THE ANALYSIS AND PERFORMANCE OF BATCHING ARBITERS †

Lindsay Kleeman and Antonio Cantoni

Department of Electrical and Computer Engineering
University of Newcastle
New South Wales 2308

† Work supported by the Australian Computer Research Board

ABSTRACT

A class of arbiters, known as batching arbiters, is introduced and defined. A particularly simple decentralised example of a batching arbiter is described, with motivation given for the batching arbiter model adopted. It is shown that under reasonable assumptions, batching arbiters can be described by a finite state Markov chain. The key steps in the analysis of the arbiter performance are the method of assigning states, evaluation of state transition probabilities and showing that the Markov chain is irreducible. Arbiter performance parameters are defined, such as proportion of time allocated to each requester and mean waiting time for each requester. Apart from results describing the steady state behavior of the arbiter for general system parameters, a number of limiting results are also obtained corresponding to light and heavy request loading.

Finally, numerical results of practical interest are presented, showing the performance parameters of the arbiter versus request rates for various configurations.

1 INTRODUCTION

An arbiter [1] is a digital circuit designed to provide mutually exclusive access to a shared resource among a number of competing requesters. Some form of arbitration can be found in virtually all digital computer based systems. Examples include bus sharing among CPU's and peripherals, multiport memories and refresh control in dynamic memories.

The characteristics of arbiters are key factors in determining the performance of computer systems [2]. Since arbiters play a dominant role in system resource allocation, their modelling and performance evaluation is of considerable interest [3]. This paper (which is a condensed version of [4]) describes a method of modelling and analysing the performance of a class of arbiters, termed batching arbiters. Batching arbiters are defined more precisely in Section 2 with a brief description given here. If a request for the resource occurs in an idling state, when no request inputs to the arbiter are active, then it and any further requests that occur within a certain time duration are batched together and serviced before any subsequent requests are considered. The batched requests are serviced in order of the requesters priorities which are assumed fixed. Any requests which occur during servicing of the batch are ignored until after all batched requests are serviced, at which time all pending requests are once again batched.

A recent example of batching arbiter can be found in the new IEEE Futurebus [5] arbitration scheme, which employs a wired-or priority resolution circuit to allocate the resource within a batch. The batching process is motivated by the desire to prevent hogging and achieve some degree of fairness. Batching arbiters can be implemented either as centralised or decentralised arbiters.

A particularly simple implementation of a decentralized batching arbiter is presented in Section 3, in which a daisy chain sequences the servicing of batched requests and a common line locks out further requests. Batching arbiters differ from conventional fixed priority arbiters [1] in the lock out mechanism which guarantees all requesters are serviced within a finite time under all loading conditions. Simple fixed priority arbiters, such as Intel's 8289 Bus Arbiter (in a daisy chain configuration) and the IEEE S-100 bus DMA arbitration, allocate the resource to the highest priority request pending without any batching mechanism, thus the possibility exists of a high priority requester locking out others by recurrent requesting. Although this hogging situation cannot occur in batching arbiters, their fairness is limited under

heavy loading conditions, as will become apparent from the results presented later in this paper.

The performance of batching arbiters cannot be analysed using standard queueing theory techniques. However it is shown in Sections 4 and 5 that under reasonable assumptions, batching arbiters can be modelled and analysed with a finite state, irreducible Markov chain.

Two broad aspects of performance are considered in [4]. The first, which is discussed in Section 6 of this paper, is concerned with efficiency and utilization characteristics of batching arbiters. The second is an analysis of the behavior of the decentralized daisy chain batching arbiter with respect to metastable behavior [4] [6]. Failure of arbiters due to metastable behavior is unavoidable due to the asynchronous nature of their inputs [7]. Reference [4] develops a technique for estimating the failure rate of the arbiter due to metastable behavior and identifies the significant factors that affect the failure rate.

Limiting cases of light and heavy request loading are considered in the theoretical analysis. Also numerical results are presented in Section 7 to illustrate the effect of various system parameters, such as circuit time delays, mean request rates, numbers of requesters and service times.

2 FIXED PRIORITY BATCHING ARBITER MODEL

The behavior of arbiters with fixed priority and batching can be described by the following model. Assuming initially no requests are pending and no requesters are being serviced, then the first request to occur initiates a time interval of duration D_3. Other requests occurring during this interval D_3 are batched together with the first request and no requests occurring after the interval D_3 are considered until all the previous requests have been serviced. The batched requests are serviced in a logical time period referred to as a batch. As shown in Figure 1, a non-zero batch consists of a time interval D_2 followed by the ordered servicing of the batched requests (in the order defined by the fixed priority of the arbiter) followed by another time interval D_1 and then, if at least one request is made during the batch, a further time interval D_3 follows.

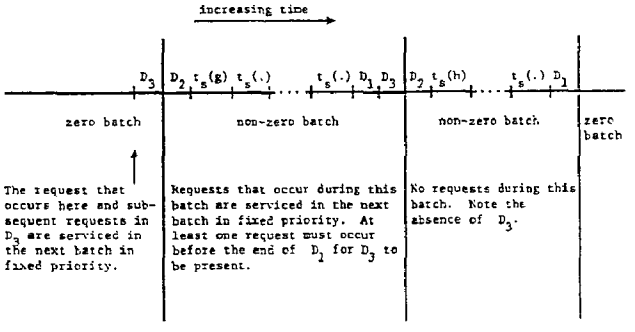


Figure 1: Batch transitions of fixed priority batching arbiter model.

