VTV – A Voting Strategy for Real-Time Systems *

Hüseyin Aysan, Sasikumar Punnekkat, and Radu Dobrin
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{huseyin.aysan, sasikumar.punnekkat, radu.dobrin}@mdh.se

Abstract

Real-time applications typically have to satisfy high dependability requirements and require fault tolerance in both value and time domains. A widely used approach to ensure fault tolerance in dependable systems is the N-modular redundancy (NMR) which typically uses a majority voting mechanism. However, NMR primarily focuses on producing the correct value, without taking into account the time dimension. In this paper, we propose a new approach, Voting on Time and Value (VTV), applicable to real-time systems, which extends the modular redundancy approach by explicitly considering both value and timing failures, such that correct value is produced at a correct time, under specified assumptions. We illustrate our voting approach by instantiating it in the context of the well-known triple modular redundancy (TMR) approach. Further, we present a generalized version targeting NMR that enables a high degree of customization from the user perspective.

Keywords: *Fault tolerance, Redundancy, Majority voting, Plurality voting, Real-time systems.*

1 Introduction

Most real-time applications typically have to satisfy high dependability requirements due to the interactions and possible impacts on the environment. Ensuring dependable performance of such systems typically involves both fault prevention and fault tolerance approaches in their design. Usage of redundancy is the key for achieving fault tolerance and it has been employed successfully in the physical, temporal, information and analytical domains of a large number of critical applications [10, 26]. Static techniques such as N-modular redundancy (NMR) have been used in safety and mission critical applications, most often in the well-known form of triple-modular redundancy (TMR), where three nodes are used for replication [19]. The key attraction of this approach lies in its low overhead and fault masking abilities, without the need for backward recovery [15]. The disadvantages include the cost of redundancy and single point failure mode of the voter. Traditionally, voters are constructed as simple electronic circuits so that a very high reliability can be achieved. Usage of distributed voters has also been employed to take care of the single-point failure mode in case of highly critical systems [7, 18]. With the additional cost of increased computation time, more enhanced voting strategies, such as *plurality*, *median* and *average* voters, can be performed in software. *Plurality* voters (or *m-out-of-n* voters) require m corresponding outputs out of n, where m is less than the majority, to reach a consensus [17, 9]. *Median* voters output the middle and *average* voters output the average value of the replica output values. Surveys and taxonomies on several voting strategies have been presented [2, 8, 16].

Replicated nodes' output values can vary slightly, resulting in a range (or a set) of values which should be considered as correct to avoid problems indicated in [4, 5]. In order to accomplish this, *inexact voting strategies* have been proposed [13, 21, 24]. This phenomenon is also observed in time domain due to several factors, such as clock drifts, node failures, processing and scheduling variations at node level, as well as communication delays. Most of the existing voting strategies, however, focus solely on tolerating variations in value domain by assuming that they are running on tightly synchronized systems, as presented in [11]. On the other hand, using loosely synchronized systems may be an attractive alternative due to low overheads, requiring, however, specifically designed asynchronous voting algorithms to compensate for the timing variations [12].

A simple approach towards tolerating both value and timing failures in replicas using the NMR approach could be adding time stamps to the replica outputs. Then, voting on time stamp values could detect possible timing anomalies of the replicas, under the unrealistic assumptions that the communication is ideal and replicas never halt. Moreover, this approach is unable to mask late timing failures since the voter has to wait for all the values to be delivered from the replicas. Real-time adaptations of majority voting

*This work was partially supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS.

techniques are proposed by Ravindran et. al., [22, 23], and Shin et. al., [25], to overcome the latter problem, where voting is performed among a quorum or a majority of responses received, rather than waiting for all the responses. In Shin et. al.'s work, two voting techniques, relaxing the tight synchronization requirements, are presented, viz., *Quorum Majority Voting (QMV)* and *Compare Majority Voting (CMV)*. QMV performs majority voting among the received values as soon as *2n+1* out of *3n+1* replicas deliver their outputs to the voter, thus, guaranteeing detection of majority of non-faulty values even in the case *n* replicas fail. CMV masks failures of *n* out of *2n+1* replicas as in basic majority voting. The main difference is that in CMV, the voter output is delivered as soon as a majority consisting of identical values has been received, without waiting for the rest of the replicas. Both QMV and CMV provide outputs within a bounded time interval, as long as the assumptions regarding the maximum number of failures hold. However, QMV and CMV are unable to detect any assumption violations in the time domain.

