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

DISEÑO DE UN CONTROLADOR DIFUSO PARA LA DISTRIBUCIÓN DE RECURSOS ENTRE PROCESADORES HETEROGÉNEOS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
LUIS ENRIQUE MENDOZA RODRÍGUEZ

TUTOR
DR. HÉCTOR BENÍTEZ PÉREZ
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS

CIUDAD UNIVERSITARIA, CD. MX. AGOSTO 2023

Resumen

El propósito de esta tesis es desarrollar un método para optimizar la utilización de los recursos en una red de dispositivos que poseen diferentes capacidades de procesamiento.

Índice general

Índice de figuras

Capítulo 1

Introducción

Podemos utilizar un ejemplo particular para poder clarificar cuál es el propósito de esta tesis.

Supongamos un caso en el que un conductor llega a detener su vehículo en un semáforo, su teléfono celular en ese momento se conecta con los teléfonos de los demás conductores para llevar a cabo alguna tarea de manera cooperativa. Para ello hay varios parámetros a considerar:

No sabemos en que momento el conductor ha llegado al semáforo, pudo llegar tanto cuando el semáforo cambió a rojo, como poco antes de cambiar a verde, por lo que no sabemos cuanto tiempo puedan estar los celulares trabajando juntos.

No todos los celulares tienen la misma capacidad de procesamiento, por lo que una tarea puede tardar más en un celular que en otro, si no se elige el celular adecuado para ejecutar la tarea, se corre el riesgo de que no termine a tiempo.

Existen aplicaciones que tienen mayor prioridad para ser terminadas o ejecutadas. Además, la prioridad en la ejecución del programa puede llegar a cambiar de un momento a otro.

Habrá momentos en que la capacidad que tienen los teléfonos no sea suficiente para ejecutar todas las tareas, esto podría deberse a que no hay suficientes celulares o a la limitada capacidad de procesamiento de los mismos.

Por estos motivos es necesario usar una aplicación que coordine las tareas a realizar, así como asignar el tiempo de ejecución, manejar las prioridades de ejecución de

las aplicaciones y elegir qué procesador es más adecuado para llevar a cabo las tareas. A una aplicación que cumpla con lo anterior se le llama manejador de recursos (RM).

El presente trabajo tiene como objetivo el diseño de un manejador de recursos mediante el uso de un control difuso para la distribución de tareas en una red de dispositivos que presenta situaciones similares a las planteadas en el ejemplo.

1.1. Objetivos

La presente tesis tiene como objetivo el diseño de un manejador de recursos por medio de un control difuso para la distribución de tareas en procesadores heterogéneos.

1.2. Contribución

La contribución de la presente tesis es la aplicación de un manejador de recursos (RM) en varios procesadores heterogéneos mediante comunicación por Wi-Fi y protocolo SSH. Se utiliza la teoría presentada en [Santos (2017)] y en [Aparicio-Santos and Benítez-Pérez (2021)] para su implementación en una red general, abordando y resolviendo problemas que no se abordan en los documentos citados.

Además, se propone un método para reducir los tiempos de ejecución de sistemas que utilicen el modelo difuso de Takagi-Sugeno, sin embargo, el resultado obtenido será una aproximación mayor o igual a un porcentaje del valor final que nosotros fijamos.

1.3. Organización de la tesis

En el capítulo 2 se expondrá la teoría necesaria para comprender el funcionamiento de un manejador de recursos híbrido, así como la teoría de los sistemas difusos que se utilizará para realizar el control del RM.

En el capítulo 3 se desarrolla la implementación de la teoría anterior en conjunto con el diseño del manejador de recursos híbrido para múltiples procesadores.

En el capítulo 4 se da a conocer algunas diferencias que tendrá la implementación del RM con lo propuesto en artículos en los cuales esta basado este trabajo. Se resuelven los problemas técnicos que se presentan en la implementación del RM.

En el capítulo 5 se centra en el código principal, el cual realiza la interconexión entre los dispositivos y el RM, además de manejar el código de este último.

En el capítulo 6 Se llevan a cabo las pruebas al RM, se presentan los resultados obtenidos y se realiza el análisis de correspondiente.

Capítulo 2

Marco teórico

2.1. Introducción

En el presente capítulo se expondrá la teoría referente al manejador de recursos RM, el servidor de ancho de banda constante CBS y el control difuso

2.2. Plataformas virtuales y Calidad de servicio (Quality of service QoS)

El manejador de recursos RM se encarga de asignar un tiempo de procesamiento justo a las aplicaciones cada periodo de tiempo. 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 la proporción de uso que hace del procesador real, es decir, por la fracción utilizada de la potencia disponible de dicho procesador.

Por ejemplo, si tenemos un procesador con una frecuencia de 1[GHz] podemos utilizar plataformas virtuales cuya suma sea igual o menor a la frecuencia total, como en el caso $v_1 = 0.5[GHz]$ $v_2 = 0.3[GHz]$ $v_3 = 0.2[GHz]$

La QoS es el efecto colectivo del desempeño del servicio, el cual determina el grado de satisfacción del usuario[ITU (1994)]

2.3. Nivel de servicio

Una variable a través de la cual se puede ajustar el rendimiento de las aplicaciones es el nivel de servicio. Los niveles de servicio dependen de las características configurables de las aplicaciones y determinan la calidad/precisión deseada. Algunos ejemplos son el ajuste de la resolución de un video o la cantidad de datos enviados por un canal para representar una página web [Maggio et al. (2013)].

En nuestro caso, el nivel de servicio está relacionado con la cantidad de tiempo de procesamiento que una aplicación solicita para ejecutarse en cada ciclo, mientras que la plataforma virtual está relacionada con el tiempo asignado a la aplicación para su ejecución. El objetivo es lograr que ambos valores coincidan, tomando en cuenta que la plataforma virtual se encuentra limitada por la capacidad del procesador en el cual se ejecuta. Por tanto, habrá situaciones en las que sea necesario disminuir el nivel de servicio para que sea adecuado.

2.4. Manejador de recursos

Un manejador de recursos RM se encarga de: (a) asignar procesadores virtuales a las aplicaciones, (b) monitorear el uso de recursos, y (c) asignar el nivel de servicio a cada aplicación. El objetivo del RM es maximizar la calidad de servicio QoS [Chasparis et al. (2016)].

Un manejador de recursos se encarga de asignar recursos de memoria y procesamiento adecuados a los procesos en ejecución. La asignación de recursos consiste en la capacidad de minimizar los compromisos de rendimiento de varias cargas de trabajo, además de supervisar el uso de los recursos [Sommerville (2011)].

Existen tres esquemas de RM: Centralizado, distribuido e híbrido.

En el esquema centralizado, el RM tiene control sobre todos los recursos de cada tarea, lo que lo hace muy estable. Sin embargo, este enfoque presenta varias debilidades. En primer lugar, la complejidad de los algoritmos utilizados para implementar el RM aumenta significativamente con el número de aplicaciones, lo que lo vuelve

impráctico, ya que puede llegar a consumir una mayor cantidad de recursos que los que debe distribuir de manera óptima [Chasparis et al. (2016)].

En segundo lugar, es difícil implementar una función de costos para estos problemas. El RM debe comparar la calidad de diferentes aplicaciones, pero esta comparación carece de sentido, ya que el concepto de calidad es dependiente de cada aplicación [Chasparis et al. (2016)].

Por último, para lograr una asignación adecuada de niveles de servicio, el RM debe tener conocimiento de las aplicaciones. Las aplicaciones deben informar al RM sobre el nivel de servicio disponible y los recursos que se consumirán para cada nivel de servicio, lo cual incrementa significativamente la complejidad de la comunicación[Chasparis et al. (2016)].

El esquema distribuido se basa en la información local que las tareas intercambian entre sí. De esta manera, cada tarea regula su propio consumo de recursos mediante un RM local para cada una. Sin embargo, la principal debilidad de este esquema radica en que la distribución global de recursos puede volverse inestable. Esto se debe a que la convergencia de cada aplicación puede no ser estable, lo que dificulta garantizar la optimización global.

El esquema híbrido aprovecha las ventajas de los sistemas centralizados y distribuidos. Mediante un RM global, se distribuyen los recursos globales, el RM global recibe apenas un mínimo de información de las tareas. Por otro lado, cada tarea regula sus propios recursos globales observando su propio desempeño local [Lee et al. (2009)].

Las ventajas que tiene el sistema híbrido, mencionadas en [Aparicio-Santos et al. (2021)] son:

- Complejidad lineal en relación al número de aplicaciones.
- Robustez frente al número y naturaleza de las aplicaciones.
- No es necesario conocer los estados internos de cada aplicación.

2.5. Servidor de ancho de banda constante CBS

El Servidor de Ancho de Banda Constante (CBS) es un algoritmo de planificación de reservación introducido en 1998. Fue diseñado para manejar tareas que se caracterizan por tener una duración de tiempo variable [Abeni et al. (2015)].

Es un mecanismo de servicio con el cual se implementa una estrategia para reservar recursos de proceso[Buttazzo (2011)]

2.5.1. Símbolos y sus definiciones

i	número de tarea
k	número de iteración
si	identificador del servidor que atiende a la tarea i
τ_i	tarea número i
$J_{i,j}$	j-ésimo trabajo (*job*) de la i-ésima tarea. Una tarea puede ser dividida en varios trabajos.
Q_{si}	máximo presupuesto del servidor que atiende a la aplicación i
T_{si}	periodo del servidor que atiende a la aplicación i
$c_{si,k}$	presupuesto actual del servicio
$r_{i,k}$	tiempo de arribo de la aplicación i
$d_{si,k}$	deadline del servidor (tiempo máximo asignado para acabar un *job*) que atiende a la aplicación i

2.5.2. Funcionamiento de CBS

El CBS reserva una fracción del ancho de banda del CPU para cada tarea por medio del presupuesto del servidor c_{si}. Una reserva es una pareja (Q_{si}, T_{si}) indicando que una tarea τ_i puede ejecutarse como máximo $c_{si} = Q_{si}$ unidades de tiempo cada periodo T_{si}.

Definimos dos tipos de tareas, tipo *hard* y tipo *soft*:

Las tareas tipo *hard* deben ser completadas antes de que se sobrepase su *deadline*.

De no lograrse esto provocará un fallo en la realización de la tarea.

Las tareas tipo *soft* puede sobrepasar su *deadline* y mientras sigan siendo ejecutadas, su resultado seguirá siendo válido.

Cada tarea pueden ser síncrona o asíncrona, y a su vez pueden estar compuestas por porciones de código sensibles al tiempo llamados *jobs*. En este trabajo se considera que las tareas están formadas exclusivamente por *jobs* que requieren atención en diferentes intervalos de tiempo.

Las características del CBS se observan en los siguientes puntos, los cuales aparecen en [Buttazzo (2011)].

- El CBS se caracteriza por tener un presupuesto c_s, un presupuesto máximo Q_s, y un periodo del servidor T_s. El valor $U_s = Q_s/T_s$ representa el ancho de banda del servidor. En cada momento, se asocia un *deadline* $d_{si,k}$ al servidor. Al inicio $d_{s,0} = 0$

- A cada *job* que entra al sistema se le asigna un *deadline* igual al actual *deadline* del servidor.

- Cada vez que se ejecuta un *job*, el presupuesto c_s se reduce en la misma cantidad de tiempo utilizado por el *job*.

- Cuando $c_s = 0$, el presupuesto se recarga al valor máximo Q_s y se genera un nuevo *deadline* del servidor como $d_{s,k+1} = d_{s,k} + T_s$.

- Un CBS se considera activo en el tiempo t si hay *jobs* pendientes

- Cuando ingresa un nuevo *job* al CBS pero este está ejecutando otro *job*, el nuevo *job* se coloca en una cola de trabajos pendientes.

- Cuando ingresa un nuevo *job* al CBS y el CBS está inactivo, si $c_s \geq (d_{s,k} - r_{i,j})U_s$, el servidor genera un nuevo *deadline* $d_{s,k+1} = r_{i,j} + T_s$ y el presupuesto c_s se recarga al valor máximo Q_s. De lo contrario, el *job* utiliza el último *deadline* $d_{s,k}$ y el presupuesto actual.

- Cuando se finaliza un *job*, se procede a ejecutar el siguiente *job* pendiente utilizando el *deadline* y presupuesto actuales. En caso de no haber trabajos pendientes, el servidor pasa a estar inactivo.

- En cualquier momento, a un *job* se le puede asignar el último *deadline* generado por el servidor.

El objetivo del CBS es lograr el cumplimiento de los *deadlines* de las aplicaciones.

La condición para lograr el cumplimiento de los *deadlines* se establece en el siguiente lema, tomado de [Buttazzo (2011)]:

Lema 2.5.1. *Dado un conjunto de* n *tareas periódicas hard cuyo uso de procesador es* U_p*, y un conjunto de* m *CBSs con uso de procesador de* $U_s = \sum_{i=1}^{m} U_{s_i}$*, el conjunto completo es planificable si y solo si*

$$U_p + U_s \leq 1 \tag{2.1}$$

2.6. Esquema híbrido con CBS

En la Figura 1 se presenta el esquema híbrido, donde se observa cómo las aplicaciones APP_i ajustan su propio nivel de servicio, mientras que el administrador de recursos calcula los nuevos valores de las plataformas virtuales. El CBS calcula el tiempo de ejecución de las aplicaciones usando como datos las plataformas virtuales y los *deadlines*.

El RM que diseñaremos debe ser difuso y distribuido. Lo primero, hace que el RM pueda lidiar con cambios no lineales en las necesidades de las tareas sin perder funcionalidad. Lo segundo permite dividir la solución del problema. Así, por un lado, cada aplicación hace una regulación local de sus recursos, mientras que por otro lado, el RM global coordina la distribución de tiempo de procesamiento asignado a cada aplicación.[Aparicio-Santos et al. (2021)]

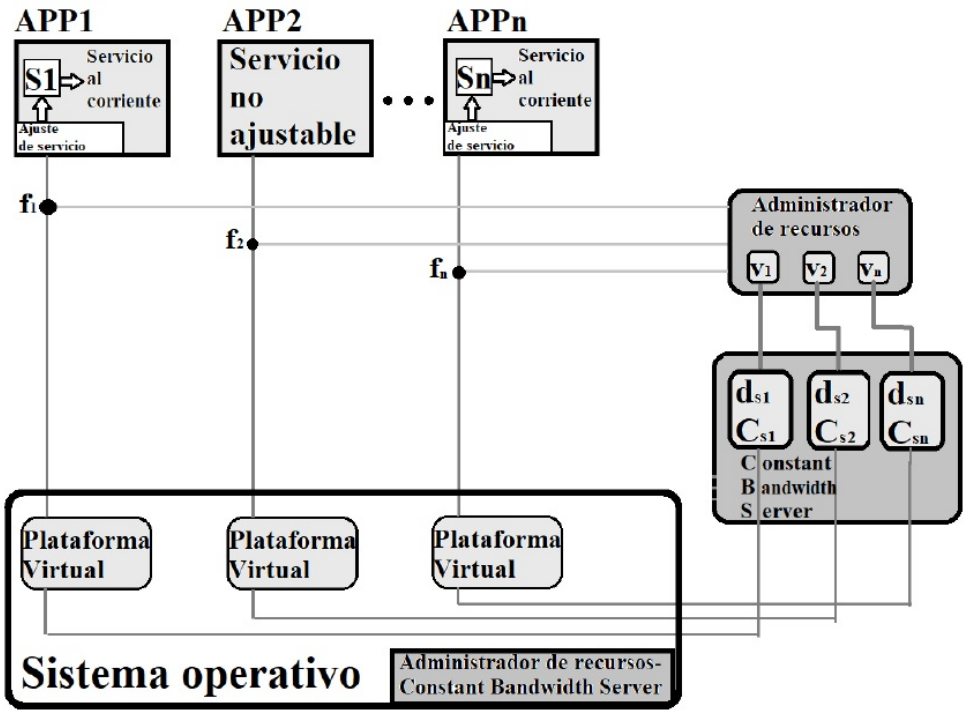


Figura 2.1: Esquema utilizado en Aparicio-Santos(2017). El manejador de recursos regula el tamaño de las plataformas virtuales, mientras el CBS se encarga de actualizar el consumo y los deadlines de las aplicaciones

2.7. Modelo Takagi-Sugeno

Para el diseño del manejador de recursos RM, se utilizará un control difuso basado en el modelo Takagi-Sugeno. El rasgo principal de un modelo Takagi-Sugeno es expresar la dinámica local de cada implicación difusa (regla) mediante un modelo lineal del sistema [Tanaka and Wang (2001)].

Se implementará un modelo Takagi-Sugeno discreto el cual esta descrito por una serie de reglas IF-THEN con la siguiente estructura, tomado de [Tanaka and Wang (2001)].

Regla modelo i:

$$IF\ z_1(t)\ is\ M_{i1}\ and\ ...\ and\ z_p(t)\ is\ M_{ip} \tag{2.2}$$

$$THEN = \begin{cases} x_i(t+1) = A_i x(t) + B_i u(t) \\ y(t) = C_i x(t) \end{cases} \quad i = 1, 2, ..., r \tag{2.3}$$

$x(t)$	Vector de estados
$A_i x(t) + B_i u(t)$	Subsistema del sistema difuso
$x_i(t+1)$	Es el vector de estados en el instante siguiente para el subsistema
$x(t+1)$	Es el vector de estados en el instante siguiente del sistema difuso
z_i	Variable premisa, pueden ser funciones de las variables de estado, perturbaciones externas y, o del tiempo.
M_{ij}	Función de membresía, asigna un valor en el intervalo de [0,1] a la variable premisa asociada z_i.
r	Número de reglas premisas
$u(t)$	Vector de entradas
$y(t)$	Vector de salida

La salida del sistema difuso es obtenida por las ecuaciones siguientes, tomadas de [Tanaka and Wang (2001)]:

$$x(t+1) = \frac{\sum_{i=1}^{r} w_i(z(t))\{A_i x(t) + B_i u(t)\}}{\sum_{i=1}^{r} w_i(z(t))} = \sum_{i=1}^{r} h_i(z(t))\{A_i x(t) + B_i u(t)\} \quad (2.4)$$

$$y(t) = \frac{\sum_{i=1}^{r} w_i(z(t)) C_i x(t)}{\sum_{i=1}^{r} w_i(z(t))} = \sum_{i=1}^{r} h_i(z(t)) C_i x(t) \quad (2.5)$$

donde:

$$z(t) = [z_1(t)\ z_2(t)\ ...\ z_p(t)] \quad (2.6)$$

$$w_i(z(t)) = \Pi_{j=1}^{p} M_{ij}(z_j(t)) \quad (2.7)$$

$$h_i(z(t)) = \frac{w_i(z(t))}{\sum_{i=1}^{r} w_i(z(t))} \quad (2.8)$$

w_i Es el peso que tiene cada variable premisa

h_i Es el peso de cada variable premisa comparado con la suma de todos los pesos de las variables premisa

Algunas propiedades que nos serán útiles son las siguientes:

$$\sum_{i=1}^{r} h_i(z(t)) = 1 \quad (2.9)$$

$$h_i(z(t)) \geq 0 \quad (2.10)$$

Para nuestro caso (2.10) será un valor entre cero y uno.

El sistema se controlará por medio de una entrada de la forma:

$$u(t) = -\sum_{j=1}^{r} h_j(z(t)) F_j x(t) \quad (2.11)$$

donde F_j es una matriz que se obtendrá en secciones posteriores.

Sustituyendo (2.11) en (2.4) y usando la propiedad de la ecuación (2.9) obtenemos el sistema en lazo cerrado:

$$x(t+1) = \sum_{i=1}^{r} \sum_{j=1}^{r} h_i(z(t)) h_j(z(t)) \{A_i - B_i F_j\} x(t) \quad (2.12)$$

La ecuación (2.12) puede ser reescrita de la siguiente manera:

$$x(t+1) = \sum_{i=1}^{r} h_i(z(t))h_i(z(t))\{A_i - B_iF_i\}x(t) + 2\sum_{i=1}^{r}\sum_{i<j} h_i(z(t))h_j(z(t))\{\frac{G_{ij}+G_{ji}}{2}\}x(t) \tag{2.13}$$

donde:

$$G_{ij} = A_i - B_iF_j \tag{2.14}$$

2.8. Estabilidad de un sistema difuso

El siguiente teorema muestra una condición suficiente para lograr la estabilidad, basado en el teorema de estabilidad de Lyapunov para tiempo discreto, teorema obtenido de [Tanaka and Wang (2001)].

Teorema 1: El punto de equilibrio de un sistema difuso (C.1) es global y asintóticamente estable si existe una matriz positiva y definida P tal que:

$$G_{ii}^T P G_{ii} - P < 0 \tag{2.15}$$

$$\left(\frac{G_{ij}+G_{ji}}{2}\right)^T P \left(\frac{G_{ij}+G_{ji}}{2}\right) - P \leq 0 \qquad i < j \leq r \quad s.t. \quad h_i \cap h_j \neq \phi \tag{2.16}$$

La ecuación (2.16) debe cumplirse para todo $i < j \leq r$ excepto cuando $h_i(z(t)) \cdot h_j(z(t)) = 0$ □

El corolario siguiente, obtenido de [Tanaka and Wang (2001)], muestra un caso especial donde el Teorema 1 se reduce a cumplir solo la condición (2.15)

Corolario 1: Cuando $B_1 = B_2 = ... = B_r$, el punto de equilibrio de un sistema difuso es global y asintótica mente estable si existe una matriz positiva P que satisfaga la ecuación (2.15)□

2.9. Desigualdades matriciales lineales LMI

El problema de diseño de control se reduce a la selección de las matrices F_i y P de manera que se cumplan las condiciones (2.15) y (2.16). Una forma de encontrar matrices F_i y P que satisfagan estas condiciones es mediante el uso de desigualdades matriciales lineales (LMI, por sus siglas en inglés). La definición siguiente y el Problema LMI se obtuvieron de [Tanaka and Wang (2001)]

Definición 1: Las desigualdades matriciales lineales LMI son de la forma

$$F(x) = F_0 + \sum_{i=1}^{m} x_i F_i > 0 \tag{2.17}$$

Donde $x^T = [x_1, x_2, ..., x_m]$ y F_i es una matriz simétrica. El símbolo de desigualdad $F(x) > 0$ significa que $F(x)$ es positiva definida.□

Problema LMI: Dado una LMI $F(x) > 0$ el objetivo es encontrar las x factibles (x^{feas}) tal que cumpla con $F(x^{feas}) > 0$ o se determine que la LMI no es posible. □

Exceptuando algunos casos especiales, las LMI no tienen soluciones analíticas. No obstante, pueden resolverse numéricamente en todos los casos mediante el uso de algoritmos en software de cálculo numérico.

2.10. Diseño de un controlador difuso estable

Para obtener estabilidad en un sistema difuso por medio de LMIs partiremos usando las ecuaciones (2.15) y (2.16). Multiplicando por $X = P^{-1}$ tanto por la derecha como por la izquierda de las desigualdades (2.15) y (2.16); y definiendo $M_i = F_i X$ se reescriben las ecuaciones como:

$$-XA_i^T - A_iX + M_i^T B_i^T + B_iM_i > 0 \tag{2.18}$$

$$-XA_i^T - A_iX - XA_j^T - A_jX + M_j^T B_i^T + B_iM_j + M_i^T B_j^T + B_jM_i \geq 0 \tag{2.19}$$

Estas desigualdades pueden ahora ser convertidas a LMIs usando el complemento de Schur. Las LMI resultantes son:

$$\begin{bmatrix} X & XA_i^T - M_i^T B_i^T \\ A_iX - B_iM_i & X \end{bmatrix} > 0 \tag{2.20}$$

$$\begin{bmatrix} X & (\frac{A_iX + A_jX - B_iM_j - B_jM_i}{2})^T \\ (\frac{A_iX + A_jX - B_iM_j - B_jM_i}{2})^T & X \end{bmatrix} \geq 0 \tag{2.21}$$

La solución de las LMI nos dan los valores de X y M_i, para obtener los valores de P y F_i se usan las siguientes ecuaciones

$$P = X^{-1} \qquad F_i = M_iX^{-1} \tag{2.22}$$

2.11. Resumen de capítulo

En este capítulo se presentaron las tres principales teorías que se utilizarán en el diseño del manejador de recursos. Estas son:

Control difuso: Se utiliza el Modelo Takagi-Sugeno, se muestran las ecuaciones que lo describen y las condiciones para lograr un control global y asintóticamente estable.

Servidor de ancho de banda constante: Se explica en qué consiste y se presenta el algoritmo que lo define.

Teoría del manejador de recursos: Nos centramos en los conceptos necesarios para su desarrollo y se introduce el modelo híbrido, el cual será el modelo a usar.

