UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Análisis de las oportunidades para el ahorro de potencia en las redes de comunicación de data centers

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Antoan Petrov Petrov

Tutor: Marina Alonso Díaz

Salvador Coll Arnau

2015/2016

Resumen

En el proyecto se realizará un estudio del tráfico en data centers a partir de trazas reales, procedentes de programas diseñados para evaluar supercomputadores paralelos. Para ello, se hará uso de un simulador de redes de interconexión a nivel de flit con un mecanismo que efectúa una conexión y desconexión dinámica de los enlaces de la red, en función del tráfico.

Se mapeará dicho tráfico sobre topologías de red empleadas en grandes centros de datos, concretamente un fat-tree, para evaluar la utilización de los enlaces. El objetivo es medir el impacto que tiene el mecanismo empleado para la técnica de ahorro sobre el consumo de la potencia de la red, calculando para ello la energía empleada, sin que esto afecte demasiado a la latencia.

Palabras clave: centro de datos, redes de comunicación, consumo, ahorro

Abstract

This project performs a research about data center's traffic starting from real traces, which are taken from programs designed to evaluate parallel supercomputers. For this, it will be used an interconnection network simulator, with a mechanism that dynamically switches on and off network links as a function of traffic.

The traffic will be mapped on highly used data center's network topologies, a fat-tree in particular, to evaluate the link using. The objective is to measure the efficacy of the mechanism used to apply the strategy for reducing the network power consumption, calculating the spended energy, without altering significantly the latency.

Keywords : data center, communication networks, consumption, saving

Índice

Índice de contenido

1. Introducción

El crecimiento del tamaño y la complejidad de las aplicaciones ha crecido con el alto nivel de potencia de cálculo que ofrecen los sistemas paralelos. Los elementos de la infraestructura que concentra estos supercomputadores conocida como *data center*, generan un consumo elevado, por lo que es necesario controlarlo a todos los niveles.

Se ha hecho un gran esfuerzo en reducir el consumo de los data centers y supercomputadores. Tenemos como referencia la lista Green50 que nos muestra los supercomputadores más eficientes del mundo desde el punto de vista energético. Sin embargo, a donde se ha enfocado ese esfuerzo es a las partes que más participan para hacer los cálculos, como el procesador o la memoria, pero no a las redes. Para dar soporte al rendimiento requerido en los centros de datos y la enorme cantidad de tráfico que circula por sus redes, se han incorporado routers y switches más potentes y con mejores prestaciones en su cálculo y su tasa de envío. El hecho de haber ido incorporando estos routers también ha despertado la inquietud por su consumo, teniendo en cuenta que los enlaces consumen una gran parte de la potencia. Nuestro trabajo consistirá pues, en hacer un estudio sobre la posibilidad de reducir este consumo.

Este proyecto partirá de la tesis de Marina Alonso y el trabajo realizado por su grupo de investigación. En esta, se define una estrategia que permite la reducción de consumo de potencia en redes de interconexión. Para ello se desarrolla un mecanismo que se aplica sobre topologías ampliamente utilizadas en data centers, como es el caso de las topologías directas, como la malla y el toro, y las topologías indirectas como los *fat-tree.* Este mecanismo efectúa una conexión/desconexión de los enlaces en función del tráfico existente en la red y gobernado por una pareja de umbrales on/off, sin modificar el algoritmo de encaminamiento y sin que esta desconexión de enlaces tenga un impacto significativo a las prestaciones de la red.

Dicho estudio se ha realizado sobre un simulador de redes de interconexión que implementa este mecanismo, sobre el cual se han elaborado distintas simulaciones inyectando al simulador tráfico sintético para cada topología y pareja de umbrales. Los resultados demuestran que el mecanismo es efectivo, y proporciona reducciones del consumo de potencia con un impacto ligero sobre la latencia.

1.1 Objetivos

Cabe destacar que es un campo en el que no hay previos estudios, puesto que hasta ahora no se han hecho pruebas con este tipo de trazas. Se conocen los efectos que tiene el mecanismo con cargas sintéticas, pero es necesario hacer una evaluación más precisa de este utilizando cargas no sintéticas, pues es la mejor manera de recopilar el comportamiento real que tienen los programas.

Para el proyecto se nos proporciona ya el simulador, por lo que nuestro objetivo es evaluar la eficiencia del mecanismo empleando para ello como entrada trazas con tráfico real. Utilizaremos las llamadas trazas en formato VEF, que recogen las comunicaciones MPI realizadas por los nodos de programas utilizados para evaluar la eficiencia de supercomputadores paralelos (también conocidos como benchmarks).

Para ello, se inyectarán el tráfico en el simulador proporcionado y se realizarán unas primeras simulaciones con el mecanismo desactivado sobre una red *fat-tree* de 64 nodos. Después de esto, se realizarán unas segundas simulaciones con el mecanismo activado con cada una de las parejas de posibles umbrales que controlan la conexión y desconexión de los enlaces de la red, y se utilizarán valores como el tiempo y la potencia para calcular la energía empleada y medir la efectividad de la estrategia.

Así pues, este trabajo constará en una primera y más breve fase de estudio teórico del mecanismo y su comportamiento con el modelo de tráfico sintético, seguida de una fase experimental en la que se realizarán las simulaciones utilizando tráfico real y se observarán los resultados. Como trabajo intermedio, se realizará también un manual de usuario del simulador y del entorno remoto donde se llevarán a cabo las simulaciones.

2. Data centers

Se denomina data center, conocido también como centro de procesamiendo de datos (CPD), a aquella infraestructura acondicionada que concentra sistemas de información y componentes asociados, en la que estos permiten un alto nivel de rendimiento y de procesamiento de datos. Es creado y mantenido por grandes organizaciones con objeto de tener acceso a la información necesaria para sus operaciones. Los dos recursos de los que dispone un centro de datos son los computadores y las redes de comunicación. [1]

La necesidad de optimizar el espacio ha hecho que los recursos, es decir, tanto los servidores, equipos de almacenamiento y elementos de la red se concentren en racks. Esta infraestructura requiere de un entorno físico especial, puesto que va a mantener una gran cantidad de equipamiento electrónico. Un data center debe disponer de una instalación específica de refrigeración para mantener una temperatura baja, necesaria para evitar el sobrecalentamiento en la sala. Aparte de esto, también necesita una alimentación eléctrica estabilizada e ininterrumpida para garantizar la alta disponibilidad, sistemas contra incendios, etc. Con todos estos elementos, el consumo de energía eléctrica de un CPD es muy elevado, convirtíendolo en el problema más importante. La métrica más utilizada para determinar la eficiencia energética de un centro de datos es el PUE (powe usage effectiveness) que comprende los gastos en mantenimiento y administración de los servidores, y de la energía para alimentarlos y refigerarlos. Es un ratio de la potencia del centro de datos total (lo consumido por los equipos de IT, además de refrigeración, consumo de luz, etc.) dividido por la potencia utilizada para el equipo de IT. El valor medio del PUE en los centros de datos de Estados Unidos en el 2007 era de 2,0 [2], lo que significa que por cada 2 vatios de potencia total, 1 va destinado para el equipamiento informático.

Los supercomputadores alojados en los data centers suponen un coste muy grande, tanto el hardware necesario para ofrecer unas prestaciones de petaflops (orden de 10^15 operaciones de como flotante por segundo). Hemos visto que la mayor parte de los recursos van destinados a los computadores y al coste asociado a los sistemas de enfriamiento, pero nos olvidamos de otro elemento que forma parte de los data centers, las redes de comunicación. Para dar servicio al tráfico generado por una gran cantidad de nodos y mantener una baja latencia, es necesario disponer de conmutadores (routers y switches) capaces de soportarlo. La tecnología más ampliamente utililizada es Infiniband, capaz de ofrecer una velocidad de 2Gbps hasta 96Gbps, dependiendo de la agrupación de los enlaces. [3]

Los fabricantes de esta clase de conmutadores, como Mellanox, ya empiezan a preocuparse por el consumo de los enlaces cuando la red tiene un uso bajo. Ellos proponen una estrategia en sus switches en el que se reduce la velocidad del enlace en momentos de baja utilización, reduciendo así el consumo de potencia. [4]

3. Topologías

La topología de una red describe el modo en el que sus elementos se conectan unos a otros. Una red es regular si todos los nodos tienen el mismo grado, es decir, el número de enlaces que conectan un nodo con sus nodos vecinos es el mismo. Este tipo de redes escalan mejor a un mayor número de nodos. Dentro de las redes regulares se encuentran las redes directas e indirectas.

- Redes directas, en las que tenemos conectado cada nodo o procesador y donde cada nodo tiene al menos un enlace en cada dirección. Un ejemplo de este tipo de redes son los toros y las mallas. Este tipo de topologías viene definido por el número de dimensiones y el número de nodos de cada dimensión. Los nodos se sitúan en un espacio de *n* dimensiones con *k* nodos en cada dimensión. También se usa la nomenclatura de *k*-ary *n*-mesh para las mallas y k-ary n-cube para los toros, por lo que en el caso de las Figuras 1 y 2 tendríamos una 8-ary 2-mesh y un 4-ary 2-cube. [5] Los toros son similares a las mallas, pero a diferencia de estas, los nodos en los extremos de cada dimensión están conectados entre sí.

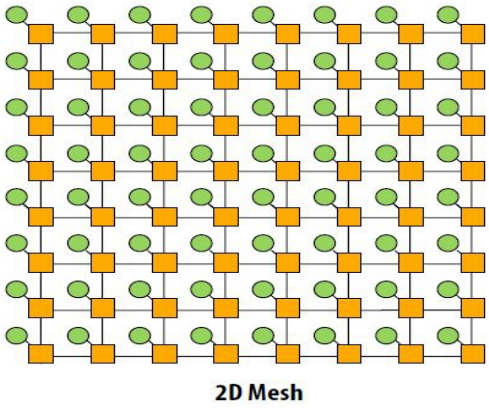


Figura 1: Malla 2D de 8x8 nodos.

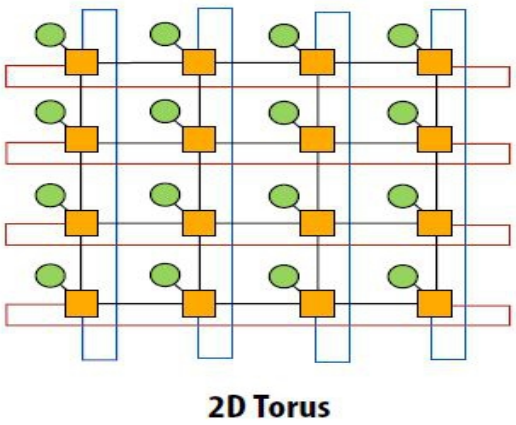


Figura 2: Toro 2D de 4x4 nodos.

- Redes indirectas en las que tenemos los nodos conectados sólo al primer nivel de switches y la forma de comunicarse unos con otros es atravesando varios niveles de switches. El ejemplo más popular de este tipo de redes son los fat-tree.

El próximo apartado está dedicado exclusivamente a las redes indirectas. Las simulaciones realizadas y evaluadas en este proyecto se han hecho únicamente para este tipo de topología regular, concretamente para un fat-tree.

3.1 Redes indirectas

La topología ideal en una red, es la que permite disponer de una conexión directa entre cada pareja de nodos. Esto se traduce a que si disponemos de *N* nodos, haríamos uso de un único switch con *N* x *N* enlaces (también conocido como *crossbar*). Obviamente, esta topología es inviable para redes de gran tamaño. Por ello se ha propuesto un gran número de topologías en los que se consigue resolver el problema de la escabilidad, a costa de que el mensaje deba atravesar varios niveles de switches para alcanzar su destino.

Estos switches suelen ser idénticos y se organizan en un conjunto de etapas, cada una conectada a la anterior y a la siguiente utilizando patrones regulares (por eso mismo, este tipo de redes se les conoce también como *multietapas*). Cada switch en una red indirecta puede conectarse a cero, uno o más nodos. Sólo los switches conectados a algún nodo pueden ser el origen o el destino de un mensaje. Además, transmitir un mensaje de un nodo a otro requiere atravesar el enlace entre el nodo origen y el switch al que se conecta, y el enlace entre el último switch de la ruta y el nodo destino.

De entre todas las topologías de redes indirectas propuestas, en la práctica la topología más ampliamente utilizada es la topología fat-tree (*k*-ary *n*-tree). Es interesante justificar por qué se utiliza la topología de los fat-tree frente a otras redes indirectas que comparten la característica de la multietapa. [6]

En el 2008, Mohammad Al-Fares, Alexander Loukissas y Amin Vahdat exponen en su documento “A Scalable, Commodity Data Center Network Architecture” que la mayoría de arquitecturas de centros de datos consiste en dos o tres niveles de switches o routers. Los switches que se conectan directamente con los hosts son GigE mientras que los switches de la capa media y superior son de 10GigE.

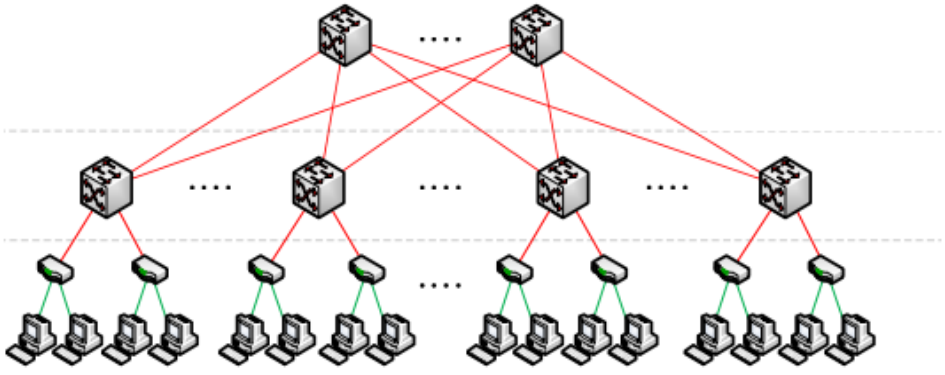


Figura 3: Esquema de una red indirecta jerárquica.

En primer lugar, exponen la enorme diferencia de precio entre los switches de 10 GigE y los de GigE. En el 2008, un switch de 10GigE de 128 puertos tenía un coste de 1.8K $/GigE mientras que un switch de GigE sólo tiene un coste de 0.3K $/GigE.

Por último, y es el detalle que más interesa de cara a este proyecto, es el consumo necesario y el calor disipado de la arquitectura jerárquica de los data centers frente a la necesaria con la arquitectura fat-tree. [7]

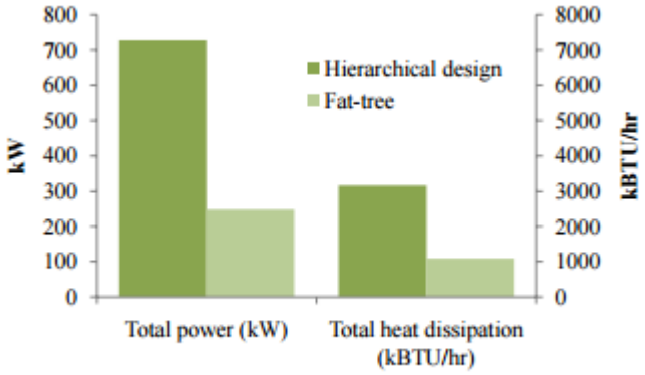


Figura 4: Comparación del consumo de potencia entre un diseño jerárquico frente a un fat-tree.

3.1.1 Fat-trees

La nomenclatura para los fat-tree es parecida a la usada para las redes directas. Se utiliza la *k* para definir el número de switches *k* x *k*, y *n* para el número de niveles o dimensiones.

Así pues, un *k-ary n-tree* está formado por dos tipos de vértices : N = k^n nodos de procesamiento y S = nk^(n−1) switches k × k (un k-ary n-tree de dimensión n = 0 está compuesto solo por un nodo de procesamiento).

Cada switch tiene 2*k* enlaces de salida de los cuales *k* están conectados a los switches o nodos de procesamiento del nivel *l* + 1 (enlaces descendentes) y los restantes *k* a los switches del nivel *l* −1 (enlaces ascendentes).

A continuación se muestra un ejemplo de un fat-tree cuaternario de 3 niveles. Es una red con 64 nodos (N=4^3) y 48 switches (3x4^(3-1)).

