Towards Energy-Aware Software-Based Fault Tolerance in Real-Time Systems[*]

Osman S. Unsal, Israel Koren, C. Mani Krishna
Architecture and Real-Time Systems Laboratory
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003
{ounsal,koren,krishna}@ecs.umass.edu

ABSTRACT

Many real-time systems employed in defense, space, and consumer applications have power constraints and high reliability requirements. In this paper, we focus on the relationship between fault tolerance techniques and energy consumption. In particular, we establish the energy efficiency of Application Level Fault Tolerance (ALFT) over other software-based fault tolerance methods. We then develop sensible energy-aware heuristics for ALFT schemes. The heuristics yield up to 40% energy savings.

1. INTRODUCTION

There is an increasing number of real-time applications that require fault-tolerance. Traditionally, fault-tolerance has been implemented using massive redundancy, by duplicating or triplicating the hardware. Such an approach obviously consumes huge amounts of energy. Recently, fault-tolerance approaches have emerged, which are less energy-intensive. One of these is Application Level Fault-Tolerance (ALFT), where information available at the application level is exploited to reduce the overhead imposed by fault-tolerance [1, 2]. In this paper, we explore the energy costs of ALFT.

Real-time systems have two attributes that set them apart from systems built for general purpose computing: (a) timeliness and (b) fault-tolerance. Real-time applications are time-constrained, i.e. the tasks have deadlines by which they have to finish execution, this requires a deterministic task response time. This is enforced through task scheduling algorithms, developed specifically for real-time applications, that guarantee timeliness. The other attribute of real-time systems is fault-tolerance, i.e. a real-time system should continue to operate correctly in the presence of faults. Fault-tolerance could be achieved through replicated execution of tasks. Power and energy analysis related to scheduling and timeliness of real-time systems have been extensively investigated, usually in the context of Dynamic Voltage Scaling (DVS) [3, 4, 5, 6, 7, 8]. However, there is no previous attempt at power and energy analysis of fault-tolerance mechanisms. To the best of our knowledge, this is the first research effort in that direction.

This paper is organized as follows: In Section 2, we examine software-based fault tolerance. In Section 3, we introduce our system and energy model. In Section 4, we discuss our results and we present our conclusions in Section 5.

[*]Supported in part by NSF grant EIA-0102696.

ISLPED'02, August 12-14, 2002, Monterey, California, USA.
Copyright 2002 ACM 1-58113-475-4/02/0008 ...$5.00.

2. SOFTWARE BASED FAULT TOLERANCE

Real-time systems, by definition, have to be reliable. The reliability of a system is linked directly to its ability to operate correctly despite the presence of faults [9]. The objective of fault tolerance techniques is to minimize the effects of faults on system operation. The additional overhead of employing fault-tolerance has energy implications, and depending on various system attributes, most notably power-aware task scheduling heuristics, the amount of power saved can be substantial, while preserving the fault-tolerance attributes. Initially, passive hardware-based redundancy approaches, such as Triple Modular Redundancy were used to ensure fault tolerance. More recently, software-based redundancy methods became preferable due to their "lightweight" characteristics. Here, we focus on such a fault tolerance scheme, the Application-Level Fault Tolerance [1, 2] (ALFT), which is an amalgam of time and software redundancy. ALFT encompasses redundancy and recovery actions within the application software. We adopt the approach taken by Haines et al. [2], according to which the task set consists of primary and secondary tasks. Secondary tasks may be identical to the primary tasks or they can be a scaled down version of them. Typically two techniques are widely used for secondary scaling: *resolution reduction* or *precision reduction*. *Resolution reduction* is more suitable for such applications as image processing or FFT, in which an iterative calculation can be scaled to $1/n$ of its size by computing every nth point and interpolating the points in between. By contrast, in *precision reduction* the scaling is achieved through ending the iterative computation earlier than the primary version. *Precision reduction* is more suitable for Increased Reward with Increased Service (IRIS) type applications such as the calculation of π or sinusoidal operations, in which the iterative computation can be aborted at any point, with appropriately reduced precision. For reliability purposes, primary and secondary tasks are assigned to run on different processors. Upon the failure of a primary task, the output of the secondary task is used. To save computing resources, a secondary task can be aborted if the primary successfully finishes its execution [10]. This can be achieved through two different but functionally equivalent mechanisms. First, in a message passing based (such as MPI) implementation of ALFT, the secondary task can be aborted by sending an abort message from the primary when it successfully finishes its execution. Second, an OS supplied abort software interrupt can be sent to the secondary when the primary finishes. In contrast, in a software-based task duplication fault-tolerance scheme, no such mechanism exists, and the secondary is executed to completion even if the corresponding primary finishes. We will compare the energy-efficiency of ALFT and task duplication in Section 4.

