Performance Evaluation of a Multi-Folder Ring Protocol for Total Ordering of Messages

Paul Ezhilchelvan*, Isi Mitrani* and Jim Webber†
*School of Computing, Newcastle University, NE4 5TG, UK
Email: [paul.ezhilchelvan, isi.mitrani]@ncl.ac.uk
†Neo4j UK, Union House, 182-194 Union Street, London, SE1 0LH
Email: jim.webber@neo4j.com

***Abstract*—In a system containing several distributed servers, messages of random sizes generated at different locations must be disseminated and processed in the same order by all hosts. A ring protocol is defined, where a number of folders carrying messages circulate in one direction without overtaking each other. A model involving parallel queues is analysed in the steady state and is solved approximately, allowing the computation of performance measures. A number of example systems are evaluated numerically and by simulations, leading to a heuristic for choosing the optimal number of folders.**

***Index Terms*—Server Replication, total order, logical ring, approximate analysis, parallel queues, bulk service.**

I. Introduction

Many distributed applications demand that cooperating processes view system events and messages in identical order [8]. Chief among these applications is replicated processing, a widely used method for building crash tolerant services [15]. A fundamental requirement in this context is that all requests (e.g. file updates), regardless of their origin, should be processed in the same order by all replicas. A client sends a message to any one of the replicated servers, which then disseminates it to all other servers so that updates everywhere are performed in a mutually consistent manner. An example of such a replicated service used in many practical applications is Apache Zookeeper [7].

There is an extensive literature on ordering protocols (see [2] for a survey). Most implementations, including the ordering protocol of Zookeeper [6] and RAFT [14], take a centralised approach. One replica is designated as the leader to whom all others send their messages for ordering. This involves two multicasts by the leader, which are typically implemented by means of multiple unicasts, one for each destination. The resulting network traffic, and the load placed on the leader, are considerable [3].

More recently, interest has focused on high throughput and scalable data processing (even at the expense of latency). The major constraint in achieving this appears to be the network capacity [18]. Consequently, alternatives to multiple unicasting have been explored and the prominent approach is to arrange server replicas in a ring. Each server sends messages only to its neighbour in one direction, and receives them only from its neighbour in the other direction. The resulting structure is a leader-free, decentralised ring. It is argued in [5] that the message ordering over such a ring can offer scalable throughput which is less affected by the network capacity constraints discussed in [18].

In this paper, we propose a ring protocol which handles messages of random sizes and allows a server to disseminate them in random batches. Several folders circulate continuously, visiting the servers in turn without overtaking each other. Every folder contains a set of dedicated storage blocks of fixed size, where each server loads the messages it wishes to pass on. The details of the protocol, and the proof that it achieves safe and consistent total order, are described in the next section.

We analyse the stochastic behaviour of the multi-folder protocol and provide an approximate solution that allows us to compute performance measures. Such an analysis does not appear to have been attempted before. Other studies have focused on the mechanics of message transport and delivery. The term 'latency' has been used to denote the interval between a server sending a message, and its delivery. We are interested in the response time, i.e. the interval between the arrival of a message into the system and its delivery. That is a sum of queueing delay at a server, plus a latency. It turns out that there is an interesting trade-off between queuing delays and latency, which influences the optimal number of folders that should be employed.

A. Related work

A ring protocol called LCR was proposed and implemented by Guerraoui et al, [5]. It ensures total order by maintaining vectors of logical clocks, whereby each station keeps track of messages sent by other stations. That set-up requires fairness control, in order to prevent a heavily loaded server from giving priority to its own messages at the expense of those sent to it for forwarding. This issue was recognised and addressed in [5] by means of a rather complex algorithm.

A different ring structure called TRAINS was introduced in Simatic et al, [16]. Its trains and wagons have similar functions to our folders and blocks. Like the LCR protocol, TRAINS requires a special algorithm for controlling and balancing the flow of messages. In our case, the problem is solved by statically fixing the block sizes allocated to each station.

The above studies deal with throughput and latency, but are not concerned with the user experience exemplified by the average response time. That performance measure was evaluated approximately by Liu et al, [10], in the case of a ring with a single folder, and messages of fixed size. The question of optimal number of folders did not arise for that system.

A circulating folder resembles a 'polling server' which visits a number of queues in sequence. While there is a large volume of work on polling systems (see [17] for a good survey), existing results do not apply to our model even in the case of a single folder. The reason is that a visit time at a host depends on the current number of non-empty blocks, i.e. not only on the queue being visited, but also on all other queues.

Another, more distantly related work is Marandi et al, [11], where a combination of a leader and a ring is proposed. There is also a class of protocols where a logical ring is used for rotating the leadership role among servers (Defago et al, [2]).

An issue that we do not consider here relates to the possibility of server crashes. After a breakdown, the ring structure would need to be reconfigured and new folders must be initialised. During this recovery process, servers would communicate with each other in the normal manner and can use any of the algorithms proposed in [5], [14], [13] and [9]. Recovery would be possible, as long as no more than one server crashes between successive reconfigurations. For our study, we assume that all servers are reliable.