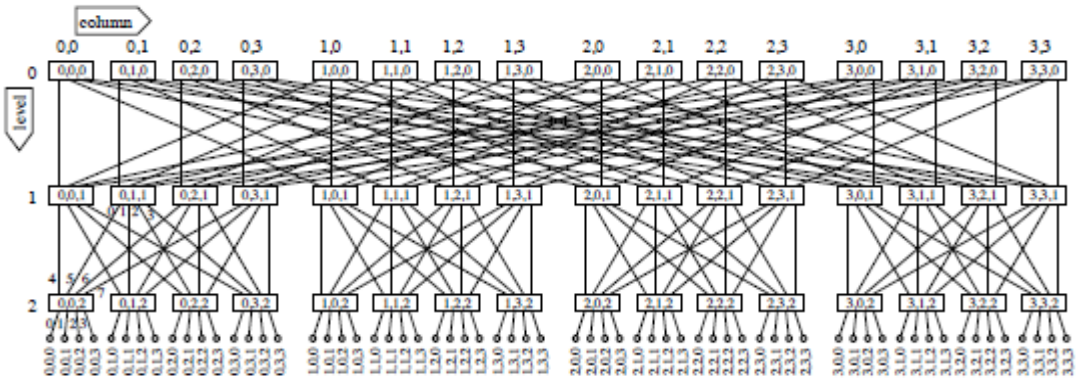


Figura 5: 4-ary 3-tree.

En la comunicación de dos nodos, el mensaje sufre dos fases de encaminamiento: una en la que es enviado desde el nodo hasta el switch ancestro más cercano, y otra en la que el mensaje desciende hasta el nodo destino.

En ésta segunda fase descendente, el camino es único para llegar al nodo, mientras que en la primera tenemos varias rutas alternativas para llegar al ancestro más cercano. Esto abre la posibilidad de utilizar un algoritmo adaptativo en que se elija el enlace de acuerdo con el estado local del switch, evitando así los enlaces congestionados. [8]

4. Mecanismo Low Power

La idea del mecanismo es cambiar el estado de los enlaces a enlaces a encendido y apagado (*on/off*), en función del tráfico de la red. Los enlaces son bidireccionales y se pueden conectar o desconectar en una determinada dirección, ascendente o descendente. Para hacer eso, cada switch de la red mide el tráfico saliente y controla el estado de los enlaces de salida dependiendo de las variaciones de tráfico.

Hay un subconjunto de los enlaces de la red, que se definen como Árbol Mínimo que no se pueden apagar para mantener la conectividad de la red. [8]

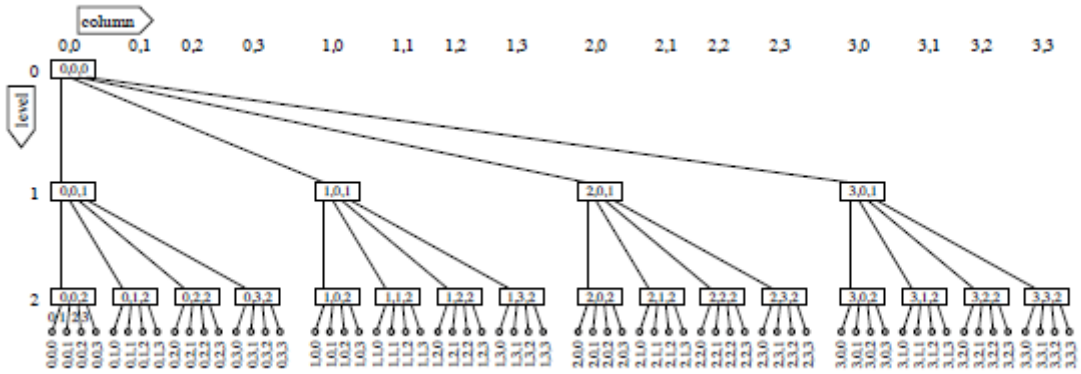


Figura 6: Árbol Mínimo.

Se sigue el siguiente criterio para realizar la conexión y desconexión de los enlaces:

- La decisión de conectar o desconectar un enlace ascendente se propaga en ambas direcciones y para ello se usa la utilización de los enlaces. De forma ascendente para garantizar que haya al menos un camino a los switches del primer nivel, y de forma descendente por la misma razón. [8]

- En los enlaces descendentes, el encendido y el apagado se hace con todos al mismo tiempo. Cuando se haya detectado que el switch no puede recibir tráfico descendente (esto ocurre cuando todos sus enlaces de entrada se han desconectado) se desconectan todos. Cuando al menos un enlace cualquiera de entrada se haya conectado, se conectarán todos los descendentes de nuevo. [8]

Para hacer efectiva la conexión y desconexión de los enlaces, una pareja de umbrales de utilización permite ajustar las características de la estrategia de ahorro de potencia: *Uoff* es el umbral de desconexión y *Uon* es el umbral de conexión.

Un nivel de tráfico que implique una baja utilización de los enlaces de un switch (por debajo de *Uoff*) provocará la desconexión de enlaces, mientras que un tráfico elevado (por encima de *Uon*) indicará al mecanismo que debe conectar más enlaces.

El comportamiento de un switch estará condicionado por la pareja de umbrales *Uon* y *Uoff* y por su posición en la red, es decir, si forma parte del árbol mínimo o no.

- Switches que pertenecen al árbol mínimo. Los enlaces que pueden ser desconectados son únicamente aquellos que no pertenecen al árbol mínimo. Es decir, todos los enlaces ascendentes menos uno; en particular, los enlaces numerados entre $k+1$ y $2k-1$. Los enlaces descendentes no se pueden desconectar porque pertenecen al árbol mínimo y proporcionan la conectividad necesaria hacia los nodos. De la misma manera, el enlace k no se puede apagar porque pertenece al árbol mínimo y proporciona la conectividad mínima hacia el nivel superior de switches. [8]

- Switches que no pertenecen al árbol mínimo. Para los enlaces de dirección ascendente se aplica la misma técnica de conexión/desconexión que para los switches del árbol mínimo. Cuando un switch detecta que uno de sus enlaces de entrada en dirección ascendente ha sido desconectado entonces desconecta un enlace de subida. Si todos los enlaces de entrada desde el nivel inferior se desconectan, todos los enlaces de salida hacia el nivel superior se pueden desconectar, puesto que el switch no va a recibir tráfico en dirección ascendente. Cuando un switch detecta que un enlace de entrada desde el nivel inferior está siendo conectado inicia la conexión del correspondiente enlace de subida. [6]

  Sin embargo, para los enlaces descendentes no se puede aplicar el mismo comportamiento ya que no pueden desconectarse de forma independiente. Mientras que un switch tenga enlaces de entrada activos desde los niveles superior o inferior, todos los enlaces descendentes tienen que estar activos.

Como ya se ha comentado, el mecanismo que activa la conexión y desconexión de los enlaces está gobernado por los umbrales *Uoff* y *Uon.*

Se define el valor medio de esta pareja de umbrales *Uavg* como (*Uoff* +*Uon*)/2. Con un valor alto de *Uavg* tendremos una política de reducción de consumo más agresiva, puesto que con cargas altas, los enlaces se desconectarán y seguirán manteniendo ese estado. El caso opuesto, con un valor de *Uavg* bajo,el mecanismo conectará los enlaces con cargas bajas.

A su vez, la diferencia entre *Uoff* - *Uon* dictará la sensibilidad del mecanismo. Con valores elevados, el mecanismo será poco sensible y hará falta variaciones significativas en el tráfico para que se produzca un cambio en el estado de los enlaces. En el caso contrario, las pequeñas variaciones en la utilización de los enlaces harán que se produzcan cambios en estado de estos. [8]

Como restricciones en los valores de los posibles umbrales, hay que nombrar las siguientes:

- Ambos valores deben ser positivos y mayores que cero.
- *Uon* debe ser mayor que *Uoff,* y menor que la carga mas alta aceptada por la red (definida como *Umax*). De no ser así, la red entraría en saturación antes de intentar conectar un enlace.
- La diferencia entre los umbrales debe ser suficiente para evitar la presencia de ciclos de conexión/desconexión.

4.1 Umbrales estáticos y dinámicos

Como ya se ha explicado anteriormente, el mecanismo hace uso de la pareja de umbrales estáticos *Uoff* y *Uon.* El mecanismo se basa en la utilización de los enlaces ascendentes considerando el ancho de banda disponible y hace una comparación con los umbrales establecidos.

Sin embargo, cuando se desconecta un enlace pero la cantidad de tráfico sigue siendo el mismo, tenemos un mayor uso de los enlaces. Por ejemplo, en un fat-tree cuaternario, si desconectamos uno de los cuatro enlaces ascendentes activos, la utilización de los tres enlaces restantes será 4/3 veces mayor que antes de desconectarlo. El nuevo valor de utilización es el que se será utilizado para compararlo con los umbrales (umbral efectivo). La limitación de los umbrales estáticos es que *Uon* debe ser dos veces mayor que *Uoff,* por lo que en casos donde hay varios enlaces activos el mecanismo será menos agresivo. [8]

Existe una versión alternativa del mecanismo con umbrales dinámicos. La idea es que cada switch distribuya el tráfico ascendente entre los enlaces cuando la red esté activa. El umbral de *Uon* se mantiene fijo, mientras que es de desconexión se calcula dinámicamente en función del número de enlaces de salida activos. Esto se convierte en una mejora para el ahorro de potencia frente a los umbrales estáticos puesto que la utilización de un switch con enlaces desconectados aumenta.

A continuación se muestra el mapa de las parejas de posibles umbrales utilizados para las simulaciones de este proyecto, ordenados siguiendo una política más sensible a una más agresiva.

Threshold	Uoff	Uon
1	0,030	0,150
2	0,110	0,230
3	0,055	0,285
4	0,145	0,375
5	0,235	0,470
6	0,085	0,435
7	0,175	0,525
8	0,265	0,615
9	0,345	0,695
10	0,025	0,495
11	0,115	0,585
12	0,205	0,675
13	0,060	0,640
14	0,200	0,320
15	0,290	0,410
16	0,380	0,500
17	0,460	0,580
18	0,325	0,555
19	0,405	0,635

Figura 7: Valores de los posibles umbrales.

4.2 Modelo de red

El simulador usado modela una red basada en wormhole a nivel de flit (la unidad de transferencia a nivel de enlace). Wormhole es una técnica de control de flujo que se ejerce sobre cada uno de los flits, en la que si el mensaje no puede continuar, no se almacena, sino que permanece bloqueado y ocupando recursos. Por ello se propone también el uso de canales virtuales, que consisten en asociar varios tampones a cada canal físico, de manera que varios mensajes que estén bloqueados puedan estar utilizando el canal. Los canales físicos del simulador se dividen en tres canales virtuales, cada canal virtual tiene asociado un buffer con capacidad para cuatro flits.

Se utiliza un algoritmo de encaminamiento adaptativo que consta de dos fases. En la primera se sigue una ruta adaptativa en direción ascendente hasta uno de los switches ancestro común más cercanos, y en la segunda se sigue una ruta determinista (ya que sólo existe un camino) en dirección descendente hasta el destino. [8]

5. Benchmark NAS

Los benchmarks NAS (Numerical Aerodynamic Simulation), son un conjunto de programas diseñados para evaluar el rendimiento de supercomputadores paralelos. Estos programas se centran en problemas de la supercomputación paralela para aplicaciones de aerodinámica.

Dentro los benchmarks NAS, nuestro trabajo se centra sobre los NPB (NAS Parallel Benchmarks), usados para el estudio de procesadores altamente paralelizables. Los NPB están formados por cinco programas principales o kernels (EP, MG, CG, FT e IS) y tres programas o simulaciones de dinámicas de fluidos (BT, SP, LU). [9]

Para cada programa, existe una clase definiendo el tamaño del problema. Para nuestras simulaciones, se ha utilizado la clase A, correspondiente a problemas de tamaño estándar. [10]

Benchmark Name	Abb.	Class A		
		Nominal Size	Operation Count (x 10^9)	Mflop/s CRAY Y-MP/1
Embarrassingly Parallel	EP	2^{28}	26.68	211
Multigrid	MG	256^3	3.905	176
Conjugate Gradient	CG	$14x10^3$	1.508	127
3-D FFT PDE	FT	256^2x128	5.631	196
Integer Sort	IS	$2^{23}x2^{19}$	0.7812	68
LU Simulated CFD Appl.	LU	64^3	64.57	194
SP Simulated CFD Appl.	SP	64^3	102.0	216
BT Simulated CFD Appl.	BT	64^3	181.3	229

Figura 8: Tamaño del problema de clase A.

Del set completo de programas, se han utilizado para las pruebas realizadas en este proyecto las mostradas a continuación. Se muestra también para cada una de ellas el patrón de comunicación entre los nodos, para representar la localidad del tráfico. Para ello, se ha procesado el fichero de trazas por medio de dos clases sencillas de Java que se adjuntan en el Anexo 1. En una primera pasada, se despliegan las operaciones colectivas en operaciones de envío punto a punto, y en la segunda se contabilizan los bytes enviados entre cada pareja de nodos origen y destino.

- MG Benchmark. Este benchmark consiste en un método (el propio método se llama Multrigrid) que resuelve una ecuación diferencial parcial 3D de Poisson. La traza de MG está formada principalmente por operaciones punto a punto. La línea en el eje X muestra una o varias operaciones Reduce, puesto que los 63

nodos envían información al nodo raíz, en este caso el 0. La línea del eje Y es propia de un Broadcast, pues el nodo origen 0, transfiere bytes al resto de los 63 nodos.

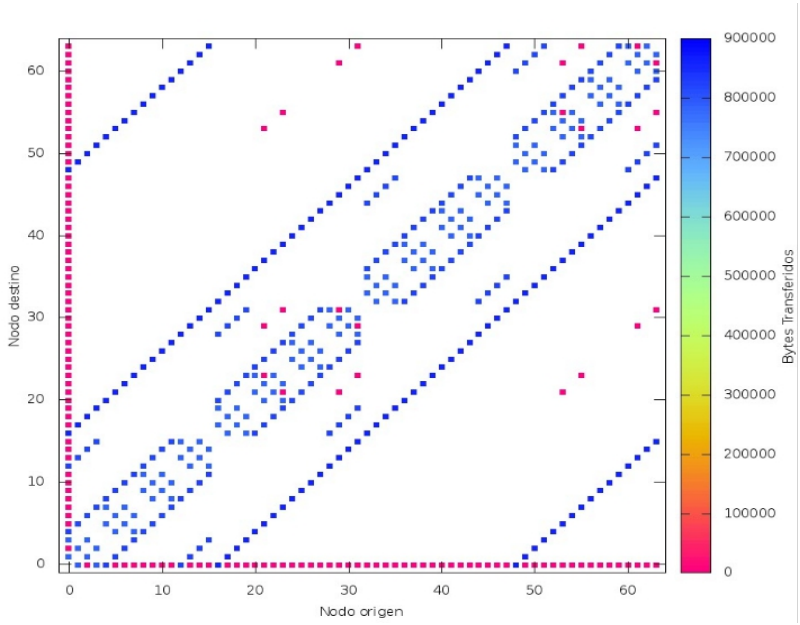


Figura 9: Patrón de comunicación de la traza MG.

- FT Benchmark. Este método vuelve a resolver una ecuación diferencial parcial 3D pero con el método de Transformación Rápida de Fourier (FFT). El fichero de trazas de FT consta principalmente de operaciones AlltoAll, por eso mismo se observa un patrón de comunicación en el que todos los nodos son origen y destino de varios mensajes.

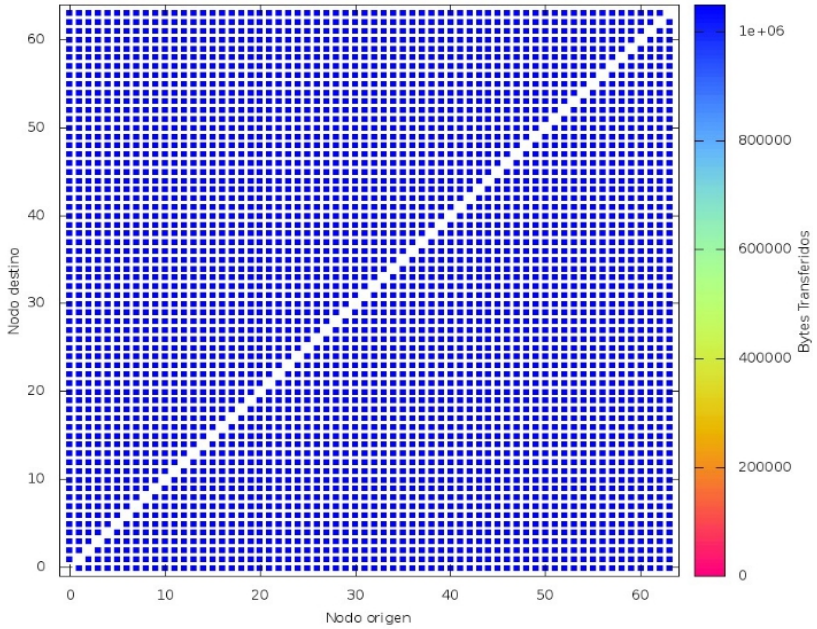


Figura 10: Patrón de comunicación de la traza FT.

