DTS: un entorno de planificación distribuido de bajo coste

Francesc Solsona[1] Francesc Giné[1] Porfidio Hernández[2] Emilio Luque[2]
francesc@eup.udl.es sisco@eup.udl.es p.hernandez@cc.uab.es iinfd@eup.udl.es
[1]Dept. Informática, Universitat de Lleida (España)
[2]Dept. Informática, Universitat Autònoma de Barcelona (España)

Resumen

El objetivo principal de nuestro trabajo es proporcionar mediante una NOW (Networks of Workstations), compuesta por estaciones Unix no dedicadas y heterogéneas, un entorno donde se puedan ejecutar eficientemente tanto aplicaciones de usuario o interactivas, como paralelas, implementadas mediante el entorno de programación PVM (Parallel Virtual Machine).

La motivación para utilizar las NOW en supercomputación está justificada si tenemos en cuenta su bajo coste, la escasa utilización de sus recursos de cómputo y además, la escalabilidad que proporciona su naturaleza distribuida.

Hemos implementado un sistema planificador distribuido denomidado DTS (Distributed Scheduler) formado por tres tipos de módulos. *El Planificador DTS* está compuesto por varios procesos distribuidos entre las diferentes estaciones. La principal función de cada uno de estos planificadores es variar de forma dinámica la cantidad de ciclos de CPU asignados a las tareas paralelas (Periodo Paralelo) dependiendo de la carga interactiva global y de la potencia de cálculo de todas las estaciones de la NOW. *CALC_LOAD* es el módulo responsable de registrar la carga en cada estación. Finalmente, con el fin de monitorizar y supervisar el estado del sistema se ha implementado un módulo, denominado *CONDTS* que actúa como cónsola del sistema DTS.

En aplicaciones con un elevado volumen de comunicaciones, el desfase en el inicio de cada *Periodo Paralelo* entre las diferentes estaciones puede provocar un elevado *overhead*. Hemos implementado un algoritmo de sincronización, basado en *sockets*, que resuelve el problema del desfase y que además, teniendo en cuenta la carga media de las aplicaciones de usuario de la NOW, asigna el mismo intervalo paralelo a todas las estaciones.

El correcto funcionamiento del sistema DTS se ha verificado ejecutando tres benchmarks paralelos de la *NAS suite*, *ep*, *is* y *mg*. Cada uno de los benchmarks de la NAS utilizados es representativo de un tipo diferente de programa paralelo. Los benchmarks se han ejecutado en tres entornos distribuidos distintos: PVM original, DTS sin sincronización y DTS con sincronización. En general, en aplicaciones con un gran volumen de comunicación (*is* y *mg*) los resultados obtenidos en los dos modos de operación del DTS, han sido bastante satisfactorios. Así mismo, en la aplicación con un elevado cómputo (*ep*), el sistema DTS aumenta el rendimiento a medida que se incrementa la carga local de cada estación.

Palabras clave – NOW, PVM, procesamiento distribuido, potencia de cálculo, índice de carga, planificación, sincronización

DTS: un entorno de planificación distribuido de bajo coste

1 Introducción

La mayoría de *estaciones de trabajo* (ws) suelen estar pensadas para ejecutar aplicaciones que requieren una gran interactividad y normalmente un moderado volumen de computación, a las cuales denotaremos mediante "*interactivas*" o "*locales*" indistintamente. Por el contrario, los *Supercomputadores* y *Mainframes* estan pensados para ejecutar aplicaciones que requieren un gran volumen de cálculo. Las ws's son mucho más baratas, por ello, en cualquier centro o laboratorio se disponen de bastantes unidades, generalmente infrautilizadas. Incluso cuando hay un usuario en cada ws, la utilización media de la CPU suele estar por debajo del 10%.

Además de por su escasa utilización, la motivación para utilizar las NOW's en supercomputación está justificada si tenemos en cuenta su bajo coste y debido a su naturaleza distribuida, supone una solución al cuello de botella que se producen en los accesos concurrentes a los diferentes recursos del sistema (E/S, mem principal, mem. secundaria, etc ...). Otro factor que favorece su uso son los continuos avances tecnológicos, los cuales permiten obtener resultados realmente esperanzadores [1]: las ws's actuales proporcionan un rendimento equivalente a un tercio del Cray C90 y redes como ATM, Fast-Ethernet o Myrinet, permiten trabajar con anchos de banda superiores a los 100Mbps y latencias inferiores a 1ms.

