UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA - CONTROL

DISEÑO DE UN COMPENSADOR DE RECURSOS PARA SISTEMAS DE CONTROL EN RED

TESIS

QUE PARA OPTAR POR EL GRADO DE
DOCTOR EN INGENIERÍA

P R E S E N T A:

JOSÉ ALBERTO APARICIO SANTOS

DIRECTOR DE TESIS:
DR. HÉCTOR BENÍTEZ PÉREZ
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y SISTEMAS

MIEMBROS DEL COMITÉ TUTOR:
DR. GERARDO RENÉ ESPINOSA PÉREZ
FACULTAD DE INGENIERÍA
DR. LUÍS AGUSTÍN ALVAREZ ICAZA LONGORIA
INSTITUTO DE INGENIERÍA
DR. PAUL ROLANDO MAYA ORTIZ
FACULTAD DE INGENIERÍA
DR. LEONID FRIDMAN
FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA, CD. MX., FEBRERO 2023

JURADO ASIGNADO:

Presidente : Dr. Gerardo René Espinosa Pérez
Secretario : Dr. Luis Agustín Álvarez Icaza Longoria
1er. Vocal : Dr. Héctor Benítez Pérez
2do. Vocal: Dr. Paúl Rolando Maya Ortiz
3er. Vocal : Dr. Leonid Fridman

IIMAS, Ciudad Universitaria, Cd. Mx.

TUTOR DE TESIS:
Dr. Héctor Benítez Pérez

FIRMA

Dedicado a mi familia:
mis padres, Simón Aparicio y Basilisa Santos,
y mis hermanos, Simón, Sandra, y Leonardo.

Agradecimientos

Agradezco a la Universidad Nacional Autónoma de México por todos estos años de formar parte de su gran comunidad, en especial al Posgrado en Ingeniería Eléctrica, así como al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas.

Agradezco Colegio Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado estos años de mi formación (CVU: 597175).

Agradezco a los miembros de la comunidad de la UNAM que me han apoyado: José Ángel, Beatriz, Oriol, Nora, Adrian en especial al Dr. Héctor Benítez Pérez por sus enseñanzas.

Agradezco a mis padres y hermanos por su eterno apoyo.

Agradezco a la Dra. Belem Hernández por ser mi inseparable compañera, por todo lo que hemos compartido y por la esperanza de un futuro juntos.

Por último quiero agradecer a los amigos y familiares que por desgracia ya no se encuentran entre nosotros, nunca los olvidaremos: Marisol, Felipe, Lorenzo, Gabriel, Celso, José Lino. Dios los tenga en su gloria.

José Alberto Aparicio Santos.

Resumen

Se presenta un administrador de recursos (*RM*) que ajusta en tiempo real los recursos computacionales requeridos por un conjunto dado de aplicaciones, con el fin de lograr una distribución justa de recursos. El diseño se basa en un nuevo modelo que describe el comportamiento de un Servidor de Ancho de Banda Constante (*CBS*), que asigna intervalos de tiempo a las tareas tipo *preemptive* cuando existe tiempo de inactividad en la ejecución de tareas periódicas de alta prioridad. Asumiendo que cada aplicación puede ser ejecutada en diferentes niveles de servicio, sin estar por debajo de un límite mínimo, este *RM* aumenta o disminuye una plataforma virtual para cada tarea. El *RM* reserva un presupuesto máximo para una tarea, que se rellena cuando es necesario sin afectar al resto de las tareas; cuando la aplicación cambia, también lo hace el presupuesto máximo, por lo que la plataforma virtual siempre está lista para servir a la tarea. Se exploran mecanismos para aplicar el *RM* en sistemas de procesadores heterogéneos. Para validar el modelo se simula en un sistema *NCS*.

Índice

Lista de figuras

Capítulo 1

Introducción

Un sistema en tiempo real (*real-time system – RTS*) está compuesto por un conjunto de aplicaciones en tiempo real (*real-time applications - RTA*), que comúnmente contienen tareas (*tasks*) que cooperan entre sí para alcanzar los objetivos de la aplicación. Estas tareas deben ser atendidas en momentos de tiempos concretos, afines con la naturaleza del sistema (Clark, 1990), por lo tanto, el correcto funcionamiento del *RTS* depende no sólo de una toma de decisiones adecuada, sino también del instante en que se comunican sus resultados (Stankovic, 1988).

Un administrador de recursos (*Resource Manager - RM*) gestiona los recursos computacionales utilizables con ayuda de las funciones de inspección de recursos, con el objetivo de que los procesos en ejecución tengan un rendimiento apropiado (Sommerville, 2011).

Un sistema de control en red (*Networked control system - NCS*), es un ejemplo de *RTA*, nacido de la tecnología embebida y de las redes, donde sensores, actuadores y elementos computacionales están acoplados por medio de una red. Las mismas tecnologías que han hecho posibles a los *NCSs* asignan restricciones en comunicación que hace la interconexión de componentes notable desde el enfoque de control. Algunos de los efectos que surgieron son (Hristu-Varsakelis et al., 2005):

- Demoras en la entrega de información entre componentes. Estas demoras pueden ser fijas o variantes en el tiempo (Y. and Chow, 2003).
- La posibilidad de que los datos no alcancen a su destino (pérdida de paquetes), esto puede ser a raíz no sólo del medio de comunicación sino también del protocolo de comunicación utilizado (Li H., 2008).

- Cuellos de botella (*bottlenecks*); puede suceder debido a que la red compartida sólo establece un número limitado de comunicaciones simultáneas entre componentes o debido a que tiene un rendimiento limitado. Los cuellos de botella ocurren debido a limitaciones computacionales (Li K., 2004).

Estas restricciones consiguen ser modeladas, aun así, se complican los problemas de control, como estabilización, estimación y seguimiento, entre otros. En la actualidad debido al aumento en la capacidad de las nuevas plataformas de *hardware*, la cifra de aplicaciones que comparten la misma plataforma de ejecución ha ido en aumento, dificultando la correcta distribución recursos (Di Natale and Sangiovanni-Vincentelli, 2010).

En este trabajo se esboza un administrador de recursos (*Resource Manager - RM*) difuso y distribuido, diseñado para sistemas de procesadores heterogéneos. Esto con el propósito de que el *RM* sea flexible a diferentes tasas de tareas y capacidades de procesadores. El *RM* converge a distribuciones de recursos justas entre las tareas.

1.1 Contexto

Los *NCS* se han aplicado en varios ámbitos, como la investigación espacial, la automatización industrial, los robots, las aeronaves, los automóviles, la supervisión de plantas de manufactura, el diagnóstico y resolución de problemas a distancia, y las teleoperaciones (Wang and Han, 2018).

Existen cinco temas de control para tratar a los *NCSs*: control de datos muestreados, control de cuantificación, control de redes, control activado por eventos y control de seguridad (Zhang et al., 2020). En este trabajo nos enfocaremos principalmente en el control de redes, como el mostrado en la Figura 1.1, este se trata de un esquema del tipo directo, ya que el controlador, junto con el administrador de recursos se encuentran comunicados por medio de la red con la planta (Mahmoud et al., 2013).

1.2 Objetivo

Diseñar y probar un sistema de compensación de cargas computacionales entre diferentes nodos que llevan a cabo *RTAs*. El esquema general adoptado se muestra

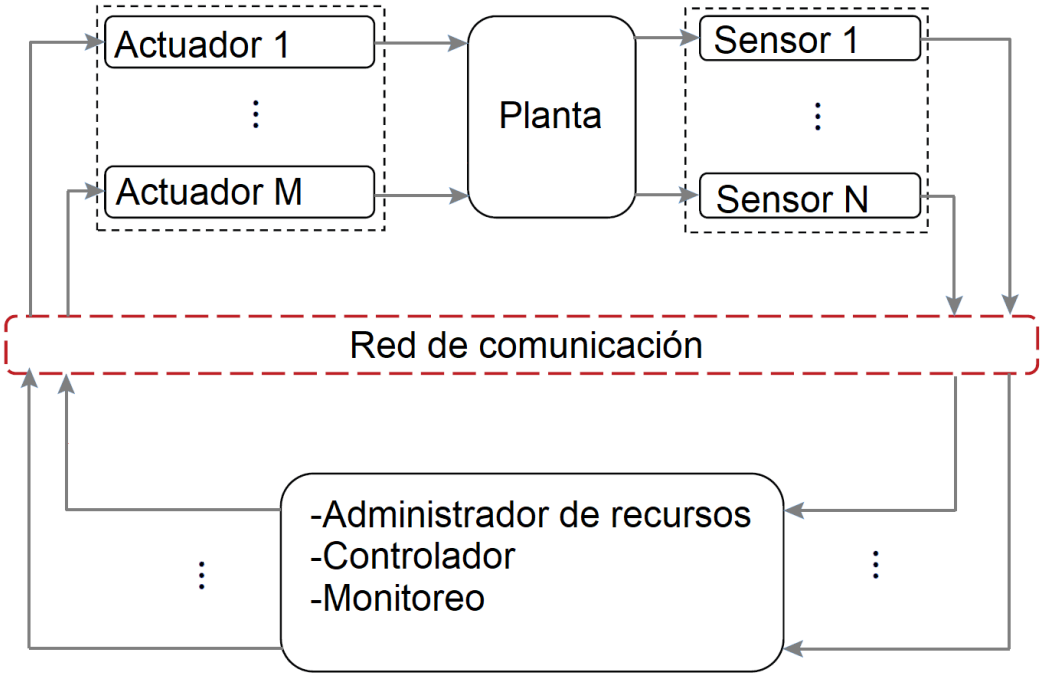

Figura 1.1: Esquema general de un *NCS*

en la Figura 1.1 y para la solución del problema se propone dividir este en cuatro procesos principales:

1. Ajuste de recursos, ejecutada por el *RM*.

2. Aplazamiento del *deadline* y asignación de presupuesto, llevada a cabo por el *CBS*.

3. Migración de tareas, ejecutada por un algoritmo de toma de decisión.

4. Adaptación del nivel de servicio, llevada de forma local por cada tarea.

La combinación de estos procesos debe llevar al sistema a una distribución de recursos justa, donde cada tarea tendrá los recursos necesarios para su óptima ejecución. Esto se demuestra, al aplicar el algoritmo a un sistema *NCS*.

1.3 Contribución

En el actual trabajo se plantea un esquema de *RM* aplicado a procesadores heterogéneos. Para alcanzar este objetivo el *RM* añade un servidor de ancho de banda constante al diseño presentado en Chasparis et al. (2016), este servidor que está definido por un conjunto de ecuaciones en diferencias y que permite integrar un esquema de control para garantizar la convergencia a distribuciones justas (*fair allocations*) y la estabilidad global para cualquier distribución inicial de recursos (por ejemplo: tiempo de procesamiento, ancho de banda de memoria, cache, etcétera). El uso del nuevo *RM* se ejemplifica a través del sistema de control de un dispositivo electromecánico.

El sistema debe cumplir las siguientes propiedades:

1. Lidiar con no linealidades propias de la distribución de recursos.
2. Ser escalable.
3. Decidir cuándo llevar a cabo la migración de tareas.
4. Converger a distribuciones justas para cualquier condición inicial dentro de los rangos de capacidad del sistema.

Para cumplir con estas propiedades, se hace uso del modelo difuso de Takagi-Sugeno (TS) Tanaka and Wang (2001), tanto para el *RM* como para el controlador del sistema bajo prueba, en este caso un sistema electromecánico, donde cada sensor y actuador es visto como una tarea. Los resultados obtenidos son generalizables.

1.4 Alcances

Las limitaciones del *RM* están en función del tipo de tareas que atiende, las tareas se toman como tipo *preemptive*, además, se considera que el consumo de cada tarea es conocido.

Debido a que hay diversas tareas vitales como las del sistema operativo, el sistema no puede asignar el 100% de la capacidad de los procesadores.

Las necesidades de recursos por parte de las tareas deben ser acotadas para poder garantizar una distribución de recursos justa, estas cotas se definen en los siguientes capítulos.

1.5 Estructura de la tesis

En el capítulo 2 se presenta una lista de variables a utilizar y los conceptos básicos para llevar a cabo la asignación de recursos, así como las herramientas básicas utilizadas.

En el capítulo 3 se aborda el estado del arte referente a la administración de recursos y se presenta el *RM* que sirve de base al *RM* desarrollado en esta tesis.

En el capítulo 4 se describe el modelo RM para sistemas homogéneos, de manera similar el capítulo 5 ahonda en el modelo para sistemas heterogéneos.

En el capítulo 6 se describe el diseño del controlador del sistema electromecánico y las ganancias que garantizan su estabilidad.

El capítulo 7 muestra los resultados obtenidos por medio de simulación y experimentación. Por último, el capítulo 8 presenta las conclusiones y se plantean recomendaciones para futuros trabajos.

Capítulo 2

Antecedentes

Un administrador de recursos (*RM*) se encarga de la distribución de distintos recursos computacionales entre distintas tareas (*tasks*) que deben ser atendidas en intervalos de tiempos específicos relacionadas a su naturaleza, lo que constituye un problema de planificación muy complejo. Ampliar el problema de planificación de recursos sobre una plataforma multiprocesador heterogénea lo dificulta todavía más (Baruah, 2006). En este trabajo se plantea un administrador de recursos para plataformas heterogéneas, difuso y distribuido. Lo primero, hace que el *RM* pueda lidiar con cambios no lineales en las necesidades de las tareas sin perder funcionalidad, gracias a las propiedades de los sistemas difusos (Tanaka and Wang, 2001). Lo segundo, permite dividir la solución del problema. Así, por un lado, cada aplicación hace una regulación local de sus recursos, mientras que, por el otro, el *RM* global coordina la distribución de tiempo de procesamiento asignado a cada tarea. Por lo anterior, es claro que el *RM* juega un papel crucial pues coadyuva a equilibrar el uso de los recursos entre las distintas tareas de un *RTS*. El *RM* que se presenta en este trabajo usa un Servidor de Banda Constante, *CBS*, cuyas dinámicas están descritas por ecuaciones en diferencias. Al combinar herramientas de la teoría control con algoritmos de planificación de recursos, se obtiene un esquema que hace posible analizar y probar la convergencia a una distribución justa (*fair allocation)* de los recursos entre las aplicaciones de un *RTS*, a partir una distribución inicial arbitraria.

En la siguiente sección se enlistan las variables, constantes y funciones utilizadas en esta tesis.

2.1 Abreviaturas

CBS: *Constant Bandwidth Server*

NCS: *Networked Control System*

RM: *Resource Manager*

RTA: *Real-Time Application*

RTS: *Real-Time System*

2.2 Símbolos y sus significados

h : número de procesador $h \in \mathbb{N}$.

i : número de tarea; $i \in \mathbb{N}$.

k : número de iteración; $k \in \mathbb{N}$.

n : número total de tareas; $i \in \mathbb{N}$.

si : identificador del servidor que atiende a la aplicación i; $si \in \mathbb{N} > 0$

ε : constante positiva, representa una pequeña cantidad de recursos que se asigna o se quita en cada iteración; $\varepsilon \in \mathbb{R} : 0 < i < 1$. Depende del número de aplicaciones $\varepsilon = 1/n$.

δ : número total de procesadores $\delta \in \mathbb{N}$.

α_i : constante positiva dependiente de la aplicación i.

β_i : constante positiva dependiente de la naturaleza de la aplicación i, $\beta_i \in \mathbb{R} : (0, \infty]$.

λ_i : prioridad estática de la aplicación i; $\lambda_i \in \mathbb{R} : [0, 1]$.

Λ_i : prioridad dinámica de la aplicación i; $\Lambda_i = \lambda_i [f_{i,k}]_{\_} \in \mathbb{R} : [-1, 0]$.

$C_{i,k}$: tiempo de ejecución nominal de la aplicación i, en una iteración k.

$c_{si,k}$: presupuesto actual del servidor que atiende a la aplicación i; $c_{si,k} \in \mathbb{R} : 0 < c_{si,k} < Q_{si}$.

$d_{si,k}$: *deadline* del servidor que atiende a la aplicación i; $d_{s,k} \in \mathbb{R} > 0$.

$f_{i,k}$: función de emparejamiento de la aplicación i en la iteración k, la cual indica si la aplicación está recibiendo suficientes o insuficientes recursos, el operador $[\,]\_$ limita el rango de $[f_{h,i,k}]$ a $[-1,0] \in \mathbb{R}$ y se define de la siguiente manera:

$$[x]\_ = \begin{cases} x & \text{si } x \leq 0 \\ 0 & \text{si } x > 0 \end{cases} \tag{2.1}$$

$F_{h,i,k}$: función de observación o equidad de la tarea i, ejecutada en el procesador h que mide el efecto de las otras tareas ejecutadas en el mismo procesador sobre ella en la iteración k; $F_{i,k} \in \mathbb{R} : [-1,1]$.

$\Phi_{h,k}$ tasa de cambio de la función de equidad nominal; $\Phi_{h,k} = |F_{h,i,m+1}| - |F_{h,i,m}|$.

P_h capacidad total del procesador h.

P_b procesador con más capacidad.

$\bar{P}_h$ procesador h normalizado con respecto al procesador más rápido P_b.