- LU Benchmark. Utiliza el método de la “lower-upper” diagonal para resolver un sistema triangular. De nuestro conjunto de programas, LU es el fichero de trazas que más ocupa y el que tiene mayor número de comunicaciones entre los nodos.

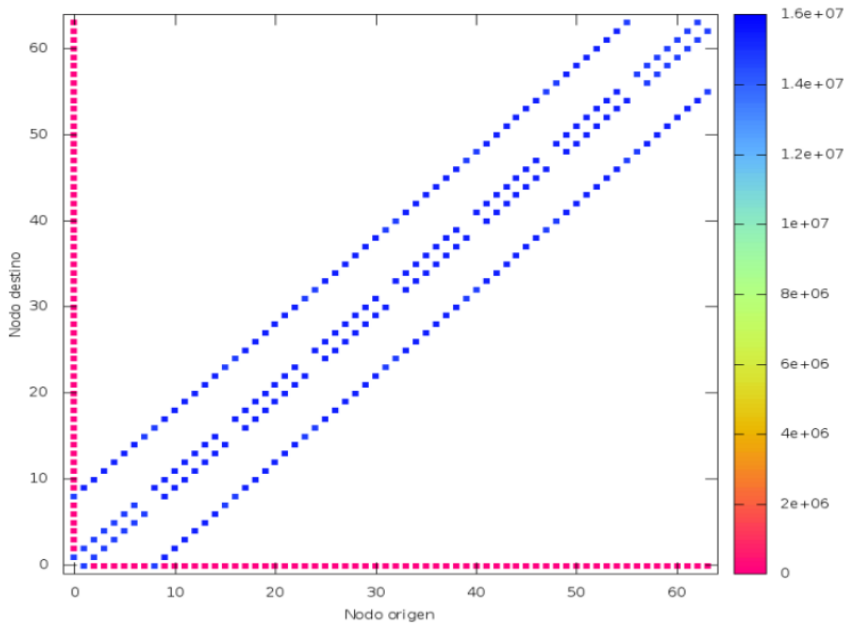


Figura 11: Patrón de comunicación de la traza LU.

- BT Benchmark. En este benchmark se resuelven sistemas de ecuaciones diagonalmente no dominantes, con bloques de ecuaciones tridiagonales. El patrón de BT es como el de FT, en el que predominan las operaciones AlltoAll.

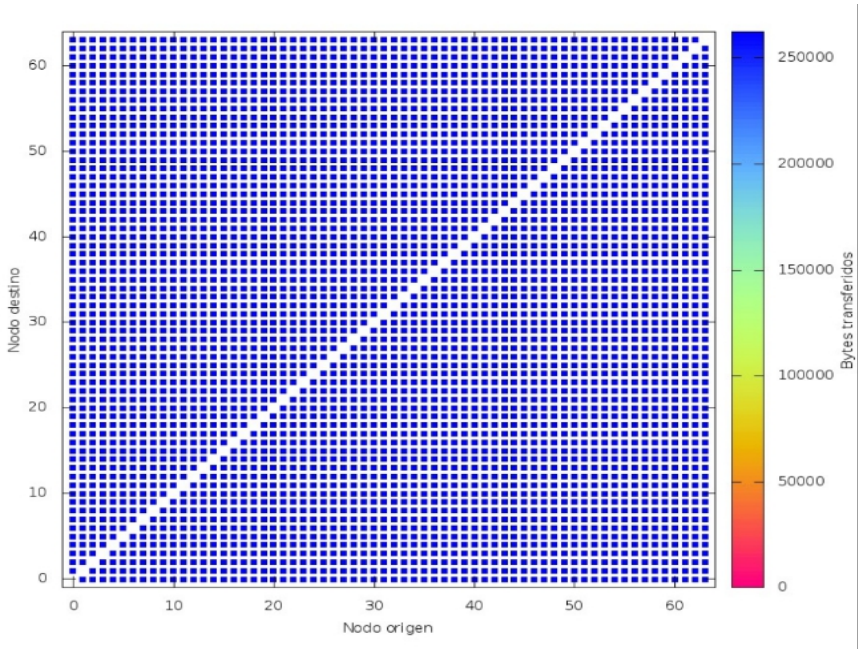


Figura 12: Patrón de comunicación de la traza BT.

6. Trazas VEF

Las trazas VEF contienen todas las comunicaciones que se realizan durante la ejecución de una aplicación MPI. Se utilizan para modelar el tráfico y las comunicaciones punto a punto y colectivas, para ser utilizado en cualquier simulador de red.

Normalmente, las trazas de tráfico contienen el tiempo absoluto desde que el mensaje se genera en los nodos, y después cuando se utilizan en la red para la simulación, el tiempo total de ejecución que depende del último mensaje enviado de la traza. Sin embargo, las trazas VEF contienen información sobre la relación entre ellas. Excepto el primer mensaje, todos los demás dependen de un mensaje enviado o recibido, o de una operación colectiva.

Así pues, todos los mensajes contienen un campo indicando el tipo de dependencia y el mensaje del que depende. A parte de este tiempo, también se necesita un tiempo para procesar el paquete en la red (que depende de la capacidad de procesamiento de la CPU, lectura y escritura en la memoria secundaria, etc..). [11]

6.1 Formato de las trazas VEF

Una traza VEF se compone de tres partes:

1. Cabecera. Contiene información básica sobre la traza
2. Comunicadores (COMMs). Es un grupo de tareas que se comunican en una operación colectiva.
3. Cuerpo. Contiene los mensajes generados por las tareas. Hay dos tipos de mensajes:
    - Aquellos que corresponden a comunicaciones punto a punto.
    - Aquellos que corresponden a comunicaciones colectivas.

6.1.1 Cabecera

La cabecera es la primera línea que se observa en la traza VEF. [11] El formato que tiene es el siguiente:

VEF NumTask NumMsg NumCOMM NumGlobalGroupComm NumLocalGroupComm onlySendFlag

donde:

- *NumTask* es el número de tareas MPI.
- *NumMsg* es el número de mensajes punto a punto.
- *NumCOMM* es el número de comunicadores (COMMs).
- *NumGlobalGroupComm* es el número de comunicaciones colectivas globales.
- *NumLocalGroupComm* es el número de comunicaciones colectivas locales. Por ejemplo, un broadcast entre cuatro tareas supone una comunicación colectiva global y cuatro comunicaciones colectivas locales.
- *OnlySendFlag* es un flag que indica si todas las dependencias de mensajes punto a punto son dependencias de envío.

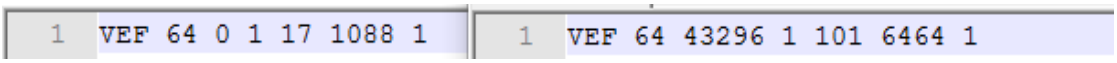


Figura 13: Cabecera de una traza VEF.

En la figura se muestran las cabeceras (la primera línea) de dos ficheros con trazas VEF. Se puede observar cómo en el primero no hay ninguna operación punto a punto (nos lo indica el segundo token con el valor 0), mientras que en el segundo éstas predominan (un valor de 43296).

También se puede ver en el primero, que consta de 17 comunicaciones colectivas globales, en los que participan las 64 tareas, por eso mismo hay 1088 comunicaciones colectivas locales.

6.1.2 Comunicadores (COMMs)

En la segunda línea de la traza tenemos la descripción de los comunicadores. Estos especifican qué tareas están involucradas en el intercambio de mensajes de una comunicación colectiva. [11] Una tarea puede usar varios COMMs para comunicarse con los distintos grupos de tareas. El formato es el siguiente:

id task_0 [task_1 ... task_N-1]

```
C0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
```

Figura 14: Comunicador formado por 64 tareas.

donde:

- *id* es el identificador del COMM. Está compuesto por la letra C y un número natural que identifica al COMM.
- *task_N* es el identificador de la tarea involucrada en el COMM.

En el caso de las trazas VEF utilizadas para este proyecto, en cada una de ellas se usa un único comunicador, formado por los 64 nodos o tareas.

6.1.3 Mensajes punto a punto

Las líneas con mensajes punto a punto representan el modelo de envío de un sólo mensaje entre dos tareas. Son generadas por funciones MPI como MPI_Send() o MPI_Recv() [11]. El formato que tienen es el siguiente:

ID src dst length Dep time IDdep

donde:

- *ID* es el identificador único para cada mensaje.
- *Src* es la tarea origen.
- *Dst* es la tarea destino.
- *Length* es el tamaño del mensaje en Bytes.
- *Dep* es el tipo de dependencia.
- *Time* es el tiempo expresado en nanosegundos que se espera para inyectar el mensaje por una dependencia.
- *IDdep* es el identificador del mensaje del cual depende.

El campo de dependencia contiene un valor basado en los siguientes valores:

0. Mensaje independiente. El mensaje se envía a tiempo.
1. Dependencia de envío. El mensaje *ID* es enviado *time* nanosegundos después del envío del mensaje *IDdep*.
2. Dependencia de recibo. El mensaje *ID* es enviado *time* nanosegundos después del recibo del mensaje *IDdep*.
3. Dependencia de comunicación colectiva. El mensaje *ID* es enviado *time* nanosegundos después de que la tarea *task* haya completado su participación en

la operación colectiva *IDdep.* En este caso, *IDdep* incluye la letra G antes del id numérico.

```
8307 0 16 34848 1 6873037 8221
8305 1 17 34848 1 7200904 8212
8283 2 18 34848 1 6428171 8201
8281 3 19 34848 1 6259994 8207
```

Figura 15: Operaciones punto a punto

En la figura se observan cuatro operaciones punto a punto. En la primera línea, la tarea 0 envía a la tarea 16 34848 Bytes. Tiene una dependencia de envío (código 1) con la tarea con ID 8221, por lo que se esperará 6873037 nanosegundos para inyectar el mensaje.

6.1.4 Operaciones colectivas

Las líneas con operaciones colectivas representan el modelo de envío de varios mensajes entre varias tareas. El formato que tienen es el siguiente: [11]

ID ID_COMM ID_OP task send_size recv_size Dep time IDdep

donde:

- *ID* es el identificador del mensaje. Para distinguirlo de los mensajes punto a punto empieza por la letra G seguido del ID. Este identificador es el mismo para todas las líneas en la traza que estén involucradas en la misma operación colectiva.
- *ID_COMM* es el identificador del COMM.
- *ID_OP* es el tipo de operación colectiva.
- *Task* es el identificador de la tarea.
- *send_size* es el número de bytes enviados por la tarea actual.
- *recv_size* es el número de bytes recibidos por la tarea actual.
- *Dep* es el tipo de dependencia.
- *Time* es el tiempo expresado en nanosegundos que se espera para inyectar el mensaje por una dependencia.
- *IDdep* es el identificador del mensaje del cual depende.

En cuanto a los tipos de operaciones colectivas que soportan las trazas VEF, son la siguientes, y se representan en la línea con éstos códigos:

0. *Broadcast*. Si la tarea actual es el root: *send_size*=X, *recv_size*=0; en caso contrario:*send_size*=0, *recv_size*=X.
1. *Reduce*. Si la tarea actual es el root: *send_size*=0, *recv_size*=X; en caso contrario:*send_size*=X, *recv_size*=0.
2. *AllReduce*. En este caso, tanto *send_size* como *recv_size* sin distintos de 0.
3. *Gather*. Si la tarea actual es el root: *send_size*=Xi, *recv_size*=SUM(Xi); en caso contrario:*send_size*=Xi, *recv_size*=0.
4. *Scatter*. Si la tarea actual es el root: *send_size*=SUM(Xi) *recv_size*=Xi; en caso *contrario*:*send_size*=0, *recv_size*=Xi.
5. *AllGather*. *send_size*=Xi y *recv_size*=SUM(Xi).
6. *Barrier*. En este caso, tanto *send_size* como *recv_size* son iguales a 0.
7. *All to all*:

   a. *All to All Básico*. *recv_size=0.* Cada tarea envía *send_size* bytes a cada una de las demás.

   b. *All to All Vectorial*. *recv_size=1.* Cada tarea envía un mensaje a la otra, pero se re parten los *send_size* bytes entre todas las tareas.

```
G6 C0 1 0 0 16 3 23152128 G5
G6 C0 1 1 16 0 3 21298699 G5
G6 C0 1 2 16 0 3 21958032 G5
G6 C0 1 3 16 0 3 21529021 G5
G6 C0 1 4 16 0 3 21455355 G5
G6 C0 1 5 16 0 3 22346406 G5
G6 C0 1 6 16 0 3 20964073 G5
G6 C0 1 7 16 0 3 20801957 G5
G6 C0 1 8 16 0 3 22039611 G5
G6 C0 1 9 16 0 3 21779238 G5
G6 C0 1 10 16 0 3 21721264 G5
```

Figura 16: Fragmento de un Broadcast.

```
G7 C0 7 0 131072 0 3 14027435 G6
G7 C0 7 1 131072 0 3 14224369 G6
G7 C0 7 2 131072 0 3 13605206 G6
G7 C0 7 3 131072 0 3 13452841 G6
G7 C0 7 4 131072 0 3 16210654 G6
G7 C0 7 5 131072 0 3 16724137 G6
G7 C0 7 6 131072 0 3 17657164 G6
G7 C0 7 7 131072 0 3 17687219 G6
G7 C0 7 8 131072 0 3 15508282 G6
G7 C0 7 9 131072 0 3 15779452 G6
G7 C0 7 10 131072 0 3 15364037 G6
```

Figura 17: Fragmento de un All to All.

En la Figura 16 se observa un fragmento de una traza VEF. Se sabe que son operaciones colectivas porque el primer campo (el identificador) empieza por la letra G. En este caso, la operación involucra al COMM C0, formado por 64 tareas. La operación es un Broadcast (código 1) y sabemos que la origina la tarea 0 ya que *send_size=0 y recv_size=X* (en este caso, 16 Bytes).

En la siguiente podemos ver otra operación colectiva, en este caso es un All to All (código 7). Todas las operaciones de esta, dependen de una comunicación colectiva (código 3).

Es muy importante destacar que todas las operaciones colectivas ya están desplegadas. Esto quiere decir que no hay un Broadcast o un All to All por cada línea que vemos, sino uno sólo, y las siguientes líneas son la consecuencia o el despliegue de esa operación colectiva.

7. Simulador previo

En este apartado se hará un manual con los pasos necesarios para compilar el simulador, crear el ejecutable y hacer una simulación.

En primer lugar, se ha de disponer una máquina con un sistema operativo Unix instalado. En nuestro caso, el entorno local de trabajo es una máquina virtual Kubuntu que se ejecuta bajo el software de virtualización Oracle VM VirtualBox.

A la hora de instalar la máquina virtual, se recomienda asignarle una cantidad de almacenamiento mínima de 100GB, ya que algunas de las trazas VEF ocupan más y en un determinado momento necesitaremos tener las trazas replicadas. En cualquier caso, si se necesita más cantidad de almacenamiento siempre se puede ampliar. Si tenemos la máquina virtual corriendo bajo VirtualBox, tenemos que abrir un terminal, navegar hasta la imagen y ejecutar el siguiente comando:

VBoxManage modifyhd NombredeHDD.vdi --resize TamañoEnMB

Después de ejecutar el comando, el nuevo espacio asignado tiene que ser particionado y formateado. Esto se puede hacer ejecutando como una ISO la aplicación *Gparted.*

Para poder trabajar con el simulador se han de realizar ciertos pasos previos de compilación y configuración. Así pues, partiendo de que ya tenemos nuestro entorno con Unix y el comprimido del simulador, tenemos que realizar lo explicado en las siguientes secciones.

7.1.1 Compilación de la librería de trazas

- Descompimir la carpeta ejecutando > tar xvzf simulator.tgz
- Situarse en la carpeta TraceLib_v2.3.1
- Ejecutar > ./configure

```
antoan@antoan-VirtualBox:~/sim-TFG/TraceLib_v2.3.1$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking whether make sets $(MAKE)... (cached) yes
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
```

- Ejecutar > make install

```
antoan@antoan-VirtualBox:~/sim-TFG/TraceLib_v2.3.1$ make install
Making install in src
make[1]: se ingresa al directorio «/home/antoan/sim-TFG/TraceLib_v2.3.1/src»
make[2]: se ingresa al directorio «/home/antoan/sim-TFG/TraceLib_v2.3.1/src»
 /bin/mkdir -p '/usr/local/lib'
 /bin/bash ../libtool   --mode=install /usr/bin/install -c   libtracelib.la '/usr/local/lib'
libtool: install: /usr/bin/install -c .libs/libtracelib.so.0.0.0 /usr/local/lib/libtracelib.so.0.0.0
/usr/bin/install: cannot create regular file `/usr/local/lib/libtracelib.so.0.0.0': Permission denied
make[2]: *** [install-libLTLIBRARIES] Error 1
make[2]: se sale del directorio «/home/antoan/sim-TFG/TraceLib_v2.3.1/src»
make[1]: *** [install-am] Error 2
make[1]: se sale del directorio «/home/antoan/sim-TFG/TraceLib_v2.3.1/src»
make: *** [install-recursive] Error 1
antoan@antoan-VirtualBox:~/sim-TFG/TraceLib_v2.3.1$
```

Si dan errores al final de la instalación, no hay ningún problema porque no se van a utilizar las librerías dinámicas sino las estáticas.

