A Formal Model to Manage the InfiniBand Arbitration Tables Providing QoS

Francisco J. Alfaro, *Member*, *IEEE Computer Society*, José L. Sánchez, Manuel Menduiña, and José Duato, *Member*, *IEEE Computer Society*

Abstract—The InfiniBand Architecture (IBA) is an industry-standard architecture for server I/O and interprocessor communication. IBA enables quality-of-service (QoS) support with certain mechanisms. These mechanisms are basically the service levels, the virtual lanes, and the table-based arbitration of those virtual lanes. In previous papers, we have examined these mechanisms and described how we can apply them to the requirements requested by the applications. We have also tested our proposals, showing that the applications achieve the level of QoS requested. In this paper, we present a formal model for the techniques previously proposed. According to this model, each application needs a sequence of entries in the IBA arbitration tables based on its requirements. These requirements are related to the mean bandwidth needed and the maximum latency tolerated by the application. Specifically, each request requires a number of entries with a maximum separation between any consecutive pair. In order to manage the requests, we propose certain algorithms and we prove some propositions and theorems, showing that our method achieves good behavior.

Index Terms—InfiniBand, QoS, scheduling, formal model.

1 Introduction

Many current applications have requirements such as guarantee of bandwidth, bounded delivery deadline, bounded interarrival delays, and so forth that not all current networks are able to provide. In that sense, the latest proposed standards incorporate some mechanisms in order to provide present and future applications with these quality-of-service (QoS) requirements [1], [2], [3].

The InfiniBand Architecture (IBA) [4] is a standard for high-speed I/O and interprocessor communication. It has been developed by the InfiniBand Trade Association (IBTA) [1], which is a group of more than 200 IT companies founded in August 1999 to develop IBA. After a very serious crisis, IBA is ramping up again, but its current use focuses almost exclusively on clusters for high-performance computing. However, IBA may expand its market in the future by being implemented in clusters for different application areas that may require QoS support.

We envision two different application areas where QoS may be very necessary. The first one is providing Internet services that require QoS guarantees (for example, video on demand) to a very large number of concurrent clients. Although it is true that Internet protocols play a critical role in providing such a QoS, it is also true that those Internet servers must be highly parallel (for example, a cluster) and will require internal QoS support when retrieving information from the disk subsystem and transmitting it through its system area network (for example, InfiniBand) from the disks to the server nodes.

The second scenario is providing bandwidth and latency guarantees to each partition when the interconnection network of a cluster is partitioned into several virtual networks. This partitioning is quickly becoming very important as the trend toward virtualization continues and an increasing number of companies are providing service to many customers by splitting a physical server into multiple virtual servers.

InfiniBand provides a series of mechanisms that are able to provide QoS to the applications. These mechanisms are mainly segregation of traffic according to categories (service levels (SLs)) among the different virtual lanes (VLs) and arbitration of the output port switches and network interfaces according to the information stored in an arbitration table. This arbitration table can be configured to provide traffic flows with QoS according to their requirements.

IBA distinguishes up to a maximum of 16 SLs, but it does not specify what characteristics the traffic of each SL should have. IBA ports have to support a minimum of 2 and a maximum of 16 VLs. The number of VLs used by a port is configured by the Subnet Manager. Since systems can be constructed with switches supporting a different number of VLs, packets are marked with an SL and a relation between the SL and VL is established at each node by means of the $SLtoVLMappingTable$.

When more than two VLs are implemented, the $VLArbitrationTable$ defines the priorities of the data lanes. Each $VLArbitrationTable$ has two tables: one for delivering packets from high-priority VLs and another one for low-priority VLs. The arbitration tables implement weighted round-robin arbitration [5] within each priority level. Up to

- *F.J. Alfaro, J.L. Sánchez, and M. Menduña are with the Departamento de Sistemas Informáticos, Escuela Politécnica Superior, Universidad de Castilla-La Mancha, 02071-Ablacete, Spain. Email: {falfaro, jsanchez, mmendu}@info-ab.uclm.es.*
- *J. Duato is with the Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, 46071-Valencia, Spain. E-mail: jduato@gap.upv.es.*

Manuscript received 27 July 2006; revised 19 Dec. 2006; accepted 23 Mar. 2007; published online 3 Apr. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0291-0706.
Digital Object Identifier no. 10.1109/TC.2007.1051.

64 table entries are cycled through, each one specifying a VL and a weight. The weight is the number of units of 64 bytes to be sent from that VL. This weight must be in the range of 0 to 255 and is always rounded up in order to transmit a whole packet. A $LimitOfHighPriority$ value specifies the maximum number of high-priority packets that can be sent before a low-priority packet is sent. More specifically, the VLs of the high-priority table can transmit $LimitOfHighPriority \times 4,096$ bytes before a packet from the low-priority table may be transmitted. If no high-priority packets are ready for transmission at a given time, low-priority packets can also be transmitted.

We have proposed a new methodology to use those IBA mechanisms providing applications with QoS guarantee [6], [7]. According to this methodology, each application needs a sequence of entries in the IBA arbitration tables based on its requirements. These requirements are related to the mean bandwidth needed and the maximum latency tolerated by the application. Specifically, each request requires a number of entries with a maximum separation between any consecutive pair.

Note that we do not consider the end-host overheads while providing QoS guarantees, which usually is very significant in high-speed cluster interconnects. However, the reduction of this software overhead is clearly out of the scope of this paper.

The results that we have obtained by simulation show that this proposal provides applications with QoS requirements that have previously been demanded [6], [7]. However, we think that this is not enough and a formal proof should be necessary. Although the performance evaluation was performed with a great variety of situations, from an engineering point of view, the correct process is to formally prove that the new methodology works correctly.

In this paper, we present a formal model for the technique previously proposed. In order to manage the requests, we propose certain algorithms and we prove some propositions and theorems, formally proving that our methodology achieves the correct behavior.

The structure of this paper is as follows: Section 2 presents related work that we have previously undertaken. In Section 3, a formal model to manage the InfiniBand arbitration tables is proposed. To that end, some definitions, initial hypotheses, and propositions are provided. Section 4 presents an algorithm to complete the table according to the model proposed in the previous section. In Section 5, some of the initial hypotheses are reviewed and two new algorithms are proposed: the disfragmentation algorithm and a further algorithm for table reordering. Section 6 deals with the global management of the table when there are, in a continuous way, new requests and releases. Finally, in Section 7, our conclusions are presented.

2 Related Work

In a previous work [8], we studied how we can provide bandwidth and/or latency guarantees to the applications. We proposed assigning all of the traffic with QoS requirements to the high-priority table, devoting the low-priority table to the traffic without QoS requirements.

In [8], we showed that a bandwidth request should be translated into a request of a certain weight in the arbitration tables. This computing is based on the bandwidth requested and the link bandwidth. However, in order to be able to achieve the latency requirement, a connection may need to have its assigned weight distributed among several entries in the arbitration table. These table entries must have a certain maximum distance between any consecutive pair in order to guarantee the maximum number of transmittable packets before one packet is transmitted from the VL associated to that connection. We say that a connection request is of *type*[1] $\boldsymbol{d}$ if it needs a maximum distance of $\boldsymbol{d}$ between any consecutive two entries of the sequence used by its VL in the arbitration tables. We have proposed categorizing the allowed distances in the following types: 2, 4, 8, 16, 32, and 64 (the request of type 64 is when the request does not have any deadline delay or is enough long). Obviously, for a table of 64 entries, there can be 64 possible distances and, so, any request will be reduced to the corresponding power of 2 immediately preceding.

Thus, a connection request of a certain mean bandwidth of B megabits per second and a bounded end-to-end delay of t ms will be treated as a request of a certain weight w and a maximum distance d between any consecutive two entries in the arbitration table. Therefore, the total number of entries used by the connection in the high-priority arbitration table will be the maximum between these two values: the entries needed by the bandwidth requirement $(\frac{w}{255})$ and the requirement of maximum distance between two consecutive entries in the table.

We have proposed that some connections requiring the same distance between two consecutive entries share a sequence of entries in the table, all of them obviously using the same VL. Each one of these entries of the sequence would have a weight based on the accumulated bandwidth for all of the connections sharing that sequence of entries. Thus, when a node sets a new request in the table, it must look for an available sequence of entries in the table to be used for that connection, that is, a sequence of entries with a given maximum distance between them. Specifically, three different situations are possible:

- We can use an already used sequence with the same maximum distance but only if it has available weight. For a sequence of entries with maximum distance d, we have $\frac{64}{d}$ entries in the table, each one with a maximum weight of 255. Hence, the total weight that can be accumulated in this sequence of entries is $\frac{64\times255}{d}$.
- There are one or some sequences in the high-priority arbitration table with the same maximum distance as that requested for a new connection, but none of them can be used. The reason is that the accumulated weight for its entries does not permit this new connection to be inserted since the maximum weight would be exceeded.

1. As type of connection and distance have the same value and meaning for a certain connection, in the following, we are going to use both terms interchangeably to refer to the same concept.

- There is no previously established sequence for this distance in the high-priority table.

The first case is very easy and we only need to recompute the weights of these entries. The other two cases require a more elaborate process to find a new sequence. This paper is focused on those two cases. We present an algorithm to find a new sequence of free entries able to locate a connection request in the table. This algorithm is part of a formal model to manage the IBA arbitration table. In the next sections, we will present a formal model to manage the IBA arbitration table and several algorithms in order to adapt this model for being used in a dynamic scenario when new requests and releases are made.

Note that, in some cases, our approach uses more entries in the table than absolutely necessary. In [6], we compare this proposal with another more exhaustive search on the basis of the number of entries improperly used and their complexity, taking into account their implementation and management. That study permits us to conclude that our proposal manages the table entries in a clever and simple way, without wasting, in a wide scope, more entries than other approaches.

3 Formal Model for the InfiniBand Arbitration Table

As has been mentioned before, we now propose a concrete algorithm to find a new sequence of free entries able to locate a connection request in the table. The treatment of the problem that we present basically consists of setting out an efficient algorithm able to select a sequence of free entries on the arbitration table. These entries must be selected with a maximum separation between any consecutive pair. To develop this algorithm, we first propose some hypotheses and definitions for establishing the correct frame to later present the algorithm and its associated theorems.

Although the algorithm and the treatment are focused on a specific frame such as InfiniBand, they can be generalized to any problem of finding a sequence of entries in a table with a certain separation between any consecutive pair. This general character can be achieved if we do not use the word *arbitration* when we are referring to the table and the word *connection* when we are talking about the origin of the entry requests. We therefore continue with a general frame, although we consider some specific characteristics of IBA: the number of table entries (64) and the value of the weight $(0 \ldots 0.255)$. All we need to know is that the requests are originated by the connections so that some requirements are guaranteed. Besides, the group of entries assigned to a request belongs to the arbitration table associated with the output ports and interfaces of the InfiniBand switches and hosts, respectively.

3.1 Definitions

For the later treatment that we shall be applying, we formally define the following concepts:

- *Table*: Round list of 64 entries.
- *Entry*: Each one of the 64 parts compounding a table.
- *Weight*: Numerical value of the entries in the table. This can vary between 0 and 255.
- *Status of an entry*: Situation of an entry of the table. The different situations can be free $(\text{weight} = 0)$ or occupied $(\text{weight} \neq 0)$.
- *Request*: A demand of a certain number of entries.
- *Distance*: Maximum separation between two consecutive entries in the table that are assigned to one request.
- *Type of request*: Each one of the different types into which the requests can be grouped. They are based on the requested distances and, so, on the requested number of entries.
- *Group or sequence of entries*: The set of entries of the table with a fixed distance between any consecutive pair. In order to characterize a sequence of entries, it will be enough to give the first entry and the distance between a consecutive pair.