Capítulo 3

Desarrollo teórico del manejador de recursos

3.1. Introducción

En el capítulo anterior se presentó una introducción al manejador de recursos y al control difuso. En este capítulo, se expondrá el desarrollo matemático para el diseño de un manejador de recursos difuso. La mayor parte de la teoría implementada en este capítulo se obtiene de [Santos (2017)], [Aparicio-Santos et al. (2021)] y [Abeni and Buttazzo (1998)]

3.1.1. Símbolos y sus definiciones

h	número de procesador
i	número de tarea
k	número de iteración
n	número total de tareas en el procesador h
κ	número total de procesadores
r	número total de subsistemas
$\bar{q}_h$	procesador h normalizado con respecto al procesador mas rápido
U_T	máxima capacidad del sistema
$v_{h,i,k}$	plataforma virtual
$s_{h,i,k}$	nivel de servicio
$\bar{v}_{h,i,k}$	plataforma virtual normalizada
$\lambda_{i,k}$	prioridad de la aplicación
$f_{i,k}$	función de emparejamiento
q_m	procesador con mayor capacidad
$F_{h,i,k}$	función de equidad nominal

3.1.2. Normalización de variables

Se debe tener en cuenta que se utilizarán microcontroladores con diferentes velocidades de procesamiento y sistemas operativos. Además, la cantidad total de aplicaciones, la cantidad de procesadores y la cantidad de aplicaciones en cada procesador pueden cambiar en cualquier momento.

La velocidad de procesamiento de un microcontrolador se puede ver como una fracción de la velocidad del microcontrolador más potente. De manera que podemos normalizar las velocidades de los procesadores respecto al de mayor potencia.

$$\bar{q}_h = \frac{q_h}{q_m} \leq 1 \tag{3.1}$$

La suma de todas las velocidades de procesamiento normalizadas da la máxima capacidad del sistema

$$\sum_{h=1}^{\kappa} \bar{q}_h = U_T \tag{3.2}$$

La plataforma virtual se definió como un procesador virtual con una fracción de la potencia del procesador real, por tanto, podemos normalizar a la plataforma virtual respecto la capacidad del procesador real.

$$\bar{v}_{h,i,k} = \frac{v_{h,i,k}}{q_h} \leq 1 \tag{3.3}$$

De esta manera la suma de las plataformas virtuales que se ejecuten en un mismo procesador no debe ser mayor que la potencia disponible en el procesador.

$$\sum_{i=1}^{n} v_{h,i,k} \leq \bar{q}_h \tag{3.4}$$

3.1.3. Función de emparejamiento

Las aplicaciones solicitan recursos del procesador a través del nivel de servicio, cada aplicación calcula su propio nivel de servicio.

Para esta tesis el nivel de servicio $s_{h,i,k}$ lo podemos definir como la potencia de procesamiento que requiere la tarea i para un rendimiento óptimo.

La función (3.5) se conoce como la función de emparejamiento, la cual indica al RM si los recursos asignados a la aplicación son suficientes, escasos o excesivos. Función obtenida de [Chasparis et al. (2016)]

$$f_{h,i,k} = \beta_i \frac{\bar{v}_{h,i,k}}{s_{h,i,k}} - 1 \tag{3.5}$$

Donde β_i es una constante positiva.

Si $\lvert f_i \rvert \leq \delta$	la tarea i tiene la cantidad correcta de recursos
Si $f_i < -\delta$	la aplicación i no tiene suficientes recursos
Si $f_i > \delta$	la aplicación i tiene recursos más que suficientes

En esta tesis se tomará $\delta = 0$ de esta forma sólo se tendrán recursos suficientes e

insuficientes

3.1.4. Función de equidad nominal

Los recursos del procesador deben ser distribuidos equitativamente entre las aplicaciones, de manera que ninguna aplicación acapare los recursos ni se quede sin recursos asignados.

Además de la cantidad de recursos que solicitan las aplicaciones para su óptimo funcionamiento, también se puede asignar a cada aplicación un nivel de prioridad $\lambda_i \in (0, 1]$. De esta manera, una aplicación con mayor prioridad tenderá a recibir más recursos que una con menor prioridad.

La función (3.6) se llama función de equidad nominal y es la encargada de repartir los recursos entre las aplicaciones. Función obtenida de [Chasparis et al. (2016)]

$$F_{h,i,k} = -(\bar{q_h} - \bar{v}_{h,i,k})\lambda_{h,i,k}[f_{h,i,k}]_- + \bar{v}_{h,i,k} \sum_{j \neq i} \lambda_{h,j,k}[f_{h,j,k}]_- \tag{3.6}$$

La notación $[x]_-$ para cualquier $x \in \mathbb{R}$ es:

$$[x]_- = \begin{cases} x & x \leq 0 \\ 0 & x > 0 \end{cases} \tag{3.7}$$

La función (3.6) captura la deficiencia en recursos de la aplicación i comparada con el resto de las aplicaciones.

Si la función (3.6) es igual a cero, la distribución de recursos es la adecuada y por tanto no hay ningún cambio en la cantidad de recursos asignados a la aplicación i. Si la función es negativa, indica que la repartición no es equitativa, ya que la aplicación i tiene más recursos de los que le corresponderían, por lo tanto, cederá recursos a las demás aplicaciones. Si la función es positiva, indica que se le cederán recursos a la aplicación i por parte de las demás aplicaciones.

La función (3.6) muestra que la distribución de recursos es equitativa si se cumple cualquiera de las siguientes ecuaciones:

1. $[f_{h,i,k}]_- = 0 \quad \forall\, i$

2. $[f_{h,i,k}]_- < 0 \quad \forall\, i$ además se debe cumplir que la proporción de recursos $v_{h,i,k}/(\bar{q_h} - v_{h,i,k})$ coincida con la proporción entre las funciones de emparejamiento multiplicadas por $\lambda_{h,i,k}$, como se muestra en la ecuación (3.8). En otras palabras, ya que en la ecuación (3.6) $F_i = 0$ cuando es equitativa, se puede reescribir la ecuación de la siguiente forma:

$$\frac{\lambda_{h,i,k}[f_{h,i,k}]_-}{\sum_{j \neq i} \lambda_{h,j,k}[f_{h,j,k}]_-} = \frac{\bar{v}_{h,i,k}}{(\bar{q}_h - \bar{v}_{h,i,k})} \tag{3.8}$$

3.2. Fuzzificación del modelo

Se expondrá el diseño del RM difuso considerando el caso en el que se encuentran tres aplicaciones en el procesador.

En cada iteración se actualiza los valores de las plataformas virtuales y el nivel de servicio a través de las siguientes dos ecuaciones:

$$\bar{v}_{h,i,k+1} = \bar{v}_{h,i,k} + \epsilon F_{h,i,k} \tag{3.9}$$

$$s_{i,k+1} = s_{i,k} + \epsilon f_{i,k} \tag{3.10}$$

Donde $\epsilon = \alpha/n$, siendo α un valor pequeño. En el caso del procesador de mayor capacidad, ϵ indica el incremento o decremento máximo permitido para las plataformas virtuales y el nivel de servicio en una iteración. El valor de ϵ depende del número de aplicaciones y se elige de manera que garantice la convergencia.

Para el control difuso, se utilizarán como estados del sistema las plataformas virtuales y las prioridades de las aplicaciones λ_i.

La actualización de las plataformas virtuales depende únicamente de la ecuación (3.9). Por otro lado, podemos colocar una entrada de referencia en el control, a la cual λ_i deberá tender, λ_i se actualizara con la entrada del control. Por tanto la matriz B

es de la forma:

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

La ecuación de equidad nominal para múltiples procesadores es:

$$F_{h,i,k} = -(\bar{q}_h - \bar{v}_{h,i,k})\lambda_{i,k}[f_{i,k}]_- + \bar{v}_{h,i,k}\sum_{j\neq i}\lambda_{j,k}[f_{j,k}]_- \tag{3.12}$$

Sustituyendo la ecuación (3.12) en la ecuación (3.9), obtenemos la siguiente expresión para la actualización de las plataformas virtuales:

$$\bar{v}_{h,1,k+1} = \bar{v}_{h,1,k} + \epsilon\{-(\bar{q}_h - \bar{v}_{h,1,k})\lambda_{1,k}[f_{1,k}]_- + \bar{v}_{h,1,k}(\lambda_{2,k}[f_{2,k}]_- + \lambda_{3,k}[f_{3,k}]_-)\} \tag{3.13}$$

$$\bar{v}_{h,2,k+1} = \bar{v}_{h,2,k} + \epsilon\{-(\bar{q}_h - \bar{v}_{h,2,k})\lambda_{2,k}[f_{2,k}]_- + \bar{v}_{h,2,k}(\lambda_{1,k}[f_{1,k}]_- + \lambda_{3,k}[f_{3,k}]_-)\} \tag{3.14}$$

$$\bar{v}_{h,3,k+1} = \bar{v}_{h,3,k} + \epsilon\{-(\bar{q}_h - \bar{v}_{h,3,k})\lambda_{3,k}[f_{3,k}]_- + \bar{v}_{h,3,k}(\lambda_{1,k}[f_{1,k}]_- + \lambda_{2,k}[f_{2,k}]_-)\} \tag{3.15}$$

$$\lambda_{1,k+1} = \lambda_{1,k} + u_1 \tag{3.16}$$

$$\lambda_{2,k+1} = \lambda_{2,k} + u_2 \tag{3.17}$$

$$\lambda_{3,k+1} = \lambda_{3,k} + u_3 \tag{3.18}$$

De las ecuaciones anteriores, se puede observar que si el término completo entre corchetes es igual a cero y la entrada u es cero, se alcanza el punto de equilibrio. Dado que estamos en un sistema de tiempo discreto, el punto de equilibrio debe cumplir la siguiente condición:

$$\begin{bmatrix} \bar{v}_{k+1} \\ \bar{\lambda}_{k+1} \end{bmatrix} = \begin{bmatrix} \bar{v}_k \\ \bar{\lambda}_k \end{bmatrix} \tag{3.19}$$

Donde $\bar{v}$ es un vector formado por las plataformas virtuales y $\bar{\lambda}$ es un vector

formado por las prioridades de las aplicaciones.

Lo anterior es lo mismo que decir que $F_{h,i,k} = 0$ que es lo que se propuso para obtener una distribución equitativa

Reescribiendo las ecuaciones de (3.13) a (3.18)

$$\bar{v}_{h,1,k+1} = \bar{v}_{h,1,k}\{1 + \epsilon(\lambda_{1,k}[f_{1,k}]_- + \lambda_{2,k}[f_{2,k}]_- + \lambda_{3,k}[f_{3,k}]_-)\} - \epsilon\bar{q}_h\lambda_{1,k}[f_{1,k}]_- \tag{3.20}$$

$$\bar{v}_{h,2,k+1} = \bar{v}_{h,2,k}\{1 + \epsilon(\lambda_{1,k}[f_{1,k}]_- + \lambda_{2,k}[f_{2,k}]_- + \lambda_{3,k}[f_{3,k}]_-)\} - \epsilon\bar{q}_h\lambda_{2,k}[f_{2,k}]_- \tag{3.21}$$

$$\bar{v}_{h,3,k+1} = \bar{v}_{h,3,k}\{1 + \epsilon(\lambda_{1,k}[f_{1,k}]_- + \lambda_{2,k}[f_{2,k}]_- + \lambda_{3,k}[f_{3,k}]_-)\} - \epsilon\bar{q}_h\lambda_{3,k}[f_{3,k}]_- \tag{3.22}$$

$$\lambda_{1,k+1} = \lambda_{1,k} + u_1 \tag{3.23}$$

$$\lambda_{2,k+1} = \lambda_{2,k} + u_2 \tag{3.24}$$

$$\lambda_{3,k+1} = \lambda_{3,k} + u_3 \tag{3.25}$$

De esta forma se tienen 6 variables premisa:

$$z_1 = 1 + \epsilon(\lambda_{1,k}[f_{1,k}]_- + \lambda_{2,k}[f_{2,k}]_- + \lambda_{3,k}[f_{3,k}]_-) \tag{3.26}$$

$$z_2 = 1 + \epsilon(\lambda_{1,k}[f_{1,k}]_- + \lambda_{2,k}[f_{2,k}]_- + \lambda_{3,k}[f_{3,k}]_-) \tag{3.27}$$

$$z_3 = 1 + \epsilon(\lambda_{1,k}[f_{1,k}]_- + \lambda_{2,k}[f_{2,k}]_- + \lambda_{3,k}[f_{3,k}]_-) \tag{3.28}$$

$$z_4 = -\epsilon\bar{q}_h\lambda_{1,k}[f_{1,k}]_- \tag{3.29}$$

$$z_5 = -\epsilon\bar{q}_h\lambda_{1,k}[f_{1,k}]_- \tag{3.30}$$

$$z_6 = -\epsilon\bar{q}_h\lambda_{1,k}[f_{1,k}]_- \tag{3.31}$$

Se pude ver que el punto de equilibrio implica que las variables premisa sean:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.32}$$

Se escribe el sistema formado por las ecuaciones (3.20) a (3.25) en forma de espacio de estados.

$$\begin{bmatrix} \bar{v}_{h,1,k+1} \\ \bar{v}_{h,2,k+1} \\ \bar{v}_{h,3,k+1} \\ \lambda_{1,k+1} \\ \lambda_{2,k+1} \\ \lambda_{3,k+1} \end{bmatrix} = \begin{bmatrix} z_1 & 0 & 0 & z_4 & 0 & 0 \\ 0 & z_2 & 0 & 0 & z_5 & 0 \\ 0 & 0 & z_3 & 0 & 0 & z_6 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{v}_{h,1,k} \\ \bar{v}_{h,2,k} \\ \bar{v}_{h,3,k} \\ \lambda_{1,k} \\ \lambda_{2,k} \\ \lambda_{3,k} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{3.33}$$

Se calculan los máximos y los mínimos de las variables premisas:

$$z_{1min} = 1 - n\epsilon \qquad z_{1max} = 1 \tag{3.34}$$

$$z_{2min} = 1 - n\epsilon \qquad z_{2max} = 1 \tag{3.35}$$

$$z_{3min} = 1 - n\epsilon \qquad z_{3max} = 1 \tag{3.36}$$

$$z_{4min} = 0 \qquad z_{4max} = \epsilon\bar{q}_h \tag{3.37}$$

$$z_{5min} = 0 \qquad z_{5max} = \epsilon\bar{q}_h \tag{3.38}$$

$$z_{6min} = 0 \qquad z_{6max} = \epsilon\bar{q}_h \tag{3.39}$$

Donde n es el número de aplicaciones, en este caso n=3 y $\epsilon = \frac{a}{n}$ donde $0 < a \leq 1$.

Sabiendo los valores máximo y mínimo de las funciones premisa, podemos definir funciones de membresía de tipo triangular que representen el valor de membresía de cada variable premisa, las funciones de membresía deben cumplir las siguientes ecuaciones:

$$z_j = M_{1,j} z_{jmax} + M_{2,j} z_{jmin} \tag{3.40}$$

$$1 = M_{1,j} + M_{2,j} \tag{3.41}$$

donde $j = 1, 2, 3, ..., 6$

Por tanto se tiene que las variables premisa son:

$$M_{1,j}(z_j) = 1 - \frac{z_j - z_{jmax}}{z_{jmin} - z_{jmax}} \tag{3.42}$$

$$M_{2,j}(z_j) = \frac{z_j - z_{jmax}}{z_{jmin} - z_{jmax}} \tag{3.43}$$

Asignamos las reglas IF-THEN, debe de generarse tantas reglas como posibles combinaciones de las funciones de membresía se puedan hacer. En el caso de tres plataformas virtuales, existen $2^6 = 64$ combinaciones posibles, con la siguiente estructura:

IF	THEN $A_i + Bu$
$M_{11}(z_1)M_{12}(z_2)M_{13}(z_3)M_{14}(z_4)M_{15}(z_5)M_{16}(z_6)$	$A_1x + Bu$
$M_{21}(z_1)M_{12}(z_2)M_{13}(z_3)M_{14}(z_4)M_{15}(z_5)M_{16}(z_6)$	$A_2x + Bu$
$\vdots$	$\vdots$
$M_{21}(z_1)M_{22}(z_2)M_{23}(z_3)M_{24}(z_4)M_{25}(z_5)M_{26}(z_6)$	$A_{64}x + Bu$

Cada regla i tiene asociado un centroide h_i que está definido por la ecuación (3.45):

$$w_i(z(t)) = \Pi_{j=1}^{p} M_{ij}(z_j(t)) \tag{3.44}$$

$$h_i(z(t)) = \frac{w_i(z(t))}{\sum_{i=1}^{r} w_i(z(t))} \tag{3.45}$$

donde p es la cantidad de estados.

La defuzzificación se logra con la ecuación (3.46):

$$x(t+1) = \sum_{i=1}^{r}\sum_{j=1}^{r} h_i(z(t))h_j(z(t))\{A_i - B_iF_j\}x(t) \tag{3.46}$$

$$y(t) = \sum_{i=1}^{r} h_i(z(t))C_ix(t) \tag{3.47}$$

3.3. Obtención de matrices F_i y P

Para garantizar que el sistema difuso sea global y asintóticamente estable, dado que se eligió que $B_1 = B_2 = ... = B_r$, se debe encontrar una matriz P positiva que satisfaga todas las desigualdades (2.15), según el Corolario 1.

Para encontrar las matrices F_i y la matriz P, se utilizan algoritmos computacionales. En el caso de esta tesis, se emplearon los LMI Solvers de MATLAB para resolver las desigualdades matriciales lineales.

En nuestro caso, las aplicaciones pueden cambiar de un dispositivo a otro en cada iteración, así como pueden agregarse nuevos dispositivos a la red o desconectarse de ella. Esto implica que en cada iteración se deben recalcular las soluciones de las desigualdades matriciales para cada dispositivo que modifique el número de aplicaciones que contiene, así como para cada dispositivo que se agregue a la red.

En el Apéndice 1 se demuestra que las matrices obtenidas mediante el uso de LMI Solvers para el dispositivo de mayor capacidad de procesamiento se pueden utilizar en los demás dispositivos.

Además, es posible obtener las matrices F_i y P para cualquier cantidad de aplicaciones que se ejecuten en un dispositivo, utilizando únicamente las matrices F_i y P obtenidas para una sola aplicación en el dispositivo de mayor potencia.

3.4. Aplicación del CBS en el manejador de recursos

La ecuación 2.1 indica que el uso de procesamiento no puede ser mayor al total que proporciona el procesador. Anteriormente, se definió una plataforma virtual como un procesador virtual que tiene una capacidad menor o igual al procesador en el que se está ejecutando. Por lo tanto, la suma de las plataformas virtuales asignadas al procesador debe ser proporcional al valor de U_s.

$$K \sum_{i}^{n} v_{h,i,k} = U_s = \frac{Q_{h,k}}{T} \tag{3.48}$$

La suma de las plataformas virtuales en un procesador dará siempre uno, inde-

pendientemente de la capacidad del procesador.

En el Apéndice 1, concluimos que cada procesador puede ser considerado como un sistema independiente en el cual no se utiliza su capacidad de procesamiento para el control difuso.

Por lo que la capacidad de procesamiento influye únicamente en dos aspectos: En el cambio de una aplicación de un dispositivo a otro y en la asignación del nivel de servicio a las aplicaciones.

A cada aplicación se le asigna un tiempo de ejecución por período, denominado presupuesto $c_{h,i,k}$. Podemos calcular este presupuesto utilizando la siguiente ecuación:

$$c_{h,i,k} = KTv_{h,i,k} \tag{3.49}$$

Debido a las redistribuciones de las aplicaciones y al uso completo del presupuesto asignado a cada una, el presupuesto se recarga siempre al valor máximo Q en cada periodo de tiempo T. La suma de los presupuestos asignados a las aplicaciones es igual al presupuesto máximo, como se muestra en la siguiente ecuación:

$$K\sum_{i}^{n} v_{h,i,k} = \frac{c_{h,1,k}}{T} + \frac{c_{h,2,k}}{T} + ... + \frac{c_{h.n,k}}{T} = \frac{Q_{h,k}}{T} \tag{3.50}$$

El valor de K se obtiene a partir de la desigualdad 2.1

$$U_s = K\sum_{i}^{n} v_{h,i,k} = K \leq 1 - U_p \tag{3.51}$$

$$\tag{3.52}$$

Podemos observar que para obtener el presupuesto de una aplicación, es necesario seleccionar un período de tiempo de ejecución. Además, la actualización del *deadline* para las aplicaciones de tipo *soft* se realizará teniendo en cuenta este período de tiempo, que será el mismo en todos los procesadores. Es importante destacar que una aplicación puede ejecutarse más de una vez antes de que se cumpla el *deadline*, dependiendo de cómo se asignen las ejecuciones mediante el CBS.

3.5. Cambio en la distribución de las aplicaciones tipo soft

El momento más crítico durante la ejecución es cuando se realiza una redistribución de las aplicaciones en los procesadores. En esta etapa, pueden surgir problemas como la pérdida de *deadlines* o la ejecución duplicada de una misma aplicación en dos procesadores diferentes. Se realizará un análisis de las posibles situaciones y se propondrán estrategias para mitigar los problemas asociados.

Una limitación importante es la necesidad de ejecutar las aplicaciones con los valores de las plataformas virtuales correspondientes a un mismo instante de tiempo. Esto es necesario para cumplir con la ecuación 2.1. Se puede decir que las aplicaciones se ejecutan en *paquetes* durante cada periodo de tiempo. Llamaremos *planificación* al proceso de determinar qué aplicaciones se van a ejecutar y durante cuánto tiempo.

Cuando se realiza una nueva distribución de las aplicaciones, se enfrenta la opción de interrumpir la ejecución de las aplicaciones faltantes de un paquete para iniciar la ejecución de las nuevas aplicaciones, o bien, esperar a que el paquete actual termine su ejecución antes de iniciar con las nuevas aplicaciones.

Sin embargo, detener la ejecución de las aplicaciones puede ocasionar la pérdida de sus respectivos *deadlines*. Por lo tanto, se opta por esperar hasta que todas las aplicaciones en ejecución hayan finalizado antes de iniciar con las nuevas aplicaciones.

Todas las aplicaciones tienen sus *deadlines* establecidos en múltiplos del periodo de tiempo. En algunos casos, un paquete de aplicaciones puede estar compuesto por aplicaciones con *deadlines* distintos, es decir, con *deadlines* que se encuentran en distintos múltiplos enteros del periodo T. En este caso, se debe ejecutar únicamente las aplicaciones con los *deadlines* más cercanos a cumplirse. La diferencia en los *deadline* de las aplicaciones solo aparece cuando se realiza una nueva distribución de las aplicaciones.

Otro problema surge cuando se ejecuta la misma aplicación en dos procesadores al mismo tiempo. Esto ocurre cuando se crea una nueva distribución de las aplicaciones y una aplicación aún no ha terminado de ejecutarse, pero una nueva planificación

indica que esa aplicación debe ser la primera en ejecutarse en otro procesador. La razón por la que la aplicación no ha terminado de ejecutarse puede ser que está adelantando su ejecución con respecto a sus *deadlines*, o que las aplicaciones en otros procesadores han terminado de ejecutarse mucho antes. Normalmente, esto se puede evitar al indicar que otra aplicación se ejecute primero, el orden de ejecución entre las aplicaciones *soft* no es importante. Si es la única aplicación ejecutándose en el nuevo procesador la única opción es esperar a que la aplicación termine su anterior ejecución.

Se puede resumir todo lo anterior en los siguientes puntos:

- Las aplicaciones tipo *soft* incrementan su *deadline* en múltiplos del periodo T
- Las aplicaciones tipo *soft* deben ejecutarse utilizando el valor de sus plataformas virtuales en el mismo instante de tiempo
- Si tienen distinto *deadline* se pueden ejecutar únicamente las aplicaciones con el *deadline* más cercano a cumplirse
- Las aplicaciones deben de activar una bandera para indicar que están ejecutándose o en espera de ejecutarse para así evitar una ejecución en paralelo
- Si una aplicación debe ejecutarse pero actualmente está en proceso de ejecución en otro procesador, se puede optar por ejecutar otra aplicación *soft* para dar tiempo a que la primera aplicación finalice su ejecución. En el peor de los casos, si no hay más aplicaciones *soft* disponibles para ejecutar, el procesador puede quedar en espera hasta que la aplicación termine su ejecución.
- El tiempo utilizado en la distribución de aplicaciones se puede añadir como tiempo adicional de ejecución para las aplicaciones *hard*, con el fin de evitar incumplimientos de *deadlines*.

Debido a lo expuesto en esta sección existen algunas limitaciones que se tiene al usar el algoritmo del CBS:

- Como la aplicación gasta todo su presupuesto excepto cuando termina su tiempo de computo, y al terminar el tiempo de cómputo se realiza una nueva distribución de las aplicaciones, no se tiene el caso donde se puede usar el presupuesto sobrante en una nueva aplicación

- Los incrementos de presupuesto en el algoritmo del CBS se dan en incrementos enteros, en nuestro caso el valor del incremento depende de la plataforma virtual por lo que puede tomar cualquier valor real positivo

- Cada conjunto de aplicaciones inicia con su presupuesto en cero por lo que tanto en la primera planificación como en posteriores el presupuesto se llena al máximo posible

3.6. Resumen de capítulo

El capítulo se centra principalmente en la combinación de la teoría del manejador de recursos y el control difuso para lograr el diseño de un manejador de recursos que, por el momento, es para un solo dispositivo.

Se analiza el comportamiento que se tendrá del CBS debido a las limitaciones que se presentan con el uso de las plataformas virtuales, además de mencionar los posibles problemas que ocurren durante la distribución de las aplicaciones.

Capítulo 4

Diseño del manejador de recursos

4.1. Introducción

En este capítulo se ampliará el análisis teórico con los detalles técnicos que han surgido durante el desarrollo. Se desarrollan métodos para lograr una mejor implementación de la teoría y para reducir los tiempos de cálculo del control.

