University of Castilla-La Mancha

A publication of the **Department of Computer Science**

Implementación y Evaluación de un Modelo de Gestión de la Tabla de Arbitraje de InfiniBand

by

J. Laborda, J.A. Villar, F.J. Alfaro, J.L. Sánchez

Technical Report **#DIAB-04-12-1** December 2004

DEPARTAMENTO DE INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

Índice general

Implementación y Evaluación de un Modelo de Gestión de la Tabla de Arbitraje de InfiniBand [1]

Joaquín Laborda, Juan A. Villar
Instituto de Investigación en Informática
Campus Universitario
02071 - Albacete, España
jlabordac@terra.es, juanan@info-ab.uclm.es

Francisco J. Alfaro, José L. Sánchez
Departamento de Informática
Escuela Politécnica Superior
Universidad de Castilla-La Mancha
02071- Albacete, España
{falfaro, jsanchez}@info-ab.uclm.es

[1]This work was partly supported by the Spanish CICYT under Grant TIC2004-1151-C07 and Junta de Comunidades de Castilla-La Mancha under Grant PBC-02-008

Resumen

InfiniBand es una nueva tecnología de interconexión entre sistemas de procesamiento y dispositivos de E/S que permite formar una red de área de sistema. InfiniBand es muy probable que sea el estándar de facto en los próximos años. InfiniBand está siendo desarrollado por la InfiniBand ^{SM} Trade Association para proporcionar funcionalidad, fiabilidad, disponibilidad, rendimiento, escalabilidad y calidad de servicio necesarios para los sistemas servidor del presente y el futuro.

Para proporcionar calidad de servicio a las aplicaciones, InfiniBand proporciona varios mecanismos tanto para servicios orientados a conexión como no. En los primeros se puede intentar garantizar calidad de servicio, sin embargo, en los servicios sin conexión no es posible dar garantías. Los mecanismos que incorpora InfiniBand para conseguir proporcionar calidad de servicio son básicamente: niveles de servicio, canales virtuales, tabla de arbitraje en dichos canales virtuales y particiones.

El trabajo que aquí se presenta ha consistido, básicamente, en la implementación de unos algoritmos para el manejo de la tabla de arbitraje, y su evaluación de prestaciones. Esos algoritmos reflejan el comportamiento de un modelo de gestión de la tabla de arbitraje que había sido previamente propuesto de una manera formal.

1. Introducción

Los sistemas basados en bus utilizados en los subsistemas de entrada/salida (E/S) se han convertido en un cuello de botella, especialmente en los servidores de alto rendimiento. Mientras que los buses tienen su principal ventaja en la simplicidad y han servido satisfactoriamente hasta este momento, no aprovechan adecuadamente la tecnología eléctrica subyacente actual, por lo que no pueden proporcionar las tasas de transferencia requeridas actualmente en los sistemas.

A pesar de las actualizaciones existentes, el sistema bus de E/S más popular (el bus PCI) ofrece ancho de banda, concurrencia y fiabilidad limitados, y estas limitaciones son inaceptables para la mayoría de las aplicaciones actuales y sistemas en general. Un simple fallo de un dispositivo puede afectar al funcionamiento de los restantes dispositivos conectados al bus.

Ésta fue la principal razón por la que un grupo de las compañías más importantes se unieron para desarrollar un estándar tanto para la comunicación entre nodos de procesamiento y dispositivos de E/S, como las comunicaciones internas del procesador, conocido como InfiniBand [7]. La InfiniBand Trade Association (IBTA) es un grupo de más de 200 compañías fundado en Agosto de 1999 para desarrollar InfiniBand. Este esfuerzo también está abierto a universidad y laboratorios de investigación. El IBTA está liderado por un comité cuyos miembros pertencen a distintas compañías como Dell, Compaq, HP, IBM, Intel, Microsoft, Sun, IBM e Intel. Otros miembros son 3Com, Cisco Systems, Fujitsu-Siemens, Hitachi, Adaptec, Lucent Technologies, NEC, y Nortel Networks.

Las primeras especificaciones de la arquitectura de InfiniBand (release 1.0) fueron publicadas en octubre del 2000 [6], y actualmente más de 200 compañías soportan esta iniciativa. La revisión más reciente (release 1.2) de la arquitectura data de septiembre de 2004.

Por otra parte, la mayoría de los productos de red comerciales han intentado alcanzar el máximo ancho de banda con una latencia mínima, dejando otros aspectos sin cubrir como pueden ser la garantía de ancho de banda, latencia máxima, retrasos [10]. Muchas aplicaciones actuales necesitan cubrir estos aspectos. Una red que actualmente cubre los aspectos de calidad de servicio es probablemente Asynchronous Transfer Mode (ATM) ([4, 5]. Sin embargo, ATM se centra más en las redes Wide Area Network (WAN), y sólo puede ser utilizada en entornos Local Area Network (LAN) con grandes dificultades. No es adecuada para proveer la comunicación entre los procesadores y los dispositivos.

La Internet Engineering Task Force (IETF) está desarrollando una arquitectura general para proveer calidad de servicio en Internet. Este esfuerzo se puede encontrar referenciado como Servicios Diferenciados [2].

Por tanto, sería importante para el éxito de InfiniBand poder satisfacer requisitos particulares de algunas latencias y a veces hacer lo mismo con características generales de calidad de servicio que demandan otro tipo de aplicaciones. InfiniBand provee una serie de mecanismos que, cuando se usan adecuadamente, son capaces de proveer ca-

lidad de servicio. Estos mecanismos son principalmente la segmentación del tráfico en base a categorías y el arbitraje de los puertos de salida según una tabla de arbitraje, la cual puede ser configurada para determinar la prioridad de los paquetes que necesitan dicha calidad de servicio.

InfiniBand distingue hasta un máximo de 16 niveles de servicio, pero no especifica las características que deben poseer el tráfico en cada uno de los niveles. Tampoco define como se asigna el contenido de las entradas en la tabla de arbitraje. Lo único que establece es la estructura que dicha tabla debe tener, dejando la implementación de la tabla como una responsabilidad de los vendedores o los usuarios finales, en base a los objetivos que éstos deseen lograr.

Se puede conseguir una implementación de los servicios diferenciados sobre InfiniBand en [8]. En este estudio el tráfico se clasifica en varias categorías y el autor propone que la tabla de arbitraje de InfiniBand debe repartir cada categoría de una forma distinta, pero no indica como rellenar estas tablas.

El trabajo que aquí se presenta ha consistido fundamentalmente en la realización y evaluación de un conjunto de algoritmos que permiten gestionar la tabla de arbitraje de los puertos de salida de conmutadores, encaminadores y terminales de una red InfiniBand. Los algoritmos obtenidos reflejan el comportamiento de un modelo particular de gestión de la tabla de arbitraje que ha sido propuesto en la tesis doctoral realizada por el profesor Francisco José Alfaro Cortés [1].

2. InfiniBand

InfiniBand define una red de área de sistema (System Area Network, SAN) para conectar ordenadores, sistemas de E/S y dispositivos de E/S tal y como se muestra en la Figura 1. InfiniBand proporciona la infraestructura adecuada para comunicación y gestión, tanto para transacciones de E/S como para comunicación entre ordenadores. Un sistema InfiniBand puede variar desde un pequeño servidor formado por un procesador y unos cuantos dispositivos de E/S conectados, hasta un supercomputador masivamente paralelo con miles de procesadores y dispositivos de E/S que está conectado vía Internet a otras plataformas de procesamiento y/o sistemas de E/S.

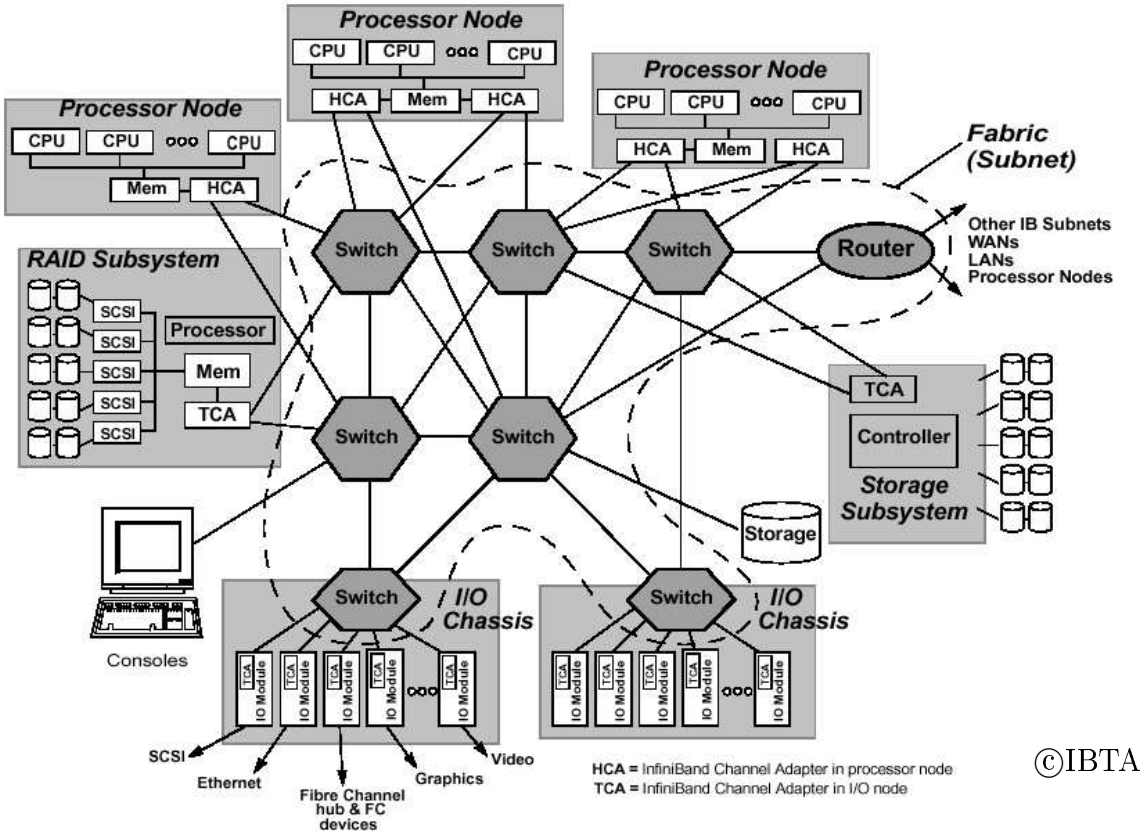


Figura 1: IBA System Area Network.

InfiniBand tiene una topología conmutada con conexiones punto a punto, lo que permite tanto topologías regulares como irregulares. El sistema de administración será capaz de identificar cualquier topología formada, y construir las tablas de encaminamiento adecuadas para permitir el intercambio de información entre dos elementos cualquiera conectados a través de la red. Una red InfiniBand está dividida en subredes interconectadas entre sí mediante routers o encaminadores. Los nodos finales estarán conectados a una única subred o a múltiples subredes por medio de distintas interfaces.

Una subred InfiniBand puede estar compuesta por nodos finales, conmutadores, encaminadores y el Subnet Manager, todos ellos interconectados por enlaces. Estos

enlaces pueden ser de cable, de fibra óptica o grabados en la placa base. Un channel adapter (CA) es el componente en un nodo final que lo conecta a la red. Los channel adapter pueden clasificarse en: Host Channel Adapter (HCA) y Target Channel Adapter (HCA). Conceptualmente tanto el HCA como el TCA son idénticos. Desde un punto de vista arquitectónico, el HCA difiere del TCA porque el HCA tiene una interfaz más estructurada que ofrece al sistema operativo.

Un encaminador (Router) se encarga de comunicar entre sí distintas subredes. Los encaminadores tampoco tienen capacidad para generar o consumir paquetes (sin contar los de control). Nuevamente, se limitan a trasvasarlos desde uno de sus puertos de entrada a uno de los de salida. En este caso la decisión de qué puerto de salida usar se toma en base a una dirección global (GID). Esta dirección global es única en todas las redes InfiniBand, mientras que la dirección local (LID) es propia de cada subred.

Cada conmutador de la subred tiene una tabla de encaminamiento que está basada en los identificadores locales, tal que cada paquete cuando viaja por la subred lleva incorporadas la dirección del puerto fuente (SLID) y la del puerto destino (DLID). En caso de que viaje entre varias subredes InfiniBand, las direcciones globales fuente (SGID) y destino (DGID) serán usadas por los encaminadores intermedios.

El funcionamiento de InfiniBand se describe mediante la interacción de una serie de niveles. El protocolo que gobierna cada nivel es independiente del resto de niveles. Cada nivel usa los servicios que le proporciona el nivel inferior y a su vez proporciona una serie de servicios al nivel inmediatamente superior.

La especificación de InfiniBand define los mensajes y protocolos de gestión de la red. Estos protocolos se agrupan en servicios de la subred. En cada subred debe existir un Subnet Manager encargado de configurar la red y administrar la información necesaria para las operaciones de la subred. Cada clase de gestión permite gestionar diversos aspectos de la red de forma independiente. El Subnet Manager (SM) se encarga de configurar y mantener la subred. El SM utiliza un formato especial de paquete llamado Subnet Management Packet (SMP) que tiene prioridad sobre el resto de paquetes.

El lector interesado puede acudir a las especificaciones de InfiniBand para más detalles. A parte de estos, existen otros documentos que puede ser útiles para la introducción del lector en InfiniBand, tales como [9, 3]

3. Calidad de Servicio en InfiniBand

InfiniBand proporciona varios mecanismos que permiten al administrador de la subred gestionar distintas garantías de calidad de servicio, tanto para servicios con conexión como sin conexión. En los servicios con conexión se puede intentar garantizar calidad de servicio. Sin embargo, en los servicios sin conexión no es posible dar garantías, pues no se conoce a priori los requisitos que necesitan las aplicaciones. Para este tipo de servicio se puede configurar la subred con algún tipo de mecanismo similar a los que hacen los servicios diferenciados.

Los mecanismos que proporciona InfiniBand para conseguir proporcionar calidad de servicio son básicamente: canales virtuales y arbitraje en dichos canales virtuales.

3.1. Canales Virtuales

Los puertos en InfiniBand tienen canales virtuales (Virtual Lane, VL). Un canal virtual representa un conjunto de buffers para transmisión y recepción en un puerto. Cada puerto puede tener un número de canales virtuales diferentes (Figura 2). Durante la fase de configuración ambos extremos de enlace deben ponerse de acuerdo para funcionar con el número mínimo de enlaces de los dos.

Por otra parte, cada paquete contiene un nivel de servicio (Service Level, SL) que determina qué VL se usará en cada uno de los enlaces por donde pasa. De esta forma, es posible que puedan funcionar en la misma subred dispositivos que implementan distinto número de canales virtuales.

Cada puerto de salida (en conmutadores, encaminadores y nodos terminales) tiene una tabla de correspondencia entre los SLs y los VLs (SLtoVLMappingTable) configurada por las entidades de gestión de la red. Una correcta configuración de esta tabla hará que cada tipo de tráfico, previamente segregado en distintos niveles de servicio, use canales virtuales diferentes.