$p_{h,k}$ capacidad del procesador h en la iteración k.

Q_{si} : máximo presupuesto del servidor que atiende a la aplicación i; $Q_{si} \in \mathbb{R} > 0$.

$r_{i,k}$: tiempo de arribo de la aplicación i.

$s_{h,i,k}$: nivel de servicio de la aplicación i en la iteración k, ejecutada en el procesador h; es un estado interno de la aplicación i, es dependiente de la plataforma virtual asignada a la aplicación i y sólo puede ser leído/escrito por la misma aplicación, $s_i \in \mathbb{R} : \underline{s_i} < s_i < \overline{s_i}$.

T_{si} : periodo del servidor que atiende a la aplicación i; $T_{si} \in \mathbb{R} > 0$.

$v_{h,i,k}$: plataforma virtual ejecutada en el procesador h, asignada a la tarea i en la iteración k; $v_{h,i,k} \in \mathbb{R} > 0$.

U_T : capacidad total del sistema, la suma de las capacidades de todos los procesadores $\sum_{h=1}^{\delta} \bar{P}_h = U_T$.

$u_{T,k}$ capacidad total disponible en el sistema en la iteración k.

$\bar{v}_{h,i,k}$: tamaño de la plataforma virtual normalizada en el procesador h, asignada a la tarea i en la iteración k, $\bar{v}_{h,i,k} = v_{h,i,k}/P_h$; $\bar{v}_{h,i,k} \in \mathbb{R} : (0,1]$.

$\bar{w}_i$ consumo de la aplicación i normalizado.

$w_{h,j}$ consumo de la aplicación j asignado al procesador h.

$\bar{w}_{h,j}$ consumo normalizado de la aplicación j asignado al procesador h.

$\Pi_{\bar{V}_i}$ regresa a $\bar{v}_{i,k}$ al dominio de $\bar{V}_i$.

$\Pi_{\mathscr{S}_i}$ regresa a $\bar{s}_{i,k}$ al dominio de $\mathscr{S}_i$.

El operador binario $\rfloor x \lceil$ se define como

$$\rfloor x \lceil = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

2.3 Modelo de tarea

Un *NCS* puede verse como una serie de tareas en tiempo real. Una tarea está conformada por *jobs*, los cuales tienen un tiempo de inicio $r_{i,k}$ y deben cumplir un plazo de tiempo *deadline* ($d_{i,k}$), cada tarea tiene cierto tiempo de ejecución nominal $C_{i,k}$ (Hristu-Varsakelis et al., 2005).

* Algunas actividades deben terminar antes de un tiempo específico (*deadline*).

* Algunos datos deben ser generados antes de un *deadline*.

* Algunos procesos/hilos deben terminar antes de un *deadline*.

Las tareas se pueden clasificar dependiendo de la frecuencia con que son ejecutadas, existen tareas que no siguen un patrón periódico por lo que son llamadas tareas esporádicas. Por otro lado, las tareas que siguen un patrón periódico de ejecución son llamadas tareas periódicas.

- La tareas esporádicas y no periódicas deben tener un tiempo de respuesta tal que no interfieran con las tareas periódicas.

- Las tareas no periódicas se planifican durante el tiempo inactivo (*idle time*).

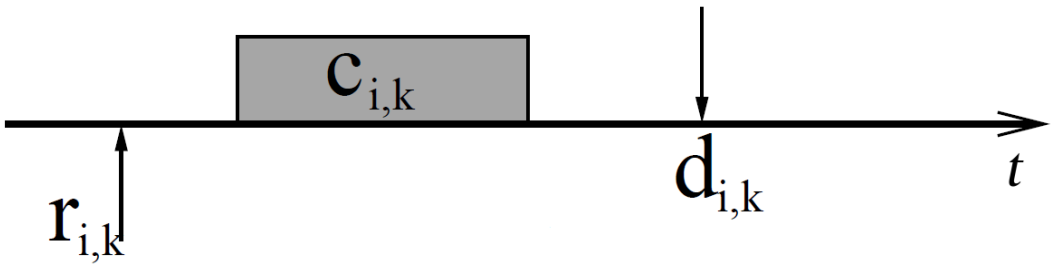

Figura 2.1: Modelo de *job*

- Una clase de soluciones modela una tarea periódica especial llamada servidor (*server*) para mejorar el tiempo de respuesta de la tarea aperiódicas. Caracterizado por un periodo T_i y la capacidad del servidor o presupuesto (*budget*) Q_i.

Otra forma de clasificar las tareas es si admiten ser suspendidas, las tareas que lo permiten se conocen como *preemtive*, las tareas que no permiten la suspensión, se clasifican como tareas *no preemtive*.

2.4 Plataformas multiprocesador

Hace un par de décadas, todos los dispositivos informáticos se basaban en un sistema uniprocesador, es decir de un solo procesador, lo que limitaba su capacidad de cómputo a un largo tiempo con un bajo rendimiento que se convirtió en un obstáculo considerable para el procesamiento, especialmente para las aplicaciones industriales y tecnológicas (Razian and MahvashMohammadi, 2017). Los sistemas multiprocesador aliviaron muchos de esos problemas al construir un número de pequeños procesadores conocidos como núcleos de procesamiento, o también llamados elementos de procesamiento, en un solo chip para permitir el procesamiento en paralelo de más de una tarea a la vez para reducir el tiempo de retardo y mejorar el rendimiento (Abdel-Basset et al., 2021). Existen diversas plataformas multiprocesador, las cuales pueden clasificarse en idénticas (u homogéneas), uniformes (o relacionada) y no relacionadas (o heterogéneas) (Carpenter et al., 2003), las cuales se describen a continuación:

- **Idénticas**: Se trata de plataformas multiprocesador en las que todos los procesadores son idénticos, en el sentido de que cada procesador de la

plataforma tiene las mismas capacidades de cómputo que los demás procesadores.

- **Uniformes**: Cada procesador de una plataforma multiprocesador uniforme (o relacionada) se caracteriza por su propia capacidad de cálculo, es decir, un *job*, que se ejecuta en cualquier procesador del sistema, lo hace a la misma tasa de ejecución.

- **No relacionadas**: En un multiprocesador no relacionado, se puede especificar una tasa de ejecución diferente para cada *job* en cada procesador.

Se puede observar que los multiprocesadores idénticos son un caso especial de los multiprocesadores uniformes, y los multiprocesadores uniformes son un caso especial de los multiprocesadores no relacionados (Baruah et al., 2015).

2.5 Plataforma virtual

Es una abstracción matemática utilizada para la distribución de recursos, nace del concepto de máquina privada virtual, donde se hacen reservaciones de los distintos recursos disponibles (procesador, memoria, ancho de banda, etcétera), obteniendo así un sistema aislado a partir de un sistema real, donde las tareas asignadas a una máquina virtual tienen a su disposición unicamente los recursos de la máquina asignada sin que las tareas ajenas a la máquina puedan acceder a estos recursos (Nesbit et al., 2008). En el caso de las plataformas virtuales cada reservación puede verse como una fracción de la capacidad del procesador real, esto implica que la suma de las capacidades de las plataformas virtuales no debe exceder la capacidad del procesador sobre el que son ejecutadas. Debido a los distintos consumos de las tareas, se suele realizar una normalización de los consumos para una distribución más sencilla (Bini et al., 2011). En esta tesis se realiza una distribución por procesador, es decir, en cada procesador puede haber varias plataformas virtuales.

2.6 Calidad de servicio (*QoS*)

La calidad de servicio (*QoS*) es una medida de satisfacción del usuario, como consecuencia, es un concepto dependiente de la aplicación, en el caso de un sistema de control este parámetro suele ser tomado como el tiempo de muestreo.

El rendimiento de un sistema de control en red puede mejorarse si la red puede garantizar la calidad del servicio (*QoS*). Debido a las demandas de tráfico de la red que varían en el tiempo y a las perturbaciones, los requisitos de calidad de servicio proporcionados pueden cambiar. En este caso, una red tiene que reasignar sus recursos y puede no ser capaz de proporcionar los requisitos de *QoS* a una aplicación de control en red según sea necesario. Por lo tanto, la aplicación puede tener que degradar su rendimiento y realizar la tarea lo mejor posible con la *QoS* de red proporcionada (Mok and Feng, 2001). Se hace uso de una abstracción para hacer frente al problema de comparar aplicaciones de distintas naturalezas.

2.7 Recapitulación

En este capítulo se hace una recopilación de los términos y definiciones utilizados para la realización de este trabajo. Debido a que la computación se fortaleció a mediados del siglo pasado, muchos de los términos han sido definidos apenas en las últimas dos décadas. Algunos términos han sido delimitados por cuestiones de simplificación, aún así, un gran número de casos es abarcado.

Entre los términos y definiciones que más ayudan al planteamiento del *RM* se encuentran las plataformas multiprocesador que contienen los recursos globales (capacidad de cada procesador), la plataforma virtual es la abstracción utilizada para su distribución, y la calidad de servicio ayuda a la distribución de los recursos locales (tiempo de muestreo).

En el siguiente capítulo se explora el estado del arte, donde se observan las debilidades y fortalezas de los esquemas de administradores de recursos (centralizado, distribuido e híbrido) propuestos a lo largo de los años.

Capítulo 3

Estado del arte

La planificación de tareas ha sido un problema ampliamente estudiado, el cuál se ha ido replanteando y aumentado su complejidad conforme los sistemas computacionales han evolucionado. A continuación se comparte una serie de trabajos relacionados con esté problema.

3.1 Abstracciones matemáticas

Uno de los primeros pasos en el diseño de un *RM* es definir una abstracción que represente los recursos a distribuir. Una manera de hacerlo es a través del concepto de calidad de servicio *(quality of service, QoS*) (Seitz, 2003), el cual depende de la naturaleza de la aplicación y de las necesidades del cliente. Así, el *RM* debe administrar la tasa de ejecución de las aplicaciones sin que estas pierdan *QoS*.

Para diseñar el *RM*, en Nesbit et al. (2008) se utiliza el esquema de máquina virtual privada, la cual está constituida por un conjunto de recursos de *hardware* virtuales, mientras que en Mok and Feng (2001) se utiliza un procesador virtual que divide a las tareas en grupos que comparten recursos, pero que no interfieren entre ellos; este procesador virtual está descrito por una proporción de uso y un retardo. Derivado de este modelo, Bini et al. (2011) y Chasparis et al. (2016) usan el concepto de plataforma virtual, la cual es una abstracción de la proporción de uso del procesador. En las técnicas de distribución de recursos, cada reservación de estos es vista como un procesador virtual (o plataforma) ejecutada en una fracción de la disponibilidad del procesador físico. La plataforma es manejada por un proceso llamado servidor (*server*), dedicado al manejo de los recursos compartidos.

Un solo servidor puede manejar diversos subconjuntos de tareas, y si este fuera el caso, las tareas a cargo del servidor compartirán recursos, pero deben estar protegidas contra sobrecostes (*overrun*) (Buttazzo, 2011). Si una tarea es manejada por un solo servidor, entonces esta tarea está aislada temporalmente y se debe garantizar que los recursos que recibe no interfieran con el desempeño de las demás tareas. En este trabajo se asume que una aplicación puede ser ejecutada a diferentes niveles de servicio (*service levels*), donde mayor nivel de servicio implica mayor *QoS*, (Chasparis et al., 2016).

3.2 Esquemas de Administrador de recursos

3.2.1 Esquema centralizado

Un esquema clásico para diseñar *RM* es el esquema centralizado (Bini et al., 2011), donde el *RM* se encarga de asignar los procesadores virtuales a las aplicaciones, a su vez que observa el uso de recursos y asigna el nivel de servicio de cada aplicación (Chasparis et al., 2013). Estos esquemas tienen la ventaja de ser muy estables, sin embargo, tienen varias debilidades. La primera de ellas es que la complejidad de los algoritmos usados para implementar el *RM* crece geométricamente con el número de aplicaciones, lo cual puede implicar que el manejo del *RM* llegue a consumir más recursos que los que debe distribuir. La segunda es que resulta difícil encontrar una función de costos para distribuir los recursos, debido a que el *RM* debe comparar la calidad de servicio de diferentes aplicaciones y, como se mencionó, el concepto de calidad de servicio depende de cada una de estas y no necesariamente es el mismo entre ellas. Por último, para una apropiada asignación de los niveles de servicio el *RM* debe tener conocimiento de los estados internos de todas las aplicaciones; para ello, las aplicaciones deben informar al *RM* sobre el nivel de servicio disponible y los recursos que se consumirán por cada nivel de servicio, lo que incrementa significativamente el costo de la comunicación (Chasparis et al., 2016).

3.2.2 Esquema distribuido

Debido a dichas debilidades, los esquemas de *RM* distribuidos han llamado la atención de manera importante (Subrata et al., 2008). El esquema de la Figura 3.1

muestra un esquema distribuido que asigna la proporción de uso del procesador y el nivel de servicio para cada aplicación independientemente. En este esquema se deben proveer condiciones para obtener convergencia a distribuciones equitativas (*fair allocations*). El esquema presenta las siguientes características:

- La complejidad crece linealmente con el número de aplicaciones.
- El *RM* no tiene que conocer los estados internos de cada aplicación.
- Es robusto frente al número y naturaleza de las aplicaciones (IEEE, 1990).

Se asume que cada RTA se compone de tareas que pueden ser síncronas o asíncronas y que a su vez estas están integradas por porciones de código sensibles al tiempo llamadas *jobs*, donde los *jobs* que integran una tarea pueden repetirse (Chasparis et al., 2013). En este trabajo se considera que la RTA sólo ejecuta una tarea integrada por *jobs* que requieren atención en diferentes intervalos de tiempo (ver Figura 3.1).

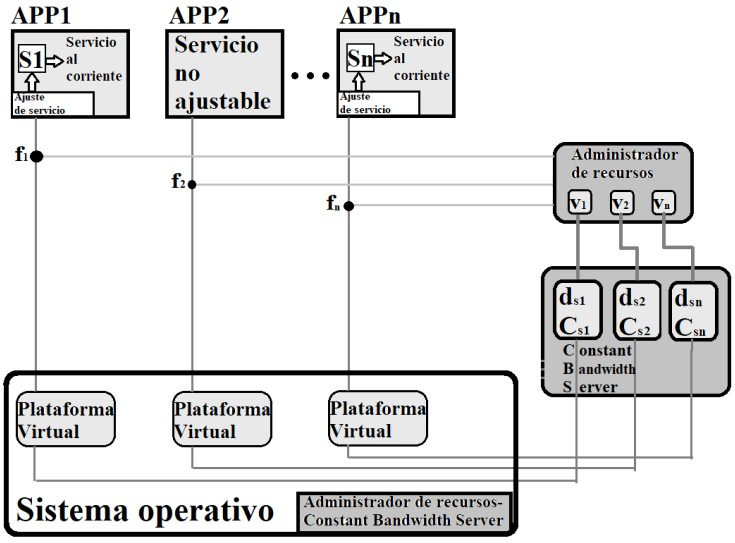

Figura 3.1: Esquema de optimización distribuido Aparicio-Santos et al. (2021), *RM* es el administrador de recursos, *APPn* es la aplicación n, f_n es el nivel de emparejamiento entre la proporción de uso v_n y el nivel de servicio s_n de la aplicación n.

El *RM* presentado en Chasparis et al. (2016) cuenta con las siguientes propiedades: evita inanición (*starvation avoidance*), obtiene balanceo (*balance*), converge síncronamente y asíncronamente (*asynchronous and synchronous convergence*) (Chasparis et al., 2013).

Aparicio-Santos (2017) retoma el modelo Chasparis et al. (2016) agregando una descripción del *CBS* e implementando una ley difusa de control para aplicarlo a sistemas de control en red.

3.3 Plataformas heterogéneas

En los últimos años los sistemas multiprocesador heterogéneos han surgido como un enfoque eficaz para el manejo eficiente de energía. Esta configuración introduce mayor complejidad al problema de distribución de recursos al incrementar el tamaño del espacio de solución (Premi et al., 2020).

La heterogeneidad puede abarcar varios aspectos, como la rapidez de acceso a memoria, la capacidad, rapidez de transmisión de los buses, etcétera (Hermosillo, 2020). En este trabajo sólo se consideran plataformas heterogéneas a la que está compuesta por procesadores de distintas tasas de rapidez. Baruah (2002) propone condiciones de suficiencia para determinar si un sistema multiprocesador heterogéneo es planificable. Chatterjee (1997) describe un algoritmo integrado de asignación y enrutamiento basado en QoS que determina qué recursos de computación y comunicación utilizar para aplicaciones de comunicación. De manera similar, Huh et al. (2000) utiliza la QoS para unificar los recursos dinámicos entre los procesadores heterogéneos, controlar el entorno de los recursos, analizar la factibilidad de la distribución y llevar a cabo un balanceo dinámico de los recursos. Premi et al. (2020) propone un enfoque de teoría de juegos para asignar recursos heterogéneos a las aplicaciones, centrándose en los requisitos de rendimiento, potencia y energía, por medio de un modelo de congestión de juego.

3.4 Diferencias entre los esquemas homogéneos y heterogéneos

Los esquemas de administración de recursos para sistemas homogéneos y heterogéneos comparten el mismo fin, el cual es distribuir los recursos de manera

equitativa entre las distintas aplicaciones del sistema.