4.2. Consideraciones tomadas respecto la teoría actual para el proyecto

En la ecuación de equidad nominal (3.6) se usa la variable $\bar{q}$ la cual indica la capacidad del procesador respecto al procesador de mayor capacidad. Esto sirve para que en un sistema con múltiples procesadores las variables de estado estén normalizadas y puedan pasar de un procesador a otro directamente.

Si bien la teoría de los últimos artículos que se han publicado sobre el tema (Aparicio-Santos et al. (2021) y Aparicio-Santos and Benítez-Pérez (2021)) se han realizado con esta variable, en el desarrollo del programa se encontraron varios problemas que afrontar, tales como:

- **Gran cantidad de matrices en memoria:** Es necesario calcular 2^{2n} matrices para cada procesador en la red, donde n es el número de aplicaciones.

- **Cálculo de matrices durante la ejecución:** Si llegara a entrar a la red un nuevo procesador cuya capacidad de procesamiento sea distinta de los que ya estaban conectados, se tiene que dedicar tiempo de procesamiento en calcular las matrices que usará. Además, si se necesita calcular matrices para un número mayor de aplicaciones de las previstas inicialmente, se deben realizar cálculos para todos los procesadores, en contraste con calcular solo para un procesador como se mostrará más adelante.

- **Lentitud del programa:** Debido al uso de $\bar{q}$, la forma más eficiente de construir el código es mediante el uso de variables simbólicas. Sin embargo, la ejecución del código con estas variables es considerablemente más lenta que sin ellas. Otras soluciones para abordar este problema conllevan inconvenientes en términos de tiempo de ejecución debido a otras circunstancias, costos de memoria o pérdida de precisión en la solución.

Por estos motivos, se utilizará la ecuación (3.6) con $\bar{q} = 1$, tal como se ha tratado en la teoría para un solo procesador. Bajo esta ecuación, se analiza cada procesador de forma individual, por lo que el paso de las plataformas virtuales de un procesador a otro ya no es tan directo.

Si se requiere cambiar la plataforma virtual de un procesador a otro, se usa la ecuación:

$$\frac{v_{h1}\bar{q}_{h1}}{\bar{q}_{h2}} = v_{h2} \tag{4.1}$$

La justificación de esta ecuación radica en que la plataforma virtual representa una fracción de la capacidad de procesamiento del procesador en el que se ejecuta. A su vez, la capacidad de procesamiento de ese procesador representa una fracción de la capacidad de procesamiento del procesador de mayor capacidad. El valor de la plataforma virtual respecto al procesador de mayor capacidad es:

$$v_{hm} = v_{h1}\bar{q}_{h1} \tag{4.2}$$

donde hm representa al procesador de mayor capacidad

Si la plataforma virtual se cambia a un procesador distinto, basta con igualar las ecuaciones (4.2) correspondientes a los procesadores involucrados.

$$v_{hm} = v_{h1}\bar{q}_{h1} = v_{h2}\bar{q}_{h2} \tag{4.3}$$

Usando estos cambios, las ventajas que provén son:

- **Cálculo previo de las matrices:** No importa si se añaden nuevos procesadores a la red, se utilizará un único conjunto de matrices que se calcula previamente y se carga en el programa al inicio de la ejecución. La única situación en la que se calcularían nuevas matrices es si se permitiera ejecutar más aplicaciones en un procesador de las que se planearon inicialmente (sin embargo, este caso no se analiza en esta tesis).
- **Menor uso de memoria:** Debido a que sólo se usa un conjunto de matrices en lugar de un conjunto para cada procesador.
- **Mejor tiempo de ejecución:** Debido a que ya no se utilizan variables simbólicas y el procesamiento se centrará principalmente en la ejecución del RM.

4.3. Cálculo de matrices iniciales

Podemos diseñar algunas matrices de antemano, y luego cargarlas en el programa durante la inicialización. Esto tiene como objetivo reducir la cantidad de datos a procesar y, por lo tanto, disminuir el tiempo de procesamiento.

4.3.1. Matrices A

Las primeras matrices que obtendremos son las matrices A de los subsistemas $x_i(t+1) = A_i x + B_i u$.

Se obtiene una primera matriz A con todas las variables premisa en sus valores mínimos. A continuación se muestra un ejemplo para el caso de tres aplicaciones:

$$A = \begin{bmatrix} a - n\epsilon & 0 & 0 & 0 & 0 & 0 \\ 0 & a - n\epsilon & 0 & 0 & 0 & 0 \\ 0 & 0 & a - n\epsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

Las demás matrices se obtienen por un algoritmo recursivo, usando como entrada la primera matriz A obtenida.

El tamaño de la matriz incrementa con el número de aplicaciones en el procesador. Es necesario tener un conjunto de matrices A para cada número de aplicaciones que el procesador pueda alojar.

En las variables premisas con la forma $a - n\epsilon$, se ha modificado el valor de $a = 1$ por un valor cercano a uno. Esto se debe a que en $a = 1$, la desigualdad matricial puede convertirse en una igualdad según lo indica el algoritmo que calcula las LMI. Los puntos eliminados son:

$$(z_i, z_j) = \begin{cases} z_i = 1 \\ z_j \in 0, \epsilon \end{cases} \quad \forall i \in [1, n], j \in [n+1, 2n] \tag{4.5}$$

De todos ellos solo hay un punto de equilibrio que se da cuando $z_i = 1$ y $z_j = 0$, como se vio en el capítulo anterior. Esto no afecta la convergencia ya que al colocar un número $a \approx 1$ se interpreta que conforme pase el tiempo nos acercamos al punto de equilibrio, sin embargo no llegamos a él.

El valor de $\epsilon = 0.1$ se puede utilizar para un número de aplicaciones menor a diez, con el fin de evitar que la resta $a - n\epsilon$ genere valores negativos que carezcan de significado en el análisis. Además, este valor de ϵ mantiene un cambio máximo constante en las variables de estado.

Sin embargo, para procesadores con un número de aplicaciones $n \geq 10$, se requiere disminuir el valor de ϵ. Esto tiene como consecuencia una reducción en el cambio

máximo de las variables de estado.

4.3.2. Matriz Az y Bz

Las ecuaciones (3.42) y (3.43) se pueden reescribir de las siguiente manera:

$$M_{1,j}(z_j) = \frac{-1}{z_{jmin} - z_{jmax}} z_j + \frac{z_{jmax}}{z_{jmin} - z_{jmax}} + 1 = a_{zj} z_j + b_{zj} \tag{4.6}$$

$$M_{2,j}(z_j) = \frac{1}{z_{jmin} - z_{jmax}} z_j + \frac{-z_{jmax}}{z_{jmin} - z_{jmax}} = a_{zj} z_j + b_{zj} \tag{4.7}$$

Podemos agrupar a todos los escalares a_{zj} de un subsistema en un vector, lo mismo para los escalares b_{zj}. Dependiendo del subsistema se usarán los escalares de la ecuación (4.6) o de la ecuación (4.7)

$$A_z = \begin{bmatrix} a_{z1} \\ a_{z2} \\ \vdots \\ a_{z2n} \end{bmatrix} \qquad B_z = \begin{bmatrix} b_{z1} \\ b_{z2} \\ \vdots \\ b_{z2n} \end{bmatrix} \tag{4.8}$$

Podemos colocar los vectores A_z de todos los subsistemas en una sola matriz, lo mismo para B_z

$$\bar{A}_z = \begin{bmatrix} A_{z1} & A_{z2} & \dots & A_{z2n} \end{bmatrix} \qquad \bar{B}_z = \begin{bmatrix} B_{z1} & B_{z2} & \dots & B_{z2n} \end{bmatrix} \tag{4.9}$$

De esta manera obtenemos la ecuación:

$$W = \bar{A}_z . * \bar{z} + \bar{B}_z \tag{4.10}$$

donde:

- W Matriz de pesos de todos los subsistemas
- $\bar{z}$ vector de variables de estado
- $.*$ Operador que multiplica al vector $\bar{z}$ con todas las columnas de la matriz $\bar{A}_z$

Si multiplicamos los elementos de cada columna de la matriz W entre sí, obtendremos un vector que contiene todos los centroides h de los subsistemas.

$$\bar{h} = \begin{bmatrix} h_1 & h_2 & \dots & h_{2^{2n}} \end{bmatrix} \tag{4.11}$$

Si se multiplica el vector $\bar{h}$ por su transpuesta, se tienen todas las combinaciones de h que se utilizaran en 2.12.

De esta manera obtenemos las matrices $\bar{A}_z$ y $\bar{B}_z$ las cuales se pueden cargar de memoria al iniciar el programa. Esto agiliza el programa ya que se omiten varios cálculos durante la ejecución, y los cálculos que aún se deben realizar son simples multiplicaciones y sumas de matrices.

4.3.3. Matrices LMI

El cálculo de las LMI es un proceso que requiere bastante tiempo de procesamiento y memoria, aumenta el uso de estos recursos cuando aumenta el número de aplicaciones que se ejecutaran en un procesador.

Las LMI entregan una matriz P positiva definida que cumple las desigualdades en (2.15) y además entrega un conjunto de matrices F las cuales se usan en las matrices $G_{i,j}$ en (2.14).

Las matrices $G_{i,j}$ obtenidas se pueden guardar en memoria y ser cargadas una vez se inicia el programa.

Como se demuestra en el Apéndice 1, es posible calcular las LMI para un caso con un número reducido de aplicaciones y, a partir de ese resultado, construir las matrices correspondientes para un mayor número de aplicaciones.

En este trabajo se calculan las matrices para el caso de dos aplicaciones y a partir de ellas se obtiene las matrices para los demás casos.

El método consiste en aprovechar la estructura de las matrices $G_{i,j}$ y de la matriz P. Los elementos en las filas k y $k+n$ de estas matrices son independientes de las

demás filas cuando se utilizan en la ecuación 2.15. Por lo tanto, si tenemos otra matriz $G_{i,j}$ de un tamaño diferente pero con los mismos elementos no nulos que la fila k, podemos asignar los elementos no nulos de la fila $k+n$ a la nueva matriz.

Como ejemplo tenemos las matrices G_1 y G_2:

$$G_1 = \begin{bmatrix} a_1 & 0 & b_1 & 0 \\ 0 & a_2 & 0 & b_2 \\ c_1 & 0 & d_1 & 0 \\ 0 & c_2 & 0 & d_2 \end{bmatrix} \tag{4.12}$$

$$G_2 = \begin{bmatrix} a_3 & 0 & 0 & b_3 & 0 & 0 \\ 0 & a_1 & 0 & 0 & b_1 & 0 \\ 0 & 0 & a_4 & 0 & 0 & b_4 \\ c_3 & 0 & 0 & d_3 & 0 & 0 \\ 0 & x & 0 & 0 & y & 0 \\ 0 & 0 & c_4 & 0 & 0 & d_3 \end{bmatrix} \tag{4.13}$$

Como el renglón 2 en G_2 tiene los mismos elementos no nulos que el renglón uno de G_1 se puede usar los elementos no nulos del renglón $k+n = 3$ de G_1 para completar la matriz G_2:

$$x = c_1 \qquad y = d_1 \tag{4.14}$$

4.4. Optimización de tiempos de ejecución

De la ecuación (2.12), se puede interpretar las sumatorias como ciclos (for), donde el incremento del número de iteraciones depende del número de aplicaciones que se ejecutan en un procesador:

$$m = 2^{4n} \tag{4.15}$$

Donde m es el número de iteraciones y n es el número de aplicaciones. Esto implica

un incremento exponencial que retrasa considerablemente los tiempos de obtención de las plataformas virtuales.

Para solucionar esto se usarán dos propuestas: El uso de programación en paralelo y el truncamiento de los valores de h.

El uso de programación en paralelo por si solo no es capaz de obtener tiempos de ejecución adecuados. Se busca que las plataformas virtuales se obtengan en un lapso de tiempo de alrededor de los cientos de milisegundos para poder asignar tiempos de ejecución a las aplicaciones del orden de los segundos.

En la siguiente sección, se presenta un método denominado "truncamiento de los valores de h"que tiene como objetivo reducir el tiempo de procesamiento. Sin embargo, esta estrategia tiene la desventaja de que el valor de la plataforma virtual será una aproximación. El truncamiento de los valores de h nos permite enfocarnos en las matrices que más contribuyen a las sumatorias en la ecuación (2.12).

Como comparación se realizo la ejecución del código con cinco aplicaciones en el procesador para generar una gráfica, obteniendo cien veces los valores de las plataformas virtuales. El tiempo que tardo el programa original es de nueve minutos Figura(4.1) mientras que para las mismas condiciones usando el truncamiento se obtuvo un tiempo de 49[s] Figura (4.2)

Los tiempos en ambos casos se muestran en la Figura(4.3) y en la Figura(4.4)

Además de la reducción en los tiempos de ejecución, se observa que a medida que las variables de estado se acercan al punto de equilibrio, el programa se ejecuta más rápidamente. Esto se debe a que el punto de equilibrio se alcanza cuando el vector de valores característicos es el siguiente:

$$\begin{bmatrix} z_1 \\ \vdots \\ z_n \\ z_{n+1} \\ \vdots \\ z_{2n} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.16}$$

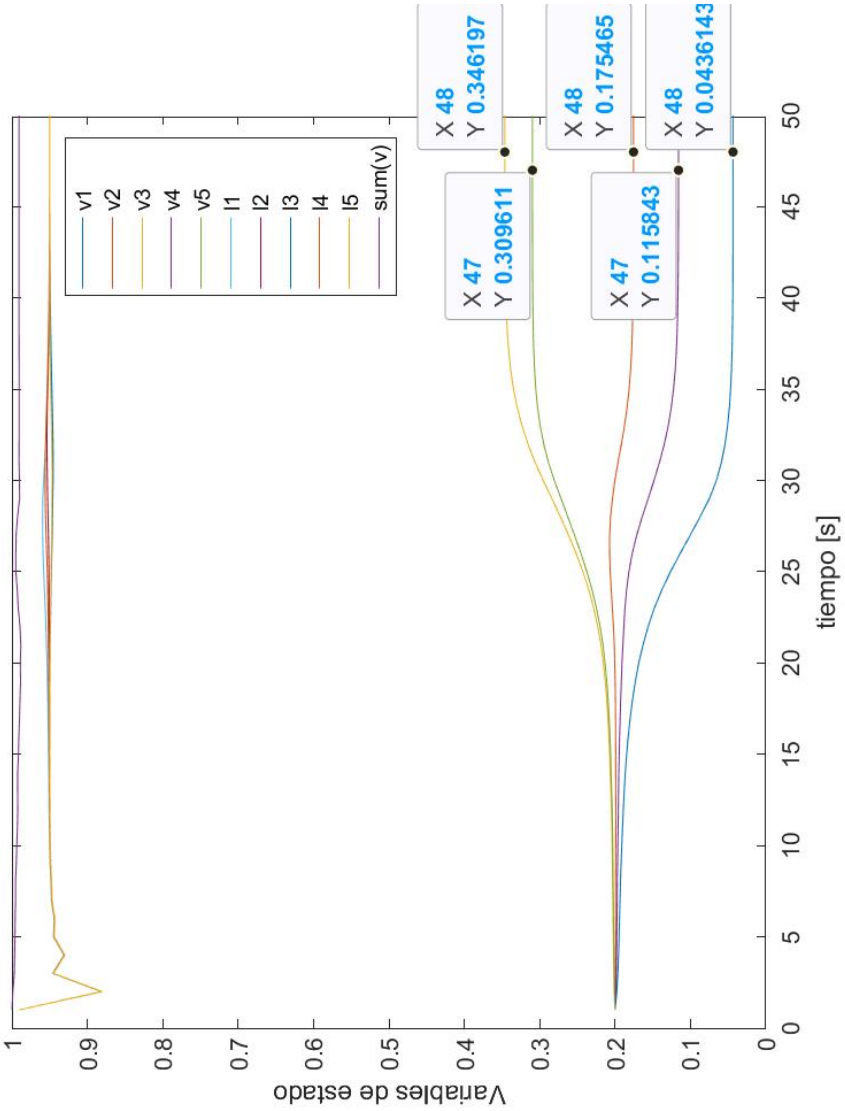


Figura 4.1: Gráfica de las variables de estados obtenidos en cincuenta iteraciones para el caso sin truncamiento.

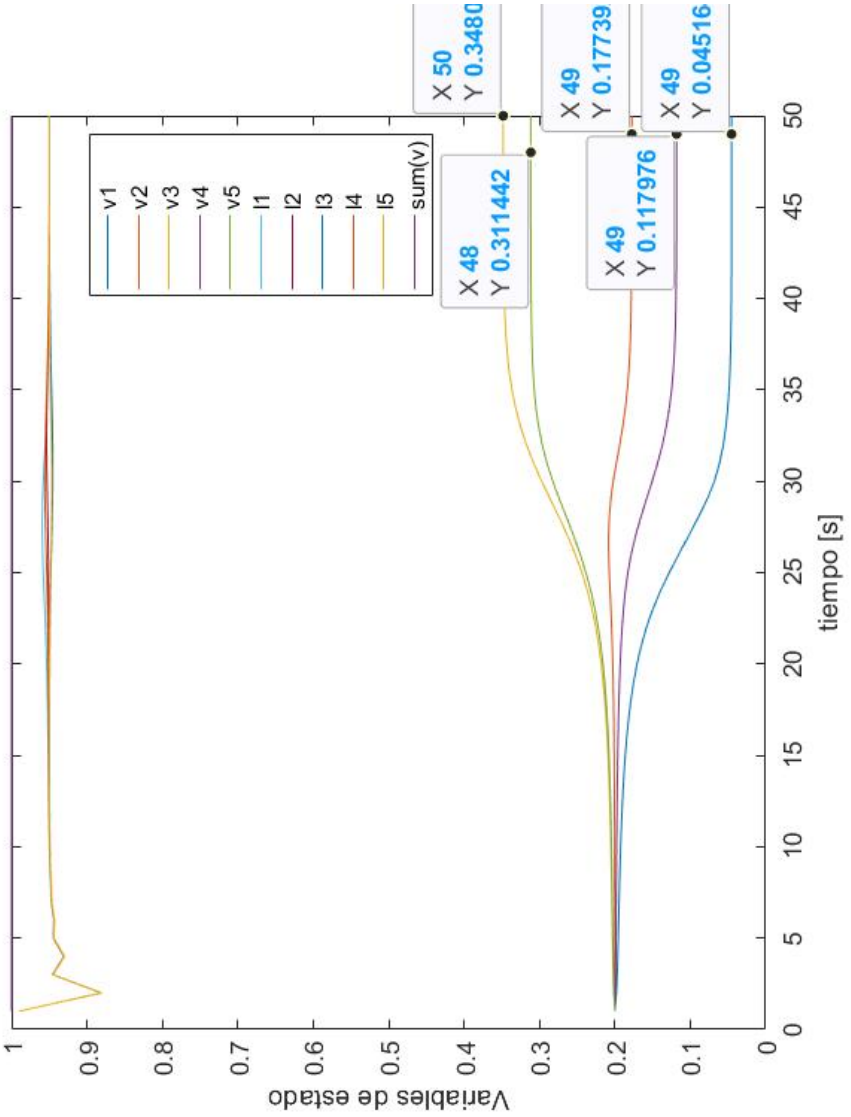


Figura 4.2: Gráfica de las variables de estados obtenidos en cincuenta iteraciones para el caso con truncamiento. Se observa lo similar que es al caso sin truncamiento

```
Elapsed time is 5.817365 seconds.
Elapsed time is 6.117217 seconds.
Elapsed time is 4.814993 seconds.
Elapsed time is 4.568661 seconds.
Elapsed time is 5.079522 seconds.
Elapsed time is 5.988969 seconds.
Elapsed time is 6.830930 seconds.
Elapsed time is 7.141553 seconds.
Elapsed time is 6.088496 seconds.
Elapsed time is 4.988465 seconds.
Elapsed time is 6.611501 seconds.
Elapsed time is 5.809165 seconds.
Elapsed time is 5.723595 seconds.
Elapsed time is 5.186129 seconds.
Elapsed time is 5.679938 seconds.
Elapsed time is 6.679113 seconds.
Elapsed time is 5.323662 seconds.
Elapsed time is 5.376272 seconds.
Elapsed time is 5.644466 seconds.
Elapsed time is 7.379964 seconds.
Elapsed time is 8.892559 seconds.
Elapsed time is 7.333530 seconds.
Elapsed time is 7.151563 seconds.
Elapsed time is 7.498174 seconds.
Elapsed time is 7.428046 seconds.
Elapsed time is 7.801318 seconds.
Elapsed time is 5.081420 seconds.
```

Figura 4.3: Tiempos por cada ciclo del programa sin truncamiento. Se observa que cada ciclo se mantiene cerca de un tiempo promedio.

```
Elapsed time is 0.084817 seconds
Elapsed time is 0.189839 seconds
Elapsed time is 0.870417 seconds
Elapsed time is 9.442192 seconds
Elapsed time is 5.434554 seconds
Elapsed time is 5.402652 seconds
Elapsed time is 6.159590 seconds
Elapsed time is 4.050222 seconds
Elapsed time is 0.666534 seconds
Elapsed time is 0.195848 seconds
Elapsed time is 0.116135 seconds
Elapsed time is 0.421934 seconds
Elapsed time is 5.848541 seconds
Elapsed time is 4.380337 seconds
Elapsed time is 5.133824 seconds
Elapsed time is 0.538654 seconds
Elapsed time is 0.313876 seconds
Elapsed time is 0.254084 seconds
Elapsed time is 0.168296 seconds
Elapsed time is 0.111467 seconds
Elapsed time is 0.072057 seconds
Elapsed time is 0.051765 seconds
Elapsed time is 0.035620 seconds
Elapsed time is 0.027256 seconds
Elapsed time is 0.026186 seconds
Elapsed time is 0.044353 seconds
Elapsed time is 0.023772 seconds
Elapsed time is 0.019627 seconds
Elapsed time is 0.034079 seconds
Elapsed time is 0.033031 seconds
```

Figura 4.4: Tiempos por cada ciclo del programa con el truncamiento. Se muestra que en ciertos puntos el tiempo es similar al caso sin truncamiento, cuando se acerca al equilibrio el tiempo disminuye.

o lo que es lo mismo, cuando se llega a la matriz:

$$A = \begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & ... & 1 \end{bmatrix} \tag{4.17}$$

Si las variables de estado están cerca del punto de equilibrio el peso h de la matriz anterior dominará al de las demás matrices. Por lo que si elegimos como valor inicial un vector de estados cercano al punto de equilibrio, el truncamiento que se realizará afectará un mayor número de subsistemas y la ejecución del programa sera rápida.

4.4.1. Truncamiento de h

Definiremos una nueva variable que es la multiplicación de dos pesos h.

$$H^{(n)} = h_i^{(n)} h_j^{(n)} \tag{4.18}$$

El superíndice se usa para distinguir entre truncamientos diferentes.

- Necesitamos realizar el truncado de manera que no se trunquen todos los valores de H, recordando que la cantidad de variables h es 2^{2n} para esta tesis. Aprovechando que la suma de todas las h debe ser igual a uno y usando el caso cuando todas las h_i son iguales, por lo que no se pueden truncar, tenemos:

$$h_i = h_j \tag{4.19}$$

$$h_1 + h_2 + ... + h_{2^{2n}} = 1 = 2^{2n} h_i \tag{4.20}$$

$$h_i^{(1)} = \frac{1}{2^{2n}} \tag{4.21}$$

$$2^{2n} h_i^{(1)} h_i^{(1)} = 2^{2n} H_{i,i}^{(1)} = h_i^{(1)} \tag{4.22}$$

$$H_{i,i}^{(1)} = \frac{h_i^{(1)}}{2^{2n}} = \frac{1}{2^{4n}} \tag{4.23}$$

- Si truncamos con un valor $T \leq H_{i,i}^{(1)}$ aseguramos que no se pierdan todos los valores de (2.12)
- Al truncar nos referimos a que cada subsistema que cumpla con $h_i h_j < T$ se omitirán de la suma en (2.12)

Realizaremos el truncamiento con la intención de que no se pierda por completo el 10 % del sistema. Para obtener el valor del truncamiento usaremos un sistema cuyo valor es el 10 % del sistema original, se analiza el caso cuando todas las h_i son iguales:

$$0.1\sum_{i=1}\sum_{j=1} hh(A_i - B_i F_j)x(t) = 0.1hh\sum_{i=1}\sum_{j=1}(A_i - B_i F_j)x(t)$$
$$= H^{(2)}\sum_{i=1}\sum_{j=1}(A_i - B_i F_j)x(t) \quad (4.24)$$

$$H^{(2)} = 0.1hh = \frac{0.1}{2^{4n}} = h^{(2)}h^{(2)} \quad (4.25)$$

$$h^{(2)} = \frac{\sqrt{0.1}}{2^{2n}} \quad (4.26)$$

$$h^{(2)} + h^{(2)} + ... + h^{(2)} = 2^{2n}h^{(2)} = \sqrt{0.1} \quad (4.27)$$

- Se puede analizar C.9 como el sistema original con $H^{(2)} = 0.1H^{(1)} = \frac{0.1}{2^{4n}}$
- Si se realiza el truncamiento para valores más pequeños que $H^{(2)}$ se garantiza que se obtendrá el 90 % del valor final de (2.12) y no siempre se perderá por completo el 10 % restante

Un problema que puede llegar a surgir es que el valor de una matriz $G_{i,j}$ de la ecuación (2.14) sea considerablemente grande y su peso H correspondiente sea truncado, de manera que no se pueda llegar al 90 % del valor final.

