Reducing Message Overhead in TMR Systems

John C. Ramirez
Rami G. Melhem
University of Pittsburgh
ramirez@cs.pitt.edu

Abstract

Traditional TMR protocols assume either single, reliable voters for each Triple-Modular Redundant Unit (TMRU) or triplicated voters (one for each processor) for each TMRU. In the first case a voter is a single point of failure for the system. In the second case, many physical messages must be sent across the communication network for each logical data item. We examine some protocols which attempt to maintain the functionality of the triplicated voter TMR protocol, while reducing the number of physical messages required by one third. Possible solutions are examined to the many issues that result from this reduction in communication. Three different Reduced-communication Triple-Modular Redundant (RTMR) protocols are considered, each of which makes different assumptions about the nature of the underlying computation.

1. Introduction

A simple, general way to incorporate fault-detection and fault-correction into computing systems is through redundancy. Initial applications of modular redundancy in computing systems were in hardware at the circuit level. Gates are duplicated (DMR) to detect circuit faults and triplicated (TMR) to mask them. Modular redundancy is now commonly used in systems at the circuit level, the processor level and the process level. DMR/TMR is a very effective way of detecting/correcting hardware errors in real time, without delay to the system. This type of performance is critical in many real time applications [3].

A fundamental problem with the simplest TMR model, however, is that each Triple Modular Redundancy Unit (TMRU) relies upon a single voter to propagate the "correct" output. Although an error in any one of the modules will be masked, an error in the voter will not be masked, and thus the voter may be the site of a single-point failure in the system. To allow for the possibility of faulty voters, TMRUs containing triplicated voters can be used with the modules in each TMRU completely interconnected to those of its neighbors. In this case, 9 physical messages need to be transmitted for each logical message between two TMRUs. This can greatly load the communication subnetwork of the system being used, especially if the message size is large and if the physical interconnection network between TMRUs is not complete.

Clearly, most of the time all 9 messages are not needed for the system to work correctly. Only when an error occurs are the extra messages needed so as to mask the error and keep the system running without delay. This observation led to the proposal in [2] of a fault-masking protocol with reduced redundant communication. The purpose of our work is to present and analyze a new set of TMR communication protocols which mask single faults in each of the TMR units while reducing the system communication required from that of the traditional TMR protocols. Specifically, we investigate protocols that reduce the number of physical messages needed from 9 to 6 while maintaining the single-error masking ability of traditional TMR systems. First, some necessary definitions and background are introduced in Section 2. Next, the protocols are presented in Section 3. Initially the Reduced Triple Modular Redundant protocol RTMR1 is presented for a general system in which TMR is to be incorporated. The protocol can mask a single fault per TMR unit and places no restrictions on the architecture or timing of the system. Under fault-free conditions, RTMR1 requires only 6 physical messages per logical message, and has minimal additional overhead. However, when an error is detected, some additional messages are required for its correction. A second protocol, RTMR2, is then presented which places some restrictions on the architecture and timing of the system, but eliminates the extra message overhead. In this protocol, when an error is detected, a TMRU delays until the error can be corrected. Finally, a third protocol, RTMR3, is presented which requires neither additional message nor time overhead for error correction. In this case some data dependency and consistency restrictions apply. The concept behind RTMR2 and RTMR3 is that of a roll-forward rather than a roll-back mechanism [7, 9].

2. Models, definitions and background

2.1. System and communication models

A *computing task* is a group of one or more *processes* which are working together to solve some *problem*. In order to execute, a computing task must be mapped onto a *computing architecture*. A computing architecture consists of one or more *processors* as well as memory, some type of *communication system* and other physical components. The processes involved in a computing task communicate with each other according to a *communication graph* for the task. Vertices in the communication graph are the processes and an edge connecting two vertices indicates that communication is required between them. In this work, an interprocess message, or *message* from any process P_i to another process P_j is defined to be a passing of data/information from P_i to P_j. The interprocess message being sent between processes is independent of the physical system used to send it. However, the physical *message copies (MCs)* used by the communication system do depend on the chosen architecture, and may vary considerably between architectures.

The architectures we consider assume a distinction between computation and communication agents, such that a fault in a computation agent will not affect the reliability of the communication network. Specifically, in our model, a processor in a system consists of two parts, a *processing unit* and a *communications unit*. The processing unit handles typical processing activities. The communications unit sends and receives messages, contains buffers for incoming and outgoing messages, and handles data flow for a processor.

We distinguish between two different communication frequency models for computing tasks: *regular* and *irregular*. In the regular communication model, messages between processes are sent and received periodically. Formally, in the regular communication model there is a time period, δ, such that, for any processes P_i and P_j in the system, if P_i receives a message from P_j at time t, it will receive another message from P_j by time $t + \delta$. Systolic systems [5] exhibit regular communication. However, unlike systolic communication, which relies on message passing at precise intervals, in regular communication δ is only an upper bound on the interval between messages. Thus pipeline [1, 8] and wavefront [6] systems also utilize regular communication, with δ being the maximum time for any pipeline stage or the maximum time between "waves", respectively.

In the irregular communications model, messages arrive at unpredictable intervals, if at all. In other words, for two processes P_i and P_j in an irregular communication system, if P_i receives a message from P_j at time t, there is no period δ during which it is guaranteed that P_i will receive another message from P_j.

A *message path* is defined to be a chain of processes P_i, P_{i+1}, $\cdots$ such that each P_j sends a message to P_{j+1}. In this work we will assume all message paths are acyclic and that each begins with a *source* process and ends with a *sink* process. Let $M_{k,j}$ denote the k^{th} message sent by process P_j to process P_{j+1}, and call the set of messages $M_{k,j}$ $j = i, i+1, \cdots$ the k^{th} *message wave*, MW_k. We will frequently abuse the notation by dropping the index j and using M_k for any message in MW_k. The index j will usually be implied by the context.

Finally, we distinguish between two data dependency models. In the *dependent* model an interprocess message M_k is dependent upon previously sent messages $M_0 \cdots M_{k-1}$. In this case, each message is somehow building on previous results, and thus a single lost message in a sequence will be fatal to the task. In the independent, or *frame* model, each interprocess message M_i is independent of any previous messages and can still be correctly processed even if a previous message was incorrect or not received. The frame model of communication is often used in applications such as image and signal processing and in pipelined computations, where each data frame is distinct and independent.

