UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN.

Desarrollo de un Sistema de Tiempo Real para Cómputo de Alto Desempeño

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA
(COMPUTACIÓN)

P R E S E N T A:

Miguel Ángel Palomera Pérez

DIRECTOR DE TESIS: Dr. Héctor Benítez Pérez

México, D.F. 2005.

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Agradecimientos

Por su apoyo continuo, desvelos, esperanzas puestas en mí: **gracias Ma.**

Por sus dudas inquietantes, su cariño, y por las tardes de películas: **gracias Hermana.**

Por esa carrera siempre en ascenso en mi vida, convirtiéndose en una parte importante de ella: **gracias Gorda.**

Por confiar en que seria capaz de terminar este trabajo: **gracias Doc. (Héctor).**

Por sus comentarios, sugerencias y tiempo: **gracias Doc. (Luis).**

Por sus charlas y su confianza: **gracias Pancho.**

A mis compañeros gracias por compartir esta experiencia conmigo.

A los profesores por dar lo mejor de sí.

Al coordinador del posgrado, Dr. Boris, por escucharnos y preocuparse por nuestra preparación.

A las secretarias del posgrado (Lulú, Diana, Amalia) por hacer esta estancia un poco más amena.

A la señora Cecilia por encontrar siempre la manera de apoyarnos.

A Juanita por tenernos el material siempre listo y sus palabras de apoyo.

A todos los que de alguna forma u otra han contribuido a hacer de mí una mejor persona.

Agradezco el apoyo prestado por el proyecto PAPIT clave IN105303 sin el cual no hubiese sido posible la realización de este trabajo.

Índice general

1. Definición del problema 1

1.1. Introducción 1

1.2. Objetivos de la tesis 2

1.3. Aportaciones de este trabajo. 3

1.4. Estructura de la tesis 3

2. Generalidades 5

2.1. Introducción 5

2.2. Tiempo real 5

2.3. La necesidad de sistemas distribuidos de alto rendimiento 7

2.4. Arquitectura propuesta 10

2.4.1. Características de las tareas 11

2.4.2. Ambiente de ejecución. 11

2.4.3. Planificación: manejador de tareas y manejador de mensajes 12

2.4.4. Proceso de transmisión de los mensajes 15

2.4.5. Proceso completo: desde la especificación del escenario hasta la entrega de resultados 16

2.5. Conclusiones 18

3. Implementación 21

3.1. Introducción 21

3.2. Usando MPI 22

3.3. Enviando mensajes en tiempo real 24

3.3.1. RTMPI_Send 27

3.3.2. RTMPI_Recv 28

3.3.3. RTMPI_Init 28

3.3.4. RTMPI_Finalize 29

3.3.5. RTMPI_Comm_rank 29

3.3.6. RTMPI_Comm_size 29

3.4. Reloj del sistema 29

3.5. Implementación de los manejadores 31

3.6. Características de la red SCI 31

3.7. Sincronización 34
3.8. Conclusiones 36

4. Análisis de planificabilidad 37
4.1. Introducción 37
4.2. Modelo del Sistema 38
4.3. Tiempos de respuesta 40
4.4. Tiempos de transmisión 43
4.5. Cálculo de los *jitters* 45
4.6. Análisis numérico y simulación 47
4.6.1. Conjuntos de prueba 47
4.6.2. Características del simulador 48
4.6.3. Experimentos realizados 53
4.7. Conclusiones 55

5. Caso de estudio 57
5.1. Introducción 57
5.2. Descripción del caso de estudio 57
5.3. Desarrollo 59
5.4. Análisis numérico y simulación 59
5.5. Codificación 62
5.5.1. Ejecución 68
5.6. Validez de los resultados 69
5.7. Conclusiones 72

6. Resultados 73
6.1. Introducción 73
6.2. Arquitectura 73
6.2.1. Red 74
6.3. Análisis 74
6.4. Conclusiones 76

7. Conclusiones 77
7.1. Trabajo Futuro 77
7.2. Comentarios Finales 78

A. Glosario 79

B. Código fuente del caso de estudio 85
B.1. Implementación del Algoritmo de las redes ART2 85
B.2. Código fuente de la primera red 89
B.3. Código fuente de la segunda red 90
B.4. Código fuente de la tercera red 92

Índice de figuras

2.1. Modelado de relaciones entre tareas. 11
2.2. Arquitectura propuesta. 13
2.3. Transmisión de mensajes. 16
2.4. Ejemplo de archivo de configuración. 17

3.1. Estructura general de los programas que emplean MPI. 22
3.2. Definición de las funciones: MPI_Send y MPI_Recv. 23
3.3. Ejemplo de uso de mpi. La tarea 0 recibe los mensajes generados por el resto de las tareas. 25
3.4. Formato de los paquetes. 27
3.5. Estructura general de los manejadores. 32
3.6. Tiempo promedio empleado en la transmisión de datos. 33
3.7. Algoritmo de sincronización. 35

4.1. Sistema de dos nodos. 39
4.2. El problema del *jitter*. 40
4.3. Interferencia y Bloqueo. La interferencia es causada por tareas de mayor prioridad mientras que el bloqueo por las de menor. 41
4.4. Función de disponibilidad $A(t)$ y función de carga $H_i(t)$. 44
4.5. *Jitters* existentes en la transmisión de los mensajes. 45
4.6. Algoritmo iterativo para realizar el análisis de planificabilidad. 46
4.7. Algoritmo empleado por el simulador. 49
4.8. Ejemplo de archivo de entrada para el simulador. 50
4.9. Ejemplo de archivo de entrada para el simulador.(Continuación) 51
4.10. Ejemplo de simulación: precedencias de tareas. 52
4.11. Ejemplo de simulación: ejecución. 52
4.12. Porcentaje de conjuntos planificables: número de mensajes, M, variable (20 %,60 % y 80 % del total de tareas); ancho de banda usado por los mensajes constante e igual a 30 %; número de nodos constante (2). 54
4.13. Porcentaje de conjuntos planificables: número de mensajes, M, constante (50 % del total de tareas); ancho de banda usado por los mensajes, U_m, variable (20 %,40 % y 60 %); número de nodos constante (2). 54

4.14. Porcentaje de conjuntos planificables: número de mensajes, M, constante (30 % del total de tareas); ancho de banda usado por los mensajes, U_m, constante (30 %); número de nodos variable (3,5,6). 55

5.1. Diagrama de la técnica traslape de redes ART. 58
5.2. División de las tareas en subtareas. (a)Tareas como son implementadas. (b) Tareas divididas en subtareas. (c) Costo y periodo de cada uno de las subtareas. 61
5.3. Extracto de la gráfica generada por el simulador. 63
5.4. Formato alternativo para implementar las redes como un solo programa. . . 64
5.5. Extracto del código fuente que implementa a la primera red (Tarea 1). . . . 65
5.6. Extracto del código fuente que implementa a la segunda red (Tarea 2). . . . 66
5.7. Extracto del código fuente que implementa a la tercera red (Tarea 3). 67
5.8. Archivo de configuración del caso de estudio (rtmpi.cfg). 68
5.9. Número de escenarios necesarios por la primera red. 70
5.10. Número de escenarios necesarios por la segunda red. 71
5.11. Número de escenarios necesarios por la tercera red. 71
5.12. Tiempo empleado por cada una de las rondas. 72

6.1. Tiempo empleado para colocar los mensajes en la cola de mensajes a enviar. 75
6.2. Tiempo de transmisión desde que el mensaje es creado hasta que llega a su destino. 76

Índice de tablas

2.1. Características de los manejadores. 14

3.1. Tipos de datos definidos en MPI. 24

4.1. Parámetros de los mensajes. 38
4.2. Parámetros empleados a través de los experimentos. 53

5.1. Parámetros empleados para realizar el análisis numérico. 60
5.2. Extracto de los tiempos de respuesta arrogados por el análisis numérico. . . 61

Capítulo 1

Definición del problema

1.1. Introducción

La correctez de la mayoría de los sistemas y dispositivos usados por la sociedad actual dependen no solo de los efectos o resultados que estos producen, sino también en el instante en que son producidos. A este tipo de sistemas se les conoce como sistemas de tiempo real y los podemos encontrar en un amplio rango de aplicaciones: desde el sistema de frenos en los automóviles hasta el sistema de monitoreo de los signos vitales de una unidad de cuidado intensivo.

Durante los últimos años, el incremento de la capacidad de los equipos de cómputo y el desarrollo de técnicas que han permitido mejorar la calidad del software han hecho que estas herramientas se emplean para monitorear y controlar a los sistemas de tiempo real. Ahora es posible encontrar sistemas operativos de propósito general que incorporan características de tiempo real para manejar vídeo sobre demanda o juegos de manera distribuida. También existen sistemas de tiempo real que se encuentra empotrados en aplicaciones especificas como son los usados por los sistemas de control de los aviones y los transbordadores espaciales.

Sin embargo, existen aplicaciones que requieren al mismo tiempo cumplir con restricciones temporales como procesar una gran cantidad de información para las cuales la capacidad de un solo nodo de procesamiento serían insuficientes. Tomemos por ejemplo el caso hipotético que se desease desarrollar un sistema que controlará el taladro de perforación de un pozo petrolero. Para ello se requiere ir procesando los datos que se van teniendo respecto a las características del terreno, posible cercanía del manto petrolífero, presión, y algunas situaciones no del todo previstas y transmitir en fracciones de segundos los comandos que controlen la velocidad del rotor, la presión aplicada etc.

Una solución factible sería contar con un sistema de cómputo de alto desempeño, como lo sería un *cluster*[1], el cual se encargará de procesar la información y transmitir los resultados por medio de una red de alta velocidad, posiblemente inalámbrica, a los dispositivos encargados de controlar al taladro. Por su parte los sensores constantemente reportarían las condiciones del medio al *cluster.*

Aunque hipotético, el sistema de control del taladro, dada la tecnología actual es posible realizarlo: al menos desde el punto de vista del equipo. En la actualidad existen sensores de

[1]Conjunto de computadores, interconectadas mediante una red de datos, configuradas de tal forma que se comportan como si se tratasen de una sola.

alta precisión y con frecuencias de muestreo en microsegundos, también se han desarrollado redes de alta velocidad, inclusive redes inalámbricas con estas características; además existen técnicas de compresión y corrección de datos que permiten transmitir grandes volúmenes de información. Por otra parte, gracias al desarrollo de tecnologías como la de los *cluster* es posible crear sistemas distribuidos de alto desempeño por una fracción del costo tradicional de una supercomputadora.

El mayor reto de emplear sistemas de alto desempeño para el procesamiento de datos en tiempo real, es que no existe todavía una tecnología que permita garantizar que las restricciones temporales de las tareas del sistema sean cumplidas. Dentro del área de los *clusters* el principal paradigma empleado para realizar tareas paralelas es el envío de mensajes entre ellas. Existiendo en la actualidad básicamente dos tecnologías MPI (*Message Passing Interface*: Interfaz de Paso de Mensajes) y PVM (*Parallel Virtual Machine*: Máquina Virtual Paralela). Sin embargo, ninguna de estas tecnologías ofrece soporte para sistemas de tiempo real.

Una tecnología capaz de garantizar la calidad de los servicios en sistemas de alto rendimiento es *The Ace ORB* [TAO] cuyo objetivo es permitir a las tareas realizar operaciones sobre objetos[2] distribuidos sin importar donde estos estén localizados, el lenguaje de programación empleado, los protocolos de comunicación, etc. Desgraciadamente ésta sigue siendo poco difundida, requiere aprender nuevos conceptos y es relativamente difícil de implementar.

1.2. Objetivos de la tesis

En vista de la falta de una plataforma de desarrollo de aplicaciones de tiempo real en sistemas de alto desempeño, se propone como objetivo de este trabajo **desarrollar una arquitectura de software que de soporte a aplicaciones de tiempo real en sistemas de cómputo distribuido de alto desempeño mediante el paradigma de envío de mensajes.**

Se propone emplear al paradigma de envío de mensajes porque es la técnica más difundida en el área de sistemas de alto rendimiento, en particular se emplea al estándar MPI con este fin, con lo que se espera disminuir la curva de aprendizaje de futuros desarrolladores así como facilitar el portar aplicaciones existentes a esta nueva tecnología.

Con esta nueva arquitectura se espera lograr que el sistema sea predecible, es decir, que sea posible garantizar el instante en que los resultados serán producidos. A su vez se desea que el ambiente resultante sea sencillo de ejecutar; sin complejos parámetros que configurar: simplemente especificar el número y características de las tareas y mensajes de los que esta compuesto.

También deberá ser posible monitorear a las tareas para determinar el instante en que éstas se ejecutan y cuanto tiempo les lleva hacerlo, del mismo modo se deberá determinar el tiempo de transmisión empleado por los mensajes. Cumplir con este requerimiento le permitirá al desarrollador realizar una etapa de pruebas con el fin de determinar alguno de los parámetros temporales de las tareas como puede ser el tiempo de ejecución de éstas, al mismo tiempo, que podrá recabar información estadística que le permita demostrar que el sistema se comporta tal como se esperaba.

[2]El término objeto tiene la misma connotación que el dado en las tecnologías orientadas a objetos, es decir, es una abstracción del mundo real que se convierte en un tipo de dato con un conjunto de operaciones para manipularlo.

1.3. Aportaciones de este trabajo.

En la actualidad, el empleo de *clusters* para resolver problemas computacionalmente demandantes es un estrategia ampliamente aceptada. Existiendo además herramientas como MPI y PVM que ayudan a los desarrolladores a crear aplicaciones para estos sistemas. Sin embargo, existen situaciones en las cuales la ejecución de las tareas tiene restricciones temporales, por ejemplo simuladores de vuelo, en cuyo caso no es posible emplear estas herramientas debido a que no toman en cuenta el comportamiento temporal del sistema. Por lo tanto, este trabajo propone una arquitectura apropiada para la ejecución y análisis de aplicaciones en tiempo real que además son computacionalmente demandantes. Dicha arquitectura está basada en un *cluster* cuyos nodos emplean el sistema operativo GNU[3]/Linux con modificaciones a la precisión del reloj, interconectados por una red de alta velocidad que implementa el protocolo TDMA (*Time Division Multiplexed Access*: Acceso Múltiple por División Temporal).

Además se desarrolló un prototipo de dicha plataforma con el objetivo de comprobar las capacidades para la ejecución de tareas en tiempo real con las que cuenta el sistema. Por otra parte, para garantizar que la aplicación a desarrollar cumple con sus restricciones temporales, se determinaron un conjunto de ecuaciones con las cuales es posible realizar un análisis numérico del sistema mediante el cual es posible encontrar cualquier violación a los requerimientos temporales.

A su vez, se construyó un simulador el cual puede emplearse como apoyo para el desarrollador para determinar la distribución de las tareas a los distintos nodos, y/o para garantizar que el sistema se comporta como se esperaba.

Utilizando éstas herramientas se desarrolló un caso de estudio el cual sirve de ejemplo del funcionamiento de esta plataforma.

1.4. Estructura de la tesis

En el **capítulo 2** se hace un resumen de las características de los sistemas de tiempo real, así como, del porqué utilizar sistemas de alto rendimiento. En este capítulo también se describe la arquitectura propuesta.

Un pequeño ejemplo de como se utiliza el estándar MPI aparece en el **capítulo 3**. La forma en que la arquitectura fue implementada usando para ello algunas características especificas del sistema operativo GNU/Linux como lo es el acceso directo al reloj físico de la computadora, es el tema principal de este capítulo. También se habla de las características de la red de datos empleada.

Para poder garantizar que un conjunto de tareas y mensajes satisfacen sus restricciones temporales es necesario realizar un análisis de planificabilidad. En el **capítulo 4** se desarrollan las ecuaciones que permiten realizar dicho análisis. También se diseña un simulador, que junto con las ecuaciones, se emplea para determinar la correctez de éstas últimas mediante un análisis estadístico.

[3]G No Unix. La letra "G" se emplea para indicar que es un término recursivo, por lo que GNU se expandirá al término No Unix, No Unix, No Unix .... indefinidamente. Ver el Apéndice B para una explicación más detallada.

En el **capítulo 5** se realiza un caso de estudio con el objetivo de corroborar que el sistema cumple con los objetivos planteados. Además dicho caso sirve de ejemplo de como utilizar a la plataforma.

Un resumen de los resultados obtenidos en cada una de las etapas del desarrollo del sistema se encuentra en el **capítulo 6**.

Los conclusiones y trabajo futuro se exponen en el **capítulo 7.**

Capítulo 2

Generalidades

2.1. Introducción

El término tiempo real es usado comúnmente para referirse a una situación que ocurre en el instante que se está hablando. Sin embargo, en el área de cómputo usarlo de esta manera sería cometer un grave error. En esta área, un sistema de tiempo real es aquel en el que se puede garantizar el instante en que se producirán los resultados esperados. Y aunque pareciera que estos sistemas sólo se encuentran dentro de los laboratorios de las universidades, forman una parte esencial de nuestra vida diaria. Por ejemplo, cuando se transmite vídeo de forma digital se debe garantizar que no existan retrasos entre la señal de audio y la imagen que está siendo desplegada.

Ahora bien, algunos de los sistemas de tiempo real, debido a su complejidad, están compuestos por más de un dispositivo. Un ejemplo clásico de un sistema distribuido de tiempo real es el sistema de frenado encontrado en la mayoría de los automóviles hoy en día.

En el caso de los sistemas de cómputo distribuido en los últimos años se ha desarrollado un conjunto de librerías y protocolos para permitir el procesamiento paralelo de datos, sin embargo, garantizar que un sistema como éste cumpla con restricciones de tiempo real continua siendo un problema abierto.

En este capítulo se propone una plataforma que permite llevar a cabo tareas de tiempo real dentro de un sistema distribuido de alto rendimiento, como lo es un *cluster*, conjunto de computadoras configuradas de tal forma que se comportan como si se tratase de una sola.

El resto del capítulo se encuentra dividido de la manera siguiente. Los términos comúnmente empleados en la área de tiempo real se definen en la sección 2.2. En la sección 2.3 se muestra porque utilizar cómputo distribuido continua, y continuará, siendo una de las mejores alternativas no obstante que la capacidad de procesamiento de las computadoras personales continué aumentando. La arquitectura propuesta se describe en la sección 2.4. Finalmente la sección 5.7 concluye al capítulo.

2.2. Tiempo real

La correctez de muchos de los sistemas y aparatos empleados en nuestros días depende no sólo de los resultados que estos producen sino también en el instante en que son producidos. Un buen ejemplo es un sistema compuesto por un brazo mecánico que necesita tomar

una pieza de una banda sinfín. Si el brazo llega tarde, la pieza ya no estará donde debía recogerla. Por lo tanto el trabajo se llevó acabo incorrectamente, aunque se haya llegado al lugar adecuado. Por el contrario, si llega antes de que la pieza llegue, ésta aun no estará ahí y el brazo puede bloquear su paso.

La definición canónica de un sistema de tiempo real de acuerdo a [Comp.realtime] es:

> "Un sistema de tiempo real es aquel en el que la correctez de las operaciones computacionales no solo depende que la lógica e implementación de los algoritmos lo sea, sino también en el tiempo en el que se entrega su resultado. Si las restricciones de tiempo no son respetadas el sistema ha fallado."

Por su parte otros han agregado:

> "...Para garantizar el comportamiento en el tiempo se requiere que el sistema sea predecible. Es también deseable que se obtenga un alto grado de utilización a la vez que se cumple con los requerimientos de tiempo."

Ejemplos de sistemas de tiempo real incluyen: versiones computarizadas de sistemas de monitoreo de signos vitales, redes de alto rendimiento y sistemas de enrutado telefónico, herramientas multimedia, sistemas de realidad virtual, aparatos de comunicación inalámbricos (tales como celulares y PDAs), telescopios astronómicos con sistemas ópticos adaptables, y muchas aplicaciones industriales consideradas como críticas.

Por su parte, las restricciones temporales de una tarea pueden ser tanto **duras** (*hard*): si una falta en su plazo conlleva resultados catastróficos para el sistema; como **suaves** (*soft*): si se tolera que en algunas ocasiones el plazo no sea cumplido. También existen aplicaciones para las cuales si no se cumple con las restricciones temporales el resultado que se genera no tiene ningún uso práctico sin provocar a una situación catastrófica, este tipo de restricciones se conocen como **firmes** (firm). En la práctica los sistemas de tiempo real puede estar formados por todos los tipos de tareas.

En el caso de sistemas de tiempo real duros se requiere que el sistema sea rigurosamente validado para demostrar que no se incurre en ninguna violación de los plazos establecidos. Para ello se emplean métodos numéricos (de los cuales se esta completamente seguro de su validez) y/o simulaciones exhaustivas. Lo mismo ocurre en sistemas de tiempo real firme, ya que aunque las faltas no sean catastróficas, si se comente los resultados generados no tienen ninguna utilidad. Mientras que en los sistemas de tiempo real suaves, gracias a la flexibilidad que se tienen en cuanto al cumplimiento de los plazos, es posible considerar otros requerimientos, como la calidad de servicios.

En ambos tipos de sistemas se requiere definir a las tareas involucradas mediante sus requerimientos temporales con el objetivo de realizar la planificación y validación de estas. Por lo general es suficiente con especificar el valor del:

Tiempo de activación (*release time*): es el instante de tiempo en el cual la tarea se convierte en elegible para ejecutarse. Cuando las tareas son aperiódicas o esporádicas este tiempo no puede conocerse con exactitud. Por su parte, cuando las tareas son periódicas y existen variaciones en su tiempo de activación se dice que la tarea sufre de *release jitter*. Se le llama *realease jitter* a las variaciones que existen del promedio del tiempo de activación. Si este es el caso este parámetro se expresa mediante un intervalo.

Plazo absoluto (*deadline*): es el instante de tiempo en el cual la tarea ya debe estar completada.

Plazo_relativo (*relative deadline*): es el plazo de la tarea con respecto al instante de activación de la tarea.

Peor tiempo de ejecución (*worts-case executing time*): es la cantidad de tiempo necesaria para que la tarea complete su ejecución considerando que se ejecuta sola y que tiene todos los recursos disponibles. Determinar este tiempo es fundamental para el correcto análisis del sistema. Una forma aproximada de obtener este valor es realizar una etapa de pruebas y emplear el mayor tiempo observado durante ésta. Sin embargo, si el sistema que se esté desarrollando requiere mayor precisión es necesario emplear métodos más confiables, como los descritos en [Cheng M.K., 2002], con este fin, esto debido a que el peor tiempo obtenido mediante experimentación puede no ser igual al encontrado cuando el sistema se encuentre en producción.

Ceder su turno (Preemptivity): es la capacidad que tiene o no una tarea de ser interrumpida, y después reanudada en el mismo punto, para ejecutar otras tareas de mayor prioridad.

Empleo de recursos (*resource requirements*): este parámetro indica los recursos que necesita la tarea para ejecutarse así como por cuanto tiempo hará uso de ellos.

En el caso que se tenga tareas periódicas, además de los parámetros anteriores, éstas se definen mediante su **periodo** y su **fase** inicial. El periodo corresponde al tiempo entre dos activaciones consecutivas, ahora bien, si los tiempos de activación sufren de *jitter* para calcular el periodo se emplea el promedio de los intervalos entre activaciones consecutivas. Por su parte la fase inicial es igual al instante de activación de la primera instancia de la tarea.

2.3. La necesidad de sistemas distribuidos de alto rendimiento

No obstante que la capacidad de cómputo continua incrementándose haciendo posible simular fenómenos que de otra forma serían imposibles de estudiar (como es el caso de choques entre galaxias), por lo general para los problemas encontrados en la ciencia y la ingeniería esta capacidad resulta insuficiente. Como ejemplo de esto, considérese que se desea predecir la condiciones climáticas de Norte América[1] (México, Estados Unidos y Canadá) cada hora durante los próximos dos días. Supóngase también que se desea modelar la atmósfera desde el nivel del mar hasta una altura de 20 km.

Una forma de resolver este tipo de problemas es modelar la región de interés como una malla y predecir el clima en cada uno de los vértices de ésta. Si se empleasen cubos de 0.1 kilómetros por cara para realizar el modelo y tomando en cuenta que la área de Norte América es de 22 millones de kilómetros cuadrados, se tendrán al menos

[1]Éste es un ejemplo ligeramente modificado del presentado en [Demmel, 1996]

$$2{,}2 * 10^7 km^2 * 20km * 10^3 cubos\ por\ km^3 \ = \ 4{,}4 * 10^{11}$$

puntos donde realizar la predicción. Ahora bien, si se requiere 100 operaciones para determinar el clima en un punto, se necesitan $4{,}4 * 10^{13}$ operaciones para predecir el clima de la siguiente hora. Como se quiere predecir el clima durante las siguientes 48 horas, se necesitan realizar aproximadamente

$$4{,}4 * 10^{13}\ operaciones\ * 48\ horas \ = \ 2{,}112 * 10^{15}\ operaciones$$

Si una computadora promedio puede realizar 10^9 operaciones por segundo[2], realizar este cálculo tomará

$$\frac{2{,}112 * 10^{15}\ operaciones}{10^9 operaciones\ por\ segundo} \ = \ 2{,}112 * 10^6\ segundos$$

lo que equivale aproximadamente a 25 días. Lo cual en este caso es inaceptable. Ahora bien, si se contase con una computadora con la capacidad de 10^{12} operaciones de punto flotante por segundo éste podría realizarse en media hora. Sin embargo, no es complicado imaginar algunas sencillas modificaciones a este problema para las cuales un computador con esta capacidad sería insuficiente: por ejemplo en lugar de Norte América podría emplearse todo el continente. Es más, es posible encontrar problemas similares con mayores requerimientos de computo como por ejemplo en lugar de predecir el clima se podría desear predecir las variaciones de temperatura en el Océano Pacífico.

Por otro lado, la velocidad de procesamiento no es el único reto a vencer cuando se construye una supercomputadora, si no es posible acceder a los datos a velocidades similares varios ciclos de reloj serán desperdiciados. Para imaginar la complejidad de este problema considérese que en una máquina se desea ejecutar el siguiente código en un segundo[3]:

```
/*x, y , z son arreglos de valores de punto flotante*/
/* con 10^12entradas cada uno*/
for (i=0; i < 10^12 ; i++)
   z[i] = x[i] + y[i];
```

En una computadora convencional, es decir basada en la arquitectura Von Newman, para realizar cualquier operación se requiere que los datos a ser operados se encuentren en los registros del procesador, a continuación se realiza la operación y finalmente se almacena el resultado. Por lo tanto para ejecutar el código anterior se requiere copiar de memoria a los registros al menos $3 * 10^{12}$ valores por segundo. Si los datos se transfieren a la velocidad de la luz ($3 * 10^8$ metros/segundo) y si "r" es la distancia entre el CPU y la memoria, entonces "r" debe satisfacer:

[2]Esto claro si se supone que en promedio cada una de las instrucciones en un procesador de 2 Giga Hertz consume a lo más dos ciclos de reloj.

[3]Ejemplo basado en el encontrado en [Demmel, 1996]

$$3 * 10^{12} * r \; metros \; = \; 3 * 10^{8} \; \frac{metros}{segundo} * 1 \; segundo$$

es decir $r = 10^{-4}$ metros. Si esto fuera poco, nuestra memoria debe tener capacidad para al menos $3 * 10^{12}$ variables de punto flotante, 32 bits por variable, algo así como 96 Giga bits de memoria y en promedio cada una de las celdas de memoria debe estar a una distancia r de los registros del procesador.

La alternativa obvia cuando se tiene un problema que demande gran capacidad de cómputo es el empleo de sistemas distribuidos: un conjunto de computadoras trabajan juntas para resolver un problema. El ejemplo más sobresaliente de este tipo de sistemas es el proyecto SETI@home [SETI] creado originalmente por la Universidad de Berkeley, California. En él las señales obtenidas por el radio-telescopio de Arecibo, Puerto Rico, son enviadas a un computador dentro de la Universidad, donde se dividen en fragmentos que se reparten entre los más de 3 millones de usuarios inscritos al proyecto. En promedio este sistema tiene una capacidad de $14 * 10^{12}$ operaciones de punto flotante por segundo.

Este tipo de proyectos son conocidos como *Grids* [GRID1, GRID2], los cuales son una forma de computación distribuida donde la cantidad de nodos de procesamiento y/o almacenamiento varia con el tiempo. La principal característica de estos sistemas es que emplean un modelo cliente-servidor para realizar sus funciones. De esta manera cuando el cliente esta disponible se conecta al servidor el cual le transfiere los datos que tiene que procesar: cuando termine los resultados son enviados de regreso.

Por su parte, un *cluster* de computadoras es un sistema estático, la cantidad de nodos disponibles es constante, lo que permite que las tareas puedan comunicarse directamente unas a otras.

Sin embargo, contar con los recursos de cómputo suficiente (capacidad de proceso y memoria) es sólo una pequeña parte a la hora de desarrollar aplicaciones en sistemas distribuidos, además se tiene que:

- diseñar e instalar la red de datos que interconecte a los distintos nodos. En algunos casos esto incluye a los módulos de memoria.
- configurar al sistema operativo para que comparta los recursos de que dispone, garantizando en todo momento la integridad de los datos.
- diseño y desarrollo de algoritmos y estructuras de datos acordes al problema.
- dividir la tarea a realizar en subtareas lo más independientemente posible unas de otras.
- identificar las comunicaciones que existen entre las subtareas para garantizar que estas pueden realizarse.
- realizar la asignación de las subtareas a los nodos de acuerdo a los requerimientos que éstas presenten.