3.2 Initial Hypothesis

In what follows, and when not indicated to the contrary, the following hypotheses will be considered:

1. There are no request eliminations, so the table is filled in when new requests are received and these requests are never removed. In other words, the entries could change from a free status to an occupied status, but it is not possible for an occupied entry to change to free. This hypothesis permits us to do a more simple and clear initial study, but, logically, it will be discarded later on.
2. It may be necessary to devote more than a group of entries to a set of requests of the same type.
3. The total weight associated with one request is distributed among the entries of the selected sequence so that the weight for the first entry of this sequence is always larger than or equal to the weight of the other entries of the sequence.
4. The distance d associated to one request will always be a power of 2 and it must be $1 \leq d \leq 64$. These are the different types of requests that we are going to consider. Therefore, in the following, *distance* and *type of request* will be equivalent terms. Thus, the maximum distance d requested will be $d = 2^i$, where $i = 0, 1, 2, \ldots, 6$.

3.3 The Model

Given a table T with N entries, we will distinguish each of them with an identifier t_i, $i = 0, 1, \cdots, N-1$. Hence, if N is 64, then we have 64 different identifiers, $t_0, t_1, \cdots, t_{62}, t_{63}$. According to the previous definitions (Section 3.1), every t_i has an associated weight w_i whose value can vary between 0 and 255. The entry t_i is said to be free if $w_i = 0$; otherwise, it is not free or occupied.

Consecutive entries of the table will not have identifiers with consecutive indexes. This does not mean a loss of generality and we will simplify the later process. The assignment of identifiers to the entries is based on the application of the bit-reversal function to the usual numbering, which would assign identifiers with

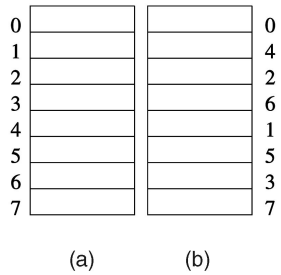


Fig. 1. Table with eight entries. (a) Using a correlative numbering of entries. (b) Numbering based on the bit-reversal function.

consecutive indexes to consecutive entries. Remember that the bit-reversal function applied to $m = b_{n-1}b_{n-2}\ldots b_1b_0$ is $\Re(m) = b_0b_1\ldots b_{n-2}b_{n-1}$. For a table with eight entries, Fig. 1a shows the correlative numbering, whereas Fig. 1b shows the new numeration.

We can see in Fig. 1b that there are only two sequences of entries to meet a request of maximum distance 2: {0, 2, 1, 3} and {4, 6, 5, 7}. Note that these sequences can be considered as sets of table entries. As there is no established order among the elements of a set, both previous sets are the same as {0, 1, 2, 3} and {4, 5, 6, 7}, respectively. As can be seen, the entries of a set have a consecutive enumeration. This is why this numeration is used. Something similar happens for other distances.

The following definition of entry set uses the new numeration. Each set contains the needed entries to meet the request of a certain distance.

Definition 1. *Given a table T for any request of type $d = 2^i$, with $0 \le i \le 6$, we define the sets $E_{i,j}$, where $j = k \times 2^{6-i}$ and $0 \le k < 2^i$:*

$$E_{i,j} = \{t_n \mid \quad j \le n < j + 2^{6-i}\}.$$

In order to simplify the notation, we also define a set $I_i = \{j \mid j = k \times 2^{6-i} \text{ with } 0 \le k < 2^i\}$. Given i, with $0 \le i < 6$, I_i is formed from the possible values of j. Thus, when this index j must be referenced, it will be indicated as $j \in I_i$. Note also that the maximum value of i $(i = 6)$ is based on the number of table entries $(6 = log_2\ 64)$.

For the eight-entry table[2] in Fig. 1b, the i value can vary from 0 to 3 $(log_2\ 8)$ and, so, the following sets are possible:

- With table entries at distance 1, $E_{0,0} = \{t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$.
- With table entries at distance 2, $E_{1,0} = \{t_0, t_1, t_2, t_3\}$ and $E_{1,4} = \{t_4, t_5, t_6, t_7\}$.
- With table entries at distance 4, $E_{2,0} = \{t_0, t_1\}$, $E_{2,2} = \{t_2, t_3\}$, $E_{2,4} = \{t_4, t_5\}$, and $E_{2,6} = \{t_6, t_7\}$.
- With table entries at distance 8, $E_{3,0} = \{t_0\}$, $E_{3,1} = \{t_1\}$, $E_{3,2} = \{t_2\}$, $E_{3,3} = \{t_3\}$, $E_{3,4} = \{t_4\}$, $E_{3,5} = \{t_5\}$, $E_{3,6} = \{t_6\}$, and $E_{3,7} = \{t_7\}$.

2. It is important to keep in mind that, for a table with eight entries, 2^{3-i} should be used in the previous definition instead of 2^{6-i}, which is for a 64-entry table.

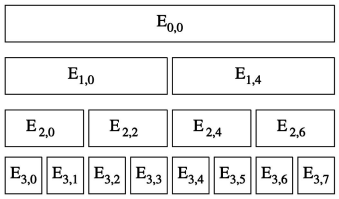


Fig. 2. Tree of sets for the table of eight entries.

Note that the sets for a distance are disjoint, each having enough entries to meet a request of that distance. With all of the sets, a binary tree can be established where a set of a certain level i is the union of two disjoint sets of level $i + 1$. The tree in Fig. 2 corresponds to the eight-entry table in Fig. 1b.

In general, for a table of 64 entries, each set $E_{i,j}$ contains the entries of the table having a distance $d = 2^i$ between any consecutive pair. In this case, the binary tree with all the sets is shown in Fig. 3. The root of the tree is the set $E_{0,0}$. This set is the union of the sets $E_{1,0}$ and $E_{1,32}$, which are the root sets of their two subtrees. These sets can also be divided into another two and so on.

For a request of distance $d = 2^i$, there are d different sets, $E_{i,j}$, corresponding to the d possible different sequences of entries in the table T that have this distance between any consecutive pair. For example, for the distance $d = 2$, there are two different sets, $E_{1,0} = \{t_0t_1t_2t_3\ldots t_{30}t_{31}\}$ and $E_{1,32} = \{t_{32}t_{33}t_{34}t_{35}\ldots t_{62}t_{63}\}$, corresponding to a set with the first part of the entries and another one with the second part of the entries. Note that this distribution would correspond to a division of even entries and odd entries in a correlative numeration. Therefore, each one of these sets has a separation of 2 between two consecutive entries and, so, each one of them has enough entries and the proper separation to meet a request of type 2. The corresponding $E_{i,j}$ sets can also be obtained for the other request types (Fig. 3).

In the following, we are going to present some definitions and propositions regarding this model. Perhaps some of them are quite trivial, but we have included them in order to be rigorous and to be able to use them later.

Definition 2. *A set $E_{i,j}$, with $0 \le i \le 6$ and $j \in I_i$, is said to be free if $w_k = 0 \ \ \forall\ t_k \in E_{i,j}$.*

This, together with the definition of $E_{i,j}$ (Definition 1), implies that a free set $E_{i,j}$ is needed for meeting a request[3] of type $d = 2^i$.

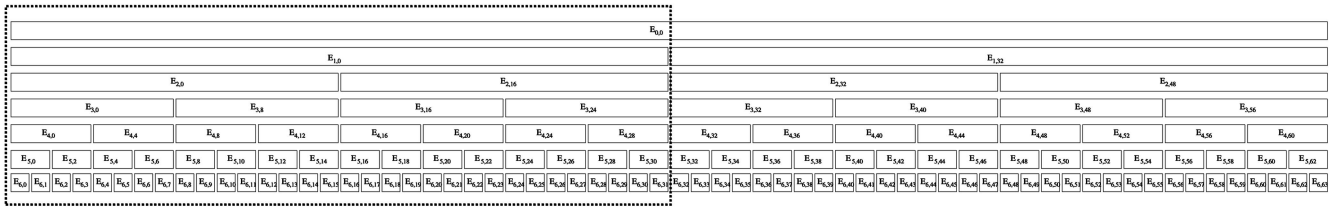
Taking the previous definitions as the starting point, the following propositions can be considered.

Proposition 1. *For any i and j, where $0 \le i < 6$ and $j \in I_i$*

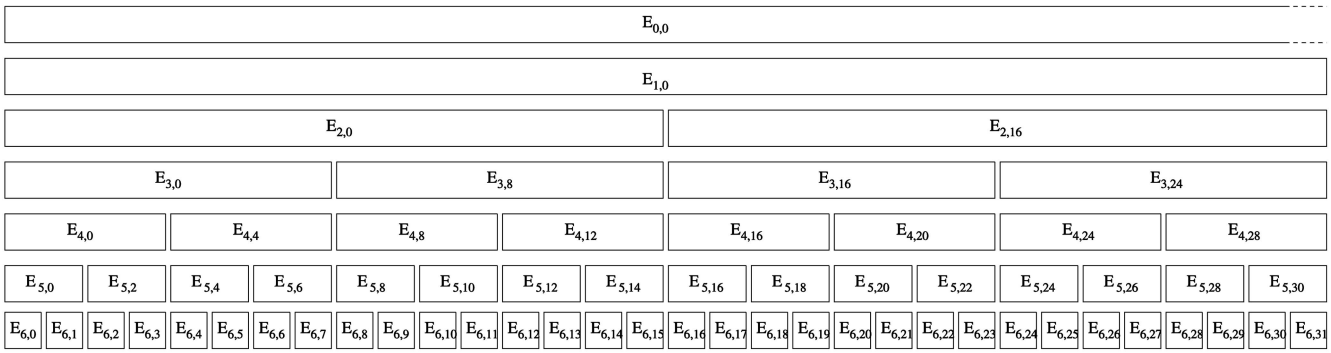
$$E_{i,j} = E_{i+1,j} \cup E_{i+1,j+2^{6-(i+1)}}.$$

In binary tree terms, this proposition could be enunciated as "*the union of both children is the father.*"

3. Remember that we are going to consider only the case where a new request needs a new sequence of free entries.



(a)



(b)

Fig. 3. Decomposition in a binary tree form of all of the possible entry sets of the table, where each level corresponds to a different type of request. (a) The whole tree. (b) A zoom of the marked half in (a).

Proof. According to Definition 1,

$$\begin{aligned} E_{i,j} &= \left\{t_n \mid \quad j \le n < j + 2^{6-i}\right\} \\ &= \left\{t_j, t_{j+1}, \ldots, t_{j+2^{6-(i+1)}}, \ldots, t_{j+2^{6-i}-1}\right\} \\ &= \left\{t_j, t_{j+1}, \ldots, t_{j+2^{6-(i+1)}-1}\right\} \cup \\ &\quad \left\{t_{j+2^{6-(i+1)}}, t_{j+2^{6-(i+1)}+1}, \ldots, t_{j+2^{6-i}-1}\right\} \\ &= \left\{t_n \mid \quad j \le n < j + 2^{6-(i+1)}\right\} \cup \\ &\quad \left\{t_m \mid \quad j + 2^{6-(i+1)} \le m < j + 2^{6-i}\right\} \\ &= E_{i+1,j} \cup E_{i+1,j+2^{6-(i+1)}}. \end{aligned}$$

□

As a consequence of this proposition, with the entries that allow a request of type $d = 2^i$ to be met, two requests of type $d' = 2^{i+1}$ can be located.