3. SYSTEM AND ENERGY MODEL

We consider distributed real-time systems: the processors are loosely coupled, see Figure 1. Each node has its private memory and each task has an associated worst-case execution time and deadline. The system is a "hard" real-time system, i.e., the missing of a task deadline due to a fault can be catastrophic. We assume that the tasks are independent of each other. The number of tasks, the fault-tolerance mechanisms, the granularity of secondaries and the task scheduling mechanism are some of the parameters in our study. Task allocation heuristics are of secondary importance for software-based fault tolerance and are not considered here. We only need to make sure that primary and secondary tasks are allocated to different processors.

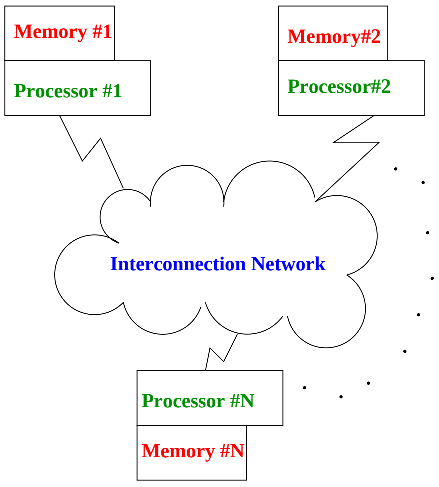

Figure 1: The System Model

As observed by Rotenberg [11], it is difficult to find standard real-time benchmarks. This difficulty is even more pronounced for real-time benchmarks which consist of multiple tasks, although most actual real-time applications consist of multiple tasks [9]. We therefore use randomized task sets averaged over multiple runs as well as available real-time target tracking [12] and Asymmetric Digital Subscriber Line (ADSL) [13] applications.

The distribution of energy consumption over the execution time of applications has been found to be independent of circuit state and more or less uniform for general purpose processors [14]. However, for other classes of processors, such as DSP's [15] the energy dissipation depends on the circuit state and thus can vary significantly over the execution time. Since general purpose COTS processors are considered for software-based fault-tolerance, we assume that the energy consumption is linearly proportional to the overall actual load, i.e., the more secondary tasks execute, the greater the energy consumption. To save energy, we want to refrain from executing the secondary tasks, but at the same time we will preserve the fault tolerance attributes of ALFT.

4. RESULTS

In this section, we study the energy implications of various fault tolerance techniques. Our intuition is that not all fault tolerance schemes have the same energy requirements. We show that some schemes, such as the ALFT, consume less energy than others. Specifically, we compare software-based task duplication with application level fault-tolerance to serve as a motivating example. We then introduce energy-efficient scheduling heuristics for ALFT. Those heuristics attempt to finish executing primaries as early as possible, while delaying executing secondaries as much as possible. This results in less secondaries having to run, thereby saving energy. In Section 4.1, we provide a simple motivational example. In Sections 4.2 and 4.3, we introduce suitable heuristics and confirm their energy-efficiency through experiments with random task sets as well as a multiprocessor Asymmetric Digital Subscriber Line (ADSL) modem application. In these sections, unless otherwise stated, each experiment is replicated and results averaged for 60 feasible task sets with random periods and for different system loads. For fault-tolerance purposes we consider the worst case load, i.e. the system load if all the secondaries were activated. For each load, the task execution times are chosen accordingly. The period of the system is the Least Common Multiple (LCM) of the task periods.