- Si no se puede llegar a una aproximación del 90 % de (2.12) cuando se realiza el truncamiento con $H^{(2)}$, se dice que existen matrices $G_{i,j}$ con valor significativo que fueron truncados.

- Para comprobar el punto anterior nos interesa demostrar si la suma de los subsistemas truncados es menor o mayor al 10 % de (2.12). De ser mayor al 10 %, la única explicación es que se truncaron valores de $G_{i,j}$ significativamente grandes.

Se demostrará en el Apéndice C los siguientes puntos:

- No existen valores de $G_{i,j}$ que sean significativamente grandes, por lo que siempre se obtendrá una aproximación de al menos el 90 % del valor de (2.12)
- La inclusión de $G_{i,j}$ en el análisis provoca que la noción de truncamiento se cambie por la de un número (suma de todos los subsistemas truncados) el cual se suma o resta a (2.12).

4.4.2. Control integral

Las plataformas virtuales y los pesos λ están diseñados para estar dentro del rango (0,1], fuera de ese rango no tienen significado. El cumplimiento de la desigualdad matricial (2.16) solo garantiza la convergencia al punto de equilibrio, sin embargo no garantiza que las variables de estado estén en el rango (0,1]. La siguiente matriz es un ejemplo de las matrices que se obtienen para el caso de tres aplicaciones, en el caso de esta tesis todas las matrices G tienen valores negativos en la mitad inferior.

$$G = \begin{bmatrix} 0.7 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -0.2030 & 0 & 0 & -0.0046 & 0 & 0 \\ 0 & -0.3065 & 0 & 0 & -0.0003 & 0 \\ 0 & 0 & -0.2125 & 0 & 0 & -0.0344 \end{bmatrix} \tag{4.28}$$

La actualización de las plataformas virtuales se obtiene de la ecuación (2.14). Si multiplicamos la matriz anterior por las variables de estado, las cuales son valores en

el rango (0,1], y luego multiplicamos por h, que también es un valor en el rango (0,1], los nuevos valores de λ resultantes pueden ser negativos, lo cual no tienen significado en el sistema.

La manera de solucionar esto es usar un control integral, el cual realiza cambios menos bruscos en las variables de estados.

El control integral esta dado por:

$$x_{k+1} = x_k + Bq_k \tag{4.29}$$

$$q_k = 0.75u_k - 0.25u_{k-1} + q_{k-1} \tag{4.30}$$

Donde x son las variables de estado, u es la entrada, y q es el valor que proporciona el control integral para estabilizar λ. Dado que se necesita una aproximación rápida al valor al que se desea que λ se acerque, denominado λ de referencia λ_{ref}, se puede inicializar $q_{k-1} \geq \lambda_{ref}$ ya que sabemos que λ pasara a ser negativo. Así los valores iniciales son:

$$q_{k-1} = \lambda_{ref} \tag{4.31}$$

$$u = x_{ref} - x_{k=0} \tag{4.32}$$

$$u_{k-1} = 0 \tag{4.33}$$

4.4.3. Método de selección de los valores iniciales

si los valores iniciales de las variables de estado están cerca del punto de equilibrio, se requerirá menos tiempo para alcanzar dicho punto. Un problema que se presenta para lograr que los valores iniciales estén cerca del punto de equilibrio es que el nivel de servicio también debe estarlo.

El nivel de servicio solo puede mantener su valor dentro del rango (0,1]. Cuando el nivel de servicio es muy grande, las plataformas virtuales tienden al valor $1/n$. A medida que el nivel de servicio disminuye, las plataformas virtuales comienzan a alejarse del valor de $1/n$ hasta llegar al punto de equilibrio.

Es por esto que no se puede elegir un valor inicial cercano al punto de equilibrio si los niveles de servicio son muy grandes.

Para elegir los valores iniciales de las plataformas virtuales se proponen dos opciones.

En la primera opción se normaliza los niveles de servicio para que como máximo el nivel de servicio valga uno.

De esta manera, es posible utilizar el valor del nivel de servicio dividido entre la suma de todos los niveles de servicio para asignar el valor inicial de la plataforma virtual (4.4.3). Esto asegura que la suma de las plataformas virtuales iniciales sea igual a uno. Dado que las plataformas virtuales son proporcionales al nivel de servicio, se espera que se mantenga en cierta medida la proporción en los valores futuros.

$$v_i = \frac{s_i}{\sum_{k=1}^{n} s_k} \tag{4.34}$$

Otra opción si no se realiza la normalización del nivel de servicio es:

$$v_i = \frac{1}{n} \tag{4.35}$$

Aunque aparecerán tiempos de cálculo de las variables de estado muy grandes, es la mejor opción para valores del nivel de servicio grandes.

Se puede usar varios métodos para determinar un conjunto de valores iniciales según sea más conveniente.

4.5. Resumen de capítulo

En este capítulo se da a conocer algunas diferencias que tendrá la implementación del RM con lo propuesto en artículos en los cuales esta basado este trabajo.

Se resuelven los problemas técnicos que se presentan en la implementación del RM. Los logros son los siguientes:

- Mejoras en tiempos de ejecución

- Mejoras en el uso de espacio de memoria
- Uso de control integral para limitar los cambios en las variables de estados
- Selección de variables de estado iniciales

Destaca la sección del uso de truncamiento en sistemas difusos por su posible implementación en los sistemas difusos de manera general

Capítulo 5

Comunicación entre dispositivos

5.1. Introducción

En este capítulo se explica el procedimiento por el cuál se realiza la comunicación entre el programa principal escrito en python, el programa del control difuso, escrito en MATLAB y la comunicación con las aplicaciones que se ejecutan en distintos dispositivos. Se analiza el código principal del proyecto

5.2. Comunicación matlab con python

Matlab proporciona a python la librería *matlab.engine* la cual permite la comunicación entre una ventana de comandos abierta de matlab y un programa en python.

Antes de ejecutar el programa de python se debe convertir una sesión abierta de matlab en una sesión compartida, usando en la ventana de comandos de matlab el comando

```
matlab.engine.shareEngine()
```

Se puede especificar un nombre a la sesión pasando como parámetro el nombre en la función anterior. De otro modo asignará un nombre por default el cuál se puede obtener por medio del comando

```
matlab.engine.engineName
```

En el código de python se debe iniciar la conexión con matlab por medio del comando

```
eng=matEng.connect_matlab('nombreDeSesiónDeMatlab')
```

Donde como parámetro se necesita el nombre de la sesión de matlab en forma de un string (en un entrecomillado). La función anterior devuelve un objeto de tipo MatlabEngine que se guarda en una variable, en nuestro caso en la variable *eng*.

A partir de la variable *eng* se manejaran tanto funciones propias de matlab, funciones que hemos creado y constantes de matlab, como se muestra en los siguientes ejemplos:

```
eng.pi
eng.sqrt(4)
eng.miFuncion()
```

Cuando una función de matlab que se manda a llamar debe devolver una matriz, se devuelve en su lugar un objeto de tipo *matlab.double*, los valores individuales de la matriz son reconocidos por python como valores de tipo *double*, pero una submatriz se reconocerá de tipo *matlab.double*.

Matlab en memoria almacena los datos por columnas a diferencia de python que los almacena por filas, por lo que la matriz estará transpuesta, para obtener la matriz como una matriz de python, con la orientación correcta, se puede usar el siguiente código.

```
c=[[] for i in range(x.size[0])]
for (i, item) in enumerate(x._data):
    c[i % x.size[0]].append(item)
```

donde x es la matriz original y c es la nueva matriz

5.3. Comunicación SSH

Para la comunicación entre el programa python y las consolas de los dispositivos se usa la librería *paramiko* la cual proporciona los protocolos de comunicación SSHv2.

Para iniciar una comunicación como cliente se utiliza la función:

```
client=paramiko.SSHClient()
```

El objeto obtenido de esta función se encarga de los aspectos de autenticación y apertura de canales comúnmente usados. Las funciones se ejecutan a través de la variable *client* donde guardamos el objeto [paramiko.org (2023)].

Utilizamos la función:

```
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

Para configurar la política que se utilizará cuando se conecte a servidores sin una clave de host conocida.

En donde

```
paramiko.AutoAddPolicy()
```

Es usado para agregar automáticamente el nombre del host y la nueva clave del host al objeto Hostkeys.

Se inicia la conexión con los dispositivos a través de la función

```
client.connect(host,username,password)
```

Que pide como parámetros la IP del dispositivo con el que nos queremos conectar, su usuario y contraseña.

Una vez iniciada la comunicación con la consola del dispositivo podemos escribir comandos en la consola con:

```
stdin,stdout,stderr=client.exec_command('comando')
```

El comando debe ser pasado como un parámetro de tipo string. Se reciben una matriz con tres elementos, la entrada estándar, la salida estándar y el error. Si la ejecución del comando devuelve algún valor, este estará codificado en stdout, para obtener el valor se usa:

```
result=stdout.read().decode()
```

Algunos ejemplos de los comandos que se pueden usar son:

```
stdin,stdout,stderr=client.exec_command('ls')
stdin,stdout,stderr=client.exec_command('python3 p1.py')
```

Donde el primero muestra los archivos y carpetas que hay en la ubicación actual y el segundo ejecuta el programa de python llamado p1.py. En este proyecto los programas ya están cargados en los dispositivos, solo se les enviara a ejecutar.

5.4. Ejecución de programa python con parámetros

En este proyecto es necesario aparte de la ejecución de los programas en los dispositivos el pasar parámetros como el tiempo que permanecerá ejecutándose y alguna variables que cambiará entre una ejecución y otra. Para ello se utilizara la función de ejecutar comando de la siguiente forma:

```
stdin,stdout,stderr=client.exec_command('python3 p2.py '+str(variable)
+' '+str(variable2))
```

Se observa que se manda a ejecutar el programa p2.py, separado con un espacio se concatena una variable y separado con otro espacio se concatena la otra variable. Se pueden colocar cualquier número de variables de esta forma con la condición que sean strings.

Por otro lado, el dispositivo que contiene el programa a ejecutar debe incluir la librería *sys* para poder interpretar las variables del comando anterior.

Para convertir los argumentos en la línea de comandos a un script de python se usa el comando:

```
arg=sys.argv
```

Donde arg[0] es el nombre del script. Después convertimos el script en una lista y convertimos cada string de la lista en su tipo correspondiente como se muestra

```
    arg=list(arg)
    v=float(arg[1])
    i=float(arg[2])
```

5.4.1. Limitar tiempo de ejecución de programas

Para ejecutar un programa durante un período determinado de tiempo, se debe utilizar hilos. Un hilo se encargará de ejecutar el programa, mientras que el otro hilo se encargará de medir el tiempo de ejecución. Para lograr esto, se utilizarán las librerías *threading* y *time*.

La estructura general de los programas que se ejecutaran se ve a continuación:

```
import time
import sys
import threading

#Pasar parametros de entrada a su correspondiente tipo de variable
arg=sys.argv
arg=list(arg)
v=float(arg[1])
i=float(arg[2])



#Función que contiene el código principal
def f(i):
    while(True):

            #El código del programa va aquí

            if stop:
```

```
            return(resultado)
            return
```

```
#Lanzamiento de hilo y contador de tiempo de ejecución
stop=False
t=threading.Thread(target=f, args=(i,))
t.start()
time.sleep(v)
stop=True
```

En la primera parte del código, se están obteniendo las variables v e i. La variable v representa el tiempo de ejecución, mientras que la variable i es una variable que se utilizará en el programa.

En la segunda parte tenemos una función que contendrá al código principal dentro de un ciclo *while* que solo termina si la variable booleana *stop* cambia a *True*. Se utiliza la función *return* o alternativamente se puede usar *print* para enviar el resultado de la ejecución del programa.

En la ultima parte del código, se inicializa la variable *stop* como *False*. Se lanza el hilo que ejecutará a la función f y se pone a dormir el hilo principal el tiempo que dicta la variable v. Una vez terminado el tiempo de espera, la variable *stop* cambia a *True* y esto hace finalizar la ejecución del hilo.

Otra manera de terminar la ejecución de un programa es enviando un segundo comando a la consola para indicar que la aplicación debe detener su ejecución, como se ve en el siguiente código:

```
import time
import sys
import threading
```

```
tiempo=10
arg=sys.argv
arg=list(arg)
v=float(arg[1])
s=float(arg[2])
data=float(arg[3])
l=0.97



def f(i):
    while(True):
        #El código del programa va aquí
        if stop:
            return(str(s)+" "+str(l)+" "+str(i))
#Lanzamiento de hilo y contador de tiempo de ejecución
stop=False
t=threading.Thread(target=f, args=(data,))
t.start()
while(stop==False):
    x=input()
    if x=="f":
        stop=True
```

En este código el hilo principal espera a que entre un dato desde la consola y si este es una f detiene el programa y envía el resultado, de esta manera el programa principal es el encargado de elegir en que momento se detiene cada aplicación.

5.5. Programa principal de python

El código principal del programa se muestra en el diagrama de flujo de la Figura 5.1

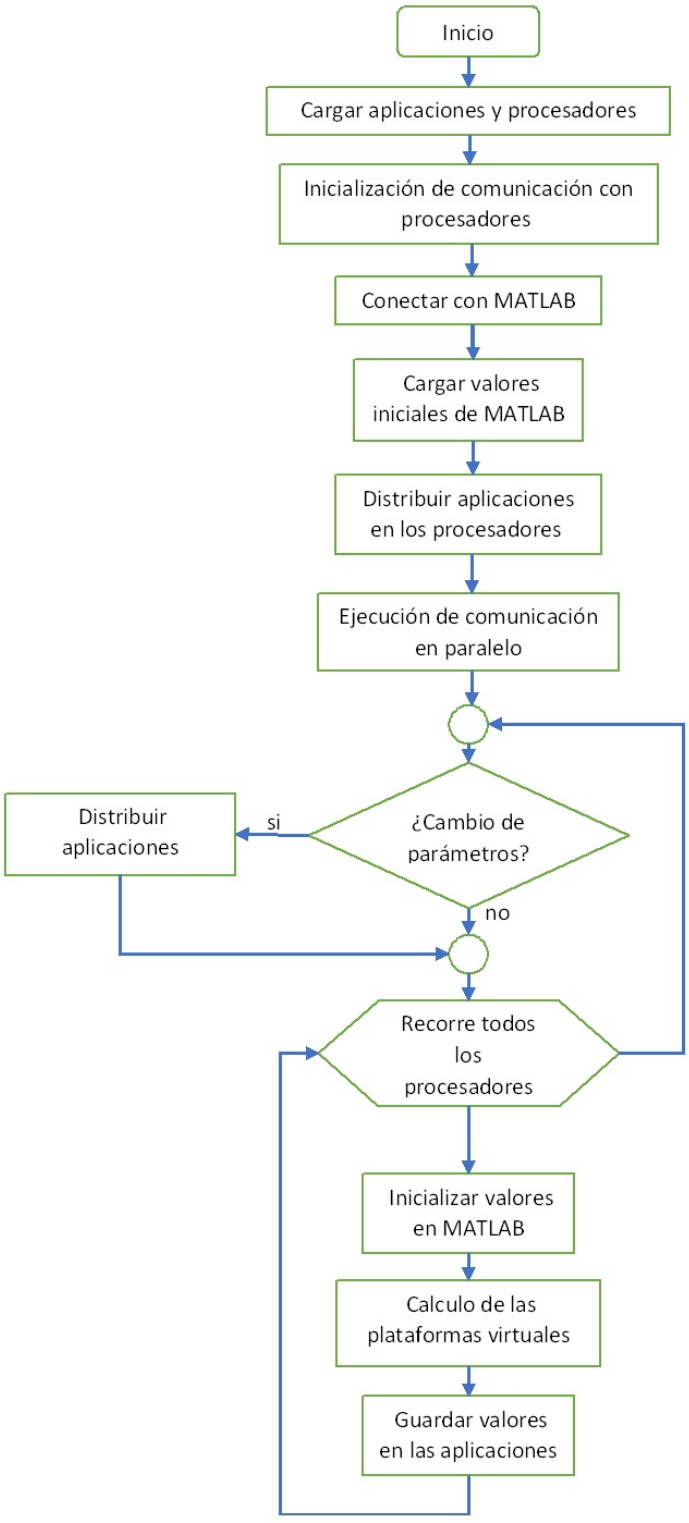


Figura 5.1: Diagrama de bloques del programa principal

En paralelo al programa principal se ejecutara la comunicación entre el RM y los procesadores que ejecutarán las aplicaciones. El diagrama de flujo se presenta en la Figura 5.2. A continuación se explicará a detalle los bloques de los diagramas

5.5.1. Cargar aplicaciones y procesadores

Al iniciar el programa se debe proporcionar los siguientes datos:

- Número de aplicaciones
- Número de procesadores
- Nombre de las aplicaciones
- Prioridad inicial de las aplicaciones
- Nivel de servicio de las aplicaciones
- Valores iniciales de los datos a manejar
- Tiempo de arribo de las aplicaciones
- Tiempo de computación de las aplicaciones
- IP de los procesadores
- Nombre de usuario de los procesadores
- Contraseñas de los procesadores
- Capacidad de procesamiento relativa de los procesadores

El tiempo de arribo es el instante de tiempo a partir del cual la aplicación puede comenzar su ejecución

El tiempo de computación es el tiempo que requiere ejecutarse la aplicación para terminar su tarea

La IP, el nombre de usuario y la contraseña son necesarios para iniciar la comunicación por medio del protocolo SSH

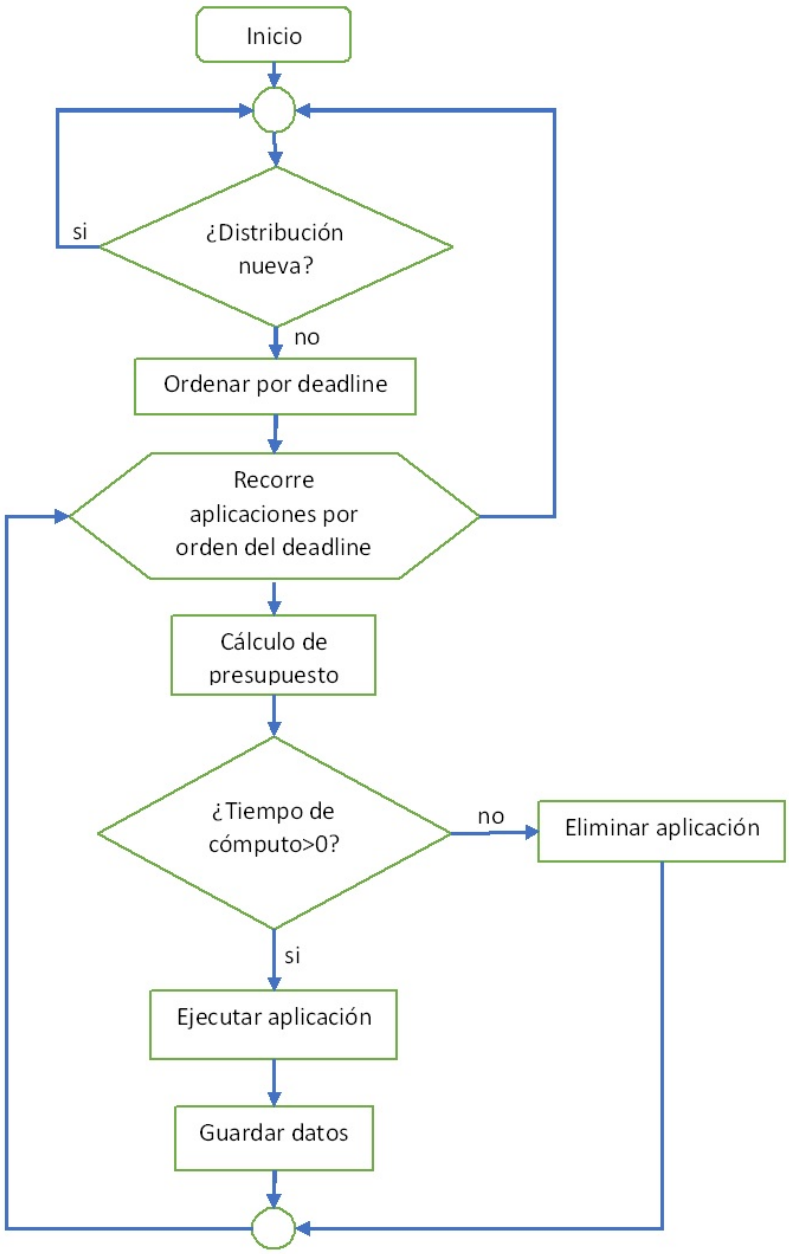


Figura 5.2: Diagrama de bloques de la comunicación en paralelo entre el RM y los procesadores

Las variables antes mencionadas y otras más que se obtendrán durante la ejecución del programa se guardaran en un par de diccionarios de python, uno dedicado a las variables de las aplicaciones y el otro para las variables de los procesadores.

Las variables pueden estar guardadas en el programa o en un archivo, con la ventaja que en este ultimo caso se puede guardar los cambios de las variables para su uso en posteriores ejecuciones

```
p={
"apps":[],
"nuevo":[],
"cliente":[],
"capacidad":[],
"c":[],
"u1":[],
"q":[],
"grafica":[],
"grafica1":[],
"grafica2":[],
"grafica3":[],
}
P=[]


Apps=[]
app={
"nombre":[],
"v":[],
"l":[],
"s":[],
"data":[],
"d":[],
```

```
"c":[],
"r":[],
"comp":[],
"grafica":[],
"grafica1":[],
"grafica2":[],
}
```

5.5.2. Inicialización de la comunicación con los procesadores

En esta parte se usa la teoría de la sección 5.3 para realizar una función que entregue como resultado un objeto encargado de la comunicación, el cual es guardado en el diccionario dedicado a las variables del procesador