From Definition 1, given a level i, the sets $E_{i,j}$s, with $0 < i \le 6$ and $j \in I_i$, are those obtained with $j = k \times 2^{6-i}$ and $0 \le k < 2^i$. Therefore, the values of j are

$$0, 1 \times 2^{6-i}, 2 \times 2^{6-i}, 3 \times 2^{6-i}, \ldots, (2^i - 2) \times 2^{6-i}, \\ (2^i - 1) \times 2^{6-i}$$

or

$$0, 2 \times 2^{6-i}, 4 \times 2^{6-i}, \ldots, (2^i - 2) \times 2^{6-i} \quad \text{and} \\ 1 \times 2^{6-i}, 3 \times 2^{6-i}, \ldots, (2^i - 1) \times 2^{6-i},$$

that is,

$$\left.\begin{matrix} 2k \times 2^{6-i} \\ (2k+1) \times 2^{6-i} \end{matrix}\right\} \; 0 \le k < 2^{i-1}.$$

Given an index k, both expressions permit us to obtain the index that identifies two brother sets. The former is the brother set *located at the left* and the latter is the brother set *located at the right*.

The following function permits us to relate the identifiers of two brother sets. Specifically, when it is applied over the index that identifies either of them, the other index is returned:

$$Brother(p, i) = \begin{cases} p + 2^{6-i} & \text{if} \quad (p \bmod 2^{6-i+1}) = 0 \\ p - 2^{6-i} & \text{if} \quad (p \bmod 2^{6-i+1}) \neq 0. \end{cases}$$

As two brothers are always located at the same level, we simplify the above expression by omitting the level, resulting in $Brother(p)$. We can test this function by applying it to a couple of brothers. For a level i $(0 < i \le 6)$, the indexes of two brother sets are j and $j + 2^{6-i}$:

For $2k \times 2^{6-i}$

$(2k \times 2^{6-i}) \bmod 2^{6-i+1} = (k \times 2^{6-i+1}) \bmod 2^{6-i+1} = 0$

Therefore

$Brother(2k \times 2^{6-i}) = (2k \times 2^{6-i}) + 2^{6-i} = (2k+1) \times 2^{6-i}$

For $(2k+1) \times 2^{6-i}$

$((2k+1) \times 2^{6-i}) \bmod 2^{6-i+1} =$

$(k \times 2^{6-i+1} + 2^{6-i}) \bmod 2^{6-i+1} = 2^{6-i} \neq 0$

Therefore

$Brother((2k+1) \times 2^{6-i}) = ((2k+1) \times 2^{6-i}) - 2^{6-i} =$

$2k \times 2^{6-i}$.

In the same way, we can also obtain an expression relating the identifier of a set to one of its ancestors. Thus, for any set $E_{k,l}$, with $0 < k \leq 6$ and $l \in I_k$, l is related to the index of any of its ancestors at the level i, with $0 \leq i < 6$, according to the expression $(l \operatorname{div} 2^{6-i}) \times 2^{6-i}$. In order to simplify the later use of this expression, we are going to define the function $Ancestor(l,i) = (l \operatorname{div} 2^{6-i}) \times 2^{6-i}$ as the ancestor of l at level i.

A generalization of Proposition 1 emerges from being successively applied to each of the descendants of a set $E_{i,j}$. This generalization indicates that, with the entries of this set, which meets a request of type $d = 2^i$, two requests of type 2^{i+1} can be met, four of type 2^{i+2}, and so on. This is shown in the following proposition:

Proposition 2. *For any i and j, where $0 \leq i < 6$ and $j \in I_i$,*

$$E_{i,j} = \bigcup_{l=0}^{2^k-1} E_{i+k,j+l\times 2^{6-(i+k)}} \qquad k = 1, 2, \ldots, 6-i.$$

Proof. The proof is by induction on k. For $k = 1$, we have $E_{i,j} = E_{i+1,j} \cup E_{i+1,j+2^{6-(i+1)}}$, which follows from Proposition 1. Let us now suppose it is true for $k-1$ and we will prove that it is also true for k. As it is true for $k-1$, we have

$$E_{i,j} = \bigcup_{l=0}^{2^{(k-1)}-1} E_{i+(k-1),j+l\times 2^{6-(i+k-1)}}. \tag{1}$$

Applying Proposition 1 to each of those sets, we obtain

$$E_{i+(k-1),j+l\times 2^{6-(i+k-1)}} =$$
$$E_{i+k,j+l\times 2^{6-(i+k-1)}} \cup E_{i+k,j+l\times 2^{6-(i+k-1)}+2^{6-(i+k)}} =$$
$$E_{i+k,j+l\times 2\times 2^{6-(i+k)}} \cup E_{i+k,j+(l\times 2+1)\times 2^{6-(i+k)}}.$$

Replacing in (1), we obtain

$$E_{i,j} = \bigcup_{l=0}^{2^{(k-1)}-1} E_{i+k,j+l\times 2\times 2^{6-(i+k)}} \cup E_{i+k,j+(l\times 2+1)\times 2^{6-(i+k)}}.$$

For the range of values of l, $\left[0\,..\,2^{(k-1)}-1\right]$, the expressions $l \times 2$ and $(l \times 2 + 1)$ include all of the values in the interval $\left[0\,..\,2^k - 1\right]$ because

$$l \times 2: \quad 0, 2, \ldots, 2(2^{(k-1)}-1) = 0, 2, \ldots, 2^k - 2$$
$$l \times 2 + 1: \quad 1, 3, \ldots, 2(2^{(k-1)}-1) + 1 = 1, 3, \ldots, 2^k - 1.$$

Therefore,

$$E_{i,j} =$$
$$\bigcup_{l=0}^{2^{(k-1)}-1} E_{i+k,j+l\times 2\times 2^{6-(i+k)}} \cup E_{i+k,j+(l\times 2+1)\times 2^{6-(i+k)}} =$$
$$\bigcup_{l=0}^{2^k-1} E_{i+k,j+l\times 2^{6-(i+k)}},$$

as we wanted to prove. □

From this last proposition, it is easy to test that if $E_{i,j} = \bigcup E_{i+k,m}$, then $m \geq j$. This means that the index of any descendant of a set at any level is always greater than or equal to its index.

Proposition 3. *The available sequences of entries for a distance $d = 2^i$ are independent of each other. In a more formal way,*

$$E_{i,j} \cap E_{i,l} = \phi \quad \forall\, l \neq j, \quad 0 < i \leq 6 \quad \text{and} \quad j, l \in I_i.$$

In binary tree terms, this proposition could be reworded as "*the nodes of the tree located at the same level are disjoint*."

Proof. The entries corresponding to these two sets are

$$E_{i,j} = \left\{t_n \mid \quad j \leq n < j + 2^{6-i}\right\} \quad \begin{cases} j = k \times 2^{6-i} \\ l = t \times 2^{6-i} \\ 0 \leq k, t < 2^i. \end{cases}$$
$$E_{i,l} = \left\{t_m \mid \quad l \leq m < l + 2^{6-i}\right\}$$

However,

$$\left.\begin{array}{l} j \leq n < j + 2^{6-i} \\ l \leq m < l + 2^{6-i} \end{array}\right\} \Longrightarrow \begin{cases} k \times 2^{6-i} \leq n < (k+1) \times 2^{6-i} \\ t \times 2^{6-i} \leq m < (t+1) \times 2^{6-i}, \end{cases}$$

As $j \neq l$, then $k \neq t$. So, $n \neq m$ and, therefore, $E_{i,j} \cap E_{i,l} = \phi$ □

Proposition 4. *Each available set of entries $E_{i,j}$ for a distance $d = 2^i$ is disjoint from all of the other available sets of entries for a distance $d' = 2^{i+1}$ that fail to satisfy Proposition 1 about $E_{i,j}$. More formally,*

$$E_{i,j} \cap E_{i+1,l} = \phi \qquad \forall l \quad \mid \quad (l \neq j) \text{ and } (l \neq j + 2^{6-(i+1)})$$
$$0 < i < 6 \qquad j \in I_i \qquad l \in I_{(i+1)}.$$

In binary tree terms, this proposition can be reworded as "*a node is disjoint with any deeper node except with its descendants.*"

Proof. According to Proposition 3, we know that $E_{i,j} \cap E_{i,l} = \phi \ \forall j \neq l$. On the other hand, from Proposition 1, $E_{i,l} = E_{i+1,l} \cup E_{i+1,l+2^{6-(i+1)}}$. Combining both expressions, we obtain

$$E_{i,j} \cap E_{i,l} = \phi$$
$$\Longleftrightarrow E_{i,j} \cap (E_{i+1,l} \cup E_{i+1,l+2^{6-(i+1)}}) = \phi$$
$$\Longleftrightarrow (E_{i,j} \cap E_{i+1,l}) \cup (E_{i,j} \cap E_{i+1,l+2^{6-(i+1)}}) = \phi$$
$$\Longrightarrow \begin{cases} (E_{i,j} \cap E_{i+1,l}) = \phi \\ (E_{i,j} \cap E_{i+1,l+2^{6-(i+1)}}) = \phi, \end{cases}$$

as we wished to prove. □

Proposition 5. *Let us consider a free set $E_{i,l}$ and another nonfree set $E_{i,m}$, with $l \neq m$, whose occupied entries are used to completely locate requests of type $d \geq 2^i$. Then, all those requests can be completely met with entries of $E_{i,l}$, thus leaving free the set $E_{i,m}$, where $0 < i \leq 6$, and $l, m \in I_i$.*

Proof. If all of the entries of the set $E_{i,m}$ are used to completely locate a request of type $d = 2^i$, then this request could also be met with the entries of any free set $E_{i,j}$ ($0 \leq j < 2^i$ and $j \neq m$) and, specifically, with the entries of the set $E_{i,l}$.

If all or part of the entries of the set $E_{i,m}$ are being used to meet completely requests of types $d > 2^i$, then it follows that we use some of the sets $E_{i+k,m+n\times 2^{6-(i+k)}}$ ($n = 0, \ldots, 2^k - 1$ and $k = 1, \ldots, 6-i$) so that $E_{i,m} = \bigcup_{n=0}^{2^k-1} E_{i+k,m+n\times 2^{6-(i+k)}}$. To meet those requests, other free sets $E_{i+k,j+n\times 2^{6-(i+k)}}$, with $0 \leq j < 2^{i+k}$

and $j \neq m$, can also be used. Specifically, the sets $E_{i+k,l+n\times 2^{6-(i+k)}}$ ($n = 0, \ldots, 2^k - 1$ and $k = 1, \ldots, 6 - i$) can be used because they are free sets since $E_{i,l}$ is free, and Proposition 2 for this set is satisfied. □

Definition 3. *A set $E_{i,j}$ is said to be* **singular** *if this set is free, but the set $E_{i,n}$ is not free, with $0 < i \leq 6$, $n = Brother(j)$, and j, $n \in I_i$. Therefore, a set $E_{i,j}$ is singular if it is free, but its brother is not free. As a consequence and using the binary tree terminology, none of the ancestors of a singular set is free.*

Definition 4. *A table is said to be normalized if $\forall i$, with $0 \leq i \leq 6$, at most there is a singular $E_{i,j}$ with $j \in I_i$. According to this definition, an empty table and a full table are normalized because there is no level with a singular set.*

Among the previously defined sets, a partial order relation can be defined based on their position in the binary tree.