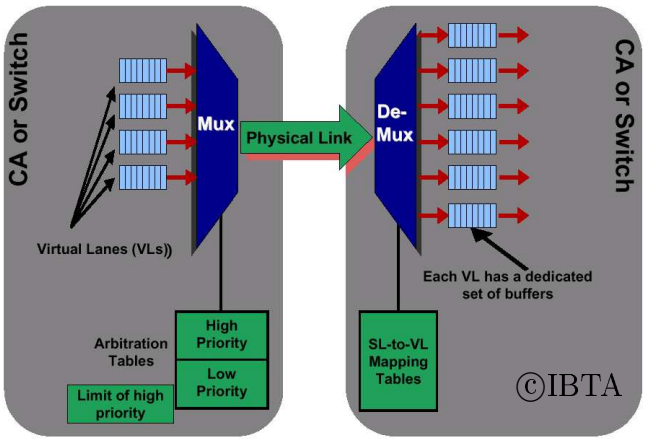


Figura 2: Funcionamiento de los canales virtuales en un canal físico.

El control de flujo se aplica a cada VL independientemente. Cuando se definen más

de dos canales virtuales, la prioridad de los canales de datos se define en la tabla de arbitraje.

3.2. Arbitraje de los canales virtuales

Los puertos de salida (tanto en los conmutadores, como en los encaminadores y en los nodos terminales) incorporan una tabla de arbitraje que permite especificar la prioridad que tendrá cada canal virtual a la hora de enviar paquetes a través de ese puerto de salida.

Es obligatorio implementar esta tabla si el puerto tiene más de dos canales virtuales. El arbitraje se realiza sólo entre los canales virtuales de datos, pues el canal virtual del tráfico de control siempre tiene preferencia.

Cada tabla de arbitraje (Figura 3) está formada por dos tablas, una para gestionar el tráfico de los canales virtuales de alta prioridad y otra para los canales virtuales de baja prioridad. Sin embargo, InfiniBand no especifica qué es alta y baja prioridad, quedando este aspecto a criterio del administrador de la subred.

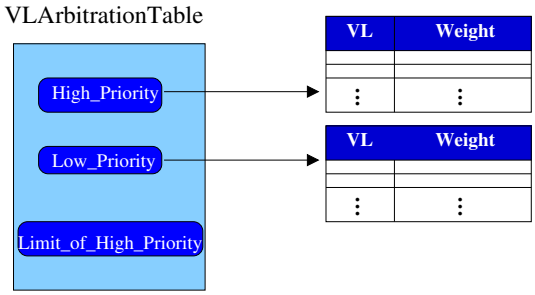


Figura 3: Estructura de la tabla de arbitraje.

Cada una de esas tablas tiene un funcionamiento cíclico y ponderado. Tanto la tabla de alta prioridad como la de baja prioridad tienen un máximo de 64 entradas. Cada entrada especifica un canal virtual y un peso. El peso indica la cantidad de unidades de 64 bytes a enviar por ese canal virtual. El peso debe estar entre 0 y 255, y siempre se redondea a un paquete completo.

El valor LimitOfHighPriority determina la cantidad máxima de información que puede sacarse de los canales virtuales de alta prioridad, antes de pasar a enviar un paquete de un canal virtual de los considerados de baja prioridad. En concreto, los canales virtuales situados en la tabla de alta prioridad pueden enviar LimitOfHighPriority x 4096 bytes antes de enviar un paquete de baja prioridad. Una vez enviado el paquete de un canal virtual de baja prioridad (si había alguno listo para enviarse), se vuelve a la tabla de alta prioridad y se continúa por la entrada donde se quedó y con el peso que le restaba.

Una característica importante del funcionamiento de la tabla de arbitraje es que si no hay paquetes de alta prioridad para ser transmitidos, se puede pasar a transmitir paquetes de baja prioridad. De esta forma no se desperdicia ancho de banda, y cuando

no haya tráfico de alta prioridad se puede pasar a transmitir otro tipo de tráfico de menos prioridad aprovechando el ancho de banda disponible.

4. Modelo formal de la tabla de arbitraje

4.1. Introducción

La tabla de arbitraje está compuesta por un número fijo de entradas. En InfiniBand son 64, aunque la teoría desarrollada es válida para cualquier número de entradas. Para identificarlas el modelo propuesto usa una numeración basada en la permutación bit-reversal. Esta permutación es tal que aplicada a una secuencia $m = b_{n-1}b_{n-2}...b_1b_0$ de dígitos, obtiene como resultado la secuencia $\Re(m) = b_0b_1...b_{n-2}b_{n-1}$. En la Figura 4.1 se ve un ejemplo de la numeración utilizada y su comparación con la numeración correlativa habitual.

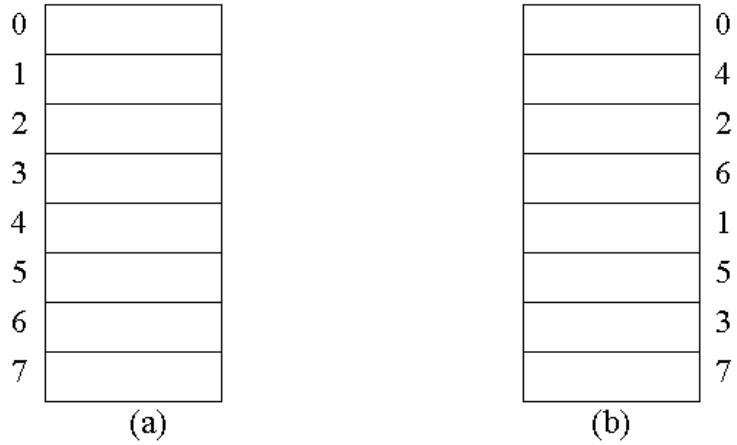


Figura 4: Tabla de 8 entradas, (a) numeración correlativa de las entradas, (b) numeración basada en la aplicación de la permutación bit-reversal

En la tabla de arbitraje se pueden distinguir diferentes secuencias de entradas en función de la distancia entre dos entradas consecutivas de la secuencia. En el ejemplo de la Figura 4 (b) podemos tener 1 secuencia de 8 entradas separadas a distancia 1 formada por todas las entradas, 2 secuencias disjuntas formadas por 4 entradas cada una separadas entre sí a distancia 2 que son (0, 2, 1, 3) y (4, 6, 5, 7), 4 secuencias de 2 entradas a distancia 4 que son (0, 1), (4, 5), (2, 3) y (6, 7), y 8 secuencias de distancia 8 formadas cada una por una única entrada. Cada una de estas secuencias forma un conjunto y son las encargadas de atender una petición de esa distancia que reciba la tabla de arbitraje.

Conjuntos con entradas a distancia 1 entre sí	{0, 1, 2, 3, 4, 5, 6, 7}
Conjuntos con entradas a distancia 2 entre sí	{0, 1, 2, 3} {4, 5, 6, 7}
Conjuntos con entradas a distancia 4 entre sí	{0, 1} {2, 3} {4, 5} {6, 7}
Conjuntos con entradas a distancia 8 entre sí	{0} {1} {2} {3} {4} {5} {6} {7}

Los conjuntos de la tabla se identifican como $E_{i,j}$ siendo i el nivel al que pertenece el conjunto y j el número de la primera entrada perteneciente a ese conjunto. Es

importante indicar que el valor de i depende de la distancia que haya entre dos entradas consecutivas de un mismo conjunto. En el Cuadro 1 se utilizan algunos de estos conjuntos como ejemplo en los que se indican las entradas que poseen.

Conjunto	Nº Entradas	Entradas que lo forman
$E_{2,16}$	16	16, 24, 20, 28, 18, 26, 22, 30, 17, 25, 21, 29, 19, 27, 23, 31
$E_{6,12}$	1	12
$E_{4,60}$	4	60, 62, 61, 63
$E_{3,40}$	8	40, 44, 42, 46, 41, 45, 43, 47

Cuadro 1: Entradas que conforman algunos conjuntos.

A partir de ahora, y con el objetivo de hacer más sencillos y claros los comentarios aquí presentados, se ha considerado una tabla con 5 niveles únicamente, y no con los 7 que realmente tiene la tabla de arbitraje de los puertos de salida en InfiniBand. De esta forma, las representaciones de la tabla y de la estructura auxiliar que más adelante se explica, podrán tener un tamaño que permita apreciar mucho mejor lo que se quiere indicar con ellas.

Cada conjunto se puede dividir en dos subconjuntos disjuntos, teniendo cada uno de ellos la mitad de entradas. Así por ejemplo, el conjunto $E_{2,16}$ puede dividirse en dos subconjuntos que son $E_{3,16}$ y $E_{2,24}$, siendo además disjuntos entre sí. A partir de esto, se puede decir que un conjunto es el padre de los dos que hay en el nivel inferior al suyo, siendo denominados éstos hijos del anterior y hermanos entre sí. Siguiendo con el conjunto $E_{2,16}$, éste forma parte de los conjuntos $E_{0,0}$ y $E_{1,0}$. Por tanto, todos los conjuntos de niveles superiores que contengan a uno dado se dice que son conjuntos ancestros de este último.

Un conjunto está libre si todas sus entradas están sin usar, y es denominado conjunto singular si a la vez de estar libre, su hermano no lo está, ya sea porque está ocupado o porque alguna de sus entradas está asignada. Como el conjunto $E_{0,0}$ no tiene hermano, no podrá ser, por definición, un conjunto singular.

En la Figura 5 se puede ver una tabla de arbitraje (de sólo 16 entradas) en la que se han sombreado los conjuntos que están libres, estando el resto ocupados. De los conjuntos libres, destacan $E_{3,0}$, $E_{2,4}$ y $E_{1,8}$, los cuales son conjuntos singulares, y están sombreados con un color más oscuro que el resto de conjuntos libres, que están incluidos dentro de ellos como antes se ha explicado.

Utilizando el concepto de conjunto singular se explican los términos de tabla normalizada y de tabla ordenada, pues dichas situaciones están marcadas por el número de conjuntos singulares que haya y por la situación que tengan entre ellos dentro de la tabla de arbitraje.

Una tabla está normalizada si a lo sumo hay un conjunto singular por nivel. Es decir, en cada nivel puede que no haya conjuntos singulares, pero en el caso de que haya, éste ha de ser único. La tabla mostrada en la Figura 5 está normalizada, porque en el nivel 4 no hay ningún conjunto singular, y los niveles 1, 2 y 3 poseen uno ($E_{3,0}$, $E_{2,4}$ y $E_{1,8}$). Sin embargo, la tabla de arbitraje de la Figura 6 no está normalizada, ya

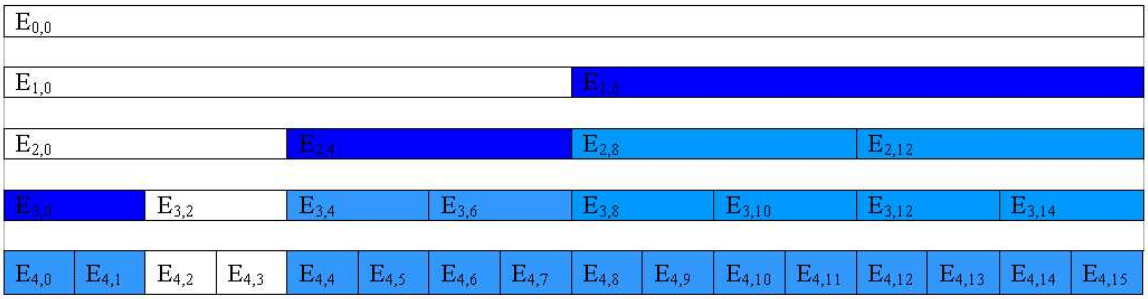


Figura 5: Tabla con tres conjuntos singulares

que el nivel 3 posee dos conjuntos singulares ($E_{3,0}$ y $E_{3,4}$).

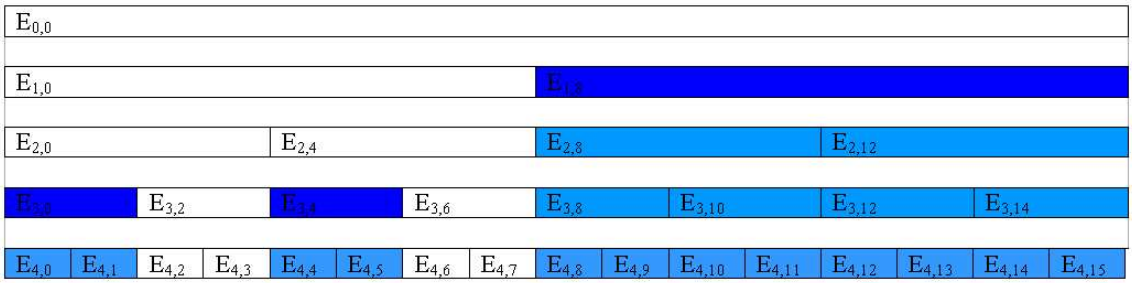


Figura 6: Ejemplo de tabla no normalizada y ordenada

Una tabla está ordenada si los conjuntos singulares están ordenados de menor a mayor tamaño de izquierda a derecha. En la Figura 6 se ve que la tabla está ordenada, puesto que se respeta el orden de menor a mayor que implica que una tabla esté ordenada. Los conjuntos $E_{3,0}$ y $E_{3,4}$ están más a la izquierda que el conjunto $E_{1,8}$ que es de mayor tamaño. Un ejemplo de tabla no ordenada es la tabla de la Figura 7, en la que se puede observar como $E_{2,0}$ está más a la izquierda que $E_{3,4}$ que es más pequeño que él.

Ahora bien, si se observan las Figuras 5, 6 y 7, se puede llegar a la conclusión de que los conceptos de tabla normalizada y tabla ordenada son totalmente independientes, ya que hay un caso de tabla no normalizada pero ordenada (Figura 6), otra que está normalizada pero no ordenada (Figura 7) y otra que sí está tanto normalizada como ordenada (Figura 5).

Tras ser descritos los conceptos básicos necesarios para poder comprender todo el trabajo que se desarrolla a continuación, se exponen las ideas básicas que se propusieron en [1] para la gestión de la tabla de arbitraje y a partir de los cuales se ha desarrollado este trabajo.

