusiones.

2. Trabajos Previos

Existe una extensa lista de trabajos relacionados con la tolerancia a las fallas. En el caso de los sistemas de computación, las acciones tendientes a disminuir las consecuencias de la ocurrencia de fallas, se pueden lograr por medio de procesadores de reemplazo por medio de re-ejecución de las tareas con errores o por combinaciones de ambas. En el primer caso el costo inicial y operativo del equipo aumenta pues se duplican o triplican los procesadores con el consiguiente aumento de peso, energía necesaria para el funcionamiento del sistema redundante, espacio físico de alojamiento, etc. Esta opción no es por lo tanto la mejor en el caso de aplicaciones tales como móviles en general y las espaciales (satélites) en particular. En éstas, todas las restricciones mencionadas deben ser consideradas y minimizada su presencia en el diseño final. La redundancia temporal, utilizada para re-ejecutar las tareas en falla, resulta más económica en estos aspectos pero exige el sobredimensionamiento del procesador para que pueda manejar los diferentes requerimientos temporales.

En [7] Liu y Layland demuestran que la condición suficiente de diagramabilidad de un sistema de tiempo real monoprocesador con n tareas periódicas, apropiables y atendidas por PMC es que el factor de utilización total sea menor que $n(2^{(1/n)} - 1)$. Si bien la condición es sólo suficiente, establece una cota de diagramabilidad que permite realizar una clasificación rápida de los sistemas ya que si poseen una utilización menor a la cota son diagramables. Una primera aproximación a la tolerancia a las fallas sería duplicar el tiempo de ejecución de todas las tareas para considerar así, al menos una re-ejecución de cada una de ellas. Si con esta consideración, el factor de utilización se mantiene por debajo de la cota de Liu y Layland, el sistema podría recuperarse frente a una falla de cada tarea en cada instanciación de la misma. Sin embargo este enfoque resulta demasiado simplista y costoso desde el punto de vista del poder de computo necesario.

En [11], se analiza la diagramabilidad de un conjunto de tareas periódicas que son diagramadas por PMC y que toleran una única falla. Cuando esta se produce todas las tareas que hasta ese momento no terminaron son re-

ejecutadas. Los autores prueban que ninguna tarea pierde su vencimiento cuando el factor de utilización del sistema no supera el 50%. El tipo de fallas que toleran son del tipo intermitentes o transitorias.

En [12] se presenta un esquema en el cual hay una asignación estática y otra dinámica para proveer tolerancia a las fallas. El algoritmo extiende el análisis de diagramabilidad realizado por Lehoczky, Sha y Ding [4] para incorporar el tiempo necesario para la re-ejecución de las tareas.

En [1] se proveen distintas pruebas de diagramabilidad exactos para conjuntos de tareas tolerantes a las fallas basados en el análisis de las respuestas temporales. En ninguno de los casos propuestos se brindan cotas para los factores de utilización de los conjuntos de tareas.

El problema de la tolerancia a las fallas también ha sido abordado a partir del uso de servidores, ideados en principio para la atención de tareas aperiódicas de baja prioridad [14, 15]. En este caso consideran la re-ejecución de tareas como tareas aperiódicas. La solución propuesta presenta la dificultad de que los servidores buscan mejorar el tiempo de respuesta de las tareas redundantes sin considerar que verifiquen sus vencimientos.

Gosh [2] propone un esquema de recuperación de fallas basado en el tiempo de *slack* disponible en el sistema. La atención de las tareas sigue un ordenamiento PMC excepto en el momento en que se produce una falla. Establece cotas en los factores de utilización a partir del ancho de banda disponible para la recuperación de las tareas de fallas simples o múltiples sujetas a restricciones en la tasa de ocurrencia de las fallas.

Liberato [6] propone un método de diagramación de tareas tolerantes a las fallas en sistemas multiprocesador. Considera para ello que las tareas pueden migrar entre los procesadores. Define dos tipos de diagramación que le sirven para distinguir a las tareas en recuperación de las otras.

El algoritmo propuesto en este trabajo difiere de los anteriores en cuanto utiliza un algoritmo simple basado en la k-diagramabilidad [9]. La sencillez del método permite de manera simple establecer no sólo cotas en el factor de utilización para la recuperación de tareas sino, además caracterizar a las tareas en críticas y no críticas con la recuperación total de las primeras en todos los casos siempre y cuando se verifiquen determinadas condiciones enunciadas.

