Dynamic Fault-Tolerance and Metrics for Battery Powered, Failure-Prone Systems

Phillip Stanley-Marbell, Diana Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890

{pstanley, dianam}@ece.cmu.edu

ABSTRACT

Emerging VLSI technologies and platforms are giving rise to systems with inherently high potential for runtime failure. Such failures range from intermittent electrical and mechanical failures at the system level, to device failures at the chip level. Techniques to provide reliable computation in the presence of failures must do so while maintaining high performance, with an eye toward energy efficiency. When possible, they should maximize battery lifetime in the face of battery discharge nonlinearities. This paper introduces the concept of *adaptive fault-tolerance management* for failure-prone systems, and a classification of local algorithms for achieving system-wide reliability.

In order to judge the efficacy of the proposed algorithms for dynamic fault-tolerance management, a set of metrics, for characterizing system behavior in terms of energy efficiency, reliability, computation performance and battery lifetime, is presented. For an example platform employed in a realistic evaluation scenario, it is shown that system configurations with the best performance and lifetime are not necessarily those with the best combination of performance, reliability, battery lifetime and average power consumption.

1. INTRODUCTION

New applications of VLSI technology pose many challenges for existing CAD methodologies. The emergence of power consumption as a critical system design constraint, particularly in battery powered computing systems, led to the development of a slew of techniques for *power management*. These efforts have further been bolstered by recent attention to the effect of application profiles on battery lifetimes. Progress in device technologies, coupled with reduction in costs, is enabling new classes of applications, such as sensor networks, and large-area surfaces with embedded computation, sensing and actuation capabilities [13]. Such flexible substrates may contain 100's or 1000's of low power microcontrollers embedded per m^2, with power distribution and communication fibers, and actuation capabilities in the form of *shape-memory-alloys* embedded in the substrate [14]. These emerging technologies pose new CAD challenges in much the same way that the burgeoning portable computing device market caused increased attention to power management techniques and algorithms.

A new dimension in requirements, is that of *reliability in the presence of runtime failures*. Runtime failures may be due, for example, to intermittent electrical failures arising from wear and tear in a wired sensor network embedded in a flexible substrate. They may likewise be due to changes in weather conditions in a networked sensor deployed outdoors. Failures due to the depletion of energy resources (such as batteries), although often predictable, may also occur.

CAD methodologies and algorithms which address a combination of issues of runtime failure of components and their induced performance and power consumption overheads, are therefore essential. Providing fault tolerance usually entails providing *redundancy*—in the presence of failures, redundant devices or spares are employed to provide correct system behavior. Invariably, portions of the executing application must be moved from failing devices to redundantly deployed ones. In *gracefully degrading systems*, the redundantly deployed devices are also employed for computation, providing better system performance in the absence of failures. In order to judge the efficacy of proposals in an appropriate design methodology framework therefore, metrics which combine energy efficiency, performance, reliability and battery lifetime (taking into consideration nonlinearities in battery and DC-DC converter characteristics) are required.

Contributions

This paper introduces the concept of *dynamic fault-tolerance management (DFTM)*, presented from the viewpoint of failure-prone battery powered systems with energy constraints. The proposal is intended to provide for failure-prone systems, what *dynamic power management* provides for energy-constrained systems. Unlike energy resources, adapting to failure requires the harnessing of fault-free devices as surrogates for failing ones. In order for such dynamic fault-management algorithms to be implemented across devices in a system, without requiring global coordination, they must employ purely local information and decisions to control the system-wide adaptation.

In order to determine the efficacy of different configurations of DFTM, we employ results from traditional performance-related reliability measures (generally referred to as *performability measures*) as well as introduce new measures that incorporate energy efficiency, battery discharge effects, performance and reliability, which will henceforth be referred to as the *ebformability measures*[1]. The effectiveness of the proposed DFTM approach, as well as the benefits of employing ebformability measures in a design methodology framework for emerging platforms, is verified through a detailed simulation study. The simulation framework employed, models computation (at the instruction level), communication (at the bit level), runtime failures in both communication and computation, power consumption, and battery discharge effects.

[1]The name *ebformability* was chosen to denote the combination of *e*nergy-efficiency, *b*attery lifetime, per*form*ance and reli*ability* measures.

The remainder of the paper begins with a survey of related research in Section 2. A description of DFTM is presented in Section 3, followed by a derivation of the *ebformability measures* in Section 4. Section 5 presents an experimental evaluation of the posited ideas, and the paper concludes with a summary of the key contributions, in Section 6.

2. RELATED RESEARCH

Particular attention has been paid to average power, peak power, energy consumption, as well as to metrics that combine the above with performance measures, such as the energy$\times$delay and energy$\times$(delay^2) metrics. These metrics have enabled the developers of CAD tools to ascertain the relative benefits of algorithms and implementations of hardware, in terms of both performance and power/energy consumption. There have been previous efforts in providing reliable computational substrates out of possibly unreliable components, dating back to von Neumann's seminal work [17]. A significant body of research has addressed combined performance and reliability measures [1], but there hitherto have been no contributions in the area of measures that combine performance, power consumption, battery lifetime and system reliability. Analytic and simulative models for battery life estimation [8, 2] provide means of determining which application workloads will provide longer battery system lifetime, but they neither provide a combined measure of battery life and performance nor a measure that takes into account battery life, performance *and* reliability.

