Adding the Time Dimension to Majority Voting Strategies

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

Industrial real-time applications typically have to cope with high dependability requirements and require fault tolerance in both value and time domain. A widely used approach to ensure fault tolerance in dependable systems is the N-modular redundancy (NMR) approach 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, Value and Time Voting (VTV), applicable to real-time systems, which extends the modular redundancy approach by explicitly considering both value and timing failures in the voting mechanism, such that correct value is produced at correct time, under specified assumptions.

1. Introduction

Many real-time applications are also characterized by their high dependability requirements due to their 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 many critical applications. Static techniques such as N-modular redundancy (NMR) have been used in safety and mission critical domains such as avionics, space and nuclear applications, most often in the particular case where three nodes are used for replication, i.e., triple-modular redundancy (TMR) [6]. The key attraction of this approach lies in its low overhead and fault masking abilities without the need for any backward recovery. The disadvantages include 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 triplicated voters can be employed to take care of the single-point failure mode in case of highly critical systems [5]. A survey and taxonomy of software voting algorithms has been presented in [4].

Timing variations in receiving outputs from replicated nodes by the voter can occur due to factors such as clock drifts, node failures, processing and scheduling variations at node level as well as communication delays. Most of the proposed voting strategies, however, focus only on masking value failures by assuming that the system is tightly synchronized as in [3]. On the contrary, loosely synchronized systems may be an attractive alternative due to low overheads, but require specifically designed asynchronous voting algorithms to compensate for the timing variations.

A simple approach towards tolerating both value and timing failures in a replica using the NMR approach could be to attach time stamps to the replica outputs. Then, performing majority voting on time stamp values could detect possible timing anomalies of the nodes, under the assumption that the communication is perfect and nodes never halt. However, this is far from true as well as late timing failures are not possible to be masked in this approach.

Shin and Dolter [8] proposed two voting techniques applicable to real-time systems, not necessarily having tight synchronization, 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 the existence of a 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, as soon as the majority of identical values has been received, the output is delivered without waiting for the rest of the replicas. Both QMV and CMV provide outputs within a bounded time as long as the assumptions for maximum number of failures hold. However, unlike standard majority voters, which can identify certain assumption violations when there is no agreed result, QMV and CMV are incapable of indicating any anomalies with respect to time.

In this paper we propose a new approach, Value and Time Voting (VTV), which performs majority voting in both time and value domains, and provides timely and correct outputs under specified assumptions or indicates assumption violations, if any. It does so by, first, using counters to identify the correct time interval in which the voter has to deliver the output, and then, by selecting the valid values (with respect to time) to be used in the voting mechanism. This enhances the fault tolerance abilities of NMR by restricting the replica outputs to be both correct in value and delivered within a specified feasible (admissible) time interval.

The rest of the paper is organized as follows: In Section 2 we present the system model and the assumptions used in this paper. Section 3 describes our approach and illustrates it by an instantiation to a system using triplicated nodes. 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. For the sake of readability, in the rest of the paper, we denote the i^{th} replica of a node *N* by N_i. The output delivered by N_i, is specified by two domain parameters, viz., value and time [1, 7]:

$$\textit{Specified output for } N_i = \; < v_i^*, t_i^*, \Delta >$$

where v_i^* is the correct value, t_i^* is the correct time point when the output should be delivered and $[t_i^* - \Delta, t_i^* + \Delta]$ is the feasible time interval for output delivery as per the real-time system specifications.

An output delivered by N_i is denoted as:

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

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 N_i as *incorrect in value domain* if $v_i \neq v_i^*$, and *incorrect in time domain* if $t_i < t_i^* - \Delta$ (*early timing failure*), or if $t_i > t_i^* + \Delta$ (*late timing failure*).

Assumptions: Our approach relies on the following set of assumptions (to a large extent based on [2]):

1. replicated nodes are characterized by deterministic executions
2. non-faulty nodes produce identical values after each computation block
3. the voter does not fail
4. incorrect values do not form majority
5. A maximum permissible *drift* δ from the global time is specified and ensured by infrequent synchronization (which is significantly less costly than tight synchronization).

