A fault-tolerant voting scheme for multithreaded environments

Bernhard Fechner, Jörg Keller
FernUniversität Hagen, Fachbereich Informatik,
Lehrgebiet Parallelität und VLSI, 58084 Hagen
{Bernhard.Fechner|Joerg.Keller}@fernuni-hagen.de

Abstract

Voting schemes are widely used in fault-tolerant systems, mainly systems which imply temporal or component redundancy.

We present a voting scheme for multithreaded environments which is based on the observation that a fault-tolerant system which does not know its history can not distinguish between transient (SEUs) and permanent errors, caused by use of a faulty component. The history of errors is used to predict future errors and to determine if a permanent or transient error occurred. Only in the former case a repair is necessary; in the latter case recovery is sufficient. Using prediction and credibility points we are able to tell if a system failure is likely to occur soon. The more credibility a version has, the more likely it will compute a correct result. Therefore we can use credibility points in connection with thread priorisation to increase performance.

1. Introduction

Voting schemes are essential for fault-tolerant systems which imply temporal or component redundancy. Starting from simple spare systems up to triple modular redundant systems, the provision of redundancy, especially hot reserve, is useless if one cannot determine quickly enough which system is faulty. In the case of two systems, a state comparator can be used to detect errors. For three or more components, a voter is used. Simultaneous multithreading is a microarchitecture for superscalar processors which tries to hide latencies, e.g. caused by control dependencies by two or more threads either sharing the same hardware or using different hardware simultaneously. For this type of architecture both types of redundancy apply. On a simultaneous multithreaded processor execution units, register sets, etc. are replicated. With this on-chip level of parallelism, used of n threads for redundancy it closely matches the definition of an NMR (N-Modular Redundancy) system.

A voter for multithreaded systems must accomplish three main requirements in a short time:

- To detect and to report an error,
- to localize the source of the error (thread number, component, etc.) and
- reliably and correctly choose one or more non-faulty systems among n systems.

Let I be the set of inputs, R the set of outputs. To obtain fault tolerance by multithreading, $n \in IN$ versions $f_i : I \to R_n$, $i \in IN$ of the same algorithm are executed. We denote by a *version* a function or program which is executed in a hardware-thread. The final output is obtained by voting among the results from different threads. Generally a voter has to choose from a set of n functions $f_i \in F$ a set $C \subseteq F$ of one or more functions $f_j \in C$, $j \in IN_n$ which are correctly computed.

A comparison among various voting schemes is reported in [1]. We present an alternative fault-tolerant voting scheme for time-redundant systems such as virtual duplex systems for multithreaded environments [2].
The remainder of this paper is organized as follows. Section 2 describes the new predicative history voting. In Section 3 we suggest some sample application scenarios where our voting scheme can be used. Section 4 concludes the paper.

2. Predicative History Voting

The main idea behind predicative history voting is the observation that a fault-tolerant system which does not know its history can not distinguish between transient (SEUs) and permanent errors, caused by use of a faulty component. In [3] the history of an N-Modular Redundant (NMR) system helps to identify the most erroneous/faulty modules of a system. In our voting scheme a history is used to predict future errors, to determine if a permanent or transient error occurred and which thread is more likely to become faulty.

For the prediction, one can use neural nets, linear predicators or methods similar to branch prediction. The environment for the algorithm can be a network with an atomic broadcast capability and bounded message delay (e.g. a local area network, SMP, on-chip-wiring). It is assumed that a fair-use policy is enforced, so that no host can indefinitely block the broadcast medium [4]. We use majority voting as (meta-) voting algorithm. For clarity we look at two common examples.

Table 1 shows the structure of a history in the case of a permanent failure of version 2, including a transient error of version 3. Each column represents the result of version 1,2 and 3 at time t, encoded by the numbers 1,2 and 3, where *F* means a fault of unknown source, *P(Vx)* a permanent fault of version x and *T(Vx)* a transient fault of version x. *N* indicates no fault, *F(Vx)* a fault of version x (cause unknown) and p_t the threshold value for the interpretation of a permanent error. Equal numbers of different versions mean that the same result was computed. For each version, credibility points are used. The more credibility a version has, the higher the probability that it works correctly. The more faults a version has, the less credibility it gets. The row '*next'* gives information about the forecasted error type and version-number. The awarding of credibility points (cp) is done with the algorithm shown below. It can be easily extended to a n-component version.

```
∀i∈{1,2,3}; cp_i:=15;
while (execution) {
 if (f_i∉C): cp_i>>1
 else
 if (cp_i<15) cp_i++;
 if (cp_i<p_t): version i has a permanent
 failure;
 # change p_t for system tuning
}
```

Table 1. History with a permanent fault of V2 and a transient error of V3 (p_t=0)