3. Sistemas *k*-PMC Diagramables

Cuando dos o más tareas requieren simultáneamente el uso del procesador, es necesario definir una disciplina de prioridades para dirimir el conflicto. En [7] se demostró que la disciplina de Períodos Monotónicos Crecientes, PMC, es óptima entre las de prioridades fijas. En la actualidad es una norma *de facto* [8] y es la disciplina adoptada en este trabajo.

Varios métodos han sido propuestos para determinar las condiciones de diagramabilidad de los STR operando bajo PMC [3, 7, 14] Todos ellos, tienen en común el hecho de que la relación de orden de prioridades está basada en periodos crecientes, con alguna regla adicional para quebrar los empates. En el Método de las Ranuras Vacías [14], el tiempo se considera ranurado y la duración de una ranura es tomada como la unidad de tiempo. Las ranuras se notan t y se numeran 1, 2, ... La expresión *al comienzo de la ranura t* e *instante t* son equivalentes. Una tarea puede ser desalojada del procesador por otra de mayor prioridad únicamente al comienzo de la ranura.

Un conjunto $\mathbf{S}(n)$ de n tareas independientes periódicas apropiables se encuentra completamente especificado por $\mathbf{S}(n)=\{(C_1, T_1, D_1), (C_2, T_2, D_2),..., (C_n, T_n, D_n)\}$, donde C_i, T_i y D_i, indican el máximo tiempo de ejecución, el período y el vencimiento de la tarea i respectivamente. En [7] se demostró que para la disciplina PMC, el peor estado de carga se produce cuando todas las tareas se generan simultáneamente en t=1. A continuación, la tarea i se generará nuevamente en las ranuras T_i, $2T_i$, etc. y tendrá vencimientos en las ranuras D_i, kT_i+D_i, etc. Cada nueva generación se denomina instanciación de la tarea. La ranura en que vence la instanciación j de la tarea i, será notada d_{ij}.

En general, en los sistemas del mundo real $D_i \leq T_i$ y en lo que sigue se acepta como hipótesis que $D_i = T_i$. En [14] se demostró formalmente que $\mathbf{S}(n)$ es diagramable por PMC sss:

$$\forall i \in (1,2,....,n) \qquad T_i \geq D_i \geq \text{menor } t \mid t = C_i + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil$$

En consecuencia, el sistema es diagramable por PMC, si antes de su vencimiento, cada tarea encuentra suficientes ranuras libres para ejecutarse y dar lugar a las tareas de mayor prioridad. Las ranuras quedan vacías únicamente si no hay tareas con ejecución pendiente.

Un sistema S(n) se define como *k*-PMC diagramable cuando

$$\forall i \in (1,2,....,n) \qquad T_i \geq D_i \geq \text{menor } t \mid t = C_i + k + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil$$

k es el mínimo valor del conjunto $\{k_i\}$. k_i se define como:

$$k_i = \max k \mid T_i \geq D_i \geq \text{menor } t \mid t = C_i + k + \sum_{h=1}^{i-1} C_h \left\lceil \frac{t}{T_h} \right\rceil$$

En la expresión anterior, el último término del miembro de la derecha se denomina *función trabajo* y se nota $W_{i-1}(t)$. Si M denota el Mínimo Común Múltiplo de los períodos de las n tareas, a menudo llamado *hiperperíodo*, la expresión

$$W_n(M) = \sum_{h=1}^{n} C_h \left\lceil \frac{t}{T_h} \right\rceil$$

da el número de ranuras necesario para procesar todas las tareas que pertenecen a **S**(n) en el intervalo [1, M]. Evidentemente, el número de ranuras vacías en el hiperperíodo será $M - W_n(M)$. Cuando esas ranuras aparecen naturalmente, luego de haber ejecutado todas las tareas pendientes hasta ese momento, se denominan ranuras *"background"*.

3.1 Detección de Singularidades

Una singularidad s, se define como una ranura en la cual todas las tareas pertenecientes al sistema **S**(n) que fueron activadas en el intervalo [1, (s-1)], han sido ejecutadas. Nótese que s-1 puede ser tanto una ranura vacía como una ranura en la cual se terminó de ejecutar la última de las tareas periódicas pendientes. s es una singularidad aún en el caso de que en t=s, se generen nuevas instancias de las tareas periódicas. Una singularidad s_i es una ranura en la cual todas las tareas del subsistema **S**(i), con activación en el intervalo [1, (s_i-1)], han sido ejecutadas.