3. Value and Time Voting (VTV)

In this section we present our novel voting strategy that explicitly considers failures in both value and time domains. As a consequence of assumption 5, in the worst case, the maximum deviation between any two replica outputs is 2δ. Hence, in VTV approach, agreement in the time domain is reached when a majority of replicas deliver their outputs within this derived time interval of 2δ (referred to as feasible window henceforth). If a node has n replicas, then at least $m = \lceil \frac{n+1}{2} \rceil$ outputs from these replicas need to match for establishing majority. Since, at a given time instant, the majority in time domain can be formed by the latest m outputs, a separate feasible window needs to be initiated upon receiving each of first m 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 for majority. If majority in value is not obtained for a particular feasible window, the process continues with subsequent feasible windows, until a majority in both time and value can be formed, or an assumption violation is detected.

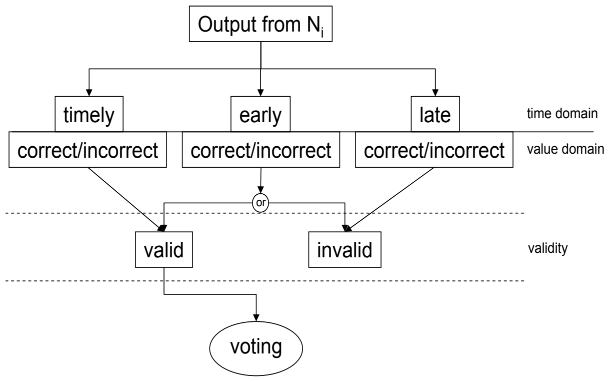

Figure 1. Replica output flow through voter

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 through the voter is given in Figure 1. 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 2. Let us assume, for example, an airbag control system where a sensor is replicated in five different nodes and produces one out of two values periodically, e.g., value a in case of a collision and 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 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, 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).

On the other hand, let us assume the same Figure 2 illustrating an altitude measurement 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, the output produced at time $(t_5 + \epsilon)$ is b.

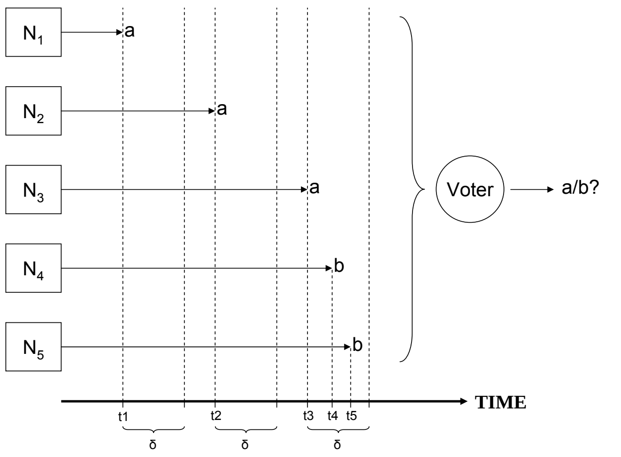

Figure 2. Voting dilemma

In case of early outputs being considered as valid outputs, upon finding a feasible window, if majority in value domain is obtained with all the values received so far, the voter delivers the majority value without waiting for the rest of the replicas. Otherwise, either majority, receipt of all replica outputs or the end of the feasible window is waited for, whichever comes first. If a majority is obtained, it is delivered as a correct output. If the end of the feasible window is reached without a majority, the process continues with a subsequent feasible window. If the last feasible window is reached, or all replica outputs are received without reaching an agreement on the majority of the values, disagreement is signalled to the rest of the system.

In the scenario where only timely outputs are considered as valid outputs, upon finding a feasible window, if majority in value domain is obtained with all the values received within the feasible window, the value is delivered as a correct output. Otherwise, either majority, receipt of all replica outputs or the end of the last feasible window is waited for, whichever comes first. If majority is obtained, the value is output. If the end of the feasible window is reached without a majority, the process continues with a subsequent feasible window. If the end of feasible window is reached or all replica outputs are received without a majority in values, majority among the timely values is voted and delivered.