In this paper, we propose a novel approach, *Voting on Time and Value* (VTV), which performs voting in both time and value domains. Our approach enhances the fault tolerance abilities of NMR by ensuring the output from the voter to be both correct in value, and delivered within a specified admissible time interval, under specified assumptions. Furthermore, our proposed approach is able to detect assumption violations regarding the maximum number of failures in time and value domains, thus enabling a fail-safe or fail-stop behavior of the system. However, we do not address Byzantine failure modes [14] and associated solutions due to their tight synchronization requirements and high overheads.

In our approach we identify a *correct time* at which the voter has to deliver the output, and perform voting in the value domain to ensure timely delivery of correct value. The *correct time* is considered to be identified when: 1) a certain number of replica outputs are delivered within a predefined time interval, and 2) a value agreement is established among *valid* values. Our approach enables the selection of valid value outputs, depending on the application type, as either *all available* replica outputs, or *timely* only.

Table 1 presents an overview of various voting strategies, including VTV, applicable to real-time systems. Clearly, VTV approach covers both time and value failures as well as provides clear indication of assumption violations regarding the maximum number of failures at the cost of slightly more complicated algorithms.

The rest of the paper is organized as follows: In Section 2 we present the system model and the basic assumptions used in this paper. Section 3 describes our approach, illustrates it by an instantiation to a system using triplicated nodes and presents a generalized algorithm for arbitrary number of nodes in replication. We conclude the paper in Section 4 outlining the on-going and future work.

2 System Model

In this paper, we assume a distributed real-time system, where each critical node is replicated for fault tolerance, and replica outputs are voted to ensure correctness in both value and time. Upon receiving identical requests or inputs, replicas of a node start their executions on dedicated processors whose clocks are allowed to drift from each other at most by a maximum deviation. This bound can be achieved by relatively inexpensive clock synchronization algorithms implemented in software (compared to expensive tight clock synchronization implementations). After the replicas complete execution, the outputs are sent to a stand-alone voting mechanism. Deviation in message transfer times from the replicas to the voter is also bounded by using reliable communication techniques. Upon receiving deliveries from replicas, the voter starts executing the voting algorithm and outputs a correct value at an admissible time or signals the non-existence of a correct output to the subsequent component in the system in a timely manner.

In our approach we use the following notations for the system properties:

Δ maximum admissible time deviation of the voter output, as per the real-time system specifications

δ maximum admissible deviation in time domain between any two replica outputs, as perceived by the voter, which includes the maximum skew between any two non-faulty replica clocks δ_{clock} and the maximum skew between any two message transmissions from replicas to the voter δ_{comm}

σ maximum admissible deviation in value domain between any two replica outputs

ϵ worst case computation time of the voting algorithm

The reader should note that the admissible time interval for the voter output is determined according to the system specifications, i.e., what the rest of the system can tolerate as per the real-time specification, i.e., Δ. On the other hand, time correctness of a replica output delivered to the voter is judged according to its behavior with respect to the timing bound δ.

Our approach builds upon the failure concepts originally introduced in [1, 3, 20]. For the sake of readability, we denote the i^{th} replica of a given node by R_i. The output delivered by R_i, is specified by two domain parameters, viz., value and time, together with their admissible deviations:

$$\textit{Specified output for } R_i = \ < v_i^*, t_i^*, \sigma, \delta >$$

Voting Strategy	Description	Voting domain(s)
QMV	1. Wait for a quorum (2n+1 out of 3n+1 replica outputs) 2. Perform majority voting among the quorum	value
CMV	Wait for a majority (n+1 out of 2n+1 replica outputs) <u>with identical values</u>	value
VTV	1. Wait for a plurality (m-out-of-n replica outputs) <u>delivered within a predefined time window</u> 2. Perform plurality voting in value domain among available replica outputs 3. In case there is no agreement in value domain, return to step one (or signal disagreement in case it was the last possible plurality in time)	value and time
VTV (timely)	1. Wait for a plurality (m-out-of-n replica outputs) <u>delivered within a predefined time window</u> 2. Perform plurality voting in value domain among available <u>timely</u> replica outputs 3. In case there is no agreement in value domain, return to step one (or signal disagreement in case it was the last possible plurality in time)	value and time

Table 1. Overview of voting strategies suitable for real-time systems

where v_i^* is the correct value, t_i^* is the correct time point (seen by a perfect observer) when the output should be delivered, $[v_i^* - \frac{\sigma}{2}, v_i^* + \frac{\sigma}{2}]$ is the admissible value range and $[t_i^* - \frac{\delta}{2}, t_i^* + \frac{\delta}{2}]$ is the admissible time interval for output delivery as per system specifications.

An output delivered by R_i is denoted as:

$$\textit{Delivered output from } R_i = \ < v_i, t_i >$$

where v_i is the value and t_i is the time point at which the value was delivered.

We define the output generated by replica R_i as *incorrect in value domain* if:

$$v_i < v_i^* - \frac{\sigma}{2} \text{ or } v_i > v_i^* + \frac{\sigma}{2}$$

and *incorrect in time domain* if:

$$t_i < t_i^* - \frac{\delta}{2}\ (early\ timing\ failure)$$

or if

$$t_i > t_i^* + \frac{\delta}{2}\ (late\ timing\ failure).$$

The notations used for the failure behavior of the replicas (seen by a perfect observer) are:

F total number of failed replicas, consisting of three main components ($F = F_t + F_v + F_{vt}$):

- F_v the number of replicas that have failed only in value domain
- F_t the number of replicas that have failed only in time domain, consisting of two subcategories:
  - F_t^e the number of replicas that produce early outputs with correct values
  - F_t^l the number of replicas that produce late outputs with correct values
- F_{vt} the number of replicas that have failed in both domains, consisting of two subcategories:
  - F_{vt}^e the number of replicas that produce early outputs with incorrect values
  - F_{vt}^l the number of replicas that produce late outputs with incorrect values

Finally, the algorithm specific notations used in the paper are:

N number of replicas

M_t minimum number of replicas required to form a consensus in time domain, as per system specification

M_v minimum number of replicas required to form a consensus in value domain, as per system specification

$valid$ is a binary variable indicating the validity of early outputs for the purpose of voting, e.g., $valid = 1$ indicates that early values are taken into account in the voting procedure

Basic assumptions: Our approach relies on the following set of basic assumptions (to a large extent based on [6]):

A1 non-faulty nodes produce values within a specified admissible range after each computation block

A2 non-faulty nodes produce values within a specified admissible time interval after each computation block

A3 replica outputs with incorrect values do not form (or contribute in forming) a consensus in value domain

A4 incorrectly timed replica outputs do not form (or contribute in forming) a consensus in time domain

A5 there exist adequate mechanisms, e.g., infrequent synchronization, which are significantly less costly than tight synchronization, to ensure a maximum permissible replica deviation from the global time

A6 the voting mechanism does not fail, as being designed and implemented as a highly reliable unit

3 Voting on Time and Value (VTV)

In this section we present our voting strategy that explicitly considers failures in both value and time domains. Obviously, the correctness of our method relies on a number of conditions regarding the permissable number of replica failures:

$\mathcal{C}1$ The number of replicas failures can not exceed the difference between the total number of replicas and the minimum number of failure free replicas required to achieve consensus in value domain.

$$F_v + F_t + F_{vt} \leq N - M_v$$

$\mathcal{C}2$ The number of replica failing in time domain is bounded by the difference between the total number of replicas and the minimum number of failure free replicas required to achieve consensus in time domain.

$$F_t + F_{vt} \leq N - M_t$$

Our goal is twofold:

1. always deliver the correct value within $[t^* - \frac{\Delta}{2}, t^* + \frac{\Delta}{2}]$, if the conditions $\mathcal{C}1$ and $\mathcal{C}2$ hold

2. provide information about violation of the conditions, otherwise.