El primero de los algoritmos propuestos es el de asignación o inserción, que se encarga de encontrar un conjunto libre para ubicar una petición. El tipo de petición viene dado por el nivel al que pertenece el conjunto solicitado. El algoritmo selecciona el conjunto de menor tamaño capaz de atender la petición que ha sido solicitada. En otras palabras, selecciona el conjunto libre situado más a la izquierda en el árbol que

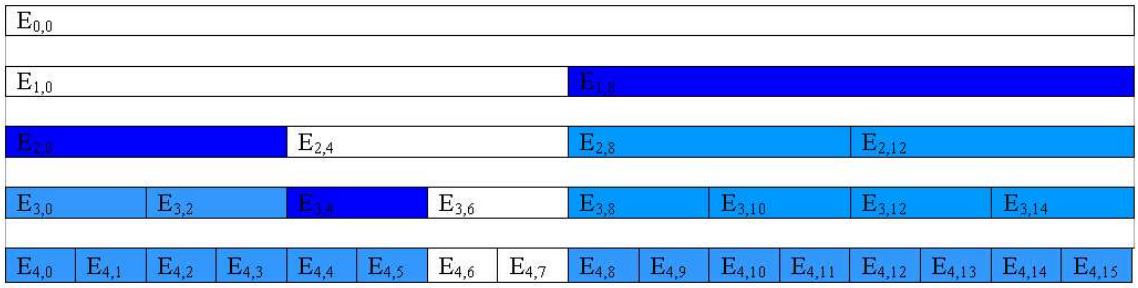


Figura 7: Ejemplo de tabla no ordenada

pueda atender a la petición que lo ha solicitado. En [1] se demuestra que siempre que haya un número suficiente de éstas, encuentra una secuencia que asignar para una distancia dada.

Ejemplo 1 Como situación inicial se tiene la mostrada en la Figura 8. En ese momento llega una petición que requiere cuatro entradas. Por tanto, se necesita un conjunto de nivel 2. Como ya se ha mencionado, el algoritmo selecciona el primero libre que encuentre buscando de izquierda a derecha. En este ejemplo, ese conjunto es $E_{2,8}$, por lo que ha de serle asociado a la nueva petición, y tras ser ocupado, el estado de la tabla de arbitraje es el mostrado en la Figura 9.

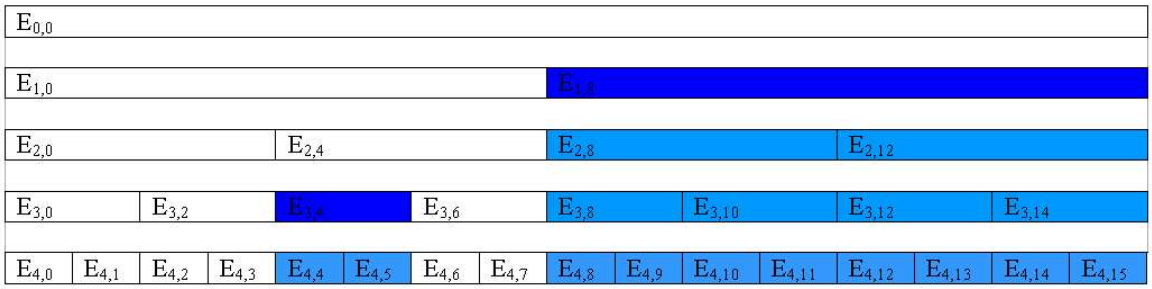


Figura 8: Tabla del Ejemplo 1 antes de aplicar el algoritmo de asignación

Cuando se realiza una eliminación, se liberan entradas ocupadas. Al liberarlas, se puede dar el caso de que la tabla deje de estar ordenada, normalizada o ambos casos de manera simultánea.

Ejemplo 2 Partiendo de la Figura 9, se da la situación en la que se ha de eliminar el conjunto que atiende a una petición, en concreto, $E_{2,0}$. La tabla inicialmente está tanto normalizada como ordenada, pero tras liberar el conjunto indicado, queda tal como se ve en la Figura 10. Se observa que en esta nueva situación la tabla no está normalizada, ya que hay dos conjuntos singulares de nivel 2 ($E_{2,0}$ y $E_{2,12}$) y tampoco está ordenada, al estar $E_{2,0}$ más a la izquierda que $E_{3,4}$ que es más pequeño.

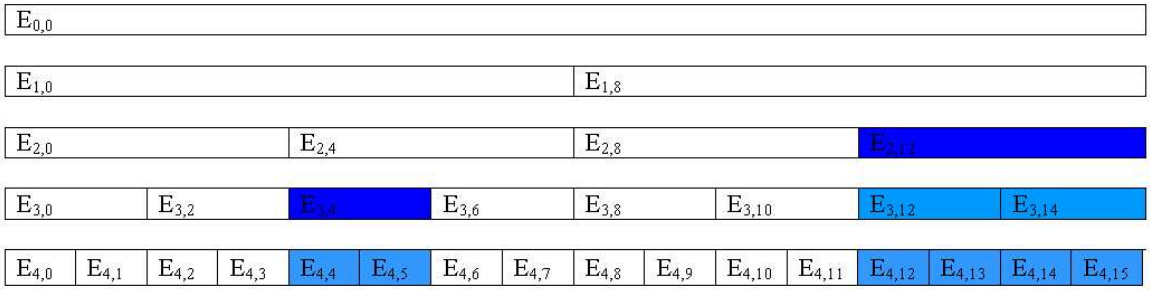


Figura 9: Tabla del Ejemplo 1 tras asignar el conjunto $E_{2,8}$

Estando la tabla no normalizada, puede que se dé el caso en el que el algoritmo de asignación no pueda ocupar una secuencia de entradas para una petición que las solicite, aún habiendo un número de entradas suficientes para ello (por ejemplo, una petición que requiera 8 entradas en la Figura 10). Este caso es consecuencia de que las entradas no pertenecen a un mismo conjunto. Casos así son el motivo de la necesidad de un procedimiento capaz de unir conjuntos libres para formar uno del máximo tamaño posible. A este algoritmo se le denomina desfragmentación.

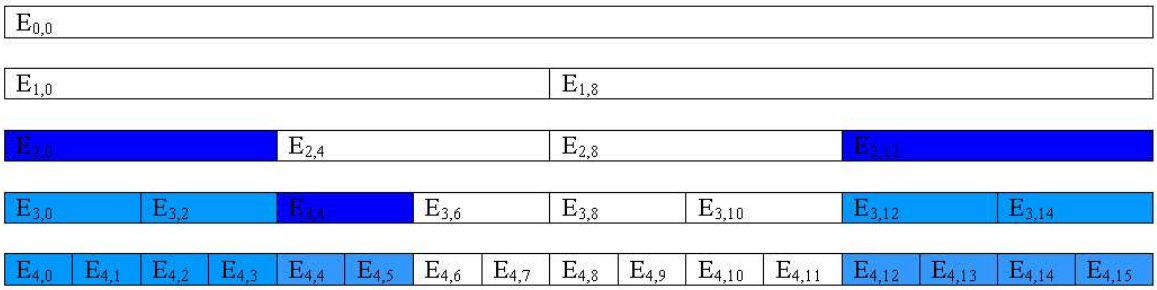


Figura 10: Tabla no ordenada ni normalizada del Ejemplo 2 tras liberar el conjunto $E_{2,0}$

Ejemplo 3 Continuando con el Ejemplo 2 por la situación reflejada en la Figura 10, en el caso de que llegase una petición que requiera 8 entradas libres (tipo 1), la petición no podrá ser atendida aún habiendo 10 entradas libres. Al no estar agrupadas en un mismo conjunto, el algoritmo de asignación no podrá asignársele un conjunto libre a la última petición solicitada. Sin embargo, uniendo los conjuntos libres $E_{2,0}$ y $E_{2,12}$ para formar uno de mayor tamaño, $E_{1,8}$, ya se puede asignar un conjunto a la petición que llegó solicitando 8 entradas puesto que $E_{1,8}$ es capaz de albergarla. El resultado de esa unión se refleja en la tabla de arbitraje de la Figura 11.

El procedimiento de desfragmentación también es útil tras una asignación de entradas. Se puede ver en el siguiente ejemplo.

Ejemplo 4 Partiendo de la Figura 10 a la que se ha llegado tras liberar $E_{2,0}$ y sin haber realizado desfragmentación alguna, llega una petición de tipo 3. Siguiendo el

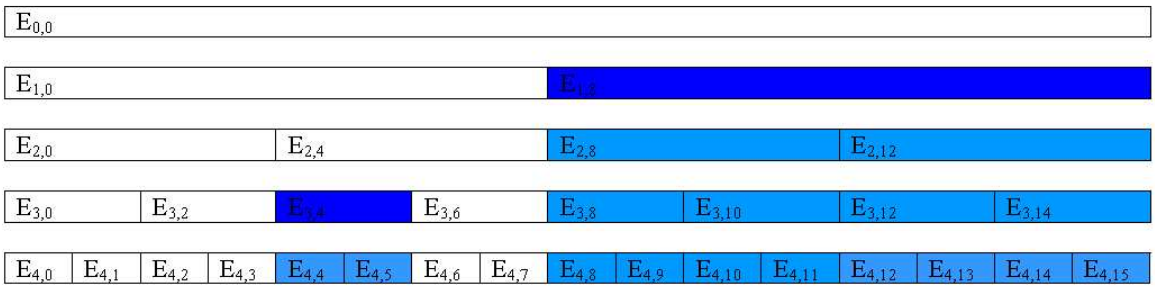


Figura 11: Tabla del Ejemplo 2 tras aplicar el algoritmo de desfragmentación

algoritmo de asignación, se ocupa el conjunto $E_{3,0}$, quedando entonces la tabla tal y como indica la Figura 12. Puede observarse que la tabla no está normalizada, ya que en el nivel 3 se pueden encontrar dos conjuntos singulares $E_{3,2}$ y $E_{3,4}$. Así pues se le ha de aplicar desfragmentación para poder atender en el futuro peticiones de tipo 1, ya que hay suficientes entradas.

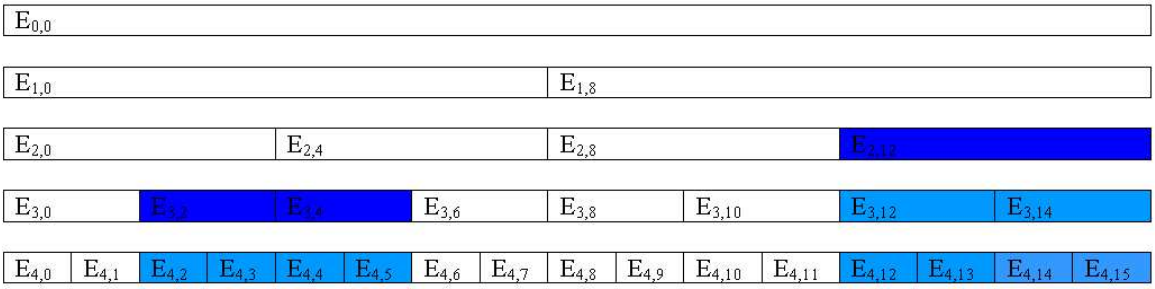


Figura 12: Situación tras la asignación de $E_{3,0}$ en el Ejemplo 4

Para conseguir detectar un conjunto desordenado y deshacer esa situación, se utiliza el algoritmo de reordenación. Este algoritmo consiste en recorrer la tabla y comprobar si a partir de un conjunto singular $E_{i,j}$, existe otro conjunto singular $E_{l,m}$ tal que este último sea de mayor tamaño ($l < i$) y esté situado en la tabla más a la izquierda que el primero ($m < j$). En ese caso se procede al intercambio de $E_{l,m}$ y del ancestro de $E_{i,j}$ en el nivel l.

Ejemplo 5 Se toma como comienzo del ejemplo la Figura 13, en la que se ha liberado previamente el conjunto $E_{3,14}$. La tabla no está ordenada, pero lo estaría si $E_{2,8}$ estuviera a la derecha del $E_{3,14}$. Por tanto, se intercambian $E_{2,8}$ y $E_{2,12}$ (Ancestro de $E_{3,14}$ en el nivel 2) quedando la situación como representa la Figura 14. Puede observarse que la tabla ahora está ordenada, y por tanto si recibiera cualquier petición que pudiera atender, seguiría estando ordenada y normalizada tras la asignación tal y como se menciona en [1] al explicar las propiedades del algoritmo de inserción.

figure

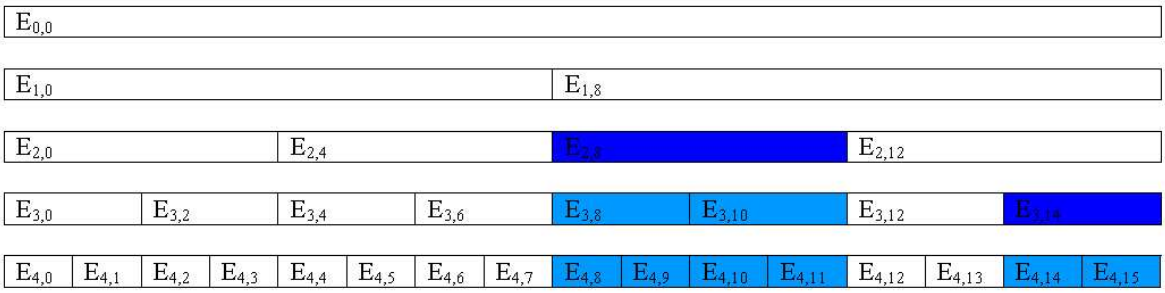


Figura 13: Situación tras la liberación de $E_{3,14}$

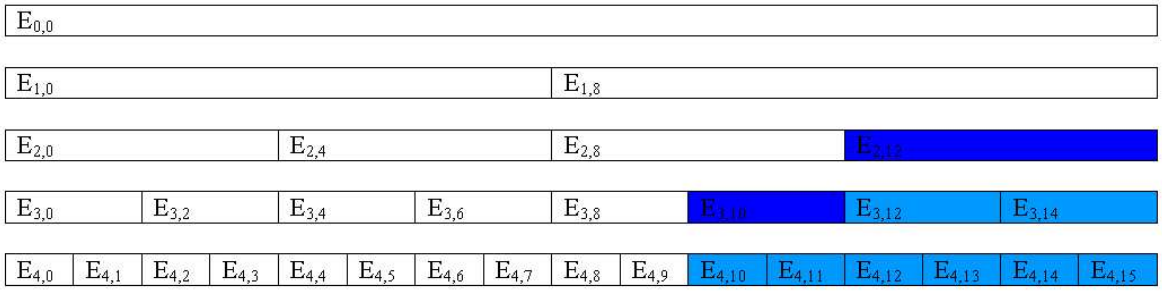


Figura 14: Situación tras realizarse la reordenación en el Ejemplo 5

El uso adecuado de los algoritmos de inserción, desfragmentación y reordenación permite gestionar de una forma eficiente la tabla de arbitraje en InfiniBand. En [1] se incluyen una serie de teoremas, y sus correspondientes demostraciones, que así lo acreditan.

Una vez descritos básicamente todos los conceptos y algoritmos básicos referentes al tratamiento general de la tabla, se presentan los programas correspondientes a esos algoritmos. Antes de ello, se incluye una descripción de una estructura en la que se apoyan esos algoritmos.

4.2. Estructura singulares

En la sección anterior se han introducido diversos conceptos relacionados con la tabla de arbitraje. De todos ellos uno muy importante es el de conjunto singular, pues aparece en varios de los procesos para el tratamiento de la tabla. Se ha pensado desarrollar una estructura que facilite la gestión de la tabla de arbitraje de InfiniBand utilizando el concepto de conjunto singular.