3.4.1 Comportamiento de los procesadores

La diferencia más evidente, está en las restricciones, el sistema heterogéneo tiene mayores restricciones comparado con el esquema homogéneo, se puede definir por el tipo de sistema, es decir si se tienen procesadores idénticos en el sistema. Aunque se debe tener cuidado en como se define un sistema homogéneo, por ejemplo, un sistema de procesadores iguales podría tener un comportamiento heterogéneo, si los procesadores están ejecutando diferentes sistemas operativos.

3.4.2 Migración de tareas

Otra diferencia entre los esquemas homogéneos y heterogéneos es la descripción de la migración de tareas, en el caso de los homogéneos no se toma en cuenta, lo cuál facilita su descripción.

En el caso del esquema heterogéneo, la migración se divide en diferentes casos, tomando como base la función de equidad nominal.

3.5 Recapitulación

En este capítulo se exploró brevemente un compendio de líneas de investigación enfocados a la administración de recursos con sus ventajas y desventajas. Con base en la literatura estudiada, este trabajo usará un esquema distribuido, basado inicialmente en el modelo descrito por (Chasparis et al., 2016) y expandido por (Aparicio-Santos et al., 2021) donde se muestra la descripción del *CBS*. Con la descripción del *CBS* se pueden diseñar controladores para una mejor distribución de recursos.

Debido a su versatilidad de poder dividir el problema de planificación en distribución de recursos globales y locales, se puede lograr una distribución justa utilizando alguna técnica de control en este trabajo se propone utilizar control difuso debido a su propiedad en el manejo de no linealidades, se profundiza sobre esto en el Capítulo 5.

Capítulo 4

Sistemas homogéneos

Los sistemas multiprocesador se pueden dividir en dos categorías, homogéneos y heterogéneos, aunque los sistemas de homogéneos se pueden ver como un caso particular de los heterogéneos, la distribución de recursos en los primeros es más sencilla. A continuación se presentan un algoritmo de distribución de recursos para sistemas homogéneos, en el siguiente capitulo este algoritmo se expande a sistemas heterogéneos.

El *RM* se encarga de distribuir los recursos, tratando de cumplir con la calidad de servicio requerida. Para elegir un esquema de *RM* se debe tomar en cuenta la información disponible. Los esquemas generales de *RM* son tres, centralizados, distribuidos e híbridos (Byeong Gi et al., 2009).

El *RM* de este trabajo está basado en Aparicio-Santos et al. (2021), donde los recursos a distribuir son el tiempo de procesamiento (recurso global) y el tiempo de muestreo (recurso local). Para administrar el tiempo de procesamiento se hace uso del concepto de plataforma virtual v_i, la cual es una representación que caracteriza a un procesador virtual por su proporción de uso, misma que representa la fracción usada de la potencia disponible en dicho procesador. La proporción de uso se compone de dos parámetros: el presupuesto Q, y el periodo T, de tal manera que dos procesadores virtuales con la misma proporción de uso pueden asignar tiempo de manera diferente. Por ejemplo, dos procesadores virtuales pueden tener una proporción de uso igual a 0.25, pero el primer procesador virtual tener un periodo de $4s$ y un presupuesto de $1s$, mientras que un segundo procesador virtual tener un periodo de $4ms$ y un presupuesto de $1ms$. En este caso, a pesar de tener la misma proporción de uso, el segundo procesador virtual es más adaptable en

el sentido de que una aplicación puede ser atendida con mayor frecuencia y, por ello, progresar más uniformemente (Bini et al., 2011).

Se asume que el nivel de servicio s_i es un estado interno de la aplicación app_i y que, por lo tanto, sólo puede ser leído/escrito por la misma aplicación. Sin embargo, es dependiente de la plataforma virtual v_i.

La desigualdad $v_i \geq v_i^{'}$ quiere decir que la calidad entregada por cualquier aplicación app_i ejecutada sobre v_i no puede estar por debajo de la calidad entregada cuando se ejecuta sobre $v_i^{'}$, mientras que, en plataformas virtuales de igual tamaño, la desigualdad $s_i \geq s_i^{'}$ significa que la calidad de servicio entregada al usuario por la aplicación app_i cuando se ejecuta al nivel s_i no es menor que la calidad de servicio cuando se ejecuta en el nivel $s_i^{'}$.

4.1 Modelo síncrono

En este trabajo se considera un modelo síncrono, es decir, se considera que cada *RTA* sincroniza su nivel de servicio en la misma iteración en la que el *RM* asigna tiempo de procesamiento a la aplicación. En (Chasparis et al., 2016) y (Chasparis et al., 2013) se propone el modelo distribuido descrito por las ecuaciones (4.1)-(4.4).

$$\bar{v}_{i,k+1} = \Pi_{\bar{v}i}\left[\bar{v}_{i,k} + \varepsilon F_{i,k}\right] \tag{4.1}$$

$$s_{i,k+1} = \Pi_{\mathscr{S}_i}[s_{i,k} + \varepsilon f_{i,k}] \tag{4.2}$$

$$F_{i,k} \doteq -(U_s - \bar{v}_{i,k}))\lambda_i[f_{i,k}]_- + \bar{v}_{i,k}\sum_{l\neq i}\lambda_l[f_{l,k}]_- \tag{4.3}$$

$$f_{i,k} = \beta_i\frac{\bar{v}_{i,k}}{s_{i,k}} - 1 \tag{4.4}$$

donde

k : número de iteración; $k \in \mathbb{N}$.

i : número de aplicación; $i \in \mathbb{N}$..

$\bar{v}_{i,k}$: tamaño de la plataforma virtual normalizada asignada a la aplicación i en la iteración k, $\bar{v}_{i,k} = v_{i,k}/\delta$; $\bar{v}_{i,k} \in \mathbb{R} : (0,1]$

$F_{i,k}$: función de observación desde la aplicación i sobre el efecto de las otras aplicaciones sobre ella en la iteración k; $F_{i,k} \in \mathbb{R} : [-1,1]$.

$\bar{v}_s$: plataforma por distribuir entre las aplicaciones *soft*, tomando en cuenta que las aplicaciones están normalizadas $\bar{v}_s = 1 - \bar{v}_H$, donde $\bar{v}_H$ es la reservación para las aplicaciones *hard*.

ε : constante positiva, representa una pequeña cantidad de recursos que se asigna o se quita en cada iteración; $\varepsilon \in \mathbb{R} : 0 < i < 1$.

$s_{i,k}$: nivel de servicio de la aplicación i en la iteración k; $s_i \in \mathbb{R} : \underline{s_i} < s_i < \overline{s_i}$.

$f_{i,k}$: función de emparejamiento, de la aplicación i en la iteración k, la cual indica si la aplicación está recibiendo suficientes o insuficientes recursos, el operador $[\,]\_$ limita su rango a $[f_{i,k}]\_ \in \mathbb{R} : [-1,0]$, el operador se define de la siguiente manera:

$$[x]\_ = \begin{cases} x & \text{si } x \leq 0 \\ 0 & \text{si } x > 0 \end{cases} \tag{4.5}$$

β_i : constante positiva dependiente de la aplicación i, está conformada por el tiempo de ejecución nominal C_i, y la constante positiva α_i, tal que, $\beta_i = \frac{C_i}{\alpha_i}$.

λ_i : prioridad de la aplicación i; $\lambda_i \in \mathbb{R} : [0,1]$.

$\Pi_{\bar{V}_i}$ regresa a $\bar{v}_{i,k}$ al dominio de $\bar{V}_i$.

$\Pi_{\mathscr{S}_i}$ regresa a $\bar{s}_{i,k}$ al dominio de $\mathscr{S}_i$.

La Ecuación (4.1) nos muestra la actualización de las plataformas virtuales, esta actualización es dependiente de la función de equidad nominal (Ecuación (4.3)), que a su vez es dependiente de los recursos asignados a las otras tareas (determinado por la Ecuación (4.4), la Ecuación (4.3) se modifica más adelante para determinar si las distribuciones heterogéneas son justas. Por último la Ecuación (4.2) se encarga de la actualización del nivel de servicio, dependiente de los recursos asignados a la aplicación observada i por medio del nivel de emparejamiento (Ecuación 4.4).

Este modelo no toma en cuenta la dinámica del servidor de ancho de banda constante (*CBS*), dinámica que es de gran utilidad para la aplicación de diversas

técnicas de control, este modelo tampoco está diseñado para un sistema multiprocesador heterogéneo, para adaptarlo a este tipo de sistemas multiprocesadores, son necesarias ciertas modificaciones, sobre todo en la Ecuación (4.3) que describe si la distribución es justa. Para implementarla, en este trabajo se aplica un control difuso que se describe en la siguiente sección.

4.2 Servidor de ancho de banda constante (*CBS*)

El *CBS* (por sus siglas en ingles *Constant Bandwidth Server*) (Abeni and Buttazzo, 1998) es un mecanismo de servicio, con el cual se implementa una estrategia de reservación. El *CBS* es caracterizado por los parámetros Q_s (presupuesto) y T_s (periodo), que garantiza que si $U_s = Q_s/T_s$ es la fracción del tiempo de procesador asignado a un servidor, su contribución a la utilización total no es mayor que U_s. El *CBS* está regido por las siguientes reglas.

1. Cuando un nuevo *job* entra al sistema, se le asigna una planificación de deadline adecuada para mantener la demanda dentro del ancho de banda reservado, y se inserta en la cola "*ready*" del *Earliest Deadline First* (*EDF*) (Horn, 1974).

2. Si el *job* trata de de ejecutar más de lo esperado, su *deadline* se pospone (por ejemplo, su prioridad decrece) para reducir la interferencia con las demás tareas. Al posponer, la tarea sigue elegible para su ejecución.

3. Si un subconjunto de tareas es manejada por un solo servidor, todas las tareas en ese subconjunto compartirán el mismo ancho de banda, no hay aislamiento entre ellas. Todas las otras tareas en el sistema están protegidas contra sobrecostes que ocurran en el subsistema.

Las ecuaciones propuestas (4.6-4.7) que describen la actualización del *deadline* y el presupuesto por cada *CBS* son:

$$c_{si,k+1} = c_{si,k} \left\rfloor (d_{si,k} - r_{ij})\delta\bar{v}_{i,k}c_{si,k} \right\lceil + Q_{si} \left\rfloor c_{si,k} - (d_{si,k} - r_{ij})\delta\bar{v}_{i,k} \right\lceil + Q_{si} \left\rfloor - c_{si,k} \right\lceil \tag{4.6}$$

$$d_{si,k+1} = d_{si,k} + T_{si} \left\rfloor (d_{si,k} - r_{ij})\delta\bar{v}_{i,k}c_{si,k} \right\lceil + T_{si} \left\rfloor - c_{si,k} \right\lceil \tag{4.7}$$

donde

si : identificador del servidor que atiende a la aplicación i; $si \in \mathbb{N} > 0$

$d_{si,k}$: *deadline* del servidor que atiende a la aplicación i; $d_{s,k} \in \mathbb{R} > 0$.

$c_{si,k}$: presupuesto actual del servidor que atiende a la aplicación i; $c_{si,k} \in \mathbb{R} : 0 < c_{si,k} < Q_{si}$.

T_{si} : periodo del servidor que atiende a la aplicación i; $T_{si} \in \mathbb{R} > 0$.

Q_{si} : máximo presupuesto del servidor que atiende a la aplicación i; $Q_{si} \in \mathbb{R} > 0$.

$v_{i,k}$: proporción de uso asignada al servidor que atiende a la aplicación i y se define como $v_{i,k} = Q_{si}/T_{si}$; $v_{i,k} \in \mathbb{R} > 0$.

$r_{i,k}$: tiempo de arribo de la aplicación i.

El operador binario $\lfloor x \rceil$ se define como

$$\lfloor x \rceil = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

4.3 Integración al modelo Chasparis

Si se asume que el total de tiempo de procesamiento a distribuir entre las aplicaciones tipo *soft* es $\bar{v}_s$, este tiempo de procesamiento se debe distribuir entre las distintas aplicaciones, en fracciones Q_{si}, en un mismo periodo T_{si} tal que:

$$V_T = \frac{\sum_i^n Q_{si}}{T_{si}} \leq 1 - \bar{v}_H \tag{4.8}$$

donde v_H es la reservación para aplicaciones tipo *hard*. A cada plataforma virtual le corresponde una fracción del tiempo de procesamiento v_s. En caso de que el periodo sea constante la relación entre el presupuesto Q_{si} y la plataforma virtual $v_{i,k}$ será lineal.

Integrando el modelo de Chasparis et al. (2016) con las dinámicas del *CBS*, se obtienen cuatro ecuaciones en diferencias, las cuales fueron presentadas en Aparicio-Santos (2017), con ayuda de estas ecuaciones se facilita el diseño de una ley de control para las distintas partes del *RM*:

$$\bar{v}_{i,k+1} = \bar{v}_{i,k} + \varepsilon F_{i,k} \tag{4.9}$$

$$s_{i,k+1} = \bar{s}_{i,k} + \varepsilon f_{i,k} \tag{4.10}$$

$$c_{si,k+1} = c_{si,k} \rfloor (d_{si,k} - r_{ij}) \delta \bar{v}_{i,k} c_{si,k} \lceil + Q_{si} \rfloor c_{si,k} - (d_{si,k} - r_{ij}) \delta \bar{v}_{i,k} \lceil + Q_{si} \rfloor - c_{si,k} \lceil \tag{4.11}$$

$$d_{si,k+1} = d_{si,k} + T_{si} \rfloor (d_{si,k} - r_{ij}) \delta \bar{v}_{i,k} c_{si,k} \lceil + T_{si} \rfloor - c_{si,k} \lceil \tag{4.12}$$

Donde la ecuación 4.9 representa la actualización de la plataforma virtual, la ecuación 4.10 describe la actualización del nivel de servicio, la ecuación 4.11 muestra la actualización del presupuesto en función de la plataforma virtual asignada a la aplicación, así como al procesador sobre la cual está siendo ejecutada. Por último la ecuación 4.12 calcula el *deadline* y su aplazamiento si es necesario.

4.4 Recapitulación

En este capítulo se presentó el modelo Chasparis expandido presentado en Aparicio-Santos et al. (2021) y sus actualizaciones para sistemas multiprocesador homogéneos, el sistema logra distribuir los recursos de forma justa. Para que pueda ser aplicado a sistemas multiprocesador heterogéneos este algoritmo debe ser modificado, esto se aborda en el siguiente capitulo.

Capítulo 5

Sistemas heterogéneos

Un problema de planificación de tareas sobre una plataforma multiprocesador homogénea es complejo, si además se considera una plataforma heterogénea, el problema se dificulta aún más (Hermosillo, 2020). Se propone una esquema utilizando funciones y conceptos definidos previamente en este trabajo.

Tanto el modelo presentado en Chasparis et al. (2016) como el expandido presentado en Aparicio-Santos et al. (2021) fueron diseñados para procesadores homogéneos, con la misma velocidad de procesamiento. De los administradores de recursos presentados en la Sección 3, se tienen que δ es el número de procesadores, ahora se debe tomar en cuenta que los procesadores tienen diferentes velocidades, se define P_h la cual es la capacidad del procesador h. Como los procesadores tienen distintas velocidades, un procesador debe ser más rápido que los otros, por lo que existe un $P_b > P_h$, se pueden normalizar δ procesadores respecto al procesador más rápido.

$$\bar{P}_h = \frac{P_h}{P_b} \leq 1 \tag{5.1}$$

La suma de todas las velocidades normalizadas da como resultado la máxima capacidad del sistema.

$$\sum_{h=1}^{\delta} \bar{P}_h = U_T \tag{5.2}$$

El sistema completo no puede superar está utilidad, si se rebasa el sistema no es factible, debido a que se estaría pidiendo más capacidad de la disponible. Para

llevar a cabo la distribución entre tareas se utiliza la actualización de plataforma virtual de la Ecuación (5.3), modificada para que la plataforma máxima que se pueda asignar a la tarea esté definida por la capacidad del procesador normalizada, la aplicaciones involucradas sólo son las que están siendo ejecutadas en el mismo procesador que la aplicación observada.

$$\bar{v}_{h,i,k+1} = \bar{v}_{h,i,k} + \varepsilon F_{h,i,k} \tag{5.3}$$

$$F_{h,i,k} \doteq -(\bar{P}_h - \bar{v}_{h,i,k})\lambda_i [f_{i,k}]_- + \bar{v}_{h,i,k} \sum_{l \neq i} \lambda_j [f_{j,k}]_- \tag{5.4}$$

donde las plataformas virtuales asignadas al procesador deben cumplir con una factibilidad local dependiente de la velocidad del procesador, ε es una variable positiva dependiente del número de aplicaciones, $\varepsilon = 1/n$, esto con el fin de garantizar convergencia (Chasparis et al., 2016). Las plataformas asignadas al procesador deben cumplir la siguiente condición de factibilidad:

$$\sum \bar{v}_{h,i,k} \leq \bar{P}_h \tag{5.5}$$

tomando en cuenta que ahora las plataformas ahora no están normalizadas respecto al número de procesadores si no respecto a la capacidad normalizada del procesador asignando $\bar{v}_{h,i,k} = \frac{v_{h,i,k}}{\bar{P}_b}$.

