

Fault Management Based on Quality of Service Criteria

Douglas Wells
The Open Group
August, 2001

This paper summarizes some thoughts on managing faults within large, complex, distributed object systems and describes initial results of an implementation experiment targeting real-time systems. The proposed method incorporates fault management within a QoS-based resource management architecture that allows trade-offs among multiple QoS dimensions, including timeliness, integrity and reliability. Initial results of this process include a fast failure detector, which can reliably detect host node failures in real-time systems with sub-second time constraints.

Introduction

In recent years the DARPA Quorum program has sponsored research in managing the end-to-end quality-of-service (QoS) characteristics of mission-critical applications. This effort has resulted in the development of technology that permits configuration decisions to be delayed until execution time. This late binding allows execution-time factors to be incorporated in the decision process, resulting in more informed assignment of resources and overall more effective defense systems.

The focus so far has been on guaranteeing essential services, often utilizing redundant components when possible, or preempting resources from other functions when necessary. The next step is to extend these capabilities to large, complex systems, where not all capabilities can be guaranteed, but where the objective is to optimize the overall system—to utilize the resources available to produce the most valuable results. This is a goal common to both commercial systems and military systems.

A characteristic of large, complex, distributed systems is that one can not expect all of it to be up simultaneously. At almost any point in time, some portions will be down, due to planned maintenance or component failures. It is highly unlikely that these outages will occur in such a manner as to maintain system optimality. Traditionally, the issue of fault recovery has been addressed during the system design phase: system engineers would select fault-tolerant components for particularly critical subsystems, build in fall-back strategies for less important functions, and allow other functions to simply fail. The result is that run-time configuration choices are limited and certain resources may be reserved and therefore unavailable for more valuable activities.

Our goal for resource management should be to select the overall best configuration based on the current situation and environment. This requires system-wide analysis of objectives and resources, including dynamic consideration of faulty components. We propose to incorporate fault management into these systems by utilizing QoS characteristics such as integrity, availability, and reliability, within a hierarchical resource management architecture.

Overview

QoS-based fault management, the systematic handling of faults and failures within a system, necessarily incorporates traditional concepts of failure detection, identification, analysis and prediction. In addition, it must include functions for analyzing component dependencies and for distributing failure and fault data to interested parties, including resource managers and information visualizers.

Large, complex systems comprise multiple applications with disparate time scales. QoS-based fault handling must accommodate these differences. Real-time subsystems must respond under tight time constraints. Resources needed for recovery must be available immediately and will often need to be pre-reserved. Other activities might allow time in which to dynamically consider alternative strategies. A run-time resource manager must balance these requirements by determining which resources are best utilized by holding them in reserve for the time-constrained subsystems.

Relationship to CORBA and ACE/TAO

CORBA—and ACE/TAO—provide an effective framework in which to address the problems of constructing large, complex, distributed systems. It provides a common typing system and a standard mechanism for invoking services. There is a common application namespace and Common Facilities includes many necessary general capabilities. In addition, its applicability to real-time systems has been proven.

At the same time, CORBA's strength, object orientation (OO), interferes with the effective provision of end-to-end properties. OO enforces opaqueness in order to encourage reuse—a component is only obligated to do what its documentation says it must do. Thus, a component that does a function is good; a fault-tolerant version of that component is better; but there is no insight into the internal behavior of a commodity component.

Quorum fostered the development of translucent layers. In order to allow the effective use of component reliability into run-time resource management, we must identify fault-related QoS metrics and characterize reusable, translucent object components. In much the same way that BBN's Quality of Objects (QuO) project identified performance characteristics and incorporated them into wrapper objects, we must make fault-related information available to resource management.

An Initial Experiment

We have been working with the Naval Surface Warfare Center (NSWC) on a prototype battle defense system built from distributed components. A fundamental problem with such a system is that in order for the overall application to meet its time deadlines in the presence of failures, subtasks must operate with much shorter time constraints. In particular, the use of group communication techniques for scalability and fault tolerance requires detection of failed members an order of magnitude faster than the end-to-end application specifications. Rather than require that the entire application be written to

these tighter requirements, we have isolated the node failure detector into a separate component that can satisfy the more stringent time constraints. Written using real-time programming techniques and utilizing reserved CPU resources, this *fast failure detector* (FFD) employs a heartbeat with deadline function to reliably detect host failures in sub-second time frames even in the presence of competing CPU loads.

The FFD notifies applications about node failures and provides regular reports to the resource manager on host status. The FFD can also provide additional metrics, such as whether a host is in danger of missing its heartbeat deadline, which would cause a *false positive* failure indication.

Note that the FFD does not actually detect group member failures. That would require that the special real-time programming techniques be applied to the overall application. Instead, each FFD actually detects failure of other FFD components on other nodes. Use of the dependency tree information mentioned earlier then allows us to reason about the effect on the overall system. In this case, failure of the FFD is highly correlated to failure of the node upon which it operates. A mission-critical application will have been extensively reviewed and tested. So, the most likely cause of its failure will be due to a problem in the underlying node, which in the most relevant context will most probably be due to battle damage, the intrusion of an foreign object into the host hardware. Finally, we retain the original failure detection capability of the group communication system, which continues to detect failures of group members—now due to less likely causes and with a much lower probability of occurrence.

Conclusion

Our initial investigation, including the FFD experiment, has supported our belief that knowledge about faults can be effectively incorporated into resource allocation decisions and that the use of this information can improve the coordination between applications relative to resource sharing. We are in the process of building a real-time group communication product that uses the FFD, and we hope to extend the concepts with further research in applying hierarchical resource management in large, complex systems built using CORBA (and ACE/TAO) technology.

— end —