- Hay que localizar el archivo libtracelib.a que está en una carpeta oculta. Lo podemos localizar con : > find .-name '*.a'

```
antoan@antoan-VirtualBox:~/sim-TFG/TraceLib_v2.3.1$ find . -name '*.a'
./src/.libs/libtracelib.a
```

Como se observa, se obtiene como resultado del comando el directorio del archivo ./src/.libs

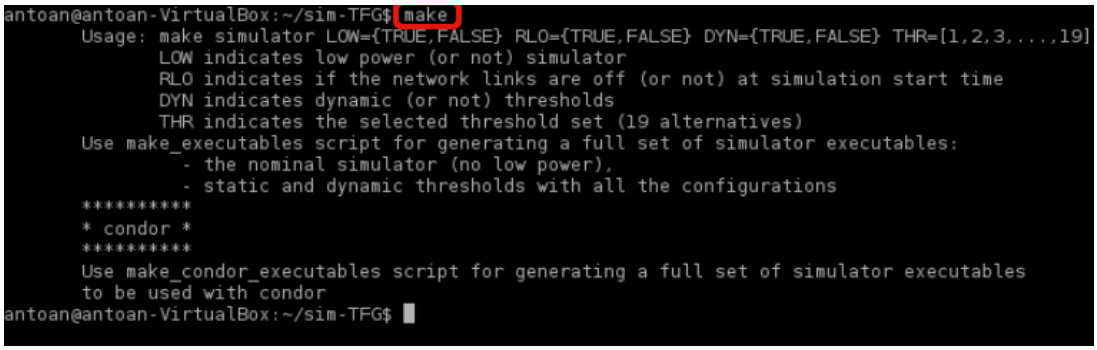
- Copiar el archivo libtracelib.a en sim-TFG/tracelib. Ésta es la librería estática que se enlazará con el simulador.

7.1.2 Compilación del simulador

- Situarse en el directorio raíz del simulador.
- > touch .depend : Crea el archivo .depend si no existe y actualiza su tiempo de creación.

- > make depend : Configura las dependencias propias del entorno de compilación.
- > make clean : Se limpian los objetos compilados anteriormente o para otra arquitectura.

- > make {nombre del ejecutable} : Compila el simulador y genera el ejecutable. Al ejecutar ">make" podemos ver los parámetros que podemos añadir al comando. Estos parámetros junto con más detalles están descritos en el fichero *"Makefile"*.



Observamos que la ayuda para el comando make nos proporciona información sobre los distintos parámetros que podemos añadir:

- El parámetro *LOW* indica si el mecanismo de ahorro del simulador está activado o no. Con LOW=TRUE nuestro mecanismo estaría activado y con LOW=FALSE estaría desactivado.

  Si ejecutásemos *make simulador* sin ningún parámetro, obtendríamos un ejecutable del simulador nominal, es decir, con el mecanismo de ahorro desactivado. Sería lo mismo que ejecutar *make simulator* LOW=FALSE.

- Con el parámetro *RLO* indicamos si al comienzo de la simulación, el estado de los enlaces es encendido o apagado. Con RLO=TRUE los enlaces estarán en off o apagados, y con RLO=FALSE los enlaces estarán en on o encendidos al inicio.

- Con el parámetro *DYN* indicamos si los umbrales que manejan la conexión y desconexión del mecanismo, tendrán un comportamiento estático o dinámico. Con DYN=TRUE los umbrales serán dinámicos y con DYN=FALSE serán estáticos.

- Por último, con *THR* indicamos el umbral o threshold (1,2,3,4...19) seleccionado.

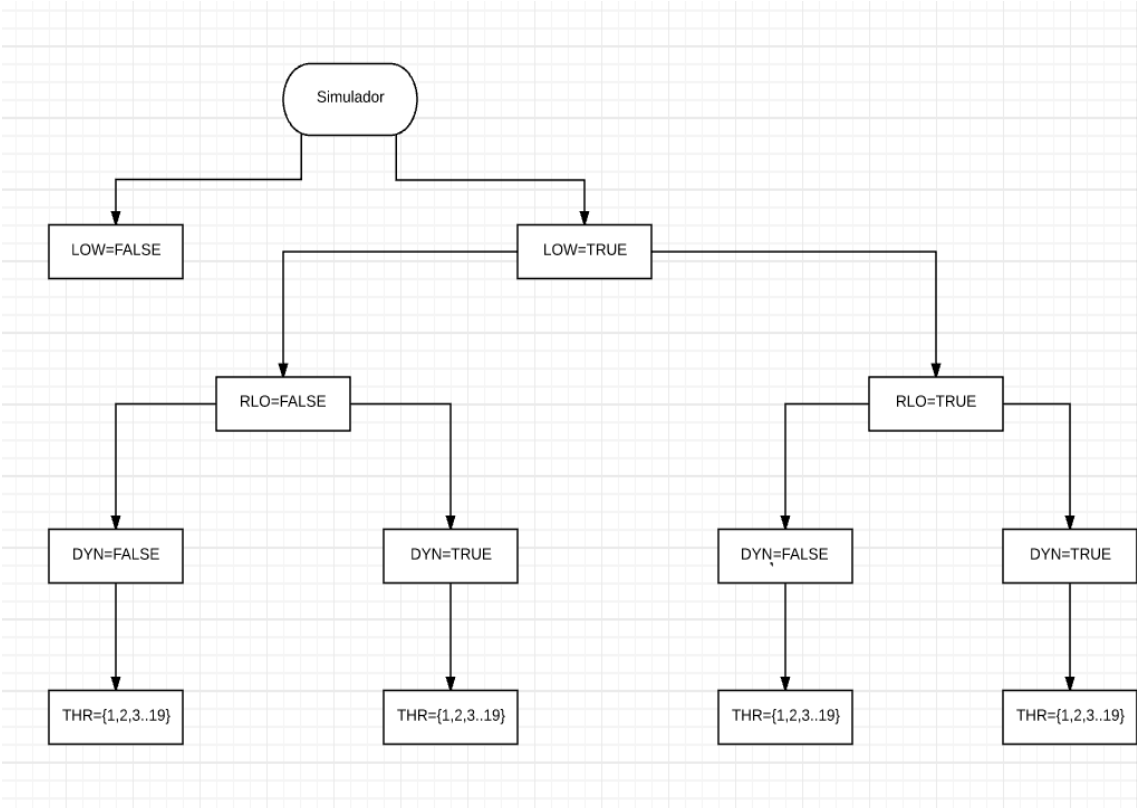


Figura 18: Posibles configuraciones de simulación.

7.1.3 Simulación

A continuación se verá el comando completo y se explicará cada elemento:

./simuladorNominal -VEFTrace ft.A.64.vef<4ary_3tree.in>ft_A_64.out

- En primer lugar tenemos *./simuladorNominal.* Es el nombre del ejecutable que hemos generado con los pasos anteriores. Lo siguiente son sus argumentos.
- Con *-VEFTrace* se le indica que vamos a usar trazas VEF y vamos a pasarle el fichero de trazas *ft.A.64.vef.*
- Con *4ary_3tree.in* se indica que la red sobre la que se ejecuta la traza es un 4-ary 3-tree (un fat-tree con switches de 4x4 y 3 niveles, 64 nodos).
- La salida se redirige a un archivo .out, por ejemplo, *ft_A_64.out.*

Como resultado de la simulación, se generan dos archivos con la extensión *.out*. El primero es el que indicamos nosotros como salida en la línea de comandos (en el ejemplo, ft_A_64.out). El segundo es un archivo oculto que podemos encontrar bajo el nombre de *OnOff_1.out*. A continuación se explicará el contenido de ambos archivos.

7.2 Archivos de salida .out

El primer archivo de salida que se genera con la simulación, y es el que indicamos en la línea de comandos.

Al comienzo de este podemos encontrar información sobre la configuración de la red: la topología seleccionada (el valor 50 indica que es un fat-tree, distinto de 50 para mallas y toros), la función de selección (un 5 indica que se elegiría el primer canal virtual del primer canal físico con mayor número de canales virtuales libres), el número de canales virtuales por canal físico (en nuestro caso 3).

La generación de los datos de entrada para las simulaciones la podemos encontrar en el fichero *genOnOff.c*. Otro archivo importante que se ha de visualizar y es en el cuál está definido el mecanismo es el *simulator.c*.

```
Resultados de la simulacion:
----------------------------

Tiempo inicial = 1179712 ; T. final ==22226040 ; T. total ==21046328

Latencia media de la cabecera =   71334.32219  Desv. tipica =   78847.72033  Valor maximo = 889158
Latencia media total =   20658.87688



SIMULADOR NOMINAL: MECANISMO ON/OFF DESCONECTADO
Utilizacion de los nodos (detector umbrales On/Off) =       0.42299
Potencia media consumida =       1.00000
Latencia media (On/Off) =   63589.50000


  Desv. tipica =  179450.43924  Valor maximo = 967041
Latencia media total desde la generacion =  122463.49696  Desv. tipica = 1110382.84241  Valor maximo = 3215985
Lat. media cabecera/Latencia media = 345.30 %%
Latencia media/Longitud media =    1291.17980
Latencia media desde la generacion/Longitud media =    7653.96856
Productividad (palabras por ciclo) =      47.52289
Productividad (palabras/ciclo/nudo) =     0.74255
```

Figura 19: Resultado de simulación con el mecanismo desconectado..

En esta parte del archivo podemos observar los resultados de la simulación. Se trata pues de una simulación hecha con el mecanismo desactivado, es decir, con el simulador nominal. Además se puede ver como la potencia media consumida es 1, ya que no se desconecta ningún enlace y no se produce ningún ahorro.

```
Resultados de la simulacion:
----------------------------

Tiempo inicial = 1175706 ; T. final ==22377826 ; T. total ==21202120

Latencia media de la cabecera =   69914.78838  Desv. tipica =   76123.35717  Valor maximo = 1074649
Latencia media total =   19458.41194



UMBRALES ESTATICOS: [ThresholdOff, ThresholdOn] = [0.235,0.470]
Utilizacion de los nodos (detector umbrales On/Off) =       0.41976
Potencia media consumida =       0.98909
Latencia media (On/Off) =   62530.23828


  Desv. tipica =  177240.27890  Valor maximo = 1172950
Latencia media total desde la generacion =   82129.36264  Desv. tipica = 1070300.70189  Valor maximo = 3060639
Lat. media cabecera/Latencia media = 359.30 %%
Latencia media/Longitud media =    1216.15075
Latencia media desde la generacion/Longitud media =    5133.08516
Productividad (palabras por ciclo) =      47.17369
Productividad (palabras/ciclo/nudo) =     0.73709
```

Figura 20: Resultado de simulación con el mecanismo conectado.

Aquí vemos otro ejemplo del mismo fragmento del archivo *.out,* esta vez con el mecanismo activado. Nos muestra que hemos elegido una configuración con umbrales estáticos, como también los valores del *ThresholdOff* y *ThresholdOn*.

Para hacer las comparaciones con los resultados de las simulaciones con el mecanismo desactivado, se puede tomar como referencia el T. final, la potencia media consumida y la latencia media.

Al final del fichero, podemos observar la utilización que han tenido los enlaces de cada nodo, con lo que también podríamos sacar conclusiones y hacer comparaciones con la utilización que han tenido estos con el simulador nominal.

```
@Utilizacion de enlaces
@Nodo  Lnk1 Lnk2 Lnk3 Lnk4 Lnk5 Lnk6 Lnk7 Lnk8 ...
@    0   60%    59%    59%    61%    0%    0%    0%    0%
@    1   68%    68%    70%    72%   60%    54%   50%    50%
@    2   0%    0%    0%    0%    68%    71%    70%    71%
@    3   62%    61%    60%    57%    0%    0%    0%    0%
@    4   72%    71%    67%    72%    63%    57%    54%    52%
@    5   0%    0%    0%    0%    68%    71%    70%    70%
@    6   60%    60%    61%    65%    0%    0%    0%    0%
@    7   70%    69%    71%    68%    62%    59%    54%    52%
@    8   0%    0%    0%    0%    68%    71%    70%    70%
@    9   60%    61%    60%    61%    0%    0%    0%    0%
@   10   70%    71%    71%    67%    60%    58%    56%    52%
@   11   0%    0%    0%    0%    69%    69%    71%    71%
@   12   56%    57%    56%    55%    0%    0%    0%    0%
@   13   71%    67%    68%    73%    58%    57%    53%    47%
@   14   0%    0%    0%    0%    69%    70%    70%    70%
@   15   57%    60%    56%    58%    0%    0%    0%    0%
@   16   75%    67%    71%    69%    59%    59%    55%    53%
@   17   0%    0%    0%    0%    70%    72%    69%    68%
@   18   53%    58%    60%    57%    0%    0%    0%    0%
@   19   65%    73%    71%    70%    62%    56%    56%    52%
@   20   0%    0%    0%    0%    70%    69%    72%    68%
@   21   57%    58%    58%    57%    0%    0%    0%    0%
@   22   68%    72%    70%    68%    59%    58%    56%    50%
@   23   0%    0%    0%    0%    68%    71%    71%    70%
@   24   53%    51%    55%    52%    0%    0%    0%    0%
@   25   75%    68%    66%    71%    61%    60%    53%    49%
@   26   0%    0%    0%    0%    71%    70%    70%    69%
@   27   55%    52%    55%    57%    0%    0%    0%    0%
```

Figura 21: Utilización de los enlaces.

Por otro lado, como ya se ha mencionado tenemos el otro fichero de salida generado *OnOff_1.out*. A diferencia del otro archivo *.out* cuyo nombre podemos elegir en el propio comando, este archivo se genera para cada simulación bajo el mismo nombre, por lo que si lanzamos varias simulaciones en el mismo directorio, estaríamos sobreescribiendo el archivo *OnOff_1.out.* En el próximo apartado se explica y se da un consejo sobre cómo evitar esto.

En él se muestra cada ciclo de reloj el tiempo, la potencia media que se consume, el tráfico expresado en flits/ciclo/nodo y el tráfico inyectado.

#Time	Power	Latency	Traffic(flits/cycle/node)	Injected Traffic(f/c/n)	ClockNumber
500	1.0000	30.2427	0.1234	239.4835	1
1000	1.0000	82.2778	0.0118	34.5621	2
1500	1.0000	0.0000	0.0000	0.0000	3
2000	1.0000	0.0000	0.0000	0.0000	4
2500	1.0000	0.0000	0.0000	0.0000	5
3000	1.0000	0.0000	0.0000	0.0000	6
3500	0.9896	0.0000	0.0000	0.0000	7
4000	0.9896	0.0000	0.0000	0.0000	8
4500	0.9896	0.0000	0.0000	0.0000	9
5000	0.9896	0.0000	0.0000	0.0000	10
5500	0.9818	0.0000	0.0000	0.0000	11
6000	0.9818	0.0000	0.0000	0.0000	12
6500	0.9818	0.0000	0.0000	0.0000	13
7000	0.9818	0.0000	0.0000	0.0000	14
7500	0.9792	0.0000	0.0000	0.0000	15
8000	0.9792	0.0000	0.0000	0.0000	16
8500	0.9792	0.0000	0.0000	0.0000	17

Figura 22: Archivo OnOff_1.out.

8. Entorno de trabajo

En este capítulo se explicará la metodología de trabajo seguida para llevar a cabo las simulaciones, y puesto que éstas se han ejecutado en una máquina en remoto, una pequeña guía del entorno.

Las primeras pruebas que se hicieron se llevaron a cabo sobre la máquina virtual Kubuntu, pero debido al tiempo necesario para terminar cada simulación y tener encendido el ordenador personal sin interrupción, se decidió realizar las pruebas en remoto en uno de los nodos del departamento GAP.