5.1 Modelo del sistema

Se propone utilizar el nivel de equidad nominal $F_{i,k}$ (Ecuación 5.10) como prioridad dinámica que nos indica qué tareas deben realizar la migración, para que esto sea posible se hace un análisis de dicha función, cuyo comportamiento típico se ilustra en la Figura 5.1. Analizando la figura, se puede observar que una función de equidad negativa se traduce en una distribución de recursos que satisface la tarea observada, y mientras más negativo sea su valor, el procesador tendrá más holgura para atender la tarea.

Integrado la modificación de la función de equidad, el modelo completo queda de la siguiente forma:

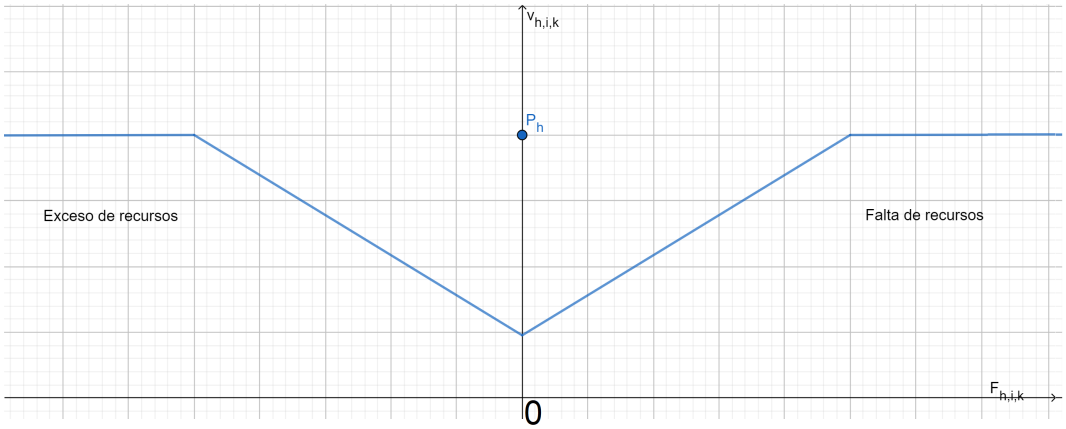

Figura 5.1: Comportamiento de la ecuación de equidad nominal $F_{h,i,k}$ (Ecuación 5.10) contra la plataforma virtual $v_{h,i,k}$ (Ecuación 5.6), si $v_{h,i,k}$ es positiva implica una falta de recursos, por el contrario si es negativa; la aplicación debe ceder recursos, al estar en un procesador de capacidad P_h, los recursos asignados no pueden exceder la capacidad de dicho procesador. Cuando $F_{h,i,k}$ es cero implica que las distribución es justa, la pendiente de cambio depende de la variable ε dependiente a su vez del número de aplicaciones.

$$\bar{v}_{h,i,k+1} = \bar{v}_{h,i,k} + \varepsilon F_{h,i,k} \tag{5.6}$$

$$s_{i,k+1} = \bar{s}_{i,k} + \varepsilon f_{i,k} \tag{5.7}$$

$$\begin{aligned} c_{si,k+1} = {} & c_{si,k} \rfloor (d_{si,k} - r_{i,k}) P_h \bar{v}_{h,i,k} c_{si,k} \lceil \\ & + Q_{si} \rfloor c_{si,k} - (d_{si,k} - r_{i,k}) P_h \bar{v}_{h,i,k} \lceil + Q_{si} \rfloor - c_{si,k} \lceil \end{aligned} \tag{5.8}$$

$$d_{si,k+1} = d_{si,k} + T_{si} \rfloor (d_{si,k} - r_{i,k}) P_h \bar{v}_{h,i,k} c_{si,k} \lceil + T_{si} \rfloor - c_{si,k} \lceil \tag{5.9}$$

donde

$$F_{h,i,k} \doteq -(\bar{P}_h - \bar{v}_{h,i,k}) \lambda_i [f_{i,k}]_- + \bar{v}_{h,i,k} \sum_{j \neq i} \lambda_j [f_{j,k}]_- \tag{5.10}$$

$$f_{i,k} = \beta_i \frac{\bar{v}_{h,i,k}}{s_{h,i,k}} - 1 \tag{5.11}$$

$$P_h = \bar{P}_h P_b \tag{5.12}$$

$$\Lambda_{i,k} = \lambda_i F_{i,k} \tag{5.13}$$

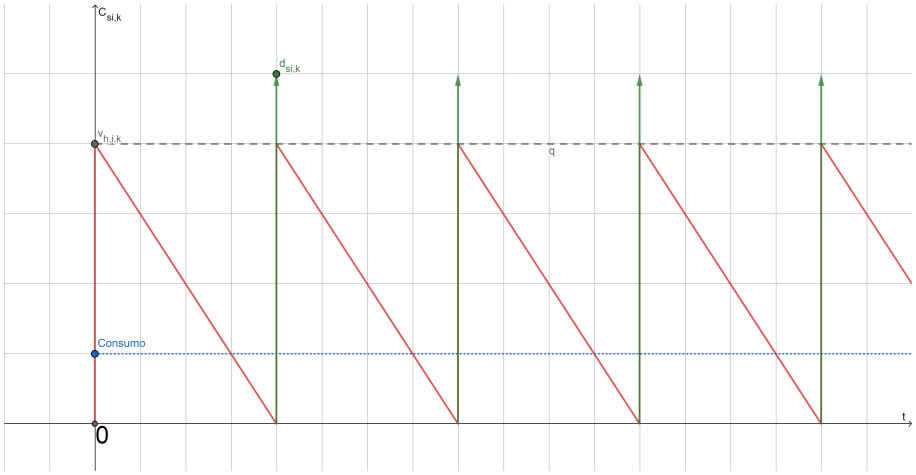

Figura 5.2: Comportamiento de la actualización del presupuesto, en la figura se toma el consumo como constante, el *CBS* se encarga de rellenar el presupuesto y posponer el *deadline*.

Cada procesador tiene su capacidad P_h, se empieza a utilizar conforme agregamos tareas al procesador por medio de las plataformas virtuales $v_{h,i,k}$, siendo la plataforma virtual i ejecutada en el procesador h en la iteración k. Las tareas ejecutadas en el procesador no deben superar la capacidad del mismo.

$$\sum_{i=1}^{n_h} v_{h,i,k} \leq P_h \tag{5.14}$$

La capacidad restante $p_{h,k}$ del procesador es:

$$p_{h,k} = P_h - \sum_{i=1}^{n_h,k} v_{h,i,k} \tag{5.15}$$

La capacidad total restante del sistema es:

$$u_{T,k} = \sum_{h=1}^{\delta} p_{h,k} \tag{5.16}$$

Se propone un vector a_h por cada procesador para manejar la asignación de tareas, de tal forma que: $a_1 = [v_1, v_2 ... v_j]$, $a_h = [a_{j+1}, a_{j+2} ... a_n]$.

Para llevar a cabo la asignación de tareas se contemplan cinco escenarios:

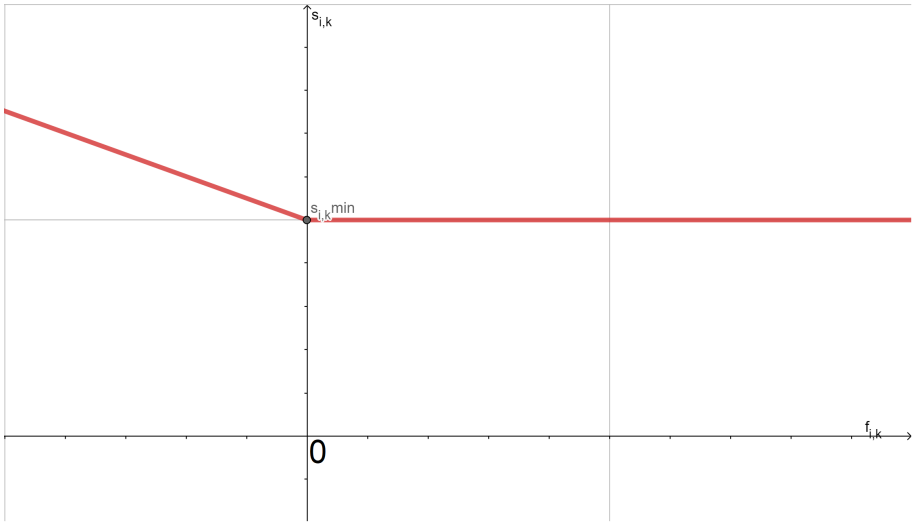

Figura 5.3: Comportamiento de la actualización del nivel de servicio contra la función de emparejamiento $f_{i,k}$.

- Una nueva tarea llega, hay espacio para atenderla.
- La tarea no puede ser atendida, no hay capacidad suficiente.
- La tarea es tipo *preemptive*, es posible atenderla.
- La tarea es tipo *preemptive*, no es posible atenderla.
- Se conoce el consumo de la tarea.

5.2 Tareas *no preemptive*

Si hay capacidad para atender una tarea *no preemptive*, utilizando la función de equidad nominal $F_{h,i,k}$ se evalúa la tarea nueva en cada procesador. Si la función de equidad nominal tiene una pendiente Φ_k negativa, indica que el procesador está otorgando recursos a la tarea. Mientras más pronunciada sea la pendiente, la convergencia será más rápida (Figura 5.1).

$$F_{h,i,k} \doteq -(\bar{P}_h - \bar{v}_{h,i,k}))\lambda_i[f_{i,k}]_- + \bar{v}_{h,i,k}\sum_{j\neq i}\lambda_j[f_{j,k}]_- \tag{5.17}$$

$$\Phi_{h,i,k+1} = |F_{h,i,k}| - |F_{h,i,k-1}| \tag{5.18}$$

Donde, en caso de tener dos procesadores h y b, la comparación de $\Phi_{h,k} < \Phi_{b,k}$ implica que el procesador a atenderá de mejor manera la tarea i. Ademas, si $F_{h,i,k} < 0$ indica que a la tarea i se le han asignado recursos suficientes.

La actualización de la plataforma virtual se lleva a cabo con el último valor calculado de la función de equidad nominal $F_{h,i,k} = F_{h,i,m+1}$.

$$\bar{v}_{h,i,k+1} = \bar{v}_{h,i,k} + \varepsilon F_{h,i,k} \tag{5.19}$$

En caso de no haber capacidad para atender una nueva tarea, la tarea puede esperar a ser atendida en cierto número de iteraciones dependiendo de su prioridad λ_i o descartada.

5.3 Tareas *preemptive*

En caso de que haya capacidad para atender la tarea, esta se maneja igual que en el caso no preemptive.

En caso de no poder atender la tarea completa en un sólo procesador, la tarea puede ser dividida para poder ser asignada a la capacidad actual de los procesadores $u_{i,c} = p_{h,k}$, donde $u_{i,c}$ es un segmento c de la tarea i, adaptado a la capacidad actual del procesador a. Esto es posible si el consumo de la tarea u_i no es mayor a la capacidad total $U_T = \sum_{h=1}^{\delta} p_h$.

En caso de no haber capacidad para ser atendida, la tarea puede esperar a que se libere capacidad en cierto número de iteraciones dependiendo de su prioridad λ_i o ser descartada.

5.4 Consumo local de la tarea

El total de la capacidad $p_{h,k}$ debe ser comparado con el consumo $w_{h,j}$, con una capacidad total del P_h. Las capacidades deben estar normalizadas respecto al procesador asignado:

$$\bar{w}_{h,j} = \frac{w_{h,j}}{P_h} \tag{5.20}$$

La actualización de la capacidad es:

$$p_{h,k} = P_h - \sum_{j \neq i}^{n_h} \bar{w}_{h,j} \tag{5.21}$$

$$p_{h,k+1} = p_{h,k} - \bar{w}_i \tag{5.22}$$

Donde n_h es el numero de tareas en el procesador h, $w_{h,j}$ es el consumo de las tareas que se encuentran en el procesador h en la iteración k.

5.5 Proyección de la capacidad restante

Migrar una tarea de procesador causa variaciones en las capacidades, por lo tanto no sólo es necesario conocer la capacidad utilizada por el procesador, sino también hacer una proyección de la capacidad que se liberará. Si el tamaño de la plataforma virtual es:

$$\bar{v}_{h,i,k+1} = \bar{v}_{h,i,k} + \varepsilon F_{h,i,k} \tag{5.23}$$

donde

$$\bar{v}_{h,i,k} = \frac{v_{h,i,k}}{P_h} \tag{5.24}$$

$$F_{h,i,k} \doteq -(\bar{P}_{h,i,k} - \bar{v}_{h,i,k}))\lambda_i [f_{i,k}]_- + \bar{v}_{h,i,k} \sum_{j \neq i} \lambda_j [f_{j,k}]_- \tag{5.25}$$

$$f_{i,k} = \beta_i \frac{\bar{v}_{h,i,k}}{s_{h,i,k}} - 1 \tag{5.26}$$

La proyección de la capacidad restante es:

$$\bar{p}_{rh,k+1} = \bar{P}_h - \sum_{i=1}^{n_h} \bar{v}_{h,i,k+1} \tag{5.27}$$

donde n_h es el número de aplicaciones ejecutadas en el procesador h. Es importante observar que si hay capacidad en un procesador después de hacer la asignación de tareas, significa que el sistema es panificable y la distribución es justa.

5.6 Condiciones iniciales

Se necesita una distribución inicial de las aplicaciones en los procesadores, en este caso se utiliza la constante $\beta_i = \alpha_i / C_i$, la cual está conformada por una constante α_i y el tiempo de ejecución nominal C_i de la aplicación i, por lo tanto se puede tener una estimación del consumo promedio de la aplicación, si $\beta_i < \beta_j$ implica que la aplicación i tiene mayor consumo que la aplicación j, basado en esto se puede hacer una distribución inicial, donde las aplicaciones con β_i menores requieren ser ejecutadas en un procesador de mayor capacidad.

5.7 Recapitulación

En este capítulo se replantearon las distintas ecuaciones que definen al *RM* para procesadores homogéneos, para modificarlas al contexto de los procesadores heterogéneo.

En el caso del los procesadores heterogéneos se sugiere verlo como distintos casos, dichos casos son dependientes de la naturaleza de las aplicaciones y la información disponible sobre ellas. El *RM* utiliza la ecuación de equidad nominal $F_{i,k}$, para decidir que hacer con la tarea en cuestión si debe ser migrada, ejecutada o incluso descartada.

Capítulo 6

Representación a través de sistemas difusos

Un esquema *RM* como el mostrado en Aparicio-Santos et al. (2021) está descrito por medio de ecuaciones en diferencia no lineales, donde su complejidad obstaculiza la aplicación de algunas técnicas de control clásico. El control difuso es una elección confiable para controlar sistemas complejos, pues son considerados aproximadores universales, es decir, que pueden aproximar una función lineal suave a cualquier exactitud prescrita en una región convexa si se cuenta con suficientes reglas difusas (Boutalis et al., 2014). Se toma la representación difusa de los sistemas heterogéneos debido a que es un caso generalizado de los sistema homogéneos.

6.1 Modelo Takagi-Sugeno

Las técnicas de control difuso representan un medio de recolección de conocimiento humano y habilidad (Fang et al., 2006). A pesar de que el método ha sido prácticamente exitoso, se ha probado que es extremadamente difícil desarrollar un análisis general y diseñar una teoría para sistemas de control difusos convencionales. Los sistemas difusos basados en el modelo de Takagi-Sugeno (Tanaka and Wang, 2001) han aparecido recientemente en la literatura, mostrando un gran número de resultados sobre análisis de estabilidad y diseño.

El modelo difuso Takagi-Sugeno, está descrito por una serie de reglas **IF** − **THEN** con la siguiente estructura Tanaka et al. (1998):

Regla modelo i:

$$IF\ z_1\ es\ M_{i1}\ and \dots\ and\ z_p(t)\ es\ M_{ip}$$
$$THEN = \begin{cases} \nabla x(t) & = A_i x(t) + B_i u(t) \\ y(t) & = C_i x(t) \end{cases} \tag{6.1}$$

M_{ij} Funciones de membresía, entregan un valor de membresía en el intervalo de $[0,1]$ a cierta variable premisa z_i asociada.

r número de reglas premisa.

$\nabla x(t)$ representa un operador:

- Para sistemas difusos continuos $\nabla x(t) = \dot{x}(t)$.
- Para sistemas difusos discretos $\nabla x(t) = x(t+1)$.

$u(t)$ vector de entrada.

$y(t)$ vector de salida.

- $z_1(t), \dots, z_p(t)$ Variables premisas, la variables premisas son dependientes de los estados y contienen las no linealidades.