4.1 Motivating Example

We assume failures are sufficiently infrequent and therefore the energy impact of fault detection is negligible. As such, we are interested in the energy signature of the fault tolerance method. We start with establishing the energy efficiency of ALFT over a more traditional fault tolerance scheme: task duplication. As our application we will consider the Real-Time Multi-Hypothesis (RTHT) [12] benchmark from the DARPA Real-Time benchmark suite. This is a general-purpose, parallel, target-tracking benchmark. The task structure is simple, all tasks have the same deadline and execution time. The ALFT approach has been applied to this benchmark, and results indicate that all targets are correctly tracked when the secondary duplicates only 15% of the primary's computation [10]. Therefore, the secondary tasks' contribution to system execution load is 15% of the primary load in the worst case. As a comparison, we have to execute the secondary copy of a task completely in task duplication; this implies that secondary tasks contribute as much as the primary tasks to the system load. This leads to the conclusion that ALFT is 42.5% more energy efficient than task duplication for this benchmark.

4.2 A Simple Heuristic: SEF

In our previous motivating example, we used the earliest-deadline-first (EDF) scheduling algorithm. We now ask the following question: "Is there a better scheduling algorithm

```
Step 1. Schedule all primaries and secondaries using EDF.

Step 2. Schedule just primaries on all processors to get $\Omega_{i,j}$.

Step 3. Set release time of secondary tasks at: $R_{i,j} = min(\Omega_{i,j}, d_{i,j} - e_i)$.

Step 4. Schedule all primaries and secondaries. If feasible stop.

Step 5. Schedule just primaries on all processors to get updated $\Omega_{i,j}$.

Step 6. Do a directed search to modify $R_{i,j}$. Go back to step 3.
```

Figure 2: The flow of the SETS algorithm. Here, each task i is iteratively executed and we denote by $d_{i,j}$ the deadline for the j'th iteration of task i, e_i the worst case execution time of task i and $\Omega_{i,j}$ is primary i's j'th iteration finish time.

for fault-tolerant energy efficient systems?" We know we should execute the primaries as early as possible to be energy efficient. To this end, the shortest-execution-time-first (SEF) seems to be a good candidate. In uniprocessors, SEF is optimal in terms of minimizing response times [16]. It is this property that lends itself to energy savings in our primary/secondary task scheme, and therefore we expect SEF to be a "good" power-aware heuristic for software-based fault-tolerant systems. We next compare EDF against SEF as well as ALFT against task duplication for a two-processor system. We consider secondaries which are either 100% or 50% of the primaries and the results in Table 1 are averaged for 44%, 66% and 88% system load. The energy results in Table 1 are normalized with respect to task duplication with EDF. Compared to EDF, in SEF, more primaries finish earlier, eliminating the need to activate secondaries. Consequently, SEF is up to 19% more energy efficient than EDF.

	Power-Unaware	Power-Aware(ALFT)	
	Task Duplication	100%	50%
EDF	100	96	77
SEF	83	77	61

Table 1: Relative energy consumption of power-unaware task duplication vs. power-aware ALFT fault tolerance policies.

4.3 An Advanced Heuristic: Secondary Execution Time Shifting (SETS)

In Section 4.2, our objective was to finish the execution of primary tasks as soon as possible to save energy. One issue with the previous heuristic is that a given task set that is feasible with EDF might not be feasible with SEF. Here, we develop a dual strategy that is more robust: we want to delay execution of secondaries as much as possible to save energy. We initially start with a feasible task set employing EDF and aggressively modify the release time of the secondaries to lead to a more energy-efficient schedule. As such, our approach is to search for a better static schedule. We term this heuristic Secondary Execution Time Shifting (SETS). We provide below a more formal discussion and explanation of SETS.

The search for an energy-efficient schedule can be formulated as an optimization problem. Let $\sigma_{i,j}$ denote the start time of iteration j of secondary i and let $\pi_{i,j}$ denote the finish time of iteration j of primary i. Note that we focus on periodic real-time tasks with a new iteration being executed before the task's deadline in every period. The optimization problem is:

Schedule the tasks to minimize

$$\sum_{i=1}^{n\_tasks} \sum_{j=1}^{n\_iter(i)} \boldsymbol{\theta_{i,j}}$$

subject to all deadlines (primary *and* secondary) being met, where $\theta_{i,j} = \pi_{i,j} - \sigma_{i,j}$ is the overlap between the execution of the primaries and their corresponding secondaries. Minimizing the overlap will also minimize the energy consumption.