3.1 Approach

As stated in the system model, the admissible time interval for output delivery, tolerated by the system, is denoted by Δ. Additionally, the worst case computation time of the algorithm ϵ needs to be taken into account in the voting procedure in order to ensure the delivery of the voter's output within Δ. Consequently, the first step is to find the maximum permissible deviation δ between any two replicas (Figure 1) such that:

$$\delta + 2\epsilon \leq \Delta$$

The reader should note that, in a loosely synchronized system, maximum deviation in replica output delivery times from t^* is the same value in both directions due to the working principles of clock synchronization mechanisms. Therefore δ is symmetrically distributed with respect to t^*. If Δ is also symmetrically distributed with respect to t^* in the system specification, derivation of δ is performed with a pessimism to the extent of the time interval equal to ϵ, shown at the beginning of the time interval Δ in Figure 1.

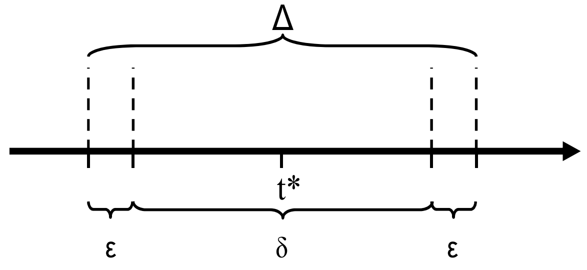

Figure 1. Relation between Δ, δ, and ϵ

The second step is to ensure that VTV detects and outputs consensus (if reached) within the derived $[t^* - \frac{\delta}{2}, t^* + \frac{\delta}{2}]$. In VTV, agreement in the time domain is reached when M_t out of N replicas deliver their outputs within the derived time interval of $[t^* - \frac{\delta}{2}, t^* + \frac{\delta}{2}]$ (referred to as *feasible window* henceforth).

The maximum number of sets, consisting of M_t consecutive replica outputs each (out of the N replicas), is $N - M_t + 1$. Since the consensus in time domain can be reached in any of these sets, a separate feasible window needs to be initiated upon receiving each of the first $N - M_t + 1$ replica outputs. We keep track of the feasible windows by using simple countdown timers. Once an agreement in time domain is obtained, then values are voted. If an agreement in value domain is not obtained within a particular feasible window, the process continues with subsequent feasible windows, until agreement in both time and value domains can be achieved, or violations of $\mathcal{C}1$ or $\mathcal{C}2$ are detected.

Depending on the real-time application characteristics, a value produced by a node may be considered *valid* or *invalid* for the purpose of voting, in case it is produced early. An illustration of replica output flow is given in Figure 2.

An issue is the choice of the set of *valid* values to be used in the voting mechanism, i.e., *all* received values vs. *all timely* received values. We illustrate this voting dilemma by using the scenario described in Figure 3. Let us assume, for