The model is described in section 2. Section 3 develops the approximate solution, while section 4 presents several examples and addresses the optimisation of the number of folders.

II. The model

The system contains N service hosts, numbered 1, 2, ..., N. Host i has a separate FIFO queue, Q_i, for incoming messages. Hosts communicate with each other by means of M 'folders', numbered 1, 2, ..., M, carrying messages. The folders travel in one direction only: from host 1 to host 2, ..., from host N to host 1. There is no overtaking, so that at each host, folder $m+1$ arrives after folder m ($m < M$), and folder 1 after folder M. If more than one folders gather at a host, they are queued and processed in order of arrival.

Each folder consists of N 'blocks', numbered 1, 2, ..., N, where block i is permanently dedicated to the transport of messages sent by station i. All blocks have the same integer size, B. Messages have random integer sizes on the interval $[1, B]$. Hence, each block carries a random number of messages which does not exceed B. The message size distribution plays an important role in the subsequent analysis.

When any folder, m, comes to host i, messages from Q_i are loaded into block i, in order of their arrival at host i, as long as their total size does not exceed B. The current contents of the folder, i.e. both the newly filled block i and the blocks loaded by the other hosts, are copied and stored locally, until they are delivered to the application. The purpose of that local store is to enable the handling algorithm to deliver all messages in the same order at all hosts, regardless of their origin. In the meantime, when folder m eventually returns to host i, a new batch of messages from Q_i is loaded into block i.

The passage of messages through the system is illustrated in Figure 1, which focuses on one of the N queues. The period between leaving the queue and departing from the system, having been transported by a folder and eventually delivered at all hosts, is represented in the diagram by the place marked 'waiting'. This period is often referred to as 'latency' in the literature. Note that the latency depends on how long folders take to visit all the hosts. Those intervals will be referred to as 'cycles'. Moreover, we shall see that the latency depends on the index of the origin host. However, it does not involve queueing. The period between a message arriving into, and departing from the system is its response time.

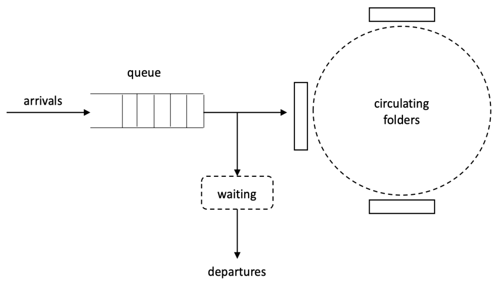

Fig. 1. The passage of messages through the system

The algorithm for transporting and delivering messages is governed by the following rules.

1) As folders circulate among the hosts, they are identified by their index, m, and also by the round, in which they are currently engaged. At time 0, a certain designated host, say host 1, generates the M folders and launches them circulating with round numbers, r, set to 1. Thereafter, every time folder m visits host 1, its round number is incremented by 1.
2) Each block carried by a folder is labeled with the round, r, during which it was loaded. Each block stored in a local store is labeled with a pair of integers, (m, r), where m is the index of the folder that carried it and r is the round number that the folder had *when the block was loaded.*
3) As soon as a host is visited by folder m with round number $r+1$, all messages in blocks labeled (m, r) in the local store are delivered. At host 1, the incrementing of r and the resulting deliveries take place before any other processing. All deliveries are performed in order of block index (i.e. host of origin) and within each block, in the order in which messages were loaded (i.e., order of arrival at the host of origin). If folder m starts round $r+1$ while still carrying blocks labeled r, such blocks are delivered to the application immediately.

Rule 3, together with the non-overtaking regime for folder circulation, implies that at all hosts, blocks are delivered in label sequence $(1, r)$,$(2, r)$,...,(M, r) (those are blocks loaded during round r), followed by $(1, r+1)$,$(2, r+1)$,..., $(M, r+1)$

(blocks loaded during round $r+1$), for all $r = 1, 2, \ldots$. Moreover, blocks with the same label, and messages within those blocks, are also delivered in a fixed prescribed order. Thus the algorithm achieves its purpose: all messages, regardless of origin, are delivered in the same order at all hosts.

Since this system is intended to model replicated servers, it is reasonable to assume that the stochastic behaviour of all hosts is the same. In particular, the time taken by any host, i, to process a visiting folder depends on the occupancy of the N blocks, but does not depend on i. The loading operation into block i takes an average time α if there is at least one message to be loaded, and time 0 if Q_i is empty. The copying of each of the other blocks also takes an average time α if that block is non-empty, and time 0 otherwise. Thus, if K of the blocks in the folder require processing when it visits host i, the average processing time, b, would be $b = K\alpha$. The maximum average processing time, $\bar{b} = N\alpha$, is realised when all blocks are non-empty.

