Fault Tolerance for Distributed and Networked Systems

Wenbing Zhao
University of California, Santa Barbara, USA

Louise E. Moser
University of California, Santa Barbara, USA

P. Michael Melliar-Smith
University of California, Santa Barbara, USA

INTRODUCTION

The services provided by computers and communication networks are becoming more critical to our society. Such services increase the need for computers and their applications to operate reliably, even in the presence of faults. Fault tolerance is particularly important for distributed and networked systems (Mullender, 1993), including telecommunication, power distribution, transportation, manufacturing, and financial systems.

Fault-tolerant computing has been oriented towards custom designs and computer hardware (Siewiorek & Swarz, 1998), towards particular kinds of applications (Wensley et al., 1978), and towards operating systems (Borg, 1989). However, as computer and communication systems have become more complex and as hardware has become cheaper, designs have moved towards commercial off-the-shelf hardware and towards fault tolerance middleware located as a software layer between the application and the operating system.

Traditional proprietary designs are now being challenged by industry standards, such as the Fault Tolerant CORBA standard (Object Management Group, 2000) for distributed object applications based on the Common Object Request Broker Architecture (CORBA), and also the Hardware Platform Interface and Application Interface Specification (Service Availability Forum, 2003) for telecommunication and other embedded systems.

Of particular importance in the development of fault-tolerant computing is the distinction between application-aware and application-transparent fault tolerance. In application-aware fault tolerance, the application is aware of, and explicitly exploits, the mechanisms provided by the fault tolerance infrastructure, using application program interfaces (APIs). The application programmer writes code corresponding to the APIs to perform specific operations, such as to checkpoint an application process and to restore the process from the checkpoint, or to send a message across the network to a process on another processor and to receive the message on that other processor.

In application-transparent fault tolerance, the application is rendered fault tolerant, without involvement by, or modification to, the application program (Bressoud, 1998; Moser, Melliar-Smith, & Narasimhan, 1998). A fault tolerance middleware library is interposed ahead of the operating system libraries, between the application and the operating system (Narasimhan, Moser, & Melliar-Smith, 2002). When the application invokes standard operating system functions, those invocations are diverted to the fault tolerance middleware library, which modifies the operating system functions to provide additional fault tolerance functionality (Zhao, Moser, & Melliar-Smith, 2004). The application programmer does not need to implement or invoke additional methods for fault tolerance or to include additional fault tolerance code in the application program.

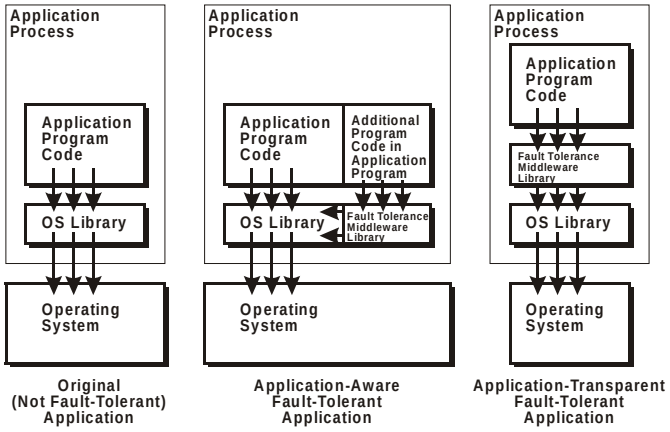
BACKGROUND

The basic terminology of fault-tolerant computing can be found in Laprie (1992). The terms failure, error, and fault were defined originally by Melliar-Smith & Randell (1977) and have become part of the ISO standard.

- A *failure* is the event of a system's generating a result that does not satisfy the system specification, or of the system's not generating a result that is required by the system specification. A failure is defined by the system specification, without reference to any components internal to the system, or to any enclosing system of which the system is a component.
- An *error* is incorrect information, or lack of information, within a system that will, unless detected and corrected, lead to a failure of the system.