Unlike efforts aimed at providing guarantees in system performance, employing a central layer of control [7], the proposal of this work, is to provide general fault-tolerance management, *using only identical local algorithms* at each node in a network. Without a central point of control, a challenge is to provide resilience to faults on the macro scale, from decisions performed at individual nodes.

The application domains that stand to benefit greatly from both techniques for reliable computation in the presence of failures, and metrics for judging the efficacy of such techniques, are the emerging technologies of sensor networks as well as wired sensor networks such those embedded into flexible substrates.

3. DYNAMIC FAULT-TOLERANCE MANAGEMENT

In traditional low power and portable computing systems, *dynamic power management* [9] exploits variations in application requirements, to adjust performance and power consumption of a system to application behavior. By employing simple rules and predictions (e.g., if the system has been idle for x minutes, spin down the disk), traditional power management techniques enable longer lifetime in the presence of workload variations and energy resource constraints. *Dynamic fault-tolerance management (DFTM)*, proposed in this paper, aims to encompass a broader range of constraints besides power consumption—to take advantage of variations in both *application* and *environment* behavior, enabling maximal application lifetime in the presence of both energy and reliability constraints, with a possible tradeoff for performance. Environment behaviors may include not just limited energy resources and battery performance, but also runtime failures, which (although not prevalent in traditional computing systems), will be a key consideration in emerging technologies. In a battery powered networked system that may witness a large number of runtime failures, for example, DFTM must determine actions to be performed to maximize system lifetime.

Unlike in the case of power constraints where decisions are based solely on local resources, failures in individual components are generally addressed by providing an external component—*redundancy*—such that failing components may be superseded by functional ones. Approaches to *manage fault tolerance* as opposed to those that *manage power consumption*, must therefore consider reconfiguration of system resources.

A structured approach to these reconfiguration decisions will be essential for tractability in defining algorithms for fault-tolerance management. In this work, we propose a structuring of algorithms which provide, in addition to local decisions (exemplified by traditional power management techniques), a framework for system reconfiguration algorithms. Prior research in reconfiguring arrays of computing systems in the presence of faults [4] has shown that the worst case slowdown due to reconfiguration (e.g., moving from the use of a failing resource to a redundantly deployed functional one) is linearly dependent on three factors:

- *Load*, l: This reflects the increase in resource utilization at a processing device, as a result of a system reconfiguration.
- *Dilation*, d: Denotes the increase in communication latency between processing elements, resulting from a system reconfiguration.
- *Congestion*, c: This is a manifestation of the decrease in system bandwidth resulting from a system reconfiguration.

Algorithms to be enabled at each node[2] in the system, must enable the minimization of the effects of *load*, *congestion* and *dilation*, minimizing the system slowdown, but without requiring global coordination between devices. Dynamic fault-tolerance management, as proposed in this work, therefore consists of three components:

- **Local Decision Algorithms (L-Class)** : *Changing the local behavior* of a node, such that the node's behavior with respect to its neighbors decreases one or all of the congestion, c, load, l or dilation, d. For example, a decision of whether or not to forward data between links to which a node in a network is connected, will affect congestion and dilation.
- **Re-mapping Decision Algorithms (M-Class)** : *Determining when to adapt* the system configuration, employing fault-free devices as surrogates for failed ones.
- **Re-mapping Destination Algorithms (D-Class)** : *Finding the appropriate alternate system configuration*. For example, experiencing excessive communication errors at a node in a network, may necessitate transfer of execution to an alternative redundantly deployed device, which provides an alternative error-free communication path.

In this paper, a specific implementation of the aforementioned classes of algorithms is presented, targeted at battery powered networked embedded systems, comprised of large

[2]In the remainder of the paper, the terms "node" and "device" are used interchangeably to refer to a processing device. Likewise, "link" and "network segment" are used interchangeably to refer to the communication interconnect that may link two or more processing devices.

Table 1: A possible set of DFTM Algorithms. Local algorithms (L0–L2), re-mapping decision algorithms (M0–M3) and migration destination algorithms (D0–D4).

Algorithm	Description
L0	Do not forward packets.
L1	Do not accept migrating applications.
L2	Never cache information.
M0	Battery too low : re-map.
M1	Too many node faults : re-map.
M2	Too many collisions : re-map.
M3	To many carrier sense errors : re-map.
D0	Pick a random redundant node to re-map to.
D1	Migrate to redundant node with lowest number of collisions.
D2	Migrate to neighbor with lowest number of carrier-sense errors.
D3	Migrate to neighbor with most energy.
D4	Migrate to redundant node with most direct links to communication target.

numbers of nodes, with a fraction of the nodes being redundantly deployed. In the presence of failures, execution of components of an application may be relocated to these redundant devices. Since multiple algorithms are defined for each class, with the possibility that multiple algorithms might be relevant at the same instant (i.e., algorithms might not necessarily be orthogonal in some settings), it will be necessary, where appropriate, to define priorities for the different algorithms. Such a priority scheme is not pursued in this work, and is a direction for future research.

To adapt a system to time-varying failure rates and modes, in order to provide *fault-free macroscopic behavior* from a *faulty substrate*, it is essential to perform on-line monitoring of failures. For a networked system consisting of battery powered devices, with high probabilities of failures in the interconnection links and devices, the statistics that may be monitored include: (1) *Remaining battery capacity*, (2) *Link carrier sense errors*, (3) *Link collisions* and (4) *Node faults*.