El estado se puede reconstruir con las siguientes ecuaciones.

$$\nabla x(t) = \sum_{i=1}^{r} h_i(z(t))\{A_i x(t) + B_i u(t)\}, \tag{6.2}$$

$$y(t) = \sum_{i=1}^{r} h_i(z(t)) C_i x(t), \tag{6.3}$$

donde

$$z(t) = [z_1(t)\ z_2(t) \dots z_p(t)] \tag{6.4}$$

$$w_i(z(t)) = \prod_{j=1}^{p} M_{ij}(z_j(t)) \tag{6.5}$$

$$h_i(z(t)) = \frac{w_i(z(t))}{\sum_{j=1}^{r} w_i(z(t))} \tag{6.6}$$

w_i Es la cuantificación de cada antecedente i.

h_i Es el peso de cada antecedente comparado con la suma de todos los antecedentes.

A_i Matriz de estados del subsistema consecuencia i.

B_i Vector de entrada del subsistema consecuencia i.

C_i Vector de salida del subsistema consecuencia i.

Regla control i:

$$\begin{aligned} IF\ \ & z_1\ es\ M_{i1}\ and \ldots\ and\ z_p(t)\ es\ M_{ip} \\ THEN\ \ & u(t) = -F_i x(t),\ i = 1,2,\ldots r \end{aligned} \tag{6.7}$$

donde la regla de control nominal $u(t)$ es:

$$u(t) = -\frac{\sum_{i=1}^{r} w_i(z(t))F_i x(t)}{\sum_{i=1}^{r} w_i(z(t))} = -\sum_{i=1}^{r} h_i(z(t))F_i x(t) \tag{6.8}$$

Sustituyendo (6.8) en (6.2), el sistema difuso en lazo cerrado puede ser representado como

$$\nabla x(t) = \sum_{i=1}^{r}\sum_{j=1}^{r} h_i(z(t))h_j(z(t))(A_i - B_i F_j)x(t) \tag{6.9}$$

6.2 Estabilidad de un sistema difuso

Existen diferentes maneras de estabilizar un sistema difuso por ejemplo, pasividad y modos deslizantes, a continuación se muestra la estabilización por asignación de polos.

Definición 6.2.1 ***(Estabilidad cuadrática de un sistema difuso)****[Tanaka and Wang (2001)] El sistema difuso (6.2) se dice cuadráticamente estabilizable si existe un controlador como en (6.8) tal que el sistema en lazo cerrado (6.9) sea cuadráticamente estabilizable.*

Se define a partir de (6.9)

$$G_{ij} = A_i - B_i F_j \tag{6.10}$$

Entonces sustituyendo (6.10) en (6.9)

$$\nabla x(t) = \sum_{i=1}^{r} \sum_{j=1}^{r} h_i(z(t)) h_j(z(t)) G_{ij} x(t) \tag{6.11}$$

Que puede ser reescrita como

$$\nabla x(t) = \sum_{i=1}^{r} h_i(z(t)) h_i(z(t)) G_{ii} x(t) + 2 \sum_{i<j}^{r} h_i(z(t)) h_j(z(t)) \left\{ \frac{G_{ij} + G_{ji}}{2} \right\} x(t) \tag{6.12}$$

Las condiciones de estabilidad para un sistema difuso continuo y uno discreto no son iguales, por lo que se presentan por separado.

Condiciones de estabilidad de un sistema difuso continuo

Un sistema difuso continuo en lazo cerrado es estable si los términos de (6.12) cumplen con las desigualdades (6.13) y (6.14) [Seidi et al. (2012)].

$$G_{ii}{}^T P + P G_{ii} < 0 \tag{6.13}$$

$$\left(\frac{G_{ij} + G_{ji}}{2} \right)^T P + P \left(\frac{G_{ij} + G_{ji}}{2} \right) \leq 0 \tag{6.14}$$

En el caso de que $B_1 = B_2 = \ldots B_r$ sólo se debe cumplir la desigualdad (6.13) para garantizar estabilidad asintótica.

Condiciones de estabilidad de un sistema difuso discreto

Un sistema difuso discreto en lazo cerrado es estable si los términos de (6.12) cumplen con las desigualdades (6.15) y (6.16).

$$G_{ii}^T P G_{ii} - P < 0 \tag{6.15}$$

$$\left(\frac{G_{ij} + G_{ji}}{2} \right)^T P \left(\frac{G_{ij} + G_{ji}}{2} \right) - P \leq 0 \tag{6.16}$$

En el caso de que $B_1 = B_2 = \ldots B_r$ sólo se debe cumplir la desigualdad (6.15) para garantizar estabilidad asintótica.

El problema de diseño consiste en encontrar la matriz P y las ganancias F_i, tales que cumplan las condiciones (6.13)-(6.14) para el caso continuo o (6.15)-(6.16) para el caso discreto, para encontrar dichos valores se recurre al método por LMI.

6.3 Modelación difusa

Como ya se explicó, se ha optado por usar sistemas difusos para modelar las no linealidades del *RM* y de las leyes de control involucradas con la aplicación seleccionada.

Aquí se plantea utilizar un modelo difuso tipo Takagi-Sugeno (Tanaka et al., 1998) el cual está descrito por una serie de reglas **IF** $-$ **THEN**. Este *RM* difuso será capaz de aproximar las no linealidades del modelo Chasparis extendido mediante un conjunto de sistemas lineales, donde para cada elemento también es posible diseñar un controlador específico. Como en todo sistema difuso, el estado actual del modelo tiene asignado un nivel de membresía a cada elemento del conjunto de sistemas no lineales. Así la combinación modelo difuso-controlador puede, a partir de cualquier distribución inicial arbitraria de recursos, llevarla esta hacia una distribución justa, sin que se pierda la estabilidad en lazo cerrado del sistema aproximado.

Además de la aproximación difusa, se propone usar un *RM* distribuido, lo que se consigue al distribuir el modelo de Chasparis extendido en tres subsistemas: i) ajuste de servicio; ii) distribución de plataforma virtual; iii) actualización del *deadline* y presupuesto (ver Figura 6.1). Estos subsistemas se pueden tratar como tres sistemas independientes para diseñar los esquemas de control que garanticen su estabilidad lazo cerrado. En todos los casos, las funciones de membresía utilizadas serán de forma triangular.

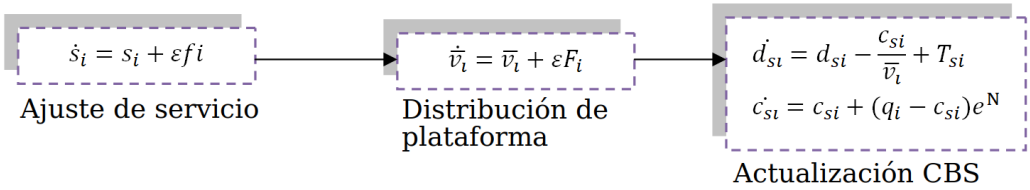

Figura 6.1: El sistema se divide en tres subsistemas, los cuales están enlazados por un mínimo de información.

6.4 Modelación difusa del nivel de plataforma virtual

Para la distribución de la plataforma virtual se necesita conocer el valor de la función de emparejamiento f_i de cada aplicación, para así poder calcular la función equidad nominal F_i; si se desarrolla dicha función se obtiene:

$$\dot{\bar{v}}_i = \bar{v}_i + \varepsilon F_i = \bar{v}_i + \varepsilon \left\{ -(\bar{P}_h - \bar{v}_{h,i,k})\lambda_i [f_{i,k}]_- + \bar{v}_{h,i,k} \sum_{l \neq i} \lambda_j [f_{j,k}]_- \right\} \tag{6.17}$$

Se define la variable premisa

$$z_{v_i} = \varepsilon \left(-(\bar{P}_h - \bar{v}_{h,i,k})\lambda_i [f_{i,k}]_- + \bar{v}_{h,i,k} \sum_{l \neq i} \lambda_j [f_{j,k}]_- \right) \tag{6.18}$$

y las funciones de membresía

$$M_{vi_1} = 1 - \frac{z_{v_i} - z_{vi_{max}}}{z_{vi_{min}} - z_{vi_{max}}} \tag{6.19}$$

$$M_{vi_2} = \frac{z_{v_i} - z_{vi_{max}}}{z_{vi_{min}} - z_{vi_{max}}} \tag{6.20}$$

Las cuales están relacionadas con las declaraciones:

- M_{vi_1} Recursos insuficientes para aplicación i
- M_{vi_2} recursos suficientes para aplicación i

y con los valores máximo vi_{max} y mínimo vi_{min} de la plataforma virtual $\bar{v}_i$.

6.5 Modelación difusa del nivel de servicio

El ajuste del nivel de servicio también se lleva a cabo dentro de cada aplicación pues, como se mencionó, este depende de la naturaleza de la misma. Para ello, se deben definir niveles de servicio máximo si_{max} y mínimo si_{min}, de manera que el nivel de servicio esté siempre entre ellos. El nivel de servicio se calcular de la siguiente ecuación

$$\dot{s}_i = s_i + \varepsilon f_i = s_i + \left[\varepsilon \left(\frac{\beta_i \kappa \bar{v}_i}{s_i} - 1 \right) \right]_{-} \tag{6.21}$$

donde se puede notar que para su cálculo la aplicación i solamente requiere información local. Se define como variable premisa para cada aplicación:

$$z_{s_i} = \varepsilon \left(\frac{\beta_i \kappa}{s_i} - 1 \right) \tag{6.22}$$

De esta variable premisa, se definen dos funciones de membresía que representan los extremos de las situaciones en las que se puede encontrar el nivel de servicio y que están descritas por las siguientes ecuaciones

$$M_{si_1} = 1 - \frac{z_{s_i} - z_{si_{max}}}{z_{si_{min}} - z_{si_{max}}} \tag{6.23}$$

$$M_{si_2} = \frac{z_{s_i} - z_{si_{max}}}{z_{si_{min}} - z_{si_{max}}} \tag{6.24}$$

Cada una de estás funciones de membresía está relacionadas con las declaraciones:

- M_{si_1} Máximo servicio en la aplicación i
- M_{si_2} Mínimo servicio en la aplicación i.

La relación entre las reglas difusas y los sistemas líneales asociados a cada una se ilustra en la Tabla 6.1, donde los sistemas $A_{vj}x_v + B_v u_v$ tienen la estructura de la ecuación (6.25), con i el número de la aplicación y n el número total de estas. Como el cálculo de la plataforma virtual $\bar{v}_i$ y del nivel de servicio se realiza de manera distribuida, el número de reglas y sistemas lineales asociados es $4n$. En el ejemplo de la Tabla 6.1, $n = 3$, por lo tanto, si se combinan las reglas del nivel de plataforma virtual y las del nivel de servicio, se obtienen 12 reglas asociadas a 12 sistemas lineales.

Tabla 6.1: Relaciones entre las funciones de membresía $M_{vi_1}(z_{v_i})$, $M_{vi_2}(z_{v_i})$, $M_{si_1}(z_{s_i})$, $M_{si_2}(z_{s_i})$; $i = 1,2,3$, y w_{vj}, $j = 1,2,3...12$ es la cuantificación de cada combinación. La forma de los sistemas lineales $A_{vj}x_v + B_v u_v$ es presentada en la ecuación (6.25)

$w_{vj}(z)$	IF	THEN $A_{vj}x_v + B_v u_v$
$w_{v1}(z)$	$M_{v1_1}(z_{v_1})$	$A_{v1}x_v + B_v u_v$
$w_{v2}(z)$	$M_{v1_2}(z_{v_1})$	$A_{v2}x_v + B_v u_v$
$w_{v3}(z)$	$M_{v2_1}(z_{v_2})$	$A_{v3}x_v + B_v u_v$
$\vdots$	$\vdots$	$\vdots$
$w_{v11}(z)$	$M_{s3_1}(z_{s_3})$	$A_{v11}x_v + B_v u_v$
$w_{v12}(z)$	$M_{s3_2}(z_{s_3})$	$A_{v12}x_v + B_v u_v$

$$\begin{bmatrix} \dot{\bar{v}}_1 \\ \vdots \\ \dot{\bar{v}}_n \\ \dot{s}_1 \\ \vdots \\ \dot{s}_n \\ \dot{\lambda}_1 \\ \vdots \\ \dot{\lambda}_n \end{bmatrix} = \begin{bmatrix} z_{v_1} & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & z_{v_n} & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & z_{s_1} & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & z_{s_n} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{v}_1 \\ \vdots \\ \bar{v}_n \\ s_i \\ \vdots \\ s_n \\ \lambda_i \\ \vdots \\ \lambda_n \end{bmatrix}$$
$$+ \begin{bmatrix} 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ & \vdots & \\ 1 & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{v1} \\ \vdots \\ u_{vn} \end{bmatrix} \tag{6.25}$$

Para garantizar la estabilidad de lazo cerrado se deben proponer las F_i y encontrar

una matriz $P > 0$ que cumpla la siguiente desigualdad para todas las matrices A_i.

$$G_{ii} = A_i - B_i F_i \tag{6.26}$$

$$0 > G_{ii}^T P G_{ii} - P \tag{6.27}$$

6.6 Modelación difusa del nivel de *CBS*

El *CBS* actualiza dos estados, el *deadline* d_{si} y el presupuesto c_{si}, de acuerdo con las siguientes ecuaciones.

$$\dot{c}_{si} = Q_{si} \rfloor c_{si} - (d_{si} - r_i)\kappa \bar{v}_i \lceil + Q_{si} \rfloor - c_{si} \lceil \tag{6.28}$$

$$\dot{d}_{si} = T_{si} \rfloor (d_{si} - r_i)\kappa \bar{v}_i c_{si} \lceil + T_{si} \rfloor - c_{si} \lceil \tag{6.29}$$

Al ser dos ecuaciones se definen dos variables premisa por aplicación, haciendo que las combinaciones sean 2^{n*2}, donde n es el número de aplicaciones.

$$z_{cs_i} = \kappa T_{si} \rfloor c_{si} - (d_{si} - r_i)\kappa \bar{v}_i \lceil \tag{6.30}$$

$$z_{ds_i} = \frac{1}{\kappa \bar{v}_i} \rfloor (d_{si} - r_i)\kappa \bar{v}_i c_{si} \lceil \tag{6.31}$$

Las funciones de membresía están definidas por las ecuaciones (6.32-6.35)

$$M_{csi_1} = 1 - \frac{z_{cs_i} - z_{csi_{max}}}{z_{csi_{min}} - z_{csi_{max}}} \tag{6.32}$$

$$M_{csi_2} = \frac{z_{cs_i} - z_{csi_{max}}}{z_{csi_{min}} - z_{csi_{max}}} \tag{6.33}$$

$$M_{dsi_1} = 1 - \frac{z_{ds_i} - z_{dsi_{max}}}{z_{dsi_{min}} - z_{dsi_{max}}} \tag{6.34}$$

$$M_{dsi_2} = \frac{z_{ds_i} - z_{dsi_{max}}}{z_{dsi_{min}} - z_{dsi_{max}}} \tag{6.35}$$

donde csi_{max}, csi_{min}, dsi_{max} y dsi_{min} son los límites máximo y mínimo de cs_i y ds_i, respectivamente. Cada una ligada a las siguientes declaraciones:

- M_{csi_1} presupuesto insuficientes para aplicación i.
- M_{csi_2} Rellenar presupuesto para aplicación i.
- M_{dsi_1} No se requiere actualizar el *deadline* de la aplicación i.

- M_{dsi_2} Actualizar el *deadline* de la aplicación i..

Los sistemas lineales $A_{ci}x_c + B_c u_c$ tienen la forma de la ecuación (6.36):

$$\begin{bmatrix} \dot{c}_{s_i} \\ \vdots \\ \dot{c}_{s_n} \\ \dot{d}_{s_i} \\ \vdots \\ \dot{d}_{s_n} \end{bmatrix} = \begin{bmatrix} 1 & \dots & 0 & z_{dc_i} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 1 & 0 & \cdots & z_{dc_n} \\ 0 & 0 & 0 & z_{d_i} & \cdots & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & z_{d_n} \end{bmatrix} \begin{bmatrix} c_{s_i} \\ \vdots \\ c_{s_n} \\ d_{s_i} \\ \vdots \\ d_{s_n} \end{bmatrix} + \begin{bmatrix} 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 0 & \cdots & 0 \\ 1 & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_i \\ \vdots \\ u_n \end{bmatrix} \tag{6.36}$$

De la misma manera que en los dos subsistemas anteriores, se requiere cumplir con la condición (6.15-6.16) para garantizar estabilidad en lazo cerrado.

6.7 Recapitulación

En este capítulo se obtuvo el la representación del modelo Takagi-Sugeno del RM para el caso heterogéneo, no es necesaria la representación del RM homogéneo debido a que es un caso particular del *RM* heterogéneo, la mayor modificación con el algoritmo presentado en (Aparicio-Santos et al., 2021) está en la función de equidad nominal, la cual hace a la plataforma virtual dependiente del procesador en el que es ejecutada, se debe tener cuidado con los valores máximos y mínimos de cada función de membrecía, ya que se puede llegar a tener dependencia lineal entre las columnas de las matrices, lo que hace al sistema indeterminado.

Capítulo 7

Simulación e implementación

Para probar el funcionamiento del *RM* y del *CBS* se recurre a la simulación de un sistema de control en red (*Networked control system - NCS*) de configuración directa (Mahmoud et al., 2013) aplicado al modelo un giroscopio de tres grados de libertad (ver Figura 7.1).