Clearly, the processing time of any folder on a visit to any host depends not only on the current state of the corresponding queue, but also on the previous history of all queues and all folders. This complex interdependency between the different queueing processes and the movements of folders implies that an exact analysis of the system is intractable. We therefore propose an approximate solution that will nevertheless enable us to compute reasonably accurate estimates of performance measures.

Assume that messages requiring transmission arrive at any host, i, according to an independent Poisson process with rate λ or, equivalently, that they arrive into the system in a Poisson process with rate $N\lambda$ and are then directed to any host with probability $1/N$. The sizes of incoming messages at all hosts are discrete random variables, independent of each other and taking value j with probability $g(j)$, $j = 1, 2, \ldots, B$.

A. Stability condition

Denote by v the average 'visit interval' of a folder at a host. The visit interval may include waiting for other folders currently at that host to complete their processing, plus the processing of this folder. More precisely, if y is the average number of other folders present, and assuming that their processing times are i.i.d. random variables distributed exponentially with mean b, we can write

$$v = (y+1)b \, . \tag{1}$$

The circulating folders may be treated as a closed queueing network where the N hosts are the nodes and the M folders are the customers. According to the Arrival Theorem (see [12]), the number of other folders encountered at a host by a visiting folder has the same distribution as the steady-state number at that host, but in a network with $M-1$ circulating folders. Since those numbers have the same distribution at all hosts, we conclude that the average number of other folders encountered at a host by a visiting folder is $y = (M-1)/N$. Hence, the maximum average visit interval is

$$\bar{v} = \frac{(N+M-1)\bar{b}}{N} = (N+M-1)\alpha \, . \tag{2}$$

The interval between two consecutive visits by any given folder to the same host has already been referred to as a cycle. The average length of a cycle reaches a maximum, $\bar{T}$, when all blocks in the folder remain non-empty during the cycle. That maximum average length is given by

$$\bar{T} = N(N+M-1)\alpha \, . \tag{3}$$

Here we have assumed that the travel times of folders between hosts are 0. Non-zero travel times could be included in the closed folder network, but that would not produce significantly more accurate results because in practice travel tmes are orders of magnitude smaller than processing times.

We need to determine the distribution of the number of messages that may be loaded into a block of size B. Denote by $G_k(j)$ the probability that the total size of k messages does not exceed j, for $j = k, k+1, \ldots, B$. The probability that at least k messages would fit in the block is then given by $G_k(B)$.

When $k = 1$, the probability that the size of a single message does not exceed j is equal to

$$G_1(j) = \sum_{s=1}^{j} g(s) \; , \; j = 1, 2, \ldots, B \; , \tag{4}$$

with $G_1(B) = 1$. For $k > 1$, the corresponding probabilities are obtained by applying the convolution recurrences, splitting the k messages into two groups containing one and $k-1$ messages respectively, and remembering that the total size of $k-1$ messages is at least $k-1$:

$$G_k(j) = \sum_{s=1}^{j-k+1} g(s) G_{k-1}(j-s) \; , \; j = k, k+1, \ldots, B \, . \tag{5}$$

In addition to these recurrences, our assumptions imply that $G_k(j) = 0$ when $j < k$, since the total size of k messages cannot be less than k. In particular, $G_k(B) = 0$ when $k > B$.

If there are more than k messages available for loading when a folder arrives, then exactly k of them are loaded when the total size of the first k messages does not exceed B, but the total size of the first $k+1$ messages does. The probability of that event, r_k, is

$$r_k = G_k(B) - G_{k+1}(B) \; , \; k = 1, 2, \ldots, B \; , \tag{6}$$

The average number of messages loaded into the block depends, in general, on the current state of the queue. It may also depend on past history. If at the previous visit the queue was not emptied, and the appropriate block was not completely filled, the first message that failed to load must have been too big to fit into whatever space remained in the block. Consequently, the distribution of the message left at the head of the queue may be skewed towards the larger sizes. This phenomenon will be referred to as the 'historical dependency'.

As it is difficult to model, and manifests itself only when the system is very heavily loaded, its effect will be evaluated by simulation.

For the purposes of approximation, we shall ignore the historical dependency and assume that when a folder finds more than k messages present in a queue, the probability that exactly k of them are are loaded is given by (6). Those probabilities cease to depend on the queue size when there are at least B messages present.

Let d be the average number of messages loaded into a visiting folder when the queue size is at least B:

$$d = \sum_{k=1}^{B} k r_k \; . \tag{7}$$

After substituting (6), this becomes

$$d = \sum_{k=1}^{B} G_k(B) \; . \tag{8}$$

Under the above assumptions, we can estimate the condition for stability of the system. Note that when all queues are long, the average cycle of any folder with respect to a particular host has its maximum value, $\bar{T}$, given by (3). During a cycle, that host is visited by all M folders and on each visit, an average of d messages are removed from its queue. Hence, a long queue would experience a downward drift, ensuring stability, if the average number of messages that arrive during a maximum cycle is lower than the corresponding number of departures. The approximate stability condition can thus be written as

$$\lambda < \frac{Md}{N(N+M-1)\alpha} \; . \tag{9}$$

The same condition would apply to all queues, i.e. to the system as a whole. In the analysis that follows, it is assumed that the inequality (9) is satisfied.