Definition 5. *Let $\mathcal{E}$ be the set made up of all the sets $E_{i,j}$, with $0 < i \leq 6$ and $j \in I_i$. In $\mathcal{E} \times \mathcal{E}$, the following binary relation is defined:*

$$E_{k,l} \lhd E_{i,j} \Longleftrightarrow \begin{cases} i \leq k \\ l < j. \end{cases}$$

$E_{k,l} \lhd E_{i,j}$ is read as: $E_{k,l}$ is placed more to the left than the set $E_{i,j}$ at the same level or at a deeper level in the binary tree, as shown in Fig. 3.

Because none of the sets $E_{i,j} \in \mathcal{E}$ are comparable (for example, $E_{3,8} \not\lhd E_{4,12}$, and $E_{4,12} \not\lhd E_{3,8}$), this is a partial order relation and, so, $\mathcal{E}$ is a partially ordered set.

We will now show new propositions that are based on Definition 5.

Proposition 6. *For $E_{k,l}$ and $E_{k,m}$, where $0 < k \leq 6$ and $l, m \in I_k$ such that $E_{k,l} \lhd E_{k,m}$, if $p \neq q$, where $p = Ancestor(l, i)$, $q = Ancestor(m, i)$, and $0 < i < k$, then $E_{i,p} \lhd E_{i,q}$.*

This means that, if two sets are related, then their respective ancestors up to the previous level of the first common ancestor are also related to each other.

Proof. In order for $E_{i,p} \lhd E_{i,q}$, both conditions of Definition 5 must be true, which are $i \leq i$ and $p < q$. As the first one is trivial, we shall study the second condition. Because $E_{k,l} \lhd E_{k,m}$, then $l < m$ and, so, $(l \operatorname{div} 2^{6-i}) \leq (m \operatorname{div} 2^{6-i})$. Moreover, $(l \operatorname{div} 2^{6-i}) \times 2^{6-i} \leq (m \operatorname{div} 2^{6-i}) \times 2^{6-i}$ and, so, $p \leq q$.

From the definition, $p \neq q$ and $p < q$, which means that $E_{i,p} \lhd E_{i,q}$. □

Proposition 7. *For $E_{k,l}$ and $E_{k,m}$, where $0 < k \leq 6$ and $l, m \in I_k$ such that $E_{k,l} \lhd E_{k,m}$, if $i < k$, and $n \neq j$, where $n = Ancestor(l, i)$ and $j = Ancestor(m, i)$, then $E_{k,l} \lhd E_{i,j}$.*

This means that, if two sets are related, then the ancestors of the second one at any level, up to the level of the first common ancestor, are also related to the first set.

Proof. Because $E_{k,l} \lhd E_{k,m}$, then $l < m$. Besides, because $i < k$ and $n \neq j$, from Proposition 6, $E_{i,n} \lhd E_{i,j}$. Moreover, because $n \neq j$, from Proposition 3, we have $E_{i,n} \cap E_{i,j} = \phi$. However, because $E_{k,l} \subset E_{i,n}$, it is also true that $E_{k,l} \cap E_{i,j} = \phi$.

Let us suppose that $E_{k,l} \not\lhd E_{i,j}$ and, so, $l \geq j$. We also have $l < m$ and, so, $j \leq l < m$. Because $E_{k,m} \subset E_{i,j}$ and, from Proposition 2, which relates the indexes of a set and its descendants, we would have $E_{k,l} \subset E_{i,j}$. However, this is a contradiction because $E_{k,l} \cap E_{i,j} = \phi$. Therefore, it must be true that $E_{k,l} \lhd E_{i,j}$. □

Proposition 8. *For $E_{k,l}$ and $E_{i,j}$, where $0 < i < k \leq 6$, $l \in I_k$, and $j \in I_i$ such that $E_{k,l} \lhd E_{i,j}$, then $E_{t,m} \lhd E_{i,j}$, where $i \leq t < k$, and $m = Ancestor(l, t)$.*

This means that, if two sets from levels k and i, where $i < k$, are related, then all the ancestors up to the level i of the set of the level k are also related to the set of the level i.

Proof. For $E_{t,m} \lhd E_{i,j}$, according to Definition 5, $i \leq t$ and $m < j$ must be true. The first condition is true according to the initial hypothesis. Let us then look at the second condition. Because $E_{k,l} \lhd E_{i,j}$, $l < j$, and, hence,

$$m = (l \operatorname{div} 2^{6-t}) \times 2^{6-t} \leq (j \operatorname{div} 2^{6-t}) \times 2^{6-t}. \quad (2)$$

Because $j \in I_i$, $j = p \times 2^{6-i}$, where $0 \leq p < 2^i$. Because $i \leq t$, then $t = i + x$, where $x \geq 0$. This way

$$j \bmod 2^{6-t} = (p \times 2^{6-i}) \bmod 2^{6-t} =$$
$$(p \times 2^{6-t+x}) \bmod 2^{6-t} = (p \times 2^x \times 2^{6-t}) \bmod 2^{6-t} = 0.$$

Because $(j \bmod 2^{6-t}) = 0$, $(j \operatorname{div} 2^{6-t}) \times 2^{6-t} = j$. Thus, from (2), it is true that $m \leq j$.

Moreover, because $m = Ancestor(l, t)$ and from Proposition 2, which relates the indexes of a set and its descendants, $m \leq l$. Therefore, we have $m \leq j$, and $m \leq l$. Also, $l < j$. If $m = j$, then we would have $l < j = m$. However, this is a contradiction. Therefore, $m \neq j$ and, so, it must be true that $m < j$.

Thus, $i \leq t$ and $m < j$ and, so, it is true that $E_{t,m} \lhd E_{i,j}$, where $i \leq t < k$. □

Definition 6. *A table is ordered if $\forall E_{i,j}$ and $\forall E_{k,l}$, both singular sets, where $0 < i < k \leq 6$, $j \in I_i$, and $l \in I_k$, then $E_{k,l} \lhd E_{i,j}$.*

So far, we have seen the model that we are using, some related definitions, and some derived propositions. In the following section, we will propose the algorithm to search for a new sequence of entries in the table that are situated with a certain distance between any consecutive pair. We will first provide the algorithm and, later, some derivable theorems.

4 Algorithm for Filling in the Table

When there is a new request of maximum distance $d = 2^i$, we must find a group of $\frac{64}{d}$ entries in the table such that two consecutive entries are separated at a maximum distance of d. Note that this is the same as finding a free set $E_{i,j}$. In order to find the specific set, we will apply an algorithm. We will first make an informal description of this algorithm in order to later formally propose it and validate it with some theorems and their corresponding proofs.

For a request of distance $d = 2^i$, the algorithm examines, in a certain order, all of the possible sets $E_{i,j}$ for this type of

request. The first one of these sets having all of its entries free is selected.

The order in which the sets are examined has as an objective to maximize the distance between two free consecutive entries that would remain in the table after carrying out the selection. This way, the table remains in the optimum condition to be able to later meet the most restrictive possible request. The order in which the algorithm examines the free sets results from testing the $E_{i,j}$ sets from left to right according to the arrangement in Fig. 3. **For a new request of maximum distance $d = 2^i$, the algorithm selects the first free set $E_{i,j}$ of the sequence** $E_{i,0}, E_{i,1\times 2^{6-i}}, E_{i,2\times 2^{6-i}}, E_{i,3\times 2^{6-i}}, \ldots, E_{i,(2^i-1)\times 2^{6-i}}$.

Note that the apparent simplicity of this algorithm (to sweep the $E_{i,j}$ sets from left to right) has been achieved due to the enumeration of the entries of the table by using the bit-reversal function instead of using a correlative enumeration.

This algorithm has some characteristics that make it very efficient for filling in the table with a series of requests with maximum distance requirements. We will now show these characteristics by proving some theorems.

Theorem 1. *If there is a group of free entries in the table separated between any consecutive pair with the requested distance $d = 2^i$, with $0 \le i \le 6$, then the algorithm finds it.*

Proof. If there is a free group of entries in the table with a distance $d = 2^i$ separating any consecutive pair, then there is a free set $E_{i,k}$, with $k \in I_i$. Since the algorithm, by definition, inspects all of the sets $E_{i,j}$ consecutively until a free one is found, it is guaranteed that this set $E_{i,k}$ will finally be found. □

Theorem 2. *After applying the filling-in algorithm, the table is normalized. This means that the filling-in algorithm leaves, at any level of the tree in Fig. 3, at most a singular set.*

Proof. Let us suppose this is false. We then have $\exists i$, where $1 < i \le 6$, with this level having more than one singular set. Let us suppose there are two sets, specifically, $E_{i,k}$ and $E_{i,l}$, where $k, l \in I_i$ and $k < l$. Because $E_{i,l}$ is singular, $E_{i,n}$ is not free, where $n = Brother(l)$. Because $k < l$ and $E_{i,k}$ and $E_{i,l}$ are singular sets, $Ancestor(k, i-1) \neq Ancestor(l, i-1)$ and, so, it is also true that $k < n$. This means that the set $E_{i,k}$ is placed to the left of the set $E_{i,n}$.

However, this situation cannot arise because the filling-in algorithm would have selected the set $E_{i,k}$ or any of its sons in the tree (according to Proposition 2) before the set $E_{i,n}$ or any of its sons in the tree (according to Proposition 2). This is because, with the first ones, we could have met all of the requests that were being met with the last ones.

Whatever situation that we consider with more than two sets, if the initial hypothesis fails, then this includes the case of two sets. As a consequence, none of them will be possible. So, $\forall i$, with $0 < i \le 6$, if there is more than one free set $E_{i,j}$, then at most one of them is a singular set and, so, the table is normalized. □

Theorem 3. *If, after applying the filling-in algorithm several times, there are still n free entries in the table, then it is possible to meet the most restrictive possible request, which is the one of type $d = 2^i$, with $\frac{64}{d} \le n < \frac{64}{\frac{d}{2}}$. Hence, this means that one free set $E_{i,m}$ exists, where $m \in I_i$.*

Proof. We know that there are n free entries in the table. If $\frac{64}{d}$ entries of them belong to the same set $E_{i,m}$, then the proof is trivial.

Now, let us suppose that there is no free set $E_{i,j}$. Therefore, the n free entries are distributed among a series of free sets $E_{k,l}$, with $i < k \le 6$ and $l \in I_k$. According to Theorem 2, after applying the filling-in algorithm several times, the table is normalized. This means that, at level $i+1$, there is at most a singular set $E_{i+1,l}$, but this is true $\forall k$, where $i < k \le 6$. However, each one of these sets $E_{k,l}$ would have $\frac{64}{2^k} = 2^{6-k}$ entries and, so, the total number of free entries that those free sets have would be

$$\sum_{k=i+1}^{6} 2^{6-k} = \sum_{k=0}^{6-i-1} 2^k = 2^0 + 2^1 + 2^2 + \ldots + 2^{6-i-1}.$$

This is the addition of a geometric progression with difference 2 with the result $2^{6-i} - 1$. As a consequence,

$$\sum_{k=i+1}^{6} 2^{6-k} = \sum_{k=0}^{6-i-1} 2^k = 2^{6-i} - 1 < 2^{6-i} = \frac{64}{2^i} = \frac{64}{d},$$

which is a contradiction of our initial assumption that there are $n \ge \frac{64}{d}$ free entries. Therefore, one free set $E_{i,k}$ must exist.