La meta del *NCS* es controlar dos de los tres grados de libertad del giroscopio, que corresponden con los ángulos ϕ y ψ en el marco de referencia mostrado en la Figura 7.1 y cuyo modelo dinámico está dado por (Robert H. Cannon, 2003)

$$M_b = J_y\ddot{\phi} - J_x^d\dot{\theta}\dot{\psi}\cos(\phi) + (J_z - J_x)\dot{\psi}^2\sin(\phi)\cos(\phi) \tag{7.1}$$

$$M_r = (J_z^r + J_z\cos^2(\phi) + J_x\sin^2(\phi))\ddot{\phi} + J_x^d\dot{\theta}\dot{\phi}\cos(\phi) + 2(J_x - J_z)\dot{\psi}\dot{\phi}\sin(\phi)\cos\phi \tag{7.2}$$

Los términos que componen las ecuaciones de movimiento (7.1-7.2) y el valor de sus parámetros dados por el fabricante (Quanser, 2012) son:

- $\dot{\theta}$: velocidad angular del disco alrededor de su propio eje de rotación x. $\dot{\theta} = 150 rad/s$.
- ϕ : posición angular de la suspensión cardán azul sobre y.
- ψ : posición angular de la suspensión cardán roja sobre Z.
- J_z^r : momento de inercia de la suspensión cardán roja sobre el eje Z. $J_z^r = 0.0342 kgm^2$
- J_z^d : momento de inercia del disco sobre el eje x. $J_z^d = 0.0056 kgm^2$

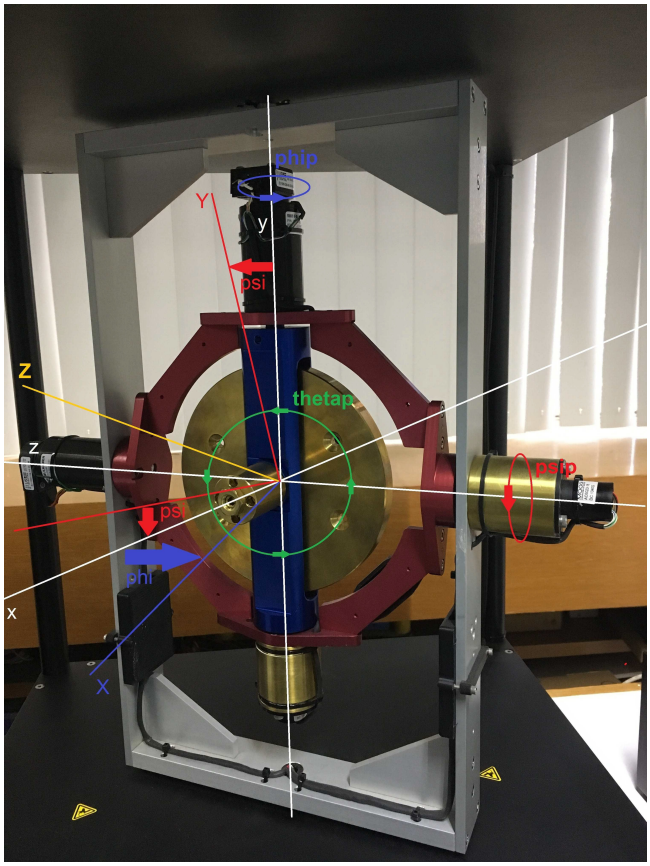

Figura 7.1: Giroscopio, marco de referencia: $theta = \theta$, $psi = \psi$, $phi = \phi$, $thetap = \dot{\theta}$, $psip = \dot{\psi}$, $phip = \dot{\phi}$.

J_x : momento de inercia del disco y la suspensión cardán azul alrededor del eje x. $J_x = 0.0074 kgm^2$

J_y : momento de inercia del disco y la suspensión cardán azul alrededor del eje y. $J_y = 0.0026 kgm^2$

J_z : momento de inercia del disco y la suspensión cardán azul alrededor del eje z. $J_z = 0.0056 kgm^2$

M_b : par externo total alrededor del eje de rotación de la suspensión cardán azul.

M_r : par externo total alrededor del eje de rotación de la suspensión cardán roja.

El control en lazo cerrado del giroscopio se realiza también a través de un sistema difuso, los detalles de su diseño se pueden encontrar en Aparicio-Santos (2017). En el *NCS* se establecen tres aplicaciones tipo *soft*, a cada de las cuales se le asignan tareas distintas, según se describe en la Figura 7.2. La *APP*1 se encarga de las tareas de lectura de los sensores, la *APP*2 del cálculo de la señal de control y la *APP*3 de las tareas de escritura de los actuadores. El *RM* se implementa en cada una de las aplicaciones, mientras el *CBS* se implementó mediante una tarea centralizada tipo *hard*. Además de realizar las tareas que tienen asignadas, las aplicaciones deben enviar por la red el valor instantáneo de sus funciones de emparejamiento f_i ; $i = 1, 2, 3$.

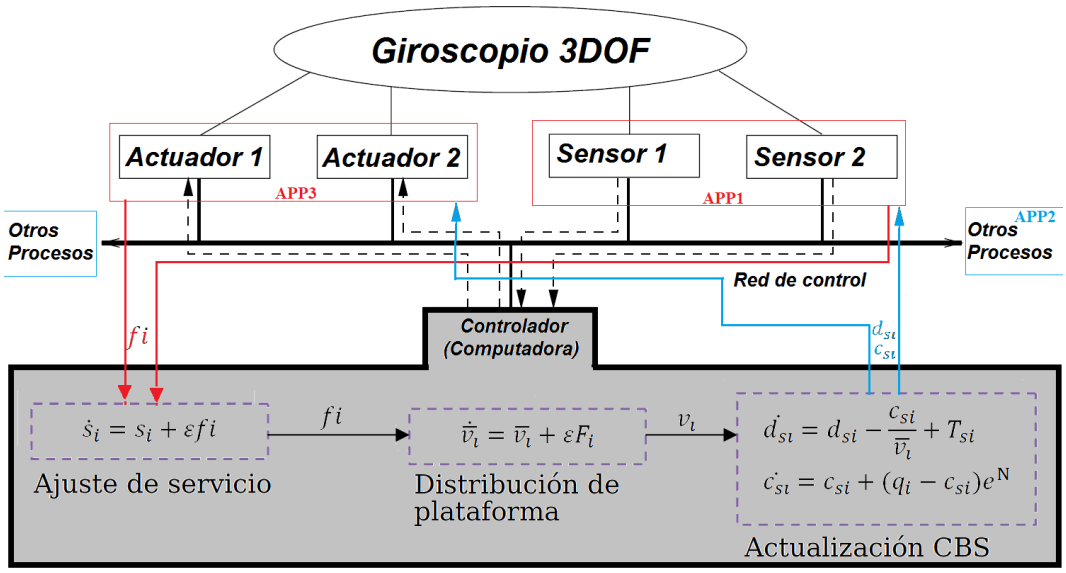

Figura 7.2: Sistema *NCS* configuración directa, conectado al administrador de recursos, se observa que la única información que se recibe de las aplicaciones es el valor de la función de emparejamiento f_i .

Para el primer experimento se usan las siguientes condiciones iniciales: Todas las plataformas virtuales son cero, se tienen tres procesadores con diferentes capacidades $P_1 = 2$, $P_2 = 3$, $P_3 = 4$, que sirven a seis tareas, con diferentes necesidades, para simplificar esto, las necesidades se pueden ver como la variable β_i y están ordenadas de mayor a menor dependiendo de la necesidad de la aplicación, es decir, $\beta_1 < \beta_2 ... < \beta_6$. Los resultado se describen a continuación:

En la Figura 7.3, se observa que la plataforma para APP6 es más exigente que las demás plataforma, por eso toma una cantidad de recursos mucho mayor. Son apreciables ciertas oscilaciones debido al balanceo de recursos entre las aplicaciones, estás tienden a disminuir conforme el algoritmo encuentra una distribución de recursos justa. En la Figura 7.4, se muestra que APP3 migra del procesador 1 al 2 en la iteración 4, el procesador 3 y 2 no presentan migraciones debido a que su capacidad es mayor que la del procesador 1. En la Figura 7.5, la función de emparejamiento presenta oscilaciones debido a los cambios en las plataformas virtuales, conforme el algoritmo se vaya acercando a una distribución justa estás irán disminuyendo, sólo APP6 llega a converger a cero en las veinte iteraciones, por lo tanto, localmente APP6 no pide más recursos. Por último en la Figura 7.6 se aprecia que a pesar de lo visto en la función de emparejamiento, la función de equidad converge a cero para todas las aplicaciones, por lo que se concluye que la distribución es justa.

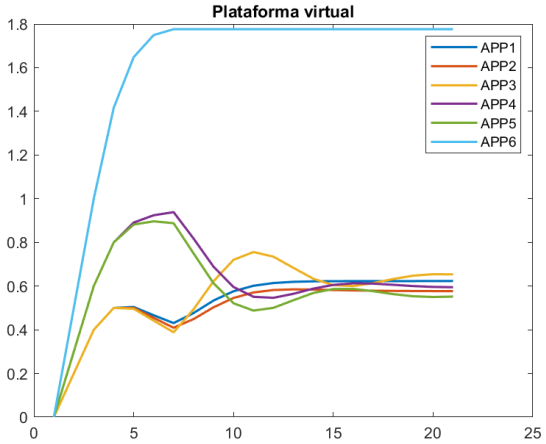

Figura 7.3: Plataforma virtual $v_{i,k}$, caso 1.

Para el segundo experimento, se tienen lo mismos parámetros de que en la simulación anterior, sólo que se disminuye la capacidad de los procesadores: Todas las plataformas virtuales son cero, se tienen tres procesadores con diferentes capacidades $P_1 = 1$, $P_2 = 2$, $P_3 = 3$, que sirven a seis tareas. Arrojando los siguientes resultados:

En la Figura 7.7, se observa que a pesar de tener menos capacidad la distribución para APP6 es igual al caso 1, presentando las mismas oscilaciones debido al in-

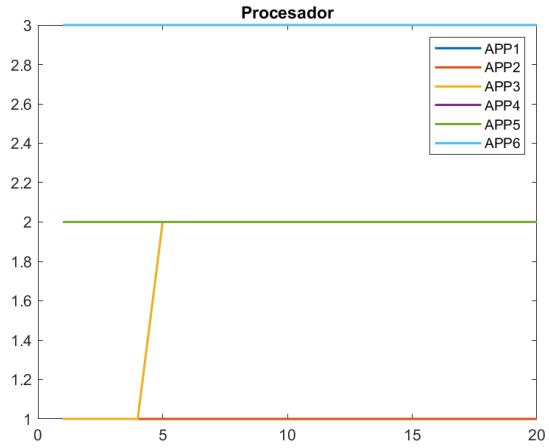

Figura 7.4: Migración entre procesadores, caso 1.

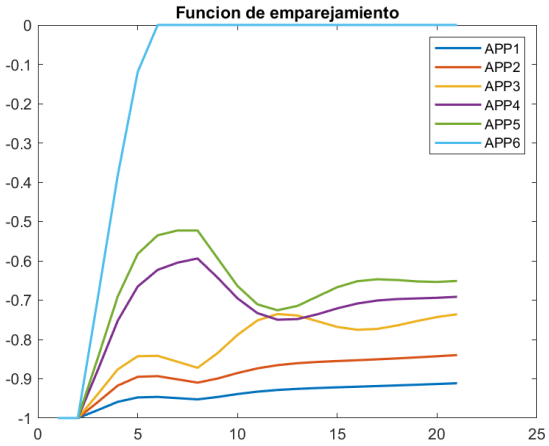

Figura 7.5: Función de emparejamiento $f_{i,k}$, caso 1.

tercambio de recursos entre las aplicaciones, sin embargo la distribución final de recursos de las demás aplicaciones se nota más dispersa. En la Figura 7.8, una vez más se nota una migración de APP3 del procesador 1 al 2. En la Figura 7.9, de manera similar que en el caso 1, APP6 llega a una satisfacción de sus recursos pues sólo es ejecutada por un procesador. Por último en la Figura 7.10 las funciones de emparejamiento vuelven a converger presentando oscilaciones que tienden a disminuir conforme el algoritmo evoluciona y hace un mejor balanceo

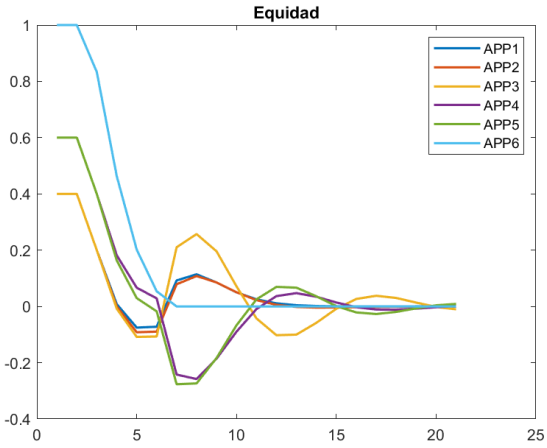

Figura 7.6: Función de equidad nominal $F_{i,k}$, caso 1.

entre los recursos de las distintas aplicaciones, por lo que se concluye que la distribución es justa.

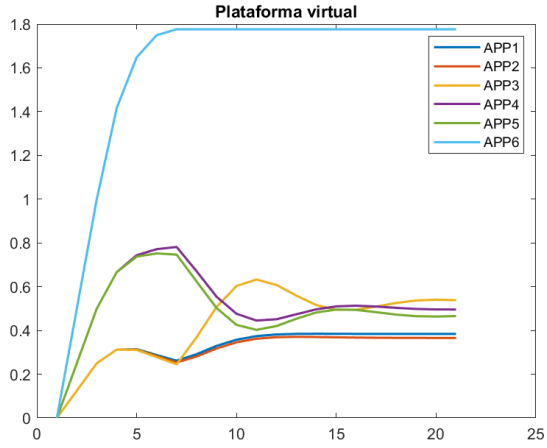

Figura 7.7: Plataforma virtual $v_{i,k}$, caso 2.

Por último se hace un experimento con características similares al segundo caso pero usando el algoritmo homogéneo, con tres procesadores iguales con capacidad $P = 2$, arrojando los siguientes resultados.

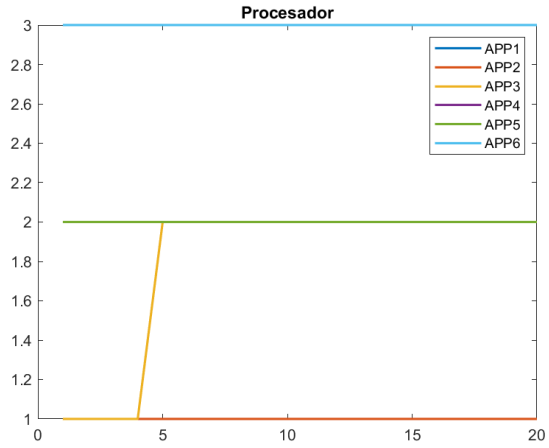

Figura 7.8: Migración entre procesadores caso 2.

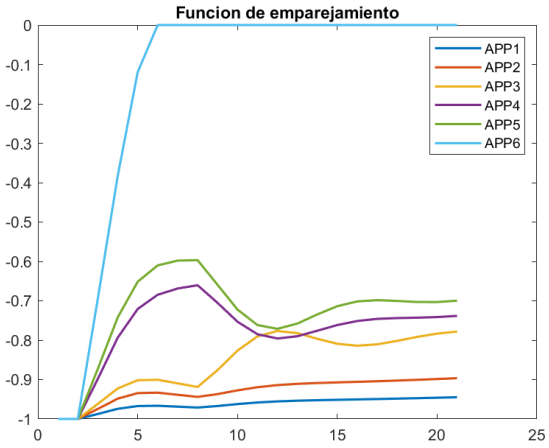

Figura 7.9: Función de emparejamiento $f_{i,k}$, caso 2.

Se puede observar que el algoritmo parece tender a la divergencia, debido al exceso de consumo por las aplicaciones; la convergencia no es posible, como se observa en la Figura 7.11. Al igual que en el caso heterogéneo, se tienen oscilaciones debido al intercambio de recursos entre las aplicaciones. La función de emparejamiento confirma parece tender a la convergencia, pero la demanda de recursos de las aplicaciones es muy grande, con lo cual no es posible obtener una

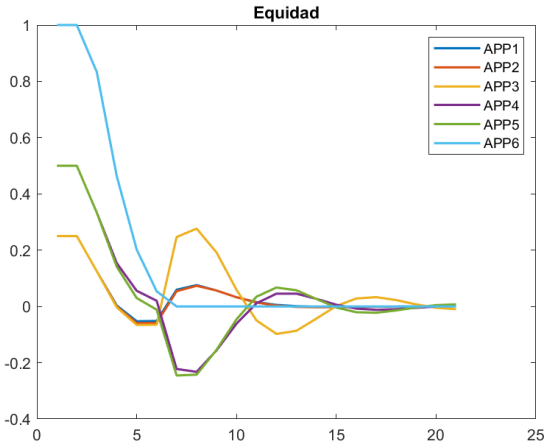

Figura 7.10: Función de equidad nominal F_i, caso 2.

distribución justa, este algoritmo en 20 iteraciones no ha alcanzado su objetivo cómo se observa en la figura 7.12, al igual que la función de equidad 7.13, se puede asegurar que bajo estas condiciones el algoritmo homogéneo no es capaz calcular la distribución a diferencia del algoritmo heterogéneo, a pesar que se ha suprimido la parte de la migración de tareas.