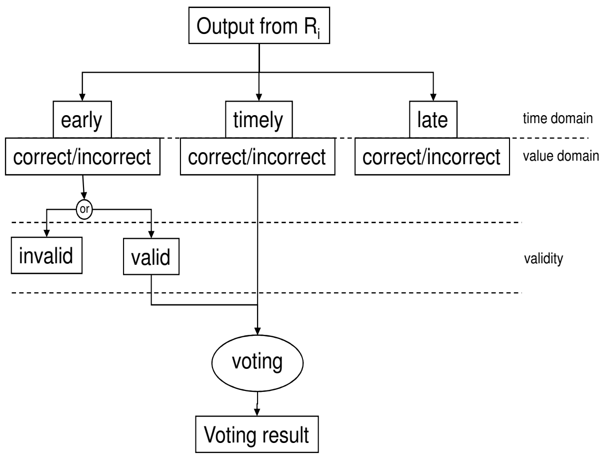

Figure 2. Replica output flow

example, an airbag control system where a collision sensor is replicated in five different nodes that produce one out of two values periodically, e.g., value a in case of a collision detection or value b otherwise. If a collision is detected at a time $t \leq t_1$ let us assume that the airbag has to inflate within a time interval $[t_{start}, t_{end}]$, where $t_2 < t_{start} \leq t_3$ and $t_5 \leq t_{end}$. In our example, the first two values are detected as *early* and the last three are identified as *timely*. However, in this case, even an early value has to be taken into consideration in the voting since an early collision detection is still a valid output with respect to the value domain. Thus, the output has to be voted upon receiving the last value at time t_5, among *all* values, i.e., $a,\ a,\ a,\ b$, and b, resulting in an output a at time $(t_5 + \epsilon)$ (where ϵ is the time required for the voting and is assumed to be negligible in this paper for simplifying the presentation). In this case, the condition $\mathcal{C}1$ becomes:

$\mathcal{C}1$ The number of replicas failures, excepting early timing ones, can not exceed the difference between the total number of replicas and the number of failure free replicas required to achieve consensus in value domain.

$$F_v + (F_t - F_t^e) + F_{vt} \leq N - M_v$$

The benefit of this observation is that the number of nodes required to mask a given number of failures (in time and value domain) can be significantly reduced, compared to traditional NMR approaches, as replica outputs failed in one domain may still be used to reach consensus in the other domain.

On the other hand, let us assume that the same Figure 3 illustrates an altitude sensor in an airplane, replicated by five nodes to read and output the altitude periodically to the voter, where data freshness may be a more desirable aspect. As the correct window of time for the output is the same as described in the previous example, the only relevant values to be taken into consideration by the voter are $a,\ b$, and b corresponding to the time points $t_3,\ t_4$, and t_5 respectively. Hence, a desirable output to be produced in this application at time $(t_5 + \epsilon)$ is b.

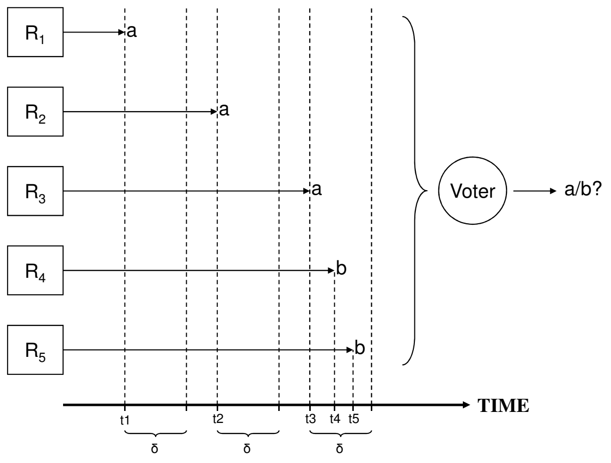

Figure 3. Voting dilemma

Upon finding a feasible window, the decision on whether the early generated replica outputs are involved in value voting or not results in two cases:

Case 1 *Early and timely outputs are considered valid.* If a plurality of corresponding values exist among all the received values, the plurality value is delivered as the correct output.

Case 2 *Only timely outputs are considered valid.* If a plurality of corresponding values exist among the timely received values, the plurality value is delivered as the correct output.

If there is not an agreement in value domain within the current feasible window, the process continues with the subsequent window. If the end of last feasible window is reached, or all replica outputs are received without reaching an agreement on the values, disagreement is signalled to the rest of the system indicating a violation of $\mathcal{C}1$ or $\mathcal{C}2$.

3.2 VTV in TMR

In this section, we present an instantiation of our approach to triple modular redundancy which can tolerate single node failures in value domain, time domain or both (Algorithm 1). In this example, we assume early timing failures as *invalid* for the purpose of voting. However, the validity of such values can be easily tuned in the algorithm. Agreement in time domain is achieved if at least two values are delivered to the voter within a time interval less than or equal to δ, since this is the maximum deviation in time among all the values as long as no failure occurs. Agreement in value domain is achieved if at least two of the *timely* outputs have the same value.