Example

The arbiter presented in this example is designed for three requesters labelled 1, 2 and 3. The highest priority is given to 1 then the next priority to 2 and 3 is the lowest. The incoming requests from requesters 1, 2 and 3 determine the behavior of the model. Fig. 2 illustrates a sequence of events.

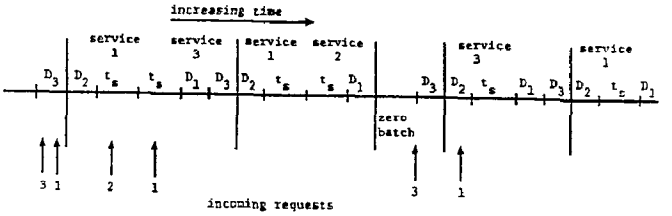


Figure 2: Example model behavior

Even though requester 3 requests before requester 1, batching and the fixed priority determines that requester 1 is serviced before requester 3. Notice that 2 requests before the servicing of 3 and, since 2 is not batched with 3, 3 is serviced before 2. Requester 1 requests soon after its service, enabling it to be serviced in the next batch. The time available for 3 to request to be included in the next batch is much shorter than for 1. In fact, if $D_1 = D_3 = 0$ in a particular arbiter, 3 can never be serviced in two consecutive batches.

Definitions of time durations D_1, D_2, D_3 and $t_s(h)$

D_1 occurs after all batched requests have been serviced

D_2 occurs after requests are batched. Requests lodged during D_2 wait until after all currently batched requests have been serviced before being batched themselves.

D_3 occurs at the end of all batches preceding a non-zero batch. D_3 is generated only if at least one request occurs before the end of D_1 in a non-zero batch, otherwise a zero batch follows D_1. In a zero batch D_3 occurs as soon as a request occurs.

$t_s(h)$ corresponds to the time duration in which requester h is serviced and holds the resource. The arbiter is assumed to have a request input, Req.h, h=1 ..k, from each requester. This is asserted to lodge a request and held until the arbiter responds by asserting the acknowledge signal, Ack. h, indicating that the resource is allocated to requester h, as shown in Fig. 3.

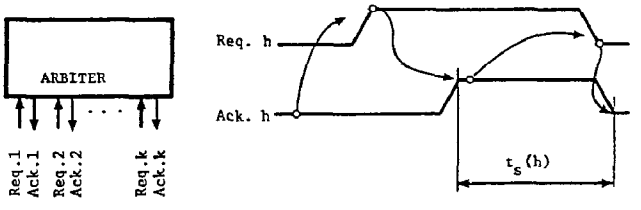


Figure 3: Arbiter request, acknowledge protocol.

When requester h finishes with the resource it drops its Req. h signal and the arbiter responds by dropping its Ack. h signal. Requester h cannot

request again until Ack. h is low. The time duration $t_s(h)$ corresponds to Ack. h being high. The order $t_s(h)$ occurs within a batch is determined by h, with the smallest index h occurring first.

Remarks

(i) The case of $D_1 = D_2 = D_3 = 0$ is not excluded.

(ii) There are 2^k possible different batches, where k is the number of requesters.

(iii) The composition of time durations in a batch is a function of not only the requesters serviced during the batch, but also whether a request occurs before D_1 ends, because the existence of D_3 depends on such a request. (This was ignored in counting batches in remark (ii) above).

(iv) A singleton batch refers to a batch in which only one request is serviced. Suppose $D_1=0$, then the last requester serviced in a batch cannot form a singleton batch immediately following, without an intervening zero batch. (It is assumed that requester h requests a non-zero time after $t_s(h)$).

(v) If $D_3=0$, a zero batch is followed by a singleton batch, assuming that no two requests occur at exactly the same time.

(vi) If $D_1=D_3=0$, the lowest priority requester serviced in a batch cannot be serviced in the following batch.

(vii) A full batch refers to the batch in which all requesters are serviced. If $D_1=D_3=0$, remarks (v) and (vi) imply the full batch never occurs.

3 EXAMPLE OF A BATCHING ARBITER

A simple decentralised daisy-chained arbiter which conforms to the batching arbiter model, is described in this section. The time durations D_1, D_2 and D_3 are identified with circuit parameters in [4]. Similar designs have appeared elsewhere [8].

In the distributed arbiter circuit shown in Fig.4 each requester has a circuit module associated with it. The modules are connected via a daisy chain and a common line. The order of the modules in the daisy chain determines the priority of the requesters.

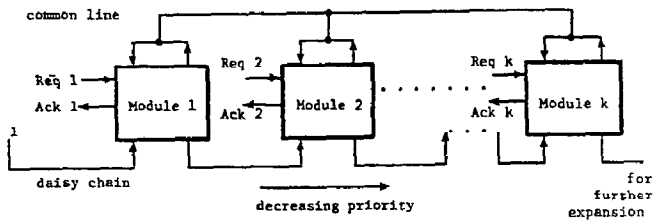


Figure 4: Structure of decentralized daisy-chained batching arbiter.

The start of the daisy chain is connected to logic '1', and the daisy chain is "threaded" through each module. When a module receives a 1 on the daisy chain from higher modules in the chain, it passes the 1 further down the chain if the module has no request latched from its requester, otherwise it "keeps" the 1 until the latched requests have been serviced. This is achieved by use of the common line which realises a wired-or function of all the latched requests of the k modules. Thus, the common line is asserted when at least one request is latched. When the common line is asserted, no new requests can be latched in the modules, and when all latched requests have been serviced, the common line resets and all pending requests are latched. This has the effect of batching requests and preventing a high priority requester hogging the system by continually locking out the other requesters.