2.2. Redundancy models

A process in a system may be replicated and executed on more than one processor. If three replicas exist for each process in a system, that system is a *triple modular redundant* (TMR) system. Each copy of the process is a *process replica* (PR). These replicas together form a Triple Modular Redundant Unit (TMRU) for each process in the system.

Due to their improved reliability over single-voter TMRUs, we will consider only TMRUs with triplicated voters. These TMRUs can handle the voting process in two ways: 1) Voting on output (VO) [4] (Figure 1(a)) and 2) Voting on input (VI) (Figure 1(b)). In the VO scheme, each PR receives only 1 MC (message copy) and voting is done just prior to passing on a single copy (per PR) of the TMRU's results. In the VI scheme, each PR receives 3 MCs, and voting must be done prior to performing the TMRU's computation. Three MCs are then output per PR, so that a receiving TMRU can again vote on input.

The VO scheme is not as resilient to faults as is the VI scheme, since a scenario can be devised in which

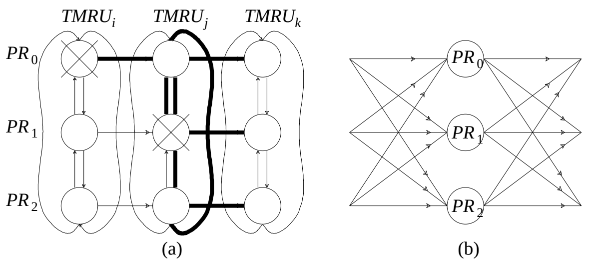

Figure 1. TMR schemes. (a) voting on output (VO) and case where errors in two successive TMRUs causes failure (faulty messages in bold). (b) voting on input (VI).

single faults in two adjacent TMRUs can be fatal to the system (Figure 1(a)). The problem occurs when an error in the voter of PR_0 in $TMRU_i$ produces an incorrect input to PR_0 in $TMRU_j$ and one of the PRs in $TMRU_j$ also produces an error. Two errors will thus exist in $TMRU_j$, from which it cannot recover. (In the remainder of this paper, we will refer to PR_i of $TMRU_j$ by $PR_{j,i}$, where i = 0, 1, 2). The VI scheme would avoid this problem since any single error, whether in a processor or a voter, would be "voted out" by the PRs in the next TMRU.

2.3. FMRRC

In [2], Echtle presented a reduced communication TMR scheme, Fault Masking with Reduced Redundant Communication (FMRRC), that is essentially a VO scheme in which voting is performed in the sending TMRU on message signatures rather than on the entire messages. In other words, prior to sending an MC, the PRs in the sending TMRU exchange and vote on message signatures. These signatures are appended to each MC, thus allowing the receiving PR to detect errors. Upon error detection, a receiving PR requests a second copy of an MC from an alternate sending PR.

In the absence of faults, for each interprocess message the FMRRC protocol requires the passing of 3 MCs (1 per PR), 6 message signatures (between sending PRs) and 3 ack/nack messages. An additional MC is passed when a fault is detected to allow for recovery.

The RTMR schemes presented in Section 3 apply this idea of reduced communication TMR, but with the VI rather than the VO model. RTMR1 has a message complexity similar to that of FMRRC, while RMTR2 and RMTR3 utilize specifics of the computation model to further reduce the message complexity.

2.4. Fault models

We will assume the commonly accepted fault model definitions, with a *fault* being a physical disruption to a processor which may manifest itself as an *error* in a process. A fault may be *detected* or *masked* by a process if the resulting error is detected or masked (respectively). A detected, unmasked fault is *fatal* to the system, while an undetected, unmasked fault is *catastrophic* to the system.

Faults can be *permanent*, *intermittent* or *transient* and may be caused by *hardware* or *software*. This work will initially assume all faults to be permanent, hardware faults. However, the protocols can also be applied to intermittent and transient faults, with varying degrees of success. Intermittent and transient faults are discussed in [10] but are not included here due to space limitations.

In redundant systems, PRs are mapped to distinct processors, and thus a hardware fault will cause an error in only a single PR in an TMRU. Hardware faults are detected and masked in redundant systems through a *voting* mechanism. In the TMR systems we will be studying, if a PR in a TMRU receives two agreeing copies and one disagreeing copy of a message, it can mask the error by choosing a copy of the agreeing messages to use. It is assumed in redundant systems that no two faults in the same TMRU will cause the same incorrect results to be sent. In other words, no two faulty versions of the same message will ever agree.

If a PR fails completely, or a link from the PR to the communication network fails, messages may never be received by a destination TMRU. These failures result in *silent faults*, which are also discussed for each RTMR protocol. In this work, we assume a network error will affect all messages sent by a PR, so that if one MC of a message is delivered correctly, all copies will be. In this way, a network error can be translated into an error in the sending PR's communications unit. Network errors involving isolated MCs can also be considered, but would involve slight modifications to the protocols.

3. Reduced communication TMR protocols

In order to develop an alternative TMR protocol, it is necessary to examine the requirements of the goal TMR system. The following are the goals any TMR protocol should be designed to meet:

1) A single permanent fault should be masked in each of the TMRUs.

2) A second permanent fault should be detected in each TMRU. Though a second fault in a TMRU is fatal, detecting it allows the system to halt gracefully when it can no longer guarantee correct results.

3) Faults in TMRUs should be detected and/or masked quickly. TMR systems are typically used because they can mask single faults with very little delay to the system.