Numerosas actividades de investigación han intentado explotar la potencia ofrecida por un entorno NOW [2, 3, 4, 5, 6, 7, 8]. Este trabajo se ha orientado hacia tres áreas distintas: el análisis, diseño y desarrollo de servicios de ejecución remota; sistemas basados en la priorización de las tareas interactivas frente a las paralelas; y entornos de ws's, las cuales sirvan como elementos de procesamiento en una Máquina Virtual (MV) que combine la ejecución de tareas locales y paralelas y que, sin necesidad de migración, asegure un buen rendimiento para ambos tipos de aplicaciones.

En la primera área de investigación, todos los sistemas comparten el mismo principio: permiten la utilización de mecanismos de migración con el fin de ejecutar eficientemente las tareas paralelas, *Condor* [2], *Process Server* [3], *Sprite* [4], *V-System* [5] y *Stardust* [6].

La segunda área de investigación, se ha basado en la implementación de entornos de procesamiento paralelo en una NOW, los cuales priorizan la asignación de recursos a las tareas de usuario o interactivas frente a las remotas o paralelas; solución propuesta por *Stealth Distributed Scheduler* [7].

El diseño de planificadores que aseguren la posibilidad de repartir los ciclos de CPU de cada ws entre tareas paralelas y locales, con bajo "*overhead*" para las tareas paralelas y un buen tiempo de respuesta para las tareas locales, es el principal objetivo de la tercera área de investigación. En SONIC [8], en el espacio de usuario del s.o. Mach, se implementa un módulo denominado "scheduling server", el cual de forma periódica e ininterrumpida y durante un breve espacio de tiempo activa las tareas paralelas, asignándoles una prioridad superior que a las tareas de usuario y a las propias del kernel. Su trabajo se centró en averiguar el tiempo que podía concederse a las tareas paralelas sin perturbar en exceso las tareas de usuario y sin profundizar en aspectos relacionados con su planificación para la obtención de mejoras en su rendimiento.

En este artículo y siguiendo la línea de trabajo iniciada en SONIC, presentamos un sistema distribuido, denomidado DTS (Distributed Scheduler) que implementa una MV en una NOW, caracterizada según [9] y [10], el cual gestiona tanto las tareas locales como paralelas (en nuestro caso distribuidas), con la principal función de garantizar un alto grado de rendimiento en la ejecución de los dos tipos de aplicaciones.

En la segunda sección se definen los parámetros que caracterizan un entorno NOW así como el diseño modular del sistema DTS. En la sección III se analiza en detalle el módulo principal del sistema, el planificador DTS. En la cuarta sección se explica un mecanismo de sincronización que se incorpora al sistema con el fin de aumentar su rendimiento. Finalmente, mostramos experimentalmente la ganancia obtenida mediante nuestro sistema con respecto al entorno PVM original.

2 El Sistema DTS

El objetivo principal de este artículo es proporcionar mediante una NOW (compuesta por ws's Unix no dedicadas y heterogéneas) un entorno donde se puedan ejecutar eficientemente tanto aplicaciones interactivas, propias de cualquier estación de trabajo, como distribuidas, implementadas mediante el entorno de programación PVM [11, 12, 13, 14].

Para entender el diseño y posterior incorporación de mejoras del sistema DTS, necesitamos primero conocer los principales parámetros que caracterizan una NOW.

2.1 Caracterización de un entorno NOW

Una NOW heterogénea se define como HN($\mathcal{M}$), siendo $\mathcal{M}$=M_1, ... ,M_N un conjunto de N ws's con idéntica latencia de comunicación entre cualquier par de máquinas.

Cada una de las máquinas que forman una NOW se caracteriza mediante su capacidad de cómputo, *Power Weight* $W_i(A)$ [9] y la carga presente en la misma, *Load Index* Q_i [10].

Power Weight $W_i(A)$: Se define como la capacidad de cómputo para ejecutar una determinada aplicación A en una ws $M_i \in$ HN($\mathcal{M}$). Formalmente:

$$W_i(A) = \frac{S_i(A)}{\max_{j=1}^{N}\{S_j(A)\}}, \ i = 1 \ldots N, \tag{1}$$

donde $S_i(A)$ es la velocidad de ejecución del programa A en la ws M_i. Podríamos decir que, $W_i(A)$ mide la velocidad de ejecución relativa de una aplicación A en la ws M_i con respecto a la más rápida, en un conjunto de ws's HN($\mathcal{M}$).

Load Index Q_i: D. Ferrari [10], demostró que la longitud media de la "*cola de preparados de la CPU*" es una buena aproximación del índice de carga, puesto que no había apenas diferencias con métricas que tenían en cuenta la totalidad de las colas de una ws. Esta se obtiene mediante un alisamiento exponencial de la longitud media de la cola durante los últimos P segundos. Formalmente:

$$Q_i = Q_{i-1}(1 - e^{-P}) + q_i e^{-P}, \tag{2}$$
$$i \geq 1, Q_0 = 0,$$

donde $Q_{i-1} \equiv$ último índice calculado, $q_i \equiv$Longitud de la cola en el instante de muestreo y $P \equiv$ Periodo de promediado tomado.