4 PROBABILITY ANALYSIS OF THE BATCHING ARBITER MODEL.

This section introduces assumptions regarding random request and service behavior. The batching arbiter model is then shown to be a finite state Markov chain. The state transition probabilities are derived in terms of requester and model parameters.

4.1 Request and Service Modelling

The protocol for requesting, described in Fig. 3, leads to the following assertion:

(i) The probability of a requester requesting after it has already requested and before it has finished being serviced is zero.

The following assumptions are made to enable the analysis to be tractable.

(ii) The probability of requester h making a request whilst it has no request pending nor is being serviced is described by an exponential distribution in time:

$$\text{prob}\,[\text{no request in } [0,t]] = e^{-\lambda_h t} \qquad (4.1)$$

where λ_h is the mean request rate for requester h.

(iii) The service time for requester h, $t_s(h)$, can have an arbitrary distribution described by the probability density function f_h:

$$\text{prob}\,[t_a \le t_s(h) \le t_b] = \int_{t_a}^{t_b} f_h(t)\,dt \qquad (4.2)$$

The mean of $t_s(h)$ is denoted $\frac{1}{\mu_h}$.

Two special cases will be considered later:

(a) constant service time of $\frac{1}{\mu_h}$:

$$f_h(t) = \delta(t - \frac{1}{\mu_h})$$

(b) exponentially distributed service times:

$$f_h(t) = \mu_h e^{-\mu_h t}$$

(iv) Requesters request independently.

(v) Service times are independent.

Remarks

(a) Assumption (ii) is a reasonable approximation

for random requests. It has the property of being memoryless : the probability of a request is independent of previous requests (provided no request is pending).

(b) Different requesters can have different request rates and service times.

4.2 State Definition

The arbiter behavior is characterized by a sequence of batches in time. An idling period corresponds to a zero batch. Each batch has an associated state, determined by the set of requesters serviced in the batch. The state of the n^{th} batch, ${}_kS(n)$, is defined as an integer in the range $0..2^k-1$, where k is the total number of requesters. $S(n)$ is a function of the n^{th} batch as follows.

Define v_n: $\{1, \ldots, k\} \to \{0,1\}$ by

$$v(h) \triangleq \begin{cases} 1, & \text{if requester } h \text{ is serviced in the } n^{th} \text{ batch, denoted } h \in S(n) \\ 0, & h \notin S(n) \end{cases}$$

Then

$$S(n) \triangleq \sum_{h=1}^{k} v(h)2^{h-1} \qquad (4.3)$$

For example, the zero batch has state zero and the full batch has state 2^k-1. There is a one to one correspondence between the set of requesters serviced in a batch and the state of the batch. This explains the use of the notation $h \in S(n)$.

4.3. Markov Property

The sequence of states of the arbiter model with requesters obeying assumptions (i)-(v) of section 3.1 forms a finite state Markov chain. That is

$$\text{prob}[S(n)=i \mid S(n-1)=j_1, S(n-2)=j_2 \ldots, S(0)=j_n] \qquad (4.4)$$

is independent of $j_2, \ldots, j_n$.

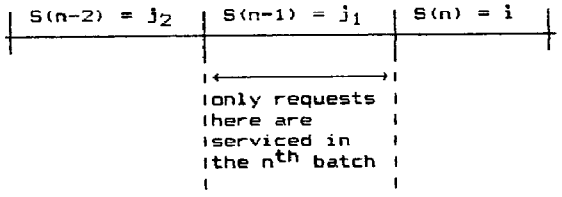


Figure 5. Markov Property of Arbiter Model.

Fig. 5 illustrates that the state of the n^{th} batch is determined only by requests made during the $(n-1)^{th}$ batch is only a function of the $(n-1)^{th}$ batch's configuration and the mean request rates. This is true because the probability of a requester requesting is independent of its history provided it has no request pending. Any request pending at the start of the $(n-1)^{th}$ batch is reflected by the servicing of that request during the $(n-1)^{th}$ batch. Thus, all the information that determines (4.4) is contained in the states i and j_1. Hence it suffices to write $\text{prob}[S(n)=i \mid S(n-1)=j_1]$ as a shorthand for (4.4).

4.4 Derivation of State Transition Probabilities

In order to derive probabilities of state transitions, some preliminary atomic expressions are introduced from which final results can be expressed. Firstly, the probability that requester h does not request during the service time of requester ℓ is derived under the condition that requester h has no request pending already

$$\text{prob}\left[\begin{matrix}\text{requester } h \text{ does not} \\ \text{request during } t_s(\ell)\end{matrix} \;\middle|\; \text{Req.}h=0 \text{ before } t_s(\ell)\right] = \int_0^\infty f_\ell(t)e^{-\lambda_h t}dt \qquad (4.5)$$

Consider the case of requester h not requesting during three consecutive time intervals $t_s(\ell_1)$, $t_s(\ell_2)$ and D, given that Req. $h = 0$ before $t_s(\ell_1)$, and where the time duration D is constant. This is given by

$$\int_0^\infty \int_0^\infty f_{\ell_1}(t_1)f_{\ell_2}(t_2)e^{-\lambda_h(t_1+t_2+D)}dt_1dt_2 = e^{-\lambda_h D}\int_0^\infty f_{\ell_1}(t_1)e^{-\lambda_h t_1}dt_1 \int_0^\infty f_{\ell_2}(t_2)e^{-\lambda_h t_2}dt_2 \qquad (4.6)$$