In the protocols considered, we examine the effect of reducing the number of messages on the criteria above. These Reduced-communication Triple Modular Redundant (RTMR) protocols use the same TMRU structure as that described in Section 2 -- each TMRU consists of three PRs which execute independently. However, the algorithm for communication between TMRUs differs from those previously described. In this new scheme, each PR in a TMRU sends its output message to only two PRs in the destination TMRU, rather than all three. Further, with each message sent, the logical interconnections between the source and destination TMRUs alternate between the three configurations shown in Figure 2. The interconnection between PRs in neighboring TMRUs is given by two sets: a send set, S_S and a receive set, R_S. Specifically, if $M_0, M_1, \cdots$ is a sequence of messages sent from $TMRU_i$ to $TMRU_j$, then a process replica $PR_{i,p}$, $p=0,1,2$ will send message M_k to each PR in set S_S(k,p) and $PR_{j,p}$, $p=0,1,2$ will receive M_k from each PR in set R_S(k,p), where S_S and R_S are as defined below:

$$S\_S(k,p) = \{(k+p) \bmod 3, (k+p+1) \bmod 3\} \quad (3.1a)$$

$$R\_S(k,p) = \{(2k+p+2) \bmod 3, (2k+p) \bmod 3\} \quad (3.1b)$$

In words, each PR sends any given message to two of the three PRs in a neighbor TMRU. The connections cycle in a regular way, so that they repeat every third message. In a message-passing system, changing configurations for the sending PRs simply means changing destinations on messages sent. It is important to realize, however, that the receiving PRs can predict which PRs in the sending TMRU will be sending any given message.

With this protocol, only 6 MCs are sent between TMRUs for each message during normal, fault-free operation. However, since PRs in a destination TMRU receive only two copies of each message, a destination PR can immediately detect an error but cannot locate or mask it. To locate and mask the fault, some additional steps must be taken. These steps will be discussed for the different communication models in the next sections.

3.1. RTMR1 for irregular communication

In a system with irregular communication, a protocol must be used which makes no assumptions about the timing of the system. If an error is detected (the two received MCs disagree), steps must be taken to correct the error before the message is processed or propagated.

To correct the error in this situation, another physical copy of the message is required. Assume $TMRU_i$ has sent a message to $TMRU_j$. If one PR in $TMRU_i$ is faulty, then an incorrect message will be received by two PRs in $TMRU_j$. These two PRs must each acquire a third copy of the message to determine which of the disagreeing messages is correct and which is faulty. This third copy must be acquired from the PR in $TMRU_i$ that did not send one of the two original copies. Assume, without loss of generality, that $PR_{j,0}$ has received disagreeing messages from $PR_{i,0}$ and $PR_{i,1}$. It then requests a copy from $PR_{i,2}$. Once $PR_{i,2}$ receives the request for a copy of the message it sends the copy back to $PR_{j,0}$. At this point $PR_{j,0}$ has three copies of the message and can locate the fault and correct the error. The faulty PR is then marked as faulty, so that future disagreeing messages can be resolved without the third copy.

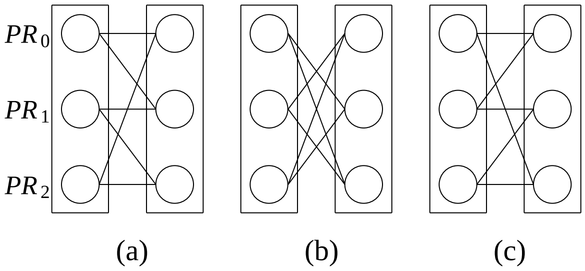

Figure 2. RTMR logical communication configurations.

Note that in order for $PR_{i,2}$ to fulfill the request for a copy of the message, it needs to have this message saved in a buffer. Yet $PR_{i,2}$ cannot keep old messages indefinitely, since it has a finite memory space. Therefore, a protocol must be used which meets all requests for copies sent within a reasonable time, yet does not waste considerable amounts of memory space. The proof for the following proposition is given in [10]:

Proposition 1: Given b input buffers and a flow control protocol in each PR, no PR will need more than $b+1$ output buffers.

When the system is fault-free, RTMR requires only 6 MCs per message, as discussed previously. However, when a fault is detected, 2 extra physical messages are required per PR to mask the fault (1 message for the request, 1 for the additional MC). However, over the two logical message cycles during which a fault is handled by all PRs, it still totals only 18 physical messages, the same as required by traditional TMR protocols. Further, faulty messages are expected to occur with a much lower probability than non-faulty messages, and once a PR is detected to be faulty, it is so marked by PRs receiving messages from it, making the overhead a one-time occurrence for each fault. Thus, even in a system which sustains many faults, the RTMR1 protocol does not increase the effective message traffic to a level above that of the traditional TMR protocol, and in a system with very few faults, the effective message traffic of RTMR1 is only 2/3 that of the traditional protocol.

Since no effective price is paid by RTMR1 in terms of message traffic, the cost of message correction must be incurred elsewhere -- namely, through delays to the system with each detected fault. The delay is equal to the time required for the third copy request to be sent, received and fulfilled. The delays are not simultaneous for all PRs in a TMRU, but since there is no requirement that communication be regular in this system, they are easily handled by the PRs. As with the message overhead, once the faulty PR has been identified, no additional delays are needed.

Handling Silent Faults

The fault detection/correction protocol as described thus far depends upon actual messages being received from 2 (or 3, if necessary) PRs, so comparisons can be made. However, if a PR fails completely, or begins misdirecting messages, then a receiving PR may never receive its message. This is the case of silent faults, and it must be handled by a time-out mechanism. Since communication is irregular, interprocess messages sent by a TMRU cannot be expected at any particular time by a receiving TMRU and thus, no absolute timeout value can be chosen. However, given that $TMRU_i$ is sending a message to $TMRU_j$, suppose (without loss of generality) $PR_{j,0}$ has already received a message from $PR_{i,2}$. Assume the current state of $PR_{j,0}$'s R_S indicates the second message should be from $PR_{i,0}$. $PR_{j,0}$ can now time out after waiting some reasonable time for the second message. The amount of time to wait before timing out is discussed below. At this point, instead of simply processing the message from $PR_{i,2}$, $PR_{j,0}$ makes a request to $PR_{i,1}$ for the third copy of the message.

Once the request for an extra copy of the message has been sent, two possibilities may occur:

1) $PR_{i,0}$ is in fact silent, and its message will never arrive. In this case, $PR_{j,0}$ will eventually receive the extra copy from $PR_{i,1}$ and processing can then proceed normally (see Figure 3).