Ejemplo 6 Se tiene una tabla de arbitraje con 16 entradas. De ella sólo se sabe que tiene como conjuntos singulares a $E_{2,4}$, $E_{2,12}$, $E_{3,0}$ y $E_{4,11}$. Con esta información se puede saber fácilmente que conjuntos están libres y cuales no.

Ya se tiene una imagen del estado de la tabla de arbitraje obteniéndola a partir de los conjuntos singulares, tal y como se puede ver en la Figura 15.

Teniendo en cuenta toda la información que pueden llegar a dar los conjuntos singulares, se proporciona la estructura singulares en la cual se apoyan todas las propuestas

Conjunto singular	Conjuntos libres dentro del conjunto singular
$E_{2,4}$	$E_{2,4} - E_{3,4}, E_{3,6} - E_{4,4}, E_{4,5}, E_{4,6}, E_{4,7}$
$E_{2,12}$	$E_{2,12} - E_{3,12}, E_{3,14} - E_{4,12}, E_{4,13}, E_{4,14}, E_{4,15}$
$E_{3,0}$	$E_{3,0} - E_{4,0}, E_{4,1}$
$E_{4,11}$	$E_{4,11}$

Cuadro 2: Conjuntos libres del Ejemplo 6

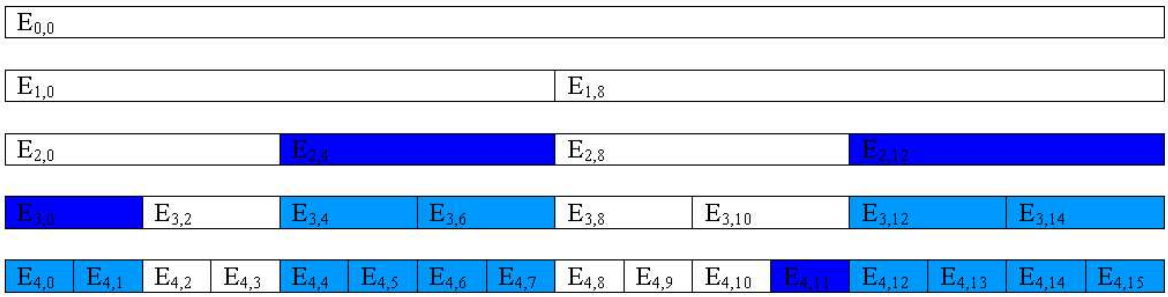


Figura 15: Estado de la tabla de arbitraje del Ejemplo 6

que se van a exponer a continuación. Esta estructura de datos almacena, para cada nivel, los índices, en concreto dos, de los conjuntos singulares que posee ordenándolos de izquierda a derecha, así como el número de ellos que hay. La razón de que sólo sea necesario almacenar los índices de como mínimo dos conjuntos por nivel es la siguiente. Una tabla de arbitraje antes de que se le aplique una inserción o eliminación siempre está normalizada con lo que el número máximo de conjuntos singulares en un nivel es uno. Sin embargo, cuando se produce una eliminación o una inserción (en este último caso puede que sea sobre una tabla no ordenada), la tabla puede quedar no normalizada. Así pues, el número máximo de conjuntos singulares que puede haber en un nivel es de 2, el que había anteriormente y el que se acaba de formar. Inmediatamente después se realizará una desfragmentación dejando la tabla nuevamente normalizada. La principal utilidad de esta estructura es la de que al saber la localización en la tabla de arbitraje de los conjuntos singulares, se puede tener cierta idea del estado que tiene ésta última ya que sabiendo donde se sitúa un conjunto singular, se pueden saber todos los conjuntos que hay libres y por tanto, también se pueden saber los que están ocupados.

La estructura de datos que se ha elegido para representar los conjuntos singulares es una tabla con tantas columnas[1] como tipos de peticiones haya (niveles en el árbol) y tres filas, dos de ellas para almacenar los índices de los dos posibles conjuntos singulares que pueda haber por nivel, y la fila 0 indica el número de conjuntos singulares que contiene cada nivel. Puede verse gráficamente en la Figura 16.

Suponiendo que se implementase en C, su declaración sería:

```
typedef char singulares[emax+1][3];
```

[1]Es importante decir que aún sabiendo que el conjunto de nivel 0 no puede ser singular por definición, es necesario para el buen funcionamiento de otras funciones, que en caso de que sea un conjunto libre, lo reconozca como singular.

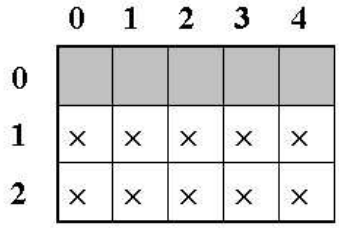


Figura 16: Representación de la estructura singulares

siendo emax el valor del nivel más alto. Por tanto, el tamaño de la estructura, en concreto para una tabla de 64 entradas sería de:

$$7\text{niveles} \times 3\frac{\text{entradas}}{\text{nivel}} \times 8\frac{\text{bits}}{\text{entrada}} = 168\text{bits} = 21\text{bytes}$$

Ejemplo 7 A partir de los datos que se dan en el Ejemplo 6 y la Figura 15, la estructura singulares que la representa sería la mostrada en la Figura 17.

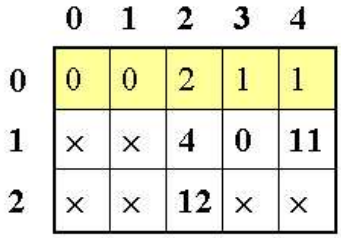


Figura 17: Representación de la estructura singulares de tabla mostrada en la Figura 15

Esta estructura se utiliza, entre otras muchas cosas, para comprobar si un conjunto está ocupado o libre, informar sobre si hay dos singulares en un mismo nivel para detectar posibles casos de tabla no normalizada y para encontrar conjuntos desordenados al reordenar. Con esto se consigue, por ejemplo, no tener que recorrer la tabla de arbitraje para realizar todas esas tareas.

4.3. Implementación de los algoritmos para el tratamiento de la tabla de arbitraje

En esta sección se describen las implementaciones que se han propuesto para los algoritmos descritos de manera básica en [1]. Se han buscado implementaciones eficientes y sencillas. También se describen otros algoritmos auxiliares que se usan en aquellos.

4.3.1. Algoritmo de asignación o inserción

Este algoritmo de asignación o inserción es el encargado de ubicar una nueva petición en la tabla. Dada una petición de tipo n, las entradas han de estar separadas entre

sí una distancia $d = 2^n$. Por esto, se necesitan max/d entradas, siendo max el número de entradas de la tabla de arbitraje. El código básico de este algoritmo se muestra en la Figura 18.

Entradas:

- **nivel:** Nivel al cual pertenece el conjunto que va a atender la petición

Salida:

- Devuelve verdadero si ha encontrado algún conjunto que pueda albergar a la petición recién llegada. En caso contrario devuelve falso.
- En *"indice"* alberga el índice del conjunto asignado en caso de encontrar un conjunto.

```
1:   booleano      insercion(nivel, *indice)
2:   nuevasEntradas=2^(emax-nivel);
3:   si (nuevasEntradas+numEntradas > max) entonces
4:         resultado= falso;
5:   sino
6:         *indice=buscaPrimeraEntradaNuevaPeticion(nivel);
7:         actualizarSingularesAntesAnnadirPeticion(E_nivel,*indice);
8:         ocupa(E_nivel,*indice);
9:         resultado=verdadero;
10:  finsi
11:  devolver resultado;
12:  Fin insercion
```

Figura 18: Algoritmo de inserción

El algoritmo propuesto comprueba inicialmente si hay suficientes entradas libres para atender en la tabla de arbitraje una petición del nivel que se le ha indicado (Figura 18, líneas 2-4). En caso de que no sea posible, es decir, no hay entradas suficientes, devuelve directamente falso (Figura 18, línea 4). En el caso de que sí pueda ser atendida, busca el conjunto de entradas adecuado.

La búsqueda la realiza otro método llamado *buscaPrimeraEntradaNuevaPeticion* (Figura 18, línea 6) almacenando en índice el valor del índice del conjunto que va a ser ocupado. Tras encontrar el conjunto que va a ser asociado a la petición que lo ha solicitado, actualiza la estructura singulares (Figura 18, línea 7). Una vez actualizada la estructura singulares, reflejando que el conjunto seleccionado ha dejado de estar libre, se puede ocupar el conjunto seleccionado ya que todas las operaciones internas que se realizan y que necesitan que la estructura singulares se encuentre actualizada, podrán funcionar correctamente (procedimiento ocupa, Figura 18, línea 8). Por último, como en este caso se ha encontrado un conjunto que atienda la petición, se devuelve verdadero.

4.3.2. Algoritmo buscaPrimeraEntradaNuevaPeticion

Este procedimiento se encarga de recorrer los conjuntos en busca de uno que esté libre capaz de atender a una petición que ha solicitado max/d entradas libres y que esté situado lo más a la izquierda posible.

Entrada:
- **nivel:** Nivel del conjunto que se está buscando.

Salida:
- Índice j del conjunto libre $E_{i,j}$ situado más a la izquierda.

```
1:  Procedimiento buscaPrimeraEntradaNuevaPeticion(nivel)
2:  menorIndice=max;
3:  para i=nivel hasta 0 hacer
4:      si (daNumConjuntosSingulares(i) == 1) entonces
5:          auxIndice=daIndiceConjuntoSingular(i,1);
6:          si (auxIndice < menorIndice) entonces
7:              menorIndice=auxIndice;
8:          finsi
9:      finsi
10: finpara
11: devolver menorIndice;
11: Fin buscaPrimeraEntradaNuevaPeticion
```

Figura 19: Algoritmo buscaPrimeraEntradaNuevaPeticion

El procedimiento recorre la estructura singulares desde el nivel solicitado por la petición hasta el nivel 0 (Figura 19, líneas 3-10), mirando el índice del primer conjunto singular de cada uno de ellos en caso de tener alguno. De todos ellos, se queda con el índice de menor valor, o sea, el situado más a la izquierda (Figura 19, líneas 4-7), y al final de la búsqueda, devuelve su valor (Figura 19, línea 11).

Ejemplo 8 Partiendo de la Figura 20, llega una petición de tipo 3. Para asignarle un conjunto, el algoritmo de asignación llama primero al método *buscaPrimeraEntradaNueva-Petición(3)*. Este procedimiento mientras recorre la estructura singulares desde el nivel 3 hasta el nivel 0, va consultando los índices de los primeros conjuntos singulares de cada uno. Se llega a que el situado más a la izquierda es el conjunto $E_{3,10}$.

Así pues, ese conjunto se le asigna a la petición que acaba de solicitar ser atendida (Figura 21).

Ejemplo 9 Partiendo de la Figura 22, llega una petición de tipo 3. Para asignarle un conjunto, el algoritmo de asignación llama primero al método *buscaPrimeraEntradaNueva-Peticion(3)*. Este procedimiento mientras recorre la estructura singulares desde

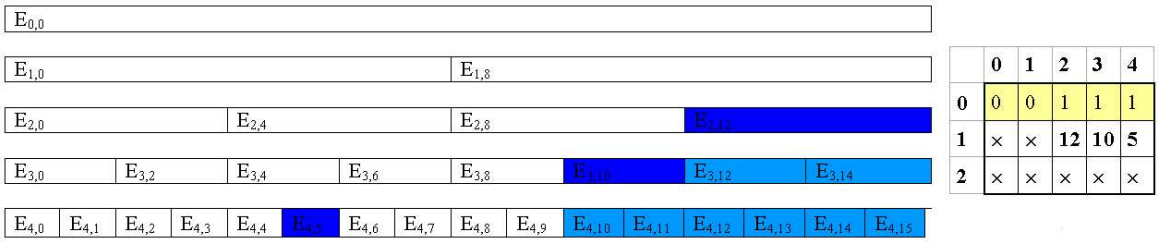


Figura 20: Situación inicial del Ejemplo 8

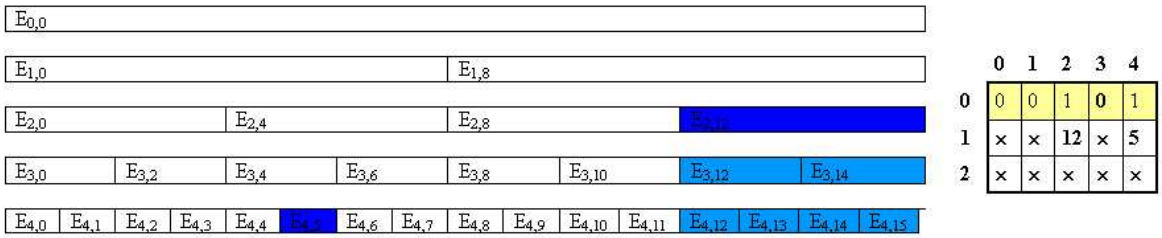


Figura 21: Situación final del Ejemplo 8

el nivel 3 hasta el nivel 0, va consultando los índices de los primeros conjuntos singulares de cada uno. Ve que en los niveles 3 y 2 no hay ningún conjunto singular. Cuando llega al nivel 1 y ve que hay un conjunto singular en dicho nivel, mira cuál es el índice del conjunto singular perteneciente a dicho nivel y descubre que el índice es el del conjunto $E_{1,8}$ y por ello es 8 el valor que devuelve *buscaPrimeraEntradaNuevaPetición*.

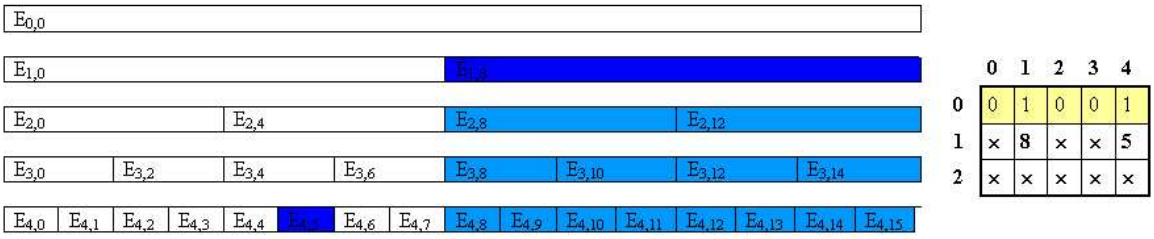


Figura 22: Situación inicial del Ejemplo 9

Así pues, el conjunto $E_{3,8}$ quedando finalmente la situación de la tabla tal y como se muestra en la Figura 23.

4.3.3. Algoritmo de desfragmentación

El propósito de este procedimiento es el de normalizar una tabla que ha dejado de serlo debido a una eliminación o una inserción. Para ello procederá a agrupar dos conjuntos singulares de un mismo nivel, repitiendo la operación tantas veces como sea necesario (hacia niveles superiores para al unir dos conjuntos en un nivel se pueden tener dos conjuntos singulares en algún nivel superior).