3.1 Example: VTV in TMR

In this section we present in 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.

Majority in time domain is achieved if at least two values are delivered to the voter within a time interval less than or equal to 2δ, since this is the maximum deviation in time among all the values as long as there is no failure. Majority in value domain is formed if at least two of the *timely* outputs have the same value.

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

The replicated nodes' output values are stored in local variables V1, V2 and V3. Values are assigned to these variables in the order of receiving inputs from the nodes (i.e., the first received value is stored in V1, the second one in V2 and the last one in V3). Two countdown timers, C1 and C2, initially set to 2δ, are used to keep track of feasible windows in order to identify majority in time domain.

The algorithm waits for the first node output to be delivered and then starts C1. It continues by waiting for the second node output and starts C2 upon its arrival. If both values have arrived before C1 expires, and match in the value domain, the voter will output the correct value. Otherwise we have two cases:

Case 1 C1 has not reached zero, but the values V1 and V2 do not match. In this case, the algorithm waits for V3 until C1 reaches zero. If the third value arrives before C1 has reached zero and matches either V1 or V2, the matching value is output since all values are within

Algorithm 1: VTV (executed in the voter)

```
/* Values from replicas are assigned to
   V1, V2 and V3 in the order of
   receiving them                        */
/* C1 and C2 are countdown timers        */
V1, V2, V3 = NULL;
C1, C2 = $2\delta$;
while V1 = NULL do wait;
start C1;
while V2 = NULL do wait;
start C2;
if $C1 > 0$ then
    if $V1 = V2$ then
        output V1;
    else
        while $C1 > 0$ and V3 = NULL do wait;
        if $C1 > 0$ and ($V3 = V1$ or $V3 = V2$) then
            output V3;
        else if $V3 <> NULL$ then
            signal disagreement;
        else
            while $C2 > 0$ and V3 = NULL do wait ;
            if $V3 = V2$ then
                output V3;
            else
                signal disagreement;
            end
        end
    end
else if $C2 > 0$ then
    while $C2 > 0$ and V3 = NULL do wait;
    if $V3 = V2$ then
        output V3;
    else
        signal disagreement;
    end
else
    signal disagreement;
end
```

the timeliness bound 2δ. In case of an assumption violation, e.g., $V1 \neq V2 \neq V3$ the algorithm signals *disagreement*. If the third value does not arrive before C1 reaches zero, the algorithm waits for V3 until C2 reaches zero. If V3 is received and matches V2 before C2 reaches zero, it is output. Otherwise the algorithm signals *disagreement*.

Case 2 C1 has reached zero. In this case, V1 is considered invalid, and the algorithm waits for V3 until C2 reaches zero, as only a match between V2 and V3 may result in an agreement. If the values do not match or V3 has not been received at all, the algorithm signals *disagreement*.

4. Conclusions

In this paper we have presented a new voting strategy called Value and Time Voting (VTV) for redundant systems, to explicitly consider both value and timing failures for achieving fault tolerance in real-time applications. Under specified failure assumptions, our method is capable of producing the correct output as well as identifying the correct window of time in which the output has to be delivered.

We have presented an algorithm for the particular case where one output is replicated in three different nodes, and illustrated the basic idea on how we perform the voting in both value and time domain.

Our ongoing research indicates that VTV, when used in the general case to mask 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*, April 2001.

[2] P. Ezhilchelvan, J.-M. Helary, and M. Raynal. Building responsive tmr-based servers in presence of timing constraints. *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, pages 267–274, 2005.

[3] H. Kopetz. Fault containment and error detection in the time-triggered architecture. *Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on*, pages 139–146, 2003.

[4] 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.

[5] 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.

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

[7] D. Powell. Failure mode assumptions and assumption coverage. *Proceedings of 22nd International Symposium on Fault-Tolerant Computing*, 1992.

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