III. Approximate solution

As indicated in Figure 1, the residence of a message in the system consists of two consecutive phases: the first, referred to as the 'queueing phase', is the sojourn in the input queue until loaded into a folder; the second, which we shall call 'delivery phase', consists of the total interval during which copies of the message remain in various waiting rooms while that folder circulates among the hosts, until all deliveries are completed. The queueing phase requires a congestion analysis, while the delivery phase involves an approximation of delivery periods. Intuitively, there is a trade-off between the two phases. Increasing the number of folders in circulation tends to increase the service rate in the right-hand side of (9), which reduces queuing. At the same time, cycles would tend to become longer, according to (3), which increases the delivery phase. The two phases will be addressed separately.

A. Queueing phase

The idea is to treat host i in isolation, with its queue evolving in a stationary environment defined by the other hosts. The resulting solution is then used in order to recalibrate the environment.

The service mechanism at Q_i is known as 'bulk services'. There is a sequence of service instants, when several messages leave the queue simultaneously. These moments correspond to the visits of folders to station i. The rate at which they occur, and the number of departures that take place, are modulated by the environment and depend on the number of messages present.

Consider the service instants when folder 1 visits host i and finds Q_i not empty. Denote by S the steady-state average number of non-empty blocks, other than block i, during the resulting folder 1 cycles. The constant S, which is yet to be determined, defines the environment in which Q_i evolves. The average length of a folder 1 cycle, $T(S)$, where block i and an average of S other blocks require processing at every station, is given by an expression similar to (3) but involving the environment S:

$$T(S) = (1+S)(N+M-1)\alpha \; . \tag{10}$$

During such a cycle, host i is visited by all folders. Hence, we can reasonably estimate the average interval between consecutive visits by a folder to host i as $T(S)/M$. Our approximation is based on assuming that those intervals are i.i.d. random variables distributed exponentially. In other words, host i is modelled as an M/M/1 queue with bulk services where, for a given environment S, the service instants occur at rate

$$\mu(S) = \frac{M}{T(S)} \; . \tag{11}$$

The number of messages that depart simultaneously from the queue at each service instant is described by the distribution $G_k(B)$, obtained from (5).

Let π_j be the steady-state probability that there are j messages in Q_i, $j = 0, 1, \ldots$. A set of balance equations for these probabilities can be derived as follows.

For each $j = 0, 1, \ldots$, divide the set of all possible states into two groups: group 1, where there are j or fewer messages present and group 2, where there are $j+1$ or more messages present. In the steady state, the rate of upward transitions between them, from group 1 to group 2 (i.e. the average number of such transitions per unit time), must be equal to the rate of downward transitions, from group 2 to group 1. Upward transition occur when there are j messages present and a new message arrives. Downward transitions occur when there are $j+k$ messages present, for $k = 1, 2, \ldots, B$, a service instant occurs and at least k messages depart, which happens with probability $G_k(B)$. These transitions are illustrated in Figure 2.

The resulting balance equations are

$$\lambda \pi_j = \mu(S) \sum_{k=1}^{B} G_k(B) \pi_{j+k} \;\; ; \;\; j = 0, 1, \ldots \; . \tag{12}$$

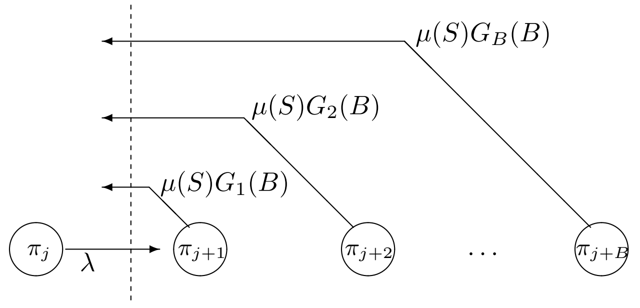

Fig. 2. Upward and downward transitions between groups 1 and 2

It is known that, if the departure batch sizes are fixed, the steady-state distribution of a queue with bulk services is geometric (e.g., see [1]). In the present model the batch sizes are random, but we still find a geometric distribution. Specifically, we can find a solution to equations (12) of the form

$$\pi_j = Cz_0^j \;\; ; \;\; j = 0, 1, \dots \; , \tag{13}$$

where C and z_0 are some positive constants. Indeed, substituting (13) into (12), we find that the equations are satisfied as long as z_0 is a zero of the polynomial of degree B

$$P(z) = \lambda - \mu(S) \sum_{k=1}^{B} G_k(B) z^k \; . \tag{14}$$

In addition, in order that we may obtain a probability distribution, z_0 must satisfy $|z_0| < 1$.