The decomposition of (4.6) generalises to any number of time intervals. Consider now the probability of requester h not having a request pending at the end of D_1 in a batch with state $i \neq 0$. This probability is denoted by $Q(i,h)$. Two cases arise:

(i) requester h is serviced in the batch with state i;

(ii) requester h is not serviced ($h \notin i$); and hence Req. $h = 0$ at the start of the batch. It follows that

$$Q(i,h) = \begin{cases} e^{-\lambda_h D_1} \prod_{\substack{\ell \in i \\ \ell > h}} \int_0^\infty f_\ell(t)e^{-\lambda_h t}dt, & \text{if } h \in i \\ e^{-\lambda_h(D_1+D_2)} \prod_{\ell \in i} \int_0^\infty f_\ell(t)e^{-\lambda_h t}dt, & \text{if } h \notin i \end{cases} \qquad (4.7)$$

where

$$\int_0^\infty f_\ell(t)e^{-\lambda_h t}dt = \begin{cases} \frac{\mu_\ell}{\mu_\ell+\lambda_h}, & \text{exponentially distributed service time} \\ e^{-\lambda_h/\mu_\ell}, & \text{constant service times} \end{cases}$$

Fig. 6 illustrates the definition of $Q(i,h)$ when $h \in i$.

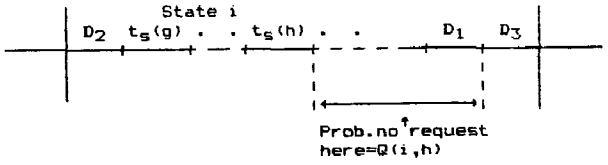


Figure 6: Definition of A(i,h), h∈i

The state transition probabilities are now derived for four cases:

Case 1: S(n-1) = S(n) = 0

$$\text{prob}\,[S(n)=0 \mid S(n-1)=0] = 0 \qquad (4.8)$$

This follows from the fact that a non-zero batch always precedes and follows a zero batch. The zero batch is defined to be the idle time between two non-zero batches, thus two consecutive zero batches cannot exist. Refer to Fig. 7.

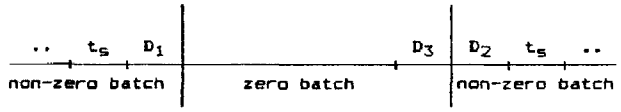


Figure 7: Zero batch between non-zero batches

Case 2: S(n-1) = j ≠ 0, S(n) = 0

$$\text{prob}\,[S(n)=0 \mid S(n-1)=j] = \prod_{h=1}^{k} Q(j,h) \qquad (4.9)$$

Equation (4.9) gives the probability of the intersection of the k independent events that each requester does not request during the $(n-1)^{th}$ batch and hence a zero batch follows.

Note that the time duration D_3 is not included in the $(n-1)^{th}$ batch because the n^{th} batch is the zero batch.

Case 3: S(n-1) = 0, S(n) = i ≠ 0.

prob [S(n)=i | S(n-1)=0] is required. Consider Fig. 8.

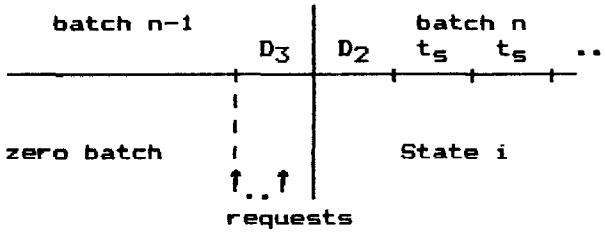


Figure 8: Zero Batch to non-zero batch transition.

For S(n)=i to follow S(n-1)=0, the first requester to end the zero batch by requesting must be in S(n). All the other requesters in S(n) must request within the following D_3 time period. But so that no more requesters request, the intersection must be taken with the event that all requesters not in S(n) do not request within D_3. Thus

$$\text{prob}\,[S(n)=i \mid S(n-1)=0]$$

$$= \sum_{h\in i} \left\{ \text{prob} \left(\begin{array}{l} \text{h is the first requester} \\ \text{to request during} \\ \text{S(n-1)=0.} \end{array} \,\middle|\, S(n-1)=0 \right) \right.$$

$$\left. \text{xprob} \left(\begin{array}{l} \text{remaining requesters } \in i \\ \text{request during } D_3 \end{array} \,\middle|\, S(n-1)=0 \right) \right\}$$

$$\text{xprob} \left(\begin{array}{l} \text{requesters } \notin i \text{ do not} \\ \text{request during } D_3 \end{array} \,\middle|\, S(n-1)=0 \right) \qquad (4.10)$$

Since the events of requesters requesting are mutually exclusive, the summation applies in (4.10). The terms in (4.10) are shown in [4] to give:

$$\text{prob}\left[S(n)=i \,\middle|\, S(n-1)=0\right] = \frac{\sum_{h\in i} \{\lambda_h \cdot \prod_{\substack{g\in i \\ g\neq h}} [1-e^{-\lambda_h D_3}]\} \cdot \prod_{f\notin i} e^{-\lambda_f D_3}}{\sum_{\ell=1}^{k} \lambda_\ell} \qquad (4.11)$$

Case 4: S(n-1) = j ≠ 0, S(n) = i ≠ 0

$$\text{prob}\,[S(n)=i \mid S(n-1)=j]$$

$$= \text{prob} \left(\begin{array}{l} \text{requesters } \in i \text{ request during} \\ \text{batch n-1 and at least} \\ \text{one of which before } D_3 \end{array} \,\middle|\, S(n-1)=j \right) \qquad (4.12)$$

$$\times \prod_{h\notin i} \text{prob} \left(\begin{array}{l} \text{h does not request} \\ \text{during batch n-1} \end{array} \,\middle|\, S(n-1)=j \right)$$

The reason for inclusion of the condition "at least one of which before D_3" is explained with the aid of Fig. 9.