Thus, if, after applying the filling-in algorithm several times, there are still n free entries, then $\frac{64}{2^i}$ entries ($\frac{64}{2^i} \le n < \frac{64}{\frac{d}{2}}$) belong to the same set $E_{i,m}$ and, so, it is possible to meet the most restrictive possible request, which is of type $d = 2^i$. □

Theorem 4. *Let there be a table with n free entries. Then, the filling-in algorithm is able to locate any set of requests that does not require more than n entries.*

Proof. Let $d_1, d_2, \ldots, d_m$ be the requests to locate in the table, which has n free entries. These requests are performed in the order indicated by the sequence and they satisfy

$$\sum_{i=1}^{m} \frac{64}{d_i} \le n.$$

Let n_i be the number of free entries in the table when the request d_i is made. Of course, $n_1 = n$. Let us analyze what happens when each of the requests d_i, with $1 \le i \le m$, is performed. When a request d_i is made, it may or may be the most restrictive one. Thus, we have two possible cases:

- *The request d_i is the most restrictive*. In this case, from Theorem 3, it can be met. After meeting it, there are still $n_{i+1} = n_i - \frac{64}{d_i}$ free entries.
- *The request d_i is not the most restrictive*. In this case, due to Theorem 3, with the n_i free entries, the most restrictive possible request (d_k so that $\frac{64}{d_k} \le n_i < \frac{64}{\frac{d_k}{2}}$) can be met. As d_i is not the most restrictive possible request, $d_i > d_k$ and, from Proposition 2, with the free entries being able to locate a request of type d_k, $\frac{d_i}{d_k}$ requests of type d_i

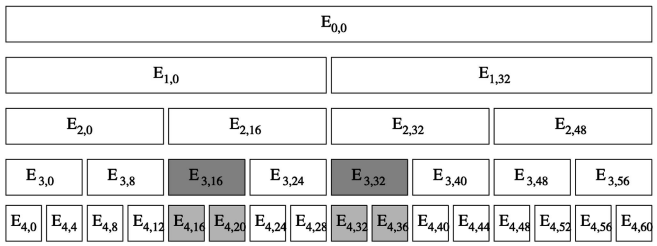


Fig. 4. Situation of the arbitration table after releasing the sets $E_{3,16}$ and $E_{3,32}$, having started with a filled table.

could be met, where $\frac{d_i}{d_k} \geq 2$. Hence, the free set required to meet the request d_i exists.

Summing up, if there are enough free entries in the table to locate a set of requests, then the filling-in algorithm is able to meet them. □

Theorem 5. *After applying the filling-in algorithm, the table stays ordered.*

Proof. According to Theorem 2, the filling-in algorithm always leaves the table normalized. Therefore, $E_{i,j}$ and $E_{k,l}$ are the only singular sets of the levels i and k, respectively, $\forall i$ and $\forall k$ so that $0 < i < k \leq 6$, $j \in I_i$, and $l \in I_k$.

Because $E_{k,l}$ is a singular set and due to the working of the filling-in algorithm, then $E_{k,m}$ is not free $\forall m$ such that $m < l$. According to Proposition 2, these sets $E_{k,m}$ are included in their ancestor sets at level i, $E_{k,m} \subset E_{i,n}$, where $n = Ancestor(m, i)$, which means that the sets $E_{i,n}$ are not free.

Moreover, because $E_{k,l}$ is singular after the filling-in algorithm has been applied, its ancestor at level i is not free. Thus, all of the ancestor sets at level i of the sets from level k placed to the left of the set $E_{k,l}$ and also $E_{k,l}$'s ancestor at level i are not free. Therefore, the free sets at level i must be placed to the right of l's ancestor at level i: $Ancestor(l, i) < j$. Because $E_{k,l} \subset E_{i,Ancestor(l,i)}$ and from Proposition 3, $E_{k,l} \not\subset E_{i,j}$, then $l < j$. This, together with $i < k$, implies that $E_{k,l} \triangleleft E_{i,j}$. As this is true $\forall i$ and $\forall k$ such that $0 < i < k \leq 6$, the table stays ordered. □

This last theorem will be used in the next section to prove that the situation in the table after releases is similar to the situation that would be achieved if we made the requests that stay in the table after those releases. This way, the application of Theorem 4 can guarantee that, if there are enough free entries to meet a sequence of requests, then these can always be met in the table. This is because the free entries are located in the best way possible to meet any later combination of maximum-distance requests.

5 Insertions and Eliminations in the Table

The algorithm presented in Section 4 has been set out based on certain initial hypotheses. The first of these requires that there are only request insertions in the table. Obviously, this is not a real situation and, so, the releasing situations must also be considered. This way, in the general behavior of the table, we could have insertions of new requests and releases of previously established requests.

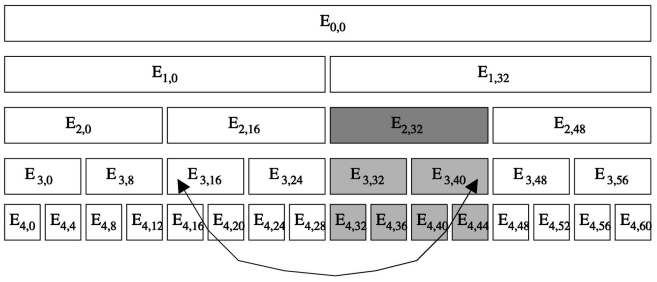


Fig. 5. Situation of the arbitration table in Example 1 after interchanging the sets $E_{3,16}$ and $E_{3,40}$.

Now, we eliminate the first of the initial hypotheses. Therefore, the elimination of requests is now possible. As a consequence, the entries used for the eliminated requests will be released. Considering the filling-in algorithm, as discussed in Section 4, in relation to this new scenario, we can achieve situations where there are enough available entries to meet a request, but it is not possible to meet it because these entries are not correctly separated. One of these situations is shown in the following example:

Example 1. We have the table filled[4] and two requests of type $d = 8$ are eliminated. These requests were made using the entries of the sets $E_{3,16}$ and $E_{3,32}$. This means that, now, the table has 16 free entries, and, so, a request of type $d' = 4$ could be met. However, there is no free set $E_{2,j}$ (see Fig. 4).[5]

Note that, in the previous example, the table is no longer normalized when the release is made. In order to solve that problem, the 16 free entries must be situated with a distance of 4 between any consecutive pair. Therefore, there should exist a free set $E_{2,j}$. It follows that the table must be normalized again. This can be achieved as indicated in Example 2. In this example, we follow a process based on leaving free entries that are occupied in order to obtain a set of free entries able to meet the most restrictive possible request.

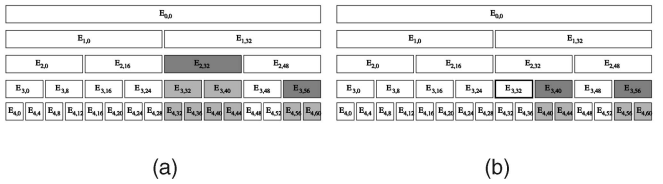
Example 2. A solution for the situation shown in Example 1 would consist of, for example, releasing the set $E_{3,40}$ and, in this way, together with the free set $E_{3,32}$, obtain $E_{2,32}$ free (by Proposition 1). Therefore, the request of type $d' = 4$ could be met.

In order to free the set $E_{3,40}$, the entries of the free set $E_{3,16}$ must be used to meet the requests that use the entries of the set $E_{3,40}$. This is possible according to Proposition 5. The final situation after this interchange is shown in Fig. 5. Note that, now, the table is normalized again.

This process, which we call *disfragmentation*, has as an objective of obtaining the greatest free set possible with the available free entries. Although the need for applying this disfragmentation process emerges as a consequence of considering request release, it is possible for it to be applied

4. We have selected this starting point in order to make the development of this example easier to understand. However, the situation shown in this example and in the following can be achieved starting from diverse table status.

5. The figure corresponds to a part of the 64-entry tree in Fig. 3.



(a) (b)

Fig. 6. Situations of the arbitration table in Example 3.

after locating the requests. This situation is shown in the following example:

Example 3. This example starts from the situation obtained in Example 2, with the entries of the set $E_{2,32}$ free and the others occupied. In this situation, the entries of the set $E_{3,56}$ are released. This situation is shown in Fig. 6a. After that, there is a new request of type $d = 8$ that must be located because there are enough free entries having the correct separation between them. When the filling-in algorithm is applied, the set $E_{3,32}$ is selected (the first free set at level 3). This creates a situation similar to that previously shown in Example 1, which can now be seen in Fig. 6b.

As a consequence, a new disfragmentation process is needed because enough entries exist to meet a request of type $d' = 4$, but they do not belong to the same set. In this case, the solution consists in freeing the set $E_{2,48}$. To that end, the entries of the set $E_{3,40}$ would meet the requests that are now being met by the entries of the set $E_{3,48}$.

Note that, in Example 3, when the set $E_{3,56}$ is released, the table is no longer ordered. When the filling-in algorithm is applied later over a nonordered table, a normalized table is no longer obtained. This is a direct consequence of the filling-in algorithm that sweeps the sets for a specific distance by selecting the first one that is free. The solution that has been applied in Example 3 consists of applying disfragmentation after a new insertion.

Another alternative would consist of ordering the singular sets generated after a release such that the table is ordered again. This way, the problem would be solved if in the situation shown in Fig. 6a, the singular set from level 3 (now $E_{3,56}$) would be to the left of the singular set from level 2 (now $E_{2,32}$). This way, the request of type $d = 8$ can be met by using the singular set from level 3, without applying disfragmentation and leaving a free set able to meet a request of type $d' = 4$. In general, the alternative solution consists of bringing about that the smaller sized free sets, whose entries will have greater separation, are situated to the left of the greater sized free sets. Obviously, this means that the table is ordered.

In order to detect this kind of situation, we must study the table, checking if there is any free set of a certain size *to the left* of another set that has a smaller size. To solve this situation, we must apply Proposition 5, but at the level of the greatest free set. Thus, the interchange must be performed between the greatest set and another same level set that includes the smallest free set. This situation is explained in Example 4.

Example 4. We will use the final situation in Example 2 as a starting point (Fig. 5). Then, the set $E_{3,56}$ is released, and the resulting situation is shown in Fig. 6a. In this situation, we study all of the tree levels until we find the situation to deal with. In order to solve this situation, we must apply Proposition 5, interchanging the sets $E_{2,32}$ and $E_{2,48}$. The resulting situation has as free sets $E_{3,32}$ and $E_{2,48}$, as can be seen in Fig. 7. If there is now a request of type $d = 8$, then it will be met with the entries of the set $E_{3,32}$, leaving the free set $E_{2,48}$ in order to meet a later request of type $d' = 4$.

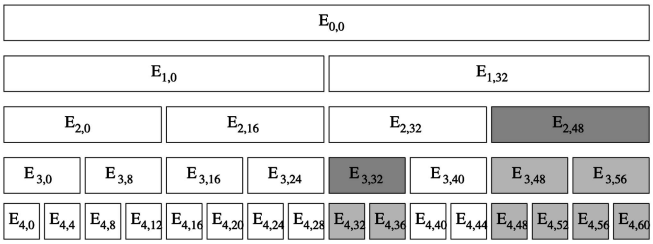


Fig. 7. The situation of the arbitration table in Example 4 after applying a reorder process.