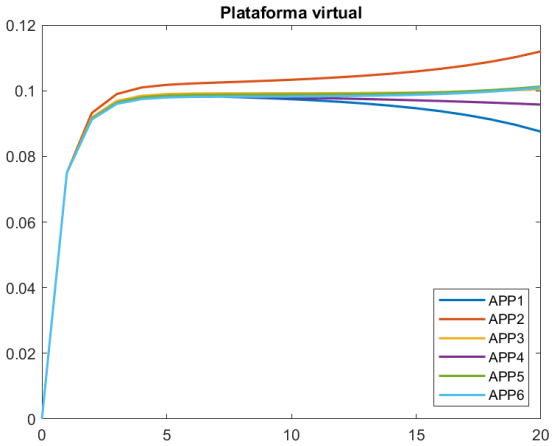

Figura 7.11: Plataforma virtual $v_{i,k}$, caso 3.

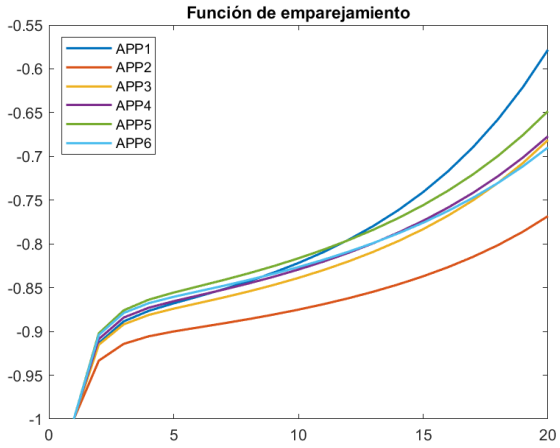

Figura 7.12: Función de emparejamiento $f_{i,k}$, caso 3.

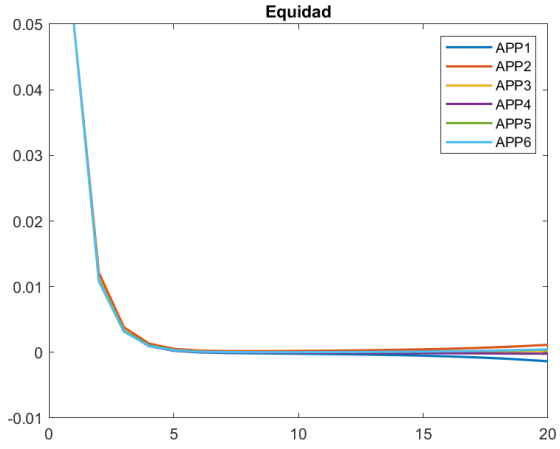

Figura 7.13: Función de equidad nominal F_i, caso 3.

7.1 Implementación Cuadricóptero

Para realizar la implementación se utilizan los bloques *Stream Client* (Figura 7.14) y *Stream Server* (Figura 7.15), estos bloques se conectan a un *host* remoto y envía y/o recibe datos de ese *host*. Estos bloques se encuentran en *Simulink* en el apartado de comunicaciones básico de *Quanser*.

7.2 Bloques de comunicación

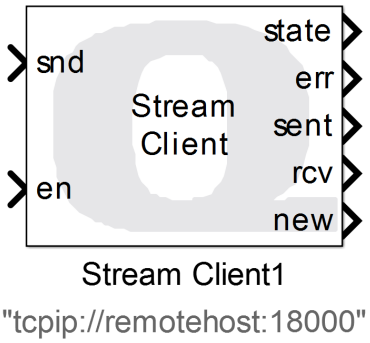

Figura 7.14: Bloque *Stream Client* en *Simulink* con la configuración predeterminada

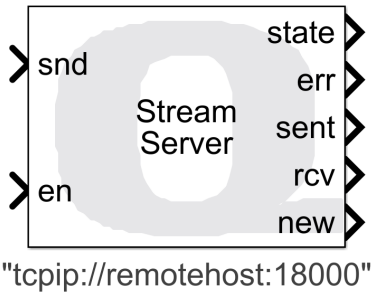

Figura 7.15: Bloque *Stream Server* en *Simulink* con la configuración por defecto

El bloque *Stream Client* permite el envío y recepción de datos, es configurable para sólo enviar o sólo recibir, en nuestro sistema los sensores envían datos (señal sensada), los actuadores reciben datos (señal de control) y el controlador recibe (señal sensada) y envía datos (señal de control).

Con estos bloque se puede utilizar una configuración cliente/servidor como la que se observa en la Figura 7.16, donde el cliente *Stream Client* hace peticiones al servidor y el servidor *Stream Server* que está conectado directamente recibiendo información de los sensores y actuadores del sistema da respuesta al cliente.
Se puede colocar más de un bloque por equipo, depende de cuantos equipos participan en el sistema.

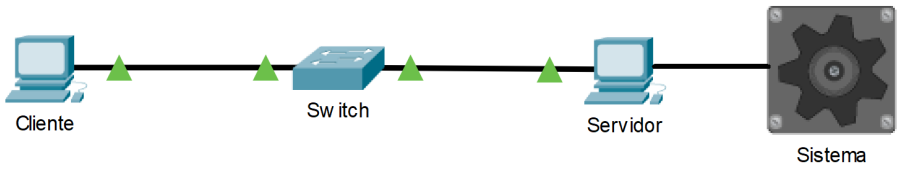

Figura 7.16: Configuración básica cliente/servidor

Es necesario conocer la IP de las computadoras a conectar, si se requiere conectar un equipo adicional es necesario colocar otro bloque. Ya sea servidor o cliente, los bloques son capaces de enviar y recibir datos.

Una vez configurado el bloque ya se puede utilizar para enviar y recibir, la conexión depende de la aplicación. En la Figura 7.17 se muestra una conexión básica para recibir/enviar datos, por la gráfica se puede observar que se envía el valor de *Constant*, que en este caso es 0, el estado del bloque se observa en el *Scope1*, los datos que se reciben se muestran en el *Scope2*.

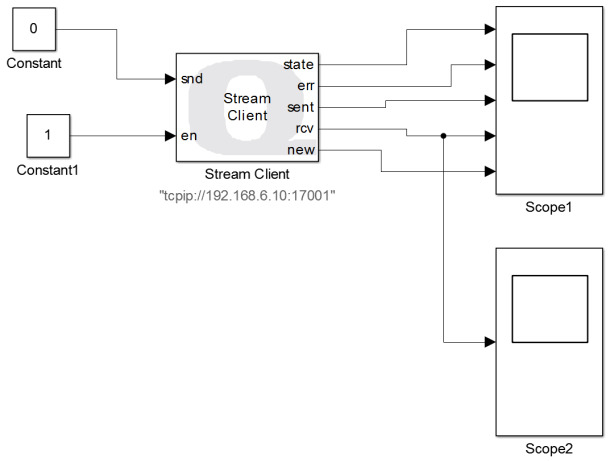

Figura 7.17: Conexión básica del bloque *Stream Client*

Los datos recibidos pueden ser extraídos de la salida **rcv**, los datos que desean ser enviados se introducen en la entrada **snd**, la entrada **en** habilita la trasmisión si es ”1”, en caso de que sea ”0”, el bloque no envía ni recibe nada, esta entrada puede ser utilizada para controlar la tasa de trasmisión.

El bloque *Stream Server*, es muy parecido al bloque *stream client*, la única diferencia entre ambos es el papel que desempeñan en la configuración cliente/servidor.

Para realizar la administración de la comunicación se hace uso de la entrada de habilitación, dado que una vez que el programa está ejecutándose no deja hacer modificaciones a otros parámetros, así se puede balancear el ancho de banda como recurso global y utilizar el tiempo de muestreo como recurso local.

La operación de cualquier sensor requiere de un mínimo de ancho de banda así como de una frecuencia de muestreo mínima, es necesario experimentar con cada uno para conocer sus límites inferiores de funcionamiento.

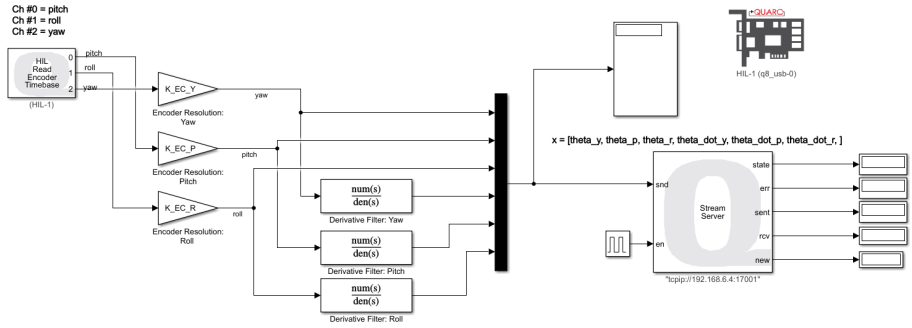

Figura 7.18: Envío de la información de encoders con el bloque *Stream Server*

7.3 Esquema del sistema

Se tiene el siguiente esquema del sistema
Pueden existir diversas configuraciones en el intercambio de información, en este caso, PC0 envía la señal de control a los actuadores por medio de PC2, y recibe la señal de los sensores por medio de PC1. Cada equipo cuenta ejecutando un programa de *Simulink* distinto, los cuales se muestran en las Figuras (7.20)-(7.22).

7.4 Computadora Controlador/RM del Cuadricóptero

El controlador utiliza dos bloques *Stream Client*; uno por cada computadora con la que se desee comunicar, recibe las señales de los sensores y envía la señal de con-

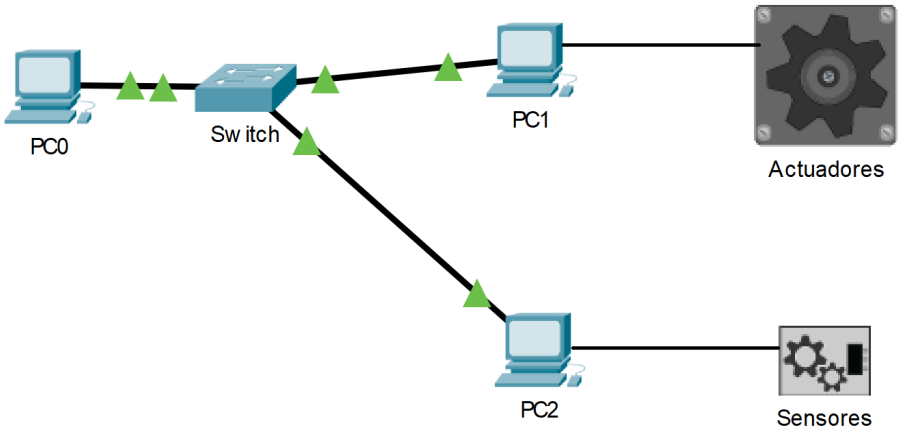

Figura 7.19: Esquema de la conexión de las PC

trol a los actuadores, también recibe señales con información sobre el rendimiento de la de las demás partes del sistema relacionadas con el ancho de banda y tiempo de muestreo.

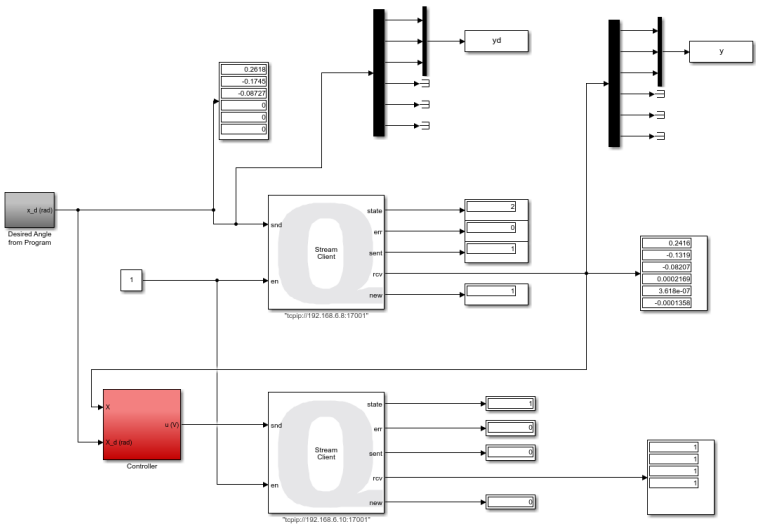

Figura 7.20: Esquema del controlador del cuadricóptero, junto con el *RM*

7.5 Computadora Actuadores del Cuadricóptero

La computadora que gestiona los actuadores utiliza el bloque *stream server*, gracias a este bloque recibe la señal de control del controlador, y envía su función de emparejamiento al RM.

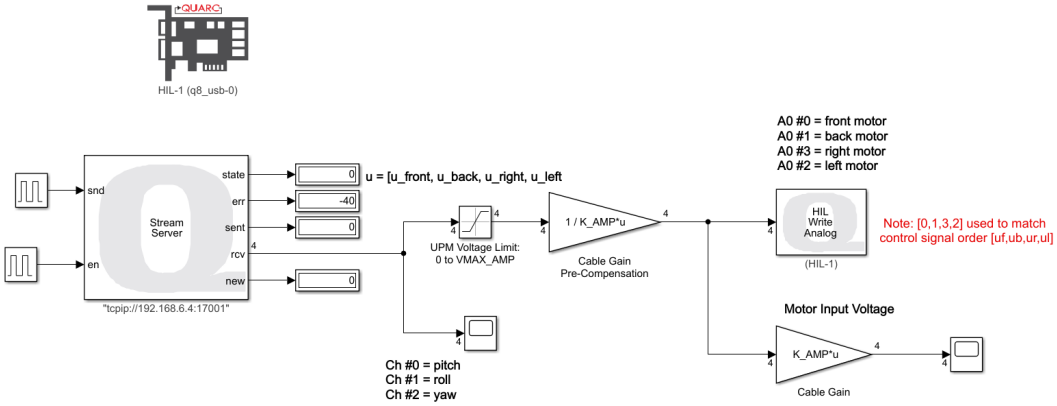

Figura 7.21: Esquema para los actuadores del cuadricóptero

7.6 Computadora Sensores del Cuadricóptero

La señal de los sensores es leída y enviada a la PC del controlador, también es enviada la función de emparejamiento para que el *RM* pueda calcular la nueva tasa de envío.

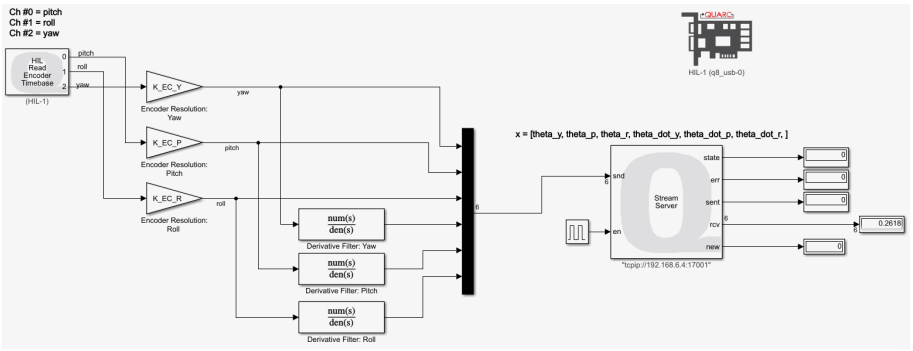

Figura 7.22: Esquema para la lectura de sensores del cuadricóptero

7.7 Resultados Cuadricóptero

Se aplicó este algoritmo a un cuadricóptero (Figura 7.23), cuya señal deseada es un escalón a las coordenadas *Yaw* y *Picth*, siguiendo el modelo presentado en Castillo (2020), se obtuvo la siguiente figura de las señales deseadas contra las señales sensadas.

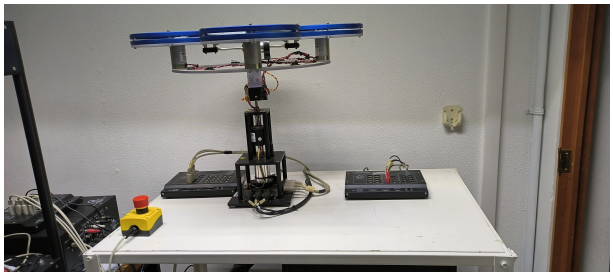

Figura 7.23: Cuadricóptero conectado a dos PCs por medio de dos tarjetas de adquisición de datos, izquierda lectura de encoders, derecha escritura a actuadores

El algoritmo calcula los tiempos habilitación de los bloques en función de las señales recibidas, hasta cumplir con los parámetros mínimos para que los recursos sean considerados justos, en la Figura 7.24 se observa las gráficas de la señal deseada contra la señal medida en los sensores.

Se observa que las señales sensadas no igualan a las señales deseadas, por lo que se tienen un error en estado estacionario. Es importante notar que este error no sólo es resultado del controlador del cuadricóptero si no también de la distribución de los recursos.