En [9] se demostró formalmente que si un STR es *k*-PMC diagramable, k ranuras de un intervalo [s, (s+k-1)] pueden ser usadas para ejecutar tareas ajenas al sistema. En el caso de sistemas tolerantes a las fallas, basta interpretar que dichas tareas son aquéllas cuya ejecución falló. El método puede refinarse generando los valores en cada s_i y se implementa mediante n contadores. El contenido de cada uno de ellos, asociado a una tarea, es notado AC_i. El método se denomina de detección de singularidades múltiples y sus pasos son:

1) $\forall g \in \{1,2,...,i\}$, $AC_g = k_g$ en $t = s_i$.

2) *Si* $\forall\, i \in \{1, 2, ..., n\}$ $AC_i \neq 0$ y es necesario repetir la ejecución de una tarea *entonces* se repite la ejecución *si no* se sigue el ordenamiento PMC.

3) $\forall i \in \{1, 2, ..., n\}$ AC_i se decrementa en una unidad por cada ranura asignada a la repetición de una ejecución.

4. *k* – Diagramabilidad y Tolerancia a las Fallas

Dado un sistema **S**(n) es posible determinar su tolerancia a las fallas por aplicación del método de recuperación por *k*-diagramabilidad y singularidades. En el peor caso, la singularidad s se presentará sólo una vez y lo hará cuando termine de ejecutarse la tarea de máximo período, que, por el ordenamiento PMC, será T_n. En consecuencia, el factor de utilización disponible para la recuperación será U_R=k/T_n. Contrariamente a lo que intuitivamente podría pensarse, con un dado U_R pueden recuperarse instancias de ejecución de tareas cuya suma de factores de utilización superan U_R.

En efecto, la expresión

$$\lceil T_n / T_i \rceil$$

indica el número de instanciaciones de t_i en el intervalo [1, T_n]. En consecuencia, la expresión

$$\lfloor k / \lceil T_n / T_i \rceil \rfloor = R_i$$

da el número de ranuras disponibles para recuperar t_i en cada instanciación. Si $R_i \geq C_i$, t_i podrá fallar una vez cada vez que se genere y ser recuperada. Si $R_i < C_i$, el tiempo de recuperación utilizado en los cálculos, notado C_i^r, será igual a R_i. Por lo tanto, $C_i^r = \min [R_i, C_i]$ y la recuperación podrá hacerse en $p_i = \lfloor \lceil T_n / T_i \rceil / \lfloor C_i / C_i^r \rfloor \rfloor$ instanciaciones, sujeto a la condición $R_i p_i \geq C_i$. La condición establece que al menos se puede recuperar una instancia.

La fallas que el sistema puede tolerar serán en consecuencia

$$\Sigma_i^n C_i^r q_i \leq k \quad \text{sujeto a } 0 \leq q_i \leq \min \{p_i, \lceil T_n / T_i \rceil\} \qquad (1)$$

en la que q_i denota el número de instanciaciones en que t_i puede fallar, y el sistema tolerarlo, en el intervalo $[1, T_n]$.

Ejemplo: Sea el sistema **S**(5) especificado en la Tabla 1.

Tabla 1

	T	***C***	***k***
1	6	1	5
2	10	2	8
3	15	1	7
4	15	2	5
5	15	1	4

Dado que $k = 4$ y $T_5 = 15$, los C_i^r y los p_i calculados aplicando las expresiones anteriores son los presentados en la Tabla 2.

Tabla 2

i	C_i^r	p_i
1	1	4
2	2	2
3	4	1
4	4	1
5	4	1

En consecuencia, el sistema puede tolerar que fallen tareas en cualquier combinación que satisfaga la expresión

$$q_1 + 2\,q_2 + q_3 + 2\,q_4 + q_5 \leq 4$$

sujeta a la condición

$$0 \leq q_i \leq \lceil T_n / T_i \rceil$$

Por lo tanto, en el intervalo $[1, T_n]$, t_1 y t_3 o t_1 y t_5 pueden fallar en todas sus instanciaciones; t_2 puede fallar en sus dos instanciaciones; t_3, t_4 y t_5 pueden fallar en todas sus instanciaciones; t_1 puede fallar en una instanciación junto con t_2 y t_3 en todas sus instanciaciones, etc. En la Fig. 1 se muestra la evolución del sistema en este último caso; la tarea tildada indica falla y con una *r* se nota su recuperación.