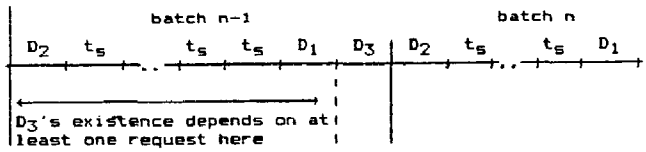


Figure 9 : Non-Zero to non-zero batch transition

If no request occurred up to the end of D_1, then D_3 would not be present in batch n-1 and a zero batch would follow.

The terms in (4.12) are derived in [4] to give:

$$\text{prob}\left[S(n)=i \,\middle|\, S(n-1)=j\right]$$

$$= \{ \prod_{h\in i} [1-Q(j,h)e^{-\lambda_h D_3}] - \prod_{h\in i} [Q(j,h)(1-e^{-\lambda_h D_3})] \} \prod_{h\in i} Q(j,h) e^{-\lambda_h D_3} \qquad (4.13)$$

Equations (4.8), (4.9), (4.11) ard (4.13) define the Markov transition matrix for the batch states of the system. Notice that in all cases prob[S(n)=i | S(n-1)=j] is independent of n. Hence the Markov chain is homogeneous. Define:

$$p_{ij} \triangleq \text{prob}[S(n)=i \mid S(n-1)=j] \quad i,j=0,\ldots,2^k-1 \qquad (4.14)$$

then the matrix P is defined by:

$$P \triangleq [p_{ij}] \qquad (4.15)$$

P is called the probability transition matrix.

5 LIMITING PROPERTIES OF THE PROBABILITY TRANSITION MATRIX

The probability transition matrix, P, is a non-negative matrix (all elements non-negative) with unity column sums. The probability vector of the n^{th} batch, p(n), represents with its i^{th} component, $p_i(n)$, the probability of the n^{th} batch being in state i. p(n) is given by:

$$p_i(n) = \sum_{j=0}^{m-1} \text{prob}\left[S(n)=i \mid S(n-1)=j\right] . p_j(n-1) = \sum_{j=0}^{m-1} p_{ij} . p_j(n-1) \qquad (5.1)$$

where $m \triangleq 2^k$. In matrix notation

$$p(n) = P.p(n-1) = P^n.p(0) \qquad (5.2)$$

where p(0) is the initial probability vector. Furthermore, p is defined as:

$$\begin{aligned} p &\triangleq \lim_{n\to\infty} p(n) \quad \text{(when the limit exists)} \\ &= \lim_{n\to\infty} P^n p(0) = P \lim_{n\to\infty} P^{n-1} p(0) = Pp \end{aligned} \qquad (5.3)$$

p is called the limiting probability vector of the probability transition matrix P. Note that (5.3) shows that p is an eigenvector of P corresponding to the eigenvalue of 1.

5.1 Principle Limiting Results as $n \to \infty$

Theorem 1: The probability transition matrix of the fixed priority batch arbiter model, P, has a unique positive limiting probability vector independent of the initial probability vector, provided $\mu_h, \lambda_h > 0$ for h=1, .., k and $D_1 + D_3 > 0$.

Proof: Firstly it is shown that P is irreducible and primitive [11] by showing that all elements of P^2 are positive (written $P^2 > 0$). This follows from the theorem:

P is irreducible and primitive if and only if some power of P is positive [9, p.801].

To show $P^2 = [p_{ij}^{(2)}]$ is positive, consider:

$$\begin{aligned} p_{ij}^{(2)} &= \sum_{u=0}^{m=1} p_{iu}p_{uj} \text{ , where } m=2^k \\ &\geq p_{iw}p_{wj}, \text{ some w, } 0\leq w\leq m-1 \end{aligned} \qquad (5.4)$$

since $p_{iu}p_{uj} \geq 0$ for all i, u, j.

So if, for every i and j, a w can be chosen such that $p_{iw}p_{wj} > 0$ then it follows that $P^2 > 0$. The state w corresponds to an intermediate state i and j as shown in Fig. 10.

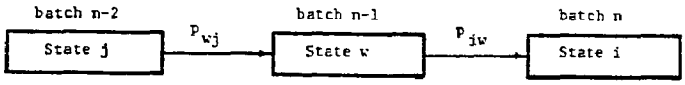


Figure 10: State transition illustrating proof of Theorem 1.

In [4] it is shown that such an intermediate state w can be chosen when $D_1 + D_3 > 0$.

□

When $D_1 = D_3 = 0$, the full batch can never occur because all transitions to it have zero probability (including the full batch to full batch transition). This implies that P is reducible in the case of $D_1 = D_3 = 0$, since P has an invariant coordinate subspace of dimension $2^k-1<2^k$ = dimension of P, consisting of all but the last state (see definition of reducibility in [9]).

Define the smaller matrix, P_1, as P with the last row and column removed (i.e. without full batch state)

$$P_1 \triangleq [p_{ij}] \quad i,j = 0, .., m-2 \qquad (5.5)$$

It is shown in [4] that P_1 has the same properties as P in theorem 1. Note that P_1 is a probability transition matrix (column sums are unity) only when $D_1=D_3=0$. The following theorem is proved in [4].