2.4. Arquitectura propuesta

La primera generación de aplicaciones de tiempo real empleaban ambientes monoprocesador debido a que los problemas que trataban de resolver eran relativamente simples. En nuestros días; soportados por los avances logrados en las áreas de: redes de alta velocidad, procesamiento y almacenamiento de datos; son cada vez más los sistemas de tiempo real en ambientes distribuidos. Por ejemplo, en [Goddard y Jeffay, 1996] utilizan los métodos de planificación encontrados en tiempo real para delimitar la latencia y utilización de memoria en sistemas distribuidos empleados para el procesamiento digital de señales. Las aplicaciones multimedia son otro ejemplo de sistemas que requieren tanto capacidad de procesamiento como satisfacer restricciones temporales para poder garantizar la calidad de los servicios que ofrecen.

Desgraciadamente, existe una enorme brecha en el soporte para tiempo real en los sistemas operativos encontrados en ambientes académicos y los desarrollados comercialmente. Una de las propuestas con mayor aceptación es contar con un sistema dual: un sistema operativo de propósito general y un microkernel que de soporte a las tareas de tiempo real. Ejemplos de este tipo de sistemas son RT-Linux y RTAI . Sin embargo, para poder construir un *cluster* de alto rendimiento con soporte para tiempo real se necesita, además, contar con redes de datos y protocolos que garanticen el cumplimiento de los requerimientos de comunicación impuestos por la aplicación que se esté desarrollando.

En [Heimfarth et al, 2003] se describe RTC (*Real Time Communication*: Comunicaciones en Tiempo Real) , un *middleware* basado en CORBA (*Common Object Request Broker Architecture*) [CORBA-FAQ] que implementa una plataforma de comunicación para tiempo real sobre un *cluster* funcionando con RTAI *(Real-Time Application Interface*). Se le llama *middleware* a una capa de software que permite la interacción entre sistemas heterogéneos: ejemplos de estos sistemas incluyen servicios de directorio (LDAP), mecanismos de envío de mensajes (MPI,PVM), agentes de solicitudes de objetos (CORBA,RMI), llamadas a procedimientos remotos, y sistemas de control de acceso a bases de datos. Cabe destacar la existencia de TAO (*The Ace Orb*)[TAO], un *middleware* que extiende las capacidades de CORBA para soportar aplicaciones de tiempo real.

A diferencia del sistema propuesto en [Heimfarth et al, 2003], en esta tesis se propone utilizar el paradigma de envió de mensajes con el mismo fin. En particular se describe una plataforma capaz de soportar al estándar MPI[MPI]. La razón de emplear este estándar es el de disminuir la curva de aprendizaje de una nueva herramienta y facilitar el traslado de antiguos proyectos a esta arquitectura.

Para el desarrollo del prototipo se emplea un *cluster Beowulf*[4], que cuenta con cuatro nodos interconectados mediante una red SCI (*Scalable Coherent Interface*: Interfaz Coherente y Escalable)[5]. Cada uno de los nodos tiene instalado el sistema operativo GNU/Linux (distribución Debian, versión Woody) sin modificaciones para soportar tiempo real, por lo que sólo puede emplearse para sistemas de tiempo real suave, sin embargo, esto es suficiente para probar el diseño y demostrar la viabilidad de un sistema como éste .

[4] http://www.beowulf.org/

[5] Para mayor información acerca de esta red se remite al lector a la sección sobre SCI en el capítulo 3.

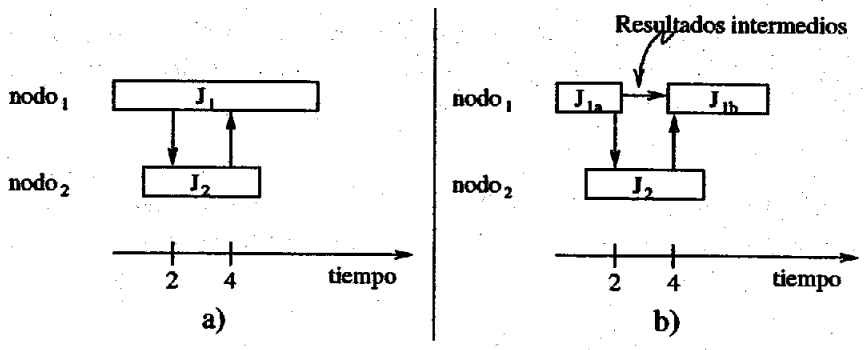


Figura 2.1: Modelado de relaciones entre tareas.

2.4.1. Características de las tareas

Para el diseño del sistema se considera que las tareas son periódicas, sin embargo es posible contar con tareas que se ejecuten una sola vez haciendo su periodo de éstas igual a cero. El caso de tareas esporádicas es más complicado, una forma de abordarlo es mediante una tarea periódica, comúnmente conocida como servidor periódico (*periodic server*), cuya función sea ejecutarlas: estableciendo un compromiso entre cuantas tareas esporádicas puedan atenderse y el tiempo que se tenga disponible para ejecutar al servidor.

Con respecto al envío de mensajes y las precedencias que de éste resultan, solo se consideran tres tipos básicos de tareas: tareas que envían mensajes, tareas que reciben mensajes y tareas que primero reciben y después envían mensajes.

En ninguno de los casos existe alguno restricción en la cantidad de mensajes que una tarea puede recibir y/o enviar, por lo tanto, contar con sólo estos tipos de tareas no representa ninguna perdidas de generalidad. Por ejemplo, consideremos el escenario mostrado por la Figura 2.1a), donde la tarea J_1 en el instante 2 le envía un mensaje a la tarea J_2, posteriormente en 4 ésta envía un mensaje de respuesta a J_1. En la Figura 2.1b) ponemos ver como esta relación se puede modelar empleando solo a los tipos antes definidos. Para ello la tarea J_1se subdivide en: dos tareas, una tarea que sólo envía mensajes J_{1a} y en otra que sólo recibe J_{1b}; y un mensaje con resultados intermedios. La tarea J_2 no necesita modificarse.

2.4.2. Ambiente de ejecución.

Las tareas dentro de un sistema distribuido requieren soporte para comunicarse, así como control de acceso a recursos compartidos. Por otra parte, si estas tareas son de tiempo real, además se debe contar con métodos para indicar sus características temporales. A todos los recursos que necesita un programa para ejecutarse se le conoce como **ambiente de ejecución**. Una forma de entender el ambiente encontrado en los sistemas distribuidos es separarlo en los procesos de:

Planificación y manejo de tareas: Incluye a las funciones encargadas de determinar en que procesador se habrá de ejecutar cada una de las tareas, así como garantizar que éstas cuenten con los recursos suficientes. Por otra parte, una vez que una tarea comienza, ésta debe controlarse. Se debe asegurar que se suspenda cuando exista otra de mayor prioridad y que posteriormente continué su ejecución en el mismo punto

donde se detuvo. En algunos sistemas estas funciones son realizadas por una pieza complicada de software; en otros se le deja al usuario que determine donde serán ejecutadas las tareas, mientras que el sistema operativo se encarga de las funciones encaminadas a controlar la ejecución de éstas.

Librería de envío de mensajes: El envió de mensajes es el principal paradigma de comunicación empleado en sistemas distribuidos. En particular, en el caso de los *clusters*, en la actualidad existen básicamente dos estándares: MPI y PVM [PVM].

Seguridad: Las funciones de seguridad incluyen procesos enfocados a mantener la consistencia del sistema. Como garantizar que las tareas no accedan a recursos de acceso exclusivo, así como que el calendarizador sólo controle a las tareas que él inició, y que la librería de envío de mensajes los entregue sólo a su destinatario. Por lo general estas funciones no se desarrollan como un módulo independiente, sino que forman parte del calendarizador y/o de la librería de mensajes.

En la mayoría de los ambientes de ejecución disponibles actualmente para sistemas distribuidos existe una tarea encargada de las funciones de planificación y manejo de tareas, por ejemplo en *mpich* [MPICH] se emplea un algoritmo *round-robin* para determinar en que nodo se han de ejecutar cada una de las tareas, las cuales se inician concurrentemente en el nodo local y de forma paralela en los nodos remotos. Pero esto no es suficiente cuando se trata de sistemas de tiempo real, en los cuales se requiere garantizar que todas las tareas cumplan con su plazo lo que implica emplear algoritmos de planificación más complejos. Ejemplos de esquemas de planificación empleados en tiempo real son:

Frecuencia Monótona (Rate Monotonic:RM): La tarea con periodo más pequeño es la que tiene la mayor prioridad.

***Plazo Monótono (Deadline Monotonic*:DM):** La tarea con el plazo más pequeño es la que tiene la mayor prioridad.

Plazo mas Cercano Primero (Earliest Deadline Firts:EDF): La tarea con el plazo más cercano al tiempo actual es la que tiene mayor prioridad.

Menor Tiempo Sobrante Primero (Least Laxity First:LLF): Supóngase que $c(t)$ es el tiempo que requiere una tarea en el instante t para terminar su ejecución, por su parte $d(t)$ es el plazo de esa misma tarea relativo a t, entonces el tiempo sobrante esta dado por $d(t) - c(t)$. En otras palabras el tiempo sobrante es igual al máximo tiempo que una tarea puede retrasar su ejecución sin perder su plazo. Por lo tanto, la tarea con menor tiempo sobrante es la que tiene mayor prioridad.

Por otra parte, debido a que el canal de comunicación es un recurso compartido, su mala utilización lo puede convertir en un cuello de botella.

2.4.3. Planificación: manejador de tareas y manejador de mensajes

Para llevar a cabo la ejecución de las tareas se propone contar en cada uno de los nodos con un calendarizador controlado por reloj, es decir, se genera una alarma que interrumpe

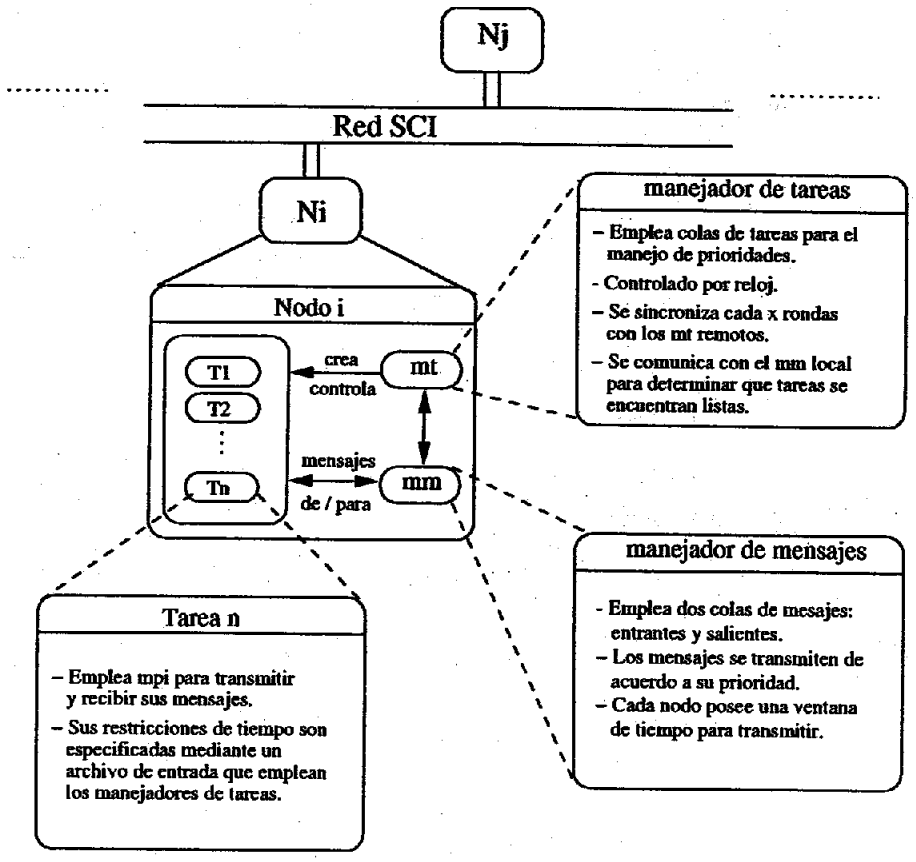


Figura 2.2: Arquitectura propuesta.

al procesador periódicamente con el objetivo de determinar cual es la siguiente tarea que debe ejecutarse. A este tiempo se le conoce como periodo del reloj o tick.

Por su parte, para transmitir los mensajes, se implementa un protocolo similar a TDMA sobre el hardware SCI, para ello se emplea una tarea que envía los mensajes durante la ventana de tiempo asignada al nodo. A lo más cada nodo posee una ventana de tiempo, haciendo uso exclusivo de ésta, es decir, si algún nodo no tiene mensajes que transmitir, este tiempo no puede ser empleado por nadie más. No obstante, la duración de la ventana puede ser diferente para cada uno de los nodos. Se llama ronda TDMA al tiempo necesario para que la secuencia en la cual los nodos transmiten se repita.

De esta forma el **manejador de tareas (mt)** será quién realice las funciones de planificación, mientras que el **manejador de mensajes (mm)** se encargará del envío de mensajes. La tabla 2.1 resume las características de estas tareas.

Por su parte en la figura 2.2 se encuentra un esquema de la arquitectura que se desea implementar. Desde el punto de vista del hardware se tiene un conjunto de nodos interconectados mediante una red SCI. En cada uno de los nodos se tiene un conjunto de tareas que puede comunicarse entre ellas o con otras en nodos remotos. Cada una de éstas tiene una prioridad única y un periodo, que puede ser cero. Además, si la tarea recibe mensajes, tiene una lista de tareas de las cuales depende.

Las tareas dentro de este sistema sólo pueden activarse cuando su periodo relativo indica que ha iniciado un nuevo periodo y, en el caso de que la tarea reciba mensajes, los mensa-

	Funciones	Formado por:
manejador de tareas (mt)	▪ determinar a la tarea con mayor prioridad. ▪ establece cuando se han satisfecho los requisitos de una tarea para que ésta pueda ejecutarse. ▪ realiza la tarea de sincronización. ▪ genera un reporte de los tiempos empleados por las tareas, el cual puede emplearse para validar que el sistema cumple con los requerimientos temporales.	tres colas: ▪ **cola de tareas listas.** Contiene las tareas que han satisfecho sus requerimientos. Se encuentra ordenada de acuerdo la prioridad de éstas, por lo tanto, la tarea en la cabeza de ella será la que se ejecute. ▪ **cola de tareas en espera.** Contiene las tareas que están en la espera de que se cumplan sus requisitos: que el tiempo actual se igual a su periodo relativo y, si la tarea recibe mensajes, que estos se encuentren disponibles. ▪ **cola de mensajes entrantes.** Es un recurso compartido entre el manejador de tareas y los manejadores de mensajes remotos, ya que es en ésta en la que los manejadores de mensajes "depositan" los destinados a este nodo, y por lo tanto, debe ser consultada por el manejador de tareas para determinar que los dependencias de éstas han sido satisfechas.
manejador de mensajes (mm)	▪ transmitir los mensajes de acuerdo en su prioridad. Para ello copia los mensajes de la cola de mensajes salientes a la cola de mensajes entrantes del nodo destino. Todas sus actividades las realiza en la ventana de tiempo asignada al nodo.	▪ **cola de mensajes salientes.** Contiene los mensajes a transmitir, divididos en paquetes y ordenados en base a su prioridad. ▪ **cola de mensajes entrantes.** Recurso compartido entre los manejadores de tareas y los manejadores de mensajes.

Tabla 2.1: Características de los manejadores.

jes de los que depende se encuentran ya disponibles en el nodo local. Es responsabilidad del manejador de tareas verificar que estas condiciones se cumplan. Para ello, cada vez se activa, revisa la cola de mensajes entrantes en busca de nuevos mensajes que hayan sido copiados por la activación de los manejadores de mensajes en los nodos remotos. Si encuentra alguno nuevo verifica si el periodo relativo de la tarea que depende de éste indica que ésta debe activarse, en caso afirmativo mueve la tarea de la cola de tareas en espera a la cola de tareas listas, por el contrario, sólo marca las dependencias de la tarea como satisfechas para que ésta sea activada a su debido tiempo.

Tanto el manejador de tareas como el manejador de mensajes dan soporte a la librería MPI. Por ejemplo, cuando una tarea desea transmitir un mensaje copia éste a una región de memoria compartida, a continuación genera una señal que es recibida por el manejador de mensajes para que se encargue de insertarlo en la cola respectiva. Todo este proceso es transparente para el usuario final, desde el punto de vista de éste lo único que tiene que hacer es llamar a la función MPI_Send() con los parámetros definidos en el estándar MPI.

Para que puedan llevar a cabo sus funciones tanto el manejador de mensajes como el manejador de tareas requieren la siguiente información:

- Los identificadores de las tareas. Pese a que estos valores pueden asignarse fácilmente se optó por que fuera el usuario quien los especificara, ya que él los necesitará, en el momento que se encuentre programando, para indicar tanto a la tarea destino como a la tarea que envía.
- La asignación de las tareas a los nodos. En el sistema que se esta desarrollando es responsabilidad del usuario determinar en que nodo se han de ejecutar las tareas, por lo tanto, es él quien debe proporcionar esta información al manejador de tareas. Para apoyar al usuario con esta actividad se ha desarrollado un análisis de planificabilidad, Capítulo 4, que determina si un conjunto de tareas, con una asignación dada, cumple con los plazos establecidos.

Para especificar toda esta información, así como las restricciones temporales de las tareas, se emplea un archivo de configuración, que posteriormente emplea el manejador de mensajes como parte de su etapa de inicialización.

2.4.4. Proceso de transmisión de los mensajes

Debido a que se requiere de la participación tanto del manejador de tareas como del manejador de mensajes durante el proceso de comunicación y estas tareas se ejecutan en intervalos preestablecidos, todas las comunicaciones soportadas por esta plataforma son asíncronas.

En la Figura 2.3 se observan las etapas involucradas en el proceso de transmitir los mensajes entre tareas ejecutándose en nodos distintos. En (1) la tarea J_1 emplea la función MPI_Send() para trasmitir un mensaje a la tarea J_2, la cual se ejecuta en el nodo 2. El mensaje se inserta, de acuerdo a su prioridad, en la cola de mensajes salientes del nodo 1. Más adelante el manejador de mensajes se activa (2) y transmite los mensajes existentes con base a su prioridad hasta que el tiempo asignado a él se agota. El mensaje, debido a que existan otros de mayor prioridad o es demasiado grande, puede requerir de varias activaciones del mm para que sea transmitido totalmente. Ahora bien, además de que el mensaje se encuentre ya en el nodo 2, se requiere que el manejador de tareas (3) en este nodo se

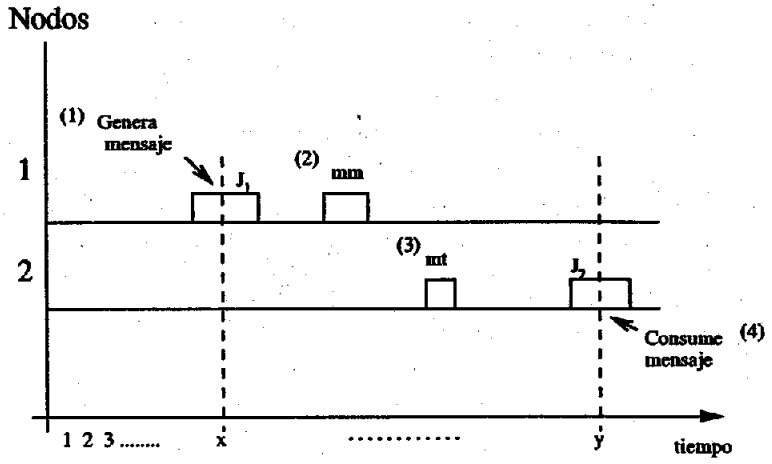


Figura 2.3: Transmisión de mensajes.

active para que se de cuenta que los requerimientos de J_2 se han satisfecho y pueda ésta ser activada. También puede suceder, dado el instante en el que se active, que otra tarea de mayor prioridad se este ejecutando por lo que J_2 tenga que esperar algún tiempo más. Finalmente en (4) la tarea J_2 emplea la función MPI_recv() para leer el mensaje que le fue transmitido. Al tiempo desde que el mensaje se genera hasta que es consumido se le conoce como tiempo de transmisión.

Ahora bien si las tareas se ejecutan en el mismo nodo, el mensaje se copia a la cola de mensajes entrantes un lugar de la de salientes por lo que ya no se requiere la participación del mm. Si embargo, aún se necesita que el mt se ejecute para que determine que los requerimientos de la tarea receptora se han cumplido.

2.4.5. Proceso completo: desde la especificación del escenario hasta la entrega de resultados

De forma similar al desarrollo de cualquier otra aplicación de *software*, el desarrollo de una aplicación de tiempo real comienza en el momento que se establecen los requerimientos del sistema, y una mala determinación de estos puede convertir al proyecto en un reto imposible de realizar.

Dentro de tiempo real, las restricciones temporales; y en el caso de sistemas con prioridades fijas, las prioridades de las tareas; que son determinadas en la etapa de análisis de los requerimientos deben ser conocidas por el calendarizador del sistema, para que éste pueda garantizar la correcta ejecución de las tareas. Como ya se ha mencionado en la plataforma propuesta se emplea un archivo de configuración con este fin. En la Figura 2.4 se observa un ejemplo de éste para un sistema formado por tres nodos, donde una sola tarea se ejecuta en cada uno de ellos. Una de las tareas recibe mensajes de las otras dos.

El archivo consta de dos tipos de declaraciones: la parte correspondiente a la configuración global y los parámetros de cada una de las tareas. Los parámetros globales se encuentran especificados dentro de la sección sistema, Figura 2.4. El orden en que estos aparecen es tomado en cuenta a la hora de hacer el análisis gramatical por lo que éste no debe alterarse. Los valores que deben proporcionarse son:

```
[Sistema]
Nummp=3.0  #número de mt
Numpus=3.0 #número de tareas
time=6000.0 #tiempo máximo de simulación

[tarea 0]
Jobid='t1' #identificador  de la tarea
Comando='./job7'#nombre del programa a ejecutar
H='n1' #identificador del nodo donde se ejecuta la tarea
T=10 #periodo
F=0.0 #fase inicial
P=63.0 #prioridad de la tarea
D='nul' #lista de dependencias
```

Figura 2.4: Ejemplo de archivo de configuración.

Nummt: número total de manejadores de tareas existentes en el sistema.

Numpus: número total de tareas.

time: tiempo de simulación o cero para indicar que el sistema funciona por siempre o hasta que ocurra un error. Este valor, al igual que el resto de los parámetros que involucren tiempo, se encuentra expresados en múltiplos del periodo del reloj. La razón de hacerlo de esta forma es que el periodo del reloj es la resolución más pequeña con la que se puede medir con exactitud en este sistema, por lo que emplear unidades como nanosegundo o microsegundos no tiene ningún sentido físico. Por defecto la duración del tick del reloj es igual a 10 milisegundos, de cualquier forma este valor pude modificarse en tiempo de compilación. En el ejemplo mostrado en la Figura, se solicita que el sistema se ejecute durante un minuto.

Por su parte, por cada una de las tareas se crea una sección tarea número (ver Figura 2.4), donde número es un valor entero en el intervalo de cero hasta el número de tareas menos uno. Nuevamente, el orden, tanto del valor numérico como de las características de las tareas, debe respetarse. Los parámetros de las tareas que deben proporcionarse son:

Jobid: identificador de la tarea. A lo más pueden emplearse cuatro caracteres para indicar este valor. Debe ser igual al empleado en las funciones de envío y recepción de mensajes.

Comando: nombre del ejecutable. Debe incluir la ruta al archivo.

H: identificador del nodo donde se ejecuta la tarea. Debe ser igual al nombre que emplea el sistema operativo como nombre de la máquina. En el caso del sistema operativo GNU/Linux este valor puede obtenerse mediante el comando hostname.

T: periodo de la tarea expresado en múltiplos del periodo del reloj.

F: fase inicial en múltiplos del reloj.

P: prioridad. Este valor debe ser único para tareas en el mismo nodo y está restringido por el tipo de dato que se emplee para almacenar este valor, por ejemplo si se utiliza un tipo que ocupe un byte a lo más pueden tenerse 255 prioridades. En este sistema entre mayor sea este número mayor será la prioridad de la tarea.

D: lista de dependencias. Este parámetro indica los mensajes que recibe la tarea o 'nul' si no recibe. Para ello se emplea el siguiente formato:

C,Jobid,Msgid,Jobid,...,Msgid

donde C corresponde al número de mensajes recibidos, Jobid es el identificador de la tarea que lo genera y Msgid es el identificador de mensaje; el cual corresponde al identificador (*tag*) usado por las funciones MPI_Send y MPI_Recv.

Una vez que este archivo es creado y asumiendo que ya se ha realizado el análisis de planificabilidad como se describe en el capítulo 4, el siguiente paso consiste en ejecutar al sistema. Para ello en cada uno de los nodos se lanza al manejador de tareas. Al finalizar éste generará un archivo con toda la información referente a los tiempos en los cuales se ejecutaron las tareas, incluyendo el instante de inicio y la duración de éstas. También incluye los tiempos en que los mensajes son generados y el instante en que son recibidos, por lo que comparando todos los archivos de salida pueden determinarse los tiempos de transmisión de los mensajes; además, puede emplearse en la etapa de pruebas para determinar el peor tiempo de ejecución de las tareas, así como para verificar que éstas cumplen con sus plazos.

En el capítulo referente al caso de estudio (capítulo 5) se puede observar un ejemplo detallado de como emplear esta arquitectura: incluyendo el código fuente para algunas de las tareas y la forma como éste debe compilarse.

2.5. Conclusiones

El avance en las capacidades de cómputo ha cambiado los paradigmas empleados en la ciencia y en la ingeniería. Tradicionalmente un científico primero observaba, a continuación desarrollaba una teoría y finalmente realizaba un conjunto de experimentos que le permitieran demostrarla. De forma similar, los ingenieros primero diseñaban (por lo general sobre papel), después construían y probaban prototipos, y finalmente realizaban la versión comercial. Ahora, es posible realizar complejas simulaciones y construir prototipos empleando sistemas de cómputo.

Los sistemas de cómputo distribuidos han demostrado ser una alternativa viable cuando se requiere tanto capacidad de procesamiento como de almacenamiento. Sin embargo, dotar a estos sistemas con soporte para aplicaciones de tiempo real sigue siendo un tema abierto. En este capítulo se describe una plataforma que emplea el paradigma de envío de mensajes para este fin. Una de las características de ésta es que trata a los mensajes como tareas que hay que "ejecutar" dentro del canal de comunicación, lo que permite acotar los tiempos de transmisión de éstas.

El sistema propuesto está formado por dos tareas: el manejador de tareas que se encarga de las funciones de planificación de las tareas y del manejador de mensajes que hace lo

propio para estos. En capítulos siguientes se harán descripción más detalladas de cada una de ellas.

Capítulo 3

Implementación

3.1. Introducción

Las aplicaciones de tiempo real están fuertemente acopladas con el ambiente que están controlando y/o monitoreando. Esto aprovecha al máximo las características del sistema para garantizar la correctez temporal de las tareas. En este caso se utiliza el soporte que brinda el sistema operativo GNU/Linux para realizar la comunicación entre tareas: en particular se emplean los métodos de memoria compartida y el envío de señales.

Como su nombre lo indica, cuando dos tareas emplean memoria compartida para comunicarse entre sí crean una región donde ambas tareas pueden leer y escribir datos. Esta es la técnica que las tareas emplean para transmitir los mensajes de o para el manejador de mensajes local. A su vez, se puede considerar a una señal como una interrupción que una tarea genera para llamar la atención de otra respecto a un evento. En particular se emplean señales de tiempo real: las cuales se diferencia de las otras en que son acumulativas, es decir, si mientras se está ejecutando la rutina que atiende a la señal ocurre otra del mismo tipo ésta se almacena para que en el instante que la rutina termine sea atendida.

Por otro lado, en lugar de describir el código fuente de la aplicación (lo cual sería tedioso y en algunas partes difícil de leer) para indicar la forma como se realizó la implementación de la plataforma se hace una descripción funcional, lo más detalladamente posible, de los requerimientos programados para cumplir con el objetivo propuesto: crear una plataforma que permita el desarrollo de aplicaciones paralelas en tiempo real en un *cluster* de alto desempeño.

En este capítulo, sección 3.2, se da un simple ejemplo de como mediante el estándar MPI puede crearse una aplicación paralela, donde una de las tareas recibe los mensajes generados por las otras. A continuación, sección 3.3, se detalla cuales son las limitantes del estándar MPI para transmitir mensajes en tiempo real, así mismo, se proponen algunas modificaciones encaminadas a subsanar este problema. La sección 3.4 describe las características del reloj en el cual esta basado el sistema. Por su parte la forma en como se implementan los manejadores se describe en la sección 3.5. Las características de la red SCI son mostradas en la sección 3.6 y en la sección 3.7 se encuentra el algoritmo de sincronización empleado. Finalmente, la sección 3.8 presenta las conclusiones del capítulo.

```
#include ''mpi.h''
      .
      .
      .
main(int argc,char *argv[]){
      .
      .
      .
   /*Ninguna función MPI debe aparecer
     antes de ésta*/
   MPI_Init(&argc,&argv);
      .
      .
      .
   MPI_Finalize();
   /*Ninguna función MPI debe aparecer
     después de ésta que */
      .
      .
      .
}
```

Figura 3.1: Estructura general de los programas que emplean MPI.