This optimization problem is NP-complete for tasks with arbitrarily chosen periods and execution times, so an efficient heuristic is needed. We developed such a heuristic, which delays the release time of the secondary task as much as possible, thereby minimizing the overlap θ and making it more likely for the secondary task not needing to be run and save energy. SETS produces an energy-efficient static schedule. See Figure 2 for the SETS algorithm. In the preparatory steps that are also common to energy-unaware EDF ALFT, we randomly select the period and execution time of primaries and scale secondaries using either *resolution* or *precision* reduction. We then randomly allocate primaries and secondaries to processors. The primary and secondary versions of a task are allocated to different processors for fault tolerance. We iterate those preliminary steps until we obtain, in Step 1, a feasible allocation of the task set with all secondary task release times set at the beginning of their respective periods. In step 3, the iterative SETS heuristic starts by moving the secondary release times as close to the secondary tasks' deadline as possible. In step 4, the feasibility of the modified schedule is checked. If infeasible, the nearest idle cycle to deadline miss is recorded and the release time of the secondary task active is relaxed up to the nearest idle cycle in step 6. This greedy process is repeated until a feasible state is reached, i.e., one that does not have any deadline misses. Note that this process is guaranteed to converge to a feasible schedule since the initial schedule is feasible. The output of SETS is the list of modified secondary task release times.

We first briefly examine a multiprocessor Asymmetric Digital Subscriber Line (ADSL) modem application. The application consists of 14 tasks, for the sake of brevity, we refer the reader to [13] for specific task deadlines and execution times. This type of application lends itself well to secondary scaling through *resolution reduction*. We therefore calculate every other second or third point to scale down the secondary task size and interpolate the points in between. Calculating every second or third point results in a secondary size of 50% or 33% of the primary, respectively. The results, in terms of percent of energy saved through SETS compared to the baseline case of EDF with the same secondary size, are shown in Table 2. Note that even for a simple application such the ADSL, the energy savings are substantial.

Secondary Size(%)	Energy Savings(%)
33	09.51
50	13.38
100	19.44

Table 2: Percent energy savings for the ADSL application. The secondary size is expressed relative to the primary task size.

In the detailed analysis we examine random task sets. This time, we assume the task set to be composed of Increased Reward with Increased Service (IRIS) type tasks [17]. This allows the use of the *precision reduction* technique for secondary scaling, therefore, the secondary calculation can be stopped at any point, with appropriately reduced precision. Secondary sizes of 60%, 80% and 100% are assumed for this study. The baseline system has 20 tasks, and six processors. The energy savings are with respect to baseline energy-unaware ALFT using EDF. We first study the impact of secondary task size on energy savings. The results are shown in Figure 3. Due to the efficiency of SETS, the savings are substantial even for higher system-loads and lower secondary sizes.

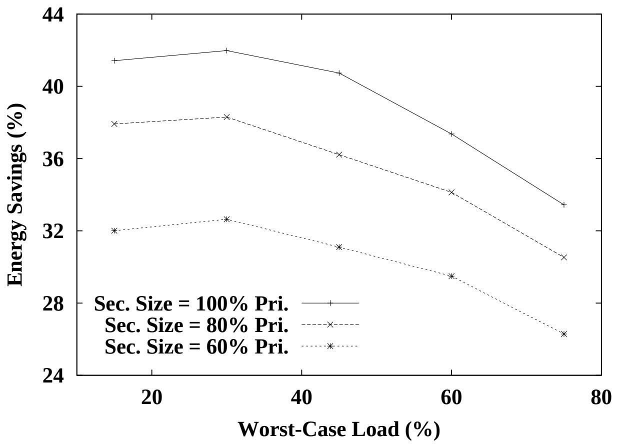

Figure 3: Energy Savings for different secondary sizes (20 tasks).