Table 1 provides an example instantiation of simple DFTM algorithms for a system with the aforementioned failure statistics monitored. Nodes must employ heuristics to ascertain these properties at their neighbors, based on locally measured values. For example, in the case of a device connected to multiple communication links, if one of those links is experiencing a significant number of failures, the device can infer that nodes attached to that same interface will be experiencing similar conditions. Since all the DFTM algorithm classes employ only information from a device's immediate neighbors, the required information could also be queried from these nodes if necessary.

3.1 L-Class Algorithms

The local decision algorithms or *L-class* algorithms aim to adapt the execution of applications to prevailing conditions, without performing application re-mapping. The L0 algorithm, which determines whether or not nodes forward packets, has a direct effect on the performance of the system as a whole, while minimizing work performed locally, and hence extending the local lifetime. If there exists a node in the system whose only communication path is through a node with the L0 algorithm enabled, for example, due to failures in its other links, then such a node will be effectively disconnected from the network. Thus, rather than greedily enabling local decision algorithms to minimize consumption of local energy resources, devices must take into consideration the role they play in the system as a whole. Rather than permanently enabling L0 to conserve energy resources, a node in a network may periodically or randomly enable this algorithm. The L1 algorithm is relevant to redundantly deployed nodes in a system. A node with L1 enabled will not permit applications to be re-mapped onto it. This can be desirable if it is more important for the node to use its energy, for example, to forward packets. Finally, the L2 algorithm determines whether or not a node should cache information. Caching data might reduce the need to communicate in some applications, but can constrain nodes from aggressive power management—e.g., going into a deep sleep mode might lead to loss of such cached data, thus the requirement to cache data might preclude devices from entering a deep-sleep state. The particular L-class policies listed above are only relevant in a system which supports multi-hop communication, and are not investigated further in this paper. They are the subject of our current investigations.

3.2 M-Class Algorithms

In systems which contain redundantly deployed nodes, it is possible to re-map execution of applications from nodes witnessing adverse conditions to those experiencing more favorable conditions. The *M-class* algorithms determine *when* to perform such re-mapping. In the particular instantiations listed in Table 1, the M0 algorithm specifies to attempt to re-map an executing application when battery levels fall below a critical threshold. The threshold associated with an M0 algorithm must be conservative enough to ensure that the re-mapping process completes before energy resources are completely exhausted. The M1, M2 and M3 algorithms cause application re-mapping to occur when a threshold in number of faults that have occurred in a node, link collisions and link carrier-sense errors respectively, have exceeded their associated, specified threshold.

3.3 D-Class algorithms

The re-mapping destination algorithm strives to determine a node that an application should be re-mapped to. Of the particular example D-class policies listed in Table 1, D1, D2 and D4, are well suited to situations in which links in a system fail and it is desirable that applications adapt around these failures. The D3 algorithm is relevant to all systems with limited battery resources, while the D0 algorithm may be enabled in systems in which it is either impossible or prohibitively expensive to determine the best neighbor to re-map to.

Online versus offline DFTM

The first step in this investigation of DFTM is to employ an offline approach, in which the best set of algorithms for the likely prevalent conditions are determined for a system at design time. An online approach in which the algorithms to be activated are themselves determined by some higher-level *meta-algorithms*, is a challenging area of future research. Before discussing the experimental evaluation of a system with a subset of the above algorithms implemented, new measures, which enable a combined evaluation of performance, power consumption, reliability and the effect of the application power consumption profile on battery life, are introduced in the following section.

4. METRICS

In gracefully degrading fault-tolerant systems [3], failure of a subset of the system resources leads to a degradation in performance, but it is not catastrophic in nature. The redundantly deployed devices in such a system can be viewed as providing additional performance in the absence of failures, and enabling continued operation in the presence of failures. In such systems therefore, it is necessary to employ metrics that take into account both system performance and reliability [1]. For systems in which performance, power (average and peak power, and overall energy consumption) and battery lifetime[3] may be traded off for reliability, similar measures are required to ascertain the effectiveness of system architectures, programming models, and the like. This section introduces such metrics.

In order to *derive* the necessary metrics using Markovian analysis, the following derivations assume that failures in the system under study are exponentially distributed, i.e., the probability of failure of a component is independent of its past histories of failures. The assumption enables the use of Markovian analysis to derive expressions for the failure probabilities over time. For situations where such an assumption will not hold, alternative means of *deriving* the expressions must be employed—the derived measures, their applicability, and meaning however, will not change.

4.1 Combined Energy, Battery, Performance and Reliability *(Ebformability)* Metrics

In the following, F is the set of failure states, a subset of the states in which a system may exist, which are indexed with the variable i. The initial system state is always denoted by I. The probability of being in a state i after n times steps is denoted $P_i(n)$. The behavior of a system (e.g., a particular computational surface) under study, can be characterized as consisting of a collection of distinct states in a Markov model, each state corresponding to a given level of performance. For example, in a gracefully degrading system with N nodes, 3 or more of which must be functioning in order for the system to be considered "alive", the system may be modeled as a set of $N+1$ states, $\{0..N\}$, of which three states, 0, 1 and 2 constitute the set of failing states, $F = \{0, 1, 2\}$.