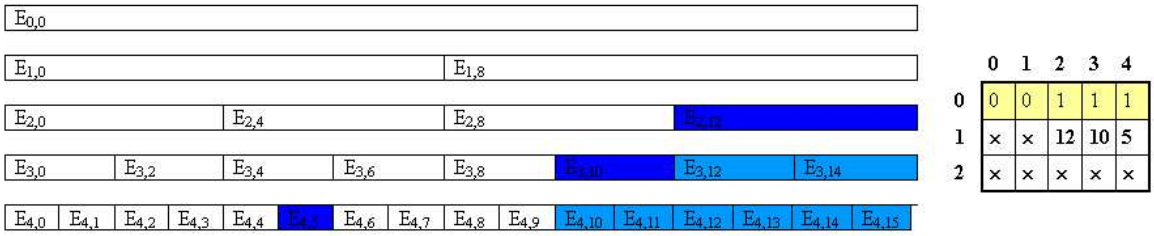


Figura 23: Situación final del Ejemplo 9

Entrada:

- **nivel:** Nivel donde se ha añadido o eliminado una petición en la tabla.

```
1:   Procedimiento desfragmentacion(nivel)
2:   para i= nivel hasta 1 hacer
3:       numSingulares=daNumConjuntosSingulares(i);
4:       si (numSingulares == 2) entonces
5:           indice1=daIndiceConjuntoSingular(i,1);
6:           indice2=daIndiceConjuntoSingular(i,2);
7:           E_{i,k}=hermano(E_{i,indice2});
8:           intercambioConjuntos(E_{i,indice1},E_{i,k});
9:       finsi
10:  finpara
11:  Fin desfragmentacion
```

Figura 24: Algoritmo desfragmentacion

El algoritmo recorre los niveles desde el que se le ha pasado como argumento hasta el nivel 1 (Figura 24, líneas 2-10). Para ello se hace uso de la estructura singulares. En ese recorrido, consulta el valor que almacena el número de conjuntos singulares que tiene cada nivel (Figura 24, línea 3). En caso de que en un nivel haya dos, simplemente coge sus índices de la estructura, calcula el hermano del segundo y lo intercambia con el primero llamando al procedimiento *intercambioConjuntos* (Figura 24, líneas 4-8). Tras el intercambio, continúa con el nivel superior. Hay que indicar que los conjuntos para realizar la desfragmentación se eligen tal que el nuevo conjunto de mayor tamaño quede más a la derecha. Con ello se quiere mantener la tabla ordenada cuando sea posible.

Ejemplo 10 Partiendo de la Figura 25, tras la eliminación del conjunto $E_{2,0}$, la tabla pasa a no estar normalizada ya que $E_{2,0}$ y $E_{2,12}$ son conjuntos singulares de un mismo nivel. El algoritmo de desfragmentación consulta el contador de conjuntos singulares de nivel 2 que hay en la estructura singulares. El resultado de esa consulta es que hay dos conjuntos singulares.

Por todo eso, se ha de realizar un intercambio entre dos conjuntos para unir a los dos conjuntos singulares y así formar uno solo. A continuación se intercambian

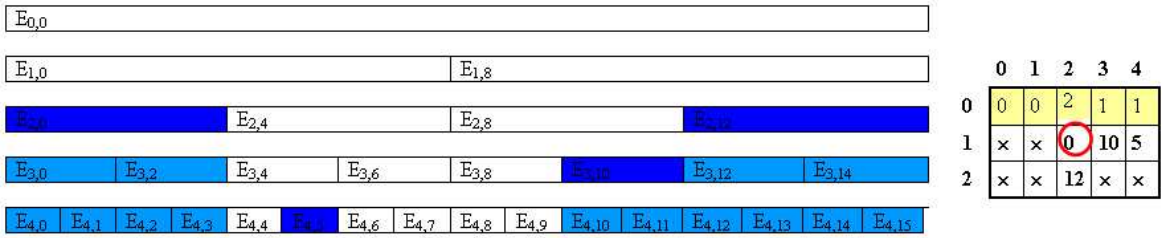


Figura 25: Situación inicial del Ejemplo 10

los conjuntos $E_{2,0}$ y $E_{2,8}$, siendo este último hermano del segundo conjunto singular ($E_{2,12}$). Una vez se ha realizado el intercambio, la situación queda tal y como indica la Figura 26. Después, el procedimiento sigue investigando sobre si hay que realizar más intercambios motivados por la posible existencia de dos conjuntos singulares en niveles superiores. Tras llegar al nivel 1 y ver que no se han de realizar más intercambios, concluye su ejecución.

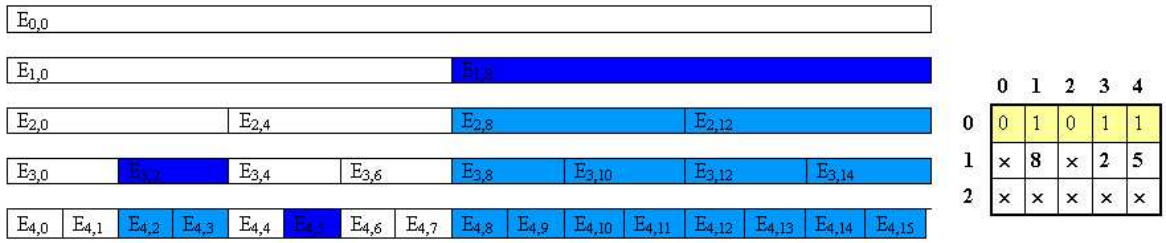


Figura 26: Situación final del Ejemplo 10

Ejemplo 11 Partiendo de la Figura 27, tras la eliminación del conjunto $E_{4,3}$, la tabla pasa a no estar normalizada ya que $E_{4,3}$ y $E_{4,5}$ son conjuntos singulares de un mismo nivel. El algoritmo desfragmentación consulta el contador de conjuntos singulares de nivel 4 que hay en la estructura singulares. El resultado de esa consulta es que hay dos conjuntos singulares.

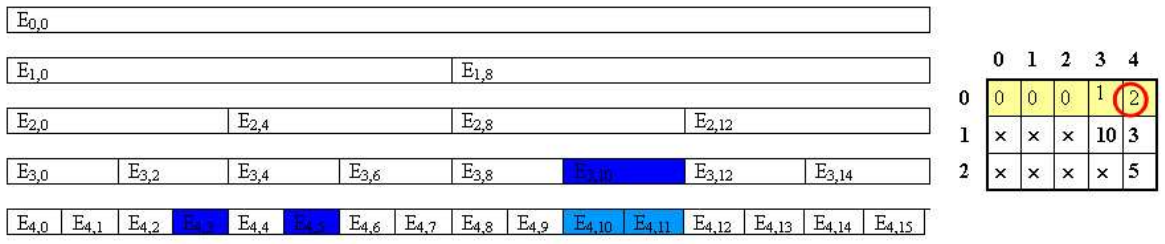


Figura 27: Situación inicial del Ejemplo 11

Por todo eso, se ha de realizar un intercambio entre dos conjuntos para unir a los dos conjuntos singulares y así formar uno solo. A continuación se intercambian los conjuntos $E_{4,3}$ y $E_{4,4}$. Una vez se ha realizado el intercambio, la situación queda tal y

como indica la Figura 28. El procedimiento sigue por el nivel 3 mirando en la estructura singulares si hay nuevamente dos conjuntos singulares en un mismo nivel y al consultar el contador de conjuntos singulares del nivel 3 que también hay dos ($E_{3,4}$ y $E_{3,10}$).

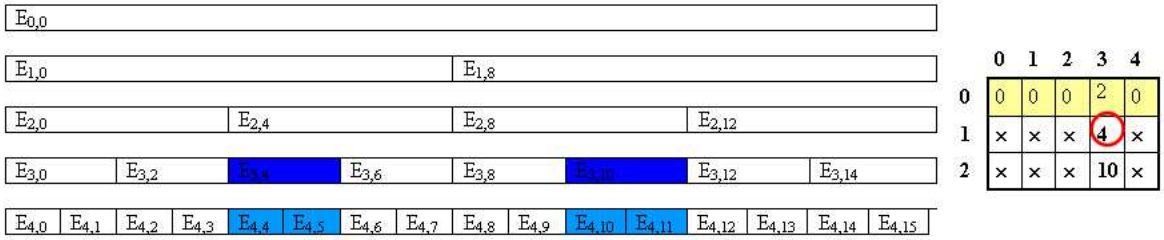


Figura 28: Situación tras el primer intercambio en el ejemplo 11

Así pues, vuelve a realizar otro intercambio para intentar que la tabla vuelva a estar normalizada siendo esta vez los conjuntos implicados $E_{3,4}$ y $E_{3,8}$, siendo el resultado del intercambio el estado que presenta la tabla de arbitraje de la Figura 29. Tras este último intercambio, el algoritmo sigue mirando si en niveles superiores hay más conjuntos singulares, siendo esta vez el resultado de dicha búsqueda negativo.

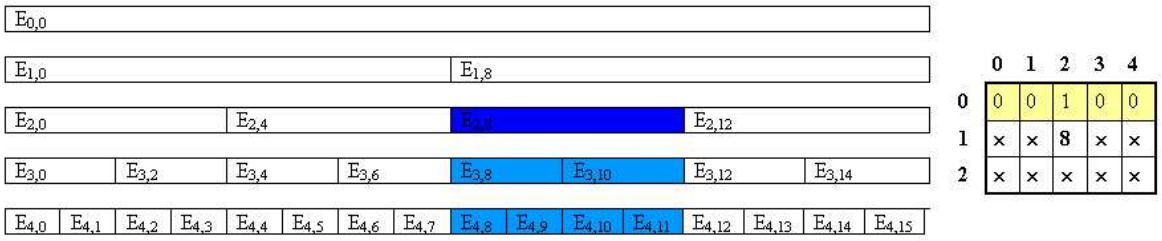


Figura 29: Situación final del Ejemplo 11

4.3.4. Algoritmo de reordenación

Al igual que el método desfragmentación, la finalidad de este algoritmo es la de que los conjuntos singulares estén dispuestos en la tabla de arbitraje tal que los conjuntos de mayor tamaño estén más a la derecha. Este algoritmo también hace uso de la estructura singulares, pues evita tener que ir recorriendo la tabla para buscar los conjuntos singulares.

El pseudocódigo del algoritmo reordenación se muestra en la Figura 30 situada en la página siguiente.

Para el funcionamiento de este algoritmo no es necesario pasarle ningún parámetro, ni tampoco devuelve un resultado. El algoritmo se limita a modificar adecuadamente la tabla de arbitraje y la estructura singulares. El algoritmo recorre todos los niveles desde *emax* hasta 2 (Figura 30, líneas 2-30) y para cada uno de los conjuntos singulares $E_{i,j}$, busca algún otro que esté desordenado respecto a él (Figura 30, líneas 11-24). Durante la búsqueda, de entre todos los conjuntos desordenados va quedándose con el situado más al a izquierda (Figura 30, líneas 19-24). Además realiza la cuenta de

cuántos lleva encontrados (Figura 30, líneas 16-17). Por último termina la búsqueda y si ha encontrado algún conjunto $E_{k,l}$ de menor nivel situado más a su izquierda, lo intercambia con el ancestro de $E_{i,j}$ en el nivel k (Figura 30, líneas 25-26). Si encuentra más de un conjunto desordenado, vuelve a repetir la búsqueda por si alguno aún sigue desordenado después de haber cambiado la ubicación de $E_{i,j}$.

Entrada:

- $E_{i,j}$: Conjunto del que se quiere saber si está libre u ocupado.

Salida:

- Devuelve verdadero si $E_{i,j}$ está libre, y en caso de que no lo esté, devuelve falso.

```
1:  Procedimiento reordenacion()
2:  para nivel=emax hasta 2 hacer
3:      numSingulares=daNumConjuntosSingulares(nivel);
4:      para posicion=1 hasta numSingulares hacer
5:          encontrados=2;
6:          mientras (encontrados>=2) hacer
7:              encontrados=0;
8:              menor=daIndiceConjuntoSingular(nivel,posición);
9:              menorNivel=nivel;
10:             auxIndex=menor;
11:             para j=1 hasta (nivel-1) hacer
12:                 auxPos=1;
13:                 auxNumSingulares=daNumConjuntosSingulares(j);
14:                 para auxPos=1 hasta auxNumConjuntosSingulares(j,auxPos) hacer
15:                     actual=daIndiceConjuntoSingular(j,auxPos);
16:                     si (auxIndex>actual) entonces
17:                         encontrados++;
18:                     finsi
19:                     si (actual<menor) entonces
20:                         menor=actual;
21:                         menorNivel=i;
22:                     finsi
23:                 finpara
24:             finpara
25:             si (menor!=auxIndex) entonces
26: intercambio conjuntos(E_menorNivel,menor,E_menorNivel,k);//l=ancestro(auxIndex,menorNivel)
27:             finsi
28:         finmientras
29:     finpara
30: finpara
31: Fin reordenacion
```

Figura 30: Algoritmo reordenación

Aunque esta comprobación se podría realizar recorriendo todas y cada una de las entradas que forman un conjunto viendo si están libres o no, se aprovecha la estructura auxiliar singulares para consultarlo partiendo de la idea de que un conjunto está libre si es singular o está incluido en otro conjunto que sí lo sea. Así pues, el procedimiento que sigue este algoritmo es primero comprobar si el conjunto que se ha pasado como argu-

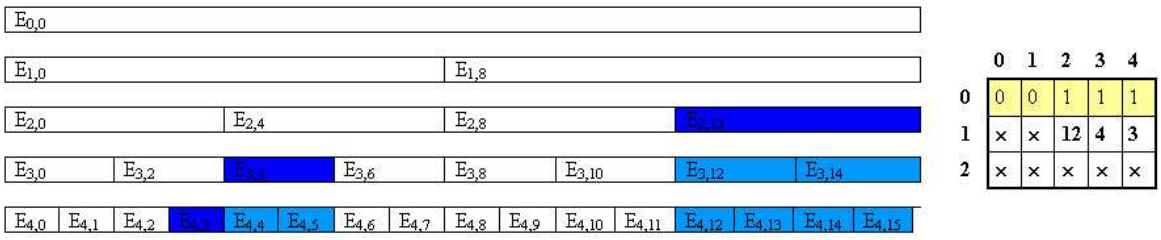
mento es singular (Figura 31, líneas 2-3). En caso de no serlo, comprueba si el conjunto que se quiere saber si está libre tiene algún conjunto ancestro que sea singular llamando a *daNivelConjuntoAncestroSingular* (Figura 31, línea 5). Esta función obtiene el nivel al que pertenece su conjunto singular ancestro ya que en caso de tenerlo devolverá un valor que estará dentro del rango de posibles niveles de la tabla de arbitraje (0-emax), y por lo tanto, el resultado será afirmativo (Figura 31, líneas 6-10).

```
1:   booleano esConjuntoLibre (Ei,j)
2:   si (esConjuntoSingular(Ei,j) entonces
3:           resultado=verdadero;
4:   sino
5:           aux=daNivelConjuntoAncestroSingular (Ei,j);
6:           si ((aux>=0) AND (aux<emax)) entonces
7:                   resultado=verdadero;
8:           sino
9:                   resultado=falso;
10:          finsi
11:  finsi
12:  devolver resultado;
13:  Fin esConjuntoLibre
```

Figura 31: Algoritmo esConjuntoLibre

En los dos siguientes ejemplos se toma como punto de partida la Figura 32.