Theorem 2: The probability transition matrix P_1 as defined in (5.5) has a unique positive limiting probability vector independent of the initial probability vector provided $\mu_h, \lambda_h>0$ for h=1, ..., k and $D_1=D_3=0$.

5.2 Heavy Request Loading Limit of the Probability Transition Matrix.

In this subsection the limiting behavior of P as the request rates tend to infinity is examined. (The zero limit can be found in [4]). The manner in which the request rates tend to infinity is defined as follows:

The average request rate, λ, is defined:

$$\lambda \triangleq \frac{\sum_{f=1}^{k} \lambda_f}{k} \qquad (5.6)$$

and relative request rates, r_h, are defined:

$$r_h \triangleq \frac{\lambda_h}{\sum_{f=1}^{k} \lambda_f} \qquad h=1,2,\ldots,k \qquad (5.7)$$

The request rates are allowed to approach infinity keeping r_h of (5.7) constant for each requester. As request rates become large, requesters request

soon after releasing the resource. The limiting behavior depends on whether D_1+D_3 is zero. For $D_1+D_3>0$, the limiting probability vector is independent of the initial probability, with all states having zero probability except for the full batch [4]. That is, the arbiter services every requester in each batch.

When $D_1+D_3=0$, the full batch can never occur, as discussed in section 2 remark (vii), and the limiting behavior is more interesting. From (4.7)

$$\lim_{\lambda\to\infty} Q(i,h) = \begin{cases} 1, & \text{if } h\in i \text{ and } h\geq \ell \text{ for all } \ell\in i \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

and from (4.9) and (5.8)

$$\lim_{\lambda\to\infty} P_{0j} = 0 \qquad j = 0, 1 \ldots, m-1 \tag{5.9}$$

and from (4.12)

$$\lim_{\lambda\to\infty} P_{i0} = \begin{cases} r_f, & \text{if } i=2^{f-1} \text{ i.e. singleton state} \\ 0, & \text{otherwise} \end{cases} \tag{5.10}$$

For $i \neq 0$ and $j \neq 0$ (4.13) and (5.8) imply.

$$\lim_{\lambda\to\infty} P_{ij} = \begin{cases} 1, & \text{if } i \text{ contains all requesters except the lowest priority requester in state } j. \\ 0, & \text{otherwise} \end{cases} \tag{5.11}$$

The powers of the matrix P formed from (5.9), (5.10) and (5.11) cycle with a period of two after the third power [4]. The limiting probability vectors in the two cycle have all zero elements except for two states corresponding to the full batch without requester k and the full batch without requester k-1. The probabilities of these states exchange after each state transition and depend on the initial state. The physical interpretation of this result is that the arbiter oscillates between the two states, with the two lowest priority requesters k and k-1 receiving half the servicing that the others receive. This interesting situation can be seen later in the numerical results for heavy request loading and small values of D_1+D_3.

6 PERFORMANCE PARAMETERS

It has been shown that the Markov chain model for the arbiter has a unique limiting probability vector for finite positive request rates. After sufficient time, the probability of a particular batch occurring will be constant regardless of the initial state of the arbiter. From the batch probabilities in the steady state, many useful performance parameters of the arbiter can be determined which are functions of the request rates, service times, the total number of requesters, k, and interbatch times D_1, D_2 and D_3.

6.1 Utilisation

Suppose the system has reached steady state with probability vector

$$p = [p_0, p_1, p_2, \ldots, p_{m-1}]^T \tag{6.1}$$

Now sample n successive batches. Let $n_j (j=0, \ldots, m-1)$ be the number of states=j, and thus

$$n = \sum_{j=0}^{m-1} n_j \tag{6.2}$$

Let t_j be the mean length of time of a batch with state j, defined as follows:

$$t_j \triangleq \begin{cases} D_1+D_2+D_3+\sum\limits_{h\in j}\frac{1}{\mu_h}, & j=1,2,\ldots,m-1 \\ \dfrac{1}{\sum\limits_{h=1}^{k}\lambda_h}, & j=0 \end{cases} \tag{6.3}$$

The proportion of time devoted to the j state is given by

$$T_j^n = \frac{n_j t_j}{\sum\limits_{i=0}^{m-1} n_i t_i} \tag{6.4}$$

taking the limit as $n\to\infty$ of (6.4) gives

$$T_j \triangleq \lim_{n\to\infty} T_j^n = \frac{t_j \lim\limits_{n\to\infty}\frac{n_j}{n}}{\sum\limits_{i=0}^{m-1} t_i \lim\limits_{n\to\infty}\frac{n_i}{n}} \tag{6.5}$$

Equation (6.5) suggests defining the following utilisation parameter

$$U_j \triangleq \frac{t_j p_j}{\sum\limits_{i=0}^{m-1} t_i p_i} \tag{6.6}$$

Similarly the proportion of time spent servicing requester ℓ, $prop_\ell$, is defined by

$$prop_\ell = \frac{\sum\limits_{\substack{i \text{ such that} \\ \ell\in i}} \frac{p_i}{\mu_\ell}}{\sum\limits_{i=0}^{m-1} p_i t_i} \tag{6.7}$$