8.1 Condor

Para realizar una simulación, se necesita lanzar la carga al scheduler (en castellano planificador, un componente del sistema que reparte la tareas al procesador) Condor en el nodo de cómputo, creando para ello un fichero de submit. Este fichero debe tener las siguientes entradas:

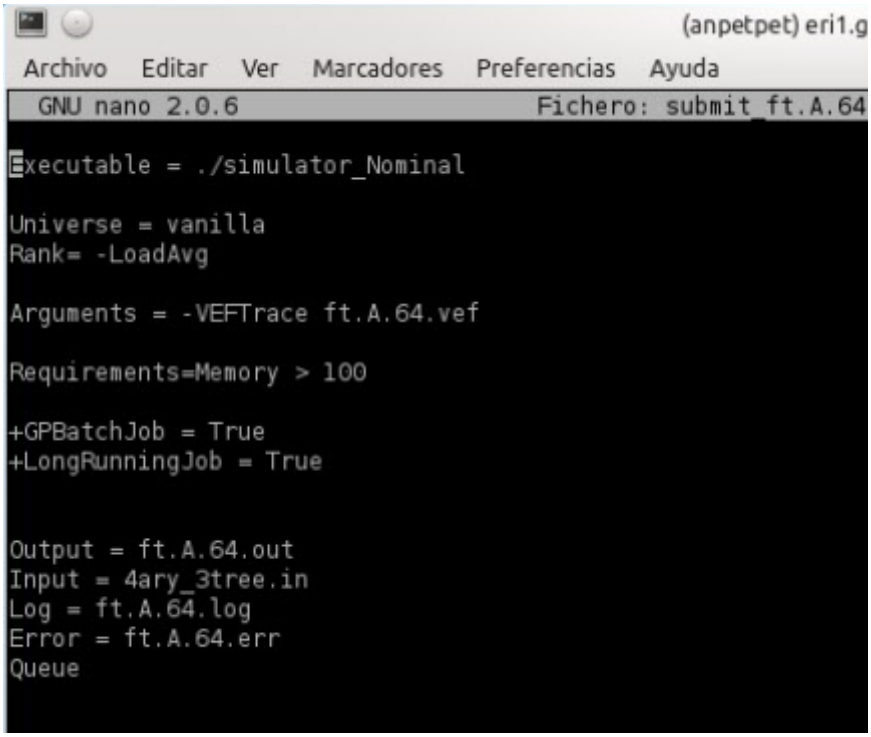


Figura 23: Archivo submit para Condor.

Las que nos interesan son *Executable* en la cual debemos poner el nombre del ejecutable del simulador. En *Arguments* se ponen la lista de argumentos que pondríamos en el comando, en este caso indicando que se usará un VEFTrace y la traza en concreto. En *Output* indicaríamos el nombre del fichero *.out,* en *Input* el fichero de entrada con la topología de nuestra red, y la única diferencia con el comando original es que necesitamos indicar también un fichero de *Log* (logs) y otro de *Error* (errores). Condor creará estos dos últimos iniciada la simulación, no es necesario crearlos previamente.

Por último, se añade la palabra *Queue* que define que el trabajo se debe poner en la cola. Cada queue es una ejecución distinta del ejecutable. Al final de esta, podríamos poner otro fragmento de Output, Input, Log, Error, Queue para realizar otra ejecución distinta con el mismo ejecutable, por ejemplo, si hiciéramos uso de varias topologías quedaría así:

```
1  Executable = ./simulator
2
3  Universe = vanilla
4  Rank= -LoadAvg
5
6  Arguments = -VEFTrace ft.A.64.vef
7
8  Requirements=Memory > 100
9
10 +GPBatchJob = True
11 +LongRunningJob = True
12
13 Output = FT_4-ary-3-mesh.out
14 Input = FT_4-ary-3-mesh.in
15 Log = FT_4-ary-3-mesh.log
16 Error = FT_4-ary-3-mesh.err
17 Queue
18
19 Output = FT_4-ary-3-cube.out
20 Input = FT_4-ary-3-cube.out
21 Log = FT_4-ary-3-cube.out
22 Error = FT_4-ary-3-cube.out
23 Queue
```

Figura 24: Archivo submit con varias ejecuciones.

Ahora, el próximo paso será ejecutar el programa en Condor. Para ello tenemos el comando:

> condor_submit *fichero_submitFT*

Otros comandos útiles que los que podamos hacer un seguimiento de los programas lanzados son los siguientes:

- > condor_q : Muestra los procesos en la cola de la máquina actual. Podemos ver la fecha y la hora en la que fue lanzado, el tiempo que lleva ejecutándose y su estado actual: R (Running) si se esta ejecutando o I (Iddle) si está en estado ocioso.

```
-- Submitter: eril.eri :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1400.4                   2/11 15:51   1+03:22:51 I  0   4394.5
2508.0                   4/28 10:47  11+03:44:50 R  0   1953.1
2508.1                   4/28 10:47  11+03:44:50 R  0   1220.7
2526.0                   4/28 17:15  10+21:16:13 R  0   1220.7
2613.0   anpetpet        5/8  15:49   0+23:17:39 R  0   976.6 simulator_Nominal
2616.1                   5/9  13:12   0+01:53:54 R  0   122.1
2616.8                   5/9  13:12   0+01:53:54 R  0   195.3
2617.1                   5/9  13:12   0+01:52:54 R  0   122.1
```

Figura 25: Resultado del comando condor_q.

- > condor_rm *id* : Borra el proceso, siempre que se haya realizado el submit desde la misma máquina.
- > condor_q -l : Nos muestra información detallada sobre los procesos encolados.
- > condor_q -run : Muestra los procesos que se están ejecutando y en qué maquinas.
- > condor_status : Muestra las máquinas que están disponibles y su estado.

8.2 Preparación del entorno

Como hemos visto en el capítulo anterior, para llevar a cabo una simulación necesitamos los siguientes elementos:

- Ejecutable del simulador con la configuración seleccionada, es decir, simulador nominal o Low Power, enlaces conectados o desconectados al principio, tipo de umbral estático o dinámico, y threshold seleccionado.
- El archivo *.vef* con la traza que deseemos hacer la simulación.
- El archivo *.in* que posee la topología de red elegida. En nuestro caso siempre será *4ary_3tree.in*.
- Además, si la simulación se ejecutará sobre Condor y no en local, necesitamos también el fichero *submit*.

Teniendo en cuenta además de que el archivo *OnOff_1.out* se genera con el mismo nombre tras cada simulación, para evitar la sobreescritura de este, se ha decidido hacer una organización por directorios con el esquema que se había mostrado en la Figura 18, y en cada uno de estos los elementos nombrados antes para realizar la simulación. Esto se puede ir haciendo de uno a uno, es decir, crear el ejecutable correspondiente y crear un directorio bajo un nombre para poder identificar la simulación, pero para agilizar el proceso, se han creado varios scripts.

En primer lugar necesitamos disponer de los ejecutables correspondientes a todas las simulaciones que vamos a realizar. Para ello se ha utilizado el siguiente script que crea cada uno de ellos con la configuración deseada.

Figura 26: Script para generar los ejecutables del simulador.

```
#!/bin/csh

# genera el ejecutable Nominal
touch simulator.c
make simulator LOW=FALSE RLO=FALSE
mv simulator simulator_Nominal

# genera el ejecutable con el mecanismo Low Power
# con thresholds Estáticos y Dinámicos
foreach sim ( Dynamic Static )
    if (${sim} == Dynamic) then
    set sim_tag = TRUE
    else
    set sim_tag = FALSE
    endif

    foreach thr ( 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 )
    touch simulator.c
    make simulator LOW=TRUE RLO=FALSE DYN=${sim_tag} THR=${thr}
    mv simulator simulator_${sim}_ThrSet${thr}
    end
end
```

Una vez hecho esto crearemos a mano, ya que es rápido, los directorios principales: LOW=T ST (mecanismo activado y con umbrales estáticos) y LOW=T DYN (mecanismo activado y con umbrales dinámicos), de los que colgará un directorio por cada traza VEF que vayamos a utilizar (en nuestro caso: FT, LU, MG, BT). Dentro de cada directorio de traza, ejecutamos un script como el que se muestra en la Figura 27.

```
#!/bin/csh
set TRAZAVEF="ft.A.64.vef"
set TRAZA="ft.A.64"
set EJECUTABLE="ft_DYN"
#Crea un directorio por cada threshold
foreach thr ( 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19)
mkdir THR${thr}
end
#En cada uno de ellos copia el ejecutable correspondiente
#el fichero 4ary_3tree.in y la traza VEF
foreach thr (  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19)
cp ~/sim-TFG/simulator_Dynamic_ThrSet${thr} ~/sim-TFG/LOW=T\ DYN/ft.A.64.vef/THR${thr}
mv ~/sim-TFG/LOW=T\ DYN/ft.A.64.vef/THR${thr}/simulator_Dynamic_ThrSet${thr} ~/sim-TFG/LOW=T\
cp ~/sim-TFG/4ary_3tree.in ~/sim-TFG/LOW=T\ DYN/ft.A.64.vef/THR${thr}
cp ~/sim-TFG/ft.A.64.vef ~/sim-TFG/LOW=T\ DYN/ft.A.64.vef/THR${thr}

# Genera el fichero submit para el scheduler Condor
printf "Executable = ./"$EJECUTABLE""${thr}"\n \
Universe = vanilla\n \
Rank= -LoadAvg\n \
Arguments = -VEFTrace "$TRAZAVEF"\n \
Requirements=Memory > 100\n \
+GPBatchJob = True\n \
+LongRunningJob = True\n \
Output = "$TRAZA".out\n \
Input = 4ary_3tree.in\n \
Log = "$TRAZA".log\n \
Error = "$TRAZA".err\n \
Queue">>~/sim-TFG/LOW=T\ DYN/ft.A.64.vef/THR${thr}/submit.txt \
end
```

Figura 27: Script con la distribución de ficheros.

Ahora, el último paso será subir el contenido de nuestro directorio a la máquina remota *eri1* por vía del comando *scp*. Para subir un directorio completo ejecutaríamos lo siguiente:

scp -P [puerto_remoto] -r [directorio_local]
[máquina_remota]:"[directorio_remoto]"

Con *-P* indicamos el puerto de la máquina en remoto, *-r* para indicar que lo que vamos a subir es el contenido de un directorio completo. Separado por un espacio, ponemos el nombre de la máquina en remoto y seguido por dos puntos, el nombre del directorio en remoto. En caso de que nos salte el siguiente mensaje: *scp: ambiguous target*, ponemos entre comillas el directorio remoto.

Cuando las simulaciones se hayan completado, para descargar el contenido de la máquina remota a nuestra máquina local, basta con alterar el orden del comando :

scp -P [puerto_remoto] -r [máquina_remota]:"[directorio_remoto]"

[directorio_local]

9. Resultados

En este capítulo se presentarán y evaluarán los resultados de las simulaciones realizadas, recordando que se ha utilizado la topología fat-tree formada por 64 nodos.

En un principio se contaba con la batería completa de los programas NAS, mostrados en la figura 8. Después de lanzar todas ellas con el simulador nominal, las trazas FT, BT, LU, MG y EP se habían completado, mientras que CG, IS y SP seguían en ejecución. Tras 17 días desde la puesta en marcha de la simulación, éstas tres últimas no habían terminado y su fichero de salida superaba los 2GB, por lo que se decidió abortar la ejecución y descartarlas de las pruebas. De las cinco restantes, por motivos desconocidos la traza EP terminaba su ejecución después del cuarto ciclo, por lo que también ha sido descartada.

Para las trazas FT, BT, LU y MG se ha completado un abanico de simulaciones acotado. Recordando todas las posibilidades de configuración del simulador, mostradas en la figura 18, se han realizado las pruebas con el mecanismo Low Power desactivado, como también las pruebas para las distintas configuraciones de la rama izquierda.

Así pues, se disponen de los resultados del mecanismo nominal, que serán el punto de partida para evaluar si el mecanismo consigue reducir el consumo de potencia. Para el caso de las pruebas con el mecanismo en marcha, se ha elegido una configuración inicial en la que los enlaces están activos (RLO=FALSE), puesto que si no lo están y hay un tráfico inicial muy elevado, la red se saturaría más rápido que si se dispone de todos ellos al principio. Con esto, para cada traza tenemos resultados utilizando umbrales estáticos y dinámicos, cada uno de ellos con las 19 parejas de valores para *Uoff* y *Uon*, haciendo un total de 156 simulaciones.

Para evaluar si el mecanismo es eficaz, nos vamos a fijar en dos valores que aparecen en el fichero de salida, el tiempo final y la potencia media consumida. Con estos podremos calcular la energía empleada:

$$E_{nominal} = 1 * t \qquad E_{LowPower} = P * t$$

Para el caso de la energía nominal, la potencia es 1 puesto que el mecanismo no está en marcha y la potencia no varía. Para cada traza y para todos los umbrales, se calculará la energía relativa, y este valor nos indicará si se consigue un ahorro o no.

$$E_{relativa} = \frac{E_{LowPower}}{E_{nominal}}$$

Si la energía relativa tiene un valor inferior a 1 el mecanismo es efectivo y se consigue reducir el consumo de potencia, en caso contrario, si es superior a 1 no se consigue.

9.1 Evaluación nominal

En primer lugar, se ha recogido del fichero de salida el tiempo que se ha tardado en completar la simulación. Este tiempo nos será necesario para calcular la energía nominal y comparar este valor con el de la energía obtenida con el mecanismo Low Power. En este caso, el valor de la energía nominal es el del tiempo, puesto que la potencia es 1.

En la figura 28 vemos una comparación entre los tiempo necesarios para completar la simulación de las cuatro aplicaciones. En la siguiente, también se observa la diferencia entre cada una respecto a la cantidad total de mensajes y megabytes enviados. Los mensajes enviados entre los nodos de LU, son en un orden de magnitud 100 veces superior al resto. Aun así, el valor de los megabytes transferidos entre los nodos de FT superan a los de LU.

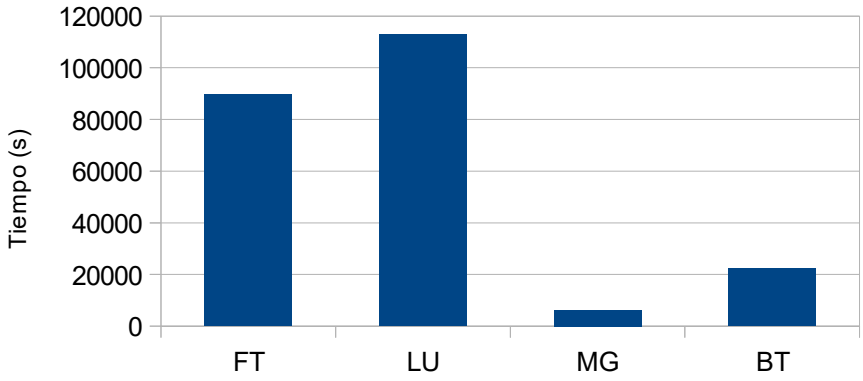


Figura 28: Tiempo nominal.

	Mensajes enviados	MB enviados
FT	32886	4032
LU	3545003	3253
MG	55581	301
BT	33075	1008

Figura 29: Cantidad total de mensajes y MB enviados.

9.2 Evaluación con mecanismo

Para poder llevar a cabo la evaluación y comparación con los resultados del simulador nominal, necesitamos recoger de cada uno de los ficheros de salida el valor de la potencia media consumida y el tiempo final. Con estos dos valores podremos calcular la energía Low Power y dividir ese valor con la energía o tiempo nominal.

Las siguientes gráficas a continuación, muestran la energía relativa obtenida para la configuración con umbrales estáticos y dinámicos ordenados por un valor de *Uon* ascendente. Recordamos para la configuración estática, los valores de la pareja *Uoff/Uon* son fijos, mientras que para la configuración dinámica se mantienen los mismos umbrales de conexión, y los de desconexión se calculan dinámicamente según el número de enlaces de salida activos.

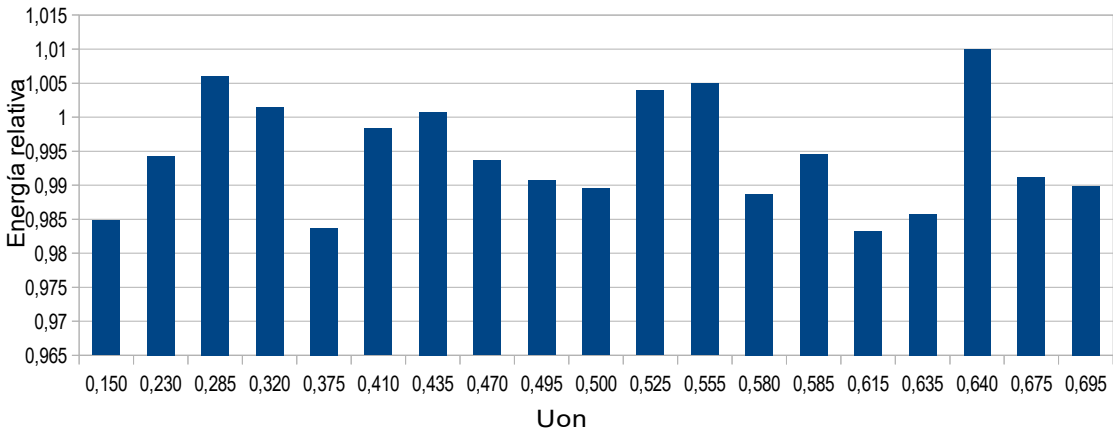


Figura 30: Energía relativa obtenida para FT utilizando umbrales estáticos.

