Matching Software Fault Tolerance and Application Needs

Eltefaat Shokri and Herbert Hecht
SoHaR Incorporated
Beverly Hills, CA, USA

Abstract

The designation "fault tolerant software" has been used for techniques ranging from roll-back and retry to N-version programming, from data mirroring to functional redundancy. If the term is to be meaningful, qualifying definitions are required. This paper attempts to provide these by analyzing the capabilities of representative software fault tolerance techniques described in prior literature and matching these with the needs of representative environments in which fault tolerance may be applied.

This paper suggests five categories for comparison of application needs and fault-tolerance capabilities: accuracy, deadline, state preservation, coverage, and economy of resources. It then shows how representative needs and capabilities can be characterized in identical terms by these categories. Algorithms are developed for either ranking (ordering) the importance of categories or assigning weighting factors to them. The algorithms suggest partially-suitable matches where there is no complete match between the application needs and the capabilities of fault-tolerance techniques. Examples of the selection technique are presented.

1. Introduction

This paper is intended for system engineers and software designers who want to specify or implement fault tolerance features and are puzzled by the different interpretations of the term "fault tolerant", ranging from check-pointing to N-version programming. The specific aim addressed here is to find classifiers that are relevant to the *needs* of the applications as well as to the *capabilities* of the software fault tolerance techniques.

The body of the paper first reviews prior work in taxonomies for system and software fault tolerance and then shows how concepts derived from the taxonomies can guide the selection of fault tolerance features in the design of a new application. The methodology emphasizes that a range of software features and architectures may be suitable with the final selection based on the developer's experience or customer preferences.

2. Prior work

At the 1975 International Conference on Reliable Software three papers were devoted to the subject. These covered, respectively, self-checking software (with implied re-execution) [1], the recovery block (RB) [2], and a general survey with emphasis on N-version programming (NVP) [3]. Taxonomies of fault tolerant systems with special reference to fault tolerant software, appeared later. Avizienis [4] laid down a framework for taxonomy of fault-tolerance.

Johnson [5] introduced a more concrete architectural taxonomy of fault-tolerant (FT) systems from the perspective of redundancy techniques employed: "time redundancy", "hardware redundancy", "information redundancy", and "software redundancy".

In [6], a new form of redundancy, called data redundancy (i.e., providing different expressions of the data and/or system state in different executions), was introduced. However, this type of redundancy can be viewed as a subtype of the information redundancy introduced in [5]. Hecht [7] added functional (analytical) redundancy in which a given application requirement can be implemented by diverse techniques, such as locating a vehicle by inertial navigation or by a radio location system.

The above taxonomies addressed the architecture of the fault tolerant configurations. In addition, a number of authors focused on the capabilities which either were provided by defined architectures or were needed by the user. In an early work of this type [8], the authors categorized the application domains based on their requirements for fault tolerance at the system level. Most of these requirements that affect software (listed below) are still significant today:
(i) Data Protection , (ii) Application Program Protection, (iii) Correctness of Results, (iv) Recovery time, (v) Cost Constraints, and (vi) Maintainability.

A more recent effort to classify the need for fault-tolerance is reported in [9], in which the applications were categorized as (i) general-purpose computing, (ii) high-availability systems, (iii) long-life systems, and (iv) critical computations. The significant software fault tolerance mechanisms that are described in categories ii - iv are checkpointing, watchdog timers, and "fast-fail" attributes.

In the following section we build on this prior work by defining requirements for software fault tolerance in a wide range of applications, and then classifying the capabilities of fault tolerance features in terms that are identical to those in which requirements are expressed. This coordinated classification facilitates identification of the features that most closely meet the user's needs.

3. Coordinated approach

To be useful for selecting the fault tolerance techniques the needs of the application and the capabilities of the techniques should be expressed in coordinated terms, called *attributes*. The five attributes introduced below represent a working set for demonstrating the selection methodology. Whether they are the best ones, or whether they need to be augmented may be explored in further research.