The algorithm signals disagreement in case agreement condition is not satisfied in any of the domains, thus enabling a fail-safe or fail-stop behavior of the system.

Algorithm 1: VTV

```
input  : $v_1, v_2, v_3 = NULL$
output: $v_{out}$ or indication of disagreement
/* Inputs are indexed with respect to
   the order of reception                        */
/* Voting in value domain is performed
   among the available timely replica
   output values                                 */
$C_1, C_2 \leftarrow \delta$ ;                  // countdown timers
while $v_1$ = NULL do wait;
start $C_1$;
while $v_2$ = NULL do wait;
start $C_2$;
if $C_1 > 0$ then
    if $v_1 = v_2$ then
        output $v_1$;
    else
        while $C_1 > 0$ and $v_3$ = NULL do wait;
        if $C_1 > 0$ and ($v_3 = v_1$ or $v_3 = v_2$) then
            output $v_3$;
        else if $v_3 <> NULL$ then
            signal disagreement;
        else
            while $C_2 > 0$ and $v_3$ = NULL do wait ;
            if $v_3 = v_2$ then
                output $v_3$;
            else
                signal disagreement;
            end
        end
    end
else if $C_2 > 0$ then
    while $C_2 > 0$ and $v_3$ = NULL do wait;
    if $v_3 = v_2$ then
        output $v_3$;
    else
        signal disagreement;
    end
else
    signal disagreement;
end
```

The replicated nodes' output values are stored in local variables v_1, v_2 and v_3. Values are assigned to these variables in the order of receiving the replica outputs (i.e., the first received value is stored in v_1, the second value is stored in v_2 and the last value is stored in v_3). Two countdown timers, C_1 and C_2, initially set to δ, are used to keep track of the feasible windows in order to identify agreement in time domain.

The algorithm waits for the first replica output to be delivered and then starts C_1. It continues by waiting for the second replica output and starts C_2 upon its arrival. If both replica outputs have arrived before C_1 expires, and have matching values, the voter will output the matching value as the correct value as a majority is formed in both time and value domains. Otherwise we have two cases:

Case 1 C_1 has not reached zero, and the values v_1 and v_2 do not match. In this case, the algorithm waits for v_3 until C_1 reaches zero. If the third value arrives before C_1 reaches zero and matches either v_1 or v_2, the algorithm outputs the matching value since all values are timely and there is a majority in value domain. In case there exists no replica output pair matching in value domain, the algorithm signals *disagreement*. If the third value does not arrive before C_1 reaches zero, the algorithm waits for v_3 until C_2 reaches zero. If v_3 is received and matches v_2 before C_2 reaches zero, the algorithm outputs the matching value. Otherwise the algorithm signals *disagreement*.

Case 2 C_1 has reached zero. In this case, v_1 is considered invalid, and the algorithm waits for v_3 until C_2 reaches zero, as only a match between v_2 and v_3 may result in an agreement. If the values do not match, or if v_3 has not been received at all, the algorithm signals *disagreement*. If v_2 and v_3 matches, the algorithm outputs the matching value.

3.3 VTV in NMR

In this section we present a generalized algorithm for value and time based voting for arbitrary nodes in replication (Algorithm 2).

The generalized VTV algorithm uses four system parameters: the total number of replicas N, the number of replicas needed for agreement in time domain M_t, the number of replicas needed for agreement in value domain M_v and the boolean parameter *timely* which indicates whether *only timely* or *all* received replica outputs will be used for value voting.