3.2. Usando MPI

El grupo que diseñó MPI en lugar de especificar un nuevo lenguaje (y por lo tanto un nuevo compilador), definió un conjunto de funciones que podrían utilizarse en programas escritos tanto en C como en Fortran. De esta manera para realizar una aplicación paralelo sólo se requiere aprender un conjunto de definiciones y de funciones. En la práctica, aunque MPI es un sistema complejo y polifacético, la mayor parte de los problemas se resuelven usando sólo seis de sus funciones.

Todo programa realizado en MPI debe incluir la sentencia:

```
#include "mpi.h"
```

este archivo, mpi.h, contiene las definiciones y declaraciones necesarias para poder compilar un programa que use MPI. Todos los identificadores de MPI empiezan con la palabra "MPI_" seguida por el nombre del tipo que representan en mayúsculas (e.g. MPI_CHAR), por su parte las funciones, emplean sólo la letra inicial en mayúsculas (e.g. MPI_Send).

La primera función de toda tarea escrita empleando el estándar MPI es la de inicializar dicho ambiente, para ello se emplea la función MPI_Init(), cuyos parámetros son punteros a los de la función principal. Al finalizar se debe emplear a la función MPI_Finalize() para liberar los recursos empleados por MPI. Por lo tanto, un programa típico de MPI tiene la estructura mostrada en la Figura 3.1.

```
int MPI_Send(
        void*           mensaje     /*in*/,
        int             tamaño      /*in*/,
        MPI_Datatype    tipodato    /*in*/,
        int             destino     /*in*/,
        int             etiqueta    /*in*/,
        MPI_Comm        comm        /*in*/)

int MPI_Recv(
        void*           mensaje     /*out*/,
        int             tamaño      /*in*/,
        MPI_Datatype    tipodato    /*in*/,
        int             origen      /*in*/,
        int             etiqueta    /*in*/,
        MPI_Comm        comm        /*in*/,
        MPI_Status*     estado      /*out*/)
```

Figura 3.2: Definición de las funciones: MPI_Send y MPI_Recv.

Ahora bien, en MPI las tareas se encuentran agrupadas en conjuntos llamados *communicators*. Sólo las tareas pertenecientes al mismo *communicator* pueden enviarse mensajes entre ellas. Por defecto todas las tareas pertenecen a MPI_COMM_WORLD[1] y por lo general no se requiere crear ningún otro. Los identificadores de las tareas son asignados por MPI y van de 0 hasta uno menos el tamaño del *communicator* al cual pertenece. Para determinar al identificador de la tarea se dispone de la función MPI_Comm_rank(), por su parte la cantidad de tareas puede determinarse mediante MPI_Comm_size().

Finalmente, para transmitir y recibir los mensajes se dispone, entre otras, de las funciones MPI_Send() y MPI_Recv(). La sintaxis de ambas funciones se observa en la Figura 3.2. El contenido de los mensajes se almacena en la dirección de memoria indicada por la variable **mensaje**. Los dos siguientes parámetros, **tamaño** y **tipodato**, son empleados por el sistema para determinar cuanta memoria es necesaria para almacenar los mensajes. El MPI_Datatype no necesariamente corresponde a los tipos de datos empleados por C: la razón de emplear este tipo de dato es para facilitar la portabilidad entre distintas plataformas. Los tipos de datos definidos en MPI y su correspondiente en C (si existe) aparecen en la Tabla 3.1.

Los parámetros **destino** y **origen** son, respectivamente, los identificadores de la tarea receptora y de la tarea que envía el mensaje. El parámetro **etiqueta** es un entero que identifica al mensaje, ésta se emplea además para garantizar el orden correcto en que deben recibirse los mensajes. El *communicator* al cual pertenecen los mensajes se especifica mediante el parámetro **comm**.

La función MPI_Recv() tiene un parámetro adicional el cual regresa información del mensaje que se ha recibido. Éste referencia a una estructura que contiene al menos tres miem-

[1]Esto sólo es valido en el estándar MPI 1.1 cuando todas las tareas se creaban al inicio. En el estándar MPI 2.0 es posible crear tareas dinámicamente, para ello primero se debe crear un *communicator* al cual pertenecerán las nuevas tareas.

Tipo de dato MPI	Tipo de dato C
MPI_CHAR	char
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Tabla 3.1: Tipos de datos definidos en MPI.

bros: el identificador de la tarea que lo envía, la etiqueta del mensaje y un código de error.

Un ejemplo completo del uso de estas funciones aparece en la Figura 3.3. En él la tarea marcada con el identificador 0 recibe los mensajes generados por las otras. Los detalles de como compilar y ejecutar este programa dependen del sistema que se éste usando. Sin embargo, compilarlo puede ser tan simple como

```
$ cc -o holamundo holamundo.c -lmpi
```

Cuando el programa esté compilado y sea ejecutado con cuatro tareas, la salida deberá ser algo similar a

```
Hola mundo desde 1
Hola mundo desde 2
Hola mundo desde 3
```

En el ejemplo, se puede observar una característica fundamentales de los programas realizados en MPI: emplean el paradigma un único-programa múltiples-datos (*single-program multiple-data, SPMD*) . Esto se debe a que es el sistema quien asigna el identificador de las tareas, por lo que éste sólo se conoce en el momento de ejecución, por lo tanto, es necesario emplear una estructura if o un switch para ejecutar el código especifico de cada una de las tareas.

3.3. Enviando mensajes en tiempo real

El estándar MPI ofrece, hasta el momento, poca o ninguna facilidad para desarrollar aplicaciones con restricciones temporales. Esta falta es evidente desde el momento que se lanzan las tareas: la segunda versión del estándar MPI sugiere contar con un programa de la forma:

```
mpiexec -n <numtarea> <programa>
```

```
1  #include <stdio.h>
2  #include <string.h>
3  #include "mpi.h"

4  int main(int argc, char *argv[])
5  {
6    int tarea_id; /*identificador de la tarea*/
7    int t;        /*numero total de tareas*/
8    int origen;   /*tare que envía el mensaje*/
9    int dest;     /*tarea que recibe el mensaje*/
10   int tag = 0;      /*identificador del mensaje*/
11   char mensaje[100]; /*mensaje*/
12   MPI_Status estado; /*estado regresado al*/
13                      /*recibir el mensaje*/
14   /*Inicialización de MPI*/
15   MPI_Init(&argc, &argv);
16   /*Se determina el identificador de la tarea*/
17   MPI_Comm_rank(MPI_COMM_WORLD,&tarea_id);
18   /*Se determina el número de tareas*/
19   MPI_Comm_size(MPI_COMM_WORLD,&t);
20   if (tarea_id != 0)
21     {
22       /*Se crea el mensaje*/
23       sprintf(mensaje,"Hola mundo desde %d",tarea_id);

24       /*La tarea destino es aquella que tenga el*/
25       /*identificador igual a cero*/
26       dest = 0;

27       /*Se transmite el mensaje*/
28       MPI_Send(mensaje,strlen(mensaje)+1,MPI_CHAR,
29                dest,tag,MPI_COMM_WORLD);
30     }
31   else
32     {
33       /*La tarea 0 recibe los mensajes de las demás*/
34       for (origen = 1;origen < p; origen++)
35         {
36           MPI_Recv(mensaje,100,MPI_CHAR,origen,tag,
37                    MPI_COMM_WORLD,&status);
38           printf("%s\n",mensaje);

39         }
40     }
41   /*MPI Termina*/
42   MPI_Finalize();
43   return 0 ;
```

Figura 3.3: Ejemplo de uso de mpi. La tarea 0 recibe los mensajes generados por el resto de las tareas.

el cual debe lanzar concurrentemente numtarea copias de programa. Pero no hace ninguna recomendación acerca de la forma como deben asignarse las tareas a los nodos, ni cual es el proceso mediante el cual se le asigna el identificador a las tareas. La implementación mpich emplea el método *round-robin* para asignar las tareas a los nodos e incrementalmente va asignando los identificadores a las tareas.

Ambos métodos resultan incompatibles cuando se requiere tener más control acerca de las tareas, por ejemplo, puede suceder que una tarea deba ejecutarse en un nodo porque éste tiene alguno de los recursos de los cuales depende. Por tal motivo se emplea un archivo de configuración para indicar en que nodo se deben ejecutar las tareas. De esta forma, la manera de inicializar al sistema sería:

```
mpiexec -configfile archivo
```

donde archivo sigue el formato especificado en la sección 2.4.5 del capítulo 2. La principal diferencia con el estándar MPI será entonces que **las tareas no se ejecutarán concurrentemente sino que dependerá de las características temporales de éstas el instante cuando sean ejecutadas.**

Por su parte los mensajes deberán ser transmitidos de acuerdo a su prioridad. Para evitar que un mensaje de gran tamaño pero de una prioridad baja bloque la transmisión de uno de mayor, todos estarán divididos en un número entero de paquetes: de esta manera se puede suspender el envío de un mensaje entre paquetes.

Cada uno de los paquetes tiene el formato indicado por la Figura 3.4. En él los primeros 8 bytes constituyen la cabecera del paquete: los primeros 5 constituyen al identificador del mensaje. Por su parte los restantes 1016 bytes contienen la información que se desea transmitir. Cabe señalar que todos los paquetes son del mismo tamaño: si la información del mensaje no es suficiente para "llenar" al último de ellos, éste se rellena con espacios en blanco hasta alcanzar el tamaño fijado. Por otra parte debido al tamaño de la cabecera, todos los datos que ésta contiene están limitados a 1 byte (solo pueden tomar valores de 0 a 255). Los campos almacenados son:

no: identificador del nodo origen.

nd: identificador del nodo destino.

to: identificador de la tarea origen.

td: identificador de la tarea destino.

et: etiqueta del mensaje. Este parámetro corresponde al especificado por el estándar MPI.

pr: prioridad del mensaje.

pt: total de paquetes en que se divide el mensaje.

pa: identificador del paquete actual.

Para cumplir con el objetivo de contar con una plataforma que permita la transmisión de mensajes en tiempo real, es suficiente con implementar el subconjunto formado por las seis funciones del estándar MPI discutidas anteriormente. Para ello se emplea la cadena

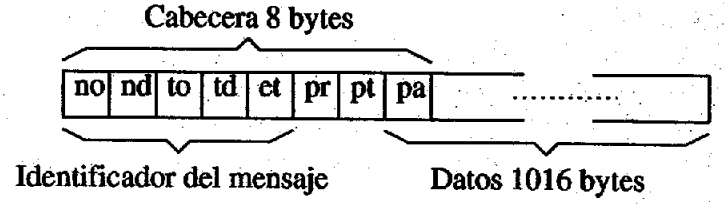


Figura 3.4: Formato de los paquetes.

RTMPI_ en lugar de MPI_ como prefijo de los nombres de las funciones, no obstante, en la medida de lo posible, se conservan los mismos parámetros y realizan la misma función. Las características de éstas funciones y su interrelación con el manejador de mensajes y/o el manejador de tareas se describe a continuación.

3.3.1. RTMPI_Send

La función RTMPI_Send() se emplea para transmitir los mensajes: como ya se a mencionado todas las comunicaciones son asíncronas. A su vez, para evitar que los datos se corrompan, durante el tiempo que estén ejecutado esta función las tareas no deben suspenderse.

Los procesos que debe realizar son:

- Copiar el mensaje a un segmento de memoria compartida. Antes de copiarse el mensaje se divide en paquetes: a cada uno de ellos se les agrega la cabecera antes indicada.
- Indicar al manejador de tareas que se desea transmitir un mensaje. Para ello envía una señal de tiempo real al manejador de tareas.

Por otro lado, cuando el manejador de tareas recibe una señal indicando que existe un nuevo mensaje, copia éste a la cola respectiva. Si el identificador del mensaje indica que es para un nodo remoto, se inserta en la cola de mensajes salientes, si por el contrario es para una tarea que se encuentra en el nodo local se copia en la lista de mensajes entrantes.

El prototipo de esta función es

```
int RTMPI_Send(
      void *            mensaje
      int               tamaño
      MPI_Datatype      tipodato
      char *            destino
      char              etiqueta
      char              prioridad)
```

Los primeros tres parámetros son equivalentes a los del estándar MPI, Figura 3.2. La variable **destino** corresponde al identificador dado por el usuario a la tarea en el archivo de configuración. Por su parte, **etiqueta** se emplea como identificador del mensaje. A su vez, se eliminó el campo referente al *communicator* y, en su lugar, se agregó el de la prioridad del mensaje.

3.3.2. RTMPI_Recv

Debido a que los mensajes son iniciados hasta que sus dependencias han sido satisfechas, el único proceso que debe realizar la función RTMPI_Recv es copiar de una dirección de memoria compartida al mensaje, obviamente éste se encontrará dividido en paquetes por lo que será necesario rensamblarlo.

El prototipo de esta función se encuentra definido como.

```
int RTMPI_Recv(
     void *          mensaje
     int             tamaño
     MPI_Datatype    tipodato
     char *          origen
     char            etiqueta
     MPI_Status *    estado)
```

Los primeros seis parámetros corresponden a los de la función RTMPI_Send. Por su parte el parámetro **estado** corresponde al definido por el estándar MPI. Al igual que en la función RTMPI_Send, el campo correspondiente al *communicator* fue eliminado, pero en esta ocasión no se agrega nada en su lugar.

Cabe destacar el caso en que se tiene una tarea que recibe múltiples instancias del mismo mensaje, es decir, existe una tarea que dentro de un ciclo este continuamente enviando un mensaje con la misma etiqueta pero con datos distintos. Por su parte la tarea (o las tareas) que recibe los mensajes emplea otro ciclo para procesarlos. En este caso la tarea es lanzada cuando se recibe al primer mensaje, en futuras llamadas a RTMPI_Recv(), si el mensaje aún no se encuentra disponible, la tarea es suspendida hasta que éste arribe.

3.3.3. RTMPI_Init

Esta función se emplea para pasarle a la nueva tarea la información necesaria para que puede llevar a cabo la transmisión de los mensajes. Dicha información incluye la dirección de memoria donde debe copiar los mensajes que desea transmitir, así como, si la tarea recibe mensajes, la región de memoria donde se encuentran estos.

A su vez, para poder generar las cabeceras de los paquetes la tarea debe conocer los identificadores del resto de las tareas, incluyendo el suyo propio y la asignación de las tareas a los nodos.

Para pasarle toda esta información a las nueva tareas, el manejador de mensajes, en el instante que las crea les agrega a su argumento de entrada una dirección de memoria donde se encuentran disponibles estos datos. Por lo tanto la función RTMPI_Init tiene como objetivo crear las estructuras de datos necesarios para copiar los datos de esa región.

Otra función realizada por RTMPI_Init() es hacer a la tarea *preemptive:* para ello se emplea dos señales de tiempo real definidas como RTMPI_STOP y RTMPI_CONTINUA. Para la primera RTMPI_Init() instala una rutina cuyo propósito es suspender a la tarea hasta que la señal RTMPI_CONTINUA sea recibida.

El prototipo de RTMPI_Init() es

```
void RTMPI_Init(
        int *       argc
        char ***    argv)
```

Ambos argumentos son apuntadores a los de la función principal de la tarea, ver Figura 3.3

3.3.4. RTMPI_Finalize

El objetivo de la función RTMPI_Finalize es liberar los recursos que hayan sido empleados en la transmisión de los mensajes. Por el momento estos son liberados ya sea por RTMPI_Send o por RTMPI_Recv, por lo que no se necesita realizar nada más en RTMPI_Finalize.

Si embargo se recomienda su uso de la forma que indica el estándar, esto porque en futuras implementaciones podría realizarse alguna actividad dentro del ámbito de esta función.

El prototipo de esta función es simplemente

```
void RTMPI_Finalize()
```

3.3.5. RTMPI_Comm_rank

La función RTMPI_Comm_rank se emplea para obtener al identificador de la tarea actual. En el caso de esta implementación esta información se obtiene de las estructuras de datos creadas por la función RTMPI_Init.

El prototipo de esta función es

```
void RTMPI_Comm_rank(
            char  *tidentificador)
```

3.3.6. RTMPI_Comm_size

El objetivo de la función RTMPI_Comm_size() es determinar el número de tareas que forman al sistema. Al igual que en RTMPI_Comm_rank() esta información se obtiene de los datos iniciales pasados a la tarea. Su prototipo es

```
void RTMPI_Comm_size(
            int    *numtareas)
```

3.4. Reloj del sistema

Debido a que el sistema es controlado por reloj se empezará por describir el soporte que para éste provee el sistema operativo GNU/Linux .No obstante que a partir de la versión 2.2 del kernel de Linux se cuenta que la función nanosleep() del estándar POSIX.4 [POSIX],

su implementación se basa en el esquema empleado por el el kernel, el cual tiene una resolución de 1 Hz (10 milisegundos).

Por lo tanto, aunque nanosleep() suspenda al proceso por menos de 10 milisegundos puede tomar este tiempo hasta que la tarea se ejecute de nuevo. De ahí, que no se considere una buena opción utilizar este tiempo como base de nuestro sistema.

Por otra parte, todas las computadores personales cuentan con un reloj dentro de ellas. Este reloj es el que mantiene la fecha y la hora de la computadora mientras ésta se encuentra apagada. Por lo general se emplea un chip Motorola MC146818 (o algún clon) con este fin. Además, este dispositivo puede emplearse para generar señales desde 2Hz hasta 8192Hz, en incrementos de las potencias de dos. Estas señales son reportadas al procesador mediante la interrupción número 8 (IRQ 8).

En Linux se puede acceder a este dispositivo mediante /dev/rtc: una tarea puede monitorizar las interrupciones generadas por éste mediante las funciones read(...)[2] o select(...)[3] sobre /dev/rtc. Estas funciones suspenden a las tarea que las llama hasta que la próxima interrupción sea generada.

Para programar la frecuencia a la que ha de trabajar se emplea la función ioctl. Ahora bien, pese a todos las variaciones que puedan existir, un uso básico de este reloj involucra los siguientes pasos.

- Abrir al dispositivo para lectura: para ello puede emplearse la función open() de la siguiente manera.

```
fd = open ("/dev/rtc", O_RDONLY);
```

- Establecer la frecuencia: se emplea la función ioctl() de la forma como se indica a continuación; donde el valor de **freq** es una potencia de dos entre 2 y 8192.

```
ioctl(fd, RTC_IRQP_SET, freq)
```

- Habilitar las interrupciones: nuevamente se emplea la función ioctl().

```
ioctl(fd, RTC_PIE_ON, 0);
```

- Esperar a que ocurra alguna interrupción: para ello se emplea la función read(); donde la variable **data** contendrá el número de interrupciones ocurridas que no fueron atendidas. Por lo general dicho valor debe ser igual a cero, si éste no es el caso significa que la frecuencia a la que se están procesando las interrupciones no es la adecuada.

```
read(fd, &data, sizeof(unsigned long));
```

[2]ssize_t read(int fd, void *buf, size_t count). La función read() lee a lo más count bytes del archivo indicado por el descriptor fd. Los datos leídos se almacenan en la región indicada por buf. Regresa el número de bytes leídos.

[3]int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout). La función select() espera hasta que un número de descriptores cambie de estado. Es decir, permite saber si un archivo ha sido modificado.

- Finalmente, antes de que el programa termine, el dispositivo debe "cerrarse" , para permitir que otra tarea pueda hacer uso de éste. Para ello primero se deshabilitan las interrupciones y a continuación se emplea la función close().

```
ioctl(fd, RTC_PIE_OFF, 0);
close(fd);
```

Por sus características se emplea este reloj a una frecuencia de 1024 Hz (aproximadamente 1 milisegundo) como la base del sistema. No obstante para determinar el tiempo empleado, ya sea por las tareas o por la transmisión de los mensajes, se emplea la función gettimeofday() que tiene una resolución de microsegundos.

3.5. Implementación de los manejadores

En lugar de implementar al manejador de tareas y al manejador de mensajes como dos tareas independientes (o como dos hilos) se hace como un sólo programa, esto con el fin de minimizar el número de cambios de contexto, al mismo tiempo que se mantiene al mínimo el número de señales generadas.

En términos generales, obviando la etapa de inicialización, el manejador de tareas se ejecuta en rondas, cuya duración es igual al periodo de éste. En la Figura 3.5 se observa la estructura interna del mt, la cual esta formada básicamente por un ciclo (Línea 2) que termina cuando se han ejecutado **maxrondas**, cabe destacar que si este valor es negativo el sistema se ejecuta por siempre o hasta que ocurra un error.

Cada **sinronda**, Líneas 3-5, los mt se sincronizan entre sí. El algoritmo que se emplea para tal fin es tratado en la sección 3.7. A continuación el manejador de tareas determina cuales están listas para ser ejecutadas, Línea 6. Para ello emplea dos colas: una con las tareas listas y otra con las tareas que están a la espera de que se satisfagan sus requerimientos. Tales requerimientos son de dos tipos: que el instante de tiempo actual sea igual a su periodo relativo y en el caso de que la tarea reciba mensajes que estos ya se encuentren disponibles localmente. Todas las tareas con sus requisitos cumplidos son insertadas en la cola de tareas listas de acuerdo a su prioridad: a mayor valor mayor prioridad.

Una vez que se ha actualizado la cola de tareas listas, se establece la alarma para que interrumpa al manejador de tareas en el instante que debe activarse al manejador de mensajes, Línea 7. Hasta que esto ocurra, el mt ejecuta las tareas en la cola de listas de acuerdo su prioridad. Si, cuando la alarma se active, se esta ejecutando una tarea, esta se suspende. Para implementar la alarma se emplea un hilo que hace uso del reloj antes descrito.

Finalmente, se continua con la ejecución de las tareas hasta que la alarma "suene" nuevamente, indicando ahora que la ronda ha concluido. Por defecto, la duración de la ronda es 10 milisegundos.

Por su parte el manejador de mensajes se implementa mediante una función la cual utiliza el soporte de la red SCI para transmitir los mensajes.

3.6. Características de la red SCI

El comité ANSI/IEEE 1592-1992 diseñó una plataforma tanto de software como de hardware capaz de soportar comunicaciones en tiempo real con una baja latencia. El resultado

```
1  ronda := 0
2  while ronda < maxrondas do
3        if ronda modulo sinronda = 0
4          call sincroniza(S, lnodo, ronda)
5        fi
6        call tareas_listas()
7        call establecer_alarma_mm()
8        call ejecutar_tareas_hasta_mm()
9        if existen_mensajes
10         call manejador_mensajes()
11       fi
13       call establecer_alarma_tiempo_restante()
14       call ejecutar_tareas_hasta_alarma()
15       ronda := ronda + 1
16 done
```

Figura 3.5: Estructura general de los manejadores.

de su trabajo se convirtió en el estándar SCI (*Scalable Coherent Interface*). Su arquitectura de dos o tres dimensiones ofrece redundancia en caso que se presenta alguna falla.

Cuando se transfiere un dato en SCI éste se copia directamente a la memoria del nodo remoto[4]: empleando el método conocido como transferencia inteligente. En dicha técnica la tarjeta SCI crea un mapa de direcciones de memoria que corresponden a una región reservada en algún nodo remoto, de esta forma el procesador puede acceder a estos datos de la misma forma que accede a los de la memoria local.

Estas características contribuyen a que la red SCI tenga una baja latencia en la transmisión de los datos. Por ejemplo[5], empleado una tarjeta D330 PCI-SCI, de la empresa *Dolphin Interconnect Solutions Inc.*[6] , en un computador con un procesador AMD 760 MPX en promedio se requieren 3 microsegundos para transferir un paquete de 512 bytes. La Figura 3.6 muestra el tiempo promedio desde que el dato se envía hasta que es recibido en la memoria remota.

Ahora bien, debido a que la red SCI no proporciona ningún protocolo para arbitrar el acceso a ésta, los manejadores de mensaje en cada uno de los nodos se ejecutan en intervalos de tiempos diferentes: logrando con ello eliminar cualquier competencia que podría existir a la hora de acceder a la red.

En general, los pasos necesarios para utilizar la red SCI son:

- En ambos nodos:
  - Inicializar el ambiente SCI. Al igual que en MPI, antes de emplear las funciones soportadas por SCI es necesario inicializar su ambiente, para ello se emplea la función SCIInitialize(). Ninguna función de SCI puede ser llamada antes de ésta.

[4]modelo de acceso a memoria remota (*Remote Memory Access*, RMA)
[5]Ejemplo tomado de [Kohmann,2004]
[6]http://www.dolphinics.com/

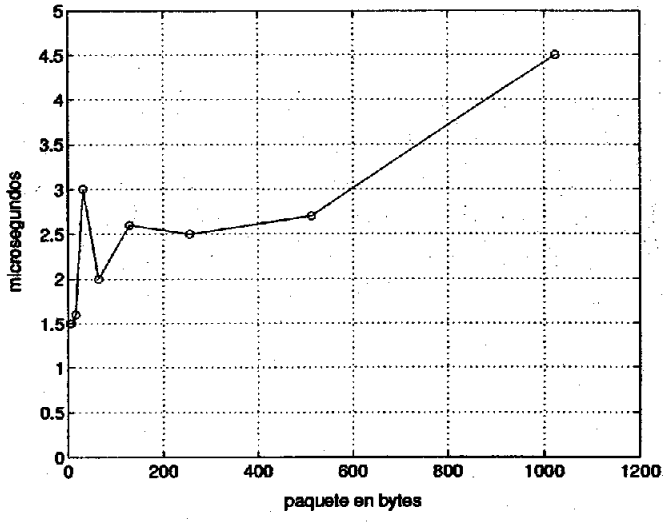


Figura 3.6: Tiempo promedio empleado en la transmisión de datos[7].

- Crear un dispositivo virtual. SCI utiliza un descriptor, al que llama dispositivo virtual, para referirse al canal de comunicación que se crea entre el hardware (la tarjeta SCI) y el software (el programa con funciones SCI). Un mismo dispositivo virtual puede utilizarse para comunicarse entre varios nodos. Para crearlo se emplea la función SCIOpen(&v_dev,...) donde v_dev es un puntero a éste. Al finalizar de usar el canal este debe destruirse usando la función SCIClose(&v_dev,...)
- Liberar los recursos empleados por SCI: para ello se emplea la función SCITermintate(). Ninguna función de SCI puede ser llamada después de ésta.

- En el nodo remoto:

  - Reservar al segmento de memoria. La asignación de un segmento de memoria se hace mediante la función SCICreateSegment(). La principal razón para contar con una función especifica para realizar esta tarea es que se requiere que la memoria empleada por la tarjeta SCI sea contigua además que siempre debe estar disponible, es decir, los contenidos de esta región nunca debe ser puesto en la memoria de intercambio.
  - Mapear el segmento de memoria al espacio de direcciones de la tarea. Para poder emplear localmente la memoria reservada en el paso anterior es necesario conocer la dirección donde ésta se encuentra: para ello se emplea la función SCIMapLocalSegment().
  - Hacer disponible el segmento de memoria a todos los nodos conectados a la red SCI. Una vez que el segmento se encuentra disponible localmente se requiere hacerlo visible para el restos de los nodos. El proceso de exportación se realiza mediante el empleo de dos funciones que deben ser llamadas secuencialmente. La primera de ellas, SCIPrepareSegment(), tiene como función mapear la dirección de memoria dentro del espacio de direcciones de 64-bits compartido por

todos los nodos dentro de la red SCI. Por su parte la función, SCISetSegmentAvailable(), hace al segmento visible para el resto de los nodos.

- Liberar la memoria. Una vez que el proceso de transmisión ha concluido, los recursos empleados por SCI deben liberarse, para ello, se deben realizar de forma secuencial los siguientes pasos: primero se inhabilita para el resto de los nodos la región de memoria empleando la función SCISetSegmentUnavailable(); a continuación, mediante las funciones SCIUnmapSegment() y SCIRemoveSegment() se libera la memoria reservada.

- En el nodo local:

  - Conectarse al segmento remoto. Lo primero que debe hacer una aplicación remota para poder utilizar un segmento remoto de memoria es conectarse a él. Dicha conexión consiste en determinar la dirección y tamaño del segmento dentro del espacio de direcciones de SCI. Esto se realiza mediante la función SCIConnectSegment().
  - Mapear el segmento de memoria al espacio de direcciones de la tarea. Al igual que de forma local, la dirección del segmento debe estar disponible dentro del espacio de direcciones destinado a la aplicación antes de poder hacer uso de ésta. La función SCIMapRemoteSegment() es empleada con este fin.
  - Liberar la memoria. Nuevamente, una vez que ya no se necesitan más, se requiere liberar los recursos empleados, esto se realiza en dos pasos. Primero se desmapea el segmento mediante la función SCIUnmapSegment() y a continuación se desconecta del nodo mediante SCIDisconnectSegment().