Por lo tanto, el estado de una NOW está determinado mediante el conjunto de los diferentes *Power Weights* y *Load Indexes* de cada ws, denotados mediante $\mathcal{W}$=$\{W_i(A)\}$ y $\mathcal{Q}$=$\{Q_i\}$, i=1...N respectivamente.

2.2 Diseño del sistema DTS

Una de las premisas de nuestro entorno es obtener un alto grado de portabilidad, hecho que ha influido en la decisión de implementarlo en el entorno de usuario de dos Sistemas Operativos UNIX, *LINUX y SOLARIS*.

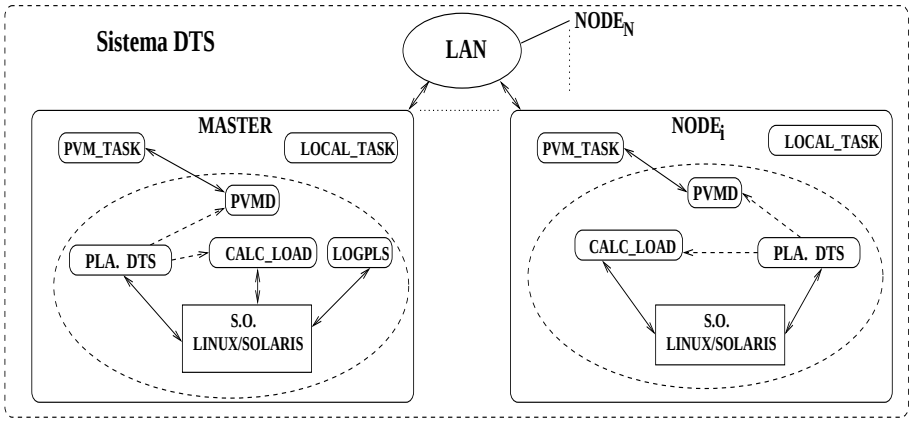

Figura 1: Diseño del sistema DTS

El sistema DTS está formado por tres tipos de módulos distribuidos entre la MV (ver figura 1), *el Planificador DTS*, *CALC_LOAD* y *CONDTS*. En la elección de este modelo ha influido enormemente la diversidad de funciones encomendadas a nuestro sistema.
El Planificador DTS: Está formado por varios procesos, distribuidos entre las diferentes ws's. La principal función de cada uno de estos planificadores es variar de forma dinámica la cantidad de ciclos de CPU asignados a las tareas distribuidas de su correspondiente ws, dependiendo de la carga interactiva de la MV.
CALC_LOAD: Al igual que el planificador DTS, está compuesto por varios procesos, uno por cada estación activa. Son los responsables de registrar la carga de cada estación (Q_i) y transferirla a CONDTS.
CONDTS: Compuesto por un solo módulo. Se puede activar desde cualquier nodo de la MV, al cual denominaremos nodo *master*, y es la cónsola del sistema DTS. Sus principales funciones son: actualizar los diferentes parámetros de la MV, monitorizar y supervisar el estado del sistema, y además proporciona un mecanismo de sincronización a toda la MV.

3 El planificador DTS

PLANIFICADOR DTS

fork&exec($CALC_LOAD$)
fork&exec($pvmd$)
set PRI($planificador_DTS$) = max($rt_priority$)
set PRI($CALC_LOAD$) = max($rt_priority$) - 1
set PRI($pvmd$) = max($rt_priority$) - 2
while($tareas_pvm$) **do**
  sleep(PP)
  stop($tareas_pvm$**)**
  sleep(PI)
  resume($tareas_pvm$)
end/*while*/

Figura 2: Algoritmo: Planificador DTS

En la MV, cada planificador DTS (ver figura 2), se ejecuta con la más alta prioridad posible, que se corresponde con la clase *REAL TIME*, tanto en el s.o. LINUX [15] como SOLARIS [16]. En la fase de inicialización crea dos procesos y a continuación ejecuta

(*fork&exec*) en sus respectivos espacios de usuario *CALC_LOAD* y *pvmd* (el daemon de pvm), asignándoles prioridades ligeramente inferiores. Este proceso de creación de proceso está representado en la figura 1 mediante flechas de trazo discontinuo. El planificador asigna un intervalo de ejecución a las tareas locales (PI) y a las distribuidas (PP). De forma continuada y hasta que no queden más tareas distribuidas en el sistema, pasado un intervalo PP se desactivan (activan) las tareas distribuidas (locales) y pasado un intervalo PI se activan (desactivan) las distribuidas (locales). Como las tareas distribuidas tienen mayor prioridad que las locales, las cuales se ejecutan en una clase con prioridad inferior, para desactivar las tareas distribuidas (y activar las locales) se envía a todas ellas la señal SIGSTOP y para reanudarlas (y desactivar las locales) se les envía la señal SIGCONT. Este proceso de activación y desactivación de tareas distribuidas se muestra en la figura 3.