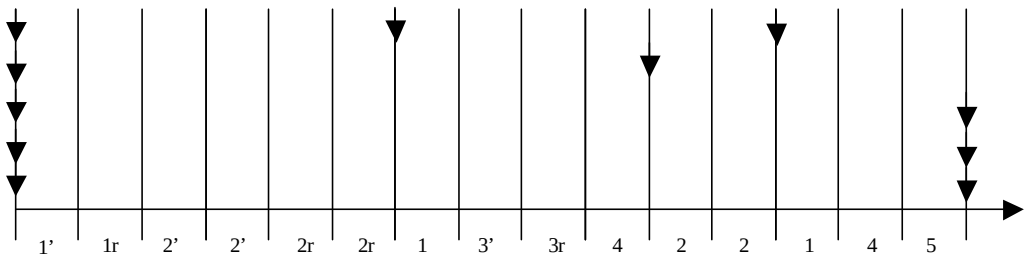

Fig. 1

Debe notarse que, en el caso general de *n* tareas, la expresión anterior genera, en el espacio *n*-dimensional, una superficie en la que y por debajo de la cual, se encuentran los puntos que hacen el sistema tolerante a las fallas.

Aparte es posible recorrer el espacio de soluciones y determinar con exactitud el grado de tolerancia a las fallas que tiene el sistema aplicando el algoritmo de recuperación propuesto.

En caso de que no todas las tareas de un sistema tengan la misma criticidad, es posible reducir las dimensiones de la inecuación anterior de modo que sólo aquéllas que de no recuperarse provocarían la salida de funcionamiento del sistema sean consideradas. Para el ejemplo anterior, la Fig. 2 muestra el caso en el que solo t_1 y t_2 resultan críticas.

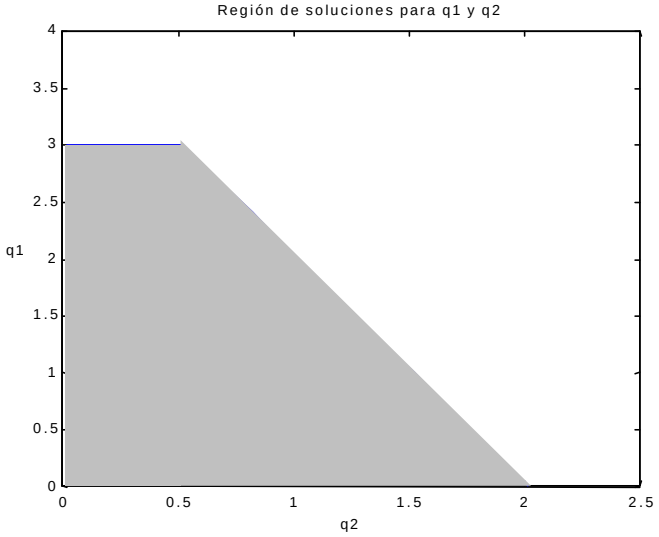

Fig.2. Superficie de soluciones del sistema cuando se consideran t_1 y t_2 únicamente

Todos los puntos del plano que se encuentren dentro o en el límite del polígono son soluciones de la inecuación y por lo tanto brindan tolerancia a las fallas de esas dos tareas en el sistema.

En [2] se enuncian las condiciones que debe cumplir un sistema tolerante a las fallas:

1) Para que una tarea t_i en falla en su instanciación j pueda ser recuperada, por lo menos debe haber C_i ranuras disponibles entre los instantes jT_i y $(j+1)\ T_i$.

2) Si una tarea t_i se ejecuta con fallas, el esquema de recuperación debe ser tal que permita su re-ejecución durante C_i ranuras antes de su vencimiento.

3) La re-ejecución de una tarea no debe ocasionar la pérdida del vencimiento de ninguna otra tarea.

El método propuesto cumple las tres condiciones:

1) El cálculo descripto previamente permite determinar con exactitud la tolerancia a las fallas del sistema, entendida como cuántas instanciaciones de qué tareas pueden ser re-ejecutadas.