2) $PR_{i,0}$ is not faulty, but was simply delayed in its sending of the message (due to resolution of a fault in another TMRU). In this situation, $PR_{j,0}$ will receive both the copy from $PR_{i,0}$ that it was expecting, as well as the third copy it had sent for from $PR_{i,1}$. In order for $PR_{i,0}$ to not be marked as faulty (incorrectly), $PR_{j,0}$ must receive the copy from $PR_{i,0}$ *before* the third copy is returned from $PR_{i,1}$. This will always be the case, and is discussed in the next paragraphs. At this point, processing continues normally using the two expected copies of the message. The third copy may still be used, however, if the two copies disagree, indicating a non-silent fault in $TMRU_i$.

It is not obvious what the wait period before a timeout should be for a PR, nor is it obvious in case 2) above that $PR_{j,0}$ receives the (second) MC from $PR_{i,2}$ before the (third) MC from $PR_{i,1}$, so these will be explained next. Under normal, fault-free conditions, six MCs of each message are sent from a sending TMRU to a receiving TMRU. The time of their arrival at the destination PRs is dependent upon the current configuration and the interconnection network of the system in use. Thus, two MCs corresponding to the same interprocess message sent from a PR may not arrive at their destination PRs at the same time. Similarly, MCs corresponding to the same interprocess message sent from two PRs to the same destination PR may not arrive at the same time. Consider two MCs sent at the same time from two PRs to the same destination PR. The difference in the arrival times of these MCs is the *single message skew* for those PRs in that configuration. For a given system, the maximum single message skew can be predetermined based on the interconnection network. Let this value be Δ. Define the *total skew* for a message to be the difference in arrival times of two MCs of the same message at a PR. It is possible for skew to be propagated and magnified as messages are passed from TMRU to TMRU, and thus the *total skew* may exceed the *single message skew*.

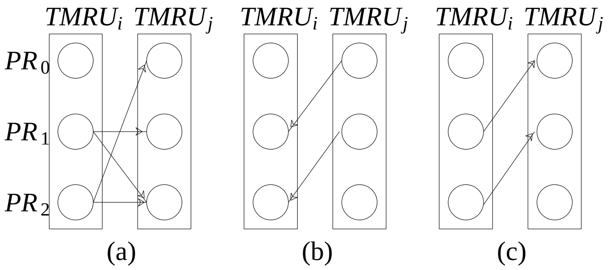

Figure 3. Handling silent faults with RTMR1. (a) $PR_{j,0}$ and $PR_{j,1}$ receive only one copy of the message and time out. (b,c) $PR_{j,0}$ and $PR_{j,1}$ request third copies of the message and subsequently receive them.

Proposition 2: If a PR waits 3Δ time after receiving its first MC prior to timing out, it will never timeout under fault-free operation.

Proof: Clearly, if a PR waits *total skew* time after receiving its first message, it will not timeout, by the definition of *total skew*. Thus we must prove that, during fault-free operation, *total skew* $\leq 3\Delta$. This proof is given in [10].

We have established in Proposition 2 above that a timeout will not occur during fault-free operation. However, when a fault in a TMRU occurs, timeouts can occur in certain circumstances, as shown in Figure 4. When $PR_{i,0}$ sends an erroneous MC (due to a fault), $PR_{j,0}$ and $PR_{j,1}$ will detect the error, and delay while obtaining the third copy of the message (Figure 4(a)). However, $PR_{j,2}$ receives two valid MCs and does not delay. Thus, $PR_{j,2}$ can process the message and immediately send its results on to $TMRU_k$ (Figure 4(b)). It is in $TMRU_k$ that the PRs will timeout, since more than 3Δ will pass before acquiring their second MCs (Figure 4(c)).

Proposition 3: If a timeout occurs due to handling of a fault, no PR will be incorrectly labeled as faulty.

Proof: Without loss of generality, again consider Figure 4(c). $PR_{j,0}$ and $PR_{j,1}$ are waiting to resolve the fault in $TMRU_i$ so that they can process and send their regular MCs. The same two PRs have also been requested to send a "third" MC to resolve each other's timeout in $TMRU_k$. When their MCs are processed and ready, they send the regular MCs, followed by the "third" MCs. In this way, the PRs in $TMRU_k$ will always receive the regular MCs before receiving the "third" MCs and will not mark the PRs as faulty (Figure 4(d)). Note that it does not matter if the requests for the third copies come before or after the actual message is processed, since no copies of the message (regular or

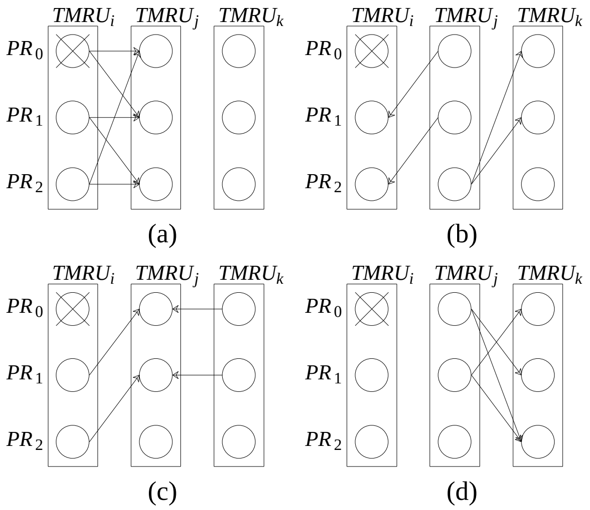

Figure 4. Handling skew due to a fault with RTMR1.

"third" copies) can be sent until the fault in $TMRU_i$ has been resolved. □

A second fault in each TMRU can also be detected by the RTMR1 protocol. If the faults are concurrent, a PR will end up with 3 different MCs, and can immediately signal a system panic. If the first fault has already been resolved, any PR that has detected the fault has marked a PR in the sending TMRU as "faulty". When a second fault occurs in the sending TMRU, each PR in the receiving TMRU will receive a pair of disagreeing messages. Since the PRs are aware of the first faulty PR in the sending TMRU, one of the receiving PRs is guaranteed to detect the second fault, since it is expecting two messages that agree. This PR can then signal to the system that the TMRU has failed. Note that two of the receiving PRs will not detect the second fault, since they are expecting messages that disagree anyway, and they may in fact propagate a faulty message. However, since a second fault in a TMRU is fatal, the whole system will subsequently shut down and the fault is not propagated to more than one TMRU. If the system is later restarted, the TMRU with corrupted data can be restored to a consistent, non-faulty state.