Based on data obtained from observing the system, or from simulation, the transition probabilities between states of the system may be used to obtain the transition probabilities between states after n steps (for a discrete time Markov chain) or over time (for a continuous time Markov chain). For a discrete time Markov chain, the steady-state probabilities are given (by the *Chapman-Kolmogorov* equation [15]) as:

$$P^{(n)} = P^{(l)} P^{(n-l)}, \qquad \text{for} \quad 0 < l < n. \tag{1}$$

where P is the matrix of the one-step transition probabilities, and $P^{(n)}$ is n-th product of P with itself. The probability of being in a given state after n steps, can be solved for using either direct or iterative methods such as the *power method* [15], using the initial conditions for state probabilities, $P_I(0) = 1$ and $P_i(0) = 0$, for some initial state $I \neq i$. The variation of probability of being in a given state as the system evolves, can now be used to derive further measures that incorporate system reliability.

In extending traditional reliability measures[4] to include performance in gracefully degrading systems, prior work [1] performed a transformation from the time domain to the computation domain, to obtain a *computation availability*, T, as shown below. To include the effect of power consumption, as proposed herein, a further transformation to the time-power domain is necessary, to obtain the *computation availability per Watt*, Tpw:

$$T = \alpha \cdot n \tag{2}$$

$$Tpw = \alpha_{pw} \cdot n = \frac{\alpha}{\text{Avg. Power Consumption}} \cdot n \tag{3}$$

where α is the *computation capacity*—the amount of useful computation per unit of time performed in a given state and n is the number of time steps. Similarly, α_{pw} is the amount of useful computation per Watt of power dissipated in a given state. The quotient of performance and power is employed in α_{pw} rather than the product (as in the case of, say, the energy-delay product), because for congruence with α, it is desired for larger values of α_{pw} to be better.

4.2 The Capacity Function

$C_i(T)$, the capacity function [1], is the probability that the system executes a task of length T before its first failure, given that the state at the start of computation was i:

$$C_i(T) = \sum_{j \notin F} P_j^*(T) \tag{4}$$

$P_j^*(T)$ (or $P_j(n)$ expressed in terms of $T = \alpha \cdot n$) is the probability of being in a given state after $T = \alpha \cdot n$ amount of computation. Larger values of $C_i(T)$ are desirable for a given T. A new metric, which extends the capacity function to include considerations of power consumption, the capacity function per Watt, $Cpw_i(Tpw)$ is given as:

$$Cpw_i(Tpw) = \sum_{j \notin F} P_j^*(Tpw), \tag{5}$$

where

$$P_j^*(Tpw) = P_j(n) \quad \text{with} \quad Tpw = \alpha_{pw} \cdot n$$

$C_i(T)$ and $Cpw_i(Tpw)$, however, do not take into account the limitation on lifetime imposed by an energy source. For example, $C_i(T)$ for a given system might be non-zero for some amount, T, of computation performed, even though for that system, the energy resources cannot sustain the system for T amount of computation. This discrepancy is the result of the model as described thus far, only taking into consideration failures (as modeled by the system states and transition probabilities) and the system performance (as modeled by the computation capacities α and α_{pw}), but having no model of the restrictions imposed by limited energy resources such as batteries. As presented thus far, it is therefore applicable to systems that are not battery-powered.

In a battery powered system, the variation of $C_i(T)$ will be affected by the battery state of charge profile, and will be bounded by the battery life. The *battery-aware capacity function*, $Cbatt_i(T)$, is defined as:

$$Cbatt_i(T) = \sum_{j \notin F} P_j^*(T) \cdot \zeta_{batt}(T) \tag{6}$$

where $\zeta_{batt}(T)$ is the normalized variation of the state of charge of the battery with the amount of computation. It is

[3] Battery lifetime is a non-linear function of the variation in power consumption profile over time.

[4] A traditional reliability metric, the *availability* [1], is given by the limit of the sum of the steady state probabilities of being in a non-fail state. The *Mean Time To Failure (MTTF)*, is equivalent to the availability with absorbing failure states.

obtained by transforming the variation of the battery state of charge versus time curve, (which can be derived from the datasheet for a particular battery cell), into the computation domain. In addition to the battery-aware capacity function, it might be important in a battery-powered system to also consider a *battery-aware capacity per Watt*, $Cbattpw(Tpw)$:

$$Cbattpw_i(Tpw) = \sum_{j \notin F} P_j^*(Tpw) \cdot \zeta_{batt}(Tpw) \quad (7)$$

$C_i(T)$, $Cbatt_i(T)$, $Cpw_i(Tpw)$ and $Cbattpw_i(Tpw)$ are used to calculate the *Mean Computation Before Failure (MCBF)* [1], and new related measures which incorporate both power consumption and the effects of the power consumption profile on battery lifetime—*Mean Computation Before Battery Failure (MCBBF)*, *Mean Computation per Watt Before Failure (MCPWBF)* and *Mean Computation per Watt Before Battery Failure (MCPWBBF)* respectively:

$$MCBF = \sum_{0}^{\infty} C_I(T), \quad (8)$$

$$MCBBF = \sum_{0}^{\infty} Cbatt_I(T) \quad (9)$$

$$MCPWBF = \sum_{0}^{\infty} Cpw_I(Tpw), \quad (10)$$

$$MCPWBBF = \sum_{0}^{\infty} Cbattpw_I(Tpw) \quad (11)$$

where I is the initial state, at $n = 0$ (and $T = 0$ or $Tpw = 0$).