2 y 3) Que el sistema, es k-PMC diagramable significa, en esencia, que las k ranuras que corresponden al tiempo redundante redistribuido, pueden ser vistas conceptualmente como ranuras en las que es factible ejecutar tareas que producen inversiones de prioridad [13] sin que el sistema pierda su diagramabilidad. Las tareas de ejecución en falla pueden considerarse como tareas ajenas al sistema, que produjeron inversiones de prioridad. Como el sistema admite esas inversiones, se cumplen las condiciones 2 y 3.

La cota inferior de recuperación determinada en la expresión (1) es pesimista. Ello se debe a que el número total de ranuras que deja vacío la ejecución del sistema **S**(n), disponibles, en consecuencia, para la recuperación de tareas en el hiperperíodo es, según se explicó en la Sección 3, $M - W_n\ (M)$. Dicho número es función únicamente del número de tareas y sus períodos y no hay algoritmo que pueda incrementarlo. Sin embargo, su utilización depende del momento en que las ranuras vacías están disponibles. Lo que hacen los métodos de k – diagramabilidad y singularidades es adelantar su aparición y tenerlas disponibles para ser usadas si ello es conveniente. No todas las ranuras vacías son redistribuidas y algunas de ellas seguirán apareciendo como ranuras *"background"*. Cuantas más ranuras adelantadas haya, mayor serán las posibilidades de recuperación del sistema en falla. En el método de detección simple de singularidades, el número de ranuras adelantadas por redistribución, es k, que es el valor usado en el cálculo de la cota. Sin embargo, según se mostró en [10], al usar el método de detección múltiple, aplicado en este caso, el número de ranuras vacías redistribuídas aumenta y aparecen más temprano, con lo cual mejoran las posibilidades de recuperación de fallas. Estos dos hechos combinados: mayor número de ranuras redistribuidas y aparición más temprana de las mismas, hace que la cota mínima pueda ser superada y el comportamiento de

recuperación del sistema sea bastante mejor que el que podría esperarse de la simple aplicación de la cota. Esto se pone de manifiesto experimentalmente en los resultados de las simulaciones de la Sección siguiente.

5. Simulaciones y Resultados

Las simulaciones se realizaron generando en forma aleatoria 10000 conjuntos de 10 tareas con periodos múltiplos de 10, distribuidos de manera uniforme en el intervalo [10, 100]. Los tiempos de ejecución también se calculan aleatoriamente de modo que el factor de utilización total del sistema tenga una distribución uniforme comprendida en el intervalo [0.3, 0.9]. Las fallas siguen una distribución estadística exponencial, por ser el modelo corriente de fallas utilizado en la evaluación de sistemas. El tiempo medio entre fallas del sistema (MTBF) se fija, de acuerdo a lo realizado por Gosh [2], en cuatro valores que son a) el promedio de los períodos del sistema; b) el periodo máximo; c) cinco veces el periodo máximo y d) 10 veces el período máximo. Los sistemas son simulados durante 100000 unidades de tiempo. Los resultados obtenidos en los cuatro casos se muestran en las Fig. 3 en la que se representan los porcentajes de tareas en falla recuperadas en función del factor de utilización total del sistema.

Los resultados obtenidos muestran un comportamiento inicial del método de recuperación de fallas con tolerancia del 100% cuando el FU del sistema es de hasta un 50%. Este hecho no es sorprendente ya que el procesador permanece ocioso más de la mitad del tiempo y por lo tanto puede recuperar cualquier tarea que falle. A partir de un factor de 0.6 en todos los casos, comienza a degradarse la tolerancia. Para un tiempo medio entre fallas de 50 ranuras, alcanza el 78% de éxito para un factor de utilización de 0.9. Cuando el tiempo medio entre fallas aumenta a 1000 ranuras, el éxito crece a 86% para la misma situación. La mejora en la tolerancia a medida que la frecuencia en las fallas decrece radica en que cuando las tareas fallan tienen disponibles ranuras que en los casos de fallas más frecuentes son utilizadas por tareas anteriores.