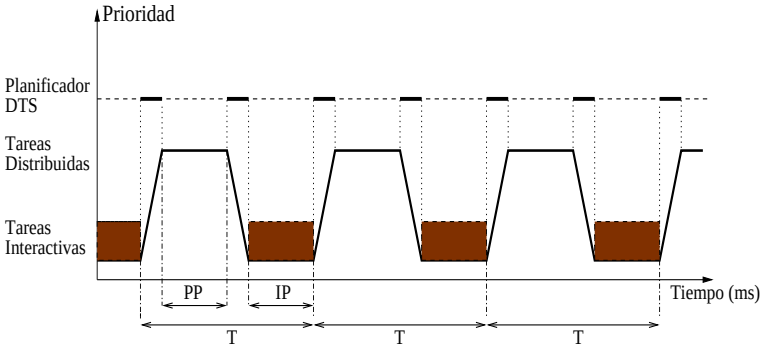

Figura 3: Diagrama de tiempo del planificador DTS

Un detalle de implementación que no se muestra en el algoritmo es que al añadir una nueva ws en la cónsola de pvm no se inicia su respectivo *pvmd* sinó nuestro planificador, siendo éste quien realmente lo inicia, de esta forma el planificador tendrá control sobre todas las tareas pvm de la estación, incluso del *pvmd*. Además, todas las tareas distribuidas pertenecen a un mismo grupo de procesos, del cual el proceso líder es *pvmd*; como éste a su vez crea todas las tareas pvm (en nuestro caso distribuidas), controlando su terminación (condición que hemos denotado mediante *WHILE(tareas_pvm)*) sabremos también que no quedan tareas pvm en la ws, y el planificador también finalizará. Con este método se consigue una forma automática de inicialización y finalización del sistema DTS sin la engorrosa necesidad de hacerlo manualmente. También facilita el control e identificación de las tareas pvm, hecho que hace al sistema mucho más eficiente.

Fijémonos en que ahora variando los intervalos PP e PI, podremos controlar el sistema DTS para que la MV, en su conjunto, de más prioridad de ejecución a las tareas pvm o a las locales. Cuanto mayor sea la relación $\frac{PP}{IP}$, más prioridad de ejecución estaremos asignando a las tareas distribuidas frente a las aplicaciones de usuario o locales.

4 Mecanismo de Sincronización

En aplicaciones con un elevado o incluso moderado volumen de comunicaciones, el desfase en el inicio de los periodos PI y PP entre las diferentes ws's puede provocar un *overhead* inaceptable (ver figura 4).

Hemos implementado un algoritmo de sincronización, basado en *sockets*, que resuelve el problema del desfase y que además teniendo en cuenta la carga local media de la MV, asigna los mismos intervalos PI y PP en todas las ws's. Al término del algoritmo, la MV ejecuta en todas las ws's, de forma síncrona y durante el mismo intervalo de tiempo solo tareas interactivas o distribuidas, que se corresponden a los periodos PI y PP respectivamente.

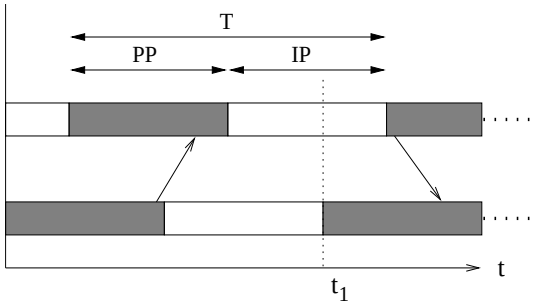

Figura 4: Máquinas no sincronizadas

CONDTS: nodo master (M_1)
int *rec*; int TS=*ts*
while(true) **do**
  sleep(TS); rec=0
  multicast($\mathcal{M}$,$Control$)
  while(($rec < N$)**and**(**not**(timeout)) **do**
    async_receive(any, Q_i); rec=rec+1
  end/*while*/
  compute(CRM,PP,PI)
  multicast($\mathcal{M}, \mathcal{PP}\&PI$)
end/*while*/

CALC_LOAD: en cada $M_i \in \mathcal{M}$
async_receive($M_1, Control$)
if ($Q_i(t) \neq Q_i(t - TS)$) **send**(M_1, Q_i)