4.3 Computation Reliability

The *computation reliability* [1], $R^*(n, T)$ is the probability the system executes a task of length T, given that the system state is i at time-step n. It is extended here to the *computation reliability per Watt* is $Rpw^*(n, Tpw)$:

$$R^*(n, T) = \sum_{i \notin F} C_i(T) P_i(n) \quad (12)$$

$$Rpw^*(n, Tpw) = \sum_{i \notin F} Cpw_i(Tpw) P_i(n) \quad (13)$$

Similar definitions can be given for *computation reliability before battery failure* and *computation reliability per Watt before battery failure*. They are omitted here for brevity, since they will not be employed in subsequent evaluations in Section 5. It is desirable for a system to have both its $R^*(n, T)$ and $Rpw^*(n, Tpw)$ decline slowly with increasing n, T and Tpw. In other words, a higher expected reliability with increasing task size (amount of computation) is desirable. Similarly, for a fixed amount of computation, a larger expected reliability with increasing task completion time (for example, due to slower computation) is desirable.

The above measures can be used to determine the efficacy of a system in providing fault-tolerant operation with the best combination of power consumption, performance, reliability and longest battery lifetime. Such an evaluation would proceed by first defining the system states, and determining the transition probabilities between states. The probability of being in a given state, i, after a number of time steps, $P_i(n)$, can then be obtained by any of the methods described in the literature. The derivation of $P_i(n)$ in this section assumed a system that obeys the Markovian property. Regardless of the method used to determine $P_i(n)$, it can then be transformed to $P_i^*(T)$, by expressing the probability of being in a given state, in terms of amount of computation performed, rather than time steps. This requires an appropriate definition for the particular system under study, of the computation availability, α. The metrics taking into account performance, reliability, battery life and power consumption can then be determined as detailed in equations 5–13 above.

5. EXPERIMENTAL EVALUATION

DFTM enables individual nodes to adapt to faults, *using* redundant computing resources, by re-mapping executing applications from failing systems to redundantly deployed ones. In this work, the re-mapping is achieved by *lightweight code migration*, although other techniques such as *remote execution* may be substituted—the ideas of DFTM are not tied to any particular re-mapping approach.

In what we term *lightweight code migration* [12, 13], in contrast to traditional process migration [6], applications are implemented in a manner in which they can be *asynchronously restarted* while maintaining *persistence* for important state. By placing information that must be persistent across restarts in the initialized and uninitialized data segments of the application, it is possible to maintain state across migration while only transferring the program code, initialized data and uninitialized data segments.

5.1 Driver Application

The driver application used in the subsequent analysis of the efficacy of DFTM is *beamforming* [16]. The goal in beamforming is to detect the location of a signal source, and "focus" a set of sensors (e.g., microphones) on this source. In a traditional implementation, sampled signals from spatially distributed sensors are sent to a central processor, which processes them to determine the location of the signal source and reconstruct a desired signal. Each sample is processed (*filtered*), and this processing is largely independent of the processing of other samples.

In a system with multiple spatially distributed processing devices, the beamforming application may be partitioned such that each filter operation for a given sensor is mapped to one processing device (henceforth referred to as a *slave node*), and the filtered samples may be communicated to one device (henceforth, *master node*) to perform a final collation step. Figure 1 illustrates the logical and a possible physical organization for an integrated computational, sensing and actuation surface [14] used to perform beamforming.

5.2 Platform

The framework used in the evaluation of DFTM is a cycle accurate simulator of computation, communication, power consumption, battery discharge characteristics and node/link failures [10]. The modeling of the battery and DC-DC converter subsystem employ a discrete-time battery model based on [2]. The simulation of instruction execution and associated power consumption is based on [11].

For example, to model the topology shown in Figure 1, 25 processing nodes and 12 communication links are instantiated in the simulator. The operating frequencies and voltages of the processing nodes may be specified, affecting both the performance (time scale) and power consumption during simulation. The instantiated communication links are configured with specific link speeds, link maximum frame sizes, transmit

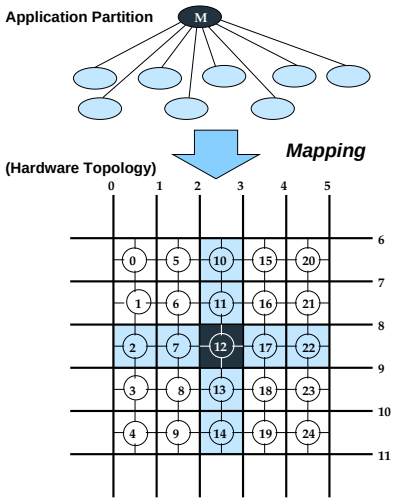

Figure 1: Nodes in the *logical organization* (top) represent units of concurrency in the application. In the *hardware topology* (bottom), heavy lines represent shared communication links and circles represent computation nodes. Connections to the communication links are shown with light lines.

Table 2: Relevant parameters employed in experimental evaluation.