3.1. Framework

The framework characterizes both user needs and the capabilities of specific fault tolerance features by five attributes, A_x ($x = 1..5$) and allows 3 levels of need or capability for each: low (L), medium (M) and high (H) (with $H > M > L$). The level of need for a given application and attribute is designated $N(A_x)$, and the capability of a given fault tolerance technique in category x is designated $C(A_x)$. The five attributes are briefly defined in Table 1.

Table 1. Software Fault Tolerance Attributes

Index	Attribute	Definition
1	Accuracy	High accuracy at every iteration
2	Deadline	Immediate response at every iteration
3	State Preserv.	Presentation of all data under failure conditions
4	Coverage	Detection of and recovery from all anomalies
5	Economy	Minim. added resources for fault tolerance

The definition of the attributes applies to both needs of an application and the capabilities of a technique. Thus, $N(A_i) = H$ means that the application has very tight requirements for response time, and $C(A_i) = H$ means that a given fault tolerance technique provides very short response times and meets tight requirements. In general,

$$C(A_i) \geq N(A_i)$$

means that with respect to the attribute A_i the technique for which the left member of the relation has been evaluated meets the needs of the application for which the right member has been evaluated. Using **C** and **N** to denote vectors of $C(A_i)$ and $N(A_i)$, respectively, for all defined indices *i*, the relation

$$\mathbf{C} \geq \mathbf{N}$$

means that the technique for which **C** has been evaluated satisfies the needs of the application for every attribute.

It is sometimes beneficial to permute the indices so that the most important attribute for a given application is associated with index 1 and the least important one with index 5. In this ordering approach, C_j and N_j denote the capability and need levels of the j-th attribute (ranked in importance).

It is also possible to form weighted norms, C_w and N_w that are defined by

$$X_w = \sum w_i A_i$$

where X represents C or N and the w_i factors are weights assigned by the designer to denote the relative importance of the attributes; these factors must be the same in the evaluation of C_w and N_w. Weighting can be accomplished by converting the levels of the attributes to numerical terms.

A mathematical notation has been used for the sake of clarity, but it should be recognized that the matching of a technique to an application is not an exact science. It can involve subjective judgment and will typically be affected by the experience of the designer in the use of specific fault tolerance techniques and by the prevailing practice in the application domain.

A number of attributes of fault tolerant software that may be of interest to the user have been omitted from Table 1 for the reasons shown in Table 2.

Table 2: Fault Types Not Considered

Fault Type	Reason for Omission
Transient hardware	FT can be provided by all SWFT techniques; effectiveness depends on implementation
Solid hardware	FT depends on hardware configuration
Hardware design	Usually manifests itself in transient faults and is handled as such
Support software	No inherent distinction between the software fault tolerance techniques
Integration	Usually manifests itself in transient faults
Environment induced	Requires protection at the interfaces; no inherent distinction

3.2. Capability attributes

This section describes the assignment of the three-level capability values (L, M, and H) for each of the five attributes shown in Table 1.

A high value is assigned for *accuracy* (i.e., application-protection) if a FT architecture offers an explicit provision for protecting the system from possible faults and/or imperfections in the application, such as functional or static software redundancy (e.g., NVP). Architectures which protect the *system-state* by use of either replicated independent memories, such as IBM's Highly Available Systems (IBM/HAS) architecture [10], or by maintaining the system-state in different hardware units (such as functional-redundancy using another computer) are rated high for this attribute. *Coverage* expresses the certainty that all expected faults will be tolerated. Static redundancy (such as NVP) is rated high because of the broad scope of error detection and the avoidance of switching for recovery. The *deadline* capability of a FT architecture identifies the expected speed of recovery from a fault. Techniques in which replacement software is immediately available are rated high. The "*economy of resource*" attribute relates to the amount of hardware/software/data resources devoted for the architecture. Architectures employing hardware redundancy (such as NVP and DRB) are classified low/medium while less resource-demanding architectures such as Rollback/Retry are classified as high.