Note that $P(0) = \lambda > 0$. When $z = 1$, we have

$$P(1) = \lambda - \mu(S) \sum_{k=1}^{B} G_k(B) = \lambda - \mu(S)d < 0 \; , \tag{15}$$

according to (8) and the stability condition (9). Remember that $T(S) \le \bar{T}$, since $S \le N - 1$.

Hence, $P(z)$ has a real zero, z_0, in the open interval $(0, 1)$. The normalized solution (13) becomes

$$\pi_j = (1 - z_0) z_0^j \;\; ; \;\; j = 0, 1, \dots \; . \tag{16}$$

The existence of a positive normalizeable solution to the balance equations implies that this irreducible and aperiodic queueing process is ergodic. Therefore, the distribution (16) is the unique solution (the fact that $P(z)$ has no other zeros in the interior of the unit disk can also be proved directly, by using Rouche's theorem).

The above distribution of the isolated queue Q_i depends on the environment S via the value of z_0, which depends on $T(S)$. To emphasise this fact, let us write z_0 as a function of S: $z_0 = z_0(S)$. Note that, according to (16), $z_0(S)$ is the probability that the queue is not empty, which is also the probability that block i in any folder is not empty. Since all N queues are statistically identical, the average number of non-empty blocks belonging to the other $N - 1$ hosts is $(N - 1)z_0(S)$. Hence, S must be the fixed point of the equation

$$S = (N - 1)z_0(S) \; . \tag{17}$$

To show that such a fixed point exists and is unique, we make the following two observations:

1) At the two end points of the interval $[0, N - 1]$, we have $z_0(0) > 0$ and $z_0(N - 1) < 1$. This follows from the existence of the geometric distribution.
2) The function $z_0(S)$ is concave. This can be seen by setting $z = z_0(S)$ in (14), so the left-hand side becomes 0, and differentiating with respect to S twice. The first derivative of $z_0(S)$ is positive, i.e. it is an increasing function of S; the second is negative.

Observation 1 implies that the functions in the left-hand side and right-hand side of (17) intersect at least once between $S = 0$ and $S = N - 1$. Observation 2 shows that they cannot intersect more than once.

The fixed point of (17), $\tilde{S}$, can be computed by various algorithms. A simple approach is to start with some initial estimate, say $S_0 = N - 1$, and evaluate successive iterations

$$S_{n+1} = (N - 1)z_0(S_n) \; , \tag{18}$$

until two consecutive iterates are sufficiently close to each other. These iterations are guaranteed to converge to $\tilde{S}$.

Having computed the fixed point, the average number of messages in any queue, L_q, is obtained from

$$L_q = \frac{z_0(\tilde{S})}{1 - z_0(\tilde{S})} \; . \tag{19}$$

The average sojourn time of a message in queue Q_i, w_q, is also independent of i and is given by Little's result:

$$w_q = \frac{Lq}{\lambda} \; . \tag{20}$$

B. Delivery phase

The average time spent in the delivery phase by a message arriving at host i, w_i, will be approximated by expressing it in terms of the average cycle time of the folder where the message is loaded. It turns out that, despite the equal traffic parameters at all hosts, w_i depends on i.

Consider a block originating at host i and loaded into folder m in round r. That block is stored in the local stores of hosts i, $i + 1$, ..., N, with label (m, r). When folder m reaches host 1, the round number becomes $r + 1$. According to rule 3, hosts 1, 2, ..., $i - 1$ deliver the block, without needing to store it, as soon as the folder reaches them. The return of the folder to host i completes a cycle during which there have been deliveries at hosts 1, 2, ..., $i - 1$. The repeated visits to hosts i, $i + 1$, ..., N form part of the next cycle and result in delivering the locally stored copies of the (m, r) block.

The delivery period, w_i, of a block loaded by host i into folder m is the interval between the loading instant and the last delivery. The latter occurs when folder m reaches host N again, during its round $r + 1$. The resulting interval includes one full i-cycle of folder m, plus $(N - i)$ 'hops' forming part of its next cycle. A hop consists of the processing time of the folder at one host.

To approximate the average delivery period we assume that the average steady-state length of any cycle is given by

(10), with $S = \tilde{S}$ computed as the solution of the fixed-point equation (17). Assume also that the average steady-state lengths of all hops are equal, so that $(N-i)$ hops take a fraction $(N-i)/N$ of a cycle. We can then write

$$w_i = T(\tilde{S}) + \frac{N-i}{N} T(\tilde{S}) \ ; \ i = 1, 2, \ldots, N \ . \tag{21}$$

Messages originating at host N have the shortest average delivery phases, while those originating at host 1 have the longest. On the other hand, host 1 is the first host to deliver all messages of blocks (m, r) as soon as it increases the round number of folder m from r to $r+1$ (see Rule 3 of Section 2).