PLANIFICADOR DTS: en cada $M_i \in \mathcal{M}$
async_receive($M_1, PP\&PI$)
set $PP\&PI$
goto *inicio_bucle_principal*

Figura 5: Algoritmo de sincronización

El algoritmo (véase figura 5) se inicia en el módulo *CONDTS* del nodo master (M1), el cual envía, cada TS segundos (periodo de sincronización) y en modo multicast (multicast($\mathcal{M}$,$Control$)), un mensaje de control a todas las ws's del sistema. Cuando el módulo *CALC_LOAD* de cada nodo recibe el mensaje de control de forma asíncrona (async_receive($any, Control$)), en caso de que $Q_i(t)$, en ese instante, sea diferente del anterior valor calculado $Q_i(t-TS)$, envía $Q_i(t)$ a *CONDTS* (send(M_1, Q_i)). *CONDTS*, cuando haya recibido un mensaje de cada ws o bién haya pasado un cierto "*timeout*", calcula la carga relativa media (CRM) mediante la siguiente expresión:

$$CRM = \frac{\sum_{i=1}^{N} \frac{Q_i}{W_i(A)}}{N}$$

CONDTS, de acuerdo a la tabla 1,[1] también calcula y posteriormente envía en modo *multicast* los valores de los intervalos PP y PI a los planificadores DTS. Ante la recepción, también asíncrona del mensaje, los planificadores ajustan adecuadamente sus

[1] Esta tabla se ha construido teniendo en cuenta que valores de la cola de preparados superiores a 1.75 son muy elevados

correspondientes intervalos PI y PP y, posteriormente, reanudan su ejecución a partir de una dirección de código preestablecida, situada al inicio del bucle principal del planificador DTS. Al término de este algoritmo todos los nodos estan sincronizados y además, los intervalos PI y PP tienen un valor acorde con la carga interactiva de la MV.

CRM			PI	PP
0≤	CRM	≤0.25	10	90
0.25<	CRM	≤0.5	20	80
0.5<	CRM	≤0.75	30	70
0.75<	CRM	≤1	40	60
1<	CRM	≤1.25	50	50
1.25<	CRM	≤1.5	60	40
1.5<	CRM	1.75	80	20
1.75<	CRM		90	10

Tabla 1: Relación entre CRM y los intervalos PP y PI.

5 Experimentación realizada

Esta sección la hemos dividido en dos partes. En la primera parte se describe el entorno experimental así como los *benchmarks* utilizados. En la segunda, se comparan los resultados obtenidos en tres sistemas distribuidos distintos: PVM original, DTS estático y DTS centralizado.

5.1 Entorno experimental

El entorno de experimentación está formado por cuatro Pentiums a 200Mh con s.o. LINUX, 32 MB de memoria y 64 MB de *swap*, conectados por una red aislada con arquitectura Ethernet de 10 Mbps y una *latencia* mínima del orden de 1 ms. Toda la experimentación se ha realizado en un entorno "no dedicado" a la ejecución de tareas distribuidas, con una carga de utilización media de la CPU de un 10% (Light) y un 70% (Heavy). La carga se ha simulado mediante la ejecución de un *script* representativo del trabajo típico que realiza un usuario local. Este script tiene la particularidad de poder fijar la carga de un usuario en el % de utilización de CPU deseado. Una carga Light del 10% equivaldría a un tamaño de la cola de preparados (*Load Index*) de $\simeq 0.2$ y Heavy, 70%, de $\simeq 1.5$. En esta experimentación no se ha tenido en cuenta el *Power Weight*, puesto que todas las estaciones tienen las mismas características.

El correcto funcionamiento del sistema DTS se ha verificado ejecutando tres benchmarks paralelos de la *NAS suite* [17], *ep*, *is* y *mg*. Cada uno de los benchmarks de la NAS utilizados es representativo de un tipo diferente de programa paralelo: *ep*, característico ejemplo de una aplicación SPMD del tipo "*embarrassingly parallel*", con un volumen de comunicaciones muy pequeño y una elevada carga de computación; *is* es una aplicación de ordenación de enteros con un elevado volumen de comunicaciones pero, por el contrario, su requerimiento de cálculo es bajo y por último, *mg*, un benchmark que soluciona el *problema discreto de Poisson*, con un gran volumen de comunicaciones pero altamente estructuradas, utilizando como patrón de conexión una estructura en anillo. Los parámetros de los tres benchmarks utilizados a lo largo de toda la experimentación se muestran en la tabla 2.