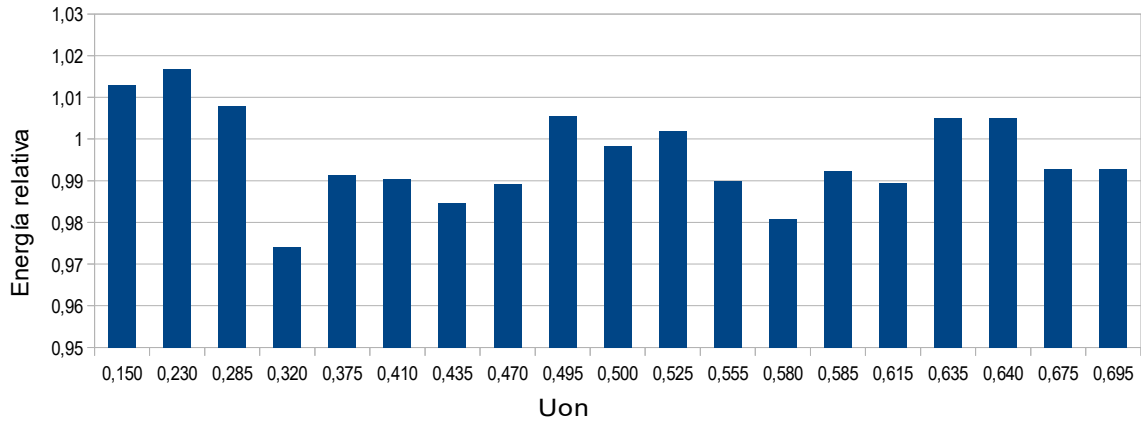


Figura 31: Energía relativa obtenida para FT utilizando umbrales dinámicos.

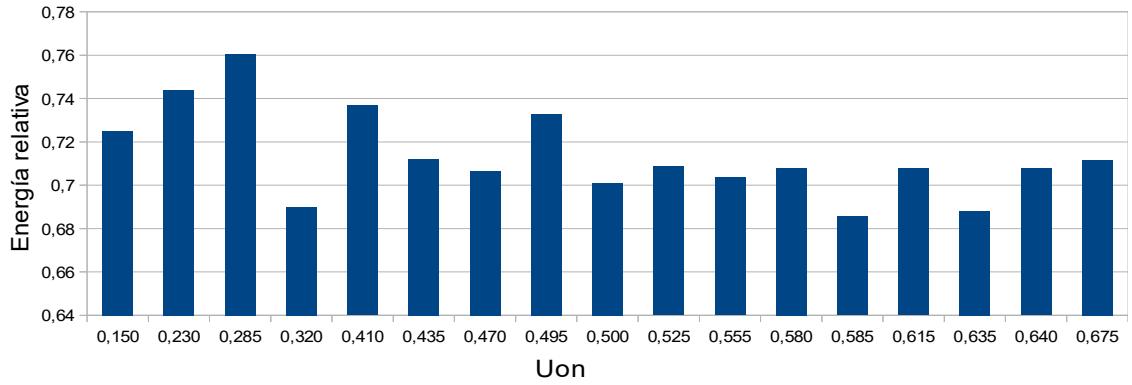


Figura 32: Energía relativa obtenida para LU utilizando umbrales estáticos.

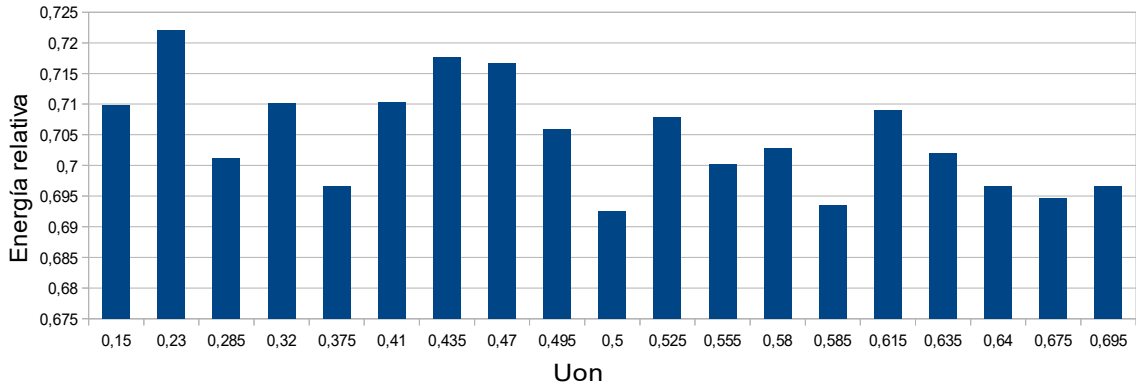


Figura 33: Energía relativa obtenida para LU utilizando umbrales dinámicos.

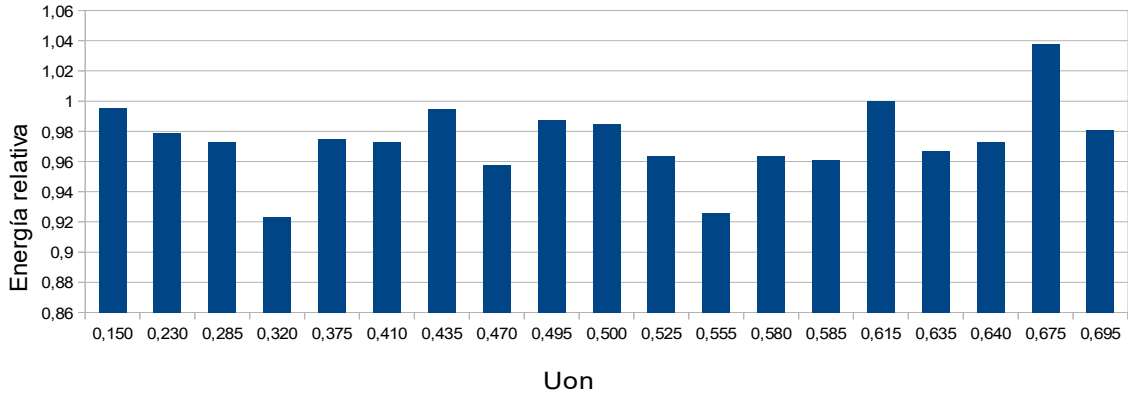


Figura 34: Energía relativa obtenida para MG utilizando umbrales estáticos.

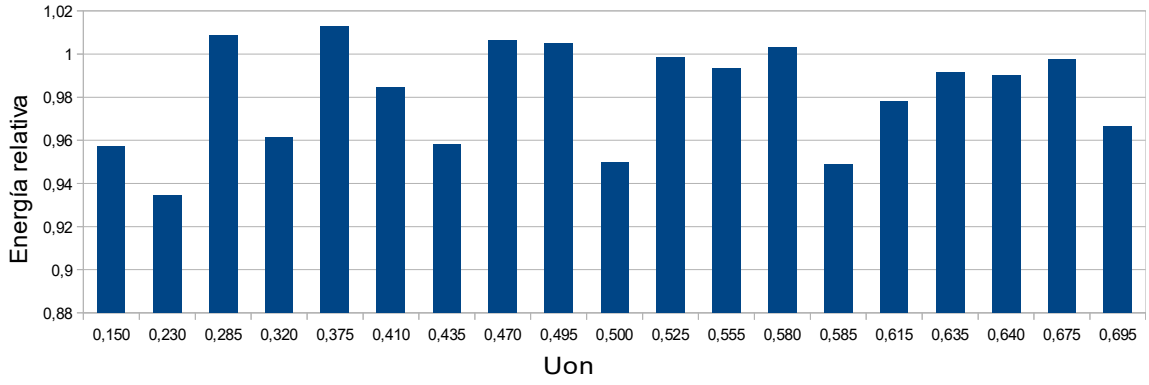


Figura 35: Energía relativa obtenida para MG utilizando umbrales dinámicos.

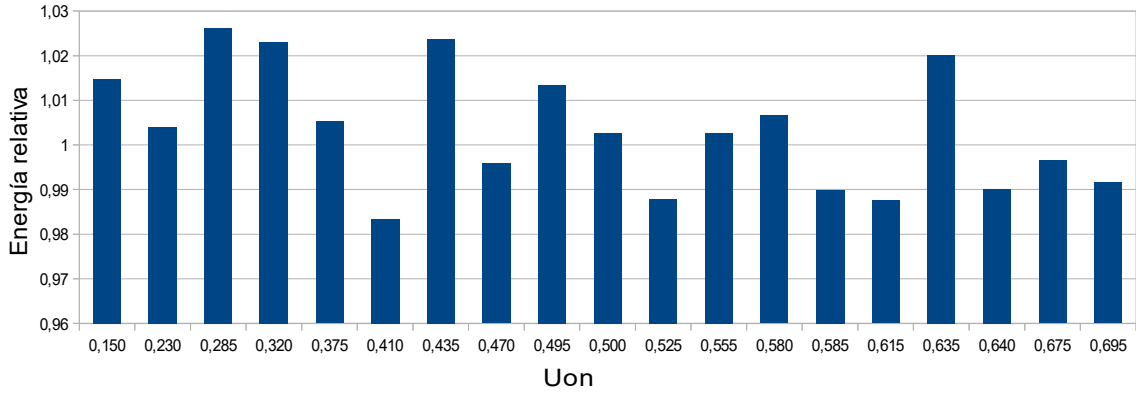


Figura 36: Energía relativa obtenida para BT utilizando umbrales estáticos.

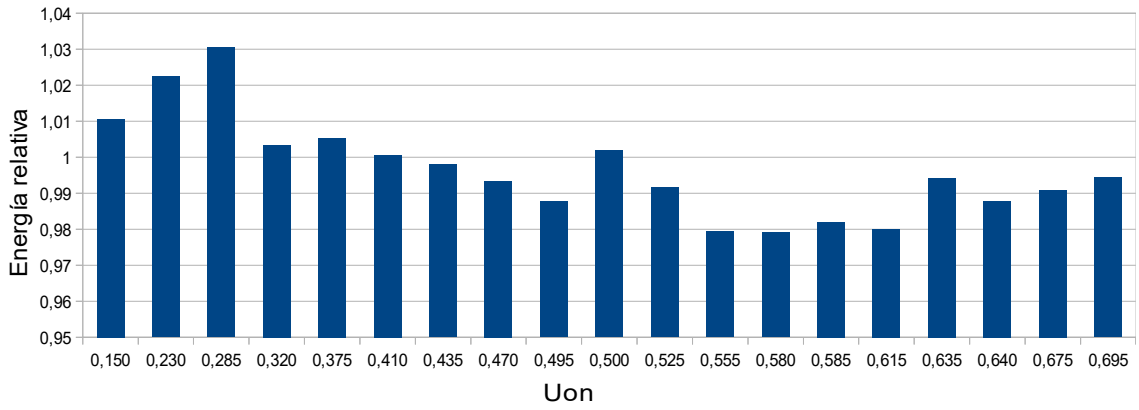


Figura 37: Energía relativa obtenida para BT utilizando umbrales dinámicos.

Para cada uno de los casos, se ha elegido el umbral con el que se consigue una menor energía relativa y por lo tanto con el que se produce un mayor ahorro de potencia. A partir del archivo *OnOff_1.out* se hará una representación gráfica del comportamiento que tiene el tráfico en cada ciclo, expresado como flits/ciclo/nodo, y como varía al respecto el consumo relativo de potencia. Puesto que existen una gran cantidad de ciclos, y el patrón del tráfico es muy irregular, se ha decidido hacer esa representación en un rango de ciclos acotado en el que se puede observar la reducción del consumo y además ver la fluctuación del tráfico.

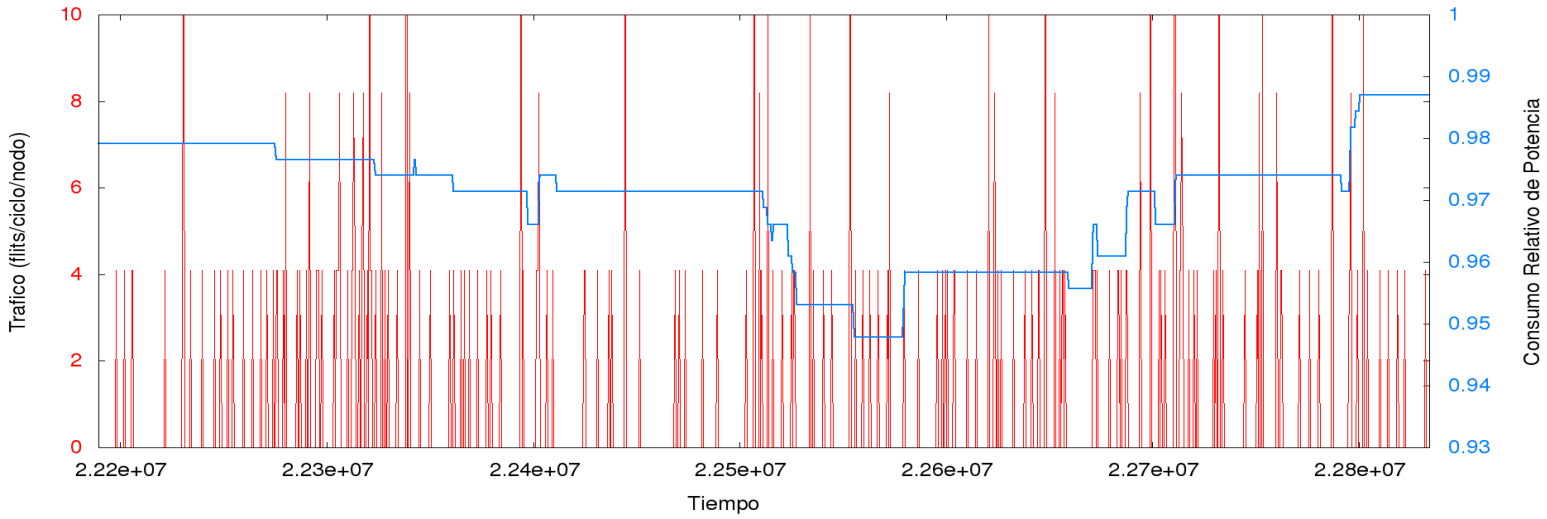


Figura 38: Consumo de potencia para FT usando configuración estática con Uon=0,615-Uoff=0,265.

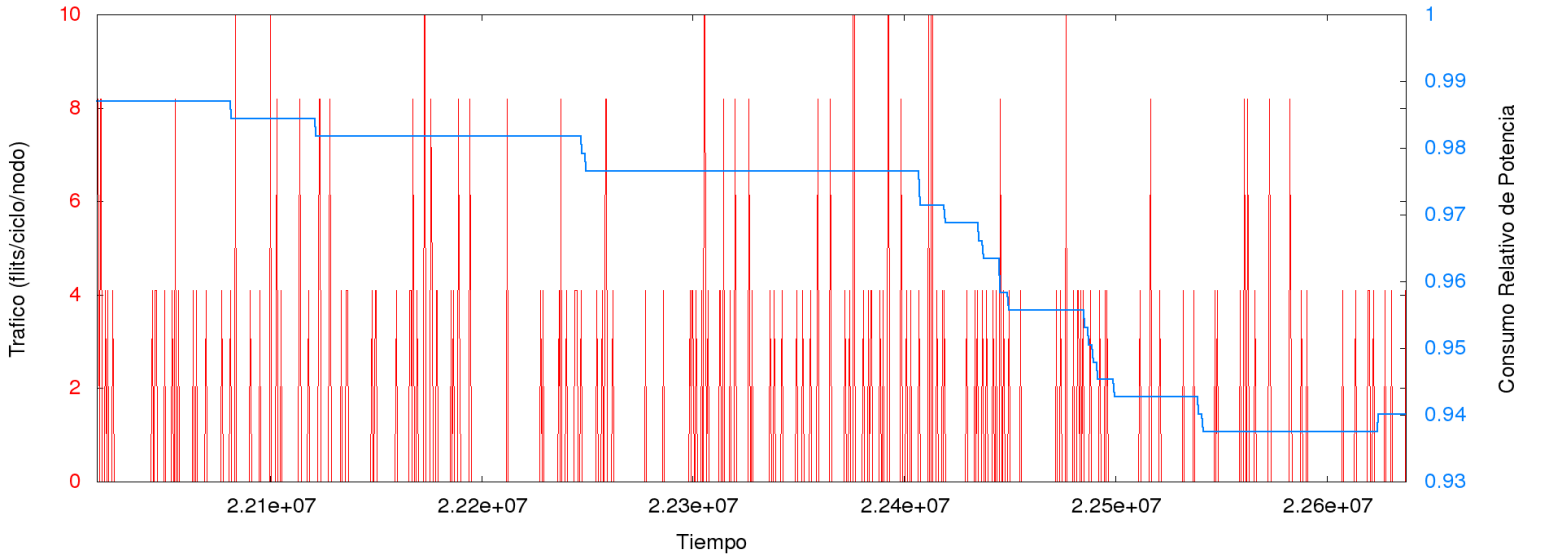


Figura 39: Consumo de potencia para FT usando configuración dinámica con Uon=0,32.