Attribute	Value
Operating Frequency Idle Mode	60 MHz @ 3.3V 15MHz @ 0.85V
Battery Capacity Battery Parameters DC-DC Conv. Efficiency	0.5 mAh Panasonic CGR18 family Maxim MAX1653
Communication Power Link Speed Link Maximum Frame Size	250mW (RX/TX) 200 Kb/s 1024 bits

and receive power consumption and other parameters. Each instantiated processor, e.g., node *0* in the lower half of Figure 1, is configured to have 4 network interfaces, and these interfaces are attached to specific network communication links, links *6*, *1*, *7* and *0* (lower half of Figure 1) in the case of node *0*. Both the nodes and the links may be configured with failure probabilities for intermittent failure, as well as maximum failure durations. In the case of Figure 1, the links are used as multiple access communication links, however, they may also be used as direct (point-to-point) links between nodes. The simulation of processing clock cycles is synchronized with that of the data transmission, thus the modeling of communication and computation are cycle accurate with respect to each other. A few simulation parameters of relevance are listed in Table 2. When not actively performing computation, nodes in the system place themselves into an *idle mode*, to conserve battery resources.

The experiments conducted can be categorized into three groups. The first group (Exp. 0–Exp. 2 in Table 3) of experiments serves as a baseline, and illustrates the performance of the system in the absence of DFTM. The second group (Exp. 3–Exp. 6 in Table 3) investigates the efficacy of the different DFTM D-class algorithms (i.e., migration destination decisions), for a system with a localized failing link[5] and migration

[5]Without loss of generality, an arbitrary choice was made to induce failures in link #9.

Table 3: Experiments used to investigate the efficacy of a subset of the proposed DFTM algorithms. The topology on which the beamforming application executes is that from Figure 1.

Exp.	Config.	Description
0	No DFTM	No intermittent faults, only low batt. Migrate to a pre-assigned redundant node, when battery levels run low.
1	No DFTM	Intermittent faults in links, rate 1E-8. Migrate to a pre-assigned redundant node, when battery levels run low.
2	No DFTM	Intermittent faults in link #9, rate 1E-6. Migrate to a pre-assigned redundant node, when battery levels run low.
3	(M0, D0)	Intermittent faults in link #9, rate 1E-6. DFTM algorithm: re-map on low battery to a random redundant node.
4	(M0, D1)	Intermittent faults in link #9, rate 1E-6. DFTM algorithm: re-map on low battery to redundant node with fewest collisions.
5	(M0, D3)	Intermittent faults in link #9, rate 1E-6. DFTM algorithm: re-map on low battery to redundant node with most energy.
6	(M0, D4)	Intermittent faults in link #9, rate 1E-6. DFTM algorithm: re-map on low battery to redundant node with most links to master node.
7	(M2, D1)	Intermittent faults in link #9, rate 1E-6. DFTM algorithm: re-map on too many collisions, to the redundant neighbor with fewest number of collisions.
8	(M3, D2)	Intermittent faults in link #9, rate 1E-6. DFTM algorithm: re-map on too many carrier sense errors, to the redundant neighbor with fewest number of carrier-sense errors.

initiated only on low battery levels (i.e., M0 from the M-class of algorithms). The last grouping (Exp. 7–Exp. 8 in Table 3) investigates the performance of the system with the logical grouping of DFTM M-class and D-class algorithms, that relate to failing links. The instantiations of the L-class algorithms are not evaluated for this application, and are a topic of our current investigations.

The failure rates specified in Table 3 are the failure probabilities per simulation time step of 16ns (corresponding to the duration of one CPU clock cycle at 60MHz). In choosing failure rates to provide appreciably adverse conditions for DFTM, a failure rate of 1E-6 for the configured faulty link was employed in all experiments with DFTM. In previous investigations of the application and simulated hardware, it was observed that a failure rate of 1E-7 was the breakpoint at which the system was not able to hide the additional cost imposed by the failures, under the available slack. Failure rates in the range 1E-6 to 1E-8 per 16ns are similar to those of first generation networking and computer hardware [5], and seem reasonable choices therefore, for emerging hardware technologies in failure-prone environments.

5.3 Effect of DFTM Algorithms on Performance

The CPU occupancy is indicative of the possibility of mapping multiple applications to a single processing element, given the requisite system software support. The product of the *idleness* (1 / (CPU occupancy)), and the application

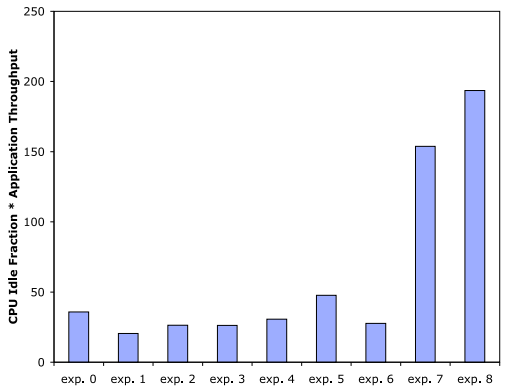

Figure 2: Variation of α (product of CPU idle-time fraction and application average sample throughput) across experiments.

throughput (average number of samples per round in beamforming application) is therefore used as the computation capacity, α, defined in Section 4. Larger values of α indicate better combined performance and efficiency in using computation resources.

Figure 2 shows the variation in α across experiments. The DFTM algorithms that exhibit the best computation capacity, α, are the (M0, D3), (M2, D1) and (M3, D2) algorithms, in order of increasing performance. From Table 3, these three algorithms aim to maximize available energy resources and minimize communication errors. Thus, the result is to be expected—in the presence of limited energy resources and faulty communication links, they provide the most efficient solutions.