Benchmark	Complejidad
EP	2^{28}
IS	2^{21}-[0..2^{19}] keys
MG	128x128x128

Tabla 2: Parámetros de los tres benchmarks

5.2 Resultados Obtenidos

Los experimentos realizados se han basado en la ejecución de los tres benchmarks en tres entornos distintos: el primero en el entorno PVM original y los dos restantes, coincidiendo con el número de modos de operación del sistema DTS (seleccionables desde la cónsola CONDTS): *estático* y *centralizado*. La diferencia entre los dos modos de operación es que en el estático no se aplica el mecanismo de sincronización (explicado en la sección 4) y, por el contrario, en el centralizado si. Se ha hecho esta clasificación para poder comparar primero los resultados obtenidos con el sistema DTS respecto a PVM y además, para mostrar las mejoras obtenidas al incorporar el mecanismo de sincronización. Por último, con objeto de comprobar el impacto que se produce en las tareas locales, se ha medido el overhead causado por el sistema DTS en la ejecución de las aplicaciones de usuario.

En las tablas se refleja el tiempo de ejecución en segundos (TE), siguiendo la notación siguiente: L indica carga Light, H, carga Heavy, T es la suma de los intérvalos PI y PP (T=PP+PI) y $f = PP \div T$, es la ratio entre el intérvalo paralelo con respecto a T.

PVM: La tabla 3 muestra los resultados obtenidos por S. White en [17], con los mismos parámetros, sin carga y utilizando 16 estaciones Sun SS1+, con 16 MB de memoria ram y 32 MB de *swap*, conectadas mediante una red aislada, también con arquitectura Ethernet de 10Mbps y con una latencia mínima de 2 milisegundos. En la tabla 4 se muestran los tiempos de ejecución de los tres benchmarks en nuestro entorno de trabajo. La comparación entre ambas tablas refleja que los resultados de partida se ajustan a otros trabajos realizados con anterioridad y que el orden de magnitud del tiempo de ejecución es el mismo. Los resultados obtenidos en nuestro entorno son mejores pero también la tecnología utilizada ofrece mayores prestaciones.

Benchmark	PVM TE(secs)
EP	1603
IS	607
MG	198

Tabla 3: Resultados obtenidos por S. White con PVM versión 3.2.6

Benchmark	PVM TE(secs)	
	Light	*Heavy*
EP	412	877
IS	420	432.1
MG	130	135

Tabla 4: Resultados obtenidos con PVM versión 3.3.11

Modo Estático: En este modo, el usuario de la MV determina desde la consola *CONDTS*, el valor de los periodos PP y PI y por consiguiente, del periodo T. Aunque el entorno DTS permite la posibilidad de asignar diferentes valores de PP y PI a distintas ws, las pruebas realizadas se han hecho asignando los mismos valores a todos los nodos de la MV. En esta experimentación, se ha investigado el impacto producido en las aplicaciones distribuidas al variar los valores de T y f.

Benchmark	T = 100ms				T = 1000ms			
	L-0.5	**L-0.8**	**H-0.5**	**H-0.8**	**L-0.5**	**L-0.8**	**H-0.5**	**H-0.8**
EP	810	530	814	531	820	532	821	530
IS	458	380	462	390	542	443	550	450
MG	180	99	182	102	194	106	197	119

Tabla 5: DTS en modo estático

De acuerdo a los valores mostrados en la tabla 5, se observa que los tiempos de ejecución obtenidos son independientes de la carga local simulada, lo que verifica el correcto funcionamiento del planificador DTS. La carga no influye en las comunicaciones, *is* y *mg* no se ven afectados; *ep* en cambio, tiene un valor fijo de ejecución (intérvalo PP), por lo tanto al variar la carga su tiempo de ejecución no debe variar. Otro dato a resaltar es que los resultados obtenidos en la aplicación con poca comunicación, *ep*, son prácticamente independientes del valor del periodo T, mientras que en aplicaciones con un elevado volumen de comunicaciones el rendimiento disminuye proporcionalmente al aumento de T. Comparando estos resultados con los obtenidos en la ejecución del PVM original, mostrados en la tabla 4, se observa una mejora en los resultados a medida que aumentamos el valor de f, acentuándose aún más en las aplicaciones con un elevado intercambio de mensajes.

Modo Centralizado: En la experimentación anterior, no se ha tenido en cuenta ni las cargas locales de cada una de las máquinas Q_i ni la sincronización entre los diferentes nodos de la MV.