Time	**1**	**2**	**3**	**4**
V1	1	1	1	1
V2	1	2	2	2
V3	1	1	1	1
Result	N	F(V2)	F(V2)	F(V2)
Cred.1	15	15	15	15
Cred.2	15	7	3	1
Cred.3	15	15	15	15
Next	N	N	F(V2)	F(V2)

Time	**5**	**6**	**7**
V1	1	1	1
V2	2	2	2
V3	1	3	1
Result	P(V2)	F/P(V2)	P(V2)
Cred.1	15	14	15
Cred.2	0	0	0
Cred.3	15	14	15
Next	P(V2)	P(V2)/T(V2)	P(V2)

At time 2 we get a fault of version 2, which can be identified by the 2-of-3 voter. When $cp_2==p_t$ version 2 turns into a permanent error. Although the error type of version 2 was identified as permanent, we are (in step 7) still able to recognize the transient failure of version 3 occurring in step 6. Table 2 shows a frequently occurring fault of version 3 which turns into a permanent failure.

Table 2. Frequently occurring fault of V3 – evolving into a permanent fault (p_t=0)

Time	**1**	**2**	**3**	**4**
V1	2	2	2	2
V2	2	2	2	2
V3	2	3	2	3
Result	N	F(V3)	N	F(V3)
Cred. 1	15	15	15	15
Cred. 2	15	15	15	15
Cred. 3	15	7	8	4
Next	N	N	F(V3)	N

Time	**5**	**6**	**7**
V1	2	2	2
V2	2	2	2
V3	2	3	3
Result	N	F(V3)	F(V3)
Cred. 1	15	15	15
Cred. 2	15	15	15
Cred. 3	5	1	0
Next	F(V3)	N	P(V3)

At times {2,4,6} we get a fault of version 3, which is identified by the 2-of-3 voter. Using prediction the faults at time 4 and 6 can be forecasted. When $cp_3==p_t$ version 3 turns into a permanent fault in step 7.

3. Applications

There exist many applications for our voting scheme, e.g. in situations where a speedy repair cannot be supplied after a fault. An example are soft mission critical systems, e.g. computers that serve scientific experiments on space missions. Here, a single experiment is not mission critical, its failure however still is expensive. In outer space transient faults are much more frequent due to radiation, and repair is impossible for obvious reasons. Also, due to increasingly smaller feature sizes, it is to be expected that transient faults due to radiation will occur much more frequently on earth missions, both in memories and in logic [5].

Since our voting scheme is general enough, it cannot only be used on a single multithreaded processor but also in symmetric multiprocessing or even in distributed computing. It could replace simple existing schemes in hardware. One example is the ABS (anti-blocking system) in a car.

Here components are triplicated, the results from various redundant sensors must be compared in real-time by a voter. If we would use standard majority voting in this case, we were only able to tell if an error occurred and which component caused the error. If we apply our voting scheme, we are furthermore able to tell:

- If a system failure is likely to occur in the future,
- if a faulty component which is accessed frequently causes an error,
- after an error, if the system will be able to work in real-time,
- if an error by a frequently accessed component or a transient error occurred,
- how to boost performance of threads or processes which show system-conformant behavior.

4. Conclusion/ extensions

We presented a voting technique which – in contrary to simpler voting schemes – is able to recognize and distinguish transient and permanent faults. Using prediction we are able to tell if a system failure will soon occur.

The more credibility a version has, the more likely it will compute a correct result. So the credibility points can be used for thread priorisation and to increase performance. The voter described in this paper can be combined with other voters to hybrid voting systems. Instead of credibility points, permanencies of observed permanent and transient errors can be used to increase prediction accuracy.

5. References

[1] Bass, J.M.; Latif-Shabgahi, G.; Bennett, S.: *Experimental Comparison of Voting Algorithms in Cases of Disagreement*, Proceeding of the 23rd Euromicro Conference, pp. 516-523, 1997

[2] Fechner, B.; Keller, J.; Sobe, P.: *Performance Estimation of Virtual Duplex Systems on Simultaneous Multithreaded Processors*. IPDPS2004, Workshop on Fault-Tolerant Parallel and Distributed Systems, 2004

[3] Latif-Shabgahi, G.; Bennett S.: *Adaptive Majority Voter: A Novel Voting Algorithm for Real-Time Fault-Tolerant Control Systems*, 25th Euromicro Conference (EUROMICRO '99), Vol. 2, p. 2113, 1999

[4] Tanenbaum, A.: *Computer Networks*, Prentice Hall, 1989

[5] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. *Modeling the effect of technology trends on the soft error rate of combinational logic*. In Proceedings IEEE International Conference on Dependable Systems and Networks (DSN'02), pp. 389–398, 2002