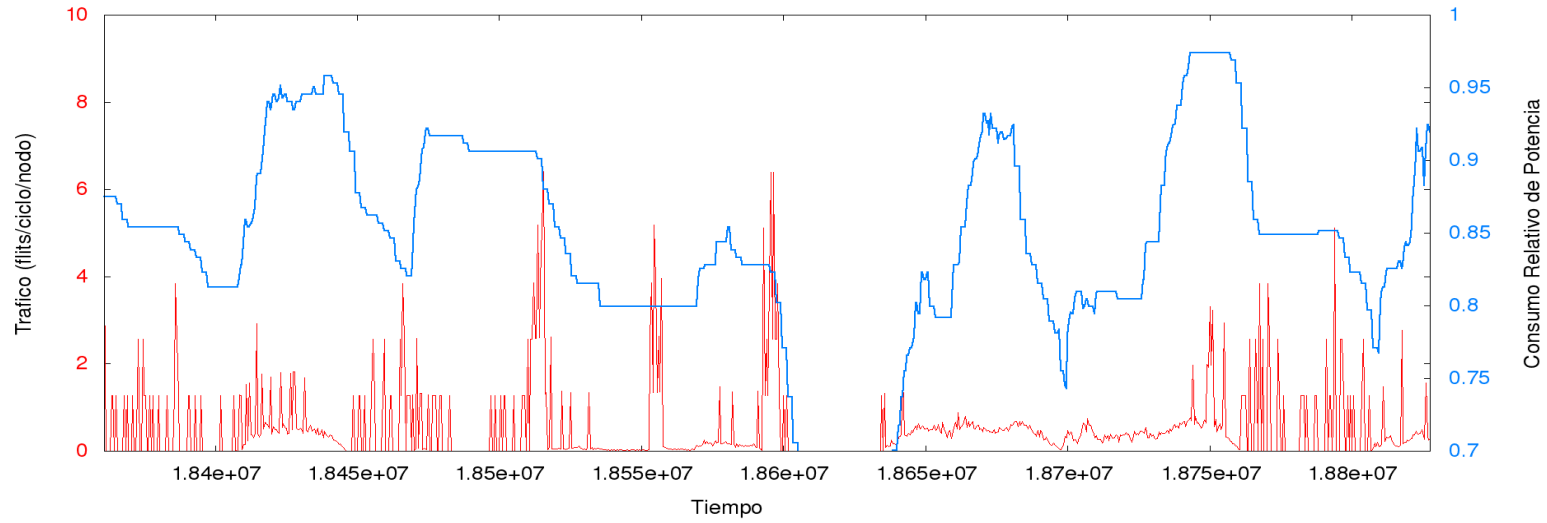


Figura 40: Consumo de potencia para LU usando configuración estática con Uon=0,585-Uoff=0,115.

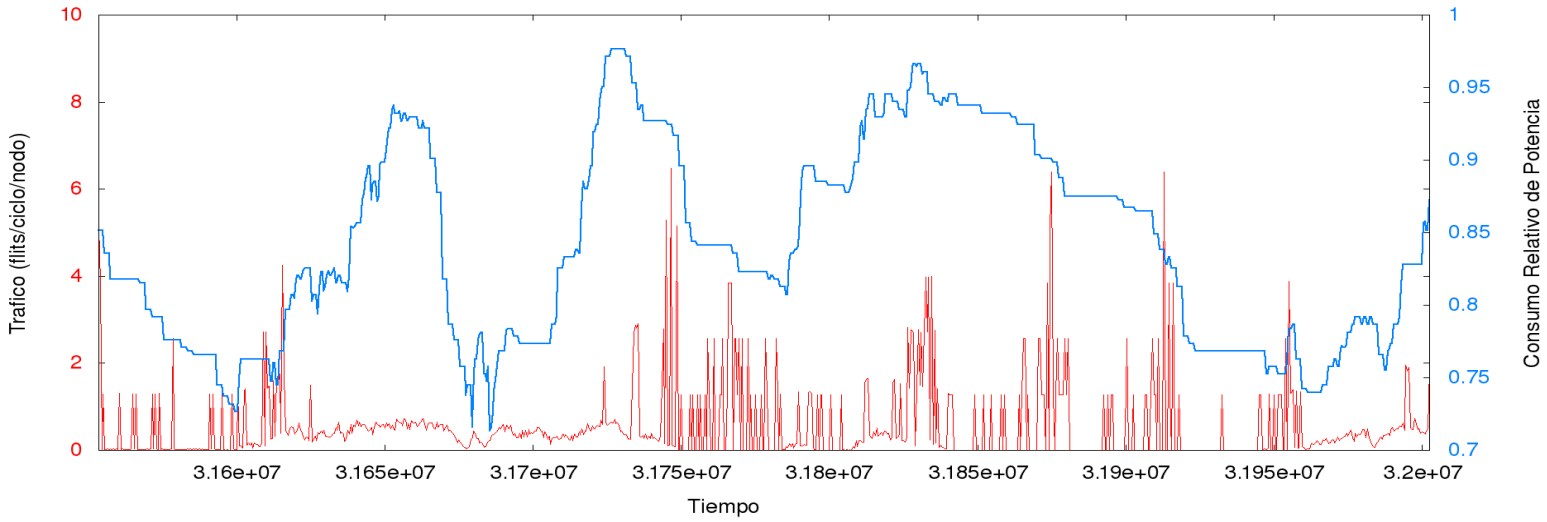


Figura 41: Consumo de potencia para LU usando configuración dinámica con Uon=0,50.

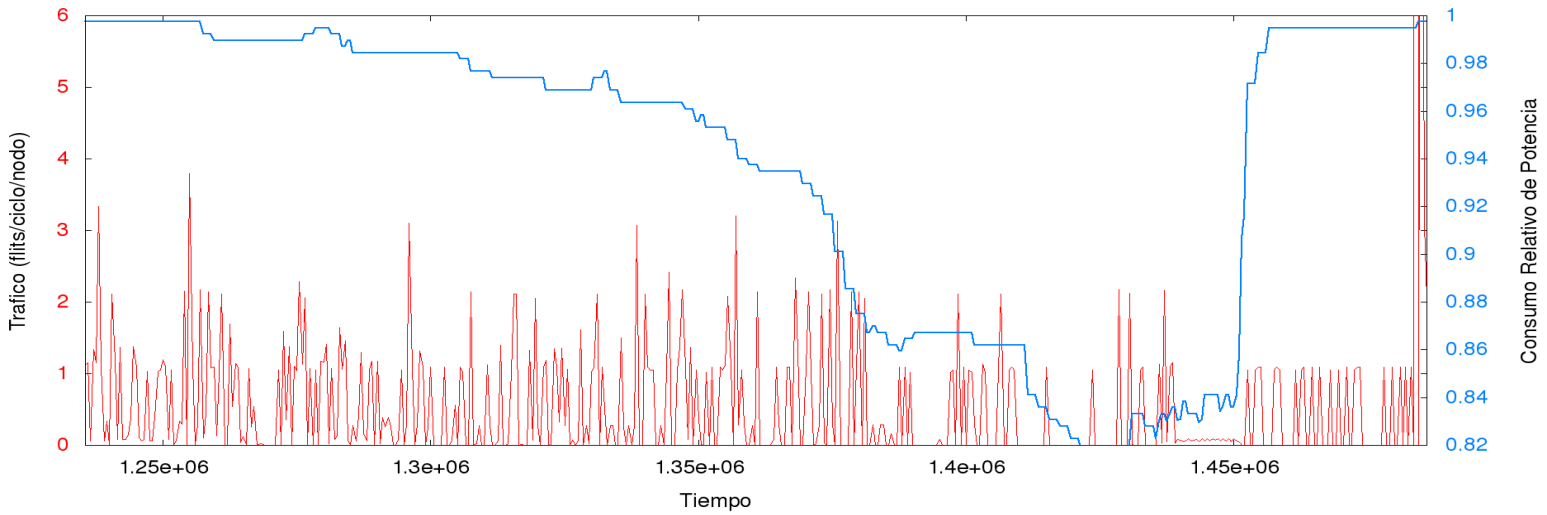


Figura 42: Consumo de potencia para MG usando configuración estática con Uon=0,32-Uoff=0,20.

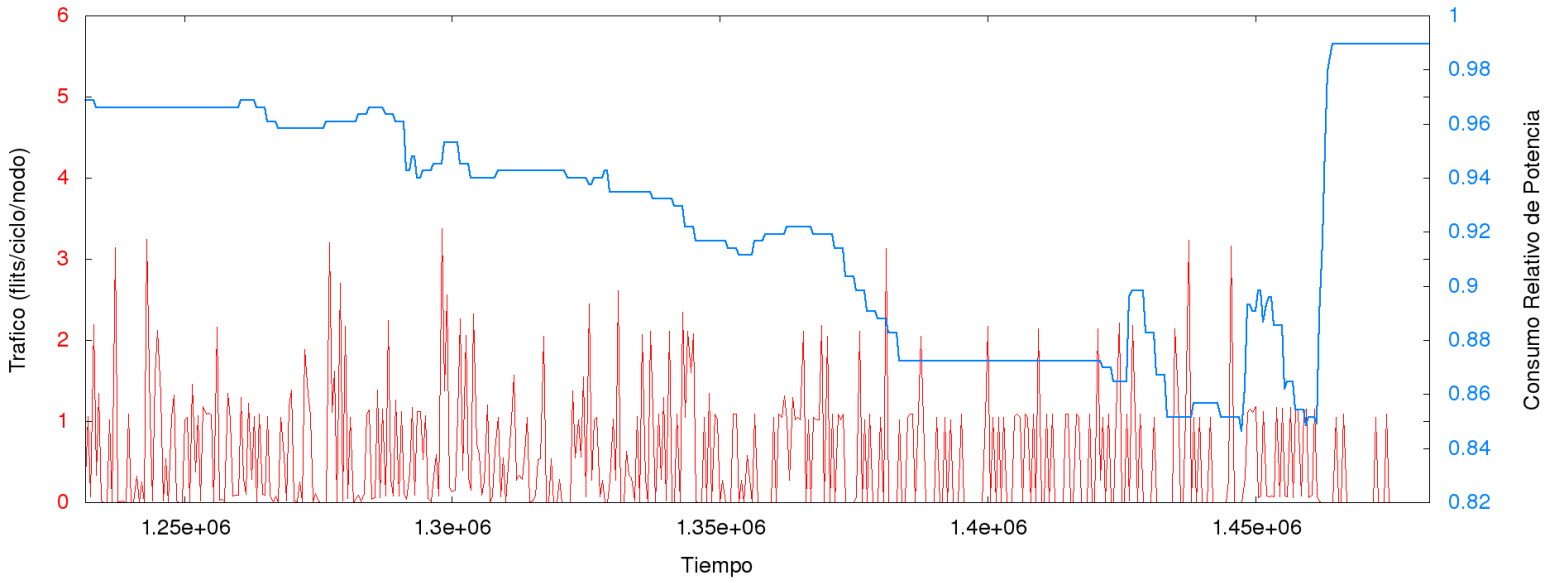


Figura 43: Consumo de potencia para MG usando configuración dinámica con Uon=0,23.

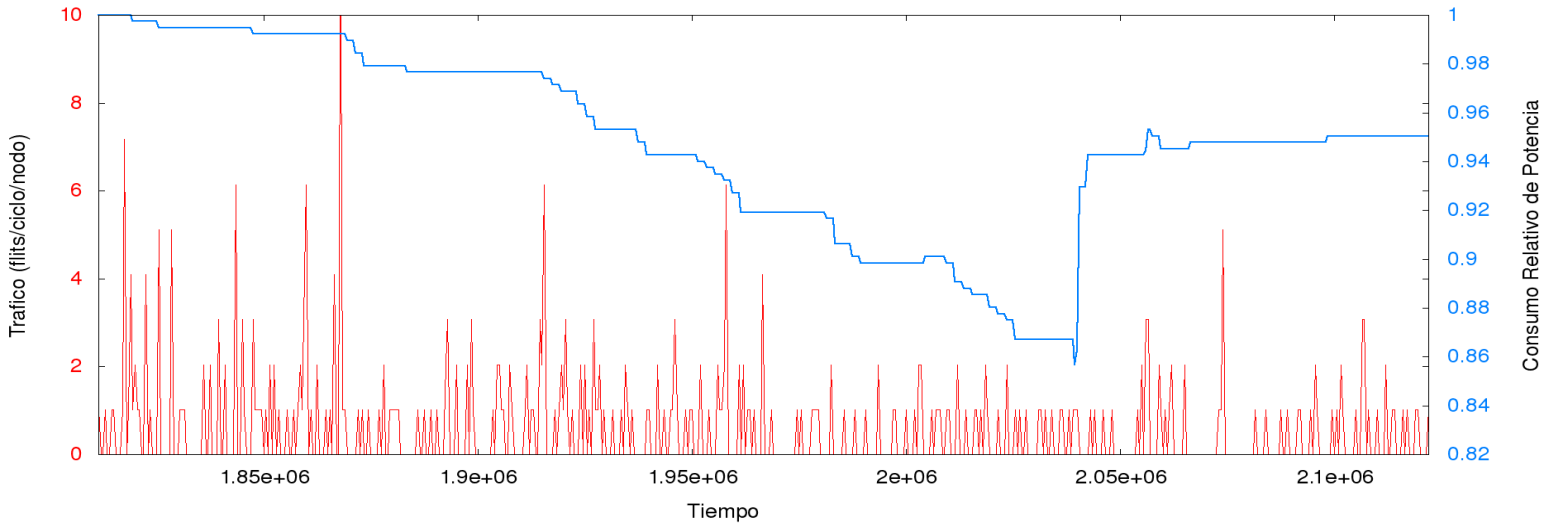


Figura 44: Consumo de potencia para BT usando configuración estática con Uon=0,41-Uoff=0,29.

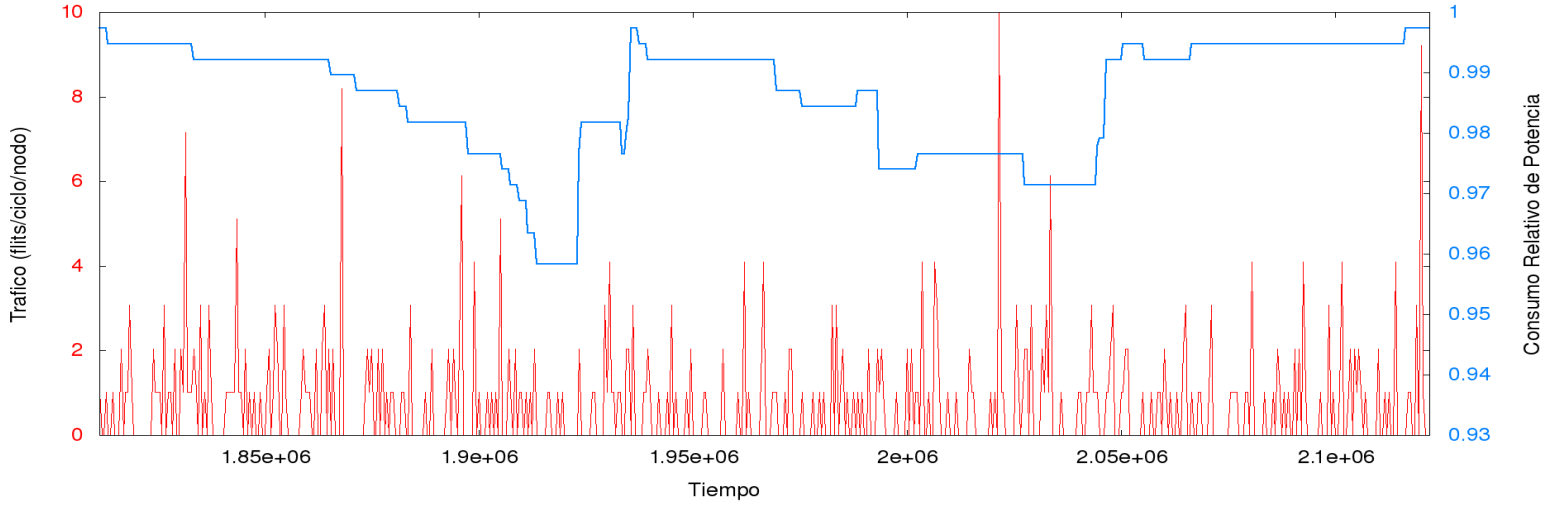


Figura 45: Consumo de potencia para BT usando configuración dinámica con Uon=0,58.