Mientras que el segmento de memoria se encuentre mapeado en ambas tareas (local y remota), es posible acceder a éste como si se tratase de uno obtenido mediante la función malloc(...)[8], es decir, se pueden emplear las operaciones definidas para punteros para leer o escribir en él: inclusive se puede emplear funciones como memcpy(...)[9]. Por lo tanto las operaciones del manejador de mensajes se reducen a copiar mensajes de la cola de mensajes salientes a la de entrantes del nodo remoto.

3.7. Sincronización

Para poder realizar sus funciones tanto los manejadores de las tareas como los manejadores de mensajes en todos los nodos que conforman al sistema deben estar de acuerdo con relación al tiempo actual. Con tal fin los manejadores de tareas periódicamente se sincronizan unos a otros mediante el empleo de una barrera.

Emplear una barrera, en lugar de usar el tiempo de alguno de ellos o de algún otro sistema como una fuente confiable, es mas apropiado para el sistema que se esta desarrollando debido a que, además de ser un sistema homogéneo (todos los nodos tiene características similares), se basa en rondas: por lo que es suficiente que los nodos estén de acuerdo en el

[8] void *malloc(size_t size); La función malloc() reserva size bytes y regresa un apuntador a dicha región.

[9] void *memcpy(void *dest, const void *src, size_t n); la función memcpy() copia n bytes de la dirección indicada por src a la región dest.

```
1  Variables en memoria compartida: enlabarrera_lnodo
2  funct syncroniza(S, lnodo, ronda)
3     N := { nodo : nodo ∈ S ∧ enlabarrera_nodo ¬ ronda}
4     enlabarrera_lnodo := ronda
5     while N ¬ φ do
6           foreach nodo ∈ N
7              if enlabarrera_nodo = ronda
8                 saca nodo de N
9              fi
10          done
11    done
```

Figura 3.7: Algoritmo de sincronización.

instante en que inicia la ronda. La principal desventaja de esta técnica es que podría suceder que uno de los nodos se "retrasara" más que los demás en llegar a la barrera en una ronda y en otra llegara más rápido, por lo que desde el punto de vista de un observador externo existirían variaciones en la duración de las rondas. Para evitar que éstas sean excesivas, en lugar de efectuar la sincronización cada rondas ésta se hace cada "x" rondas.

Para implementar el algoritmo se emplean el soporte para acceso a memoria remota con que cuenta la red SCI. En él cada uno de los nodos tiene una región de memoria que emplea para indicar que el nodo ha alcanzado la barrera (**enlabarrera**), los otros nodos pueden "ver" ésta y así determinar cuando todos han llegado al mismo punto. Los detalles del algoritmo pueden apreciarse en la Figura 3.7. Como parámetros de entrada el algoritmo, Línea 1, recibe: un conjunto formado por todos los identificadores de los nodos que conforman al sistema exceptuando al nodo local, representado por la variable S; el identificador del propio nodo, **lnodo**; y el número de ronda que se supone debe iniciar, **ronda**. Como primera tarea el algoritmo determinada al conjunto N, Línea 2, formado por los nodos que aún no llegan a la barrera. A continuación indica al resto de los nodos que la ha alcanzado, Línea 3. Finalmente entra en un ciclo donde espera al resto de los nodos, Líneas 4 - 10.

Para el diseño del algoritmo, y del propio sistema, se considera que los nodos no presentan ningún tipo de fallas. Por su parte éste debe garantizar que todos los nodos salen de la barrera prácticamente al mismo tiempo. Esto puede demostrarse considerando el caso del último nodo que llega a ella: para él el conjunto N estará vació, mientras que el resto de los nodos sólo estarán esperando por éste, por lo que en el instante que iguale su bandera a la ronda actual todos los nodos saldrán de la barrera.

La sincronización de los nodos permite implementar el protocolo TDMA sobre la red SCI para controlar el acceso a ésta. De esta manera, cada nodo cuanta con una venta exclusiva para transmitir sus mensajes. El inicio de cada una de las ventanas ocurre al instante Δ_{η_i} después que ha ocurrido la barrera. Δ_{η_i}es el desplazamiento temporal que sufren los manejadores de mensajes con el fin de evitar cualquier competencia por acceder al medio de comunicación y puede obtenerse mediante:

$$\Delta_{\eta_i} = iC_m \tag{3.1}$$

3.8. Conclusiones

Durante el presente capítulo se analizaron los requerimientos funcionales de la plataforma que se está desarrollando, y a su vez se establecieron algunas directivas de la forma en que estos pueden satisfacerse. Uno de ellos, el tiempo, quedó establecido por la señal generada por el reloj físico de los equipos de cómputo. Por su parte una barrera se emplea para realizar la sincronización entre los distintos nodos.

Para disminuir la latencia generada por la intervención del sistema operativo, se limitó el número de llamadas a éste. Por tal motivo en lugar de implementar al manejador de tareas y al de mensajes como programas separados (o hilos) se hace como uno sólo. Además como sólo una tarea puede estar activa a la vez, es posible utilizar una sola región de memoria compartida (de tamaño suficiente) para copiar los mensajes que las tareas deseen transmitir.

Capítulo 4

Análisis de planificabilidad

4.1. Introducción

Planificar un conjunto de tareas involucra determinar cuando ejecutar cada una de ellas; y en el caso de sistemas con múltiples procesadores o en sistemas distribuidos, además, se debe asignar cada tarea a un procesador en específico. El principal objetivo del calendarizador en sistemas que no son de tiempo real es maximizar el rendimiento promedio (número de tareas completadas por unidad de tiempo); mientras que en el caso de sistemas de tiempo real, el objetivo es cumplir con los plazos de cada una de las tareas asegurando que éstas terminen antes de que estos expiren. Estos plazos son impuestos por las restricciones temporales de la aplicación.

El análisis de planificabilidad implica determinar si un conjunto de tareas satisface sus restricciones temporales empleando un esquema específico de planificación. La planificabilidad en un sistema distribuido considerando tanto a los mensajes como a las tareas recibió considerable atención durante la década pasada, destacando el trabajo realizado por Tindell y Clark.

En [Tindell y Clark, 1994] los autores calculan el peor tiempo de ejecución para un sistema distribuido formado por un conjunto de nodos conectados mediante una red con soporte para TDMA. Las tareas en dicho sistema tienen prioridades fijas y las tareas con dependencias se activan con la recepción de los mensajes de los cuales depende: lo que implica que están limitadas a recibir un sólo mensaje. En el sistema propuesto en la tesis, es fácil remover esta restricción dado que dentro del ambiente de ejecución existe el manejador de tareas quien entre sus funciones determina si se han satisfecho todos los requerimientos de una tarea antes de que ésta sea activada.

Por otro lado, el ambiente de ejecución también incluye al manejador de mensajes que transmite los mensajes con base a su prioridad durante el intervalo de tiempo asignado al nodo. Una forma de modelar esta tarea es como un servidor periódico que "ejecuta" los mensajes existentes, los cuales se encuentran divididos en paquetes de tamaño fijo, por lo que se puede suspender la transmisión de cualesquiera de ellos para enviar a otros de mayor prioridad. Dado estas características se considera más adecuado utilizar el método conocido como diseño de servidor *(server design)* propuesto en [Almeida y Pedreiras, 2004], que consiste en determinar el periodo máximo y costo mínimo de un servidor que sea capaz de ejecutar a un conjunto de tareas con prioridades fijas y con capacidad para ceder su turno *(preemptives)* garantizando que todas ellas se ejecutan dentro de su plazo, para determinar

ς_i	número de paquetes en que fue dividido el mensaje.
τ_i	periodo del mensaje.
δ_i	plazo relativo.
ϕ_i	prioridad.
Tg_i	identificador de la tarea que lo genera.
Tr_i	identificador de la tarea que lo recibe.

Tabla 4.1: Parámetros de los mensajes.

el peor tiempo de transmisión.

El análisis de planificabilidad de las tareas se basa principalmente en [Tindell y Clark, 1994], pero a diferencia de éste considera la posible existencia de múltiples mensajes, además de la sobrecarga generada por la tarea encargada de transmitirlos. Por su parte, el cálculo del peor tiempo de transmisión emplea el método propuesto en [Almeida y Pedreiras, 2004], obviamente con las modificaciones necesarias para adaptarlo a la arquitectura descrita.

En la sección 4.2 se desarrolla un modelo matemático del ambiente de ejecución descrito en el Capítulo 2 de esta tesis, este modelo es necesario para poder establecer las ecuaciones del peor tiempo de respuesta de las tareas y del peor tiempo de transmisión: secciones 4.3 y 4.4, respectivamente. En la sección 4.5, se describe como calcular el *jitter* que sufren las tareas que reciben mensajes. Por último, en la sección 4.6, se describen una serie de experimentos realizados con el propósito de comprobar estadísticamente que el análisis propuesto es correcto. Las conclusiones al respecto del análisis realizado se pueden observar en la sección 5.7.

4.2. Modelo del Sistema

El sistema está formado por un conjunto S de N nodos, $S = \{\eta_i, i = 1...N\}$, que se comunican entre si mediante una red de memoria compartida-distribuida, SCI. Sin embargo, para el propósito de este análisis, las características especificas de SCI no son determinantes, de ahí que sin perdida de generalidad, se referirá al canal de comunicación como una red TDMA.

Por cada nodo, η_i, existe un conjunto Π_{η_i} de N_{η_i} tareas periódicas y totalmente *preemptives*, $\Pi_{\eta_i} = \{J_i(C_i, T_i, D_i, P_i), i = 1...N_{\eta_i}\}$. Cada tarea, J_i, se caracteriza por

- peor tiempo de ejecución, C_i
- su periodo, T_i
- plazo relativo, D_i, menor o igual a su periodo; y
- una prioridad fija, P_i, única para cada tarea en el nodo que puede obtenerse a partir del periodo de la tarea, plazo o cualquier otro criterio.

Como ya se ha mencionado en capítulos anteriores (Capítulos 2 y 3), las tareas pueden comunicarse con otras en el mismo nodo o en nodos diferentes mediante el envió de mensajes entre ellas. Llamaremos Θ_{η_i} al conjunto de mensajes enviados por el nodo η_i, el cual

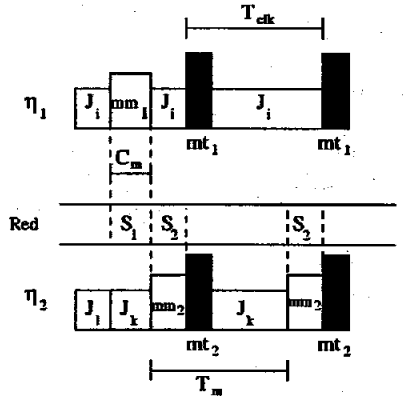


Figura 4.1: Sistema de dos nodos.

contiene M_{η_i} mensajes, $\Theta_{\eta_i} = \{\mu_i(\varsigma_i, \tau_i, \delta_i, \phi_i, Tg_i, Tr_i), i = 1...M_{\eta_i}\}$. La definición de cada uno de los parámetros del mensaje μ_i se muestra en la tabla 4.1.

Ahora bien, para evitar que un mensaje de alto costo de transmisión pero de baja prioridad haga uso exclusivo del canal de comunicación dado su tamaño, cada mensaje esta dividido en un número entero, ς_i, de paquetes; por lo tanto, es posible interrumpir la transmisión entre paquetes para enviar aquellos de mayor prioridad. Por su parte cada paquete requiere C_p unidades de tiempo para ser transmitido. Tanto su periodo τ_i como su prioridad ϕ_i se heredan de los parámetros respectivos de la tarea Tg_i que lo genera. Por otro lado, para garantizar que la tarea Tr_i que lo recibe tiene suficiente tiempo para procesarlo su plazo, δ_i, debe ser menor al de ésta.

Por último, el periodo del manejador de tareas, T_{clk}, es igual al máximo común divisor de los periodos de las tareas en todos los nodos. Por otro lado, para garantizar prontitud a la hora de transmitir los mensajes el periodo del manejador de mensajes, T_m, es igual a T_{clk}, es decir, al menor periodo en el sistema. El máximo tiempo disponible para transmitir los mensajes, el cual se considerará igual al tiempo de ejecución de mm, C_m, se obtiene mediante la Ecuación 4.1. Desde el punto de vista de la red TDMA: C_m es el tiempo asignado al nodo, mientras que T_m es igual a la duración de la ronda TDMA . Cabe resaltar que el manejador de mensajes puede ejecutarse durante un tiempo menor que C_m dependiendo del estado actual de la cola de mensajes a transmitir: el tiempo sobrante se emplea para ejecutar alguna otra tarea.

$$\begin{aligned} C_m &= kC_p \\ k &= \left\lfloor \frac{T_m}{(N+1)C_p} \right\rfloor \end{aligned} \tag{4.1}$$

En la figura 4.1 vemos la ejecución de un sistema compuesto por dos nodos durante dos periodos del manejador de tareas. Durante ese intervalo en η_1 sólo se ejecuta la tarea J_i, la cual se suspende para ejecutar tanto al tm como al mm, cabe destacar que como todos los mensajes son transmitidos durante la primera activación del manejador de mensajes, éste ya no vuelve a ejecutarse, por lo tanto la tarea J_i utiliza ése tiempo. Mientras tanto en η_2 se ejecutan las tareas J_l y J_k, es importante notar que en cuanto la tarea J_l termina la tarea J_k se activa.

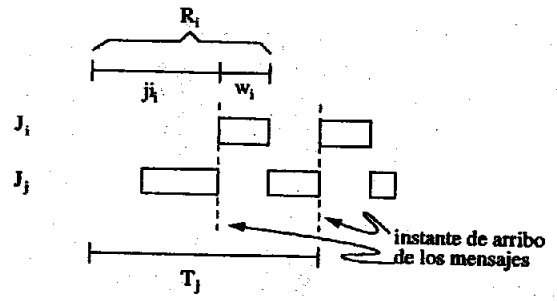


Figura 4.2: El problema del *jitter*.

4.3. Tiempos de respuesta

Determinar el peor tiempo de respuesta de las tareas es una parte fundamental del análisis: ser demasiado optimista al calcularlo podría ocasionar que una vez que el sistema sea implementado éste no cumpla con los requerimientos planteados como lo establece la teoría; por el contrario, si la valoración es demasiado pesimista sólo muy pocos conjuntos de tareas serán encontrados planificables mientras que en la practica podría resultar que si lo son.

Por otro lado, cuando se tienen tareas con precedencias, como es el caso de las tareas que reciben mensajes, éstas no se activan hasta que sus dependencias son resueltas , es decir, hasta que sus mensajes están disponibles. Ahora bien, la transmisión de los mensajes depende de dos factores: el instante en que son generados y la cantidad de mensajes de mayor prioridad que existan en la cola de mensajes a enviar. Debido a esto, el tiempo que emplea un mensaje para transmitirse puede variar entre ejecuciones, lo cual genera un *jitter* tanto en la generación de los mensajes como en la activación de las tareas que dependen de ellos. En la sección 4.5 se detalla como es posible calcularlos. Para obtener el tiempo de respuesta de las tareas y el tiempo de transmisión de los mensajes se considerará que estos valores ya se conocen. Llamaremos ji_i al *jitter* de la tarea J_i, y ι_i al del mensaje μ_i.

Que exista *jitter* en la activación de las tareas es un problema debido a que el peor caso entre activaciones consecutivas[1] de una tarea puede ser menor que el tiempo indicado por su periodo. Por ejemplo consideremos el caso en que la tarea J_i de mayor prioridad que la tarea J_j, Figura 4.2, llega a la cola de tareas listas en el instante 0, pero debido a sus precedencias no se activa hasta que han transcurrido ji_i unidades de tiempo, en ese instante la tarea J_j se estaba ejecutando, por lo que ocurre un cambio de contexto para ejecutar J_i. Al cumplirse su periodo (T_i), J_i es puesta en la cola de tareas listas, pero ahora los mensajes de los cuales depende ya se encuentran disponibles por lo que se activa inmediatamente. Desde el punto de vista de la tarea J_j, J_i se activa con un tiempo entre activaciones igual a $T_i - ji_i$. Por lo tanto, el peor tiempo de ejecución de una tarea, además de considerar que ésta puede activarse al mismo instante que todas las tareas con mayor prioridad, debe tomar en cuenta el peor caso entre activaciones consecutivas.

El peor tiempo de respuesta con respecto al instante de activación de la tarea, R_i, de la tarea J_i esta dado por el peor instante de activación de ésta, *jitter,* más el tiempo de ejecución de la tarea, w_i, Ecuación 4.2.

[1]El peor caso entre activaciones consecutivas es el menor tiempo existente entre la activación de dos instancias de la misma tarea.

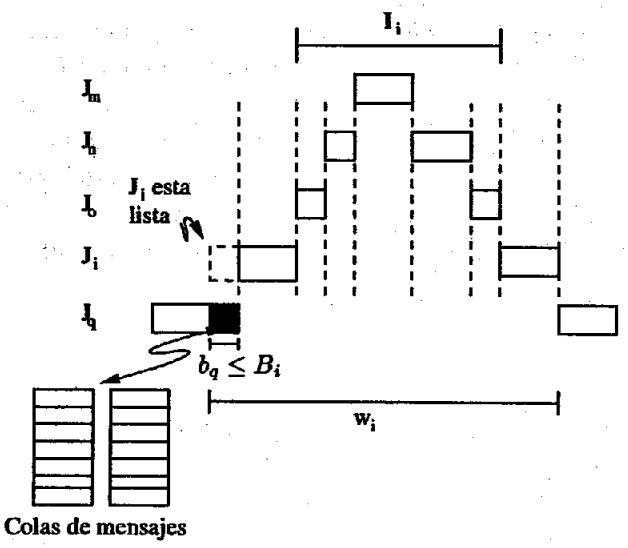


Figura 4.3: Interferencia y Bloqueo. La interferencia es causada por tareas de mayor prioridad mientras que el bloqueo por las de menor.

$$R_i = ji_i + w_i \tag{4.2}$$

Para determinar el tiempo de ejecución, se deben considerar todas las posibles contribuciones al retraso de la tarea. En la Figura 4.3, se muestran dos de ellas. I_i es la interferencia causada por las tareas de mayor prioridad, mientras que B_i es el bloqueo que la tarea puede sufrir cuando se activa en el instante que una tarea de menor prioridad se encuentra copiando un mensaje en o de alguna de las colas de mensajes. La interferencia I_i depende de w_i, mediante la Ecuación 4.3 se obtiene este valor, dicha ecuación considera a $hp(i)$ como el conjunto de tareas con mayor prioridad que la tarea J_i. Por su parte B_i se calcula con la Ecuación 4.4 donde $lq(i)$ es el conjunto de tareas con menor prioridad que la tarea J_i.

$$I_i(w_i) = \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + ji_j}{T_j} \right\rceil C_j \tag{4.3}$$

$$B_i = \max_{\forall j \in lp(i)} (b_j) \tag{4.4}$$

Otro factor a tomar en cuenta es el tiempo empleado por el manejador de tareas. Sin embargo, suponer que en cada activación del mt se emplea el máximo tiempo de ejecución es pesimista, por lo que se considerará independientemente cada uno de los procesos que éste realiza. Primero debe sincronizarse con los manejadores de tareas en otros nodos (Capítulo 3, sección 3.7), de cualquier forma el tiempo empleado para ello no necesita ser considerado porque es a partir del instante que los nodos están sincronizados que inicia la ronda (Capítulo 3, sección 3.5). A continuación, el mt debe mover las tareas que se encuentren listas para ser ejecutadas de la cola de tareas en espera a la cola de tareas listas. En seguida,

compara la prioridad de la tarea que esta al frente de la cola de tareas listas con la prioridad de la tarea que actualmente se esta ejecutando. Para realizar estas actividades se requiere a lo más C_{pr} unidades de tiempo. Si la tarea en la cola tiene mayor prioridad entonces ocurre un cambio de contexto que emplea C_{cs}. Para determinar si existen tareas listas mt necesita C_{clk}. Por lo tanto, mientras que C_{clk} se emplea en cada activación tanto C_{pr} como C_{cs} dependen de las prioridades y periodos de las otras tareas. La Ecuación 4.5 determina el número de veces, L, que el manejador de tareas se ejecuta durante el intervalo w_i:

$$L(w_i) = \left\lfloor \frac{w_i}{T_{clk}} \right\rfloor \tag{4.5}$$

Dentro de ese intervalo el número de tareas que pasan de una cola a otra, KT_i, esta dado por la Ecuación 4.6:

$$KT_i(w_i) = \sum_{\forall j \in \Pi_\eta} \left\lfloor \frac{w_i + ji_j}{T_j} \right\rfloor \tag{4.6}$$

Finalmente, el número de tareas de mayor prioridad que la tarea J_i que podrían activarse es igual a:

$$KH_i(w_i) = \sum_{\forall j \in hp(i)} \left\lfloor \frac{w_i + ji_j}{T_j} \right\rfloor \tag{4.7}$$

No obstante, debido a que a lo más puede ocurrir un cambio de contexto por cada ejecución del tm, el número de tareas que efectivamente se despachan durante w_i tiene una cota superior dada por L_i. Por lo tanto, el retraso en la ejecución de la tarea J_i generado por el manejador de tareas esta dado por la Ecuación 4.8.

$$Ts_i(w_i) = L(w_i)C_{clk} + KT_i(w_i)C_{pr} + \min(L(w_i), KH_i(w_i))C_{cs} \tag{4.8}$$

Un análisis similar para determinar el retraso ocasionado por un calendarizador controlado por reloj se encuentra en [Tindell y Clark, 1994].

Finalmente, es necesario considerar el tiempo empleado para transmitir los mensajes. Una forma simple de hacerlo es incluir una tarea con la mayor prioridad, periodo T_m y costo C_m en cada uno de los conjuntos Π_η. Una opción menos pesimista, consiste en determinar el número de mensajes generados durante w_i y a continuación calcular el tiempo necesario para transmitirlos.

La Ecuación 4.9 obtiene el número de paquetes producidos durante w_i. Es importante notar que sólo es necesario considerar el conjunto de mensajes que se envían a tareas que se ejecutan en otro nodo ,$\lambda_\eta \subseteq \Theta_\eta$, ya que como se describe en el Capítulo 3 los mensajes generados localmente se copian directamente a la cola de mensajes entrantes sin la intervención del mm.

$$G_i(w_i) = \sum_{\forall m \in \lambda_\eta} \left\lfloor \frac{w_i + \iota_i}{\tau_j} \right\rfloor \varsigma_i \tag{4.9}$$

Ahora bien, si durante el tiempo asignado se tiene capacidad para ψ paquetes, con $\psi = \frac{C_m}{C_p}$, entonces se requieren $\left\lceil \frac{G_i(w_i)}{\psi} \right\rceil$ activaciones del mm para transmitir todos los paquetes, pero durante w_i a lo más pueden ocurrir $\left\lceil \frac{w_i}{T_m} \right\rceil$ de ellas. Por lo tanto el tiempo necesario para transmitir los mensajes esta dado por:

$$M_i(w_i) = \text{mín}(\left\lceil \frac{G_i(w_i)}{\psi} \right\rceil, \left\lceil \frac{w_i}{T_m} \right\rceil) C_m \tag{4.10}$$

Como se ha podido observar la mayoría de los factores que determinan el tiempo de ejecución de la tarea dependen a su vez de éste, por lo que se requiere emplear una ecuación recursiva 4.11 para determinarlo. Dicha ecuación puede resolverse iterativamente con $w_i^0 = C_i$. La Ecuación 4.11 converge a $w_i^n = w_i^{n+1}$ o crece por encima del plazo de alguna de las tarea dentro de un número finito de iteraciones.

$$w_i^{n+1} = C_i + B_i + I_i(w_i^n) + Ts_i(w_i^n) + M_i(w_i^n) \tag{4.11}$$

Finalmente, el conjunto de tareas Π_{η_i} en el nodo η_i será planificable si satisface la Condición 4.12.

$$\forall i \in \Pi_{\eta_i} \; : \; R_i \leq D_i \tag{4.12}$$

4.4. Tiempos de transmisión

Para determinar el tiempo de transmisión se considera que el manejador de mensajes así como el tiempo dentro de la ronda TDMA asociado a éste se comportan como un servidor periódico, encargado de despachar los mensajes de acuerdo a su prioridad. Por lo tanto habrá un servidor por nodo. Ahora bien, el número de activaciones del mm durante un periodo de tiempo, t, establece la capacidad que tiene el servidor para transmitir los mensajes. Por lo tanto, definimos la función de disponibilidad $A(t)$, Ecuación 4.13, que determina para cada instante t la capacidad del servidor.

$$A(t) = \begin{cases} kC_m, & kT_m \leq t < kT_m + \Delta \\ t - (k+1)\Delta, & kT_m + \Delta \leq t < (k+1)T_m \end{cases} \tag{4.13}$$

$$\Delta = T_m - C_m$$

$$k = \left\lfloor \frac{t}{T_m} \right\rfloor$$

Finalmente, es necesario saber la carga máxima que un mensaje entrega al servidor para determinar el tiempo que le llevará procesarla. Dicha carga ocurre cuando el mensaje se genera al mismo tiempo que los de mayor prioridad. La función de carga, 4.14, calcula este valor; donde $m_{hp}(i)$ es el conjunto de mensajes con mayor prioridad que el mensaje i.

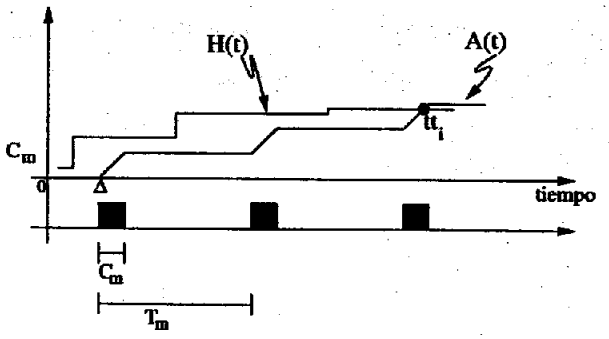


Figura 4.4: Función de disponibilidad $A(t)$ y función de carga $H_i(t)$.

$$H_i(t) = \left(\sum_{\forall j \in m_{hp}(i)} \left\lceil \frac{t + \iota_j}{\tau_j} \right\rceil \varsigma_j + \varsigma_i \right) C_p \tag{4.14}$$

Por lo tanto, el tiempo de transmisión del mensaje i, tt_i, esta dado por el mínimo instante en que el servidor tiene suficiente capacidad para procesar la carga, Figura 4.4. El valor de tt_i puede calcularse empleado la Ecuación 4.15, o más eficientemente utilizando 4.16 que emplea la función inversa de la función de disponibilidad, $A^{inv}(u)$, Ecuación 4.17

$$tt_i = \text{mín}(t) \;:\; A(t) = H_i(t) \tag{4.15}$$

$$tt_i = \text{mín}(t) \;:\; t = A^{inv}(H_i(t)) \tag{4.16}$$

$$A^{inv}(u) = \Delta + mT_m + (u - mC_m), \quad m = \left\lceil \frac{u}{C_m} \right\rceil - 1 \tag{4.17}$$

Estrictamente, invertir a $A(t)$ requiere especificar para cada intervalo el valor de su función inversa, sin embargo, para el análisis que se esta realizando, es suficiente con conocer el límite inferior de ellos, con lo que se obtiene la menor disponibilidad, Ecuación 4.17.

La Ecuación 4.16 pude resolverse iterativamente con $tt_i^0 = H_i(0)$. Nuevamente, 4.16 converge a $tt_i^{n+1} = tt_i^n$ o crece por encima del plazo de alguno de los mensajes dentro de un número finito de iteraciones. Para demostrar este hecho, basta con notar que la función de carga es monotónicamente creciente en múltiplos de C_p.

Ahora bien, el peor tiempo de transmisión, ρ_i, del mensaje i estará dado por:

$$\rho_i = \iota_i + tt_i \tag{4.18}$$

Finalmente el conjunto de mensajes Θ_{η_i} generados por el nodo η_i es planificable si se cumple con la Condición 4.19.

$$\forall i \in \Theta_{\eta_i} : \; \rho_i \leq \delta_i \tag{4.19}$$

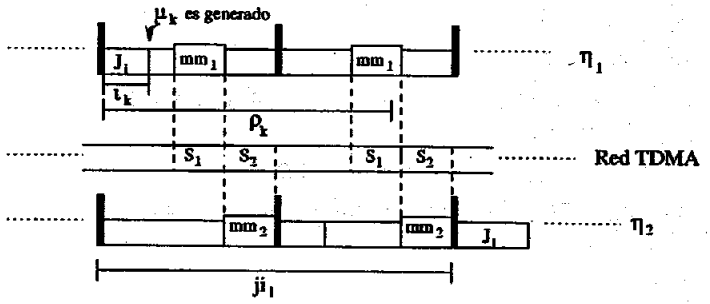


Figura 4.5: *Jitters* existentes en la transmisión de los mensajes.