Copyright © 2005, Idea Group Inc., distributing in print or electronic forms without written permission of IGI is prohibited.

Figure 1. The original application is shown at the left. In application-aware fault-tolerance (shown in the middle), the application program includes additional code that invokes the fault tolerance middleware library. In application-transparent fault-tolerance (shown at the right), the fault tolerance middleware library is interposed between the application and the operating system library.



- A *fault* is the original cause of an error, whether hardware or software. Sometimes the cause of the error is strictly objective but sometimes, particularly for software, it is a matter of subjective opinion.

A failure of a system might be a fault within a larger enclosing system. Similarly, a fault might be the failure of one of the components from which the system is constructed.

Faults (Cristian, 1991) are further classified as follows:

- A *crash fault* occurs when a component operates correctly up to some point in time, after which it produces no further results.
- A *timing fault* occurs when a component produces results at the wrong time, either too early or too late.
- An *omission fault* occurs when a component produces some results but not others.
- A *commission fault* occurs when a process or processor generates incorrect results. A *Byzantine* or *malicious fault* is a form of commission fault in which a process or processor generates incorrect results that are intentionally designed to mislead the algorithms or components of the system.

The metrics used in fault-tolerant computing include:

- *Reliability* is a measure of the uptime of a system in the absence of failure, and is given by the Mean Time Between Failure (MTBF).
- *Repairability* is a measure of how quickly a failed component or system can be restored to service, and is given by the Mean Time To Repair (MTTR).
- *Availability* is a measure of the uptime of a system, and is related to MTBF and MTTR by the formula: Availability = MTBF/(MTBF + MTTR). High availability typically means five nines (99.999%) or better, which corresponds to 5.25 minutes of planned and unplanned downtime per year.

TECHNOLOGY OF FAULT TOLERANCE

Replication

The basic strategy used in fault-tolerant systems to protect an application against faults is *replication*, so that if one replica becomes faulty, another replica is available to provide the service. The unit of replication can be an entire processor, a process, a Java container, a CORBA object or some other component. In this article, we refer to such a unit as a component. The replicas of a component constitute a group of two or more components. To provide fault isolation, the replicas of a component must be independent of each other, with no shared memory or data and with communication controlled by the fault tolerance middleware, so that the failure of one replica does not disable another replica. Several kinds of replication are possible (Powell, 1991).

Passive replication, shown in *Figure 2*, distinguishes one of the server replicas as the primary server replica and the other as the backup server replica. The primary executes methods invoked on the group, and the backup does not execute those methods. In cold passive replica-

5 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/chapter/fault-tolerance-distributed-networked-systems/14409

Related Content

Systems Requirements and Prototyping
Vincent C. Yen (1997). *Cases on Information Technology Management In Modern Organizations (pp. 107-120).*
www.irma-international.org/chapter/systems-requirements-prototyping/33463

Setting Up to Fail: The Case of Midwest MBA
Andrew Urbaczewskiand Jo Ellen Moore (1999). *Success and Pitfalls of Information Technology Management (pp. 143-148).*
www.irma-international.org/chapter/setting-fail-case-midwest-mba/33487

The Expert's Opinion
Beth Green (1995). *Information Resources Management Journal (pp. 37-38).*
www.irma-international.org/article/expert-opinion/51017

Developing Efficient Processes and Process Management in New Business Creation in the ICT-Sector
Arla Juntunen (2010). *Information Resources Management: Concepts, Methodologies, Tools and Applications (pp. 346-364).*
www.irma-international.org/chapter/developing-efficient-processes-process-management/54488

Using Quick Response Codes with Videos in the Laboratory
Marina Duarte, Andresa Baptistaand Gustavo Pinto (2016). *Journal of Cases on Information Technology (pp. 70-80).*
www.irma-international.org/article/using-quick-response-codes-with-videos-in-the-laboratory/173725