7.8 Implementación con el Giroscopio

Llevando a cabo una distribución similar que con el cuadricóptero, se aplicó el algoritmo al sistema de giroscopio mostrado en la Figura 7.1.

Los esquemas de *Simulink* en las computadoras difieren principalmente en la parte del controlador debido a la diferencia entre los mecanismos. Se presentan a continuación dichos esquemas.

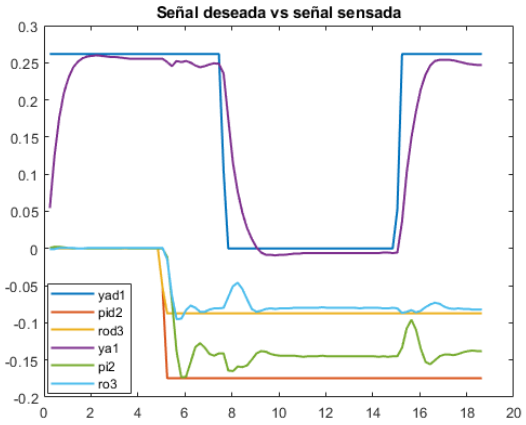

Figura 7.24: Resultados del cuadricóptero, donde las señales deseadas son yad1 (*yaw*), pid2 (*pitch*), rod3 (*roll*); ya1, pi2, ro3 son las señales sensadas

7.9 Computadora Controlador/RM del Giroscopio

El controlador utiliza dos bloques *Stream Client*; debido a que esta computadora hará el papel de servidor, se muestra el esquema en la Figura 7.25

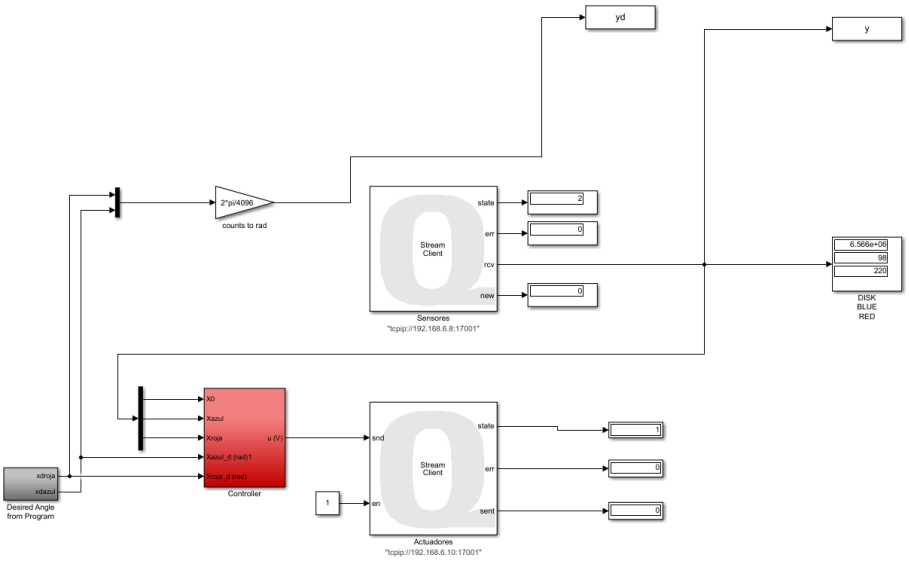

Figura 7.25: Esquema del controlador del giroscopio, junto con el *RM*

7.10 Computadora Actuadores del Giroscopio

La computadora que envía la señal a los actuadores recibe la información por medio del bloque *stream server*.

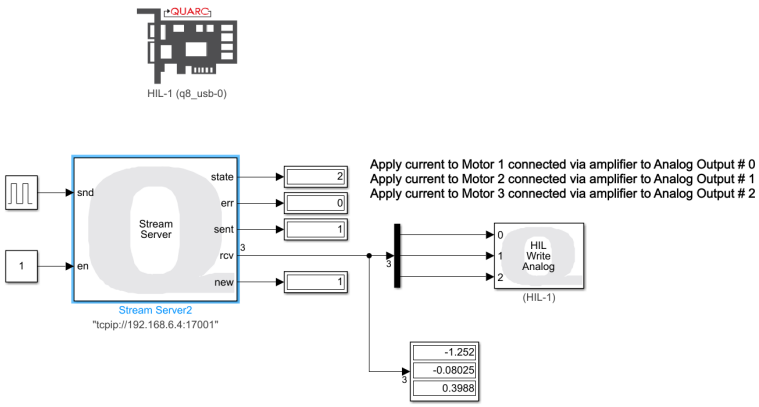

Figura 7.26: Esquema para los actuadores del giroscopio

7.11 Computadora Sensores del Giroscopio

La información de los sensores se envían al controlador por medio del bloque *stream server*.

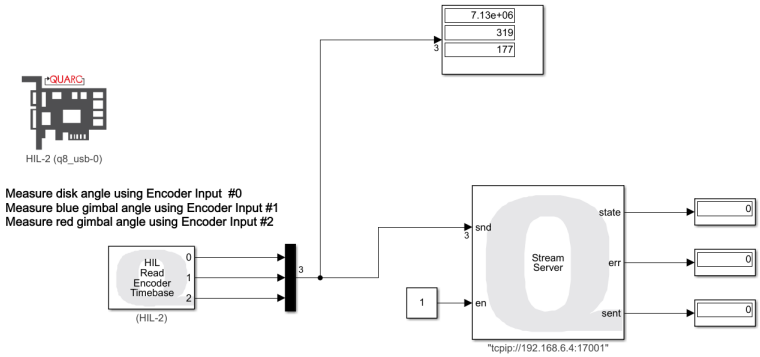

Figura 7.27: Esquema para la lectura de sensores del giroscopio

7.12 Resultados Giroscopio

Lo que se controla en el giroscopio son sus dos suspensiones, una roja y un azul (ver Figura 7.1), en esté caso a la suspensión roja (y2) se aplica una señal deseada escalón (yd2), la suspensión azul (y1), a pesar enviarle una señal de control cero (yd11) es perturbada por el movimiento de la suspensión roja.

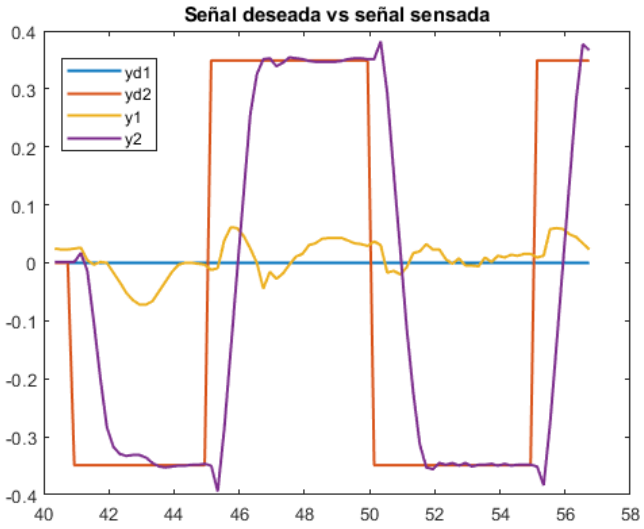

Figura 7.28: Resultado, donde las señales deseadas son yd1 (suspensión azul) y yd2 (suspensión roja), la señales sensadas son y1 (suspensión azul) y y2 (suspensión roja)

7.13 Recapitulación

Como se observa en las gráficas los efectos de la red se hacen presentes en ambos sistemas, se llevaron a cabo prueba para estimar sus limites de funcionamiento mínimos, ya que ese será su nivel de servicio, se debe tener cuidado al accionar los sistemas ya que un orden de activación incorrecto puede llevar a la inestabilidad antes de ejecutar el algoritmo, el orden seguido es sensores, actuadores, controlador.

Un punto importante a tomar en cuenta en esta implementación es la incapacidad de migrar tareas debido a que los sistemas están conectados físicamente a las tarjetas y se tendría que necesitar alguna especia de multiplexor entre las diferentes tarjetas.

Capítulo 8

Conclusiones

Se presentó un modelo de *RM* para procesadores heterogéneos descrito por ecuaciones en diferencias, las cuales toman en cuenta la distribución de recursos globales y locales, así como el comportamiento del *CBS*. Este *RM* toma en cuenta el desempeño de las tareas en cada partición del procesador, en caso de no cumplir con un desempeño esperado, la tarea se puede migrar a otro procesador o ser descartada.

La principal desventaja de este esquema es que la velocidad de convergencia está ligada al número de tareas, a mayor número de tareas la convergencia tiende a ser más lenta, tomando más iteraciones para lograr su objetivo, lo que puede ser impráctico, sobre todo en *RTAs*.

Se diseñó una representación por medio de ecuaciones en diferencias del *RM*, esto con el fin de obtener una descripción matemática de cada componentes del sistema. La ecuaciones describen la distribución de recursos, y el rellenado de presupuesto del *CBS*, así como el aplazamiento del *deadline*.

A partir de esta descripción, se pudo diseñar una ley de control difuso para buscar convergencia a distribuciones justas, cuya convergencia se probó a partir de un análisis por linealización. Se optó por el control difuso por su versatilidad como aproximador lineal, sin embargo, el modelo no está restringido a este esquema de control.

Se comparó el esquema heterogéneo con el esquema homogéneo descrito en el capítulo 4 y en Aparicio-Santos et al. (2021), estos dos esquemas tienen como

principal diferencia las restricciones en el sistema a nivel *hardware* debido la naturaleza de los procesadores, de igual manera se puede tener restricción a nivel *software*, por ejemplo, a causa de diferentes sistemas operativos.

Se presentaron resultados de simulaciones que confirman la viabilidad del esquema propuesto y enfatizan la necesidad de contar con los recursos mínimos para su correcta ejecución.

Se desarrolló experimentalmente una red de sensores y actuadores para administrar el ancho de banda entre cada participante de la red.

Bibliografía

Abdel-Basset, M., Mohamed, R., Abouhawwash, M., Chakrabortty, R. K., and Ryan, M. J. (2021). Ea-msca: An effective energy-aware multi-objective modified sine-cosine algorithm for real-time task scheduling in multiprocessor systems: Methods and analysis. *Expert Systems with Applications*, 173:114699.

Abeni, L. and Buttazzo, G. (1998). Integrating multimedia applications in hard real-time systems. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 4–13.

Aparicio-Santos, J.-A. (2017). Diseño de un controlador difuso para compensar cargas de comunicación en tiempo real. Master's thesis, Universidad Nacional Autónoma de México, México.

Aparicio-Santos, J.-A., Ángel Hermosillo-Gómez, J., Benítez-Pérez, H., and Álvarez Icaza-Longoria, L. (2021). Controlador difuso para compensar cargas de comunicación en sistemas en tiempo real. *Revista Iberoamericana de Automática e Informática industrial*, 18(3).

Baruah, S. (2002). Robustness results concerning edf scheduling upon uniform multiprocessors. In *Proceedings 14th Euromicro Conference on Real-Time Systems. Euromicro RTS 2002*, pages 95–102.

Baruah, S. (2006). The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32:9–20.

Baruah, S., Bertogna, M., and Buttazzo, G. (2015). *Multiprocessor Scheduling for Real-Time Systems*. Springer Publishing Company, Incorporated.

Bini, E., Buttazzo, G., Eker, J., Schorr, S., Guerra, R., Fohler, G., Arzen, K. E., Romero, V., and Scordino, C. (2011). Resource management on multicore systems: The actors approach. *IEEE Micro*, 31(3):72–81.

Boutalis, Y., Theodoridis, D., Kottas, T., and Christodoulou, M. A. (2014). *System Identification and Adaptive Control: Theory and Applications of the Neurofuzzy and Fuzzy Cognitive Network Models*. Springer.

Buttazzo, G. C. (2011). *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition.

Byeong Gi, L., Daeyoung, P., and Hanbyul, S. (2009). *Wireless Communications Resource Managemen*. John Wiley and Sons.

Carpenter, J., Funk, S., Holman, P., Anderson, J., and Baruah, S. (2003). A categorization of real-time multiprocessor scheduling problems and algorithms.

Castillo, O. O. (2020). *Sistemas muestreados estabilizados mediante señal de control pulsada*. PhD thesis, Universidad Nacional Autónoma de México.

Chasparis, G., Maggio, M., Arzen, K. E., and Bini, E. (2013). Distributed management of cpu resources for time-sensitive applications. In *2013 American Control Conference*, pages 5305–5312.

Chasparis, G. C., Maggio, M., Bini, E., and Arzen, K.-E. (2016). Desing and implementation of distributed resource management for time-sensitive applications. *Automatica*, 64:44 – 53.

Chatterjee, S. (1997). A quality of service based allocation and routing algorithm for distributed, heterogeneous real time systems. In *Proceedings of 17th International Conference on Distributed Computing Systems*, pages 235–242.

Clark, R. K. (1990). *Scheduling Dependent Real-Time Activities*. PhD thesis, Carnegie Mellon University, USA. AAI9107552.

Di Natale, M. and Sangiovanni-Vincentelli, A. L. (2010). Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proceedings of the IEEE*, 98(4):603–620.

Fang, C.-H., Liu, Y.-S., Kau, S.-W., Hong, L., and Lee, C.-H. (2006). A new lmi-based approach to relaxed quadratic stabilization of t-s fuzzy control systems. *IEEE Transactions on Fuzzy Systems*, 14(3):386–397.

Hermosillo, J.-A. (2020). *Algoritmos de planificación de tiempo real en sistemas distribuidos hetérogeneos considerando la migración de tareas*. PhD thesis, Universidad Nacional Autónoma de México.

Horn, W. (1974). Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185.

Hristu-Varsakelis, D., Levine, W. S., Alur, R., Arzen, K.-E., Baillieul, J., and Henzinger, T. A. (2005). *Handbook of Networked and Embedded Control Systems (Control Engineering)*. Birkhauser.

Huh, E.-N., Welch, L., Shirazi, B., and Cavanaugh, C. (2000). Heterogeneous resource management for dynamic real-time systems. In *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556)*, pages 287–296.

IEEE (1990). Ieee standard glossary of software engineering terminology.

Li H., Sun Z., L. H. C. W. (2008). Predictive observer-based control for networked control systems with delay and packet dropout. *Asian Journal of Control*, 10(6):1–3.

Li K., B. J. (2004). Robust quantization for digital finite communication bandwidth (dfcb) control. *IEEE Trans. on Automatic Control*, 49(9):1573–1584.

Mahmoud, M., of Engineering, I., and Technology (2013). *Distributed Control and Filtering for Industrial Systems*. Control, Robotics and Sensors. Institution of Engineering and Technology.

Mok, A. K. and Feng, X. (2001). Resource partition for real-time systems. In *Proceedings Seventh IEEE Real-Time Technology and Applications Symposium*, pages 75–84.

Nesbit, K. J., Moreto, M., Cazorla, F. J., Ramirez, A., Valero, M., and Smith, J. E. (2008). Multicore resource management. *IEEE Micro*, 28(3):6–16.

Premi, L., Reghenzani, F., Massari, G., and Fornaciari, W. (2020). A game theory approach to heterogeneous resource management: Work-in-progress. In *2020 International Conference on Embedded Software (EMSOFT)*, pages 25–27.

Quanser (2012). *USER MANUAL 3 DOF Gyroscope Experiment Set Up and Configuration*. Quanser inc.

Razian, S. and MahvashMohammadi, H. (2017). Optimizing raytracing algorithm using cuda. *Italian Journal of Science and Engineering*, 1:167–178.

Robert H. Cannon, J. (2003). *Dynamics Of Physical Systems*. Dover Publications, INC.

Seidi, M., Hajiaghamemar, M., and Segee, B. (2012). Fuzzy control systems: Lmi-based design.

Seitz, N. (2003). Itu-t qos standards for ip-based networks. *Communications Magazine, IEEE*, 41:82 – 89.

Sommerville, I. (2011). *Software engineering*. Addison-Wesley, 9th ed. edition.

Stankovic, J. A. (1988). Misconceptions about real-time computing: a serious problem for next-generation systems. *Computer*, 21(10):10–19.

Subrata, R., Zomaya, A. Y., and Landfeldt, B. (2008). A cooperative game framework for qos guided job allocation schemes in grids. *IEEE Transactions on Computers*, 57(10):1413–1422.

Tanaka, K., Ikeda, T., and Wang, H. O. (1998). Fuzzy regulators and fuzzy observers: relaxed stability conditions and lmi-based designs. *IEEE Transactions on Fuzzy Systems*, 6(2):250–265.

Tanaka, K. and Wang, H. O. (2001). *Fuzzy Control Systems Design and Analysis: A Linear Matrix Inequality Approach*. Wiley-Interscience.

Wang, Y.-L. and Han, Q.-L. (2018). Network-based modelling and dynamic output feedback control for unmanned marine vehicles in network environments. *Automatica*, 91:43–53.

Y., T. and Chow, M. Y. (2003). Control methodologies in networked control systems. *Control Engineering Practice*, 11:1099–1111.

Zhang, X.-M., Han, Q.-L., Ge, X., Ding, D., Ding, L., Yue, D., and Peng, C. (2020). Networked control systems: a survey of trends and techniques. *IEEE/CAA Journal of Automatica Sinica*, 7(1):1–17.