The energy savings are due to SETS' ability to decrease the overlaps. We have therefore analyzed the reduction in primary and secondary overlap due to SETS. Figures 4 and 5 show this reduction for different secondary sizes and for two task granularities (20 tasks in Figure 4 and 50 tasks in Figure 5). The results indicate that the overlap reduction due to SETS is quite high suggesting that near optimal behavior can be expected with SETS. We can also see that the efficiency of SETS is almost independent of the secondary size but is highly dependent on the task granularity. This dependence is more pronounced at high system loads for coarse task granularities. This is to be expected since there is more task overlap, for the same load, for coarser tasks which have comparatively longer execution times. At high system loads, it also becomes more difficult to find idle cycles in the tighter schedule to shift the secondary task release times.

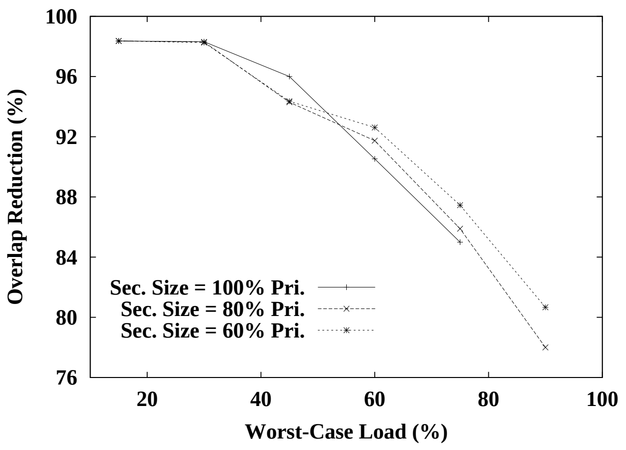

Figure 4: Overlap reduction for different secondary sizes (for 20 tasks).

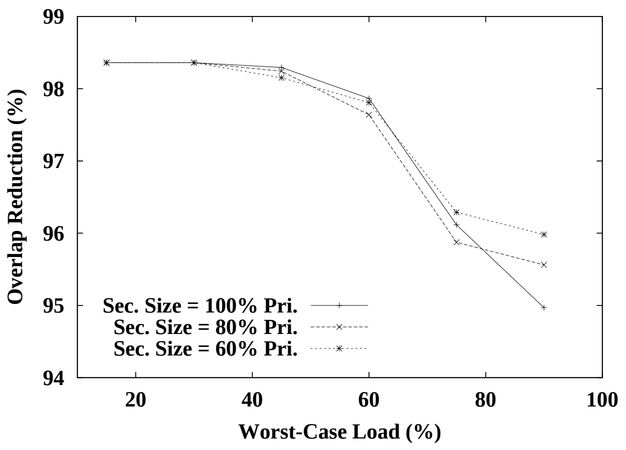

Figure 5: Overlap reduction for different secondary sizes (for 50 tasks).

To further check the impact of task granularity on the effectiveness of SETS we have analyzed the resulting energy savings when the workload is divided among 10 to 60 tasks. Our results in Figure 6 indicate that tasks with finer granularity do better than tasks with coarser granularity. This is to be expected, since the overhead can not be reduced by

SETS beyond a certain limit if tasks are of coarser granularity, see Figure 7. Note that this would even be the case for an optimal scheme.

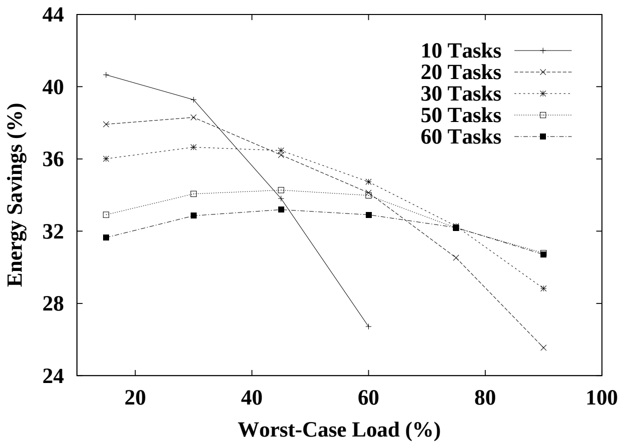

Figure 6: The effect of task granularity on energy savings for 80% secondary size.