Table 3. Capabilities Classification

FT ARCH.		Ref.	Accu.	State	Cover.	Dead.	Eco.
Time Redundancy	Rollback/ Retry	[11]	M	M	L	M	H
	Cold Restart		M	L	L	L	H
Functional Redundancy	System-level		L	H	M	M	L
	Different computer	[12]	M	H	M	M	L
	Same computer		M	L	M	M	L
Application-level Software Redundancy	Static Redun.	[13]	H	H	H	H	L
	DRB	[14]	H/M	H	M	H	M
	RB	[2]	H/M	H	M	M	H
Data Redundancy	NCP	[6]	M	M	L	H	L
	Retry-Block	[6]	M	M	L	M	H
	IBM/HAS	[10]	M	H	M	H	M

3.3. Needs attributes

The assignment of values (L, M, or H) to the needs attributes follows similar reasoning, and examples of the classification are shown in Table 4. The *accuracy* attribute will generally be assigned a high value if catastrophic consequences may arise from a short-term deviation from the specified accuracy. This is particularly true for applications in which irreversible actions are taken, such as shutting down a nuclear reactor or launching a ballistic missile. The *deadline* attribute is particularly important in closed loop control systems because these can go unstable if corrective computer outputs are delayed.

State preservation is particularly important in transaction systems where the consequence of data loss can be very substantial. Financial transactions, such as electronic funds transfer, are particularly demanding in this area.

Demands for *coverage* depend on the alternatives available for timely detection of an anomaly and for recovering from the consequences of a failure. Where the software constitutes the last line of defense, such as in a nuclear plant shut-down system or in fly-by-wire aircraft controls, a very high degree of coverage is required.

The need for *economy of resources* arises not from the software proper but from the computer architecture that is required to run fault tolerant software. Physical resource constraints are most severe in spacecraft and in high performance military aircraft. Monetary resource constraints are highest in automotive applications.

3.4. Outline of selection procedures

There are two alternatives for selecting the most suitable fault-tolerance technique for an application:

- *Ordering*: In this approach, the attributes needed by an application are ordered based on their importance, so that N_1 is the most important need which cannot be sacrificed and N_5 is the least important need attribute. The objective here is to find a fault-tolerance technique with $C > N$. If there is no completely satisfactory fault tolerance technique for the application, the least important attribute can be relaxed in order to find the best suited alternatives.

- *Weighted*: In this approach, a weight is assigned to each need. Then a suitable architecture is the architecture with the weighted sum of capabilities equal to or greater that the weighted sum of needs of the application.

The following describes the ordering approach and one illustrative example.

Assume that $<N_1, N_2, N_3, N_4, N_5>$ is the set of ordered needs of the application for which a suitable fault-tolerance technique is sought. If there exist architectures with $<C_1, C_2, C_3, C_4, C_5>$ such that $C_j \geq N_j$ ($\forall$ j=1..5), then the set of suitable architectures (SSA) for the application includes all such architectures. However, if no architecture qualifies for the above requirements, we relax the less important need and check if this results in a matching architecture for the application. If not, then sacrificing the next less-important need will be considered.

As an example, consider selection of a fault tolerance technique for the *fly-by-wire aircraft control* application, the ordered needs of which are <Coverage, Deadline, Economy, Accuracy, State>. According to Table 4, the example application has the needs equal to $<N_1 = H, N_2 = H, N_3 = M , N4 = M , N5 = L>$. Table 3, shows no technique that can satisfy all needs of the example application. The relaxation starts with the least important attribute, in this case the "state" attribute that has the value L and thus cannot be relaxed further. The next candidate for relaxation is "accuracy" with the value M.

Table 4. Examples of Needs Classification

TYPE	EXAMPLE	Accuracy	State	Coverage	Deadline	Economy
Closed Loop Control	Fly-By-Wire Military Aircraft Control	M	L	H	H	M
	Autopilot, Transport	M	L	M	L	L
	Attitude, Spacecraft	M	M	M	M	H
	Nuclear Power, Feedwater Control	L	L	M	L	L
	Chemical Process Control	L	L	M	L/M	L
	Automobile, Cruise Control	L	L	L	L	H
Signaling, Switching, Shut-Down	Rail, Collision Avoidance	H	M	H	M	L
	Rail, Routing	M	H	M	L	L
	Elevator Control	L	M	L	L	L
	Nuclear Power Plant Shut-Down	H	H	H	L	L
	Missile, Range Safety	M	M	H	M	L
Transaction	Automatic Teller	L/H	H	L	L	H/L
	Electronic Funds Transfer	H	H	M	L	L
	Battle Management	H	H	H	M	L
	Telephone Switching	M	H	M	L	L
	Telephone Billing	L	M	L	L	L
Navigation	Aircraft	H	M	L	L	M
	Spacecraft	H	M/H	M	L	H
	Air Traffic Control–Aircraft Location	H	M	L	L	L
	Air Traffic Control – Warning Services	H	M	M	M	L