Benchmark	Light	Heavy
EP	690	820
IS	360	380
MG	86	116

Tabla 6: DTS en modo centralizado

Los valores mostrados en la tabla 6 han sido obtenidos para un periodo de sincronización (TS) de 30 segundos. Observar la mejora experimentada en aquellas aplicaciones con elevado volumen de comunicaciones, lo que corrobora el buen funcionamiento de la sincronización realizada. El empeoramiento de los resultados en la aplicación con elevado cómputo (*ep*) se compensa con la menor perturbación en el trabajo del usuario local, dado que en este caso el valor de PP varía de acuerdo a la carga Q_i.

En general, en los dos modos de operación del DTS, los resultados obtenidos en aplicaciones con gran comunicación han sido bastante satisfactorios. Al iniciar y parar constantemente el daemon de pvm, *pvmd*, se inicializan también del mismo modo las colas de mensajes del sistema operativo, hecho que ha podido influir de forma favorable en nuestro entorno, aunque faltaría probarlo experimentalmente. Al reiniciar constantemente el daemon de pvm, las comunicaciones entre los diferentes procesos distribuidos también pueden verse favorecidas debido al hecho que se ha reducido considerablemente el número de colisiones y compense de este modo, el overhead introducido por el sistema de planificación.

Todos los resultados de las tablas de este trabajo son medias estadísticas de varias muestras tomadas en un gran número de experimentaciones, las cuales se caracterizan por una elevada dispersión. Este hecho también es consecuencia directa del elevado número de colisiones que se producen tanto en el entorno PVM como en el DTS al ejecutar las aplicaciones con alto número de comunicaciones. La impredictibilidad de la ejecución de cada aplicación distribuida se ve potenciada aún más por el efecto *captura* [18]. El efecto captura es el término usuado para describir una conocida característica, producto del funcionamiento del algoritmo de retransmisiones del protocolo ethernet [19], el cual

sostiene que el intérvalo entre sucesivas retransmisiones depende del número de colisiones.

Impacto del sistema DTS en la carga local: Con objeto de medir el overhead introducido por el sistema DTS en la carga local de cada estación, se ha elaborado un *script* que simula la ejecución de las aplicaciones propias de un usuario, como son compilación, lecturas y escrituras en ficheros, etc ..., con un tiempo de ejecución en una máquina totalmente dedicada de 86 s.

Una métrica que puede ayudarnos a computar el impacto introducido en el rendimiento de las aplicaciones locales es el *Overhead Relativo* (OR), definido como la diferencia en tiempo de la ejecución no dedicada con respecto a la dedicada, es decir:

$$OR = \frac{Tiempo_ejecu_no_dedicada - Tiempo_ejecu_dedicada}{Tiempo_ejecu_dedicada}$$

En la tabla 7, se muestran los tiempos obtenidos al ejecutar el *script* conjuntamente con los diferentes benchmarks en el sistema DTS en modo estático. Como podemos ver, el overhead producido al ejecutar *is* y *mg* es bajo, especialmente para el *is*, con menor cómputo que el *mg*; en *ep*, en cambio, el Overhead Relativo indica que se ha duplicado y hasta incluso triplicado su ejecución, hecho normal si tenemos en cuenta que requeriere una gran capacidad de cálculo; es imprescindible en estos casos encontrar un compromiso entre ambos tipos de aplicaciones (locales y distribuidas) o bien priorizar las unas con respecto las otras. No obstante, la mayoría de las tareas de usuario son interactivas y por ello, este overhead no es ni mucho menos preocupante ya que la variación del tiempo de respuesta del sistema es casi inapreciable por el usuario.

Benchmark	**T = 100ms**				**T = 1000ms**			
	f = 0.5		**f = 0.8**		**f = 0.5**		**f = 0.8**	
	TE	OR	TE	OR	TE	OR	TE	OR
EP	155	0.8	280	2.25	159	0.8	285	2.3
IS	95	0.1	102	0.1	97	0.1	103	0.1
MG	122	0.4	125	0.4	113	0.3	124	0.4

Tabla 7: Evaluación del rendimiento local. TE:Tiempo de Ejecución, OR: Overhead Relativo.

6 Conclusiones

En este artículo hemos introducido un entorno de trabajo que sin perturbar el rendimiento de las tareas locales mejora el rendimiento global de la ejecución de las tareas distribuidas en una NOW, compuesta de estaciones heterogéneas no dedicadas. Nuestros esfuerzos se han centrado en balancear la ejecución de dichas tareas haciendo un buen uso de los ciclos muertos de CPU que no utilizan las tareas interactivas para ejecutar en esos períodos las distribuidas. Siempre que no ha sido posible sacar un buen provecho de dicha situación, hemos repartido proporcionalmente los ciclos de CPU.

En el futuro, estamos interesados en sincronizar las estaciones de trabajo en función del tipo de aplicaciones distribuidas que se ejecuten. Para ello, deberíamos realizar un amplio estudio de los diferentes tipos de estructuras de comunicación y sincronización de tareas distribuidas y en función de ello, implementar diferentes métodos de sincronización.