The idling time for the system is defined as the proportion of time not spent servicing requests

$$\text{idle time} \triangleq 1 - \sum_{\ell=1}^{k} prop_\ell \tag{6.8}$$

6.2 Mean waiting time for each requester.

The waiting time for requester h is the time from when requester h requests to when requester h receives the resource. In [4] an expression is derived for w(i,j,h), the mean waiting time for requester h given that S(n)=i follows S(n-1)=j where requester h is serviced during state i. The uncondition mean waiting time for requester h, MWT(h), can then be expressed:

$$MWT(h) = \lim_{n\to 0} \sum_{j=0}^{\infty} \sum_{\substack{i \text{ such} \\ \text{that } h\in i}} W(i,j,h)\ \text{prob}\left[S(n-1)=j,\ S(n)=i \,\middle|\, h\in S(n)\right]$$

$$= \sum_{j=0}^{m-1} \sum_{\substack{i \\ \text{such that} \\ h\in i}} \frac{W(i,j,h)\cdot p_{ij}\cdot p_j}{\sum_{\substack{\text{such that} \\ h\in u}} p_u} \qquad (6.9)$$

6.3 Heavy Request Loading Limit of Performance Parameters

Two cases arise: (i) $D_1 + D_3 > 0$ (ii) $D_1 + D_3 = 0$. The limit as request rates approach infinity with fixed ratios as defined in (5.7) is an idealisation of a heavily loaded request situation. In practice, batching arbiters have parameters D_1 and D_3 much smaller than mean service times and consequently the probability of requests occurring during D_1+D_3 at the end of a batch is quite small, even for large request rates. As a result the case of $D_1 + D_3 = 0$ approximates the behavior of most batching arbiters under heavy loading and will be considered in more detail here. In order that the heavy loading limit for $D_1 + D_3 > 0$ be borne out in practice, inconceivably large request rates must occur in most batching arbiters. As can be seen in [4], the case (i) limit gives the full batch occurring with probability 1 and limiting performance parameters can then be derived. For $D_1 + D_3 = 0$.

$$\lim_{\lambda\to\infty} p = \lim_{\lambda\to\infty} \lim_{n\to\infty} p(n) = \lim_{\lambda\to\infty} \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n)$$

$$= \lim_{N\to\infty} \lim_{\lambda\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \qquad (6.10)$$

$$= \left[0,\ 0,\ \ldots,\ \underbrace{\tfrac{1}{2},\ \tfrac{1}{2}}_{k-1},\ \underbrace{\tfrac{1}{2},\ 0}_{k}\right]^T$$

Where $\overline{k-1}$ is the state with all requesters serviced except k-1.

The steps in (6.10) are justified in [4], by showing

$$\lim_{n\to\infty} p(n) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \quad \text{and}$$

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \quad \text{converges uniformly.}$$

Equation (6.10) can be interpreted physically by observing that the arbiter alternates between the batch servicing all but the lowest priority requester and the batch servicing all but the second lowest priority requester. This occurs because the last requester serviced in a batch is unable to request in time to be included in the next batch.

It follows from (6.3), (6.7) and (6.10) that

$$\lim_{\lambda\to\infty} prop_h = \begin{cases} \dfrac{2}{\mu_h T_1}, & h\neq k \text{ and } j\neq k-1 \\ \dfrac{1}{\mu_h T_1}, & h=k \text{ or } h=k-1 \end{cases}$$

$$\text{where } T_1 = 2\{D_2 + \sum_{g=1}^{k-2} \frac{1}{\mu_g}\} + \frac{1}{\mu_{k-1}} + \frac{1}{\mu_k}$$

$$\text{and } D_1 + D_2 = 0 \qquad (6.11)$$

It follows from (6.9), [4] and (6.10) that

$$\lim_{\lambda\to\infty} MWT(h) = \begin{cases} \dfrac{T_1}{2} - \dfrac{1}{\mu_h}, & h\neq k \text{ and } h\neq k-1 \\ T_1 - \dfrac{1}{\mu_h}, & h=k \text{ or } h=k-1 \end{cases}$$

where T_1 is defined in (6.11) and $D_1 + D_3 = 0$ (6.12)

Note the unfairness in (6.11) and (6.12) for the two lowest priority requesters, who receive approximately half the servicing that other requesters receive.

7 COMPUTER STUDY AND NUMERICAL RESULTS

Much of the interesting and relevant behavior of batching arbiters occurs between the limiting extremes of light and heavy request loading. This behavior may best be examined using numerical computer techniques since the theoretical expressions are not easily intepreted for intermediate request rates.

7.1 Assumptions

The following simplyfying assumptions are made in the computer study

$$u_\ell = u \qquad (7.1)$$

$$\lambda_\ell = \lambda \qquad (7.2)$$

$$t_s(\ell) = \frac{1}{\mu} \qquad (7.3)$$

$$\ell = 1,2,\ldots,k$$

All requesters are assumed to have identical request and service statistical characteristics, with constant service times.

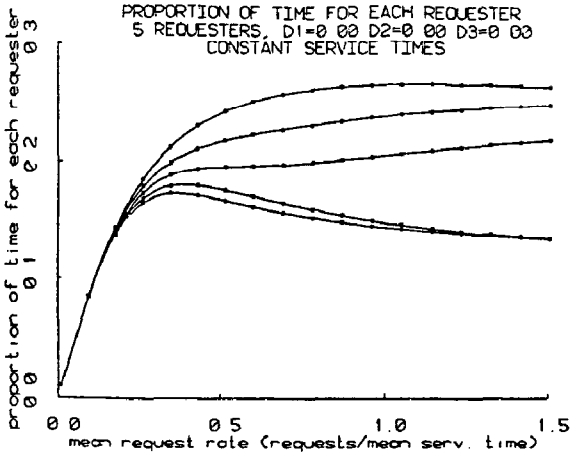
7.2 Description of Results

In graphs 1, 2 and 3 some results of the computer study are presented. The interbatch time durations D_1, D_2 and D_3 are expressed in units of service time and are selected to correspond approximately to an arbiter design such as that shown in section 3. For applications with large service times, for example in an arbiter for resolving multiple interrupts, D_1, D_2 and D_3 can be approximated as zero. For a bus arbiter, for example, much shorter service times are typical, and the value of 0.2 service times was chosen for that case.