4.5. Cálculo de los *jitters*

Para comprender como es posible calcular el *jitter*, tanto de los mensajes como de las tareas, consideremos el ejemplo dado por la Figura 4.5, que consiste en una tarea J_i en el nodo η_1 que envía el mensaje μ_k a la tarea J_l en el nodo η_2. Ambas tareas llegan a su respectiva cola de tareas listas en el mismo instante, pero debido a sus precedencias la tarea J_l no se activa. En el peor de los casos μ_k se genera como la última instrucción de la tarea J_i, es decir en R_i, por lo tanto el *jitter* del mensaje, ι_k, es igual a este valor. De forma general el *jitter* del mensaje m es igual al peor tiempo de ejecución de la tarea que lo genera, Ecuación 4.20. Por otra parte, debido a la cantidad de mensajes existentes en la cola de mensajes salientes en el nodo η_i, μ_k no se trasmite hasta el instante ρ_k, pero es necesario que sea ejecute el manejador de tareas en η_2 para que la tarea J_l sea activada. Por lo tanto el valor del *jitter* de las tareas que reciben mensajes está dado por la Ecuación 4.21, donde $m_r(i)$ es el conjunto de mensajes que recibe la tarea i.

$$\iota_m = R_m \tag{4.20}$$

$$ji_i = \max_{\forall m \in m_r(i)} \left(\left\lceil \frac{\rho_m}{T_{clk}} \right\rceil T_{clk} \right) \tag{4.21}$$

Sin embargo, si se contara con el mejor caso tanto para las tareas como para los mensajes sería posible determinar el valor del *jitter* de una manera más precisa: básicamente, el *jitter* esta dado por el valor absoluto de la diferencia entre el mejor y el peor caso. Si sólo se considera el peor caso, el pesimismo introducido en el cálculo de éste es mayor. De cualquier forma, por considerar al *jitter* generado por el peor tiempo de ejecución se determina al mismo tiempo el peor tiempo de ejecución con respecto al periodo de la transacción[2] , es decir, el peor tiempo de transmisión del mensaje ρ_i es a su vez el peor tiempo de respuesta de la tarea que envía el mensaje, si se considera que la tarea termina cuando el mensaje es entregado a su destino; y R_i para una tarea receptora es el peor tiempo de ejecución con respecto al instante de activación de la tarea que envía el mensaje. Por lo tanto, al determinar R_i de la tarea receptora considerando este *jitter* se esta encontrando el retraso total (*end to*

[2]Se considera transacción a la triada formada por: la tarea que envía el mensaje, el mensaje y la tarea que lo recibe.

```
1  funct determina_planificabilidad(S)
2    call S := init_jitter(S)
3    call (S, J_inicial, Es_planificable) = haz_analisis(S)
4    terminar := False
5    while (Es_planficable ∧ ¬terminar) do
6          call (S, J_final, Es_planificable) := haz_analisis(S)
7          if J_incial = J_final
8            then
9                 terminar := True
10           else
11                J_inicial := J_final
12         fi
13   done
14   return Es_planificable
15 end
```

Figura 4.6: Algoritmo iterativo para realizar el análisis de planificabilidad.

end) de la transacción que incluye el tiempo de ejecución de la tarea que envía el mensaje, el tiempo de transmisión del mensaje y el tiempo de ejecución de la tarea que lo recibe. Esto puede observarse claramente en la figura, 4.5. Este resultando es muy importante porque en la mayoría de los casos con lo que se trabaja es con el deadline de la transacción y no con el de las tareas y mensajes por separado.

Ahora bien, tanto el peor tiempo de respuesta R_i como el peor tiempo de trasmisión ρ_m, Ecuaciones 4.2 y 4.18 respectivamente, dependen del valor de los *jitters*. Debido a esto se hace necesario emplear un algoritmo iterativo, Figura 4.6, para realizar el análisis. El algoritmo recibe como parámetro de entrada, al conjunto S formado por todos los nodos del sistema, posteriormente, Línea 2, procede a inicializar a ceros los posibles *jitters*. Para llevar a cabo el análisis se emplea la función *haz_analisis()*, la cual emplea al conjunto S como parámetro de entrada y devuelve una terna compuesta por: un conjunto de nodos similar al que recibió como entrada pero con el valor de sus *jitters* igual al calculado, el conjunto de valores de *jitter* obtenidos y una variable de tipo falso o verdadero donde indica si el sistema es o no planificable. Dicha función determina si se satisfacen las condiciones 4.12 y 4.19, para ello realiza el cálculo del peor tiempo de ejecución y del peor tiempo de transmisión de la forma como se explicó en las secciones previas. Por último, calcula el valor de los *jitters* de las tareas y de los mensajes, que se emplearán en futuras llamadas a esta función. A continuación, en la Línea 5 se tiene el ciclo: que termina ya sea porque se determina que el sistema no es planificable o porque durante dos iteraciones consecutivas el valor de los *jitters* se mantuvo constante, lo cual es suficiente para garantizar que a partir de ese momento los *jitters* permanecerán constantes.

Para demostrar que el algoritmo de la Figura 4.6 termina dentro de un finito número de iteraciones es necesario mostrar que tanto el *jitter* de los mensajes como el de las tareas son funciones monotónicamente crecientes, en ambos casos es suficiente con demostrar que su crecimiento se encuentra limitado por una cota mínima. Como se muestra en la Ecuación

4.20, el *jitter* en la generación de los mensajes depende del peor tiempo de ejecución de la tarea que lo genera y éste, a su vez, aumenta en términos del costo de las tareas de mayor prioridad, Ecuación 4.3, más el tiempo empleado por el sistema, es decir por el manejador de tareas y el manejador de mensajes (Ecuaciones 4.8 y 4.10) respectivamente. Por lo tanto, el *jitter* asociado a los mensajes tienen una cota mínima dada por la ecuación:

$$M_{cm} = \text{mín}(C_{clk}, C_{pr}, C_{cs}, C_p, \min_{\forall j \in \Pi_{\eta_i}} (C_j)) \tag{4.22}$$

Por otra parte, dado que es necesario que el manejador de tareas se ejecute para que las tareas que reciben mensajes se activen (Capítulo 2, sección 2.4.4), el *jitter* asociado a éstas crece en múltiplos del periodo del reloj, siendo la duración de éste la cota mínima.

4.6. Análisis numérico y simulación

Con el propósito de verificar estadísticamente que las ecuaciones obtenidas proporcionan resultados confiables, esto es, si mediante el análisis numérico se concluye que el conjunto de tareas y mensajes es planificable entonces éste debe serlo. Por el contrario, dado a que se realizó el análisis considerando el peor caso posible y en la práctica este caso puede no presentarse, un conjunto marcado como no planificable puede realmente serlo. Por lo tanto, además de corroborar el análisis, los experimentos proporcionan información acerca de cuan pesimista es éste.

En cada uno de los experimentos se realiza tanto el análisis numérico mediante las ecuaciones descritas en este capítulo, como empleando un simulador construido para tal fin.

4.6.1. Conjuntos de prueba

Para poder realizar cualquier análisis se requiere contar primero con conjuntos de tareas y mensajes representativos del sistema que se esta evaluando. En el caso particular de esta tesis, se decidió generar estos conjuntos de manera semi-aleatoria: esto porque existen características como el costo de las tareas que deben controlarse para lograr una utilización total preestablecida.

En el caso de las tareas cada uno de sus parámetros se genera de la siguiente manera:

Periodo (T_i): como se establece en la sección 4.3, la duración del periodo del reloj es igual al máximo común divisor del periodo de todas las tareas: para simplificar este cálculo se estableció primero la duración de éste, Tabla 4.2, y a continuación se determinó el periodo de las tareas como múltiplo de T_{clk} en el rango de 10 a 100 milisegundos. Por otra parte, las tareas que se comunican entre ellas deben tener el mismo periodo: para garantizar que existan suficientes parejas de tareas con esta característica; se emplea un conjunto de periodos, generados de la forma antes mencionada, menor que el número total de tareas que se desea obtener y el periodo de éstas se selecciona dentro de este conjunto.

Peor tiempo de ejecución (C_i): dado que, junto con el periodo, el costo de la tarea determina el porcentaje de utilización de ésta ($U_i = \frac{C_i}{T_i}$), se requiere repartir "equitativamente" el ancho de banda total disponible entre el número de tareas que se desea. Una

forma muy simple de realizar esto es dividiendo el ancho de banda entre el número de tareas, lo cual no es del todo recomendable debido a que las tareas generadas presentarán poca variabilidad. Por lo tanto, para determinar C_i se optó por una solución intermedia: en ella el ancho de banda utilizado por la tarea se seleccionó aleatoriamente, entre un valor mínimo que garantiza un costo mínimo y un máximo que varía dinámicamente dependiendo del número de tareas que falten por asignar y el ancho de banda disponible en ese momento. De esta manera a la última tarea se le asigna el ancho de banda sobrante; al mismo tiempo, si el ancho de banda total disponible no es suficiente para garantizar que todas las tareas tendrán un costo al menos igual al mínimo, el total de tareas se reduce hasta que esta condición se satisface.

Plazo (D_i): el plazo se fijó igual al periodo de la tarea.

Prioridad (P_i): la prioridad de las tareas se asignó empleando el criterio de *Deadline Monotonic* (menor plazo mayor prioridad). Cuando las tareas tienen plazos iguales, aleatoriamente se establece cual de ellas tiene mayor prioridad.

Por su parte, para generar los mensajes se emplean consideraciones similares, específicamente en el caso del plazo (δ_i) este se estableció igual al de la tarea que lo recibe menos un porcentaje obtenido aleatoriamente del costo de ésta.

4.6.2. Características del simulador

La simulación de los conjunto de tareas se realizó durante un periodo de tiempo correspondiente a dos macro-periodos, es decir, dos veces el mínimo común múltiplo de todos los periodos. Tal duración garantiza que el peor caso ocurrirá durante la simulación.

El simulador se encuentra principalmente basado en SimHol [Calha y Fonseca], pero a diferencia de éste considera la existencia de una red TDMA, además del manejador de tareas y del manejador de mensajes.

A la hora de desarrollar al simulador se plantearon los siguientes requerimientos:

- Una interfaz simple e intuitiva.
- Descripción de los escenarios mediante archivos texto: de tal forma que tanto los usuarios como algún otro programa puedan fácilmente especificar las características del sistema que desean simular.
- Toda la información obtenida por el simulador es salvada a archivos de texto: lo que realizar su posterior procesamiento, al mismo tiempo que es fácil de leer.
- Emplear el menor tiempo posible para llevar a cabo la simulación.
- Operación del simulador lo más similar al sistema real.
- Contar con dos versiones: una con gui ("*graphical user interfaz*: interfaz gráfica de usuario") que permita observar las dependencias y la ejecución del sistema; y otra en la forma de línea de comandos que pueda ser llamada por otros programas: esto con el objetivo de integrarla al programa que realiza la generación de los conjuntos de tareas.

```
1  tiempo_actual := 0
2  while tiempo_actual ≤ 2 * macro_periodo do
3        foreach η ∈ S do
4          call manejador_tareas
5        done
6        tiempo := tiempo_actual + C_pr + C_cs + C_clk
7        foreach η ∈ S do
8          ta := tiempo_disponible_antes_mm
9          at := tiempo_actual
10         call despacha_tareas(ta, at)
11         if existen_mensajes
12           then
13                 call manejador_mensajes()
14         fi
15         ta := tiempo_disponible_despues_mm
16         call despacha_tareas(ta, at)
17       done
18       tiempo_actual := ⌈at / T_clk⌉ T_clk
19 done
```

Figura 4.7: Algoritmo empleado por el simulador.

De manera similar a como se realizó el manejador de tareas, el algoritmo del simulador también se basa en rondas. Al final de la ronda se considera que ha transcurrido un periodo del manejador de tareas. En la Figura 4.7 se observa el algoritmo empleado por el simulador. Al inició de la ronda (Líneas 3-5), en cada uno de los nodos, se llama a la función que simula al manejador de tareas, la cual determina que tareas están listas para ser ejecutadas y cual de ellas tiene la mayor prioridad. Por simplicidad, se considera que esta función consume todo el tiempo asignado a ella (Línea 6). Posteriormente (Líneas 8-9), se calcula el tiempo disponible antes de que sea llamado el manejador de mensajes, y mediante la función despacha_tareas() se procede a la ejecución de las mismas de acuerdo a su prioridad durante el tiempo asignado. Dicha función también determina si alguna de las tareas viola su plazo: en el caso de que esto ocurre se genera una interrupción, se indica que tarea fue y la simulación termina en estado de error. Si durante la ronda actual o en rondas anteriores se han generado mensajes, la función que simula al manejador de mensajes se activa (Línea 11). Por último, se calcula el tiempo disponible antes de que concluya la ronda para continuar con la ejecución de las tareas correspondientes.

Para especificar el escenario que se desea simular se emplea un archivo de configuración. Éste incluye los detalles de la simulación: nombre, capacidad del canal y duración de la ronda; y las características de los nodos, tareas y mensajes involucrados. El formato del archivo es una letra (S, B, W, N, T, C y P) que indica el parámetro que se está especificando seguido del valor de éste, Figuras 4.8 y 4.9.

```
#Nombre de la simualción
#S nombre
S simtest

#Tipo de canal de comunicacion
#B Tipo BitRate(b/s) Tamaño_del_paquete
B TDMA 1000000000 1000

#Duracion de la ronda
#W duracion en microsegundos
W 10000

#Declaración de los nodos
#N Id
# - Id: 1 < Id < 255
N 1
N 2

#Declaración de las tareas
#T Id C P D F Nodo_Id Prio
# - Id: 1 < 255
# - C: Costo
# - P: Periodo
# - D: Plazo
# - F: Fase inicial
# - Nodo_Id: Identificador del nodo
# - Prio: Prioridad, mayor valor numérico mayor prioridad.
T 1 971 80000 80000 0 1 10
T 2 997 90000 90000 0 1 9
T 3 1007 90000 90000 0 1 8
T 4 1045 100000 100000 0 1 7
T 5 1051 100000 100000 0 1 6
T 6 1086 100000 100000 0 1 5
T 7 1100 100000 100000 0 1 4
T 8 1126 100000 100000 0 1 3
T 9 1148 100000 100000 0 1 2
.
.
.
```

Figura 4.8: Ejemplo de archivo de entrada para el simulador.

```
.
.
.
#Declaracion de los mensajes
# P Id NP TP NTP
# - Id: Igual al de la tarea que lo produce
# - NP: Número de paquetes
# - TP: Identificador tarea productora
# - NTP: Identificador del nodo de TP
# C Id TP NTP TC NTC
# - Id: Igual al de la tarea que lo produce
# - TP: Identificador tarea productora
# - NTP: Identificador del nodo de TP
# - TC: Identificador de la tarea consumidora
# - NTC: Identificador del nodo de TC
P 2 5027 2 1
C 2 2 1 10 2
P 4 9413 4 1
C 4 4 1 14 2
P 14 433 14 2
C 14 14 2 18 2
P 17 14566 17 2
C 17 17 2 7 1
```

Figura 4.9: Ejemplo de archivo de entrada para el simulador.(Continuación)

Especial mención requiere la forma como se especifican los mensajes. Para ello se emplea una dupla, primero se especifican los detalles de la tarea que lo produce (letra P) y a continuación los de la tarea que lo consumen (letra C). No obstante que alguna de la información que se describe mediante estas letras es duplicada, es necesario hacerlo de esta manera para simplificar la función que realiza el análisis gramatical.

Los resultados de la simulación del escenario descrito anteriormente se muestran en las Figuras 4.10 y 4.11. En la Figura 4.10 se observan las características de la simulación así como la duración del macro-periodo. También se despliegan las precedencias de las tareas y mensajes, junto con sus características: costo (C), plazo (D), periodo (T), nodo (N), peor tiempo de ejecución (EC).

Por su parte en la Figura 4.11 aparece la gráfica de la simulación. Horizontalmente cada cuadro representa 10 milisegundos, verticalmente tenemos (por cada nodo): MT duración del manejador de tareas, MM duración del manejador de mensajes, Tx-Ty ejecución de las tareas de la x a la y, Nz utilización del procesador en el nodo z. Resulta interesante observar la ejecución de las tareas que tienen precedencias, por ejemplo, la T4 en el nodo 1 le envía el mensaje M4 a T14 que a su vez le envía el mensaje M14 a la tarea T18, pero como estas dos últimas tareas se ejecutan en el mismo nodo, no es necesario que se active el manejador de mensajes para transmitir éste, sin embargo, el manejador de tareas debe ejecutarse para que determine que las dependencias de T18 se han satisfecho.

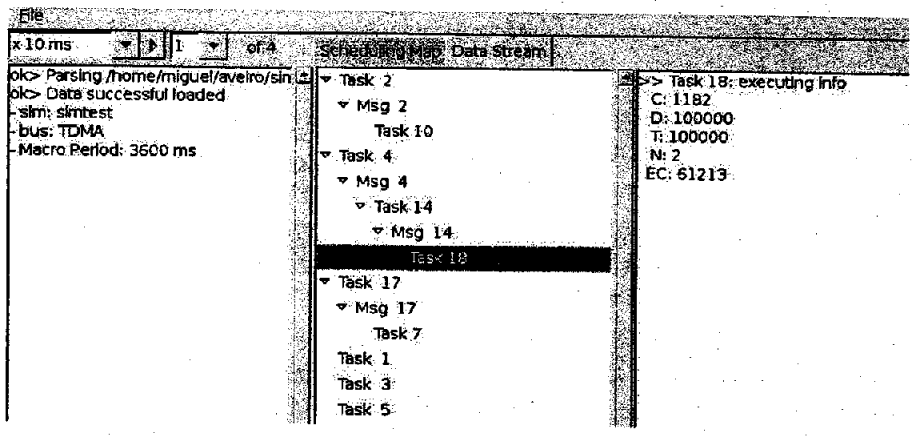


Figura 4.10: Ejemplo de simulación: precedencias de tareas.

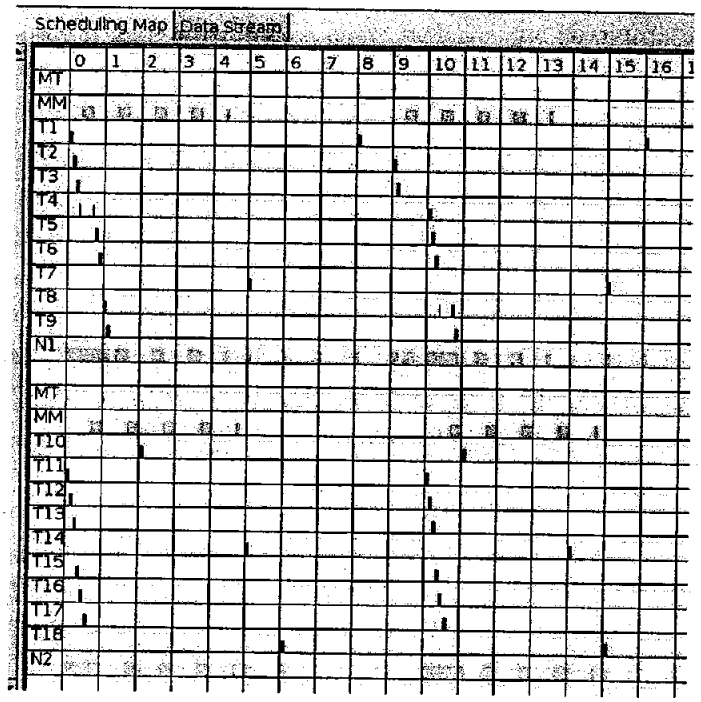


Figura 4.11: Ejemplo de simulación: ejecución.

B_i	T_{clk}	C_{clk}	C_{pr}	C_{cs}	C_p
$10\mu s$	$10ms$	$10\mu s$	$10\mu s$	$10\mu s$	$1\mu s$

Tabla 4.2: Parámetros empleados a través de los experimentos.

4.6.3. Experimentos realizados

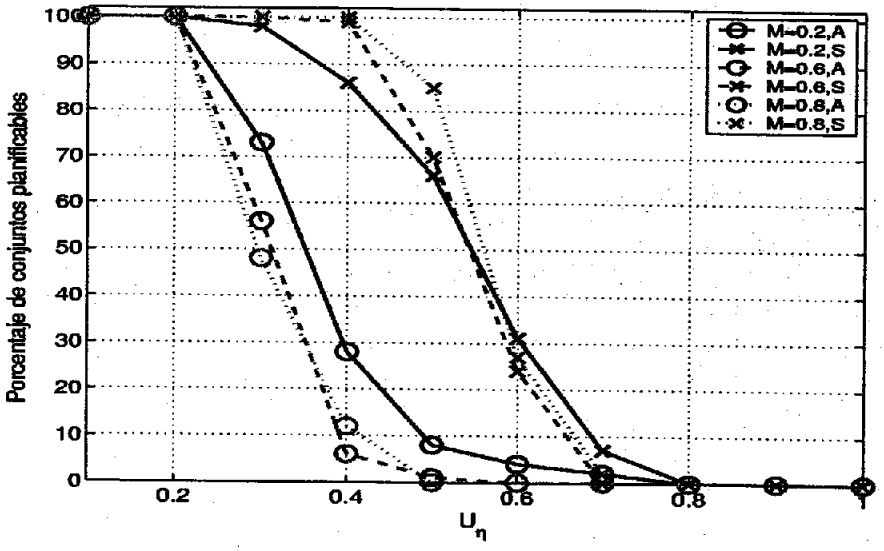
En total se efectuaron tres experimentos enfocados en características especificas que podrían tener gran impacto en el análisis de planificabilidad del sistema. En todos ellos se evalúa el nivel de planificabilidad expresado por el porcentaje de conjuntos planificables como función de la utilización del nodo, U_η, variando ésta de 0.1 a 1.0 en incrementos de 0.01. Por cada valor de U_m se generan 100 conjuntos de tareas y mensajes. Finalmente, tanto para realizar el análisis numérico como para la simulación se emplearon los datos de la Tabla 4.2.

En el primero de ellos el objetivo es evaluar al sistema conforme el número de mensajes aumenta: esta característica es importante debido a que las tareas que reciben mensajes no se activan hasta que estos han sido recibidos, lo que genera un "retraso" en la ejecución de estas tareas y dado a que no existe ninguna restricción en la asignación de prioridades podría suceder que la tarea que recibió el mensaje suspenda a otra que envía algún mensaje con lo que el tiempo de ejecución de esta última aumenta lo que a su vez afecta a otra en otro nodo que recibe el mensaje y a si sucesivamente. Esta situación se refleja en la forma iterativa en que se realiza el análisis de planificabilidad, Figura 4.6.

Para realizar este experimento se generaron aleatoriamente 10,000 conjuntos de tareas y mensajes para cada una de las variaciones. Los conjuntos se generaron de la forma descrita anteriormente. El total de mensajes, M, se expresa en un porcentaje del número de tareas, variando de 0 a 1 en múltiplos 0.1. Sin embargo, el porcentaje de la utilización de la red se mantuvo constante e igual a 0.3. La longitud del mensaje, y por lo tanto su costo de transmisión, se obtuvo de manera similar al costo de las tareas. Para este experimento se consideró que se contaba con dos nodos y con 20 tareas por nodo.

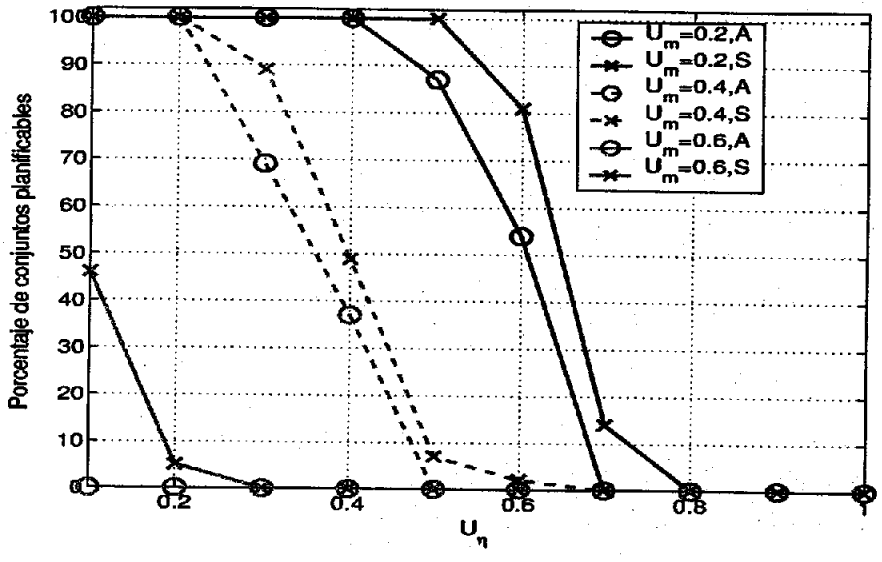
Los resultados para los casos en que el 20 %, 60 % y 80 % de las tareas envían mensajes se observan en la Figura 4.12. Como se esperaba, el número de mensajes afecta drásticamente al análisis numérico. Esto se debe a que el análisis se realiza iterativamente y al aumentar el número de mensajes aumenta el número de tareas con *jitter* lo que incrementa el número de iteraciones que se deben realizar, lo cual a su vez contribuye aumentar el pesimismo del análisis realizado. Por otro lado, el resultado de la simulación establece que en general el número de mensajes no es un factor preponderante siempre y cuando el ancho de banda utilizado por éste se mantenga constante.

El segundo experimento tiene como propósito evaluar al análisis conforme los mensajes hacen mayor uso del ancho de banda disponible. En el experimento el 50 % de las tareas envían mensajes, el total del ancho de banda empleado por estos ,U_m, varía entre 10 y 100 % en múltiplos de 10 %. Nuevamente, por cada variación, se emplean 10,000 conjuntos de 20 tareas por nodo: considerando 2 nodos en total. Los resultados para U_m igual a 20 %, 40 % y 60 % se muestran en la Figura 4.13. En ella podemos constatar que el ancho de banda es un cuello de botella real en los sistemas distribuidos: tanto los resultados de la simulación como el análisis numérico decaen rápidamente en cuanto el ancho de banda utilizado por los mensajes aumenta.



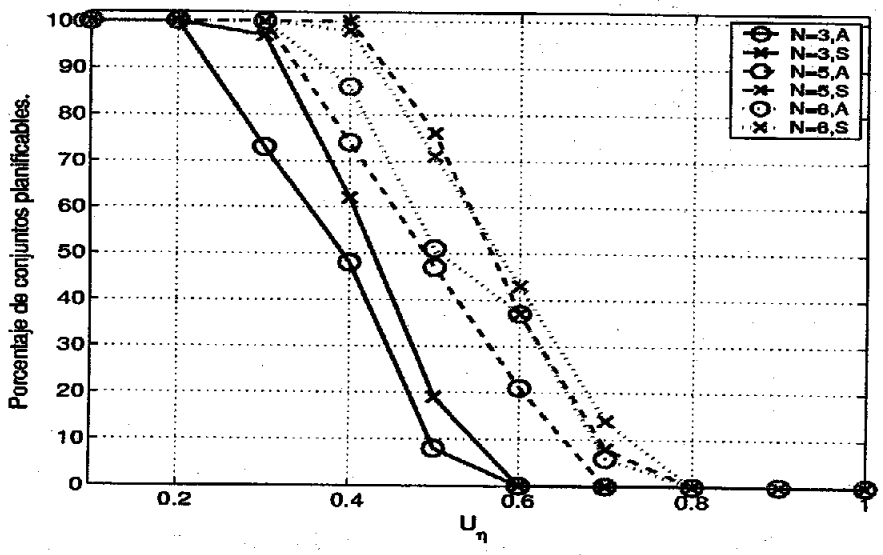
A: Análisis; S:Simulación.

Figura 4.12: Porcentaje de conjuntos planificables: número de mensajes, M, variable (20 %,60 % y 80 % del total de tareas); ancho de banda usado por los mensajes constante e igual a 30 %; número de nodos constante (2).



A: Análisis; S:Simulación.

Figura 4.13: Porcentaje de conjuntos planificables: número de mensajes, M, constante (50 % del total de tareas); ancho de banda usado por los mensajes, U_m, variable (20 %,40 % y 60 %); número de nodos constante (2).



A: Análisis; S:Simulación.

Figura 4.14: Porcentaje de conjuntos planificables: número de mensajes, M, constante (30 % del total de tareas); ancho de banda usado por los mensajes, U_m, constante (30 %); número de nodos variable (3,5,6).

En el último experimento se varía el número de nodos, manteniendo el total de mensajes constante e igual al 30 % del total de tareas. Por su parte, el total del ancho de banda utilizado se fijó a 30 %. Al igual que en los otros experimentos se generaron 20 tareas por nodo, con un total de 10,000 conjuntos por cada variación. En la Figura 4.14 se observan los resultados obtenidos para el caso de 3, 5 y 6 nodos. Cabe destacar que el análisis y la simulación muestran que el número de nodos no es factor tan preponderante como lo es el ancho de banda empleado por los mensajes.

4.7. Conclusiones

En este capítulo se desarrollaron las ecuaciones que permiten determinar analíticamente si un conjunto de tareas y mensajes es planificable. Además se construyó un simulador con el objetivo de validar estadísticamente la correctez de los resultados obtenidos numéricamente.

Por otra parte, en todos los experimentos realizados se observó que el análisis de planificabilidad es confiable, esto es, cuando numéricamente se determina que un conjunto de tareas y mensajes son planificables, estos lo son. También, como se puede observar en las Figuras 4.12, 4.13 y 4.14, para la mayoría de los casos, cuando no existe "demasiados" mensajes, el pesimismo introducido por el análisis es aceptable.