Note that when host 1 delivers messages of blocks (m, r), host N also has a copy of all delivered messages in its waiting room; host $N-1$ has all of them except those originating at host N; host i has all except those originating at hosts N, $N-1$, …, $i+1$. Therefore, even if host 1 crashes immediately after delivering (m, r) messages, after reconfiguration there will be at least one other host that can supply all delivered messages to hosts that have not received them. Thus, the protocol tolerates one host crash between successive reconfigurations and satisfies the well-known reliability criterion that any message delivered by a crashed host will also be delivered by all operative hosts.

The protocol can be generalised to tolerate f simultaneous host crashes, for $1 < f < N$. Such a generalisation would require that any message delivered by any host must have been seen by at least f other hosts. Without going into details, this can be achieved by adding $f-1$ extra hops to Rule 3 in Section 2, and hence to the delivery period in equation (21). The resulting generalised result becomes

$$w_i = T(\tilde{S}) + \frac{N+f-i-1}{N} T(\tilde{S}) \ ; \ i = 1, 2, \ldots, N \ . \tag{22}$$

The total average response time of a message submitted at host i, W_i, is the interval between its arrival into the system and its departure from the system. That average is approximated as

$$W_i = \frac{L_q}{\lambda} + w_i \ ; \ i = 1, 2, \ldots, N \ , \tag{23}$$

where L_q and w_i are given by (19) and (21). The total average number of messages submitted at host i and present in the system at any time in the steady state, L_i, is

$$L_i = L_q + \lambda w_i \ ; \ i = 1, 2, \ldots, N \ , \tag{24}$$

Both these performance measures depend on the host of origin, i. A simple way of avoiding that dependency is to slightly overestimate the average delivery period at hosts other than 1, and use w_1 for all messages. The resulting estimate for the overall average response time will be denoted by W and will be used as the performance measure in the numerical examples.

IV. Numerical and simulation results

We have experimented with different systems where all hosts have the same parameters. The block size is fixed at $B = 10$ and message sizes have three possible sizes: 1 with probability 0.5, 2 with probability 0.3 and 3 with probability 0.2. The average number of messages that may fit in a block, if plenty are available, is roughly $d = 5.8$. The average block processing time is also fixed, at $\alpha = 10^{-3}$ secs. The arrival rate is varied, as are the number of hosts and the number of circulating folders. The average response time, W, is approximated as described in the previous section, and is also evaluated by simulation, for purposes of comparison.

In the first example, a single folder circulates among 5 replicated hosts. The approximated and simulated values of the average response time W are plotted against λ, the arrival rate of messages per station. For this example, the inequality (9) suggests that the queues become unstable when $\lambda > 231$. The range we have chosen, from $\lambda = 180$ to $\lambda = 220$, implies an increase in the offered load from about 80% to 95%. Over this range, the response times are dominated by the queueing phase.

Each simulated point is the result of a run during which two million messages arrive into the system. The simulation assumes that the processing time of a folder containing K non-empty blocks is distributed exponentially with mean Ka. Messages with the given distribution are loaded into the appropriate block until either the queue is emptied or the next message would not fit. In the latter case, the size of the rejected message is remembered and is used when loading at the next folder visit. Thus the simulation keeps track of the historical dependency.

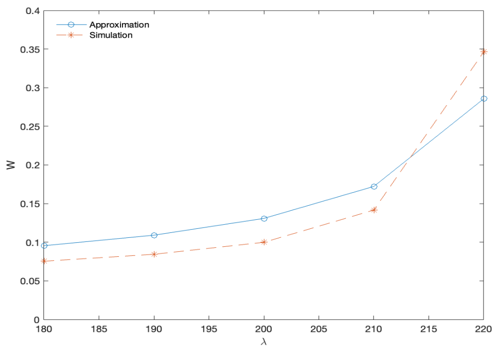

Fig. 3. Average response time for different arrival rates
$N = 5$, $M = 1$, $\alpha = 0.001$

The results are shown in Figure 3. The approximated and simulated values of W are seen to be reasonably close over the entire range of offered loads. However, it is notable that the approximation overestimates the average response time while the queue is less than 90% loaded, and underestimates it when the load reaches 95%. There are two factors which may explain this. In the case of a single folder, the simulated intervals between visits to a given host are sums of N exponentially distributed random variables with mean $T(S)/N$ each. On the other hand, the approximation assumes an exponentially distributed interval between visits, with mean $T(S)$. Thus the approximated interval has a larger coefficient of variation than the simulated one, and larger coefficients of variation are known to cause larger queueing times.

The second factor is the historical dependency, whose effect becomes significant at the 95% load. The slight decrease in the average number of messages fitting in a block leads to an increase in the queueing time that dominates the variance factor.

It is perhaps worth mentioning that the approximation plot was very quick to compute, taking a few seconds, while the simulation one was rather slow, taking tens of minutes. If the simulation runs were repeated or enlarged several times for the purpose of computing confidence intervals, they would take even longer.