3.2. RTMR2 for regular communication

The general protocol discussed in Section 3.1 can be modified to increase efficiency in computation structures with regular communication. Assume a fault is first exhibited by $PR_{i,0}$ in interprocess message M_k (Figure 5(a)). $PR_{j,0}$ and $PR_{j,1}$ will each receive a copy of this faulty message as well as a copy of a non-faulty message. Since the messages do not match, both receiving PRs will at this point detect the fault. In the RTMR2 scheme, once an error is detected by a PR, the PR delays until the fault can be corrected.

During the delay, both conflicting copies of messages received are saved in buffers as shown in Figure 5(d-f). The determination of whether or not the messages match is also maintained for each pair of messages received in the Message Equality Bit (MEB). After two pairs of messages have been received by a PR (starting with the first disagreeing pair) the PR can determine which PR in the sending TMRU is faulty. Clearly, MEB_0 is a 0, and MEB_1 is now either a 0 or a 1. If $MEB_1 = 0$ then the senders of message M_{k+1} also disagree, and therefore one of the two PRs can be assumed to be faulty. In this case, the PR that sent a copy of both M_k and M_{k+1} is determined to be faulty, since it disagrees with each of the other two PRs. This case is shown in Figure 5(d-f). $PR_{j,1}$ initially receives disagreeing messages from $PR_{i,0}$ and $PR_{i,1}$ (Figure 5(d)). It then receives disagreeing messages from $PR_{i,0}$ and $PR_{i,2}$, indicating that $PR_{i,0}$ is the faulty PR (Figure 5(e)). At this point, the receiving PR processes the non-faulty copy of M_k.

In the case where $MEB_2 = 1$, both physical copies of message M_{k+1} agree. The receiving PR assumes from this that the disagreement in M_k was due to a fault in the PR that did *not* send M_{k+1} and therefore can also process the non-faulty copy of M_k at this point. $PR_{j,0}$ can make this determination in Figure 5(b) (buffer for this PR not shown). Assuming only one fault in any TMRU, message M_{k+2} received by a PR will confirm the faulty PR. In Figure 5(f) this is shown as a 1 in MEB_3 of $PR_{j,1}$.

Proposition 4: Masking of a fault in RTMR2 incurs a delay of at most δ, the regular communication interval.

Proof: As shown in Figure 5 and the previous passages, a PR only delays until it receives another interprocess message, which, by the definition of regular communication, requires at most δ time. □

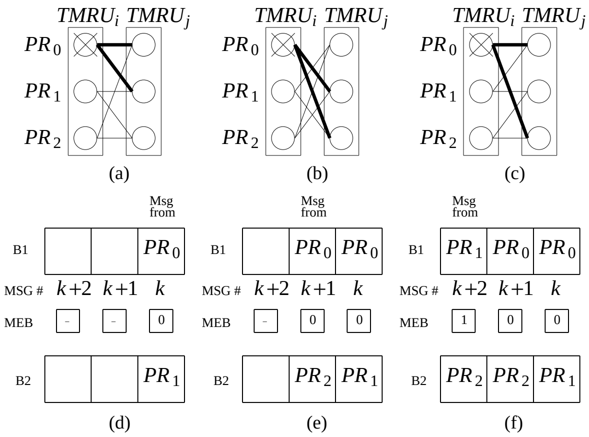

Figure 5. (a-c) configuration cycles during masking of a fault in RTMR2. (d-f) logical buffer structure of the communications unit within $PR_{j,1}$ in (a-c).

Once each PR locates the fault, it marks the sending PR as "faulty", discards the incorrect saved message, processes the correct saved message and continues on with its normal activities. Due to the communication scheme, the PRs within a TMRU will not all detect/correct the error at the same time, but none will delay more than δ time, and the error will not be propagated. Also, it is clear that with this scheme, single, concurrent errors can be handled in all of the TMRUs (since the first TMRU in the chain will never delay, the system will never deadlock) without any corruption of results, since no erroneous data is ever propagated. However, a delay of up to δ is required for each fault. Thus, in a system in which F faults have occurred, at total of up to $F\delta$ extra time may be required for the computation to complete. This delay will significantly impact a job only if a large number of faults develop in the system during a job with a relatively short run-time. As long as a PR in the sending TMRU remains faulty, the receiving PR will continue to process any message M_k only after receiving message M_{k+1}. However, since the communication is regular, there will be no additional effective delay caused by that TMRU after the initial delay.

Handling Silent Faults

In the case of handling silent faults the RTMR2 protocol employs a timeout mechanism. Each receiving PR in the RTMR2 protocol is expecting a given message M_k within time δ of message M_{k-1}. However, if the sending TMRU had to resolve a fault, its PRs will have delayed up to δ time, causing a gap of up to 2δ between messages M_{k-1} and M_k. Matters are further complicated by the fact that PRs in a TMRU do not all delay in the same message cycle due to the logical configuration rotations. Thus, a timeout mechanism for this protocol must be able to distinguish between a delay caused by fault handling and an actual silent fault.

The solution to this problem is actually quite simple. When a PR receives a copy of message M_k, it waits for the other copy without timing out until one of the following events occurs:

1) it receives the second copy of message M_k
2) it receives a copy of message M_{k+1}.