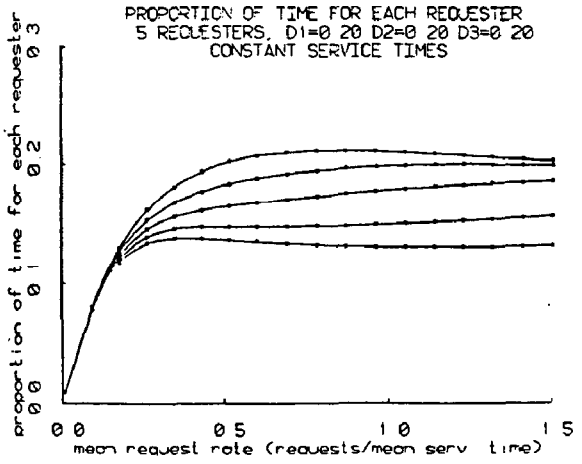
In Graph 1, the proportion of time allocated to each requester is plotted against the request rate, λ, in (7.2). Higher priority requesters receive more time than lower priority requesters. However for request rates below 0.25 requests/service

time, all requesters receive approximately equal time.

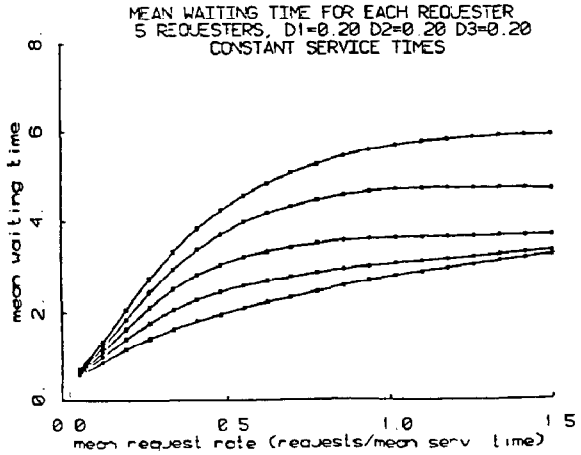
Note that for large request rates in Graph 1, the lowest priority requesters receive about half the time that the others receive as predicted by results in section 6.3. A physical explanation for this unfairness is that the lowest priority requester is serviced last when serviced in a batch, and thus cannot request in time to be included in the next batch. Then the second lowest priority requester is serviced last in this next batch and consequently suffers the same problem. For non-zero interbatch times, this effect is moderated, as seen in Graph 2.



Graph 1



Graph 2



Graph 3

8 CONCLUSION

This paper has presented theoretical and numerical results on the performance of a class of arbiters termed batching arbiters. The modelling and analysis presented has been shown to give results of practical interest for computer system designers.

In the utilisation results, an important feature of batching arbiters was highlighted: The two lowest priority requesters receive approximately half the servicing compared with other requests, under heavy loadings and small interbatch times. Another feature which may apply more generally than to batching arbiters, is that the "fairness" of the arbiter depended very much on the criterion for assessment. In terms of proportion of time each requester received, the arbiter was fair up to the saturation levels of requesting. However, in terms of relative mean waiting times for requesters, the arbiter was not fair for even moderate request loadings. Thus, for throughput applications the arbiter allocates the resource fairly but not for applications sensitive to access times of the requesters. In all cases batching arbiters have bounded waiting times independently of request loadings - a property fixed priority unbatched arbiters lack.

REFERENCES

[1] K.J. Thurber, E.D. Jensen, L.A. Jack, L.L. Kinney, P.C. Patton, L.C. Anderson, "A systematic approach to the design of digital bussing structures", *AFIPS Conference Proceedings*, Full Joint Conference, 1972, 41, part 11.

[2] J. Kriz, "A queueing analysis of a symmetric multiprocessor with shared memories and buses" *IEE Proc.*, Vol. 130, Pt E, No. 3, May 1983, pp 83-89.

[3] A.B. Kovaleski, "High-speed bus arbiter for multiprocessors", *IEE Proc.*, Vol. 130, Pt E, No. 2, March 1983, pp 49-56

[4] L. Kleeman and A. Cantoni, "A class of arbiters - structures and performance analysis", Tech. Report EE8421, Department of Electrical and Computer Engineering, University of Newcastle, Australia, June 1984.

[5] J. Theus, M. Taub and R.V. Ballakrishnan, "Futurebus anticipates coming needs", Electronics, July 12, 1984, pp 108-112.

[6] L. Kleeman and A. Cantoni, "Metastable Behaviour and digital system reliability", Tech. Report EE8441, Department of Electrical and Computer Engineering, University of Newcastle, Australia, Sept. 1984, also to appear IEEE Design and Test.

[7] L.R. Marino, "General theory of metastable operation", *IEEE Trans. Comput.*, Vol. C-30, February 1981, pp. 107-115.

[8] G. Coiffi, P. Velardi, "A fully distributed arbiter for multiprocessor systems" Microprocessing and Microprogramming 11, (1983), pp. 15-22.

[9] Gantmacher, Matrix Theory, Vol. II, N.Y. Chelsea Publishing Co., 1974.