will consider running a third execution in order to make a majority vote on three results. We will identify criteria that the scheduler can use to determine whether it is safe to run another execution of a task, or if the node must be shut-down in order to prevent a fail-silence violation in either the value or the time domain.

Other key problems are how to implement the use of time-redundancy for the execution of the kernel program, and optimal placement of checkpoints.

Reviewer comment:

I find their terminology confusing, e.g., they use the terms static versus fixed priority scheduling without defining them. However, fixed priority scheduling is a type of static scheduling so this is confusing at best.

Response:

I use the same terminology as Tindell[3] and Poledna[4]. The term fixed priority scheduling is clearly defined by Tindells work. Both Tindell and Poledna use the term “dynamic” when scheduling decisions are made on-line, and the term “static” when schedules are calculated off-line.

I am aware that other researchers use the term “dynamic” exclusively to denote techniques that make scheduling decisions based on dynamic priorities, e.g. earliest deadline first (EDF) scheduling.

Reviewer comment:

I also don’t understand their claim that fixed priority scheduling and FT has not been researched; in what sense? I don’t agree with this at all;

Response:

I claimed that it has not been researched extensively, and this was perhaps a bit misleading. There has been several papers on scheduling theory and a few implementation studies, but there have been very few papers that have addressed the problem of validating the failure mode assumptions of fault-tolerant fixed-priority scheduling. In particular, I am not aware of any fault injection paper that has addressed this problem.

Reviewer comment:

You cannot generalize from 1 system.

Response:

Yes, it is difficult to draw general conclusions from experiments. But in most scientific disciplines experimentation is necessary in order to assess the accuracy of theoretical models, and to validate the assumptions on which models are based. Computer science is no exception. We expect that the fault injection methodology for parameter estimation and the scheduling techniques that we will develop in this project could be used in a wide range of systems that make use of behavior-based error detection and massive time-redundancy. We also believe that our results could provide valuable insight to scheduling theoreticians. However, the specific results regarding, for example, detection coverage or the probability for missing a deadline will of course only be valid for our target system.

References

[1] S. Ghosh, R. Melhem and D. Mossé, Fault-Tolerant Rate-Monotonic Scheduling, The Journal of Real-Time Systems, vol 15, no. 2, Kluwer Academic Publishers, pp. 149 - 181, Sept 1998.

[2] D. Powell, Failure Mode Assumptions and Assumption Coverage, Proc. of 22nd Int. Symp. on Fault-Tolerant Computing, IEEE Computer Society Press, Boston, MA, USA, pp. 386-395. July 1992.

[3] K.W. Tindell, Fixed Priority Scheduling of Hard Real-Time Systems, Ph.D. thesis, Univ. of York, UK, 1994.

[4] S. Poledna, Fault-Tolerant Real-Time Systems: The problem of replica determinism, Kluwer Academic Publishers, 1996.

CHALMERS TEKNISKA HöGSKOLA
Institutionen för datorteknik
Universitetslektor Johan Karlsson
412 96 Göteborg

Besöksadress: Hörsalsvägen 11
Telefon: 031 - 772 16 70
Telefax: 031 - 772 36 63
E-post: johan@ce.chalmers.se
www.ce.chalmers.se

Comments and clarifications in response to the reviewer comments concerning the project proposal “Node-level Fault-tolerance for Fixed Priority Scheduling” submitted to ARTES.

First I would like to clarify the overall goal of our research. We consider the problem of implementing fault-tolerant hard-real time system using low-cost techniques. We propose the combined use of behavior based error detection and massive time-redundancy. Recent developments in microprocessor technology have made it economically feasible to use high-performance microprocessors also in low-cost application; MIPS are cheap! This has made it possible to consider the use of massive time-redundancy to achieve high dependability in applications with low or moderate requirements of processing power. Massive time-redundancy makes use of double or triple execution of tasks, checkpoints, and rollback or roll-forward recovery techniques. These techniques have previously not been commonly used for hard real-time systems - except for double execution - because they impose a rather large time overhead.

Reviewer comment:

I don’t find any novelty in this proposal. The problem itself is not motivated. There are few details about the fault tolerance techniques to be studied (no novelty?), and even if they accomplish what they say I don’t really see much as an outcome. I feel that they failed to motivate any key problems here.

Response:

The key research problem that we intend to investigate is validation of the failure mode assumptions made in fault-tolerant fixed-priority scheduling.

There has been a few papers that have described how time redundant execution can be incorporated into fixed-priority scheduling. A recent paper by Ghosh et al. [1] considers time redundancy in rate-monotonic scheduling. The authors have citations to only three other papers that have addressed this problem.

Most papers dealing with fault-tolerant scheduling rely on naive assumptions regarding system behavior under the presence of faults. It is often assumed that error detection coverage is perfect, error detection latency is negligible, or that recovery time is deterministic or negligible, etc. These assumptions are not realistic, especially when low-cost fault-tolerance techniques are used. To deal with this kind of uncertainty Powell has defined the notion of *assumption coverage* [2]. In reality error detection coverage, error detection latency, and recovery time are random variables that depend on where and when a fault occurs, and how the workload exercise the hardware (workload dependency).

We intend to use fault injection to estimate assumption coverage, error detection coverage, error detection latency, recovery time, etc, and create a model in which these parameters are used to estimate the probability that any task will miss its deadline given that a fault has occurred. I am not aware of any experimental research that has addressed this important problem.

Another key problem is the trad-off between the probability of fail-silence violations in the value domain and the probability of missing a deadline. We will use double execution of tasks and result comparison to detect data errors that escape the behavior based error detection mechanisms (e.g. hardware exceptions). However, our previous research has shown that up to 95% of transient faults can be detected by behavior based mechanisms. When an error is detected by these mechanisms in, say, the second execution of a task, we have the possibility to use the result of the first execution, without re-executing the second execution. This will decrease the probability of missing a deadline, but it will increase the probability of producing erroneous results (because we rely on a single instance of the result).

We intend to investigate different scheduling policies that make on-line decisions about whether to run single or double execution of tasks that has been affected by a fault. For data errors detected by result comparison, we

CHALMERS TEKNISKA HöGSKOLA
Institutionen för datorteknik
Universitetslektor Johan Karlsson
412 96 Göteborg

Besöksadress: Hörsalsvägen 11
Telefon: 031 - 772 16 70
Telefax: 031 - 772 36 63
E-post: johan@ce.chalmers.se
www.ce.chalmers.se