9.3 Observaciones

Comenzaremos comentando las figuras que muestran la energía relativa obtenida para cada configuración. Los umbrales que nos interesan, son aquellos para los que se ha obtenido un valor de energía por debajo de 1, lo cual significa que se consigue un ahorro. El caso más llamativo es para la traza de LU. A diferencia de las otras en las que los valores de la energía relativa oscilan entre los 0,92-1,01 aquí tenemos unos valores por debajo de 0,73. Este valor indica que el consumo de energía empleando el mecanismo de ahorro de potencia se reduce en un 17%.

Para observar mejor el comportamiento del tráfico y la potencia nos fijamos en las siguiente tipo de gráficas. En el caso de FT se puede observar si trazamos dos líneas verticales, en que hay una banda con un tráfico de 2 flits/ciclos/nodo, otra de 4 y varios casos en los que supera el valor de 8 y de 10 o para MG por ejemplo, se suele mantener en en 1 flit/ciclo/nodo. Se observa también, la cantidad de ciclos en los que no se envía ningún mensaje. Para el caso de LU, lo que más llama la atención es la oscilación del consumo de potencia y los valores en los que se mantiene. Mientras en FT se puede apreciar como entre los ciclos 2,23e+07 hasta 2,24e+07 se mantiene estable y no varía, en LU a simple vista es más irregular que los otros.

10. Conclusiones

En primer lugar, el valor observado más favorable después de recopilar los datos para las simulaciones es el tiempo. A nivel teórico, lo que se espera de unos resultados con el mecanismo activado, es que el tiempo nominal sea inferior que el tiempo Low Power puesto que la estrategia consiste en ir apagando los enlaces a medida que el tráfico disminuya, consiguiendo así un consumo de potencia menor, a cambio de una penalización en la latencia y por lo tanto un mayor tiempo en procesar todo el tráfico. Sin embargo, en algunos casos ocurre todo lo contrario, el tiempo conseguido es incluso menor que el tiempo nominal.

Es de esperar también, que a menor energía consumida se tarde un mayor tiempo puesto que hemos ido apagando enlaces y viceversa, si consumimos una mayor energía es porque tenemos más enlaces conectados, y tardaríamos menos tiempo enviar todos los mensajes. La siguiente gráfica muestra el efecto contrario conseguido, en los casos que consumimos menos energía, obtenemos también un tiempo relativo menor al nominal. Esto es un efecto positivo, puesto que conseguimos un ahorro considerable de energía además de un tiempo de simulación menor.

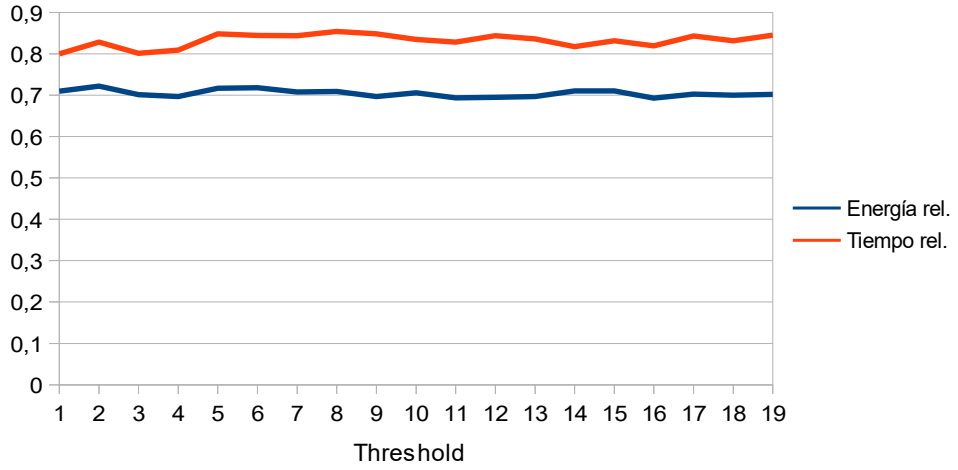


Figura 46: Relación entre la energía relativa y el tiempo relativo para LU.

Por otro lado, a pesar de que en las gráficas del capítulo anterior se ha elegido un rango en el que se ve cómo la potencia puede llegar a bajar hasta 0,9, es decir, un 10% de ahorro, o en nuestro caso especial (LU), un 25%, esto no ocurre así durante toda la ejecución puesto que hay ciclos en los que el simulador emplea más potencia. Para MG, conseguimos en los mejores umbrales un 5%, mientras que para LU de 13 a 18% de ahorro.

Centrándonos en el caso para el que se obtienen mejores prestaciones, la traza LU con umbrales estáticos y *Uoff*=0,115 y *Uon*=0,585, veremos algunos detalles curiosos que nos proporciona el archivo de salida de la simulación.

El tiempo de simulación para LU con el simulador nominal es de 112.902 segundos, mientras que con la mejor pareja de umbrales estáticos de 89.086 segundos. Esto es un tiempo relativo de 0,789, es decir, casi un 22% menos de tiempo. La latencia media nominal es de 8348,92 ciclos mientras que con el mecanismo activado es de 9100.35. Es algo inesperado ya con una latencia media mayor, es lógico que el tiempo final sea superior también, aunque no ocurre así. Para razonar esto, los datos que nos proporciona el fichero es la productividad (palabras por ciclo) y la utilización de los enlaces. Para el caso nominal la productividad es de 30,19 palabras por ciclo mientras que para el mecanismo de 38,28, haciendo posible que la penalización de la latencia no resulte significativa puesto que la productividad aumenta. La utilización media de los enlaces también aumenta, de un 9% para la simulación nominal a un 14%.

El estudio llevado a cabo demuestra que la estrategia de reducción de consumo de potencia implementada en el simulador de redes, sigue siendo efectiva para cargas de trabajo reales. Haber utilizado este tipo de trazas, nos ayudará a comprender y a predecir mejor el comportamiento que tendrá el mecanismo a la hora de implementarlo sobre una red de comunicación real. El hecho de saber que se puede conseguir un ahorro de potencia considerable en uno de los elementos en el que menos esfuerzo se ha realizado, hace que siga siendo un campo importante en el que hay que seguir investigando.

10.1 Trabajo futuro

Habiendo visto los resultados anteriores, no cabe duda de que queda un trabajo por delante en el que se ha de resolver y averiguar el por qué de los valores obtenidos con el mecanismo Low Power. Así, podríamos considerar lo siguiente para ampliar el proyecto:

- Descubrir por qué se obtienen mejores tiempos frente al simulador nominal, y además de eso un ligero ahorro. Es un efecto positivo el que se produzca un aumento en la productividad de la red y la utilización de los enlaces con el mecanismo activado, pero se desconoce el motivo.
- Utilizar herramientas como Automake y Autoconf para una configuración e instalación automática del simulador.
- Probar el comportamiento del mecanismo en redes directas, como también implementar el mismo sobre la topología jerárquica DragonFly.

11. Bibliografía

[1] Data center: https://en.wikipedia.org/wiki/Data_center (23 August 2016)

[2] Report to Congress on Server and Data Center Energy Efficiency Public Law 109-431 U.S. Environmental Protection Agency ENERGY STAR Program August 2, 2007: https://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf

[3] An Introduction to the InfiniBand Architecture by Odysseas Pentakalos: http://archive.oreilly.com/pub/a/network/2002/02/04/windows.html

[4] Power Saving Features in Mellanox Products, Link-Level Power Saving: http://www.mellanox.com/pdf/whitepapers/WP_ECONET.pdf

[5] Multiprocessor Interconnection Networks : http://15418.courses.cs.cmu.edu/spring2013/article/30

[6] Análisis y evaluación de técnicas de reconfiguración dinámica de la red de interconexión en sistemas masivamente paralelos, José Luis Sánchez García, Enero de 1998: https://books.google.es/books?id=uT3tHvJTB40C&pg=PA19&lpg=PA19&dq=redes+indirectas&source=bl&ots=ZnH_cUFQz1&sig=bpwXYKPmkl5ndlv8n1vsBtTTI74&hl=es&sa=X&ved=0ahUKEwj__-uekrXOAhULWxQKHbxdAJYQ6AEIHDAA#v=onepage&q=redes%20indirectas&f=false

[7] A Scalable Commodity Data Center Network Architecture, Mohammad Al-Fares, Alexander Loukissas, Amin Vahdat: http://www.sigcomm.org/sites/default/files/ccr/papers/2008/October/1402946-1402967.pdf

[8] Una estrategia para la reducción del consumo de potencia en redes de interconexión, Marina Alonso Díaz (Junio 2012) : https://riunet.upv.es/bitstream/handle/10251/16186/tesisUPV3835.pdf?sequence=1

[9] NAS Parallel Benchmarks : http://www.nas.nasa.gov/publications/npb.html

[10] NAS Parallel Benchmark (Version 1.0) Results 11-96 Subhash Saini1 and David H. Bailey 2 Report NAS-96-18, November 1996: http://www.nas.nasa.gov/assets/pdf/techreports/1996/nas-96-018.pdf

[11] VEF Traces: A Framework for Modelling MPI Traffic in Interconnection Network Simulators, Francisco J. Andújar, Juan A. Villar, Jose L. Sánchez, Francisco J. Alfaro and Jesús Escudero-Sahuquillo (Sep 2015): http://www.i3a.info/VEFtraces/

Anexo 1

Las dos clases en código Java, que extraen información del fichero de trazas VEF para que sea procesada posteriormente por Gnuplot.

```
import java.io.File;
import java.io.FileWriter;
import java.util.Scanner;
import java.util.StringTokenizer;

public class DespliegaOperaciones {
    public static void main(String[] args) {
        File fichero = new File("K:\\workspace\\TFG\\NAS-64n-bt-A.vef");
        Scanner s = null;
        FileWriter ficheroAux = null;
        String idAComparar="";
        try {
            s = new Scanner(fichero);
            ficheroAux = new FileWriter("K:\\workspace\\TFG\\aux.txt");
            while (s.hasNextLine()) {
            String linea = s.nextLine();
            int origen=0;
            int destino=0;
            int bytesEnviados=0;
            String idColectiva;
            int tipoColectiva;
            String [] tokens = linea.split("\\s+");

                if (!tokens[0].substring(0,1).equals("G")){//La operación es Punto a Punto
                        origen=Integer.parseInt(tokens[1]);
                        destino=Integer.parseInt(tokens[2]);
                        bytesEnviados=Integer.parseInt(tokens[3]);
                        if(origen!=destino){
                            ficheroAux.write(""+origen+" "+destino+" "+bytesEnviados+"\n");
                        }
                }
                //La operación es Colectiva
                else {
                idColectiva=tokens[0];
                tipoColectiva=Integer.parseInt(tokens[2]);
                origen=Integer.parseInt(tokens[3]);
                bytesEnviados=Integer.parseInt(tokens[4]);
                if(bytesEnviados!=0){//Sólo nos interesa detectar el envío de datos
                    if(tipoColectiva==0){   // Es un Broadcast
                        for(int i=1; i<64;i++){
                            if(origen!=i){
                                ficheroAux.write(""+origen+" "+i+" "+bytesEnviados+"\n");
                            }
                        }
                    }
                    if(tipoColectiva==1){   // Es un Reduce
                      ficheroAux.write(""+origen+" "+0+" "+bytesEnviados+"\n");
                    }
                    if(tipoColectiva==2){  // Es un AllReduce
                        if(!idColectiva.equals(idAComparar)){
                    //Aquí se despliega la operación colectiva, por lo que las siguientes líneas
                    //en las que se muestra la misma operación (mismo ID) no han de ser desplegadas.
                    //Se hace la misma comparación para AlltoAll
                            for(int j=1; j<64;j++){ //Es lo mismo que un Reduce
                                ficheroAux.write(""+j+" "+0+" "+bytesEnviados+"\n");
                            }
                            for(int i=1; i<64;i++){//seguido de un Broadcast
                                ficheroAux.write(""+0+" "+i+" "+bytesEnviados+"\n");
                        }
                        }
                        idAComparar=idColectiva;
```

```
                }
                if(tipoColectiva==7){// Es un AlltoAll
                    if(!idColectiva.equals(idAComparar)){
                        for(int i=0; i<64;i++){
                            for(int j=0; j<64;j++){
                                if(i!=j){
                                    ficheroAux.write(""+i+" "+j+" "+bytesEnviados+"\n");
                                }
                        }
                        }
                        }
                }
                idAComparar=idColectiva;
            }
            }
        }
        ficheroAux.close();

public class sumaBytes {
    public static void main(String[] args) {
        File fichero = new File("K:\\workspace\\TFG\\aux.txt");
        Scanner s = null;
        FileWriter fichero2 = null;
        int[][]  comunicacionNodos = new int[4033][3];
        //Utilizamos un array para almacenar la comunicación entre los nodos
        // Origen Destino BytesEnviados
        for(int i=0; i<4033;i++){
            for(int j=0; j<3;j++){
                if (j==2){
                    comunicacionNodos[i][j]=0;
                    }
                else comunicacionNodos[i][j]=-1;
            }
            }
        try {
            s = new Scanner(fichero);
            fichero2 = new FileWriter("K:\\workspace\\TFG\\NAS-64n-bt-A.RESULTADO.txt");
            int origen=0;
            int destino=0;
            int bytesEnviados=0;
            while (s.hasNextLine()) {
            String linea = s.nextLine();
            boolean parEncontrado=false;
            String [] tokens = linea.split("\\s+");
            origen=Integer.parseInt(tokens[0]);
            destino=Integer.parseInt(tokens[1]);
            bytesEnviados=Integer.parseInt(tokens[2]);
                //La pareja de nodos todavía no ha sido insertada en el array
                for(int i=0; i<4033 && !parEncontrado;i++){
                    if(comunicacionNodos[i][0]==-1){
                    comunicacionNodos[i][0]=origen;
                    comunicacionNodos[i][1]=destino;
                    comunicacionNodos[i][2]=bytesEnviados;
                    parEncontrado=true;
                            }
                //La pareja de nodos ya había sido insertada. Acumular los bytes enviados
                    else if(comunicacionNodos[i][0]==origen){
                            if(comunicacionNodos[i][1]==destino){
                                comunicacionNodos[i][2]+=bytesEnviados;
                                parEncontrado=true;
                                }
                            }
            }
    }
    //Escribimos en el fichero de resultado el contenido del array.
    for(int i=0; i<4033;i++){
        if(comunicacionNodos[i][0]!=-1){
         fichero2.write(""+comunicacionNodos[i][0]+" "+comunicacionNodos[i][1]+" "+comunicacionNodos[i][2]+"\n");
        }
        }
    fichero2.close();
```

Anexo 2

#Comandos Gnuplot para las gráficas del patrón de comunicación para cada traza

```
set xlabel 'Nodo origen'
set ylabel 'Nodo destino'
set cblabel 'Bytes transferidos'
set view map
set palette rgbformulae 10,13,33
set xrange [-1:64]
set yrange [-1:64]
set term png size 1024,1024
set output 'LU_2_2.png'
splot 'lu.A.64RESULTADO.txt' using 1:2:3 notitle with points palette ps 1 pt 5
```

#Comandos Gnuplot para representar el tráfico y la potencia frente al tiempo

```
set terminal png size 1920,720 enhanced font "Helvetica,20"
set output 'prueba.png'
unset key
set xlabel 'Tiempo'
set xrange [1811500:1486000]
set ylabel 'Trafico (flits/ciclo/nodo)'
set yrange [0:10]
set ytics textcolor lt 1
set y2label 'Consumo Relativo de Potencia'
set y2range[0.82:1]
set y2tics textcolor lt 3;
plot "OnOff_1.out" using 1:4 linetype 1 with lines,"OnOff_1.out" using 1:2 axes x1y2
linetype 3 linewidth 2 with lines
```