Ejemplo 12 Se invoca a *esConjuntoLibre*($E_{3,12}$). El algoritmo primero comprueba si es un conjunto singular, siendo el resultado de esa consulta negativo. A continuación, comprueba si tiene algún conjunto ancestro singular llamando a *daNivelConjuntoAncestroSingular* ($E_{3,12}$) devolviendo el valor 2, ya que el conjunto $E_{2,12}$ es un conjunto ancestro suyo y singular a la vez. Por lo tanto, el conjunto $E_{3,12}$ si está libre.



	0	1	2	3	4
0	0	0	1	1	1
1	×	×	12	4	3
2	×	×	×	×	×

Figura 32: Situación inicial de los Ejemplos 12 y 13

Ejemplo 13 Se invoca a *esConjuntoLibre*($E_{4,9}$). Aquí también mira primero si el conjunto $E_{4,9}$ es un conjunto singular y ve que no lo es. A continuación, llama a *daNivelConjuntoAncestroSingular* ($E_{4,9}$) para ver si tiene algún conjunto singular ancestro

devolviendo el valor -1, que indica que no posee ninguno. Por tanto, el conjunto $E_{4,9}$ no está libre y se devolverá falso.

4.3.5. Algoritmo daNivelConjuntoAncestroSingular

Este procedimiento busca, a través de los conjuntos ancestros del conjunto pasado como argumento $E_{i,j}$, uno que sea singular para devolver el nivel del citado conjunto ancestro. El pseudocódigo del algoritmo *daNivelConjuntoAncestroSingular* se muestra en la Figura 33.

Primero se comprueba si $E_{i,j}$ es un conjunto singular (Figura 33, línea 2). Si no lo es, se obtienen sus ancestros uno a uno, y con la ayuda de la estructura singulares va comprobando si alguno de ellos es singular (Figura 33, líneas 5-14). En caso de encontrarlo, se devuelve el nivel al que pertenece el que cumple el requisito de ser singular (Figura 33, líneas 9-11). Si el conjunto pasado como argumento no tiene ningún ancestro singular, se devuelve el valor -1 para indicar esa situación.

Ejemplo 14 Partiendo de la Figura 34, se hace una llamada a *daNivelConjuntoAncestroSingular ($E_{4,13}$)*. Primero comprueba si el conjunto $E_{4,13}$ es singular, viendo que no lo es. Por tanto, se han de calcular sus ancestros mientras no se encuentre ninguno que sea singular. $E_{3,12}$ es el primero en ser calculado, pero utilizando la estructura singulares, se comprueba que este no es el ancestro que se aestán buscando. Lo mismo ocurre con $E_{2,12}$. Al llegar a $E_{1,8}$, la consulta sobre si es un conjunto singular da resultado afirmativo, dando fin a la búsqueda y quedando sólo devolver 1.

4.3.6. Algoritmo esConjuntoSingular

Este procedimiento comprueba si un conjunto es singular. Esta función es utilizada por las tareas más importantes que se llevan a cabo tanto directa como indirectamente (desfragmentación, reordenar, comprobar si un conjunto está libre, etc).

Todo el proceso se lleva a cabo utilizando la estructura singulares. Primero examina si hay conjuntos singulares en el nivel del que está realizando la consulta (Figura 35, líneas 3-4). Si hay alguno, consulta los índices de ese nivel en búsqueda de uno que coincida (Figura 35, líneas 4-9), y si lo encuentra, devuelve verdadero. Si no lo hay, simplemente devuelve falso.

Para los siguientes ejemplos, se toma como situación inicial la Figura 36.

Ejemplo 15 Se invoca a *esConjuntoSingular($E_{4,13}$)*. El procedimiento comprueba si en el nivel 4 hay algún conjunto singular y ve que sí lo hay. Entonces procede a buscar el índice 13 en singulares, sin tener éxito en dicha búsqueda, devolviendo falso y dando así constancia de que no es un conjunto singular.

Ejemplo 16 Esta vez se invoca a *esConjuntoSingular($E_{3,4}$)*. Siguiendo otra vez los pasos, comprueba si en el nivel 3 hay algún conjunto singular consultando el contador que

Entrada:

- $E_{i,j}$: Conjunto del que se quiere conocer el nivel de su ancestro singular, si lo tiene.

Salida:

- El nivel del citado conjunto ancestro en caso de que exista, y -1 si $E_{i,j}$ no tiene ningún ancestro singular.

```
1:  entero daNivelConjuntoAncestroSingular(E_{i,j})
2:  si (esConjuntoSingular(E_{i,j}) entonces
3:          resultado=-1;
4:  sino
5:          nivel=i;
6:          continuar=verdadero
7:          mientras ((nivel>0) AND (continuar)) hacer
8:                  E_{nivel-1,k}=conjuntoPadre(nivel-1, E_{i,j});
9:                  si esConjuntoSingular(E_{nivel-1,k}) entonces
10:                         resultado=nivel-1;
11:                         continuar=false;
12:                 finsi
13:                 nivel--
14:         finmientras
15:         si (continuar) entonces
16:                 resultado=-1;
17:         finsi
18: finsi
19: devolver resultado
20: Fin daNivelConjuntoAncestroSingular
```

Figura 33: Algoritmo daNivelConjuntoAncestroSingular

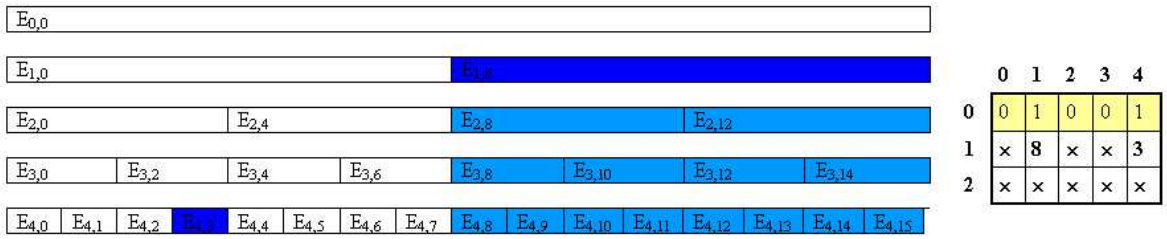


Figura 34: Situación del Ejemplo 14

hay en la estructura singulares para ese nivel. Una vez se ha comprobado que hay uno, se examina si alguno de los índices de los conjuntos singulares de ese nivel es el 4, que corresponde con el índice del conjunto que se ha tomado como argumento, por lo que se procede a informar que $E_{3,4}$ sí es un conjunto singular.

Entrada:

- $E_{i,j}$: Conjunto del que se quiere comprobar si es singular.

Salida:

- Devuelve verdadero si $E_{i,j}$ es un conjunto singular, y en caso de que no lo sea, devuelve falso.

```
1:   booleano esConjuntoSingular(E_{i,j})
2:   resultado=false;
3:   numSingulares=daNumConjuntosSingulares(i);
4:   k=1;
5:   mientras (k>=numSingulares) AND (NOT (resultado)) hacer
6:           auxIndex=daIndiceConjuntoSingular(i, k);
7:           si (auxIndex==j) entonces
8:                   resultado=verdadero;
9:           finsi
10:          k++;
11:  finmientras
12:  devolver falso;
13:  Fin esConjuntoSingular
```

Figura 35: Algoritmo esConjuntoSingular

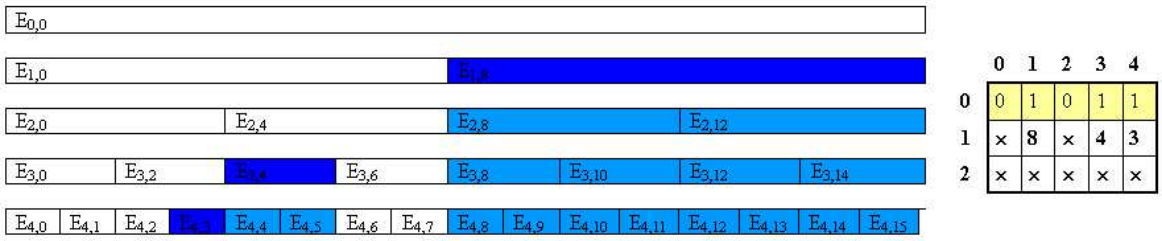


Figura 36: Situación para los Ejemplos 15 y 16

4.4. Inserción inteligente

En la propuesta que se acaba de presentar en la sección anterior, tanto el algoritmo de desfragmentación como el de reordenación surgen para que la asignación no cause ningún problema debido a su funcionamiento. A continuación, se suponen dos casos en los que se deja de utilizar uno de ellos (desfragmentación o reordenación), viendo qué efectos se producen y posibles soluciones para suplir su ausencia.

Si se dejase de utilizar el algoritmo de desfragmentación, la asignación no podría en muchos casos atender la petición más restrictiva ya que la tabla estaría en muchas ocasiones no normalizada y aún habiendo el número de entradas necesario, no estarían formando un conjunto capaz de atender una petición. A este inconveniente se suma el hecho de que al poderse dar casos en los que la tabla de arbitraje no esté normalizada, puede haber más de dos conjuntos singulares en algunos niveles, siendo preferible tener

que implementar dicha estructura de datos singulares de manera dinámica, lo cual ralentizaría todo lo referido al mantenimiento de dicha estructura de datos. La única solución sería la de coger cierto número de entradas que puedan atender la petición solicitada y que a su vez, no pertenezcan a un mismo conjunto. Al analizar esta posible solución, se ve que no está de acuerdo con la idea de tabla de arbitraje que se propuso utilizando el concepto de conjunto. Por lo tanto, no hay solución que pueda mejorar a priori el rendimiento de lo propuesto.

Si se eliminara el algoritmo de reordenación se tendrían situaciones donde habría que aplicar inserciones sobre una tabla no ordenada. Al producirse esto, podría suceder que la tabla pasase a estar también no normalizada debido al funcionamiento del algoritmo de asignación, que busca el conjunto libre lo más a la izquierda posible capaz de albergar la petición solicitada. Esto haría necesario desfragmentar también tras una inserción. En este caso sí hay solución factible. Esa solución se llama inserción inteligente y se desarrolla a continuación.

Ejemplo 17 El punto de partida de este ejemplo es la tabla de arbitraje de la Figura 37. Se observa que la tabla no está ordenada ya que el no aplicar el algoritmo de reordenación hace que el conjunto $E_{1,0}$ esté más a la izquierda que el conjunto $E_{3,8}$ y este a su vez más a la izquierda que $E_{4,15}$.

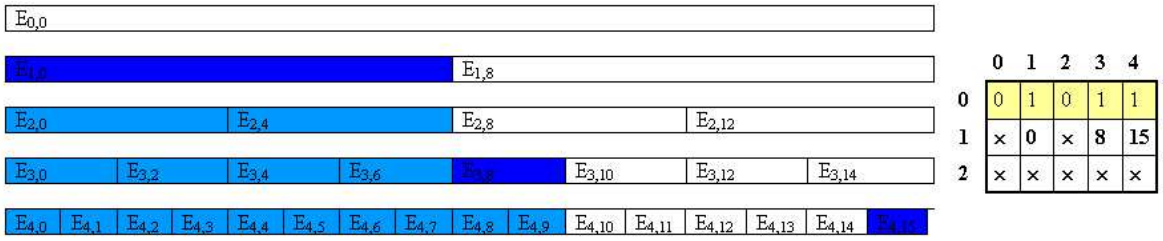


Figura 37: Situación inicial del Ejemplo 17

El algoritmo de asignación que se utilizaba antes elegiría al conjunto $E_{3,0}$ si se recibiera una petición de tipo 3. Esto provocaría que la tabla quedara tal y como se ve en la Figura 38.

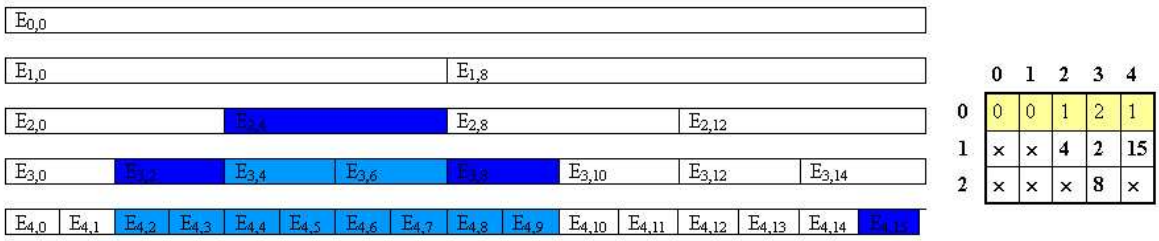


Figura 38: Situación tras ocupar el conjunto $E_{3,0}$ en el Ejemplo 17

La tabla de la Figura 38 no está normalizada, por lo que hay que llevar a cabo una desfragmentación, siendo el resultado final de este ejemplo la tabla de la Figura 39.

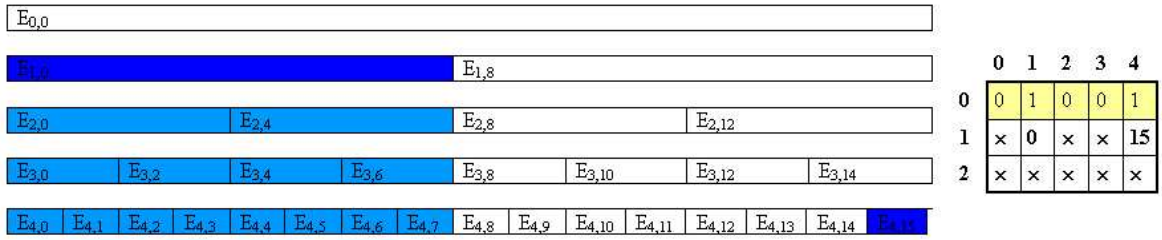


Figura 39: Situación final tras desfragmentar en el Ejemplo 17

Ahora el algoritmo de asignación (se seguirá denominando así) hay que modificarlo para que en vez de llamar a *buscaPrimeraEntradaNuevaPeticion* para buscar un conjunto que cumpla las condiciones solicitadas, llame a *busquedaInteligenteDeConjunto*. También se elimina la utilización del algoritmo reordenación reflejando así la situación que se supuso en el párrafo anterior. El pseudocódigo del algoritmo *busquedaInteligenteDeConjunto* se encuentra en la Figura 40.

En *busquedaInteligenteDeConjunto* se sigue utilizando la estructura singulares. El procedimiento a seguir es simplemente recorrer a partir del nivel pasado como argumento hasta el 0 (Figura 40, líneas 2-6) los contadores de conjuntos singulares que poseen (Figura 40, línea 3), y en el momento en que se encuentre que hay alguno en uno de ellos, devolver el índice del conjunto singular que posea (Figura 36, línea 4). Si no hay ninguno, simplemente devuelve -1.