In the first case, processing continues normally without any change to the system. However, in the second case, the PR which was supposed to send the unreceived copy of M_k is marked as faulty, and the received copy of M_k is processed. An example showing this timeout mechanism is shown in Figure 6. The propagation of message M_k is shown in bold. Initially, $PR_{i+1,0}$ and $PR_{i+1,1}$ receive only one copy of M_k (Figure 6(a)). Upon receipt of message M_{k+1}, $PR_{i+1,0}$ and $PR_{i+1,1}$ mark $PR_{i,0}$ as faulty and process M_k. Note that $PR_{i+1,1}$ and $PR_{i+1,2}$ receive only one copy of message M_{k+1}. For $PR_{i+1,1}$ this is no problem, as it has just determined that $PR_{i,0}$ is faulty, so it can process M_{k+1} without waiting for another message (Figure 6(b)). $PR_{i+1,2}$, however, must wait until receiving M_{k+2} before it can time out and process M_{k+1} (Figure 6(c)). In this way the PRs in $TMRU_{i+1}$ resynchronize so that M_{k+1} and future messages are sent in synch.

As in RTMR1, we want to be sure that RTMR2 will not timeout incorrectly, thereby incorrectly labeling a PR as faulty. An incorrect timeout can only occur in RTMR2 if a PR, due to the configuration changes, receives a copy of message M_{k+1} prior to a second, expected copy of M_k. The following proposition, which is proved in [10], indicates that this cannot happen.

Proposition 5: The RTMR2 protocol will only timeout in the event of a silent fault.

The timeout protocols and the proof of their correctness are very similar in RTMR1 and RTMR2. There are some fundamental differences between the two, however. RTMR1 is active in this case, with a PR requesting a third copy of message M_k after waiting 3Δ and subsequently timing out. RTMR2, on the other hand, is passive, with a timeout occurring only after receipt of a copy of message M_{k+1}. For both RTMR1 and RTMR2, the inherent synchronization of the communication scheme assures the protocols of being correct.

Detection of a Second Fault in a TMRU

Consider now the effect of a second fault in the sending TMRU. Since we assume that two faulty processors will not produce the same incorrect result, a second fault in the sending TMRU will cause all MEBs in the receiving PRs to be 0's. A PR determines this state to be fatal, since there is no way to correct the second fault. The PR can then send a panic message to the rest of the system.

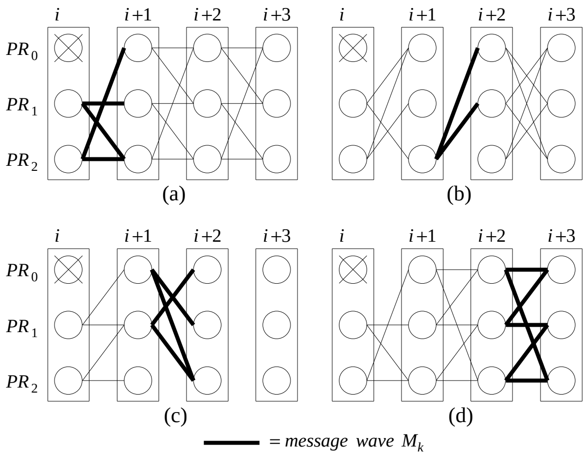

Figure 6. Handling of silent fault in RTMR2 protocol.

3.3. RTMR3 for frame based computations

The protocol RTMR2 improves performance over traditional TMR schemes in regard to the number of MCs required per interprocess message. However, this message reduction is done at the expense of some delays in masking errors when they occur. The delays are necessary when the *dependent* communication model is used, since data must be consistent at all times and no messages can be missed. However, if the *frame* based communication model is used, the protocol can be relaxed to eliminate delays altogether.

The protocol RTMR3 is a simple variation of RTMR2. Each PR in a receiving $TMRU_i$ receives two MCs from PRs in a sending $TMRU_{i-1}$. The PR then processes the message and propagates its result as indicated in Table 1. Under non-faulty conditions (Case 1) RTMR3 acts in identical fashion to RTMR1 and RTMR2 and outputs the value, f(X), its computation produces after receiving two copies of message X. However, now, in order to avoid a delay, when conflicting messages X and Y are received by a PR (Case 2), it sends null messages to the PRs in the next TMRU. Through Cases 3 and 4, the null messages are eventually worked out of the processing chain. A PR which receives a message X and a null message N will process X and produce f(X) as output (Case 3). If a PR receives 2 null messages, it propagates another null message (Case 4). Cases 5 and 6 deal with the system after a fault has been located and the sending PR has been marked as faulty. In these cases, a PR will either process the other copy of the message (Case 5) or propagate a null message (Case 6).

An example of this protocol is shown in Figure 7. Assume a fault is first exhibited by $PR_{i,0}$ in message M_k. The fault in $PR_{i,0}$ is subsequently detected by $PR_{i+1,0}$ and $PR_{i+1,1}$ when their copies of M_k do not agree (Figure 7(a)). These PRs then propagate null messages to $TMRU_{i+2}$ while $PR_{i+1,2}$ propagates valid copies of M_k (Figure 7(b)). $PR_{i+2,2}$ then has two null messages for M_k, and thus propagates a null message. However, $PR_{i+2,0}$ and $PR_{i+2,1}$ each have one null message and one valid message, and thus propagate the valid message (Figure 7(c)). By the time M_k reaches $TMRU_{i+3}$, all of the PRs have at least one copy of the valid message, and thus from $TMRU_{i+3}$ through the rest of the chain of TMRUs, M_k will be valid (Figure 7(d)). Thus, $TMRU_{i+4}$ and all subsequent TMRUs will see no effects of the fault.

To once again stabilize the system, faulty PRs are identified in a manner identical to that in RTMR2. That is, after two messages a receiving PR can determine which sending PR is faulty, and it labels this PR as "faulty". Subsequent messages from the faulty PR are still checked, but are assumed to be faulty if they disagree (Case 5 of Table 1). Referring again to Figure 7, when message M_{k+1} is sent to $TMRU_{i+1}$, $PR_{i+1,0}$ and $PR_{i+1,1}$ locate the fault and can thus propagate correct copies of M_{k+1} (Figure 7(b-c)). Similarly, upon receipt of M_{k+2}, $PR_{i+1,2}$ locates the fault and can thus propagate a correct copy (Figure 7(c-d)).

N = null message, F = known faulty message						
Case	1	2	3	4	5	6
Input Msgs	X X	X Y	X N	N N	X F	F N
Output Msgs	f(X)	N	f(X)	N	f(X)	N

Table 1. Input and output messages for a PR using RTMR3.