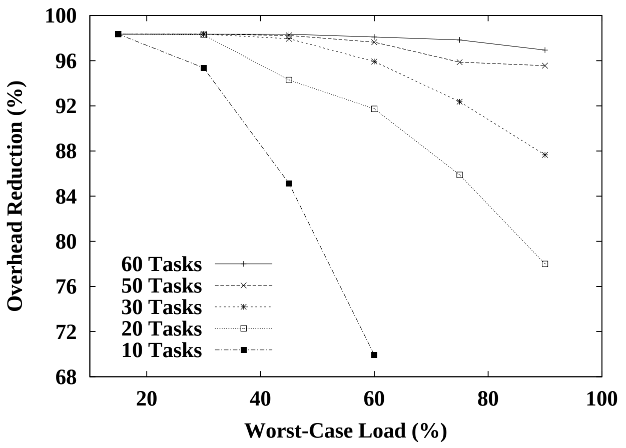

Figure 7: The effect of task granularity on overlap reduction for 80% secondary size.

5. CONCLUSION

We have examined, for the first time, the energy-signature of various fault-tolerance techniques. We introduced energy-aware fault-tolerance heuristics that lead to substantial energy savings: up to 40% depending on the configuration. Energy awareness in fault-tolerant real-time systems is a new research area; one possible future research direction includes examining energy-aware checkpointing schemes.

6. REFERENCES

[1] A. Beguelin, E. Seligman, and P. Stephan. Application level fault tolerance in heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 43(2):147–155, June 1997.

[2] J. Haines, V. Lakamraju, I. Koren, and C.M. Krishna. Application-level fault tolerance as a complement to system-level fault tolerance. *The Journal of Supercomputing*, 16:53–68, 2000.

[3] Trevor Pering and Robert Brodersen. Energy efficient voltage scheduling for real-time operating systems. In *4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session*, June 1998.

[4] C.M. Krishna and Y.-H. Lee. Voltage-clock-scaling techniques for low power in hard real-time systems. In *IEEE Real-Time Technology and Applications Symposium*, pages 156–165, May 2000.

[5] V. Swaminathan and K. Chakrabarty. Real-time task scheduling for energy-aware embedded systems. In *IEEE Real-Time Systems Symposium, Work in Progress Session*, November 2000.

[6] F. Gruian. Hard real-time scheduling for low energy using stochastic data and dvs processors. In *International Symposium on Low-Power Electronics and Design ISLPED'01*, August 2001.

[7] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th ACM Symposium on Operating Systems Principles*, October 2001.

[8] H. Aydin, R. Melhem, D. Moss, and P.M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Real-Time Systems Symposium RTSS'01*, December 2001.

[9] C.M. Krishna and K.G. Shin. *Real Time Systems*. Mc Graw Hill, 1997.

[10] J. Haines, V. Lakamraju, I. Koren, and C.M. Krishna. Development of application-level fault tolerance in a real-time benchmark. In *Proceedings of EFTS'98, IEEE Workshop On Embedded Fault-Tolerant Systems*, pages 28–33, May 1998.

[11] Eric Rotenberg. Using variable-mhz microprocessors to efficiently handle uncertainty in real-time systems. In *34th Annual International Symposium on Microarchitecture, Micro-34*, pages 28–39, December 2001.

[12] D.A. Castanon and R. Jha. Multi-hypothesis tracking (draft). *DARPA Real-Time Benchmarks, Technical Information Report (A006)*, 1997.

[13] P. Yang et al. Energy-aware runtime scheduling for embedded-multiprocessor socs. *IEEE Design and Test of Computers*, 18(5):46–58, September 2001.

[14] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration Systems*, 2(4):437–445, December 1994.

[15] M.T. Lee, V. Tiwari, S. Malik, and M. Fujita. Power analysis and minimization techniques for embedded dsp software. *IEEE Transactions on Very Large Scale Integration Systems*, 5(1):123–133, March 1997.

[16] M. Newman. http://guir.cs.berkeley.edu/projects/osprelims/summaries/concurrency.html.

[17] J.K. Dey, J.F. Kurose, D. Towsley, C.M. Krishna, and M. Girkar. Efficient on-line processor scheduling for a class of iris (increasing reward with increasing service) real-time tasks. In *ACM SIGMETRICS Conference*, pages 217–228, May 1993.