In the next example, the arrival rate at each queue is kept fixed at $\lambda = 220$ and the number of folders is varied. The other parameters are as before. Remember that, when $M = 1$, all queues are heavily loaded at that arrival rate. We may guess that, as M increases, and hence the frequency of folder visits to any host increases, the average queue sizes and average queuing times would decrease. The delivery phase would be expected to increase due to the longer cycle times.

In Figure 4, the approximated and simulated average response times are plotted against the number of folders, which increases from $M = 1$ to $M = 8$.

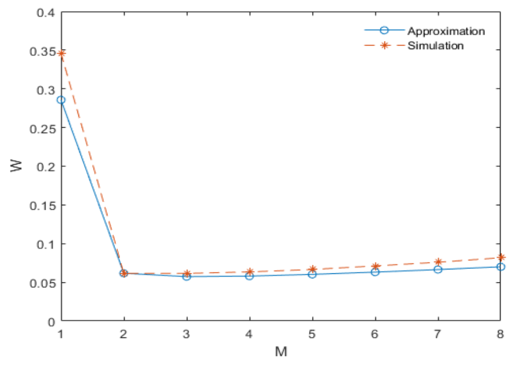

Fig. 4. Average response time for different numbers of folders
$N = 5$, $\alpha = 0.001$, $\lambda = 220$

Again, the two plots follow each other closely. We observe a steep decrease in response time, followed by an almost flat portion and then a gradual increase. Changing the number of folders from 1 to 2 effectively doubles the service capacity and reduces the average queuing time by about a factor of 6. Further additional folders have very little effect on the queues, but tend to increase the average cycle times and hence the delivery component of W. Eventually the response time starts to increase, slowly.

In this example, both the approximation and the simulation find the optimal number of processors to be $M = 3$, although there is very little difference between the minimum response times at $M = 3$ and its neighbours at $M = 2$ and $M = 4$.

We carried out the same experiment at a lighter, 50% load, $\lambda = 115$. The results were predictable: there is a slight drop in response time with the increment from $M = 1$ to $M = 2$, after which the plots look like the ones in Figure 4.

Less predictable is the behaviour of the system at higher arrival rates, requiring more than one folder in order to ensure stability of the queues. This is illustrated in the next experiment, where the same system is subjected to increasingly heavy message traffic. Four values of λ were chosen so that the minimum numbers of folders necessary for stability, as determined by (9), are 2, 3, 4 and 5, respectively. At those minimum numbers, all queues are 95% loaded. In Figure 5, the approximated average response times are again plotted against the number of folders (the simulated plots do not provide new insights and are omitted).

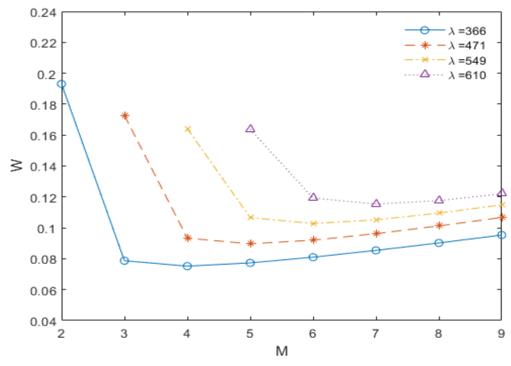

Fig. 5. Different numbers of folders and different arrival rates
$N = 5$, $\alpha = 0.001$

It is noticeable that, as the offered load increases, the decreasing portion of the plot becomes less steep and the minimum average response time grows. When several folders are required for stability, the queuing reduction produced by each additional folder diminishes, while the average cycle length, and hence the delivery period, increases. Clearly, there is a limit on the arrival rate that the system can cope with. Letting $M \to \infty$ in (9), we see that the right-hand side approaches $d/(Na)$, which would be the departure rate from each queue if there were no gaps between folders, with all blocks non-empty. If λ exceeds that quantity, the system would be unstable regardless of the number of folders employed.

On the basis of the experiments carried out so far, we propose the following ‘rule of thumb’ for choosing the number of folders in heavily loaded systems.

Optimisation Heuristic. Let M^* be the minimum number of folders required for stability, computed according to (9). Using $M^* + 2$ folders would minimise, or nearly minimise, the average response time.

This heuristic works for all examples in figures 4 and 5. Of course, in lightly loaded systems, $M = 1$ may be sufficient and close to optimal.

In practice, the number of replicated servers would be quite small, on the order of $N = 2$ or $N = 3$. The main implication for the behaviour of such systems is that the maximum average cycle length is small and therefore higher arrival rates can be supported. This is illustrated in our last example, where the approximated value of W is plotted against M, for $N = 2$ and $N = 3$, and two arrival rates for which the queues are 95% loaded at the corresponding minimum number of folders,1 and 2, respectively.