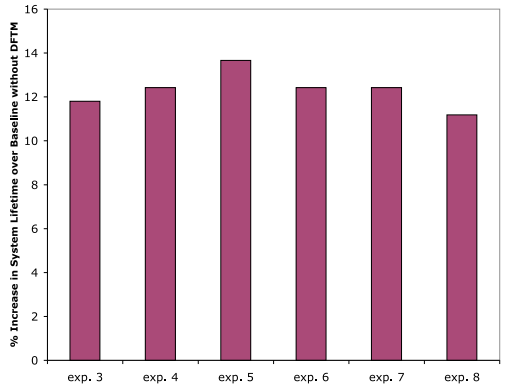

Figure 3: Increase in system lifetime over baseline without DFTM.

The percentage increase in system lifetime, over the baseline without DFTM, is shown in Figure 3. The algorithm grouping which witnesses the longest lifetime is Exp. 5, the (M0, D3) algorithm tuple (which aims to maximize available energy resources), with a 13.7% improvement over the baseline. Comparing the lifetime trends to those for computation capacity in Figure 2, it is apparent that the system configuration with the longest lifetime is not the most computationally efficient. These results however, neither provide a measure of which set of algorithms provides better reliability in the limit, nor do they provide a measure of the combined reliability, power consumption and battery life.

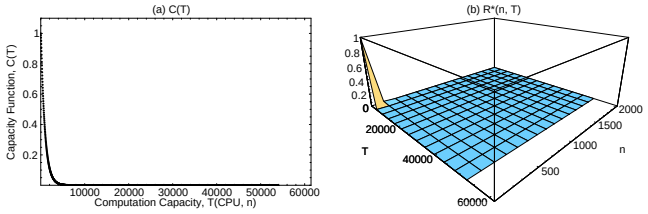

Figure 4: Variation of (a) capacity $C_i(T)$ and (b) reliability $R^*(n,T)$ for baseline system without DFTM (Exp. 2).

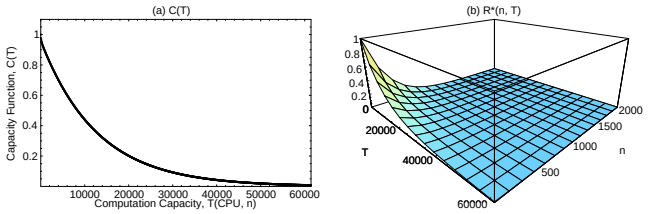

Figure 5: Variation of (a) capacity $C_i(T)$ and (b) reliability $R^*(n,T)$ for DFTM algorithm (M0, D3) which aims to maximize battery life.

5.4 Reliability and Mean Computation

The previous set of results evaluated the performance of the various DFTM algorithm tuples in terms of traditional measures of performance. The algorithm tuple leading to the longest system lifetime (M0, D3) may not indeed provide the best performance (from Figure 2, this was attained by Exp. 8, the (M3, D2) algorithm). However, neither of these pieces of information provide any insight into which system is more reliable during its lifetime, and which system has the best combination of performance, battery lifetime and reliability. The measures derived in Section 4 however make it possible to reach such conclusions with a combination of constraints.

Figure 4(a) shows the variation of the capacity function, the probability that the system executes a task of a given length, starting from its initial conditions, for the baseline system without DFTM. The probability of the system executing a task of a given length approaches zero as the task length (T), approaches 4000 units. The reliability of the baseline system, the probability that the system executes a task of a specified length T, given that the system is in a non-failed state at time step n, is shown in Figure 4(b). The front-left face of the cube is equivalent to the plot of the capacity function, since the system starts of, in that case, with $n = 0$. The rear-left face of the cube likewise gives the amount of computation than can be performed with increasing time steps, for a task of minimal length.

Equivalent trends for the algorithm setting which achieves the longest lifetime (Exp. 5, See Figure 3) are shown in Figure 5. Compared to the baseline system without DFTM, this configuration exhibits a much slower decline in the probability of the system being in a non-failure state—thus, it provides a better reliability of executing a task of a given length for increasing task lengths, and for increasing durations of time in which to do so. The system maintains a non-zero probability of being in a non-failed state past $T = 50,000$. The best reliability is however provided by the algorithm groupings that explicitly address the issue of failures (as opposed to lifetime). The capacity function and reliability plots for the (M3, D2) algorithm grouping is shown in Figure 6.

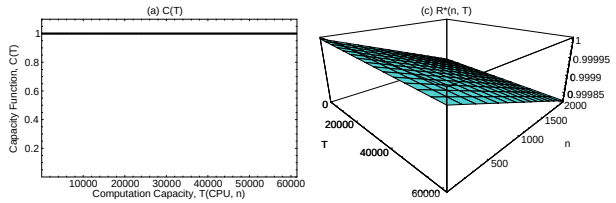

Figure 6: Variation of (a) capacity $C_i(T)$ and (b) reliability $R^*(n,T)$ for DFTM algorithm (M3, D2).