Capítulo 5

Caso de estudio

5.1. Introducción

Con el objetivo de corroborar que la arquitectura propuesta cumple que el requerimiento planteado de proporcionar soporte para tiempo real en sistemas de alto rendimiento, se diseñó un caso de estudio que hace uso exhaustivo de los recursos que dicha arquitectura proporciona.

Para que la plataforma sea probada al máximo el caso de estudio debe ser:

computacionalmente demandante: debe hacer uso de las características de alto rendimiento con que cuenta un *cluster.*

uso intensivo del canal de comunicación: la cantidad de mensajes transmitidos debe ser considerable, con el fin de probar el correcto manejo de estos.

sencillo de implementar: el caso de estudio debe ser simple, no debe perderse de vista que el objetivo es probar que la plataforma funciona de acuerdo a lo planeado.

Un problema que cumple con estos requisitos es la caracterización de la técnica "traslape de redes ART" (Overlappend ART Networks: OAN) descrita en [Benítez-Pérez y García-Nocetti,] para el caso que se tiene como entrada una señal sinusoidal con ruido. Dicho caso cuenta con la ventaja de que ha sido estudiado previamente por lo que demostrar que ha sido correctamente implementado, desde el punto de vista del algoritmo, será simple. En la sección 5.2 se detallan las características de dicha técnica. Las tareas que conforman al caso de estudio se describen en la sección 5.3. Por su parte, en la sección 5.5 se discute el código fuente correspondiente a dichas tareas. La validez tanto algorítmica como temporal de la aplicación se muestra en la sección 5.6. Finalmente la sección 5.7 concluye al capítulo.

5.2. Descripción del caso de estudio

El objetivo de la técnica OAN, empleada como caso de estudio, es el de la localización de fallas en sistemas físicos. Para ello se utilizan tres redes neuronales del tipo ART2 [S. Grossberg] (*Adaptive Resonance Theory*: Teoría de Resonancia Adaptiva). La red ART más simple es un clasificador de vectores: toma un vector de entrada y lo clasifica dentro

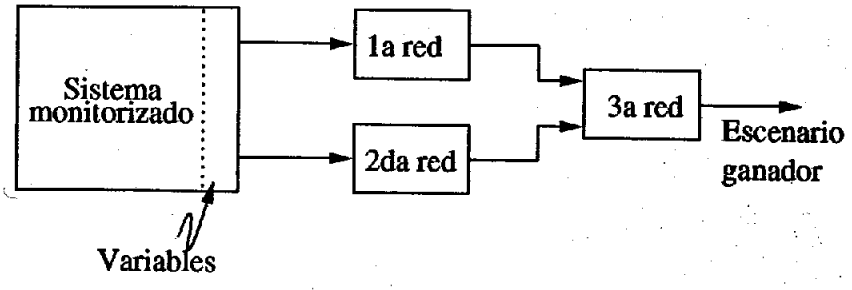


Figura 5.1: Diagrama de la técnica traslape de redes ART.

de una categoría con base a un grupo de patrones previamente almacenados. Cuando se encuentra un patrón que coincide, éste es modificado (entrenado) para que refleje las características del nuevo vector. Si el vector de entrada no coincide con ninguno se crea una nueva categoría agregándolo a los patrones. Por su parte las redes ART2 son una modificación a este tipo de red para permitir entradas tanto digitales como analógicas.

Las redes ART2 se encuentran conectadas de acuerdo al diagrama mostrado en la Figura 5.1. Las señales de entrada de las dos primeras se encuentran traslapadas en un porcentaje previamente dado. La tercera red combina los resultados obtenidos por las otras con el objetivo de detectar las principales diferencias en las partes no comunes. El vector de salida de esta última red caracteriza el estado actual del sistema.

El algoritmo de las redes ART realiza una clasificación de los vectores dados como entrada con base a una matriz de pesos, W. Para ello emplea dos parámetros: el parámetro de vigilancia (ρ), el cual determina cuando un nuevo escenario se convierte en un nuevo patrón, y el factor de aprendizaje (η), que establece que tanto deben modificarse las características del patrón que coincidió con el vector de entrada.

Los vectores de entrada de las redes ART2 deben estar normalizados mediante:

$$I = \frac{A}{\|A\|} \tag{5.1}$$

donde I es el vector normalizado, A es el vector de entrada y $\|.\|$ es la norma Euclidiana.

Para determinar al vector de salida se realiza el producto punto entre el vector normalizado y la matriz de pesos. El vector ganador, W_g, será el que arrogue una mayor correspondencia con el de entrada, es decir, el que genere el mayor producto punto. Si el producto ganador, p_g, satisface la Ecuación 5.2 ocurre una adaptación del vector de pesos que lo generó de la forma dada por la Ecuación 5.3. Por el contrario si el producto es menor del umbral fijado por el parámetro de vigilancia el vector de entrada se convierte en un nuevo patrón de la matriz de pesos.

$$p_g \geq \rho \tag{5.2}$$

$$W_g^{nuevo} = \eta * I + (1 - \eta) * W_g^{anterior} \qquad 0 \leq \eta \leq 1 \tag{5.3}$$

La etapa de aprendizaje de las tres redes se realiza fuera de línea con base a datos históricos que contienen tanto escenarios con fallas como sin éstas.

5.3. Desarrollo

Cada una de las redes ART2 se modela como una tarea independiente, así las dos primeras redes tienen que transmitir su vector ganador a la tercera red. Cada una de éstas se ejecuta en un nodo diferente.

Para el caso de estudio se toma como valor de entrada una señal sinusoidal en presencia de ruido gausiano. Por cada pareja de valores ρ y η se clasifican 100 señales, las cuales varían su frecuencia de 1 a 100 hertz. A su vez, ρ y η varían entre 0 y 1 en incrementos de 0.01.

Por cada una de las señales se toman 200 muestras, ahora bien, si cada una de ellas se representa con una variable del tipo double (de 8 bytes de tamaño), cada mensaje tendrá una longitud de 1600 bytes. Para un tamaño de paquete de 1024 bytes se requieren dos paquetes por mensajes.

Por otra parte, para probar por completo las capacidades del sistema se requiere que las tareas sean periódicas. Por lo tanto, cada una de las tareas sólo calcula una pareja de valores ρ y η: de tal forma que para evaluar al sistema completo se requiera ejecutar varias veces las tareas para distintos valores de ρ y η. Para determinar los valores actuales se emplea un archivo de configuración (art2.cfg) el cual es leído por cada una de las tareas inicialmente y modificado de tal forma que la siguiente activación de la tarea continúe donde concluyó la anterior.[1]

Al mismo tiempo, mediante art2.cf, se indica el porcentaje de traslape. El formato de este archivo es muy simple: sólo contiene una línea con tres valores fraccionarios separados por espacios: siendo el primero de ellos el valor de la variable η, seguida del de ρ y finalmente el traslape. Por ejemplo un archivo con la línea:

```
0.27    0.15    0.5
```

indica un valor η de 0.27, ρ de 0.15 y un traslape de 50 %.

5.4. Análisis numérico y simulación

Para implementar el caso de estudio (como se verá en la siguiente sección) se utilizó un ciclo tanto para transmitir como recibir a los mensajes, por lo que la tarea que implementa la tercera red será lanzada cuando el primer par de mensajes destinados a ella arriben y no cuando la totalidad de ellos (200) lo hayan hecho. La implementación se realizó de esta manera con el objetivo de disminuir el tiempo total de ejecución: mientras que en el primer par de nodos se esta procesando al segundo vector de entrada, la tercera red puede estar clasificando los resultados del primero.

Ahora bien, dado que el análisis considera que una tarea que tiene restricciones debe satisfacerlas antes de ser lanzada si el análisis se realizara tal como las redes fueron implementadas se consideraría que la tercera tarea empieza a ejecutarse hasta que las otras dos terminaron de hacerlo, lo cual no corresponde con lo que pasa en el sistema. Por lo tanto, para realizar el análisis numérico y la simulación se modela a las tareas que implementan a las redes como si estuviesen compuesta por subtareas. Cada una de éstas sólo transmite un

[1] Ver Apéndice B código fuente correspondiente a las redes.

mt		subtareas			mm		mensajes	
T_{clk}	$C_{clk} + C_{cs} + C_{pr}$	T_i	C_i	B_i	T_m	C_m	C_p	ς_i
10 ms	30 μs	350 ms	2670 μs	30 μs	10 ms	2500 μs	57 μs	2

Tabla 5.1: Parámetros empleados para realizar el análisis numérico.

mensaje, en lugar de los 100[2] de la tarea completa. Por lo tanto las subtareas que conforman a la tercera red sólo reciben dos mensajes.

La Figura 5.2 ejemplifica como se dividen las tareas. Cada una de las tareas, Figura 5.2(a), se divide en 100 subtareas, Figura 5.2(b). Las subtareas poseen un identificador único (un numero entero): las primeras 100 forman a la primera tarea, las siguientes a la segunda y las últimas 100 a la tercera. Todas las subtareas tienen el mismo periodo que la tarea original, Figura 5.2(c). Ahora bien, pese a que el tiempo para clasificar a los vectores de entrada es diferente (en teoría se requiere más tiempo para los últimos, porque existen más patrones, y por lo tanto mayor cantidad de operaciones a realizar), para realizar el análisis se considerará el tiempo empleado por cada una de las subtareas idéntico y aproximadamente igual a una centésima parte del tiempo empleado por la tarea.

Para determinar el tiempo de ejecución de las tareas y los periodos de las mismas se realizó una etapa de pruebas.Durante ésta solo se ejecutaron las tareas correspondientes al caso de estudio, es decir, el *cluster* se empleó de manera exclusiva para esta aplicación. Se obtuvo que se requiere [3] alrededor de 280 milisegundos por tarea. Ahora bien, este tiempo incluye el empleado por el manejador de mensajes y por el manejador de tareas. Experimentalmente se determinó que se requieren, en promedio, 57 microsegundos para transmitir un paquete. Por su parte el manejador de tareas emplea 30 microsegundos. Si se requieren transmitir 200 paquetes[4], se necesitan 11.4 milisegundos para hacerlo. Por otra parte, el periodo del manejador de tareas es de 10 milisegundos, por lo que en 280 ocurren 28 activaciones de éste: empleando un total de 840 microsegundos. Por lo tanto, para la ejecución de las subtareas se disponen de 267.760 milisegundos, lo que resulta en 2.67 milisegundos por subtarea. El periodo de las subtareas, y por lo tanto el periodo de las tareas, se determina con ayuda del análisis numérico de tal forma que sea posible garantizar la ejecución de las mismas teniendo como plazo la duración del periodo. En la tabla 5.1 se resumen el valor de los parámetros empleados para hacer el análisis numérico.

El plazo de las subtareas se fija al periodo de las mismas. Por su parte el periodo y el plazo de los mensajes son iguales a los de las subtareas que los generan. El análisis se realiza empleando las ecuaciones discutidas en el capítulo 4.

La tabla 5.2 presenta un extracto de los tiempos de respuesta obtenidos mediante el análisis. La primera columna de la tabla contiene al identificador de las subtareas expresado mediante la ecuación $x + (n_{id} * 100)$, donde x es un número entero entre 1 y 100. La razón de hacerlo de esta forma es la de colocar en el mismo renglón; con el objetivo de compararlos; los valores obtenidos para la primera, segunda, tercera, etc. subtarea de cada una de las redes: ya que la subtarea 201 es la primera de la tercera red, al igual que la 101 de la segunda y la 1 de la primera.

[2]Se envían un mensaje por cada señal que se clasifica, como existen 100 señales se transmiten 100 mensajes.

[3]El peor tiempo de ejecución se presenta en la tercera red.

[4]Cada mensaje esta formado por dos paquetes.

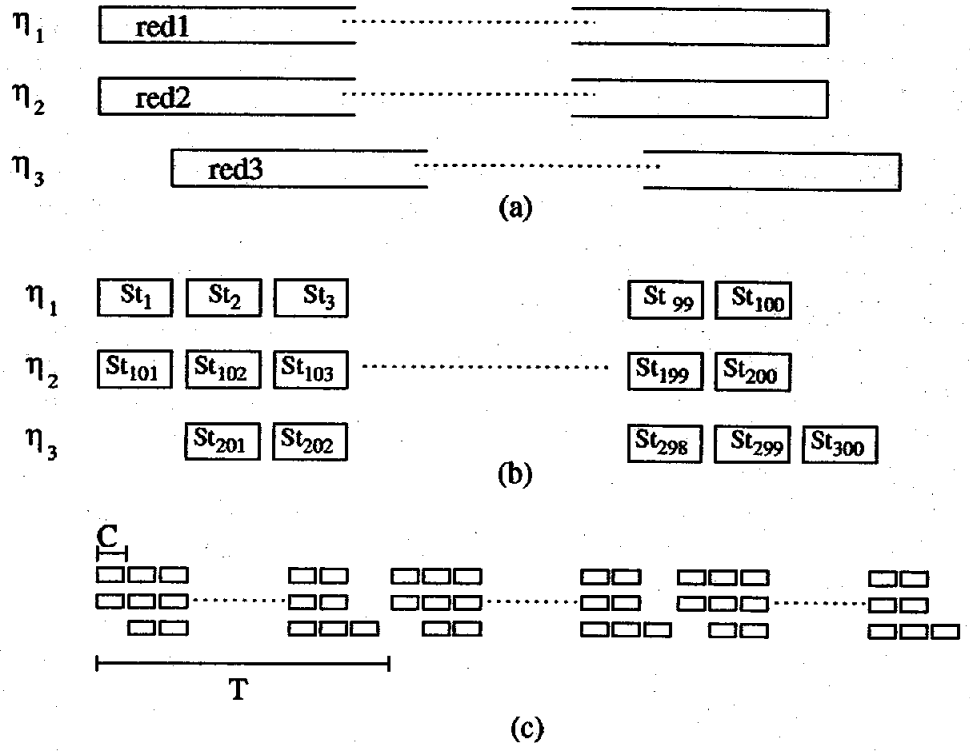


Figura 5.2: División de las tareas en subtareas. (a)Tareas como son implementadas. (b) Tareas divididas en subtareas. (c) Costo y periodo de cada uno de las subtareas.

	red_1 ($n_{id}=0$)		red_2 ($n_{id}=1$)		red_3 ($n_{id}=2$)	
Subtareas	Tiempo R (μs)	Tiempo T (μs)	Tiempo R (μs)	Tiempo T (μs)	Tiempo R (μs)	Fase (μs)
$1+n_{id}*100$	2700	12614	2700	5114	22700	20000
$2+n_{id}*100$	5370	12728	5484	15228	25370	20000
$3+n_{id}*100$	8040	12842	8154	15342	28040	20000
$4+n_{id}*100$	10740	12956	10854	15456	30740	20000
$5+n_{id}*100$	13866	23070	13524	15570	33440	30000
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$43+n_{id}*100$	119720	134902	119844	137402	148780	140000
$44+n_{id}*100$	122430	142516	122544	145016	152700	150000
$45+n_{id}*100$	125556	152630	125556	155130	162700	160000
$46+n_{id}*100$	128226	152744	128226	155244	165370	160000
$47+n_{id}*100$	130926	152858	130926	155358	168040	160000
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$98+n_{id}*100$	273558	323672	273330	316172	332700	330000
$99+n_{id}*100$	276228	323786	276342	326286	335370	330000
$100+n_{id}*100$	278898	323900	279012	326400	338040	330000

Tabla 5.2: Extracto de los tiempos de respuesta arrogados por el análisis numérico.

Para las subtareas de las dos primeras redes se reporta el peor tiempo de respuesta (Tiempo R) y el tiempo en que el mensaje generado por esta arriba a su destino, es decir, su tiempo de transmisión (Tiempo T). Es interesante resaltar que el primer mensaje que llega a su destino es el generado por la primera subtarea de la segunda red (a los 5114 μs), esto ocurre así porque la primera activación del manejador de mensajes en el nodo 1 ocurre a los 2500μs y en ese tiempo la primera subtarea aún no termina, por lo que no hay mensajes que transmitir. Por su parte, en el nodo 2 la activación del mm ocurre a los 5000μs por lo que ya existe un mensaje que transmitir.

Por su parte, para la tercera red además del tiempo de respuesta de las subtareas se incluye el instante en que se activan éstas. La activación ocurre cuando los mensajes ya han llegado a su destino y el manejador de tareas se percata de ello. Esta forma de proceder genera "huecos", ya que habrá instantes en que el procesador quede libre debido a que aún no han llegado los mensajes o que el manejador de tareas no se ha ejecutado: por ejemplo, la subtarea 244 termina a los 152700 μs , pero como los mensajes de los que depende la siguiente no llegan hasta los 152630 μs y los 155130 μs, respectivamente, la subtarea 245 no se activa sino hasta los 160000 μs; por lo que el procesador del nodo 3 estará ocioso durante 7300 μs: tiempo suficiente para ejecutar dos subtareas.

En la Figura 5.3 se presenta un extracto de la gráfica generada por el simulador. En dicha figura cada cuadro horizontal representan 5 ms de simulación. Verticalmente se encuetran cada una de las subtareas. Se presentan las primeras y las últimas cuatro subtareas de cada una de las redes, además de los manejadores de tareas (MT) y manejadores de mensajes (MM) de cada uno de los nodos. En ella se observa fácilmente como las subtareas que pertenecen a la tercera red inician hasta que los mensajes de la primera subtarea han llegado. Por otra parte, el tiempo de respuesta de la última subtarea reportado por el simulador es de 288215 μs en contraparte de los 338040 μs indicados por el análisis. Esta discrepancia se debe al pesimismo inherente del análisis numérico.

Tanto el análisis numérico como la simulación determinan que el conjunto de tareas es planificable dentro de lo parámetros temporales establecidos.

5.5. Codificación

Restando la parte de inicialización de las variables, y algunos detalles de la implementación de las redes ART[5], las Figuras 5.5, 5.6 y 5.7 muestran el código correspondiente a la primera, segunda y tercera red, respectivamente.

La principal diferencia que se puede observar con respecto al paradigma usando en MPI es que existe un programa independiente por cada una de las tareas: lo cual se puede realizar porque es el usuario quien, mediante el archivo de configuración, especifica los identificadores de las tareas. Si embargo, es posible emplear el paradigma "clásico" de MPI, como se muestra en la Figura 5.4.

Como cualquier otro programa escrito en MPI, es necesario incluir la librería "mpi.h" y al igual que emplear a la función RTMPI_Init() antes de cualquier llamada a otra función de este estándar.

[5]El código fuente completo de las tareas que implementan al caso de estudio se encuentra en el Apéndice B.

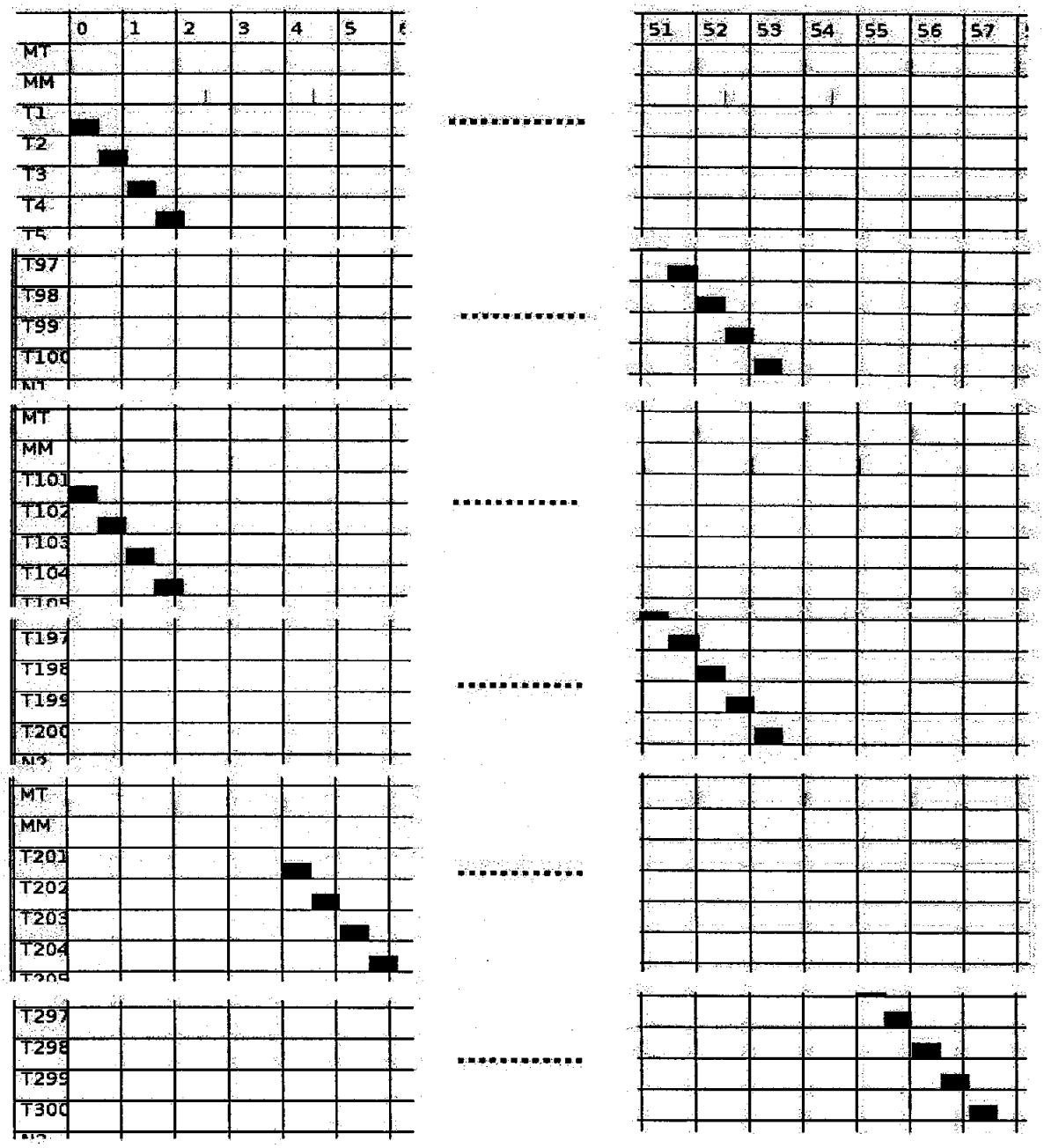


Figura 5.3: Extracto de la gráfica generada por el simulador.

```
1  #include ''mpi.h''
2
3  int int main(int argc, char *argv[])
4  {
5    /*Iniciamos el sistema*/
6    RTMPI_Init(&argc,&argv);
7
8    RTMPI_Comm_rank(myid);
9    if (!strncmp(myid,"t1"))
10     {
11       /*Código de la primera red*/
12     }
13   else if (!strncmp(myid,"t2"))
14     {
15       /*Código de la segunda red*/
16     }
17   else
18     {
19       /*Código de la tercera red*/
20     }
21
22   RTMPI_Finalize();
23   return 0;
24 }
25
```

Figura 5.4: Formato alternativo para implementar las redes como un solo programa.

Básicamente los algoritmos de la primera y segunda red son idénticos, ambos constan de un ciclo, en el cual generan la señal sinusoidal de entrada (Líneas 17 y 12 de las Figuras 5.5 y 5.6, respectivamente), la clasifican mediante la función ART2_eval(...) y finalmente transmiten al vector ganador (G1 y G2, respectivamente) a la tercera red empleando la función RTMPI_Send(). La única diferencia entre éstas, es que la señal de entrada de la segunda red se encuentra desfasada el porcentaje indicado por la variable tras (Líneas 20 y 21 de la Figura 5.6).

Por su parte la tercera red (Figura 5.7) esta compuesta por un ciclo en el cual se reciben los vectores ganadores de las otras redes mediante la función RTMPI_Recv(...). Por otro lado, para garantizar que los mensajes se entreguen correctamente la prioridad de estos disminuye en el orden en que son generados (Líneas 29 y 25 de las Figuras 5.5 y 5.6, respectivamente): mayor prioridad a los primeros, menor a los últimos.

```
#include "mpi.h"

int main(int argc,char *argv[])
{
        .
  tag = 23;
  /*Se inicializa el sistema*/
  RTMPI_Init(&argc,&argv);
        .
        .

  /*Se genera al ruido gausiano*/
  srand(27);
  for(i=0;i<N;i++)
    R[i] = 0.1*rand()/(RAND_MAX+1.0);

  for(j=0;j<100;j++)
  {
    /*Se genera la señal de entrada*/
    for(i=0;i<N;i++)
      I[i] = sin((j + 1.0)*((double)i/(double)N)*6.2832) + R[i];
    normalize(I,N);

    /*Se evalua la red neuranal y se obtiene al vector ganador
    G1 */
    G1 = ART2_eval(&W1,I,rho,eta,&RES_art1[2]);

    /*Transmitimos el  G1 a la red 3*/
    RTMPI_Send(G1,N,MPI_DOUBLE,"t3",tag,255-prio);
    prio++;

  }
     .
     .

  RTMPI_Finalize();
  return 0;
}
```

Figura 5.5: Extracto del código fuente que implementa a la primera red (Tarea 1).

```
#include "mpi.h"

int main(int argc,char *argv[])
{
      .
  tag = 14;
  /*Se inicializa al sistema*/
  RTMPI_Init(&argc,&argv);
      .
      .

  for(j=0;j<100;j++)
  {
    /*Se genera la señal*/
    for(i=0;i<N;i++)
      I[i] = sin((j + 1.0)*((double)i/(double)N)*6.2832) + R[i];
    normalize(I,N);

    /*Se desfasa la señal el porcentaje indicado*/
    for(i=0;i<N;i++)
      TI[i] = I[((int)(i + ceil((1.0 - tras)*N)))%N];

    G2 = ART2_eval(&W2,TI,rho,eta,&RES_art2[2]);
    /*Transmitimos G2 a la red 3*/
    RTMPI_Send(G2,N,MPI_DOUBLE,"t3",tag,255-prio);
    prio++;


   }
        .
        .
        .
  RTMPI_Finalize();
  return 0;
}
```

Figura 5.6: Extracto del código fuente que implementa a la segunda red (Tarea 2).

```
1  #include "mpi.h"
2
3  int main(int argc,char *argv[])
4  {
5
6    /*Iniciamos el sistema*/
7    RTMPI_Init(&argc,&argv);
8            .
9            .
10           .
11   for(j=0;j<100;j++)
12   {
13     /*Se solicitan los datos de la red 1*/
14     RTMPI_Recv(G,N,MPI_DOUBLE,"t1",23,&status);
15     ART2_eval(&W3,G,rho,eta,&RES_art3[2]);
16
17     /*Se solicitan los datos de la red 2*/
18     RTMPI_Recv(G,N,MPI_DOUBLE,"t2",14,&status);
19     ART2_eval(&W3,G,rho,eta,&RES_art3[2]);
20
21   }
22         .
23         .
24         .
25   RTMPI_Finalize();
26   return 0;
27 }
28
```

Figura 5.7: Extracto del código fuente que implementa a la tercera red (Tarea 3).

```
[Sistema]
Nummp=3.0  #número de mp
Numpus=3.0 #número de tareas
time=350000.0 # 10000 rondas de 350 milisegundos


[tarea 0]

Jobid='t1'
Comando='./tarea1'
H='n1'
T=35.0
F=0.0
P=63.0
D='nul'

[tarea 1]
Jobid='t2'
Comando='./tarea2'
H='n3'
T=35.0
F=0.0
P=63.0
D='nul'

[tarea 2]
Jobid='t3'
Comando='./tarea3'
H='n4'
T=35.0
F=0.0
P=63.0
D='2,t1,23,t2,14'
```

Figura 5.8: Archivo de configuración del caso de estudio (rtmpi.cfg).

5.5.1. Ejecución

Una vez que se han realizado los programas (o el programa) que conforman a la aplicación, se requiere realizar los siguientes pasos para ejecutarla.

- Compilar los programas. Para ello se utiliza al compilador gcc de la manera siguiente:

  ```
  gcc -o tarea1 tarea1.c mpi.o xipc.o -lpthread
  ```

  donde tarea1.c es el código del programa que se desea compilar. Por su parte tanto mpi.o como xipc.o contienen la definición de las funciones del estándar MPI.
  Por último para poder emplear las interrupciones de tiempo real es necesario agregar el soporte para hilos mediante la bandera -lpthread.

- Determinar las características temporales de las tareas. Para ello se propone realizar una etapa de pruebas en la cual se miden los tiempos máximos empleados por las

tareas para llevar a cabo sus funciones. Para el caso de estudio se realizaron varios experimentos con el objetivo de determinar estos tiempos, encontrando que en promedio se requiere un poco más de 280 milisegundos para el caso que se tiene un traslape de cincuenta por ciento entre las señales. Por su parte, el periodo de las tareas se obtiene considerando los resultados arrogados por el análisis, dando un valor de 350 milisegundos.