Therefore, combining the filling-in algorithm, the disfragmentation algorithm, and the reordering algorithm, we can manage the general treatment of the table. We have shown two alternatives. In the first case, we maintain the table ordered and normalized, whereas, in the second case, the table only needs to be normalized and it is not necessary for the table to be ordered.

Summing up, we have just outlined two methods to treat those situations that have arisen as a result of having eliminated the restriction imposed by the first initial hypothesis in Section 3.2. These methods basically consist of the following:

1. We must study the resulting status of the table both after meeting a new request (using free entries) and after releasing a request (releasing occupied entries). In both cases, we must study if the table has enough free entries to meet a request, but this is not possible because there is no free set correctly sized. If this situation is found, then the disfragmentation algorithm must be applied to solve it. In other words, if, after an insertion or a release, the table is no longer normalized, then the disfragmentation algorithm must be applied to normalize it.
2. We must study the resulting status of the table only when a request is released. In this case, we must first check if there are free sets of a greater size than other sets situated to the left and, in this case, we must reorder. We must also check if, since we have enough free entries to meet a certain request, the latter cannot be met because there is no correctly sized free set. In this case, the disfragmentation algorithm must be applied. The order of these checks is irrelevant.

   Summing up, this second method should check after a release if the table is no longer normalized and/or nonordered. In this case, the disfragmentation algorithm and/or the reordering algorithm must be executed to leave the table once more normalized and ordered.

```
1: Procedure Disfragmentation ( set E_{i,k} )
2: while i > 1 do
3:    found = Find(E_{i,l})          // E_{i,l} singular
4:    if found then
5:       swap(E_{i,k}, E_{i,m})      // E_{i,k} is E_{i,m}'s brother
6:    end if
7:    i = i - 1
8:    k = Ancestor(m, i)
9: end while
10: End Disfragmentation
```

Fig. 8. Disfragmentation algorithm.

Therefore, there are two new algorithms to be developed: one for the disfragmentation and another for the reordering. In the following sections, these new algorithms and the generic situations when they must be used will be explained. We will also study some new theorems that can be obtained by applying these new algorithms.

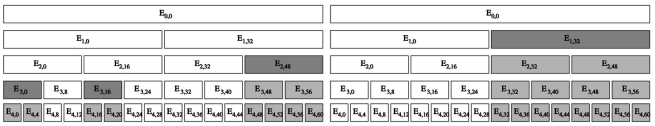
5.1 Disfragmentation Algorithm

The basic idea of this algorithm is to group all of the free entries of the table into several free sets that permit meeting any request needing a number of entries equal to or lower than the available table entries. Thus, the objective of the algorithm is to perform a grouping of the free entries in order to be able to apply Theorem 4. This algorithm repeats, as often as necessary, a process that consists of joining the entries of two free sets of the same size in a unique free set. This joining will be effective only if the two free sets do not already belong to the same greater free set. Therefore, the algorithm is restricted to only singular sets. The goal is to have a free set of the biggest size in order to be able to meet a request of this size. For that purpose, the table has enough free entries which, however, belong to two small free sets that are not able to meet that request.

Fig. 8 shows the code of the disfragmentation algorithm. Starting from a free set $E_{i,k}$, a check is made to see if there is another free set $E_{i,l}$ that is not $E_{i,k}$'s brother (function $Find$, line 3). If this exists, all of the entries of both sets are joined in a new free set of double size. This is achieved by applying Proposition 5 (procedure $swap$, line 5). As the new free set cannot be the only one at level $i-1$, the process must continue. This process can continue up to level 1 (the loop from lines 2 to 9). The resulting set from the joining is the $E_{i,k}$ of its level (lines 7 and 8).

The basic action of this disfragmentation algorithm is the execution of the procedure $swap()$. This procedure makes a transformation in the table through an interchange of sets. This fact was shown in an informal way in Example 1, where we showed that the interchange of the suitable sets ends up with a unique singular set at a specific level. This would permit us to pass from a situation where a request of the most restrictive type cannot be met to another situation where this is possible. This interchange is reflected in Theorem 6.

Theorem 6. *From a situation in which $\exists E_{i,k}$ and $\exists E_{i,l}$ are both singular sets, $1 < i \leq 6$ and $k \neq l$, it is possible to bring about a new situation, where $\exists\, E_{i-1,j}$, which is free so that $E_{i-1,j} = E_{i,m} \cup E_{i,l}$, k, l, $m \in I_i$, $j \in I_{i-1}$.*



(a) (b)

Fig. 9. Situation (a) before and (b) after applying the disfragmentation algorithm successively at several levels.

Proof. Because $\exists\, E_{i,k}$ and $\exists\, E_{i,l}$, which are singular sets, $\nexists E_{i-1,j}$ such that $E_{i-1,j} = E_{i,k} \cup E_{i,l}$. Proposition 5 permits us to interchange the singular set $E_{i,k}$ and the nonfree set $E_{i,m}$, where $m = Brother(l)$. Thus, we interchange the free set $E_{i,k}$ with $E_{i,l}$'s brother, which is occupied. This process creates a free $E_{i-1,j}$ such that $E_{i,m} \cup E_{i,l} = E_{i-1,j}$. □

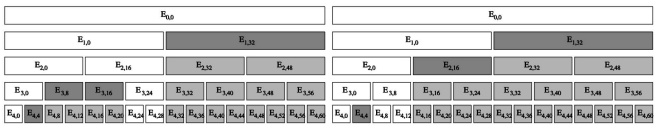
However, it is possible that, in some situations, the problem is not corrected with only an interchange. In these cases, the disfragmentation algorithm must perform several iterations. This is shown in the following example:

Example 5. After the release of the sets $E_{3,0}$ or $E_{3,16}$, the situation shown in Fig. 9a arises. Then, the disfragmentation algorithm is applied. In the first iteration, the algorithm interchanges the sets $E_{3,0}$ and $E_{3,24}$ and, so, the set $E_{2,16}$ is now free. However, there are 32 free entries and there is no free set at level 1 to meet a possible request of type $d = 2$. Obviously, this is because the table is not normalized and there are two singular sets at level 2. Therefore, the disfragmentation algorithm is going to perform another iteration. At the starting point of the second iteration, the singular sets are $E_{2,16}$ and $E_{2,48}$. The algorithm now interchanges $E_{2,16}$ and $E_{2,32}$ such that both $E_{2,32}$ and $E_{2,48}$ end up free and, so, the singular set now is $E_{1,32}$. This situation is shown in Fig. 9b. Note that the table is now normalized. If we now have a request of type $d = 2$, then this would be met with the entries of the set $E_{1,32}$.

Therefore, the disfragmentation algorithm consists of applying Theorem 6 at all levels where there is more than a singular set. Thus, starting from a table that is not normalized, a normalized table is achieved. This is reflected in the following theorem:

Theorem 7. *The disfragmentation algorithm permits us to obtain a normalized table from a table that was not normalized, which was obtained after a release of a request in a normalized table.*

Proof. The disfragmentation algorithm applies Theorem 6 successively at all levels where there are more than two singular sets. Starting from the level where the release has happened, it studies all of the levels up to level 1, joining couples of free sets of the same level. This way, after applying the algorithm, not more than one singular set can exist at each level. Thus, after applying the disfragmentation algorithm, the table is normalized again. □



(a) (b)

Fig. 10. Situation (a) before and (b) after applying the disfragmentation algorithm. Note that, starting from an ordered table, the disfragmentation algorithm leaves the table ordered and normalized.

Let now see how disfragmentation influences the ordering of a table.

Example 6. After the release of the set $E_{3,8}$ or $E_{3,16}$, the situation in Fig. 10a is achieved. Note that the table is ordered because the smaller sets are always placed at the left of the greater sets. However, the table is not normalized because, at level 3, there are two singular sets. After applying the disfragmentation algorithm, the situation shown in Fig. 10b is obtained. Note that the table remains ordered but now is also normalized.

If the disfragmentation algorithm were to be applied successively to several levels, then the order in the table would not be modified either because, starting from an ordered table, the greater sets will be always placed to the right. Although the algorithm sweeps several levels by grouping the singular set formed in the previous iteration with the singular set that existed at that level, the order will not be altered.

Note that we are neither claiming that the disfragmentation algorithm orders the table nor that, after a release, the table will always end up ordered, as shown in Example 7. When we want to recover the order in the table, we must apply the reordering algorithm that we will study in the following section. What we want to highlight is that the disfragmentation algorithm does not cause any disorder in the table.

In the previous example, we have seen that applying the disfragmentation algorithm on an ordered table that is not normalized obtains an ordered and normalized table. We are going to see that in a formal way with the following theorem:

Theorem 8. *After applying the disfragmentation algorithm to an ordered table, in which there is at most a level i, with $1 < i \leq 6$, having two singular sets, the table ends up normalized and remains ordered.*

Proof. Let $E_{i,k}$ and $E_{i,l}$ be the two singular sets of the unique level i that has two singular sets, where $k < l$ and $k, l \in I_i$. Applying Theorem 7, a free set $E_{t,u}$ is obtained, with $0 < t < i$ and $u \in I_t$.

Let $E_{p,q}$ and $E_{r,s}$ be any two singular sets such that $0 < r < i < p \leq 6$, $q \in I_p$, and $s \in I_r$. As the table was initially ordered, $E_{p,q} \lhd E_{i,k} \lhd E_{i,l} \lhd E_{r,s}$.

Having this situation as the initial point, we have two possibilities:

- If $r < t$, then what must be proven is $E_{p,q} \lhd E_{t,u} \lhd E_{r,s}$.

  Let us first study the case when $t = i - 1$. In this case, $u = Ancestor(l, t)$. Because $E_{i,k} \lhd E_{i,l}$, $E_{i,k} \lhd E_{t,u}$ from Proposition 7. Moreover, because $E_{p,q} \lhd E_{i,k}$ and $E_{i,k} \lhd E_{t,u}$, from the transitive proposition of the order relation, $E_{p,q} \lhd E_{t,u}$.

  If $r < t < i - 1$, then the proof is similar to, but applies in a successive way, Proposition 7 and the transitive proposition of the order relation. Therefore, whichever is the case, if $r < t$, then $E_{p,q} \lhd E_{t,u}$.

  On the other hand, $E_{t,u} \lhd E_{r,s}$ must also be true. For this to be so, it should be true that $r \leq t$ and $u < s$. The first is in the initial hypothesis. Let us study the second one for the particular case where $t = i - 1$. Because $E_{i,l} \lhd E_{r,s}$, $l < s$. Moreover, as $u = Ancestor(l, t)$ and following from Proposition 2, which relates the indexes of a set and its descendants, $u \leq l$ and, so, $u < s$. Therefore, $E_{t,u} \lhd E_{r,s}$.

  Again, for the general case where $r < t < i - 1$, the proof is similar, but it must be performed in a successive way. Therefore, in any case, if $r < t$, then $E_{t,u} \lhd E_{r,s}$.

  This way, $E_{p,q} \lhd E_{t,u} \lhd E_{r,s}$ and, so, if $r < t$, then the new singular set $E_{t,u}$, which the disfragmentation algorithm has generated, keeps the table ordered.
- If $r = t$, then the disfragmentation algorithm, using $E_{t,u}$ and $E_{r,s}$, will generate a singular set $E_{r-1,h}$, where $h \in I_{r-1}$ and $h = Ancestor(s, r - 1)$. In this case, we only need to prove that $E_{p,q} \lhd E_{r-1,h}$.

  This proof is similar to the one performed in the previous point, where Proposition 7 and the transitive proposition of the order relation must be applied successively.