```
def inicioConexion(i):
    client=paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(host[i],username=user[i],password=password[i])
    return(client)
```

5.5.3. Conexión con MATLAB y carga de valores iniciales de MATLAB

Se utiliza la teoría vista en la sección 5.2 para realizar la conexión entre MATLAB y python, los valores que se cargan en MATLAB al iniciar son las matrices obtenidas en el Capitulo 4 para agilizar la ejecución del código en MATLAB.

5.5.4. Distribución de aplicaciones en los procesadores

Inicialmente no hay relación entre los procesadores y las aplicaciones por lo que se debe encontrar una manera conveniente de distribuir las tareas en los procesadores.

En nuestro caso se usa la siguiente ecuación:

$$\frac{\lambda_1 s_1 + \lambda_2 s_2 + ... + \lambda_n s_n}{\sum p} = x \tag{5.1}$$

donde λ es la prioridad de la aplicación, s es el nivel de servicio y p es la capacidad normalizada del procesador. Una vez obtenido el valor de x se usa un algoritmo que seleccione las combinaciones que cumplan con:

$$\frac{\sum \lambda_i s_i}{p_1} \approx \frac{\sum \lambda_j s_j}{p_2} \approx \frac{\sum \lambda_k s_k}{p_n} \approx x \tag{5.2}$$

La justificación de este método se da en el Apéndice B

5.5.5. Ejecución de comunicación en paralelo

Al mismo tiempo que se calculan las plataformas virtuales, se correrá en paralelo el algoritmo de la Figura 5.2 para cada procesador.

El objetivo de este bloque es iniciar la comunicación en paralelo, cada hilo que se inicia se le da como argumento el diccionario del procesador al cual estará asociado

5.5.6. Cambio de parámetros

Cada vez que las aplicaciones realicen un cambio en su nivel de servicio o en su prioridad, se realiza una nueva distribución de las aplicaciones.

Si las aplicaciones tipo soft se están ejecutando en el momento que se inicia una nueva distribución, continuarán ejecutándose hasta que terminen.

Cuando se asigna el orden de ejecución de las aplicaciones en un procesador, se realiza una copia del diccionario de dicho procesador. De esta manera, aunque las aplicaciones dentro del diccionario original cambien, podemos saber qué aplicaciones aún faltan por ejecutarse y podemos guardar el resultado de su ejecución en la aplicación correspondiente. Si las aplicaciones terminan de ejecutarse antes de obtener la nueva distribución, se entra en un bucle de espera hasta que se obtenga dicha distribución.

Las aplicaciones tipo hard, no cambian de procesador por lo que seguirán ejecutándose de manera normal

5.5.7. Actualización de plataforma virtual

El ciclo *for* en la parte final del diagrama es la encargada del cálculo de los nuevos valores de las plataformas virtuales, niveles de servicio y prioridad de las aplicaciones. Se realiza el cálculo de los valores para un procesador a la vez.

5.6. Comunicación en paralelo entre RM y los procesadores

En esta sección analizaremos el diagrama de bloques de la Figura 5.2

Si el programa está generando una nueva distribución, la ejecución de las siguientes aplicaciones planificadas de tipo *soft* se detendrá hasta que la nueva distribución esté completada.

Posteriormente, se asigna el orden en el que se ejecutarán las aplicaciones, de manera que se prioricen aquellas cuyo deadline está más próximo a cumplirse.

Una vez obtenido el orden, se calcula el tiempo de ejecución de cada aplicación, además de calcular el siguiente deadline de las aplicaciones.

Si el tiempo de computo es cero, la aplicación terminó su labor y podemos dejar de considerarla para el cálculo de las plataformas virtuales.

5.7. Resumen del capítulo

En este capítulo se presentó el programa principal del proyecto, el cual tiene las siguientes funciones principales:

- Realizar la conexión del RM con los dispositivos en la red.
- Enviar, recibir y guardar parámetros y datos de las aplicaciones y procesadores.

- Activar y manejar el RM, aplicaciones, CBS y el algoritmo de distribución.

Capítulo 6

Resultados

6.1. Introducción

En esta sección se presenta la parte experimental del proyecto, junto con los resultados obtenidos y su correspondiente análisis. A menos que se indique lo contrario, los siguientes valores son los que se utilizaron en las pruebas.

Se realiza una distribución de recursos con 3 raspberry con los siguientes parámetros:

Raspberry	Modelo	sistema operativo	frecuencia
1	Raspberry Pi 3 Modelo B V1.2	Pi Os (32-bit)	0.6[GHz] a 1.2[GHz]
2	Raspberry Pi 3 Modelo B V1.2	Pi Os (64-bit)	0.6[GHz] a 1.2[GHz]
3	Raspberry Pi 3 Modelo B V1.2	Pi Os Lite (32-bit)	0.6[GHz] a 1.2[GHz]

Como las aplicaciones a utilizar son relativamente simples, las raspberry trabajarán en las frecuencias bajas.

Como en estas pruebas no interesa el contenido de la aplicación sino la distribución de los tiempos de las aplicaciones, se puede elegir la relación de las capacidades de los procesadores, para estas pruebas se elige las mostradas en la siguiente tabla.

Raspberry	Capacidad relativa
1	1
2	0.9
3	0.8

Se utilizaron 10 aplicaciones de las cuales 3 son tipo *hard* y 7 son tipo *soft*, en la siguiente tabla se dan las características de las aplicaciones.

Aplicación	Tipo	Nivel de servicio	Periodo [s]	Tt [s]	Tiempo de computo [s]
1	Hard	N/A	4	0.5	N/A
2	Soft	3.8	3	N/A	200
3	Soft	2.3	3	N/A	200
4	Soft	3.7	3	N/A	200
5	Hard	N/A	5	0.5	N/A
6	Hard	N/A	5	0.75	N/A
7	Soft	1.9	3	N/A	200
8	Soft	3.0	3	N/A	200
9	Soft	2.3	3	N/A	200
10	Soft	2.4	3	N/A	200

En la tabla anterior, se eligieron niveles de servicio relativamente cercanos entre sí para evitar grandes diferencias en los recursos asignados. Además, todos los niveles de servicio son mayores que uno, lo que significa que los niveles de servicio disminuirán hasta alcanzar el intervalo (0,1], que es el rango donde se puede lograr el punto de equilibrio.

El periodo indica cada cuanto tiempo se actualiza el *deadline*. En el caso de las aplicaciones tipo *soft*, el periodo debe ser el mismo para todas ellas. El tiempo de trabajo (Tt) indica la duración necesaria para ejecutar las aplicaciones tipo *hard*. Por último, el tiempo de cómputo indica la cantidad de tiempo requerida para que una aplicación se complete.

Se muestra en qué raspberry se ejecutó la aplicación por medio del color, la si-

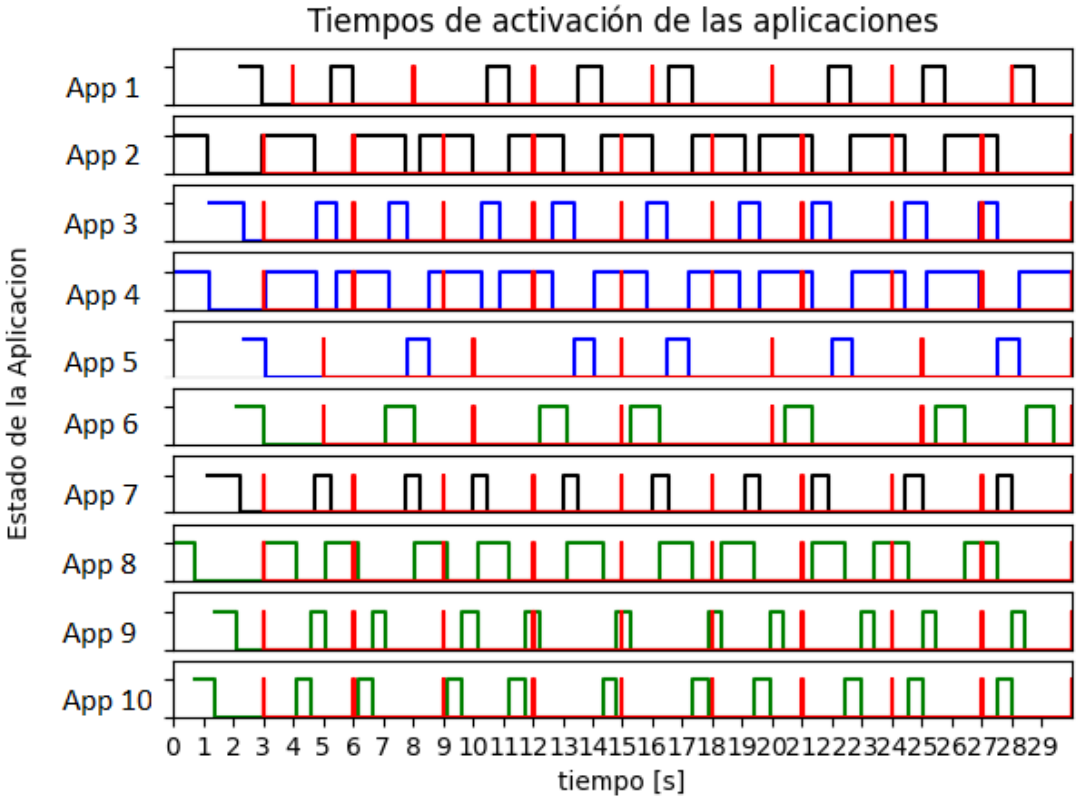


Figura 6.1: La figura pertenece a la operación sin cambios, se muestra los instantes de tiempo en que las aplicaciones fueron activadas y el tiempo que permanecieron activas. El orden de las aplicaciones es de arriba a abajo de la aplicación 1 a la 10. El color indica en que procesador se ejecuta la aplicación. Las lineas rojas indican los deadlilnes.

guiente tabla muestra la relación:

Raspberry	Color
1	Negro
2	Azul
3	Verde

6.2. Operación sin cambio

En esta prueba se ejecuta el RM sin que se produzca una nueva distribución durante la ejecución.

La Figura 6.1 muestra en que instante de tiempo se ejecuto la aplicación y cuanto

tiempo permaneció activa, las aplicaciones se muestran desde la aplicación uno a la diez de arriba a abajo, los colores de los pulsos indican en que procesador se ejecutó la aplicación, las lineas rojas indican los *deadline* de las aplicaciones.

Las siguientes características son generales y se encuentran en todas las pruebas que se presentarán en este capítulo.

- En todo momento, salvo en algunos casos que aparecen cuando hay una nueva distribución, las aplicaciones *soft* se ejecutan como un solo conjunto y en el mismo orden. Las aplicaciones *hard* se ejecutan siempre antes o después de que se ejecuten todas las aplicaciones *soft*.

- La prioridad de ejecución entre las aplicaciones *hard* y *soft* depende del *deadline* más cercano a cumplirse.

- Desde el inicio hasta los tres segundos, las aplicaciones tipo *soft* se ejecutaron con el valor de la plataforma virtual $1/n$ donde n es el número de aplicaciones en el dispositivo. Se puede observar que en la primera ejecución las aplicaciones en un mismo dispositivo se ejecutan la misma cantidad de tiempo.

- Se observa que la suma de los tiempos de ejecución de las aplicaciones *soft* en un dispositivo es menor o igual al periodo.

- En las gráficas, las plataformas virtuales aparecen normalizadas con respecto al dispositivo de mayor capacidad, por lo que si por ejemplo el valor asignado a la plataforma inicial de una aplicación es 0.5 y esta en el dispositivo de capacidad 0.9 el valor en la gráfica es 0.45.

- En la distribución inicial, es cuando más tiempo se tarda en alcanzar el equilibrio. Esto se debe a que los valores iniciales de los niveles de servicio están fuera del intervalo (0,1].

- A partir de la segunda ejecución de las aplicaciones y hasta que se realice una nueva distribución, los tiempos asignados a las aplicaciones son los correspon-

dientes al punto de equilibrio, Esto se debe a que el equilibrio se alcanza poco después del segundo uno.

- El periodo de las aplicaciones tipo *soft* se eligió lo suficientemente grande para que sea poco probable que el tiempo de ejecución sea menor a los 0.23[s] requeridos como mínimo para ejecutarse. Ese tiempo es debido a el tiempo de procesamiento que requiere la aplicación para decodificar la información proveniente de las raspberry.

- Inmediatamente después de que una aplicación termina su ejecución en una raspberry la siguiente aplicación inicia su ejecución inmediatamente, por lo que en todo momento se encuentran ocupadas las raspberrys.

- Todas las aplicaciones terminan su operación antes de su *deadline* correspondiente.

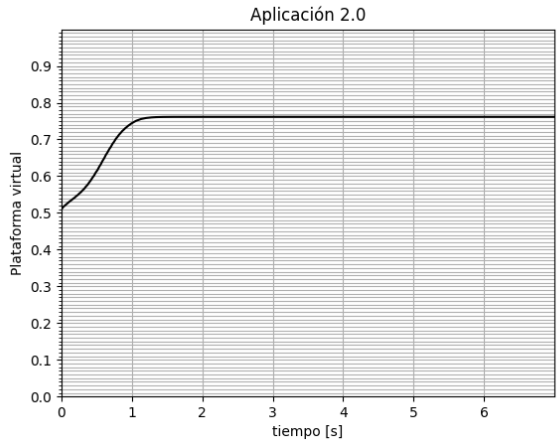
Las siguientes son características particulares de la prueba en esta sección

- Cada raspberry contiene únicamente una aplicación tipo *hard*

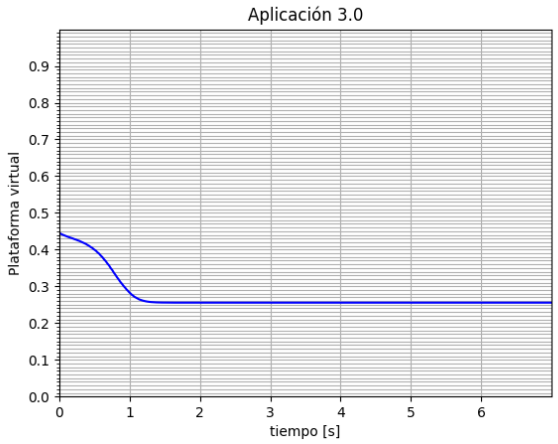
- Se observa, en las raspberry uno y dos, que se asigna una gran cantidad de recursos a una aplicación *soft* en comparación con la otra aplicación con la que comparte dispositivo , esto es debido a como funciona el algoritmo de distribución.

- En la aplicación dos, en su cuarta ejecución parece que no alcanza a terminar antes del *deadline*, pero realmente el *deadline* que le corresponde es el siguiente. La tendencia a que se ejecuten las aplicaciones con cada vez mayor tiempo de anticipación es debido a que el tiempo asignado a cada conjunto de aplicaciones *soft* puede ser menor a su periodo

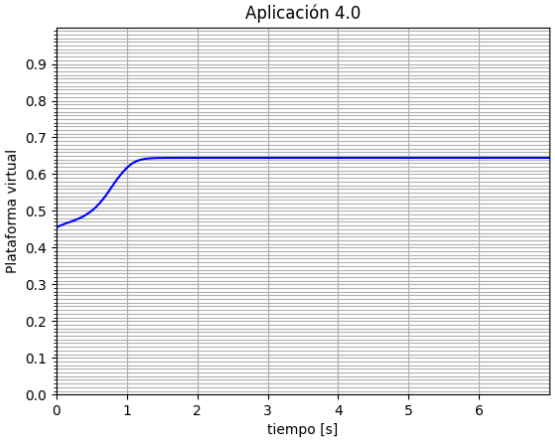
Si comparamos la relación de las plataformas virtuales y la capacidad relativa con respecto a los niveles de servicio iniciales, tenemos la siguiente tabla.



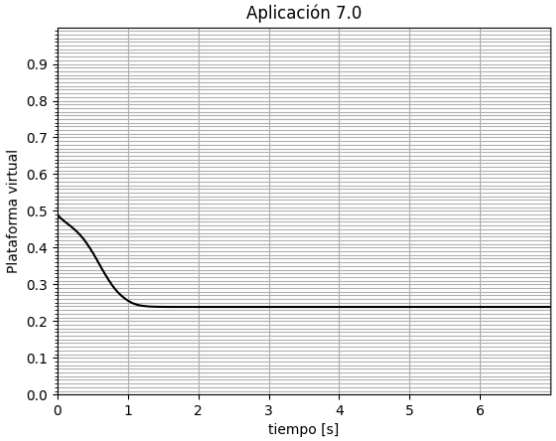
(a) Aplicación 2



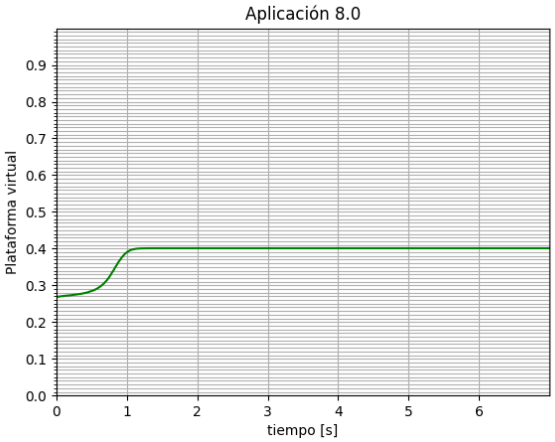
(b) Aplicación 3



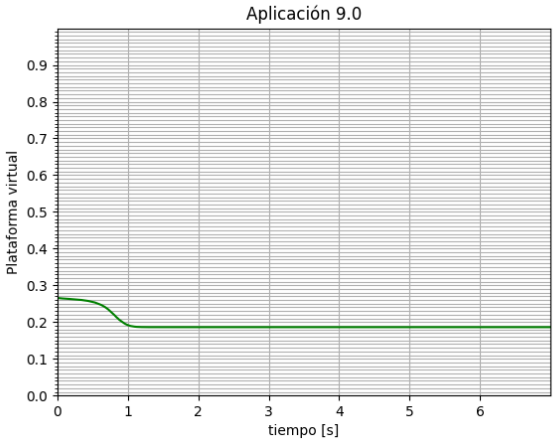
(c) Aplicación 4



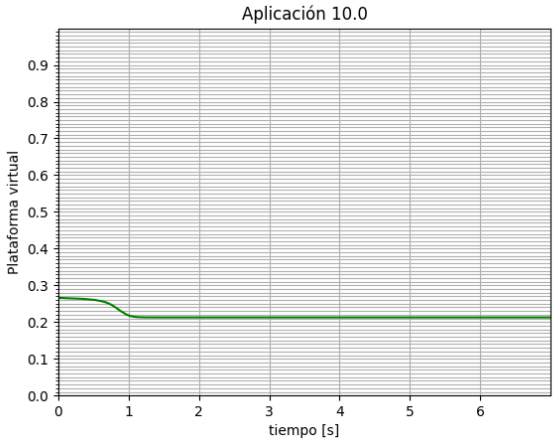
(d) Aplicación 7



(e) Aplicación 8



(f) Aplicación 9



(g) Aplicación 10

Figura 6.2: Comportamiento de las plataformas virtuales a lo largo del tiempo para el caso donde no se produce nuevas distribuciones. El color de las gráficas indica el procesador donde se han ejecutado, el valor de las plataformas virtuales está normalizada respecto al dispositivo de capacidad de computo mayor

Aplicación	Raspberry	C	v	C*v	Nivel de servicio
1	1	1	N/A	N/A	N/A
2	1	1	0.76	0.76	3.8
3	2	0.9	0.283	0.255	2.3
4	2	0.9	0.72	0.648	3.7
5	2	0.9	N/A	N/A	N/A
6	3	0.8	N/A	N/A	N/A
7	1	1	0.24	0.24	1.9
8	3	0.8	0.5	0.4	3.0
9	3	0.8	0.23	0.184	2.3
10	3	0.8	0.27	0.216	2.4

De la tabla anterior podemos notar que entre mayor sea el nivel de servicio inicial mayor serán los recursos recibidos, los recursos se normalizan respecto al procesador de mayor capacidad en (C*v) para poder realizar la comparación. En la tabla hay una excepción a la regla anterior, la cual aparece en la aplicación tres, esto puede ser debido a que solo en el caso ideal la distribución de recursos se puede realizar sin que

haya un cambio.

En la siguiente sección se vera que en el caso concreto en el que una aplicación gana más recursos de los que tenía no siempre se cumple esta regla.

6.3. Operación con agotamiento de tiempo de computo

En esta prueba se da a la aplicación tres un tiempo de computo de 8[s], por lo que una vez la aplicación se ha ejecutado ese tiempo deja de requerir recursos y una nueva distribución se lleva a cabo. Una vez realizada la nueva distribución no se presenta ningún otro cambio.

Las siguientes son características particulares de esta prueba que se observan en las figuras (6.3) y (6.4).

- A los 12.42[s] la aplicación dos termina de ejecutarse, lo que provoca una nueva distribución. Nuevamente se usa como plataforma virtual inicial $1/n$

- El cambio no se da de manera inmediata ya que se espera hasta que la ejecución de todo el conjunto de aplicaciones *soft* termine. También se observa que las aplicaciones tipo *hard* no cambian su distribución, esto es debido a que se diseño de manera que no cambien, se puede realizar también el cambio de las aplicaciones *hard* pero se debe de tener en cuenta todos los posibles casos en que se pueda perder un *deadline*.

- Las gráficas de las plataformas virtuales muestran que todas las aplicaciones recibieron más recursos de los que tenían. Una propiedad de este control es que si todas las aplicaciones en un mismo dispositivo reciben igual o más recursos de los que solicitan los niveles de servicio, el cambio de estos últimos es mucho mayor que el cambio en el valor de las plataformas virtuales (esto es debido a la forma de la función de emparejamiento), por lo que se llega al equilibrio sin que el cambio en la plataforma virtual sea perceptible. Esto se cumple siempre

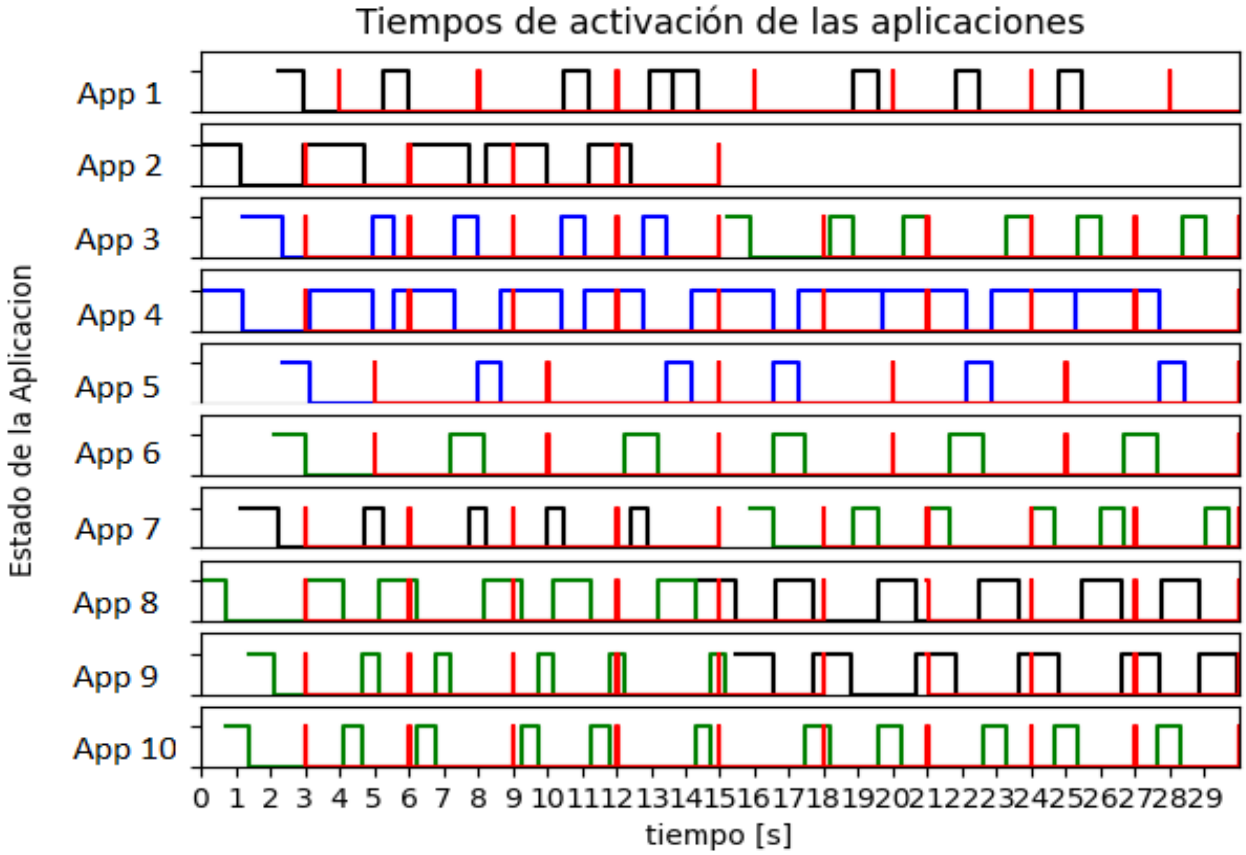


Figura 6.3: La figura muestra el caso donde se finaliza la aplicación dos, muestra los instantes de tiempo en que las aplicaciones fueron activadas y el tiempo que permanecieron activas. El color indica en que procesador se ejecuta la aplicación. Las lineas rojas indican los deadlines.

y cuando la suma de las plataformas virtuales sea uno. En la Figura 6.5 se puede ver un ejemplo del cambio del nivel de servicio en azul y el cambio en la plataforma virtual en rojo.

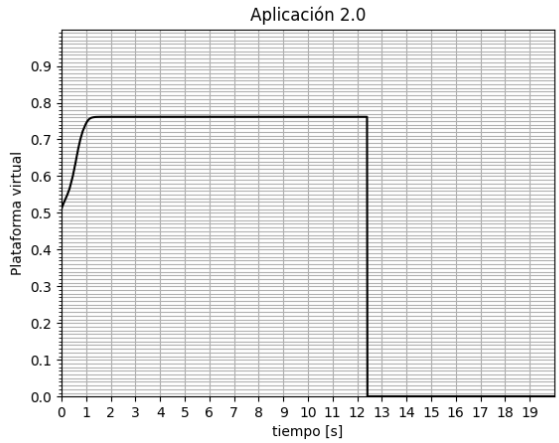
Esto lleva a que la idea de que una aplicación con un nivel de servicio inicial mayor reciba más recursos que una con un nivel de servicio inicial menor pueda dejar de cumplirse. Sin embargo, las aplicaciones no compiten entre sí, cada una se ocupa únicamente de recibir los recursos necesarios para un funcionamiento adecuado. Si las aplicaciones aceptan las disminuciones en su nivel de servicio, tal como lo dicta el control, es posible que dos aplicaciones con niveles de servicio iniciales muy diferentes terminen teniendo las mismas plataformas virtuales. Si a una aplicación no le resultan aceptables los recursos que se le asignan, puede cambiar su nivel de servicio por sí misma.

- Después del cambio del nivel de servicio, de las aplicaciones *soft*, la aplicación cuatro es la primera en ser ejecutada a los 14.14[s] con los nuevos valores de las plataformas virtuales calculados después de la distribución.

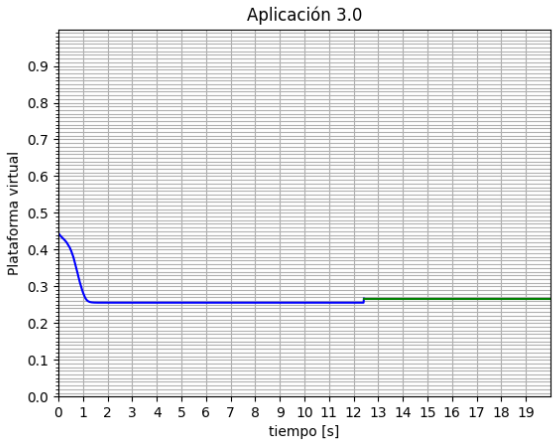