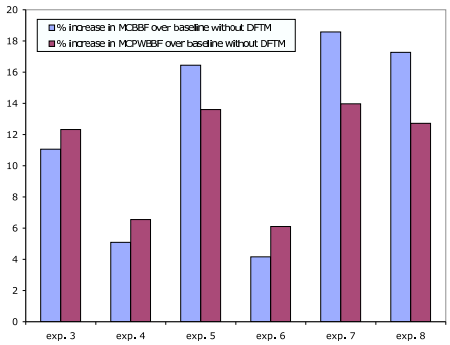

Figure 7: Percentage increase in MCBBF and MCPWBBF for experiments with DFTM. System lifetime is limited by a single re-mapping step.

The above results might suggest that, for a given goal in computation to be performed in the presence of failures, the configuration (M3, D2) has a clear advantage, since it provides both the best performance (largest α, Figure 2) and the greatest reliability (Figure 6). The above measures however, do not include the effects of the associated power dissipation, and its effects on battery lifetime.

The mean computation before battery failure (MCBBF), represents the limiting amount of computation that can be obtained from a particular battery (i.e., for a given batteries variation in state of charge with discharge, captured by ζ defined in Section 4). The percentage increase in MCBBF for the different DFTM algorithm groupings are shown in Figure 7.

Despite having the best performance and reliability, the (M3, D2) algorithm grouping provides neither the greatest MCBBF nor the best MCPWBBF (the MCBBF per Watt of power dissipated on average). This is due to its associated energy costs, which lead in increased power consumption and a reduced battery lifetime, offsetting the benefits of increased reliability. Furthermore, the trends in the figure indicate that the algorithm groupings which provide the best combination of metrics, do so at a cost in average power consumption—the increase in MCPWBBF (MCBBF per Watt) over the baseline in the three best cases is consistently worse than the increase in the MCBBF, but not so for the other algorithm groupings.

6. SUMMARY AND CONCLUSION

This paper introduced the concept of *dynamic fault-tolerance management (DFTM)*, and classifications for DFTM algorithms. New metrics for characterizing *ebformability*, a combination of system *e*nergy-efficiency, *b*attery lifetime, per*forma*nce and reli*ability* were presented. The proposed metrics can be used to assess the quality of various design methodologies and tools for emerging platforms characterized by joint energy, reliability and performance constraints.

Using the proposed techniques, it was shown that techniques providing best performance do not necessarily provide the best combined performance, reliability, power consumption and battery life. For battery powered devices, inclusion of battery discharge characteristics into the model enables better judgment as to the potential computation that may be performed by a system in the presence of runtime failures, before it reaches an absorbing failure state.

Acknowledgments

This research was supported in part by DARPA Information Processing Technology Office under contract F33615-02-1-4004 and Semiconductor Research Corporation under grant 2002-RJ-1052G.

7. REFERENCES

[1] M. D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, c-27(6):540–547, June 1978.

[2] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi. A discrete-time battery model for high-level power estimation. In *Proceedings of the conference on Design, automation and test in Europe, DATE'00*, pages 35–39, January 2000.

[3] B. R. Borgerson and R. F. Freitas. A reliability model for gracefully degrading and standby-sparing systems. *IEEE Transactions on Computers*, c-24:517–525, May 1975.

[4] R. J. Cole, B. M. Maggs, and R. K. Sitaraman. Reconfiguring arrays with faults part I: worst-case faults. *SIAM Journal on Computing*, 26(6):1581–1611, December 1997.

[5] F. E. Heart, S. M. Ornstein, W. R. Crowther, and W. B. Barker. A New Minicomputer/Multiprocessor for the ARPA Network. In *Proceedings of the 1973 NCC, AFIPS Conference Proceedings*, pages 529–537, 1973.

[6] D. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process Migration. *ACM Computing Surveys*, 32(3):241–299, September 2000.

[7] M. Perillo and W. Heinzelman. Optimal Sensor Management Under Energy and Reliability Constraints. In *Proc. of the IEEE Wireless Communications and Networking Conference*, March 2003.

[8] D. Rakhmatov, S. Vrudhula, and D. A. Wallach. Battery Lifetime Prediction for Energy-Aware Computing. In *International Symposium on Low Power Electronics and Design, ISLPED'02*, pages 154–159, August 2002.

[9] T. Simunic, L. Benini, P. W. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *Mobile Computing and Networking*, pages 11–19, 2000.

[10] P. Stanley-Marbell. Myrmigki Simulator Reference Manual. Technical report, CSSI, Dept. of ECE, Carnegie Mellon, 2003.

[11] P. Stanley-Marbell and M. Hsiao. Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 141–146, August 2001.

[12] P. Stanley-Marbell and D. Marculescu. Exploiting Redundancy through Code Migration in Networked Embedded Systems. Technical Report 02-14, CSSI, Carnegie Mellon, April 2002.

[13] P. Stanley-Marbell, D. Marculescu, R. Marculescu, and P. K. Khosla. Modeling, Analysis, and Self-Management of Electronic Textiles. *IEEE Trans. on Computers*, 52(8):996–1010, August 2003.

[14] P. Stanley-Marbell, D. Marculescu, R. Marculescu, and P. K. Khosla. Modeling Computational, Sensing and Actuation Surfaces. In C. Piguet, editor, *Low-Power Electronics Design*. CRC Press, 2003.

[15] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[16] B. D. Van Veen and K. M. Buckley. Beamforming: a versatile approach to spatial filtering. *IEEE ASSP Magazine*, 5(2):4–24, April 1988.

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