- Crear el archivo de configuración. Una vez que se han determinado las restricciones temporales de las tareas y que se ha establecido en que nodos se han de ejecutar: resulta simple crear el archivo de configuración. El correspondiente al caso de estudio se observa en la Figura 5.8. En la parte destinada a las características del sistema se indica que se tienen tres manejadores de tareas y tres tareas. A continuación, mediante la variable **time**, se especifica cuanto ha de durar la ejecución del sistema: el tiempo estimado es de aproximadamente 58 minutos. Este valor se determina considerando que se requieren 10000 activaciones para calcular todos las posibles combinaciones de ρ y η [6], con una duración 350 milisegundos de cada una de ellas. Este valor se especifica en un unidades de tick de reloj (10 milisegundos). Cada una de las tareas se describe dentro de su propia sección ([tarea x]) : destacando el caso de la última, la cual depende de las otras dos como lo establece la cadena[7] D='2,t1,23,t2,14'. Dicha cadena indica que la tarea depende de dos mensajes: con identificadores 23 y 14; y generados por las tareas t1 y t2, respectivamente.

- Finalmente en cada uno de los nodos se ejecuta el comando:

```
mpiexec -configfile rtmpi.cfg
```

donde rtmpi.cfg es el archivo anteriormente creado. No existe ninguna restricción en el orden en que se debe lanzar el comando en los nodos, ni siquiera tiene que ejecutarse de manera simultanea, ya que los manejadores de tareas "esperan" a que la cantidad de ellos indicada por el archivo de configuración sea lanzada para sincronizarse y empezar con la ejecución de las tareas.

5.6. Validez de los resultados

Como se ha mencionado anteriormente las aplicaciones de tiempo real requieren no sólo que el algoritmo sea implementado correctamente sino además que los resultados que generan deben ocurrir en el instante establecido.

En el caso de estudio, el objetivo del algoritmo es agrupar las señales de entrada de acuerdo a su similitud. El factor ρ indica que tan similares deben ser dos señales para que pertenezcan al mismo grupo: siendo 1 la máxima posible. Por lo tanto, se esperaría que conforme el valor de éste parámetro aumente, el número de escenarios necesarios para clasificar las señales también lo haga.

Por otro lado, η determina cuanto se debe modificar al escenario patrón para adecuarlo a la nueva señal que coincidió con éste. Como lo establece la ecuación 5.3, si este valor es igual

[6] 100 valores de η por 100 valores de ρ

[7] Ver el capítulo 2, subsección 2.4.5, para una explicación detallada del formato del archivo de configuración.

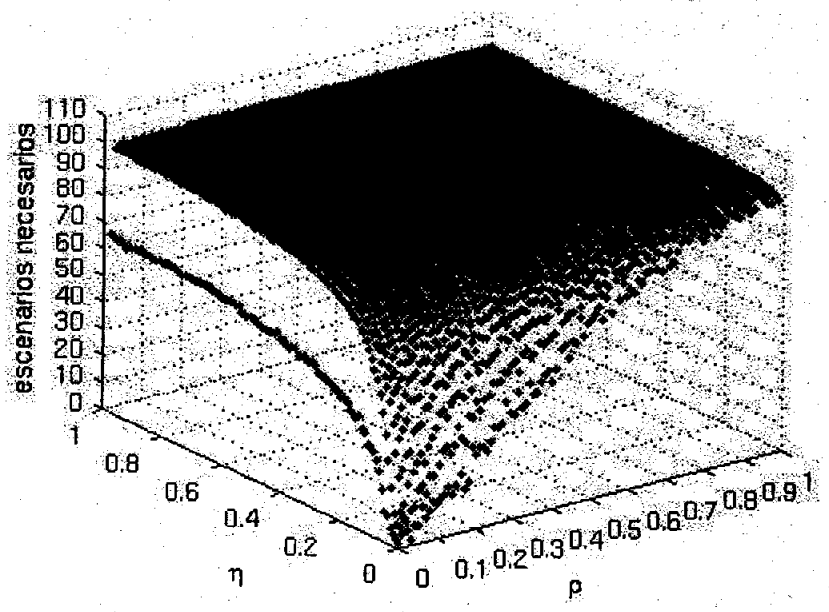


Figura 5.9: Número de escenarios necesarios por la primera red.

a 1 la nueva señal se convierte en un nuevo patrón. Por lo tanto, de manera similar a ρ, se esperaría que el número de patrones aumente conforme η lo haga.

En las Figuras 5.9, 5.10 y 5.11 se gráfica el número de patrones necesarios como función de los parámetros ρ y η: para el caso que las señales tienen un traslape del cincuenta por ciento. En todas ellas se verifica que el número de patrones necesarios para clasificar a las señales se incrementa conforme lo hacen ρ y η, llegando al máximo de ellos (100 de 100 señales por clasificar para el caso de las dos primeras redes, y 150 de 200 señales para el caso de la tercera red) cuando ambos tienen el valor de 1. Por lo tanto, se verifica que el algoritmo se implementó de manera correcta.

Por otro lado, el tiempo empleado por los tareas es medido mediante la función gettimeofday() [8] y almacenado en un archivo para su posterior procesamiento. Para el caso mostrado en las Figuras 5.9, 5.10 y 5.11, se emplearon alrededor de 48 minutos, 10 menos de lo estimado. Con el propósito de realizar una gráfica más legible se agruparon con base al valor de ρ los distintos valores de η, es decir, se sumaron los tiempos obtenidos para todas las parejas con el mismo valor de ρ. Dando 99 valores (esto porque el caso donde tanto ρ como η son iguales a 0 no fue explorado) por red a graficar. La gráfica resultante se muestra en la Figura 5.12. Como se observa en todos los casos el máximo se mantuvo dentro del tiempo esperado, con lo cual se garantiza que las condiciones temporales impuestas al sistema fueron cumplidas.

[8] Ver Apéndice B parte final del código de las redes uno, dos y tres.

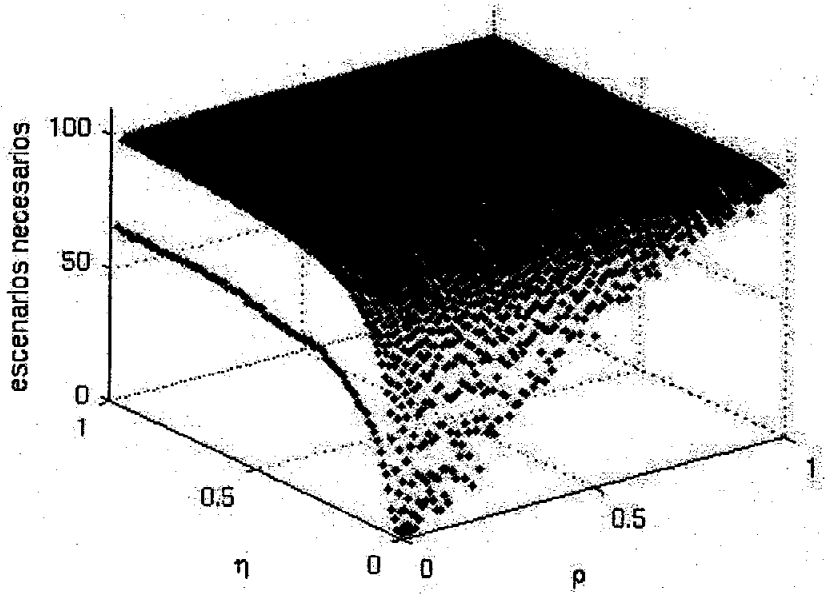


Figura 5.10: Número de escenarios necesarios por la segunda red.

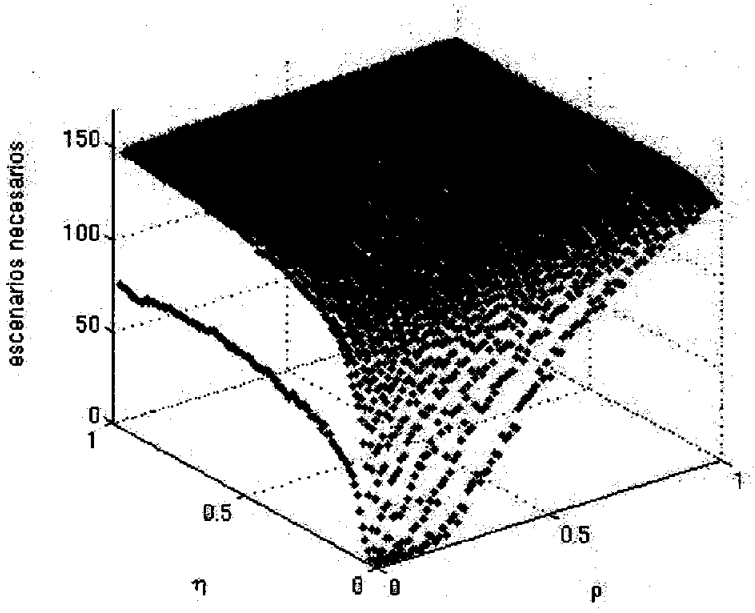


Figura 5.11: Número de escenarios necesarios por la tercera red.

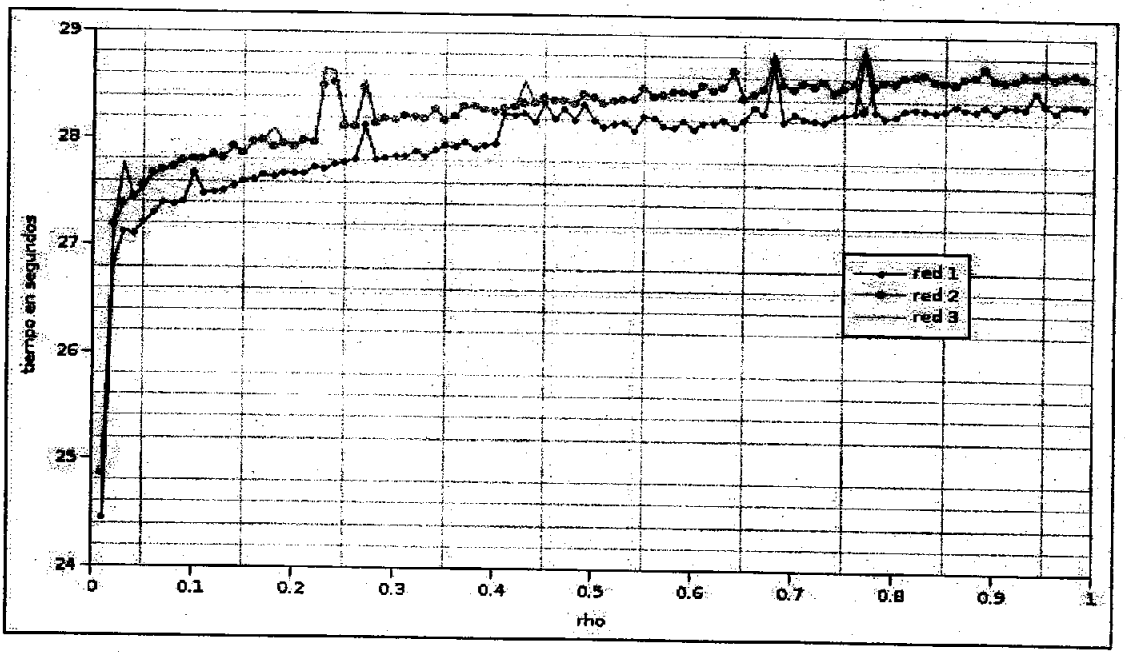


Figura 5.12: Tiempo empleado por cada una de las rondas.

5.7. Conclusiones

Mediante el empleo de un caso de estudio fue posible verificar que la plataforma proporciona el soporte de tiempo real para el envío de mensajes tal como se esperaba. Además dicho caso sirve como ejemplo de la forma como debe realizarse el análisis numérico.

Por otro lado, el desarrollo de la aplicación se hizo empleando las mismas funciones del estándar MPI discutidas en el capítulo anterior. También se discutió la posibilidad de emplear un programa por tarea lo cual no es posible en otras implementaciones del estándar. A su vez, se indicó la manera en que la aplicación debe ejecutarse.

Finalmente, se verificó que la aplicación desarrollada cumpliera tanto con las restricciones temporales como que el algoritmo fuera correcto.

Capítulo 6

Resultados

6.1. Introducción

Tradicionalmente el enfoque de tiempo real se ha aplicado a sistemas de control como el caso de la aeronáutica, control de procesos industriales, robótica, así como en la electrónica del automóvil. Siendo la principal características de estos sistemas estar fuertemente acoplados, es decir, la aplicación se construye es profeso para ese sistema.

Por otra lado, los *clusters* de alto rendimiento se emplean para ejecutar programas computacionalmente intensivos: por lo general de especial interés dentro de la comunidad científica. Ejemplos de estas aplicaciones incluyen simulación de proteínas, modelos financieros, procesamiento de información proporcionada por sismógrafos en busca de yacimientos petroleros a sí como la posible predicción de terremotos.

Combinar estas dos ramas del conocimiento es un campo abierto a la experimentación y a las propuestas. En este trabajo se plantea una de ellas: emplear el paradigma de envío de mensajes, en particular al estándar MPI, para crear una plataforma de desarrollo que de soporte a aplicaciones de tiempo real en sistemas de alto desempeño.

En los capítulos precedentes se ha detallado las características de esta propuesta, al igual que se ha presentado un ejemplo de como utilizarla. Finalmente, en este capítulo se describen los principales resultados obtenidos.

6.2. Arquitectura

El empleo de una tarea cuyo único objetivo fuese la transmisión de los mensajes (el manejador de mensajes) permitió tratar a estos de manera similar a las tareas, es decir, asignarles una prioridad así como determinar el tiempo máximo empleado para transmitirlos, con lo cual se logró que el sistema fuese predecible.

A su vez la existencia del manejador de tareas hizo posible que éstas pudiesen recibir cualquier cantidad de mensajes. Además, ambos manejadores, facilitaron poder monitorizar temporalmente al sistema, de tal forma que se pudiese realizarse una etapa de pruebas que permitiera establecer las características temporales de las tareas.

Por otro lado, aunque esta arquitectura, se diseñó con el objetivo de ser implementada en un *cluster*, es lo suficientemente general para poder ser portada a otras plataformas. Con el mismo fin, permitir la portabilidad, la aplicación se desarrolló empleando funciones del

estándar POSIX, por lo que, un posible trabajo futuro sería utilizar un sistema operativo de tiempo real en sustitución de GNU/Linux con el objetivo de soportar aplicaciones de tiempo real duro.

6.2.1. Red

En tiempo real el canal de comunicación es un recurso valioso, de ahí que por lo general se emplean redes y protocolos específicos para estos fines. Ejemplos de estas redes son: CAN (Control Area Network) diseñada originalmente para su aplicación en vehículos; DeviceNet utiliza como base el bus CAN, e incorpora una capa de aplicación orientada a objetos; Profibus diseñada también para la industria automotriz; ProfiNet diseñada para manejar simultáneamente transmisiones basadas en el estándar Ethernet juntamente con transmisiones de tiempo real hechas mediante Profibus.

Sin embargo en cómputo de alto rendimiento, por lo general, se transmiten grandes cantidades de datos, y estas redes, aunque predecibles, no tienen capacidad para soportarlo. Por otra parte, la red Ethernet, ampliamente difundida, no esta diseñada para garantizar el tiempo que le llevará transmitir un dato; esto se debe principalmente a la forma que tiene de acceder al medio: cuando existe una colisión se espera un tiempo aleatorio para volver a intentar transmitir. Sin contar que el principal grupo de protocolos que se emplea en esta plataforma son los formados por el conjunto TCP/IP, los cuales fueron diseñados para ser altamente confiables y cuando existe un error en la transmisión de los datos se privilegia la confiabilidad en decremento de la duración de la transmisión: por lo general se tienen ilimitadas retransmisiones por lo que es imposible establecer un límite en el tiempo de transmisión.

Debido a esto se optó por emplear la red SCI: la cual cuenta con poco latencia y tiene la capacidad de transmitir grandes volúmenes de información. Uno de los principales logros del uso de este canal de comunicación, fue el corroborar la baja latencia de esta red. Durante el experimento realizado con el objetivo de determinar el tiempo de transmisión empleado por los mensajes se observó una variación máxima de 2 microsegundos en la transmisión de mensajes del mismo tamaño. Los características de este experimento se detallan en la siguiente sección.

6.3. Análisis

El desarrollo de las ecuaciones que permiten determinar analíticamente la planificabilidad de un conjunto de tareas y mensajes, junto con la arquitectura son las principales contribuciones de este trabajo.

Sin embargo, para realizar el análisis numérico se requiere conocer ciertos parámetros que caracterizan al sistema. Algunos de ellos como la duración del tick son especificados por el usuario y otros deben ser obtenidos experimentalmente. El tiempo empleado por el manejador de tareas es uno de estos últimos; el cual se determinó durante la etapa de pruebas a las que se sometió el caso de estudio encontrando un valor máximo de 30 microsegundos.

Por otro lado, para determinar el tiempo de transmisión y el tiempo que le toma a una tarea copiar sus mensajes a la cola de salida fue necesario desarrollar experimentos específicos. El tiempo que le toma a una tarea copiar sus mensajes a la cola de salida, es decir, el tiempo

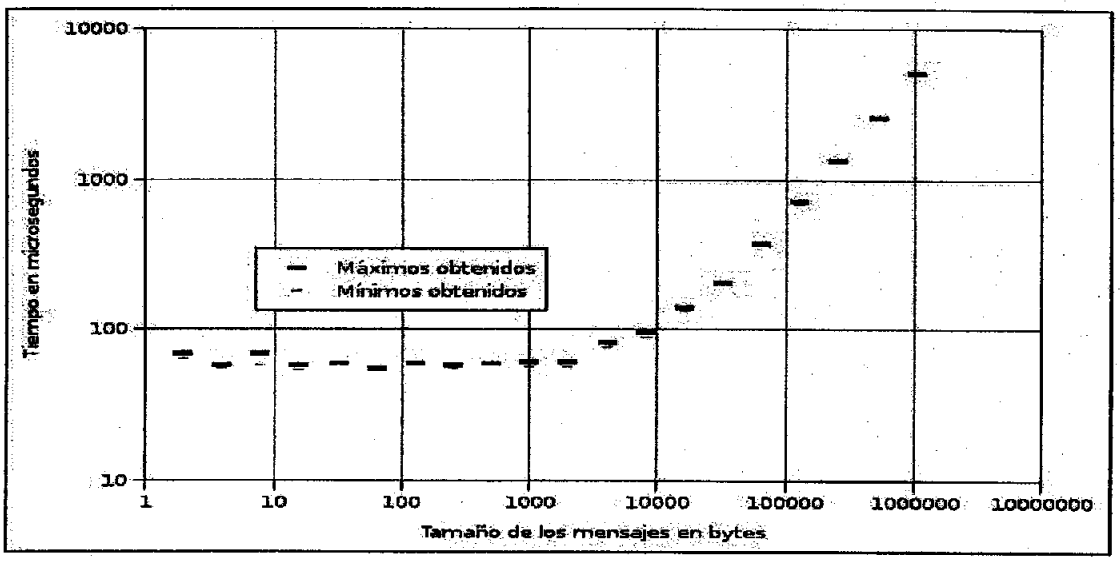


Figura 6.1: Tiempo empleado para colocar los mensajes en la cola de mensajes a enviar.

de bloqueo, se determina fácilmente consultando el tiempo actual antes y después del uso de la función RTMPI_Send(...), ya que es esta tarea la que copia los mensajes a dicha cola. En la Figura 6.1 se observan los resultados obtenidos para distintos tamaños del mensaje: el tamaño del mensaje se varió de 2 bytes hasta 1 Mega byte en múltiplos de las potencias de 2. El experimento se realizó 100 ocasiones y en la Figura se muestran los valores máximos y mínimos encontrados, teniendo estos una diferencia promedio de 6 microsegundos. Es interesante notar que el tiempo para copiar ha esta cola es prácticamente constante dentro de los primeros 4 kilo bytes de datos, oscila entre los 54 y 61 microsegundos, a partir de los cuales el tiempo se incrementa exponencialmente.

Determinar el tiempo de transmisión desde que el mensaje es creado hasta que llega a su destino es más complicado, especialmente por el ruido que pueden causar el manejador de tareas y el manejador de mensajes. Por lo tanto, para determinar este parámetro, se limitó la participación de estas tareas: de tal forma que en cuanto los datos son puestos en la cola de salida estos se trasmiten. Por su parte las tareas fueron lanzadas simultáneamente, aprovechando la sincronización de los manejadores de tareas. Para medir el tiempo de transmisión se modificó a la función RTMPI_Recv(...) de tal forma que suspendiera a la tarea que la llama hasta que lo mensajes estuviesen disponibles, por lo tanto para determinar este valor solo se requiere consultar el tiempo antes y después de esta función.

En la Figura 6.2 se muestran el valor máximo y el valor mínimo obtenido durante 100 repeticiones del experimento. El tamaño de los mensajes se varió de forma similar al experimento anterior. De manera similar al tiempo empleado para colocar los mensajes en la cola, este tiempo permanece prácticamente constante e igual a 114 microsegundos siempre y cuando el tamaño de los mensajes sea menor o igual a 4 kilo bytes. Las variaciones entre el máximo y mínimo tiempo empleados por mensajes del mismo tamaño no sobrepasaron los 5 microsegundos.

Otra aportación realizada es que se realizaron una serie de experimentos con el objetivo de demostrar la validez del análisis numérico de manera estadística. También se desarrolló un simulador con el objetivo de ayudar al desarrollador en la tarea de determinar la manera en

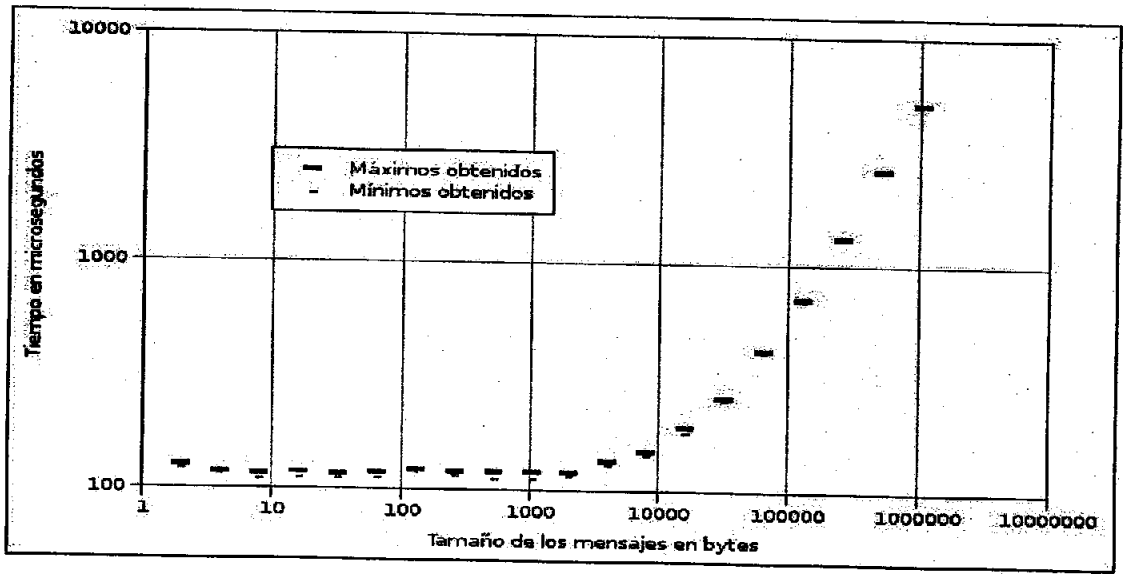


Figura 6.2: Tiempo de transmisión desde que el mensaje es creado hasta que llega a su destino.

que se ha de realizar la asignación de las tareas a los nodos. Dicho simulador también se empleo para determinar que tan pesimista es el análisis propuesto.

6.4. Conclusiones

Al margen que existen tecnologías como CORBA que están haciendo su incursión a los sistemas de tiempo real, el paradigma de envío de mensaje es el que sigue predominando dentro del campo de sistemas distribuidos. Por lo tanto, dotar ha esta tecnología con soporte para tiempo real disminuirá la curva de aprendizaje, al mismo tiempo que motivará a sus usuarios para explorar el campo de tiempo real.

El trabajo realizado ha permitido la construcción de una plataforma de desarrollo con soporte para aplicaciones de tiempo real en sistemas de cómputo de alto rendimiento mediante el paradigma de mensajes.

Capítulo 7

Conclusiones

En el presente trabajo se planteó como objetivo desarrollar un sistema de alto desempeño que contenga acotamientos en el tiempo de ejecución de tareas dando por hecho un comportamiento en tiempo real del sistema en su conjunto.

Para cumplirlo se propuso una arquitectura compuesta por dos tareas: el manejador de tareas y el manejador de mensajes; lo cual permitió tratar de manera similar a las tareas como a los mensajes, es decir, se les pudo asignar una prioridad, determinar sus tiempos máximos de ejecución y transmisión, hacerlos de ejecución preferente (tanto la tarea como el mensaje de mayor prioridad suspenden al de menor para ejecutarse -transmitirse- en su lugar). Con lo cual se logró cumplir satisfactoriamente con el objetivo.

Por otro lado, al margen de los conocimientos obtenidos (tanto de tiempo Real, como de técnicas de programación), el análisis de planificabilidad desarrollado para determinar que un conjunto de tareas y mensajes en esta plataforma satisfacen con sus requerimientos temporales, así como demostrar, mediante un caso de estudio, que la arquitectura cumple con los requerimientos planteados son los principales logros de ésta tesis.

7.1. Trabajo Futuro

El trabajo desarrollado ha permitido probar que es posible realizar aplicaciones de tiempo real en un sistema de alto desempeño utilizando el paradigma de envío de mensajes para comunicar a las tareas. Dicha arquitectura permite la construcción de sistemas que requieran garantizar la calidad de los servicios que ofrecen, como sería el caso de la transmisión de vídeo bajo demanda.

Desgraciadamente, debido a que se utilizó un sistema operativo de propósito general para realizar la implementación, no es posible emplear esta plataforma para aplicaciones de misión crítica. Por lo que una futura contribución a ésta sería su aportación a un sistema operativo de tiempo real como sería el caso de RTLinux.

Por otro lado, el emplear una tecnología como SCI para la red de datos garantiza una baja latencia en la transmisión, pero el sistema resultante es poco escalable y además su costo podría hacerlo inviable. Una alternativa sería utilizar tarjetas Ethernet de alta velocidad (100 MB - 1GB), que combinadas con un sistema operativo de tiempo real y eliminando al conjunto de protocolos TCP/IP se logra latencias de transmisión de unas cuantas decenas de microsegundos. También es posible, dependiendo el tipo de tarjeta, mediante software,

evitar que éstas hagan uso de su algoritmo de acceso al medio; sin embargo, para evitar la perdida de datos se requiere de un protocolo que garantice que no existirán colisiones.

Otra contribución futura al presente trabajo sería el continuar con la implementación del resto de las funciones que forman al estándar MPI. De especial interés serían las que tienen que ver con comunicaciones *multicast* y *broadcast*. Una forma de implementar comunicaciones *multicast* sería hacer que la etiqueta que dentro del paquete indica al destinatario aceptase múltiples valores, el manejador de mensajes revisaría ésta y determinaría en que nodos están las tareas destino, generando un mensaje por nodo. Sin embargo existen múltiples problemas inherentes de resolver, por ejemplo ¿a que nodo se le enviaría primero el mensaje?, ¿como manejar la prioridad de las comunicaciones *multicast*?

7.2. Comentarios Finales

No obstante, el envío de mensajes sea el principal paradigma empleado en los *clusters* en la actualidad, como se ha venido mencionando, no es la única tecnología disponible para realizar aplicaciones en sistemas distribuidos. En especial las tecnologías basadas en ORB (*Object Request Broker*) como son CORBA, COM+, .NET, SOAP, cuentan con gran aceptación y el número de aplicaciones para ellas sigue en aumento.

En el campo de tiempo real y sistemas de alto desempeño TAO (basada en CORBA, y por tanto en ORB) es la tecnología que lleva la delantera. Pero, aún no hay nada definitivo, y por lo general no existen soluciones únicas, sino más bien dependiendo de la aplicación se selecciona a la que más se adapte a las necesidades. Por lo que, es muy probable que conforme las necesidades por sistemas de tiempo real aumenten el estándar MPI evolucione para satisfacer éstas, además, puedan surgir otras tecnologías.

Por el lado de los sistemas operativos es un hecho que cada vez más incorporarán características de los sistemas de tiempo real, como los algoritmos de planificación, para soportar de mejor manera tareas consideradas como críticas.

El tiempo real y los sistemas de alto desempeño son dos tecnologías que poco a poco abarcan más áreas de nuestra vida diaria.

Apéndice A

Glosario

A

ART (***Adaptive Resonance Theory***): Teoría de Resonancia Adaptiva. Tipo de red neuronal empleado como clasificador de vectores: toma un vector analógico de entrada y lo clasifica dentro de una categoría con base a un grupo de patrones previamente almacenados. Las redes ART2 son una modificación a este tipo de red para permitir tanto entradas digitales como analógicas.

Traslape de redes ART (***Overlappend ART Networks; OAN***): Técnica de localización de fallas propuesta en [Benítez-Pérez y García-Nocetti,] que emplea tres redes neuronales del tipo ART2 para cumplir con su objetivo. Las señales de las dos primeras redes se encuentran traslapadas. Por su parte la tercera red recibe como entrada los resultados de las otras dos y determina que tan "parecidas" son estos.