- Después de realizar la nueva distribución, se puede observar en la Figura 6.3 que la aplicación uno de tipo *hard* se ejecuta dos veces consecutivas. Esto se debe a que las aplicaciones ocho y nueve estaban en ejecución durante el cambio de distribución. Sin embargo, esto no afecta al cumplimiento de sus *deadlines*, ya que esta ultima ejecución estaba adelantada para las aplicaciones ocho y nueve, siendo sus *deadline* a los 18[s] para ese momento.

- Otro caso interesante es el de la aplicación cuatro, al ser la única aplicación tipo *soft* en el dispositivo se le asigna automáticamente todos los recursos.

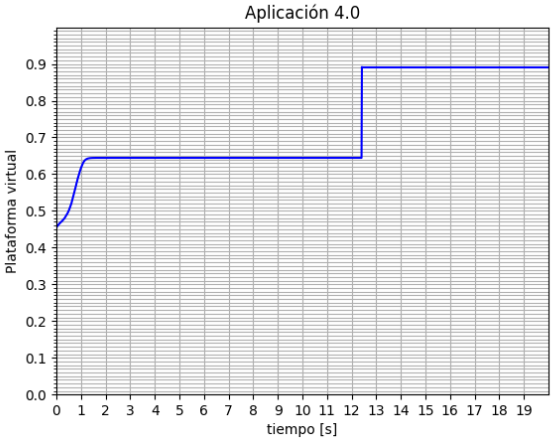
- Se cumple con todos los *deadline*.



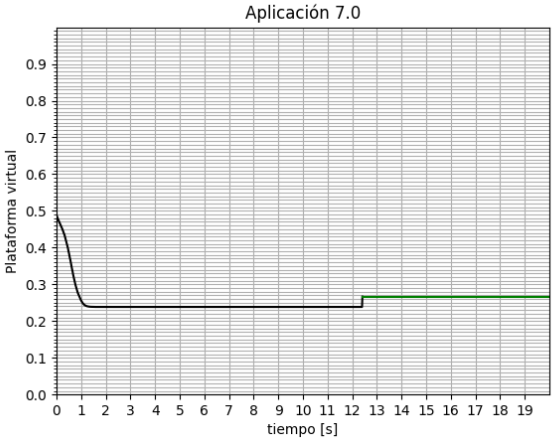
(a) Aplicación 2



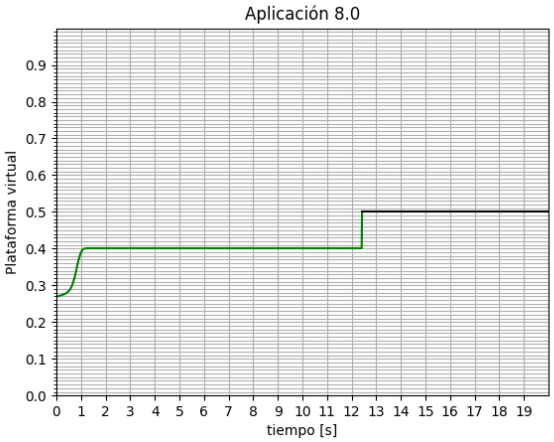
(b) Aplicación 3



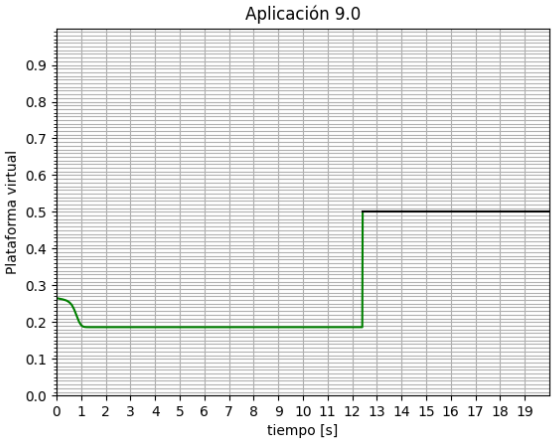
(c) Aplicación 4



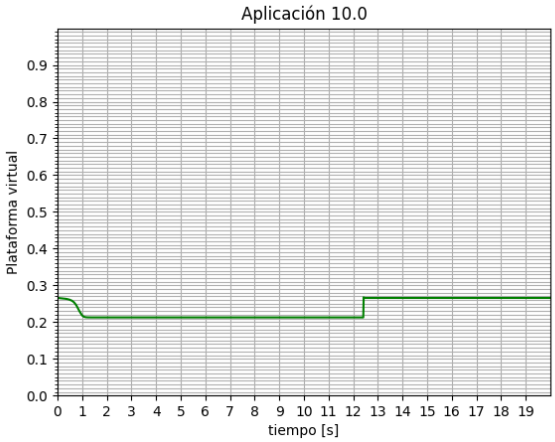
(d) Aplicación 7



(e) Aplicación 8



(f) Aplicación 9



(g) Aplicación 10

Figura 6.4: Comportamiento de las plataformas virtuales a lo largo del tiempo para el caso en que se finaliza una aplicación. El color de las gráficas indica el procesador donde se han ejecutado, el valor de las plataformas virtuales está normalizada respecto al dispositivo de capacidad de computo mayor

Aplicación	Raspberry	C	v	C*v	Nivel de servicio inicial
1	1	1	N/A	N/A	N/A
2	N/A	1	N/A	N/A	3.8
3	3	0.8	0.333	0.266	2.3
4	2	0.9	1	0.9	3.7
5	2	0.9	N/A	N/A	N/A
6	3	0.8	N/A	N/A	N/A
7	3	0.8	0.33	0.266	1.9
8	1	1	0.5	0.5	3.0
9	1	1	0.5	0.5	2.3
10	3	0.8	0.33	0.266	2.4

En la tabla, se observa que los recursos asignados no se distribuyen proporcionalmente con respecto a los niveles de servicio iniciales. Si deseamos mantener la distribución de recursos, podemos incrementar proporcionalmente los niveles de servicio, de modo que sean mayor que las plataformas virtuales, y así restablecer el equilibrio respetando la proporcionalidad inicial entre las plataformas virtuales.

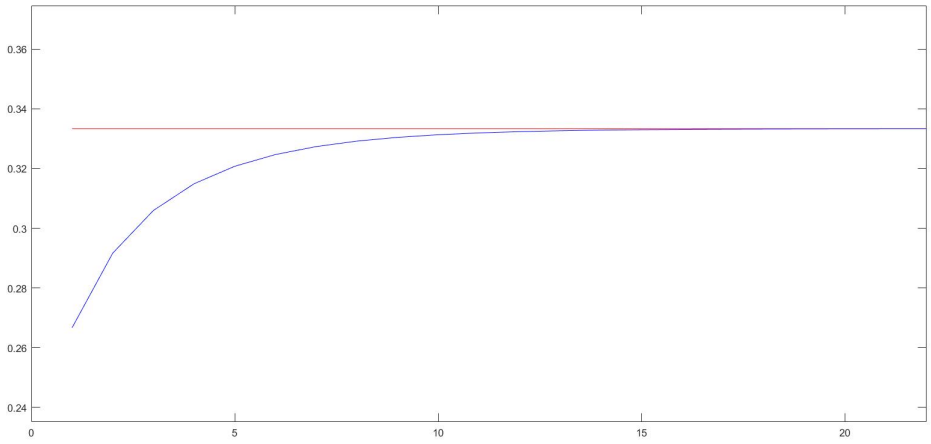


Figura 6.5: Comportamiento del control para tres aplicaciones cuando los niveles de servicio son menores que las plataformas virtuales. En rojo las plataformas virtuales inician en 1/3, en azul los niveles de servicio, inician en 0.2

6.4. Operación con cambio en el nivel de servicio

En esta prueba se modifico el contenido de la aplicación ocho para que cuando se ejecute al menos 2.5 segundos, su nivel de servicio cambie a 2. La Figura (6.6) muestra la operación del RM.

Las siguientes son características particulares de la prueba que se realizo en esta sección, estas características se pueden ver en las Figuras (6.6) y (6.7).

- A diferencia de la sección anterior, en las gráficas en (6.7) se puede observar la acción del control después del cambio en la distribución.
- A los 9.29[s] ocurre el cambio del nivel de servicio de la aplicación ocho, comienza una nueva distribución de las aplicaciones
- El algoritmo que distribuye las aplicaciones decidió que la aplicación ocho con nivel de servicio 2 compartiera procesador con la aplicación diez con nivel de servicio sin normalizar de 0.25
- Se observa que únicamente la aplicación nueve cambio de dispositivo
- El valor final del nivel de servicio de la aplicación ocho, sin normalizar, es de 0.762 y el de la aplicación diez es de 0.238

- En las gráficas de las plataformas virtuales Figura 6.7 , después del segundo 9.29[s], se observa el cambio de las plataformas virtuales hasta alcanzar el equilibrio, a diferencia de la sección anterior.

- Una vez realizada la nueva distribución, se observa que el mayor cambio en las plataformas virtuales ocurre en las aplicaciones diez y ocho. Esto se debe a que la aplicación ocho utiliza la mayor parte de los recursos del procesador.

- La aplicación cuatro es la primera en ser ejecutada utilizando los valores de las plataformas virtuales obtenidos después de la distribución, se ejecuta a los 11.02[s]. Debido a que el control comienza a calcular las plataformas virtuales en t=9.29[s], y la primera aplicación se inicia en t=11.02[s], las aplicaciones se ejecutan con el tiempo asignado por el punto de equilibrio.

Aplicación	Raspberry	C	v	C*v	Nivel de servicio
1	1	1	N/A	N/A	N/A
2	1	1	0.57	0.57	3.8
3	2	0.9	0.28	0.252	2.3
4	2	0.9	0.45	0.405	3.7
5	2	0.9	N/A	N/A	N/A
6	3	0.8	N/A	N/A	N/A
7	1	1	0.43	0.43	1.9
8	3	0.8	0.76	0.608	3.0
9	2	0.9	0.27	0.243	2.3
10	3	0.8	0.24	0.192	2.4

En la tabla, se puede observar que la aplicación ocho, que incrementó su nivel de servicio, es la que termina recibiendo la mayor cantidad de recursos. En cuanto a las demás aplicaciones, la mayoría respeta la proporción de los niveles de servicio iniciales, y cuando no se respeta, la diferencia entre los recursos asignados no es tan grande. Al analizar la teoría utilizada en la distribución de aplicaciones, se nota que solo en el

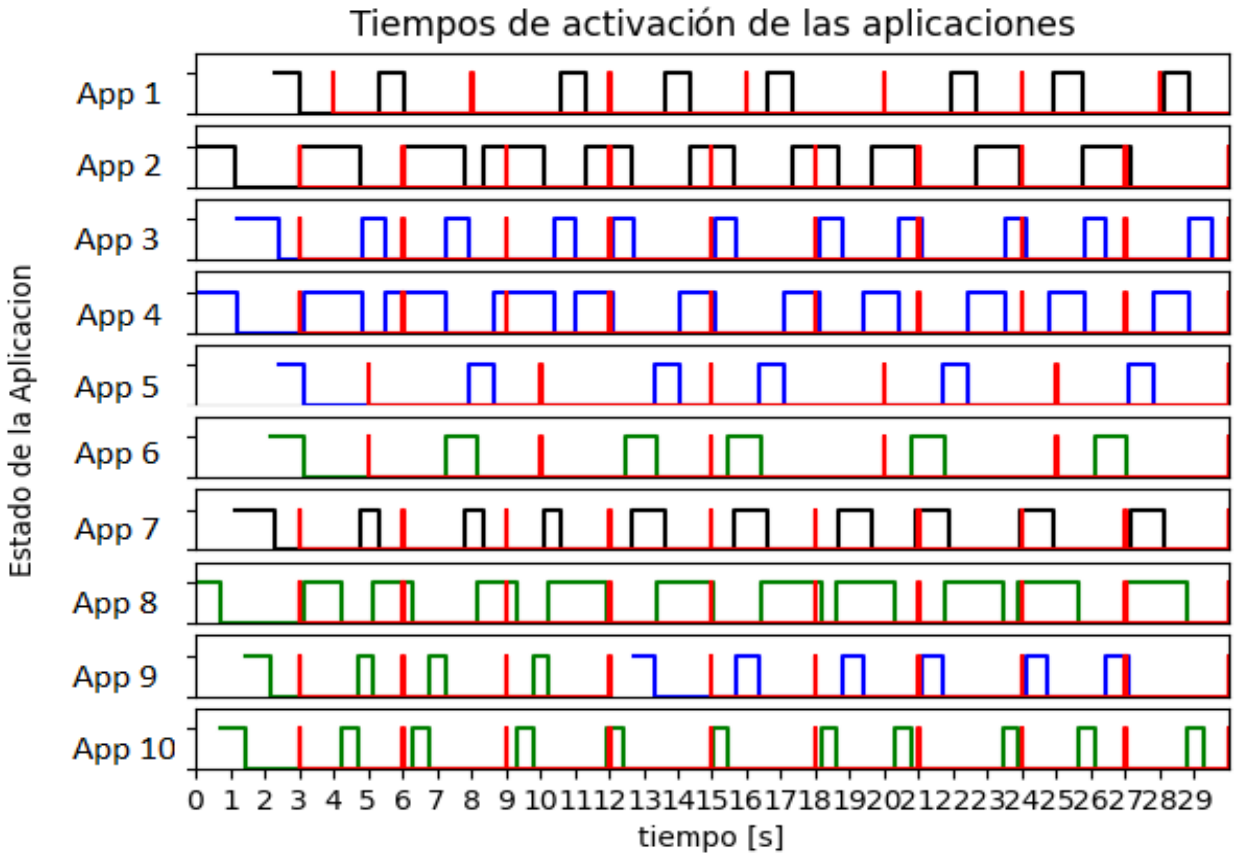
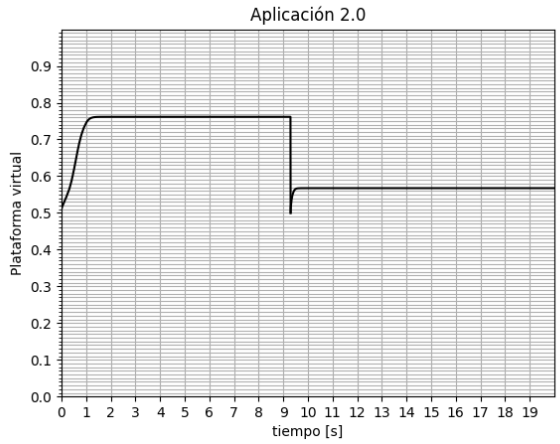
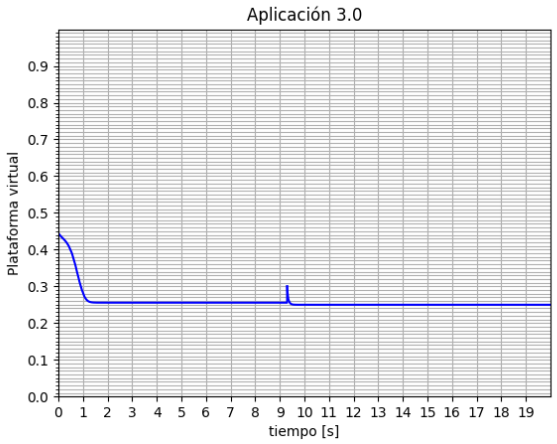


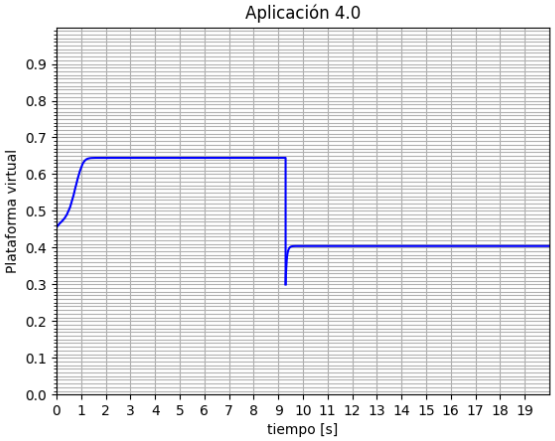
Figura 6.6: La figura pertenece al caso en que se modifica el nivel de servicio de una aplicación, se muestra los instantes de tiempo en que las aplicaciones fueron activadas y el tiempo que permanecieron activas. El color indica en que procesador se ejecuta la aplicación. Las lineas rojas representan los deadlines.



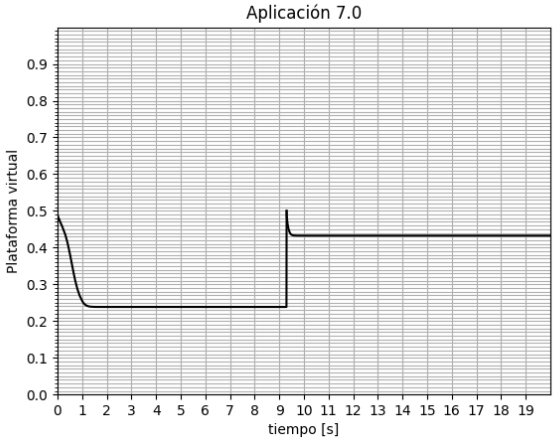
(a) Aplicación 2



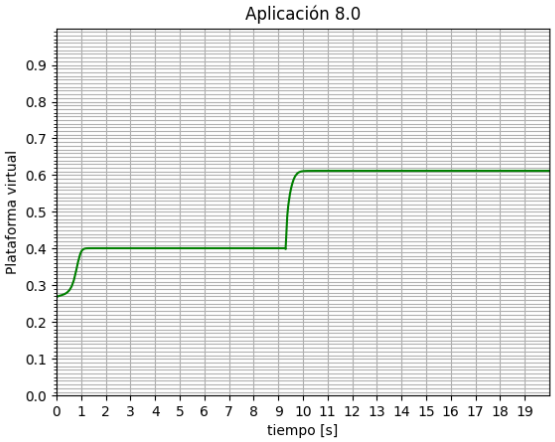
(b) Aplicación 3



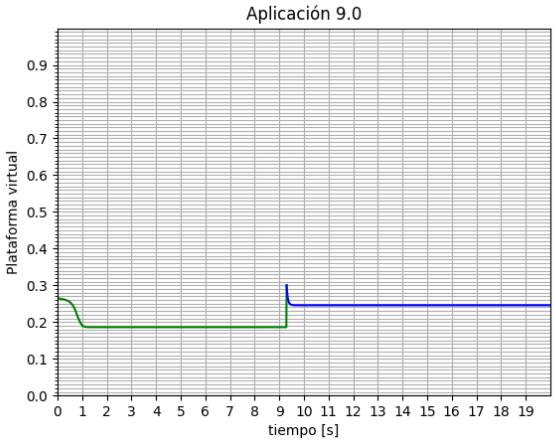
(c) Aplicación 4



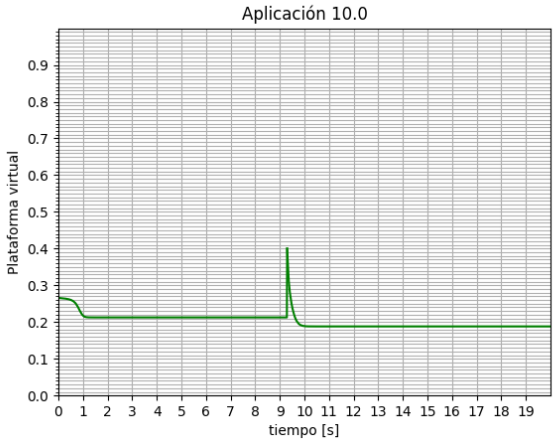
(d) Aplicación 7



(e) Aplicación 8



(f) Aplicación 9



(g) Aplicación 10

Figura 6.7: Comportamiento de las plataformas virtuales en el caso donde se modifica el nivel de servicio de una aplicación. El color de las gráficas indica el procesador donde se han ejecutado, el valor de las plataformas virtuales está normalizada respecto al dispositivo de capacidad de computo uno

caso ideal se mantienen las proporciones de recursos. En ocasiones, se invierten estas proporciones para aprovechar mejor las capacidades de los procesadores.

6.5. Cambio en nivel de servicio y finalización de tarea

En esta sección se muestra el resultado de la combinación de ambos factores. Nuevamente la aplicación dos termina cuando se ha ejecutado al menos 8[s] y el nivel de servicio de la aplicación ocho se actualiza cuando se ha ejecutado al menos 2.5[s]

Las siguientes son características particulares de la prueba que se realizo en esta sección, estas características se pueden ver en las Figuras (6.8) y (6.9)

- El cambio en la distribución debido a el nivel de servicio de la aplicación ocho se presenta a los 9.11[s]
- El cambio en la distribución debido a la finalización de la tarea dos se da a los 12.18[s]

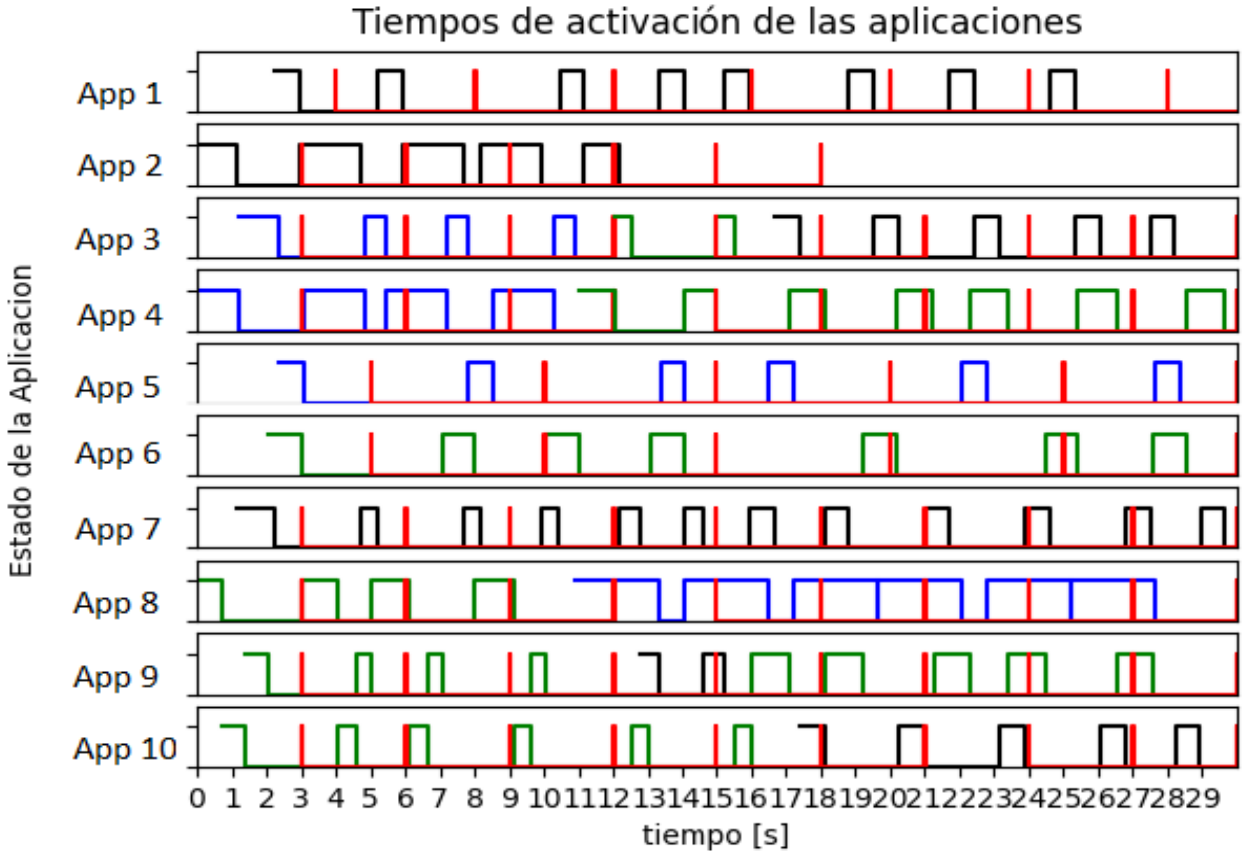
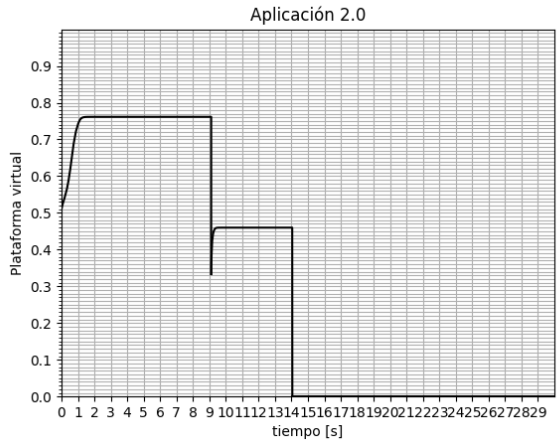
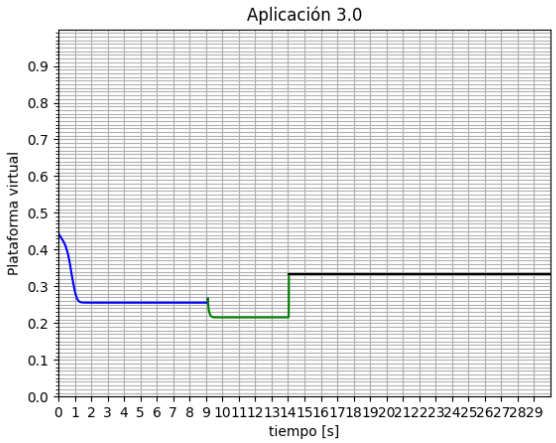


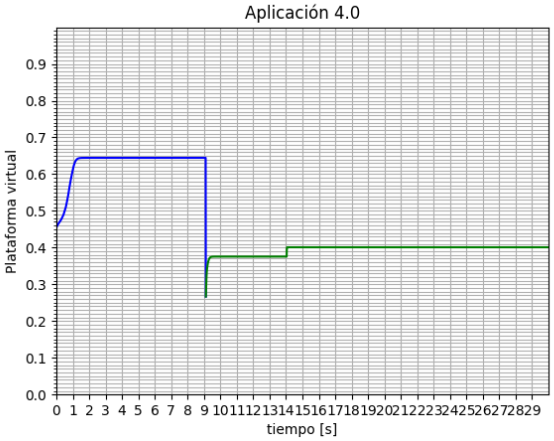
Figura 6.8: La figura muestra el caso donde una aplicación se finaliza y otra modifica su nivel de servicio, se muestra los instantes de tiempo en que las aplicaciones fueron activadas y el tiempo que permanecieron activas. El color indica en que procesador se ejecuta la aplicación. Las lineas rojas indican los deadlines.



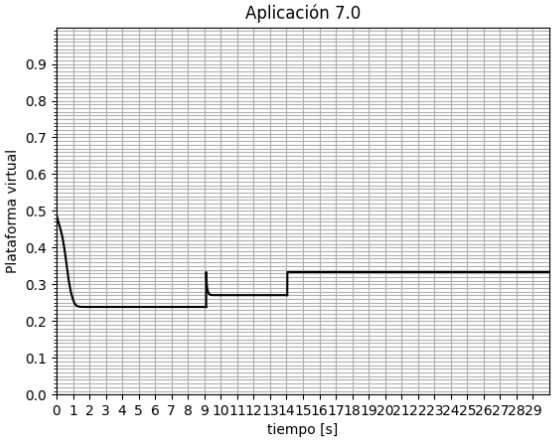
(a) Aplicación 2



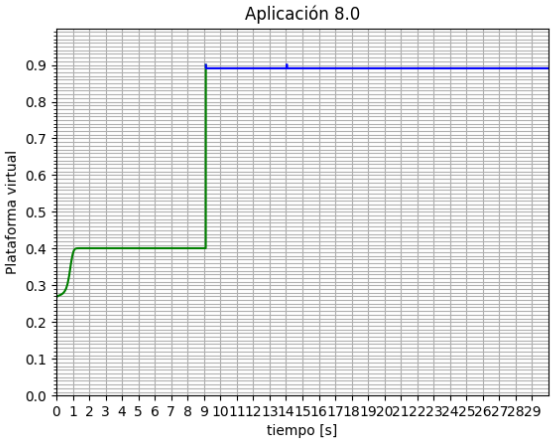
(b) Aplicación 3



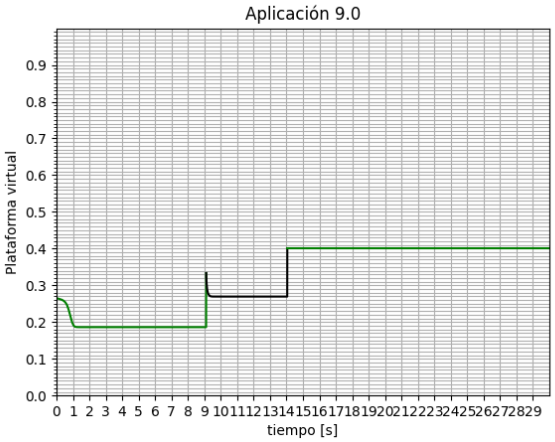
(c) Aplicación 4



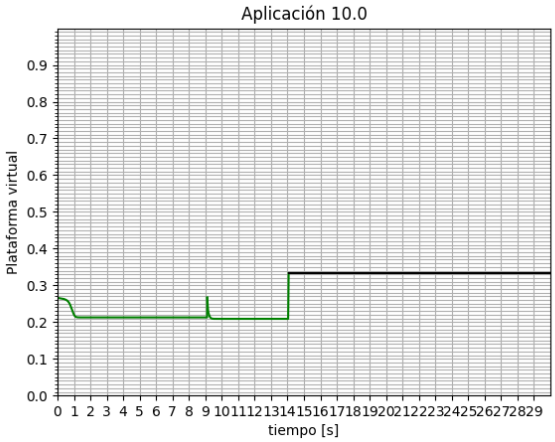
(d) Aplicación 7



(e) Aplicación 8



(f) Aplicación 9



(g) Aplicación 10

Figura 6.9: Comportamiento de las plataformas virtuales en el caso donde una aplicación termina su ejecución y otra cambia su nivel de servicio. El color de las gráficas indica el procesador donde se han ejecutado, el valor de las plataformas virtuales está normalizada respecto al dispositivo de capacidad de computo mayor.

- La primera aplicación en ejecutarse con los valores posteriores a la primera distribución es la aplicación ocho en el segundo 10.92[s]

- La primera aplicación en ejecutarse con los valores posteriores a la segunda distribución es la aplicación siete a los 15.95[s]

- La aplicación dos debería ejecutarse una vez más, pero el tiempo necesario para su ejecución es menor al tiempo que tarda la comunicación en decodificar los datos recibidos de la raspberry, por lo que se omite la ejecución. Debido a esto, del paquete formado por las aplicaciones dos y siete, solamente se ejecuta la aplicación siete en su sexta ejecución.

- Se puede ver en las gráficas de las plataformas virtuales Figura (6.9) que el cambio en la distribución se da a los 14.17[s] mientras que en la Figura (6.8) pareciera que la distribución debería ser antes, la razón se explico en el punto anterior

- En esta ocasión a la aplicación ocho, al momento de cambiar su nivel de servicio, se le asignó un dispositivo exclusivamente para su uso.

- En las gráficas de las plataformas virtuales se puede ver los dos comportamientos del control presentados en las secciones anteriores
- En la aplicación ocho en los dos momentos que se realiza la distribución se puede ver que disminuye un poco el valor de la plataforma virtual respecto su máximo, puede deberse al redondeo o al truncamiento realizado en el cálculo de las plataformas virtuales

6.6. Resumen de capítulo

En este capítulo se presentan cuatro pruebas realizadas al RM las cuales son:

- Sin cambio en la ejecución
- Finalización de tarea
- Cambio de nivel de servicio
- Cambio de nivel de servicio y finalización de tarea

Se analizan los datos obtenidos y se profundiza en algunos casos especiales que se observan en los resultados.

Capítulo 7

Conclusiones y trabajo futuro

- El manejador de recursos presentado en este trabajo cumple con su propósito de repartir los recursos de manera justa o proporcional a los requerimientos de las aplicaciones.

- Tiene la capacidad de adaptarse a nuevos requerimientos de las aplicaciones o a nuevas cantidades de aplicaciones y procesadores.

- También puede adaptarse a los cambios de las aplicaciones tipo *hard*.

- Como el tiempo requerido para que el control llegue al equilibrio es menor que el tiempo requerido para la ejecución de las aplicaciones, rara vez se ejecutarán las aplicaciones fuera de la repartición equitativa de recursos.

- Tiene la ventaja de que requiere cortos periodos de tiempo para su ejecución, por lo que una interrupción en la red no presenta una perdida grande.

- Tiene la ventaja de que la parte de mayor costo computacional se puede paralelizar.

- Aunque se está usando la teoría en la repartición de recursos en computadoras, es posible adaptarlo a otros ámbitos como por ejemplo la repartición de transporte en el MetroBus.

- Aunque se usó la combinación de MATLAB y Python en el trabajo, se puede usar únicamente Python evitando el tener que pagar alguna licencia o pudiendo adaptar el RM a dispositivos que no puedan correr MATLAB.

- Tiene la posibilidad de dividir la ejecución de una aplicación en dos, una parte tipo hard y otra tipo soft. Es decir, si una aplicación requiere ser ejecutada cierto tiempo cada deadline pero le es conveniente ejecutarse un poco más se podría hacer la división sin problemas.

Este trabajo puede ser ampliado de diversas formas. A continuación se presentan algunos casos:

- El RM tiene la ventaja de poder anexar nuevas aplicaciones o procesadores sin mayores complicaciones.

- Aunque el incrementar el número de aplicaciones incrementa el número de operaciones de manera exponencial, con los métodos para reducir tiempos de ejecución y usando dos o más manejadores de recursos en paralelo en vez de uno solo puede mejorar bastante los tiempos de ejecución.

- En el trabajo presente se propuso las aplicaciones tipo *hard* ligadas a un procesador, se podría buscar la manera de distribuirlas sin que se pierdan sus *deadline*. A demás se podrían ejecutar en al menos dos procesadores a la vez para tener robustez y no se pierda la información si hay algún problema con la red.

- Una limitante que se tuvo es el tiempo que requiere la comunicación, esta se puede mejorar si se encuentra otro método de comunicación, o ya que las aplicaciones tipo *soft* en un mismo procesador se tienen que ejecutar juntas, es posible crear una aplicación en los procesadores que reciba y envíe toda la información a la vez, además de ser quien mande a ejecutar a las aplicaciones, de manera que serán menores los tiempos perdidos por la comunicación, esto puede reducir los periodos de tiempo que requieren los programas en ejecutarse.

Apéndice A

Prueba de hipótesis

A.1. Problema

Si en un procesador cambia el número de aplicaciones que está ejecutando o si entra un nuevo procesador a la red, se debe calcular nuevamente las matrices obtenidas por medio del algoritmo de las LMI. El tiempo del calculo de las LMI lleva desde unos minutos a horas según el número de aplicaciones.

Se plantea demostrar que las matrices obtenidas para el procesador con mayor capacidad se pueden usar para los demás procesadores. Además también se demuestra que los renglones de la matriz $G_{i,j}$ son independientes entre ellos, por lo que a partir de la matriz $G_{i,j}$ correspondiente a unas pocas aplicaciones se puede obtener las matrices $G_{i,j}$ para cualquier número de aplicaciones.

El planteamiento realizado en este apéndice se basa en la teoría presentada en Aparicio-Santos et al. (2021). Sin embargo, se observará que el resultado obtenido con el análisis de este apéndice es similar a realizar el RM considerando que cada procesador es un sistema independiente de los demás.

A.2. Demostración de la hipótesis

El punto de equilibrio de un sistema difuso es global y asintóticamente estable si existe una matriz positiva definida P, tal que:

$$G_{ii}^T P G_{ii} - P < 0 \tag{A.1}$$

El objetivo es probar que la anterior desigualdad se cumple para $0 < \bar{q} < 1$ usando el conjunto de matrices obtenidas para $\bar{q} = 1$. Sabemos que se cumple para $\bar{q} = 1$ debido a que es el caso que se calcula con el *LMI solver* de MATLAB. El análisis se hace para cuando el procesador tiene 3 aplicaciones pero es el mismo para cualquier número de aplicaciones.

De la desigualdad (A.1) se tiene que:

$$G_{ii} = A_i - B_i F_i \tag{A.2}$$

$$A_i = \begin{bmatrix} z_1 & 0 & 0 & z_4 & 0 & 0 \\ 0 & z_2 & 0 & 0 & z_5 & 0 \\ 0 & 0 & z_3 & 0 & 0 & z_6 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

$$B_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

$$F_i = \begin{bmatrix} f_1 & 0 & 0 & f_4 & 0 & 0 \\ 0 & f_2 & 0 & 0 & f_5 & 0 \\ 0 & 0 & f_3 & 0 & 0 & f_6 \end{bmatrix} \tag{A.5}$$

Las matrices F_i son obtenidas por medio del algoritmo que calcula las LMI, se sabe por experiencia que siempre tienen la misma forma.

Se ha observado que la matriz P es simétrica, todos sus elementos son positivos y tiene la siguiente forma:

$$P = \begin{bmatrix} p_{1,1} & 0 & 0 & p_{1,4} & 0 & 0 \\ 0 & p_{2,2} & 0 & 0 & p_{2,5} & 0 \\ 0 & 0 & p_{3,3} & 0 & 0 & p_{3,6} \\ p_{1,4} & 0 & 0 & p_{4,4} & 0 & 0 \\ 0 & p_{2,5} & 0 & 0 & p_{5,5} & 0 \\ 0 & 0 & p_{3,6} & 0 & 0 & p_{6,6} \end{bmatrix} \tag{A.6}$$

La matriz G_{ii} la podemos reescribir para evitar los signos y separar el valor de $\bar{q}$:

$$G_{ii} = \begin{bmatrix} g_{1,1} & 0 & 0 & g_{1,4}\bar{q}_h & 0 & 0 \\ 0 & g_{2,2} & 0 & 0 & g_{2,5}\bar{q}_h & 0 \\ 0 & 0 & g_{3,3} & 0 & 0 & g_{3,6}\bar{q}_h \\ g_{4,1} & 0 & 0 & g_{4,4} & 0 & 0 \\ 0 & g_{5,2} & 0 & 0 & g_{5,5} & 0 \\ 0 & 0 & g_{6,3} & 0 & 0 & g_{6,6} \end{bmatrix} \tag{A.7}$$

Reordenamos la desigualdad (A.1), premultiplicamos por x^T y posmultiplicamos por x

$$x^T G^T P G x < x^T P x \tag{A.8}$$

Donde x es un vector de dimensión 6x1 cuyos elementos pertenecen a los reales. Si para algún conjunto de valores de x no se cumple la desigualdad entonces la hipótesis que se intenta probar es falsa.

Realizamos las multiplicaciones de las matrices:

$$Gx = \begin{bmatrix} g_{1,1}x_1 + g_{1,4}\bar{q}_h x_4 \\ g_{2,2}x_2 + g_{2,5}\bar{q}_h x_5 \\ g_{3,3}x_3 + g_{3,6}\bar{q}_h x_6 \\ g_{4,1}x_1 + g_{4,4}x_4 \\ g_{5,2}x_2 + g_{5,5}x_5 \\ g_{6,3}x_3 + g_{6,6}x_6 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \\ w_{41} \\ w_{51} \\ w_{61} \end{bmatrix} \tag{A.9}$$

$$PGx = \begin{bmatrix} p_{1,1}w_{1,1} + p_{1,4}w_{4,1} \\ p_{2,2}w_{2,1} + p_{2,5}w_{5,2} \\ p_{3,3}w_{3,1} + p_{3,6}w_{6,3} \\ p_{1,4}w_{1,1} + p_{4,4}w_{4,1} \\ p_{2,5}w_{2,1} + p_{5,5}w_{5,1} \\ p_{3,6}w_{3,1} + p_{6,6}w_{6,1} \end{bmatrix} \tag{A.10}$$

$$\begin{aligned} x^T G^T PGx = {} & w_{1,1}p_{1,1}w_{1,1} + w_{1,1}p_{1,4}w_{4,1} + w_{4,1}p_{1,4}w_{1,1} + w_{4,1}p_{4,4}w_{4,1} \\ & + w_{2,1}p_{2,2}w_{2,1} + w_{2,1}p_{2,5}w_{5,2} + w_{5,2}p_{2,5}w_{2,1} + w_{5,2}p_{5,5}w_{5,1} \\ & + w_{3,1}p_{3,3}w_{3,1} + w_{3,1}p_{3,6}w_{6,3} + w_{6,3}p_{3,6}w_{3,1} + w_{6,3}p_{6,6}w_{6,1} \end{aligned} \tag{A.11}$$

En la ecuación (A.11) podemos notar que los términos en el primero, segundo y tercer renglón son independientes entre si. Se puede hacer el análisis para los elementos del primer renglón y será el mismo análisis para lo demás. Se analizara para el caso del primer renglón, con las variables x_1 y x_4, las demás variables de x serán cero.

De la ecuación (A.11) se obtiene:

$$p_{1,1}w_{1,1}^2 + 2p_{1,4}w_{11}w_{4,1} + p_{4,4}w_{4,1}^2 < p_{1,1}x_1^2 + 2p_{1,4}x_1x_4 + p_{4,4}x_4^2 \tag{A.12}$$

Sustituyendo las variables $w_{1,1}$ y $w_{4,1}$:

$$\begin{aligned}(p_{1,1}g_{1,1}^2 + 2p_{1,4}g_{1,1}g_{4,1} + p_{4,4}g_{4,1}^2)x_1^2 + \\ (2p_{1,1}g_{1,1}g_{1,4}\bar{q}_h + 2p_{1,4}g_{1,1}g_{4,4} + 2p_{1,4}g_{1,4}g_{4,1}\bar{q}_h + 2p_{4,4}g_{4,1}g_{4,4})x_1x_4 + \\ (p_{1,1}g_{1,4}^2\bar{q}_h^2 + 2p_{1,4}g_{1,4}g_{4,4}\bar{q}_h + p_{4,4}g_{4,4}^2)x_4^2 < p_{1,1}x_1^2 + 2p_{1,4}x_1x_4 + p_{4,4}x_4^2 \quad \text{(A.13)}\end{aligned}$$

Podemos ver que la desigualdad se puede dividir en tres desigualdades:

$$(p_{1,1}g_{1,1}^2 + 2p_{1,4}g_{1,1}g_{4,1} + p_{4,4}g_{4,1}^2)x_1^2 < p_{1,1}x_1^2 \quad \text{(A.14)}$$

$$(2p_{1,1}g_{1,1}g_{1,4}\bar{q}_h + 2p_{1,4}g_{1,1}g_{4,4} + 2p_{1,4}g_{1,4}g_{4,1}\bar{q}_h + 2p_{4,4}g_{4,1}g_{4,4})x_1x_4 < 2p_{1,4}x_1x_4 \quad \text{(A.15)}$$

$$(p_{1,1}g_{1,4}^2\bar{q}_h^2 + 2p_{1,4}g_{1,4}g_{4,4}\bar{q}_h + p_{4,4}g_{4,4}^2)x_4^2 < p_{4,4}x_4^2 \quad \text{(A.16)}$$

Si $x_4 = 0$ solo queda la desigualdad (A.14). Como $g_{1,1}$ puede tener el valor de 1 y todos los términos son positivos salvo $g_{4,1}$ que se desconoce, implica que $g_{4,1} < 0$

Si $x_1 = 0$ solo queda la desigualdad (A.16). Sabemos que cuando $\bar{q}_h = 1$ se cumple la desigualdad debido a que se calcula con el algoritmo de las LMI. También cumple la desigualdad para $0 < \bar{q}_h < 1$ si $|g_{4,4}| < 1$ ya que $\bar{q}_h$ afecta disminuyendo la parte positiva del lado izquierdo de la desigualdad:

$$\bar{q}_h(p_{1,1}g_{1,4}^2\bar{q}_h + 2p_{1,4}g_{1,4}g_{4,4}) < p_{4,4}(1 - g_{4,4}^2) \quad \text{(A.17)}$$

La desigualdad (A.15) no siempre se cumple, pero se puede verificar que no afecte el signo de la desigualdad (A.13) si cumple con:

$$a^2x^2 + b^2y^2 \geq 2|ab|xy > 2|c|xy \quad \text{(A.18)}$$

Reescribimos la desigualdad (A.13) de la siguiente manera:

$$(p_{1,1}-(p_{1,1}g_{1,1}^2+2p_{1,4}g_{1,1}g_{4,1}+p_{4,4}g_{4,1}^2))x_1^2+(p_{4,4}-(p_{1,1}g_{1,4}^2\bar{q}_h^2+2p_{1,4}g_{1,4}g_{4,4}\bar{q}_h+p_{4,4}g_{4,4}^2))x_4^2$$
$$> ((2p_{1,1}g_{1,1}g_{1,4}\bar{q}_h + 2p_{1,4}g_{1,1}g_{4,4} + 2p_{1,4}g_{1,4}g_{4,1} + 2p_{4,4}g_{4,1}g_{4,4}) - 2p_{1,4})x_1x_4 \quad \text{(A.19)}$$

Con este nuevo orden se puede ver que:

$$a^2 = p_{1,1} - (p_{1,1}g_{1,1}^2 + 2p_{1,4}g_{1,1}g_{4,1} + p_{4,4}g_{4,1}^2) \quad \text{(A.20)}$$
$$b^2 = p_{4,4} - (p_{1,1}g_{1,4}^2\bar{q}_h^2 + 2p_{1,4}g_{1,4}g_{4,4}\bar{q}_h + p_{4,4}g_{4,4}^2) \quad \text{(A.21)}$$
$$2c = (2p_{1,1}g_{1,1}g_{1,4}\bar{q}_h + 2p_{1,4}g_{1,1}g_{4,4} + 2p_{1,4}g_{1,4}g_{4,1} + 2p_{4,4}g_{4,1}g_{4,4}) - 2p_{1,4} \quad \text{(A.22)}$$

Donde ya demostramos que a^2 y b^2 son positivas y reales, solo faltaría demostrar:

$$|ab| > |c| \quad \text{(A.23)}$$

Que podemos elevar al cuadrado sin afectar el signo y reordenar para que quede:

$$a^2b^2 - c^2 > 0 \quad \text{(A.24)}$$

Sustituyendo los valores de a,b y c; y hacemos la factorización respecto a $\bar{q}_h$

$$(g_{4,1}^2p_{1,1}p_{4,4} - g_{4,1}^2p_{1,4}^2 - p_{1,1}^2)g_{1,4}^2\bar{q}_h^2$$
$$+ (2g_{4,1}p_{1,4}^2 + 2g_{1,1}p_{1,1}p_{1,4} - 2g_{4,4}p_{1,1}p_{1,4} + 2g_{1,1}g_{4,1}g_{4,4}p_{1,4}^2 - 2g_{1,1}g_{4,1}g_{4,4}p_{1,1}p_{4,4})g_{1,4}\bar{q}_h$$
$$- g_{1,1}^2g_{4,4}^2p_{1,4}^2 + p_{1,1}g_{1,1}^2g_{4,4}^2p_{4,4} - p_{1,1}g_{1,1}^2p_{4,4} - 2g_{1,1}g_{4,1}p_{1,4}p_{4,4} + 2g_{1,1}g_{4,4}p_{1,4}^2 - g_{4,1}^2p_{4,4}^2$$
$$+ 2g_{4,1}g_{4,4}p_{1,4}p_{4,4} - p_{1,1}g_{4,4}^2p_{4,4} - p_{1,4}^2 + p_{1,1}p_{4,4} > 0 \quad \text{(A.25)}$$

Podemos ver que es un polinomio de segundo grado respecto a $\bar{q}_h$, nos interesa saber los valores de $\bar{q}_h$ para los que se cumple la desigualdad anterior, por lo que buscaremos las raíces del polinomio. Sabemos que $\bar{q}_h = 1$ cumple con la desigualdad,

por lo que los valores de $\bar{q}_h$ que cumplen con la desigualdad se encuentran en el intervalo de valores de $\bar{q}_h$ que contiene a $\bar{q}_h = 1$ y que está limitado por las raíces reales de $\bar{q}_h$ si estas existen.

Por lo que se demuestra que la desigualdad (A.14) se cumple, (A.16) se cumple y al hacer la suma de (A.14),(A.15) y (A.16) los términos de (A.15) no afecta el sentido de la desigualdad, por tanto se cumple la desigualdad (A.13) para un conjunto alrededor de $\bar{q}_h = 1$

A.2.1. Ejemplo

Se realiza el análisis para una de las matrices G_{ii} para el caso en que se tiene tres plataformas virtuales. La matriz G_{ii} elegida tiene los dos posibles casos en que el valor $\bar{q}_h$ afecta a la desigualdad.

$$G_{ii} = \begin{bmatrix} 1 & 0 & 0 & 0.1\bar{q}_h & 0 & 0 \\ 0 & 0.7 & 0 & 0 & 0.1\bar{q}_h & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -0.3112 & 0 & 0 & -0.0321 & 0 & 0 \\ 0 & -0.2180 & 0 & 0 & -0.03119 & 0 \\ 0 & 0 & -0.3113 & 0 & 0 & -0.0011 \end{bmatrix} \tag{A.26}$$

$$P = \begin{bmatrix} 0.0506 & 0 & 0 & 0.0124 & 0 & 0 \\ 0 & 0.0506 & 0 & 0 & 0.0124 & 0 \\ 0 & 0 & 0.0506 & 0 & 0 & 0.0124 \\ 0.0124 & 0 & 0 & 0.0398 & 0 & 0 \\ 0 & 0.0124 & 0 & 0 & 0.0398 & 0 \\ 0 & 0 & 0.0124 & 0 & 0 & 0.0398 \end{bmatrix} \tag{A.27}$$

Para $x = [x_1\ 0\ 0\ x_4\ 0\ 0]^T$ la ecuación (A.24) es:

$$(-2.3765)(10^{-5})\bar{q}_h^2 + (1.1596)(10^{-4})\bar{q}_h - (1.5090)(10^{-7}) > 0 \tag{A.28}$$

Las raíces son:

$$\bar{q}_h = 4.878 \qquad \bar{q}_h = 0.0013 \tag{A.29}$$

Sabemos que la desigualdad se cumple para $\bar{q}_h = 1$ por lo que el rango donde se cumple es (0.0013,4.878).

Se realiza el análisis para $x = [\ 0\ x_2\ 0\ 0\ x_5\ 0]^T$, la ecuación (A.24) es:

$$-0.0025\bar{q}_h^2 + (8.3137)(10^{-4})\bar{q}_h + (9.4733)(10^{-4}) > 0 \tag{A.30}$$

Las raíces son:

$$\bar{q}_h = -4.7359 \qquad \bar{q}_h = 8.1042 \tag{A.31}$$

El caso $x = [\ 0\ 0\ x_3\ 0\ 0\ x_6\ ]^T$ no depende de $\bar{q}_h$ por lo que se sabe que cumple.

Entonces todos los procesadores que sean $\bar{q}_h > 0.0013$ cumplen la desigualdad (A.1)

Los valores limites de $\bar{q}_h$ están definidos por las matrices F que entrega el algoritmo de las LMI.

Al poder usar las matrices para $\bar{q} = 1$ en $\bar{q}_h > 0.0013$ puede darse casos como el siguiente.

Se han calculado dos plataformas virtuales con el mismo conjunto de matrices, $v_1 = 1$ y $v_2 = 1$, sin embargo v_1 se ejecuta en un procesador con $\bar{q} = 1$ y v_2 en un procesador con $\bar{q} = 0.5$, por lo que aún teniendo el mismo valor realmente no representan la misma capacidad de procesamiento para un procesador virtual.

Puede asumirse que cada procesador tiene una capacidad de procesamiento de $\bar{q} = 1$ y ver el efecto de su capacidad de procesamiento real solamente cuando se emigra una aplicación a otro procesador.

A.3. Consideraciones que se deben de cumplir

- G_{ii} debe tener la forma de (A.7)
- P debe tener la forma de (A.6)
- Los elementos de la matriz P son todos positivos
- $|g_{4,4}| < 1$

Si no se cumplen los puntos no se demuestra que no se cumpla la desigualdad (A.1), se tendría que hacer otro analizar de la desigualdad.

A.4. Reducción de costo de procesamiento de las LMI

Cada una de las matrices G_{ii} debe ser guardada en memoria. En la sección anterior, se demuestra que solo es necesario guardar las matrices del caso con $\bar{q}_h = 1$ sin embargo aún se debe guardar un conjunto de matrices de tamaño 2^{2n} desde n=1 hasta el máximo número de aplicaciones.

El cálculo de las LMI también se ve afectado por el número de aplicaciones, llegando a un punto que en una computadora convencional no puede correr el algoritmo de las LMI. Usaremos la teoría de la sección pasada para obtener las matrices G_{ii} sin calcular las LMI

En la matriz G_{ii} en (A.7) cada par de renglones x y n+x son independientes de los demás renglones, donde x es el número del renglón y n es el número de aplicaciones, se puede plantear que en una matriz G_{ii}, que puede ser de diferente dimensión, si uno de sus renglones tiene los mismos valores distintos de cero que el renglón x de una matriz conocida, entonces se puede usar los valores distintos de cero del renglón n+x de la matriz conocida para completar la matriz G_{ii}.

Así por ejemplo, usando la matriz G_{ii} del ejemplo de la sección anterior, podemos tener la matriz:

$$G_{ii} = \begin{bmatrix} 1 & 0 & 0.1\bar{q}_h & 0 \\ 0 & 0.7 & 0 & 0.1\bar{q}_h \\ -0.3112 & 0 & -0.0321 & 0 \\ 0 & -0.2180 & 0 & -0.03119 \end{bmatrix} \tag{A.32}$$

que sería una matriz valida G_{ii} para dos aplicaciones a partir de los valores de la matriz G_{ii} para tres aplicaciones. Si los valores no nulos de los primeros dos renglones coinciden con los de la matriz del ejemplo de la sección anterior, asignamos los valores no nulos correspondientes de la matriz del ejemplo a los valores no nulos de los últimos dos renglones.

Sin embargo, tenemos el problema que algunos de los primeros n renglones de G_{ii} dependen del número de aplicaciones por lo que los valores nunca llegan a ser iguales. estos casos son:

$$g_{j,j} = 1 - n\epsilon \tag{A.33}$$

$$g_{j,j+n} = \epsilon \tag{A.34}$$

donde g es un valor de la matriz G_{ii}

En el trabajo se usa ϵ fijo para tener el máximo incremento fijo sin importar el número de aplicaciones, también podría dejarse fijo el valor $1 - n\epsilon$ y que ϵ fuera quien variara.

La manera de poder usar el recurso planteado en el inicio de esta sección, es demostrando que para un rango de valores de las variables g, modificando ya sea n o ϵ, se cumple la desigualdad (A.13) si se usan estos valores en una matriz donde los renglones x+n pertenecen a una matriz previamente calculada que cumple con la desigualdad (A.13).

La demostración para el caso con $1-n\epsilon$ fija es idéntica a la demostración planteada en la sección anterior. Para el caso con ϵ fija se demuestra a continuación.

Partimos de las desigualdades (A.14), (A.15) y (A.16). El termino que se ve afec-

tado por el cambio de n es $g_{1,1} = 1 - n\epsilon$, este termino no aparece en la desigualdad (A.16) así que la podemos omitir. En (A.14) se puede calcular una región donde sean validos los valores de n. Se usa (A.24) en lugar de la desigualdad (A.15) para encontrar una región en la que sea valida. La intersección de ambas regiones da los valores de n validos

A.4.1. Ejemplo

En el siguiente ejemplo se usan los valores usados en este trabajo.

$$P = \begin{bmatrix} 0.0969 & 0 & 0.0231 & 0 \\ 0 & 0.0969 & 0 & 0.0231 \\ 0.0231 & 0 & 0.07406 & 0 \\ 0 & 0.0231 & 0 & 0.07406 \end{bmatrix} \tag{A.35}$$

$$G = \begin{bmatrix} 1 - n\epsilon & 0 & 0.1 & 0 \\ 0 & 1 - n\epsilon & 0 & 0 \\ -0.2473 & 0 & -0.0324 & 0 \\ 0 & -0.2406 & 0 & 0.0025 \end{bmatrix} \tag{A.36}$$

Los renglones uno y dos de la matriz G son las condiciones que se deben cumplir donde $g_{i,i} = 1 - n\epsilon$ y $g_{i,i+n}$ tiene el valor de ϵ y 0.

Se sustituirán los valores correspondientes en las desigualdades (A.14) y (A.24), se obtendrán las raíces del polinomio, con las cuales podemos saber el intervalo donde se cumplen las desigualdades y n es valida

El polinomio de la desigualdad (A.14) para el primer renglón de G es:

$$-0.001n^2 + 0.0182n + 0.0069 \tag{A.37}$$

Las raíces del polinomio son [-0.3694, 19.1895]

El polinomio de la desigualdad (A.14) para el segundo renglón de G es:

$$-0.001n^2 + 0.0183n + 0.0068 \tag{A.38}$$

Las raíces del polinomio son [-0.3653, 19.2176]

El polinomio de la desigualdad (A.24) para el primer renglón de G es:

$$-0.0001n^2 + 0.0013n + 0.0003 \tag{A.39}$$

Las raíces del polinomio son [-0.2293, 18.4909]

El polinomio de la desigualdad (A.24) para el segundo renglón de G es:

$$-0.0001n^2 + 0.0013n + 0.0003 \tag{A.40}$$

Las raíces del polinomio son [-0.2292, 18.4571]

Como n solo puede tener valores enteros positivos el rango de valores factibles es desde $n = 1$ a $n = 18$

Apéndice B

Distribución de aplicaciones en los procesadores

B.1. Introducción

En la distribución de las aplicaciones se busca que exista un equilibrio entre la cantidad de recursos a usar del procesador y la capacidad del mismo, de manera que no se sature algún procesador o se desperdicie su capacidad.

B.2. Desarrollo

Si una aplicación necesita ejecutarse 1[s] en el procesador de mayor capacidad, en uno con la mitad de la capacidad requerirá 2[s]. De manera general el tiempo requerido para realizar la labor es:

$$\text{tiempo requerido} = \frac{\text{plataforma virtual}}{\text{capacidad de procesador}} \tag{B.1}$$

La distribución de las aplicaciones en los procesadores debe de ser de tal forma que el tiempo requerido para terminar la ejecución de las aplicaciones sea similar entre los procesadores. De lo contrario tendremos distribuciones donde unos procesadores terminan su trabajo quedando inactivos y otros tienen un exceso de trabajo y no

logran terminar a tiempo.

Como no se tiene los valores de las plataformas virtuales en el punto de equilibrio, se puede usar la multiplicación del nivel de servicio por la prioridad, esto es bajo la suposición que a un mayor nivel de servicio se le asigna una plataforma virtual mayor, la misma idea se aplica a la prioridad de las aplicaciones.