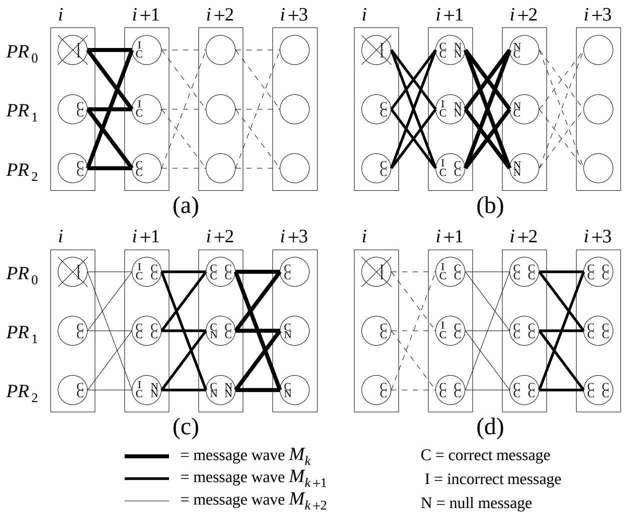

Figure 7. Single fault masking via RTMR3 protocol.

Because a new fault causes null messages to be propagated, there are some fault configurations in which a fault in a second TMRU cannot be immediatedly corrected even after a previous fault in a TMRU has been handled. These situations occur when the second fault occurs in a TMRU immediately before one which has already had a fault. In other words, if a PR in $TMRU_i$ becomes faulty, and then, some time later, a PR in $TMRU_{i-1}$ becomes faulty, a single null message may be propagated into the computation chain (See Figure 8). In Figure 8(a), a new fault is exhibited by $PR_{i,2}$ at message M_k. Faulty messages from $PR_{i,2}$ are detected by $PR_{i+1,0}$ and $PR_{i+1,2}$, while $PR_{i+1,1}$ receives two valid copies of M_k. However, $PR_{i+1,1}$ has previously been determined to be faulty, and thus propagates only faulty messages. Thus, when $TMRU_{i+1}$ propagates M_k, $PR_{i+1,0}$ and $PR_{i+1,2}$ send null messages while $PR_{i+1,1}$ sends faulty messages, and $TMRU_{i+2}$ receives no valid copies of M_k (Figure 8(b)). Since the PRs in $TMRU_{i+2}$ previously knew that $PR_{i+1,1}$ was faulty, they do not propagate the erroneous copy of M_k that they receive, but instead send null messages (Case 6 in Table 1) (Figure 8(c)). Thus, message M_k has been lost and null messages will propagate for the rest of the processing chain. Since the frame computation model is being used, this may degrade the task but will not be fatal to it.

Although a null message is propagated in this instance, subsequent messages will be correct and thus the null message will just be a single interruption in the computations. This can be seen in the propagation of message M_{k+1} in Figure 8. Since the fault in $PR_{i,2}$ can be identified by $PR_{i+1,0}$ and $PR_{i+1,2}$ with receipt of message M_{k+1}, $PR_{i+1,0}$ and $PR_{i+1,2}$ can then propagate the correct M_{k+1}, and the faulty output of $PR_{i+1,1}$ no longer causes the loss of the message.

Proposition 6: Using RTMR3 in a frame based computation and assuming no two faults occur within time 3δ, any single fault in a TMRU will either be masked or result in a null value at the sink process.

Proof: Based on the values stated in Table 1 and shown in Figure 7, a fault in $TMRU_i$ in an initially fault-free system will propagate 2 incorrect MCs to $TMRU_{i+1}$. However, $TMRU_{i+1}$ will propagate only correct MCs and null messages to $TMRU_{i+2}$. Finally, $TMRU_{i+2}$ will propagate only correct MCs to $TMRU_{i+3}$, and the fault is removed within 3δ. Note that even if the fault occurs in the last TMRU prior to the sink process, the fault will still be masked. This is possible because the sink process is not triplicated, and it simply votes on three MCs, thereby masking a fault in the final TMRU. A subsequent fault in a TMRU will again be masked as above, or, as discussed previously, will cause a null message to be propagated to the sink process. □

Null messages are not fatal to a frame computation, but they do degrade its performance, so it is desirable to have as few of them as possible. We have simulated random fault injection into RTMR3 and found that on

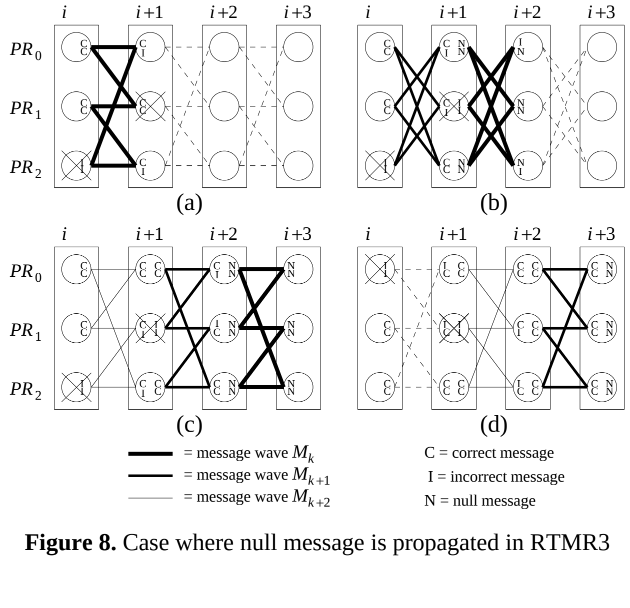

Figure 8. Case where null message is propagated in RTMR3

average less than one null message is produced before the system completely fails. The results of the fault injection experiments show that the average number of null messages at failure asymptotically approach 0.5 as the number of TMRUs approaches infinity. This value is supported intuitively. Note that the null message propagation as described above occurs for a fault in only one out of the three possible PRs in $TMRU_{i-1}$. However, a fault in either of the remaining PRs in $TMRU_i$ will cause the system to fail. Thus, for each new fault in the system, it is twice as likely that the fault will cause a system failure as it is the fault will cause a null message to be propagated. Further, the number of faults generated in a system must greatly exceed the average faults to failure for the average number of null messages to approach and surpass one. Consequently, although null messages may occur, their effect on the performance of the frame based computation will be minimal.