B

Barrera: Técnica de sincronización empleada en sistemas de cómputo distribuidos que consiste en forzar que las tareas que conforman la aplicación esperen hasta que todas hayan alcanzado el mismo punto de ejecución: a dicho punto es al que se le conoce como barrera.

C

Calendarizador: Tarea o rutina que se encarga de determinar cual es la siguiente tarea que se ha de ejecutar. Puede ser tan simple como una lista creada fuera de línea que indique el orden en que se han de ejecutar las tareas o puede emplear complejos algoritmos dependiendo cual sea su objetivo: lograr ejecutar al mayor número de tareas en el menor tiempo posible, garantizar los tiempos de ejecución de las tareas, etc.

Cluster: Conjunto de computadoras conectadas mediante una red de datos configuradas de tal forma que se comportan como si se tratasen de una sola. Por lo general en los

clusters a las computadoras que lo forman se les denomina nodos; distinguiendo dos tipos: nodo maestro y nodos esclavos. El nodo maestro es el encargado de mantener la configuración del sistema.

CORBA (*Common Object Request Broker Arquitecture*): Es un estándar para la construcción de aplicaciones basadas en objetos remotos distribuidos, permitiendo que los objetos interactúen entre sí independientemente de la plataforma y lenguaje empleado para programarlos.

F

Frecuencia Monótona (*Rate Monotonic; RM*): Esquema de planificación en tiempo real para tareas periódicas que consiste en asignarle la mayor prioridad a la tarea con el periodo más pequeño, es decir, a la que tiene la frecuencia mayor.

G

GNU: El proyecto GNU [Proyecto GNU]fue creando en 1984 con el objetivo de desarrollar un sistema operativo con características similares a UNIX, el cual fuera software libre. GNU es un acronimo recursivo para "GNU's Not UNIX". La Fundación de Software Libre (*Free Software Foundation*) es la principal institución detras del proyecto GNU. La misión de la FSF es preserver, proteger y promover la libertad para usar, estudiar, copiar, modificar y redistribuir software, a su vez que defender los derechos de los usuarios del software libre.

GNU/Linux: El principal objetivo de GNU de crear un sistema operativo completo aún no se ha cumplido del todo debido a que el nucleo del sistema operativo GNU, llamado GNU Hurd, aún no se encuentra listo para producción. Afortunadamente, otro núcleo se encuentra disponible. En 1991, Linus Torvalds desarrolló un nucleo compatible con UNIX, llamado Linux. Alrededor de 1992, la combinación de Linux con las herramientas desarrolladas para al sistema GNU (compiladores, editores, ambientes gáficos, etc.) originaron un sistema operativo completo. A este sistema se le conoce como GNU/Linux, con el objetivo de indicar la combinación de estos dos sistemas.

I

IRQ (*Interrupt Request Line*; Linea de Solicitud de Interrupción): Son puntos de acceso (líneas físicas, regiones de memoria) através de las cuales los dispositivos, como el disco duro o tarjetas de red, pueden enviar señales de interrupción al procesador.

J

Jitter: La expresión *jitter* se emplea para referirse a variaciones que se presentan de un valor promedio.

Jitter de activación (*release jitter*): Es el *jitter* que se presenta en el instante de activación de las tareas o de los mensajes.

M

Manejador de mensajes(mm): Tarea base de RTMPI, la cual se encarga de transmitir los mensajes destinados a los nodos remotos. Cada mensaje es transmitido con base a su prioridad, y el que estén divididos en paquetes permite suspender su transmisión para enviar a otros de mayor prioridad.

Manejador de tareas (mt): Tarea base de RTMPI, la cual se encarga de ejecutar las tareas que forman a la aplicación con base a su prioridad. Dentro de sus funciones está el determinar cuando se han satisfecho las precedencias de una tarea para que ésta pueda empezar a ejecutarse.

Menor Tiempo Sobrante Primero (*Least Laxity Firts: LLF*): Esquema de planificación de tiempo real en el cual se determina cual es el tiempo sobrante (máximo tiempo que una tarea puede retrasar su ejecución sin perder su plazo) de la tarea, y a la tarea con menor tiempo sobrante es a la que se le asigna mayor prioridad.

MPI (*Message Passing Interface*): MPI es la especificación de una librería para el envío de mensajes, propuesta como un estándar por un comite de vendedores, desarrolladores y usuarios.

Mpich: Principal implementación del estándar MPI.

O

ORB (*Object Request Broker*): Es la base de las tecnologías de objetos distribuidos. ORB representa la habilidad para manipular objetos remotamente. Su implementación más sencilla provee un medio para que los clientes puedan solicitar que se realicen operaciones, y pasar parámetros entre el servidor y los clientes.

P

Paso de mensajes: Paradigma empleado en sistemas distribuidos que consiste en la habilidad de las tareas de enviarse mensajes entre sí.

Plazo Monótono (*Deadline Monotonic; DM*): Esquema de planificación en tiempo real que consiste en asignarle la mayor prioridad a la tarea con el plazo más pequeño.

Plazo mas Cercano Primero (*Earliest Deadline Firts; EDF*): Esquema de planificación en tiempo real que consiste en asignarle la mayor prioridad a la tarea con el plazo mas cercano con respecto al tiempo actual.

POSIX (Portable Operating System Interface): Un estándar basado en UNIX desarrollado por la IEEE. En 1988, el comite encargado emitio la primera version del estándar (1003.1), el cual representa la parta principal de las llamadas al sistema. La existencia de este estándar permite a los desarrolladores construir aplicaciones diseñadas para un ambiente en especifico en lugar que preocuparse por el sistema operativo donde serán ejecutadas. POSIX define características del sistema operativo, manejadores de bases de datos, servicios de red, interfaces de usuario y ambientes de programación.

Preemptive: Termino que se le aplica a las tareas que tienen la capacidad de suspender su ejecución para que otra de mayor prioridad se ejecute en su lugar. Posteriormente continua en el punto donde fue interrumpida. En esta tesis se empleó la frase "ceder el turno" como sinónimo de éste.

PVM (Máquina Virtual Paralela): Es una aplicación de software que permite a un conjuto de sistemas operativos heterogeneos dependientes de una misma red ser usados como si se tratasen de una sola computadora paralela. De esta forma problemas computacionalmente demandantes pueden ser resueltos.

R

RMA (*Remote Memory Access*): Es una tecnología que permite a las computadoras que comparten una red intercambiar información existente en su memoria de manera similar a la forma con la que accesan a datos almacenados en su memoria local.

Round-robin: Esquema de planifiación que consiste en asignar durante un determinado monto de tiempo el procesador a una tarea, cuando el tiempo ha expirado se continua con la siguiente y así sucesivamente. Cuando todas las tareas han sido atendidas de esta forma se vuelve a empezar hasta que todas las tareas sean completadas.

S

SCI (*Scalable Coherent Interface*): El estándar SCI (ANSI/IEEE 1596-1992) define una interfaz tanto de software como hardware para comunicaciones punto a punto. Los protocolos de SCI soportan comunicaciones mediante memoria compartida mediante la encapsulación de las instrucciones de solicitud y de respuesta en paquetes SCI.

Server design: Técnica propuesta en [Almeida y Pedreiras, 2004] para diseñar al servidor mínimo que sea capaz de ejecutar a un conjunto de tareas periódicas.

Sistemas de tiempo real: Es aquel en el que la correctez de las operaciones computacionales no solo depende que la lógica e implementación de los algoritmos lo sea, sino también en el tiempo en el que se entrega su resultado. Si las restricciones de tiempo no son respetadas el sistema ha fallado.

tiempo real duro (hard): Sistema de tiempo real en el cuál si las tareas no cumplen con sus plazos se tienen consecuencias catastróficas.

tiempo real firme (firm): Sistema de tiempo real en el cuál si los resultados de las tareas no son obtenidos dentro de su plazo no tienen ninguna utilidad, sin que existan consecuencias catastróficas.

tiempo real suave (soft): Sistema de tiempo real en el cuál se tolera que algunas tareas puedan no cumplir con sus plazos.

T

Tareas: Programa que realiza una actividad especifica. También se emplea para indicar cada una de las partes que conforman a una aplicación distribuida.

Tareas de tiempo real: Tareas que tienen restricciones temporales. Por lo general se especifican mediante un conjunto de parámetros como son:

Tiempo de activación (*Release time*): es el instante de tiempo en el cual la tarea se convierte en elegible para ejecutarse. Por su parte, cuando las tareas son periódicas y existen variaciones en su tiempo de activación se dice que la tarea sufre de *release jitter.*

Plazo absoluto (*Deadline*): es el instante de tiempo en el cual la tarea ya debe estar completada.

Plazo_relativo (*Relative deadline*): es el plazo de la tarea con respecto al instante de activación de la tarea.

Peor tiempo de ejecución (*Worts-case executing time*): es la cantidad de tiempo necesaria para que la tarea complete su ejecución considerando que se ejecuta sola y que tiene todos los recursos disponibles. Determinar este tiempo es fundamental para el correcto análisis del sistema. Una forma aproximada de obtener este valor es realizar una etapa de pruebas y emplear el mayor tiempo observado durante ésta. Sin embargo, si el sistema que se esté desarrollando requiere mayor precisión es necesario emplear métodos más confiables, como los descritos en [Cheng M.K., 2002], con este fin, esto debido a que el peor tiempo obtenido mediante experimentación puede no ser igual al encontrado cuando el sistema se encuentre en producción.

Ceder su turno (*Preemptivity*): es la capacidad que tiene o no una tarea de ser interrumpida, y después reanudada en el mismo punto, para ejecutar otras tareas de mayor prioridad.

Empleo de recursos (*Resource requirements*): este parámetro indica los recursos que necesita la tarea para ejecutarse así como por cuanto tiempo hará uso de ellos.

Tareas periódicas: Tareas que se repiten cada determinado tiempo. En el caso que se tenga tareas periódicas, además de los parámetros anteriores, éstas se definen mediante su **periodo** y su **fase** inicial. El periodo corresponde al tiempo entre dos activaciones consecutivas, ahora bien, si los tiempos de activación sufren de *jitter* para calcular el periodo se emplea el promedio de los intervalos entre activaciones consecutivas. Por su parte la fase inicial es igual al instante de activación de la primera instancia de la tarea.

TDMA (***Time Division Multiplexed Access***): Protocolo empleado para arbitrar el acceso a los medios de comunicación. Consiste en asignar un determinado tiempo a cada uno de los participantes para que use exclusivamente el canal de comunicación.

TDMA round: Recibe este nombre el periodo de tiempo necesario para que todos los participantes usen por lo menos una vez el canal de comunicación, es decir, es el tiempo que se requiere para que la secuncia en la cual los participantes accesan al medio se repita.

TDMA slot: Recibe este nombre cada una de las divisiones temporales. La duración de cada una de ellas puede ser diferente.

Apéndice B

Código fuente del caso de estudio

En este apéndice se incluye el código fuente de las tareas que forman parte del caso de estudio. En la sección B.1, se presentan las funciones que implementan al algoritmo de las redes ART. Por su parte la sección B.2, contiene el código correspondiente a la primera red ART, la sección B.3 a la segunda. Finalmente el código de la tercera red se describe en la sección B.4.

B.1. Implementación del Algoritmo de las redes ART2

art2.h

```
/* Nombre: art2.h */
/* Autor: Miguel Angel Palomera Perez */
/* Descripcion: Clase art2 */

#ifndef _ART2_H_
#define _ART2_H_
/*Tipo matriz*/
typedef char * Matriz;
typedef  double * Vector;
/*Tipo de dato empleado por matlab*/
typedef struct
{
  long type;
  long mrows;
  long ncols;
  long imagf;
  long namelen;
}Fmatrix;


/*Funciones para manipular la matriz*/
int M_init(Matriz *W,int r,int c);
int M_append_col(Matriz *W,Vector I);

/*Funciones para manipular la red*/
Vector ART2_eval(Matriz *W,Vector I,double rho,double eta,double *col);

/*Funciones de utileria*/
/*Función para salvar los datos en el formato empleado por matlab*/
int saveW(char *archivo,Matriz W,char *name);
```

```
/*Función para determinar el tiempo transcurrido*/
int time_elapsed(struct timeval *result, struct timeval *x, struct timeval *cy);

/*Macros*/
#define REN(l) *((int *)(l))
#define COL(l) *((int *)((l)+sizeof(int)))
#define XY(M,x,y) *((double *)(M + 2*sizeof(int) + y*REN(M)* sizeof(double)\
                    + x*sizeof(double)))
#define size_of(l) 2*sizeof(int) + REN(l) * COL(l) * sizeof(double)

#define M_multi_vec_matriz(I,W,C)\
{\
 int i,j;\
 for(j=0;j<COL(W);j++)\
    {\
      C[j] = 0.0;\
      for(i=0;i<REN(W);i++)\
        C[j] +=  I[i] * XY(W,i,j);\
    }\
}

#define normalize(I,size)\
{\
  double sum;\
  int i;\
  sum = 0.0;\
  for(i=0;i<size;i++)\
    sum += I[i] * I[i];\
  sum = sqrt(sum);\
  for(i=0;i<size;i++)\
    I[i] = I[i] / sum;}

#define V_max_pos(I,size,val,pos)\
{\
  int i;\
  val=I[0];\
  pos=0;\
  for(i=0;i<size;i++)\
    if(val < I[i])\
      {\
        val = I[i];\
        pos = i;\
      }}

#define M_adaptation(A,c,eta,I)\
 for(i=0;i<REN(A);i++)\
      XY(A,i,c) = I[i]*eta + (1 - eta)*XY(A,i,c)

/*Variables*/
#define N 200
#endif
```

art2.c

```
/* Nombre: matriz.c */
/* Autor: Miguel Angel Palomera Perez */
/* Descripcion: Clase Matriz */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "art2.h"

int print_error(char *s)
{
  printf("%s\n",s);
  return 0;
}

int M_init(Matriz *W,int r,int c)
{

  int i=0;
  if((*W = (Matriz)malloc(2*sizeof(int) + r * c *sizeof(double))) != NULL)
  {
    REN(*W) = r;
    COL(*W) = c;
    return 1;
  }
  return print_error("Memoria insuficente");
}

int M_append_col(Matriz *W,Vector I)
{
  int i;
  if((*W = (Matriz)realloc(*W,2*sizeof(int) + REN(*W) * (COL(*W) + 1)\
            *sizeof(double))) != NULL)
  {
    for(i=0;i<REN(*W);i++)
      XY(*W,i,COL(*W)) = I[i];
    COL(*W) +=  1;

    return 1;
  }
  return print_error("Memoria insuficiente");
}

int saveW(char *filename,Matriz W,char *name)
{
  FILE *fp;
  Fmatrix x;

  fp = fopen(filename,"wb");
  if(fp == NULL)
    return print_error("Imposible abrir archivo.");
  else
    {
      x.type = 0;
      x.mrows = REN(W);
      x.ncols = COL(W);
      x.imagf = 0;
      x.namelen = strlen(name) + 1;
```

```
      fwrite(&x,sizeof(Fmatrix),1,fp);
      fwrite(name,sizeof(char),x.namelen,fp);
      fwrite(&(XY(W,0,0)),sizeof(double),COL(W)*REN(W),fp);
      fclose(fp);
    }
  return 1;
}

Vector ART2_eval(Matriz *W,Vector I,double rho,double eta,double *col)
{
  Vector C,NG;
  int cmax,i,cmax_aux;
  double valmax;
  C = (Vector)malloc(COL(*W)*sizeof(double));
  M_multi_vec_matriz(I,*W,C);
  V_max_pos(C,COL(*W),valmax,cmax);

  if(valmax >= rho || valmax <= 0.0)
      M_adaptation(*W,cmax,eta,I);
  else
    {

      M_append_col(W,I);
      free(C);
      C = (Vector)malloc(COL(*W)*sizeof(double));
      M_multi_vec_matriz(I,*W,C);
      V_max_pos(C,COL(*W),valmax,cmax_aux);
      if(valmax >= rho || valmax <= 0.0)
        {
          M_adaptation(*W,cmax_aux,eta,I);
          cmax = cmax_aux;
        }
    }
  free(C);
  *col = (double)COL(*W);
  NG = (Vector) malloc(REN(*W)*sizeof(double));
  for(i=0;i<REN(*W);i++)
    NG[i] = XY(*W,i,cmax);
  return NG;
}

int time_elapsed(struct timeval *result, struct timeval *x, struct timeval *cy)
{
  struct timeval y;
  y.tv_sec = cy->tv_sec;
  y.tv_usec = cy->tv_usec;

  if (x->tv_usec < y.tv_usec) {
    int nsec = (y.tv_usec - x->tv_usec) / 1000000 + 1;
    y.tv_usec -= 1000000 * nsec;
    y.tv_sec += nsec;
  }
  if (x->tv_usec - y.tv_usec > 1000000) {
    int nsec = (x->tv_usec - y.tv_usec) / 1000000;
    y.tv_usec += 1000000 * nsec;
    y.tv_sec -= nsec;
  }
```

```
  result->tv_sec = x->tv_sec - y.tv_sec;
  result->tv_usec = x->tv_usec - y.tv_usec;

  return x->tv_sec < y.tv_sec;
}
```

B.2. Código fuente de la primera red

tarea1.c

```
/* Nombre: tarea1.c */
/* Autor: Miguel Angel Palomera Perez */
/* Descripcion: Primera red neuronal del caso de  */
/*              estudio */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include "art2.h"
#include "mpi.h"

int main(int argc,char *argv[])
{
  Matriz W1;
  Vector G1;
  double rho=0.0,eta=0.0;
  FILE *inout;
  Matriz art1 = NULL; /*Datos almacenar*/
  double RES_art1[3],tras,I[N],W[N],R[N];
  int i,j,k,l,m;
  char filename[40];
  char data[15];
  struct timeval tt,ti,tf;
  char tag=23;
  char prio=0;

  /*Iniciamos el sistema*/
  RTMPI_Init(&argc,&argv);
  gettimeofday(&ti,NULL);
  inout = fopen("art2.cfg","r+");
  fscanf(inout,"%s",data);
  eta = atof(data);
  fscanf(inout,"%s",data);
  rho = atof(data);
  fscanf(inout,"%s",data);
  tras = atof(data);
  fseek(inout, 0L, SEEK_SET);
  if(eta == 1.0)
    fprintf(inout,"%f %f %f",0.01,rho + 0.01,tras);
  else
    fprintf(inout,"%f %f %f",eta + 0.01, rho,tras);
  fclose(inout);
```

```
  srand(time(NULL));
  for(i=0;i<N;i++)
    W[i] = 1.0*rand()/(RAND_MAX + 1.0);
  srand(27);
  for(i=0;i<N;i++)
    R[i] = 0.1*rand()/(RAND_MAX+1.0);

  prio = 0;
  M_init(&W1,N,1);
  for(i=0;i<N;i++)
    XY(W1,i,0) = W[i];
  for(j=0;j<100;j++)
  {
    for(i=0;i<N;i++)
      I[i] = sin((j + 1.0)*((double)i/(double)N)*6.2832) + R[i];
    normalize(I,N);
    G1 = ART2_eval(&W1,I,rho,eta,&RES_art1[2]);
    /*Transmitimos el  G1 a la red 3*/
    RTMPI_Send(G1,N,MPI_DOUBLE,"t3",tag,255-prio);
    prio++;
  }

  RES_art1[0] = rho;
  RES_art1[1] = eta;
  M_init(&art1,3,1);
  for(i=0;i<3;i++)
    XY(art1,i,0) = RES_art1[i];
  free(W1);
  free(G1);


  sprintf(data,"art1_%d_%d",(int)(ceil(100.0*tras)),(int)(ceil(100.0*rho)));
  sprintf(filename,"art1_%.2f_%.2f.mat",tras,rho);
  saveW(filename,art1,data);
  free(art1);
  gettimeofday(&tf,NULL);
  time_elapsed(&tt,&tf,&ti);
  print("Art1 %.2f > %ld %ld\n",rho,tt.tv_sec,tt.tv_usec);
  RTMPI_Finalize();
  return 0;
}
```

B.3. Código fuente de la segunda red

tarea2.c

```
/* Nombre: tarea2.c */
/* Autor: Miguel Angel Palomera Perez */
/* Fecha: 04/03/2005 */
/* Descripcion: Segunda red neuronal del caso de  */
/*              estudio */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
```

```
#include <sys/time.h>
#include "art2.h"
#include "mpi.h"

int main(int argc,char *argv[])
{
  Matriz W2;
  Vector G2;
  double rho=0.0,eta=0.0;
  FILE *inout;
  Matriz art2 = NULL; /*Datos almacenar*/
  double RES_art2[3],tras,I[N],W[N],TI[N],R[N];
  int i,j,k,l,m;
  char filename[40];
  char data[15];
  struct timeval tt,ti,tf;
  char tag = 14;
  char prio = 0;

  /*Iniciamos el sistema*/
  RTMPI_Init(&argc,&argv);
  gettimeofday(&ti,NULL);
  inout = fopen("art2.cfg","r+");
  fscanf(inout,"%s",data);
  eta = atof(data);
  fscanf(inout,"%s",data);
  rho = atof(data);
  fscanf(inout,"%s",data);
  tras = atof(data);
  fseek(inout, 0L, SEEK_SET);
  if(eta == 1.0)
    fprintf(inout,"%f %f %f",0.01,rho + 0.01,tras);
  else
    fprintf(inout,"%f %f %f",eta + 0.01, rho,tras);
  fclose(inout);

  srand(time(NULL));
  for(i=0;i<N;i++)
    W[i] = 1.0*rand()/(RAND_MAX + 1.0);
  srand(27);
  for(i=0;i<N;i++)
    R[i] = 0.1*rand()/(RAND_MAX+1.0);

  prio = 0;
  M_init(&W2,N,1);
  for(i=0;i<N;i++)
    XY(W2,i,0) = W[i];
  for(j=0;j<100;j++)
  {
    for(i=0;i<N;i++)
      I[i] = sin((j + 1.0)*((double)i/(double)N)*6.2832) + R[i];
    normalize(I,N);
    /*Señal, traslapada*/
    for(i=0;i<N;i++)
      TI[i] = I[((int)(i + ceil((1.0 - tras)*N)))%N];

    G2 = ART2_eval(&W2,TI,rho,eta,&RES_art2[2]);
```

```
    /*Transmitimos G2 a la red 3*/
    RTMPI_Send(G2,N,MPI_DOUBLE,"t3",tag,255-prio);
    prio++;

  }
  RES_art2[0] = rho;
  RES_art2[1] = eta;
  M_init(&art2,3,1);
  for(i=0;i<3;i++)
    XY(art2,i,0) = RES_art2[i];

  free(W2);
  free(G2);

  sprintf(data,"art2_%d_%d",(int)(ceil(100.0*tras)),(int)(ceil(100.0*rho)));
  sprintf(filename,"art2_%.2f_%.2f.mat",tras,rho);
  saveW(filename,art2,data);
  free(art2);
  gettimeofday(&tf,NULL);
  time_elapsed(&tt,&tf,&ti);
  print("Art2 %.2f > %ld %ld\n",rho,tt.tv_sec,tt.tv_usec);
  RTMPI_Finalize();
  return 0;
}
```

B.4. Código fuente de la tercera red

tarea3.c

```
/* Nombre: tarea3.c */
/* Autor: Miguel Angel Palomera Perez */
/* Descripcion: Tercera red neuronal del caso de  */
/*              estudio */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include "art2.h"
#include "mpi.h"

int main(int argc,char *argv[])
{
  Matriz W3;
  double G[N];
  double rho=0.0,eta;
  FILE *inout;
  Matriz art3 = NULL; /*Datos almacenar*/
  double RES_art3[3],tras,W[N];
  int i,j,k,l,m;
  char filename[40];
  char data[15];
  struct timeval tt,ti,tf;
  MPI_Status status;
```

```
  /*Iniciamos el sistema*/
  RTMPI_Init(&argc,&argv);
  gettimeofday(&ti,NULL);
  inout = fopen("art2.cfg","r+");
  fscanf(inout,"%s",data);
  eta = atof(data);
  fscanf(inout,"%s",data);
  rho = atof(data);
  fscanf(inout,"%s",data);
  tras = atof(data);
  fseek(inout, 0L, SEEK_SET);
  if(eta == 1.0)
    fprintf(inout,"%f %f %f",0.01,rho + 0.01,tras);
  else
    fprintf(inout,"%f %f %f",eta + 0.01, rho,tras);
  fclose(inout);

  srand(time(NULL));
  for(i=0;i<N;i++)
    W[i] = 1.0*rand()/(RAND_MAX + 1.0);

  M_init(&W3,N,1);
  for(i=0;i<N;i++)
    XY(W3,i,0) = W[i];
  for(j=0;j<100;j++)
  {
    /*Se solicitan los datos de la red 1*/
    RTMPI_Recv(G,N,MPI_DOUBLE,"t1",23,&status);
    ART2_eval(&W3,G,rho,eta,&RES_art3[2]);

    /*Se solicitan los datos de la red 2*/
    RTMPI_Recv(G,N,MPI_DOUBLE,"t2",14,&status);
    ART2_eval(&W3,G,rho,eta,&RES_art3[2]);
  }

  RES_art3[0] = rho;
  RES_art3[1] = eta;
  M_init(&art3,3,1);
  for(i=0;i<3;i++)
    XY(art3,i,0) = RES_art3[i];
  free(W3);


  sprintf(data,"art3_%d_%d",(int)(ceil(100.0*tras)),(int)(ceil(100.0*rho)));
  sprintf(filename,"art3_%.2f_%.2f.mat",tras,rho);
  saveW(filename,art3,data);
  free(art3);
  gettimeofday(&tf,NULL);
  time_elapsed(&tt,&tf,&ti);
  print("Art3 %.2f > %ld %ld\n",rho,tt.tv_sec,tt.tv_usec);
  RTMPI_Finalize();
  return 0;
}
```

Referencias

Almeida Luis, Pedreiras Paulo *Scheduling within temporal partitions: Response-time analysis and server design.* En el compendio de la cuarta Conferencia Internacional ACM sobre Software Embebido, EMSOFT 2004.

Benítez Pérez Héctor, García Nocetti Francisco. *Fault Classification for a Class of Time Variable Systems by using a group of three ART2 Networks*

Calha Mário J., Fonseca José Alberto. *SimHol- A Graphical Simulator for the Joint Scheduling of Messages and Tasks in Distributed Embedded Systems.* En el compendio de la quinta Conferencia Internacional IFAC sobre Fieldbus y sus aplicaciones,Fet 2003.

Albert M.K. Cheng. *Real Time Systems: Scheduling , Analysis, and Verification.* Ed. Wiley-Interscience, 2002. pp 43

Comp.realtime. *Frequently Asked Questions (FAQs) (version 3.6)* Disponible en Internet en la dirección: http://www.faqs.org/faqs/realtime-computing/faq/

CORBA-FAQ. Object Management Group. Disponible en Internet en la dirección: http://www.omg.org/gettingstarted/corbafaq.htm

Jammes Demmel. *Lecture Notes for Intro to Parallel Computing.* Verano 1996. Disponible en Internet en la dirección: http://www.cs.berkeley.edu/~demmel/cs267/lecture01.html

S. Goddard K. Jeffay. *Distributed Real-Time Dataflow: An Execution Paradigm for Image Processing and Anti-Submarine Warfare Applications* En el compendio del Simposio sobre Sistemas de Tiempo Real IEEE, Washington DC, Diciembre 1996, pp 55-58.

S. Grossberg. Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. Biological Cybernetics, 23:121-134, 1976.

GRID1. Proyecto grid.org. Sitio en Internet http://www.grid.org/home.htm

GRID2. Proyecto GRIDS Center. Sitio en Internet http://www.grids-center.org/index.asp

Proyecto GNU. Sitio en Internet http://www.gnu.org/

Tales Heimfarth, Marcelo Götz, Franz J. Ramming, Fávio R. Wagner. *RTC: A Real-time Communication Middleware on Top of RTAI-Linux.* En el compendio del décimo sexto Simposio Internacional sobre Orientación a Objetos, Tiempo Real y Computación Distribuida (ISORC'03) IEEE.

Hugo Kohmann, *SCI Weds Reliability and Real-Time* Disponible en Internet en la dirección:http://www.rtcmagazine.com/

MPI. MPI consorcio. *The Message Passing Interface (MPI) standar* Disponible en Internet en la dirección: http://www-unix.mcs.anl.gov/mpi/index.htm

MPICH. Proyecto mpich. *MPICH: A Portable Implementation of MPI.* Disponible en Internet en la dirección: http://www-unix.mcs.anl.gov/mpi/mpich/

POSIX.4 *Portable Operating System Interface* Disponible en Internet en la dirección: http://www.pasc.org/plato/

PVM. Proyecto PVM. *Parallel Virtual Machine.* Disponible en Internet en la dirección:http://www.csm.ornl.gov/pvm/pvm_home.html

SETI. Proyecto SETI@home. *Search for Extraterrestrial Intelligence.* Sitio en Internet del proyecto: http://setiathome.ssl.berkeley.edu/

TAO. The ACE ORB. Sitio en Internet del proyecto: http://www.cs.wustl.edu/ schmidt/TAO.html

Tindell K. and Clark J. *Holistic schedulability for distributed hard real-time systems.* Microprocessing and Microprogramming,40:117–134, 1994.