Therefore, in any case, after applying the disfragmentation algorithm to an ordered table in which there is at most one level i, with $1 < i \leq 6$, having two singular sets, the table ends up normalized and remains ordered. □

Thus, the disfragmentation algorithm normalizes a nonnormalized table and, if it was already normalized, then it remains ordered after applying this algorithm. This result will be used again in Section 6.

We are now going to present the reordering algorithm. Just as in the case of the disfragmentation algorithm, we will first include an informal description by using simple examples and, later, we will show its properties, stating and proving several theorems.

5.2 Reordering Algorithm

The reordering algorithm basically consists of an order algorithm, but applies it at a level of sets. This algorithm has been designed to be applied to a table that is nonordered, with the purpose of leaving the table ordered according to Definition 6. The reordering algorithm sets an order from a lower to a larger set size in ascending order and from left to right (according to Fig. 3). Thus, when the order fails as a consequence of the apparition of new singular incorrectly

```
1: Procedure Reordering
2: for k = 6 downto 2 do
3:    para todo E_{k,l} singular do
4:       for i = k − 1 downto 2 do
5:          found = Find(E_{i,j}) // E_{i,j} singular, E_{k,l} ◄ E_{i.j}
6:          if found then
7:             swap(E_{i,j}, E_{i,Ancestor(k,i)})
8:          end if
9:       end for
10:   end for
11: end for
12: End Reordering
```

Fig. 11. Reordering algorithm.

situated sets, the algorithm will act to reestablish the order right again.

When a set is released, the algorithm checks if this set is correctly placed in relation to the other free sets. Hence, this new free set must have to *its left* all of the smaller free sets and placed to *its right* all of the greater free sets. If its position is not correct, then it will perform the needed movements to reestablish the order in the table. These movements will be performed by making interchanges between sets, in the same way as the disfragmentation algorithm works.

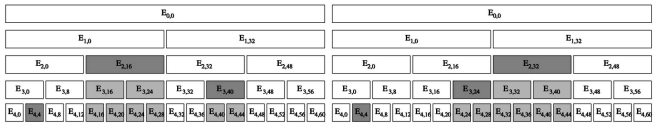
As stated at the beginning of Section 5, the reordering algorithm will be used after every entry release. This release always happens in an ordered table. Therefore, just before applying the reordering algorithm, there will be in the table only one singular set that is not ordered in relation to the other singular sets. It should be noted that the way in which this algorithm works is very similar to that of the well-known order algorithm *Direct Selection* [9].

In Fig. 11, the basic code that makes up the algorithm is shown. The algorithm studies all of the possible levels (lines 2-11), starting from the bottom. In each of them, there will be at most one singular set $E_{k,l}$ except at one of the levels where there may be two singular sets. In that case, only one of them can be badly ordered. Therefore, at each level, if there is a singular set $E_{k,l}$, it will be tested to learn if there is a singular set $E_{i,j}$ at the upper levels (lines 4-9) that is placed to its left. If there are several sets, then the one that is at the leftmost will be selected (lines 5 and 6) and the ordering takes place by applying Proposition 5 over $E_{i,j}$ and the ancestor of $E_{k,l}$ at level i (procedure $swap$, line 7).

As we did in the previous section with the disfragmentation algorithm, we will now see a simple example of the working of the reordering algorithm.

Example 7. Let us take as our initial situation the one shown in Fig. 12a. The table has arrived at that situation after the release of the set $E_{2,16}$ or $E_{3,40}$ in an ordered table. The reordering algorithm will interchange the sets $E_{2,16}$ and $E_{2,32}$. Therefore, the final situation will be the one shown in Fig. 12b, where the singular sets are now $E_{4,4}$, $E_{3,24}$, and $E_{2,32}$. Note that the table is ordered again.

Thus, with the correct interchange of sets, we can pass from a situation where some singular sets do not maintain the order relation of Definition 5 to another situation where



(a) (b)

Fig. 12. The situation (a) before and (b) after applying the reordering algorithm.

the new singular sets do maintain that order relation. This is shown in a formal way in the following theorem:

Theorem 9. *Let $E_{k,l}$ and $E_{i,j}$ be the only singular sets at levels k and i, respectively, with $0 < i < k \leq 6$, $l \in I_k$, and $j \in I_i$, such that no order relation exists between them. Thus, $E_{k,l} \not\lhd E_{i,j}$. However, it is possible to change to another situation with $E_{k,n} \lhd E_{i,m}$, with $n \in I_k$, and $m \in I_i$, where $E_{k,n}$ and $E_{i,m}$ are now the only singular sets of levels k and i, respectively. Moreover, all of the accepted requests until that moment are maintained.*

Proof. Proposition 5 permits the sets $E_{i,j}$ and $E_{i,m}$ to be interchanged, where $m = Ancestor(l, i)$. Therefore, the singular set at level i is now $E_{i,m}$, whereas $E_{i,j}$ is no longer free. The same happens with the singular set at level k, which is now $E_{k,n}$, with $n = l - m + j$. What must be proven is that $E_{k,n} \lhd E_{i,m}$. For that purpose, it will be enough to prove that $E_{k,n} \lhd E_{k,l}$, which, together with the application of Proposition 7, allows us to obtain $E_{k,n} \lhd E_{i,m}$.

Let us suppose $E_{k,n} \not\lhd E_{k,l}$, which means that $n \geq l$. If $n \neq l$ were false, then $E_{k,n}$ and $E_{k,l}$ would be the same set and $n > l$. However, $E_{k,l} \lhd E_{k,n}$ and, according to Proposition 7, $E_{k,l} \lhd E_{i,j}$, which is false according to the initial hypotheses. Therefore, $E_{k,l} \not\lhd E_{k,n}$, $n < l$, and $E_{k,n} \lhd E_{k,l}$, and, from Proposition 7, $E_{k,n} \lhd E_{i,m}$. □

This result permits us to obtain two singular ordered sets from two singular nonordered singular sets. By applying this result in a successive way, we can obtain an ordered table. This result is shown in the following theorem:

Theorem 10. *The reordering algorithm permits us to pass from a table that is not ordered to another one that is ordered.*

Proof. The reordering algorithm applies Theorem 9 successively to any couple of sets $E_{i,j}$ and $E_{k,l}$, which are singular, with $0 < i < k \leq 6$, $j \in I_i$, and $l \in I_k$ such that $E_{k,l} \not\lhd E_{i,j}$. After applying the theorem, $E_{k,n} \lhd E_{i,m}$, with $n = l - m + j$ and $m = Ancestor(l, i)$. Therefore, after applying the reordering algorithm, the table becomes ordered. □

As stated in the description of the reordering algorithm, its working is similar to the well-known *Direct Selection* ordering algorithm. We therefore have an even stronger guarantee that our reordering algorithm works properly and obtains an ordered table.

To conclude our study of the reordering algorithm, let us see what happens when this algorithm is applied over a table that is normalized. Specifically, we want to know if it remains normalized. For that purpose, we use the fact that

the reordering algorithm does not generate more singular sets than were previously there, but it simply changes their order. We will see this in the following theorem:

Theorem 11. *A normalized table remains normalized after applying the reordering algorithm.*

Proof. If the table was normalized, then it must have at most one singular set at each level. However, the reordering algorithm always interchanges a singular set $E_{i,j}$ with another set $E_{i,m}$ which is not free. Thus, the algorithm will never generate more free sets than those existing at each level. Therefore, we have at most a singular set per level and, so, the table also remains normalized. □

Once the filling-in, disfragmentation, and reordering algorithms have been presented, we must draw together the elements studied and check that the global treatment of the table is correct.

6 Global Management of the Table

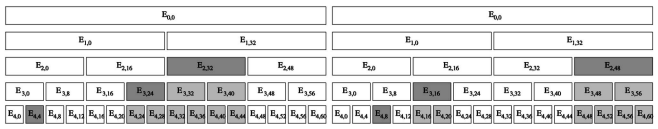
For the global management of the table, having both insertions and releases, we have shown that a combination of the filling-in and disfragmentation algorithms (and even the reordering algorithm, if needed) must be used. However, the properties of the filling-in algorithm have been proven with a certain initial hypothesis in which the releases were not considered. Therefore, when releases are considered and we use the disfragmentation algorithm (and the reordering algorithm, if needed), we must prove that the table will always have a correct status in order that the propositions of the filling-in algorithm continue to be true (for example, Theorem 4). We will now prove that the final situation achieved in the arbitration table after insertions and releases of requests with their corresponding disfragmentations and reorderings is equivalent to that achieved with only the insertions of the requests finally remaining in the table. The term *equivalent* refers to the capacity to meet requests and is exactly defined in the following:

Definition 7. *Tables* T *and* T′ *are said to be equivalent if they can meet the same requests.*

This means that, regardless of which sets of both tables are free, they must be able to meet the same requests. As we shall see in the following, two tables will be equivalent if they have the same number of singular sets and these sets are placed at the same levels.

As an example, the tables shown in Figs. 9a and 9b are not equivalent, even though they have the same number of free entries. Note that, with the second table, requests of type $d \geq 2$ can be met, whereas, with the first table, only requests of type $d' \geq 4$ can be met. In the same way, the tables in Figs. 10a and 10b also are not equivalent. We shall see in the following an example with two tables that are not equal but are equivalent.

Example 8. In Fig. 13, we can see two tables that are equivalent. In the one shown in Fig. 13a, the singular sets are $E_{4,4}$, $E_{3,24}$, and $E_{2,32}$, whereas, in Fig. 13b, the singular sets are $E_{4,8}$, $E_{3,16}$, and $E_{2,48}$. The other sets, which are not descendants of these, are busy. However, both tables are able to meet exactly the same request sequences.



(a) (b)

Fig. 13. Example of two equivalent tables.

The following theorem proves that two normalized tables that have the same number of singular sets located at the same levels have the same number of free entries.

Theorem 12. *If two normalized tables,* T *and* T′*, such that* $\forall E_{i,j} \in T$*, where* $E_{i,j}$ *is singular,* $\exists E_{i,k} \in \mathrm{T}'$*, where* $E_{i,k}$ *is also singular, and* $\forall E_{i,k} \in \mathrm{T}'$*, where* $E_{i,k}$ *is singular,* $\exists E_{i,j} \in T$*, where* $E_{i,j}$ *is singular, with* $0 < i \leq 6$ *and* $j, k \in I_i$*, then both tables have the same number of free entries.*

Proof. This proof works by *reductio ad absurdum*. Let us suppose that both tables T and T′ do not have the same number of entries. Therefore, a level l, with $0 \leq l \leq 6$, must exist such that, from l (toward lower levels), the tables do not have the same number of free entries. Because the number of singular sets is the same at all of the levels of both tables, it must be a number n of free nonsingular sets in one of the tables which are not found in the other. In order for these n free sets to be nonsingular sets, their brother sets must also be free. However, this would form, in a table, one or more singular sets at some level t, with $0 < t < i$, that are not in the other table. Obviously, this contradicts the initial hypothesis.

Therefore, if two tables, T and T′, are normalized and $\forall E_{i,j} \in T$, where $E_{i,j}$ is singular, $\exists E_{i,k} \in \mathrm{T}'$, where $E_{i,k}$ is also singular, and $\forall E_{i,k} \in \mathrm{T}'$, where $E_{i,k}$ is singular, $\exists E_{i,j} \in T$, where $E_{i,j}$ is singular, with $0 < i \leq 6$ and $j, k \in I_i$, then both tables have the same number of free entries. □

Using the previous theorem, we are going to prove that two normalized tables are equivalent if they have the same number of singular sets and these are located at the same levels.