Entrada:
- **nivel:** Nivel del conjunto que se está buscando.

Salida:
- Índice j del conjunto libre $E_{i,j}$ situado más a la izquierda.

```
1:  entero busquedaInteligenteDeConjunto(nivel)
2:  para i=nivel hasta 0 hacer
3:      si (daNumConjuntosSingulares(i) == 1) entonces
4:          devolver daIndiceConjuntoSingular(i,1);
5:      finsi
6:  finpara
7:  devolver -1;
8:  Fin busquedaInteligenteDeConjunto
```

Figura 40: Algoritmo busquedaInteligenteDeConjunto

Ejemplo 18 El punto de partida de este ejemplo es la tabla de arbitraje de la Figura 41. Puesto que ya no se utiliza el método reordenar, se aplicará una nueva inserción directamente sobre esa situación. En esas circunstancias llega una petición de tipo 3.

El método *busquedaInteligente* empieza a recorrer la estructura singulares en el nivel 3. En ese nivel, ya hay un conjunto singular. Por ello, selecciona el primer conjunto

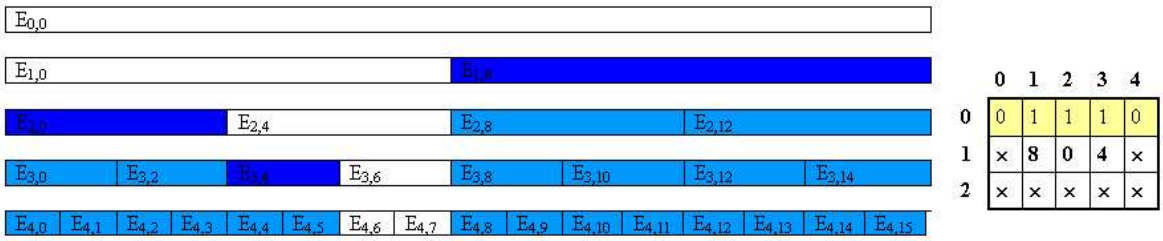


Figura 41: Situación inicial del Ejemplo 18

singular de ese nivel que es $E_{3,4}$. El resultado al ocuparlo es la tabla de arbitraje de la Figura 42(a). Sin embargo con el método de búsqueda anterior, se hubiera seleccionado a $E_{3,0}$ siendo su resultado la tabla de la Figura 42(b). Se ve como en el caso de la inserción inteligente, la tabla de arbitraje sigue estando normalizada, pero con la inserción anterior la tabla deja de estarlo.

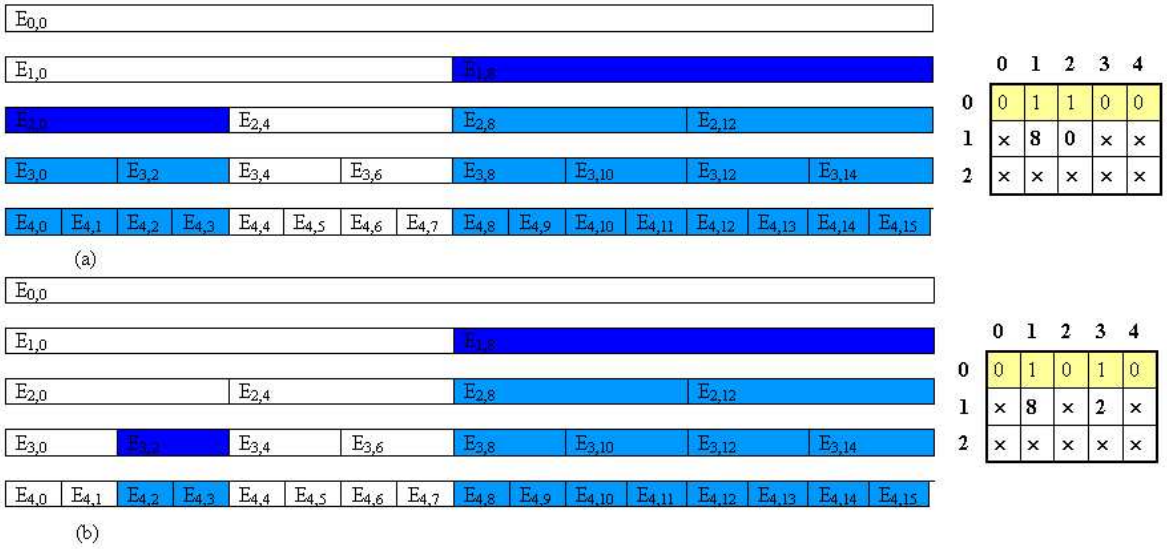


Figura 42: (a) Situación final del Ejemplo 18 tras la inserción inteligente.(b): Situación final del Ejemplo 18 tras el método de inserción no inteligente.

5. Evaluación de prestaciones y análisis de resultados

Para realizar la evaluación de prestaciones se ha seguido el método de simulación, creando un simulador que abstrae el comportamiento del modelo de gestión de la tabla de arbitraje expuesto anteriormente.

5.1. Modelos simulados

Para comprobar el funcionamiento de la tabla de arbitraje se simulan diferentes flujos de datos que requieren el uso de los enlaces de la red. Cada canal físico tiene un conjunto determinado de canales virtuales que son asignados a esos flujos de datos. El uso del canal físico vendrá determinado por la tabla de arbitraje de dicho canal. En esa tabla se dispone de una serie de entradas que son asignadas a los canales virtuales. El número de entradas asignadas a un flujo, la separación entre ellas y el peso de cada una de ellas establece el ancho de banda del canal, y la latencia máxima, para ese flujo.

Según vayan apareciendo flujos se van asignando entradas (o modificando pesos de unas ya existentes) y conforme finalicen esos flujos se liberan dichas entradas (o se actualizan los pesos correspondientes). Por tanto, a lo largo del tiempo, se van produciendo asignaciones (también se usa los términos reservas o inserciones para indicar lo mismo) y liberaciones de las entradas de las tablas. En el capítulo anterior se han presentado diversos algoritmos para realizar esta gestión de la tabla de arbitraje. La combinación de esos algoritmos tras una inserción o una eliminación ha dado lugar a varios modelos, de los cuales se quiere saber su comportamiento para determinar finalmente cuál de ellos es el mejor. Se indican a continuación cuáles son esos modelos.

Modelo 1 En este modelo, tras producirse una inserción o una eliminación en la tabla de arbitraje se aplica el algoritmo de desfragmentación.

Modelo 2 Tras una eliminación se aplica el algoritmo de reordenación primero y el de desfragmentación después. Tras una inserción no es necesario realizar operaciones de mantenimiento de la tabla de arbitraje.

Modelo 3 Este caso es casi idéntico al Modelo 2, excepto que tras una eliminación se aplica primero el algoritmo de desfragmentación y después el de reordenación. Tampoco en este caso es necesario realizar operaciones de mantenimiento de la tabla de arbitraje, después de una inserción.

5.2. Índices de prestaciones

Las medidas se obtienen realizando un cierto número de simulaciones, consistentes cada una de ellas en un millón de operaciones distribuidas aleatoriamente entre inserciones y eliminaciones, sobre la tabla de arbitraje. Se usará a partir de ahora el término operación para hacer referencia indistintamente a inserción o eliminación.

Una parte de los índices de prestaciones que se obtienen son tiempos empleados para realizar ciertas tareas. De estos tiempos, se obtiene:

- **Valor medio.**
- **Desviación típica.**
- **Peor caso.**
- **Muestras de cada tiempo.**

Los tiempos obtenidos son los siguientes:

- **Tiempo de ejecución de una operación de inserción.** Tiempo que tardan en realizarse todas las acciones necesarias para completar una inserción.
- **Tiempo de ejecución de una operación de inserción fallida.** Tiempo que tarda en ejecutarse un caso en el que se ha realizado una petición de inserción a la tabla de arbitraje, pero no ha podido ser atendida.
- **Tiempo de ejecución de una operación de eliminación.** Tiempo que tardan en realizarse todas las acciones necesarias para completar una eliminación.
- **Tiempo de una inserción propiamente dicha.** Tiempo que se emplea en ocupar las entradas que requiere una petición solicitada. Además, este tiempo se desglosa en los tiempos invertidos en actualizar la estructura singulares tras una inserción y en el tiempo utilizado en buscar un conjunto de la tabla para ser asignado.
- **Tiempo de una eliminación propiamente dicha.** Tiempo que se emplea en liberar las entradas asignadas a una petición, y de ese tiempo se tiene en cuenta el utilizado en actualizar la estructura singulares tras la eliminación.
- **Tiempo de desfragmentación.** Se mide el tiempo utilizado para llevar a cabo una desfragmentación, el tiempo empleado en realizar cada uno de los intercambios realizados durante esta operación y el tardado en buscar los dos conjuntos que han de ser intercambiados.
- **Tiempo de reordenación.** Es el tiempo usado para realizar una reordenación, desglosado en el tiempo invertido en cada intercambio realizado dentro de una reordenación y el que se tarda en encontrar un conjunto desordenado respecto de otro.
- **Tiempo de intercambio.** Tiempo empleado durante los intercambios realizados en desfragmentaciones y reordenaciones. Además se obtiene el tiempo utilizado en actualizar la estructura singulares tras el intercambio.

Otros datos que se obtienen son:

- **Peticiones solicitadas.** Se agrupan en función del nivel al que pertenezca el conjunto solicitado.

- **Peticiones atendidas.** De todas las peticiones solicitadas, se obtienen las que han sido aceptadas. También se agrupan por niveles.
- **Peticiones no atendidas.** Peticiones que no han sido aceptadas debido a que la tabla estaba totalmente ocupada.
- **Peticiones eliminadas.** Número de peticiones que han dejado de ser utilizadas, y han de abandonar la tabla de arbitraje.
- **Número de intercambios.** Distribución por niveles de los intercambios llevados a cabo durante la simulación.
- **Desfragmentaciones con intercambio.** Muestra el porcentaje de desfragmentaciones en las que se ha producido algún intercambio. La idea es comprobar el porcentaje de intercambios con respecto al número de desfragmentaciones.
- **Reordenaciones con intercambio.** Al igual que con las desfragmentaciones y con el mismo propósito, se toma el número de reordenaciones en las que se ha producido algún intercambio.
- **Número de intercambios en desfragmentaciones tras una inserción.** Esta medida solamente se toma en el modelo 1, con el fin de estudiar cuál es el efecto provocado por la inserción sobre la tabla de arbitraje haciendo que deje de estar normalizada.
- **Número de intercambios en desfragmentaciones tras una eliminación.** Al igual que la medida anterior, solamente se toma en el modelo 1 con el mismo fin respecto de la eliminación.

Los resultados de todos estos índices permitirán, además de conocer el comportamiento de cada uno de los modelos y por tanto determinar cuál se comporta mejor, descubrir e identificar las partes del código que mayor contribución tienen al tiempo total invertido. Esto ayudará, si fuera el caso, a encontrar alternativas mejores a las propuestas, lo que redundará en unas mejores prestaciones del modelo.

5.3. Presentación y análisis de resultados

Antes de nada, hay que indicar que todos los resultados que aquí se presentan están condicionados a que su valor sea menor a un umbral de tiempo establecido previamente. La causa que ha provocado el tener que establecer el umbral ha sido que la función utilizada para medir los tiempos de ejecución de cada tarea del simulador mide el tiempo que se invierte en otros procesos del sistema operativo que se estén ejecutando simultáneamente así como también los tiempos dedicados a consultas a memoria tras producirse fallos en cache. Este último caso provoca altos picos en las muestras obtenidas, tales que falsean las medias de todos los índices medidos, aún siendo el porcentaje de estos casos menor al 3 % del total. Es por ello que las medidas que se muestran son todas respecto a operaciones válidas.

5.3.1. Modelo 1

En este modelo, una operación de eliminación está compuesta por la eliminación propiamente dicha, que libera un conjunto de la tabla de arbitraje tras haber servido a la conexión que realizó una petición, seguida de una desfragmentación, con el objetivo de normalizar la tabla ante la posibilidad que existe de que la tabla de arbitraje deje de estarlo tras la liberación del conjunto. Una operación de inserción está compuesta a su vez por la reserva de un conjunto para una petición que ha realizado una solicitud seguida por una desfragmentación.

En la simulación de este modelo ha habido aproximadamente un 3 % de operaciones invalidadas.

Operación / Tarea	**Tiempo medio (microsegundos)**
Operación Inserción	29.51606979
Ocupación de conjunto	14.7126913
Búsqueda Inserción	2.29820253
Actualización singulares	3.15501726
Desfragmentación	11.0030499
Búsqueda	48.9749696
Intercambio	2.36846504
Operación Eliminación	32.6688208
Liberación de conjunto	12.256478
Actualización singulares	2.986545
Desfragmentación	12.256987
Búsqueda	35.236453
Intercambio	2.356487
Operación Inserción fallida	4.50739351

Cuadro 3: Operaciones

En el Cuadro 5.3.1 se ve que el hecho de haber pocos intercambios durante las desfragmentaciones que se realizan tras ocupar el conjunto hace que el tiempo invertido en la operación de inserción esté prácticamente marcado por el tiempo empleado durante la ocupación del conjunto.

En lo que a la operación de eliminación se refiere, viendo la Figura 44 se aprecia como su comportamiento es prácticamente similar al de la operación de inserción, salvo que al producirse más intercambios durante la desfragmentación, el porcentaje de operaciones marcadas por el tiempo transcurrido durante la liberación del conjunto es menor.

5.3.2. Modelo 2

Para este modelo se ha establecido que la operación eliminación esté formada por una eliminación seguida de una reordenación y tras ella una desfragmentación. Una operación de inserción está compuesta únicamente por la tarea encargada de asignar un conjunto a una petición que lo haya encargado.

En la simulación de este modelo, un 97.38 % de las operaciones fueron validas.

Operación / Tarea	**Tiempo medio (microsegundos)**
Operación Inserción	20.4631379
Ocupación de conjunto	15.4860794
Búsqueda Inserción	2.448428
Actualización singulares	3.29179197
Operación Eliminación	45.6902477
Liberación de conjunto	12.0599192
Actualización singulares	2.91499289
Reordenación	11.8999243
Búsqueda	2.9573179
Intercambio	40.6167184
Desfragmentación	11.9228966
Búsqueda	2.57360484
Intercambio	46.1881146
Operación Inserción Fallida	4.81503764

Cuadro 4: Tiempos de inserción

La operación de inserción, tal y como se ve en la Figura 43, prácticamente se comporta totalmente igual que el de ocupación del conjunto. Si se desplazase a la izquierda el tiempo de ocupación un intervalo igual al tiempo que se emplea en actualizar ciertas estructuras internas ajenas a la estructura singulares.