Handling Silent Faults

Timeouts are used to handle silent faults in RTMR3 in a manner identical to that of RTMR2. When a PR receives one copy of a message M_k, it will timeout only if it receives a copy of message M_{k+1} before the second copy of M_k. As explained in Section 3.2, based on the underlying assumptions of the system used, this timeout strategy will never cause inconsistencies within the TMRU chain. It will, however, introduce a delay into the system of at most δ per silent fault. Yet, any traditional TMR protocol would require a similar timeout mechanism, and thus also a delay. Therefore the RTMR3 even in this case is as fast as a traditional TMR protocol.

Detection of a Second Fault in a TMRU

The RTMR3 protocol maintains MEBs for fault detection and isolation, just like RTMR2. Thus, if a PR receives three message pairs in succession, none of which match, it determines that a second fault has occurred in the sending TMRU and sends a system panic.

4. Conclusions

We have examined three protocols which reduce the number of messages required to maintain TMR in a computing system. These protocols reduce the number of messages required by as much as 1/3, which is a significant improvement, especially in systems with relatively low communication bandwidth. All of the protocols utilize rotating logical configurations in the system so as to enable detection and correction of a single fault in any TMRU. However, each protocol is most useful under a different system specification. Thus, the choice of a protocol RTMR1, RTMR2 or RTMR3 will depend greatly upon the physical and logical configuration of the system being used.

The RTMR1 protocol is the most general protocol of the three, and can be used in any distributed system with acyclic message paths. Under fault-free conditions, two copies of each message are required, with a third copy being acquired only if necessary. Thus, in a fault-free state, the RTMR1 protocol has a message savings of 1/3 with no additional overhead required. With faults in the system, extra messages are required, but no more than are required in traditional TMR systems. Instead, the payment for faults in the system is with time, as two message cycles are required for each acquisition of a third copy of a message. Note that the message overhead may be further reduced by utilizing message signatures as proposed in [2]. In cases where sending simply a message signature is less costly than sending an entire MC, 3 of the 6 originally sent MCs can be substituted by message signatures.

The RTMR2 protocol allows faults to be detected and corrected without ever requiring the third copy of any message. Thus, each interprocess message will require exactly two physical messages and the message savings will always be 1/3. When a fault is detected at message M_k, the system uses message M_{k+1} to determine the location, delaying the process. The payment for the message savings is again time, but in this case it is the time required for the *next* message to arrive. RTMR2 also pays in restricting the system to regular communication.

RTMR3 is a relaxation of RTMR2, with null messages taking the place of delays in the system. With this protocol, the only delays are from timeouts in the system due to silent faults, and these are required in traditional TMR protocols as well. Thus, the biggest advantage of RTMR3 is that it is just as fast as a traditional TMR protocol, and still provides the 1/3 message savings of RTMR2. The payment in this case is occasional null messages when certain fault combinations occur. These, however, are few, and they are not fatal when the frame computation model is used.

A comparison of the three RTMR protocols, the traditional TMR protocol and the FMRRC protocol described in Section 2.3 is shown in Table 2.

Protocol	Trans. faults	Silent faults	# of msgs^	Delay	Restrictions
TMR	Yes	Timeout	(9, 9)	none	none
FMRRC	Yes	Timeout	(9,10)*	2 msg cycles per fault	none
RTMR1	Yes	Timeout after 3 Δ	(6,8)+	2 msg cycles per fault	none
RTMR2	No (see [10])	Timeout at next msg	(6, 6)	δ per fault	Reg. Comm.
RTMR3	Yes	As RTMR2	(6, 6)	none	Frame Mod.

^ (no faults, with fault)
* Incl. 6 signature msgs - excl. 3 ack msgs.
+ May incl. 3 signature msgs - excl. 2 nack msgs.

Table 2. Comparison of TMR protocols

References

1. M. J. Berger and S. H. Bokhari, ‘‘A Partitioning Strategy for Nonuniform Problems on Multiprocessors,’’ *IEEE Transactions on Computers* **36**(5)(May 1987).
2. K. Echtle, ‘‘Fault-Masking with Reduced Redundant Communication,’’ *Proceedings of the 16th Annual International Symposium on Fault-Tolerant Computing Systems*, (1986).
3. A. L. Hopkins, T. B. Smith III, and J. H. Lala, ‘‘FTMP--A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft,’’ *Proceedings of the IEEE* **66**(10)(October, 1978).
4. D. L. Kiskis and K. G. Shin, ‘‘Embedding Triple-Modular Redundancy into a Hypercube Architecture,’’ *The Third Conference on Hypercube Concurrent Computers and Applications* **1** pp. 337-345 (1988).
5. H.T. Kung, ‘‘Why Systolic Architectures?,’’ *IEEE Computer* **15**(1)(January, 1982).
6. S. Y. Kung, S. C. Lo, S. N. Jean, and J. N. Hwang, ‘‘Wavefront Array Processors: From Concept to Implementation,’’ *IEEE Computer*, (July 1987).
7. J. L. Long, W. K. Fuchs, and J. A. Abraham, ‘‘Forward Recovery Using Checkpointing in Parallel Systems,’’ *Proceeedings of the 1990 International Conference on Parallel Processing*, pp. I272-I275 (1990).
8. D. M. Nicol and D. R. O’Hallaron, ‘‘Improved Algorithms for Mapping Pipelined and Parallel Computations,’’ *IEEE Transactions on Computers* **40**(3)(March 1991).
9. D. K. Pradhan and N. H. Vaidya, ‘‘Roll-Forward Checkpointing Scheme: A Novel Fault-Tolerant Architecture,’’ *IEEE Transactions on Computers* **43**(10)(October 1994).
10. J. C. Ramirez, ‘‘Flexible Fault-Tolerance Using Redundancy in Mesh Connected Processor Arrays,’’ *Ph.D Dissertation*, (1995). University of Pittsburgh