Theorem 13. *If two normalized tables,* T *and* T′*, such that* $\forall E_{i,j} \in T$*, where* $E_{i,j}$ *is singular,* $\exists E_{i,k} \in \mathrm{T}'$*, where* $E_{i,k}$ *is also singular, and* $\forall E_{i,k} \in \mathrm{T}'$*, where* $E_{i,k}$ *is singular,* $\exists E_{i,j} \in T$*, where* $E_{i,j}$ *is also singular, with* $0 < i \leq 6$ *and* $j, k \in I_i$*, then both tables are equivalent.*

Proof. If T and T′ are normalized, then it is because they have at most a singular set per level. Moreover, $\forall E_{i,j} \in T$, where $E_{i,j}$ is singular, $\exists E_{i,k} \in \mathrm{T}'$, where $E_{i,k}$ is also singular, with $0 < i \leq 6$, and vice versa. From Theorem 12, both tables have the same number of free entries. Therefore, from Theorem 4, the tables T and T′ are able to meet the same number of requests and are therefore equivalent. □

We shall now prove that the resulting situation of insert and release requests is equivalent to the final situation if there were only the requests that remain after the releases. Thus, all of the work developed for the case when this did not have releases is also applicable to the case when we have request release (considering the disfragmentation and reordering algorithms). The reason for this is that we have an equivalent situation regarding new requests and, in this case, we can apply Theorem 4. This way, we can successfully unite both parts of the theory developed. We shall now see the theorem to prove this equivalence.

Theorem 14. *Let a series of requests $d_1, d_2, \cdots, d_n$ be followed by a series of releases $d'_1, d'_2, \cdots, d'_m$ after each of the reordering and/or disfragmentation algorithms are applied, when needed, for the table to be normalized and ordered, where $\exists d_i / d'_j = d_i$, $\forall j \in [1, m]$. This sequence of insertions d_i, followed by these releases d'_j, produces a table equivalent to that which results from the insertion of only the following requests, $\{d_1, d_2, \cdots, d_n\} - \{d'_1, d'_2, \cdots, d'_m\}$, which are the requests that remain after the releases when all the requests have been satisfied.*

Proof. Let T be the resulting table after realizing the insertion of the requests $d_1, d_2, \cdots, d_n$, followed in any order by the releases $d'_1, d'_2, \cdots, d'_m$. On the other hand, let T′ be the resulting table if the insertions $\{d_1, d_2, \cdots, d_n\} - \{d'_1, d'_2, \cdots, d'_m\}$ are directly carried out. What we want to prove is that T and T′ are equivalent tables. For that purpose, we are going to show that both tables have the same number of singular sets and are placed at the same levels. Thus, according to Theorem 13, both tables are able to meet exactly the same requests.

As explained in Section 5, we have two options to dynamically manage the table:

1. After each insertion d_i and each release d'_j, the disfragmentation algorithm is applied. Thus, according to Theorem 7, the table T remains normalized. On the other hand, according to Theorem 2, the table T′ is also normalized.
2. In this case, after each release d'_j, the reordering algorithm and/or the disfragmentation algorithm are applied. Thus, according to Theorems 7, 8, and 11, table T is normalized. For the same reason as before, by Theorem 2, the table T′ is also normalized.

Thus, in both cases, the tables T and T′ are normalized and at most have only one singular set per level. We shall see that both tablesalso the same number of singular sets, placed at the same levels.

If T and T′ do not have the same number of singular sets or, if they do, they are not situated at the same levels, then it is because, for some level i, with $0 < i \leq 6$, $\exists E_{i,j} \in T$, which is singular, $j \in I_i$, but $\nexists E_{i,k} \in \mathrm{T}'$, with $E_{i,k}$ singular, $k \in I_i$. The sets $E_{i,l} \in \mathrm{T}'$, which are not free, with $l \in I_i$, could be due to requests of type $d = 2^i$ or, by Proposition 2, to requests of type $d' = 2^{i+p}$, with $1 \leq p \leq 6 - i$ such that $E_{i,l} = \bigcup_{m=0}^{2^p-1} E_{i+p,j+m\times 2^{6-(i+p)}}$. This would mean that the request of type $d = 2^i$ (or, in a similar way, the request of type $d' = 2^{i+p}$) in which both tables differ would be in a sequence of requests $d_1, d_2, \cdots, d_n$ and not in the other sequence $\{d_1, d_2, \cdots, d_n\} - \{d'_1, d'_2, \cdots, d'_m\}$. However, we know that this is not possible.

Hence, we have proven that both tables T and T′ have the same number of singular sets and are located at the same levels. Thus, from Theorem 13, both tables are able to meet the same requests. Therefore, we have proven that both tables T and T′ are equivalent. □

After having proven this theorem, we have linked both parts of the theory developed. Now, it is possible to apply Theorem 4 in any situation in order to meet new requests. The conclusion is very important because, whatever the request may be, with our proposal, it is possible to meet it in the table by simply having enough free entries.

7 Conclusions

In [6] and [7], we proposed a scheduling methodology to provide applications with QoS in an InfiniBand subnet. For a certain maximum latency, the maximum distance between two consecutive entries in the arbitration table is computed. We proposed categorizing the distances in the powers that are divisors of 64, which simplifies the management of the requests and achieves a good filling in of the table. This model has been evaluated in previous papers using simulations and results show that all applications achieve the QoS requirements that they had previously requested. Although these simulation results are very important, a formal proof of our proposed methodology is also needed.

In this paper, we have defined a methodology that is based on sets of entries in the arbitration table. Each set is the sequence of entries in the table that have a certain distance between any two consecutive entries. After defining the model, we have stated and proven a series of propositions that have further allowed us to state the theorems. With these theorems, we have proven that, if we have enough free entries in the arbitration table, then, with our algorithm, we can always situate the request in the table. This is because the algorithm always uses the entries in a concrete way, leaving the other free entries in the best place to meet other requests later.

We have studied the different possible situations in which requests are met and released from the arbitration table. We have analyzed the different situations, and we have found some problematic situations causing our filling in algorithm to not behave correctly. In order to solve these situations, we have proposed two new algorithms. The first is the disfragmentation algorithm, which must be executed every time a request is released on the table. The idea of this algorithm is to join loose sets (sequences) in order to form other sets able to meet requests that are more restrictive than the previous ones. For that purpose, an interchange between a free set and another occupied one can be made. This is in order to put together the two free sets and, in this way, to form a large free set able to meet a more restrictive request. The second algorithm proposed is used to maintain the order among the sets. The idea is that we might need to have the free sets in a certain order based on their size. As in the previous case, to achieve our propose, the algorithm will interchange different sets.

We have shown that if we only want to change the table after a release, we need both the disfragmentation and the reordering algorithms. If we perform the disfragmentation algorithm both after an insertion and after a release, then it is not necessary to maintain an order among the sets and, hence, the reordering algorithm becomes useless.

The algorithm to fill in the arbitration table, together with the disfragmentation algorithm and the algorithm to maintain the order among the free sets when there are releases in the table permit us to make a dynamic use of the arbitration table of InfiniBand when there are requests to be settled in the table and also requests to be released from the table.

ACKNOWLEDGMENTS

This work was partly supported by the Spanish CICYT under CONSOLIDER INGENIO 2010 CSD-2006-46 and TIN2006-15516-C04-02 grants and the Junta de Comunidades de Castilla-La Mancha under Grant PBC-05-005.

REFERENCES

[1] InfiniBand Trade Assoc., http://infinibandta.com, 1999.
[2] *Advanced Switching Core Architecture Specification Rev 1.1.* Advanced Switching Interconnect Group, Mar. 2005.
[3] HyperTransport Technology Consortium, *HyperTransport™ I/O Link Specification,* http://www.hypertransport.org, 1999.
[4] *InfiniBand Architecture Specification Vol. 1. Release 1.0.* InfiniBand Trade Assoc., Oct. 2000.
[5] M. Katevenis, S. Sidiropoulos, and C. Corcoubetis, "Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch," *IEEE J. Selected Areas in Comm.,* Oct. 1991.
[6] F.J. Alfaro, J.L. Sánchez, M. Menduiña, and J. Duato, "Formalizing the Fill-In of the InfiniBand Arbitration Table," Technical Report DIAB-03-02-35, Dept. of Informatics, Univ. of Castilla-La Mancha, http://www.info-ab.uclm.es/trep.php, Mar. 2003.
[7] F.J. Alfaro, J.L. Sánchez, and J. Duato, "A New Proposal to Fill In the InfiniBand Arbitration Tables," *Proc. 32nd IEEE Int'l Conf. Parallel Computing (ICPP '03),* pp. 133-140, Oct. 2003.
[8] F.J. Alfaro, J.L. Sánchez, and J. Duato, "QoS in InfiniBand Subnetworks," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 9, pp. 194-205, Sept. 2004.
[9] D.E. Knuth, "Sorting and Searching," *The Art of Computer Programming 3,* Addison-Wesley, 1973.

Francisco J. Alfaro received the MS degree in computer science from the University of Murcia, Spain, in 1995 and the PhD degree from the University of Castilla-La Mancha, Spain, in 2003. He is currently an assistant professor of computer architecture and technology in the Department of Computer Systems at the University of Castilla-La Mancha. His research interests include high-performance local area networks, quality of service (QoS), design of high-performance routers, and design of on-chip interconnection networks for multicore systems. He is a member of the IEEE Computer Society.

José L. Sánchez received the PhD degree from the Technical University of Valencia, Spain, in 1998. Since November 1986, he has been a member of the Department of Computer Systems (formerly the Department of Computer Science) at the University of Castilla-La Mancha, Spain. He is currently an associate professor of computer architecture and technology. His research interests include multicomputer systems, quality of service (QoS) in high-speed networks, interconnection networks, parallel algorithms, and simulation.

Manuel Menduiña is a high school teacher of mathematics. In 1990, he joined the Department of Computer Systems, University of Castilla-La Mancha, Spain, as a part-time assistant professor. His main research fields are formal models and operation researches.

José Duato received the MS and PhD degrees in electrical engineering from the Polytechnic University of Valencia (UPV), Spain, in 1981 and 1985, respectively, where he is now a professor in the Department of Computer Engineering (DISCA). He served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers* and is serving as an associate editor of the *IEEE Computer Architecture Letters (CAL)*. He was the general cochair of the 30th International Conference on Parallel Processing (ICPP '01), a program chair of the 10th International Symposium on High-Performance Computer Architecture (HPCA-10), and a program cochair of ICPP 2005. Also, he served as a cochair, Steering Committee member, vicechair, or program vommittee member for more than 55 conferences, including HPCA, the International Symposium on Computer Architecture (ISCA), the International Parallel Processing Symposium/Symposium on Parallel and Distributed Processing (IPPS/SPDP), the International Parallel and Distributed Processing Symposium (IPDPS), ICPP, the International Conference on Distributed Computing Systems (ICDCS), Euro-Par, and the International Conference on High Performance Computing (HiPC). His research interests include interconnection networks and multiprocessor architectures. His research results have been used in the design of the Alpha 21364 microprocessor and the Cray T3E and IBM BlueGene/L supercomputers. He has published more than 350 papers. He is the first author of the book *Interconnection Networks: An Engineering Approach*. He is a member of the IEEE Computer Society.