Inputs to Algorithm 2 are the replica outputs which are indexed in the order of reception. The output of the algorithm is either a value considered correct that is generated at a correct time or an indication of disagreement. Countdown timers, $C_1, C_2, \ldots, C_{N-M_t+1}$, initially set to δ, are used to keep track of feasible windows. These timers are started upon the reception of the first $N - M_t + 1$ replica outputs. Set R keeps the values of replica outputs which

Algorithm 2: VTV(N,M_t,M_v,timely)

```
input : $v_1, \ldots, v_n = NULL$
output: $v_{out}$ or indication of disagreement
/* Inputs to the algorithm are indexed
   in the order of replica outputs'
   reception                                */
/* $N$: the total number of replicas         */
/* $M_t$: the number of replica outputs
   needed for an agreement in time
   domain                                   */
/* $M_v$: the number of replica outputs
   needed for an agreement in value
   domain                                   */
/* Voting in value domain is performed
   among the available timely replica
   output values if the parameter timely
   is true, or all the available replica
   output values if it is false             */
/* Countdown timers                         */
$C_1, \ldots, C_{N-M_t+1} \leftarrow \delta$
/* Set of all received values               */
$R \leftarrow \emptyset$
/* The earliest timely element of set
   R's arrival index                        */
$earliestTimely \leftarrow 1$
for i = 1 to N do
    while $v_i = NULL$ and $C_{N-M_t+1} > 0$ do wait;
    if $C_{N-M_t+1} = 0$ then
        signal disagreement;
        break;
    end
    $R = R \cup \{v_i\}$;
    if $i \leq N - M_t + 1$ then
        start $C_i$;
    end
    if $i \geq max(M_t, M_v)$ then
        for j = earliestTimely to $N - M_t + 1$ do
            if $C_j > 0$ then
                if $i - j + 1 \geq M_t$ then
                    if ValueAgreement($R, M_v$) then
                        return $v_{out}$;
                    end
                else
                    break;
                end
            else
                earliestTimely++;
                if timely then
                    $R = R \setminus \{v_j\}$;
                end
            end
        end
    end
    i++;
end
```

are considered valid. If parameter *timely* is true, the values of replica outputs are removed from this set whenever their corresponding countdown timers reach zero. Variable *earliestTimely* holds the arrival index of set R's earliest timely element.

When the algorithm starts to execute, it waits for replica outputs to be delivered. As soon as a replica output is received, the countdown timer with the current index is activated and the value of the replica output is added to the set R. After the reception of $max(M_t, M_v)$ replica outputs, the algorithm starts checking for agreement in time domain within each active *feasible window*. Once an agreement in time domain is achieved, function *ValueAgreement* is called for value voting on the elements of set R. If an agreement on values is achieved as well, the algorithm outputs the value. Otherwise the procedure is continued with the consecutive *feasible windows*. If the last *feasible window* is ended without any agreement in both value and time domains, the algorithm signals *disagreement* to the rest of the system.

Algorithm 2 is intended as a general structure, implementation of which will require a set of suitable data structures and operators for efficient handling of the inputs and their order of occurrences.

4 Conclusions

In this paper, we have presented a new voting strategy, Voting on Time and Value (VTV), for redundant real-time systems, to explicitly consider both value and timing failures for achieving fault tolerance in real-time applications. Our method produces the correct output value, as well as identifies the correct window of time in which the output has to be delivered, provided the conditions on the maximum number of permissible failures are not violated. Moreover, our method is capable of providing information about violation of such conditions, if any.

We have presented an algorithm for the particular case where the nodes are triplicated, and illustrated the basic idea on how to perform the voting in both value and time domains. We have further presented a generalized version of the algorithm that enables a high degree of customization with respect to the number of replicas, plurality levels, as well as the validity of early values for the purpose of voting.

Our ongoing research indicates that VTV, when used in the general case to mask an arbitrary number of value and timing failures, is cost-effective in comparison with the number of nodes required by majority voting in NMR. The main reason is that, in our approach, a non-faulty node can be successfully used to mask both a value and a timing failure in the voting procedure.

References

[1] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. *Research Report N01145, LAAS-CNRS*, 2001.

[2] D. Blough and G. Sullivan. A comparison of voting strategies for fault-tolerant distributed systems. *Proceedings of the 9th Symposium on Reliable Distributed Systems*, pages 136–145, 1990.