Lowering it to L will still not result in a qualifying technique. Thus, the relaxation of needs should continue to the "economy" attribute. When the "economy" attribute is lowered to L, the '*Static Redundancy*' technique is identified as a partially suitable technique. Analysis of the fault-tolerance techniques that have been employed in the past for this application shows that this is indeed a practicable selection. The extent to which requirements had to be relaxed also shows that much further research needs to be devoted to exploring better fault-tolerance techniques for this application. Where the need for fault-tolerance arises from regulatory requirements or is taken for granted by the decision-makers, the procedures discussed here may conclude the selection process.

4. Conclusions

The diversity of available software fault tolerance techniques presents at once an opportunity for meeting needs for fault tolerance in the most effective way and a source of bewilderment to the potential user in how to organize the selection process. The techniques described in this paper should reduce the bewilderment and improve the matching of needs and capabilities.

Five attributes have been identified that can guide the selection of fault tolerance techniques for critical application programs. The need for the attributes and the capabilities of the fault tolerance techniques are expressed in identical terms, and this permits a technically substantiated matching of needs to capabilities.

5. References

[1] Yau, S., et al, "Design of Self-Checking Software", *Proc. of Int'l Conf. on Reliable Software*, Los Angeles, CA, 1975.

[2] Randell, B., "System Structure for Software Fault Tolerance", *Proc. of Int'l Conf. on Reliable Software*, Los Angeles, CA, 1975.

[3] Avizienis, A., "Fault-Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Software", *Proc. of Conf. on Reliable Software*, Los Angeles, CA, 1975.

[4] Avizienis, A., "Framework for a Taxonomy of Fault-Tolerance Attributes in Computer Systems", *Proc. l0th Symposium on Computer Architecture*, Sweden, 1983.

[5] Johnson, B.W., "An Introduction to the Design and Analysis of Fault-Tolerant Systems", in *Fault-Tolerant Computer System Design*, Ed., Pradhan, D., 1996.

[6] Ammann P., and Knight J., "Data Diversity: An Approach to Software Fault-Tolerance", *IEEE Trans. Computers*, 1988.

[7] Hecht, H., "Reliability During Space Mission Concept Exploration", *in Space mission analysis and design*, Ed., Wertz, J., and Larson, W., 1991.

[8] Wensely, J., et al, "A Comparative Study of Architectures for Fault-Tolerance", *Proc. of Int'l Symp. of Fault-Tolerant Computing Systems*, Urbana, Ill., 1974.

[9] Siewiorek, D., "Architecture of Fault-Tolerant Computers", in *Fault-Tolerant System Design*, Ed., Pradhan, 1996.

[10] Cristian, F., "Fault-Tolerance in the Advanced Automation System", *Proc. of International Symposium on Fault Tolerant Computing*, Newcastle, UK, June 1990.

[11] Gray, J., "Why Do Computers Stop and What Can Be Done About It", *Proc. of 6th Symp. on Reliability in Distributed Software and Databases*, Los Angeles, CA, Jan., 1986.

[12] Bodson, M., "Analytic Redundancy for Software Fault-Tolerance in Hard Real-Time Systems", *CMU Tech. Rep.*, 1994.

[13] Avizienis, A., and Chen, L., "On the Implementation of N-Version Programming for Software Fault Tolerance During Execution", *Proc. of IEEE COMPSAC*, 1975.

[14] Kim, K. H., "Action-Level Fault Tolerance", in *Advances in Real-Time Systems'*, Ed. Sang Son, Prentice Hall, 1994.