También existe la posibilidad de pensar en una verdadera sincronización física de los temporizadores de las estaciones. De esta forma, el tiempo absoluto es una referencia válida para sincronizarlas. Un buen punto de partida sería utilizar el protocolo de sincronización NTP (Network Time Protocol) implementado por D. L. Mill [20, 21].

Otra línea a seguir puede ser la asignación estática o dinámica de tareas distribuidas, en función de la potencia de cálculo y de la carga de cada estación. En PVM, se pueden

crear tareas gestoras de recursos y en especial una de ellas, puede utilizarse para gestionar la asignación de tareas a las distintas estaciones de trabajo.

Referencias

[1] T. E. Anderson, D. E. Culler, and D. Patterson, "A case for now (networks of workstations)," *IEEE Micro*, pp. 54–64, 1995.

[2] M.Litzkow, M.Livny, and M.W.Mutka, "Condor - a hunter of idle workstations," *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104–111, June 1988.

[3] R. Hagman, "Process server: Sharing processing power in a workstation environment," *Proc. 6th Int'l Conference on Distributed Computing Systems*, pp. 260–267, May 1986.

[4] J.K. Ousterhout, A. R. Cherenson, F. Douglis, M.Ñ. Nelson, and B. B. Welch, "The sprite network operating system," *Computer*, pp. 23–26, Feb. 1988.

[5] D. R. Cheriton and W. Zwaenepoel, "The distributed v kernel and its performance for diskless workstations," *Proc. of the 9th ACM Symposium on Operating Systems Principles*, pp. 128–139, Oc. 1983.

[6] G. Cabillic and I. Puajit, "Stardust: An environment for parallel programming of networks of heterogeneous workstations," *Journal of Parallel and Distributed Computing*, vol. 40, pp. 65–80, 1997.

[7] P. Krueger and R. Chawla, "The stealth distributed scheduler," *Int'l. Conference on Distributed Computing Systems, IEEE*, pp. 336–343, 1991.

[8] Andreas Polze, "How to partition a workstation," *Proc. IAESTED International Conference Parallel asd Distribted Computing ans Systems '87, 12th Int'l Symp. Computer Performance Modeling, Measurement, and Evaluation, North-Holland, Amsterdam*, pp. 184–187, 1996.

[9] X. Zhang and Y. Yan, "Modeling and characterizing parallel computing performance on heterogeneous networks of workstations," *Proc. Seventh IEEE Symp. Parallel and Distributed Processing*, pp. 25–34, 1995.

[10] D. Ferrari and S. Zhou, "An empirical investigation of load indices for load balancing applications," *Proc. Performance '87, 12th Int'l Symp. Computer Performance Modeling, Measurement, and Evaluation, North-Holland, Amsterdam*, pp. 515–528, 1987.

[11] International Business Machines Corporation, *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*, May 1988.

[12] A. L. Beguelin, J. J. Dongarra, A. Geist, R. J. Mancheck, and V. S. Sunderam, "Heterogeneous network computing," *Sixth SIAM Conference on Parallel Processing*, 1993.

[13] A. L. Beguelin, A. Geist, R. J. Mancheck, and V. S. Sunderam, "Integrated pvm framework supports heterogeneous network computing," *Computers in Physics*, 1993.

[14] A. L. Beguelin, J. J. Dongarra, A. Geist, R. J. Mancheck, S. W. Otto, and J. Walpole, "Pvm: Experiences, current status and future direction," *Supercomputing '93 Proceedings*, pp. 765–776, 1993.

[15] M. Beck, H. Bohme, and et al., *Linux Kernel Internals*, Addison-Wesley, 1997.

[16] Sun Microsystems Inc., *System Adiministration Guide, Volume II*, November 1995.

[17] S. White, A. Alud, and V. S. Sunderam, "Performance of the nas parallel benchmarks on pvm based networks," RNR 94-008, Emory University, May 1994, 27 pages.

[18] Mart L. Molle, "A new binary logarithmic arbitration method for ethernet," Technical Report CSRI-298, Computer Systems Reserach Institute, University of Toronto, 1994.

[19] ISO/IEC International Standard 8802-3, Information Technology-Local and metropolitan area networks, *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, Fourth edition*, 1993.

[20] D. L. Mill, "Network time protocol (version 3) specification implentation and analysis," RFC 1305, University of Delaware, March 1992, 113 pages.

[21] D. L. Mill, "Improved algorithms for syncrhonizing computer network clocks," *IEEE/ACM Transactions on Networks*, vol. 3, no. 3, pp. 245–254, 1995.