Supongamos ahora el caso en que tenemos dos aplicaciones, en el procesador de capacidad relativa 1 tardan 1[s] y 2[s] respectivamente, queremos repartirlas entre los procesadores con capacidad relativa 1 y 0.5, la mejor opción es colocar la aplicación de 2[s] en el procesador de capacidad 1 y la aplicación de 1[s] en el procesador de capacidad 0.5. Al hacer la división en (B.1) ambas dan un tiempo de 2[s].

De manera general, la distribución que buscamos se da cuando la división en (B.1) es aproximadamente igual para todos los procesadores.

$$\frac{\sum \lambda_i s_i}{p_1} \approx \frac{\sum \lambda_j s_j}{p_2} \approx \frac{\sum \lambda_k s_k}{p_n} \approx x \tag{B.2}$$

Donde las sumatorias usan los valores de $\lambda_i s_i$ de las aplicaciones que se ejecutan en el procesador p correspondiente.

Se puede usar la siguiente ecuación para realizar la distribución:

$$\frac{\lambda_1 s_1 + \lambda_2 s_2 + ... + \lambda_n s_n}{\sum p} = x \tag{B.3}$$

Lo que nos da la ecuación anterior es el valor al cual debe tender la ecuación (B.2), de esta forma se puede hacer un algoritmo que sume los valores λs de las aplicaciones para que se aproximen a x

Cuando (B.2) son igualdades se puede ver que la ecuación (B.3) es correcta por medio de la siguiente sustitución:

$$\frac{\lambda_1 s_1 + \lambda_2 s_2 + ... + \lambda_n s_n}{\sum p} = \frac{p_1 \frac{\sum \lambda_i s_i}{p_1} + p_2 \frac{\sum \lambda_j s_j}{p_2} + ... + p_m \frac{\sum \lambda_n s_n}{p_m}}{\sum p} = \frac{p_1 + p_2 + ... + p_m}{\sum p} x = x \tag{B.4}$$

Aun cuando no se cumpla la igualdad en (B.2) se busca que sus valores se aproximen a x, por lo que sigue siendo útil para realizar la distribución.

B.3. Ejemplo

Si tenemos las siguientes aplicaciones y procesadores:

apps	s	λ	λs
1	2	0.9	1.8
2	2.5	0.8	2
3	1.8	0.95	1.71
4	2.2	0.75	1.65
5	1.9	0.7	1.33

cpu	capacidad
1	0.8
2	0.6
3	1

Usando la ecuación (B.3) para obtener el valor de referencia al cual queremos aproximar

$$\frac{1.8 + 2 + 1.71 + 1.65 + 1.33}{1 + 0.8 + 0.6} = 3.5375 \tag{B.5}$$

La ecuación en (B.2) debe de aproximarse al valor anterior para cada procesador. Para este ejemplo la distribución queda de la siguiente manera

cpu	apps	aproximacion
1	4,5	3.725
2	2	3.3333
3	1,3	3.51

Aunque pensar en la distribución como tiempos de ejecución facilita el entendimiento de lo propuesto en este apéndice, en realidad, en este trabajo, el tiempo de ejecución de todas las aplicaciones en un procesador está definido por un periodo de ejecución que nosotros establecemos. Lo que realmente se ve afectado es la repartición de recursos.

Apéndice C

Truncamiento

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

$$G_{ij} = A_i - B_i F_j \tag{C.2}$$

C.0.1. Truncamiento de h

Definiremos una nueva variable que es la multiplicación de dos pesos h

$$H^{(n)} = h_i^{(n)} h_j^{(n)} \tag{C.3}$$

El superíndice se usa para distinguir entre truncamientos diferentes

- Necesitamos realizar el truncado de manera que no se trunquen todos los valores de H, recordando que la cantidad de variables h es 2^{2n} para esta tesis. Aprovechando que la suma de todas las h debe ser igual a uno y usando el caso cuando todas las h_i son iguales, por lo que no se pueden truncar, tenemos

$$h_i = h_j \tag{C.4}$$

$$h_1 + h_2 + ... + h_{2^{2n}} = 1 = 2^{2n} h_i \tag{C.5}$$

$$h_i^{(1)} = \frac{1}{2^{2n}} \tag{C.6}$$

$$2^{2n} h_i^{(1)} h_i^{(1)} = 2^{2n} H_{i,i}^{(1)} = h_i^{(1)} \tag{C.7}$$

$$H_{i,i}^{(1)} = \frac{h_i^{(1)}}{2^{2n}} = \frac{1}{2^{4n}} \tag{C.8}$$

- Si truncamos con un valor $T \leq H_{i,i}^{(1)}$ aseguramos que no se pierdan todos los valores de (C.1)
- Se llamará subsistema a los elementos $h_i h_j G_{i,j} x$ de (C.1)
- Cada subsistema que cumpla con $h_i h_j < T$ se omitirán de la suma en (C.1)

Realizaremos el truncamiento con la intención de que no se pierda por completo el 10 % del sistema. Para obtener el valor del truncamiento usaremos un sistema cuyo valor es el 10 % del sistema original, se analiza el caso cuando todas las h_i son iguales:

$$\begin{aligned} 0.1 \sum_{i=1} \sum_{j=1} hh(A_i - B_i F_j) x(t) = 0.1hh \sum_{i=1} \sum_{j=1} (A_i - B_i F_j) x(t) \\ = H^{(2)} \sum_{i=1} \sum_{j=1} (A_i - B_i F_j) x(t) \end{aligned} \tag{C.9}$$

$$H^{(2)} = 0.1hh = \frac{0.1}{2^{4n}} = h^{(2)} h^{(2)} \tag{C.10}$$

$$h^{(2)} = \frac{\sqrt{0.1}}{2^{2n}} \tag{C.11}$$

$$h^{(2)} + h^{(2)} + ... + h^{(2)} = 2^{2n} h^{(2)} = \sqrt{0.1} \tag{C.12}$$

- Se puede analizar C.9 como el sistema original con $H^{(2)} = 0.1 H^{(1)} = \frac{0.1}{2^{4n}}$

- Si se realiza el truncamiento para valores más pequeños que $H^{(2)}$ se garantiza que se obtendrá el 90 % del valor final de (2.12) y no siempre se perderá por completo el 10 % restante

Un problema que puede llegar a surgir es que el valor de una matriz $G_{i,j}$ de la ecuación (2.14) sea considerablemente grande y su peso H correspondiente sea truncado, de manera que no se pueda llegar al 90 % del valor final.

- Si no se puede llegar a una aproximación del 90 % de (C.1) cuando se realiza el truncamiento con $H^{(2)}$, se dice que existen matrices $G_{i,j}$ con valor significativo que fueron truncados.

- Para comprobar el punto anterior nos interesa demostrar si la suma de los subsistemas truncados es menor o mayor al 10 % de (C.1). De ser mayor al 10 %, la única explicación es que se truncaron valores de $G_{i,j}$ significativamente grandes.

Se demostrará a continuación los siguientes puntos:

- No existen valores de $G_{i,j}$ que sean significativamente grandes, por lo que siempre se obtendrá una aproximación de al menos el 90 % del valor de (C.1)

- La inclusión de $G_{i,j}$ en el análisis provoca que la noción de truncamiento se cambie por la de un número (suma de todos los subsistemas truncados) el cual se suma o resta a (C.1).

Para la demostración se usarán dos sistemas que actuaran como los casos extremos del sistema a analizar. Se observará que en estos sistemas como máximo se truncará el 10 % y por tanto el sistema real también tendrá un truncamiento menor al 10 %. Además de estos dos sistemas se utilizará el sistema (C.9) con los valores $G_{i,j}x$ iguales a los del sistema a analizar.

- El análisis se puede hacer usando solo el k-esimo elemento de cada uno de los vectores $G_{i,j}x$, así todos los elementos involucrados en el análisis son escalares.

- El análisis se realiza para el caso de un truncamiento del 10 % pero se puede generalizar para cualquier truncamiento
- Recordemos también que $0 \leq h \leq 1$

C.1. Sistema con los vectores $G_{i,j}x \geq 0$

El primer sistema a analizar tiene las siguientes características:

- Sus valores de $H_{i,j} = h_i h_j$ son los mismos que el del sistema real
- Sus valores de $G_{i,j}x$ son todos positivos y tienen el mismo valor que el valor absoluto de $|G_{i,j}x|$ del sistema real

Análisis para el sistema con los valores $G_{i,j}x \geq 0$:

- Si solo hay un subsistema no truncado se tiene el peor caso debido a que si hay más de un subsistema no truncado la suma de los subsistemas truncados disminuye
- El sistema se puede representar de la siguiente manera:

$$\sum_i \sum_j h_i h_j G_{i,j}x = \sum_{i \neq a} \sum_{j \neq a} h_i h_j G_{i,j}x + \sum_{i \neq a} h_i h_a G_{i,a}x + \sum_{j \neq a} h_j h_a G_{a,j}x + h_a h_a G_{a,a}x \tag{C.13}$$

  donde $h_a h_a G_{a,a}x$ es el único subsistema que no se trunca

- Probaremos que se cumple la desigualdad de la siguiente ecuación para cuando todos los vectores $G_{i,j}x$ son positivos

$$0.1 \sum_i \sum_j hh G_{i,j}x = h^{(2)} h^{(2)} \sum_i \sum_j G_{i,j}x > \sum_{i \neq a} \sum_{j \neq a} h_i h_j G_{i,j}x + \sum_{i \neq a} h_i h_a G_{i,a}x + \sum_{j \neq a} h_j h_a G_{a,j}x \tag{C.14}$$

- Para que se trunquen los subsistemas se debe cumplir que: $h_i h_j < h^{(2)}h^{(2)}$, $h_j h_a < h^{(2)}h^{(2)}$ y $h_i h_a < h^{(2)}h^{(2)}$
- Los vectores $G_{i,j}x$ en ambos lados de la desigualdad son los mismos. Por lo que al juntar lo visto en este punto con el punto anterior se tiene que el lado derecho de la desigualdad es menor
- Además se tiene que la suma de matrices $G_{i,j}x$ en el lado izquierdo de la desigualdad tiene un termino más que en el derecho el cual es $h^{(2)}h^{(2)}G_{a,a}x$.
- Podemos concluir que el truncamiento en sistemas donde $G_{i,j}x > 0$ será como máximo de un 10 % del sistema y por tanto los valores de la matriz G no impedirán que se obtenga como mínimo el 90 % del valor del sistema

C.2. Sistema con los vectores $G_{i,j}x \leq 0$

Se analizara un sistema con las siguientes características:

- Sus valores de $H_{i,j} = h_i h_j$ son los mismos que el del sistema real
- Sus valores de $G_{i,j}x$ son todos negativos y su valor absoluto es el mismo que el valor absoluto de $|G_{i,j}x|$ del sistema real
- Si solo hay un subsistema no truncado se tiene el peor caso debido a que si hay más de un subsistema no truncado el valor absoluto de la suma de los subsistemas truncados disminuye
- El sistema se puede representar de la siguiente manera:

$$\sum_i \sum_j hhG_{i,j}x = \sum_{i\neq a} \sum_{j\neq a} h_i h_j G_{i,j}x + \sum_{i\neq a} h_i h_a G_{i,a}x + \sum_{j\neq a} h_j h_a G_{a,j}x + h_a h_a G_{a,a}x \tag{C.15}$$

donde $h_a h_a G_{a,a}x$ es el único subsistema que no se trunca

- Probaremos que se cumple la desigualdad de la siguiente ecuación para cuando todos los $G_{i,j}x$ son negativos

$$|0.1\sum_{i}\sum_{j} hhG_{i,j}x| = |h^{(2)}h^{(2)}\sum_{i}\sum_{j} G_{i,j}x| > |\sum_{i\neq a}\sum_{j\neq a} h_ih_jG_{i,j}x + \sum_{i\neq a} h_ih_aG_{i,a}x + \sum_{j\neq a} h_jh_aG_{a,j}x| \tag{C.16}$$

- Para que se trunquen los subsistemas se debe cumplir que: $h_ih_j < h^{(2)}h^{(2)}$, $h_jh_a < h^{(2)}h^{(2)}$ y $h_ih_a < h^{(2)}h^{(2)}$

- Los vectores $G_{i,j}x$ en ambos lados de la desigualdad son los mismos. Por lo que al juntar lo visto en este punto con el punto anterior se tiene que el lado derecho de la desigualdad es menor

- Además se tiene que la suma de matrices $G_{i,j}x$ en el lado izquierdo de la desigualdad tiene un termino más, el cual es $h_2h_2G_{a,a}x$.

- Del análisis de este sistema se observa que se debe cambiar la idea de truncamiento por la de un valor (suma de todos los subsistemas truncados) que se resta al sistema

$$\sum_{i}\sum_{j} h_ih_jG_{i,j}x - (\sum_{i\neq a}\sum_{j\neq a} h_ih_jG_{i,j}x + \sum_{i\neq a} h_ih_aG_{i,a}x + \sum_{j\neq a} h_jh_aG_{a,j}x) \tag{C.17}$$

  Para este sistema el realizar la resta implica obtener un valor mayor que el del sistema, mientras que para el sistema de la sección anterior se obtendrá un valor menor que el del sistema

- Podemos concluir que la suma de los subsistemas truncados en los sistemas que cumplen con $G_{i,j}x < 0$ será como máximo de un 10 % y por tanto los valores

de la matriz $G_{i,j}$ no impedirán que se obtenga un valor que varíe en un 10 % respecto al valor del sistema

C.3. Sistema general

En esta sección se usaran los resultados de las dos secciones anteriores para resolver el caso general.

- En un sistema general los valores en los vectores $G_{i,j}x$ pueden tener signos contrarios.
- La suma de los subsistemas truncados podrá ser tanto positiva como negativa.
- Si el subsistema no truncado es positivo la comparación se realiza con el sistema que tiene todos los vectores $G_{i,j}x > 0$. Si es negativo se compara con el sistema con $G_{i,j}x < 0$
- El valor absoluto de la suma de los subsistemas truncados en el caso general será menor o igual que en los casos extremos. Esto es debido a que en los casos extremos todos los subsistemas tienen el mismo signo.
- Como los sistemas de las secciones anteriores son iguales al sistema real salvo en los signos del vector $G_{i,j}x$ y debido a que el truncamiento solo depende del valor de las h_ih_j el subsistema que no se truncó es el mismo para el sistema general que el del caso extremo con el cual se está comparando
- Debido a lo visto en las secciones anteriores el subsistema no truncado variará respecto al valor total del sistema de las secciones anteriores en un 10 %
- Como el valor absoluto de la parte truncada en el caso general es menor que en los casos extremos, el valor del subsistema no truncado en el sistema general tendrá también una variación del 10 % respecto al total del sistema general

  a=Total caso extremo

b=Subsistema no truncado

c=Truncamiento caso extremo

A=Total caso general

C=Truncamiento caso general

$$a = b + c$$

$$A = b + C$$

$$-0.1a \leq c \leq 0.1a$$

$$0.9a \leq b \leq 1.1a$$

$$|c| \geq |C|$$

Si la ultima desigualdad fuera $|c| > |C|$ el valor de **A** estará más cercano a **b** que el valor de **a**, por lo que **b** representaría un porcentaje más próximo al 100 % para **A** y dado que **c** asegura que el valor de **b** varía como máximo en un 10 % respecto a **a**, entonces **b** variará en menos del 10 % respecto a **A**

$$0.9A \leq b \leq 1.1A$$

De todo lo anterior se puede concluir que:

- No existen valores de $G_{i,j}$ que sean significativamente grandes, por lo que siempre se obtendrá una aproximación de más del 90 % del valor de (C.1)
- La inclusión de $G_{i,j}$ en el análisis provoca que la noción de truncamiento se cambie por la de un número (suma de todos los subsistemas truncados) el cual se suma o resta a (C.1).

- Existen dos sistemas que actúan como limites, en al menos un instante de tiempo, de un sistema controlado, con un punto de equilibrio global y asintóticamente estable. Las características de estos sistemas se mencionaron en las secciones anteriores

- El truncamiento visto en este apéndice puede funcionar de manera general para los sistemas con control difuso, únicamente necesitando como dato el número de variables **h** y el truncamiento que se quiere realizar.

Referencias

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. IEEE.

Abeni, L., Lipari, G., and Lelli, J. (2015). Constant bandwidth server revisited. *Acm Sigbed Review*, 11(4):19–24.

Aparicio-Santos, J. and Benítez-Pérez (2021). Administrador de recursos para procesadores heterogéneos. *Revista Iberoamericana de Automática e Informática Industrial*.

Aparicio-Santos, J., Hermosillo-Gómez, J., Benítez-Pérez, H., and Alvarez-Icaza, L. (2021). Controlador difuso para compensar cargas de comunicación en sistemas en tiempo real. *Revista Iberoamericana de Automática e Informática Industrial*, 18(3):288–299.

Buttazzo, G. C. (2011). *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media.

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

ITU, T. (1994). Terms and definitions related to quality of service and network performance including dependability. *Recommendation E*, 800.

Lee, B. G., Park, D., and Seo, H. (2009). *Wireless communications resource management*. John Wiley & Sons.

Maggio, M., Bini, E., Chasparis, G., and Årzén, K.-E. (2013). A game-theoretic resource manager for rt applications. In *2013 25th Euromicro Conference on*

Real-Time Systems, pages 57–66. IEEE.

paramiko.org (2023). Pramiko's documentation (2023). paramiko. `https://docs.paramiko.org/en/stable/`.

Santos, A. (2017). *Diseño de un controlador difuso para compensar cargas de comunicación en tiempo real*. Tesis.

Sommerville, I. (2011). *Software Engineering, 9/E*. Pearson Education India.

Tanaka, K. and Wang, H. O. (2001). *Fuzzy control systems design and analysis*. Wiley Online Library.