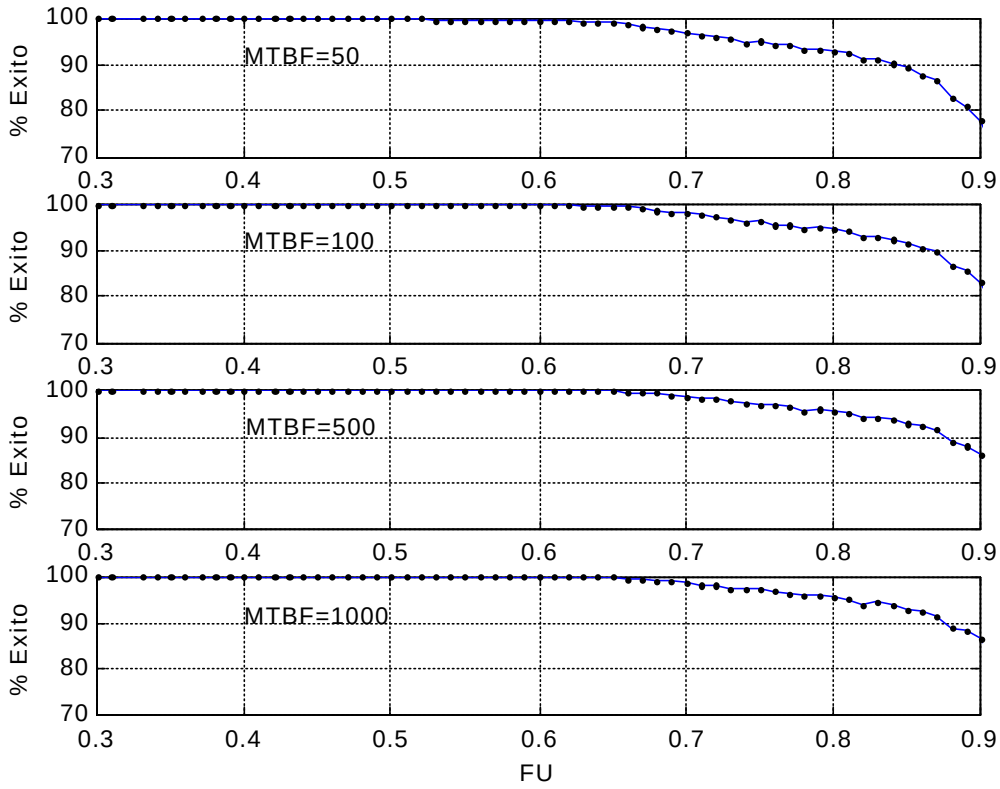

Fig. 3 Porcentaje de tareas en falla recuperadas en función del factor de utilización para distintos MTBF.

6. Conclusiones

Se presentó un método para la recuperación de tareas en falla en sistemas de tiempo real duro. El método propuesto redistribuye el tiempo ocioso del procesador de modo que pueda ser utilizado para re-ejecutar aquellas tareas que hayan tenido errores. Por tratarse de tiempo real duro, se tomaron las peores condiciones de carga para el procesador y se consideró que la detección de la falla de la tarea sólo se producía al término de la ejecución de la misma (caso más desfavorable pues se consumieron las c_i ranuras de la ejecución en falla). A pesar de estas condiciones restrictivas para el sistema, el algoritmo desarrollado posee una elevada relación de éxito cuando el factor de

utilización se encuentra por debajo de la cota de Liu Layland. A medida que aquél aumenta se produce una degradación en el rendimiento, previsible de antemano.

Los sistemas 100% seguros no existen pues en el peor caso todos los algoritmos y los procesadores de respaldo podrían fallar. Para tareas que requieran por su importancia una confiabilidad muy alta, seguramente se destinará un procesador especial para su atención con la suficiente redundancia de "hardware". Sin embargo, en la mayor parte de los casos, con brindar una confiabilidad media basada en la redundancia temporal de las tareas puede ser suficiente. El método propuesto aquí se orienta en este sentido.

El método sólo necesita para su implementación la incorporación de un contador por tarea para llevar cuenta de la cantidad de ranuras ociosas utilizadas para la recuperación de fallas. Estos contadores pueden ser actualizados al término de la ejecución de las tareas en forma muy rápida por el despachador del sistema operativo. El costo del cálculo de los mismos previo al inicio del servicio se puede realizar en tiempo polinomial lo que convierte al método en realizable para sistemas en los cuales haya variaciones en la cantidad de tareas que conforman el sistema en tiempo de corrida o modificaciones en los tiempos de ejecución o periodos de generación.