[3] A. Bondavalli and L. Simoncini. Failure classification with respect to detection. *Proceedings of the 2nd IEEE Workshop on Future Trends in Distributed Computin*, pages 47–53, 1990.

[4] S. S. Brilliant, J. C. Knight, and N. G. Leveson. The consistent comparison problem in N-version software. *IEEE Transactions on Software Engineering*, 15(11):1481–1484, 1989.

[5] L. Chen and A. Avizienis. N-version programming: A fault-tolerance approach to reliability of software operatlon. *Proceedings of the 25th International Symposium on Fault-Tolerant Computing, ' Highlights from Twenty-Five Years'.*, page 113, 1995.

[6] P. Ezhilchelvan, J.-M. Helary, and M. Raynal. Building responsive TMR-based servers in presence of timing constraints. *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 267–274, 2005.

[7] V. D. Florio, G. Deconinck, and R. Lauwereins. The EFTOS voting farm: A software tool for fault masking in message passing parallel environments. *Proceedings of the 24th Euromicro Conference*, 1:379–386, 1998.

[8] F. D. Giandomenico and L. Strigini. Adjudicators for diverse-redundant components. *Proceedings of the 9th Symposium on Reliable Distributed Systems*, pages 114–123, 1990.

[9] A. Grnarov, J. Arlat, and A. Avizienis. Modeling of software fault-tolerance strategies. *Proceedings of the 11th Annual Pittsburgh Modeling and Simulation Conference*, pages 571–578, 1980.

[10] A. Hopkins, T. Smith, and J. Lala. FTMP: A highly reliable fault-tolerant multiprocessor for aircraft. *Proceedings of the IEEE*, 66(10):1221–1239, 1978.

[11] H. Kopetz. Fault containment and error detection in the time-triggered architecture. *Proceedings of the 6th International Symposium on Autonomous Decentralized Systems*, pages 139–146, 2003.

[12] J. Lala, L. Alger, S. Friend, G. Greeley, S. Sacco, and S. Adams. An analysis of redundancy management algorithms for asynchronous fault tolerant control systems. *Research Report NASA-TM-100007, NASA*, 1987.

[13] J. Lala and R. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, 1994.

[14] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[15] J.-C. Laprie. Dependable computing and fault-tolerance: Concepts and terminology. *Proceedings of the 25th International Symposium on Fault-Tolerant Computing, ' Highlights from Twenty-Five Years'.*, 1995.

[16] G. Latif-Shabgahi and a. S. B. J.M. Bass. A taxonomy for software voting algorithms used in safety-critical systems. *IEEE Transactions on Reliability*, 53(3):319–328, 2004.

[17] P. Lorczak, A. Caglayan, and D. Eckhardt. A theoretical investigation of generalized voters for redundant systems. *Proceedings of the 19th International Symposium on Fault-Tolerant Computing, FTCS-19. Digest of Papers.*, pages 444–451, 1989.

[18] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *Journal of Research and Development*, 6:200–209, 1962.

[19] J. V. Neuman. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.

[20] D. Powell. Failure mode assumptions and assumption coverage. *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, pages 386–395, 1992.

[21] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabejac, and A. Wellings. GUARDS: a generic upgradable architecture for real-time dependable systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):580–599, 1999.

[22] K. Ravindran, K. Kwiat, and A. Sabbir. Adapting distributed voting algorithms for secure real-time embedded systems. *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, pages 347–353, 2004.

[23] K. Ravindran, K. Kwiat, A. Sabbir, and B. Cao. Replica voting: a distributed middleware service for real-time dependable systems. *Proceedings of the 1st International Conference on Communication System Software and Middleware*, pages 1–7, 2006.

[24] J. Rushby. Formal methods and the certification of critical systems. *Computer Science Laboratory, SRI International, Tech. Rep CSL-93-7*, 1993.

[25] K. Shin and J. Dolter. Alternative majority-voting methods for real-time computing systems. *IEEE Transactions on Reliability*, 38(1):58–64, 1989.

[26] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Milliar-Smith, R. E. Shostak, and C. B. Weinstock. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, 1978.