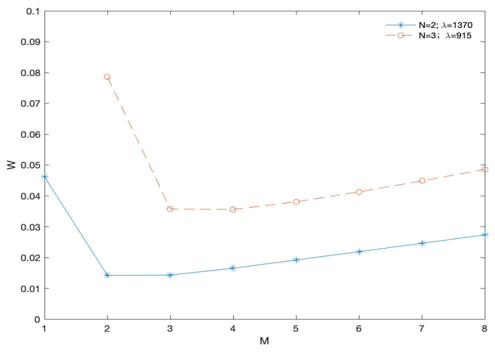

Fig. 6. $N = 2$ and $N = 3$; different numbers of folders and arrival rates $\alpha = 0.001$

In Figure 6, we observe similarly shaped plots to the ones in Figure 5. The average response times are smaller, which is not really surprising. It is also encouraging to find that the optimisation heuristic suggested above still works.

V. Conclusion

We have defined a multi-folder ring protocol which ensures that messages are delivered to a number of replicated servers in the same order. A queueing model of the system was analysed and solved approximately. The results obtained are sufficiently accurate to yield valuable insights into system behaviour. In particular, the numerical experiments suggest a heuristic for choosing the number of folders when the queues are heavily loaded.

It may be of interest to consider systems where arrival rates, and possibly message size distributions, are different at different hosts. The fixed-point approach could be generalised by defining the environment of an isolated queue as a vector of occupancies, instead of a single number. That would require a more complicated sequence of iterations whose convergence would be less predictable.

Another open topic for future work is the historical dependence which begins to affect performance at very heavy loads. Rather than ignoring that effect, it may be possible to approximate it. The difficulty of introducing the dependency into the model is that the isolated queue would no longer be a Markov process. Additional approximate assumptions would be needed in order to take it into account. Nevertheless, the effort would be worth it, if it leads to a more accurate sufficient condition for stability.

References

[1] N.T.J. Bailey, “On Queueing Processes with Bulk Service”, Journal of the Royal Statistical Society, Series B, 16, 1, pp. 80-87, 1954.

[2] X. Défago, A. Schiper and P. Urbán, “Total order broadcast and multicast algorithms: Taxonomy and survey”, ACM Computing Surveys (CSUR), 36, 4, pp. 372-421, 2004.

[3] A. Ejem and P. Ezhilchevan, “Design and Performance Evaluation of Raft Variations”, 39 th Annual UK Performance Engineering Workshop, 2023.

[4] P. Fouto, N. Preguiça and J. Leitão, “High throughput replication with integrated membership management”, 2022 USENIX Annual Technical Conference (USENIX ATC 22), pp. 575-592, 2022.

[5] R. Guerraoui, R. Ron, B. Pochon and V. Quéma, “Throughput optimal total order broadcast for cluster environments”, ACM Transactions on Computer Systems (TOCS), 28, 2, pp. 1-32, 2010.

[6] F. Junqueira, B. Reed and M. Serafini, “Zab: High-performance broadcast for primary-backup systems”, IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), pp. 245-256, 2011.

[7] F. Junqueira and B. Reed, “ZooKeeper: distributed process coordination”, O'Reilly Media, Inc., 2013.

[8] L. Lamport, “Time, clocks and the ordering of events in a distributed system”, Communications of the ACM (CACM), 21, 7, pp. 558-565, 1978.

[9] B. Liskov and J. Cowling, “Viewstamped replication revisited”, https://pmg.csail.mit.edu/papers/vr-revisited.pdf, 2012.

[10] Y. Liu, P. Ezhilchelvan and I. Mitrani, “Design and Analysis of Distributed Message Ordering over a Unidirectional Logical Ring”, EPEW24, Venice, 2024.

[11] P.J. Marandi, M. Primi, N. Schiper and F. Pedone, “Ring Paxos: A high-throughput atomic broadcast protocol”, 40th IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), pp. 527-536, 2010.

[12] I. Mitrani, *Probabilistic Modelling*, Cambridge University Press, 1998.

[13] B.M. Oki and B.H. Liskov, “Viewstamped replication: A new primary copy method to support highly-available distributed systems”, Procs., 7th annual ACM Symposium on Principles of distributed computing, pp. 8-17, 1988.

[14] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm”, 2014 USENIX annual technical conference (USENIX ATC 14), pp. 305-319, 2014.

[15] F.B. Schneider, “Implementing fault-tolerant services using the state machine approach: A tutorial”, ACM Computing Surveys (CSUR), 22, 4, pp. 299-319, 1990.

[16] M. Simatic, A. Foltz, D. Graux, N. Hascoét, S. Ouillon, N. Reboud and T. Wang “TRAINS: A Throughput-Efficient Uniform Total Order Broadcast Algorithm”, Int. Conf. on Protocol Engineering (ICPE) and Int. Conf. on New Technologies of Distributed Systems (NTDS), Paris, pp. 1-8, 2015.

[17] H. Takagi, “Queuing analysis of polling models”, ACM Comp. Surveys, 20, pp. 5-28, 1988.

[18] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili and X. Bao, “Amazon aurora: Design considerations for high throughput cloud-native relational databases”, Procs. 2017 ACM International Conference on Management of Data, pp. 1041-1052, 2017.