Tan importante como lo anterior es que el algoritmo propuesto puede determinar la tolerancia a las fallas de un sistema cualquiera sin introducir modificaciones severas. Es decir que con él se puede responder a la pregunta, ¿cuán tolerante a las fallas es este sistema? La diferencia de aproximación con los otros métodos no es trivial ya que en este caso dado el sistema, definido a partir de los costos de producción o disponibilidad de elementos del mercado, se puede brindar una garantía sobre el nivel de tolerancia a las fallas, y esto de hecho brinda una métrica sobre la calidad de servicio que ofrece el sistema.

Al mismo tiempo, con pocas modificaciones, el método permite distinguir entre tareas críticas y no críticas. Reduciendo el universo de soluciones, se quitan grados de libertad en la inecuación, la identificación de las superficies de soluciones se puede realizar rápidamente y así brindar respuestas absolutas sobre la tolerancia a las fallas del sistema, independiente de la tasa de error que la tarea pudiera tener.

Como trabajo futuro queda realizar un análisis más profundo del comportamiento del sistema con detección múltiple de singularidades. Otra línea de investigación puede ser el estudio de la partición de las tareas en clases que compartan un contador común. Finalmente si cada tarea tiene asociado un valor de criticidad con mayores recompensas asociadas a mayores criticidades, es posible que mediante técnicas de programación lineal, se pueda determinar la configuración de recuperaciones que maximiza la recompensa.

Referencias

[1] Burns, A., R. Davis, S. Punnekkat, "Feasibility Analysis of Fault-Tolerant Real-Time Tasks Sets", *Proc. 8th Euromicro Workshop on Real-Time Systems*, 29-33, 1996.

[2] Gosh, S.,R. Melhem, D. Mossé, J. Sen Sarma, "Fault-Tolerant Rate-Monotonic Scheduling", *Real Time Systems Journal*, 15, 149-181, 1998.

[3] Joseph, M., P. Pandya, "Finding response Times in a Real-Time System", *the Computer Journal*, 29 (5), 390-395, 1986.

[4] Lehozcky, J., L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour", *Proc. Real Time Systems Symposium*, 166-171, 1989.

[5] Lehozcky, J., L. Sha, J. Strosnider, "Enhanced Aperiodic Responsivness in Hard-Real-Time Environments", *Proc. Real Time Systems Symposium*, 261-270, 1987.

[6] Liberato, F., S. Lauzac, R. Melhem, D. Mossé, "Fault-Tolerant Real-Time Global Scheduling on Multiprocessors", *proc. 11th Euromicro Conference on real-Time Systems*, 252-259, York, 1999.

[7] Liu, C. L., J. Layland, "Scheduling Algorithm for Multiprograming in a Hard Real-Time Environments", *Journal of the ACM*, 20 (1), 46-61, 1973.

[8] Obenza, J., "Rate Monotonic Analysis for Real-Time Systems", *IEEE Computer*, (26) 3, 73-74, 1993.

[9] Orozco J., R. Santos, J. Santos and R. Cayssials, "Taking advantage of priority inversions to improve the processing of non-hard real-time tasks in mixed systems", *Proc. WIP 21st IEEE Real-Time Systems Symposium*, pp. 13-16, 2000.

[10] Orozco J., J. Santos, R. Cayssials, E. Ferro, "Mixing Real and non Real-Time Processing: Priority Inversions are not so Bad After All", *Proc. XII Euromicro Conference, WIP Session*, 2000.

[11] Pandya, M., M. Malek, "Minimum Achievable Utilization for Fault-Tolerant Processing of Periodic Tasks", *IEEE Trans. On Computers*, 47 (10) 1102-1112,1998.

[12] Ramos-Thuel, S.,J.K. Strosnider, “Scheduling Fault Recovery Operations for Time-Critical Applications”, *Proc. 4th IFIP Conference on Dependable Computing for Critical Applications,* 1995.

[13] Santos, J., E. Ferro, J. Orozco, and R. Cayssials, "A heuristic approach to the multitask-multiprocessor assignment problem using the empty-slots method and rate monotonic scheduling", *Real-Time Systems Journal*, (13) 2 pp. 167-200, 1997.

[14] Santos, J, J. Orozco, “Rate Monotonic Scheduling in Hard Real-Time Systems”, *Information Processing Letters*, 48, 39-45, 1993.

[15] Sprunt, B., L. Sha, J. Lehoczky: "Aperiodic Task Scheduling for Hard Real-Time Systems", *Real-Time Systems Journal* 1 (1), 27-60, 1989.