En la Figura 44, se puede ver que un gran número de las operaciones de eliminación siguen el comportamiento de la tarea de liberación del conjunto. Estas operaciones son las que simplemente llevan a cabo la liberación, seguidas de una reordenación y una desfragmentación sin intercambio alguno. El resto de casos, que es un porcentaje bajo del total de operaciones puesto que en sólo el 9.05 % de reordenaciones y el 8.72 % de las desfragmentaciones se lleva a cabo algún intercambio, y hacen que los tiempos de ejecución vayan mucho más allá del empleado por la mayoría de operaciones de este tipo.

5.3.3. Modelo 3

Este modelo difiere del Modelo 2 en el orden en el que se ejecutan la reordenación y la desfragmentación. En este caso, se realiza primero la desfragmentación seguida por la reordenación.

El porcentaje de ocurrencias de operaciones invalidadas durante esta simulación es de un 2.55 %.

Al observar la Figura 45, se aprecia que la operación de inserción se comporta prácticamente igual a la tarea que ocupa el conjunto que ha sido solicitado por una petición, tal y como pasaba en el modelo anterior.

Con la operación de eliminación ocurre igual que con la inserción, se comporta tal y como ocurre la operación eliminación en el Modelo 2.

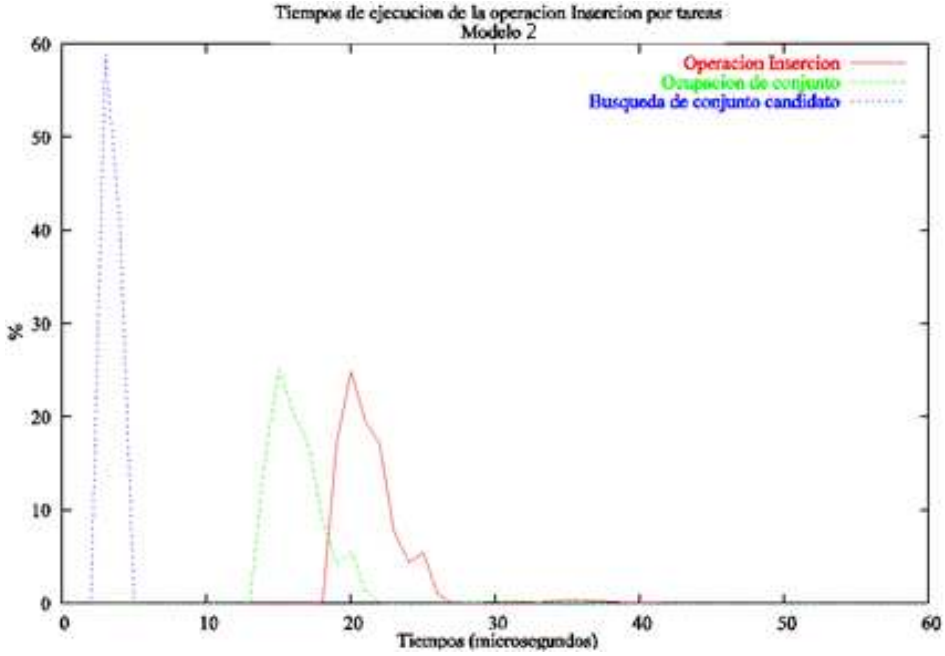


Figura 43: Tiempo de inserción

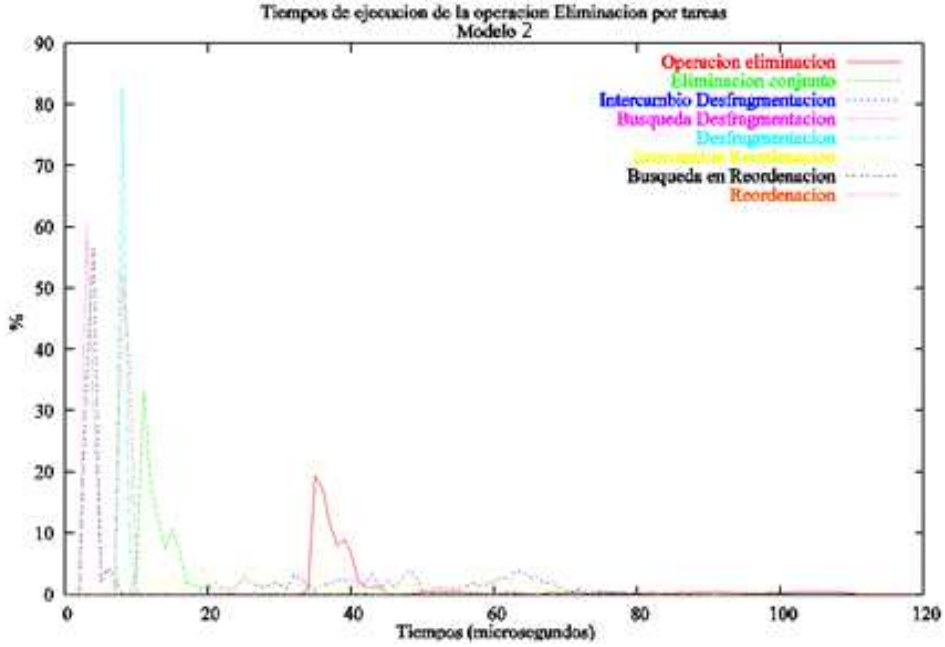


Figura 44: Tiempo de eliminación

5.3.4. Comparativa entre modelos

Una vez expuestos los 3 modelos, se procede a realizar un estudio comparativo entre los tiempos obtenidos por cada uno. Por ello, se expone primero una tabla con todos los tiempos que se han obtenido en cada uno de los modelos.

Viendo esta tabla, se puede observar como el Modelo 1 es el que mejor tiempo medio tiene. La principal razón de este resultado es el porcentaje de intercambios que se da

Operación / Tarea	Tiempo medio (microsegundos)
Operación Inserción	19.7402106
Ocupación de conjunto	14.9527087
Búsqueda Inserción	2.23590386
Actualización singulares	3.10793627
Operación Eliminación	44.7582443
Liberación de conjunto	12.0174165
Actualización singulares	2.78933425
Reordenación	10.1673254
Búsqueda	2.70758744
Intercambio	39.2435057
Desfragmentación	13.2055076
Búsqueda	2.45667072
Intercambio	49.2571936
Operación Inserción Fallida	4.73061304

Cuadro 5: Tiempos de inserción

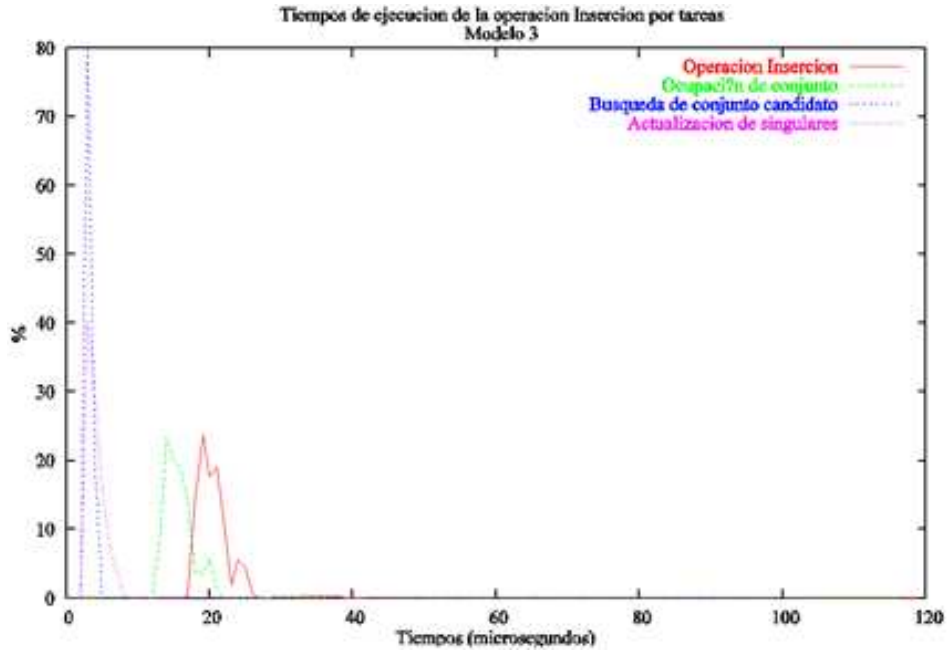


Figura 45: Tiempos de inserción

en el Modelo 1, siendo el intercambio la tarea que más tiempo necesita para llevarse a cabo. En concreto, el número medio de intercambios por operación en cada modelo se indica en la siguiente tabla.

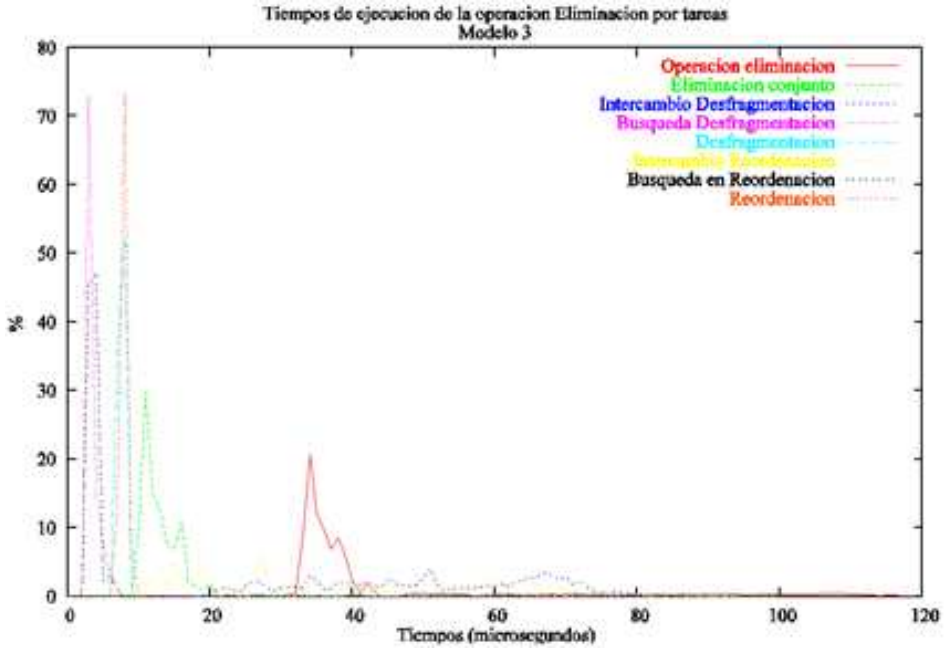


Figura 46: Tiempos de eliminación

Tiempos medios (microsegundos)			
Operación / Tarea	**Modelo**		
	1	2	3
Operación Inserción	29.51606979	20.4631379	19.7402106
Ocupación de conjunto	14.7126913	15.4860794	14.9527087
Búsqueda Inserción	2.29820253	2.448428	2.23590386
Actualización singulares	3.15501726	3.29179197	3.10793627
Desfragmentación	11.0030499	x	x
Búsqueda conjunto	2.36846504	x	x
Intercambio	48.9749696	x	x
Operación Eliminación	32.6688208	45.6902477	44.7582443
Liberación de conjunto	11.5595146	12.0599192	12.0174165
Actualización singulares	2.75405247	2.91499289	2.78933425
Reordenación	x	11.8999243	10.1673254
Búsqueda	x	2.89573179	2.70758744
Intercambio	x	40.6167184	39.2435057
Desfragmentación	11.0030499	11.9228966	13.2055076
Búsqueda	48.9749696	2.57360484	2.45667072
Intercambio	2.36846504	46.1881146	49.2571936
Operación Inserción Fallida	4.50739351	4.81503764	4.73061304
Tiempo medio de Operación	29.04465481	30.537596	29.979255

Modelo 1	Modelo 2	Modelo 3
0.0715	0.1207	0.1126

Cuadro 6: Número de intercambios por operación

6. Conclusiones

En este informe se han presentado los algoritmos correspondientes al modelo de gestión de la tabla de arbitraje de los puertos de salida de InfiniBand, presentado en la tesis doctoral Francisco J. Alfaro [1].

En un primer momento se realizó la correspondiente verificación de su adecuado funcionamiento y se ha desarrollado una forma de evaluación de sus prestaciones.

Se han definido tres modelos:

Modelo 1 En este modelo, tras producirse una inserción o una eliminación en la tabla de arbitraje se aplica el algoritmo de desfragmentación.

Modelo 2 Tras una eliminación se aplica el algoritmo de reordenación primero y el de desfragmentación después. Tras una inserción no es necesario realizar operaciones de mantenimiento de la tabla de arbitraje.

Modelo 3 Este caso es casi idéntico al Modelo 2, excepto que tras una eliminación se aplica primero el algoritmo de desfragmentación y después el de reordenación. Tampoco en este caso es necesario realizar operaciones de mantenimiento de la tabla de arbitraje, después de una inserción.

Como trabajo futuro, en los modelos 2 y 3, se podría modificar el método de búsqueda que hasta ahora se utiliza, haciendo que en vez de realizar la búsqueda desde el nivel del que se ha solicitado un conjunto hasta el nivel 0, se realice hasta que encuentre un conjunto singular, ya que al estar la tabla de arbitraje ordenada, por definición, dicho conjunto singular estará más a la izquierda que otros conjuntos singulares de mayor tamaño.

Además, en el modelo 1, sí se realizará una búsqueda tal y como se indica en la propuesta anterior, aún sin estar ordenada la tabla pero sí normalizada, tras ocupar el conjunto solicitado con ese tipo de búsqueda, la tabla seguiría normalizada. Por tanto, se podría plantear un modelo que consista en realizar únicamente la desfragmentación tras eliminar y dejar de llevarla a cabo tras la inserción.

Bibliografía

[1] F. J. Alfaro. *Una sencilla metodología para garantizar calidad de servicio en subredes InfiniBand*. Number 188 in Colección Tesis Doctorales. Servicio de publicaciones de la Universidad de Castilla-La Mancha, 2004. ISBN: 84-8427-320-2.

[2] S. Blake, D. Back, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. Internet Request for Comment RFC 2475, Internet Engineering Task Force, Dec. 1998.

[3] P. by ISSG Technology Communications. Infiniband architectural technology. Technical Report TC000702TB, Compaq Computer Corporation, July 2000.

[4] A. Forum. *ATM Forum traffic management specification. Version 4.0*, May 1995.

[5] N. Giroux and S. Ganti. *Quality of Service in ATM Networks*. Prentice Hall, 1999.

[6] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1. Release 1.0*, Oct. 2000.

[7] InfiniBand^{TM} Trade Association. 1999. `http://infinibandta.com`.

[8] J. Pelissier. Providing Quality of Service over Infiniband Architecture Fabrics. In *Proceedings of the 8th Symposium on Hot Interconnects*, Aug. 2000.

[9] G. Pfister. *High Performance Mass Storage and Parallel I/O*, chapter 42: An Introduction to the InfiniBand Architecture, pages 617–632. IEEE Press and Wiley Press, 2001.

[10] M. Schwartz and D. Beaumount. Quality of service requirements for audio-visual multimedia services. *ATM Forum*, ATM94-0640, July 1994.